

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства совместного
редактирования документов**

Студент

(подпись, дата)

Зеляпугин Иван Алексеевич
(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026 г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой

(протокол заседания ГЭК от №)

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Зеляпугин Иван Алексеевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Программные средства совместного редактирования документов

- Тема утверждена приказом ПГУ № 1583/о от "30" декабря 2025 г.
3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.
4. Задание на работу (назначение разработки, исходные данные и т.п.) _____
- 1) Назначение: программные средства совместного редактирования документов.
- 2) Среда выполнения: Windows 10.
- Языковые средства разработки: C#, SQL, TypeScript, UML.
- Инструментальные средства: Visual Studio 2022, Docker, Visual Studio Code, Visual Paradigm.
- Технологии разработки: ООП, принципы SOLID, сервис-ориентированная архитектура.
- 3) Основные функции приложения:
- а) регистрация и авторизация;
- б) добавление, изменение, удаление документов;
- в) добавление участников в документ;
- г) выбор документа из списка;
- д) редактирование профиля пользователя;
- е) добавление, изменение, удаление блоков документа;
- ж) экспорт документа.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 113 страниц, 32 рисунка, 24 таблицы, 20 источников, 8 листингов и 7 приложений.

СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА, БАЗА ДАННЫХ, ТЕКСТОВЫЙ РЕДАКТОР, UML, WEB-ПРИЛОЖЕНИЕ, POSTGRESQL, REDIS, NGINX, ASP.NET CORE, DOCKER, KAFKA, SIGNALR, REACT.

Объект исследования – программные средства для совместного редактирования документов, включающее серверную часть с распределенной архитектурой и клиентское веб приложение для взаимодействия с пользователем.

Цель выпускной квалификационной работы – спроектировать и разработать программные средства совместного редактирования документов, обеспечивающее контроль ролей пользователей, управление документами и взаимодействие между клиентскими устройствами и серверной инфраструктурой.

Технологии разработки – разработка программных средств для серверной части сервиса проводилась в среде разработки Microsoft Visual Studio 2022 на языке программирования C#. Конструирование базы данных проводилось в СУБД PostgreSQL. Развертывание приложения производится посредством контейнеризации в системе Docker.

Среда исполнения – Windows 10, веб-браузер.

Результат работы – в ходе выполнения выпускной квалификационной работы были спроектированы и разработаны программные средства для совместного редактирования документов. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08BP222.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства совместного редактирования документов. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Зеляпугин И.А.						
Пров.		Балашова И.Ю.					4	113
Реценз.						Группа 22ВП2		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и выявление требований на разработку программных средств совместного редактирования текстовых документов.....	9
1.1 Анализ предметной области программных средств	9
1.2 Анализ технологий решения задач синхронного редактирования.....	10
1.3 Анализ аналогичных программных средств.....	11
1.4 Постановка задачи на разработку	13
1.4.1 Цели и задачи программных средств	13
1.4.2 Классы пользователей сервиса	14
1.5 Анализ требований	15
1.5.1 Анализ функциональных требований	15
1.5.2 Анализ нефункциональных требований	21
1.6 Планирование разработки и оценка бюджета	22
2 Проектирование программных средств совместного редактирования текстовых документов	25
2.1 Высокоуровневое проектирование	25
2.2 Проектирование структур данных.....	28
2.2.1 Концептуальная модель базы данных.....	28
2.2.2 Логическая модель базы данных	30
2.3 Проектирование серверной части	31
2.3.1 Проектирование сервиса AuthService	31
2.3.2 Проектирование сервиса DocumentService.....	32
2.3.3 Проектирование сервиса BlockService	34
2.3.4 Проектирование сервиса EmailService	36
2.4 Проектирование клиентской части.....	37
3 Реализация программных средств совместного редактирования документов.....	41
3.3 Реализация базы данных.....	41

3.4	Реализация серверной части приложения.....	43
3.4.2	Реализация авторизации приложения	44
3.4.3	Реализация взаимодействия с Kafka.....	46
3.4.4	Реализация взаимодействия в реальном времени и маршрутизации через Nginx	48
3.5	Реализация клиентской части приложения.....	51
4	Контроль качества программного обеспечения	54
4.1	Выбор режима тестирования и метрик кода.....	54
4.2	Тестирование программных средств совместного редактирования документов	56
4.2.1	Функциональное тестирование.....	56
4.2.2	Модульное тестирование.....	58
4.2	Оценка качества кода	62
4.3.1	Метрики Холстеда.....	62
	Заключение	65
	Список используемых источников.....	67
	Приложение А (справочное) Глоссарий	70
	Приложение Б (рекомендуемое) Диаграммы классов.....	72
	Приложение В (обязательное) Листинг серверной части.....	76
	Приложение Г (обязательное) Листинг клиентской части	90
	Приложение Д (рекомендуемое) Графическая часть	98
	Приложение Е (рекомендуемое) Функциональное тестирование	102
	Приложение Ж (рекомендуемое) Листинг модульных тестов.....	107

Введение

В настоящее время текстовые документы стали неотъемлемой частью повседневной жизни, особенно в образовательной и профессиональной сферах. Разработка программной документации, научных статей, отчетов и других материалов все чаще осуществляется несколькими участниками одновременно. Современные приложения для совместного редактирования должны обеспечивать мгновенную синхронизацию изменений, поддержку структурированного контента, удобное распределение ролей и прав доступа, а также возможность эффективного взаимодействия пользователей в реальном времени.

Актуальность темы совместного редактирования обусловлена активным распространением гибридных и полностью удаленных форм работы. Возможность одновременно вносить изменения в документы позволяет сократить время принятия решений, повысить продуктивность и улучшить качество результата.

Современные веб-приложения, реализующие совместную работу с данными в режиме реального времени, все чаще строятся на основе микросервисной архитектуры. Такой подход обеспечивает гибкость, независимость компонентов и возможность масштабирования системы, однако одновременно повышает требования к организации взаимодействия между бизнес-логикой приложения и хранилищем данных.

Для удобства сопровождения и снижения зависимости от конкретных технологий приложение проектируется с использованием многослойной архитектуры. Такой подход позволяет разделить ответственности между компонентами системы и изолировать бизнес-логику от деталей хранения данных.

Целью выпускной квалификационной работы является проектирование и разработка программных средств совместного редактирования документов,

которое позволит снизить время коллективной работы над документами за счет автоматической синхронизации изменений в реальном времени.

Для реализации поставленной цели были выявлены следующие задачи:

- проанализировать предметную область и существующие аналогичные программные продукты;
- спроектировать программные средства, а также структуры данных разрабатываемого приложения;
- разработать программные средства;
- осуществить контроль качества программного обеспечения.

Успешное решение поставленных задач позволит создать эффективный инструмент для расчета ипотечных условий.

1 Анализ предметной области и выявление требований на разработку программных средств совместного редактирования текстовых документов

1.1 Анализ предметной области программных средств

Программные средства совместного редактирования текстовых документов представляют собой программные решения, предназначенные для создания, редактирования, хранения и управления текстовыми документами с поддержкой совместной работы пользователей в реальном времени. Цель работы – предоставить удобный, безопасный и гибкий инструмент для индивидуальной и групповой работы с текстовой информацией.

Ниже представлены основные понятия предметной области.

Совместное редактирование – процесс одновременной работы нескольких пользователей над одним и тем же текстовым документом с обеспечением синхронного и асинхронного обновления.

Документ – структурированный набор текстовых и графических блоков, которые могут редактироваться участниками.

Блок документа – элемент документа, содержащий текст или изображение, которое редактируется независимо от других блоков.

Пользователь – человек, взаимодействующий с системой.

Роль – набор прав, определяющий возможности участника документа.

Защита данных реализуется через сервис авторизации, обеспечивающий регистрацию, вход в систему и подтверждение электронной почты через сервис сервиса рассылки. После успешного входа пользователь получает доступ к совместному редактированию файлов.

Документооборот реализуется преимущественно в электронной форме. Основой является текстовый документ, содержащий блоки и метаданные (название, создатель, перечень участников с ролями). Бумажные документы рассматриваются как производные.

1.2 Анализ технологий решения задач синхронного редактирования

В современных системах совместного редактирования одной из ключевых задач является обеспечение своевременной и корректной синхронизации изменений между участниками. Для достижения этой цели в программном средстве используются REST-интерфейсы в сочетании с WebSocket соединениями, что позволяет эффективно сочетать преимущества обеих технологий [1].

REST представляет собой архитектурный стиль, ориентированный на обмен данными посредством HTTP-запросов. В контексте данной работы он используется для получения начального состояния документов, а также для обработки отдельных операций изменения, которые не требуют мгновенной синхронизации с другими участниками.

Однако при использовании REST возникает ограничение. Каждый клиент должен периодически опрашивать сервер для получения актуального состояния документа. Что приводит к задержкам, увеличивает нагрузку сеть, а также может приводить к несогласованности данных при одновременной работе нескольких пользователей. Для решения этой проблемы программные средства используют WebSocket, который после рукопожатия устанавливает постоянное двустороннее соединение. Оно открывается на долгое время и позволяет передавать множество сообщений в обоих направлениях [2].

Обмен данными и управление состоянием документа в разрабатываемом программном средстве осуществляется через событийную модель. При внесении изменений в документ сервер обновляет данные в базы данных и транслирует соответствующее сообщение всем пользователям в рамках текущей группы. Это позволяет мгновенно обновлять графический интерфейс без необходимости повторно запроса страниц.

Таким образом можно сказать, что совместное использование описанных ранее технологий является хорошей практикой, так как REST используется для

редких или тяжелых операций, таких как обновление профиля пользователя или получение всех документов. А WebSocket в свою очередь берет на себя обновление и синхронизацию данных при редактировании внутреннего содержания документов.

1.3 Анализ аналогичных программных средств

Проведя анализ схожих по функционалу приложений, были выделены популярные платформы, представление в таблице 1.

Таблица 1 – Краткое описание общих сведений об аналогах

Наименование	Производитель	Описание
Google Docs	Google LLC	Облачный редактор документов с развитой системой совместного редактирования, поддержкой комментариев, истории изменений и синхронизацией в реальном времени
Notion	Notion Labs	Многофункциональная платформа для совместной работы, включая редактор документов, базы данных, систему управления задачами и рабочим пространством
Microsoft Office Online	Microsoft Corporation	Облачная версия офисного пакета, поддерживающая одновременную работу нескольких пользователей в Word, Excel и других приложениях
Quip	Salesforce	Платформа для корпоративной совместной работы, объединяющая документы, таблицы, чаты и встроенную систему обсуждений

Каждая из перечисленных платформ обладает широким функционалом и ориентирована на разные категории пользователей. Далее приведены их ключевые преимущества и недостатки.

Google Docs является наиболее известным решением в области облачного редактирования документов.

К основным плюсам можно отнести:

- высокая скорость синхронизации и стабильность работы даже при большом количестве участников;
- подробная история изменений и развитая система комментариев;

- возможность совместного редактирования без установки программного обеспечения.

К минусам можно отнести:

- отсутствие блочного представления документов, усложняющее работу над структурированными материалами.

Notion представляет универсальную систему управления информацией с акцентом на многофункциональность.

К основным плюсам можно отнести:

- развитая система блоков, позволяющих хранить сложные структурированные документы;

- гибкая настройка интерфейса под индивидуальные сценарии.

К минусам можно отнести:

- задержка синхронизации при высокой нагрузке;
- сложность интерфейса при работе с большими документами.

Office Online предлагает облачные версии классических офисных приложений.

К основным плюсам Office Online можно отнести:

- знакомый большинству пользователей интерфейс;
- совместимость с локальным Office.

Quip ориентирован на корпоративные команды и глубокую интеграцию в экосистему Salesforce.

К основным плюсам относятся:

- сочетание документов, таблиц и чатов в едином рабочем пространстве;

- стабильная работа в больших рабочих группах.

К минусам относятся:

- ориентированность на корпоративный сегмент и высокая стоимость;
- ограниченная гибкость структуры документов.

Для проведения сравнительного анализа разрабатываемых программных средств с существующими аналогами были выбраны ключевые функциональные и экономические критерии. Результаты сравнения приведены в таблице 2.

Таблица 2 – Сравнительный анализ аналогичных программных средств

Критерий	Google Docs	Notion	Office Online	Quip
Стоимость использования	Бесплатно (есть корпоративные тарифы)	Бесплатно (платная подписка для корпоративных решений)	Бесплатно версия с ограничениями	Только платные тарифы
Организация данных	Стандартная структура документа	Блочная структура данных	Стандартная структура документа	Документы со встроенными таблицами и чатами
Синхронизация	Мгновенная	С небольшой задержкой при совместной работе	Мгновенная	Высокая скорость
Права доступа	Гибкие (просмотр, редактирование, комментирование)	Продвинутые (доступ к отдельным блокам)	Стандартные (чтение и редактирование)	Командные (на уровне папок и чатов)
Интерфейс	Классический, интуитивно понятный	Требует время на освоение	Классический, привычный пользователям MS Office	Ориентирован на командные чаты и обсуждения
Нагрузка на систему	Низкая	Средняя	Средняя	Низкая

Подводя итог, можно сказать, что доступные на рынке программные средства предлагают близкий набор опций. Однако ни один из готовых продуктов не способен закрыть весь спектр требований, предъявленных к сервису. Поэтому данное обстоятельство говорит об необходимости создания своего решения, реализующего следующие требования:

- удобную работу с документом, разбитым на независимые блоки;
- высокую скорость доставки изменений благодаря WebSocket;
- удобный интерфейс и устойчивую работу на широком спектре устройств.

1.4 Постановка задачи на разработку

1.4.1 Цели и задачи программных средств

Основной целью работы является разработка программного обеспечения для совместного редактирования документов, включающего серверную и

клиентскую часть и ориентированную на работу в условиях распределенных команд и студенческих групп.

В процессе разработки нужно решить следующие задачи:

- обеспечить регистрацию и авторизацию;
- разработать механизм хранения документов, их структуры и содержания;
- реализовать блочную модель, позволяющую разбить документы на независимые фрагменты;
- обеспечить доставку изменений в реальном времени;
- организовать синхронизацию состояния документа между пользователями через события;
- реализовать операции создания, удаления и изменения документов и отдельных блоков;
- реализовать интерфейс, позволяющий удобно редактировать документ и визуально отображать блокировки;
- обеспечить взаимодействие между сервисами через Kafka.

Успешное достижение поставленных задач позволит создать приложение с интуитивно понятным интерфейсом и сквозной сервисной логикой. Взаимодействие между сервисами будет организовано через классические REST-запросы, постоянное WebSocket соединение и брокер сообщений Kafka.

1.4.2 Классы пользователей сервиса

В разрабатываемом приложении можно выделить три основных класса пользователей, представленных в таблице 3.

Таблица 3 – Классы пользователей приложения

№	Наименование класса	Бизнес-процессы	Используемые объекты
1	Наблюдатель	Просмотр документов в реальном времени	Документ, блоки
		Экспорт документа	Документ
2	Редактор	Создание новых блоков	Блоки
		Редактирование существующих блоков	Блоки

Продолжение таблицы 3

№	Наименование класса	Бизнес-процессы	Используемые объекты
3	Создатель	Удаление документов	Документ
		Управление настройками документа (управление ролями, переименование)	Документ, участники
		Управление составом участников (удаление, добавление)	Участники

Таким образом сформирована четкая ролевая модель, которая позволит права доступа к элементам системы. Ее реализация позволит обеспечить необходимый уровень безопасности данных.

1.5 Анализ требований

1.5.1 Анализ функциональных требований

Для описания функциональных возможностей приложения традиционно используется Use Case диаграмма [3].

Общая диаграмма вариантов использования представлена на рисунке 1.

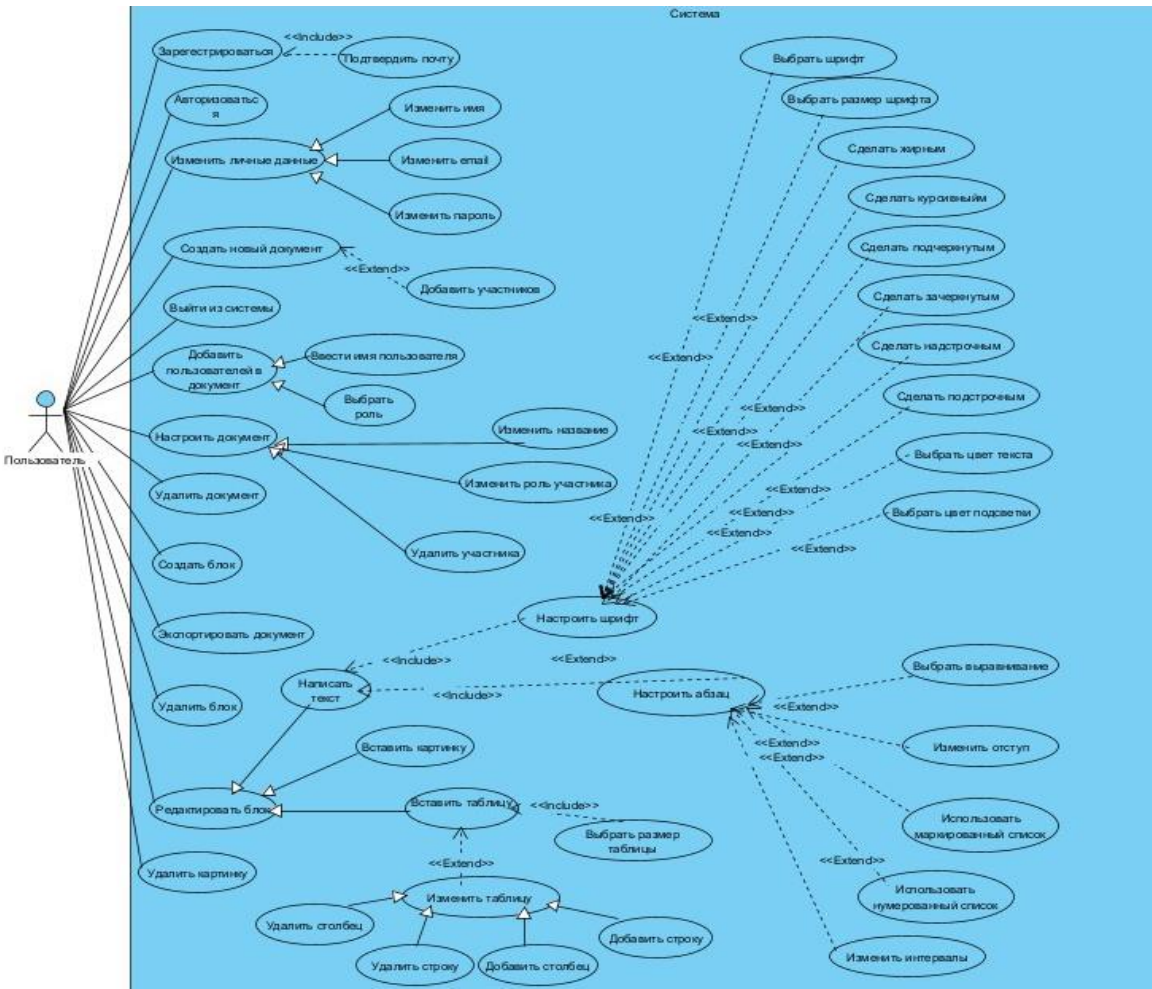


Рисунок 1 – Общая диаграмма вариантов использования

Как только заинтересованный в приложении человек регистрируется, он становится пользователем. После создания документа он становится его создателем и имеет полный контроль над ним и его участниками.

К основным вариантам использования можно отнести:

- зарегистрироваться;
- авторизоваться;
- изменить личные данные;
- создать новый документ;
- выйти из системы;
- добавить пользователя в документ;
- настроить документ;
- удалить документ;
- создать блок;
- экспортировать документ;
- удалить блок;
- редактировать блок;
- удалить картинку.

Сценарий варианта использования «Зарегистрироваться» представлен на рисунке 2.

Препедент: зарегистрироваться
Идентификатор препедента: 1
Краткое описание: пользователь вводит «Имя», «Email», «Пароль» и нажимает кнопку «Зарегистрироваться». Система создает учетную запись и отправляет письмо с кодом подтверждения на указанный email.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> 1. пользователь зашел на сайт.
Основной поток: <ol style="list-style-type: none"> 1. система предлагает пользователю ввести email, имя и пароль; 2. пользователь вводит данные; 3. система отправляет одноразовый код для подтверждения почты.
Постусловия: <ol style="list-style-type: none"> 1. данные сохраняются в базу данных.

Рисунок 2 – Сценарий варианта использования «Зарегистрироваться»

Сценарий варианта использования «Авторизоваться» представлен на рисунке 3.

Прецедент: авторизоваться
Идентификатор прецедента: 2
Краткое описание: пользователь вводит логин и пароль и отправляет форму входа. Система проверяет учетные данные и открывает главную страницу с документами, если аутентификация успешна.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> у пользователя уже ранее был создан аккаунт.
Основной поток: <ol style="list-style-type: none"> система предлагает пользователю ввести логин и пароль; пользователь вводит необходимые данные; система проверяет наличие пользователя.
Постусловия: <ol style="list-style-type: none"> пользователь вошел в систему

Рисунок 3 – Сценарий варианта использования «Авторизоваться»

Сценарий варианта использования «Изменить личные данные» представлен на рисунке 4.

Прецедент: изменить свои данные
Идентификатор прецедента: 3
Краткое описание: пользователь редактирует имя или пароль в настройках профиля и сохраняет изменения. Система обновляет данные в базе данных и отображает уведомление об успешном сохранении.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> пользователь авторизовался; пользователь зашел в профиль.
Основной поток: <ol style="list-style-type: none"> пользователь вводит новые данные (имя, пароль); система обновляет базу данных.
Постусловия: <ol style="list-style-type: none"> изменились данные пользователя.

Рисунок 4 – Сценарий варианта использования «Изменить свои данные»

Сценарий варианта использования «Создать документ» представлен на рисунке 5.

Прецедент: создать документ
Идентификатор прецедента: 4
Краткое описание: пользователь нажимает кнопку «Создать документ» на главной странице. Система создает пустой документ.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> пользователь авторизовался; пользователь решил создать документ.
Основной поток: <ol style="list-style-type: none"> пользователь вводит название документа; пользователь вводит имена и роли добавленных пользователей.
Постусловия: <ol style="list-style-type: none"> документ создан с указанными пользователями

Рисунок 5 – Сценарий варианта использования «Создать документ»

Сценарий варианта использования «Выйти из системы» представлен на рисунке 6.

Прецедент: выйти из системы
Идентификатор прецедента: 5
Краткое описание: пользователь нажимает кнопку «Выйти» в меню. Система завершает сессию и перенаправляет его на страницу входа.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> 1. пользователь авторизовался; 2. пользователь решил выйти.
Основной поток: <ol style="list-style-type: none"> 1. пользователь нажимает кнопку «Выйти»; 2. систем обнуляет токены доступа.
Постусловия: <ol style="list-style-type: none"> 1. токены обнулены; 2. пользователь на странице авторизации.

Рисунок 6 – Сценарий варианта использования «Выйти из системы»

Сценарий варианта использования «Добавить пользователя в документ» представлен на рисунке 7.

Прецедент: добавить пользователей в документ
Идентификатор прецедента: 6
Краткое описание: пользователь вводит имя участника и выбирает роль. Система добавляет пользователя в документ.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> 1. документ создан.
Основной поток: <ol style="list-style-type: none"> 1. администратор вводит имя и роль нового пользователя.
Постусловия: <ol style="list-style-type: none"> 1. новый участник добавлен.

Рисунок 7 – Сценарий варианта использования «Добавить пользователя в документ»

Сценарий варианта использования «Настроить документ» представлен на рисунке 8.

Прецедент: настроить документ
Идентификатор прецедента: 7
Краткое описание: пользователь изменяет название документа, а также количество участников и их роли в панели свойств и подтверждает изменения. Система сохраняет новое название и отображает его в заголовке документа.
Действующие лица: пользователь
Предусловия: <ol style="list-style-type: none"> 1. документ создан.
Основной поток: <ol style="list-style-type: none"> 1. администратор вводит новое имя документа, удаляет участников или меняет их роли; 2. система рассылает событие всем подписанным пользователям.
Постусловия: <ol style="list-style-type: none"> 1. параметры изменены.

Рисунок 8 – Сценарий варианта использования «Настроить документ»

Сценарий варианта использования «Удалить документ» представлен на рисунке 9.

Прецедент: удалить документ
Идентификатор прецедента: 8
Краткое описание: пользователь выбирает документ в списке и подтверждает его удаление. Система удаляет документ из хранилища и обновляет список доступных документов.
Действующие лица: пользователь
Предусловия:
1. документ создан.
Основной поток:
1. администратор выбирает документ и удаляет его;
2. система рассылает событие всем подписанным пользователям.
Постусловия:
1. документ удален.

Рисунок 9 – Сценарий варианта использования «Настроить документ»

Сценарий варианта использования «Создать блок» представлен на рисунке 10.

Прецедент: создать блок
Идентификатор прецедента: 9
Краткое описание: пользователь нажимает кнопку «Создать блок» на панели инструментов. Система вставляет новый текстовый блок в конец.
Действующие лица: пользователь
Предусловия:
1. документ создан.
Основной поток:
1. администратор или редактор создают новый блок;
2. система рассылает событие всем подписанным пользователям.
Постусловия:
1. блок создан.

Рисунок 10 – Сценарий варианта использования «Создать блок»

Сценарий варианта использования «Экспортировать документ» представлен на рисунке 11.

Прецедент: экспортировать документ
Идентификатор прецедента: 10
Краткое описание: пользователь экспортирует документ.
Действующие лица: пользователь
Предусловия:
1. документ создан.
Основной поток:
1. участник экспортирует документ себе на устройство в удобном для него формате.
Постусловия:
1. получена экспортированная версия.

Рисунок 11 – Сценарий варианта использования «Экспортировать документ»

Сценарий варианта использования «Удалить блок» представлен на рисунке 12.

Прецедент: удалить блок
Идентификатор прецедента: 11
Краткое описание: пользователь выбирает документ в списке и подтверждает его удаление. Система удаляет документ из хранилища и обновляет список доступных документов.
Действующие лица: пользователь
Предусловия: 1. документ создан.
Основной поток: 1. администратор или редактор удаляют блок; 2. система рассылает событие всем подписанным пользователям.
Постусловия: 1. блок удален.

Рисунок 12 – Сценарий варианта использования «Экспортировать документ»

Сценарий варианта использования «Редактировать блок» представлен на рисунке 13.

Прецедент: редактировать блок
Идентификатор прецедента: 12
Краткое описание: пользователь вводит или удаляет текст внутри существующего текстового блока. Система сохраняет изменения в реальном времени
Действующие лица: пользователь
Предусловия: 1. документ создан.
Основной поток: 1. администратор или редактор редактируют блок; 2. система рассылает событие всем подписанным пользователям.
Постусловия: 1. блок изменен.

Рисунок 13 – Сценарий варианта использования «Редактировать блок»

Сценарий варианта использования «Удалить картинку» представлен на рисунке 14.

Прецедент: удалить картинку
Идентификатор прецедента: 13
Краткое описание: пользователь кликает по изображению и выбирает «Удалить изображение». Система удаляет изображение, оставляя блок пустым
Действующие лица: пользователь
Предусловия: 1. документ создан.
Основной поток: 1. администратор или редактор удаляют картинку из блока; 2. система рассылает событие всем подписанным пользователям.
Постусловия: 1. картинка удалена.

Рисунок 14 – Сценарий варианта использования «Удалить картинку»

На основе сформированных функциональных критериев был определен перечень прецедентов, необходимых для полноценной реализации программных средств совместного редактирования документов. Выявленные сценарии отражают все ключевые действия будущих пользователей.

1.5.2 Анализ нефункциональных требований

Одним из наиболее важных этапов анализа нефункциональных требований является стал выбор архитектурного подхода. Выбранный паттерн должен гарантировать отказоустойчивость, надежность и высокую отзывчивость интерфейса [4]. Исходя из ключевых особенностей системы были сформулированы следующие требования:

- производительность;
- масштабируемость;
- доступность;
- надежность и безопасность;
- удобство и простота использования.

Такие требования были выведены на основе специфики приложения, так как придется обрабатывать большие массивы данных.

С учетом сформулированных критериев наиболее подходящей стала архитектура, основанная на сервисах. За счет высокой декомпозиции и автономности каждого программного модуля достигается необходимая гибкость и изолированность.

Ниже приведен анализ всех нефункциональных требований, обосновывающий выбранную архитектуру приложения.

Производительность – система должна обеспечивать минимальную задержку в передаче обновлений между пользователями [5].

Масштабируемость – приложение должно корректно работать при увеличении числа пользователей и открытых документов [5].

Доступность – система должна быть доступна всегда и прилагать все усилия для восстановления в случае сбоя [5].

Удобство и простота использования – приложение должно предоставлять удобный и понятный интерфейс, быть легким к изучению и защищать от ошибок пользователя [5].

Инструментальной основой для реализации указанных критериев выступает технологический стек от компании Microsoft. Программные средства совместного редактирования документов создаются на объектно – ориентированном языке C# в рамках платформы .NET. В роли среды разработки используется Visual Studio 2022.

Для хранения и структурирования данных применяется реляционная СУБД PostgreSQL. Взаимодействие между программным кодом и базой данных реализуется с помощью легковесной ORM Dapper. Использование этого инструмента позволяет выполнять SQL-запросы с минимальными расходами на маппинг данных, что существенно увеличивает пропускную способность системы.

Развертывание и последующая эксплуатация серверного компонента осуществляется в изолированной контейнерной среде на базе платформы Docker, под управлением операционной системы Linux. Веб-интерфейс стабильно отображается и функционирует в современных интернет-обозревателях, включая Google Chrom, Mozilla Firefox и Microsoft Edge.

1.6 Планирование разработки и оценка бюджета

Чтобы гарантировать успешное завершение проекта особое внимание уделяется бюджету и календарному планированию. Данный этап позволяет четко определить этапы и расставить приоритеты [6].

В целях эффективного мониторинга и оперативного управления ходом выполнения работ в качестве инструмента был выбран Microsoft Office Project. В данные программные средства были добавлены сведения обо всех ключевых этапах и вехах жизненного цикла создания приложения. Структура сформированных задач и их последовательность представлена на рисунке 15.

Название задачи	Длительность	Начало	Окончание
Начало проекта	0 дней	Вт 20.01.26	Вт 20.01.26
Анализ предметной области и выявление требований на разработку	12 дней	Вт 20.01.26	Ср 04.02.26
Анализ целевой аудитории	1 день	Вт 20.01.26	Вт 20.01.26
Анализ предметной области	6 дней	Ср 21.01.26	Ср 28.01.26
Анализ программных средств совместного редактирования документов	1 день	Чт 29.01.26	Чт 29.01.26
Анализ функциональных требований	1 день	Пт 30.01.26	Пт 30.01.26
Анализ нефункциональных требований	1 день	Пн 02.02.26	Пн 02.02.26
Анализ требований к интерфейсу	1 день	Вт 03.02.26	Вт 03.02.26
Планирование процесса разработки приложения и оценка затрат на разработку	1 день	Ср 04.02.26	Ср 04.02.26
Анализ требований завершен	0 дней	Ср 04.02.26	Ср 04.02.26
Проектирование программных средств совместного редактирования документов	20 дней	Чт 05.02.26	Ср 04.03.26
Проектирование структуры базы данных	5 дней	Чт 05.02.26	Ср 11.02.26
Проектирование архитектуры серверной части	6 дней	Чт 12.02.26	Чт 19.02.26
Проектирование архитектуры клиентской части	4 дней	Пт 20.02.26	Ср 25.02.26
Проектирование дизайна приложения	5 дней	Чт 26.02.26	Ср 04.03.26
Проектирование ПО завершенно	0 дней	Ср 04.03.26	Ср 04.03.26
Реализация программных средств совместного редактирования документов	45 дней	Чт 05.03.26	Ср 06.05.26
Реализация базы данных	4 дней	Чт 05.03.26	Вт 10.03.26
Реализация серверной части приложения	22 дней	Ср 11.03.26	Чт 09.04.26
Реализация клиентской части приложения	19 дней	Пт 10.04.26	Ср 06.05.26
Реализация завершенна	0 дней	Ср 06.05.26	Ср 06.05.26
Контроль качества программного обеспечения	7 дней	Чт 07.05.26	Пт 15.05.26
Выбор режима тестирования и метрик кода	1 день	Чт 07.05.26	Чт 07.05.26
Функциональное тестирование	1 день	Пт 08.05.26	Пт 08.05.26
Модульное тестирование	3 дней	Пн 11.05.26	Ср 13.05.26
Оценка качества кода	2 дней	Чт 14.05.26	Пт 15.05.26
Контроль качества программного обеспечения завершен	0 дней	Пт 15.05.26	Пт 15.05.26
Оформление отчета	9 дней	Пн 18.05.26	Чт 28.05.26
Оформление пояснительной записки	9 дней	Пн 18.05.26	Чт 28.05.26
Оформление отчета завершенно	0 дней	Чт 28.05.26	Чт 28.05.26
Конец проекта	0 дней	Вс 31.05.26	Вс 31.05.26

Рисунок 15 – Список задач проекта

Для более удобного представление перечня задач и их длительности в среде есть диаграмма Ганта. Подробная визуализация сформированного календарного плана представлена на рисунке 16 [7].

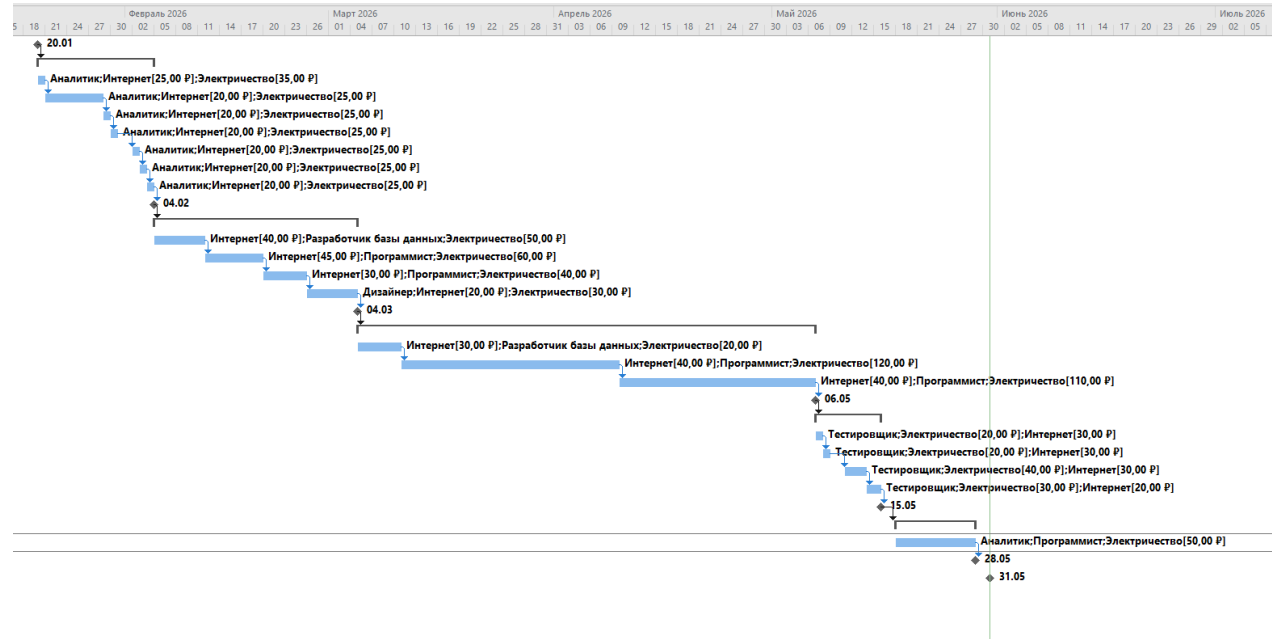


Рисунок 16 – Диаграмма Ганта

Из плана разработки можно увидеть, что наиболее продолжительным этапом является этап разработки программных средств, а завершение проекта планируется 31 мая 2026 года.

Для расчета бюджета проекта была определена стоимость каждого задействованного ресурса. Присвоенные базовые тарифные ставки отражены на рисунке 17.

Название ресурса	Тип	Краткое название	Стандартная ставка	Затраты на
Аналитик	Трудовой	А	300,00 Р/ч	0,00 Р
Программист	Трудовой	П	350,00 Р/ч	0,00 Р
Дизайнер	Трудовой	Д	300,00 Р/ч	0,00 Р
Разработчик базы данных	Трудовой	Р	250,00 Р/ч	0,00 Р
Тестировщик	Трудовой	Т	250,00 Р/ч	0,00 Р
Компьютер	Материальный	К	70 000,00 Р	0,00 Р
Электричество	Затраты	Э		
Интернет	Затраты	И		

Рисунок 17 – Ресурсы проекта

Таким образом общая длительность проекта составляет 94 дня. За дату начала разработки выбрано 20 января 2026 года. Стоимость реализации составляет 263 675 рублей. Сводка по проекту представлена на рисунке 18.

	Начало	Окончание
Текущее	Вт 20.01.26	Вс 31.05.26
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	Од	Од

	Длительность	Трудозатраты	Затраты
Текущие	94д	816ч	263 675,00 Р
Базовые	0д	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	94д	816ч	263 675,00 Р

Процент завершения

Длительность: 0% Трудозатраты: 0%

Заккрыть

Рисунок 18 – Сводка по проекту

Таким образом, на основе информации об используемых ресурсах и оценке бюджета проекта был сделан вывод о целесообразности реализации проекта по разработке программных средств совместного редактирования документов.

2 Проектирование программных средств совместного редактирования текстовых документов

2.1 Высокоуровневое проектирование

Проанализировав функциональные и нефункциональные требования в качестве целевого архитектурного подхода, была выбрана сервис-ориентированная архитектура. Это распределенная макроуровневая структура, состоящая из отдельно развернутого пользовательского интерфейса, отдельно развернутых крупномодульных сервисов и монолитной базы данных. Единая база данных сервисам применять SQL-запросы точно так же, как и в монолитной многоуровневой архитектуре, что облегчает разработку и уменьшает обмен сообщениями между сервисами.

Из-за крупномодульности предметных областей обычные ACID-транзакции базы данных выполняются в рамках одного сервиса. С другой стороны, архитектура с высоким уровнем распределенности, такая как микросервисная, обладает более высокой степенью детализации сервисов и использует технологию распределенной транзакции, которая полагается на окончательную согласованность, поэтому в ней не поддерживается такой же сохранности базы данных, как у ACID-транзакций [8].

С целью декомпозиции программного обеспечения на логические компоненты была разработана диаграмма компонентов, представленная на рисунке 19. Данная модель служит для визуализации внутренней структуры приложения и определения связей между его программными модулями. Методология построения данной схемы опирается на отображение ключевых компонентов и их интерфейсов межмодульного взаимодействия [9].

Отраженная на схеме структура наглядно демонстрирует слабую связанность модулей, реализованную через четко определенные контракты интерфейсов. Это позволяет тестировать и обновлять отдельные компоненты, не рискуя сломать всю систему.

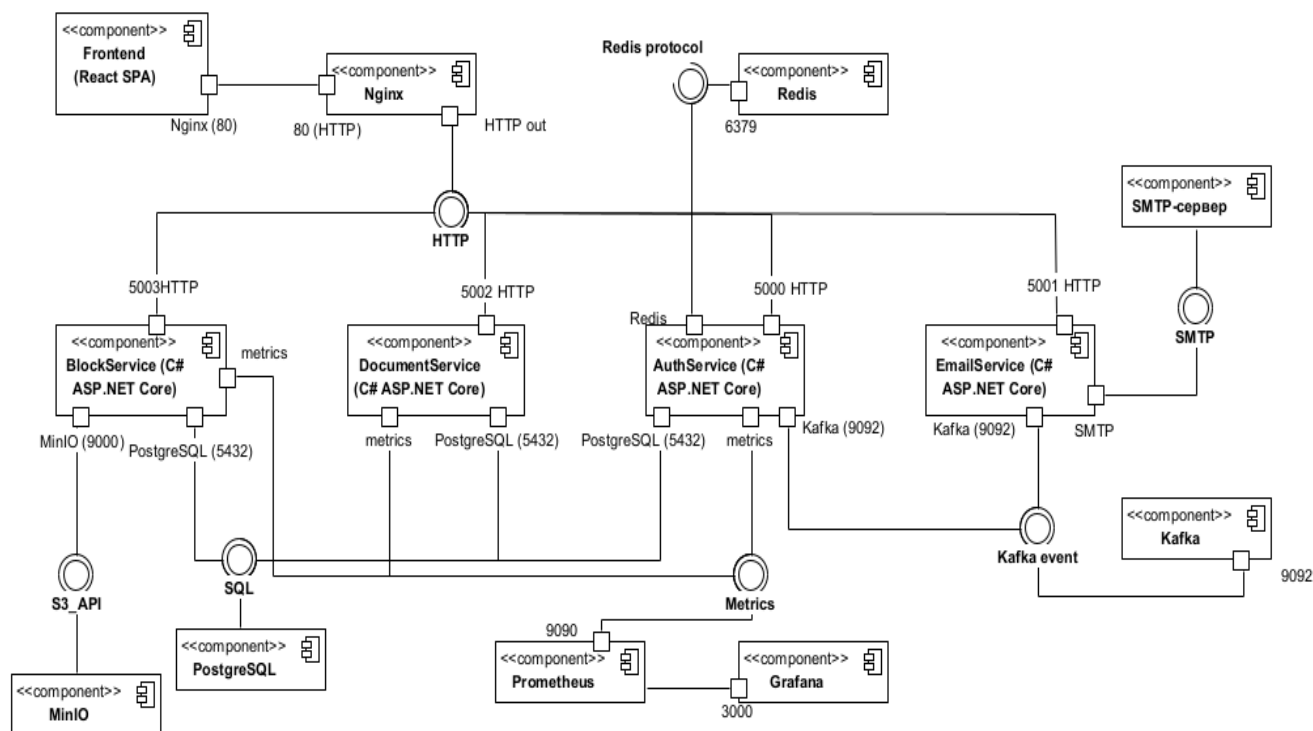


Рисунок 19 – Диаграмма компонентов

Модули, представленные на диаграмме, функционируют независимо друг от друга, выполняя задачи исключительно в рамках своей предметной области:

- AuthService – регистрация, аутентификация пользователей и выдача токенов доступа;
- DocumentService – управляет документами пользователей, хранит их метаданные, отвечает за создание, обновление и предоставление данных о документах, а также за управление участниками и их ролями;
- BlockService – обрабатывает содержимое документов, включая текстовые и графические блоки, выполняет загрузку, хранение и удаление изображений с использованием MinIO;
- EmailService – получает одноразовые коды подтверждения через Kafka и отправляет их пользователям по электронной почте;
- PostgreSQL – централизованное хранилище данных для сервисов авторизации, документов и блоков;

- Redis – хранилище одноразовых кодов, используемых сервисом авторизации;
- MiniO – хранилище, в котором сохраняются изображения, прикрепляемые к блокам документов;
- Kafka – брокер сообщений, через который сервис авторизации пересылает события о создании одноразовых кодов в сервис рассылки;
- Nginx – реверс-прокси и единая точка входа для клиентских запросов, обеспечивающий маршрутизацию на внутренние сервисы;
- Prometheus – собирает метрики со всех сервисов;
- Grafana – визуализирует данные, полученные из Prometheus, и используется для мониторинга состояния системы.

Подобная декомпозиция обеспечивает высокую адаптивность программного комплекса, существенно облегчая последующую модернизацию кода и позволяет осуществить горизонтальное масштабирование как по всей системе, так и ее отдельных компонентов.

Информационный обмен между внутренними сервисами организован по двум независимым каналам. Первый реализует классическую клиент-серверную модель на основе REST API, в то время как второй отвечает за доставку сообщений через распределенную шину данных Kafka.

REST API используется для синхронного взаимодействия между сервисами и клиентской частью. Apache Kafka применяется для асинхронной передачи событий. Например, при регистрации, сервис авторизации генерирует одноразовый код, сохраняет его в Redis и публикует сообщение в Kafka, откуда сервис рассылки его считывает и отправляет пользователю. Такой подход обеспечивает высокую производительность и отсутствие блокировок при выполнении длительных операций.

Nginx служит входной точкой для всех HTTP-запросов, обеспечивает маршрутизацию между сервисами и раздачу статических файлов клиентского приложения.

Для мониторинга используются Prometheus и Grafana. Первый собирает метрики путем периодического обращения к сервисам, второй отображает данные в виде графиков.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Переход к непосредственному созданию физической схемы базы данных требует построения промежуточного абстрактного уровня. В качестве такого базиса выступает концептуальная модель. Разрабатываемая на данном этапе модель абстрагирована от конкретной СУБД и фокусируется исключительно на отображении главных сущностей и логических связей, определяющих структуру приложения [10].

Спецификация первоначального набора сущностей представлена в таблице 4.

Таблица 4 – Описание сущностей

№	Наименование	Описание
1	Пользователь	Содержит сведения о зарегистрированных пользователях
2	Документ	Содержит сведения о документах пользователя
3	Участник	Содержит сведения об участниках документов и их роли
4	Блок	Содержит сведения о блоках документов
5	Изображение	Содержит сведения об изображениях документов
6	Роль	Содержит сведения о доступных ролях

На основе полученной информации была подготовлена концептуальная диаграмма в нотации «сущность – связь». Были определены основные типы связей между сущностями, а также их атрибуты. Для реализации связи много ко много была разработана сущность «Участник», которая соединяет пользователей с документами [11]. Внедрение данной промежуточной сущности обеспечивает возможность гибкого разграничения прав доступа и хранения данных о роли каждого пользователя.

Графическое представление концептуальной модели представлено на рисунке 20.

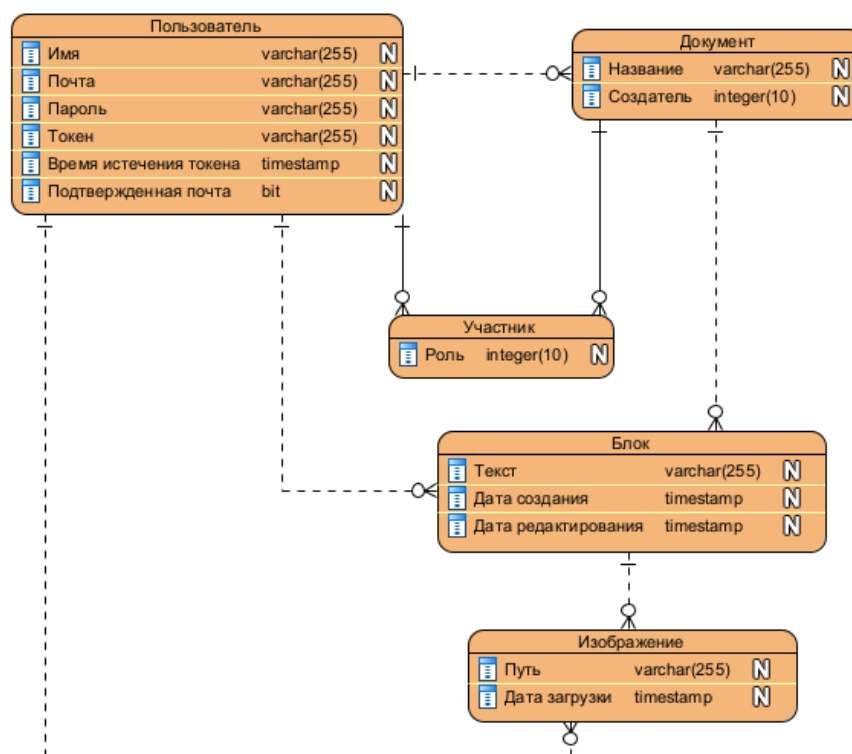


Рисунок 20 – Концептуальная модель

В таблице 5 представлена спецификация связей между сущностями.

Таблица 5 – Спецификация связей между сущностями

№	Сущность 1	Сущность 2	Описание	Мощность
1	Пользователь	Документ	Один пользователь может создать несколько документов	1:M
2	Пользователь	Участник	Один пользователь может быть участником нескольких документов	1:M
3	Документ	Участник	Документ может иметь множество участников	1:M
4	Документ	Блок	Документ содержит множество блоков	1:M
5	Участник	Блок	Участник документа может управлять множеством блоков	1:M
6	Блок	Изображение	Блок может иметь множество изображений	1:M
7	Участник	Изображение	Участник документа может загрузить множество изображений	1:M

Построенная концептуальная модель отражает целостную логику функционирования информационной системы, описывая, каким образом организованы данные и какие зависимости между ними существуют.

2.2.2 Логическая модель базы данных

Логическая модель данных представляет из себя расширение концептуальной модели. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи. Также на этом этапе уточняется тип данных для каждого поля. В ходе логического проектирования базы данных элементы концептуальной.

Так как программные средства будут разрабатываться с учетом сервис – ориентированной архитектуры, то база данных будет единая, но сущности будут находиться в разных схемах, на основе логического разделения.

На рисунке 21 представлена логическая модель базы данных.

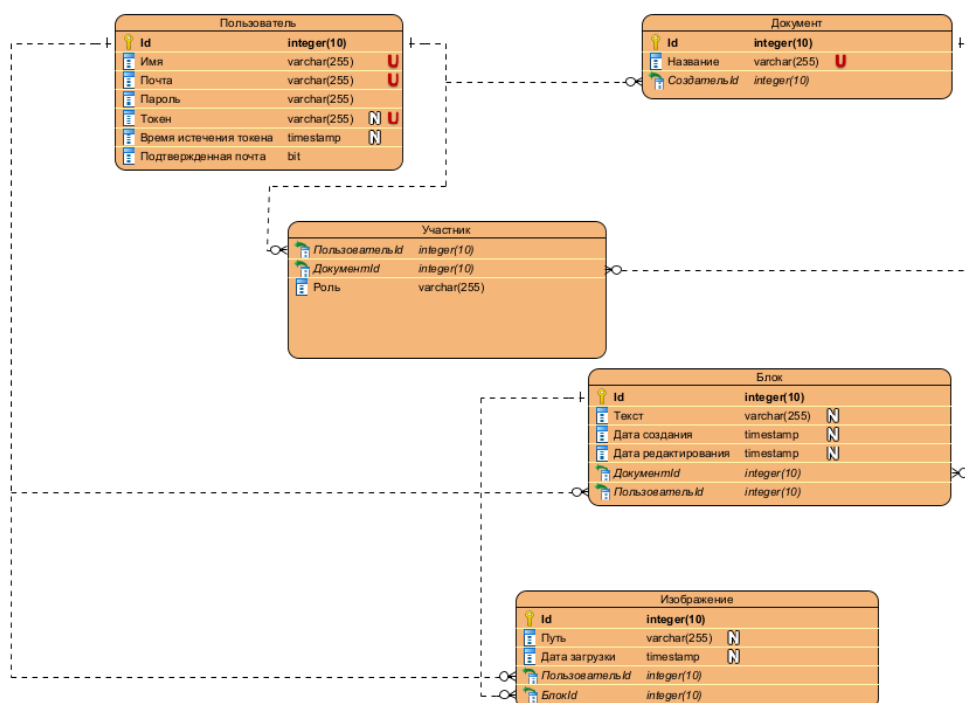


Рисунок 21 – Диаграмма сущность-связь логической модели

Таким образом на этапе логического проектирования базы данных особое внимание уделялось правилам ссылочной целостности, так как они обеспечивают корректность взаимодействия данных и сохраняют связи между таблицами при обновлении или удалении записей. Для реализации данного подхода на уровне СУБД были определены ограничения внешних ключей.

2.3 Проектирование серверной части

2.3.1 Проектирование сервиса AuthService

Сервис AuthService предназначен для управления учетными записями пользователей и обеспечения безопасной аутентификации в программном средстве управления текстовыми документами. Он отвечает за регистрацию новых пользователей, подтверждение email адресов, выдачу и обновление JWT токенов, а также управление профилем и паролем. Сервис взаимодействует с базой данных, брокером сообщений Kafka и системой мониторинга Prometheus, обеспечивая надежность и безопасность.

Архитектура сервиса разделена на следующие логические слои:

- слой обработки HTTP-запросов (AuthController, UsersController);
- сервисный слой (AuthService, TokenService, UserService);
- слой репозитория (UserRepository).

Слой обработки HTTP-запросов реализован через контроллеры AuthController и UsersController. Он принимает внешние запросы и делегирует их соответствующим сервисам.

Сервисный слой содержит основную бизнес-логику:

- AuthService управляет регистрацией, входом и подтверждением email;
- TokenService генерирует access- и refresh-токены с использованием криптографического ключа;
- UserService отвечает за операции с профилем (обновление email, смена пароля).

При успешной регистрации AuthService публикует событие в Kafka для асинхронной отправки сообщений на почту.

Слой репозитория представлен интерфейсом IUserRepository и его реализацией, которая работает напрямую с PostgreSQL через Dapper. Также используется Redis для временного хранения кодов подтверждения. При

регистрации генерируется одноразовый код, который сохраняется в Redis с ограниченным временем жизни и привязывается к email адресу.

Основной функционал:

- регистрация нового пользователя (с подтверждением по email);
- аутентификация по логину и паролю с выдачей JWT токена;
- изменение email;
- смена пароля;
- хранение и валидация временных кодов подтверждения в Redis.

В приложении Б, на рисунке Б.1 представлена диаграмма классов данного сервиса [12].

Ниже в таблице 6 представлено описание API сервиса.

Таблица 6 – Спецификация API AuthService

Метод доступа	Адрес	Описание
POST	/api/auth/register	Регистрация нового пользователя. Возвращает ID пользователя и сообщение об отправке кода
	/api/auth/login	Аутентификация пользователя. Возвращает JWT-токен и данные пользователя
	/api/auth/confirm-email	Подтверждение email-адреса с помощью одноразового кода. Возвращает токены при успехе
PUT	/api/auth/refresh-token	Обновление access-токена с использованием refresh-токена
	/api/users/me	Обновление профиля текущего пользователя (только на уже подтвержденный email)
	/api/users/me/password	Смена пароля текущего пользователя
GET	/api/users/me	Получение данных текущего авторизованного пользователя

Все операции, связанные с управлением учетной записью, доступны только авторизованным пользователям и защищены с помощью механизма JWT аутентификации. Для повышения безопасности пароли хранятся в зашифрованном виде, а временные коды подтверждения имеют ограниченное время.

2.3.2 Проектирование сервиса DocumentService

Сервис Document Service предназначен для создания, хранения и совместного использования документов в системе CollaboCraft. Он обеспечивает управление метаданными документов, а также контроль доступа на основе ролей.

Архитектура сервиса разделена на следующие логические слои:

- слой обработки HTTP-запросов и SignalR-хабов (DocumentController, DocumentHub, ParticipantHub);
- сервисный слой (DocumentService, DocumentParticipantService);
- слой репозитория (DocumentRepository, DocumentParticipantRepository).

Слой обработки запросов включает REST-контроллер DocumentController, который предоставляет API для получения списка документов пользователя и деталей конкретного документа, а также два SignalR-хаба:

- DocumentHub отвечает за операции с самим документом (создание, удаление, переименование);
- ParticipantHub управляет участниками (добавление, изменение ролей, удаление).

При подключении к хабу пользователь автоматически добавляется во все группы документов, в которых участвует, что обеспечивает мгновенную синхронизацию изменений.

Сервисный слой содержит бизнес-логику:

- DocumentService управляет документом;
- DocumentParticipantService обрабатывает операции с участниками и ролями.

Слой репозитория реализует взаимодействие с PostgreSQL через Dapper. Сущности documents и document_participants хранят метаданные и права доступа соответственно.

Все операции с документами учитывают контекст авторизованного пользователя и его роль.

Основной функционал:

- создание нового документа;
- получение списка документов, доступных пользователю;
- получение полного содержимого документа по идентификатору;

- обновление метаданных документа;
- назначение и отзыв прав доступа для других пользователей.

В приложении Б, на рисунке Б.2 представлена диаграмма классов данного сервиса.

Таблица 7 – Спецификация API сервиса управления документами

Метод доступа	Адрес	Описание
GET	/api/document/my	Получение списка документов, доступных текущему пользователю
	/api/document/{id}	Получение подробной информации о документе

В таблице 8 представлено описание методы SignalR-хабов.

Таблица 8 – Методы SignalR-хабов сервиса управления документами

Хаб	Метод	Описание
DocumentHub	CreateDocument	Создает новый документ, добавляет участников и уведомляет их через группу
	DeleteDocument	Удаляет документ. Удаляет группу и уведомляет участников
	RenameDocument	Переименовывает документ и рассылает новое имя участникам
ParticipantHub	AddUsersToDocument	Добавляет пользователей в документ
	ChangeUserRoleInDocument	Меняет роль пользователя в документе
	RemoveUserFromDocument	Удаляет пользователя из документа

Таким образом, сервис управления документами обеспечивает как синхронное взаимодействие через RESTful API, так и поддержку совместного редактирования в реальном времени посредством SignalR-хабов. Это позволяет реализовать гибкую модель совместной работы с документами, где изменения мгновенно синхронизируются между всеми участниками. Проектирование сервиса управления блоками.

2.3.3 Проектирование сервиса BlockService

Сервис Block Service предназначен для хранения, редактирования и синхронизации структурных единиц документа – блоков. Каждый документ представляет собой упорядоченную коллекцию блоков, а все операции над ними должны поддерживать совместное редактирование в реальном времени. Сервис

реализован на C# с использованием ASP.NET Core и обеспечивает мгновенную синхронизацию изменений между всеми участниками документа через SignalR.

Архитектура сервиса разделена на следующие логические слои:

- слой обработки запросов (BlockController, BlockImageController, BlockHub);
- сервисный слой (BlockService, BlockImageService);
- слой репозитория (BlockRepository, BlockImageRepository, IDocumentParticipantRepository).

Слой обработки запросов включает:

- REST-контроллеры BlockController и BlockImageController для первоначальной загрузки блоков и изображений при открытии документа;
- SignalR-хаб BlockHub, который обрабатывает все операции в реальном времени.

При выполнении любой операции хаб проверяет права пользователя и рассылает изменения всем участникам документа.

Сервисный слой содержит основную бизнес-логику:

- BlockService управляет текстовыми и структурными блоками и правами доступа;
- BlockImageService отвечает за загрузку и удаление медиафайлов, включая взаимодействие с MinIO.

Слой репозитория реализует работу с PostgreSQL через Dapper. Таблицы blocks и block_images хранят метаданные и ссылки на контент.

Основной функционал:

- создание, обновление и удаление блоков внутри документа;
- автоматическая синхронизация изменений со всеми участниками документа;
- валидация операций на основе прав пользователя.

Внешние зависимости:

- PostgreSQL – хранение метаданных и содержимого блоков;

- Prometheus – экспорт метрик;
- MinIO – хранение медиафайлов, если блок содержит изображение или вложение.

В приложении Б, на рисунке Б.3 представлена диаграмма классов данного сервиса.

Таблица 9 – Методы API и хабов сервиса управления блоками

Тип взаимодействия	Метод	Описание
REST	GET /api/block/{documentId}	Получение всех блоков документа, измененных после указанного времени
	GET /api/blockimage/blocks/{blockId}/images	Получение списка изображений, прикрепленных к блоку
SignalR	SendBlock	Создание нового блока и рассылка события ReceiveBlock участникам
	EditBlock	Обновление содержимого блока, рассылка BlockEdited
	DeleteBlock	Удаление блока, рассылка BlockDeleted
	SendBlockImage	Загрузка изображения к блоку, рассылка ReceiveBlockImage
	DeleteBlockImage	Удаление изображения, рассылка BlockImageDeleted

Таким образом, сервис управления блоками обеспечивает как загрузку структуры документа при первом подключении, так и мгновенную синхронизацию изменений в реальном времени. Все операции строго контролируют права доступа на основе участия пользователя в документе.

2.3.4 Проектирование сервиса EmailService

Сервис EmailService предназначен исключительно для фоновой обработки событий, поступающих из Apache Kafka, и отправки email уведомлений конечным пользователям. В отличие от других сервисов, он не предоставляет публичного REST API и не взаимодействует с клиентами напрямую. Его единственная задача заключается в потреблении событий и надежной доставке сообщения через внешний SMTP сервер.

Основной функционал:

- подписка на Kafka-топик send-email-code;
- десериализация события SendEmailCodeEvent, содержащего email адрес и одноразовый код подтверждения;
- отправка письма через SMTP с использованием библиотеки MailKit.

Внешние зависимости:

- Kafka – источник событий;
- SMTP-сервер – внешний почтовый сервис.

Сервис полностью соответствует принципу единой ответственности. Он не хранит данные, не обрабатывает запросы пользователей и не зависит от состояния других сервисов, кроме Kafka и SMTP.

Ниже на рисунке 22 представлена диаграмма классов данного сервиса.

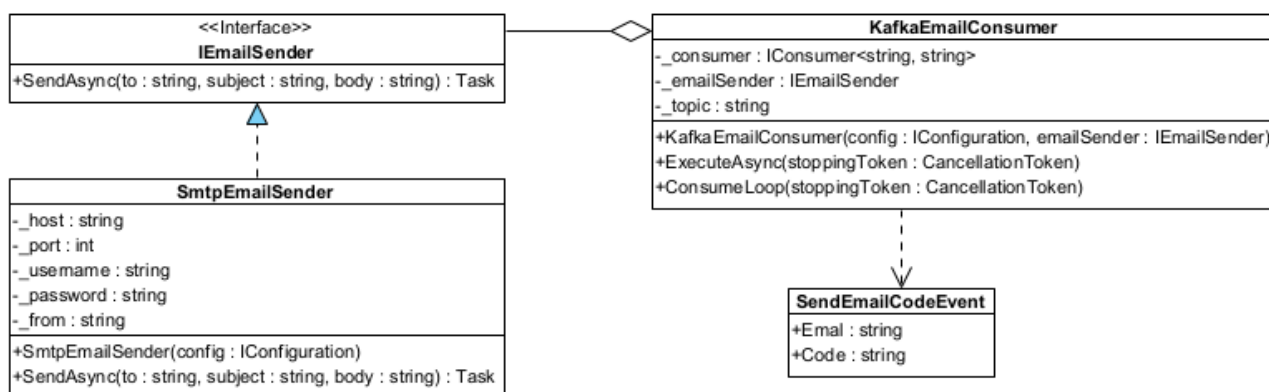


Рисунок 22 – Диаграмма классов EmailService

Такой подход обеспечивает слабую связанность между Auth Service и механизмом отправки писем. Авторизационный сервис не блокируется ожиданием отправки email, а ошибка в рассылке не влияет на успешность регистрации пользователя.

2.4 Проектирование клиентской части

Разработка клиентской части программных средств совместного редактирования документов велась в рамках концепции одностраничных приложений с использованием фреймворка React. Применение данной

технологии обусловлено необходимостью оперативного управления состоянием интерфейса и рендеринга сложных динамических элементов [13].

Архитектура проекта построена модульно и отражает разделение кода по функциональным областям. Логическая организация каталогов обеспечивает удобство масштабирования, поддержку читаемости и возможность дальнейшего развития приложения. Каждый раздел проекта отвечает за собственный аспект работы интерфейса и взаимодействия с серверной частью. За счет такой изоляции изменение внутренней логики одного модуля не влечет за собой каскадных правок. Это позволяет вести параллельную разработку, минимизируя возникновение конфликтов.

В ходе проектирования клиентской части приложения был разработан интерфейс, представленный на рисунках 23 – 26.

Прототип интерфейса авторизации представлен на рисунке 23.

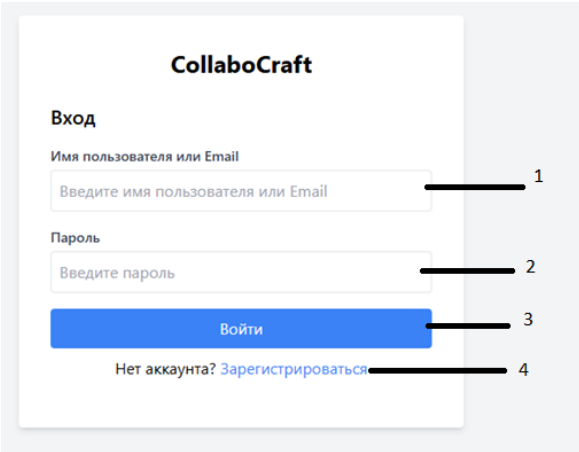
Прототип формы авторизации для CollaboCraft. Форма имеет белый фон и заголовок 'CollaboCraft'. Под заголовком находится раздел 'Вход'. В этом разделе есть два текстовых поля: 'Имя пользователя или Email' и 'Пароль'. Первое поле имеет подсказку 'Введите имя пользователя или Email'. Второе поле имеет подсказку 'Введите пароль'. Под полями находится синяя кнопка 'Войти'. В самом низу формы есть ссылка 'Нет аккаунта? Зарегистрироваться'. Справа от формы, на сером фоне, есть четыре черные линии с цифрами 1, 2, 3 и 4, указывающие на соответствующие элементы: 1 – поле ввода логина, 2 – поле ввода пароля, 3 – кнопка 'Войти', 4 – ссылка на регистрацию.

Рисунок 23 – Прототип формы авторизации

Рассмотрим основные элементы интерфейса формы авторизации, представленные на рисунке 23:

- 1 – поле ввода логина;
- 2 – поле ввода пароля;
- 3 – кнопка для входа;
- 4 – кнопка для перехода на страницу регистрации.

Прототип интерфейса регистрации, представленный на рисунке 24.

Регистрация

1 Email

2 Имя пользователя

3 Пароль

4 Подтверждение пароля

5 Зарегистрироваться

6 Уже есть аккаунт? Войти

Рисунок 24 – Прототип формы регистрации

Рассмотрим основные элементы интерфейса формы регистрации, представленные на рисунке 24:

- 1 – поле ввода email;
- 2 – поле ввода имени;
- 3 – поле ввода пароля;
- 4 – поле повторного ввода пароля;
- 5 – кнопка для регистрации;
- 6 – кнопка для перехода на страницу авторизации.

На рисунке 25 приведен прототип интерфейса главной страницы.

Мои документы

1 Создать документ

2 Профиль

3 Выйти

Документ_1 (Creator)

4

5 +

6 ⚙

7 🗑

Рисунок 25 – Прототип интерфейса главной страницы

Рассмотрим основные элементы интерфейса главной страницы, представленные на рисунке 25:

- 1 – кнопка для создания документа;
- 2 – кнопка для перехода на страницу профиля пользователя;
- 3 – кнопка для перехода на страницу авторизации;
- 4 – кнопка для перехода к содержимому документа;

- 5 – кнопка для добавления новых участников;
- 6 – кнопка для настройки документа;
- 7 – кнопка для удаления документа.

Прототип интерфейса страницы редактирования документа представлен на рисунке 26.

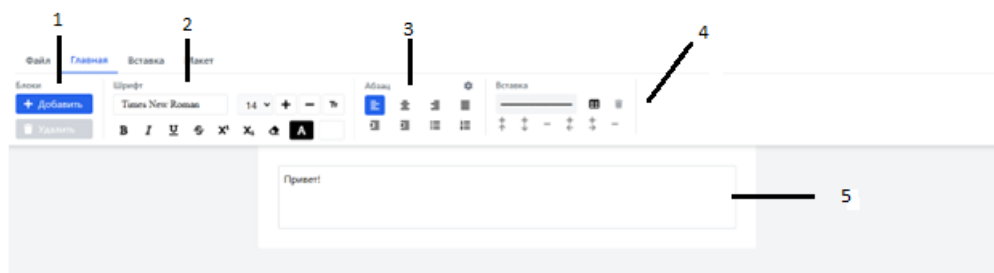


Рисунок 26 – Прототип интерфейса окна редактирования документа

Рассмотрим основные элементы интерфейса формы авторизации, представленные на рисунке 26:

- 1 – кнопки для управления блоками;
- 2 – инструменты для управления шрифтом;
- 3 – инструменты для управления абзацем;
- 4 – инструменты для управления таблицами.

Файловая структура клиентского приложения имеет следующий вид:

- `api` – набор модулей, инкапсулирующих взаимодействие с серверной частью через HTTP-запросы;
- `Components` – каталог пользовательских компонентов, включая крупные функциональные элементы интерфейса (редактор документов, страницы авторизации, панель инструментов и другое);
- `Extensions` – набор расширений и утилит, интегрирующих сторонние библиотеки с логикой приложения;
- `Models` – каталог, содержащий модели данных, используемых в процессе взаимодействия с бэкендом.

3 Реализация программных средств совместного редактирования документов

3.3 Реализация базы данных

Физическая ER-диаграмма – это разновидность блок-схем, где показано, как разные сущности связаны между собой внутри системы. Данные должны быть структурированы и связаны в конкретной СУБД и поэтому важно учитывать ограничения используемой СУБД [14].

Для построения физической модели базы данных использовалась программа Visual Paradigm, нотация IDEF1X. Разработанная модель представлена на рисунке 27.

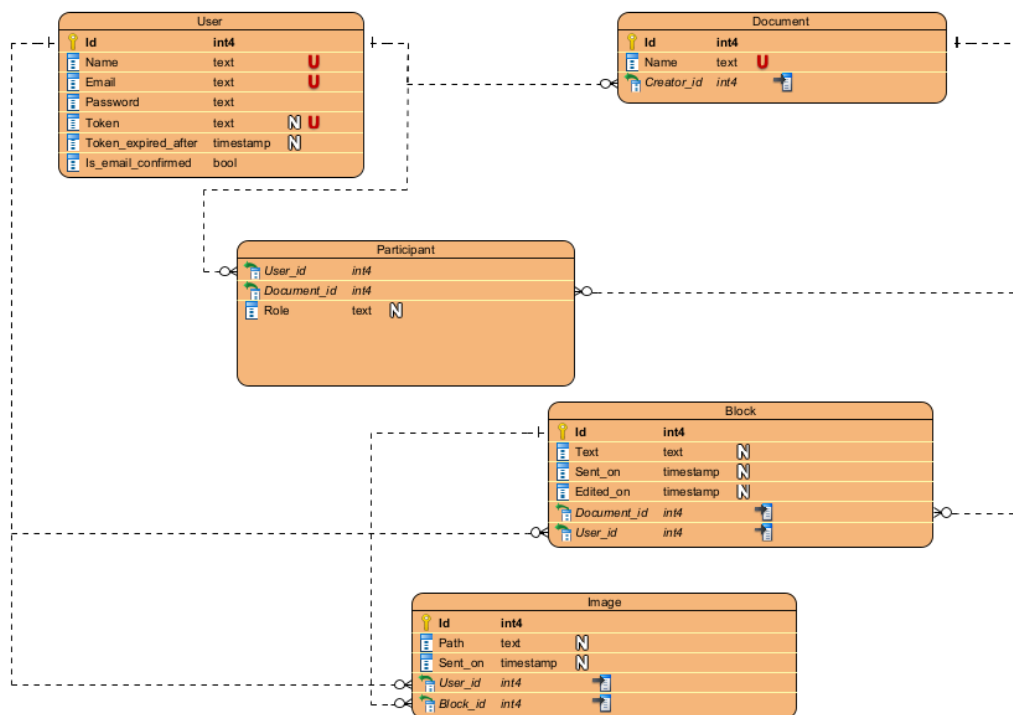


Рисунок 27 – Физическая схема базы данных

В процессе разработки были выбраны подходящие типы данных и определены стратегии индексирования, позволяющие ускорить операции чтения. Внешние ключи были дополнены политиками ON DELETE и ON UPDATE, позволяющими корректно изменять родительские записи без нарушения целостности базы данных.

Описание сущности «Документ» приведено в таблице 10.

Таблица 10 – Сущность «Документ»

Атрибут	Тип	Обязательность	Ключ	Ограничения	Индекс
Id	integer	Да	PK	UNIQUE	Уникальный индекс В-дерева
Name	text	Да	-	UNIQUE	-
Creator_id	integer	Да	FK	-	Индекс В-дерева

Описание сущности «Пользователь» приведено в таблице 11.

Таблица 11 – Сущность «Пользователь»

Атрибут	Тип	Обязательность	Ключ	Ограничения	Индекс
Id	integer	Да	PK	UNIQUE	Уникальный индекс В-дерева
Name	text	Да	-	UNIQUE	-
Email	text	Да	-	UNIQUE	-
Password	text	Да	-	-	-
Token	text	Да	-	UNIQUE	-
Token_expired_after	timestamp	Да	-	-	-
Is_email_confirmed	bool	Да	-	-	-

Описание сущности «Участник» приведено в таблице 12.

Таблица 12 – Сущность «Участник»

Атрибут	Тип	Обязательность	Ключ	Ограничения	Индекс
User_id	integer	Да	FK	-	Индекс В-дерева
Document_id	integer	Да	FK	-	Индекс В-дерева
Role	text	Да	-	-	-

Описание сущности «Блок» приведено в таблице 13.

Таблица 13 – Сущность «Блок»

Атрибут	Тип	Обязательность	Ключ	Ограничения	Индекс
Id	integer	Да	PK	UNIQUE	Уникальный индекс В-дерева
Text	text	Да	-	-	-
Sent_on	timestamp	Да	-	-	-
Document_id	integer	Да	FK	-	Индекс В-дерева
User_id	integer	Да	FK	-	Индекс В-дерева
Edited_on	timestamp	Да	-	-	-

Описание сущности «Изображение» приведено в таблице 14.

Таблица 14 – Сущность «Изображение»

Атрибут	Тип	Обязательность	Ключ	Ограничения	Индекс
Id	integer	Да	PK	UNIQUE	Уникальный индекс В-дерева
Path	text	Да	-	-	-
Sent_on	timestamp	Да	-	-	-
User_id	integer	Да	FK	-	Индекс В-дерева
Block_id	integer	Да	FK	-	Индекс В-дерева

База данных, соответствующая разработанной физической модели, была развернута с помощью механизма миграций с использованием библиотеки DbUp. Миграции позволяют последовательно применять SQL-скрипты для

управления изменениями в схеме базы данных и автоматически отслеживать их версии.

3.4 Реализация серверной части приложения

Реализация основного приложения проводилась как набор сервисов в среде разработки Microsoft Visual Studio 2022, на языке программирования C#, с использованием среды ASP.NET Core. Каждый сервис спроектирован с использованием слоистой архитектуры, где каждый слой четко отделен от остальных. На рисунке 28 представлен типовой вид многослойной архитектуры.

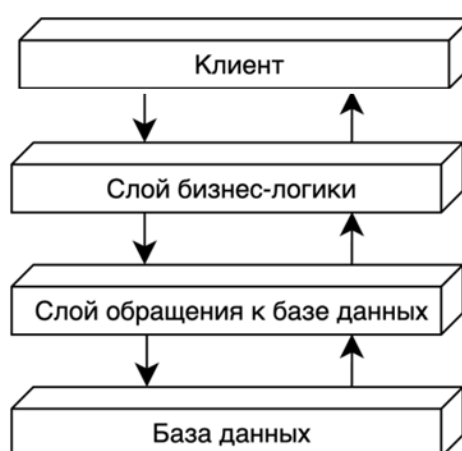


Рисунок 28 – Типовая архитектура сервисов

Такая реализация позволяет обеспечить масштабируемость и четкое разделение ответственности между компонентами системы. Фрагмент кода серверной части приложения приведен в приложении В. Полный код расположен на сервисе git [15].

Также была разработана диаграмма развертывания, отражающая архитектуру приложения, представленная на рисунке 29. Данное графическое представление визуализирует физическое распределение программных компонентов по узлам системы и определяет топологию их сетевого взаимодействия. В частности, схема демонстрирует контейнеризацию серверной части, где фронтенд приложение и бизнес-логика изолированы внутри среды Docker.

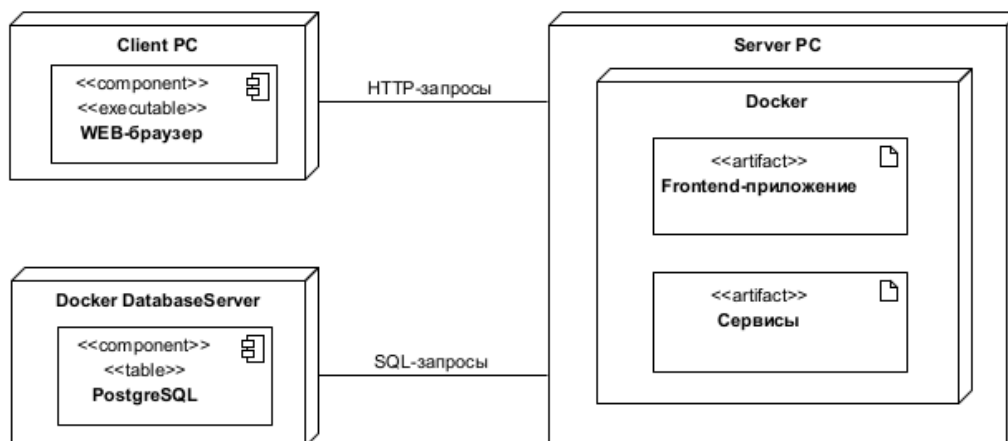


Рисунок 29 – Диаграмма развертывания

Представленная архитектурная схема разделена на три ключевых инфраструктурных узла, обеспечивающих функционирование программных средств. В их число входит клиентский терминал, удаленный выделенный сервер, а также сервер базы данных, отвечающий за структурированное хранение и обработку информации.

3.4.2 Реализация авторизации приложения

Сервис авторизации необходим для обеспечения безопасности в распределенной системе. Он обеспечивает:

- аутентификацию пользователей;
- авторизацию;
- управление сессиями;
- регистрацию.

Генерация JWT токена с настраиваемыми claims, а также генерация refresh-токена представлена в листинге 1.

Листинг 1 – Генерация refresh-токена

```

public class TokenService(IAuthSettings authSettings,
    IUserRepository userRepository) : ITokenService
{
    public string CreateAccessToken(IEnumerable<Claim> claims)
    {

```

```

        var authSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(authSettings.Key));

        var token = new JwtSecurityToken(
            issuer: authSettings.Issuer,
            audience: authSettings.Audience,
            claims: claims,
            expires:
DateTime.UtcNow.AddMinutes(authSettings.AccessTokenExpiresInMinute
s),
            signingCredentials: new
SigningCredentials(authSigningKey, SecurityAlgorithms.HmacSha256)
        );
        return new JwtSecurityTokenHandler().WriteToken(token);
    }

    public async Task<AuthResponse> RefreshToken(string
refreshToken)
    {
        var user = await
userRepository.GetUserByRefreshToken(refreshToken);
        if (user.RefreshTokenExpiredAfter < DateTime.UtcNow)
            throw new SecurityTokenExpiredException("Refresh token
expired");
        var claims = Jwt.GetClaims(user.Id, user.Role, user.Email,
user.Username);
        var newAccessToken = CreateAccessToken(claims);
        var refreshThreshold = TimeSpan.FromDays(1);

        if (user.RefreshTokenExpiredAfter - DateTime.UtcNow <
refreshThreshold)
        {
            newRefreshToken = GenerateRefreshToken();
            await userRepository.UpdateRefreshToken(
                user.Id,
                newRefreshToken,
            );
        }

        return new AuthResponse
        {
            AccessToken = newAccessToken,
            RefreshToken = newRefreshToken
        };
    }
}

```

Пароли хранятся в базе данных в виде хэшей с использованием соли. Код хеширования представлен в листинге 2.

Листинг 2 – Код хеширования

```
public static string HashPassword(string password, byte[] pepper)
{
    using var rng = RandomNumberGenerator.Create();
    byte[] salt = new byte[32];
    rng.GetBytes(salt);

    byte[] pepperedPassword = CombinePepperAndPassword(password,
        pepper);

    var pbkdf2 = new Rfc2898DeriveBytes(pepperedPassword, salt,
        100_000, HashAlgorithmName.SHA256);
    byte[] hash = pbkdf2.GetBytes(32);

    var hashBytes = new byte[64];
    Array.Copy(salt, 0, hashBytes, 0, 32);
    Array.Copy(hash, 0, hashBytes, 32, 32);

    return $"{HashVersion}:{Convert.ToBase64String(hashBytes)}";
}
```

На основе реализованных алгоритмов аутентификации и авторизации, включающих процедуры регистрации, верификации учетных данных и генерации сессионных ключей, достигается необходимый уровень информационной безопасности.

3.4.3 Реализация взаимодействия с Kafka

Для межсервисного взаимодействия используется брокер сообщений Kafka. Он обеспечивает подтверждение почты путем отправки сообщений из сервиса авторизации в сервис рассылки.

Процесс отправки кода подтверждения начинается с метода ProduceAsync, который вызывается после успешного создания пользователя. Он формирует сообщение, состоящее из email и четырехзначного кода и помещает его в очередь. Реализация отправления сообщения представлена в листинге 3.

Листинг 3 – Kafka-продюсер

```
public class KafkaProducer : IKafkaProducer, IDisposable
{
    private readonly IProducer<string, string> _producer;

    public KafkaProducer(IConfiguration config)
    {
        var producerConfig = new ProducerConfig
```

```

        {
            BootstrapServers = config["Kafka:BootstrapServers"] ??
"localhost:9092"
        };

        _producer = new ProducerBuilder<string,
string>(producerConfig).Build();
    }

    public async Task ProduceAsync<T>(string topic, T message)
    {
        var json = JsonSerializer.Serialize(message);
        await _producer.ProduceAsync(topic, new Message<string,
string>
        {
            Key = Guid.NewGuid().ToString(),
            Value = json
        });
    }

    public void Dispose()
    {
        _producer?.Dispose();
    }
}

```

Сервис рассылки получает сообщение из очереди, десериализует его и отправляет код подтверждения на указанную почту. Реализация потребителя сообщений представлена в листинге 4.

Листинг 4 – Реализация потребителя сообщений

```

public class KafkaEmailConsumer : BackgroundService
{
    public KafkaEmailConsumer(IConfiguration config, IEmailSender
emailSender)
    {
        var consumerConfig = new ConsumerConfig
        {
            BootstrapServers = config["Kafka:BootstrapServers"] ??
"kafka:9092",
            GroupId = "email-service",
            AutoOffsetReset = AutoOffsetReset.Earliest,
            EnableAutoCommit = true
        };
        _consumer = new ConsumerBuilder<string,
string>(consumerConfig).Build();
        _topic = config["Kafka:Topics:SendEmailCode"] ?? "send-
email-code";
        _emailSender = emailSender;
    }
}

```

```

        protected override Task ExecuteAsync(CancellationToken
stoppingToken)
        {
            Task.Run(() => ConsumeLoop(stoppingToken), stoppingToken);
            return Task.CompletedTask;
        }

        private async Task ConsumeLoop(CancellationToken
stoppingToken)
        {
            _consumer.Subscribe(_topic);
            while (!stoppingToken.IsCancellationRequested)
            {
                try
                {
                    var cr = _consumer.Consume(stoppingToken);
                    var evt =
JsonSerializer.Deserialize<SendEmailCodeEvent>(cr.Message.Value);
                    if (evt != null)
                    {
                        try
                        {
                            await _emailSender.SendAsync(
                                evt.Email,
                                "Подтверждение Email",
                                $"Ваш код подтверждения: {evt.Code}"
                            );
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Ошибка при обработке
сообщения: {ex.Message}");
                }
            }
        }
    }
}

```

Таким образом брокер сообщений Kafka обеспечивает асинхронный обмен сообщениями между сервисами авторизации и рассылки сообщений. Он позволяет сервисам работать независимо друг от друга и обеспечивает доставку и обработку сообщений даже при высоких нагрузках.

3.4.4 Реализация взаимодействия в реальном времени и маршрутизации через Nginx

Основой для программных средств совместного редактирования документов является взаимодействие в реальном времени. Для более слабой

связанности, коммуникация между сервисами строиться на основе событий, а сама реализация взаимодействия построена с использованием библиотеки SignalR. Пример реализации SignalR-хаба представлен в листинге 5.

Листинг 5 – Реализация хаба управления документами

```
public class DocumentHub
    (IConnectionTracker connectionTracker,
     IDocumentService documentService,
     IDocumentParticipantService documentParticipantService,
     IUserRepository userRepository
    ) : BaseHub
{
    public async Task CreateDocument(CreateDocumentRequest
request)
    {
        {
            try
            {
                request.CreatorId = Id;
                var users = request.Usernames != null &&
request.Usernames.Any()
                    ? await
userRepository.GetUsersByUsernamesAsync(request.Usernames)
                        : new List<DbUser>();
                request.UserIds = users?.Select(u =>
u.Id).ToList() ?? new List<int>();
                var document = await
documentService.CreateDocument(request);
                var connectionIds = connectionTracker
                    .SelectConnectionIds(request.UserIds)
                    .Append(Context.ConnectionId);
                await
Task.WhenAll(connectionIds.Select(connectionId =>
Groups.AddToGroupAsync(connectionId,
$"Document{document.Id}")));
                await Clients.Caller.SendAsync("DocumentCreated",
document);
                await Clients.Group($"Document{document.Id}")
                    .SendAsync("AddedToDocument",
request.UserIds);
                DocumentHubMetrics.DocumentsCreated.Inc();
            }
            catch (Exception e)
            {
                throw new HubException(e.Message);
            }
        }
    }

    public async Task DeleteDocument(int documentId)
```

```

{
    var documentParticipants = await
documentParticipantService.GetDocumentParticipantsById(Id);
    var documentIds = documentParticipants.Select(x =>
x.DocumentId);
    await Task.WhenAll(documentIds.Select(documentId =>
        Groups.AddToGroupAsync(Context.ConnectionId,
        $"Document{documentId}")));
    await base.OnConnectedAsync();
}
}

```

Для контейнеризации инфраструктуры используется Docker. Все сервисы, база данных, Kafka и MinIO объединены в одну bridge сеть, а маршрутизация осуществляется через Nginx, который проксирует HTTP и WebSocket запросы к соответствующим сервисам. Код конфигурации Nginx представлен в листинге 6.

Листинг 6 – Конфигурация nginx

```

server {
    listen 80;
    server_name _;

    root /usr/share/nginx/html;
    index index.html;

    location /api/document/ {
        proxy_pass http://document_service:80;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api/auth/ {
        proxy_pass http://auth_service:80;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Authorization $http_authorization;
    }

    location /api/block/ {
        proxy_pass http://block_service:80;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
    }
}

```

```

        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Authorization $http_authorization;
    }

    location /api/blockimage/ {
        proxy_pass http://block_service:80;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Authorization $http_authorization;
    }

    location /participanthub {
        proxy_pass http://document_service:80/participanthub;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location / {
        try_files $uri /index.html;
    }
}

```

Таким образом была разработана гибкая инфраструктура, использующая событийно-ориентированное взаимодействие. А использование Docker для контейнеризации и Nginx в качестве обратного прокси сервера позволило организовать централизованную маршрутизацию.

3.5 Реализация клиентской части приложения

Создание клиентского компонента программных средств совместного редактирования документов осуществлялась с использованием редактора кода Visual Studio Code. В качестве технологий разработки был выбран строго типизированный язык TypeScript, а построение интерфейсной логики и модулей выполнялось на базе фреймворка React.

Для взаимодействия с серверной частью используется библиотека `axios`, который автоматически добавляет JWT токен к запросам, а для стилизации готовые стили из библиотеки `Tailwind CSS`. Функция, отвечающая за добавление токена представлена в листинге 7. Код фрагмента клиентской части приложения приведен в приложении Г, а графическое представление в приложении Д. Полный код расположен на сервисе `git` [15].

Листинг 7 – Добавление токена

```
export const api = axios.create({ baseURL: '/api' });

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

api.interceptors.response.use(
  res => res,
  async (error) => {
    if (error.response?.status === 401) {
      await refreshAccessToken();
      return api(error.config);
    }
    return Promise.reject(error);
  }
);
```

Данная функция обеспечивает простую работу с аутентификацией. При истечении токена приложение автоматически его обновляет и повторяет исходный запрос без участия пользователя.

Для подключения к сервисам реального времени была реализована фабрика `SignalR`-хабов, которая создает и управляет подключениями, а также предотвращает попытки дублирования подключения. Код фабрики представлен в листинге 8.

Листинг 8 – Фабрика для создания подключения к хабу

```
class SignalRHubFactory {
  private hubs: Map<string, HubInstance> = new Map();
  public createHub(url: string, tokenKey = 'accessToken'):
    HubInstance {
    const connection = new signalR.HubConnectionBuilder()
      .withUrl(url, {
```

```

        accessTokenFactory: () => localStorage.getItem(tokenKey)
    || '',
        skipNegotiation: true,
        transport: signalR.HttpTransportType.WebSockets,
    })
    .build();
    const instance = { connection, isConnecting: false };
    this.hubs.set(url, instance);
    return instance;
}
public async sendMessage(url: string, method: string, args:
any[]) {
    return this.hubs.get(url)!.connection.invoke(method, ...args);
}
}

```

Клиентская часть приложения реализована как одностраничное веб-приложение, а каждая часть интерфейса реализована в виде компонента с локальным управлением состоянием и валидацией. Пример модуля «Редактирование профиля» представлен в листинге Г.4 приложения Г.

Таким образом вся клиентская часть стилизуется с использованием Tailwind CSS, а статика раздается через Nginx, а реализованное приложение обеспечивает отзывчивый интерфейс, мгновенную синхронизацию и надежную работу с аутентификацией.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования и метрик кода

Тестирование позволяет гарантировать отказоустойчивость приложения, своевременно выявить логические дефекты, а также подтвердить корректность интеграции изолированных сервисов в рамках единой архитектуры. Проведение контрольных мероприятий ориентировано на выявление критических ошибок и архитектурных уязвимостей [16].

Современная методология обеспечения качества включает в себя широкий спектр подходов к проверке корректной работоспособности программного обеспечения. Классификация наиболее востребованных методов верификации, а также их описание приведены далее [17].

Виды тестирования:

- функциональное;
- интеграционное;
- тестирование производительности;
- модульное.

Ключевая цель функционального тестирования заключается в подтверждении того, что поведение приложения полностью соответствует заданным критериям. В рамках данного подхода осуществляется проверка, как отдельных модулей и компонентов, так и сквозных пользовательских сценариев [18].

Преимущества функционального тестирования:

- имитирует фактическое использование ПО;
- позволяет выявить ошибки в логике работы ПО.

Недостатки функционального тестирования:

- может быть избыточным;
- не позволяет выявить ошибки в производительности, надежности и безопасности.

Проведение интеграционных проверок ориентировано на оценку работоспособности взаимосвязанных элементов системы. Данный подход позволяет своевременно обнаружить дефекты на стыках интерфейсов взаимодействия изолированных компонентов системы.

Преимущества интеграционного тестирования:

- позволяет выявить ошибки во взаимодействии компонентов;
- оценить, как работает приложение в целом.

Недостатки интеграционного тестирования:

- может быть трудоемким;
- зависимость от готовности смежных компонентов.

Тестирование производительности направлено на проверку того, как приложение работает при различных нагрузках. Оно включает в себя измерение времени выполнения операций и количества пользователей, которые могут одновременно работать с ПО.

Преимущества тестирования производительности:

- позволяет оценить работу приложения в условиях близких к реальным;
- позволяет выявить проблемы в производительности системы.

Недостатки тестирования производительности:

- требует специального оборудования;
- сложность и высокая стоимость;
- позднее выявление ошибок.

Модульное тестирование направлено на проверку работоспособности отдельных модулей программы в полной изоляции от остальной системы. Они пишутся разработчиками в процессе разработки основного кода.

Преимущества модульного тестирования:

- позволяет выявить ошибки на ранних этапах разработки;
- позволяют в будущем безопасно производить рефакторинг кода;
- служат документацией к функциям.

Недостатки модульного тестирования:

- не позволяют выявить ошибки во взаимодействии модулей;
- необходимо обновлять при изменении функционала.

Чтобы покрыть весь спектр бизнес-логики, требуется разработать детализированные тест-кейсы. В основу закладываются пользовательские сценарии, что позволяет последовательно протестировать критически важные функции системы в условиях, максимально приближенных к реальной эксплуатации.

В качестве метода автоматизированного тестирования было выбрано модульное тестирование. Данный метод позволяет эффективно и в короткий срок провести проверку отдельных компонентов.

Метрики кода представляют из себя набор программных показателей, которые позволяют разработчикам лучше понять код. Они помогают понять какие типы и методы следует переработать или более тщательно протестировать [19].

В качестве методической основы оценки качества программного кода были определены классические параметры, среди которых ключевая роль отведена метрикам Холстеда, оценивающим объемную сложность [20].

4.2 Тестирование программных средств совместного редактирования документов

4.2.1 Функциональное тестирование

Пользовательская часть приложения в рамках основного функционала представлена созданием документа с заданным названием. Для поля «Название документа» были определены требования, которым соответствуют выделенные классы эквивалентности, представленные в таблице 15.

Таблица 15 – Классы эквивалентности

Требование	Правильные КЭ	Неправильные КЭ
Длина названия документа	1 – 100 (1)	0 (2) 101 - ∞ (3)

Продолжение таблицы 15

Требование	Правильные КЭ	Неправильные КЭ
Содержание	Кириллица(4) Латиница(5) Цифры(6) Пробелы внутри строки (7)	Спецсимволы (@#\$\$%^&*) (8)
Уникальность	Уникальное имя (9)	Дубликат существующего имени (10)

Проведенное разбиение на классы эквивалентности легло в основу создания тест кейсов, представленных в таблице 16. Результаты тестирования приведены в приложении Е.

Таблица 16 – Тест-кейсы для выделенных классов эквивалентности

Номер теста	Классы эквивалентности	Входные данные	Выходные данные
1	Все правильные 1, 4, 5, 6, 7, 9	Мой Doc 12	Успешно (Рисунок Е.1)
2	1,4,7,9	Документ1	Успешно (Рисунок Е.2)
3	1,6,9	2026	Успешно (Рисунок Е.3)
4	Заменяем длину 2	«»	Ошибка: название пустое (Рисунок Е.4)
5	Заменяем содержание 1,8,9	Doc@1	Ошибка: недопустимые символы (Рисунок Е.5)
6	Заменяем уникальность 1,5,7,10	Doc 12	Ошибка: документ уже существует (Рисунок Е.6)
7	Проверка пробелов 1,5,8,9	My Doc	Успешно (Рисунок Е.7)
8	Проверка длины 3, 5, 9	DDDDD(101)	Ошибка: слишком длинное название (Рисунок Е.8)
9	Комбинация: цифры и пробел 1,6,7,9	2026 1	Успешно (Рисунок Е.9)

Для повышения надежности сформированный набор классов эквивалентности был расширен за счет определения предельных значений. Перечень граничных значений, установленных для валидации ввода наименования документа описан в таблице 17.

Таблица 17 – Граничные условия

Тип границы	Границы
Физические (ф1)	Отрицательная длина строки (невозможное значение)
Логическая (л1)	Дубликат существующего названия
Произвольная (п1)	Минимальная допустимая длина = 1
Произвольная (п2)	Максимальная допустимая длина 100 символов
Произвольная (п3)	Наличие запрещенных спецсимволов

На основе определенных граничных значений был сформирован дополнительный комплекс тестовых сценариев, представленных в таблице 18.

Таблица 18 – Тест-кейсы для выделенных граничных условий

№	Границы	Входные данные	Выходные данные
1	П1	А	Успешно (Рисунок Е.10)
2	П1	«»	Ошибка: название не может быть пустым (Рисунок Е.11)
3	П2	D (100 символов)	Успешно (Рисунок Е.12)
4	П2	D (101 символ)	Ошибка: превышена максимальная длина (Рисунок Е.13)
5	П3	Doc@1	Ошибка: недопустимые символы (Рисунок Е.14)
6	П3	Doc1	Успешно (Рисунок Е.15)
7	Л1	Doc 12	Ошибка: документ с таким названием уже существует (Рисунок Е.16)

Программа успешно прошла все тесты и подтвердила корректность обработки минимального и максимального названия документа, а также выявило ожидаемое поведение системы при превышении допустимых значений.

4.2.2 Модульное тестирование

Для проверки корректности работы было проведено модульное тестирование. В качестве инструмента тестирования был выбран популярный фреймворк xUnit.

Для каждого сервиса были разработаны тестовые наборы, покрывающие как успешные, так и неуспешные случаи. Все тесты реализованы в соответствии с принципами изолированного unit-тестирования. Генерация данных производится автоматически с использованием библиотеки AutoFixture, а проверка поведения с помощью анализа возвращаемых значений. Фрагмент кода модульных тестов приведен в приложении Ж. Полный код расположен на сервисе git [15].

Для сервиса авторизации было разработано 13 тестов, представленных в таблице 19.

Таблица 19 – Тесты для сервиса авторизации

№	Название теста	Сценарий
1	Register_ValidModel_Success	Успешная регистрация нового пользователя с уникальными username/email
2	Register_PasswordsDoNotMatch_ThrowsInvalidInputException	Ошибка при несовпадении пароля и подтверждения
3	Register_EmptyFields_ThrowsInvalidInputException	Ошибка при пустых обязательных полях

Продолжение таблицы 19

№	Название теста	Сценарий
4	Register_UsernameAlreadyTaken_ThrowsUsernameAlreadyTakenException	Ошибка при попытке зарегистрироваться с занятым username
5	Register_EmailAlreadyTaken_ThrowsEmailAlreadyTakenException	Ошибка при попытке зарегистрироваться с занятым email
6	Login_ValidCredentials_Success	Успешный вход при правильных кредах и подтвержденном email
7	Login_InvalidCredentials_ThrowsBadCredentialsException	Ошибка при неверных логине/пароле
8	Login_EmailNotConfirmed_ThrowsInvalidOperationException	Ошибка при входе с неподтвержденным email
9	ConfirmEmail_ValidCode_Success	Успешное подтверждение email по корректному коду
10	ConfirmEmail_UserNotFound_ThrowsInvalidInputException	Ошибка при подтверждении несуществующего пользователя
11	ConfirmEmail_InvalidCode_ThrowsInvalidInputException	Ошибка при неверном коде подтверждения
12	ConfirmEmail_ExpiredCode_ThrowsInvalidInputException	Ошибка при истекшем коде подтверждения
13	Logout_CallsUpdateRefreshTokenWithNulls	Успешный logout – бнуление refresh token

Результат выполнения тестов представлен ниже на рисунке 30.

AuthService.Tests (13)

847 мс

AuthService.Tests (13)

847 мс

AuthServiceTests (13)

847 мс

ConfirmEmail_ExpiredCode_ThrowsInvalidInp...

1 мс

ConfirmEmail_InvalidCode_ThrowsInvalidInpu...

3 мс

ConfirmEmail_UserNotFound_ThrowsInvalidIn...

< 1 мс

ConfirmEmail_ValidCode_Success(userReposit...

8 мс

Login_EmailNotConfirmed_ThrowsInvalidOper...

269 мс

Login_InvalidCredentials_ThrowsBadCredentia...

1 мс

Login_ValidCredentials_Success(userRepositor...

176 мс

Logout_CallsUpdateRefreshTokenWithNulls(u...

6 мс

Register_EmailAlreadyTaken_ThrowsEmailAlre...

287 мс

Register_EmptyFields_ThrowsInvalidInputExce...

< 1 мс

Register_PasswordsDoNotMatch_ThrowsInvali...

< 1 мс

Register_UsernameAlreadyTaken_ThrowsUser...

2 мс

Register_ValidModel_Success(userRepository...

94 мс

Сводка по группе

AuthService.Tests

Тесты в группе: 13

🕒 Общая длительность: 847 мс

Результаты

✔ 13 Пройден

Рисунок 30 – Тестирование сервиса авторизации

Для сервиса управления документами и правами было разработано 22 теста, представленных ниже в таблице 20.

Таблица 20 – Тесты для сервиса управления документами и правами

№	Название теста	Сценарий
1	CreateDocument_ValidRequest_Success	Успешное создание документа: все участники существуют, роли корректны, транзакция завершена
2	CreateDocument_UserNotFound_ThrowsUserNotFoundException	Ошибка при указании несуществующего пользователя в списке участников
3	DeleteDocument_Creator_Success	Создатель документа может удалить документ
№	Название теста	Сценарий
4	DeleteDocument_NonCreator_ThrowsPermissionDeniedException	Попытка удаления документа пользователем с ролью Editor или Viewer
5	DeleteDocument_DocumentNotFound_ThrowsDocumentNotFoundException	Ошибка при удалении несуществующего документа

Продолжение таблицы 20

№	Название теста	Сценарий
6	GetDocumentsByUserId_ReturnsMappedDocuments	Пользователь получает список своих документов
7	GetDocumentDetails_DocumentExists_ReturnsDetails	Получение полной информации о документе
8	GetDocumentDetails_DocumentNotFound_ThrowsDocumentNotFoundException	Ошибка при запросе деталей несуществующего документа
9	AddUsersToDocument_ValidRequest_Succeeds	Успешное добавление новых пользователей в документ с корректными ролями
10	AddUsersToDocument_UserAlreadyInDocument_ThrowsUserAlreadyInDocumentException	Ошибка при попытке добавить пользователя, уже находящегося в документе
11	RenameDocument_Creator_Success	Создатель документа может изменить его имя
12	RenameDocument_Editor_Success	Редактор может изменить имя документа
13	RenameDocument_Viewer_ThrowsPermissionDeniedException	Пользователь с ролью Viewer не может переименовывать документ
14	RemoveUserFromDocument_CreatorRemovesParticipant_Success	Создатель документа успешно удаляет другого участника
15	RemoveUserFromDocument_NonCreator_ThrowsPermissionDeniedException	Попытка удаления участника пользователем без роли Creator
16	RemoveUserFromDocument_DocumentNotFound_ThrowsDocumentNotFoundException	Ошибка при удалении участника из несуществующего документа
17	RemoveUserFromDocument_UserNotInDocument_ThrowsDocumentParticipantNotFoundException	Ошибка при удалении пользователя, не являющегося участником документа
18	ChangeUserRoleInDocument_CreatorChangesRole_Success	Создатель успешно изменяет роль участника
19	ChangeUserRoleInDocument_NonCreator_ThrowsPermissionDeniedException	Пользователь без роли Creator не может менять роли
20	ChangeUserRoleInDocument_UserNotInDocument_ThrowsDocumentParticipantNotFoundException	Ошибка при смене роли несуществующего участника
21	ChangeUserRoleInDocument_InvalidRole_ThrowsArgumentException	Передача неверного строкового значения роли
22	GetDocumentParticipantsByUserId_ReturnsMappedParticipants	Пользователь получает список всех документов, в которых он участвует, с корректными ролями

Результат выполнения тестов представлен ниже на рисунке 31.

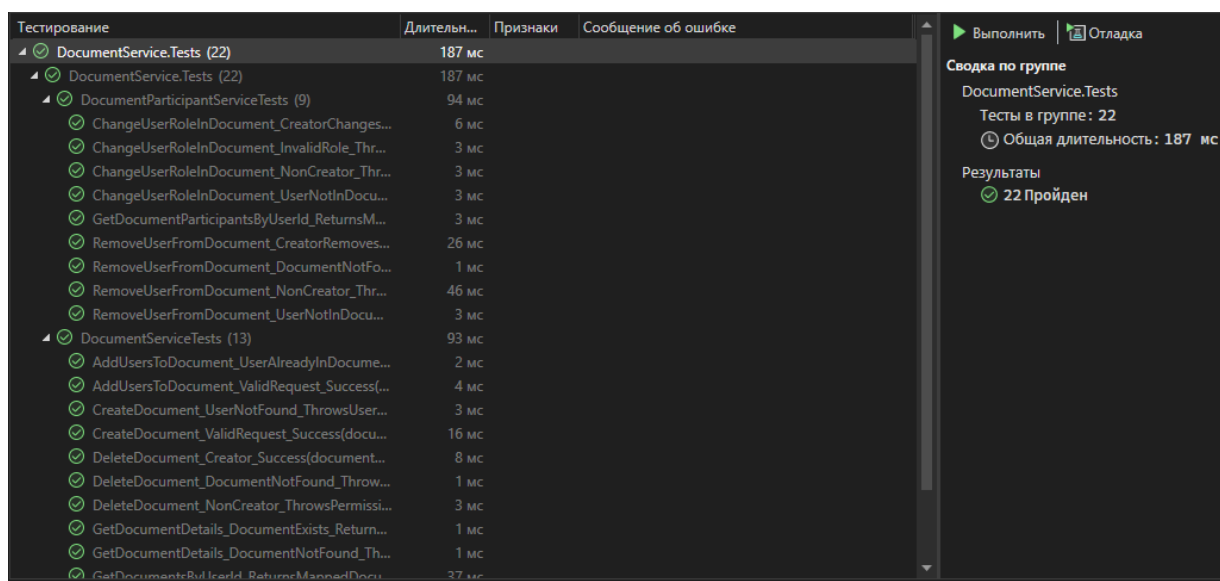


Рисунок 31 – Тестирование сервиса управления документами и ролями

Для сервиса управления блоками было разработано 11 тестов, представленных ниже в таблице 21.

Таблица 21 – Тесты для сервиса управления блоками

№	Название теста	Сценарий
1	SendBlock_ValidRequest_Success	Участник документа с ролью «Editor» или «Creator» успешно отправляет текстовый блок
2	SendBlock_DocumentNotFound_ThrowsDocumentNotFoundException	Ошибка при попытке отправить блок в несуществующий документ
3	SendBlock_UserNotInDocument_ThrowsDocumentParticipantNotFoundException	Ошибка при отправке пользователем, не являющимся участником документа
4	SendBlock_UserWithoutWriteAccess_ThrowsPermissionDeniedException	Пользователь с ролью «Viewer» (или ниже) не имеет права на запись
5	GetBlocksByDocument_ValidRequest_ReturnsBlocks	Участник документа получает список всех блоков с фильтрацией по дате
6	EditBlock_EditorEditsOtherBlock_Success	Пользователь с ролью «Editor» или «Creator» может редактировать любой блок в документе
7	EditBlock_BlockNotFound_ThrowsBlockNotFoundException	Ошибка при попытке отредактировать несуществующий блок
8	SendBlockImage_ValidRequest_Success	Участник с правом записи успешно загружает изображение к существующему блоку
9	SendBlockImage_BlockNotFound_ThrowsBlockNotFoundException	Ошибка при привязке изображения к несуществующему блоку
10	DeleteBlockImage_EditorDeletesImage_Success	Участник документа успешно удаляет изображение
11	GetBlockImagesByBlock_ValidRequest_ReturnsImages	Участник документа получает список всех изображений, привязанных к блоку

Результат выполнения тестов представлен ниже на рисунке 32.

Тестирование	Длительность	Признаки	Сообщение об ошибке
BlockService.Tests (11)	122 мс		
BlockService.Tests (11)	122 мс		
BlockImageServiceTests (4)	55 мс		
DeleteBlockImage_EditorDeletesImage_Succe...	32 мс		
GetBlockImagesByBlock_ValidRequest_Return...	6 мс		
SendBlockImage_BlockNotFound_ThrowsBloc...	1 мс		
SendBlockImage_ValidRequest_Success(image...	16 мс		
BlockServiceTests (7)	67 мс		
EditBlock_BlockNotFound_ThrowsBlockNotFo...	1 мс		
EditBlock_EditorEditsOtherBlock_Success(bloc...	8 мс		
GetBlocksByDocument_ValidRequest_Returns...	9 мс		
SendBlock_DocumentNotFound_ThrowsDocu...	1 мс		
SendBlock_UserNotInDocument_ThrowsDocu...	39 мс		
SendBlock_UserWithoutWriteAccess_ThrowsP...	1 мс		
SendBlock_ValidRequest_Success(blockReposi...	8 мс		

Выполнить | Отладка

Сводка по группе

BlockService.Tests

Тесты в группе: 11

Общая длительность: 122 мс

Результаты

11 Пройден

Рисунок 32 – Тестирование сервиса управления блоками

В результате проведенного модульного тестирования подтверждена корректность работы всех сервисов в изолированной среде, а также отсутствие критических багов в логике приложения. Все внешние зависимости, включая запросы к базе данных и сторонним API, были успешно изолированы с помощью фиктивных объектов.

4.2 Оценка качества кода

4.3.1 Метрики Холстеда

В качестве объекта для расчета метрик Холстеда была выбрана функция `CreateDocument`. Реализованная в ней логика обеспечивает создание документов и распределение прав доступа между приглашенными участниками. С полным текстом функции можно ознакомиться в листинге В.6 приложения В.

Для расчета параметров сложности необходимо подсчитать число уникальных параметров ($n1$) и операндов ($n2$), а также их общий объем использования в коде ($N1$, $N2$). Операторы функции `CreateDocument` и число их вхождений приведены в таблице 22.

Таблица 22 – Операторы функции `CreateDocument`

№	Оператор	Число вхождений
1	<code>async</code>	1
2	<code>await</code>	4
3	<code>foreach</code>	1
4	<code>for</code>	1
5	<code>if</code>	2
6	<code>throw</code>	2
7	<code>try</code>	1
8	<code>catch</code>	1
9	<code>using</code>	1
10	<code>return</code>	1
11	<code>new</code>	2
12	<code>=</code>	14
13	<code>==</code>	1
14	<code><</code>	2
15	<code>&&</code>	1
16	<code>?</code>	1
17	<code>:</code>	1
18	<code>++</code>	1
19	<code>.</code>	31
20	<code>(</code>	21
21	<code>)</code>	21
22	<code>{</code>	7
23	<code>}</code>	7
24	<code>;</code>	20
25	<code>,</code>	26
26	<code>CreateDocument</code>	1
27	<code>BeginTransaction</code>	1
28	<code>IsUserExistsById</code>	1
29	<code>MapToDb</code>	2
30	<code>CreateDocument (repository)</code>	1
31	<code>CreateDocumentParticipants</code>	1
32	<code>Commit</code>	1
33	<code>Rollback</code>	1

Продолжение таблицы 22

№	Оператор	Число вхождений
34	Add	2
35	TryParse	1
36	MapToDomain	1
	36	183

Операнды функции CreateDocument и число их вхождений приведены в таблице 23.

Таблица 23 – Операнды функции CreateDocument

№	Операнд	Число вхождений
1	request	13
2	UserIds	4
3	CreatorId	2
4	id	5
5	userRepository	2
6	documentRepository	2
7	transaction	6
8	dbDocument	3
9	documentId	4
10	documentParticipants	4
11	DbDocumentParticipant	1
12	userId	5
13	roleStr	2
14	Roles	2
15	parsed	1
16	role	2
17	roleEnum	4
18	DocumentRole	3
19	User	1
20	Creator	1
21	Exception	1
22	i	6
23	Count	2
24	Document	1
25	CreateDocumentRequest	1
	25	87

Характеристики программы, рассчитанные по формулам из метрик Холстеда приведены в таблице 24.

Таблица 24 – Результаты вычислений по метрике Холстеда

Значения	Результат
Словарь программы (n)	61
Длина программы (N)	270
Теоретическая длина программы (N')	302
Объем программы (V)	1485
Теоретический объем программы (V')	1661
Уровень качества программирования (L)	1,11

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2 = 36 + 25 = 61.$$

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2 = 183 + 87 = 270.$$

Теоритическая длина программы рассчитывается по формуле:

$$N' = n_1 \times \log_2 n_1 + n_2 \times \log_2 n_2 = 36 \times 5.17 + 25 \times 4.64 = 302.$$

Объем программы рассчитывается по формуле:

$$V = N \times \log_2 n = 270 \times 5.93 = 1485.$$

Теоритический объем программы рассчитывается по формуле:

$$V' = N' \times \log_2 n = 302 \times 5.5 = 1661.$$

Уровень качества программирования рассчитывается по формуле:

$$L = \frac{V'}{V} = \frac{1661}{1485} = 1.11.$$

На основе проведенных расчетов можно сделать вывод, что характеристики приложения не превышают критических значений. Полученные значения подтверждают архитектуры исходного кода и отсутствие избыточного загромождения алгоритмов.

Заключение

В рамках выполнения выпускной квалификационной работы спроектированы и разработаны программные средства для совместного редактирования документов.

На этапе анализа предметной области описаны основные понятия и описаны участники бизнес-процесса и их функциональные возможности. Проведен анализ существующих информационных систем аналогичного назначения, таких как Google Docks, Notion, Microsoft Office Online и Quip. На основе проведенного анализа определены сильные и слабые стороны аналогов.

Проведен детальный расчет финансово-экономических показателей. Итоговый бюджет необходимый для успешной реализации программных средств совместного редактирования документов составил 263 675 рублей при суммарной затрате рабочего времени в размере 392 часов, при этом жизненный цикл разработки охватил интервал с 20 января по 31 мая текущего года.

На начальном этапе анализа четко определены задачи, а также выявлены основные функциональные требования, предъявляемые к программному средству. Для визуализации логики взаимодействия компонентов и бизнес-логики было осуществлено моделирование прецедентов.

На стадии проектирования сформирована инфологическая схема базы данных, определена общая архитектурная концепция и принципы межмодульного взаимодействия. Также определены ключевые требования к интерфейсу веб-приложения, необходимые для удобного использования конечным пользователем.

Практическая реализация включала в себя создание серверной и клиентской части приложения развернута реляционная база данных под управлением СУБД PostgreSQL, а также разработаны три сервиса на платформе ASP.NET Core в среде Visual Studio 2022. Клиентская часть реализовывалась в среде Visual Studio Code с использованием строго типизированного языка TypeScript и фреймворка React, а стилизация проводилась помощью инструмента

Tailwind CSS. Интеграция и автоматизация запуска всех компонентов обеспечивается за счет технологии контейнеризации Docker.

На этапе контроля качества созданного веб-приложения был выполнен комплекс мероприятий, включающий проведение функционального и модульного тестирования, а также проведение анализа кода с расчетом метрик Холстеда. Полученные результаты проверки подтвердили полную техническую исправность приложения, а количественная оценка метрических параметров обосновала оптимальную сложность архитектуры и высокий потенциал сопровождаемости.

Таким образом в результате выполнения выпускной квалификационной работы разработаны программные средства совместного редактирования документов, позволяющие структурировать текст по отдельным независимым блокам и отказаться от привычного деления на страницы, быстро добавлять и удалять ненужные части, а также приглашать в документ других участников и настраивать для них разные роли и права доступа.

Список используемых источников

1. Таненбаум, Э. С. Распределенные системы: принципы и парадигмы / Э. С. Таненбаум, М. ван Стеен. – СПб.: Питер, 2003. – 877 с.
2. WebSocket – основные концепции // WebSocket [Электронный ресурс]. URL: <https://websocket.org/> (дата обращения: 20.02.26);
3. UML – диаграмма вариантов использования // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 24.02.2026).
4. Архитектура приложений и интеграции // Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/itq_group/articles/705598 (дата обращения: 28.02.2026)
5. Ричардс, М. Фундаментальный подход к программным архитектурам: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд. – СПб.: Питер. 2023. – 448с.
6. Трофимов, В. В. Управление проектами: Учебное пособие / В. В. Трофимов. – СПб.: Изд-во СПбГЭУ, 2019. – 174 с.
7. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс]. URL: <https://www.unisender.com/ru/blog/что-такое-диаграмма-ganta-i-kak-ee-sdelat/> (Дата обращения: 10.03.2026).
8. ACID // Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/ACID/> (дата обращения: 15.03.2026).
9. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 20.03.2026).
10. Просто о сложном. Концептуальная модель. // MLKV [Электронный ресурс]. URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 27.03.2026).
11. Моделирование данных: обзор // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения: 29.03.2026)

12. Диаграмма классов. Унифицированный язык моделирования (UML) // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/> (дата обращения: 11.04.2026).

13. Процесс разработки пользовательского интерфейса: 6 простых шагов для создания отличного пользовательского интерфейса // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/ui-design-process-everything-you-need-to-know/> (дата обращения: 20.04.2026).

14. Проектирование и разработка баз данных. Руководство для разработчиков баз данных. Концептуальная, логическая и физическая модели данных // Официальная документация Visual Paradigm [Электронный ресурс]. URL: <https://circle.visual-paradigm.com/docs/database-design-engineering/database-designers-guide/conceptual-logical-and-physical-data-model/> (дата обращения: 27.04.2026).

15. Зеляпугин, И. А. Программные средства совместного редактирования документов // GitHub [Электронный ресурс]. URL: <https://github.com/Ivan-Zelyapugin/MicroServices> (дата обращения: 29.04.2026).

16. Учебное пособие по тестированию программного обеспечения // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-testing-tutorial/> (дата обращения: 12.05.2026).

17. Крауч, Р. Тестирование ПО. Основы, методы и термины / Р. Крауч, В. Д. Влиет. – СПб.: Питер, 2003. – 256 с.

18. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – Минск.: Четыре четверти, 2016. – 300 с.

19. Значения метрик кода // Официальная документация Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата обращения: 20.05.2026).

20. Метрики как средства управления качеством // OSP [Электронный ресурс] URL: <https://www.osp.ru/os/2009/08/10748698> (Дата обращения: 21.05.2026).

Приложение А
(справочное)
Глоссарий

БД (База данных) – структурированный массив информации, описывающий текущие характеристики объектов, их свойства и взаимосвязи в рамках конкретной предметной области.

СУБД (Система управления базами данных) – комплекс специализированного программного обеспечения и языковых инструментов, предназначенных для организации, администрирования, совместного использования и защиты хранилищ информации.

REST (Representational State Transfer) – архитектурный стиль проектирования распределенных систем, регламентирующий свод правил и ограничений для построения масштабируемых веб-служб.

API (Application Programming Interface) – стандартизированный интерфейс взаимодействия программ, представляющий собой готовый набор классов, процедур, системных констант и структур, с помощью которых сторонние приложения могут обращаться к функциям операционной системы или сервиса.

Фреймворк – платформа, определяющая каркас архитектуры и диктующая единые правила проектирования, модификации и интеграции компонентов при создании программного обеспечения.

Web-приложение – распределенный программный комплекс, построенный по клиент-серверной модели, вычислительная логика которого развернута на удаленном сервере, а взаимодействие с пользователем осуществляется через веб-браузер посредством передачи данных по протоколам семейства HTTP.

.NET – универсальная экосистема разработки, объединяющая в себе, компилятор, языки программирования, среды выполнения и кроссплатформенные библиотеки для проектирования широкого спектра программного обеспечения.

ASP.NET Core – специализированное расширение программного стека .NET, содержащее инструменты, ориентированные на создание высокопроизводительных веб-служб и облачных приложений.

Приложение Б
(рекомендуемое)
Диаграммы классов

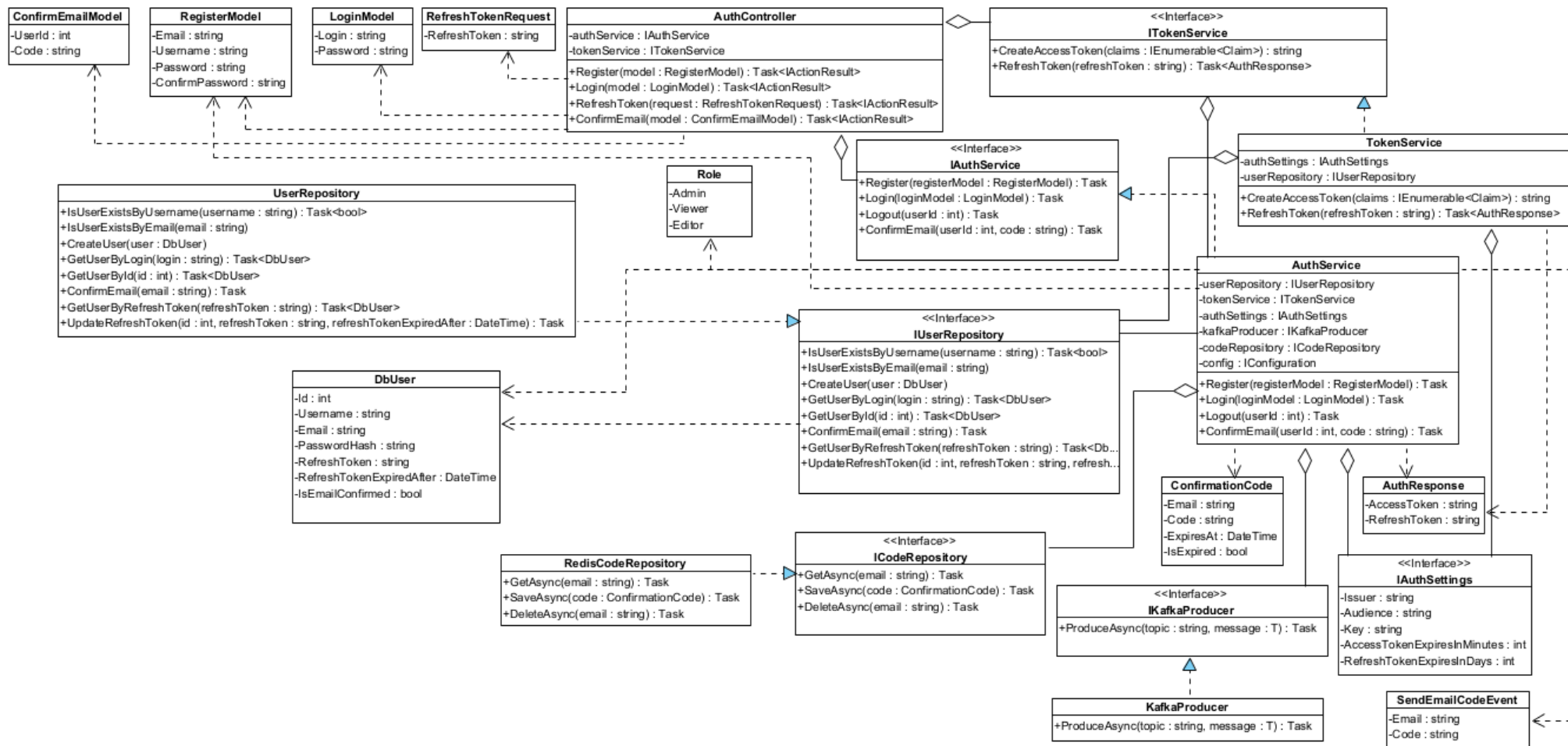


Рисунок Б.1 – Диаграмма классов AuthService

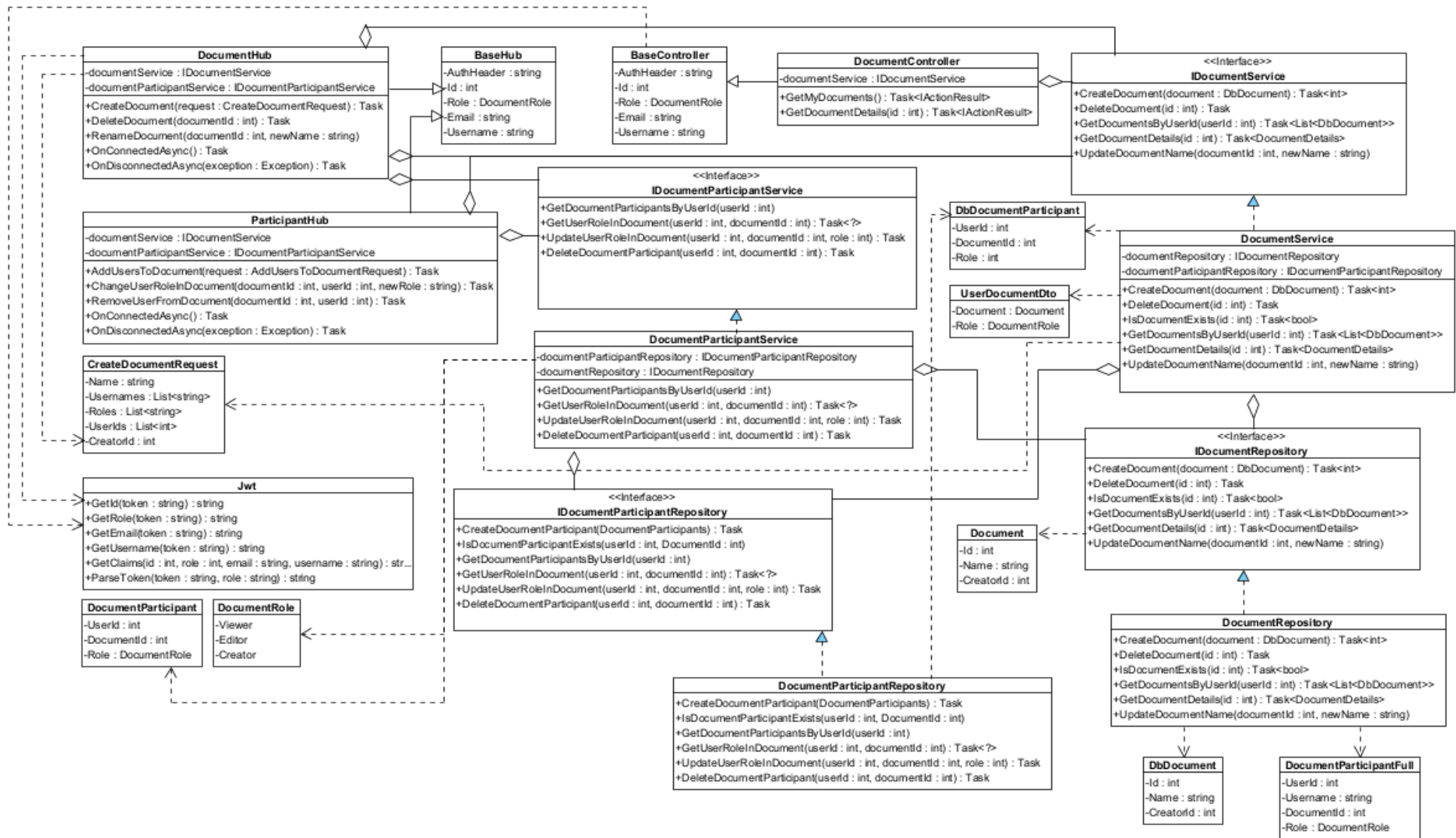


Рисунок Б.2 – Диаграмма классов DocumentService

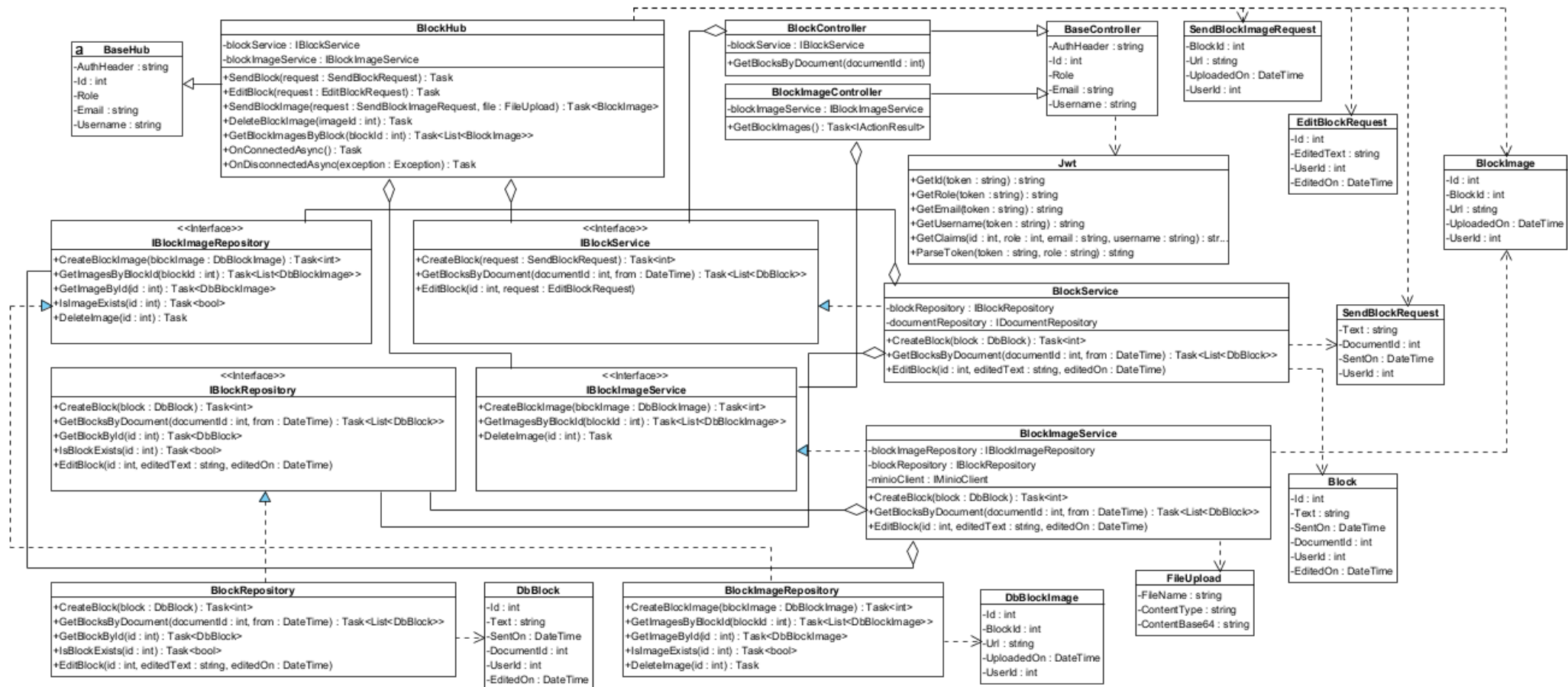


Рисунок Б.3 – Диаграмма классов BlockService

Приложение В
(обязательное)
Листинг серверной части

Листинг В.1 – Файл BaseController.cs

```
using DocumentService.Common;
using DocumentService.Models.Permission;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DocumentService.Api.Controllers
{
    [Authorize]
    [ApiController]
    public class BaseController : ControllerBase
    {
        private string AuthHeader =>
HttpContext.Request.Headers.Authorization.ToString();

        protected int Id => int.Parse(Jwt.GetId(AuthHeader));
        protected DocumentRole Role =>
Enum.Parse<DocumentRole>(Jwt.GetRole(AuthHeader));
        protected string Email => Jwt.GetEmail(AuthHeader);
        protected string Username => Jwt.GetUsername(AuthHeader);
    }
}
```

Листинг В.2 – Файл DocumentController.cs

```
using DocumentService.DataAccess.Repositories.Interfaces;
using DocumentService.Models.Document;
using DocumentService.Services.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace DocumentService.Api.Controllers
{
    [Route("api/[controller]")]
    public class DocumentController(IDocumentService
documentService, IUserRepository userRepository) : BaseController
    {

        [HttpGet("my")]
        public async Task<IActionResult> GetMyDocuments()
        {
            return Ok(await
documentService.GetDocumentsByUserId(Id));
        }

        [HttpGet("me")]
        public IActionResult GetCurrentUser()
        {
            return Ok(new
            {
                id = Id,
                username = Username,
                email = Email,
            });
        }
    }
}
```

```

        role = Role.ToString()
    });
}

[HttpGet("{id:int}")]
public async Task<IActionResult> GetDocumentDetails(int
id)
{
    var details = await
documentService.GetDocumentDetails(id);

    if (details == null)
        return NotFound();

    var result = new DocumentInfoDto
    {
        Id = details.Id,
        Name = details.Name,
        CreatorUsername = details.Creator?.Username,
        Users = details.Participants
        .Select(p => new DocumentUserDto
        {
            UserId = p.UserId,
            Username = p.Username,
            Role = p.Role.ToString()
        })
        .ToList()
    };

    return Ok(result);
}
}
}

```

Листинг В.3 – Файл DocumentHub.cs

```

using DocumentService.Api.HubMetrics;
using DocumentService.DataAccess.Models;
using DocumentService.DataAccess.Repositories.Interfaces;
using DocumentService.Models.Document;
using DocumentService.Services.Interfaces;
using Microsoft.AspNetCore.SignalR;
using Prometheus;

namespace DocumentService.Api.Hubs
{
    public class DocumentHub
        (IConnectionTracker connectionTracker,
        IDocumentService documentService,
        IDocumentParticipantService documentParticipantService,
        IUserRepository userRepository
        ) : BaseHub
    {

```

```

        public async Task CreateDocument(CreateDocumentRequest
request)
        {
            using
(DocumentHubMetrics.CreateDocumentDuration.NewTimer())
            {
                try
                {
                    request.CreatorId = Id;

                    var users = request.Usernames != null &&
request.Usernames.Any()
                        ? await
userRepository.GetUsersByUsernamesAsync(request.Usernames)
                        : new List<DbUser>();

                    request.UserIds = users?.Select(u =>
u.Id).ToList() ?? new List<int>();

                    var document = await
documentService.CreateDocument(request);

                    var connectionIds = connectionTracker
                        .SelectConnectionIds(request.UserIds)
                        .Append(Context.ConnectionId);

                    await
Task.WhenAll(connectionIds.Select(connectionId =>
                        Groups.AddToGroupAsync(connectionId,
$"Document{document.Id}")));

                    await
Clients.Caller.SendAsync("DocumentCreated", document);
                    await Clients.Group($"Document{document.Id}")
                        .SendAsync("AddedToDocument",
request.UserIds);

                    DocumentHubMetrics.DocumentsCreated.Inc();
                }
                catch (Exception e)
                {
                    throw new HubException(e.Message);
                }
            }
        }

        public async Task DeleteDocument(int documentId)
        {
            using
(DocumentHubMetrics.DeleteDocumentDuration.NewTimer())
            {
                try
                {

```

```

        await
documentService.DeleteDocument(documentId, Id);
        await
Clients.Group($"Document{documentId}").SendAsync("DocumentDeleted"
, documentId);

        var connectionIds =
connectionTracker.SelectConnectionIds(new List<int> { Id });
        await
Task.WhenAll(connectionIds.Select(connectionId =>
Groups.RemoveFromGroupAsync(connectionId,
$"Document{documentId}")));

        DocumentHubMetrics.DocumentsDeleted.Inc();
    }
    catch (Exception e)
    {
        throw new HubException(e.Message);
    }
}

public async Task RenameDocument(int documentId, string
newName)
{
    using
(DocumentHubMetrics.RenameDocumentDuration.NewTimer())
    {
        try
        {
            await
documentService.RenameDocument(documentId, newName, Id);

            await Clients.Group($"Document{documentId}")
                .SendAsync("DocumentRenamed", documentId,
newName);

            DocumentHubMetrics.DocumentsRenamed.Inc();
        }
        catch (Exception e)
        {
            throw new HubException(e.Message);
        }
    }
}

public override async Task OnConnectedAsync()
{
    connectionTracker.TrackConnection(Context.ConnectionId
, Id);

    DocumentHubMetrics.ActiveConnections.Inc();

```

```

        var documentParticipants = await
documentParticipantService.GetDocumentParticipantsByUserId(Id);
        var documentIds = documentParticipants.Select(x =>
x.DocumentId);
        await Task.WhenAll(documentIds.Select(documentId =>
            Groups.AddToGroupAsync(Context.ConnectionId,
$"Document{documentId}")));

        await base.OnConnectedAsync();
    }

    public override async Task OnDisconnectedAsync(Exception
exception)
    {
        connectionTracker.UntrackConnection(Context.Connection
Id);

        DocumentHubMetrics.ActiveConnections.Dec();

        await base.OnDisconnectedAsync(exception);
    }
}

```

Листинг В.4 – Файл ParticipantHub.cs

```

using DocumentService.Api.HubMetrics;
using DocumentService.DataAccess.Repositories.Interfaces;
using DocumentService.Models.Document;
using DocumentService.Services.Interfaces;
using Microsoft.AspNetCore.SignalR;
using Prometheus;

namespace DocumentService.Api.Hubs
{
    public class ParticipantHub(IConnectionTracker
connectionTracker,
        IDocumentService documentService,
        IDocumentParticipantService documentParticipantService,
        IUserRepository userRepository
    ) : BaseHub
    {
        public async Task
AddUsersToDocument(AddUsersToDocumentRequest request)
        {
            using
(ParticipantHubMetrics.AddUsersDuration.NewTimer())
            {
                try
                {
                    request.RequestingUserId = Id;

```

```

        var users = await
userRepository.GetUsersByUsernamesAsync(request.Usernames);
        if (users == null || !users.Any())
        {
            throw new HubException("Не удалось найти
пользователей по указанным email");
        }

        request.UserIds = users.Select(u =>
u.Id).ToList();

        await
documentService.AddUsersToDocument(request);

        var connectionIds =
connectionTracker.SelectConnectionIds(request.UserIds);
        await
Task.WhenAll(connectionIds.Select(connectionId =>
Groups.AddToGroupAsync(connectionId,
$"Document{request.DocumentId}")));

        await
Clients.Group($"Document{request.DocumentId}")
            .SendAsync("AddedToDocument",
request.UserIds);

        ParticipantHubMetrics.UsersAddedToDocument.Inc
();
    }
    catch (Exception e)
    {
        throw new HubException(e.Message);
    }
}

public async Task ChangeUserRoleInDocument(int documentId,
int userId, string newRole)
{
    using
(ParticipantHubMetrics.ChangeUserRoleDuration.NewTimer())
    {
        try
        {
            await
documentParticipantService.ChangeUserRoleInDocument(documentId,
userId, newRole, Id);

            await Clients.Group($"Document{documentId}")
                .SendAsync("UserRoleChanged", documentId,
userId, newRole);

            ParticipantHubMetrics.UserRolesChanged.Inc();

```

```

        }
        catch (Exception e)
        {
            throw new HubException(e.Message);
        }
    }
}

public async Task RemoveUserFromDocument(int documentId,
int userId)
{
    using
(ParticipantHubMetrics.RemoveUserDuration.NewTimer())
    {
        try
        {
            await
documentParticipantService.RemoveUserFromDocument(documentId,
userId, Id);

            var connectionIds =
connectionTracker.SelectConnectionIds(new List<int> { userId });
            await
Task.WhenAll(connectionIds.Select(connectionId =>
Groups.RemoveFromGroupAsync(connectionId,
$"Document{documentId}")));

            await Clients.Group($"Document{documentId}")
                .SendAsync("UserRemoved", documentId,
userId);

            ParticipantHubMetrics.UsersRemovedFromDocument
.Inc();
        }
        catch (Exception e)
        {
            throw new HubException(e.Message);
        }
    }
}

public override async Task OnConnectedAsync()
{
    connectionTracker.TrackConnection(Context.ConnectionId
, Id);

    ParticipantHubMetrics.ActiveConnections.Inc();

    var documentParticipants = await
documentParticipantService.GetDocumentParticipantsById(Id);
    var documentIds = documentParticipants.Select(x =>
x.DocumentId);
    await Task.WhenAll(documentIds.Select(documentId =>

```

```

        Groups.AddToGroupAsync(Context.ConnectionId,
$"Document{documentId}"));

        await base.OnConnectedAsync();
    }

    public override async Task OnDisconnectedAsync(Exception
exception)
    {
        connectionTracker.UntrackConnection(Context.Connection
Id);

        ParticipantHubMetrics.ActiveConnections.Dec();

        await base.OnDisconnectedAsync(exception);
    }
}

```

Листинг В.5 – Файл BaseController.cs

```

using BlockService.Common;
using BlockService.Models.Permission;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace BlockService.Api.Controllers
{
    [Authorize]
    [ApiController]
    public class BaseController : ControllerBase
    {
        private string AuthHeader =>
HttpContext.Request.Headers.Authorization.ToString();

        protected int Id => int.Parse(Jwt.GetId(AuthHeader));
        protected DocumentRole Role =>
Enum.Parse<DocumentRole>(Jwt.GetRole(AuthHeader));
        protected string Email => Jwt.GetEmail(AuthHeader);
        protected string Username => Jwt.GetUsername(AuthHeader);
    }
}

```

Листинг В.6 – Файл DocumentService

```

using DocumentService.DataAccess.Models;
using DocumentService.DataAccess.Repositories.Interfaces;
using DocumentService.Models.Document;
using DocumentService.Models.Permission;
using DocumentService.Services.Exceptions;
using DocumentService.Services.Interfaces;
using DocumentService.Services.Mappers;
namespace DocumentService.Services
{

```

```

public class DocumentService(
    IDocumentRepository documentRepository,
    IUserRepository userRepository,
    IDocumentParticipantRepository
documentParticipantRepository
) : IDocumentService
{
    public async Task<Document>
CreateDocument(CreateDocumentRequest request)
    {
        request.UserIds.Add(request.CreatorId);
        foreach (var id in request.UserIds)
        {
            if (!await userRepository.IsUserExistsById(id))
            {
                throw new UserNotFoundException(id);
            }
        }
        using var transaction =
documentRepository.BeginTransaction();
        try
        {
            var dbDocument = request.MapToDb();
            var documentId = await
documentRepository.CreateDocument(request.MapToDb(), transaction);
            var documentParticipants = new
List<DbDocumentParticipant>();
            for (int i = 0; i < request.UserIds.Count; i++)
            {
                var userId = request.UserIds[i];
                var roleStr = request.Roles != null && i <
request.Roles.Count ? request.Roles[i] : "User";
                var parsed =
Enum.TryParse<DocumentRole>(roleStr, true, out var role);
                var roleEnum = parsed ? role :
DocumentRole.User;
                if (userId == request.CreatorId)
                    roleEnum = DocumentRole.Creator;
                documentParticipants.Add(new
DbDocumentParticipant
                {
                    UserId = userId,
                    DocumentId = documentId,
                    Role = (int)roleEnum
                });
            }
            await
documentParticipantRepository.CreateDocumentParticipants(documentP
articipants, transaction);
            transaction.Commit();
            dbDocument.Id = documentId;
            return dbDocument.MapToDomain();
        }
    }
}

```

```

        catch (Exception)
        {
            transaction.Rollback();
            throw;
        }
    }

    public async Task DeleteDocument(int documentId, int
requestingUserId)
    {
        if (!await
documentRepository.IsDocumentExists(documentId))
        {
            throw new DocumentNotFoundException(documentId);
        }
        var participant = await
documentParticipantRepository.IsDocumentParticipantExists(requesti
ngUserId, documentId);
        if (!participant)
        {
            throw new
DocumentParticipantNotFoundException(requestingUserId,
documentId);
        }
        var roleInt = await
documentParticipantRepository.GetUserRoleInDocument(requestingUser
Id, documentId);
        if (roleInt is null)
            throw new
DocumentParticipantNotFoundException(requestingUserId,
documentId);
        var role = (DocumentRole)roleInt;
        if (role != DocumentRole.Creator)
            throw new PermissionDeniedException("Удаление
документа разрешено только его создателю.");
        using var transaction =
documentRepository.BeginTransaction();
        try
        {
            await
documentRepository.DeleteDocument(documentId, transaction);
            transaction.Commit();
        }
        catch (Exception)
        {
            transaction.Rollback();
            throw;
        }
    }

    public async Task<List<UserDocumentDto>>
GetDocumentsByUserId(int userId)
    {
        var dbDocuments = await
documentRepository.GetDocumentsByUserId(userId);

```

```

        var result = new List<UserDocumentDto>();
        foreach (var dbDoc in dbDocuments)
        {
            var roleInt = await
documentParticipantRepository.GetUserRoleInDocument(userId,
dbDoc.Id);

            if (roleInt == null)
                continue;
            var role = (DocumentRole)roleInt;
            result.Add(new UserDocumentDto
            {
                Document = dbDoc.MapToDomain(),
                Role = role
            });
        }
        return result;
    }

    public async Task<DocumentDetails> GetDocumentDetails(int
id)
    {
        var documentDetails = await
documentRepository.GetDocumentDetails(id);
        if (documentDetails == null)
        {
            throw new DocumentNotFoundException(id);
        }
        return documentDetails;
    }

    public async Task
AddUsersToDocument(AddUsersToDocumentRequest request)
    {
        if (!await
documentRepository.IsDocumentExists(request.DocumentId))
            throw new
DocumentNotFoundException(request.DocumentId);
        if (!await
documentParticipantRepository.IsDocumentParticipantExists(request.
RequestingUserId, request.DocumentId))
            throw new
DocumentParticipantNotFoundException(request.RequestingUserId,
request.DocumentId);
        var participants = new List<DbDocumentParticipant>();
        for (int i = 0; i < request.UserIds.Count; i++)
        {
            var userId = request.UserIds[i];
            if (!await
userRepository.IsUserExistsById(userId))
                throw new UserNotFoundException(userId);
            if (await
documentParticipantRepository.IsDocumentParticipantExists(userId,
request.DocumentId))
                throw new
UserAlreadyInDocumentException(userId, request.DocumentId);

```

```

        string roleStr = request.Roles != null && i <
request.Roles.Count ? request.Roles[i] : "User";
        var parsed = Enum.TryParse<DocumentRole>(roleStr,
true, out var role);
        var roleEnum = parsed ? role : DocumentRole.User;
        participants.Add(new DbDocumentParticipant
        {
            DocumentId = request.DocumentId,
            UserId = userId,
            Role = (int)roleEnum
        });
    }
    await
documentParticipantRepository.CreateDocumentParticipants(participa
nts);
}
    public async Task RenameDocument(int documentId, string
newName, int requestingUserId)
    {
        if (!await
documentRepository.IsDocumentExists(documentId))
            throw new DocumentNotFoundException(documentId);
        var participantExists = await
documentParticipantRepository.IsDocumentParticipantExists(requesti
ngUserId, documentId);
        if (!participantExists)
            throw new
DocumentParticipantNotFoundException(requestingUserId,
documentId);
        var roleInt = await
documentParticipantRepository.GetUserRoleInDocument(requestingUser
Id, documentId);
        if (roleInt is null)
            throw new
DocumentParticipantNotFoundException(requestingUserId,
documentId);
        var role = (DocumentRole)roleInt;
        if (role != DocumentRole.Creator && role !=
DocumentRole.Editor)
            throw new PermissionDeniedException("Смена имени
документа разрешена только создателю или редактору.");
        using var transaction =
documentRepository.BeginTransaction();
        try
        {
            await
documentRepository.UpdateDocumentName(documentId, newName,
transaction);
            transaction.Commit();
        }
        catch
        {
            transaction.Rollback();

```

```
        throw;  
    }  
}  
}
```

Приложение Г
(обязательное)
Листинг клиентской части

Листинг Г.1 – Файл signalr.ts

```
import { hubFactory } from './signalrHubFactory';
// Создаем хабы
export const documentHub = hubFactory.createHub('/documenthub');
export const blockHub = hubFactory.createHub('/blockhub');
export const participantHub =
hubFactory.createHub('/participanthub');
export const voiceHub = hubFactory.createHub('/voicehub');
// Стартеры хабов
export const startDocumentHub = () =>
hubFactory.startHub('/documenthub');
export const startBlockHub = () =>
hubFactory.startHub('/blockhub');
export const startParticipantHub = () =>
hubFactory.startHub('/participanthub');
export const startVoiceHub = () =>
hubFactory.startHub('/voicehub');
// Отправка сообщений
export const sendDocumentMessage = (method: string, args: any[])
=>
    hubFactory.sendMessage('/documenthub', method, args);
export const sendBlockMessage = (method: string, args: any[]) =>
    hubFactory.sendMessage('/blockhub', method, args);
export const sendParticipantMessage = (method: string, args:
any[]) =>
    hubFactory.sendMessage('/participanthub', method, args);
export const sendVoiceMessage = (method: string, args: any[]) =>
    hubFactory.sendMessage('/voicehub', method, args);
```

Листинг Г.2 – Файл signalrHubFactory.ts

```
import * as signalR from '@microsoft/signalr';
interface HubInstance {
    connection: signalR.HubConnection;
    isConnecting: boolean;
}
class SignalRHubFactory {
    private hubs: Map<string, HubInstance> = new Map();

    public createHub(url: string, tokenKey = 'accessToken'):
HubInstance {
        if (this.hubs.has(url)) {
            return this.hubs.get(url)!;
        }
        const connection = new signalR.HubConnectionBuilder()
            .withUrl(url, {
                accessTokenFactory: () => localStorage.getItem(tokenKey)
|| '',
            })
            .withAutomaticReconnect([0, 1000, 3000, 5000, 10000])
            .configureLogging(signalR.LogLevel.Information)
            .build();
```

```

    const instance: HubInstance = { connection, isConnecting:
false };
    connection.onclose(() => console.warn(`[SignalR] ${url}
closed`));
    connection.onreconnecting(() => console.log(`[SignalR] ${url}
reconnecting...`));
    connection.onreconnected(() => console.log(`[SignalR] ${url}
reconnected`));
    this.hubs.set(url, instance);
    return instance;
}
public async startHub(url: string) {
    const hub = this.hubs.get(url);
    if (!hub) throw new Error(`Hub ${url} not found`);
    if (hub.isConnecting || hub.connection.state !==
signalR.HubConnectionState.Disconnected) return;
    hub.isConnecting = true;
    try {
        await hub.connection.start();
        console.log(`[SignalR] ${url} connected`);
    } catch (err) {
        console.error(`[SignalR] ${url} connection error:`, err);
    } finally {
        hub.isConnecting = false;
    }
}
public async stopHub(url: string) {
    const hub = this.hubs.get(url);
    if (!hub) return;
    if (hub.connection.state ===
signalR.HubConnectionState.Connected) {
        await hub.connection.stop();
        console.log(`[SignalR] ${url} disconnected`);
    }
}
public async sendMessage(url: string, method: string, args:
any[]) {
    const hub = this.hubs.get(url);
    if (!hub) throw new Error(`Hub ${url} not found`);
    return hub.connection.invoke(method, ...args);
}
}
export const hubFactory = new SignalRHubFactory();

```

Листинг Г.3 – Файл editorUtils.ts

```

// utils/editorUtils.ts
import { Editor } from "@tiptap/react";
import { EditorAttributes } from "../ToolBar/HomeTab/Ts/types";
export function getEditorAttributes(editor: Editor):
EditorAttributes {
    return {

```

```

    fontFamily: editor.getAttributes('textStyle')?.fontFamily ||
'Times New Roman',
    fontSize: editor.getAttributes('textStyle')?.fontSize || 14,
    bold: editor.isActive('bold'),
    italic: editor.isActive('italic'),
    superscript: editor.isActive('superscript'),
    subscript: editor.isActive('subscript'),
    color: editor.getAttributes('textStyle')?.color || '#000000',
    highlight: editor.getAttributes('highlight')?.color || null,
    textAlign: editor.getAttributes('paragraph')?.textAlign ||
'left',
    bulletList: editor.isActive('bulletList'),
    orderedList: editor.isActive('orderedList'),

    indent: editor.getAttributes('paragraph')?.indent || 0,
    indentLeft: editor.getAttributes('paragraph')?.indentLeft ||
0,
    indentRight: editor.getAttributes('paragraph')?.indentRight ||
0,
    indentFirstLine:
editor.getAttributes('paragraph')?.indentFirstLine || 0,
    spacingBefore:
editor.getAttributes('paragraph')?.spacingBefore || 0,
    spacingAfter: editor.getAttributes('paragraph')?.spacingAfter
|| 0,
    lineHeight: editor.getAttributes('paragraph')?.lineHeight ||
1.5,
  };
}

```

Листинг Г.4 – Файл UserProfile.tsx

```

import React, { useEffect, useState } from 'react';
import {
  getMe,
  updateMe,
  changePassword,
  User,
  UpdateUserRequest,
  ChangePasswordRequest,
} from '../api/user';

export const UserProfile: React.FC = () => {
  const [user, setUser] = useState<User | null>(null);
  const [originalUser, setOriginalUser] = useState<User |
null>(null);

  const [passwordData, setPasswordData] =
useState<ChangePasswordRequest>({
    currentPassword: '',
    newPassword: '',
  });

```

```

const [message, setMessage] = useState('');
const [error, setError] = useState('');
const [loadingProfile, setLoadingProfile] = useState(false);
const [loadingPassword, setLoadingPassword] = useState(false);

useEffect(() => {
  getMe()
    .then((data) => {
      setUser(data);
      setOriginalUser(data);
    })
    .catch(() => setError('Ошибка загрузки профиля'));
}, []);

if (!user || !originalUser) return <div>Загрузка...</div>;

const isProfileChanged =
  user.username !== originalUser.username || user.email !==
originalUser.email;

const handleUpdate = async () => {
  if (!isProfileChanged) return;

  setLoadingProfile(true);
  setMessage('');
  setError('');

  try {
    const data: UpdateUserRequest = {};

    // Отправляем username, только если изменился
    if (user.username !== originalUser.username) {
      data.username = user.username;
    }

    // Отправляем email, только если изменился
    if (user.email !== originalUser.email) {
      data.email = user.email;
    }

    if (Object.keys(data).length === 0) {
      setMessage('Нет изменений');
      return;
    }

    await updateMe(data);
    setOriginalUser({ ...user });
    setMessage('Профиль обновлён');
  } catch (e: any) {
    setError(
      e.response?.data?.message ||
      e.response?.data?.detail ||
      'Ошибка обновления профиля'
    );
  }
};

```

```

    } finally {
      setLoadingProfile(false);
    }
  };

  const handleChangePassword = async () => {
    if (!passwordData.currentPassword ||
!passwordData.newPassword) {
      setError('Заполните все поля пароля');
      return;
    }

    if (passwordData.newPassword.length < 6) {
      setError('Новый пароль должен содержать минимум 6
СИМВОЛОВ');
      return;
    }

    setLoadingPassword(true);
    setMessage('');
    setError('');

    try {
      await changePassword(passwordData);
      setMessage('Пароль успешно изменён');
      setPasswordData({ currentPassword: '', newPassword: '' });
    } catch (e: any) {
      setError(
        e.response?.data?.message ||
        e.response?.data?.detail ||
        'Ошибка смены пароля'
      );
    } finally {
      setLoadingPassword(false);
    }
  };

  return (
    <div className="max-w-md mx-auto mt-10 bg-white p-6 rounded
shadow">
      <h2 className="text-xl font-semibold mb-4">Профиль</h2>

      {message && <div className="mb-4 text-green-
600">{message}</div>}
      {error && <div className="mb-4 text-red-600">{error}</div>}

      <label className="block font-semibold mt-2">Имя
пользователя</label>
      <input
        className="w-full mb-2 p-2 border rounded"
        value={user.username}
        onChange={(e) => setUser({ ...user, username:
e.target.value })}

```

```

        disabled={loadingProfile || loadingPassword}
      />

      <label className="block font-semibold mt-2">Email</label>
      <input
        className="w-full mb-4 p-2 border rounded"
        type="email"
        value={user.email}
        onChange={(e) => setUser({ ...user, email: e.target.value
    }}}
        disabled={loadingProfile || loadingPassword}
      />

      <button
        onClick={handleUpdate}
        disabled={!isProfileChanged || loadingProfile ||
loadingPassword}
        className={`w-full py-2 rounded mb-6 ${
          isProfileChanged
            ? 'bg-green-500 hover:bg-green-600 text-white'
            : 'bg-gray-300 text-gray-500 cursor-not-allowed'
        }}`
      >
        {loadingProfile ? 'Сохранение...' : 'Сохранить'}
      </button>

      <h3 className="text-lg font-semibold mb-2">Смена пароля</h3>

      <label className="block font-semibold mt-2">Текущий
пароль</label>
      <input
        type="password"
        className="w-full mb-2 p-2 border rounded"
        value={passwordData.currentPassword}
        onChange={(e) =>
          setPasswordData({
            ...passwordData,
            currentPassword: e.target.value,
          })
        }
        disabled={loadingProfile || loadingPassword}
      />

      <label className="block font-semibold mt-2">Новый
пароль</label>
      <input
        type="password"
        className="w-full mb-4 p-2 border rounded"
        value={passwordData.newPassword}
        onChange={(e) =>
          setPasswordData({
            ...passwordData,
            newPassword: e.target.value,

```

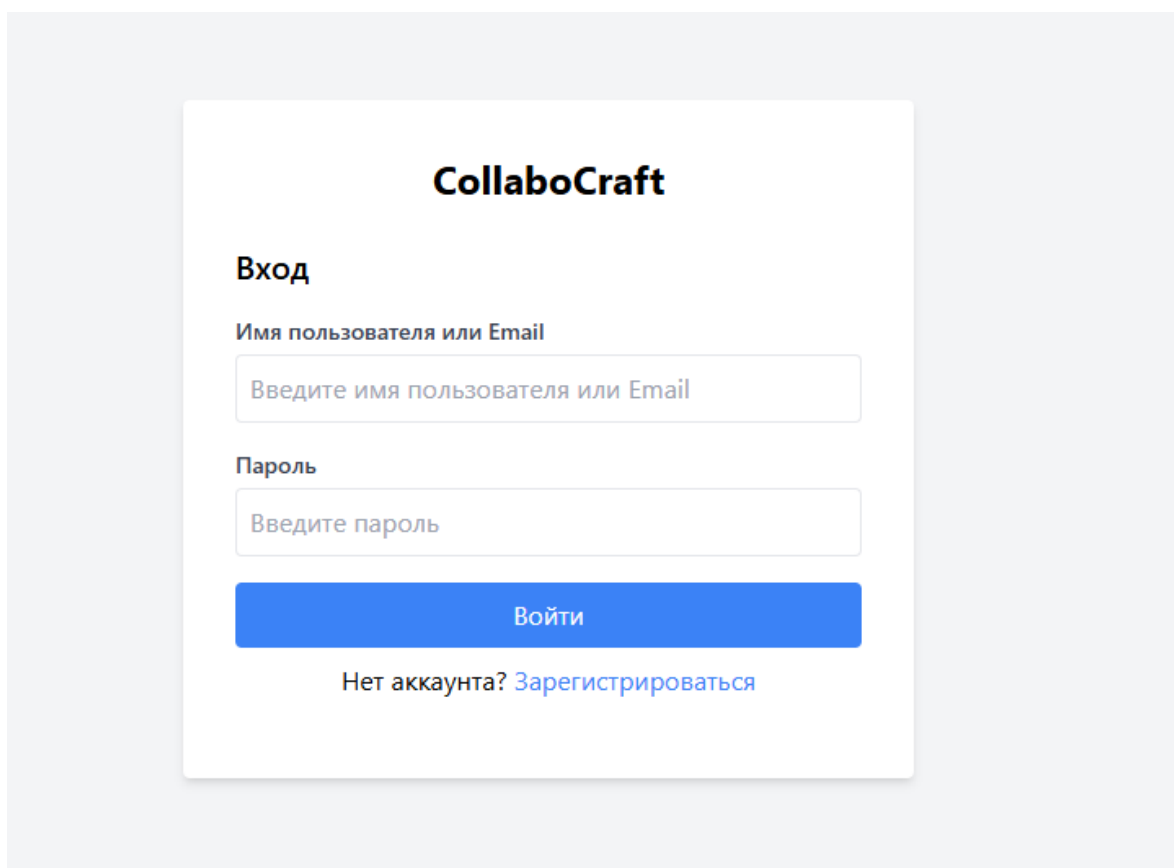
```

        })
    }
    disabled={loadingProfile || loadingPassword}
  />

  <button
    onClick={handleChangePassword}
    disabled={loadingProfile || loadingPassword}
    className={`w-full py-2 rounded ${
      loadingPassword
        ? 'bg-gray-400 text-gray-200 cursor-not-allowed'
        : 'bg-blue-500 hover:bg-blue-600 text-white'
    }`}
  >
    {loadingPassword ? 'Смена пароля...' : 'Сменить пароль'}
  </button>
</div>
);
};

```

Приложение Д
(рекомендуемое)
Графическая часть



CollaboCraft

Вход

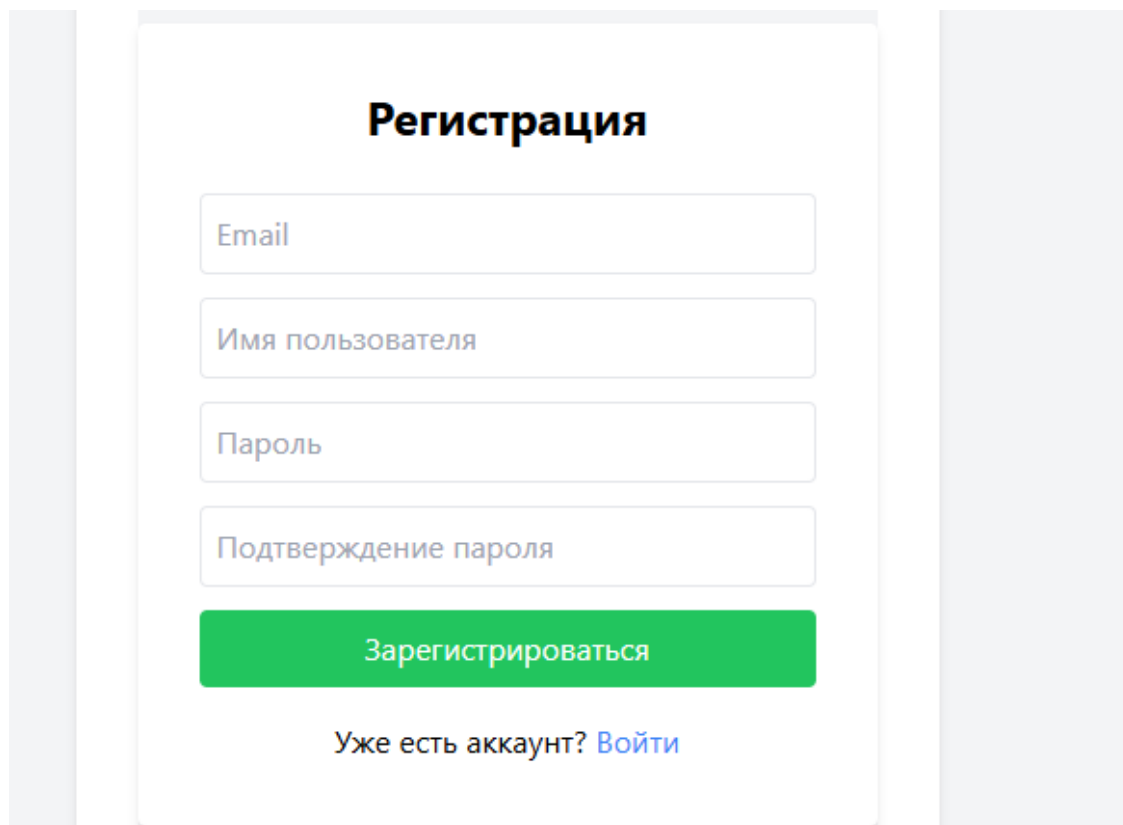
Имя пользователя или Email

Пароль

Войти

Нет аккаунта? [Зарегистрироваться](#)

Рисунок Д.1 – Форма входа



Регистрация

Зарегистрироваться

Уже есть аккаунт? [Войти](#)

Рисунок Д.2 – Форма регистрации

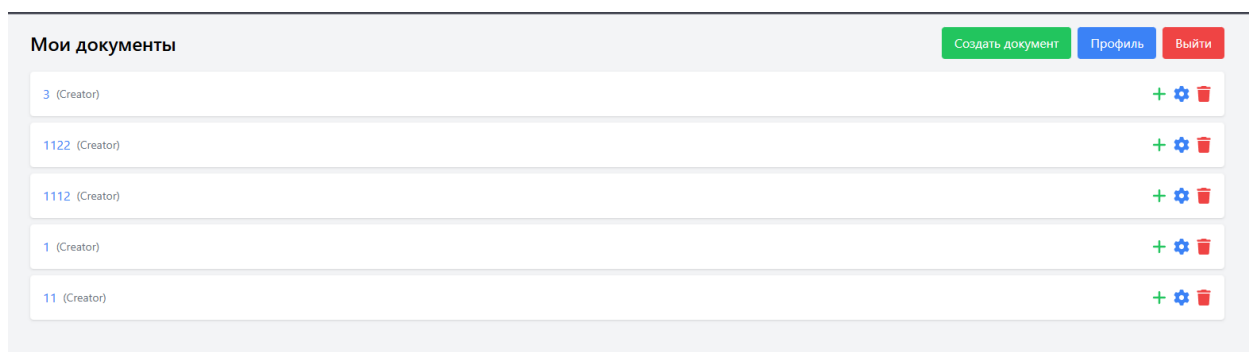


Рисунок Д.3 – Стартовая страница пользователя

Рисунок Д.4 – Форма для создания нового документа

Рисунок Д.5 – Форма для управления документом

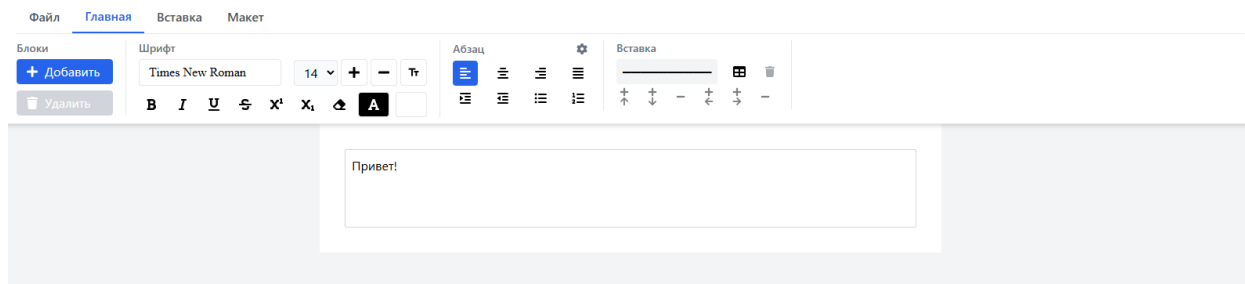


Рисунок Д.6 – Редактируемый документ с блоками

Параметры абзаца

Отступы

Слева: см

Справа: см

Первая строка: см

Интервалы

Перед: см

После: см

Межстрочный: ×

Отмена

ОК

Рисунок Д.7 – Форма для настройки параметров абзаца

Вставить таблицу

Строк:

Столбцов:

Выберите размер: 0x0



Отмена

Вставить

Рисунок Д.8 – Форма для вставки

Приложение Е
(рекомендуемое)
Функциональное тестирование

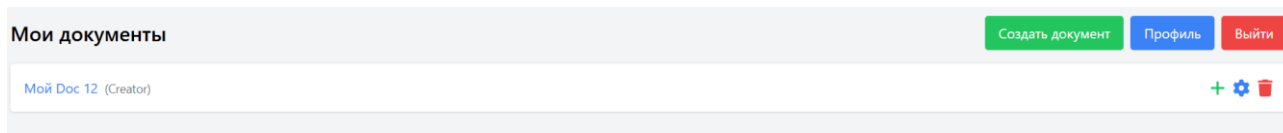


Рисунок Е.1 – Пользователь создал документ «Мой Doc 12»

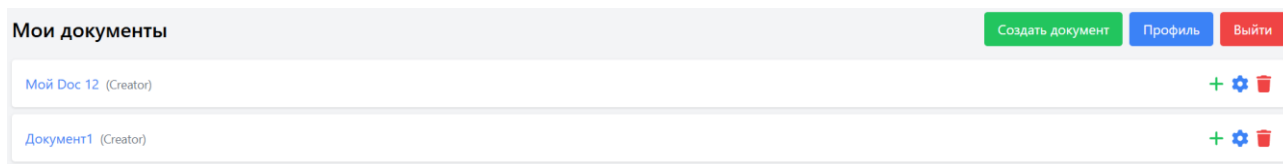


Рисунок Е.2 – Пользователь создал документ «Документ1»

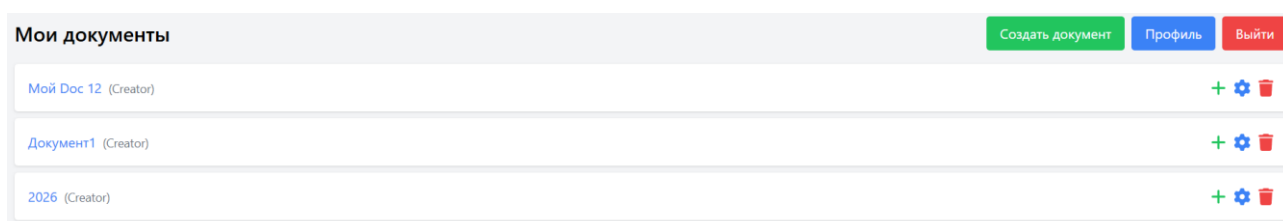


Рисунок Е.3 – Пользователь создал документ «2026»

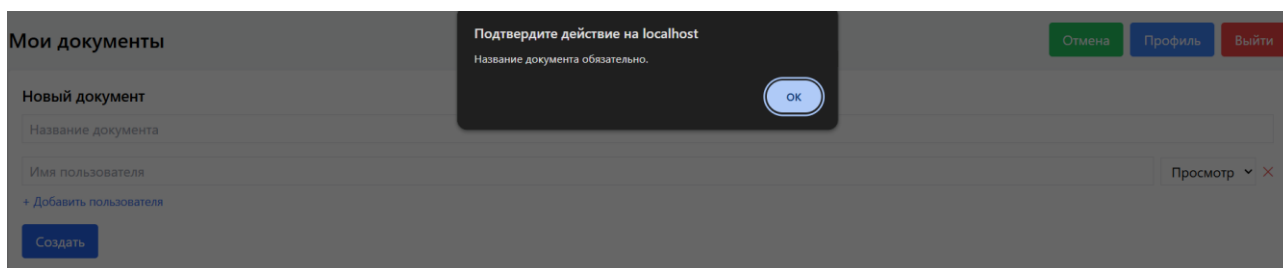


Рисунок Е.4 – Пользователь попытался создать документ «<>»

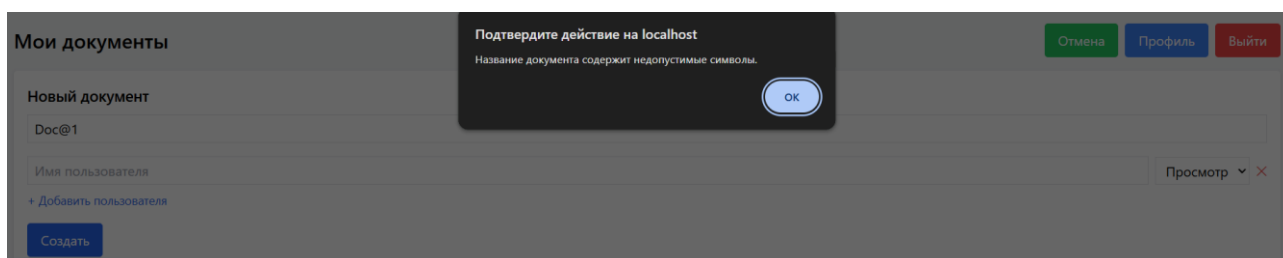


Рисунок Е.5 – Пользователь попытался создать документ «Doc@1»

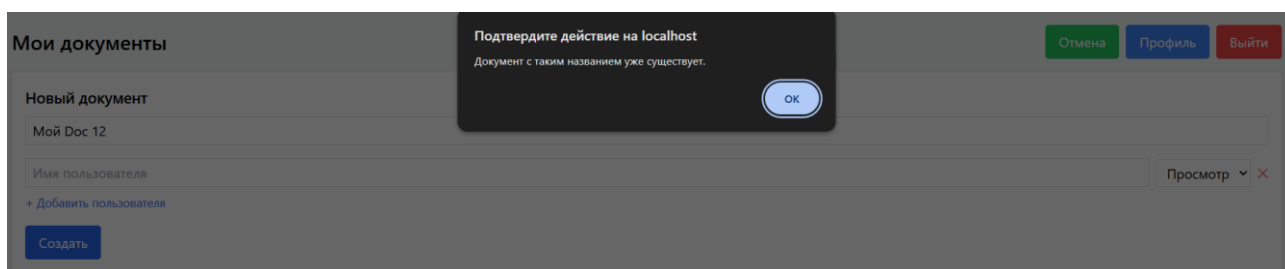


Рисунок Е.6 – Пользователь попытался создать документ «Мой Doc 12»



Рисунок Е.7 – Пользователь создал документ «My Doc»

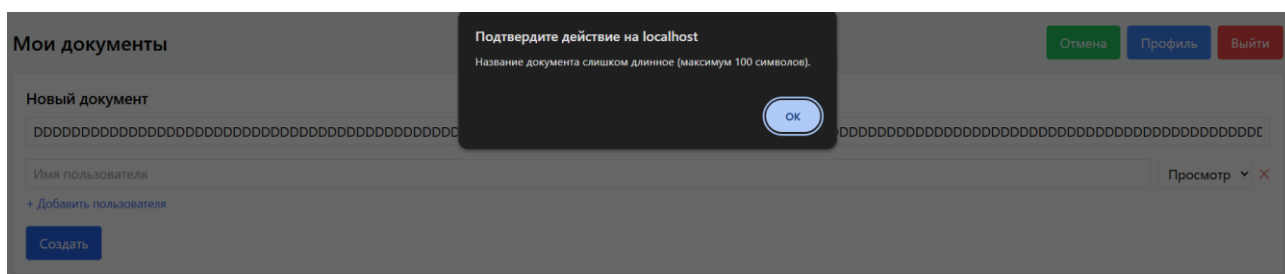


Рисунок Е.8 – Пользователь попытался создать документ «DDDDDD(101)»



Рисунок Е.9 – Пользователь создал документ «2026 1»



Рисунок Е.10 – Пользователь создал документ «А»

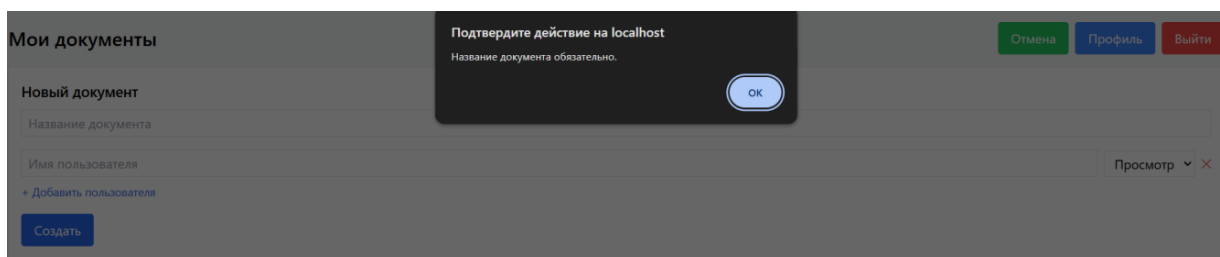


Рисунок Е.11 – Пользователь попытался создать документ «»

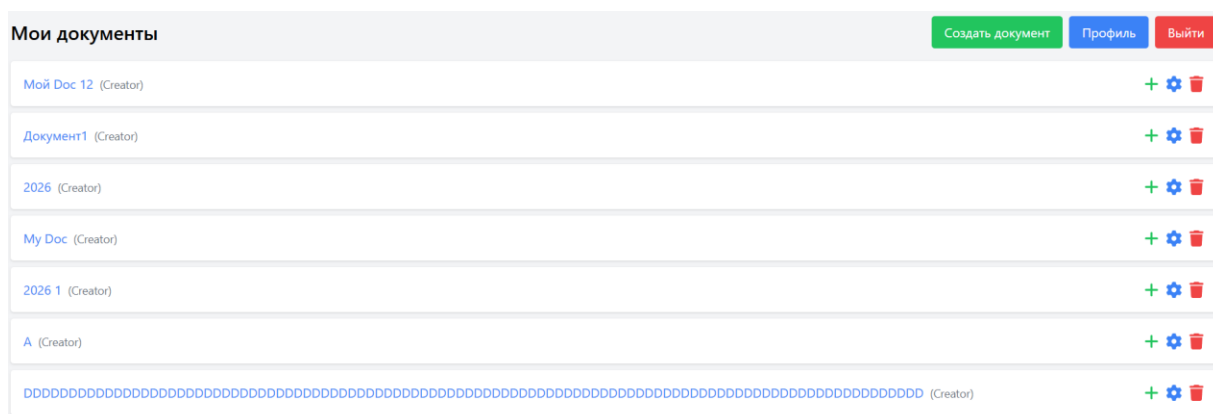


Рисунок Е.12 – Пользователь создал документ «DDDDD(100)»

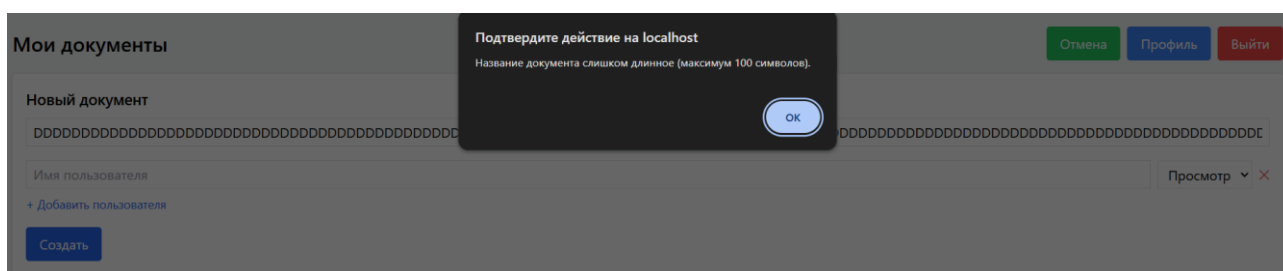


Рисунок Е.13 – Пользователь попытался создать документ «DDDDD(101)»

[illegible]

Рисунок Е.16 – Пользователь попытался создать документ «Мой Дос 12»

Приложение Ж
(рекомендуемое)
Листинг модульных тестов

```

using AuthService.Common;
using AuthService.DataAccess.Models;
using AuthService.DataAccess.Repositories.Interfaces;
using AuthService.Models;
using AuthService.Models.Enums;
using AuthService.Models.Events;
using AuthService.Models.Interfaces;
using AuthService.Services.Exceptions;
using AuthService.Services.Interfaces;
using Microsoft.Extensions.Configuration;
using System.Text;
namespace AuthService.Tests;
public class AuthServiceTests
{
    private const string PEPPER =
"6h5nDgI5vV4kYbL0lmB6eJTTdD0w2w5d2NZJwX4xYYkB0QF7C1mV2xM7yQtJbRkS"
;

    static AuthServiceTests()
    {
        Environment.SetEnvironmentVariable("AUTH_PEPPER", PEPPER);
    }
    // Успешная регистрация пользователя
    [Theory, AutoMoqWithPepper]
    public async Task Register_ValidModel_Success(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        [Frozen] Mock<ICodeRepository> codeRepositoryMock,
        [Frozen] Mock<IKafkaProducer> kafkaProducerMock,
        [Frozen] Mock<IConfiguration> configMock,
        RegisterModel registerModel,
        Services.AuthService authService)
    {
        // Arrange
        registerModel.ConfirmPassword = registerModel.Password;

        var dbUser = new DbUser
        {
            Id = 999,
            Username = registerModel.Username,
            Email = registerModel.Email,
            PasswordHash = "hashed",
            IsEmailConfirmed = false,
            Role = (int)Role.User,
            RefreshToken = string.Empty,
            RefreshTokenExpiredAfter = DateTime.UtcNow
        };
        userRepositoryMock.Setup(x =>
x.IsUserExistsByUsername(registerModel.Username)).ReturnsAsync(false);
        userRepositoryMock.Setup(x =>
x.IsUserExistsByEmail(registerModel.Email)).ReturnsAsync(false);

```

```

        userRepositoryMock.Setup(x =>
x.GetUserByLogin(registerModel.Username)).ReturnsAsync((DbUser) null);

        userRepositoryMock.Setup(x =>
x.CreateUser(It.IsAny<DbUser>())) .ReturnsAsync(dbUser.Id);
        codeRepositoryMock.Setup(x =>
x.SaveAsync(It.IsAny<ConfirmationCode>())) .Returns(Task.CompletedTask);

        const string topic = "email-code-topic";
        configMock.Setup(x =>
x["Kafka:Topics:SendEmailCode"]).Returns(topic);
        kafkaProducerMock.Setup(x => x.ProduceAsync(topic,
It.IsAny<SendEmailCodeEvent>())) .Returns(Task.CompletedTask);
        // Act
        var userId = await authService.Register(registerModel);
        // Assert
        Assert.Equal(dbUser.Id, userId);
        userRepositoryMock.Verify(x =>
x.CreateUser(It.Is<DbUser>(u =>
            u.Username == registerModel.Username &&
            u.Email == registerModel.Email &&
            u.IsEmailConfirmed == false &&
            u.Role == (int)Role.User)), Times.Once);
        codeRepositoryMock.Verify(x =>
x.SaveAsync(It.Is<ConfirmationCode>(c =>
            c.Email == registerModel.Email &&
            c.Code.Length == 4 &&
            c.ExpiresAt > DateTime.UtcNow)), Times.Once);

        kafkaProducerMock.Verify(x => x.ProduceAsync(topic,
It.Is<SendEmailCodeEvent>(e =>
            e.Email == registerModel.Email)), Times.Once);
    }
    // Ошибка при занятом имени
    [Theory, AutoMoqWithPepper]
    public async Task
Register_UsernameAlreadyTaken_ThrowsUsernameAlreadyTakenException(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        RegisterModel registerModel,
        Services.AuthService authService)
    {
        registerModel.ConfirmPassword = registerModel.Password;
        userRepositoryMock.Setup(x =>
x.IsUserExistsByUsername(registerModel.Username)).ReturnsAsync(true);
        userRepositoryMock.Setup(x =>
x.IsUserExistsByEmail(registerModel.Email)).ReturnsAsync(false);
        userRepositoryMock.Setup(x =>
x.GetUserByLogin(registerModel.Username))
            .ReturnsAsync(new DbUser { IsEmailConfirmed = true });

        await Assert.ThrowsAsync<UsernameAlreadyTakenException>(()
=> authService.Register(registerModel));
    }

```

```

    }
    // Ошибка при занятом Email
    [Theory, AutoMoqWithPepper]
    public async Task
    Register_EmailAlreadyTaken_ThrowsEmailAlreadyTakenException(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        RegisterModel registerModel,
        Services.AuthService authService)
    {
        registerModel.ConfirmPassword = registerModel.Password;
        userRepositoryMock.Setup(x =>
            x.IsUserExistsByUsername(registerModel.Username)).ReturnsAsync(false);
        userRepositoryMock.Setup(x =>
            x.IsUserExistsByEmail(registerModel.Email)).ReturnsAsync(true);
        userRepositoryMock.Setup(x =>
            x.GetUserByLogin(registerModel.Username))
            .ReturnsAsync(new DbUser { IsEmailConfirmed = true });
        await Assert.ThrowsAsync<EmailAlreadyTakenException>(() =>
            authService.Register(registerModel));
    }

    // Успешный вход при правильных кредах и подтвержденном Email
    [Theory, AutoMoqWithPepper]
    public async Task Login_ValidCredentials_Success(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        [Frozen] Mock<ITokenService> tokenServiceMock,
        [Frozen] Mock<IAuthSettings> authSettingsMock,
        LoginModel loginModel,
        Services.AuthService authService)
    {
        // Arrange
        var password = "SecurePass123!";
        loginModel.Password = password;
        var passwordHash = HashHelper.HashPassword(password,
            Encoding.UTF8.GetBytes(PEPPER));
        var user = new DbUser
        {
            Id = 777,
            Username = loginModel.Login,
            Email = "user@example.com",
            PasswordHash = passwordHash,
            IsEmailConfirmed = true,
            Role = (int)Role.User
        };
        userRepositoryMock.Setup(x =>
            x.GetUserByLogin(loginModel.Login)).ReturnsAsync(user);
        tokenServiceMock.Setup(x =>
            x.CreateAccessToken(It.IsAny<IEnumerable<System.Security.Claims.Claim>>()))
            .Returns("access-token-123");
        authSettingsMock.Setup(x =>
            x.RefreshTokenExpiresInDays).Returns(30);
    }

```

```

// Act
var result = await authService.Login(loginModel);
// Assert
Assert.Equal("access-token-123", result.AccessToken);
Assert.NotNull(result.RefreshToken);
Assert.NotEmpty(result.RefreshToken);
userRepositoryMock.Verify(x => x.UpdateRefreshToken(
    user.Id,
    It.IsNotNull<string>(),
    It.Is<DateTime>(d => d >
DateTime.UtcNow.AddDays(29))), Times.Once);
}

// Ошибка при входе с неподтвержденным Email
[Theory, AutoMoqWithPepper]
public async Task
Login_EmailNotConfirmed_ThrowsInvalidOperationException(
    [Frozen] Mock<IUserRepository> userRepositoryMock,
    LoginModel loginModel,
    Services.AuthService authService)
{
    var testLogin = "testuser";
    var testPassword = "pass";

    loginModel.Login = testLogin;
    loginModel.Password = testPassword;
    var pepperBytes = Encoding.UTF8.GetBytes(PEPPER);
    var passwordHash = HashHelper.HashPassword(testPassword,
pepperBytes);
    Assert.True(HashHelper.VerifyPassword(passwordHash,
testPassword, pepperBytes));
    var user = new DbUser
    {
        Id = 555,
        Username = testLogin,
        Email = "unconfirmed@example.com",
        PasswordHash = passwordHash,
        IsEmailConfirmed = false,
        Role = (int)Role.User
    };
    userRepositoryMock.Setup(x =>
x.GetUserByLogin(testLogin)).ReturnsAsync(user);

    var ex = await
Assert.ThrowsAsync<InvalidOperationException>(() =>
authService.Login(loginModel));
    Assert.Contains("Email не подтвержден", ex.Message);
}

[Theory, AutoMoqWithPepper]
public async Task ConfirmEmail_ValidCode_Success(
    [Frozen] Mock<IUserRepository> userRepositoryMock,
    [Frozen] Mock<ICodeRepository> codeRepositoryMock,
    [Frozen] Mock<IKafkaProducer> kafkaProducerMock,
    [Frozen] Mock<IConfiguration> configMock,

```

```

[Frozen] Mock<ITokenService> tokenServiceMock,
[Frozen] Mock<IAuthSettings> authSettingsMock,
Services.AuthService authService)
{
    // Arrange
    var userId = 101;
    var email = "test@example.com";
    var code = "1234";
    var user = new DbUser
    {
        Id = userId,
        Email = email,
        Username = "testuser",
        Role = (int)Role.User,
        IsEmailConfirmed = false
    };
    var confirmationCode = new ConfirmationCode
    {
        Email = email,
        Code = code,
        ExpiresAt = DateTime.UtcNow.AddMinutes(10)
    };
    userRepositoryMock.Setup(x =>
x.GetUserById(userId)).ReturnsAsync(user);
    codeRepositoryMock.Setup(x =>
x.GetAsync(email)).ReturnsAsync(confirmationCode);
    codeRepositoryMock.Setup(x =>
x.DeleteAsync(email)).Returns(Task.CompletedTask);
    userRepositoryMock.Setup(x =>
x.ConfirmEmail(email)).Returns(Task.CompletedTask);
    tokenServiceMock.Setup(x =>
x.CreateAccessToken(It.IsAny<IEnumerable<System.Security.Claims.Claim>>()))
        .Returns("new-access-token");
    authSettingsMock.Setup(x =>
x.RefreshTokenExpiresInDays).Returns(7);

    const string topic = "user-created-topic";
    configMock.Setup(x =>
x["Kafka:Topics:UserCreated"]).Returns(topic);
    kafkaProducerMock.Setup(x => x.ProduceAsync(topic,
It.IsAny<UserCreatedEvent>()))).Returns(Task.CompletedTask);
    // Act
    var result = await authService.ConfirmEmail(userId, code);
    // Assert
    Assert.Equal("new-access-token", result.AccessToken);
    Assert.NotNull(result.RefreshToken);
    codeRepositoryMock.Verify(x => x.DeleteAsync(email),
Times.Once);
    userRepositoryMock.Verify(x => x.ConfirmEmail(email),
Times.Once);

```

```

        kafkaProducerMock.Verify(x => x.ProduceAsync(topic,
It.Is<UserCreatedEvent>(e => e.Id == userId && e.Email == email)),
Times.Once);
        userRepositoryMock.Verify(x =>
x.UpdateRefreshToken(userId, It.IsNotNull<string>()),
It.Is<DateTime>(d => d > DateTime.UtcNow.AddDays(6))),
Times.Once);
    }
    // Ошибка пользователь код не верный в ConfirmEmail
    [Theory, AutoMoqWithPepper]
    public async Task
ConfirmEmail_InvalidCode_ThrowsInvalidInputException(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        [Frozen] Mock<ICodeRepository> codeRepositoryMock,
        Services.AuthService authService)
    {
        var user = new DbUser { Id = 1, Email = "test@example.com"
};
        userRepositoryMock.Setup(x =>
x.GetUserById(1)).ReturnsAsync(user);
        codeRepositoryMock.Setup(x =>
x.GetAsync("test@example.com"))
            .ReturnsAsync(new ConfirmationCode { Code = "1111",
ExpiresAt = DateTime.UtcNow.AddMinutes(5) });

        await Assert.ThrowsAsync<InvalidInputException>(() =>
authService.ConfirmEmail(1, "2222"));
    }
    // Ошибка пользователь код истек в ConfirmEmail
    [Theory, AutoMoqWithPepper]
    public async Task
ConfirmEmail_ExpiredCode_ThrowsInvalidInputException(
        [Frozen] Mock<IUserRepository> userRepositoryMock,
        [Frozen] Mock<ICodeRepository> codeRepositoryMock,
        Services.AuthService authService)
    {
        var user = new DbUser { Id = 1, Email = "test@example.com"
};
        userRepositoryMock.Setup(x =>
x.GetUserById(1)).ReturnsAsync(user);
        codeRepositoryMock.Setup(x =>
x.GetAsync("test@example.com"))
            .ReturnsAsync(new ConfirmationCode { Code = "1234",
ExpiresAt = DateTime.UtcNow.AddMinutes(-1) });

        await Assert.ThrowsAsync<InvalidInputException>(() =>
authService.ConfirmEmail(1, "1234"));
    }
}

```