

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

СИСТЕМЫ ЭЛЕКТРОННОГО
ДОКУМЕНТООБОРОТА СКЛАДА ВООРУЖЕНИЯ
И ВОЕННОЙ ТЕХНИКИ

Студент

21.06.19 Тумасов Андрей Андреевич
(подпись, дата) (ФИО полностью)

Руководитель

22.6.19 А.В. Дубравин
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

22.06.19 А.В. Кучин
(подпись, дата) (фамилия, инициалы)

Рецензент т.п. специалист
ООО, Бизнес - программа 21.06.19 А.В. Своякин
Работа допущена к защите (протокол зас. кафедры от 31.05.2019 г. № 38-09-18)

Заведующий кафедрой

М М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
05.07.2019 № 4)

Секретарь ГЭК

Е Е.И. Гурин

Пенза 2019



Заведующему кафедрой

"Вычислительная техника"

от студента группы 14BO2

Глумскова Андрея Олеговича

Заявление

Прошу утвердить в качестве руководителя ВКР: Дубравина А.В., старшего преподавателя каф. ВТ Пензенского государственного университета.

Тема ВКР: Система электронного документооборота склада вооружения и военной техники

Дата: ____ . ____ .20__ г.

Руководитель _____
(подпись)

Студент _____
(подпись)

ОТЗЫВ

на выпускную квалификационную работу специалиста

Пензенского государственного университета

Глумскова Андрея Олеговича

Тема работы: "Система электронного документооборота склада вооружения и военной техники"

Дипломная работа посвящена разработке специализированного приложения, обеспечивающего упрощение и автоматизацию работы с документацией в воинской части.

Актуальность темы заключается в том, что на данный момент в воинских частях нет программ для обеспечения электронного документооборота и все операции выполняются с применением бумажных носителей. Программное решение проблемы позволяет упростить для должностных лиц работу, связанную с документооборотом и автоматизировать процесс обработки, согласования, передачи и хранения документов.

В процессе проектирования проведено моделирование базы данных в системе управления базами данных PostgreSQL, проведена разработка серверного и клиентского приложений на языке C# в среде Visual Studio 2017.

Глумсков А.О. справился с поставленной задачей. Он проявил самостоятельность при выполнении дипломной работы. Дипломный проект полностью соответствует заданию на проектирование. В процессе работы Глумсков А.О. показал хорошие знания по специальным дисциплинам, проявил себя сформировавшимся специалистом, способным самостоятельно решать инженерные задачи. Он хорошо владеет современными средствами разработки программного обеспечения.

Считаю, что работа соответствует требованиям к дипломным работам специалистов, заслуживает оценки "отлично", а Глумсков Андрей Олегович - присвоения квалификации специалиста по направлению 09.05.01 "Применение и эксплуатация автоматизированных систем специального назначения."

Руководитель ст. преподаватель каф. ВТ



А.В. Дубравин

Ректору Пензенского
государственного университета
А.Д.Гулякову
студента 5 курса ФВТ очной
формы обучения
Глумскова Андрея
Олеговича

ЗАЯВЛЕНИЕ

Я, Глумсков Андрей Олегович, студент 5 курса очной формы обучения направления подготовки 09.05.01 Применение и эксплуатация автоматизированных систем специального назначения, прошу провести проверку с использованием системы «Антиплагиат» выпускной квалификационной работы на тему «Система электронного документооборота склада вооружения и военной техники», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.

Подпись «__» _____ 2019 г.

ВКР представлена на проверку «5» июня 2019 г.

Руководитель ВКР

 А. В. Дубравин

Протокол

проверки на оригинальность в системе «АНТИПЛАГИАТ» (<https://www.antiplagiat.ru>) выпускной квалификационной работы на тему «Система электронного документооборота склада вооружения и военной техники» студента 5 курса Глумскова А.О. направления подготовки/специальности 09.05.01, Применение и эксплуатация автоматизированных систем специального назначения.

Руководитель ВКР старший преподаватель кафедры «ВТ» Дубравин А.В.



Отчет о проверке на заимствования №1



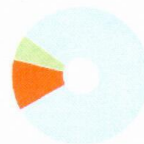
Автор: Митрохин Максим Александрович ID: 1214
Проверяющий: Митрохин Максим Александрович ID: 1214
Организация: Пензенский государственный университет
Отчет предоставлен сервисом «Антиплагиат» - <https://online.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 38
Начало загрузки: 07.06.2019 01:40:18
Длительность загрузки: 00:00:40
Имя исходного файла: кр. глумсков, 14862
Размер текста: 4863 kb
Тип документа: Выпускная квалификационная работа
Символьная длина: 546624
Слов в тексте: 35233
Число предложений: 5457

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет пред:
Начало проверки: 07.06.2019 01:40:54
Длительность проверки: 00:00:12
Комментарий: не указан
Модуль поиска: Модуль поиска ИСК "Антилет", Модуль выделения библиографических записей, Словная коллекция 396, Коллекция ИТБ, Цитирование, Модуль поиска первоисточников заимствований, Коллекция ИИ ВРАНУ.Ю, Коллекция ГАРАНТ, Модуль поиска Интернет, Коллекция Медицина, Модуль поиска перефразирований и ИИ ВРАНУ.Ю, Модуль поиска перефразирований Интернет, Коллекция Патенты, Модуль поиска ПЕРУ, Модуль поиска общепотребительных выводов, Коллекция ВУЗов



ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
11.85%	5.96%	82.19%

«7» июня 2019 г. Дубравин А.В.

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« » 20 г.

**ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ**

1. Студент Глумсков Андрей Олегович гр. 14ВО3 факультета ФВТ

2. Тема работы Система электронного документооборота склада вооружения и военной техники

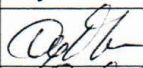
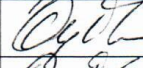
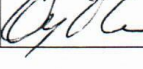
Тема утверждена приказом ПГУ № 554/О от "29" апреля 2019 г.

3. Руководитель работы Дубравин Алексей Викторович

4. Задание на работу (назначение разработки, исходные данные и т.п.) Разработать систему электронного документооборота склада вооружения и военной техники. Система должна обеспечивать хранение и обработку входящих документов, исходящих документов и документов для внутреннего использования, разграничение доступа пользователей к документам, рассылку документов пользователям, постановку документов на маршрут, обеспечивать возможность подписания документов, блокировку доступа к документам при редактировании и на время согласования, возможность разделения документов по каталогам, сдачу документов в архив, уничтожение документов не подлежащих хранению, версию документов

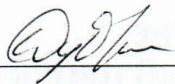
5. Перечень подлежащих разработке Разработка базы данных онлайн сервиса, разработка логической модели базы данных, разработка физической модели базы данных, разработка программного обеспечения, разработка и анализ пользовательского интерфейса, разработка программного обеспечения клиентской части, разработка программного обеспечения серверной части

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Выбор темы и разработка индивидуального плана проведения работ	10	11.02.19 - 12.02.19	
2	Подбор и изучение материала по теме работы	464	12.02.19 - 17.04.19	
3	Получение и анализ результатов	300	17.04.19 - 25.05.19	
4	Оформление отчёта	90	25.05.19 - 02.06.19	

Дата выдачи задания " _____ " _____ 20____ г.

Руководитель работы
(подпись, дата)

 22.6.19

Задание к исполнению принял студент
(подпись, дата)



Работу к защите допустить
Декан факультета
(подпись, дата)

 28.09.2019

РЕЦЕНЗИЯ

на дипломный проект студента

Пензенского государственного университета

Глумскова Андрея Олеговича

**Тема проекта: “ Система электронного документооборота склада
вооружения и военной техники “**

Проект Глумскова Андрея Олеговича посвящен разработке системы электронного документооборота склада вооружения и военной техники.

Актуальность данного проекта заключается в том, в настоящее время документооборот в вооруженных силах Российской Федерации происходит с использованием бумажных носителей. Разработанная система электронного документооборота способна упростить и автоматизировать процесс работы с документами.

Дипломный проект полностью соответствует заданию на дипломное проектирование. Пояснительная записка и графическая часть выполнены грамотно и с высоким качеством.

По материалам дипломного проекта, очевидно, что Глумсков А.О. умеет решать инженерные задачи, владеет реальными практическими навыками разработки ПО.

Материалы дипломного проекта имеют практическую значимость и, могут быть использованы по назначению.

Считаю, что работа соответствует требованиям к дипломным работам специалистов, установленных государственным образовательным стандартом, заслуживает оценки "отлично", а Глумсков Андрей Олегович - присвоения квалификации специалиста по направлению 09.05.01 "Применение и эксплуатация автоматизированных систем специального назначения."

Рецензент
К.т.н., гл. специалист
ООО «Бизнес-программа»



Севастьянов А.В.

РЕФЕРАТ

Пояснительная записка содержит 286 листов, 39 рисунков, 22 таблицы, 7 источников, 2 приложения.

СИСТЕМА ЭЛЕКТРОННОГО ДОКУМЕНТООБОРОТА СКЛАДА ВООРУЖЕНИЯ И ВОЕННОЙ ТЕХНИКИ.

Объектом разработки является система электронного документооборота склада вооружения и военной техники.

Цель работы – проведение предпроектного анализа деятельности организации и программная реализация части требуемых доработок системы под требования предприятия.

В результате проведенной работы создан проект, удовлетворяющей поставленной задаче.

В пояснительной записке описано: проведение предпроектного анализа, проектирование системы и сама система.

В проектной части описаны выполненные действия по проектированию системы электронного документооборота склада вооружения и военной техники.

В разделе безопасности жизнедеятельности описаны психофизиологические факторы, воздействующие на оператора ЭВМ, а так же рассмотрено воздействие работы с ПК на зрение человека.

Графическая часть содержит схемы базы данных, а так же блок-схемы алгоритма работы разработанного приложения.

					ПГУ 1.090501.03.001 ПЗ				
Изм.	Лист	№ докум.	Подп.	Дат.	Система электронного документооборота склада вооружения и военной техники. Пояснительная записка.	Лит.	Лист	Листов	
Разраб.		Глумсков А.О.		11.06.19					
Пров.		Дубравин А.В.		11.06.19			3	286	
Н. контр.		Кучин А.В.		11.06.19		Гр. 14В02			
Утв.									

Содержание

1. Предпроектный анализ.....	8
1.1. Основание для разработки.....	8
1.2. Назначение разработки.....	8
1.3. Требования к программе.....	9
1.3.1. Требования к функциональным характеристикам.....	9
1.3.2. Требования к надежности.....	10
1.3.3. Требования к составу и параметрам технических средств.....	10
1.3.4. Требования к информационной и программной совместимости.....	11
1.4. Требования к программной документации.....	11
1.5. Состав и содержание работ по созданию системы.....	11
1.6. Порядок контроля и приемки.....	12
2. Проектирование приложения "Система электронного документооборота склада вооружения и военной техники"	13
2.1. Цель проектирования.....	13
2.2. Описание CASE-средства Open ModelSphere.....	13
2.3. Разработка базы данных.....	14
2.3.1. Разработка концептуальной диаграммы.....	14
2.3.2. Разработка связей.....	21
2.3.3. Разработка реляционной модели данных.....	26
2.3.4. Разработка физической модели данных.....	34
2.4. Алгоритм работы программы.....	45
2.5. Иерархия форм.....	54
3. Технология разработки и отладки отчётных форм АСУ.....	56

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Введение.....	56
3.1. Разработка базы данных.....	56
3.1.1. Реляционные базы данных.....	56
3.1.2. Сущности и атрибуты в реляционной модели.....	57
3.1.3. Связи в реляционной модели.....	58
3.1.4. Краткое описание возможностей PostgreSQL.....	58
Функции.....	59
3.2. Описание модуля Windows Communication Foundation.....	63
3.3. Методика испытаний.....	68
4. Психофизиологические факторы, воздействующие на оператора.....	71
Введение.....	71
4.1. Аспекты производственной безопасности при работе на ПК.....	71
4.1.1. Психофизиологические факторы.....	72
4.1.2. Оборудование рабочих мест с ПК.....	74
4.1.3. Освещение рабочего места.....	75
5. Руководство пользователя по эксплуатации.....	77
5.1. Меню "Действия".....	80
5.1.1. Действие создания сотрудника.....	80
5.1.2. Действие создания пользователя.....	81
5.1.3. Действия создания типа.....	82
5.1.4. Действие создания подразделения.....	83
5.1.5. Действие создания должности.....	84
5.1.6. Действие создания воинского звания.....	84
5.1.7. Действие создания роли.....	85

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

5.1.8. Действие создания маршрута.....	86
5.1.9. Действие создания стадии.....	87
5.2. Списки.....	88
5.3. Работа с деревом документов.....	90
Приложение А	101
Приложение Б.....	280

Введение

Система электронного документооборота (СЭД) — это система,

ПГУ 1.090501.03.001 ПЗ

					Лист
Изм.	Лист	№ докум.	Подпись	Дат	280

которая позволяет организовать и автоматизировать, а тем самым и упростить работу электронными документами в процессе работы с ними. СЭД должна обеспечивать функции создания, изменения, хранения и маршрутизации документов, а также ряда сервисных возможностей, таких как поиск, классификация и т.д.

Кроме того, такие системы предназначены для автоматизации процессов взаимодействия между сотрудниками (передачи документов, отправки документов, выдаче заданий и т.д.). При этом СЭД могут обеспечивать работу как с текстовыми документами (.doc,.txt,.docx,.odt и т.д.), так и с изображениями(.pdf, .jpg, .jpeg и т.д.).

Документооборот связан с работой с корреспонденцией, документами распорядительного и организационного характера и т.д. Системы электронного документооборота позволяют также связать каждый документ с действиями, которые будут выполняться в процессе работы с ними. Благодаря этому, обеспечивается возможность отслеживать выполнение сотрудниками назначенных работ. Таким образом, системы электронного документооборота позволяют обеспечить не только автоматизацию управления потоками документов и обеспечить информационную безопасность в компании, но и повысить контроль исполнения работ по документам и продуктивность работы сотрудников.

1. Предпроектный анализ

1.1. Основание для разработки

Основанием для разработки системы электронного документооборота места хранения вооружения и военной техники является задание на дипломное проектирование, выданное руководителем дипломного

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

проектирования и утвержденное заведующим кафедрой ВТ.

1.2. Назначение разработки

Должностным лицам, ответственным за места хранения вооружения и военной техники поступают разнотипные документы, информация о поставках, приказы по войсковой части и приказы от министерства обороны Российской Федерации на различных носителях данных, как на бумажных носителях, так и в электронном виде. Для ее обработки необходимо разработать единую систему, которая имеет общий формат для каждого документа и обеспечивает хранение, корректировку и согласование полученных документов, с возможностью прикрепления документов различного формата к конкретной записи о них в системе.

Большое число документов разного типа усложняет работу сотрудников и снижает эффективность работы, а так же для хранения большого количества документов выполненных на бумажных носителях требуется отведение для них специально оборудованного для этого места.

Для того чтобы упростить и повысить скорость обработки данных следует обобщить и структурировать имеющиеся документы. Решением этой задачи является консолидация данных. В этом случае данные и документы хранятся в единой базе данных и предоставляются для обработки в удобной для администратора форме.

Программа предназначена для упрощения работы военнослужащих, работающих с документами, связанными с местом хранения вооружения и военной техники.

1.3. Требования к программе

1.3.1. Требования к функциональным характеристикам

					ПГУ 1.090501.03.001 ПЗ	Лист 280
Изм.	Лист	№ докум.	Подпись	Дат		

Система электронного документооборота, предназначенная для автоматизации работы с документами, относящимися к местам хранения вооружения и военной техники, должна обеспечивать возможность выполнения следующих функций:

- обеспечивать хранение, обработку и согласование входящих документов, исходящих и документов для внутреннего использования
- разграничение доступа пользователей к документам;
- рассылка документов должностным лицам;
- согласование документов;
- блокировка доступа при редактировании документа;
- формирование каталогов для документов относящихся к одному вопросу;
- сдача документов в архив (запрет на редактирование документов);
- удаление документов;
- версию документов;

Разрабатываемая информационная система электронного документооборота позволит ускорить и упростить процесс работы с документами, обмена документами между должностными лицами войсковой части, согласования документов и их хранения.

Данное ПО легкое в применении и эффективное в эксплуатации, которое не требует много усилий от пользователя. Программа разработана так, чтобы пользователи затратили минимум усилий для выполнения своих должностных обязанностей связанных с работой с документами и документооборотом.

Общим требованием системы является создание дружественного интерфейса, с помощью которого пользователь мог бы легко и быстро использовать программу по вышеперечисленным требованиям. Работа с

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

программой не требует предварительного изучения сопроводительной документации. Это говорит об удобстве обучения.

1.3.2. Требования к надежности

Программа должна работать без остановки по программным ошибкам. Если при разборе документов обрабатываемые данные нарушают условия, которым должны удовлетворять хранимые в базе данные, выдается сообщение об ошибке. Затем спорные и ошибочные данные уточняются в присутствии администратора базы данных и лиц, ответственных за создание документов. Корректные данные заносятся в базу данных.

Неавторизованный пользователь не должен иметь свободный доступ к документам.

1.3.3. Требования к составу и параметрам технических средств

Для обеспечения нормальной работоспособности данного программного продукта необходим стандартный комплекс технических средств из состава персональных компьютеров:

- операционная система Windows 7 и выше;
- ОЗУ 4Gb и выше;
- от 40Gb свободного места на жёстком диске для хранения документов.

1.3.4. Требования к информационной и программной совместимости

Система должна быть разработана в соответствии с требованиями технического задания.

Структура базы данных системы должна соответствовать структуре, разработанной в процессе логического проектирования.

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

Программа должна быть реализована в среде программирования Microsoft Visual Studio 2017 с использованием системы управления базами данных PostgreSQL. Дополнительного программного обеспечения не требуется.

Рекомендуемые операционные системы –Windows 7 и выше.

1.4. Требования к программной документации

В процессе проектирования системы необходимо подготовить следующие документы:

- описание программы;
- текст программы;
- программа и методика испытания;
- описание применения.

1.5. Состав и содержание работ по созданию системы

- Построение инфологической модели;
- структурный анализ;
- структурное проектирование системы;
- логическое проектирование базы данных;
- разработка технического задания;
- создание базы данных;
- написание пояснительной записки.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

1.6. Порядок контроля и приемки

Контроль и прием разработанной системы осуществляется заказчиком, разработчиком и представителем организации, на базе которой ведется разработка. По мере разработки новых модулей, реализующих функцию, должны быть разработаны тестовые последовательности действий, проверяющие правильность функционирования этих модулей.

Приемка программного приложения должна проводиться при представлении его работоспособности: при предъявлении системе в качестве входных данных специально разработанных тестовых наборов, а также при наличии полной документации к программе.

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

1. Проектирование приложения "Система электронного документооборота склада вооружения и военной техники"

1.7. Цель проектирования

Целью проектирования является структурный анализ представленной в задании предметной области. Необходимо разработать систему электронного документооборота места хранения вооружения и военной техники.

1.8. Описание CASE-средства Open ModelSphere

Разработка функциональной модели проводилась с помощью CASE – средства Open ModelSphere.

Open Modelsphere – свободно распространяемое, достаточно простое и удобное в освоении CASE-средство. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной модели (диаграмм потоков данных и диаграмм «сущность-связь»). Не требовательно к ресурсам системы. Open Modelsphere поддерживает все системы управления базами данных в общем виде. Так же включает в себя модули экспертов для проверки целостности архитектуры. Такая проверка может сэкономить время проектирования. Таким образом, для проектирования базы данных автоматизированной системы выбрано CASE-средство Open Modelsphere 3.2. На основе вышесказанного для разработки и функционирования визуального приложения для оповещения работников предприятия в экстренных ситуациях, были выбраны следующие программные средства: — в качестве СУБД выбрана СУБД PostgreSQL; — в качестве средства для проектирования базы данных выбрано CASE-средство Open Modelsphere 3.2.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

1.9. Разработка базы данных

1.9.1. Разработка концептуальной диаграммы.

В результате анализа изучаемой предметной области было обозначено 19 основных сущностей для создания базы данных системы электронного документооборота склада вооружения и военной техники. Сущности в разработанной ER-диаграмме описаны в таблице 2.1.

На этапе логического проектирования дается обоснование разбиения базы данных на таблицы. Разбиение осуществляется в соответствии с семантическим анализом предметной области, при этом, каждому объекту предметной области ставится в соответствие определенная сущность, характеристикам объекта соответствуют атрибуты сущности, идентификатору сущности ключевой атрибут в соответствии с таблицей 2.1. Полученная в результате моделирования диаграмма, называется концептуальной моделью базы данных или ER-диаграммой. Описание сущностей, соответствующих им атрибутов и ключевых полей каждой сущности приведено в таблице 2.1. Диаграмма концептуальной модели данных приведена на рисунке В1 приложения В.

Таблица 2.1 – Описание сущностей

Сущность	Атрибут
Документы	Идентификатор (РК – первичный ключ)
	Идентификатор родительского документа
	Наименование
Продолжение таблицы 2.1	
Сущность	Атрибут

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

	Наименование при создании
	Наименование при изменении
	Расположение
	Описание
	Автор изменения
	Текущая стадия
Сотрудники	Идентификатор (РК – первичный ключ)
	Фамилия
	Имя
	Отчество
	Тип удостоверяющего документа
	Документ
	Дата рождения
	Рабочий телефон
	Мобильный телефон
	Домашний телефон
Продолжение таблицы 2.1	
Сущность	Атрибуты
	Признак удаления
Входящие письма	Идентификатор входящего письма (РК – первичный ключ)

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

	Тема письма
	Получатель
	Отправитель
	Сообщение
	Прочитано
	Удалено
	Дата отправки
Исходящие письма	Идентификатор исходящего письма (РК – первичный ключ)
	Тема письма
	Получатель
	Отправитель
	Сообщение
	Прочитано
Продолжение таблицы 2.1	
Сущность	Атрибут
	Признак удаления
	Дата отправки
	Отправлено
	Авто
Подразделения	Идентификатор подразделения (РК – первичный ключ)

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

	Идентификатор родительского подразделения
	Наименование
Должности	Идентификатор должности (РК – первичный ключ)
	Наименование
Адрес	Идентификатор адреса (РК – первичный ключ)
	Почтовый индекс
	Страна
	Адрес
Продолжение таблицы 2.1	
Сущность	Атрибут
Доступ	Идентификатор доступа (РК – первичный ключ)
	Наименование
Фото	Идентификатор фото (РК – первичный ключ)
	Фото
Звания	Идентификатор звания (РК – первичный ключ)
	Наименование

Пользователь	Идентификатор пользователя (РК – первичный ключ)
	Логин
	Пароль
Роли	Идентификатор роли (РК – первичный ключ)
	Наименование
	Контроль доступа
Продолжение таблицы 2.1	
Подписи	Идентификатор подписи (РК – первичный ключ)
	Дата подписи
	Стадия
Маршруты	Идентификатор маршрута (РК – первичный ключ)
	Наименование маршрута
	Маршрут
Тип	Идентификатор типа (РК – первичный ключ)
	Наименование
Группа типов	Идентификатор группы (РК – первичный ключ)
	Наименование

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

	Код
Стадии	Идентификатор стадии (РК – первичный ключ)
Продолжение таблицы 2.1	
Сущность	Атрибут
	Наименование
	Тип подписи
	Заблокирована
	Разблокирована
	Отправка назад
Файлы	Идентификатор файла (РК – первичный ключ)
	Путь к файлу
	Наименование
	Версия
История	Идентификатор записи в истории (РК – первичный ключ)
	Тип объекта
	Дата изменения

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Продолжение таблицы 2.1	Автор изменения
	Источник изменения
Сущность	Атрибут
	Изменение

1.9.2. Разработка связей

После этого разрабатывается структура связей отношений системы электронного документооборота мест хранения вооружения и военной техники.

К примеру сущность «Подразделения» связана с сущностью «Сотрудники» связью «один – ко – многим», это означает, что одно подразделение может иметь несколько сотрудников, а связь является неидентифицирующей. В свою очередь «Сотрудники» связана с сущностью «Подразделения» связью «один – к – одному», это означает, что один сотрудник может состоять только в одном подразделении, а связь является идентифицирующей. Описание связей родительских сущностей с дочерними приведено в таблице 2.2.

Таблица 2.2 - Описание сущностей

Родительская сущность	Дочерняя сущность	Описание связи
-----------------------	-------------------	----------------

Подразделения	Сотрудники	Связь неидентифицирующая, один ко многим. В одном подразделении может состоять одновременно множество сотрудников.
---------------	------------	--

Продолжение таблицы 2.2

Родительская сущность	Дочерняя сущность	Описание связи
Сотрудники	Доступ	Связь неидентифицирующая, один ко многим. Один сотрудник может иметь множество доступов.
Подразделения	Доступ	Связь неидентифицирующая, один ко многим. Одно подразделение может иметь множество доступов.
Документы	Доступ	Связь неидентифицирующая, один ко многим. Один документ может иметь множество доступов.
Должности	Сотрудники	Связь неидентифицирующая, один ко многим. На одной должности может состоять множество сотрудников.
Адрес	Сотрудники	Связь идентифицирующая, один к

Изм.	Лист	№ докум.	Подпись	Дат	ПГУ 1.090501.03.001 ПЗ	Лист
						280

Продолжение таблицы 2.2		одному. Один адрес может быть только у одного сотрудника.
Родительская сущность	Дочерняя сущность	Описание связи
Звание	Сотрудники	Связь неидентифицирующая, один ко многим. Одно звание может быть присвоено множеству сотрудников.
Фото	Сотрудники	Связь идентифицирующая, один к одному. Одно фото может быть только у одного сотрудника.
Сотрудники	Пользователь	Связь неидентифицирующая, один ко многим. Один сотрудник может иметь множество учетных записей пользователя.
Роли	Пользователь	Связь неидентифицирующая, один ко многим. Одна роль может предоставляться множеству пользователей.
Стадии	Документы	Связь неидентифицирующая, один ко многим. Одна стадия может быть связана с множеством документов.

Родительская сущность	Дочерняя сущность	Описание связи
Документы	История	Связь неидентифицирующая, один ко многим. Один документ может соответствовать множеству записей в истории.
Документы	Входящие сообщения	Связь неидентифицирующая, один к многим. Один документ может быть связан с множеством входящих сообщений.
Документы	Исходящие сообщения	Связь неидентифицирующая, один к многим. Один документ может быть связан с множеством исходящих сообщений.
Маршруты	Подписи	Связь неидентифицирующая, один к многим. Один маршрут может быть связан с множеством подписей.
Типы документов	Подписи	Связь неидентифицирующая, один ко многим. Один тип документа может соответствовать множеству подписей.

Родительская сущность	Дочерняя сущность	Описание связи
Группа типов	Типы документов	Связь неидентифицирующая, один ко многим. Одна группа может содержать множество типов.
История	Файлы	Связь идентифицирующая, один к одному. Одна запись в истории соответствует одному файлу
Входящие сообщения	Исходящие сообщения	Связь идентифицирующая, один ко многим. Одно входящее сообщение соответствует одному исходящему, в свою очередь одно исходящее сообщение соответствует одному входящему.

Для дочерних таблиц базы данных справедливо, то, что все связи с родительскими таблицами будут один - к - одному и будут являться идентифицирующими.

1.9.3. Разработка реляционной модели данных

На этапе проектирования реляционной модели данных осуществляется определение внешних ключей (далее FK(Foreign key)) для таблиц, организация ссылочной целостности.

Все поля таблиц являются не нулевыми. В случае если поле было не заполнено, то оно заполняется автоматически через программу.

Описание сущностей с созданными внешними ключами приведено в таблице 2.3

Таблица 2.3 – Описание сущностей с внешними ключами

Сущность	Атрибут
Документы	Идентификатор (РК – первичный ключ)
	Идентификатор родительского документа
	Наименование
	Наименование при создании
	Наименование при изменении
	Расположение
	Описание
	Автор изменения
Продолжение таблицы 2.3	
Сущность	Атрибут
Документы(продолжение)	Текущая стадия

	Идентификатор стадии(FK)
	Идентификатор сотрудника
Сотрудники	Идентификатор (РК – первичный ключ)
	Фамилия
	Имя
	Отчество
	Тип удостоверяющего документа
	Документ
	Дата рождения
	Рабочий телефон
	Мобильный телефон
	Домашний телефон
	Удалён ли сотрудник
	Идентификатор адреса(FK)
	Идентификатор должности(FK)
Продолжение таблицы 2.3	
Сущность	Атрибут
Сотрудники (продолжение)	Идентификатор подразделения(FK)
	Идентификатор фото(FK)
	Идентификатор звания(FK)
Входящие письма	Идентификатор входящего письма (РК – первичный ключ)

	Тема письма
	Получатель
	Отправитель
	Сообщение
	Прочитано
	Удалено
	Дата отправки
	Идентификатор исх. сообщения(FK)
	Идентификатор документа(FK)
	Идентификатор сотрудника(FK)
Продолжение таблицы 2.3	Идентификатор исходящего письма (РК – первичный ключ)
Сущность	Атрибут
Исходящие письма (продолжение)	Тема письма
	Идентификатор пользователя(FK)
Исходящие письма(продолжение)	Отправитель
	Получатель
	Сообщение
	Прочитано
	Удалено
	Дата отправки
	Отправлено

Подразделения	Авто
	Идентификатор документа(FK)
	Идентификатор подразделения (РК – первичный ключ)
	Идентификатор родительского подразделения
Наименование	

Продолжение таблицы 2.2

Сущность	Атрибуты
Должности	Идентификатор должности (РК – первичный ключ)
	Наименование
Адрес	Идентификатор адреса (РК – первичный ключ)
	Почтовый индекс
	Страна
	Адрес
Доступ	Идентификатор доступа (РК – первичный ключ)
	Наименование
	Идентификатор сотрудника(FK)
	Идентификатор подразделения(FK)

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

	Идентификатор документа(FK)
Фото	Идентификатор фото (PK – первичный ключ)
	Фото
Продолжение таблицы 2.3	

Сущность	Атрибуты
Звания	Идентификатор звания (PK – первичный ключ)
	Наименование
Пользователь	Идентификатор пользователя (PK – первичный ключ)
	Логин
	Пароль
	Идентификатор роли(FK)
	Идентификатор сотрудника(FK)
Роли	Идентификатор роли (PK – первичный ключ)
	Наименование
	Контроль доступа
Подписи	Идентификатор подписи (PK – первичный ключ)
	Дата подписи
	Стадия

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Продолжение таблицы 2.3	Идентификатор маршрута(FK)
	Атрибут
Сущность	Атрибут
Подписи(продолжение)	Идентификатор типа(FK)
	Идентификатор сотрудника(FK)
	Идентификатор документа(FK)
Маршруты	Идентификатор маршрута (PK – первичный ключ)
	Наименование маршрута
	Маршрут
	Идентификатор документа(FK)
Тип	Идентификатор типа (PK – первичный ключ)
	Наименование
	Идентификатор группы типов(FK)
Группа типов	Идентификатор группы (PK – первичный ключ)
	Наименование
	Код
Стадии	Идентификатор стадии (PK – первичный ключ)
Продолжение таблицы 2.3	
Сущность	Атрибут
Стадии(продолжение)	Наименование

	Тип подписи
	Заблокирована
	Разблокирована
	Отправка назад
	Отправка назад
Файлы	Идентификатор файла (РК – первичный ключ)
	Путь к файлу
	Наименование
	Версия
	Идентификатор документа(FK)
	Идентификатор записи в истории(FK)
История	Идентификатор записи в истории (РК – первичный ключ)
	Тип объекта

Продолжение таблицы 2.3

Сущность	Атрибуты
История(продолжение)	Дата изменения
	Автор изменения
	Источник изменения
	Изменение
	Идентификатор документа(FK)

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

1.9.4. Разработка физической модели данных

На этапе физического проектирования осуществляется определение таблиц и полей: имени поля, типа данных поля, свойств поля. Тип данных определяет вид информации, которая будет храниться в данном поле.

На основании реляционной модели данных описанной в пункте 2.3.3, была разработана физическая модель данные. Её описание приведено в таблицах 2.4 - 2.22

Таблица 2.4 –Описание полей таблицы Документы

Поле	Физическое имя	Тип данных
Идентификатор (PK – первичный ключ)	ID	Integer
Идентификатор родительского	ParentID	Integer
Продолжение таблицы 2.4		
Поле	Физическое имя	Тип данных
Наименование	Name	Character(255)
Дата создания	CreateDate	Date
Дата изменения	EditDate	Date
Обозначение	Designation	Character(255)
Описание	Description	Character(255)
Автор изменения	EditAuthor	Integer
Текущая стадия	StageNum	Integer
Поле	Физическое имя	Тип данных

Идентификатор стадии(FK)	StagesID	Integer
Идентификатор сотрудника(FK)	EmployeesID	Integer

Таблица 2.5 – Описание полей таблицы Пользователи

Поле	Физическое имя	Тип данных
Идентификатор (PK – первичный ключ)	ID	Integer
Продолжение таблицы 2.5	FirstName	Character(200)
Поле	Физическое имя	Тип данных
Имя	MiddleName	Character(200)
Отчество	LastName	Character(200)
Тип удостоверяющего документа	TypeDoc	Integer
Документ	Doc	Character(200)
Дата рождения	Birthday	Date
Рабочий телефон	WorkPhone	Character(200)
Мобильный телефон	MobilePhone	Character(200)
Домашний телефон	HomePhone	Character(200)
Удалён ли сотрудник	Deleted	Logic
Идентификатор адреса(FK)	AddressID	Integer
Идентификатор должности(FK)	AppointmentID	Integer

Идентификатор подразделения(FK)	SubdivisionID	Integer
Идентификатор фото(FK)	PhotoID	Integer
Идентификатор звания(FK)	MilitaryRankID	Integer

Таблица 2.6 – Описание полей таблицы Входящие сообщения

Поле	Физическое имя	Тип данных
Идентификатор входящего письма (PK – первичный ключ)	ID	Integer
Тема письма	Theme	Character(200)
Получатель	ToUser	Integer
Отправитель	FromUser	Integer
Сообщение	Message	Character(255)
Прочитано	Readed	Logic
Удалено	Deleted	Logic
Дата отправки	DatePost	Date
Идентификатор исх. сообщения(FK)	PostMessageOutID	Integer
Идентификатор документа(FK)	DocumentsID	Integer
Идентификатор сотрудника(FK)	EmployeesID	Integer

Таблица 2.7 – Описание полей таблицы Исходящие сообщения

Поле	Физическое имя	Тип данных
Идентификатор исходящего письма (РК – первичный ключ)	ID	Integer
Тема письма	Theme	Character(200)
Получатель	ToUser	Integer
Отправитель	FromUser	Integer
Сообщение	Message	Character(255)
Прочитано	Readed	Logic
Удалено	Deleted	Logic
Дата отправки	DatePost	Date
Отправлено	Sent	Logic
Авто	Auto	Logic
Поле	Физическое имя	Тип данных
Идентификатор документа(FK)	DocumentsID	Integer
Идентификатор сотрудника(FK)	EmployeesID	Integer

Таблица 2.8 – Описание полей таблицы Подразделения

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Поле	Физическое имя	Тип данных
Идентификатор подразделения (РК – первичный ключ)	SubdivisionID	Integer
Идентификатор родительского подразделения	ParentID	Integer
Наименование	Name	Character(200)

Таблица 2.9 – Описание полей таблицы Должности

Поле	Физическое имя	Тип данных
Идентификатор должности (РК – первичный ключ)	ID	Integer
Наименование	Name	Character(255)

Таблица 2.10 – Описание полей таблицы Адресс

Поле	Физическое имя	Тип данных
Идентификатор адреса (РК – первичный ключ)	ID	Integer
Поле	Физическое имя	Тип данных
Почтовый индекс	PostIndex	Character(200)
Страна	Country	Character(200)
Адрес	Addr	Character(200)

Таблица 2.11 – Описание полей таблицы Доступ

Поле	Физическое имя	Тип данных
Идентификатор доступа (РК –	ID	Integer

первичный ключ)		
Тип доступа	AccessType	Integer
Идентификатор сотрудника(FK)	EmployeesID	Integer
Идентификатор подразделения(FK)	SubdivisionID	Integer
Идентификатор документа(FK)	DocumentsID	Integer

Таблица 2.12 – Описание полей таблицы Фото

Поле	Физическое имя	Тип данных
Идентификатор фото (PK)	ID	Integer
Photo	Photo	Bytea

Таблица 2.13 – Описание полей таблицы Звания

Поле	Физическое имя	Тип данных
Идентификатор звания(PK)	ID	Integer
Наименование	Name	Character(255)

Таблица 2.14 – Описание полей таблицы Пользователь

Поле	Физическое имя	Тип данных
Идентификатор пользователя (PK)	ID	Integer
Логин	Login	Character(255)

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Пароль	Password	Character(255)
Идентификатор роли(FK)	RolesID	Integer
Идентификатор сотрудника(FK)	EmployeesID	Integer

Таблица 2.15 – Описание полей таблицы Роли

Поле	Физическое имя	Тип данных
Идентификатор роли(PK)	ID	Integer
Наименование	Name	Character(255)
Поле	Физическое имя	Тип данных
Контроль доступа	ControlAccess	Character(255)

Таблица 2.16 – Описание полей таблицы Подписи

Поле	Физическое имя	Тип данных
Идентификатор подписи (PK – первичный ключ)	ID	Integer
Дата подписи	SignDate	Date
Стадия	SignNum	Integer
Идентификатор маршрута(FK)	RoutesID	Integer
Идентификатор типа(FK)	TypeID	Integer
Идентификатор сотрудника(FK)	EmplyesID	Integer
Идентификатор документа(FK)	DocumentsID	Integer

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Таблица 2.17 – Описание полей таблицы Маршруты

Поле	Физическое имя	Тип данных
Идентификатор маршрута (PK – первичный ключ)	ID	Integer
Поле	Физическое имя	Тип данных
Наименование маршрута	Name	Character(255)
Маршрут	Route	Character(255)
Идентификатор документа(FK)	DocumentsID	Integer

Таблица 2.18 – Описание полей таблицы Тип документа

Поле	Физическое имя	Тип данных
Идентификатор типа (PK – первичный ключ)	ID	Integer
Наименование	Name	Character(255)
Идентификатор группы типов(FK)	TypeGroupID	Integer

Таблица 2.19 – Описание полей таблицы Группа типов

Поле	Физическое имя	Тип данных
Идентификатор группы (PK – первичный ключ)	ID	Integer
Наименование	Name	Character(200)
Код	Code	Integer

Таблица 2.20 – Описание полей таблицы Стадии

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Поле	Физическое имя	Тип данных
Идентификатор стадии (РК – первичный ключ)	ID	Integer
Наименование	Name	Character(255)
Продолжение таблицы 2.20	SignType	Integer
Заблокирована	Lock	Logic
Разблокирована	Unlock	Logic
Отправка назад	Back	Logic

Таблица 2.21 – Описание полей таблицы Документы

Поле	Физическое имя	Тип данных
Идентификатор файла (РК – первичный ключ)	ID	Integer
Путь к файлу	FullPath	Character(255)
Наименование	Name	Character(255)
Версия	Version	Integer
Идентификатор документа(FK)	DocumentsID	Integer
Поле	Физическое имя	Тип данных
Идентификатор записи в истории(FK)	HistoryID	Integer

Таблица 2.22 – Описание полей таблицы Документы

Поле	Физическое имя	Тип данных
Идентификатор записи в истории (PK – первичный ключ)	ID	Integer
Тип объекта	ObjectType	Integer
Дата изменения	DataChange	Date
Автор изменения	ChangeAuthor	Integer
Источник изменения	ChangeHost	Character(255)
Изменение	Action	Character(255)
Идентификатор документа(FK)	DocumentsID	Integer

1.10. Алгоритм работы программы

Для работы приложения необходимо, чтобы были запущены следующие службы:

- Служба файлового сервера(FS.Host.exe)
- Служба сервера баз данных(Doc.Host.exe)

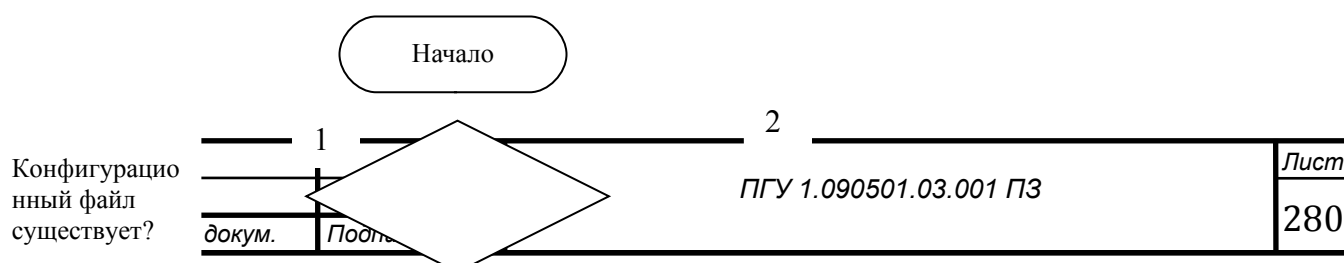
При первом запуске служб автоматически создаются конфигурационные файлы с настройками для подключения к сервисам WCF для работы с файловым хранилищем и с сервером баз данных. Из конфигурационного файла для файлового сервера считывается имя сервера, порт для

прослушивания, который впоследствии будет открыт для обеспечения взаимодействия между клиентским приложением и сервером. А из конфигурационного файла для сервера баз данных считывается ещё и строка подключения к серверу баз данных. Внутри служб запускаются соответствующие сервисы (Doc.Service.dll и FS.Sevice.dll), и внутри каждой службы открывается канал для приёма запросов от клиента.

Алгоритм работы служб приведён на рисунках 2.1 и 2.2.

При первом запуске клиентского приложения также создаётся конфигурационный файл с настройками по умолчанию, которые впоследствии считываются из него, и создаётся подключение к каналу для работы с сервером баз данных и каналу для работы с файловым хранилищем.

Если настройки неверны или службы недоступны, то пользователю будет выдано соответствующее сообщение и программа завершит работу. Затем, если подключение, к каналу для работы с базой прошло успешно, пользователь вводит свои учетные данные, которые проверяются на сервере.



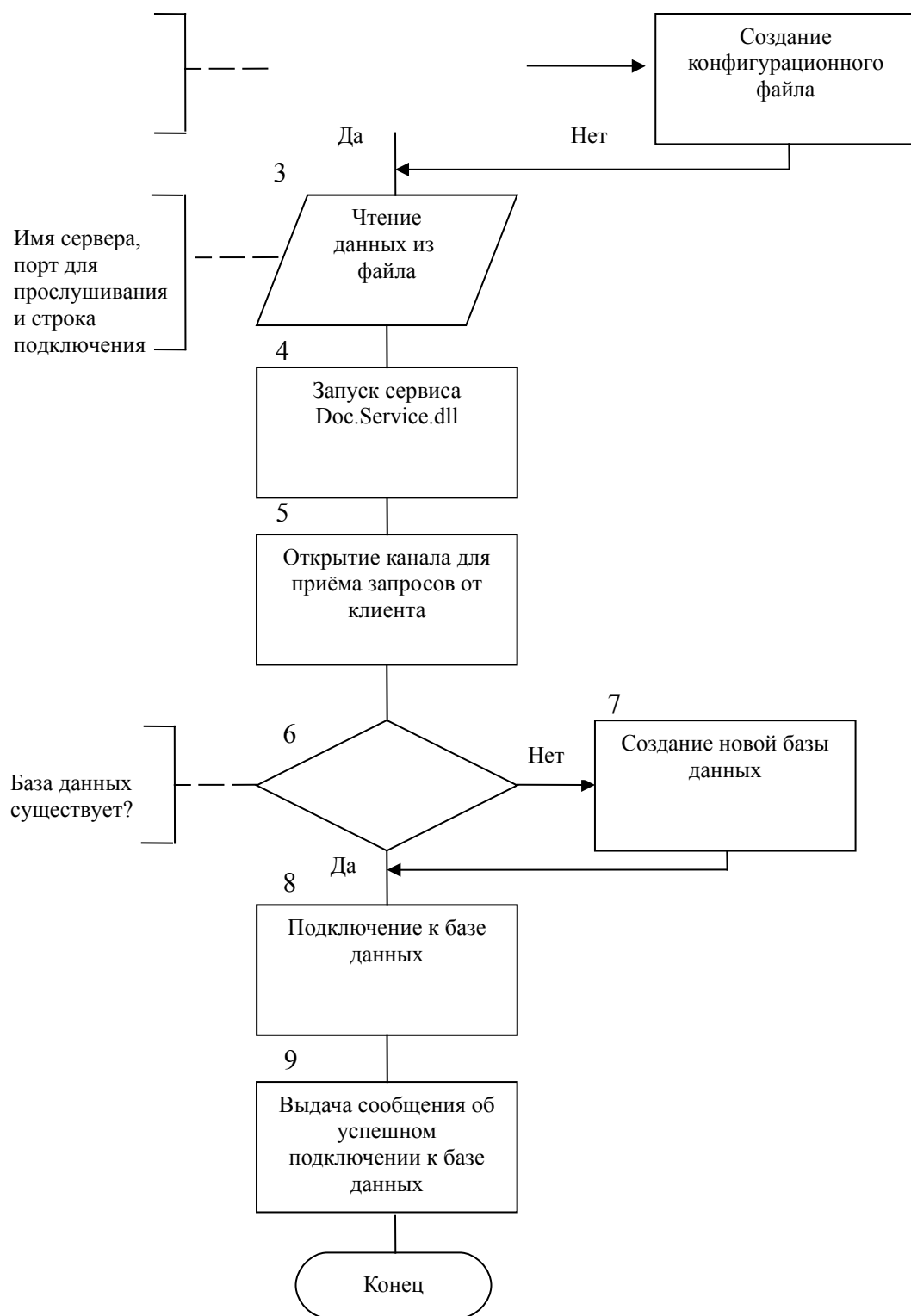


Рисунок 2.1 - Алгоритм работы службы баз данных

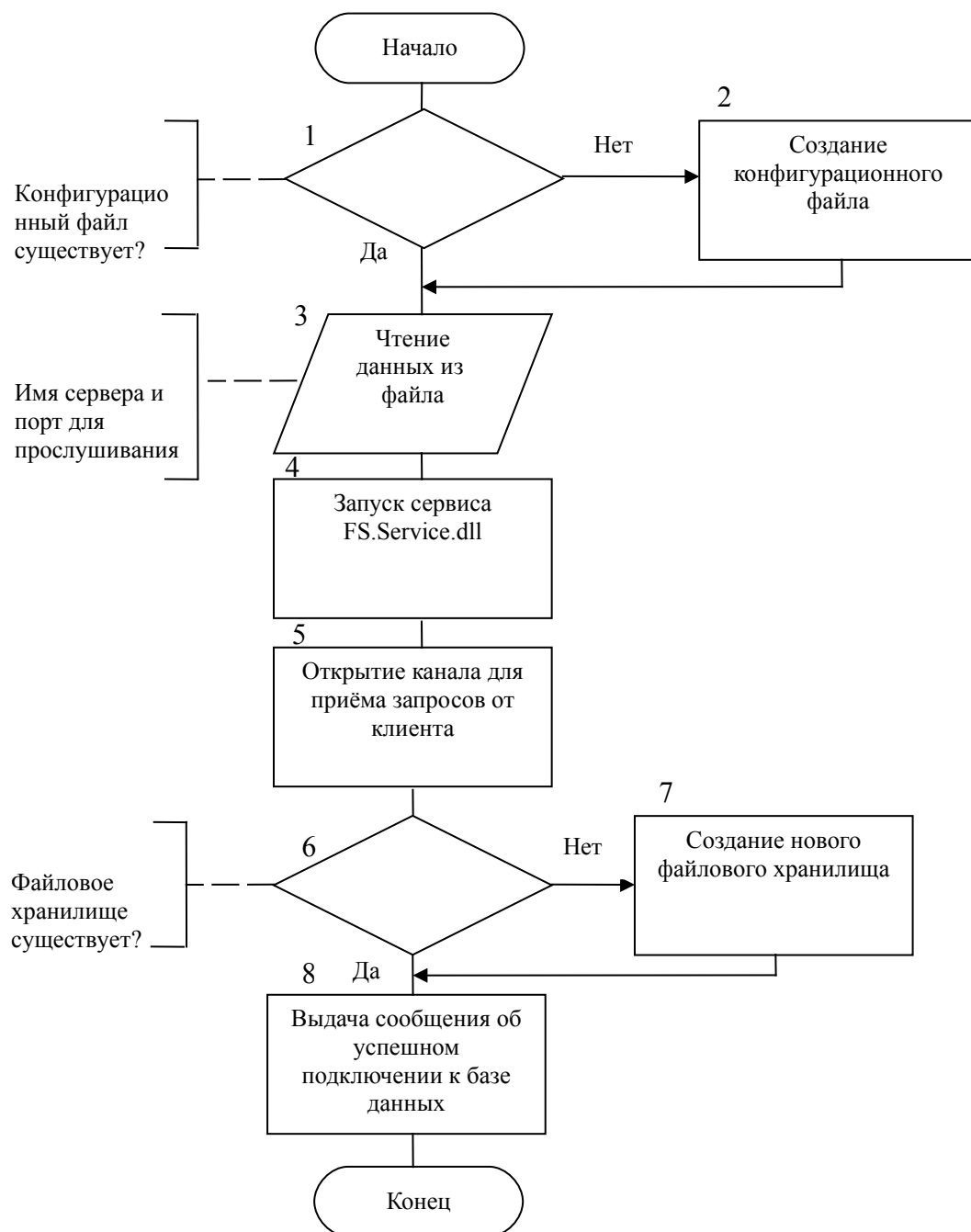


Рисунок 2.2 - Алгоритм работы службы файлового сервера

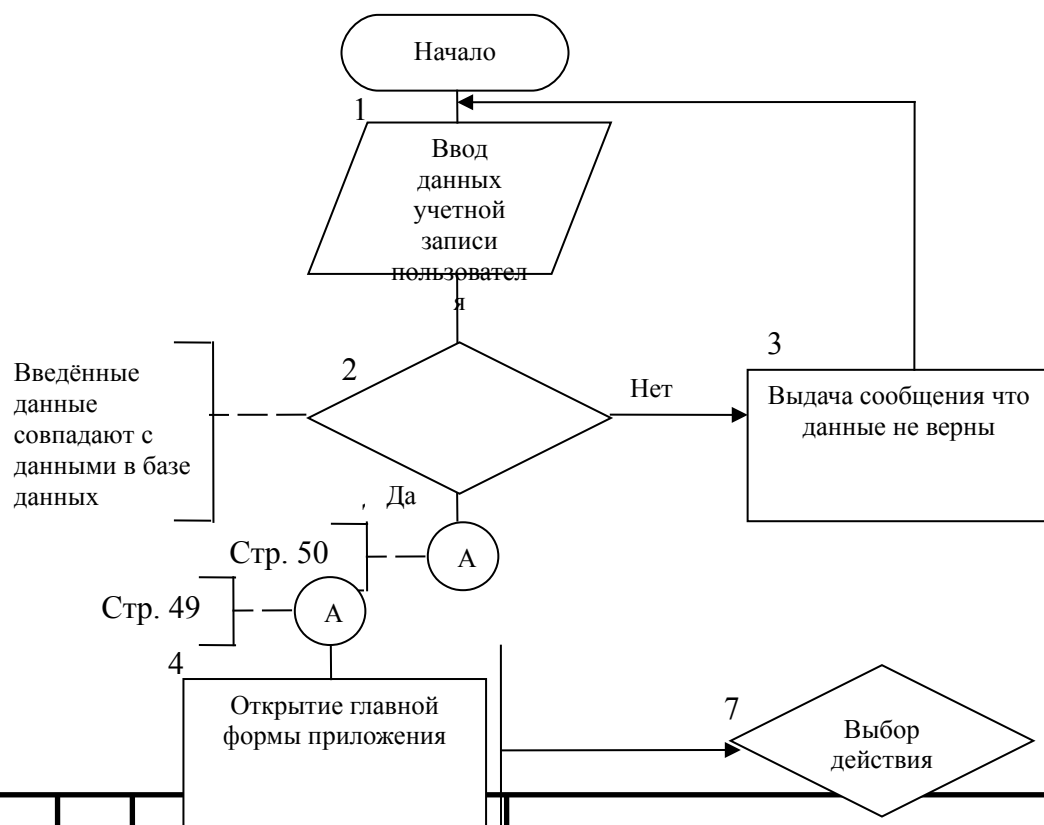
Если учетные данные совпадают с имеющимися в базе данных, то открывается главное окно приложения, иначе выдается сообщение, что данные неверны.

После этого с учетом роли пользователя и разрешений на документы устанавливаются ограничения на использование ресурсов системы (сокрытие

пунктов меню и кнопок на панели инструментов, блокирование функций редактирования, запрет на доступ к документам и т.д.) и появляется главное окно приложения. Через главное меню и панель инструментов в зависимости от назначенной роли можно получить доступ к инструментарию управления справочниками системы, такими как:

- справочник сотрудников;
- справочник пользователей системы;
- справочник подразделений;
- справочник типов;
- справочник должностей;
- справочник стадий проектирования;
- справочник маршрутов;
- справочник воинских званий;
- справочник ролей,

и к управлению документами. Также предусмотрена возможность смены пароля пользователя системы.



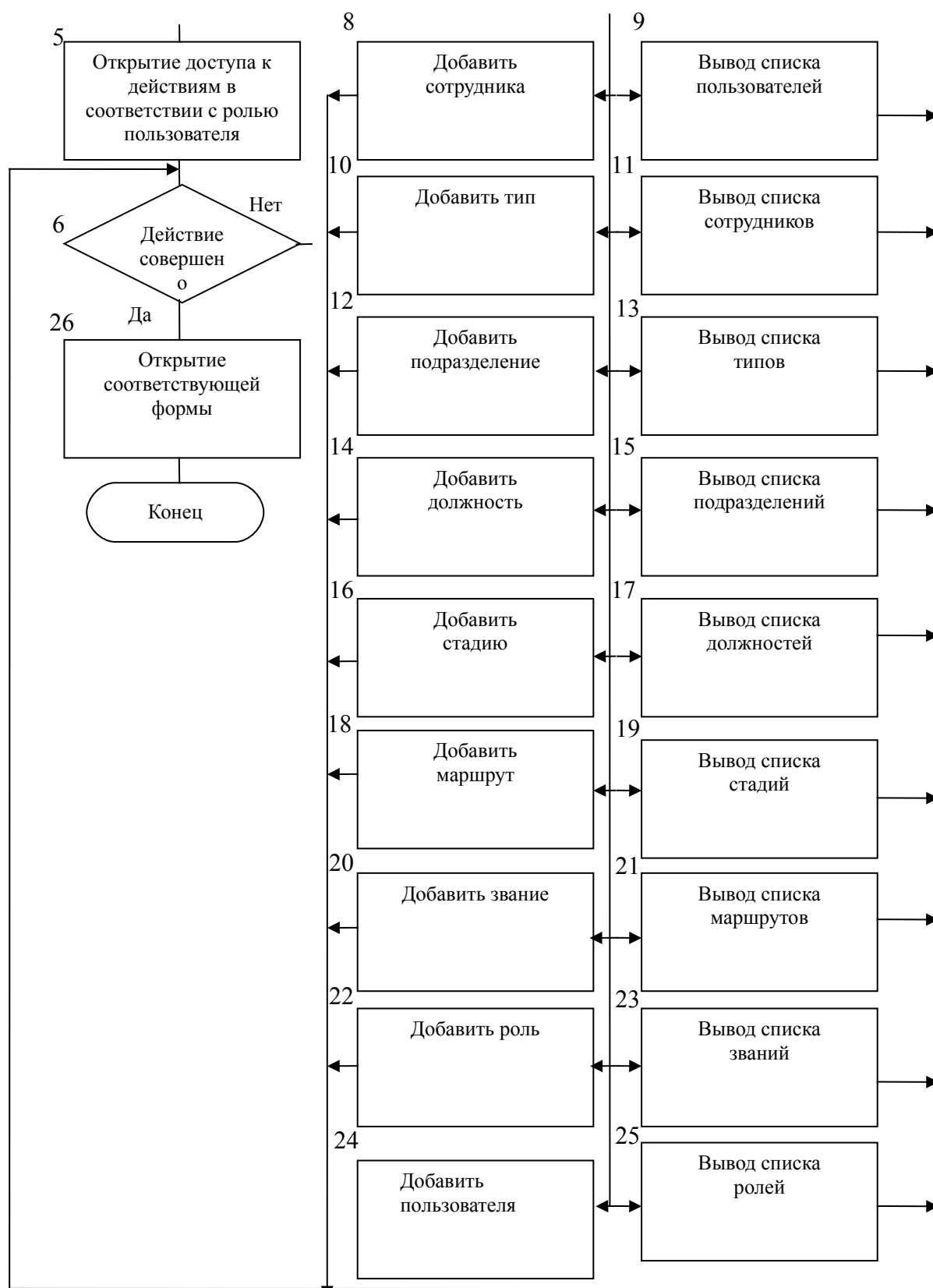


Рисунок 2.3 - Алгоритм работы главного окна

Работа с хранилищем документов начинается с формирования его структуры (создание папок в дереве), далее в нужной папке создаётся

документ выбранного типа. Работа системы построена так, что все изменения в хранилище сразу же распространяются на всех подключенных клиентов без необходимости обновления дерева документов самостоятельно. Документ состоит из двух частей: карточка и файлы. Сначала создаётся карточка, затем к ней присоединяется файл или файлы. При нажатии кнопки сохранить если есть файл, то он сохраняется на сервере, иначе сохраняется только карточка на документ в базе данных. При удалении документа, находящегося не на стадии хранения, он не удаляется физически из системы, а лишь помечается на удаление, и всегда есть возможность восстановить его. То же с файлами. Со стадии хранения документ удаляется безвозвратно.

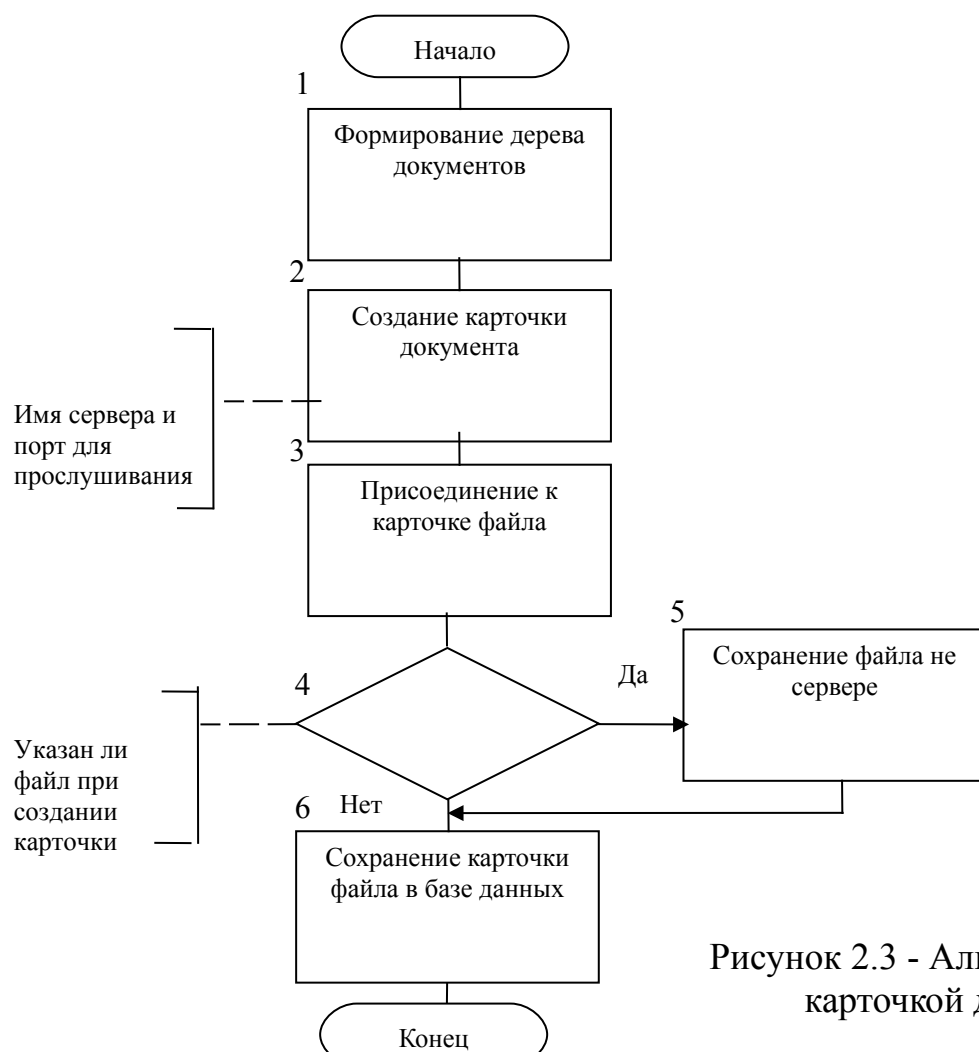


Рисунок 2.3 - Алгоритм работы с карточкой документа

При присоединении или изменении файла он копируется в папку на рабочем месте, предопределённую системой. При сохранении файл копируется в хранилище на файловом сервере. Если файл был изменён, то

создаётся его новая версия, которая становится рабочей. Всегда можно вернуться к любой из версий файла.

При взятии документа на изменение доступ на редактирование для других пользователей блокируется до тех пор, пока пользователь, взявший документ на редактирование, не снимет блокировку. Блокировку можно снять, сохранив документ, или просто разблокировать его без сохранения.

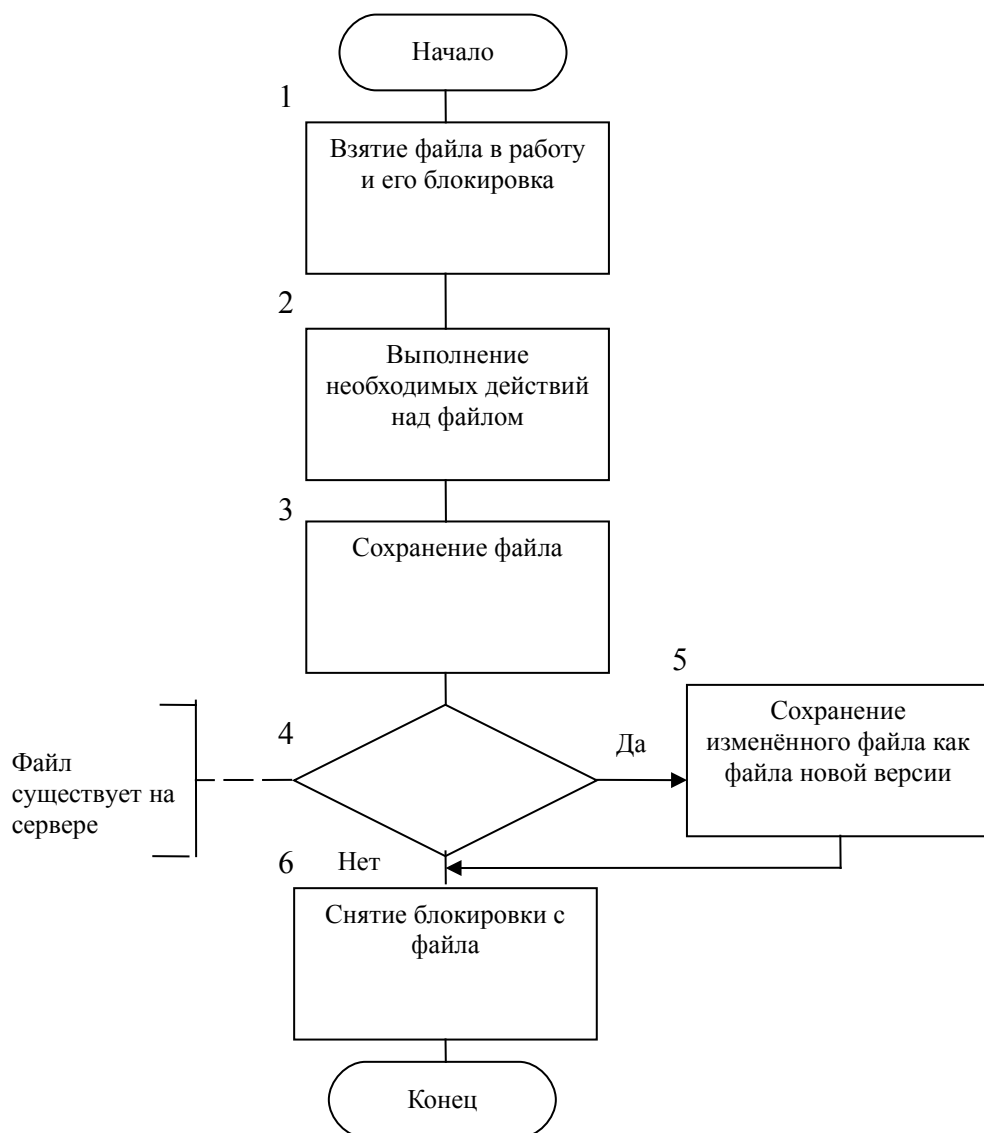


Рисунок 2.4 - Алгоритм изменения документа

При постановке на маршрут и движении его по маршруту автоматически формируются почтовые сообщения. Почтовые сообщения также можно создавать самостоятельно. Причём, отправитель может видеть

прочитано ли сообщение получателем. К почтовым сообщениям можно прикреплять документы.

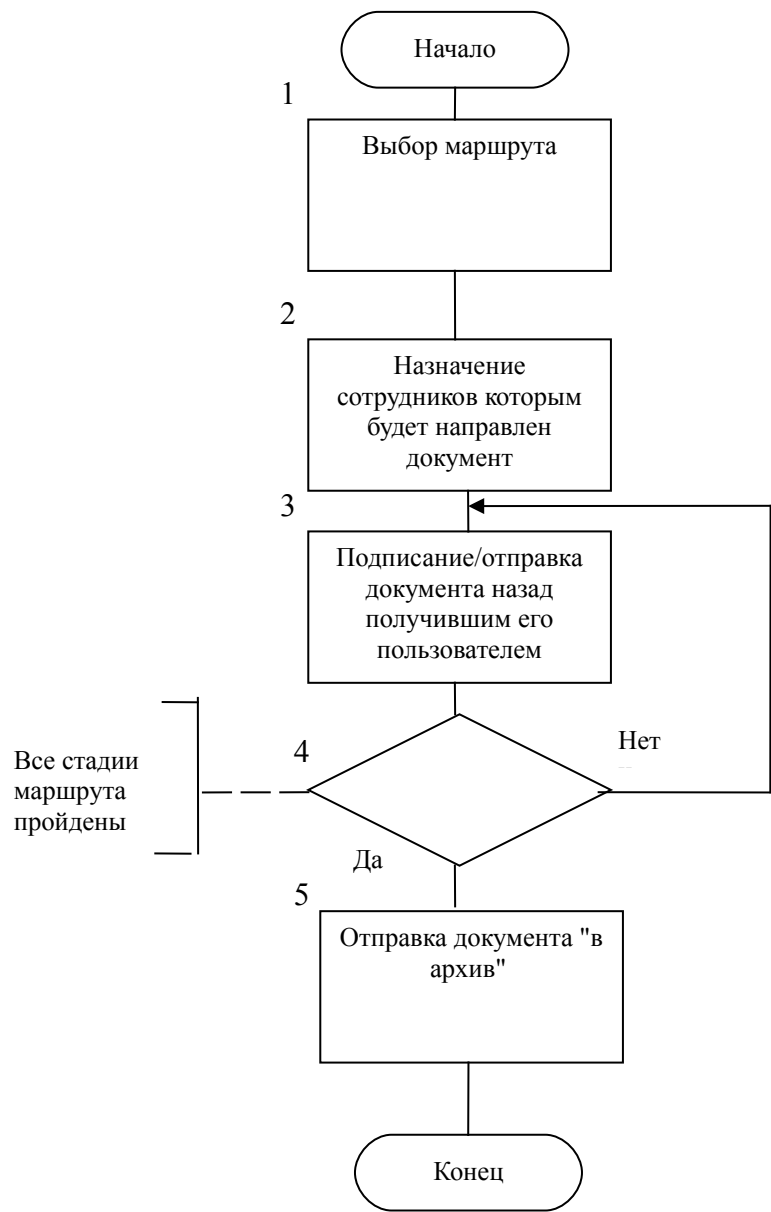


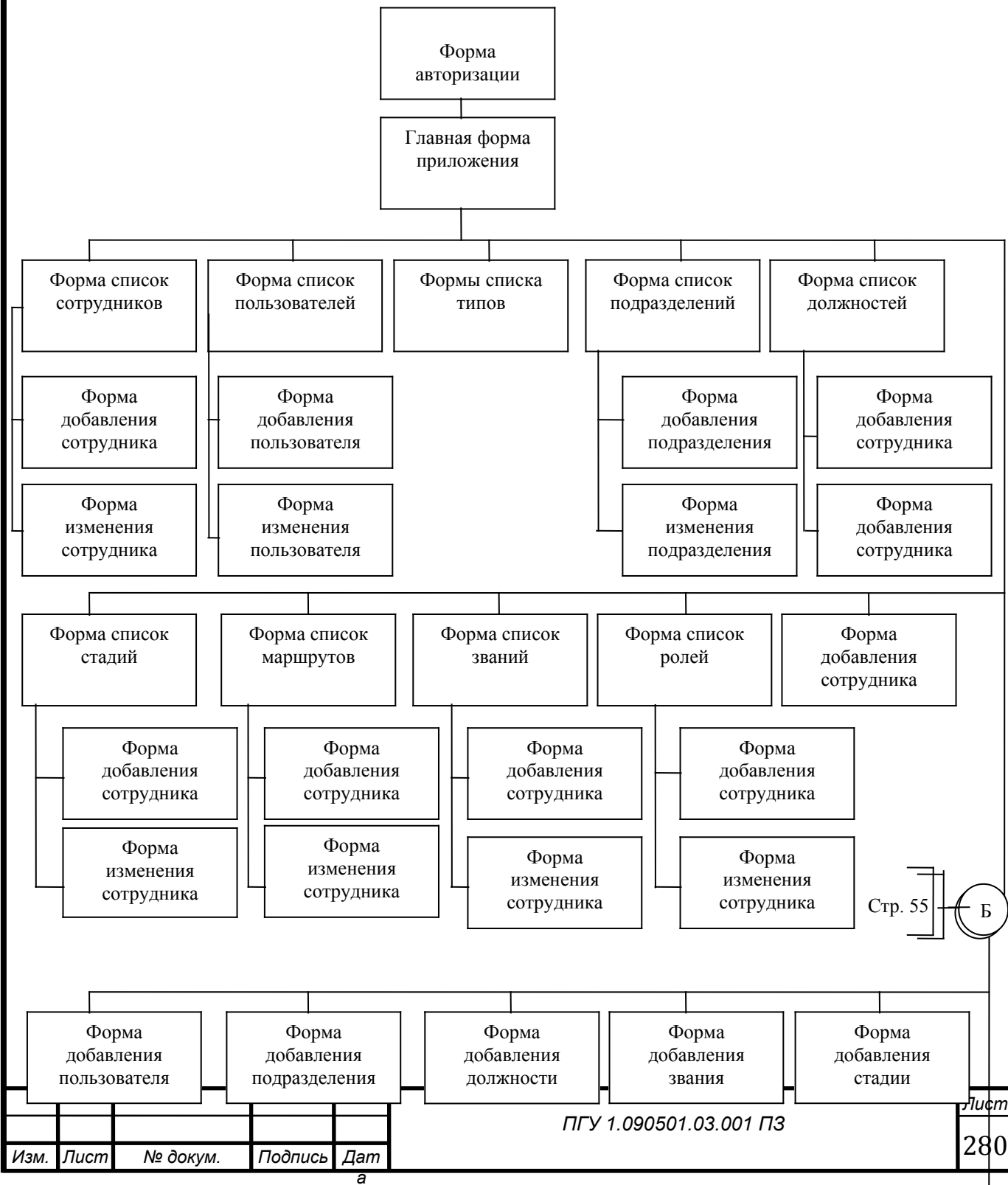
Рисунок 2.5 - Алгоритм работы маршрутов

1.11. Иерархия форм

В данном разделе приводятся схемы иерархии форм приложения с описанием их предназначения. Полное описание форм приведено в

руководстве по эксплуатации приложения. Для удобства отображения дерево форм представлено на трех схемах:

- формы связанные с главным окном приложения (рис. 2.6);
- формы связанные со списком типов (вынесены отдельной схемой, в отличии от остальных форм главного окна) (рис. 2.7);



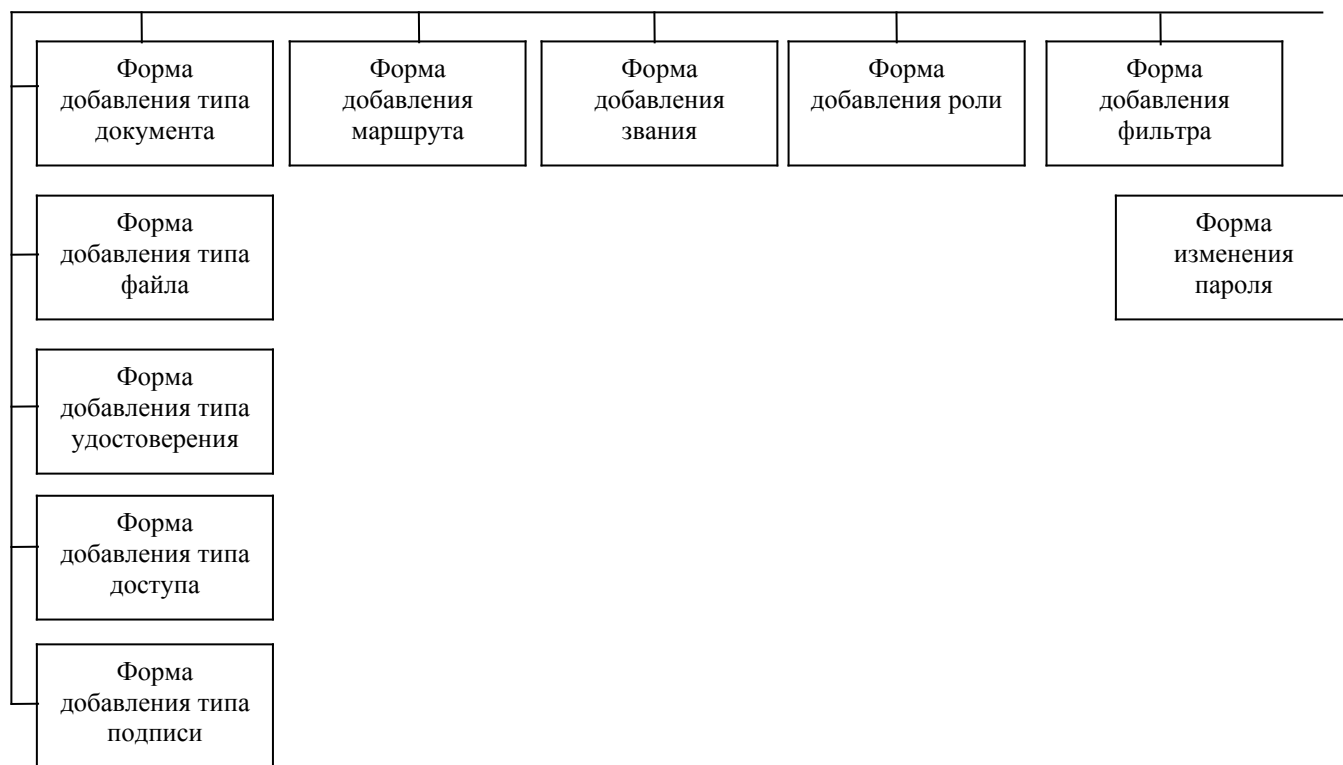


Рисунок 2.6 - Иерархия форм главного окна приложения

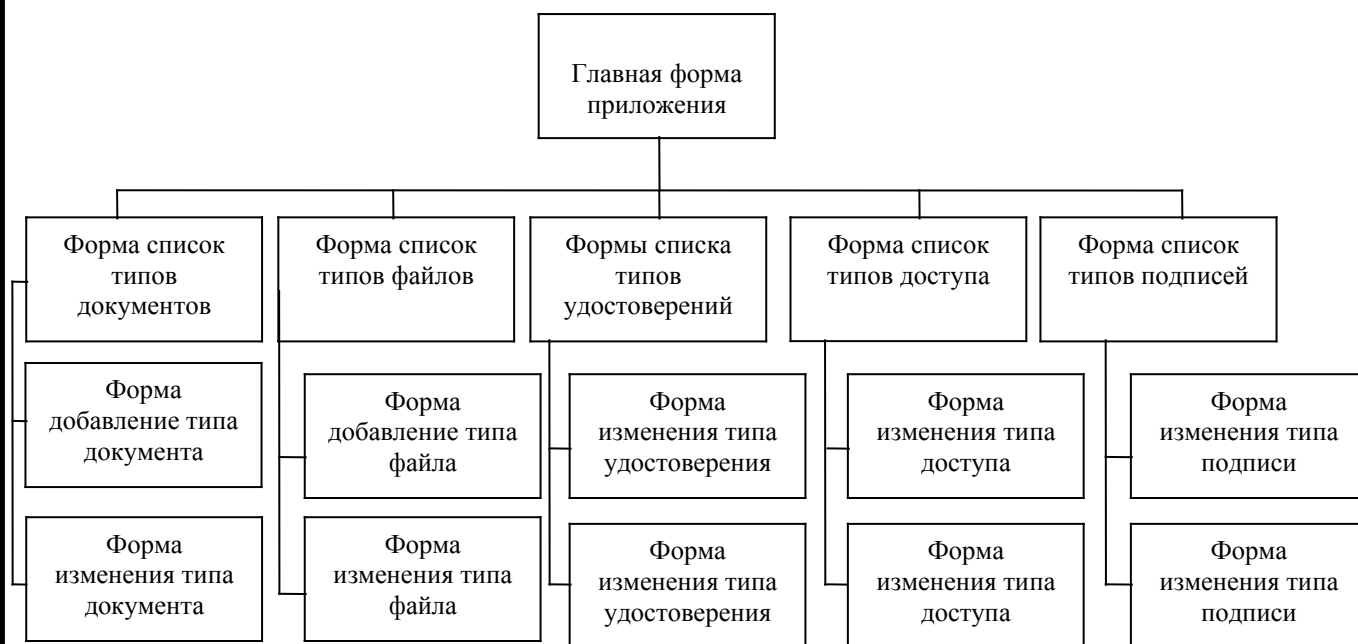


Рисунок 2.7 - Иерархия форм списка типов документов

3. Технология разработки и отладки отчётных форм АСУ

Введение

В данном разделе приводится описание технологий, которые использовались при разработке системы электронного документооборота склада вооружения и военной техники. Одним из важных элементов системы является база данных, поэтому рассмотрены вопросы технологии создания баз данных и их администрирования с помощью системы управления базами данных. В данном разделе рассмотрены основные понятия реляционных баз данных и описаны методы работы с БД при помощи PostgreSQL, а так же, использованного при написании приложения, модуля Windows Communication Foundation.

1.12. Разработка базы данных

1.12.1. Реляционные базы данных

В реляционных базах данных все данные представляются как факты о объектах реального мира в виде сущностей и связей между ними. Например, в проектируемой базе данных для системы электронного документооборота склада вооружения и военной техники содержится информация о сущностях «Сотрудник», «Подразделение», «Документ» и т.д. Между сущностями определяются связи.

Сущность – это, отражение объекта реального мира, например человек, адрес, документы по сделкам. Сущности именуются обычно существительными, такими как «пользователь», «склад», «номенклатура», «товар» и т.д.

Связь – это зависимость между объектами реального мира (сущностями). Например, «сотрудник» состоит в «подразделении».

Кроме связи могут быть вида "один-к-одному", "один-ко-многим", "многие-ко-многим". Что отражает каким образом сущности связаны между

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

собой. Например Между сущностями "сотрудник" и "подразделение", на стороне "сотрудника" установлена связь "один-к-одному", это означает что один сотрудник может состоять в одном подразделении, в свою очередь на стороне "подразделения" связь "один-ко-многим" - в одном подразделении может состоять множество сотрудников.

Каждая сущность обладает атрибутами. Атрибут – это свойство объекта, характеризующее его экземпляр. Например, сущность «Подразделение» имеет атрибуты «Идентификатор подразделения», «Идентификатор родительского подразделения» и «Наименование».

Общепринятым видом графического представления отношений между сущностями и структуры проектируемой базы данных является ER-диаграмма. На такой диаграмме сущности изображаемые прямоугольниками, соединенными между собой связями в виде линий. Такое графическое представление облегчает восприятие структуры базы данных по сравнению с текстовым описанием.

1.12.2. Сущности и атрибуты в реляционной модели

Таблицы в реляционной БД состоят из строк данных, однородных по своей природе. Другими словами, каждая строка таблицы описывает один экземпляр некоторой сущности, причем набор атрибутов каждого экземпляра постоянен.

Для примера приведён фрагмент таблицы, отображающая список с воинскими званиями, таблица содержит 2 колонки:

ID записи Наименование

1 Рядовой

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

2	Ефрейтор
3	Мл. сержант
4	Сержант

Таблицы в реляционной модели соответствуют (не обязательно совпадают по имени) сущностям, а колонки – атрибутам.

1.12.3. Связи в реляционной модели

Если между некоторыми сущностями существует связь, то определённые атрибуты из одной сущности связаны с определёнными атрибутами из другой сущности. Поддержание непротиворечивости функциональных зависимостей между сущностями называется ссылочной целостностью. Поскольку связи содержатся «внутри» реляционной модели, реализация ссылочной целостности может выполняться как приложением, так и самой СУБД (с помощью механизмов декларативной ссылочной целостности – внешних и альтернативных ключей, ограничений, триггеров).

1.12.4. Краткое описание возможностей PostgreSQL

PostgreSQL базируется на языке SQL и поддерживает многие из возможностей стандарта SQL:2011

В PostgreSQL версии 9.5.3 есть следующие параметры:

Максимальный размер базы данных	Нет ограничений
---------------------------------	-----------------

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Максимальный размер таблицы	32 Тбайт
Максимальный размер записи	1,6 Тбайт
Максимальный размер поля	1 Гбайт
Максимум записей в таблице	Нет ограничений
Максимум полей в записи	250—1600, в зависимости от типов полей
Максимум индексов в таблице	Нет ограничений

Функции

Функции представляют собой написанные пользователем программы, исполняемыми на сервере баз данных, а не на клиенте. Они могут быть написаны на SQL, однако реализация дополнительной логики, например, циклов, выходит за рамки возможностей языка SQL и требует использования других языков программирования. Функции могут писаться с использованием одного из следующих языков:

- Встроенный процедурный язык PL/pgSQL,;
- Скриптовые встроенные языки программирования — PL/Lua, PL/LOLCODE, PL/Perl, PL/PHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl, PL/Scheme, PL/v8 (Javascript);
- Классические языки — C, C++, Java (через модуль PL/Java);
- Статистический язык R (через модуль PL/R).

Триггеры

Триггеры определяются как функции, инициируемые операциями языка SQL. Например, операция INSERT может запускать триггер, который будет

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

проверять, добавляемую запись, на соответствие, определённым в триггере, условиям.

Индексы

В PostgreSQL имеется поддержка индексов следующих типов: В-дерево, хеш, GiST, GIN, BRIN, Bloom. Однако, если необходимо, администратор базы данных может создавать типы индексов самостоятельно

Индексы в PostgreSQL обладают следующими свойствами:

- просмотр индекса не только в прямом, но и в обратном порядке
- возможно создание индекса над несколькими столбцами таблицы, в том числе над столбцами различных типов данных;
- индексы могут быть функциональными, то есть строиться не на базе набора значений некоего столбца/столбцов, а на базе набора значений функции от набора значений;
- индексы могут быть частичными, то есть строиться только по части таблицы (по некоторой её проекции); в некоторых случаях это помогает создавать намного более компактные индексы или достигать улучшения производительности за счёт использования разных типов индексов для разных (например, с точки зрения частоты обновления) частей таблицы;
- планировщик запросов может использовать несколько индексов одновременно для выполнения сложных запросов.

Многоверсионность (MVCC)

PostgreSQL поддерживает внесение изменений в базу данных одновременно несколькими пользователями с помощью механизма Multiversion Concurrency Control (MVCC). Благодаря этому практически отпадает нужда в блокировках чтения.

Типы данных

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

PostgreSQL поддерживает следующий набор встроенных типов данных:

- Численные типы
 - Целые
 - С фиксированной точкой
 - С плавающей точкой
 - Денежный тип (отличается только форматом вывода, а в остальном идентичен типу с фиксированной точкой с двумя знаками после запятой)
- Символьные типы произвольной длины
- Двоичные типы (включая BLOB)
- Типы «дата/время» (полностью поддерживающие различные форматы, точность, форматы вывода, включая последние изменения в часовых поясах)
- Булев тип (Логический тип)
- Перечисление
- Геометрические примитивы
- Сетевые типы
 - IP и IPv6-адреса
 - CIDR-формат
 - MAC-адрес
- UUID-идентификатор
- XML-данные
- Массивы
- JSON
- Идентификаторы объектов БД
- Псевдотипы

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

Кроме того, пользователь может создавать новые необходимые ему типы и программировать для них механизмы индексирования.

Пользовательские объекты

PostgreSQL может быть расширен пользователем для собственных нужд практически в любом аспекте. Есть возможность добавлять собственные:

- Преобразования типов
- Типы данных
- Домены (пользовательские типы с изначально наложенными ограничениями)
- Функции (включая агрегатные)
- Индексы
- Операторы (включая переопределение уже существующих)
- Процедурные языки

Наследование

Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.

1.13. Описание модуля Windows Communication Foundation

Windows Communication Foundation (WCF) — это платформа для построения сервисноориентированных приложений. С помощью WCF, можно отправлять данные как асинхронные сообщения из одной конечной точки службы в другую. Конечная точка службы может входить в постоянно

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

доступную службу, размещаемую в IIS, или представлять службу, размещаемую в приложении. Конечная точка может быть клиентом службы, которая запрашивает данные от конечной точки службы. Сообщения могут представлять одиночный символ или одно слово, отправляемое в формате XML, или иметь вид сложного потока двоичных данных. Далее представлено несколько образцов сценариев.

- Защищенная служба для обработки бизнес-транзакций.
- Служба, передающая другим объектам текущие данные, такие как отчет о трафике, или другая служба наблюдения.
- Служба бесед, которая позволяет двум пользователям общаться и обмениваться данными в реальном времени.
- Приложение панели мониторинга, которая опрашивает одну или несколько служб и дает логическое представление полученных данных.
- Предоставление доступа к рабочему процессу, реализованному с помощью Windows Workflow Foundation, в виде службы WCF.
- Приложение Silverlight для запроса последних каналов данных в службе.

При создании таких приложений до появления WCF, WCF делает разработку конечных точек проще, чем когда-либо. Таким образом WCF позволяет предлагает удобный подход к созданию веб-служб и клиентов веб-служб.

Возможности WCF

WCF включает следующий набор функций. Дополнительные сведения см. в разделе подробные сведения о возможностях WCF.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

- Сервис-ориентированность

Следствием применение стандартов WS является то, что WCF позволяет создавать ориентированные на службы приложений. Сервисноориентированная архитектура (SOA) подразумевает применение веб-служб для отправки и получения данных. Общим преимуществом служб является слабая связанность вместо жесткой запрограммированности для различных приложений. Слабая связь означает, что любой клиент, созданный на любой платформе, может подключаться к любой службе при условии, что выполняются необходимые контракты.

- Взаимодействие

WCF реализует современные отраслевые стандарты для веб-службы взаимодействие. Дополнительные сведения о поддерживаемых стандартах см. в разделе взаимодействие и интеграция.

- Несколько шаблонов сообщений

Обмен сообщениями выполняется по одному из нескольких шаблонов. Чаще всего используется шаблон «запрос-ответ», когда одна конечная точка запрашивает данные от другой конечной точки. Вторая конечная точка отвечает. Существуют и другие шаблоны, например одностороннее сообщение, когда одна конечная точка отправляет сообщение, не ожидая ответа. Более сложным является шаблон дуплексного обмена, когда две конечные точки устанавливают соединение и отправляют данные в обоих направлениях подобно программе обмена мгновенными сообщениями. Дополнительные сведения о реализации обмена сообщениями различных шаблонов с помощью WCF см. в разделе контракты.

- Метаданные службы

WCF поддерживает публикация метаданных службы, с использованием форматов, указанных в отраслевых стандартах, таких как WSDL, схемы XML и WS-Policy. Эти метаданные можно использовать для автоматического создания и настройки клиентов для доступа к службам WCF. Метаданные могут публиковаться через HTTP и HTTPS или с использованием стандарта обмена метаданными веб-служб. Дополнительные сведения см. в разделе метаданных.

- Контракты данных

Поскольку WCF построена с использованием .NET Framework, он также включает код с поддержкой методы передачи контрактов, вы хотите применить. Одним из универсальных типов контрактов является контракт данных. Если код службы создается на языке Visual C# или Visual Basic, то самым простым способом обработки данных фактически является создание классов, которые представляют сущность данных со свойствами, принадлежащими сущности данных. WCF включает сложную систему для работы с данными этим удобным способом. После создания классов, представляющих данные, служба автоматически создает метаданные, которые позволяют клиентам обеспечивать соответствие заданным типам данных. Дополнительные сведения см. в разделе Using Data Contracts

- Безопасность

Сообщения можно шифровать для защиты конфиденциальности и требовать от пользователей проходить проверку подлинности перед приемом сообщений. Можно реализовать широко известные стандарты безопасности, такие как SSL и WS-SecureConversation. Дополнительные сведения см. в разделе Безопасность.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

- Несколько транспортов и кодировок

Сообщения могут отправляться по любому из нескольких встроенных транспортных протоколов в различных кодировках. Наиболее распространенным является отправка текст в кодировке сообщения SOAP при помощи протокола передачи гипертекста (HTTP) для использования в Интернете. Кроме того WCF позволяет отправлять сообщения по протоколу TCP, именованные каналы или MSMQ. Сообщения можно кодировать в виде текста или использовать оптимизированный двоичный формат. Двоичные данные можно эффективно отправлять с использованием стандарта MTOM. Если ни один из предоставляемых транспортов и кодировок не подходит к текущим требованиям, вы можете создать собственный пользовательский транспорт или кодировку. Дополнительные сведения о транспорте и кодировках, поддерживаемых WCF см. в разделе транспортов.

- Надежные сообщения и сообщения в очереди

WCF поддерживает надежный обмен сообщениями с использованием надежных сеансов, реализованных на WS-Reliable Messaging и с помощью MSMQ. Дополнительные сведения о надежной поддержке обмена сообщениями в WCF см. в разделе очереди и надежные сеансы.

- Устойчивые сообщения

Устойчивые сообщения не теряются в случае перебоев связи. Сообщения, передаваемые по устойчивому шаблону, всегда сохраняются в базе данных. Если происходит перебой связи, база данных позволяет возобновить обмен сообщениями после восстановления соединения. Можно также создать постоянное сообщение с помощью

Windows Workflow Foundation (WF).Дополнительные сведения см. в разделе служб рабочих процессов.

- Транзакции

WCF также поддерживает транзакции, с помощью одного из трех моделей: WS-AtomicTransactions, API-интерфейсы в System.Transactions пространства имен и координатор распределенных транзакций. Дополнительные сведения о транзакции поддержки в WCF см. в разделе транзакции.

- Поддержка AJAX и REST

REST - это пример развития технологии Web 2.0. WCF можно настроить для обработки «обычный» XML-данных, не упакованных в конверт SOAP. Также можно расширить WCF для поддержки определенных форматов XML, таких как ATOM (распространенный стандарт RSS) и даже не XML-форматы, такие как JavaScript Object Notation (JSON).

- Расширение среды

Архитектура WCF состоит множество точек расширения. Если требуются дополнительные возможности, поддерживаются точки входа, посредством которых можно настроить поведение службы. Дополнительные сведения о расширяемости точек см. в разделе расширение WCF.

1.14. Методика испытаний

Тестирование является одним из наиболее важных элементов обеспечения качества программного обеспечения. Многие фирмы-разработчики программных продуктов проводят тестирование своих

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

приложений в несколько этапов, которые отличаются видами проводимых работ и привлекаемыми ресурсами.

Тестирование проводилось методом «белого ящика»

Тестирование методом «белого ящика» – это тестирование на этапе реализации.

Тестирование методом белого ящика – метод тестирования программного обеспечения, который предполагает, что внутренняя устройство системы известны лицу проводящему тестирование системы. Выбираются входные значения, основываясь на знании содержания кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники.

Далее приведён фрагмент протокола тестирования.

№ теста	Описание теста	Дата	Результат тестирования	Описание ошибки или примечание	Исправление ошибки
1.1	Запуск программы под учётной записью администратора	10.05.19	Выводится основное окно приложения с доступом ко всем действиям и отображением дерева документов.		
1.2	Использование команды Действие>Создать>Сотрудник, соответствующей кнопки на панели инструментов	17.04.05	Активизируется диалоговое окно “Сотрудник:Новый сотрудник”.	Ошибка в названии окна "СотрудниК:Новый сотрудник" Отсутствует буква "к" на конце слова сотрудник	Исправлено. Внесены изменения в форму
1.3	Использование	29.04.05	Выводится окно	.	.

№ теста	Описание теста	Дата	Результат тестирования	Описание ошибки или примечание	Исправление ошибки
	команды Списки > Список воинских званий соответствующ ей кнопки на панели инструментов		содержащее список воинских званий.		
1.4	Использование команды Добавить на панели инструментов окна "список воинских званий"	29.04.05	Выводится окно добавления нового пользователя, после введения данных и подтверждения данные заносятся в список.		
1.5	Использование команды создать документ>по типу> приказ	29.04.05	Открывается окно создания карточки документа.	Отсутствует кнопка добавления документа при создании карточки.	Исправлено. Была добавлена кнопка в окне создания карточки и внесены изменения в код.
1.6	Постановка документа на маршрут через контекстное меню	29.04.05	Открывается окно постановки на маршрут, выбирается маршрут и сотрудник у документа изменяется стадия на стадию указанному в маршруте у пользователя которому отправлен документ появляется почтовое сообщение о приходе документа после прохождения		

№ теста	Описание теста	Дата	Результат тестирования	Описание ошибки или примечание	Исправление ошибки
			всех стадий и подписания всеми назначенными пользователями документ блокируется.		

Все ошибки и недочёты, выявленные в процессе реальной эксплуатации, были успешно локализованы и устранены.

2. Психофизиологические факторы, воздействующие на оператора

Введение

Персональные компьютеры (ПК) являются сложной по составу техникой и, требуют при работе с ними соблюдения определенных правил безопасности, незнание или пренебрежение, которыми может причинить урон здоровью оператора.

Однако, большая часть этих проблем решаются при правильной

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

организации рабочего места, соблюдении правил техники безопасности и разумном распределении рабочего времени.

В данном разделе приводятся рекомендации по правильной организации рабочего места оператора, а также основные методы уменьшения вредных факторов воздействия компьютера на человека.

1.15. Аспекты производственной безопасности при работе на ПК

Имеющийся в настоящее время комплекс разработанных организационных мероприятий и технических средств защиты, позволяет значительно понизить воздействие вредных и опасных факторов на оператора ПК.

Опасным считается производственный фактор, воздействие которого на сотрудника при нарушении техники безопасности или правил эксплуатации может привести к травме или другому внезапному резкому ухудшению здоровья. Если же производственный фактор приводит к заболеванию или снижению трудоспособности, то его считают вредным. В зависимости от уровня и продолжительности воздействия вредный производственный фактор может стать опасным. Опасные и вредные производственные факторы подразделяются на четыре группы: физические, химические, биологические и психофизиологические.

1.15.1. Психофизиологические факторы

Психофизиологические факторы в зависимости от характера действия делятся на следующие группы: физические перегрузки (статические, динамические), умственные перегрузки, монотонность труда и эмоциональные перегрузки.

Монотонность - психическое состояние человека, вызванное однообразием выполняемых им действий. Монотонность труда характерна

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

для работы сотрудника, которая связана с обработкой однообразной информации. Наибольшая вероятность появления стрессового состояния у сотрудника может наблюдаться при выполнении монотонной работы, не дающей сразу некого положительного отклика и требующее постоянной концентрации при работе. Примером такой работы может являться работа, связанная с большим количеством документов. При этом риск возникновения стресса возрастает, если работа связана с постоянными отвлекающими либо раздражающими внешними факторами.

Для предупреждения возникновения рабочего стресса рекомендуется соблюдать следующие правила:

- суммарное время работы за вычислительной машиной не должно превышать 50% всего рабочего времени оператора;
- при обычной работе за компьютером необходимо делать 15-минутные перерывы через каждые два часа, а при интенсивной работе - через каждый час;
- не следует превышать темп работы порядка 10 тысяч нажатий клавиш в час (примерно 1500 слов);
- предпочтительнее использовать жидкокристаллические дисплеи с высокой разрешающей способностью (разрешением) и удобным размером экрана;
- лучше выбирать видеоадаптеры с высоким разрешением и, по возможности, частотой обновления экранного изображения не менее 70-72 Гц (рекомендуется 85 или 100 Гц);
- обязательно ставить на дисплеи экранные, в частности, поляризационные, фильтры, в несколько раз снижающие утомляемость глаз;

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

- наконец, при вводе данных с клавиатуры рекомендуется не зажимать телефонную трубку между плечом и ухом.

Правильное расположение рабочего и его поза оказывает значительное влияние на эффективность работы человека. Основные требования к рабочим местам при выполнении сидячей работы приведены в ГОСТ 12.2.032-78 "ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования". Рабочее место должно быть оборудовано так, чтобы исключать неудобные позы и длительные статические напряжения тела. Комфортабельным может быть рабочее место, которое можно приспособить, как минимум, для двух позиций (при этом положение оборудования должно соответствовать выполняемой работе и привычкам пользователя).

К обслуживанию и работе на ПК допускаются лица прошедшие медосмотр при поступлении на работу. Последующий медосмотр проводится раз в два года. Также необходимо соблюдать ограничения на работу с персональными компьютерами для служащих, страдающих заболеваниями опорно-двигательного аппарата, глаз (или нарушениями зрения), кожи.

1.15.2. Оборудование рабочих мест с ПК

Требования к организации и оборудованию рабочего места оператора приведены в ГОСТ 12.2.032-78. Высота рабочей поверхности стола для пользователей должна регулироваться в пределах 680-800 мм; при отсутствии таковой возможности высота рабочей поверхности стола должна составлять 725 мм.

Модульными размерами рабочей поверхности стола для ПК, на основании которых должны рассчитываться конструктивные размеры, следует считать: ширину 800, 1200, 1400 мм, глубину 800 и 1000 мм при перегулируемой высоте, равной 725 мм.

					ПГУ 1.090501.03.001 ПЗ		Лист
							280
Изм.	Лист	№ докум.	Подпись	Дат			

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также – расстоянию спинки до переднего края сиденья.

Рабочее место необходимо оборудовать подставкой для ног, имеющей ширину не менее 300 мм, глубину не менее 400 мм, регулировку по высоте в пределах до 150 мм и по углу наклона опорной поверхности подставки до 20 градусов. Поверхность подставки должна быть рифленой и иметь по переднему краю бортик высотой 10 мм.

Клавиатуру следует располагать на поверхности стола на расстоянии 100-300 мм от края, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

Помимо всего манипулятор(компьютерная мышь) и клавиатура должны быть эргономичными, во избежание возникновения у оператора заболеваний связанных с кистями рук, как например т.н. туннельный синдром.

1.15.3. Освещение рабочего места

. Освещенность рабочего места не должна быть очень яркой и должна быть меньше, чем при работе с бумагами. При одновременной работе с бумагами и экраном необходимо использовать настольную лампу для освещения документов, с которыми работает оператор. При этом свет от настольной лампы не должен попадать в глаза и на экран, во избежание раздражающих для глаз оператора факторов (прямой свет, блики). Рядом с монитором нельзя располагать глянцевые предметы либо поверхности, а также иные поверхности хорошо отражающие свет.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Во избежание бликов экран монитора нужно располагать перпендикулярно к окну. Не следует располагать монитор на фоне яркого окна, во избежание снижения нагрузки на глаза оператора. Если окно будет расположено за спиной оператора, изображение на мониторе будет выглядеть блеклым, что также потребует дополнительного зрительного напряжения. Наилучшим считается, когда окно расположено слева. При этом требуется иметь на окнах жалюзи или шторы, для регуляции освещения.

Освещенность должна соответствовать характеру труда, который определяется объектом различия, фоном, контрастом объекта с фоном.

Необходимо обеспечить достаточно равномерное распределение яркости на рабочей поверхности, а также в пределах окружающего пространства. Светлая окраска потолка, стен и производственного оборудования способствует созданию равномерного распределения яркости в поле зрения.

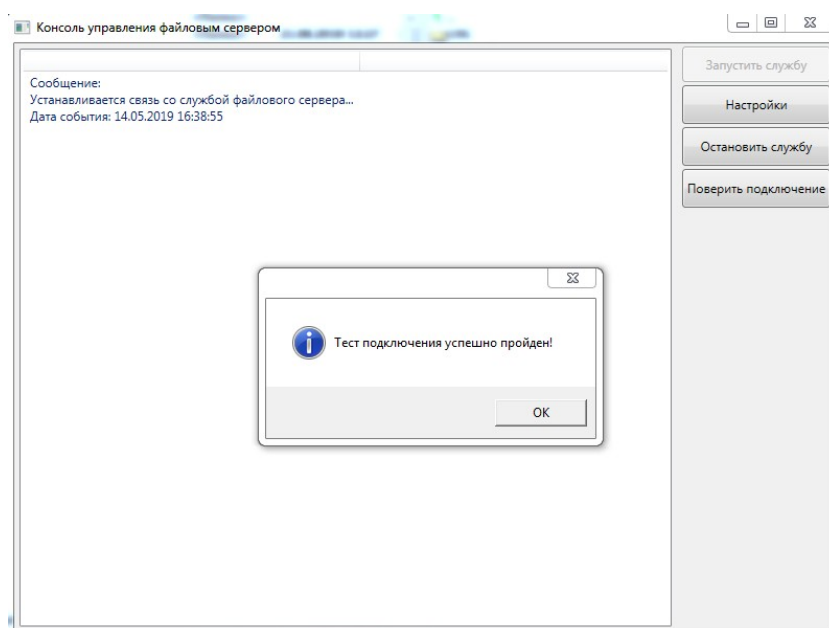
Величина освещенности должна быть постоянной во времени. Колебания освещенности, вызванные резким изменением напряжения в сети, приводят к значительному утомлению. Следует выбирать необходимый состав спектра освещения. Это существенно при работах, где требуется правильная цветопередача. Все элементы осветительных установок должны быть достаточно долговечными, электро- и взрывобезопасными.

Свечение со стороны экрана, а также частая смена заставок на экране при большой продолжительности трудовой деятельности может отрицательно воздействовать на зрение. Такой режим работы утомляет зрительные органы, и это необходимо учитывать при работе за компьютером.

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

3. Руководство пользователя по эксплуатации

Перед запуском программы необходимо запустить службы файлового сервера и сервера баз данных через их консоли управления FS.Console.exe и Doc.Console.exe, соответственно, а так же проверить подключение(рисунки 6.1 и 6.2)



					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Рисунок 6.1 - Окно консоли управления файловым сервером

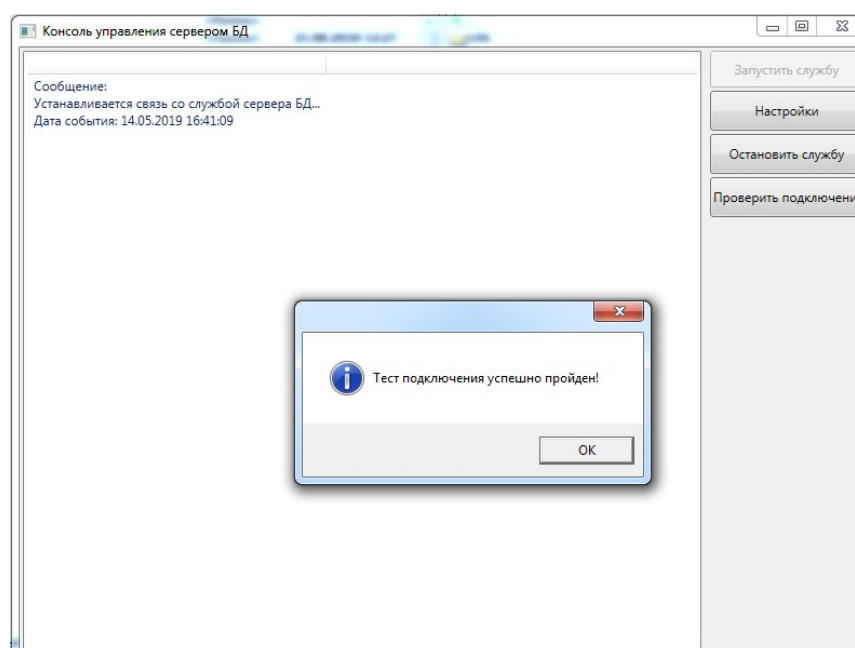


Рисунок 6.2 - Окно консоли управления сервером баз данных

После этого необходимо запустить основное приложение системы электронного документооборота, приложение Doc.App.exe. Сначала появится форма авторизации, в которой необходимо ввести логин и пароль для того чтобы начать работу (рисунок 6.3). После этого, в соответствии с допуском появляются кнопки и действия на панели инструментов главного окна (рисунок 6.4).

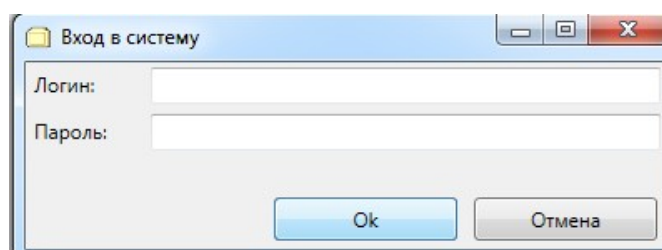


Рисунок 6.3 - Форма авторизации

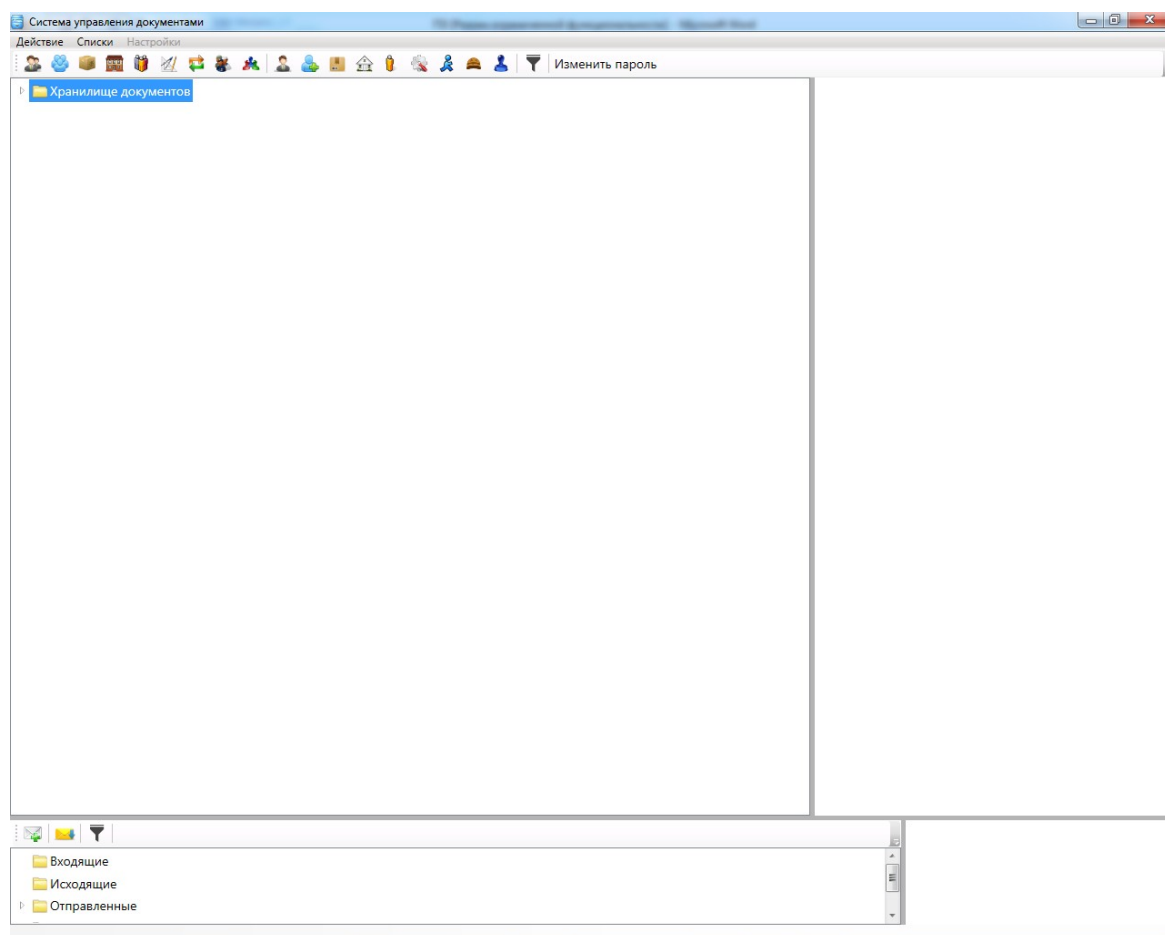


Рисунок 6.4 - Форма главного окна приложения

Так как, вход был осуществлён под именем администратора, на форме отображается полный набор доступных действий и функций. В верхней части окна располагаются выпадающие меню "Действия" (рисунок 6.5) и "Списки" (рисунок 6.6)

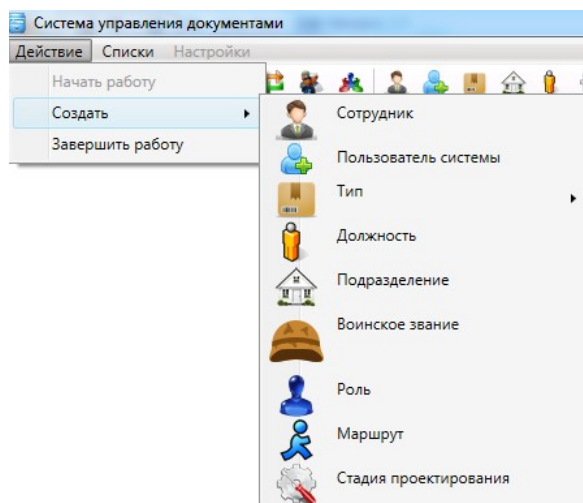


Рисунок 6.5 - Выпадающее меню "Действия"

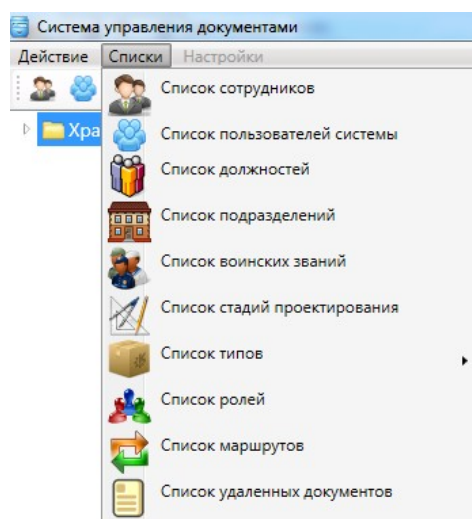


Рисунок 6.6 - Выпадающее меню "Списки"

Аналогичные действия можно выполнить, используя кнопки с панели инструментов в верхней части окна. Помимо этого на панели инструментов есть кнопка, позволяющая пользователю сменить пароль, а так же кнопка фильтров.

1.16. Меню "Действия"

Меню "Действия" содержит кнопки "Начать работу", по нажатии на которую открывается форма авторизации, для получения доступа к работе с документами, "Завершить работу", завершающая работу текущего пользователя, а так же выпадающее меню "Создать" содержащее кнопки для создания "Сотрудник", "Пользователь", "Тип", "Должность", "Подразделения", "Воинское звание", "Роль", "Маршрут", "Стадия проектирования". Каждый элемент выпадающего меню соответствует аналогичной кнопке в правой части (после разделителя) панели инструментов.

1.16.1. Действие создания сотрудника

При выборе действия Создать > Сотрудник открывается форма для создания сотрудника(рисунок 6.7).

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Рисунок 6.7 - Форма создания нового сотрудника

В верхней части формы, на панели инструментов располагаются кнопки для добавления фотографии, удаления фотографии и удаления сотрудника.

При нажатии на кнопку добавления фотографии открывается диалоговое окно через которое необходимо выбрать загружаемый файл.

Форма заполняется необходимыми данными, после чего необходимо нажать на кнопку сохранить после чего информация добавляется в базу данных. Информацию о сотрудниках можно просмотреть в меню списки.

1.16.2. Действие создания пользователя

При выборе действия Создать > Пользователь открывается форма для создания сотрудника(рисунок 6.8).

Пользователь

Логин:

Пароль:

Подтверждение:

Сотрудник:

Роль:

Ok Сохранить Отмена

Рисунок 6.8 Окно создания сотрудника

В открывшейся форме необходимо ввести данные нового пользователя, а именно его логин и пароль, а так же выбрать сотрудника и его роль из выпадающего списка. После этого необходимо нажать на кнопку "Ok" либо сохранить. После этого данные о сотруднике будут доступны в соответствующем списке.

1.16.3. Действия создания типа

При выборе пункта меню "Тип" выпадает меню, содержащее список уточняющий для чего создается тип (рисунок 6.9).

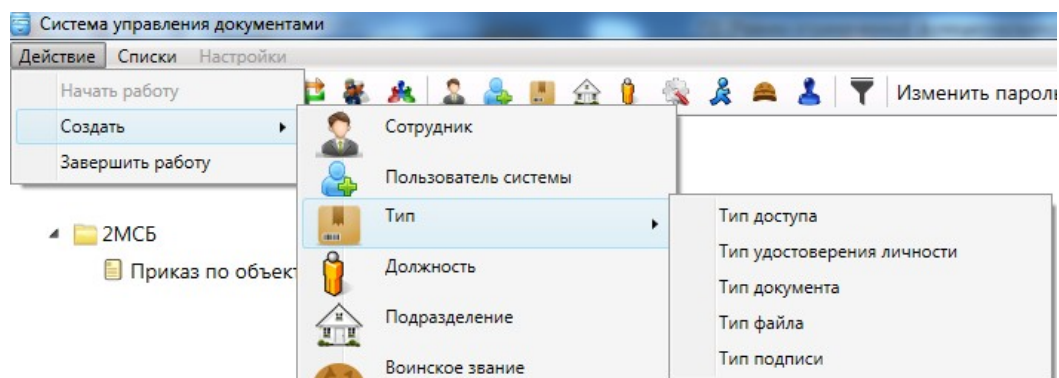


Рисунок 6.9 - Выпадающее меню тип

Все формы имеют идентичный вид и отличаются только названием, в форме присутствует строка для ввода наименования типа и кнопки "Сохранить" и "Отмена"(рисунок 6.10).

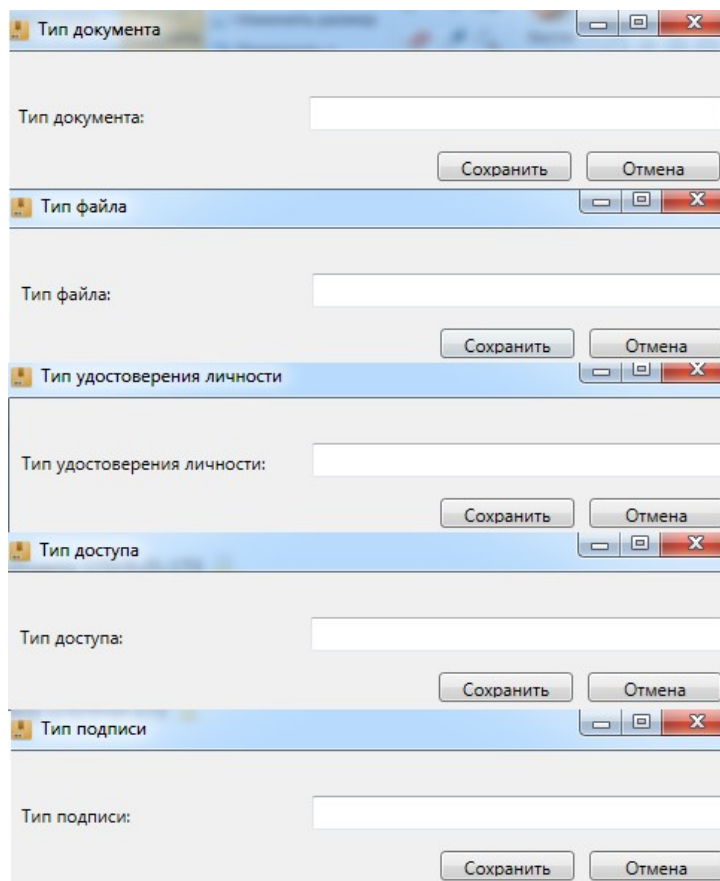


Рисунок 6.10 - Внешний вид окон добавления типа

1.16.4. Действие создания подразделения

При выборе действия Создать > Подразделение, появляется форма для создания подразделения(рисунок 6.11), содержащая строку для наименования подразделения, кнопки "Ok", "Сохранить", "Отмена", а так же выпадающий список для выбора родительского подразделения, который берётся из базы данных. После подтверждения, введённые данные добавляются в соответствующий список, а так же в базу данных.

Рисунок 6.11 - Форма добавления подразделения

1.16.5. Действие создания должности

При выборе действия Создать > Должность, появляется форма для создания должности (рисунок 6.12), содержащая строку для наименования должности, кнопки "Ok", "Сохранить", "Отмена". После подтверждения, введённые данные добавляются в соответствующий список, а так же в базу данных.

Рисунок 6.12 - Форма добавления должности

1.16.6. Действие создания воинского звания

При выборе действия Создать > Воинское звание, появляется форма для создания воинского звания (рисунок 6.13), содержащая строку для наименования воинского звания, кнопки "Ok", "Сохранить", "Отмена". После подтверждения, введённые данные добавляются в соответствующий список, а так же в базу данных.

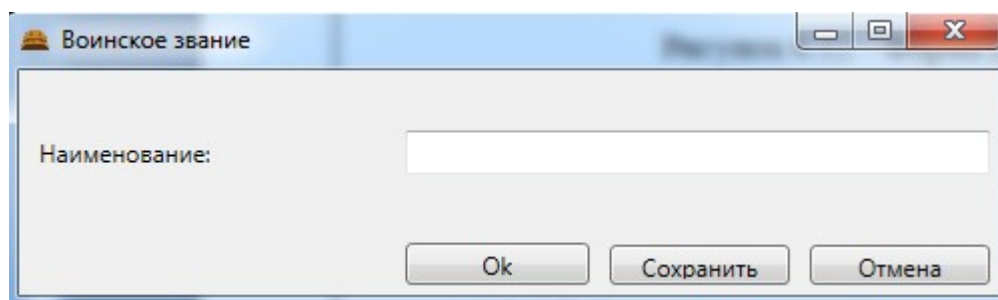


Рисунок 6.13 - Форма добавления звания

1.16.7. Действие создания роли

При выборе действия Создать > Воинское звание, появляется форма для создания роли (рисунок 6.14). На форме в верхней части расположена строка для ввода наименования. Ниже следует список доступных для создаваемой роли действий, а также кнопки "Ok", "Сохранить", "Отмена". После подтверждения, введенные данные добавляются в соответствующий список, а так же в базу данных.

Роль

Наименование:

- ☐ Управление сотрудниками
- ☐ Управление пользователями системы
- ☐ Управление документами
- ☐ Управление типами
- ☐ Управление группами типов
- ☐ Управление ролями
- ☐ Управление маршрутами
- ☐ Управление стадиями проектирования
- ☐ Управление доступом к объектам
- ☐ Управление должностями
- ☐ Управление воинскими званиями
- ☐ Управление подразделениями
- ☐ Управление блокировкой документа
- ☐ Просмотр удаленных документов
- ☐ Удаление документов из архива

Ok Сохранить Отмена

Рисунок 6.14 - Форма создания роли

1.16.8. Действие создания маршрута

При выборе действия Создать > маршрут, появляется форма для создания маршрута (рисунок 6.15). На форме в нижней части расположена строка для ввода наименования. Форма разбита на две части. С левой стороны находится список стадий доступных для добавления в маршрут. На правой части находится место для добавления стадий. Стадии перемещённые в правую часть будут добавлены в маршрут. Между окнами находятся кнопки добавить, убрать, добавить всё, убрать всё, а также в нижней части кнопки Вверх, Вниз для изменения очередности стадий и кнопки "Ok" и "Отмена".

После подтверждения, введённые данные добавляются в соответствующий список, а так же в базу данных.

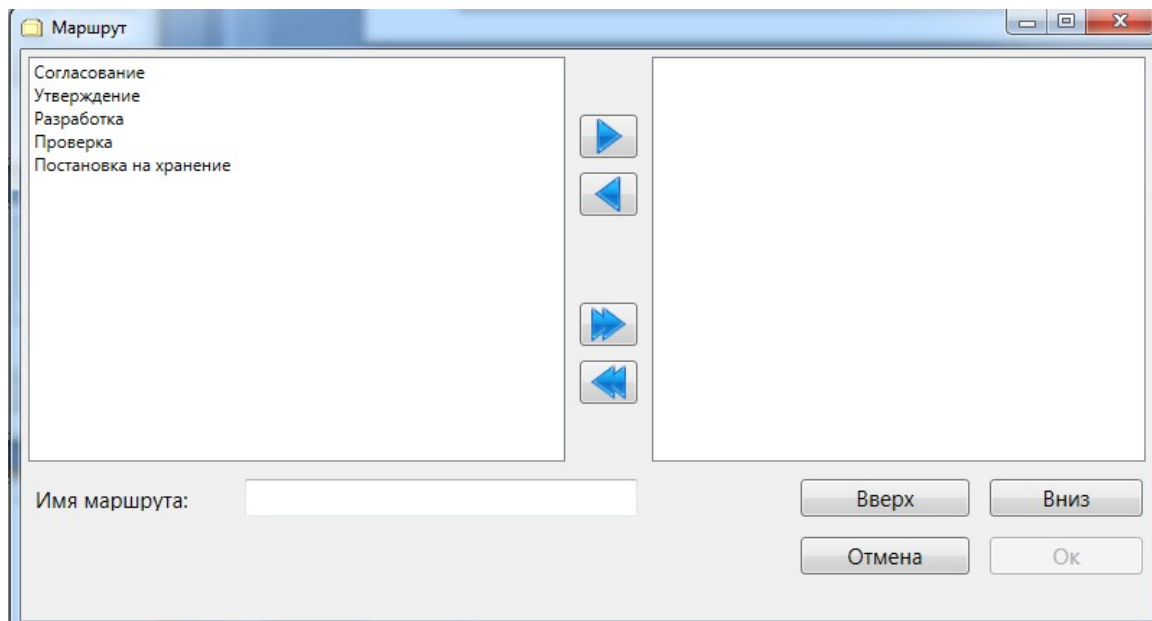


Рисунок 6.15 - Форма создания маршрута

1.16.9. Действие создания стадии

При выборе действия Создать > Стадия проектирования, появляется форма для создания маршрута (рисунок 6.16). На форме расположена строка для ввода наименования стадии, ниже расположен выпадающий список позволяющий выбрать какой подписью будет подписываться документ на создаваемой стадии. Ниже расположены функции выбора возможностей взаимодействия с документом на выбранной стадии проектирования, а также в нижней части кнопки "Ок", "Сохранить" и "Отмена".

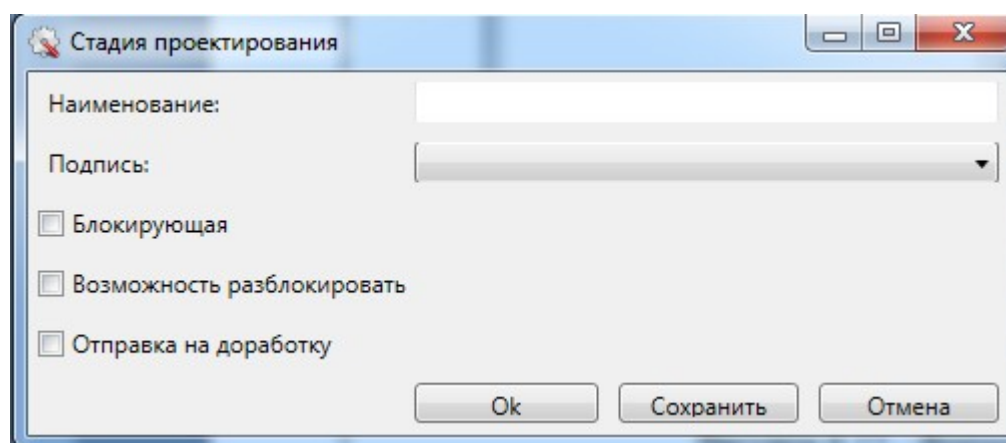


Рисунок 6.16 - Создание стадии проектирования

1.17. Списки

Меню "Списки" содержит кнопки "Список сотрудников", "Список пользователей", "Список типов", "Список должностей", "Список подразделений", "Список воинских званий", "Список ролей", "Список маршрутов", "Список Стадий проектирования" и соответствуют аналогичным кнопкам на панели инструментов в левой части (до разделителя).

Все списки формируются в одинаковой форме. Для примера взята форма списка сотрудники.(рисунок 6.17).

Фамилия	Имя	Отчество	Дата рождения	Подразделение	Уволен
Системы	Администратор		1/1/1900 12:00:00		Нет
Филимонов	Олег	Валерьевич	5/4/1979 12:00:00	Штаб	Нет
Сергеев	Семён	Иванович	9/3/1989 12:00:00	1МСБ\1 Рота	Нет

Рисунок 6.17 - Форма список сотрудников

В верхней части располагается панель инструментов содержащая кнопки "добавить", аналогичной кнопке создать на панели инструментов и в меню Действие>Создать, кнопка "Изменить" по нажатию которой вызывается форма, аналогичная форме добавления, но с уже заполненными данными (рисунок 6.18), кнопка удалить, удаляющая запись из списка, кнопка скрыть/показать панель фильтров, которая расположена в правой части экрана(рисунок 6.17). Далее идут кнопки взаимодействия с фильтрами, а именно кнопки "Подтвердить фильтр", "Очистить фильтр", "Обновить" и кнопка отобразить удалённых, позволяющая отобразить удалённые строки.

Сотрудник (Сергеев Семён Иванович)

Дата рождения: 03.09.1989

Подразделение: 1МСБ\1 Рота

Должность: Командир роты

Воинское звание: Капитан

Тип документа: Паспорт

Документ: 5888967784

Страна: Россия

Почтовый индекс: 442254

Адрес: г. Москва, ул. Пушкина 4 кв. 12

Раб. телефон: 60-12-15

Дом. телефон: 60-07-47

Моб. телефон: 89605241278

Фамилия: Сергеев

Имя: Семён

Отчество: Иванович

Ok Сохранить Отмена

Рисунок 6.18 - Форма изменения сотрудника

1.18. Работа с деревом документов

Дерево документов находится на центральной форме главного окна приложения и занимает большую его часть(рисунок 6.4). Для просмотра содержимого каталогов, необходимо нажать на него левой кнопкой мыши(рисунок 6.19).

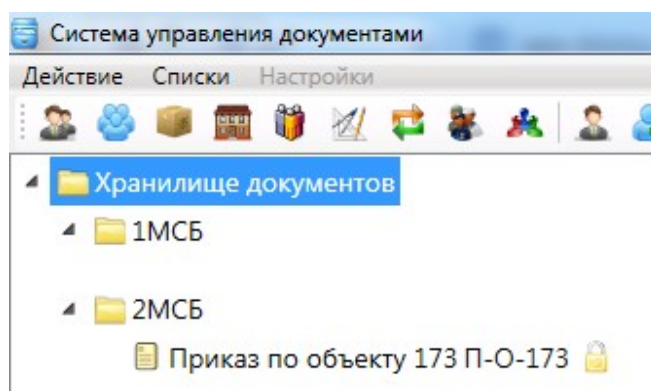


Рисунок 6.19 - Просмотр содержимого каталогов

Для того, чтобы создать новый каталог, необходимо нажать правой кнопкой мыши по основному каталогу "Хранилище документов" либо по любому другому каталогу в дереве документов, откроется контекстное меню в котором необходимо выбрать Создать>Папку (рисунок 6.20). При этом появится форма в которую необходимо ввести наименование папки и подтвердить(рисунок 6.21).

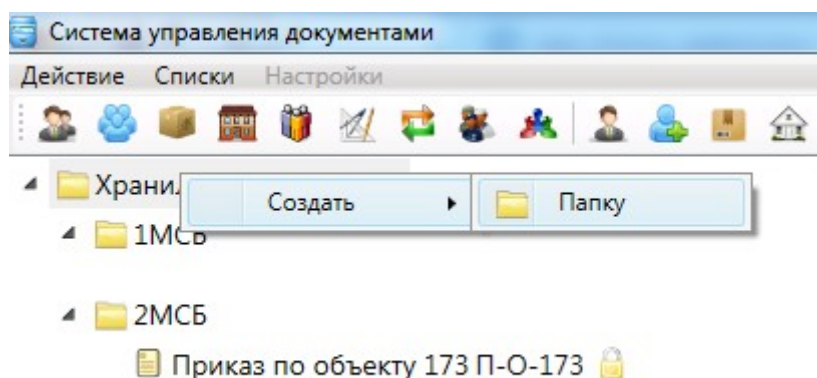


Рисунок 6.20 - Контекстное меню для создания каталога

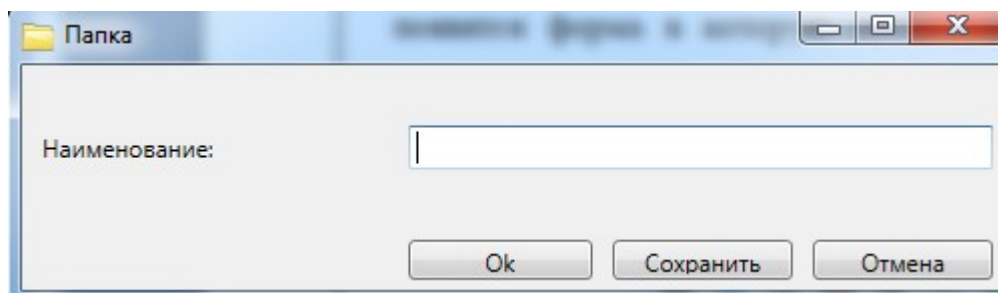


Рисунок 6.21 - Форма создания каталога.

Любому созданному каталогу администратор системы, либо пользователь имеющий возможность разграничения доступа, может назначить доступ к папке, делается это через контекстное меню, вызываемое нажатием правой кнопки мыши по папке и выбором пункта "Назначить доступ"(рисунок 6.22)

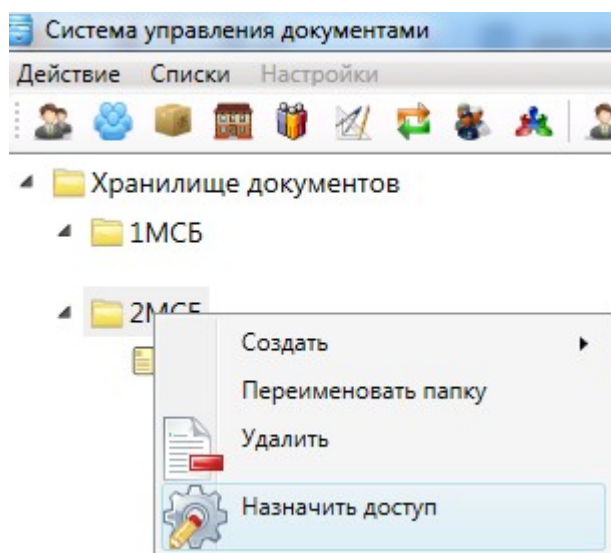


Рисунок 6.22 - Вызов контекстного меню для назначения доступа

При выборе данного пункта меню появляется форма для назначения доступа(рисунок 6.23). В которой нажатием кнопки "Добавить" или "Удалить", а так же опциональным выбором, можно изменять доступ пользователей к папке. Назначение доступа для документов выполняется аналогичным образом.

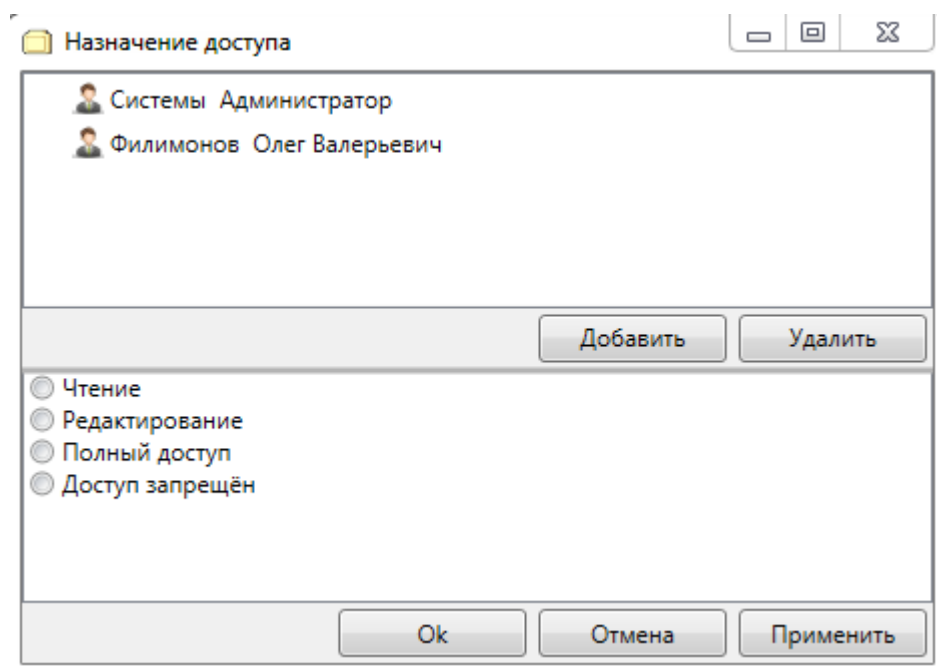


Рисунок 6.23 - Форма для назначения доступа

Для того чтобы создать в дереве документ необходимо нажать правой кнопкой мыши по каталогу и выбрать пункт контекстного меню Создать>Документ и далее выбрать создать документ с типом либо без типа(рисунок 6.24).

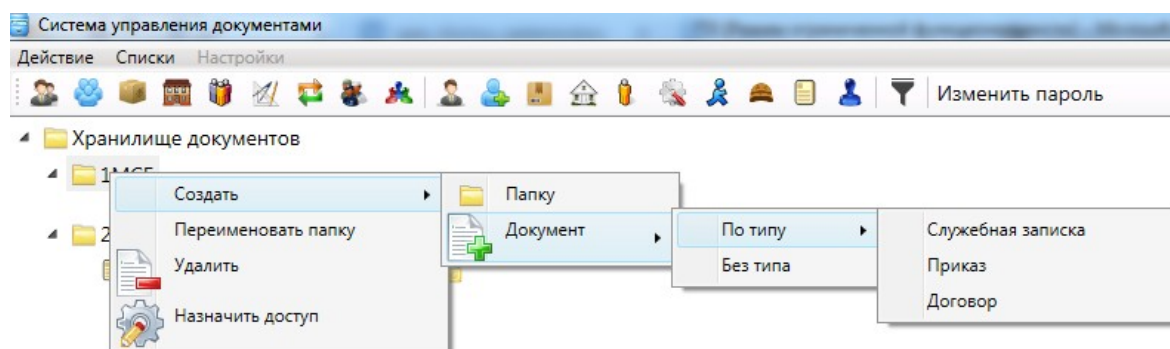


Рисунок 6.24 - Контекстное меню для создания документа

Выбор пунктов меню "По типу" и "Без типа" отличается только тем что в открывшейся форме(рисунок 6.25) будет или не будет соответственно заранее объявленный тип документа.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Рисунок 6.25 - Форма создания документа.

На вкладке "Карточка" располагается кнопка для сохранения документа, в верхней части окна, а так же поля для ввода основных сведений о документе. На вкладке "Файлы" (рисунок 6.26) есть возможность добавить к карточке файл. Для этого необходимо нажать кнопку добавить в верхней части окна, вкладки "Файлы". После чего откроется диалоговое окно в котором необходимо выбрать присоединяемый файл.

На вкладках "Подписи" и "История изменений" в уже созданных карточках будет вноситься информация о том, кем документ подписан и история вносимых в документ изменений, соответственно.

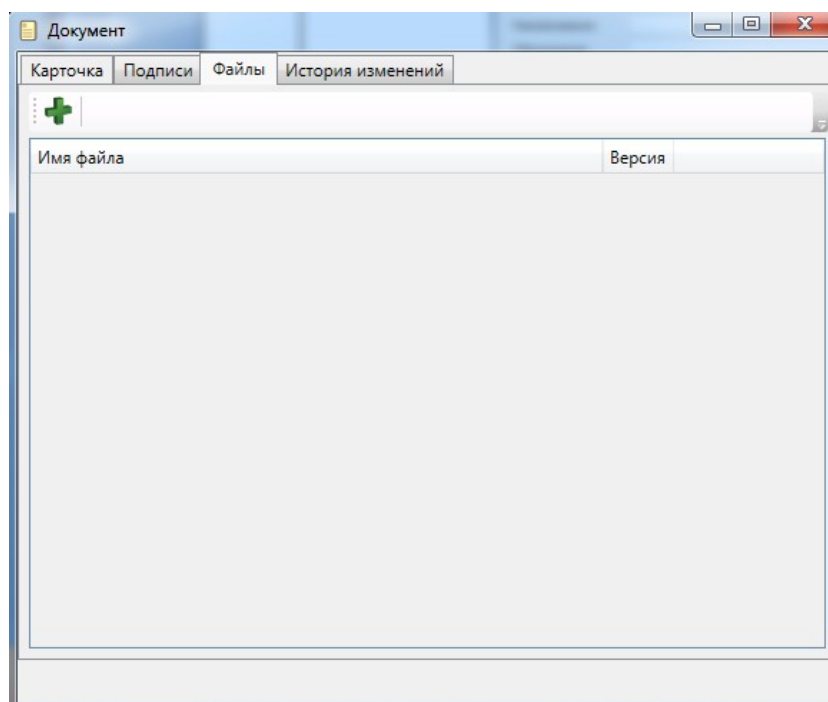


Рисунок 6.26 - Вкладка "Файлы"

Для постановки документа на маршрут необходимо открыть контекстное меню нажатием правой кнопкой мыши по документу (Рисунок 6.27). При этом откроется форма постановки документа на маршрут (Рисунок 6.28). В этом окне необходимо выбрать по какому маршруту будет двигаться документ, а так же кто будет выполнять первую стадию маршрута.

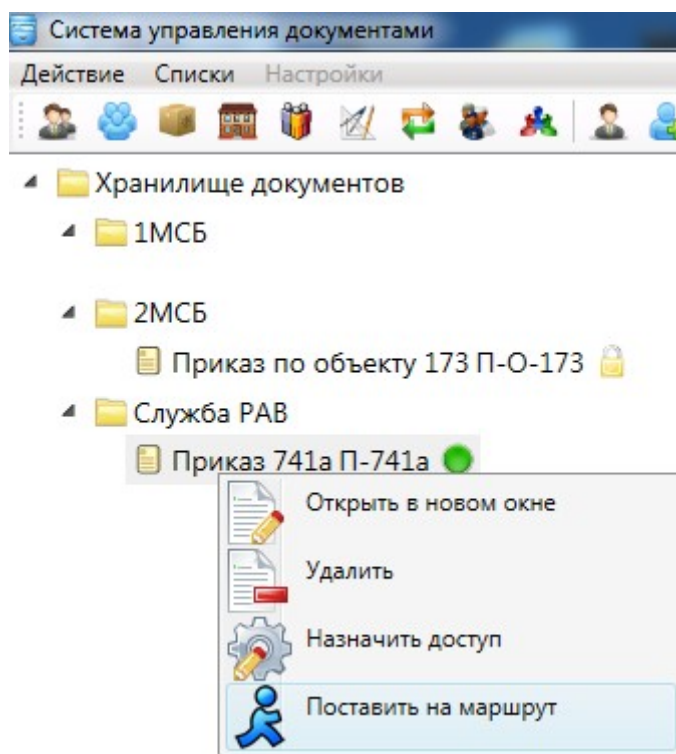


Рисунок 6.27 - Контекстное меню для постановки на маршрут

Рисунок 6.28 - Форма постановки документа на маршрут

При отправки документа по маршруту, исполнителю текущей стадии приходит почтовое сообщение (рисунок 6.29). Через это же сообщение исполнитель стадии может найти документ в списке и подписать его.(рисунок 6.30). Так же получатель может вернуть документ на доработку либо перенаправить его. После подписания документа получатель должен

направить документ следующему должностному лицу, нажатием на соответствующую кнопку.

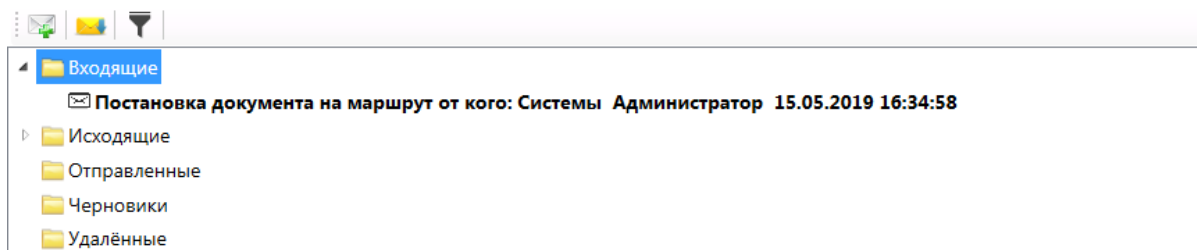


Рисунок 6.29 - Сообщение о постановке документа на маршрут

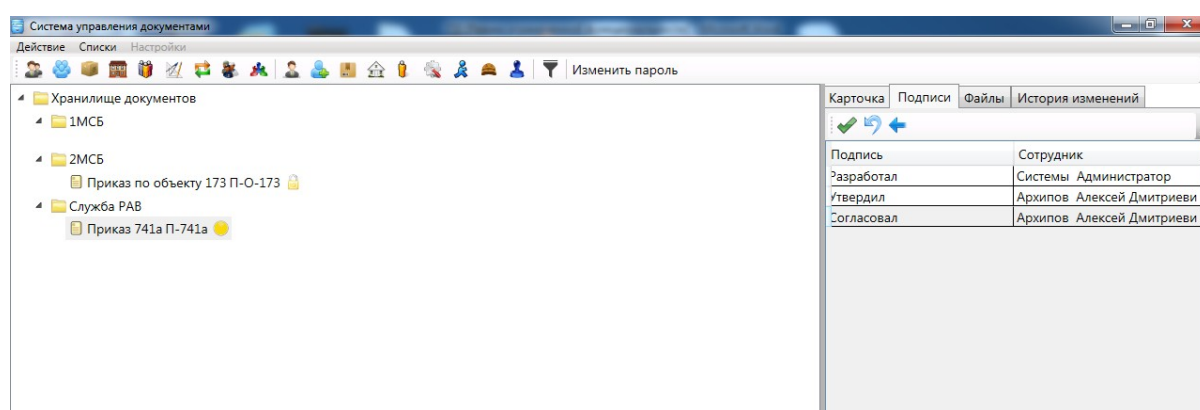


Рисунок 6.30 - Подписание документа должностным лицом

Помимо автоматических почтовых сообщений в системе реализована и отправка сообщений в ручную. Отправить сообщение можно нажатием на соответствующую кнопку в нижней части главного окна (рисунок 6.31). При этом откроется форма сообщения (рисунок 6.32). На форме располагается поле для ввода темы, отправителя сообщения, выбора адресата сообщения, а так же поля для ввода самого сообщения. В верхней части окна располагаются кнопки отправки сообщения, сохранения сообщения в черновиках и кнопка для прикрепления к сообщению документа, при нажатии на которую откроется окно с деревом документов, в котором можно выбрать какой документ прикрепить к отправляемому сообщению (рисунок 6.33).

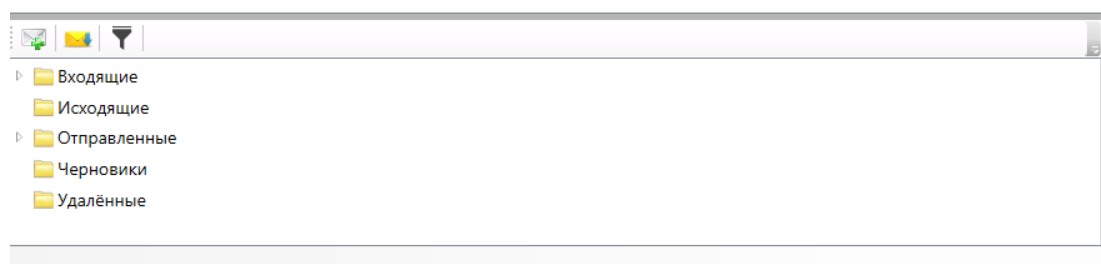


Рисунок 6.31 - Почтовые сообщения

Рисунок 6.32 - Форма для создания сообщения.

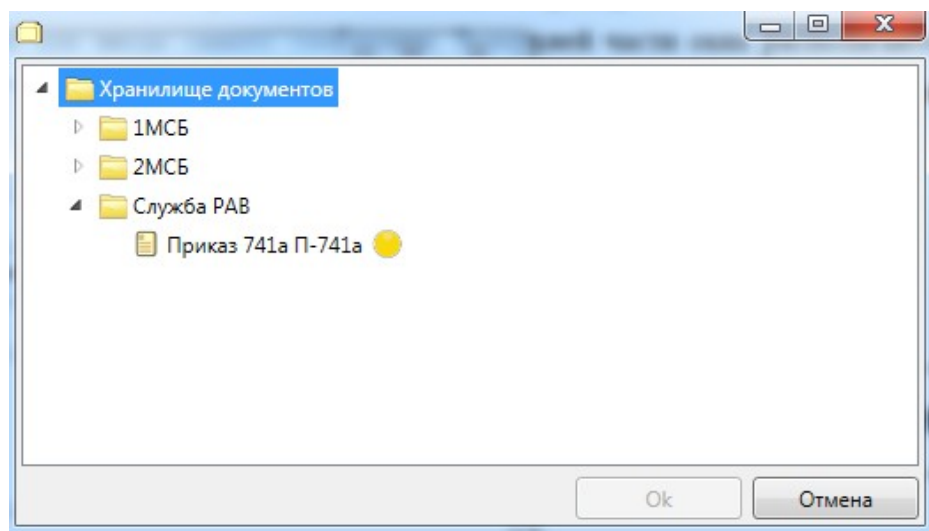


Рисунок 6.33 - Окно выбора документа

Заключение

В процессе выполнения данной выпускной квалификационной работы проведены анализ требований, проектирование и разработка и тестирование программного обеспечения для системы электронного документооборота склада вооружения и военной техники.

В процессе анализа требований, были выделены основные функциональные характеристики. Проведен анализ приоритетов вариантов использования. Разработаны концептуальная, логическая и физическая модели базы данных, иерархия форм. В результате выполненной выпускной квалификационной работы реализовано приложение, являющееся, системой электронного документооборота склада вооружения и военной техники.

Проект реализован на языках C# в среде разработки Visual Studio 2017 и языке SQL в системе управления базами данных PostgreSQL. На основе анализа, полученных результатов тестирования, можно сделать вывод, что все функции, разработанной программы, работают корректно, на всех наборах тестовых данных. Следовательно, цель выпускной квалификационной работы достигнута.

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

Список использованных источников

1. Microsoft Development Network (MSDN), URL: <https://msdn.microsoft.com/ru-ru/>
2. Wikipedia, статья "PostgreSQL", URL: <https://ru.wikipedia.org/wiki/PostgreSQL>
3. Дейтел Х. – С# / Х. Дейтел, Дейтел П., Листфид Дж., Нието Т., Йегер Ш., Златкина М. - СПб: БХВ-Петербург, 2016.
4. Арлоу Д. Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт – СПб: Символ-Плюс, 2017.
5. Мандел Т. Разработка пользовательского интерфейса / Т. Мандел – М.: ДМК Пресс, 2018.
6. Фримен А. – LINQ: язык интегрированных запросов в С# 2016 для профессионалов / А. Фримен, Раттц-мл Д. – М.: Вильямс, 2018.
7. Троелсен Э. – Язык программирования С# 5.0 и платформа .NET 4.5/ Э. Троелсен – М.: Вильямс, 2015.

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Приложение А

(текст программы)

CallbackEventArgs.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    public class CallbackEventArgs : EventArgs
    {
        public readonly string Name;
        public readonly string Message;
        public CallbackEventArgs(string name, string message)
        {
            Name = name;
            Message = message;
        }
    }
}
```

DocServiceCallback.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using Utils;
using Utils.Config;
using System.Diagnostics;
using System.IO;

namespace Doc.App
{
    [CallbackBehavior(UseSynchronizationContext = false, ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class DocServiceCallback : IDocServiceCallback
    {
        public Document NewDocument = null;

        public delegate void AddMessage(string message, DateTime timestamp);
        public event AddMessage AddMessageEvent;

        private static IDocService _proxy = null;
        private static ConfigManager _cfgManager;
        private static string _fileName = "";
        private static ConfigApp _cfg;

        public bool Connect()
        {
            _cfgManager = new ConfigManager();
            Process proc = Process.GetCurrentProcess();

            _fileName = proc.ProcessName + ".cfg";
        }
    }
}
```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        if (!File.Exists(_fileName))
        {
            if (!File.Exists(_fileName))
            {
                _cfgManager.SaveConfiguration(_fileName, new ConfigApp("localhost", "9000", "localhost",
"9100"));
            }
        }

        _cfg = _cfgManager.GetConfiguration<ConfigApp>(_fileName);

        InstanceContext instanceContext = new InstanceContext(this);

        NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);
        binding.MaxReceivedMessageSize = 104857600;
        EndpointAddress address = new EndpointAddress("net.tcp://" + _cfg.HostDB + ":" + _cfg.DbPort +
"/DocService");

        DuplexChannelFactory<IDocService> pipeFactory =
            new DuplexChannelFactory<IDocService>(instanceContext,
                binding, address);

        _proxy = pipeFactory.CreateChannel();

        ///note the "DuplexChannelFactory". This is necessary for Callbacks.
        /// A regular "ChannelFactory" won't work with callbacks.
        ///DuplexChannelFactory<IDocService> pipeFactory =
        ///    new DuplexChannelFactory<IDocService>(
        ///        new InstanceContext(this),
        ///        new NetTcpBinding(),
        ///        new EndpointAddress("net.tcp://localhost:8000/ISubscribe"));

        try
        {
            //Open the channel to the server
            _proxy = pipeFactory.CreateChannel();
            //Now tell the server who is connecting
            _proxy.Subscribe();
            return true;
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            return false;
        }
    }

    public void Close()
    {
        _proxy.Unsubscribe();
    }

    ///This function sends a string to the server so that it can broadcast
    /// it to all other clients that have called Subscribe().
    public void SendMessage(string message, bool isedit, bool delete)
    {
        try
        {
            App.Proxy.AddMessage(message, isedit, delete);
        }
        catch (Exception e)
        {
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//This is the function that the SERVER will call
public void OnMessageAdded(string message, bool isedit, bool delete)
{
    if(delete)
        App.DelList.Add(message);
    else
    {
        if (isedit)
            App.ChangeList.Add(message);
        else
            App.NewList.Add(message);
    }
}

//We need to tell the server that we are leaving
public void Dispose()
{
    _proxy.Unsubscribe();
}
}

```

IDocServiceCallback.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;

namespace Doc.App
{
    [ServiceContract]
    interface IDocServiceCallback
    {
        //This is the callback interface declaration for the client
        [OperationContract(IsOneWay = true)]
        void OnMessageAdded(string message, bool isedit, bool delete);
    }
}

```

IDocService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using System.Collections;
using System.Runtime.Serialization;

namespace Doc.App
{
    [ServiceContract(CallbackContract = typeof(IDocServiceCallback))]
    public interface IDocService
    {
        /// <summary>
        /// Проверка подключения
        /// </summary>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <returns></returns>
[OperationContract]
bool IsConnected();

/// <summary>
/// Выполняет запрос к БД
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="param">Передаваемые параметры</param>
/// <returns>Результат запроса</returns>
[OperationContract]
ArrayList ExecuteQuery(string query, Hashtable param);

/// <summary>
/// Выполняет вставку удаление, изменение записей в таблицах
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="lstparam">Передаваемые параметры</param>
[OperationContract]
void ExecuteNonQueryTr(string query, List<Hashtable> lstparam);
/// <summary>
/// Выполняет вставку удаление, изменение записей в таблицах
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="param">Передаваемые параметры</param>
[OperationContract]
void ExecuteNonQuery(string query, Hashtable param);

[OperationContract]
Hashtable GetPhoto(int id);

//these are the interface declerations for the server.
[OperationContract]
void AddMessage(string message, bool isedit, bool delete);
[OperationContract]
bool Subscribe();
[OperationContract]
bool Unsubscribe();
/// <summary>
/// Добавить документ
/// </summary>
[OperationContract]
void InsertDocument(Hashtable querys, Hashtable param, List<Hashtable> parAdminAccess,
List<Hashtable> parFiles);
/// <summary>
/// Удалить документ
/// </summary>
[OperationContract]
bool DeleteDocument(Hashtable queryes, Hashtable param, List<long> lstDel, bool del);
/// <summary>
/// Добавить файл в хранилище
/// </summary>
/// <param name="querys">Запросы</param>
/// <param name="param">Параметры</param>
[OperationContract]
void InsertFile(Hashtable querys, Hashtable param);

/// <summary>
/// Изменяет запись о файле в базе
/// </summary>
/// <param name="querys"></param>
/// <param name="param"></param>
[OperationContract]
void UpdateFile(Hashtable querys, Hashtable param);

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Заменяет текущую версию файла
/// </summary>
/// <param name="querys"></param>
/// <param name="param"></param>
[OperationContract]
void SetFileCurrentVersion(Hashtable querys, Hashtable param);

/// <summary>
/// Восстанавливает документ из удаленных
/// </summary>
/// <param name="queryes"></param>
/// <param name="pardoc"></param>
/// <param name="parhis"></param>
[OperationContract]
void RestoreDocument(Hashtable queryes, List<Hashtable> pardoc, List<Hashtable> parhis);
}

```

```

}

```

IFSServise.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Utils;
using System.ServiceModel;
namespace Doc.App
{
    [ServiceContract]
    public interface IFSService
    {
        /// <summary>
        /// Проверка подключения
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        bool IsConnected();

        /// <summary>
        /// Запись файла в хранилище
        /// </summary>
        /// <param name="stream">Поток с клиента</param>
        /// <param name="filename">Имя файла в хранилище</param>
        /// <param name="oldfilename">Новое имя старого файла</param>
        /// <returns>Признак перезаписи</returns>
        [OperationContract]
        bool SaveFile(byte[] stream, string filename, string fullpath);

        /// <summary>
        /// Получает байтовый поток из файла в хранилище
        /// </summary>
        /// <param name="filename">Имя файла в хранилище</param>
        /// <returns>Байтовый поток из файла в хранилище</returns>
        [OperationContract]
        byte[] GetFile(string filename, string fullpath);

        /// <summary>
        /// Удаляет все версии файла из хранилища
        /// </summary>
        /// <param name="fullpath">Панка файла</param>
        /// <returns></returns>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

[OperationContract]
bool DeleteFile(string fullpath);

/// <summary>
/// Устанавливает выбранную версию текущей
/// </summary>
/// <param name="oldversion">Текущая версия</param>
/// <param name="newversion">Устанавливаемая версия</param>
/// <param name="newfilename">Новое имя текущей версии</param>
[OperationContract]
bool SetCurrentVersion(string oldversion, string newversion);
}
}

```

ServerManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using System.Diagnostics;
using Utils;
using System.IO;
using Utils.Config;
using System.Windows;

namespace Doc.App
{
    public static class ServerManager
    {
        private static IDocService _proxy = null;
        private static IFSService _proxyFS = null;
        private static ConfigManager _cfgManager;
        private static string _fileName = "";
        private static ConfigApp _cfg;

        /// <summary>
        /// Устанавливает связь с каналом
        /// </summary>
        /// <returns></returns>
        public static IDocService GetProxyDb()
        {
            _cfgManager = new ConfigManager();
            Process proc = Process.GetCurrentProcess();

            _fileName = proc.ProcessName + ".cfg";

            if (!File.Exists(_fileName))
            {
                if (!File.Exists(_fileName))
                {
                    _cfgManager.SaveConfiguration(_fileName, new ConfigApp("localhost", "9000", "localhost",
"9100"));
                }
            }

            _cfg = _cfgManager.GetConfiguration<ConfigApp>(_fileName);

            //сервер БД
            DocServiceCallback callback = new DocServiceCallback();

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

InstanceContext instanceContext = new InstanceContext(callback);

NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);
binding.MaxReceivedMessageSize = 104857600;
EndpointAddress address =
    new EndpointAddress("net.tcp://" + _cfg.HostDB + ":" + _cfg.DbPort + "/DocService");

DuplexChannelFactory<IDocService> pipeFactory =
    new DuplexChannelFactory<IDocService>(instanceContext,
        binding, address);

_proxy = pipeFactory.CreateChannel();
// _proxy.Subscribe();

return _proxy;
}

public static IFSService GetProxyFS()
{
    _cfgManager = new ConfigManager();
    Process proc = Process.GetCurrentProcess();

    _fileName = proc.ProcessName + ".cfg";

    if (!File.Exists(_fileName))
    {
        if (!File.Exists(_fileName))
        {
            _cfgManager.SaveConfiguration(_fileName, new ConfigApp("localhost", "9000", "localhost",
"9100"));
        }
    }

    _cfg = _cfgManager.GetConfiguration<ConfigApp>(_fileName);

    NetTcpBinding binding = new NetTcpBinding(SecurityMode.None);
    binding.MaxReceivedMessageSize = 104857600;
    EndpointAddress address = new EndpointAddress("net.tcp://" + _cfg.HostFS + ":" + _cfg.FSPort +
"/FSService");

    _proxyFS = ChannelFactory<IFSService>.CreateChannel(binding, address);
    return _proxyFS;
}

/// <summary>
/// Проверка подключения
/// </summary>
public static void IsConnected()
{
    //проверка наличия канала
    try
    {
        App.Proxy.IsConnected();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Служба сервера БД недоступна.\n " +
            "Приложение завершает работу. Обратитесь к администратору системы.", "",
            MessageBoxButton.OK, MessageBoxImage.Error);
        throw;
    }

    try
    {
        App.ProxyFS.IsConnected();
    }

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }
    catch (Exception ex)
    {
        MessageBox.Show("Служба файлового сервера недоступна.\n " +
            "Приложение завершает работу. Обратитесь к администратору системы.", "",
            MessageBoxButton.OK, MessageBoxImage.Error);
        throw;
    }
}
}
}

```

AccessManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления доступом к документам
    /// </summary>
    class AccessManager
    {
        public Hashtable Filter = new Hashtable();
        private DocumentsManager _documentsManager = null;

        /// <summary>
        /// Конструктор класса AccessManager
        /// </summary>
        public AccessManager()
        {
            Filter.Add(":_emplid", null);
            Filter.Add(":_subid", null);
            Filter.Add(":_accesstype", null);
            Filter.Add(":_objectid", null);

            _documentsManager = new DocumentsManager();
        }

        /// <summary>
        /// Добавляет права доступа к документу
        /// </summary>
        /// <param name="access">Права доступа</param>
        public void Insert(List<Access> list)
        {
            List<Hashtable> lstPar = new List<Hashtable>();

            string query = "CALL sp_addaccess(:_objectid, :_emplid, :_subid, :_accesstype)";
            foreach (var access in list)
            {
                var param = new Hashtable();
                param.Add(":_objectid", access.ObjectID);
                param.Add(":_subid", access.SubID);
                param.Add(":_emplid", access.EmplID);
                param.Add(":_accesstype", access.AccessType);
                lstPar.Add(param);
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

try
{
    App.Proxy.ExecuteNonQueryTr(query, lstPar);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.AccessManager.Insert", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}
}
/// <summary>
/// Получает права доступа
/// </summary>
/// <returns></returns>
public List<Access> Select()
{
    string query = "SELECT * FROM fn_getaccess(:_objectid, :_emplid, :_subid, :_accesstype);";
    List<Access> list = new List<Access>();
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Access access = new Access(Int32.Parse(t[0]), Int64.Parse(t[1]), Int32.Parse(t[2]), Int32.Parse(t[3]),
Int32.Parse(t[4]));
            list.Add(access);
        }
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AccessManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter[":_emplid"] = null;
    Filter[":_subid"] = null;
    Filter[":_accesstype"] = null;
    Filter[":_objectid"] = null;

    return list;
}

//private void GetTreeDocuments(Access access)
//{
//    // папки верхнего уровня
//    _documentsManager.Filter[":_parentid"] = access.ObjectID;
//    List<Document> docs = _documentsManager.Select(false, "");

//    List<Access> accesses = new List<Access>();

//    foreach (var doc in docs)
//    {
//        //TreeViewItem item = CreateTreeViewItem(sub);

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//      //tvUsers.Items.Add(item);
//    }
//    tvUsers.Items.Refresh();
//    //дочерние
//    if (tvUsers.HasItems)
//        foreach (TreeViewItem item in tvUsers.Items)
//        {
//            GetChildrenSub(item);
//        }

//    //сотрудники без подразделения
//    _employeesManager.Filter[":_subdivid"] = 0;
//    List<Employee> employees = _employeesManager.Select(false);
//    if (employees.Count > 0)
//    {
//        foreach (var empl in employees)
//        {
//            TreeViewItem item = CreateTreeviewItem(empl);
//            tvUsers.Items.Add(item);
//        }
//    }
//}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="access">Права доступа</param>
public void Update(List<Access> list)
{
    List<Hashtable> lstPar = new List<Hashtable>();

    string query = "CALL sp_updateaccess(:_id, :_accesstype); ";

    foreach (var access in list)
    {
        var param = new Hashtable();
        param.Add(":_id", access.ID);
        param.Add(":_accesstype", access.AccessType);
        lstPar.Add(param);
    }

    try
    {
        App.Proxy.ExecuteNonQueryTr(query, lstPar);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AccessManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор доступа</param>
public void Delete(List<Access> delList)
{
    string query = "CALL sp_delaccess(:_id); ";
    List<Hashtable> lstPar = new List<Hashtable>();
    foreach (var access in delList)

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    {
        var param = new Hashtable();
        param.Add(":_id", access.ID);
        lstPar.Add(param);
    }

    try
    {
        App.Proxy.ExecuteNonQueryTr(query, lstPar);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AccessManager.Delete", MessageType.Error, e,
        DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
}

```

AddressManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления адресами
    /// </summary>
    class AddressManager
    {
        public Hashtable Filter = new Hashtable();

        public AddressManager()
        {
            Filter.Add(":_id", null);
        }
        /// <summary>
        /// Добавляет новый адрес
        /// </summary>
        /// <param name="addr">Адрес</param>
        public void Insert(Address addr)
        {
            string query = "CALL sp_addaddress(:_postindex, :_country, :_addr)";
            var param = new Hashtable();
            param.Add(":_postindex", addr.PostIndex);
            param.Add(":_country", addr.Country);
            param.Add(":_addr", addr.Addr);
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.AddressManager.Insert", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}
}
/// <summary>
/// Получает адрес
/// </summary>
/// <returns></returns>
public Address Select()
{
    string query = "SELECT * FROM fn_getaddress(:_id);";
    Address addr = null;
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        if (arr.Count > 0)
        {
            string[] t = ((string)arr[0]).Split('|');
            addr = new Address(Int32.Parse(t[0]), t[1], t[2], t[3]);
        }
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AddressManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter[":_id"] = null;

    return addr;
}

public Address SelectLast()
{
    string query = "SELECT * FROM fn_getaddress(NULL) " +
        "ORDER BY ID DESC LIMIT 1;";

    Address addr = null;
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        addr = new Address(Int32.Parse(t[0]), t[1], t[2], t[3]);
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
        Process proc = Process.GetCurrentProcess();

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Logger.Log log = new Logger.Log("Doc.App.AddressManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter[":_id"] = null;

    return addr;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="addr">Адрес</param>
public void Update(Address addr)
{
    string query = "CALL sp_updateaddress(:_id, :_postindex, :_country, :_addr); ";

    var param = new Hashtable();
    param.Add(":_id", addr.ID);
    param.Add(":_postindex", addr.PostIndex);
    param.Add(":_country", addr.Country);
    param.Add(":_addr", addr.Addr);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AddressManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор адреса</param>
public void Delete(int id)
{
    string query = "CALL sp_deladdress(:_id); ";

    var param = new Hashtable();

    param.Add(":_id", id);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AddressManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дат

```

    }
  }
}

```

AppointmentManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления справочником должностей
    /// </summary>
    class AppointmentManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса AppointmentManager
        /// </summary>
        public AppointmentManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
        }

        /// <summary>
        /// Создаёт новую должность
        /// </summary>
        /// <param name="appoint">Должность</param>
        public void Insert(Appointment appoint)
        {
            string query = "CALL sp_addappoint(:_name)";

            var param = new Hashtable();
            param.Add(":_name", appoint.Name);
            try
            {
                App.Proxy.ExecuteNonQuery(query, param);
            }
            catch (Exception e)
            {
                Process proc = Process.GetCurrentProcess();

                Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Insert", MessageType.Error, e,
                    DateTime.Now);
                string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
                Logger.WriteLog(logFile, log);
                MessageBox.Show(e.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        /// <summary>
        /// Получает список должностей
        /// </summary>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <returns>Список должностей</returns>
public List<Appointment> Select()
{
    List<Appointment> list = new List<Appointment>();

    string query =
        "SELECT * FROM fn_getappointments(:_id, :_name)";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Appointment sub = new Appointment(Int32.Parse(t[0]), t[1]);
            list.Add(sub);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_name", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="appoint">Должность</param>
public void Update(Appointment appoint)
{
    string query = "CALL sp_updateappoint(:_id, :_name); ";

    var param = new Hashtable();
    param.Add(":_id", appoint.ID);
    param.Add(":_name", appoint.Name);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <param name="id">Идентификатор должности</param>
public bool Delete(int id)
{
    string query = "CALL sp_delappoint(:_id); ";
    List<Employee> list = new List<Employee>();
    EmployeesManager employeesManager = new EmployeesManager();
    employeesManager.Filter[":_appointid"] = id;
    list = employeesManager.Select(true);
    if (list.Count > 0)
    {
        MessageBox.Show("Есть сотрудники с указанной должностью.\nУдаление невозможно",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
    var param = new Hashtable();
    param.Add(":_id", id);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
        return true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Delete", MessageType.Error, e,
            DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}

/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public Appointment SelectLast()
{
    Appointment appoint = null;
    string query = "SELECT * FROM 'Appointments' ORDER BY 'ID' DESC LIMIT 1;";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        appoint = new Appointment(Int32.Parse(t[0]), t[1]);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
            DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return appoint;
}
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

DocumentsManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Documents;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления документами
    /// </summary>
    class DocumentsManager
    {
        public Hashtable Filter = new Hashtable();
        private UserManager _userManager = null;

        /// <summary>
        /// Конструктор класса DocumentsManager
        /// </summary>
        public DocumentsManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_parentid", null);
            Filter.Add(":_name", null);
            Filter.Add(":_currhid", null);
            Filter.Add(":_remhid", null);
            Filter.Add(":_authorid", null);
            Filter.Add(":_lockid", null);
            Filter.Add(":_stageid", null);
            Filter.Add(":_typeid", null);
            Filter.Add(":_datecreatebeg", null);
            Filter.Add(":_datecreateend", null);
            Filter.Add(":_dateeditbeg", null);
            Filter.Add(":_dateeditend", null);
            Filter.Add(":_designation", null);
            Filter.Add(":_editauthor", null);

            _userManager = new UserManager();
        }
        /// <summary>
        /// Добавляет новый документ
        /// </summary>
        /// <param name="doc"></param>
        public void Insert(Document doc, List<DocFile> files)
        {
            //Несколько запросов в одну транзакцию объединить. На сервере написать процедуру для
            //выполнения в пределах транзакции
            History history = new History(0, (int)HistoryObjectEnum.Document, doc.ID, doc.CreateDate,
            doc.AuthorID
            , Environment.MachineName, "Создан новый документ");

            var paramHis = new Hashtable();
            paramHis.Add(":_objecttype", history.ObjectType);
            paramHis.Add(":_objectid", history.ObjectID);
            paramHis.Add(":_datechange", history.DateChange);
        }
    }
}
```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

paramHis.Add(":_changeauthor", history.ChangeAuthor);
paramHis.Add(":_changehost", history.ChangeHost);
paramHis.Add(":_action", history.Action);

//1. Получаем последний документ
string queryLastDoc = "SELECT \"ID\" FROM \"Documents\" ORDER BY \"ID\" DESC LIMIT 1; ";
//2. Insert в Document
string queryInsDoc = "CALL
sp_adddocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_authorid, :_lockid, :_lockhost, :_routeid, :_stageid, :_typeid, " +
":_createdate, :_editdate, :_designation, :_description, :_editauthor, :_stagenum); ";
//3. Инсерт в History
string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";

//4. Получаем последний HID
string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1;

string queryLastFiles = "";
string queryInsertFiles = "";
Hashtable querys = new Hashtable();
List<Hashtable> parFiles = new List<Hashtable>();
List<Hashtable> parHisFiles = new List<Hashtable>();
Hashtable param = new Hashtable();

if (files.Count > 0)
{
    //Если есть файлы
    //6. Последний ID файла
    queryLastFiles = "SELECT \"ID\" FROM \"Files\" ORDER BY \"ID\" DESC LIMIT 1; ";
    //7. вставка в таблицу Files
    queryInsertFiles = "CALL sp_addfile(:_id, :_name, :_fullpath, :_currhid, :_remhid, :_docid, :_version); ";

    //Параметры для файлов

    foreach (var file in files)
    {
        Hashtable par = new Hashtable();
        par.Add(":_id", file.ID);
        par.Add(":_name", file.Name);
        par.Add(":_fullpath", file.FullPath);
        par.Add(":_currhid", file.CurHID);
        par.Add(":_remhid", file.RemHID);
        par.Add(":_docid", file.DocID);
        par.Add(":_version", file.Version);

        parFiles.Add(par);
    }

    Hashtable parHisFile = new Hashtable();

    History hisFile = new History(0, (int)HistoryObjectEnum.File, 0, DateTime.Now, doc.AuthorID
, Environment.MachineName, "Создан новый файл");

    parHisFile.Add(":_objecttype", hisFile.ObjectType);
    parHisFile.Add(":_objectid", hisFile.ObjectID);
    parHisFile.Add(":_datechange", hisFile.DateChange);
    parHisFile.Add(":_changeauthor", hisFile.ChangeAuthor);
    parHisFile.Add(":_changehost", hisFile.ChangeHost);
    parHisFile.Add(":_action", hisFile.Action);

    param.Add("HisFiles", parHisFile);

    querys.Add("LastFile", queryLastFiles);

```

```

    querys.Add("InsertFile", queryInsertFiles);
}

```

```

var paramDoc = new Hashtable();
paramDoc.Add(":_id", doc.ID);
paramDoc.Add(":_parentid", doc.ParentID);
paramDoc.Add(":_name", doc.Name);
paramDoc.Add(":_currhid", doc.CurHID);
paramDoc.Add(":_remhid", doc.RemHID);
paramDoc.Add(":_authorid", doc.AuthorID);
paramDoc.Add(":_lockid", doc.LockID);
paramDoc.Add(":_createdate", doc.CreateDate);
paramDoc.Add(":_editdate", doc.EditDate);
paramDoc.Add(":_lockhost", doc.LockHost);
paramDoc.Add(":_routeid", doc.RouteID);
paramDoc.Add(":_stageid", doc.StageID);
paramDoc.Add(":_typeid", doc.TypeID);
paramDoc.Add(":_designation", doc.Designation);
paramDoc.Add(":_description", doc.Description);
paramDoc.Add(":_editauthor", doc.EditAuthorID);
paramDoc.Add(":_stagenum", doc.StageNum);

```

//11. Автору документа назначаем полный доступ по умолчанию

```

Access access = new Access(0, doc.ID, doc.AuthorID, 0, (int)AccessType.Full);

```

```

string queryInsertAccess = "CALL sp_addaccess(:_objectid, :_emplid, :_subid, :_accesstype); ";
var paramAccess = new Hashtable();
paramAccess.Add(":_objectid", access.ObjectID);
paramAccess.Add(":_subid", access.SubID);
paramAccess.Add(":_emplid", access.EmplID);
paramAccess.Add(":_accesstype", access.AccessType);

```

//Запросы

```

querys.Add("LastDocument", queryLastDoc);
querys.Add("InsertDocument", queryInsDoc);
querys.Add("InsertHistory", queryHis);
querys.Add("LastHistory", queryLastHis);
querys.Add("InsertAccess", queryInsertAccess);

```

//параметры

```

param.Add("Document", paramDoc);
param.Add("History", paramHis);
param.Add("Access", paramAccess);

```

```

_userManager.Filter[":_emplid"] = doc.AuthorID;
List<User> users = _userManager.Select();
List<Hashtable> parAdminAccess = new List<Hashtable>();

```

```

if (users[0].Role.Name != "Администратор")
{
    //доступ администраторам
    //Выбираем всех с ролью администратор
    List<User> lst = new List<User>();
    RoleManager roleManager = new RoleManager();
    roleManager.Filter[":_name"] = "Администратор";
    List<Role> roles = roleManager.Select();

```

```

    if (roles.Count > 0)
    {

```

```

        _userManager.Filter[":_roleid"] = roles[0].ID;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        lst = _userManager.Select();
        //string query = "CALL sp_addaccess(:_objectid, :_emplid, :_subid, :_accesstype); ";

        foreach (var user in lst)
        {
            var paramAccessAdmin = new Hashtable();
            paramAccessAdmin.Add(":_objectid", access.ObjectID);
            paramAccessAdmin.Add(":_subid", access.SubID);
            paramAccessAdmin.Add(":_emplid", user.EmplID);
            paramAccessAdmin.Add(":_accesstype", (int)AccessType.Full);

            parAdminAccess.Add(paramAccessAdmin);
        }
    }

    try
    {
        App.Proxy.InsertDocument(querys, param, parAdminAccess, parFiles);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.Insert", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }
}

/// <summary>
/// Получить список по фильтру
/// </summary>
/// <returns></returns>
public List<Document> Select(bool withdeleted, string where)
{
    List<Document> list = new List<Document>();
    if(!withdeleted)
        Filter[":_remhid"] = 0;

    string query = "SELECT * FROM
fn_getdocuments(:_id, :_parentid, :_name, :_currhid, :_remhid, :_authorid, :_lockid, " +
        ":_stageid, :_typeid, :_datecreatebeg, :_datecreateend, :_dateeditbeg, :_dateeditend, " +
        ":_designation, :_editauthor) " + where + ";";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Document doc = new Document(Int64.Parse(t[0])
                ,Int64.Parse(t[1])
                , t[2]
                , Int64.Parse(t[3])
                , Int64.Parse(t[4])
                , Int32.Parse(t[5])
                , Int32.Parse(t[6])
                , t[7]
                , Int32.Parse(t[8])
                , Int32.Parse(t[9])
                , Int32.Parse(t[10])
                , DateTime.Parse(t[11])
                , DateTime.Parse(t[12])
            );
        }
    }
}

```

```

        , t[13]
        , t[14]
        , Int32.Parse(t[15])
        , Int32.Parse(t[16]));
    list.Add(doc);
    }
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.Select", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

Filter.Clear();
Filter.Add(":_id", null);
Filter.Add(":_parentid", null);
Filter.Add(":_name", null);
Filter.Add(":_currhid", null);
Filter.Add(":_remhid", null);
Filter.Add(":_authorid", null);
Filter.Add(":_lockid", null);
Filter.Add(":_stageid", null);
Filter.Add(":_typeid", null);
Filter.Add(":_datecreatebeg", null);
Filter.Add(":_datecreateend", null);
Filter.Add(":_dateeditbeg", null);
Filter.Add(":_dateeditend", null);
Filter.Add(":_designation", null);
Filter.Add(":_editauthor", null);

return list;
}

public Document SelectLast(int emplid)
{
    Document doc = null;

    string query = string.Format("SELECT * FROM \\"Documents\\" WHERE \\"AuthorID\\" = {0} " +
        "ORDER BY \\"ID\\" DESC LIMIT 1; ", emplid);

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = arr[0].ToString().Split('|');
        doc = new Document(Int64.Parse(t[0])
            , Int64.Parse(t[1])
            , t[2]
            , Int64.Parse(t[3])
            , Int64.Parse(t[4])
            , Int32.Parse(t[5])
            , Int32.Parse(t[6])
            , t[7]
            , Int32.Parse(t[8])
            , Int32.Parse(t[9])
            , Int32.Parse(t[10])
            , DateTime.Parse(t[11])
            , DateTime.Parse(t[12])
            , t[13]
            , t[14]
            , Int32.Parse(t[15])
            , Int32.Parse(t[16]));
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.SelectLast", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_parentid", null);
    Filter.Add(":_name", null);
    Filter.Add(":_currhid", null);
    Filter.Add(":_remhid", null);
    Filter.Add(":_authorid", null);
    Filter.Add(":_lockid", null);
    Filter.Add(":_stageid", null);
    Filter.Add(":_typeid", null);
    Filter.Add(":_datecreatebeg", null);
    Filter.Add(":_datecreateend", null);
    Filter.Add(":_dateeditbeg", null);
    Filter.Add(":_dateeditend", null);
    Filter.Add(":_designation", null);
    Filter.Add(":_editauthor", null);

    return doc;
}

private List<long> GetChildDocuments(Document doc, bool withdeleted, List<long> lstChild)
{
    Filter[":_parentid"] = doc.ID;
    List<Document> lstDoc = Select(withdeleted, "");
    List<long> lst = lstChild;
    foreach (var d in lstDoc)
    {
        if (d.LockID != 0 || doc.StageID != 0)
        {
            lst.Clear();
            break;
        }
        lst.Add(d.ID);

        lst = GetChildDocuments(d, withdeleted, lst);
    }

    return lst;
}

public bool Delete(Document doc, int emplID, string workdir)
{
    DateTime changeDate = DateTime.Now;
    //1. Получаем удаляемые документы
    List<long> lst = new List<long>();
    lst.Add(doc.ID);
    lst = GetChildDocuments(doc, false, lst);
    if (lst.Count == 0)
    {
        MessageBox.Show("Удаляемый объект содержит дочерние, находящиеся в работе или на
хранении.\n" +
        "Удаление невозможно");
        return false;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }

    //string queryDelDoc = string.Format("SELECT * FROM \"Documents\" WHERE \"ID\" IN({0}); ", str);
    //2. Инсерт в History
    string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";

    History history = new History(0, (int)HistoryObjectEnum.Document, doc.ID, changeDate, emplID
    , Environment.MachineName, "Документ удалён");

    var paramHis = new Hashtable();
    paramHis.Add(":_objecttype", history.ObjectType);
    paramHis.Add(":_objectid", history.ObjectID);
    paramHis.Add(":_datechange", history.DateChange);
    paramHis.Add(":_changeauthor", history.ChangeAuthor);
    paramHis.Add(":_changehost", history.ChangeHost);
    paramHis.Add(":_action", history.Action);

    //3. Получаем последний HID
    string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1";
    //4. Update документа
    string queryUpdDoc = "UPDATE \"Documents\" " +
        "SET \"EditDate\" = :_editdate, \"EditAuthor\" = :_editauthor, " +
        "\"RemHID\" = :_remhid " +
        "WHERE \"ID\" = :_id AND \"RemHID\" = 0 ; ";

    var paramDoc = new Hashtable();
    paramDoc.Add(":_id", doc.ID);
    paramDoc.Add(":_remhid", doc.RemHID);
    paramDoc.Add(":_editdate", changeDate);
    paramDoc.Add(":_editauthor", emplID);

    //Запросы
    Hashtable querys = new Hashtable();

    // querys.Add("SelectDelDoc", queryDelDoc);
    querys.Add("InsertHistory", queryHis);
    querys.Add("LastHistory", queryLastHis);
    querys.Add("UpdateDocument", queryUpdDoc);

    //параметры
    Hashtable param = new Hashtable();
    param.Add("Document", paramDoc);
    param.Add("History", paramHis);

    try
    {
        if (!App.Proxy.DeleteDocument(querys, param, lst, false))
        {
            return false;
        }

        foreach (var docid in lst)
        {
            //1. Ищем связанные файлы
            FileManager filesManager = new FileManager();
            filesManager.Filter[":_docid"] = docid;
            List<DocFile> lstFile = filesManager.Select(false);
            //2. Удаляем связанные файлы
            foreach (var file in lstFile)
            {
                //удаляем из хранилища0
                //if (App.ProxyFS.DeleteFile(file.Name))
                //{
                //удаляем с рабочего стола
            }
        }
    }

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        if (File.Exists(workdir + "\\" + file.FullPath + "\\" + file.Name))
        {
            //Удаляем файл с рабочего стола
            DirectoryInfo inf = new DirectoryInfo(workdir + "\\" + file.FullPath);
            DirectoryInfo parent = inf.Parent;
            Directory.Delete(workdir + "\\" + file.FullPath, true);
            //проверяем не пустая ли родительская директория файла
            if (parent.GetDirectories().Length == 0)
                Directory.Delete(parent.FullName);
        }
        ///получаем связанный документ
        //Filter[":_id"] = docid;
        //List<Document> lstDoc = Select(true, "");
        //Document document = lstDoc[0];
        ///удаляем из базы
        //filesManager.Update(file, document, true, true);
        //}
    }
}
return true;
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.Delete", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
    return false;
}
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="doc">Документ</param>
public void Update(Document doc)
{
    string query = "CALL
sp_updatedocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_lockid, :_lockhost, :_typeid, :_routeid, :_stageid, " +
        ":_editdate, :_designation, :_description, :_editauthor, :_stagenum); ";

    var param = new Hashtable();

    param.Add(":_id", doc.ID);
    param.Add(":_parentid", doc.ParentID);
    param.Add(":_name", doc.Name);
    param.Add(":_currhid", doc.CurHID);
    param.Add(":_remhid", doc.RemHID);
    param.Add(":_lockid", doc.LockID);
    param.Add(":_lockhost", doc.LockHost);
    param.Add(":_typeid", doc.TypeID);
    param.Add(":_routeid", doc.RouteID);
    param.Add(":_stageid", doc.StageID);
    param.Add(":_editdate", doc.EditDate);
    param.Add(":_designation", doc.Designation);
    param.Add(":_description", doc.Description);
    param.Add(":_editauthor", doc.EditAuthorID);
    param.Add(":_stagenum", doc.StageNum);

    try
    {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

}

public bool Equal(Document doc1, Document doc2)
{
    if (doc1.ID == doc2.ID && doc1.LockID == doc2.LockID &&
        doc1.Name == doc2.Name && doc1.ParentID == doc2.ParentID &&
        doc1.RemHID == doc2.RemHID && doc1.RouteID == doc2.RouteID &&
        doc1.StageID == doc2.StageID && doc1.TypeID == doc2.TypeID &&
        doc1.AuthorID == doc2.AuthorID && doc1.CreateDate == doc2.CreateDate &&
        doc1.CurHID == doc2.CurHID && doc1.Description == doc2.Description &&
        doc1.Designation == doc2.Designation && doc1.EditAuthorID == doc2.EditAuthorID &&
        doc1.EditDate == doc2.EditDate)
        return true;
    else
        return false;
}

public bool Delete(Document doc)
{
    List<DocFile> listFiles = new List<DocFile>();
    FilesManager filesManager = new FilesManager();
    filesManager.Filter["_docid"] = doc.ID;
    listFiles = filesManager.Select(true);

    Hashtable querys = new Hashtable();
    Hashtable param = new Hashtable();

    string path = "";

    StringBuilder filesIds = new StringBuilder();
    if (listFiles.Count > 0)
    {
        foreach (var file in listFiles)
        {
            if (filesIds.Length == 0)
                filesIds.Append(file.ID.ToString() + ",");
            else if (!filesIds.ToString().Contains(file.ID.ToString()))
                filesIds.Append(file.ID.ToString() + ",");
        }
        filesIds.Remove(filesIds.Length - 1, 1);
    }

    // Чистим историю
    string queryDelHis =
        "DELETE FROM \"History\" WHERE \"ObjectType\" = :_objecttype AND \"ObjectID\" IN (:_ids); ";

    if (filesIds.Length > 0)
    {
        //1. Чистим историю, связанную с файлами
        Hashtable parDelFileHis = new Hashtable();
        parDelFileHis.Add("_objecttype", (int)HistoryObjectEnum.File);
    }

```

Изм.	Лист	№ докум.	Подпись	Дат

```

parDelFileHis.Add(":_ids", fileIds.ToString());

//2. Удаляем все записи о файлах
string queryDelFiles = "DELETE FROM \"Files\" WHERE \"DocID\" = :_docid; ";
Hashtable parDelFiles = new Hashtable();
parDelFiles.Add(":_docid", doc.ID);

querys.Add("DelFiles", queryDelFiles);

param.Add("DelFiles", parDelFiles);
param.Add("DelFilesHis", parDelFileHis);
}

//3. Чистим историю, связанную с документом
Hashtable parDelDocHis = new Hashtable();
parDelDocHis.Add(":_objecttype", (int)HistoryObjectEnum.Document);
parDelDocHis.Add(":_ids", doc.ID);

//4. Удаляем все разрешения, связанные с документом
string queryDelAccess = "DELETE FROM \"Access\" WHERE \"ObjectID\" = :_objectid; ";
Hashtable parDelAccess = new Hashtable();
parDelAccess.Add(":_objectid", doc.ID);

//5. Удаляем подписи, связанные с документом
string queryDelSign = "DELETE FROM \"Signatures\" WHERE \"ObjectID\" = :_objectid; ";
Hashtable parDelSign = new Hashtable();
parDelSign.Add(":_objectid", doc.ID);

//5. Удаляем документ
string queryDelDoc = "DELETE FROM \"Documents\" WHERE \"ID\" = :_id; ";
Hashtable parDelDoc = new Hashtable();
parDelDoc.Add(":_id", doc.ID);

querys.Add("DelDoc", queryDelDoc);
querys.Add("DelAccess", queryDelAccess);
querys.Add("DelHis", queryDelHis);
querys.Add("DelSign", queryDelSign);

param.Add("DelDocHis", parDelDocHis);

param.Add("DelDoc", parDelDoc);
param.Add("DelAccess", parDelAccess);
param.Add("DelSign", parDelSign);

try
{
    App.Proxy.DeleteDocument(querys, param, null, true);
    //5. Удаляем все связанные файлы из хранилища
    foreach (var file in listFiles)
    {
        path = file.FullPath.Substring(0, file.FullPath.IndexOf("\\"));
        App.ProxyFS.DeleteFile(path);
    }
    return true;
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.Delete", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

Изм.	Лист	№ докум.	Подпись	Дат

```

        return false;
    }
}

public List<Document> SelectDeleted(bool isfolder, string where)
{
    List<Document> lst = new List<Document>();

    string query = "";

    if (isfolder)
    {
        query = "SELECT * FROM \"Documents\" d1 INNER JOIN " +
            "(SELECT \"ID\", MAX(\"CurHID\") \"CurHID\" FROM \"Documents\" " +
            "WHERE \"RemHID\" > 0 AND \"TypeID\" = 0 AND \"ID\" = COALESCE(:_id, \"ID\") " +
            "AND \"ParentID\" = COALESCE(:_parentid, \"ParentID\") " +
            "AND \"ID\" NOT IN(SELECT \"ID\" FROM \"Documents\" WHERE \"RemHID\" = 0 " +
            "AND \"TypeID\" = 0 AND \"ID\" = COALESCE(:_id, \"ID\") " +
            "AND \"ParentID\" = COALESCE(:_parentid, \"ParentID\")) " +
            "GROUP BY \"ID\") d2 " +
            "ON d1.\"ID\" = d2.\"ID\" AND d1.\"CurHID\" = d2.\"CurHID\"; ";
    }
    else
    {
        query = "SELECT * FROM \"Documents\" d1 INNER JOIN " +
            "(SELECT \"ID\", MAX(\"CurHID\") \"CurHID\" FROM \"Documents\" " +
            "WHERE \"RemHID\" > 0 AND \"TypeID\" != 0 " +
            "AND \"ParentID\" = COALESCE(:_parentid, \"ParentID\") " +
            "AND \"ID\" NOT IN(SELECT \"ID\" FROM \"Documents\" WHERE \"RemHID\" = 0
AND \"TypeID\" != 0) " +
            "AND \"ParentID\" = COALESCE(:_parentid, \"ParentID\") " +
            "GROUP BY \"ID\") d2 " +
            "ON d1.\"ID\" = d2.\"ID\" AND d1.\"CurHID\" = d2.\"CurHID\" " + where + ";";
    }

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Document doc = new Document(Int64.Parse(t[0])
                , Int64.Parse(t[1])
                , t[2]
                , Int64.Parse(t[3])
                , Int64.Parse(t[4])
                , Int32.Parse(t[5])
                , Int32.Parse(t[6])
                , t[7]
                , Int32.Parse(t[8])
                , Int32.Parse(t[9])
                , Int32.Parse(t[10])
                , DateTime.Parse(t[11])
                , DateTime.Parse(t[12])
                , t[13]
                , t[14]
                , Int32.Parse(t[15])
                , Int32.Parse(t[16]));
            lst.Add(doc);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.SelectDeleted", MessageType.Error,
e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxImage.Error);
    }

    return lst;
}
/// <summary>
/// Восстанавливает удалённый документ
/// </summary>
/// <param name="restlist"></param>
/// <param name="emplid"></param>
/// <returns></returns>
public bool RestoreDocument(List<Document> restlist, int emplid)
{
    List<Hashtable> parHis = new List<Hashtable>();
    List<Hashtable> parDoc = new List<Hashtable>();

    foreach (var doc in restlist)
    {
        History history = new History(0, (int)HistoryObjectEnum.Document, doc.ID, DateTime.Now, emplid
, Environment.MachineName, "Документ восстановлен");

        var paramHis = new Hashtable();
        paramHis.Add(":_objecttype", history.ObjectType);
        paramHis.Add(":_objectid", history.ObjectID);
        paramHis.Add(":_datechange", history.DateChange);
        paramHis.Add(":_changeauthor", history.ChangeAuthor);
        paramHis.Add(":_changehost", history.ChangeHost);
        paramHis.Add(":_action", history.Action);

        parHis.Add(paramHis);

        var paramDoc = new Hashtable();
        paramDoc.Add(":_id", doc.ID);
        paramDoc.Add(":_parentid", doc.ParentID);
        paramDoc.Add(":_name", doc.Name);
        paramDoc.Add(":_currhid", doc.CurHID);
        paramDoc.Add(":_remhid", 0);
        paramDoc.Add(":_authorid", doc.AuthorID);
        paramDoc.Add(":_lockid", doc.LockID);
        paramDoc.Add(":_createdate", doc.CreateDate);
        paramDoc.Add(":_editdate", DateTime.Now);
        paramDoc.Add(":_lockhost", doc.LockHost);
        paramDoc.Add(":_routeid", doc.RouteID);
        paramDoc.Add(":_stageid", doc.StageID);
        paramDoc.Add(":_typeid", doc.TypeID);
        paramDoc.Add(":_designation", doc.Designation);
        paramDoc.Add(":_description", doc.Description);
        paramDoc.Add(":_editauthor", emplid);
        paramDoc.Add(":_stagenum", doc.StageNum);

        parDoc.Add(paramDoc);
    }

    //1. Инсерт в History
    string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";

    //2. Insert в Document
    string queryInsDoc = "CALL
sp_adddocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_authorid, :_lockid, :_lockhost, :_routeid, :_stagei
d, :_typeid, "

```

					ПГУ 1.090501.03.001 ПЗ	Лист 280
Изм.	Лист	№ докум.	Подпись	Дат		

```

        ":_createdate, :_editdate, :_designation, :_description, :_editauthor, :_stagenum); ";

//4. Получаем последний HID
string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1";

//Запросы
Hashtable queryys = new Hashtable();

queryys.Add("InsertDocument", queryInsDoc);
queryys.Add("InsertHistory", queryHis);
queryys.Add("LastHistory", queryLastHis);

try
{
    App.Proxy.RestoreDocument(queryys, parDoc, parHis);
    return true;
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.DocumentsManager.RestoreDocument",
    MessageType.Error, e, DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
    return false;}

    }
}
}

```

EmployeesManager.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections;
using System.Diagnostics;
using System.IO;
using Utils;
using System.ServiceModel;
using System.Windows;
using Utils.Config;

namespace Doc.App
{
    public class EmployeesManager
    {
        public Hashtable Filter = new Hashtable();

        public EmployeesManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_firstname", null);
            Filter.Add(":_middlename", null);
            Filter.Add(":_typedoc", null);
            Filter.Add(":_lastname", null);
            Filter.Add(":_doc", null);
            Filter.Add(":_appointid", null);
            Filter.Add(":_rankid", null);
            Filter.Add(":_subdivid", null);
            Filter.Add(":_birthday", null);
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Filter.Add(":_deleted", null);
    }

    public void Insert(Employee employee)
    {
        string query = "CALL
sp_addemployee(:_lastname, :_firstname, :_middlename, :_typedoc, :_doc, :_addrid, :_subdivid, " +
        ":_appointid, :_photoid, :_rankid, :_birthday, :_workphone, :_homephone, :_mobilephone, :_deleted)";

        var param = new Hashtable();
        param.Add(":_lastname", employee.LastName);
        param.Add(":_firstname", employee.FirstName);
        param.Add(":_middlename", employee.MiddleName);
        param.Add(":_typedoc", employee.TypeDoc);
        param.Add(":_doc", employee.Doc);
        param.Add(":_addrid", employee.AddrID);
        param.Add(":_subdivid", employee.SubdivID);
        param.Add(":_appointid", employee.AppointID);
        param.Add(":_photoid", employee.PhotoID);
        param.Add(":_rankid", employee.RankID);
        param.Add(":_birthday", employee.Birthday);
        param.Add(":_workphone", employee.WorkPhone);
        param.Add(":_homephone", employee.HomePhone);
        param.Add(":_mobilephone", employee.MobilePhone);
        param.Add(":_deleted", employee.Deleted ? "1" : "0");

        try
        {
            App.Proxy.ExecuteNonQuery(query, param);
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.EmployeesManager.Insert", MessageType.Error, e,
            DateTime.Now);
            string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(_logFile, log);
        }
    }

    /// <summary>
    /// Получить список по фильтру
    /// </summary>
    /// <returns></returns>
    public List<Employee> Select(bool withdel)
    {
        List<Employee> list = new List<Employee>();

        if (withdel)
            Filter[":_deleted"] = null;
        else Filter[":_deleted"] = "0";

        string query =
            "SELECT * FROM fn_getemployes(:_id, :_lastname, :_firstname, :_middlename, :_typedoc, :_doc, " +
            ":_subdivid, :_appointid, :_rankid, :_birthday, :_deleted)";
        try
        {
            ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
            foreach (string item in arr)
            {
                string[] t = item.Split('|');
                Employee empl = new Employee(Int32.Parse(t[0]), t[1], t[2], t[3], Int32.Parse(t[4]), t[5],
                Int32.Parse(t[6]), Int32.Parse(t[7]),

```

```

        Int32.Parse(t[8]), Int32.Parse(t[9]), Int32.Parse(t[10]), DateTime.Parse(t[11]), t[12], t[13], t[14],
        Boolean.Parse(t[15]));
        list.Add(empl);
    }
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
    DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

Filter.Clear();
Filter.Add(":_id", null);
Filter.Add(":_firstname", null);
Filter.Add(":_middlename", null);
Filter.Add(":_typedoc", null);
Filter.Add(":_lastname", null);
Filter.Add(":_doc", null);
Filter.Add(":_appointid", null);
Filter.Add(":_rankid", null);
Filter.Add(":_subdivid", null);
Filter.Add(":_birthday", null);
Filter.Add(":_deleted", null);

return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="empl">Сотрудник</param>
public void Update(Employee empl)
{
    string query = "CALL sp_updateemployee(:_id, :_lastname, :_firstname, :_middlename, :_typedoc, :_doc,
:_addrid, " +
        ":_subdivid, :_appointid, :_rankid, :_photoid, :_birthday, :_workphone, :_homephone, " +
        ":_mobilephone, :_deleted); ";

    var param = new Hashtable();

    param.Add(":_id", empl.ID);
    param.Add(":_lastname", empl.LastName);
    param.Add(":_firstname", empl.FirstName);
    param.Add(":_middlename", empl.MiddleName);
    param.Add(":_typedoc", empl.TypeDoc);
    param.Add(":_doc", empl.Doc);
    param.Add(":_addrid", empl.AddrID);
    param.Add(":_subdivid", empl.SubdivID);
    param.Add(":_appointid", empl.AppointID);
    param.Add(":_rankid", empl.RankID);
    param.Add(":_photoid", empl.PhotoID);
    param.Add(":_birthday", empl.Birthday);
    param.Add(":_workphone", empl.WorkPhone);
    param.Add(":_homephone", empl.HomePhone);
    param.Add(":_mobilephone", empl.MobilePhone);
    param.Add(":_deleted", empl.Deleted ? "1" : "0");

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
        if (empl.Deleted)

```

```

    {
        //удаляем связанных пользователей
        UserManager userManager = new UserManager();
        userManager.Filter[":_emplid"] = empl.ID;
        List<User> list = userManager.Select();
        foreach (User user in list)
        {
            userManager.Delete(user);
        }
    }
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.EmployeesManager.Update", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

public Employee SelectLast()
{
    Employee empl = null;
    string query = "SELECT * FROM \"Employees\" ORDER BY \"ID\" DESC LIMIT 1;";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        empl = new Employee(Int32.Parse(t[0]), t[1], t[2], t[3], Int32.Parse(t[4]), t[5], Int32.Parse(t[6]),
Int32.Parse(t[7]),
        Int32.Parse(t[8]), Int32.Parse(t[9]), Int32.Parse(t[10]), DateTime.Parse(t[11]), t[12], t[13], t[14],
Boolean.Parse(t[15]));
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return empl;
}
}
}

```

FilesManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.ServiceModel.Security;
using System.Text;
using System.Threading.Tasks;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления записями о файлах
    /// </summary>
    class FileManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса DocumentsManager
        /// </summary>
        public FileManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
            Filter.Add(":_fullpath", null);
            Filter.Add(":_curhid", null);
            Filter.Add(":_remhid", null);
            Filter.Add(":_docid", null);
            Filter.Add(":_version", null);
        }

        /// <summary>
        /// Добавляет новую запись о файле
        /// </summary>
        /// <param name="file">Запись о файле</param>
        public void Insert(DocFile file, Document doc)
        {
            //1. Делаем update документа(editdate, editauthor)
            string queryUpdDoc = "CALL
sp_updatedocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_lockid, :_lockhost, :_typeid, :_routeid, :_stag
eid, " +
                ":_editdate, :_designation, :_description, :_editauthor, :_stagenum); ";

            var paramDoc = new Hashtable();

            paramDoc.Add(":_id", doc.ID);
            paramDoc.Add(":_parentid", doc.ParentID);
            paramDoc.Add(":_name", doc.Name);
            paramDoc.Add(":_currhid", doc.CurHID);
            paramDoc.Add(":_remhid", doc.RemHID);
            paramDoc.Add(":_lockid", doc.LockID);
            paramDoc.Add(":_lockhost", doc.LockHost);
            paramDoc.Add(":_typeid", doc.TypeID);
            paramDoc.Add(":_routeid", doc.RouteID);
            paramDoc.Add(":_stageid", doc.StageID);
            paramDoc.Add(":_editdate", doc.EditDate);
            paramDoc.Add(":_designation", doc.Designation);
            paramDoc.Add(":_description", doc.Description);
            paramDoc.Add(":_editauthor", doc.EditAuthorID);
            paramDoc.Add(":_stagenum", doc.StageNum);
            //2. Получаем последний ID файла
            string queryLastFile = "SELECT \"ID\" FROM \"Files\" ORDER BY \"ID\" DESC LIMIT 1";

            //3. Инсерт в History
            string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";

            History history = new History(0, (int)HistoryObjectEnum.File, file.ID, doc.EditDate, doc.EditAuthorID
, Environment.MachineName, "Создан новый файл");

            var paramHis = new Hashtable();

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

paramHis.Add(":_objecttype", history.ObjectType);
paramHis.Add(":_objectid", history.ObjectID);
paramHis.Add(":_datechange", history.DateChange);
paramHis.Add(":_changeauthor", history.ChangeAuthor);
paramHis.Add(":_changehost", history.ChangeHost);
paramHis.Add(":_action", history.Action);

//4. Получаем последний HID
string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1";

//5. Insert в Files
string queryInsFile = "CALL sp_addfile(:_id, :_name, :_fullpath, :_currhid, :_remhid, :_docid, :_version); ";

var paramFile = new Hashtable();
paramFile.Add(":_id", file.ID);
paramFile.Add(":_name", file.Name);
paramFile.Add(":_fullpath", file.FullPath);
paramFile.Add(":_currhid", file.CurHID);
paramFile.Add(":_remhid", file.RemHID);
paramFile.Add(":_docid", file.DocID);
paramFile.Add(":_version", file.Version);

//Запросы
Hashtable querys = new Hashtable();
querys.Add("LastFile", queryLastFile);
querys.Add("UpdateDocument", queryUpdDoc);
querys.Add("InsertFile", queryInsFile);
querys.Add("InsertHistory", queryHis);
querys.Add("LastHistory", queryLastHis);
// querys.Add("UpdateFile", queryUpdFile);

//параметры
Hashtable param = new Hashtable();
param.Add("Document", paramDoc);
param.Add("History", paramHis);
param.Add("File", paramFile);
// param.Add("UpdFile", paramUpdFile);

try
{
    App.Proxy.InsertFile(querys, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.FilesManager.Insert", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

}

/// <summary>
/// Получить список по фильтру
/// </summary>
/// <returns></returns>
public List<DocFile> Select(bool deleted)
{
    List<DocFile> list = new List<DocFile>();
    string query = "";

    if (!deleted)
    {
        Filter[":_remhid"] = 0;
    }

```

```

    }

    query = "SELECT * FROM fn_getfiles(:_id, :_name, :_fullpath, :_curhid, :_remhid, :_docid, :_version);";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            DocFile file = new DocFile(Int64.Parse(t[0]), t[1], t[2], Int64.Parse(t[3]), Int64.Parse(t[4]),
                Int64.Parse(t[5]), Int32.Parse(t[6]));
            list.Add(file);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.FilesManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_name", null);
    Filter.Add(":_fullpath", null);
    Filter.Add(":_curhid", null);
    Filter.Add(":_remhid", null);
    Filter.Add(":_docid", null);
    Filter.Add(":_version", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="file">Запись о файле</param>
public void Update(DocFile file, Document doc, bool del, bool deldoc)
{
    Hashtable querys = new Hashtable();
    Hashtable param = new Hashtable();
    string queryUpdDoc = "";
    var paramDoc = new Hashtable();

    if (!deldoc)
    {
        //1. Делаем update документа(editdate, editauthor)
        queryUpdDoc = "CALL
sp_updatedocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_lockid, :_lockhost, :_typeid, :_routeid, :_stag
eid, " +
            ":_editdate, :_designation, :_description, :_editauthor, :_stagenum);";

        paramDoc.Add(":_id", doc.ID);
        paramDoc.Add(":_parentid", doc.ParentID);
        paramDoc.Add(":_name", doc.Name);
        paramDoc.Add(":_currhid", doc.CurHID);
        paramDoc.Add(":_remhid", doc.RemHID);
        paramDoc.Add(":_lockid", doc.LockID);
        paramDoc.Add(":_lockhost", doc.LockHost);
    }

```

Изм.	Лист	№ докум.	Подпись	Дат

```

        paramDoc.Add(":_typeid", doc.TypeID);
        paramDoc.Add(":_routeid", doc.RouteID);
        paramDoc.Add(":_stageid", doc.StageID);
        paramDoc.Add(":_editdate", doc.EditDate);//
        paramDoc.Add(":_designation", doc.Designation);
        paramDoc.Add(":_description", doc.Description);
        paramDoc.Add(":_editauthor", doc.EditAuthorID);
        paramDoc.Add(":_stagenum", doc.StageNum);//
    }

    //2. Insert в History
    string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";
    History history;
    if (del)
    {
        history = new History(0, (int)HistoryObjectEnum.File, file.ID, doc.EditDate, doc.EditAuthorID
        , Environment.MachineName, "Файл удалён");
    }

    else
    {
        history = new History(0, (int)HistoryObjectEnum.File, file.ID, doc.EditDate, doc.EditAuthorID
        , Environment.MachineName, "Файл изменён");
    }

    var paramHis = new Hashtable();
    paramHis.Add(":_objecttype", history.ObjectType);
    paramHis.Add(":_objectid", history.ObjectID);
    paramHis.Add(":_datechange", history.DateChange);
    paramHis.Add(":_changeauthor", history.ChangeAuthor);
    paramHis.Add(":_changehost", history.ChangeHost);
    paramHis.Add(":_action", history.Action);

    //3. Получаем последний HID
    string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1";
    //4. Update старого файла в remhid пишем последний HID, в имя файла
    //то, что вернулось после сохранения
    string queryUpdFile = "CALL sp_updatefile(:_id, :_currhid, :_remhid, :_name, :_version); ";

    var paramUpdFile = new Hashtable();

    paramUpdFile.Add(":_id", file.ID);
    paramUpdFile.Add(":_remhid", file.RemHID);//HID
    paramUpdFile.Add(":_currhid", file.CurHID);
    paramUpdFile.Add(":_name", file.Name);
    paramUpdFile.Add(":_version", file.Version);
    //5. Определяем последнюю версию файла

    string queryLastVersion =
        "SELECT MAX(\"Version\") FROM \"Files\" WHERE \"ID\" = :_id AND \"DocID\" = :_docid GROUP
BY \"ID\"; ";
    Hashtable parLastVersion = new Hashtable();
    parLastVersion.Add(":_id", file.ID);
    parLastVersion.Add(":_docid", file.DocID);
    //6. Создаём новую запись с id от старого, curhid = HID и remhid = 0
    if(!del)
    {
        string queryInsFile = "CALL sp_addfile(:_id, :_name, :_fullpath, :_currhid, :_remhid, :_docid, :_version);
";

        var paramFile = new Hashtable();
        paramFile.Add(":_id", file.ID);
        paramFile.Add(":_name", file.Name);

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

paramFile.Add(":_fullpath", file.FullPath);//путь тоже последняя версия + 1
paramFile.Add(":_currhid", file.CurHID);//HID
paramFile.Add(":_remhid", file.RemHID);
paramFile.Add(":_docid", file.DocID);
paramFile.Add(":_version", file.Version);//меняем на последнюю + 1

querys.Add("InsertFile", queryInsFile);
param.Add("File", paramFile);
}

//Запросы
if (!deldoc)
{
    querys.Add("UpdateDocument", queryUpdDoc);
    param.Add("Document", paramDoc);
}

querys.Add("InsertHistory", queryHis);
querys.Add("LastHistory", queryLastHis);
querys.Add("UpdateFile", queryUpdFile);
querys.Add("LastVersion", queryLastVersion);

//параметры
param.Add("History", paramHis);
param.Add("UpdFile", paramUpdFile);
param.Add("LastVersion", parLastVersion);

try
{
    App.Proxy.UpdateFile(querys, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.FilesManager.Update", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
/// <summary>
/// Заменяет версию файла
/// </summary>
/// <param name="newfile">Новая версия</param>
/// <param name="oldFile">Старая версия</param>
/// <param name="doc">Связанный документ</param>
public void SetCurrentVersion(DocFile newfile, DocFile oldFile, Document doc)
{
    Hashtable querys = new Hashtable();
    Hashtable param = new Hashtable();
    string queryUpdDoc = "";
    var paramDoc = new Hashtable();

    //1. Делаем update документа(editdate, editauthor)
    queryUpdDoc = "CALL
sp_updatedocument(:_id, :_parentid, :_name, :_currhid, :_remhid, :_lockid, :_lockhost, :_typeid, :_routeid, :_stag
eid, " +
        ":_editdate, :_designation, :_description, :_editauthor, :_stagenum); ";

    paramDoc.Add(":_id", doc.ID);
    paramDoc.Add(":_parentid", doc.ParentID);
    paramDoc.Add(":_name", doc.Name);
    paramDoc.Add(":_currhid", doc.CurHID);

```

Изм.	Лист	№ докум.	Подпись	Дат

```

paramDoc.Add(":_remhid", doc.RemHID);
paramDoc.Add(":_lockid", doc.LockID);
paramDoc.Add(":_lockhost", doc.LockHost);
paramDoc.Add(":_typeid", doc.TypeID);
paramDoc.Add(":_routeid", doc.RouteID);
paramDoc.Add(":_stageid", doc.StageID);
paramDoc.Add(":_editdate", doc.EditDate);//
paramDoc.Add(":_designation", doc.Designation);
paramDoc.Add(":_description", doc.Description);
paramDoc.Add(":_editauthor", doc.EditAuthorID);//
paramDoc.Add(":_stagenum", doc.StageNum);

//2. Insert в History
string queryHis = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";
History history;

if(oldFile != null)
    history = new History(0, (int)HistoryObjectEnum.File, oldFile.ID, doc.EditDate, doc.EditAuthorID
        , Environment.MachineName, "Версия файла " + oldFile.Version.ToString() + " заменена на
версию " + newfile.Version.ToString());
else
    history = new History(0, (int)HistoryObjectEnum.File, newfile.ID, doc.EditDate, doc.EditAuthorID
        , Environment.MachineName, "Восстановлена версия файла " + newfile.Version.ToString());

var paramHis = new Hashtable();
paramHis.Add(":_objecttype", history.ObjectType);
paramHis.Add(":_objectid", history.ObjectID);
paramHis.Add(":_datechange", history.DateChange);
paramHis.Add(":_changeauthor", history.ChangeAuthor);
paramHis.Add(":_changehost", history.ChangeHost);
paramHis.Add(":_action", history.Action);

//3. Получаем последний HID
string queryLastHis = "SELECT \"HID\" FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1";

if (oldFile != null)
{
    //4.Update старого файла
    string queryUpdFile = "CALL sp_updatefile(:_id, :_currhid, :_remhid, :_name, :_version); ";

    var paramUpdFile = new Hashtable();

    paramUpdFile.Add(":_id", oldFile.ID);
    paramUpdFile.Add(":_remhid", oldFile.RemHID);//HID
    paramUpdFile.Add(":_currhid", oldFile.CurHID);
    paramUpdFile.Add(":_name", oldFile.Name);
    paramUpdFile.Add(":_version", oldFile.Version);

    queries.Add("UpdateFile", queryUpdFile);
    param.Add("UpdOldFile", paramUpdFile);
}

//5. Insert новой строки с 0 RemHID

string queryInsFile = "CALL sp_addfile(:_id, :_name, :_fullpath, :_currhid, :_remhid, :_docid, :_version); ";

var paramInsNewFile = new Hashtable();

paramInsNewFile.Add(":_id", newfile.ID);
paramInsNewFile.Add(":_name", newfile.Name);
paramInsNewFile.Add(":_fullpath", newfile.FullPath);
paramInsNewFile.Add(":_remhid", newfile.RemHID);//0
paramInsNewFile.Add(":_currhid", newfile.CurHID);//меняется на последний

```

Изм.	Лист	№ докум.	Подпись	Дат

```

paramInsNewFile.Add(":_docid", newfile.DocID);
paramInsNewFile.Add(":_version", newfile.Version);

quers.Add("UpdateDocument", queryUpdDoc);
quers.Add("InsertHistory", queryHis);
quers.Add("LastHistory", queryLastHis);
quers.Add("InsertFile", queryInsFile);

param.Add("Document", paramDoc);
param.Add("History", paramHis);
param.Add("InsNewFile", paramInsNewFile);

try
{
    App.Proxy.SetFileCurrentVersion(quers, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.FilesManager.SetCurrentVersion", MessageType.Error,
e, DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}
}

```

HistoryManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления историческими данными
    /// </summary>
    class HistoryManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса HistoryManager
        /// </summary>
        public HistoryManager()
        {
            Filter.Add(":_hid", null);
            Filter.Add(":_objecttype", null);
            Filter.Add(":_objectid", null);
            Filter.Add(":_datechange", null);
            Filter.Add(":_changeauthor", null);
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Добавляет новую историческую запись
/// </summary>
/// <param name="history">Историческая запись</param>
public void Insert(History history)
{
    string query = "CALL
sp_addhistory(:_objecttype, :_objectid, :_datechange, :_changeauthor, :_changehost, :_action); ";

    var param = new Hashtable();
    param.Add(":_objecttype", history.ObjectType);
    param.Add(":_objectid", history.ObjectID);
    param.Add(":_datechange", history.DateChange);
    param.Add(":_changeauthor", history.ChangeAuthor);
    param.Add(":_changehost", history.ChangeHost);
    param.Add(":_action", history.Action);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.HistoryManager.Insert", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }
}

/// <summary>
/// Получить список по фильтру
/// </summary>
/// <returns></returns>
public List<History> Select()
{
    List<History> list = new List<History>();

    string query = "SELECT * FROM
fn_gethistory(:_hid, :_objecttype, :_objectid, :_datechange, :_changeauthor); ";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            History history = new History(Int64.Parse(t[0]), Int32.Parse(t[1]), Int64.Parse(t[2]),
DateTime.Parse(t[3])
, Int32.Parse(t[4]), t[5], t[6]);
            list.Add(history);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.HistoryManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

Filter.Clear();
Filter.Add(":_hid", null);
Filter.Add(":_objecttype", null);
Filter.Add(":_objectid", null);
Filter.Add(":_datechange", null);
Filter.Add(":_changeauthor", null);

return list;
}
/// <summary>
/// Получает исторические данные о файлах документа
/// </summary>
/// <returns></returns>
public List<HisFile> SelectFilesHistory(long docid)
{
    List<HisFile> lst = new List<HisFile>();

    string query = "SELECT f.\"ID\" fileid, f.\"CurHID\", f.\"Name\", f.\"Version\" +
        \", f.\"RemHID\", h.\"DateChange\", e.\"ID\" EmplID +
        \", (e.\"LastName\" || ' ' || e.\"FirstName\" || ' ' || e.\"MiddleName\") fio +
        \", h.\"ChangeHost\", h.\"Action\", f.\"FullPath\" \" +
        \"FROM \"History\" h INNER JOIN \"Files\" f \" +
        \" ON(h.\"ObjectID\" = f.\"ID\" AND ((h.\"HID\" = f.\"CurHID\" AND f.\"RemHID\" > h.\"HID\") \" +
        \"OR(h.\"HID\" = f.\"CurHID\" AND f.\"RemHID\" = 0) OR (f.\"CurHID\" < h.\"HID\" \" +
        \"AND f.\"RemHID\" = h.\"HID\" AND h.\"Action\" LIKE '%удалён')) \" +
        \"INNER JOIN \"Employee\" e ON h.\"ChangeAuthor\" = e.\"ID\" \" +
        \"WHERE h.\"ObjectType\" = :_type AND f.\"DocID\" = :_docid\" ;

    Hashtable param = new Hashtable();
    param.Add(":_type", (int)HistoryObjectEnum.File);
    param.Add(":_docid", docid);
    try
    {
        ArrayList arrList = App.Proxy.ExecuteQuery(query, param);
        if(arrList.Count > 0)
            foreach (string arr in arrList)
            {
                string[] t = arr.Split('|');
                HisFile his = new HisFile(Int64.Parse(t[0]), Int64.Parse(t[1]), t[2], Int32.Parse(t[3]),
                    Int64.Parse(t[4]), DateTime.Parse(t[5]), Int32.Parse(t[6]), t[7], t[8], t[9], t[10]);
                lst.Add(his);
            }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.HistoryManager.SelectFilesHistory", MessageType.Error,
e, DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }
    return lst;
}

public History SelectLast()
{
    History history = null;
    string query = "SELECT * FROM \"History\" ORDER BY \"HID\" DESC LIMIT 1; ";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = arr[0].ToString().Split('|');

```

```

        history = new History(Int64.Parse(t[0]), Int32.Parse(t[1]), Int64.Parse(t[2]), DateTime.Parse(t[3])
            , Int32.Parse(t[4]), t[5], t[6]);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.HistoryManager.Select", MessageType.Error, e,
            DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_hid", null);
    Filter.Add(":_objecttype", null);
    Filter.Add(":_objectid", null);
    Filter.Add(":_datechange", null);
    Filter.Add(":_changeauthor", null);

    return history;
}
}
}

```

MilitaryRanksManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления справочником MilitaryRanks
    /// </summary>
    class MilitaryRanksManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса MilitaryRanksManager
        /// </summary>
        public MilitaryRanksManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
        }

        /// <summary>
        /// Создаёт новое воинское звание
        /// </summary>
        /// <param name="rank">Должность</param>
        public void Insert(MilitaryRank rank)
        {
            string query = "CALL sp_addmilitaryrank(:_name)";

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

var param = new Hashtable();
param.Add(":_name", rank.Name);
try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.MilitaryRanksManager.Insert", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

/// <summary>
/// Получает список воинских званий
/// </summary>
/// <returns>Список воинских званий</returns>
public List<MilitaryRank> Select()
{
    List<MilitaryRank> list = new List<MilitaryRank>();

    string query =
        "SELECT * FROM fn_getmilitaryranks(:_id, :_name)";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            MilitaryRank rank = new MilitaryRank(Int32.Parse(t[0]), t[1]);
            list.Add(rank);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.MilitaryRanksManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_name", null);

    return list;
}

/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public MilitaryRank SelectLast()
{
    MilitaryRank rank = null;
    string query = "SELECT * FROM 'MilitaryRanks' ORDER BY 'ID' DESC LIMIT 1;";

```

Изм.	Лист	№ докум.	Подпись	Дат

```

try
{
    ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

    string[] t = ((string)arr[0]).Split('|');
    rank = new MilitaryRank(Int32.Parse(t[0]), t[1]);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

return rank;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="rank">Воинское звание</param>
public void Update(MilitaryRank rank)
{
    string query = "CALL sp_updatemilitaryrank(:_id, :_name); ";

    var param = new Hashtable();
    param.Add(":_id", rank.ID);
    param.Add(":_name", rank.Name);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.MilitaryRanksManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор воинского звания</param>
public bool Delete(int id)
{
    string query = "CALL sp_delmilitaryrank(:_id); ";
    List<Employee> list = new List<Employee>();
    EmployeesManager employeesManager = new EmployeesManager();
    employeesManager.Filter[":_rankid"] = id;
    list = employeesManager.Select(true);
    if (list.Count > 0)
    {
        MessageBox.Show("Есть сотрудники с указанным званием.\nУдаление невозможно",
            "Ошибка.", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

var param = new Hashtable();
param.Add(":_id", id);
try
{
    App.Proxy.ExecuteNonQuery(query, param);
    return true;
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.MilitaryRanksManager.Delete", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxImage.Error);
    return false;
}
}
}
}

```

PhotoManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления фотографиями
    /// </summary>
    class PhotoManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса Photo
        /// </summary>
        public PhotoManager()
        {
            Filter.Add(":_id", null);
        }

        /// <summary>
        /// Создаёт новое фото
        /// </summary>
        /// <param name="photo">Фотография</param>
        public void Insert(Photo photo)
        {
            string query = "CALL sp_addphoto(:_photo)";

            var param = new Hashtable();
            param.Add(":_photo", photo.PhotoEmpl);
            try

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        {
            App.Proxy.ExecuteNonQuery(query, param);
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.PhotoManager.Insert", MessageType.Error, e,
DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
            MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    /// <summary>
    /// Получает фотографию
    /// </summary>
    /// <returns>Фотография</returns>
    public Photo Select(int id)
    {
        //TODO: заменить на отдельную функцию в DocService, которая будет возвращать List
        // Так не работает
        Photo photo = null;

        try
        {
            Hashtable arr = App.Proxy.GetPhoto(id);

            if (arr.Count > 0)
            {
                foreach (DictionaryEntry de in arr)
                {
                    photo = new Photo((int)de.Key, (byte[])de.Value);
                }
            }
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Select", MessageType.Error, e,
DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
            MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        Filter[":_id"] = null;

        return photo;
    }

    public Photo SelectLast()
    {
        Photo photo = null;
        string query =
            "SELECT * FROM fn_getphoto(:_id) ORDER BY ID DESC LIMIT 1";
        try
        {
            ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

            string[] t = ((string)arr[0]).Split('|');
            photo = new Photo(Int32.Parse(t[0]), Encoding.UTF8.GetBytes(t[1]));
        }
    }

```

Изм.	Лист	№ докум.	Подпись	Дат

```

    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.AppointmentManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    Filter[":_id"] = null;

    return photo;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="photo">Фотография</param>
public void Update(Photo photo)
{
    string query = "CALL sp_updatephoto(:_id, :_photo); ";

    var param = new Hashtable();
    param.Add(":_id", photo.ID);
    param.Add(":_photo", photo.PhotoEmpl);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PhotoManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор фотографии</param>
public void Delete(int id)
{
    string query = "CALL sp_delphoto(:_id); ";

    var param = new Hashtable();
    param.Add(":_id", id);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PhotoManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);

```

					Лист
					280
Изм.	Лист	№ докум.	Подпись	Дат	

```

        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}

```

PostMessageInManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления входящими почтовыми сообщениями
    /// </summary>
    class PostMessagesInManager
    {
        /// <summary>
        /// Фильтр
        /// </summary>
        public Hashtable Filter = new Hashtable();
        /// <summary>
        /// Конструктор класса PostMessagesInManager
        /// </summary>
        public PostMessagesInManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_touser", null);
            Filter.Add(":_fromuser", null);
            Filter.Add(":_objectid", null);
            Filter.Add(":_outid", null);
            Filter.Add(":_datebeg", null);
            Filter.Add(":_dateend", null);
        }
        /// <summary>
        /// Создаёт входящее сообщение
        /// </summary>
        /// <param name="mes">Сообщение</param>
        public void Insert(PostMessageIn mes)
        {
            string query = "CALL
sp_addpostmessagein(:_theme, :_touser, :_fromuser, :_message, :_objectid, :_readed, :_deleted, :_outid, :_date
post); ";

            var param = new Hashtable();
            param.Add(":_theme", mes.Theme);
            param.Add(":_touser", mes.ToUserID);
            param.Add(":_fromuser", mes.FromUserID);
            param.Add(":_message", mes.Message);
            param.Add(":_objectid", mes.ObjectID);
            param.Add(":_readed", mes.Readed ? "1" : "0");
            param.Add(":_deleted", mes.Deleted ? "1" : "0");
            param.Add(":_outid", mes.OutID);

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

param.Add(":_datepost", mes.DatePost);

try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.PostMessagesInManager.Insert", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
}
}
/// <summary>
/// Получает список сообщений
/// </summary>
/// <returns></returns>
public List<PostMessageIn> Select(string where)
{
    List<PostMessageIn> list = new List<PostMessageIn>();

    string query =
        string.Format("SELECT * FROM fn_getpostmessagesin(:_id, :_touser, " +
            ":_fromuser, :_objectid, :_outid, :_datebeg, :_dateend) {0}; ", where);
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            PostMessageIn mes = new PostMessageIn(Int64.Parse(t[0]), t[1], Int32.Parse(t[2]),
Int32.Parse(t[3]), t[4],
            Int64.Parse(t[5]), Boolean.Parse(t[6]), Boolean.Parse(t[7]), Int64.Parse(t[8]),
DateTime.Parse(t[9]));
            list.Add(mes);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesInManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_touser", null);
    Filter.Add(":_fromuser", null);
    Filter.Add(":_objectid", null);
    Filter.Add(":_outid", null);
    Filter.Add(":_datebeg", null);
    Filter.Add(":_dateend", null);

    return list;
}

```

/// <summary>

/// Изменяет текущую запись

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

/// </summary>
/// <param name="mes">Сообщение</param>
public void Update(PostMessageIn mes)
{
    string query = "CALL sp_updatepostmessagein(:_id, :_readed, :_deleted); ";

    var param = new Hashtable();

    param.Add(":_id", mes.ID);
    param.Add(":_readed", mes.Readed ? "1" : "0");
    param.Add(":_deleted", mes.Deleted ? "1" : "0");
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesInManager.Update", MessageType.Error,
e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор сообщения</param>
/// <param name="del">Признак окончательного удаления</param>
public void Delete(PostMessageIn mess, bool del)
{
    string query = "";
    var param = new Hashtable();

    if (del)
    {
        param.Add(":_id", mess.ID);
        query = "CALL sp_delpostmessagein(:_id); ";
        try
        {
            App.Proxy.ExecuteNonQuery(query, param);
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.PostMessagesInManager.Delete",
MessageType.Error, e, DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
            MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    else
    {
        Update(mess);
    }
}
}
}

```

PostMessageOutManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    class PostMessagesOutManager
    {
        /// <summary>
        /// Фильтр
        /// </summary>
        public Hashtable Filter = new Hashtable();
        /// <summary>
        /// Конструктор класса PostMessagesOutManager
        /// </summary>
        public PostMessagesOutManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_touser", null);
            Filter.Add(":_fromuser", null);
            Filter.Add(":_objectid", null);
            Filter.Add(":_datebeg", null);
            Filter.Add(":_dateend", null);
            Filter.Add(":_auto", null);
        }
        /// <summary>
        /// Создаёт исходящее сообщение
        /// </summary>
        /// <param name="mes">Сообщение</param>
        public void Insert(PostMessageOut mes)
        {
            string query = "CALL sp_addpostmessageout(:_theme, :_touser, :_fromuser, :_message, " +
                ":_readed, :_deleted, :_sent, :_objectid, :_datepost, :_auto); ";

            var param = new Hashtable();
            param.Add(":_theme", mes.Theme);
            param.Add(":_touser", mes.ToUserID);
            param.Add(":_fromuser", mes.FromUserID);
            param.Add(":_message", mes.Message);
            param.Add(":_readed", mes.Readed ? "1" : "0");
            param.Add(":_deleted", mes.Deleted ? "1" : "0");
            param.Add(":_sent", mes.Sent ? "1" : "0");
            param.Add(":_objectid", mes.ObjectID);
            param.Add(":_datepost", mes.DatePost);
            param.Add(":_auto", mes.IsAuto ? "1" : "0");

            try
            {
                App.Proxy.ExecuteNonQuery(query, param);
            }
            catch (Exception e)
            {
                Process proc = Process.GetCurrentProcess();
            }
        }
    }
}
```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesOutManager.Insert", MessageType.Error,
e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
    }
}
/// <summary>
/// Получает список сообщений
/// </summary>
/// <returns></returns>
public List<PostMessageOut> Select(string where)
{
    List<PostMessageOut> list = new List<PostMessageOut>();

    string query =
        string.Format("SELECT * FROM fn_getpostmessagesout(:_id, :_touser, " +
            ":_fromuser, :_objectid, :_datebeg, :_dateend, :_auto) {0}; ", where );
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split("|");
            PostMessageOut mes = new PostMessageOut(Int64.Parse(t[0]), t[1], Int32.Parse(t[2]),
Int32.Parse(t[3]), t[4]
                , Boolean.Parse(t[5]), Boolean.Parse(t[6]), Boolean.Parse(t[7])
                , Int64.Parse(t[8]), DateTime.Parse(t[9]), Boolean.Parse(t[10]));
            list.Add(mes);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesOutManager.Select", MessageType.Error,
e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_touser", null);
    Filter.Add(":_fromuser", null);
    Filter.Add(":_objectid", null);
    Filter.Add(":_datebeg", null);
    Filter.Add(":_dateend", null);
    Filter.Add(":_auto", null);

    return list;
}

/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public PostMessageOut SelectLast(int fromid)
{
    PostMessageOut mess = null;
    string query =
        string.Format("SELECT * FROM \"PostMessagesOut\" WHERE \"FromUser\" = {0} ORDER BY \"ID\"
DESC LIMIT 1", fromid);
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        string[] t = arr[0].ToString().Split("|");
        mess = new PostMessageOut(Int64.Parse(t[0]), t[1], Int32.Parse(t[2]), Int32.Parse(t[3]), t[4]
            , Boolean.Parse(t[5]), Boolean.Parse(t[6]), Boolean.Parse(t[7]), Int64.Parse(t[8]),
            DateTime.Parse(t[9]), Boolean.Parse(t[10]));

    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesOutManager.SelectLast",
MessageTypes.Error, e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
    }

    return mess;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="mes">Сообщение</param>
public void Update(PostMessageOut mes)
{
    string query = "CALL sp_updatepostmessageout(:_id, :_theme, :_touser, :_message, " +
        ":_readed, :_deleted, :_sent, :_objectid, :_datepost); ";

    var param = new Hashtable();

    param.Add(":_id", mes.ID);
    param.Add(":_theme", mes.Theme);
    param.Add(":_touser", mes.ToUserID);
    param.Add(":_message", mes.Message);
    param.Add(":_readed", mes.Readed ? "1" : "0");
    param.Add(":_deleted", mes.Deleted ? "1" : "0");
    param.Add(":_sent", mes.Sent ? "1" : "0");
    param.Add(":_objectid", mes.ObjectID);
    param.Add(":_datepost", mes.DatePost);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.PostMessagesOutManager.Update",
MessageTypes.Error, e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор сообщения</param>
/// <param name="del">Признак окончательного удаления</param>
public void Delete(PostMessageOut mess, bool del)
{
    if (del)
    {
        string query = "CALL sp_delpostmessageout(:_id); ";
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дат

```

var param = new Hashtable();

param.Add(":_id", mess.ID);

try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.PostMessagesOutManager.Delete",
MessageTypeError.Error, e, DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
else
{
    Update(mess);
}
}
}
}

```

RolesManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления ролями пользователей
    /// </summary>
    class RoleManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса RoleManager
        /// </summary>
        public RoleManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
        }
        /// <summary>
        /// Добавить новую роль
        /// </summary>
        /// <param name="role">Роль</param>
        public void Insert(Role role)
        {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string query = "CALL sp_addrole(:_name, :_controlaccess)";

var param = new Hashtable();
param.Add(":_name", role.Name);
param.Add(":_controlaccess", role.ControlAccess);
try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.RoleManager.Insert", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}
}

/// <summary>
/// Получить список по фильтру
/// </summary>
/// <returns></returns>
public List<Role> Select()
{
    List<Role> list = new List<Role>();

    string query =
        "SELECT * FROM fn_getroles(:_id, :_name)";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Role type = new Role(Int32.Parse(t[0]), t[1], t[2]);
            list.Add(type);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.RoleManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_name", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="type">Тип</param>
public void Update(Role role)
{
    string query = "CALL sp_updaterole(:_id, :_name, :_controlaccess)";

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

var param = new Hashtable();
param.Add(":_id", role.ID);
param.Add(":_name", role.Name);
param.Add(":_controlaccess", role.ControlAccess);

try
{
    App.Proxy.ExecuteNonQuery(query, param);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.RoleManager.Update", MessageType.Error, e,
DateTime.Now);
    string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(logFile, log);
    MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор роли</param>
public bool Delete(int id)
{
    string query = "CALL sp_delrole(:_id); ";

    var param = new Hashtable();
    param.Add(":_id", id);
    try
    {
        UserManager userManager = new UserManager();
        userManager.Filter[":_roleid"] = id;
        List<User> list = userManager.Select();
        if (list.Count > 0)
        {
            MessageBox.Show("Данная роль присвоена " + list.Count.ToString() +
                " пользователям. \nУдаление невозможно.", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
        else
        {
            App.Proxy.ExecuteNonQuery(query, param);
            return true;
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.RoleManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}
/// <summary>
/// Возвращает последнюю запись
/// </summary>

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <returns></returns>
public Role SelectLast()
{
    Role role = null;

    string query =
        "SELECT * FROM \"Roles\" ORDER BY \"ID\" DESC LIMIT 1; ";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        string[] t = arr[0].ToString().Split('|');
        role = new Role(Int32.Parse(t[0]), t[1], t[2]);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.RoleManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return role;
}
}
}

```

RoutesManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления маршрутами
    /// </summary>
    class RoutesManager
    {
        public delegate void SendAutomateMessage(PostMessageOut message);

        public event SendAutomateMessage OnSendAutomatetMessage;

        IDocService _proxy = null;
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса RoutesManager
        /// </summary>
        public RoutesManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        _proxy = ServerManager.GetProxyDb();
    }
    /// <summary>
    /// Добавляет маршрут
    /// </summary>
    /// <param name="route">Маршрут</param>
    public void Insert(Route route)
    {
        string query = "CALL sp_addroute(:_name, :_route);";
        var param = new Hashtable();
        param.Add(":_name", route.Name);
        param.Add(":_route", route.RouteAlg);

        try
        {
            _proxy.ExecuteNonQuery(query, param);
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.RoutesManager.Insert", MessageType.Error, e,
DateTime.Now);
            string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(_logFile, log);
        }
    }
    /// <summary>
    /// Получает список маршрутов
    /// </summary>
    /// <returns></returns>
    public List<Route> Select()
    {
        string query = "SELECT * FROM fn_getroutes(:_id, :_name);";
        List<Route> list = new List<Route>();
        try
        {
            ArrayList arr = _proxy.ExecuteQuery(query, Filter);
            foreach (string item in arr)
            {
                string[] t = item.Split('|');
                Route route = new Route(Int32.Parse(t[0]), t[1], t[2]);
                list.Add(route);
            }
        }
        catch (Exception e)
        {
            MessageBox.Show(e.Message);
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.RoutesManager.Select", MessageType.Error, e,
DateTime.Now);
            string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(_logFile, log);
        }

        Filter.Clear();
        Filter.Add(":_id", null);
        Filter.Add(":_name", null);

        return list;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="route">Маршрут</param>
public void Update(Route route)
{
    string query = "CALL sp_updateroute(:_id, :_name, :_route);";

    var param = new Hashtable();
    param.Add(":_id", route.ID);
    param.Add(":_name", route.Name);
    param.Add(":_route", route.RouteAlg);

    try
    {
        _proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.RoutesManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор маршрута</param>
public bool Delete(int id)
{
    string query = "CALL sp_delroute(:_id);";

    var param = new Hashtable();

    param.Add(":_id", id);
    try
    {
        _proxy.ExecuteNonQuery(query, param);
        return true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.RoutesManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return false;
    }
}

}
/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public Route SelectLast()
{
    Route rout = null;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string query = "SELECT * FROM \"Routes\" ORDER BY \"ID\" DESC LIMIT 1;";

try
{
    ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

    string[] t = ((string)arr[0]).Split('|');
    rout = new Route(Int32.Parse(t[0]), t[1], t[2]);
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.RoutesManager.SelectLast", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

return rout;
}

public Document GoToStage(Document doc, int fromuser, int touser,
int route, Stage stage, string theme, string message)
{
    Document newdoc = null;
    int lockid = doc.LockID;
    if (stage.IsLock)
        lockid = touser;
    try
    {
        //1.Update Doc
        newdoc = new Document(doc.ID, doc.ParentID, doc.Name,
            doc.CurHID, doc.RemHID, doc.AuthorID, lockid,
            doc.LockHost, route, stage.ID, doc.TypeID,
            doc.CreateDate, DateTime.Now, doc.Designation,
            doc.Description, doc.EditAuthorID, doc.StageNum);
        DocumentsManager documentsManager = new DocumentsManager();
        documentsManager.Update(newdoc);
        //2. Исходящее сообщение
        PostMessageOut messOut = new PostMessageOut(0,
            theme, touser, fromuser,
            message, false, false, true, newdoc.ID, DateTime.Now, true);
        PostMessagesOutManager postMessagesOutManager = new PostMessagesOutManager();
        postMessagesOutManager.Insert(messOut);
        messOut = postMessagesOutManager.SelectLast(fromuser);

        //3. Входящее сообщение
        PostMessageIn messIn = new PostMessageIn(0, messOut.Theme, messOut.ToUserID,
            messOut.FromUserID, messOut.Message, messOut.ObjectID, false, false,
            messOut.ID, DateTime.Now);
        PostMessagesInManager postMessagesInManager = new PostMessagesInManager();
        postMessagesInManager.Insert(messIn);
        //отправляем сообщение
        if (OnSendAutomatetMessage != null)
            OnSendAutomatetMessage(messOut);

        return newdoc;
    }
    catch (Exception ex)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.RoutesManager.GoToSage", MessageType.Error, ex,
DateTime.Now);

```

					ПГУ 1.090501.03.001 ПЗ		Лист
							280
Изм.	Лист	№ докум.	Подпись	Дат			

```

        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
        return null;
    }
}
}

```

SignatureManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления подписями
    /// </summary>
    public class SignaturesManager
    {
        /// <summary>
        /// Фильтр
        /// </summary>
        public Hashtable Filter = new Hashtable();
        /// <summary>
        /// Конструктор класса SignaturesManager
        /// </summary>
        public SignaturesManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_typeid", null);
            Filter.Add(":_emplid", null);
            Filter.Add(":_objectid", null);
            Filter.Add(":_signdate", null);
            Filter.Add(":_routeid", null);
        }
        /// <summary>
        /// Создаёт новую подпись
        /// </summary>
        /// <param name="signature">Подпись</param>
        public void Insert(Signature signature)
        {
            string query = "CALL sp_addsignature(:_typeid, :_emplid, :_objectid, :_signdate, :_routeid, :_stagenum);";

            var param = new Hashtable();
            param.Add(":_typeid", signature.TypeID);
            param.Add(":_emplid", signature.EmplID);
            param.Add(":_objectid", signature.ObjectID);
            param.Add(":_signdate", signature.SignDate);
            param.Add(":_routeid", signature.RouteID);
            param.Add(":_stagenum", signature.StageNum);
            try
            {
                App.Proxy.ExecuteNonQuery(query, param);
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.SignaturesManager.Insert", MessageType.Error, e,
DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
        }
    }
    /// <summary>
    /// Получает список подписей
    /// </summary>
    /// <returns></returns>
    public List<Signature> Select()
    {
        List<Signature> list = new List<Signature>();

        string query =
            "SELECT * FROM fn_getsignatures(:_id, :_typeid, :_emplid, :_objectid, :_signdate, :_routeid)";
        try
        {
            ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
            foreach (string item in arr)
            {
                string[] t = item.Split('|');
                Signature signature = new Signature(Int64.Parse(t[0]), Int32.Parse(t[1]), Int32.Parse(t[2]),
                    Int64.Parse(t[3]), DateTime.Parse(t[4]), Int32.Parse(t[5]), Int32.Parse(t[6]));
                list.Add(signature);
            }
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.SignaturesManager.Select", MessageType.Error, e,
DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
        }

        Filter.Clear();
        Filter.Add(":_id", null);
        Filter.Add(":_typeid", null);
        Filter.Add(":_emplid", null);
        Filter.Add(":_objectid", null);
        Filter.Add(":_signdate", null);
        Filter.Add(":_routeid", null);

        return list;
    }

    /// <summary>
    /// Удаляет текущую запись
    /// </summary>
    /// <param name="id">Идентификатор подписи</param>
    public void Delete(long id)
    {
        string query = "CALL sp_delsignature(:_id); ";

        var param = new Hashtable();

        param.Add(":_id", id);
        try
        {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.SignaturesManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
}

```

StagesManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

namespace Doc.App
{
    /// <summary>
    /// Класс для управления справочником стадий проектирования
    /// </summary>
    class StagesManager
    {
        public Hashtable Filter = new Hashtable();

        /// <summary>
        /// Конструктор класса StagesManager
        /// </summary>
        public StagesManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
        }

        /// <summary>
        /// Создаёт новую стадию
        /// </summary>
        /// <param name="stage">стадия</param>
        public void Insert(Stage stage)
        {
            string query = "CALL sp_addstage(:_name, :_signtype, :_lock, :_unlock, :_back)";

            var param = new Hashtable();
            param.Add(":_name", stage.Name);
            param.Add(":_signtype", stage.SignType);
            param.Add(":_lock", (stage.IsLock ? "1" : "0"));
            param.Add(":_unlock", (stage.IsUnlock ? "1" : "0"));
            param.Add(":_back", (stage.IsBack ? "1" : "0"));
            try
            {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.StagesManager.Insert", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

public Stage SelectLast()
{
    Stage stage = null;
    string query = "SELECT * FROM \"Stages\" ORDER BY \"ID\" DESC LIMIT 1;";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        stage = new Stage(Int32.Parse(t[0]), t[1], Int32.Parse(t[2]), Boolean.Parse(t[3]), Boolean.Parse(t[4]),
Boolean.Parse(t[5]));
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.StageManager.SelectLast", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return stage;
}

public bool VerifyName(string name)
{
    bool result = false;
    string query = "SELECT * FROM \"Stages\" WHERE \"Name\" = :_name; ";
    var param = new Hashtable();

    param.Add(":_name", name);

    try
    {
        ArrayList list = App.Proxy.ExecuteQuery(query, param);
        if (list.Count > 0)
            result = true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.StagesManager.VerifyName", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    return result;
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Получает список стадий
/// </summary>
/// <returns>Список стадий</returns>
public List<Stage> Select()
{
    List<Stage> list = new List<Stage>();

    string query =
        "SELECT * FROM fn_getstages(:_id, :_name)";
    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Stage stage = new Stage(Int32.Parse(t[0]), t[1], Int32.Parse(t[2]), Boolean.Parse(t[3]),
Boolean.Parse(t[4]), Boolean.Parse(t[5]));
            list.Add(stage);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.StagesManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    Filter.Clear();
    Filter.Add(":_id", null);
    Filter.Add(":_name", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="stage">Стадия</param>
public void Update(Stage stage)
{
    string query = "CALL sp_updatestage(:_id, :_name, :_signtype, :_lock, :_unlock, :_back);";

    var param = new Hashtable();
    param.Add(":_id", stage.ID);
    param.Add(":_name", stage.Name);
    param.Add(":_signtype", stage.SignType);
    param.Add(":_lock", stage.IsLock ? "1" : "0");
    param.Add(":_unlock", stage.IsUnlock ? "1" : "0");
    param.Add(":_back", stage.IsBack ? "1" : "0");

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Logger.Log log = new Logger.Log("Doc.App.StagesManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxImage.Error);
    }
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор стадии</param>
public bool Delete(int id)
{
    string query = "CALL sp_delstage(:_id); ";

    List<Document> list = new List<Document>();

    DocumentsManager documentsManager = new DocumentsManager();
    documentsManager.Filter["_stageid"] = id;
    list = documentsManager.Select(false, "");
    if (list.Count > 0)
    {
        MessageBox.Show("Есть документы на указанной стадии.\nУдаление невозможно",
            "Ошибка.", MessageBoxButtons.OK, MessageBoxImage.Error);
        return false;
    }

    //TODO: Проверка маршрута

    var param = new Hashtable();
    param.Add("_id", id);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
        return true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.StagesManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxImage.Error);
        return false;
    }
}
}
}
}

```

SubdivisonsManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

namespace Doc.App
{
    /// <summary>
    /// Класс для управления справочником подразделений
    /// </summary>
    public class SubdivisionManager
    {
        /// <summary>
        /// Фильтр
        /// </summary>
        public Hashtable Filter = new Hashtable();
        /// <summary>
        /// Конструктор класса SubdivisionManager
        /// </summary>
        public SubdivisionManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_parentid", null);
            Filter.Add(":_name", null);
        }
        /// <summary>
        /// Создаёт новое подразделение/организацию
        /// </summary>
        /// <param name="sub"></param>
        public void Insert(Subdivision sub)
        {
            string query = "CALL sp_addsubdivision(:_parentid, :_name)";

            var param = new Hashtable();
            param.Add(":_name", sub.Name);
            param.Add(":_parentid", sub.ParentID);
            try
            {
                App.Proxy.ExecuteNonQuery(query, param);
            }
            catch (Exception e)
            {
                Process proc = Process.GetCurrentProcess();

                Logger.Log log = new Logger.Log("Doc.App.SubdivisionManager.Insert", MessageType.Error, e,
DateTime.Now);
                string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
                Logger.WriteLog(logFile, log);
            }
        }
        /// <summary>
        /// Получает список подразделений/организаций
        /// </summary>
        /// <returns></returns>
        public List<Subdivision> Select()
        {
            List<Subdivision> list = new List<Subdivision>();

            string query =
                "SELECT * FROM fn_getsubdivisions(:_id, :_parentid, :_name)";
            try
            {
                ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

                Filter.Clear();

                Filter.Add(":_id", null);
                Filter.Add(":_parentid", null);
                Filter.Add(":_name", null);
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            Subdivision sub = new Subdivision(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2]);
            sub.FullName = GetFullNameSub(sub, sub.Name);
            list.Add(sub);
        }
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.SubdivisionManager.Select", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
    }

    Filter.Clear();

    Filter.Add(":_id", null);
    Filter.Add(":_parentid", null);
    Filter.Add(":_name", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="sub">Подразделение</param>
public void Update(Subdivision sub)
{
    string query = "CALL sp_updatesubdivision(:_id, :_parentid, :_name); ";

    var param = new Hashtable();

    param.Add(":_id", sub.ID);
    param.Add(":_parentid", sub.ParentID);
    param.Add(":_name", sub.Name);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.SubdivisionManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор подразделения</param>
public bool Delete(int id)
{
    List<Subdivision> list = new List<Subdivision>();
    Filter[":_parentid"] = id;
    list = Select();
    if(list.Count > 0)

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    {
        MessageBox.Show("Подразделение является родительским для "
            + list.Count.ToString() + " подразделений.\n " +
            "Рекомендуется вывести из него дочерние подразделения.", "Ошибка!",
            MessageBoxButton.OK, MessageBoxImage.Error);
        return false;
    }

    List<Employee> listEmpl = new List<Employee>();
    EmployeesManager employeesManager = new EmployeesManager();
    employeesManager.Filter[":_subdivid"] = id;
    listEmpl = employeesManager.Select(true);

    if (listEmpl.Count > 0)
    {
        MessageBox.Show("Есть сотрудники в указанном подразделении.\nУдаление невозможно",
            "Ошибка.", MessageBoxButton.OK, MessageBoxImage.Error);
        return false;
    }
    string query = "CALL sp_delsubdivision(:_id); ";

    var param = new Hashtable();
    param.Add(":_id", id);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
        return true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.SubdivisionManager.Delete", MessageType.Error, e,
            DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        return false;
    }
}

public bool VerifyName(string name, int parentid)
{
    bool result = false;
    string query = "SELECT * FROM \"Subdivisions\" WHERE \"Name\" = :_name AND \"ParentID\"
= :_parentid; ";
    var param = new Hashtable();

    param.Add(":_name", name);
    param.Add(":_parentid", parentid);
    try
    {
        ArrayList list = App.Proxy.ExecuteQuery(query, param);
        if (list.Count > 0)
            result = true;
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.SubdivisionManager.VerifyName", MessageType.Error,
            e, DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дат

```

    }
    return result;
}

/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public Subdivision SelectLast()
{
    Subdivision sub = null;
    string query = "SELECT * FROM \"Subdivisions\" ORDER BY \"ID\" DESC LIMIT 1;";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        sub = new Subdivision(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2]);
        sub.FullName = GetFullNameSub(sub, sub.Name);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return sub;
}

/// <summary>
/// Получает полное имя подразделения
/// </summary>
/// <param name="sub">Подразделение</param>
/// <param name="subname">Строковое имя подразделения</param>
/// <returns></returns>
private string GetFullNameSub(Subdivision sub, string subname)
{
    string name = "";
    string result = "";
    //собираем имя указанного подразделения с родителями
    if (sub.ParentID != 0)
    {
        Filter[":_id"] = sub.ParentID;
        List<Subdivision> list = Select();
        Subdivision subParent = list[0];
        name = subParent.Name + "\"\" + subname;
        name = GetFullNameSub(subParent, name);
    }
    else
    {
        name = subname;
    }
    result = name;
    return result;
}
}
}

```

TypeGroupManager.cs

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Utils;

namespace Doc.App
{
    class TypeGroupManager
    {
        public Hashtable Filter = new Hashtable();
        private string _group = "";

        public TypeGroupManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
            Filter.Add(":_code", null);
        }

        /// <summary>
        /// Получить список по фильтру
        /// </summary>
        /// <returns></returns>
        public List<TypeGroup> Select()
        {
            List<TypeGroup> list = new List<TypeGroup>();

            string query =
                "SELECT * FROM fn_gettypegroups(:_id, :_name, :_code)";
            try
            {
                ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
                foreach (string item in arr)
                {
                    string[] t = item.Split('|');
                    TypeGroup typeGroup = new TypeGroup(Int32.Parse(t[0]), t[1], Int32.Parse(t[2]));
                    list.Add(typeGroup);
                }
            }
            catch (Exception e)
            {
                Process proc = Process.GetCurrentProcess();

                Logger.Log log = new Logger.Log("Doc.App.TypeGroupManager.Select", MessageType.Error, e,
                    DateTime.Now);
                string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
                Logger.WriteLog(_logFile, log);
            }

            Filter.Clear();
            Filter.Add(":_id", null);
            Filter.Add(":_name", null);
            Filter.Add(":_code", null);

            return list;
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

TypeManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections;
using System.Diagnostics;
using System.IO;
using System.Windows;
using Utils;

namespace Doc.App
{
    class TypeManager
    {
        public Hashtable Filter = new Hashtable();

        public TypeManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_groupid", null);
            Filter.Add(":_name", null);
        }

        public void Insert(Type type)
        {
            string query = "CALL sp_addtype(:_groupid, :_name)";

            var param = new Hashtable();
            param.Add(":_name", type.Name);
            param.Add(":_groupid", type.Group);
            try
            {
                App.Proxy.ExecuteNonQuery(query, param);
            }
            catch (Exception e)
            {
                Process proc = Process.GetCurrentProcess();

                Logger.Log log = new Logger.Log("Doc.App.TypeMnager.Insert", MessageType.Error, e,
DateTime.Now);
                string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
                Logger.WriteLog(_logFile, log);
            }
        }

        /// <summary>
        /// Получить список по фильтру
        /// </summary>
        /// <returns></returns>
        public List<Type> Select()
        {
            List<Type> list = new List<Type>();

            string query =
                "SELECT * FROM fn_gettypes(:_id, :_groupid, :_name)";
            try
            {
                ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
                foreach (string item in arr)
                {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        string[] t = item.Split('|');
        Type type = new Type(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2] );
        list.Add(type);
    }
}
catch (Exception e)
{
    Process proc = Process.GetCurrentProcess();

    Logger.Log log = new Logger.Log("Doc.App.TypeManager.Select", MessageType.Error, e,
DateTime.Now);
    string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
    Logger.WriteLog(_logFile, log);
}

Filter.Clear();
Filter.Add(":_id", null);
Filter.Add(":_groupid", null);
Filter.Add(":_name", null);

return list;
}
/// <summary>
/// Последняя запись
/// </summary>
/// <returns></returns>
public Type SelectLast()
{
    Type type = null;
    string query = "SELECT * FROM \"Types\" ORDER BY \"ID\" DESC LIMIT 1;";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);

        string[] t = ((string)arr[0]).Split('|');
        type = new Type(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2]);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.FrameWork.GetEmployees", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    return type;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="type">Тип</param>
public void Update(Type type)
{
    string query = "CALL sp_updatetype(:_id, :_groupid, :_name); ";

    var param = new Hashtable();
    param.Add(":_id", type.ID);
    param.Add(":_groupid", type.Group);
    param.Add(":_name", type.Name);

    try

```

```

    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.TypeManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор типа</param>
public bool Delete(Type type)
{
    //Проверка на связанные записи
    TypeGroupManager typeGroupManager = new TypeGroupManager();
    typeGroupManager.Filter[":_code"] = type.Group;
    List<TypeGroup> listGroup = typeGroupManager.Select();
    TypeGroup group = listGroup[0];
    if(group.Code == (int)TypeGroupsEnum.AccessType)
    {
        AccessManager accessManager = new AccessManager();
        accessManager.Filter[":_accesstype"] = type.ID;
        List<Access> list = accessManager.Select();
        if (list.Count > 0)
        {
            MessageBox.Show("Существуют назначенные права доступа указанного типа.\n" +
                "Удаление невозможно.", "Ошибка.", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }
    else if (group.Code == (int)TypeGroupsEnum.DocType)
    {
        DocumentsManager docManager = new DocumentsManager();
        docManager.Filter[":_typeid"] = type.ID;
        List<Document> list = docManager.Select(true, "");
        if (list.Count > 0)
        {
            MessageBox.Show("Существуют документы указанного типа.\n" +
                "Удаление невозможно.", "Ошибка.", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }
    else if (group.Code == (int)TypeGroupsEnum.PassType)
    {
        EmployeesManager emplManager = new EmployeesManager();
        emplManager.Filter[":_typedoc"] = type.ID;
        List<Employee> list = emplManager.Select(true);
        if (list.Count > 0)
        {
            MessageBox.Show("Существуют сотрудники с документами указанного типа.\n" +
                "Удаление невозможно.", "Ошибка.", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }
    else if (group.Code == (int)TypeGroupsEnum.SignatureType)
    {
        SignaturesManager signManager = new SignaturesManager();
        signManager.Filter[":_typeid"] = type.ID;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        List<Signature> list = signManager.Select();
        if (list.Count > 0)
        {
            MessageBox.Show("Существуют подписи указанного типа.\n" +
                "Удаление невозможно.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
        string query = "CALL sp_deltypes(:_id);";

        var param = new Hashtable();
        param.Add(":_id", type.ID);
        try
        {
            App.Proxy.ExecuteNonQuery(query, param);
            return true;
        }
        catch (Exception e)
        {
            Process proc = Process.GetCurrentProcess();

            Logger.Log log = new Logger.Log("Doc.App.TypeManager.Delete", MessageType.Error, e,
                DateTime.Now);
            string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
            Logger.WriteLog(logFile, log);
            MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false;
        }
    }
}
}

```

UserManager.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.ServiceModel;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Utils;
using Utils.Config;

namespace Doc.App
{
    public class UserManager
    {
        public Hashtable Filter = new Hashtable();

        public UserManager()
        {
            Filter.Add(":_id", null);
            Filter.Add(":_emplid", null);
            Filter.Add(":_login", null);
            Filter.Add(":_roleid", null);
        }

        public void Insert(User user)

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

{
    string query = "CALL sp_adduser(:_emplid, :_login, :_password, :_roleid)";
    var param = new Hashtable();
    param.Add(":_emplid", user.EmplID);
    param.Add(":_login", user.Login);
    param.Add(":_password", user.Password);
    param.Add(":_roleid", user.RoleID);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.UserManager.Insert", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }
}

public List<User> Select()
{
    List<User> list = new List<User>();

    string query = "SELECT * FROM fn_getusers(:_id, :_emplid, :_login, :_roleid)";

    try
    {
        ArrayList arr = App.Proxy.ExecuteQuery(query, Filter);
        foreach (string item in arr)
        {
            string[] t = item.Split('|');
            var user = new User(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2], t[3], Int32.Parse(t[4]) );
            list.Add(user);
        }
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.UserManager.Select", MessageType.Error, e,
DateTime.Now);
        string _logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(_logFile, log);
    }

    Filter.Clear();

    Filter.Add(":_id", null);
    Filter.Add(":_emplid", null);
    Filter.Add(":_login", null);
    Filter.Add(":_roleid", null);

    return list;
}

/// <summary>
/// Изменяет текущую запись
/// </summary>
/// <param name="user">Пользователь</param>
public void Update(User user)

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

{
    string query = "CALL sp_updateuser(:_id, :_login, :_password, :_roleid); ";

    var param = new Hashtable();
    param.Add(":_id", user.Login);
    param.Add(":_login", user.Login);
    param.Add(":_password", user.Password);
    param.Add(":_roleid", user.RoleID);

    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.UserManager.Update", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Удаляет текущую запись
/// </summary>
/// <param name="id">Идентификатор Пользователя</param>
public void Delete(User user)
{
    string query = "CALL sp_deluser(:_id); ";

    var param = new Hashtable();

    param.Add(":_id", user.ID);
    try
    {
        App.Proxy.ExecuteNonQuery(query, param);
        //удалить доступ
        AccessManager accessManager = new AccessManager();
        accessManager.Filter[":_emplid"] = user.EmplID;
        List<Access> list = accessManager.Select();
        accessManager.Delete(list);
    }
    catch (Exception e)
    {
        Process proc = Process.GetCurrentProcess();

        Logger.Log log = new Logger.Log("Doc.App.UserManager.Delete", MessageType.Error, e,
DateTime.Now);
        string logFile = Logger.GetLogName(proc.ProcessName, proc.StartTime);
        Logger.WriteLog(logFile, log);
        MessageBox.Show(e.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Возвращает последнюю запись
/// </summary>
/// <returns></returns>
public User SelectLast()
{
    User user = null;
    ArrayList list = new ArrayList();
    string query = "SELECT * FROM VUsers ORDER BY VID DESC LIMIT 1; ";

```

```

list = App.Proxy.ExecuteQuery(query, Filter);
if(list.Count > 0)
{
    string[] t = list[0].ToString().Split('|');
    user = new User(Int32.Parse(t[0]), Int32.Parse(t[1]), t[2], t[3], Int32.Parse(t[4]));
}
return user;
}
}
}

```

Access.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Права доступа
    /// </summary>
    public class Access
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public readonly int ID;
        /// <summary>
        /// Идентификатор документа
        /// </summary>
        public readonly long ObjectID;
        /// <summary>
        /// Идентификатор сотрудника
        /// </summary>
        public readonly int EmplID;
        /// <summary>
        /// Идентификатор подразделения
        /// </summary>
        public readonly int SubID;
        /// <summary>
        /// Тип доступа
        /// </summary>
        public readonly int AccessType;
        /// <summary>
        /// Сотрудник
        /// </summary>
        public readonly Employee Employee = null;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// Подразделение
/// </summary>
public readonly Subdivision Subdivision = null;

/// <summary>
/// Конструктор класса Access
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="objid">Идентификатор документа</param>
/// <param name="emplid">Идентификатор пользователя</param>
/// <param name="subid">Идентификатор подразделения</param>
/// <param name="access">Идентификатор доступа</param>
public Access(int id, long objid, int emplid, int subid, int access)
{
    ID = id;
    ObjectID = objid;
    EmplID = emplid;
    SubID = subid;
    AccessType = access;

    if (emplid > 0)
    {
        EmployeesManager employeesManager = new EmployeesManager();
        employeesManager.Filter[":_id"] = emplid;
        List<Employee> list = employeesManager.Select(false);
        Employee = list[0];
    }
    if (subid > 0)
    {
        SubdivisionManager subdivisionManager = new SubdivisionManager();
        subdivisionManager.Filter[":_id"] = subid;
        List<Subdivision> list = subdivisionManager.Select();
        Subdivision = list[0];
    }
}
}
}

```

Address.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Адрес
    /// </summary>
    public class Address
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Почтовый индекс
        /// </summary>
        public string PostIndex { get; }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Страна
/// </summary>
public string Country { get; }

/// <summary>
/// Адрес
/// </summary>
public string Addr { get; }

/// <summary>
/// Конструктор класса Address
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="postindex">Почтовый индекс</param>
/// <param name="country">Страна</param>
/// <param name="addr">Адрес</param>
public Address(int id, string postindex, string country, string addr)
{
    ID = id;
    PostIndex = postindex;
    Country = country;
    Addr = addr;
}
}
}

```

Appointment.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Должность
    /// </summary>
    public class Appointment
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Наименование должности
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Конструктор класса Appointment
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="name">Наименование должности</param>
        public Appointment(int id, string name)
        {
            ID = id;
            Name = name;
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }
}

```

DocFile.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Файл документа
    /// </summary>
    class DocFile
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public long ID { get; }

        /// <summary>
        /// Имя файла
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Путь к файлу на файловом сервере
        /// </summary>
        public string FullPath { get; }

        /// <summary>
        /// Текущий исторический идентификатор
        /// </summary>
        public long CurHID { get; }

        /// <summary>
        /// Исторический идентификатор записи, которая заменила данную
        /// </summary>
        public long RemHID { get; }

        /// <summary>
        /// Идентификатор связанного документа
        /// </summary>
        public long DocID { get; }

        /// <summary>
        /// Версия файла
        /// </summary>
        public int Version { get; }

        /// <summary>
        /// Конструктор класса Files
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="name">Имя файла</param>
        /// <param name="fullpath">Путь к файлу на файловом сервере</param>
        /// <param name="currhid">Текущий исторический идентификатор</param>
        /// <param name="remhid">Исторический идентификатор записи, которая заменила данную</param>
        /// <param name="docid">Идентификатор связанного документа</param>
        /// <param name="version">Версия файла</param>
        ///
        public DocFile(long id, string name, string fullpath, long currhid
            , long remhid, long docid, int version)
        {
            ID = id;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        Name = name;
        FullPath = fullpath;
        CurHID = currhid;
        RemHID = remhid;
        DocID = docid;
        Version = version;
    }
}
}

```

Document.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Документ
    /// </summary>
    public class Document
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public long ID { get; }
        /// <summary>
        /// Идентификатор родителя
        /// </summary>
        public long ParentID { get; }
        /// <summary>
        /// Наименование документа
        /// </summary>
        public string Name { get; }
        /// <summary>
        /// Текущий исторический идентификатор
        /// </summary>
        public long CurHID { get; }
        /// <summary>
        /// Исторический идентификатор записи, которая заменила данную
        /// </summary>
        public long RemHID { get; }
        /// <summary>
        /// Идентификатор автора документа
        /// </summary>
        public int AuthorID { get; }
        /// <summary>
        /// Автор документа
        /// </summary>
        public Employee Author { get; }
        /// <summary>
        /// Идентификатор заблокировавшего документ
        /// </summary>
        public int LockID { get; }
        /// <summary>
        /// Заблокировавший сотрудник
        /// </summary>
        public Employee LockEmpl { get; }
        /// <summary>
        /// Заблокировавший хост
        /// </summary>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

public string LockHost { get; }

/// <summary>
/// Идентификатор автора изменений
/// </summary>
public int EditAuthorID { get; }
/// <summary>
/// Автор изменений
/// </summary>
public Employee EditAuthor { get; }
/// <summary>
/// Идентификатор стадии разработки
/// </summary>
public int StageID { get; }
/// <summary>
/// Стадия разработки
/// </summary>
public Stage Stage { get; }
/// <summary>
/// Идентификатор типа документа
/// </summary>
public intTypeID { get; }
/// <summary>
/// Тип документа
/// </summary>
public Type DocType { get; }
/// <summary>
/// Дата создания
/// </summary>
public DateTime CreateDate { get; }
/// <summary>
/// Дата изменения
/// </summary>
public DateTime EditDate { get; }
/// <summary>
/// Обозначение
/// </summary>
public string Designation { get; }
/// <summary>
/// Описание
/// </summary>
public string Description { get; }
/// <summary>
/// Идентификатор маршрута
/// </summary>
public int RouteID { get; }
/// <summary>
/// Порядковый номер стадии в маршруте
/// </summary>
public int StageNum { get; set; }

/// <summary>
/// Конструктор класса Document
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="parentid">Идентификатор родителя</param>
/// <param name="name">Наименование документа</param>
/// <param name="currhid">Текущий исторический идентификатор</param>
/// <param name="remhid">Исторический идентификатор записи, которая заменила данную</param>
/// <param name="authorid">Идентификатор автора документа</param>
/// <param name="lockid">Идентификатор заблокировавшего документ</param>
/// <param name="lockhost">Заблокировавший хост</param>
/// <param name="routeid">Идентификатор маршрута</param>
/// <param name="editauthorid">Автор изменений</param>
/// <param name="stageid">Идентификатор стадии разработки</param>

```

Изм.	Лист	№ докум.	Подпись	Дат

```

/// <param name="typeid">Идентификатор типа документа</param>
/// <param name="createdate">Дата создания</param>
/// <param name="editdate">Дата изменения</param>
/// <param name="designation">Обозначение</param>
/// <param name="description">Описание</param>
/// <param name="stagenum">Порядковый номер стадии в маршруте</param>

public Document(long id, long parentid, string name, long currhid, long remhid, int authorid
, int lockid, string lockhost, int routeid, int stageid, int typeid, DateTime createdate
, DateTime editdate, string designation, string description, int editauthorid, int stagenum)
{
    ID = id;
    ParentID = parentid;
    Name = name;
    CurHID = currhid;
    RemHID = remhid;
    AuthorID = authorid;
    LockID = lockid;
    EditAuthorID = editauthorid;
    StageID = stageid;
   TypeID = typeid;
    CreateDate = createdate;
    EditDate = editdate;
    Designation = designation;
    Description = description;
    RouteID = routeid;
    LockHost = lockhost;
    StageNum = stagenum;

    EmployeesManager employeesManager = new EmployeesManager();
    employeesManager.Filter[":_id"] = lockid;
    List<Employee> listEmpl = employeesManager.Select(true);
    if (listEmpl.Count > 0)
        LockEmpl = listEmpl[0];

    employeesManager.Filter[":_id"] = authorid;
    listEmpl = employeesManager.Select(true);
    if (listEmpl.Count > 0)
        Author = listEmpl[0];

    if (authorid == editauthorid)
        EditAuthor = Author;
    else
    {
        employeesManager.Filter[":_id"] = editauthorid;
        listEmpl = employeesManager.Select(true);
        if (listEmpl.Count > 0)
            EditAuthor = listEmpl[0];
    }

    StagesManager stagesManager = new StagesManager();
    stagesManager.Filter[":_id"] = stageid;
    List<Stage> listStage = stagesManager.Select();
    if (listStage.Count > 0)
        Stage = listStage[0];

    TypeManager typeManager = new TypeManager();
    typeManager.Filter[":_id"] = typeid;
    List<Type> listType = typeManager.Select();
    if (listType.Count > 0)
        DocType = listType[0];
}
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

Employees.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mime;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Сотрудник
    /// </summary>
    public class Employee
    {
        /// <summary>
        /// Идентификатор сотрудника
        /// </summary>
        public readonly int ID;

        /// <summary>
        /// Фамилия сотрудника
        /// </summary>
        public string LastName { get; }

        /// <summary>
        /// Имя сотрудника
        /// </summary>
        public string FirstName { get; }

        /// <summary>
        /// Фамилия, Имя, Отчество
        /// </summary>
        public string FIO { get; }

        /// <summary>
        /// Отчество сотрудника
        /// </summary>
        public string MiddleName { get; }

        /// <summary>
        /// Тип документа
        /// </summary>
        public int TypeDoc { get; }

        /// <summary>
        /// Документ
        /// </summary>
        public string Doc { get; }

        /// <summary>
        /// Идентификатор адреса
        /// </summary>
        public int AddrID { get; }

        /// <summary>
        /// Идентификатор подразделения
        /// </summary>
        public int SubdivID { get; }

        /// <summary>
        /// Идентификатор должности
        /// </summary>
        public int AppointID { get; }

        /// <summary>
        /// Идентификатор фотографии
        /// </summary>
        public int PhotoID { get; }
    }
}
```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Дата рождения
/// </summary>
public DateTime Birthday { get; }

/// <summary>
/// Рабочий телефон
/// </summary>
public string WorkPhone { get; }

/// <summary>
/// Домашний телефон
/// </summary>
public string HomePhone { get; }

/// <summary>
/// Мобильный телефон
/// </summary>
public string MobilePhone { get; }
/// <summary>
/// Признак уволенного сотрудника
/// </summary>
public bool Deleted { get; set; }

/// <summary>
/// Военское звание
/// </summary>
public int RankID { get; }

///Поля по требованию

/// <summary>
/// Фото сотрудника
/// </summary>
public Photo Photo { get; }

/// <summary>
/// Адрес сотрудника
/// </summary>
public Address Address { get; }

/// <summary>
/// Подразделение сотрудника
/// </summary>
public Subdivision Subdivision { get; }

/// <summary>
/// Военское звание
/// </summary>
public MilitaryRank MilitaryRank { get; }

/// <summary>
/// Должность
/// </summary>
public Appointment Appointment { get; }

/// Текстовое представление признака уволенного
/// </summary>
public string StringDeleted { get; }

/// <summary>
/// Конструктор класса Employee
/// </summary>
/// <param name="id">Идентификатор сотрудника</param>

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <param name="lastname">Фамилия сотрудника</param>
/// <param name="firstname">Имя сотрудника</param>
/// <param name="middlename">Отчество сотрудника</param>
/// <param name="typedoc">Тип документа</param>
/// <param name="doc">Документ</param>
/// <param name="addrid">Идентификатор адреса</param>
/// <param name="subdivid">Идентификатор подразделения</param>
/// <param name="appointid">Идентификатор должности</param>
/// <param name="photoid">Идентификатор фото</param>
/// <param name="rankid">Идентификатор воинского звания</param>
/// <param name="birthday">Дата рождения</param>
/// <param name="workphone">Рабочий телефон</param>
/// <param name="homephone">Домашний телефон</param>
/// <param name="mobilephone">Мобильный телефон</param>
/// <param name="deleted">Признак уволенного сотрудника</param>
public Employee(int id, string lastname, string firstname, string middlename, int typedoc, string doc, int
addrid
, int subdivid, int appointid, int photoid, int rankid, DateTime birthday, string workphone, string
homephone, string mobilephone, bool deleted)
{
    ID = id;
    LastName = lastname;
    FirstName = firstname;
    MiddleName = middlename;
    TypeDoc = typedoc;
    Doc = doc;
    AddrID = addrid;
    SubdivID = subdivid;
    AppointID = appointid;
    PhotoID = photoid;
    RankID = rankid;
    Birthday = birthday;
    WorkPhone = workphone;
    HomePhone = homephone;
    MobilePhone = mobilephone;
    Deleted = deleted;
    FIO = lastname + " " + " " + firstname + " " + middlename;

    SubdivisionManager subdivisionManager = new SubdivisionManager();
    subdivisionManager.Filter[":_id"] = subdivid;
    List<Subdivision> sub = subdivisionManager.Select();
    if (sub.Count > 0)
    {
        Subdivision = sub[0];
    }

    else Subdivision = null;

    PhotoManager photoManager = new PhotoManager();
    photoManager.Filter[":_id"] = photoid;
    Photo = photoManager.Select(photoid);

    AddressManager addressManager = new AddressManager();
    addressManager.Filter[":_id"] = addrid;
    Address = addressManager.Select();

    MilitaryRanksManager militaryRanksManager = new MilitaryRanksManager();
    militaryRanksManager.Filter[":_id"] = rankid;
    List<MilitaryRank> ranks = militaryRanksManager.Select();
    if (ranks.Count > 0)
        MilitaryRank = ranks[0];
    else MilitaryRank = null;

    AppointmentManager appointmentManager = new AppointmentManager();
    appointmentManager.Filter[":_id"] = appointid;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        List<Appointment> appoint = appointmentManager.Select();
        if (appoint.Count > 0)
            Appointment = appoint[0];
        else Appointment = null;

        StringDeleted = deleted ? "Да" : "Нет";
    }
}
}

```

HisFiles.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mime;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Сотрудник
    /// </summary>
    public class Employee
    {
        /// <summary>
        /// Идентификатор сотрудника
        /// </summary>
        public readonly int ID;

        /// <summary>
        /// Фамилия сотрудника
        /// </summary>
        public string LastName { get; }

        /// <summary>
        /// Имя сотрудника
        /// </summary>
        public string FirstName { get; }

        /// <summary>
        /// Фамилия, Имя, Отчество
        /// </summary>
        public string FIO { get; }

        /// <summary>
        /// Отчество сотрудника
        /// </summary>
        public string MiddleName { get; }

        /// <summary>
        /// Тип документа
        /// </summary>
        public int TypeDoc { get; }

        /// <summary>
        /// Документ
        /// </summary>
        public string Doc { get; }

        /// <summary>
        /// Идентификатор адреса
        /// </summary>
        public int AddrID { get; }

        /// <summary>

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// Идентификатор подразделения
/// </summary>
public int SubdivID { get; }
/// <summary>
/// Идентификатор должности
/// </summary>
public int AppointID { get; }
/// <summary>
/// Идентификатор фотографии
/// </summary>
public int PhotoID { get; }
/// <summary>
/// Дата рождения
/// </summary>
public DateTime Birthday { get; }

/// <summary>
/// Рабочий телефон
/// </summary>
public string WorkPhone { get; }

/// <summary>
/// Домашний телефон
/// </summary>
public string HomePhone { get; }

/// <summary>
/// Мобильный телефон
/// </summary>
public string MobilePhone { get; }
/// <summary>
/// Признак уволенного сотрудника
/// </summary>
public bool Deleted { get; set; }

/// <summary>
/// Военское звание
/// </summary>
public int RankID { get; }

///Поля по требованию

/// <summary>
/// Фото сотрудника
/// </summary>
public Photo Photo { get; }

/// <summary>
/// Адрес сотрудника
/// </summary>
public Address Address { get; }

/// <summary>
/// Подразделение сотрудника
/// </summary>
public Subdivision Subdivision { get; }

/// <summary>
/// Военское звание
/// </summary>
public MilitaryRank MilitaryRank { get; }

/// <summary>
/// Должность

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

/// </summary>
public Appointment Appointment { get; }

/// Текстовое представление признака уволенного
/// </summary>
public string StringDeleted { get; }

/// <summary>
/// Конструктор класса Employee
/// </summary>
/// <param name="id">Идентификатор сотрудника</param>
/// <param name="lastname">Фамилия сотрудника</param>
/// <param name="firstname">Имя сотрудника</param>
/// <param name="middlename">Отчество сотрудника</param>
/// <param name="typedoc">Тип документа</param>
/// <param name="doc">Документ</param>
/// <param name="addrid">Идентификатор адреса</param>
/// <param name="subdivid">Идентификатор подразделения</param>
/// <param name="appointid">Идентификатор должности</param>
/// <param name="photoid">Идентификатор фото</param>
/// <param name="rankid">Идентификатор воинского звания</param>
/// <param name="birthday">Дата рождения</param>
/// <param name="workphone">Рабочий телефон</param>
/// <param name="homephone">Домашний телефон</param>
/// <param name="mobilephone">Мобильный телефон</param>
/// <param name="deleted">Признак уволенного сотрудника</param>
public Employee(int id, string lastname, string firstname, string middlename, int typedoc, string doc, int
addrid
, int subdivid, int appointid, int photoid, int rankid, DateTime birthday, string workphone, string
homephone, string mobilephone, bool deleted)
{
    ID = id;
    LastName = lastname;
    FirstName = firstname;
    MiddleName = middlename;
    TypeDoc = typedoc;
    Doc = doc;
    AddrID = addrid;
    SubdivID = subdivid;
    AppointID = appointid;
    PhotoID = photoid;
    RankID = rankid;
    Birthday = birthday;
    WorkPhone = workphone;
    HomePhone = homephone;
    MobilePhone = mobilephone;
    Deleted = deleted;
    FIO = lastname + " " + " " + firstname + " " + middlename;

    SubdivisionManager subdivisionManager = new SubdivisionManager();
    subdivisionManager.Filter[":_id"] = subdivid;
    List<Subdivision> sub = subdivisionManager.Select();
    if (sub.Count > 0)
    {
        Subdivision = sub[0];
    }

    else Subdivision = null;

    PhotoManager photoManager = new PhotoManager();
    photoManager.Filter[":_id"] = photoid;
    Photo = photoManager.Select(photoid);

    AddressManager addressManager = new AddressManager();
    addressManager.Filter[":_id"] = addrid;

```

Изм.	Лист	№ докум.	Подпись	Дат

```

        Address = addressManager.Select();

        MilitaryRanksManager militaryRanksManager = new MilitaryRanksManager();
        militaryRanksManager.Filter[":_id"] = rankid;
        List<MilitaryRank> ranks = militaryRanksManager.Select();
        if (ranks.Count > 0)
            MilitaryRank = ranks[0];
        else MilitaryRank = null;

        AppointmentManager appointmentManager = new AppointmentManager();
        appointmentManager.Filter[":_id"] = appointid;
        List<Appointment> appoint = appointmentManager.Select();
        if (appoint.Count > 0)
            Appointment = appoint[0];
        else Appointment = null;

        StringDeleted = deleted ? "Да" : "Нет";
    }
}
}

```

History.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// История изменений
    /// </summary>
    class History
    {
        /// <summary>
        /// Исторический идентификатор
        /// </summary>
        public long HID { get; }
        /// <summary>
        /// Тип изменённого объекта
        /// </summary>
        public int ObjectType { get; }
        /// <summary>
        /// Идентификатор изменённого объекта
        /// </summary>
        public long ObjectID { get; }
        /// <summary>
        /// Дата изменения
        /// </summary>
        public DateTime DateChange { get; }
        /// <summary>
        /// Идентификатор автора изменений
        /// </summary>
        public int ChangeAuthor { get; }
        /// <summary>
        /// Имя хоста, с которого произвели изменения
        /// </summary>
        public string ChangeHost { get; }
        /// <summary>
        /// Описание изменений
        /// </summary>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

public string Action { get; }

/// <summary>
/// Конструктор класса History
/// </summary>
/// <param name="hid">Исторический идентификатор</param>
/// <param name="objtype">Тип изменённого объекта</param>
/// <param name="objid">Идентификатор изменённого объекта</param>
/// <param name="datechange">Дата изменения</param>
/// <param name="changeauthor">Идентификатор автора изменений</param>
/// <param name="changehost">Имя хоста, с которого произвели изменения</param>
/// <param name="action">Описание изменений</param>
public History(long hid, int objtype, long objid, DateTime datechange, int changeauthor, string changehost,
string action)
{
    HID = hid;
    ObjectType = objtype;
    ObjectID = objid;
    DateChange = datechange;
    ChangeAuthor = changeauthor;
    ChangeHost = changehost;
    Action = action;
}
}
}

```

MilitaryRank.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    public class MilitaryRank
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }
        /// <summary>
        /// Военское звание
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Конструктор класса Appointment
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="name">Наименование должности</param>
        public MilitaryRank(int id, string name)
        {
            ID = id;
            Name = name;
        }
    }
}

```

Photo.cs

```

using System;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Фото сотрудника
    /// </summary>
    public class Photo
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }
        /// <summary>
        /// Фото сотрудника
        /// </summary>
        public byte[] PhotoEmpl { get; }

        /// <summary>
        /// Конструктор класса Photo
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="photo">Фото сотрудника</param>
        public Photo(int id, byte[] photo)
        {
            ID = id;
            PhotoEmpl = photo;
        }
    }
}
```

PostMessageIn.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Входящее почтовое сообщение
    /// </summary>
    public class PostMessageIn
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public long ID { get; }
        /// <summary>
        /// Тема письма
        /// </summary>
        public string Theme { get; }
        /// <summary>
        /// Кому
        /// </summary>
        public int ToUserID { get; }
        /// <summary>
        /// От кого
        /// </summary>
    }
}
```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

public int FromUserID { get; }
/// <summary>
/// Сообщение
/// </summary>
public string Message { get; }
/// <summary>
/// Связанный документ
/// </summary>
public long ObjectID { get; }
/// <summary>
/// Прочтено
/// </summary>
public bool Readed { get; set; }
/// <summary>
/// Удалено
/// </summary>
public bool Deleted { get; set; }
/// <summary>
/// Идентификатор исходящего сообщения
/// </summary>
public long OutID { get; set; }
/// <summary>
/// Дата сообщения
/// </summary>
public DateTime DatePost { get; }

/// <summary>
/// Отправитель
/// </summary>
public Employee FromUser { get; }
/// <summary>
/// Получатель
/// </summary>
public Employee ToUser { get; }
/// <summary>
/// Прикрепленный документ
/// </summary>
public Document Document { get; }

/// <summary>
/// Конструктор класса PostMessageIn
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="theme">Тема письма</param>
/// <param name="touser">Кому</param>
/// <param name="fromuser">От кого</param>
/// <param name="message">Сообщение</param>
/// <param name="objid">Связанный документ</param>
/// <param name="readed">Прочтено</param>
/// <param name="deleted">Deleted</param>
/// <param name="outid">Идентификатор исходящего сообщения</param>
/// <param name="date">Дата сообщения</param>
public PostMessageIn(long id, string theme, int touser, int fromuser, string message, long objid, bool
readed,
bool deleted, long outid, DateTime date)
{
    ID = id;
    Theme = theme;
    ToUserID = touser;
    FromUserID = fromuser;
    Message = message;
    ObjectID = objid;
    Readed = readed;
    Deleted = deleted;
    OutID = outid;

```

```

DatePost = date;

EmployeesManager employeesManager = new EmployeesManager();
employeesManager.Filter[":_id"] = touser;
List<Employee> lst = employeesManager.Select(false);
ToUser = lst[0];
lst.Clear();

employeesManager.Filter[":_id"] = fromuser;
lst = employeesManager.Select(false);
FromUser = lst[0];

if (objid > 0)
{
    List<Document> lstDoc = new List<Document>();
    DocumentsManager documentsManager = new DocumentsManager();
    documentsManager.Filter[":_id"] = objid;
    lstDoc = documentsManager.Select(false, "");
    if (lstDoc.Count > 0)
        Document = lstDoc[0];
}
else
{
    Document = null;
}
}
}
}

```

PostMessageOut.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Исходящее почтовое сообщение
    /// </summary>
    public class PostMessageOut
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public long ID { get; }
        /// <summary>
        /// Тема сообщения
        /// </summary>
        public string Theme { get; }
        /// <summary>
        /// Кому
        /// </summary>
        public int ToUserID { get; }
        /// <summary>
        /// От кого
        /// </summary>
        public int FromUserID { get; }
        /// <summary>
        /// Сообщение
        /// </summary>
        public string Message { get; }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Прочтено получателем
/// </summary>
public bool Readed { get; set; }
/// <summary>
/// Удалено
/// </summary>
public bool Deleted { get; set; }
/// <summary>
/// Отправлено
/// </summary>
public bool Sent { get; set; }
/// <summary>
/// Идентификатор связанного документа
/// </summary>
public long ObjectID { get; }

/// <summary>
/// Дата сообщения
/// </summary>
public DateTime DatePost { get; }
/// <summary>
/// Признак автоматического сообщения
/// </summary>
public bool IsAuto { get; }

/// <summary>
/// Отправитель
/// </summary>
public Employee FromUser { get; }
/// <summary>
/// Получатель
/// </summary>
public Employee ToUser { get; }
/// <summary>
/// Прикрепленный документ
/// </summary>
public Document Document { get; }

/// <summary>
/// Конструктор класса PostMessageOut
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="theme">Тема сообщения</param>
/// <param name="touser">Кому</param>
/// <param name="fromuser">От кого</param>
/// <param name="message">Сообщение</param>
/// <param name="readed">Прочтено получателем</param>
/// <param name="deleted">Удалено</param>
/// <param name="sent">Отправлено</param>
/// <param name="objid">Идентификатор связанного документа</param>
/// <param name="date">Дата сообщения</param>
/// <param name="isauto">Признак автоматического сообщения</param>
public PostMessageOut(long id, string theme, int touser, int fromuser, string message, bool readed,
    bool deleted, bool sent, long objid, DateTime date, bool isauto)
{
    ID = id;
    Theme = theme;
    ToUserID = touser;
    FromUserID = fromuser;
    Message = message;
    Readed = readed;
    Deleted = deleted;
    Sent = sent;
    ObjectID = objid;
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

DatePost = date;
IsAuto = isauto;

EmployeesManager employeesManager = new EmployeesManager();
List<Employee> Ist = new List<Employee>();

if (touser > 0)
{
    employeesManager.Filter[":_id"] = touser;
    Ist = employeesManager.Select(false);
    ToUser = Ist[0];
    Ist.Clear();
}

employeesManager.Filter[":_id"] = fromuser;
Ist = employeesManager.Select(false);
FromUser = Ist[0];

if (objid > 0)
{
    List<Document> IstDoc = new List<Document>();
    DocumentsManager documentsManager = new DocumentsManager();
    documentsManager.Filter[":_id"] = objid;
    IstDoc = documentsManager.Select(false, "");
    if (IstDoc.Count > 0)
        Document = IstDoc[0];
}
else
{
    Document = null;
}
}
}
}

```

Role.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    public class Role
    {
        /// <summary>
        /// Идентификатор роли
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Имя роли
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Доступ к контролам приложения
        /// </summary>
        public string ControlAccess { get; }

        /// <summary>
        /// Конструктор класса Types

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    /// </summary>
    /// <param name="id">Идентификатор роли</param>
    /// <param name="name">Имя роли</param>
    /// <param name="controlaccess">Доступ к контролам приложения</param>
    public Role(int id, string name, string controlaccess )
    {
        ID = id;
        Name = name;
        ControlAccess = controlaccess;
    }
}

```

Route.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Маршрут
    /// </summary>
    public class Route
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }
        /// <summary>
        /// Имя маршрута
        /// </summary>
        public string Name { get; }
        /// <summary>
        /// Маршрут
        /// </summary>
        public string RouteAlg { get; }

        /// <summary>
        /// Конструктор класса Route
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="name">Имя маршрута</param>
        /// <param name="route">Маршрут</param>
        public Route(int id, string name, string route)
        {
            ID = id;
            Name = name;
            RouteAlg = route;
        }
    }
}

```

Signature.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

using Utils;

namespace Doc.App
{
    /// <summary>
    /// Подпись
    /// </summary>
    public class Signature
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public long ID { get; }
        /// <summary>
        /// Идентификатор типа подписи
        /// </summary>
        public int TypeID { get; }
        /// <summary>
        /// Идентификатор сотрудника
        /// </summary>
        public int EmplID { get; }
        /// <summary>
        /// Идентификатор документа
        /// </summary>
        public long ObjectID { get; }
        /// <summary>
        /// Дата подписи
        /// </summary>
        public DateTime SignDate { get; }
        /// <summary>
        /// Сотрудник, пописавший документ
        /// </summary>
        public Employee Employee { get; }
        /// <summary>
        /// Тип подписи
        /// </summary>
        public Type SignType { get; }
        /// <summary>
        /// Идентификатор маршрута
        /// </summary>
        public int RouteID { get; }
        /// <summary>
        /// Порядковый номер стадии в маршруте
        /// </summary>
        public int StageNum { get; set; }

        /// <summary>
        /// Конструктор класса Signature
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="typeid">Идентификатор типа подписи</param>
        /// <param name="emplid">Идентификатор сотрудника</param>
        /// <param name="objid">Идентификатор документа</param>
        /// <param name="signdate">Дата подписи</param>
        /// <param name="routeid">Идентификатор маршрута</param>
        public Signature(long id, int typeid, int emplid, long objid, DateTime signdate, int routeid, int stagenum)
        {
            ID = id;
            TypeID = typeid;
            EmplID = emplid;
            ObjectID = objid;
            SignDate = signdate;
            RouteID = routeid;
            StageNum = stagenum;
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        EmployeesManager employeesManager = new EmployeesManager();
        employeesManager.Filter[":_id"] = emplid;
        List<Employee> lst = employeesManager.Select(true);
        Employee = lst[0];

        TypeManager _typeManager = new TypeManager();
        _typeManager.Filter[":_id"] = typeid;

        List<Type> lstType = _typeManager.Select();
        SignType = lstType[0];
    }
}
}

```

Stage.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Стадия проектирования
    /// </summary>
    public class Stage
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Наименование стадии
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Тип подписи
        /// </summary>
        public int SignType { get; }

        /// <summary>
        /// Признак блокирующей стадии
        /// </summary>
        public bool IsLock { get; }

        /// <summary>
        /// Возможность разблокировать
        /// </summary>
        public bool IsUnlock { get; }

        /// <summary>
        /// Возможность отправки на доработку
        /// </summary>
        public bool IsBack { get; }

        /// <summary>
        /// Порядковый номер стадии в маршруте
        /// </summary>
        public int Number { get; set; }

        /// <summary>
        /// Конструктор класса Stage
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="name">Наименование стадии</param>
        /// <param name="islock">Признак блокирующей стадии</param>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <param name="isunlock">Возможность разблокировать</param>
/// <param name="isback">Возможность отправки на доработку</param>
public Stage(int id, string name, int signtype, bool islock, bool isunlock, bool isback)
{
    ID = id;
    Name = name;
    SignType = signtype;
    IsLock = islock;
    IsUnlock = isunlock;
    IsBack = isback;
}
}
}

```

Subdivisions.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Подразделение
    /// </summary>
    public class Subdivision
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }
        /// <summary>
        /// Идентификатор родительской записи
        /// </summary>
        public int ParentID { get; }
        /// <summary>
        /// Наименование подразделения
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Полное наименование подразделения
        /// </summary>
        public string FullName { get; set; }

        /// <summary>
        /// Конструктор класса Subdivision
        /// </summary>
        /// <param name="id">Идентификатор записи</param>
        /// <param name="parentid">Идентификатор родительской записи</param>
        /// <param name="name">Наименование подразделения</param>
        public Subdivision(int id, int parentid, string name)
        {
            ID = id;
            ParentID = parentid;
            Name = name;
        }
    }
}

```

Type.cs

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Тип
    /// </summary>
    public class Type
    {
        /// <summary>
        /// Идентификатор типа
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Имя типа
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Группа
        /// </summary>
        public int Group { get; }

        /// <summary>
        /// Конструктор класса Types
        /// </summary>
        /// <param name="id">Идентификатор типа</param>
        /// <param name="name">Имя типа</param>
        /// <param name="group">Группа</param>
        public Type(int id, int group, string name)
        {
            ID = id;
            Name = name;
            Group = group;
        }
    }
}

```

TypeGroup.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Группа типа
    /// </summary>
    class TypeGroup
    {
        /// <summary>
        /// Идентификатор группы
        /// </summary>
        public int ID { get; }

        /// <summary>
        /// Имя группы

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    /// </summary>
    public string Name { get; }
    /// <summary>
    /// Код группы
    /// </summary>
    public int Code { get; }

    /// <summary>
    /// Конструктор класса TypeGroup
    /// </summary>
    /// <param name="id">Идентификатор группы</param>
    /// <param name="name">Имя группы</param>
    /// <param name="code">Код группы</param>
    public TypeGroup(int id, string name, int code)
    {
        ID = id;
        Name = name;
        Code = code;
    }
}

```

User.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Doc.App
{
    /// <summary>
    /// Пользователь системы
    /// </summary>
    public class User
    {
        /// <summary>
        /// Идентификатор записи
        /// </summary>
        public int ID { get; }
        /// <summary>
        /// Идентификатор сотрудника
        /// </summary>
        public int EmplID { get; }
        /// <summary>
        /// Логин
        /// </summary>
        public string Login { get; }
        /// <summary>
        /// Пароль
        /// </summary>
        public string Password { get; }

        /// <summary>
        /// Роль пользователя
        /// </summary>
        public int RoleID { get; }

        public Employee Employee { get; }

        public Role Role { get; }

        /// <summary>

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

/// Конструктор класса User
/// </summary>
/// <param name="id">Идентификатор записи</param>
/// <param name="emplid">Идентификатор сотрудника</param>
/// <param name="login">Логин</param>
/// <param name="passwd">Пароль</param>
public User(int id, int emplid, string login, string passwd, int roleid)
{
    ID = id;
    EmplID = emplid;
    Login = login;
    Password = passwd;
    RoleID = roleid;

    EmployeesManager employeesManager = new EmployeesManager();
    employeesManager.Filter[":_id"] = EmplID;
    List<Employee> list = employeesManager.Select(false);
    if(list.Count > 0)
        Employee = list[0];

    RoleManager roleManager = new RoleManager();
    roleManager.Filter[":_id"] = RoleID;
    List<Role> list1 = roleManager.Select();
    if(list1.Count > 0)
        Role = list1[0];
}
}
}

```

DocServiceCallback.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;

namespace Doc.Console
{
    [CallbackBehavior(UseSynchronizationContext = false)]
    public class DocServiceCallback : IDocServiceCallback
    {
        IDocService pipeProxy = null;
        public bool Connect()
        {
            //note the "DuplexChannelFactory". This is necessary for Callbacks.
            // A regular "ChannelFactory" won't work with callbacks.
            DuplexChannelFactory<IDocService> pipeFactory =
                new DuplexChannelFactory<IDocService>(
                    new InstanceContext(this),
                    new NetTcpBinding(),
                    new EndpointAddress("net.tcp://localhost:8000/ISubscribe"));

            try
            {
                //Open the channel to the server
                pipeProxy = pipeFactory.CreateChannel();
                //Now tell the server who is connecting
                pipeProxy.Subscribe();
                return true;
            }
            catch (Exception e)
            {
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        {
            return false;
        }
    }
    public void Close()
    {
        pipeProxy.Unsubscribe();
    }

    //This function sends a string to the server so that it can broadcast
    // it to all other clients that have called Subscribe().
    public string SendMessage(string message)
    {
        try
        {
            pipeProxy.AddMessage(message);
            return "sent >>>> " + message;
        }
        catch (Exception e)
        {
            return e.Message;
        }
    }

    //This is the function that the SERVER will call
    public void OnMessageAdded(string message, DateTime timestamp)
    {
        // Console.WriteLine(message + ": " + timestamp.ToString("hh:mm:ss"));
    }

    //We need to tell the server that we are leaving
    public void Dispose()
    {
        pipeProxy.Unsubscribe();
    }
}
}

```

IDocService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;
using Utils;

namespace Doc.Console
{
    [ServiceContract(CallbackContract = typeof(IDocServiceCallback))]
    interface IDocService
    {
        /// <summary>
        /// Создаёт новую базу данных
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        void CreateDB();

        /// <summary>
        /// Проверяет наличие БД
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// </summary>
/// <returns>Признак успешности</returns>
[OperationContract]
bool IsDBExists(string name);

/// <summary>
/// Получает конфигурацию СБД
/// </summary>
/// <returns></returns>
[OperationContract]
ConfigSBD GetConfigSbd();

//these are the interface declerations for the server.
[OperationContract]
void AddMessage(string message);
[OperationContract]
bool Subscribe();
[OperationContract]
bool Unsubscribe();
}
}

```

IDocServiceCallback.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;

namespace Doc.Console
{
    [ServiceContract]
    interface IDocServiceCallback
    {
        //This is the callback interface declaration for the client
        [OperationContract(IsOneWay = true)]
        void OnMessageAdded(string message, DateTime timestamp);
    }
}

```

Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;

namespace Doc.Host
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        static void Main()
        {
            ServiceBase[] ServicesToRun;
            ServicesToRun = new ServiceBase[]
            {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        new DocServiceHost()
    };
    ServiceBase.Run(ServicesToRun);
}
}
}

```

DocService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using Npgsql;
using System.Collections;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization.Json;
using Utils;
using NpgsqlTypes;

namespace Doc.Service
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode =
ConcurrencyMode.Reentrant)]
    /// <summary>
    /// Класс для работы с БД
    /// </summary>
    public class DocService : IDocService
    {
        private ConfigSBD _cfg;
        protected Hashtable Clients = new Hashtable();
        private static List<IDocServiceCallback> subscribers = new List<IDocServiceCallback>();
        public ServiceHost host = null;

        /// <summary>
        /// Проверка подключения
        /// </summary>
        /// <returns></returns>
        public bool IsConnected()
        {
            return true;
        }

        public void Connect()
        {
            //I'm doing this next part progromatically instead of in app.cfg
            // because I think it makes it easier to understand (and xml is stupid)
            using (ServiceHost host = new ServiceHost(
                typeof(DocService),
                new Uri("net.tcp://localhost:8000")))
            {
                //notice the NetTcpBinding? This allows programs instead of web stuff
                // to communicate with each other
                host.AddServiceEndpoint(typeof(IDocService),
                    new NetTcpBinding(),
                    "ISubscribe");

                try
                {
                    host.Open();
                    //Console.WriteLine("Successfully opened port 8000.");
                }
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

        //Console.ReadLine();
        host.Close();
    }
    catch (Exception e)
    {

        //Console.WriteLine(e.Message);
    }

}

public bool Subscribe()
{
    try
    {
        //Get the hashCode of the connecting app and store it as a connection
        IDocServiceCallback callback =
OperationContext.Current.GetCallbackChannel<IDocServiceCallback>();
        if (!subscribers.Contains(callback))
            subscribers.Add(callback);
        return true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        return false;
    }
}

public bool Unsubscribe()
{
    try
    {
        //remove any connection that is leaving
        IDocServiceCallback callback =
OperationContext.Current.GetCallbackChannel<IDocServiceCallback>();
        if (subscribers.Contains(callback))
            subscribers.Remove(callback);
        return true;
    }
    catch
    {
        return false;
    }
}

public void AddMessage(string message, bool isedit, bool delete)
{
    //Go through the list of connections and call their callback funciton
    //for (int i = 0; i < subscribers.Count; i++)
    //{
    //    if(subscribers[i] != OperationContext.Current.GetCallbackChannel<IDocServiceCallback>()
    //        && ((ICommunicationObject)subscribers[i]).State == CommunicationState.Opened)
    //        subscribers[i].OnMessageAdded(message, DateTime.Now);
    //    else
    //    {
    //        subscribers.Remove(subscribers[i]);
    //    }
    //}

    subscribers.ForEach(delegate (IDocServiceCallback callback)
    {
        if (((ICommunicationObject)callback).State == CommunicationState.Opened)
        {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        // Console.WriteLine("Calling OnMessageAdded on callback ({0}).", callback.GetHashCode());
        callback.OnMessageAdded(message, isedit, delete);
    }
    //else
    //{
    //    subscribers.Remove(callback);
    //}
});
}

/// <summary>
/// Возвращает строку подключения к БД
/// </summary>
/// <returns>Строка подключения</returns>
public string GetConnectionString()
{
    //чтение конфига службы
    Process[] pr = Process.GetProcessesByName("Doc.Host");
    Process pr1 = pr[0];
    string servConfigPath = pr1.MainModule.FileName.Replace(".exe", ".cfg");

    ConfigManager cfManager = new ConfigManager();
    _cfg = cfManager.GetConfiguration<ConfigSBD>(servConfigPath);

    return _cfg.ConnectionString;
}

/// <summary>
/// Создаёт новую базу данных
/// </summary>
/// <returns>Признак успешности</returns>
public void CreateDB()
{
    string connString = "";
    string tempConnString = GetConnectionString();
    string[] arr = tempConnString.Split(';');
    string nameDB = "";
    foreach (string str in arr)
    {
        if (str.StartsWith("Database"))
        {
            connString = tempConnString.Replace(str, "Database=postgres");
            string[] arr1 = str.Split('=');
            nameDB = arr1[1];

            break;
        }
    }

    StringBuilder createDB = new StringBuilder();
    //Создаем базу
    createDB.AppendFormat("CREATE DATABASE \"{0}\" WITH ", nameDB);
    createDB.AppendLine("OWNER = postgres");
    createDB.AppendLine("ENCODING = 'UTF8'");
    createDB.AppendLine("LC_COLLATE = 'Russian_Russia.1251' ");
    createDB.AppendLine("LC_CTYPE = 'Russian_Russia.1251' ");
    createDB.AppendLine("TABLESPACE = pg_default");
    createDB.AppendLine("CONNECTION LIMIT = -1;");

    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        using (NpgsqlCommand comm = new NpgsqlCommand(createDB.ToString(), conn))

```

Изм.	Лист	№ докум.	Подпись	Дат

```

        {
            comm.ExecuteNonQuery();
        }
        conn.Close();
    }

    CreateTables();
    CreateInsertProcedures();
    CreateUpdateProcedures();
    CreateDelProcedures();
    CreateSelectFunctions();
    //CreateConstraints();
    CreateReferences();
    CreateAdmin();
}

private void CreateTables()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();
    //Таблицы
    //Access
    commString.AppendLine("CREATE TABLE public.\"Access\" ");
    commString.AppendLine("(\"ID\" serial NOT NULL, ");
    commString.AppendLine("\"ObjectID\" bigint NOT NULL, ");
    commString.AppendLine("\"EmplID\" integer NOT NULL, ");
    commString.AppendLine("\"SubID\" integer NOT NULL, ");
    commString.AppendLine("\"AccessType\" integer NOT NULL, ");
    commString.AppendLine("CONSTRAINT \"Access_pkey\" PRIMARY KEY(\"ID\")); ");

    //Address
    commString.AppendLine("CREATE TABLE public.\"Address\" ");
    commString.AppendLine("(\"ID\" serial NOT NULL, ");
    commString.AppendLine("\"PostIndex\" character varying(200) COLLATE pg_catalog.\"default\" NOT
NULL, ");
    commString.AppendLine("\"Country\" character varying(200) COLLATE pg_catalog.\"default\" NOT
NULL, ");
    commString.AppendLine("\"Addr\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
    commString.AppendLine("CONSTRAINT \"Address_pkey\" PRIMARY KEY (\"ID\")); ");

    //Appointments
    commString.AppendLine("CREATE TABLE public.\"Appointments\" ");
    commString.AppendLine("(\"ID\" serial NOT NULL, ");
    commString.AppendLine("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
    commString.AppendLine("CONSTRAINT \"Appointments_pkey\" PRIMARY KEY (\"ID\")); ");

    //Documents
    commString.AppendLine("CREATE TABLE public.\"Documents\" ");
    commString.AppendLine("(\"ID\" bigint NOT NULL, ");
    commString.AppendLine("\"ParentID\" bigint NOT NULL, ");
    commString.AppendLine("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
    commString.AppendLine("\"CurHID\" bigint NOT NULL, ");
    commString.AppendLine("\"RemHID\" bigint NOT NULL, ");
    commString.AppendLine("\"AuthorID\" integer NOT NULL, ");
    commString.AppendLine("\"LockID\" integer NOT NULL, ");
    commString.AppendLine("\"LockHost\" character varying(255) COLLATE pg_catalog.\"default\" NOT
NULL, ");
    commString.AppendLine("\"RouteID\" integer NOT NULL, ");
    commString.AppendLine("\"StageID\" integer NOT NULL, ");
    commString.AppendLine("\"TypeID\" integer NOT NULL, ");
    commString.AppendLine("\"CreateDate\" timestamp without time zone NOT NULL, ");
    commString.AppendLine("\"EditDate\" timestamp without time zone NOT NULL, ");

```

Изм.	Лист	№ докум.	Подпись	Дат

```

commString.AppendLine("\\"Designation\\" character varying(255), ");
commString.AppendLine("\\"Description\\" character varying(255), ");
commString.AppendLine("\\"EditAuthor\\" integer, ");
commString.AppendLine("\\"StageNum\\" integer, ");
commString.AppendLine("CONSTRAINT \\"Documents_pkey\\" PRIMARY KEY(\\"ID\\", \\"CurHID\\"); ");

//Employees
commString.AppendLine("CREATE TABLE public.\\"Employees\\" ");
commString.AppendLine("(\\"ID\\" serial NOT NULL, ");
commString.AppendLine("\\"LastName\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"FirstName\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"MiddleName\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"TypeDoc\\" integer NOT NULL, ");
commString.AppendLine("\\"Doc\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT NULL, ");
commString.AppendLine("\\"AddrID\\" integer NOT NULL, ");
commString.AppendLine("\\"SubdivID\\" integer NOT NULL, ");
commString.AppendLine("\\"AppointID\\" integer NOT NULL, ");
commString.AppendLine("\\"PhotoID\\" integer NOT NULL, ");
commString.AppendLine("\\"RankID\\" integer NOT NULL, ");
commString.AppendLine("\\"Birthday\\" date NOT NULL, ");
commString.AppendLine("\\"WorkPhone\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"HomePhone\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"MobilePhone\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"Deleted\\" bit NOT NULL, ");
commString.AppendLine("CONSTRAINT \\"Employese_pkey\\" PRIMARY KEY (\\"ID\\"); ");

//Files
commString.AppendLine("CREATE TABLE public.\\"Files\\" ");
commString.AppendLine("(\\"ID\\" bigint NOT NULL, ");
commString.AppendLine("\\"Name\\" character varying(200) COLLATE pg_catalog.\\"default\\" NOT NULL,
");
commString.AppendLine("\\"FullPath\\" character varying(255) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"CurHID\\" bigint NOT NULL, ");
commString.AppendLine("\\"RemHID\\" bigint NOT NULL, ");
commString.AppendLine("\\"DocID\\" bigint NOT NULL, ");
commString.AppendLine("\\"Version\\" integer NOT NULL, ");
commString.AppendLine("CONSTRAINT \\"Files_pkey\\" PRIMARY KEY (\\"ID\\", \\"CurHID\\"); ");

//History
commString.AppendLine("CREATE TABLE public.\\"History\\" ");
commString.AppendLine("(\\"HID\\" bigserial NOT NULL, ");
commString.AppendLine("\\"ObjectType\\" integer NOT NULL, ");
commString.AppendLine("\\"ObjectID\\" bigint NOT NULL, ");
commString.AppendLine("\\"DateChange\\" timestamp without time zone NOT NULL, ");
commString.AppendLine("\\"ChangeAuthor\\" integer NOT NULL, ");
commString.AppendLine("\\"ChangeHost\\" character varying(255) COLLATE pg_catalog.\\"default\\" NOT
NULL, ");
commString.AppendLine("\\"Action\\" character varying(255) COLLATE pg_catalog.\\"default\\" NOT NULL,
");
commString.AppendLine("CONSTRAINT \\"History_pkey\\" PRIMARY KEY (\\"HID\\"); ");

//MilitaryRanks
commString.AppendLine("CREATE TABLE public.\\"MilitaryRanks\\" ");
commString.AppendLine("(\\"ID\\" serial NOT NULL, ");
commString.AppendLine("\\"Name\\" character varying(255) COLLATE pg_catalog.\\"default\\" NOT NULL
NOT NULL, ");
commString.AppendLine("CONSTRAINT \\"MilitaryRanks_pkey\\" PRIMARY KEY (\\"ID\\"); ");

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//Photo
commString.AppendLine("CREATE TABLE public.\"Photo\" ");
commString.AppendLine("(\"ID\" serial NOT NULL, ");
commString.AppendLine("\"Photo\" bytea NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Photo_pkey\" PRIMARY KEY (\"ID\")); ");

//PostMessagesIN
commString.AppendLine("CREATE TABLE public.\"PostMessagesIN\" ");
commString.AppendLine("(\"ID\" bigserial NOT NULL, ");
commString.AppendLine("\"Theme\" character varying(200) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"ToUser\" integer NOT NULL, ");
commString.AppendLine("\"FromUser\" integer NOT NULL, ");
commString.AppendLine("\"Message\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"ObjectID\" bigint NOT NULL, ");
commString.AppendLine("\"Readed\" bit(1) NOT NULL, ");
commString.AppendLine("\"Deleted\" bit(1) NOT NULL, ");
commString.AppendLine("\"OutID\" bigint NOT NULL, ");
commString.AppendLine("\"DatePost\" timestamp without time zone NOT NULL, ");
commString.AppendLine("CONSTRAINT \"PostMessagesIN_pkey\" PRIMARY KEY (\"ID\")); ");

//PostMessagesOUT
commString.AppendLine("CREATE TABLE public.\"PostMessagesOut\" ");
commString.AppendLine("(\"ID\" bigserial NOT NULL, ");
commString.AppendLine("\"Theme\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"ToUser\" integer NOT NULL, ");
commString.AppendLine("\"FromUser\" integer NOT NULL, ");
commString.AppendLine("\"Message\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"Readed\" bit(1) NOT NULL, ");
commString.AppendLine("\"Deleted\" bit(1) NOT NULL, ");
commString.AppendLine("\"Sent\" bit(1) NOT NULL, ");
commString.AppendLine("\"ObjectID\" bigint NOT NULL, ");
commString.AppendLine("\"DatePost\" timestamp without time zone NOT NULL, " +
    "\"Auto\" bit(1), ");
commString.AppendLine("CONSTRAINT \"PostMessagesOut_pkey\" PRIMARY KEY (\"ID\")); ");

//Roles
commString.AppendLine("CREATE TABLE public.\"Roles\" ");
commString.AppendLine("(\"ID\" serial NOT NULL, ");
commString.AppendLine("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"ControlAccess\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Roles_pkey\" PRIMARY KEY (\"ID\")); ");

//Routes
commString.AppendLine("CREATE TABLE public.\"Routes\" ");
commString.AppendLine("(\"ID\" serial NOT NULL, ");
commString.AppendLine("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("\"Route\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Routes_pkey\" PRIMARY KEY (\"ID\")); ");

//Signatures
commString.AppendLine("CREATE TABLE public.\"Signatures\" ");
commString.AppendLine("(\"ID\" bigserial NOT NULL, ");
commString.AppendLine("\"TypeID\" integer NOT NULL, ");
commString.AppendLine("\"EmpID\" integer NOT NULL, ");
commString.AppendLine("\"ObjectID\" bigint NOT NULL, ");
commString.AppendLine("\"SignDate\" timestamp without time zone NOT NULL, ");
commString.AppendLine("\"RouteID\" integer NOT NULL, ");

```

Изм.	Лист	№ докум.	Подпись	Дат

```

commString.AppendLine(@"StageNum" integer NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Signatures_pkey\" PRIMARY KEY(\"ID\"); ");

//Stages
commString.AppendLine("CREATE TABLE public.\"Stages\" ");
commString.AppendLine("("\"ID\" serial NOT NULL, ");
commString.AppendLine("("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
commString.AppendLine("("\"SignType\" integer NOT NULL, ");
commString.AppendLine("("\"Lock\" bit(1) NOT NULL, ");
commString.AppendLine("("\"UnLock\" bit(1) NOT NULL, ");
commString.AppendLine("("\"Back\" bit(1) NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Stages_pkey\" PRIMARY KEY(\"ID\"); ");

//SubDivisions
commString.AppendLine("CREATE TABLE public.\"Subdivisions\" ");
commString.AppendLine("("\"ID\" serial NOT NULL, ");
commString.AppendLine("("\"ParentID\" integer NOT NULL, ");
commString.AppendLine("("\"Name\" character varying(200) COLLATE pg_catalog.\"default\" NOT NULL,
");
commString.AppendLine("CONSTRAINT \"Subdivisions_pkey\" PRIMARY KEY (\"ID\"); ");

//TypeGroup
commString.AppendLine("CREATE TABLE public.\"TypeGroup\" ");
commString.AppendLine("("\"ID\" serial NOT NULL, ");
commString.AppendLine("("\"Name\" character varying(200) COLLATE pg_catalog.\"default\" NOT NULL,
");
commString.AppendLine("("\"Code\" integer NOT NULL, ");
commString.AppendLine("CONSTRAINT \"TypeGroup_pkey\" PRIMARY KEY (\"ID\"); ");

//Types
commString.AppendLine("CREATE TABLE public.\"Types\" ");
commString.AppendLine("("\"ID\" serial NOT NULL, ");
commString.AppendLine("("\"GroupID\" integer NOT NULL, ");
commString.AppendLine("("\"Name\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
commString.AppendLine("CONSTRAINT \"Types_pkey\" PRIMARY KEY (\"ID\"); ");

//Users
commString.AppendLine("CREATE TABLE public.\"Users\" ");
commString.AppendLine("("\"ID\" serial NOT NULL, ");
commString.AppendLine("("\"EmplID\" integer NOT NULL, ");
commString.AppendLine("("\"Login\" character varying(255) COLLATE pg_catalog.\"default\" NOT NULL,
");
commString.AppendLine("("\"Password\" character varying(255) COLLATE pg_catalog.\"default\" NOT
NULL, ");
commString.AppendLine("("\"RoleID\" integer NOT NULL, ");
commString.AppendLine("CONSTRAINT \"Users_pkey\" PRIMARY KEY (\"ID\"); ");

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
    conn.Close();
}

private void CreateInsertProcedures()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//Процедуры
//INSERT
//Access
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addaccess(" +
    "_objectid bigint, " +
    "_emplid integer, " +
    "_subid integer, " +
    "_accesstype integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Access\" " +
    "VALUES(nextval(\"Access_ID_seq\"), _ObjectID, _EmplID, _subid, _AccessType); " +
    "$BODY$;");

//Address
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addaddress(" +
    "_postindex character varying, " +
    "_country character varying, " +
    "_addr character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Address\" " +
    "VALUES(nextval(\"Address_ID_seq\"), _PostIndex, _Country, _Addr); " +
    "$BODY$;");

//Appointments
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addappoint(_name
character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Appointments\" VALUES(nextval(\"Appointments_ID_seq\"), _name); " +
    "$BODY$;");

//Documents
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_adddocument(" +
    "_id bigint, " +
    "_parentid bigint, " +
    "_name character varying, " +
    "_currhid bigint, " +
    "_remhid bigint, " +
    "_authorid integer, " +
    "_lockid integer, " +
    "_lockhost character varying, " +
    "_routeid integer, " +
    "_stageid integer, " +
    "_typeid integer, " +
    "_createdate timestamp without time zone, " +
    "_editdate timestamp without time zone, " +
    "_designation character varying, " +
    "_description character varying, " +
    "_editauthor integer, " +
    "_stagenum integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Documents\" VALUES(_ID " +
    ", _ParentID, _Name, _CurrHID, _RemHID " +
    ", _AuthorID, _LockID, _LockHost, _RouteID, _StageID, _TypeID, _CreateDate " +
    ", _EditDate, _Designation, _Description, _EditAuthor, _StageNum); " +
    "$BODY$;");

//Employees
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addemployee(" +
    "_lastname character varying, " +
    "_firstname character varying, " +
    "_middlename character varying, " +
    "_typedoc integer, " +
    "_doc character varying, " +
    "_addrid integer, " +
    "_subdivid integer, "

```

```

        "_appointid integer," +
        "_photoid integer," +
        "_rankid integer," +
        "_birthday date," +
        "_workphone character varying," +
        "_homephone character varying," +
        "_mobilephone character varying," +
        "_deleted bit)" +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"INSERT INTO \"Employes\" VALUES(nextval(\"\"Employes_ID_seq\"\")) " +
", _LastName, _FirstName, _MiddleName, _TypeDoc " +
", _Doc, _AddrID, _SubdivID, _AppointID, _PhotoID, _RankID, _Birthday, _WorkPhone,
_HomePhone, _MobilePhone, _Deleted); " +
"$BODY$;");

//Files
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addfile(_id bigint, " +
    "_name character varying," +
    "_fullpath character varying,_currhid bigint," +
    "_remhid bigint,_docid bigint, _version integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"INSERT INTO \"Files\" VALUES(_ID, _name, _fullpath, _currhid, _remhid, _docid,
_version); " +
"$BODY$;");

//History
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addhistory(_objecttype integer, " +
    "_objectid bigint," +
    "_datechange timestamp without time zone, " +
    "_changeauthor integer, " +
    "_changehost character varying, " +
    "_action character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"INSERT INTO \"History\" " +
"VALUES(nextval(\"\"History_HID_seq\"\"), _objecttype, _objectid, _datechange, " +
    "_changeauthor, _changehost, _action); " +
"$BODY$;");

//MilitaryRanks
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addmilitaryrank(_name
character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ INSERT INTO \"MilitaryRanks\" VALUES(nextval(\"\"MilitaryRanks_ID_seq\"\"),
_name); $BODY$;");

//Photo
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_addphoto(_photo bytea) " +
"LANGUAGE 'sql' " +
"AS $BODY$ INSERT INTO \"Photo\" VALUES(nextval(\"\"Photo_ID_seq\"\"), _Photo);
$BODY$;");

//PostMessageIn
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addpostmessagein(_Theme
character varying, " +
    "_ToUser integer, _FromUser integer, _Message character varying, " +
    "_ObjectID bigint, _Readed bit, _Deleted bit, _OutID bigint, _DatePost timestamp without
time zone) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"INSERT INTO \"PostMessagesIN\" VALUES(nextval(\"\"PostMessagesIN_ID_seq\"\"), " +
    "_Theme, _ToUser, _FromUser, _Message, _ObjectID, _Readed, _Deleted, _OutID,
_DatePost); " +
"$BODY$;");

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//PostMessagesOut
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addpostmessageout" +
    "(" _Theme character varying, _ToUser integer, _FromUser integer, " +
    "_Message character varying, _Readed bit, _Deleted bit, _Sent bit, _ObjectID bigint,
    _DatePost timestamp without time zone," +
    "_Auto bit )" +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"PostMessagesOut\" VALUES(nextval(\"PostMessagesOut_ID_seq\"), " +
    "_Theme, _ToUser, _FromUser, _Message, _Readed, _Deleted, _Sent, _ObjectID,
    _DatePost, _Auto); " +
    "$BODY$; ");

//Roles
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addrole(_Name character varying,
" +
    "_controlaccess character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Roles\" VALUES(nextval(\"Roles_ID_seq\"), _Name, _controlaccess); " +
    "$BODY$; ");

//Routes
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addroute(_Name character
varying, " +
    "_Route character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Routes\" VALUES(nextval(\"Routes_ID_seq\"), _Name, _Route); " +
    "$BODY$; ");

//Signatures
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addsignature(_TypeID integer, " +
    "_EmplID integer, _ObjectID bigint, " +
    "_signdate timestamp without time zone, _routeid integer, _stagenum integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Signatures\" VALUES(nextval(\"Signatures_ID_seq\"), _ " +
    "TypeID, _EmplID, _ObjectID, _SignDate, _routeid, _stagenum); " +
    "$BODY$; ");

//Stages
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addstage(_Name character
varying, _SignType integer, " +
    "_Lock bit, _UnLock bit, _Back bit) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Stages\" VALUES(nextval(\"Stages_ID_seq\"), _Name, _SignType,
    _Lock, _UnLock, _Back); " +
    "$BODY$; ");

//Subdivisions
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addsubdivision(_ParentID integer,
" +
    "_Name character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Subdivisions\" VALUES(nextval(\"Subdivisions_ID_seq\"), " +
    "_ParentID, _Name); $BODY$; ");

//TypeGroup
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addtypegroup(_Name character
varying, _code int) " +
    "LANGUAGE 'sql' ";

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        "AS $BODY$ " +
        "INSERT INTO \"TypeGroup\" VALUES(nextval(\"TypeGroup_ID_seq\"), _Name, _code); "
+
        "$BODY$; ");

//Types
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_addtype(_GroupID integer, " +
    "_Name character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Types\" VALUES(nextval(\"Types_ID_seq\"), _GroupID, _Name); " +
    "$BODY$; ");

//Users
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_adduser(_EmplID integer, " +
    "_Login character varying, _Password character varying, _RoleID integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "INSERT INTO \"Users\" VALUES(nextval(\"Users_ID_seq\"), " +
    "_EmplID, _Login, _Password, _RoleID); " +
    "$BODY$; ");

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
    conn.Close();
}

}

private void CreateSelectFunctions()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();
    //Функции
    //Access
    commString.AppendLine(
        "CREATE OR REPLACE FUNCTION public.fn_getaccess(_objectid bigint, _emplid integer, _subid
integer, _accesstype integer) " +
        "RETURNS TABLE(id integer, objectid bigint, emplid integer, subid integer, accesstype integer) " +
        "LANGUAGE 'sql' " +
        "AS $BODY$ SELECT * FROM \"Access\" " +
        "WHERE \"EmplID\" = COALESCE(_emplid, \"EmplID\") " +
        "AND \"ObjectID\" = COALESCE(_objectid, \"ObjectID\") " +
        "AND \"SubID\" = COALESCE(_subid, \"SubID\") " +
        "AND \"AccessType\" = COALESCE(_accesstype, \"AccessType\"); " +
        "$BODY$; ");

    //Address
    commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getaddress(in _id integer) " +
        "RETURNS TABLE(id integer " +
        ", postindex character varying, country character varying, addr character varying) " +
        "LANGUAGE 'sql' AS " +
        "$BODY$ " +
        "SELECT * FROM \"Address\" " +
        "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
        "$BODY$; ");

    //Appointments
    commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getappointments(in _id integer " +
        ", _name character varying) ");

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

"RETURNS TABLE(id integer" +
", name character varying)" +
"LANGUAGE 'sql' AS " +
"$BODY$ " +
"SELECT * FROM \"Appointments\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")) " +
"$BODY$;");

//Documents
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_getdocuments(_id bigint, _parentid bigint, _name
character varying, _curhid bigint, " +
"_remhid bigint, _authorid integer, _lockid integer, _stageid integer, " +
"_typeid integer, _datecreatebeg timestamp without time zone, " +
"_datecreateend timestamp without time zone, " +
"_dateeditbeg timestamp without time zone, " +
"_dateeditend timestamp without time zone, " +
"_designation character varying, " +
"_editauthor integer) " +
"RETURNS TABLE(id bigint, parentid bigint, name character varying, curhid bigint, remhid bigint,
authorid integer, " +
"lockid integer, lockhost character varying, routeid integer, stageid integer, typeid integer, createdate
timestamp without time zone, " +
"editdate timestamp without time zone, designation character varying, description character varying, "
+
"editauthor integer, stagenum integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT* FROM \"Documents\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND \"ParentID\" = COALESCE(_parentid, \"ParentID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")) " +
"AND \"CurHID\" = COALESCE(_curhid, \"CurHID\") " +
"AND \"RemHID\" = COALESCE(_remhid, \"RemHID\") AND \"AuthorID\" =
COALESCE(_authorid, \"AuthorID\") " +
"AND \"LockID\" = COALESCE(_lockid, \"LockID\") " +
"AND \"StageID\" = COALESCE(_stageid, \"StageID\") " +
"AND \"TypeID\" = COALESCE(_typeid, \"TypeID\") " +
"AND \"CreateDate\" BETWEEN COALESCE(_datecreatebeg, \"CreateDate\") " +
"AND COALESCE(_datecreateend, \"CreateDate\") " +
"AND \"EditDate\" BETWEEN COALESCE(_dateeditbeg, \"EditDate\") " +
"AND COALESCE(_dateeditend, \"EditDate\") " +
"AND UPPER(\"Designation\") LIKE UPPER(COALESCE(_designation, \"Designation\")); " +
"$BODY$;");

//Employees
commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getemployees(in _id integer" +
", _lastname character varying" +
", _firstname character varying" +
", _middlename character varying" +
", _typedoc integer" +
", _doc character varying" +
", _subdivid integer" +
", _appointid integer" +
", _rankid integer" +
", _birthday date" +
", _deleted bit) " +
"RETURNS TABLE(id integer, lastname character varying, firstname character varying,
middlename character varying " +
", typedoc integer, doc character varying, addrid integer, subdivid integer, appointid integer
" +
", photoid integer, rankid integer, birthday date, workphone character varying, homephone
character varying" +
", mobilephone character varying, deleted bit) " +

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

"LANGUAGE 'sql' AS " +
"$BODY$ " +
"SELECT * FROM \"Employees\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"FirstName\") LIKE UPPER(COALESCE(_firstname, \"FirstName\")) " +
"AND UPPER(\"MiddleName\") LIKE UPPER(COALESCE(_middlename, \"MiddleName\"))
" +
"AND UPPER(\"LastName\") LIKE UPPER(COALESCE(_lastname, \"LastName\")) " +
"AND \"TypeDoc\" = COALESCE(_typedoc, \"TypeDoc\") " +
"AND UPPER(\"Doc\") LIKE UPPER(COALESCE(_doc, \"Doc\")) " +
"AND \"AppointID\" = COALESCE(_appointid, \"AppointID\") " +
"AND \"SubdivID\" = COALESCE(_subdivid, \"SubdivID\") " +
"AND \"RankID\" = COALESCE(_rankid, \"RankID\") " +
"AND \"Birthday\" = COALESCE(_birthday, \"Birthday\") " +
"AND \"Deleted\" = COALESCE(_deleted, \"Deleted\"); " +
"$BODY$; ");

```

```

//Files
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_getfiles(_id bigint, _name character varying, _fullpath
character varying " +
", _curhid bigint, _remhid bigint, _docid bigint, _version integer) " +
"RETURNS TABLE(id bigint, name character varying, fullpath character varying, " +
"curhid bigint, remhid bigint, docid bigint, version integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT* FROM \"Files\" WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")) " +
"AND UPPER(\"FullPath\") = UPPER(COALESCE(_fullpath, \"FullPath\")) " +
"AND \"CurHID\" = COALESCE(_curhid, \"CurHID\") " +
"AND \"RemHID\" = COALESCE(_remhid, \"RemHID\") " +
"AND \"DocID\" = COALESCE(_docid, \"DocID\") " +
"AND \"Version\" = COALESCE(_version, \"Version\"); " +
"$BODY$; ");

```

```

//History
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_gethistory(_hid bigint, _objecttype integer, _objectid
bigint " +
", _datechange timestamp without time zone " +
", _changeauthor integer) " +
"RETURNS TABLE(hid bigint, objecttype integer, objectid bigint, datechange timestamp without time
zone " +
", changeauthor integer, changehost character varying, action character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT * FROM \"History\" " +
"WHERE \"HID\" = COALESCE(_hid, \"HID\") " +
"AND \"ObjectType\" = COALESCE(_objecttype, \"ObjectType\") " +
"AND \"ObjectID\" = COALESCE(_objectid, \"ObjectID\") " +
"AND \"DateChange\" = COALESCE(_datechange, \"DateChange\") " +
"AND \"ChangeAuthor\" = COALESCE(_changeauthor, \"ChangeAuthor\"); " +
"$BODY$; ");

```

```

//MilitaryRanks
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_getmilitaryranks(_id integer, _name character varying)
" +
"RETURNS TABLE(id integer, name character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT * FROM \"MilitaryRanks\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")); " +
"$BODY$; ");

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

//Photo
commString.AppendLine("CREATE OR REPLACE FUNCTION public.fn_getphoto(_id integer) " +
    "RETURNS TABLE(id integer, photo bytea) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ SELECT * FROM \"Photo\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\"); " +
    "$BODY$;");

//PostMessagesIN
commString.AppendLine(
    "CREATE OR REPLACE FUNCTION public.fn_getpostmessagesin(_id bigint, _touser integer,
    _fromuser integer, _objectid integer " +
    ", _outid bigint, " +
    "_datebeg timestamp without time zone, _dateend timestamp without time zone) " +
    "RETURNS TABLE(id bigint, theme character varying, touser integer, fromuser integer, message
    character varying " +
    ", objectid bigint, readed bit, deleted bit, outid bigint, " +
    "datepost timestamp without time zone) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ SELECT * FROM \"PostMessagesIN\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
    "AND \"ToUser\" = COALESCE(_touser, \"ToUser\") " +
    "AND \"FromUser\" = COALESCE(_fromuser, \"FromUser\") " +
    "AND \"ObjectID\" = COALESCE(_objectid, \"ObjectID\") " +
    "AND \"OutID\" = COALESCE(_outid, \"OutID\") " +
    "AND \"DatePost\" BETWEEN COALESCE(_datebeg, \"DatePost\") AND
    COALESCE(_dateend, \"DatePost\"); " +
    "$BODY$;");

//PostMessagesOut
commString.AppendLine(
    "CREATE OR REPLACE FUNCTION public.fn_getpostmessagesout(_id bigint, _touser integer,
    _fromuser integer, " +
    "_objectid integer, _datebeg timestamp without time zone, " +
    "_dateend timestamp without time zone, _auto bit) " +
    "RETURNS TABLE(id bigint, theme character varying, touser integer, fromuser integer, message
    character varying, readed bit " +
    ", deleted bit, sent bit, objectid bigint, " +
    "datepost timestamp without time zone, auto bit ) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ SELECT * FROM \"PostMessagesOut\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
    "AND \"ToUser\" = COALESCE(_touser, \"ToUser\") " +
    "AND \"FromUser\" = COALESCE(_fromuser, \"FromUser\") " +
    "AND \"ObjectID\" = COALESCE(_objectid, \"ObjectID\") " +
    "AND \"DatePost\" BETWEEN COALESCE(_datebeg, \"DatePost\") " +
    "AND COALESCE(_dateend, \"DatePost\") " +
    "AND \"Auto\" = COALESCE(_auto, \"Auto\"); " +
    "$BODY$;");

//Roles
commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getroles(in _id integer " +
    ", _name character varying ) " +
    "RETURNS TABLE(id integer " +
    ", name character varying " +
    ", controlaccess character varying ) " +
    "LANGUAGE 'sql' AS " +
    "$BODY$ " +
    "SELECT * FROM \"Roles\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
    "AND UPPER(\"Name\") = UPPER(COALESCE(_name, \"Name\")) " +
    "$BODY$;");

//Routes
commString.AppendLine(

```

```
"CREATE OR REPLACE FUNCTION public.fn_getroutes(_id integer, _name character varying) " +
"RETURNS TABLE(id integer, name character varying, route character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT * FROM \"Routes\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")); " +
"$BODY$";
```

//Signatures

```
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_getsignatures(_id bigint, _typeid integer, " +
"_emplid integer, _objectid bigint, " +
"_signdate timestamp without time zone, _routeid integer) " +
"RETURNS TABLE(id bigint, typeid integer, emplid integer, objectid bigint) " +
", signdate timestamp without time zone, routeid integer, stagenum integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT * FROM \"Signatures\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND \"TypeID\" = COALESCE(_typeid, \"TypeID\") " +
"AND \"EmplID\" = COALESCE(_emplid, \"EmplID\") " +
"AND \"ObjectID\" = COALESCE(_objectid, \"ObjectID\") " +
"AND \"SignDate\" = COALESCE(_signdate, \"SignDate\") " +
"AND \"RouteID\" = COALESCE(_routeid, \"RouteID\"); " +
"$BODY$";
```

//Stages

```
commString.AppendLine(
"CREATE OR REPLACE FUNCTION public.fn_getstages(_id integer, _name character varying) " +
"RETURNS TABLE(id integer, name character varying, signtype integer, lock bit, unlock bit, back bit) " +
+
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"SELECT * FROM \"Stages\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")); " +
"$BODY$";
```

//Subdivisions

```
commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getsubdivisions(in _id integer " +
", _parentid integer, _name character varying) " +
"RETURNS TABLE(id integer) " +
", parentid integer, name character varying) " +
"LANGUAGE 'sql' AS " +
"$BODY$ " +
"SELECT* FROM \"Subdivisions\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND \"ParentID\" = COALESCE(_parentid, \"ParentID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")); " +
"$BODY$";
```

//TypeGroup

```
commString.AppendLine("CREATE OR REPLACE FUNCTION fn_gettypegroups(in _id integer " +
", _name character varying, _code integer) " +
"RETURNS TABLE(id integer) " +
", name character varying, code integer) " +
"LANGUAGE 'sql' AS " +
"$BODY$ " +
"SELECT * FROM \"TypeGroup\" " +
"WHERE \"ID\" = COALESCE(_id, \"ID\") " +
"AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")) " +
"AND \"Code\" = COALESCE(_code, \"Code\") " +
"$BODY$";
```

//Types

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

commString.AppendLine("CREATE OR REPLACE FUNCTION fn_gettypes(in _id integer " +
    ", _groupid integer, _name character varying )" +
    "RETURNS TABLE(id integer " +
    ", groupid integer, name character varying) " +
    "LANGUAGE 'sql' AS " +
    "$BODY$ " +
    "SELECT * FROM \"Types\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
    "AND \"GroupID\" = COALESCE(_groupid, \"GroupID\") " +
    "AND UPPER(\"Name\") LIKE UPPER(COALESCE(_name, \"Name\")) " +
    "$BODY$;");

//Users
commString.AppendLine("CREATE OR REPLACE FUNCTION fn_getusers(in _id integer " +
    ", _emplid integer " +
    ", _login character varying " +
    ", _roleid integer) " +
    "RETURNS TABLE(id integer " +
    ", emplid integer " +
    ", login character varying" +
    ", password character varying" +
    ", roleid integer ) " +
    "LANGUAGE 'sql' AS " +
    "$BODY$ " +
    "SELECT * FROM \"Users\" " +
    "WHERE \"ID\" = COALESCE(_id, \"ID\") " +
    "AND \"EmplID\" = COALESCE(_emplid, \"EmplID\") " +
    "AND UPPER(\"Login\") = UPPER(COALESCE(_login, \"Login\")) " +
    "AND \"RoleID\" = COALESCE(_roleid, \"RoleID\") " +
    "$BODY$;");

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
    conn.Close();
}

}

private void CreateUpdateProcedures()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();

    //Access
    commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updateaccess(_id bigint,
    _accesstype integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Access\" " +
    "SET \"AccessType\" = _accesstype " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

    //Address
    commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updateaddress(_id integer, _postindex character varying,
    " +
    " _country character varying, " +
    " _addr character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Address\" " +
    "SET \"PostIndex\" = _postindex, \"Country\" = _country, \"Addr\" = _addr " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");
}

```

```

"AS $BODY$ UPDATE \"Address\" " +
"SET \"PostIndex\" = _postindex, " +
"\"Country\" = _country, " +
"\"Addr\" = _addr " +
"WHERE \"ID\" = _id; " +
"$BODY$;");

//Appointments
commString.AppendLine(
"CREATE OR REPLACE PROCEDURE sp_updateappoint(_id integer, _name character varying) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"UPDATE \"Appointments\" " +
"SET \"Name\" = _name " +
"WHERE \"ID\" = _id; " +
"$BODY$;");

//Documents
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updatedocument(_id bigint,
_parentid bigint, " +
" _name character varying, _currhid bigint, _remhid bigint, _lockid integer, _lockhost
character varying, _typeid integer, _routeid integer, " +
" _stageid integer, _editdate timestamp without time zone, " +
" _designation character varying, _description character varying, " +
" _editauthor integer, _stagenum integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ UPDATE \"Documents\" " +
"SET \"ParentID\" = _parentid, \"Name\" = _name, \"RemHID\" = _remhid, " +
"\"LockID\" = _lockid, \"LockHost\" = _lockhost, " +
"\"RouteID\" = _routeid, \"StageID\" = _stageid, " +
"\"EditDate\" = _editdate, \"Designation\" = _designation, \"Description\" = _description, " +
"\"EditAuthor\" = _editauthor, " +
"\"CurHID\" = _currhid, " +
"\"TypeID\" = _typeid, " +
"\"StageNum\" = _stagenum " +
"WHERE \"ID\" = _id AND \"RemHID\" = 0; " +
"$BODY$;");

//Employees
commString.AppendLine(
"CREATE OR REPLACE PROCEDURE public.sp_updateemployee(_id integer, _lastname character
varying, " +
" _firstname character varying, _middlename character varying, _typedoc integer, " +
" _doc character varying, _addrid integer, _subdivid integer, _appointid integer, " +
" _rankid integer, _photoid integer, _birthday date, _workphone character varying, _homephone
character varying, " +
" _mobilephone character varying, _deleted bit) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +
"UPDATE \"Employees\" " +
"SET \"LastName\" = _lastname, \"FirstName\" = _firstname, \"MiddleName\" = _middlename, " +
"\"TypeDoc\" = _typedoc, \"Doc\" = _doc, \"AddrID\" = _addrid, \"SubdivID\" = _subdivid, \"AppointID\"
= _appointid, " +
"\"PhotoID\" = _photoid, \"RankID\" = _rankid, \"Birthday\" = _birthday, \"WorkPhone\" = _workphone"
+
" , \"HomePhone\" = _homephone, \"MobilePhone\" = _mobilephone, \"Deleted\" = _deleted " +
"WHERE \"ID\" = _id; " +
"$BODY$;");

//Files
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updatefile(_id bigint, _currhid
bigint, " +
" _remhid bigint, _name character varying, _version integer) " +
"LANGUAGE 'sql' " +
"AS $BODY$ " +

```

ПГУ 1.090501.03.001 ПЗ

Лист

280

Изм.	Лист	№ докум.	Подпись	Дат

```

        "UPDATE \"Files\" SET \"RemHID\" = _remhid, " +
        "\"Name\" = _name, " +
        "\"Version\" = _version " +
        "WHERE \"ID\" = _id AND \"CurHID\" = _currhid; " +
        "$BODY$;");

//MilitaryRanks
commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updatemilitaryrank(_id integer, _name character varying)
" +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"MilitaryRanks\" " +
    "SET \"Name\" = _name " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Photo
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updatephoto(_id integer, _photo
bytea) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Photo\" " +
    "SET \"Photo\" = _photo " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//PostmessagesOut
commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updatepostmessageout(_id bigint, _theme character
varying, " +
    "_touser integer, _message character varying, _readed bit, _deleted bit, " +
    "_sent bit, _objectid bigint, _datepost timestamp without time zone) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"PostMessagesOut\" " +
    "SET \"Theme\" = _theme, \"ToUser\" = _touser, " +
    "\"Message\" = _message, \"Readed\" = _readed, \"Deleted\" = _deleted, " +
    "\"Sent\" = _sent, \"ObjectID\" = _objectid, " +
    "\"DatePost\" = _datepost " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//PostmessagesIN
commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updatepostmessagein(_id bigint, _readed bit, _deleted
bit) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"PostMessagesIN\" " +
    "SET \"Readed\" = _readed, \"Deleted\" = _deleted WHERE \"ID\" = _id; " +
    "$BODY$;");

//Roles
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updaterole(_id integer, _name
character varying, _
    "controlaccess character varying ) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Roles\" " +
    "SET \"Name\" = _name, \"ControlAccess\" = _controlaccess " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Routes

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updateroute(_id integer, _name
character varying, _ " +
    "route character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Routes\" " +
    "SET \"Name\" = _name, \"Route\" = _route " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Stages
commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updatestage(_id integer, _name character varying,
    _signtype integer, " +
    "_lock bit, _unlock bit, _back bit) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Stages\" SET \"Name\" = _name, " +
    "\"SignType\" = _signtype, " +
    "\"Lock\" = _lock, " +
    "\"UnLock\" = _unlock, " +
    "\"Back\" = _back " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Subdivisions
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updatesubdivision(_id integer,
    _parentid integer, " +
    "_name character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Subdivisions\" " +
    "SET \"Name\" = _name, " +
    "\"ParentID\" = _parentid " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Types
commString.AppendLine(
    "CREATE OR REPLACE PROCEDURE sp_updatetype(_id integer, _groupid integer, _name
character varying) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Types\" " +
    "SET \"GroupID\" = _groupid, \"Name\" = _name " +
    "WHERE \"ID\" = _id; " +
    "$BODY$;");

//Users
commString.AppendLine("CREATE OR REPLACE PROCEDURE sp_updateuser(_id integer, _login
character varying, " +
    "_password character varying, _roleid integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ " +
    "UPDATE \"Users\" " +
    "SET \"Login\" = _login, \"Password\" = _password, " +
    "\"RoleID\" = _roleid " +
    "WHERE \"ID\" = _id; $BODY$;");

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }
    conn.Close();
}

private void CreateConstraints()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();

    //Access
    commString.AppendLine("ALTER TABLE \"Access\" " +
        "ADD CONSTRAINT \"Access_Documents_ID\" FOREIGN KEY(\"ObjectID\") " +
        "REFERENCES \"Documents\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE CASCADE; " +
        "CREATE INDEX \"fki_Access_Documents_ID\" " +
        "ON \"Access\"(\"ObjectID\");");

    commString.AppendLine("ALTER TABLE \"Access\" " +
        "ADD CONSTRAINT \"Acces_User_ID\" FOREIGN KEY(\"UserID\") " +
        "REFERENCES \"Users\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE CASCADE; " +
        "CREATE INDEX \"fki_Acces_User_ID\" " +
        "ON \"Access\"(\"UserID\");");

    commString.AppendLine(
        "ALTER TABLE \"Access\" ADD CONSTRAINT \"Access_Type_ID\" FOREIGN KEY(\"AccessType\") "
+
        "REFERENCES \"Types\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE RESTRICT; " +
        "CREATE INDEX \"fki_Access_Type_ID\" " +
        "ON \"Access\"(\"AccessType\");");

    commString.AppendLine("ALTER TABLE \"Documents\" " +
        "ADD CONSTRAINT \"Documents_Stage_ID\" FOREIGN KEY(\"StageID\") " +
        "REFERENCES \"Stages\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE RESTRICT; " +
        "CREATE INDEX \"fki_Documents_Stage_ID\" " +
        "ON \"Documents\"(\"StageID\");");

    commString.AppendLine("ALTER TABLE \"Documents\" " +
        "ADD CONSTRAINT \"Documents_Type_ID\" FOREIGN KEY(\"TypeID\") " +
        "REFERENCES \"Types\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE RESTRICT; " +
        "CREATE INDEX \"fki_Documents_Type_ID\" " +
        "ON \"Documents\"(\"TypeID\");");

    commString.AppendLine("ALTER TABLE \"Documents\" " +
        "ADD CONSTRAINT \"Documents_Route_ID\" FOREIGN KEY(\"RouteID\") " +
        "REFERENCES \"Routes\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE NO ACTION; " +
        "CREATE INDEX \"fki_Documents_Route_ID\" " +
        "ON \"Documents\"(\"RouteID\");");

    //Signatures
    commString.AppendLine("ALTER TABLE \"Signatures\" " +
        "ADD CONSTRAINT \"Signatures_Types_ID\" FOREIGN KEY(\"TypeID\") " +
        "REFERENCES \"Types\" (\"ID\") MATCH SIMPLE " +
        "ON UPDATE NO ACTION " +
        "ON DELETE RESTRICT; " +
        "CREATE INDEX \"fki_Signatures_Types_ID\" " +
        "ON \"Signatures\"(\"TypeID\");");

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

//Types
commString.AppendLine(
    "ALTER TABLE \"Types\" ADD CONSTRAINT \"Type_TypeGroup_ID\" FOREIGN KEY(\"GroupID\") \"
+
    \"REFERENCES \"TypeGroup\" (\"ID\") MATCH SIMPLE \" +
    \"ON UPDATE NO ACTION \" +
    \"ON DELETE RESTRICT; \" +
    \"CREATE INDEX \"fki_Type_TypeGroup_ID\" \" +
    \"ON \"Types\"(\"GroupID\");\";

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
    conn.Close();
}

private void CreateDelProcedures()
{
    StringBuilder commString = new StringBuilder();
    string connString = GetConnectionString();

    //Access
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE sp_delaccess(_id bigint) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"Access\" WHERE \"ID\" = _id; $BODY$; \");

    //Appointments
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE sp_delappoint(_id integer) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"Appointments\" WHERE \"ID\" = _id; $BODY$; \");

    //MilitaryRanks
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE public.sp_delmilitaryrank(_id integer) \"
+
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"MilitaryRanks\" WHERE \"ID\" = _id; $BODY$; \");

    //Photo
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE public.sp_delphoto(_id integer) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"Photo\" WHERE \"ID\" = _id; $BODY$; \");

    //PostMessagesIN
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE public.sp_delpostmessagein(_id
bigint) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"PostMessagesIN\" WHERE \"ID\" = _id; $BODY$; \");

    //PostMessagesOut
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE public.sp_delpostmessageout(_id
bigint) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"PostMessagesOut\" WHERE \"ID\" = _id; $BODY$; \");

    //Routes
    commString.AppendLine(\"CREATE OR REPLACE PROCEDURE public.sp_delroute(_id integer) \" +
        \"LANGUAGE 'sql' \" +
        \"AS $BODY$ DELETE FROM \"Routes\" WHERE \"ID\" = _id; $BODY$; \");

    //Roles

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_delrole(_id integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Roles\" WHERE \"ID\" = _id; $BODY$; ");

//Signatures
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_delsignature(_id bigint) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Signatures\" WHERE \"ID\" = _id; $BODY$; ");

//Stages
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_delstage(_id integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Stages\" WHERE \"ID\" = _id; $BODY$; ");

//Subdivisions
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_delsubdivision(_id integer) "
+
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Subdivisions\" WHERE \"ID\" = _id; $BODY$; ");

//Types
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_deltypes(_id integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Types\" WHERE \"ID\" = _id; $BODY$; ");

//Users
commString.AppendLine("CREATE OR REPLACE PROCEDURE public.sp_deluser(_id integer) " +
    "LANGUAGE 'sql' " +
    "AS $BODY$ DELETE FROM \"Users\" WHERE \"ID\" = _id; $BODY$; ");

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
    {
        comm.ExecuteNonQuery();
    }
    conn.Close();
}

private void CreateReferences()
{
    string connString = GetConnectionString();
    //Роли
    string controlAdmin = "$" + ((int)RoleAccess.DocControl).ToString() + "$" +
        ((int)RoleAccess.EmployeesControl).ToString() + "$" +
        ((int)RoleAccess.RouteControl).ToString() + "$" +
        ((int)RoleAccess.StageControl).ToString() + "$" +
        ((int)RoleAccess.TypeControl).ToString() + "$" +
        ((int)RoleAccess.UserControl).ToString() + "$" +
        ((int)RoleAccess.AccessControl).ToString() + "$" +
        ((int)RoleAccess.AppointmentControl).ToString() + "$" +
        ((int)RoleAccess.MilitaryRankControl).ToString() + "$" +
        ((int)RoleAccess.RoleControl).ToString() + "$" +
        ((int)RoleAccess.SubdivisionControl).ToString() + "$" +
        ((int)RoleAccess.TypGroupControl).ToString() + "$" +
        ((int)RoleAccess.UnlockDocument).ToString() + "$" +
        ((int)RoleAccess.BrowseDeletedDocument).ToString() + "$" +
        ((int)RoleAccess.DelArchiveDocument).ToString() + "$";

    string controlUser = "$" + ((int)RoleAccess.DocControl).ToString() + "$";

```

Изм.	Лист	№ докум.	Подпись	Дат

```

string commString = string.Format("CALL sp_addrole('Администратор', '{0}'); " +
    "CALL sp_addrole('Пользователь', '{1}'); " +
    //Группы типов
    "CALL sp_addtypegroup('Тип доступа', 1);" +
    "CALL sp_addtypegroup('Тип удостоверения личности', 2);" +
    "CALL sp_addtypegroup('Тип документа', 3);" +
    "CALL sp_addtypegroup('Тип файла', 4);" +
    "CALL sp_addtypegroup('Тип подписи', 5); ", controlAdmin, controlUser);

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    conn.Open();
    using (NpgsqlCommand comm = new NpgsqlCommand(commString, conn))
    {
        comm.ExecuteNonQuery();
    }

    //Типы
    string selectTypeGroup = "SELECT * FROM fn_gettypegroups(NULL, NULL, NULL)";

    Hashtable types = new Hashtable();

    using (NpgsqlCommand selectCommand = new NpgsqlCommand(selectTypeGroup, conn))
    {
        using (IDataReader reader = selectCommand.ExecuteReader())
        {
            int typeGroup = 0;

            while (reader.Read())
            {
                typeGroup = (int)reader["ID"];
                ArrayList list = null;
                switch (reader["Name"].ToString())
                {
                    case "Тип доступа":
                        list = new ArrayList(new ArrayList { "Чтение", "Редактирование", "Полный доступ",
"Доступ запрещён" });
                        break;
                    case "Тип удостоверения личности":
                        list = new ArrayList(new ArrayList { "Паспорт" });
                        break;
                    case "Тип документа":
                        list = new ArrayList(new ArrayList { "Служебная записка", "Приказ", "Договор" });
                        break;
                    case "Тип файла":
                        list = new ArrayList(new ArrayList { "*.doc", "*.docx", "*.xls", "*.odt", "*.pdf", "*.jpg",
"*.bmp", "*.png", "*.tiff" });
                        break;
                    case "Тип подписи":
                        list = new ArrayList(new ArrayList { "Разработал", "Проверил", "Согласовал",
"Утвердил", "Поставил на хранение" });
                        break;
                }
                types.Add(typeGroup, list);
            }
        }

        foreach (DictionaryEntry type in types)
        {
            int typeGroup = (int)type.Key;
            foreach (string item in (ArrayList)type.Value)
            {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        using (NpgsqlCommand commType = new NpgsqlCommand(string.Format("CALL
sp_addtype({0}, '{1}']", typeGroup, item), conn))
        {
            commType.ExecuteNonQuery();
        }
    }
}

conn.Close();
}

}

private void CreateAdmin()
{
    string connString = GetConnectionString();
    string addEmployee = "CALL sp_addemployee('Системы', 'Администратор', ", 0, ", 0, 0, 0, 0, 0, '1900-
01-01'" +
        ", ", ", ", ", '0')";

    string getRole = "SELECT * FROM fn_getroles(NULL, 'Администратор')";

    string getEmployee = "SELECT * FROM fn_getemployees(NULL, 'Системы' , NULL, NULL, NULL, NULL,
" +
        "NULL, NULL, NULL, NULL, '0')";

    int emplid = 0;
    int roleid = 0;

    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        //Добавляем сотрудника
        using (NpgsqlCommand comm = new NpgsqlCommand(addEmployee, conn))
        {
            comm.ExecuteNonQuery();
        }

        //получаем id сотрудника
        using (NpgsqlCommand comm = new NpgsqlCommand(getEmployee, conn))
        {
            using (IDataReader reader = comm.ExecuteReader())
            {
                while (reader.Read())
                {
                    emplid = (int)reader["id"];
                }
            }
        }

        //получаем роль администратора

        using (NpgsqlCommand comm = new NpgsqlCommand(getRole, conn))
        {
            using (IDataReader reader = comm.ExecuteReader())
            {
                while (reader.Read())
                {
                    roleid = (int)reader["id"];
                }
            }
        }
    }
}

```

//Создаем пользователя системы с логином 1 и паролем 1

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        string addUser = string.Format("CALL sp_adduser({0}, '1', '{1}', {2})", emplid,
Encrypt.HashPassword("1"), roleid);

        using (NpgsqlCommand comm = new NpgsqlCommand(addUser, conn))
        {
            comm.ExecuteNonQuery();
        }

        conn.Close();
    }
}

/// <summary>
/// Проверяет наличие БД
/// </summary>
/// <returns>Признак успешности</returns>
public bool IsDBExists(string name)
{
    string connString = "";
    string tempConnString = GetConnectionString();
    string[] arr = tempConnString.Split(';');
    foreach (string str in arr)
    {
        if (str.StartsWith("Database"))
        {
            connString = tempConnString.Replace(str, "Database=postgres");
            break;
        }
    }

    StringBuilder commString = new StringBuilder();

    commString.AppendFormat("SELECT * FROM pg_database WHERE datname = '{0}';", name);
    bool isexists = false;

    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        using (NpgsqlCommand comm = new NpgsqlCommand(commString.ToString(), conn))
        {
            if (comm.ExecuteScalar() != null)
            {
                isexists = true;
            }
        }
    }

    return isexists;
}

/// <summary>
/// Выполняет запрос к БД
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="param">Передаваемые параметры</param>
/// <returns>Результат запроса</returns>
public ArrayList ExecuteQuery(string query, Hashtable param)
{
    ArrayList list = new ArrayList();
    string connString = GetConnectionString();
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        using (NpgsqlCommand comm = new NpgsqlCommand(query, conn))

```

```

    {
        if (param != null)
        {
            foreach (DictionaryEntry de in param)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),

                    Value = de.Value ?? DBNull.Value
                };
                comm.Parameters.Add(par);
            }
        }
    }

    using (IDataReader reader = comm.ExecuteReader())
    {
        while (reader.Read())
        {
            StringBuilder str = new StringBuilder();
            for (int i = 0; i < reader.FieldCount; i++)
            {
                str.Append(reader[i].ToString() + "|");
            }
            str.Remove(str.Length - 1, 1);
            list.Add(str.ToString());
        }
    }
}

return list;
}

/// <summary>
/// Выполняет вставку удаление, изменение записей в таблицах
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="lstparam">Передаваемые параметры</param>
public void ExecuteNonQueryTr(string query, List<Hashtable> lstparam)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;

    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            transaction = conn.BeginTransaction();
            foreach (var param in lstparam)
            {
                using (NpgsqlCommand comm = new NpgsqlCommand(query, conn))
                {
                    comm.Transaction = transaction;
                    foreach (DictionaryEntry de in param)
                    {
                        var par = new NpgsqlParameter
                        {
                            ParameterName = de.Key.ToString(),
                            Value = de.Value
                        };
                        if (par.ParameterName == ":_photo")
                            par.NpgsqlDbType = NpgsqlDbType.Bytea;
                        comm.Parameters.Add(par);
                    }
                }
            }
        }
        catch { }
        finally { transaction?.Dispose(); }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        }
        comm.ExecuteNonQuery();
    }
    transaction.Commit();
}
catch (Exception e)
{
    transaction.Rollback();
}
finally
{
    conn.Close();
}
}
}

/// <summary>
/// Выполняет вставку удаление, изменение записей в таблицах
/// </summary>
/// <param name="query">Запрос</param>
/// <param name="param">Передаваемые параметры</param>
public void ExecuteNonQuery(string query, Hashtable param)
{
    string connString = GetConnectionString();
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        using (NpgsqlCommand comm = new NpgsqlCommand(query, conn))
        {
            foreach (DictionaryEntry de in param)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),
                    Value = de.Value
                };
                if (par.ParameterName == "._photo")
                    par.NpgsqlDbType = NpgsqlDbType.Bytea;
                comm.Parameters.Add(par);
            }

            comm.ExecuteNonQuery();
        }
        conn.Close();
    }
}

public Hashtable GetPhoto(int id)
{
    Hashtable result = new Hashtable();
    string connString = GetConnectionString();
    string query = string.Format("SELECT * FROM fn_getphoto({0});", id);
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        conn.Open();
        using (NpgsqlCommand comm = new NpgsqlCommand(query, conn))
        {
            using (IDataReader reader = comm.ExecuteReader())
            {
                while (reader.Read())
                {
                    result.Add(reader[0], reader[1]);
                }
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }
    conn.Close();
}

return result;
}

/// <summary>
/// Добавить документ
/// </summary>
public void InsertDocument(Hashtable querys, Hashtable param, List<Hashtable> parAdminAccess,
List<Hashtable> parFiles)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;
    Hashtable parDoc = null;
    long lastDocId = 0;
    long lastHID = 0;
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            //Открываем транзакцию
            transaction = conn.BeginTransaction();

            //1. Последний документ
            string strLastDocument = (string)querys["LastDocument"];
            using (NpgsqlCommand commLastDocument = new NpgsqlCommand(strLastDocument, conn))
            {
                commLastDocument.Transaction = transaction;

                var res = commLastDocument.ExecuteScalar();

                if (res != null)
                    lastDocId = (long)res;
            }

            //3. Инсерт в History
            string strInsertHistory = (string)querys["InsertHistory"];
            using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
            {
                commInsertHistory.Transaction = transaction;

                Hashtable parHis = (Hashtable)param["History"];

                foreach (DictionaryEntry de in parHis)
                {
                    var par = new NpgsqlParameter
                    {
                        ParameterName = de.Key.ToString(),
                        Value = de.Value
                    };
                    if (par.ParameterName == ":_objectid")
                        par.Value = lastDocId + 1;

                    commInsertHistory.Parameters.Add(par);
                }
                commInsertHistory.ExecuteNonQuery();
            }

            //4. Получаем последний HID
            string strLastHistory = (string)querys["LastHistory"];
            using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))

```

```

{
    commLastHistory.Transaction = transaction;

    var res = commLastHistory.ExecuteScalar();

    if (res != null)
        lastHID = (long)res;
}

//2. Insert в Document
string strInsertDoc = (string)queryrs["InsertDocument"];
using (NpgsqlCommand commInsertDocument = new NpgsqlCommand(strInsertDoc, conn))
{
    commInsertDocument.Transaction = transaction;

    parDoc = (Hashtable)param["Document"];

    foreach (DictionaryEntry de in parDoc)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_id")
            par.Value = lastDocId + 1;
        if (par.ParameterName == ":_currhid")
            par.Value = lastHID;

        commInsertDocument.Parameters.Add(par);
    }
    commInsertDocument.ExecuteNonQuery();
}

if (parFiles.Count > 0)
{
    string strLastFile = (string)queryrs["LastFile"];
    long lastFileId = 0;
    foreach (var parFile in parFiles)
    {
        //6. Последний ID файла
        using (NpgsqlCommand commLastFile = new NpgsqlCommand(strLastFile, conn))
        {
            commLastFile.Transaction = transaction;
            var res = commLastFile.ExecuteScalar();

            if (res != null)
                lastFileId = (long)res;
        }

        //8. вставка в таблицу History queryHis
        using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
        {
            commInsertHistory.Transaction = transaction;

            Hashtable parHisFiles = (Hashtable)param["HisFiles"];

            foreach (DictionaryEntry de in parHisFiles)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),
                    Value = de.Value
                };
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        if (par.ParameterName == ":_objectid")
            par.Value = lastFileId + 1;

        commInsertHistory.Parameters.Add(par);
    }
    commInsertHistory.ExecuteNonQuery();
}

//9. Последний HID для файла
using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
{
    commLastHistory.Transaction = transaction;

    var res = commLastHistory.ExecuteScalar();

    if (res != null)
        lastHID = (long)res;
}

//7. вставка в таблицу Files
string strInsertFiles = (string)query["InsertFile"];
using (NpgsqlCommand commInsertFile = new NpgsqlCommand(strInsertFiles, conn))
{
    commInsertFile.Transaction = transaction;

    foreach (DictionaryEntry de in parFile)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_id")
            par.Value = lastFileId + 1;
        if (par.ParameterName == ":_docid")
            par.Value = lastDocId + 1;
        if (par.ParameterName == ":_currhid")
            par.Value = lastHID;

        commInsertFile.Parameters.Add(par);
    }
    commInsertFile.ExecuteNonQuery();
}
}

//6. Назначаем полный доступ для автора
string strInsertAccess = (string)query["InsertAccess"];
Hashtable parAccess = (Hashtable)param["Access"];
using (NpgsqlCommand commInsertAccess = new NpgsqlCommand(strInsertAccess, conn))
{
    commInsertAccess.Transaction = transaction;
    foreach (DictionaryEntry de in parAccess)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };

        if (par.ParameterName == ":_objectid")
            par.Value = lastDocId + 1;

        commInsertAccess.Parameters.Add(par);
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    }

    commInsertAccess.ExecuteNonQuery();
}
//Доступ администраторам
if(parAdminAccess.Count > 0)
{
    foreach (var parameter in parAdminAccess)
    {
        using (NpgsqlCommand comm = new NpgsqlCommand(strInsertAccess, conn))
        {
            comm.Transaction = transaction;
            foreach (DictionaryEntry de in parameter)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),
                    Value = de.Value
                };
                if (par.ParameterName == ":_objectid")
                    par.Value = lastDocId + 1;
                comm.Parameters.Add(par);
            }

            comm.ExecuteNonQuery();
        }
    }
}

transaction.Commit();
}
catch (Exception ex)
{
    transaction.Rollback();
    // throw;
}
finally
{
    if (conn != null) conn.Close();
}
}
}

/// <summary>
/// Удаляет документ
/// </summary>
/// <param name="queries"></param>
/// <param name="param"></param>
/// <param name="del"></param>
public bool DeleteDocument(Hashtable queries, Hashtable param, List<long> lstDel, bool del)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;
    Hashtable parDoc = null;
    long lastHID = 0;
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            transaction = conn.BeginTransaction();
            //Помечаем документ на удаление
            if (!del)
            {
                //4. Получаем ID удаляемых документов

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

foreach (long id in lstDel)
{
    //1. Инсерт в History удаляемого документа
    string strInsertHistory = (string)queries["InsertHistory"];
    using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
    {
        commInsertHistory.Transaction = transaction;

        Hashtable parHis = (Hashtable)param["History"];

        foreach (DictionaryEntry de in parHis)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };
            if (par.ParameterName == ":_objectid")
                par.Value = id;

            commInsertHistory.Parameters.Add(par);
        }
        commInsertHistory.ExecuteNonQuery();
    }
    //3. Получаем последний HID
    string strLastHistory = (string)queries["LastHistory"];
    using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
    {
        parDoc = (Hashtable)param["Document"];
        commLastHistory.Transaction = transaction;

        var res = commLastHistory.ExecuteScalar();

        if (res != null)
            lastHID = (long)res;
    }
    //4. Update документа
    string strUpdateDoc = (string)queries["UpdateDocument"];
    using (NpgsqlCommand commUpdateDocument = new NpgsqlCommand(strUpdateDoc,
conn))
    {
        commUpdateDocument.Transaction = transaction;

        foreach (DictionaryEntry de in parDoc)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };
            if (par.ParameterName == ":_remhid")
                par.Value = lastHID;
            if (par.ParameterName == ":_id")
                par.Value = id;

            commUpdateDocument.Parameters.Add(par);
        }
        commUpdateDocument.ExecuteNonQuery();
    }
}
//Удаляем документ физически
else
{
    //4. Чистим историю, связанную с файлами

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string strDelHis = (string) queries["DelHis"];

using (NpgsqlCommand commDellHis = new NpgsqlCommand(strDelHis, conn))
{
    Hashtable parHis = (Hashtable) param["DelFilesHis"];
    commDellHis.Transaction = transaction;
    if (param.Contains("DelFilesHis"))
    {
        foreach (DictionaryEntry de in parHis)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            commDellHis.Parameters.Add(par);
        }
        commDellHis.ExecuteNonQuery();
    }

    //3. Чистим историю, связанную с документом
    commDellHis.Parameters.Clear();
    parHis = (Hashtable) param["DelDocHis"];
    foreach (DictionaryEntry de in parHis)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };

        commDellHis.Parameters.Add(par);
    }
    commDellHis.ExecuteNonQuery();
}

//2. Удаляем все записи о файлах
if (queries.Contains("DelFiles"))
{
    string strDelFiles = (string) queries["DelFiles"];
    using (NpgsqlCommand commDellFiles = new NpgsqlCommand(strDelFiles, conn))
    {
        Hashtable parDelFiles = (Hashtable) param["DelFiles"];
        commDellFiles.Transaction = transaction;

        foreach (DictionaryEntry de in parDelFiles)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            commDellFiles.Parameters.Add(par);
        }
        commDellFiles.ExecuteNonQuery();
    }
}

//4. Удаляем разрешения
string strDelAccess = (string) queries["DelAccess"];
using (NpgsqlCommand commDellAccess = new NpgsqlCommand(strDelAccess, conn))
{
    Hashtable parDelAccess = (Hashtable) param["DelAccess"];

```

					Лист
					280
Изм.	Лист	№ докум.	Подпись	Дат	

```

        commDellAccess.Transaction = transaction;

        foreach (DictionaryEntry de in parDelAccess)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            commDellAccess.Parameters.Add(par);
        }
        commDellAccess.ExecuteNonQuery();
    }

    //Удаляем подписи
    string strDelSign = (string)queries["DelSign"];
    using (NpgsqlCommand commDellSign = new NpgsqlCommand(strDelSign, conn))
    {
        Hashtable parDelSign = (Hashtable)param["DelSign"];
        commDellSign.Transaction = transaction;

        foreach (DictionaryEntry de in parDelSign)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            commDellSign.Parameters.Add(par);
        }
        commDellSign.ExecuteNonQuery();
    }

    //5. Удаляем документ
    string strDelDoc = (string)queries["DelDoc"];
    using (NpgsqlCommand commDelIDoc = new NpgsqlCommand(strDelDoc, conn))
    {
        Hashtable parDelDoc = (Hashtable)param["DelDoc"];
        commDelIDoc.Transaction = transaction;

        foreach (DictionaryEntry de in parDelDoc)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            commDelIDoc.Parameters.Add(par);
        }
        commDelIDoc.ExecuteNonQuery();
    }
}

transaction.Commit();
return true;
}
catch (Exception ex)
{
    transaction.Rollback();
    return false;
    // throw;
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        finally
        {
            if (conn != null) conn.Close();
        }
    }
}

/// <summary>
/// Восстанавливает документ из удаленных
/// </summary>
/// <param name="queries"></param>
/// <param name="pardoc"></param>
/// <param name="parhis"></param>
public void RestoreDocument(Hashtable queries, List<Hashtable> pardoc, List<Hashtable> parhis)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;

    long lastHID = 0;
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            transaction = conn.BeginTransaction();
            string strInsertDoc = (string) queries["InsertDocument"];
            string strInsertHistory = (string) queries["InsertHistory"];
            string strLastHistory = (string) queries["LastHistory"];

            for (int i = 0; i < parhis.Count; i++)
            {
                //Пишем историю
                using (NpgsqlCommand commInsHis = new NpgsqlCommand(strInsertHistory, conn))
                {
                    commInsHis.Transaction = transaction;
                    foreach (DictionaryEntry de in parhis[i])
                    {
                        {
                            var par = new NpgsqlParameter
                            {
                                ParameterName = de.Key.ToString(),
                                Value = de.Value
                            };

                            commInsHis.Parameters.Add(par);
                        }
                    }
                    commInsHis.ExecuteNonQuery();
                }
                //Получаем последний HID
                using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
                {
                    commLastHistory.Transaction = transaction;

                    var res = commLastHistory.ExecuteScalar();

                    if (res != null)
                        lastHID = (long)res;
                }
                //Восстанавливаем документ
                using (NpgsqlCommand commInsDoc = new NpgsqlCommand(strInsertDoc, conn))
                {
                    commInsDoc.Transaction = transaction;
                    foreach (DictionaryEntry de in pardoc[i])
                    {
                        {
                            var par = new NpgsqlParameter
                            {
                                ParameterName = de.Key.ToString(),
                                Value = de.Value
                            };

                            commInsDoc.Parameters.Add(par);
                        }
                    }
                    commInsDoc.ExecuteNonQuery();
                }
            }
        }
        catch { }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        ParameterName = de.Key.ToString(),
        Value = de.Value
    };
    if (par.ParameterName == ":_currhid")
        par.Value = lastHID;

    commInsDoc.Parameters.Add(par);
}
commInsDoc.ExecuteNonQuery();
}

}

transaction.Commit();
}
catch (Exception e)
{
    transaction.Rollback();
}
finally
{
    conn.Close();
}
}
}

/// <summary>
/// Добавить файл в хранилище
/// </summary>
/// <param name="querys">Запросы</param>
/// <param name="param">Параметры</param>
public void InsertFile(Hashtable querys, Hashtable param)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;
    Hashtable parFile = null;
    long lastFileId = 0;
    long lastHID = 0;
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            transaction = conn.BeginTransaction();
            //1. Делаем update документа(editdate, editauthor)
            string strUpdDocument = (string)querys["UpdateDocument"];

            using (NpgsqlCommand commUpdDocument = new NpgsqlCommand(strUpdDocument, conn))
            {
                commUpdDocument.Transaction = transaction;

                var parDoc = (Hashtable)param["Document"];

                foreach (DictionaryEntry de in parDoc)
                {
                    var par = new NpgsqlParameter
                    {
                        ParameterName = de.Key.ToString(),
                        Value = de.Value
                    };
                    commUpdDocument.Parameters.Add(par);
                }
                commUpdDocument.ExecuteNonQuery();
            }
        }
        //2. Получаем последний ID файла
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string strLastFile = (string)query["LastFile"];
using (NpgsqlCommand commLastFile = new NpgsqlCommand(strLastFile, conn))
{
    commLastFile.Transaction = transaction;

    var res = commLastFile.ExecuteScalar();

    if (res != null)
        lastFileId = (long)res;
}
//3. Инсерт в History
string strInsertHistory = (string)query["InsertHistory"];
using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
{
    commInsertHistory.Transaction = transaction;

    Hashtable parHis = (Hashtable)param["History"];

    foreach (DictionaryEntry de in parHis)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_objectid" && (long)par.Value == 0)
            par.Value = lastFileId + 1;

        commInsertHistory.Parameters.Add(par);
    }
    commInsertHistory.ExecuteNonQuery();
}
//4. Получаем последний HID
string strLastHistory = (string)query["LastHistory"];
using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
{
    commLastHistory.Transaction = transaction;

    var res = commLastHistory.ExecuteScalar();

    if (res != null)
        lastHID = (long)res;
}
//5. Insert в Files
string strInsertFile = (string)query["InsertFile"];
using (NpgsqlCommand commInsertFile = new NpgsqlCommand(strInsertFile, conn))
{
    commInsertFile.Transaction = transaction;

    parFile = (Hashtable)param["File"];

    foreach (DictionaryEntry de in parFile)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_id" && (long)par.Value == 0)
            par.Value = lastFileId + 1;
        if (par.ParameterName == ":_currhid")
            par.Value = lastHID;

        commInsertFile.Parameters.Add(par);
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        }
        commInsertFile.ExecuteNonQuery();
    }

    transaction.Commit();
}
catch (Exception ex)
{
    transaction.Rollback();
}
finally
{
    conn.Close();
}
}
}
}
/// <summary>
/// Изменяет запись о файле в базе
/// </summary>
/// <param name="querys"></param>
/// <param name="param"></param>
public void UpdateFile(Hashtable querys, Hashtable param)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;
    Hashtable parFile = null;
    long lastHID = 0;
    int lastVersion = 0;
    using (NpgsqlConnection conn = new NpgsqlConnection(connString))
    {
        try
        {
            conn.Open();
            transaction = conn.BeginTransaction();
            if (querys.Contains("UpdateDocument"))
            {
                //1. Делаем update документа(editdate, editauthor)
                string strUpdDocument = (string)querys["UpdateDocument"];

                using (NpgsqlCommand commUpdDocument = new NpgsqlCommand(strUpdDocument, conn))
                {
                    commUpdDocument.Transaction = transaction;

                    var parDoc = (Hashtable)param["Document"];

                    foreach (DictionaryEntry de in parDoc)
                    {
                        var par = new NpgsqlParameter
                        {
                            ParameterName = de.Key.ToString(),
                            Value = de.Value
                        };
                        commUpdDocument.Parameters.Add(par);
                    }
                    commUpdDocument.ExecuteNonQuery();
                }
            }

            //2. Insert в History
            string strInsertHistory = (string)querys["InsertHistory"];
            using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
            {
                commInsertHistory.Transaction = transaction;

                Hashtable parHis = (Hashtable)param["History"];
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

foreach (DictionaryEntry de in parHis)
{
    var par = new NpgsqlParameter
    {
        ParameterName = de.Key.ToString(),
        Value = de.Value
    };

    commInsertHistory.Parameters.Add(par);
}
commInsertHistory.ExecuteNonQuery();
}
//3. Получаем последний HID
string strLastHistory = (string)query["LastHistory"];
using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
{
    commLastHistory.Transaction = transaction;

    var res = commLastHistory.ExecuteScalar();

    if (res != null)
        lastHID = (long)res;
}
//4. Update старого файла в remhid пишем последний HID, в имя файла
//то, что вернулось после сохранения
string strUpdateFile = (string)query["UpdateFile"];
using (NpgsqlCommand commUpdFile = new NpgsqlCommand(strUpdateFile, conn))
{
    commUpdFile.Transaction = transaction;

    Hashtable parUpdFile = (Hashtable)param["UpdFile"];

    foreach (DictionaryEntry de in parUpdFile)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_remhid")
            par.Value = lastHID;

        commUpdFile.Parameters.Add(par);
    }
    commUpdFile.ExecuteNonQuery();
}

//5. Определяем последнюю версию файла

string strLastVersion = (string)query["LastVersion"];
Hashtable parLastVersion = (Hashtable)param["LastVersion"];

using (NpgsqlCommand commLastVersion = new NpgsqlCommand(strLastVersion, conn))
{
    commLastVersion.Transaction = transaction;

    foreach (DictionaryEntry de in parLastVersion)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
    };
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        commLastVersion.Parameters.Add(par);
    }

    var res = commLastVersion.ExecuteScalar();
    if (res != null)
        lastVersion = (int)res;
}

if (querys.Contains("InsertFile"))
{
    //6. Создаём новую запись с id от старого, curhid = HID и remhid = 0
    string strInsertFile = (string)querys["InsertFile"];
    using (NpgsqlCommand commInsertFile = new NpgsqlCommand(strInsertFile, conn))
    {
        commInsertFile.Transaction = transaction;

        parFile = (Hashtable)param["File"];

        foreach (DictionaryEntry de in parFile)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };

            if (par.ParameterName == ":_currhid")
                par.Value = lastHID;
            if (par.ParameterName == ":_version")
                par.Value = lastVersion + 1;
            if (par.ParameterName == ":_fullpath")
                par.Value = ((string)parFile[":_fullpath"]).Replace("\\\" + parFile[":_version"].ToString()
                    , "\\\" + (lastVersion + 1).ToString());

            commInsertFile.Parameters.Add(par);
        }
        commInsertFile.ExecuteNonQuery();
    }
}

transaction.Commit();
}
catch (Exception ex)
{
    transaction.Rollback();
}
finally
{
    conn.Close();
}
}

}

/// <summary>
/// Заменяет текущую версию файла
/// </summary>
/// <param name="querys"></param>
/// <param name="param"></param>
public void SetFileCurrentVersion(Hashtable querys, Hashtable param)
{
    string connString = GetConnectionString();
    NpgsqlTransaction transaction = null;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

Hashtable parFile = null;
long lastFileId = 0;
long lastHID = 0;

using (NpgsqlConnection conn = new NpgsqlConnection(connString))
{
    try
    {
        conn.Open();
        transaction = conn.BeginTransaction();

        //1. Делаем update документа(editdate, editauthor)
        string strUpdDocument = (string)query["UpdateDocument"];

        using (NpgsqlCommand commUpdDocument = new NpgsqlCommand(strUpdDocument, conn))
        {
            commUpdDocument.Transaction = transaction;

            var parDoc = (Hashtable)param["Document"];

            foreach (DictionaryEntry de in parDoc)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),
                    Value = de.Value
                };
                commUpdDocument.Parameters.Add(par);
            }
            commUpdDocument.ExecuteNonQuery();
        }

        //2. Insert в History
        string strInsertHistory = (string)query["InsertHistory"];
        using (NpgsqlCommand commInsertHistory = new NpgsqlCommand(strInsertHistory, conn))
        {
            commInsertHistory.Transaction = transaction;

            Hashtable parHis = (Hashtable)param["History"];

            foreach (DictionaryEntry de in parHis)
            {
                var par = new NpgsqlParameter
                {
                    ParameterName = de.Key.ToString(),
                    Value = de.Value
                };

                commInsertHistory.Parameters.Add(par);
            }
            commInsertHistory.ExecuteNonQuery();
        }

        //3. Получаем последний HID
        string strLastHistory = (string)query["LastHistory"];
        using (NpgsqlCommand commLastHistory = new NpgsqlCommand(strLastHistory, conn))
        {
            commLastHistory.Transaction = transaction;

            var res = commLastHistory.ExecuteScalar();

            if (res != null)
                lastHID = (long)res;
        }

        if (query.Contains("UpdateFile"))

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

{
    //4. Update старого файла в remhid пишем последний HID, в имя файла
    //то, что вернулось после сохранения
    string strUpdateFile = (string)querys["UpdateFile"];
    using (NpgsqlCommand commUpdFile = new NpgsqlCommand(strUpdateFile, conn))
    {
        commUpdFile.Transaction = transaction;

        Hashtable parUpdFile = (Hashtable)param["UpdOldFile"];

        foreach (DictionaryEntry de in parUpdFile)
        {
            var par = new NpgsqlParameter
            {
                ParameterName = de.Key.ToString(),
                Value = de.Value
            };
            if (par.ParameterName == ":_remhid")
                par.Value = lastHID;

            commUpdFile.Parameters.Add(par);
        }
        commUpdFile.ExecuteNonQuery();
    }
}

//5. Insert новой строки с 0 RemHID

string strInsFile = (string)querys["InsertFile"];

Hashtable parInsNewFile = (Hashtable)param["InsNewFile"];

using (NpgsqlCommand commInsFile = new NpgsqlCommand(strInsFile, conn))
{
    commInsFile.Transaction = transaction;
    foreach (DictionaryEntry de in parInsNewFile)
    {
        var par = new NpgsqlParameter
        {
            ParameterName = de.Key.ToString(),
            Value = de.Value
        };
        if (par.ParameterName == ":_remhid")
            par.Value = 0;
        if (par.ParameterName == ":_currhid")
            par.Value = lastHID;

        commInsFile.Parameters.Add(par);
    }

    commInsFile.ExecuteNonQuery();
}

transaction.Commit();
}
catch (Exception ex)
{
    transaction.Rollback();
}
finally
{
    conn.Close();
}
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```
}
}
```

IDocService.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using Utils;

namespace Doc.Service
{
    /// <summary>
    /// Сервис работы с БД
    /// </summary>
    [ServiceContract(CallbackContract = typeof(IDocServiceCallback), SessionMode = SessionMode.Required)]
    public interface IDocService
    {
        /// <summary>
        /// Возвращает строку подключения к БД
        /// </summary>
        /// <returns>Строка подключения</returns>
        [OperationContract]
        string GetConnectionString();

        /// <summary>
        /// Создаёт новую базу данных
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        void CreateDB();

        /// <summary>
        /// Выполняет запрос к БД
        /// </summary>
        /// <param name="query">Запрос</param>
        /// <param name="param">Передаваемые параметры</param>
        /// <returns>Результат запроса</returns>
        [OperationContract]
        ArrayList ExecuteQuery(string query, Hashtable param);

        /// <summary>
        /// Выполняет вставку удаление, изменение записей в таблицах
        /// </summary>
        /// <param name="query">Запрос</param>
        /// <param name="lstparam">Передаваемые параметры</param>
        [OperationContract]
        void ExecuteNonQueryTr(string query, List<Hashtable> lstparam);

        /// <summary>
        /// Выполняет вставку удаление, изменение записей в таблицах
        /// </summary>
        /// <param name="query">Запрос</param>
        /// <param name="param">Передаваемые параметры</param>
        [OperationContract]
        void ExecuteNonQuery(string query, Hashtable param);

        /// <summary>
        /// Проверяет наличие БД
        /// </summary>
        /// <param name="name"></param>
```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <returns></returns>
[OperationContract]
bool IsDBExists(string name);
/// <summary>
/// Проверка подключения
/// </summary>
/// <returns></returns>
[OperationContract]
bool IsConnected();

/// <summary>
/// Добавляет сообщение для уведомления всех клиентов об изменениях
/// </summary>
/// <param name="message"></param>
[OperationContract]
void AddMessage(string message, bool isedit, bool delete);

/// <summary>
/// Подписывает клиента на уведомления
/// </summary>
/// <returns></returns>
[OperationContract]
bool Subscribe();

/// <summary>
/// Отписывает клиента от уведомлений
/// </summary>
/// <returns></returns>
[OperationContract]
bool Unsubscribe();

/// <summary>
/// Получает фотографию
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
[OperationContract]
Hashtable GetPhoto(int id);

/// <summary>
/// Добавить документ
/// </summary>
[OperationContract]
void InsertDocument(Hashtable querys, Hashtable param, List<Hashtable> parAdminAccess,
List<Hashtable> parFiles);
/// <summary>
/// Удалить документ
/// </summary>
[OperationContract]
bool DeleteDocument(Hashtable queryes, Hashtable param, List<long> IstDel, bool del);
/// <summary>
/// Добавить файл в хранилище
/// </summary>
/// <param name="querys">Запросы</param>
/// <param name="param">Параметры</param>
[OperationContract]
void InsertFile(Hashtable querys, Hashtable param);
/// <summary>
/// Изменяет запись о файле в базе
/// </summary>
/// <param name="querys"></param>
/// <param name="param"></param>
[OperationContract]
void UpdateFile(Hashtable querys, Hashtable param);
/// <summary>
/// Заменяет текущую версию файла

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    /// </summary>
    /// <param name="querys"></param>
    /// <param name="param"></param>
    [OperationContract]
    void SetFileCurrentVersion(Hashtable querys, Hashtable param);

    /// <summary>
    /// Восстанавливает документ из удаленных
    /// </summary>
    /// <param name="queryes"></param>
    /// <param name="pardoc"></param>
    /// <param name="parhis"></param>
    [OperationContract]
    void RestoreDocument(Hashtable queryes, List<Hashtable> pardoc, List<Hashtable> parhis);
}
}

```

IDocServiceCallback.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ServiceModel;

namespace Doc.Service
{
    [ServiceContract]
    interface IDocServiceCallback
    {
        [OperationContract(IsOneWay = true)]
        void OnMessageAdded(string message, bool isedit, bool delete);
    }
}

```

IFSService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Utils;
using System.ServiceModel;
namespace FS.Console
{
    [ServiceContract]
    interface IFSService
    {
        /// <summary>
        /// Запись файла в хранилище
        /// </summary>
        /// <param name="stream">Поток с клиента</param>
        /// <param name="filename">Имя файла в хранилище</param>
        /// <param name="oldfilename">Новое имя старого файла</param>
        /// <returns>Признак перезаписи</returns>
        [OperationContract]
        bool SaveFile(byte[] stream, string filename, ref string oldfilename);

        /// <summary>
        /// Получает байтовый поток из файла в хранилище
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

    /// </summary>
    /// <param name="filename">Имя файла в хранилище</param>
    /// <returns>Байтовый поток из файла в хранилище</returns>
    [OperationContract]
    byte[] GetFile(string filename);
    /// <summary>
    /// Проверяет соединение с сервером
    /// </summary>
    /// <param name="fullpath"></param>
    /// <returns></returns>
    [OperationContract]
    bool IsConnectFsStorage(string fullpath);
}
}

```

Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceProcess;
using System.Text;
using System.Threading.Tasks;

namespace FS.Host
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        static void Main()
        {
            ServiceBase[] ServicesToRun;
            ServicesToRun = new ServiceBase[]
            {
                new FSServiceHost()
            };
            ServiceBase.Run(ServicesToRun);
        }
    }
}

```

FSService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Collections;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization.Json;
using Utils;

namespace FS.Service
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode =
    ConcurrencyMode.Reentrant)]

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

public class FSService : IFSService
{
    /// <summary>
    /// Проверка подключения
    /// </summary>
    /// <returns></returns>

    public bool IsConnected()
    {
        return true;
    }

    /// <summary>
    /// Проверка наличия сервера и каталога хранилища
    /// </summary>
    /// <param name="fullpath"></param>
    /// <returns></returns>
    public bool IsConnectFsStorage(string fullpath)
    {
        try
        {
            if (!Directory.Exists(fullpath))
            {
                Directory.CreateDirectory(fullpath);
            }
            return true;
        }
        catch (Exception e)
        {
            return false;
        }
    }

    /// <summary>
    /// Запись файла в хранилище
    /// </summary>
    /// <param name="stream">Поток с клиента</param>
    /// <param name="filename">Имя файла в хранилище</param>
    /// <param name="oldfilename">Новое имя старого файла</param>
    /// <returns>Признак перезаписи</returns>
    public bool SaveFile(byte[] stream, string filename, string fullpath)
    {
        try
        {
            string dir = GetStoragePath() + "\\" + fullpath.Substring(0, fullpath.IndexOf("\\"));
            if (!Directory.Exists(dir))
                Directory.CreateDirectory(dir);

            //находим последнюю версию
            DirectoryInfo dirInfo = new DirectoryInfo(dir);
            DirectoryInfo[] childs = dirInfo.GetDirectories();
            int vers = -1;

            for (int i = 0; i < childs.Length; i++)
            {
                if (i == 0)
                    vers = Int32.Parse(childs[i].Name);
                else if (Int32.Parse(childs[i].Name) > Int32.Parse(childs[i - 1].Name))
                    vers = Int32.Parse(childs[i].Name);
            }
            string storagePath = dir + "\\" + (vers + 1).ToString();

            if (!IsConnectFsStorage(storagePath))
                return false;
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string path = storagePath + @"\" + filename;
//1. Найти в файловом хранилище файл с именем из параметров
//if (File.Exists(path))
//{
//    byte[] arr = null;
//    //2. Получить из него байтовый поток
//    arr = File.ReadAllBytes(path);
//    //3. сравнить с полученным потоком
//    //Если равны, ничего не делаем, возвращаем true

//    if (arr.SequenceEqual(stream))
//    {
//        return true;
//    }

//    //выделяем из имени файла расширение
//    string[] name = filename.Split('.');
//    //переименовываем файл
//    string newName = name[0] + "_" + DateTime.Now.ToString("ddMMyyHHmmss") +
//        "." + name[1];
//    //путь к хранилищу из конфига. Переименование файла с датой изменения, например
//    string newPath = storagePath + @"\" + newName;
//    File.Move(path, newPath); //Переименовываем файл

//    //И пишем новый из полученного потока по указанному пути и с указанным именем.
//    using (FileStream fstream2 = new FileStream(path, FileMode.CreateNew, FileAccess.Write))
//    {
//        fstream2.Write(stream, 0, stream.Length);
//    }

//    newfilename = newName;
//}
//else
//{
//    using (FileStream fs = new FileStream(path, FileMode.CreateNew, FileAccess.Write))
//    {
//        fs.Write(stream, 0, stream.Length);
//    }
//}
return true;
}
catch (Exception e)
{
    return false;
}
}

/// <summary>
/// Удаляет все версии файла из хранилища
/// </summary>
/// <param name="fullpath">Полная ссылка на файл</param>
/// <returns></returns>
public bool DeleteFile(string fullpath)
{
    try
    {
        string storagePath = GetStoragePath() + @"\" + fullpath;

        if (Directory.Exists(storagePath))
            Directory.Delete(storagePath, true);
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        return false;
    }
}

/// <summary>
/// Получает байтовый поток из файла в хранилище
/// </summary>
/// <param name="filename">Имя файла в хранилище</param>
/// <returns>Байтовый поток из файла в хранилище</returns>
public byte[] GetFile(string filename, string fullpath)
{
    string storagePath = GetStoragePath() + "\\ " + fullpath;

    if (!IsConnectFsStorage(storagePath))
        return null;
    //1. Найти в файловом хранилище файл с именем из параметров
    byte[] buf = null;
    string path;
    path = storagePath + @"\" + filename;
    if (File.Exists(path))
    {
        //2. Получить из него байтовый поток
        buf = File.ReadAllBytes(path);
    }

    return buf;
}

/// <summary>
/// Устанавливает выбранную версию текущей
/// </summary>
/// <param name="oldversion">Текущая версия</param>
/// <param name="newversion">Устанавливаемая версия</param>
public bool SetCurrentVersion(string oldversion, string newversion)
{
    string storagePath = GetStoragePath();

    if (!IsConnectFsStorage(storagePath))
        return false;

    //находим текущую версию
    string pathold = "";
    string newName = "";
    string pathnew = "";

    if (oldversion != "")
    {
        //переименовываем текущую версию
        pathold = storagePath + @"\" + oldversion; //без цифр
        pathnew = storagePath + @"\" + newversion; //с цифрами
        //выделяем из имени файла расширение
        string[] name = oldversion.Split('.');
        //переименовываем файл
        newName = name[0] + "_" + DateTime.Now.ToString("ddMMyyHHmmss") +
            "." + name[1];
    }
    else
    {
        pathold = storagePath + @"\" + newversion; //с цифрами
        string[] name = newversion.Split('.');
        newName = name[0].Substring(0, name[0].IndexOf('_')) + "." + name[1];
        pathnew = storagePath + @"\" + newName; //без цифр
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

string newpath = storagePath + @"\" + newName;
if (File.Exists(pathold))
{
    File.Move(pathold, newpath);

    if(oldversion != "")
    {
        //переименовываем устанавливаемую версию
        if (File.Exists(pathnew))
            File.Move(pathnew, pathold);
        else
            return false;
    }

    return true;
}
return false;
}

private string GetStoragePath()
{
    Process[] pr = Process.GetProcessesByName("FS.Host");
    Process pr1 = pr[0];
    string servConfigPath = pr1.MainModule.FileName.Replace(".exe", ".cfg");
    ConfigManager cfManager = new ConfigManager();
    ConfigFS cfg = cfManager.GetConfiguration<ConfigFS>(servConfigPath);
    return cfg.StoragePath;
}
}
}

```

IFSService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace FS.Service
{
    [ServiceContract (SessionMode = SessionMode.Required)]
    public interface IFSService
    {
        /// <summary>
        /// Проверка подключения
        /// </summary>
        /// <returns></returns>
        [OperationContract]
        bool IsConnected();
        /// <summary>
        /// Запись файла в хранилище
        /// </summary>
        /// <param name="stream">Поток с клиента</param>
        /// <param name="filename">Имя файла в хранилище</param>
        /// <param name="oldfilename">Новое имя старого файла</param>
        /// <returns>Признак перезаписи</returns>
        [OperationContract]
        bool SaveFile(byte[] stream, string filename, string fullpath);

        /// <summary>
        /// Получает байтовый поток из файла в хранилище
        /// </summary>
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <param name="filename">Имя файла в хранилище</param>
/// <returns>Байтовый поток из файла в хранилище</returns>
[OperationContract]
byte[] GetFile(string filename, string fullpath);

[OperationContract]
bool IsConnectFsStorage(string fullpath);
/// <summary>
/// Удаляет все версии файла из хранилища
/// </summary>
/// <param name="fullpath">Папка файла</param>
/// <returns></returns>
[OperationContract]
bool DeleteFile(string fullpath);

/// <summary>
/// Устанавливает выбранную версию текущей
/// </summary>
/// <param name="oldversion">Текущая версия</param>
/// <param name="newversion">Устанавливаемая версия</param>
[OperationContract]
bool SetCurrentVersion(string oldversion, string newversion);

}

}

```

ImageManager.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media.Imaging;

namespace Utils
{
    public static class ImageManager
    {
        public static BitmapImage GetBitmap(Stream stream)
        {
            var bitmap = new BitmapImage();
            bitmap.BeginInit();
            bitmap.StreamSource = stream;
            bitmap.CacheOption = BitmapCacheOption.OnLoad;
            bitmap.EndInit();
            bitmap.Freeze();

            return bitmap;
        }
    }
}

```

ConfigApp.cs

```

using System;
using System.Collections.Generic;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace Utils.Config
{
    [XmlRoot("ConfigApp", Namespace = "", IsNullable = false)]
    public class ConfigApp
    {
        /// <summary>
        /// Имя компьютера на котором запущена служба сервера БД.
        /// </summary>
        [XmlElement]
        public string HostDB;

        /// <summary>
        /// Номер порта для службы сервера БД.
        /// </summary>
        [XmlElement]
        public string DbPort;

        /// <summary>
        /// Имя компьютера на котором запущена служба файлового сервера.
        /// </summary>
        [XmlElement]
        public string HostFS;

        /// <summary>
        /// Номер порта для службы файлового сервера.
        /// </summary>
        [XmlElement]
        public string FSPort;

        /// <summary>
        /// Инициализирует новый экземпляр Config.
        /// </summary>
        /// <param name="">Строка подключения.</param>
        /// <param name="hostdb">Хост.</param>
        /// <param name="dbport">Порт доступа к БД.</param>
        /// <param name="hostfs">Имя компьютера на котором запущена служба файлового
сервера.</param>
        /// <param name="fsport">Номер порта для службы файлового сервера.</param>
        public ConfigApp(string hostdb, string dbport, string hostfs, string fsport)
        {
            HostDB = hostdb;
            DbPort = dbport;
            HostFS = hostfs;
            FSPort = fsport;
        }

        public ConfigApp()
        {
        }
    }
}

```

Config.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

using System.Threading.Tasks;
using System.Xml.Serialization;

namespace Utils
{
    /// <summary>
    /// Конфигурация службы файлового сервера БД
    /// </summary>
    [XmlRoot("ConfigFS", Namespace = "", IsNullable = false)]
    public class ConfigFS
    {
        /// <summary>
        /// Имя компьютера на котором запущена служба файлового сервера.
        /// </summary>
        [XmlElement]
        public string Host;

        /// <summary>
        /// Номер порта для службы файлового сервера.
        /// </summary>
        [XmlElement]
        public string FsPort;

        /// <summary>
        /// Путь к хранилищу файлов на сервере
        /// </summary>
        [XmlElement]
        public string StoragePath;

        /// <summary>
        /// Инициализирует новый экземпляр Config.
        /// </summary>
        /// <param name="host">Хост.</param>
        /// <param name="fsPort">Порт доступа к БД.</param>
        /// <param name="path">Путь к хранилищу файлов на сервере</param>
        public ConfigFS(string host, string fsPort, string path)
        {
            Host = host;
            FsPort = fsPort;
            StoragePath = path;
        }

        /// <summary>
        /// Инициализирует новый экземпляр Config.
        /// </summary>
        public ConfigFS()
        {
        }
    }
}

```

ConfigManager.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

using System.Net.Mime;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using System.Threading;

namespace Utils
{
    public enum AppType
    {
        DocHost,
        DocConsole,
        DocApp,
        FSHost,
        FSConsole,
    }

    public class ConfigManager
    {
        private static readonly object lockObj = new object();
        /// <summary>
        /// Конструктор класса ConfigManager
        /// </summary>
        public ConfigManager()
        {
        }

        /// <summary>
        /// Получает конфигурацию для сервера БД
        /// </summary>
        /// <returns>Конфигурация</returns>
        public T GetConfiguration<T>(string filename)
        {
            T cfg = default(T);

            string fileName = filename;
            lock (lockObj)
            {
                try
                {
                    //Пытаемся загрузить файл с диска и десериализовать его
                    using (var fs = new FileStream(fileName, FileMode.Open, FileAccess.Read))
                    {
                        var xs = new XmlSerializer(typeof(T));
                        cfg = (T) xs.Deserialize(fs);
                    }
                }
                catch (Exception ex)
                {
                    Process curProc = Process.GetCurrentProcess();
                    String logFile = Logger.GetLogName(curProc.ProcessName, curProc.StartTime);

                    Logger.Log mes = new Logger.Log("Ошибка.", MessageType.Error, ex, DateTime.Now);
                    Logger.WriteLog(logFile, mes);
                }
            }
            return cfg;
        }
        /// <summary>
        /// Записывает конфигурацию сервера БД
        /// </summary>
        /// <param name="cfg"></param>
        public void SaveConfiguration<T>(string filename, T cfg)
        {

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

lock (lockObj)
{
    string fileName = filename;

    if(fileName == "")
    {
        Process proc = Process.GetCurrentProcess();
        fileName = proc.ProcessName + ".cfg";
    }
    try
    {
        using (var wStream = new FileStream(fileName, FileMode.Create))
        {
            XmlSerializer xs = new XmlSerializer(typeof(T));
            xs.Serialize(wStream, cfg);
        }
    }
    catch (Exception e)
    {
        Process curProc = Process.GetCurrentProcess();
        String logFile = Logger.GetLogName(curProc.ProcessName, curProc.StartTime);

        Logger.Log mes = new Logger.Log("Ошибка.", MessageType.Error, e, DateTime.Now);
        Logger.WriteLog(logFile, mes);
    }
}
}
}
}
}

```

ConfigSBD.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace Utils
{
    /// <summary>
    /// Конфигурация службы сервера БД
    /// </summary>
    [XmlRoot("ConfigSBD", Namespace = "", IsNullable = false)]
    public class ConfigSBD
    {
        /// <summary>
        /// Строка подключения к БД.
        /// </summary>
        [XmlElement]
        public string ConnectionString;

        /// <summary>
        /// Имя компьютера на котором запущена служба сервера БД.
        /// </summary>
        [XmlElement]
        public string Host;

        /// <summary>
        /// Номер порта для службы сервера БД.
        /// </summary>
        [XmlElement]
        public string DbPort;
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

/// <summary>
/// Инициализирует новый экземпляр Config.
/// </summary>
/// <param name="connectionString">Строка подключения.</param>
/// <param name="host">Хост.</param>
/// <param name="dbPort">Порт доступа к БД.</param>
public ConfigSBD(string connectionString, string host, string dbPort)
{
    ConnectionString = connectionString;
    Host = host;
    DbPort = dbPort;
}

public ConfigSBD()
{
}
}
}

```

Encrypt.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace Utils
{
    public static class Encrypt
    {
        public static string HashPassword(string password)
        {
            byte[] salt;
            byte[] buffer2;
            if (password == null)
            {
                throw new ArgumentNullException("password");
            }
            using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, 0x10, 0x3e8))
            {
                salt = bytes.Salt;
                buffer2 = bytes.GetBytes(0x20);
            }
            byte[] dst = new byte[0x31];
            Buffer.BlockCopy(salt, 0, dst, 1, 0x10);
            Buffer.BlockCopy(buffer2, 0, dst, 0x11, 0x20);
            return Convert.ToBase64String(dst);
        }

        public static bool VerifyHashedPassword(string hashedPassword, string password)
        {
            byte[] buffer4;
            if (hashedPassword == null)
            {
                return false;
            }
            if (password == null)
            {
                throw new ArgumentNullException("password");
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        byte[] src = Convert.FromBase64String(hashedPassword);
        if ((src.Length != 0x31) || (src[0] != 0))
        {
            return false;
        }
        byte[] dst = new byte[0x10];
        Buffer.BlockCopy(src, 1, dst, 0, 0x10);
        byte[] buffer3 = new byte[0x20];
        Buffer.BlockCopy(src, 0x11, buffer3, 0, 0x20);
        using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, dst, 0x3e8))
        {
            buffer4 = bytes.GetBytes(0x20);
        }
        return buffer3.SequenceEqual(buffer4);
    }
}

```

AccessType.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Utils
{
    public enum AccessType
    {
        Full = 1,
        ReadAndWrite,
        Read,
        Denied,
    }
}

```

HistoryObjectEnum.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Utils
{
    public enum AccessType
    {
        Full = 1,
        ReadAndWrite,
        Read,
        Denied,
    }
}

```

RoleEnums.cs

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Utils
{
    public enum RoleType
    {
        Administrator = 1,
        User,
    }

    public enum RoleAccess
    {
        EmployeesControl = 1,
        UserControl,
        DocControl,
        TypeControl,
        TypGroupControl,
        RouteControl,
        StageControl,
        RoleControl,
        AccessControl,
        AppointmentControl,
        MilitaryRankControl,
        SubdivisionControl,
        UnlockDocument,
        BrowseDeletedDocument,
        DelArchiveDocument,
    }
}
```

TypeGroupEnum.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Utils
{
    public enum TypeGroupsEnum
    {
        AccessType = 1,
        PassType,
        DocType,
        FileType,
        SignatureType,
    }
}
```

Logger.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

					ПГУ 1.090501.03.001 ПЗ	Лист
						280
Изм.	Лист	№ докум.	Подпись	Дат		

```

using System.Windows;

namespace Utils
{
    public enum MessageType
    {
        Message,
        Error,
    }

    public static class Logger
    {
        private static readonly object lockObj = new object();
        /// <summary>
        /// Запись лога
        /// </summary>
        public static void WriteLog(string filelog, Log log)
        {
            lock (lockObj)
            {
                try
                {
                    string fullPath = GetOrCreateFileFullName(filelog);

                    string str = "";
                    FileStream fs = File.Open(fullPath, FileMode.Append, FileAccess.Write, FileShare.None);

                    str = (log.MesType == MessageType.Error ? "Ошибка: \r\n" : "Сообщение: \r\n") + log.Message +
                        "\r\n" + "Дата события: " + DateTime.Now.ToString() + "\r\n" +
                        ExceptionToString(log.Ex)
                        + "\r\n!-----\r\n";

                    byte[] buff = Encoding.UTF8.GetBytes(str);

                    fs.Write(buff, 0, buff.Length);
                    fs.Close();
                }
                catch (Exception e)
                {
                    MessageBox.Show(e.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private static string ExceptionToString(Exception e)
        {
            StringBuilder result = new StringBuilder();
            if (e != null)
            {
                result.AppendFormat("Исключение: {0}\r\n", e.Message);
                result.AppendFormat("Тип исключения: {0}\r\n", e.GetType());
                result.AppendFormat("Приложение: {0}\r\n", e.Source + "(" +
                    Process.GetCurrentProcess().Id.ToString() + ")");
                result.AppendFormat("Метод: {0}\r\n", e.TargetSite);
                result.AppendFormat("Стек вызова: {0}\r\n", e.StackTrace);

                if (e.InnerException != null)
                {
                    result.Append("\r\n");
                    result.Append(ExceptionToString(e.InnerException));
                }
            }
            return result.ToString();
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

private static string GetOrCreateFileFullName(string filename)
{
    string fullPath = "";
    try
    {
        if (File.Exists(filename))
            fullPath = filename;
        else
        {
            string logDir = Directory.GetCurrentDirectory() + "\\LOG";
            if (!Directory.Exists(logDir))
                Directory.CreateDirectory(logDir);
            if (!File.Exists(logDir + "\\" + filename))
            {
                FileStream fs = File.Create(logDir + "\\" + filename);
                fs.Close();
            }
            fullPath = logDir + "\\" + filename;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка при создании лога.",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    return fullPath;
}

/// <summary>
/// Чтение лога
/// </summary>
/// <param name="logname"></param>
/// <param name="pos"></param>
/// <returns></returns>
public static Result ReadLog(string logname, long pos)
{
    List<Log> messageList = new List<Log>();
    StringBuilder str = new StringBuilder(""); //результатирующая строка
    long position = pos;
    Result res = null;
    string ret = "";
    string fullPath = GetOrCreateFileFullName(logname);
    //Только BinaryReader позволяет запоминать позицию
    //и при следующем открытии файла ставить указатель на нужную позицию.
    //Таким образом осуществляем построчное чтение файла и в то же время не держим его все
    время открытым
    BinaryReader reader = new BinaryReader(File.Open(fullPath, FileMode.Open));
    try
    {
        reader.BaseStream.Position = pos;
        while (position < reader.BaseStream.Length)
        {
            Char ch = reader.ReadChar();
            if (ch == '!')
            {
                while (ch != '\r')
                {
                    ch = reader.ReadChar();
                }
                position = reader.BaseStream.Position + 1;
                break;
            }
        }
    }
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

```

        str.Append(ch);

        position = reader.BaseStream.Position;

    }

    reader.Close();
    ret = str.Length > 0 ? str.ToString() : "Записей больше нет.";
    ret.IndexOf("Дата события: ");
}
catch (Exception e)
{
    MessageBox.Show(e.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

res = new Result(ret, position);

return res;
}

public static string GetLogName(string processname, DateTime starttime)
{
    string logname = processname + "-" + starttime.Year.ToString() +
        "-" + starttime.Month.ToString() + "-" + starttime.Day.ToString() +
        "-" + starttime.Hour.ToString() + "-" + starttime.Minute.ToString() +
        "-" + starttime.Second.ToString() + ".log";

    return logname;
}

public class Log
{
    public string Message;
    public MessageType MesType;
    public Exception Ex;
    public DateTime MesDateTime;

    public Log(string message, MessageType messType, Exception ex, DateTime dt)
    {
        Message = message;
        MesType = messType;
        Ex = ex;
        MesDateTime = dt;
    }
}

public class Result
{
    public string Str;
    public long Pos;
    public Result(string str, long pos)
    {
        Str = str;
        Pos = pos;
    }
};
}
}

```

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

					ПГУ 1.090501.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дат		280

