

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

Единая информационная система
медицинских организаций города

Студент

Косенко Артем Сергеевич

Руководитель

Шибанов С.В.

Нормоконтролер

Такташкин Д.В.

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

Козлов А.Ю.

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

Попова Н.А.

Пенза, 2024

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
«25» декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Косенко Артем Сергеевич гр. 20ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Единая информационная система медицинских организаций города

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение разработки: информационная система для взаимодействия с медицинскими организациями города*

2) *Операционная система: Windows 10 и выше*

Используемые языки разработки: Kotlin, C#

Среды разработки: Visual Studio 2022 и IntelliJ IDEA

Основные библиотеки и фреймворки: Spring Framework и WPF

3) *Основной функционал информационной системы:*

а) запись к врачу;

б) просмотр медицинской карты;

в) просмотр списка медицинских организаций;

г) просмотр списка врачей;

д) управление личными данными в рамках ИС.

5. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи на разработку;
2) Проектирование информационной системы;
3) Реализация программы;
4) Тестирование информационной системы;
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%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 100 листов, 45 рисунков, 17 таблиц, 21 использованный источник и 4 приложения.

МЕДИЦИНСКИЕ ОРГАНИЗАЦИИ, СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ЗАПИСЬ К ВРАЧУ, KOTLIN, C#, SPRING FRAMEWORK, KAFKA.

Объект исследования – медицинские организации и их рабочие процессы.

Цель выпускной квалификационной работы – разработать информационную систему для работы с медицинскими организациями города.

Технологии разработки – серверная часть приложения разработана на языке программирования Kotlin с применением фреймворка Spring. Клиентское приложение реализовано на языке C#. В качестве сред разработки были использованы IntelliJ IDEA и Visual Studio 22. Для хранения данных применяется реляционная СУБД PostgreSQL.

Среда исполнения – Windows 10 и выше.

Результаты работы – в ходе выполнения выпускной квалификационной работы была разработана информационная система медицинских организаций города, позволяющая записаться к врачу и просмотреть данные в своей медицинской карте.

					ПГУ 09.03.04– 08BP201.27 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Косенко А.С.			Единая информационная система медицинских организация города. Пояснительная записка	Лит.	Лист	Листов	
Пров.		Шибанов С.В.						4	100
						Группа 20ВП1			
Н. Контр.		Такташкин Д.В							
Утв.		Козлов А.Ю.							

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку единой информационной системы медицинских организаций города	9
1.1 Анализ предметной области информационной системы.....	9
1.1.1 Анализ структурной организации предметной области	9
1.1.2 Анализ свойств информационных объектов предметной области..	10
1.1.3 Анализ бизнес-процессов предметной области.....	11
1.2 Сравнительный анализ приложений аналогов.....	14
1.2.1 Цели и задачи сравнительного анализа	14
1.2.2 Выбор приложений-аналогов для анализа	15
1.2.3 Обзор приложений-аналогов	16
1.2.4 Результаты сравнительного анализа	21
1.3 Постановка задачи на разработку информационной системы	21
1.3.1 Цели разработки и назначение информационной системы.....	21
1.3.2 Классы и роли пользователей информационной системы	22
1.3.3 Автоматизируемые функции	23
1.4 Планирование и оценка процесса разработки информационной системы.....	24
1.4.1 Выбор нотации и программных средств планирования и оценки процесса разработки информационной системы	24
1.4.2 Планирование процесса разработки информационной системы.....	25
1.4.3 Оценка временных и финансовых затрат информационной системы	28
2 Проектирование единой информационной системы медицинских организаций города	31
2.1 Моделирование функциональных возможностей информационной системы.....	31
2.2 Разработка концептуальной модели информационной системы.....	35

2.3 Разработка программной архитектуры информационной системы	36
2.4 Разработка требований к аппаратно-программному обеспечению	38
2.5 Обоснование выбора средств разработки.....	39
2.5.1 Обоснование выбора средств автоматизации	39
2.5.2 Обоснование выбора инструментальных средств разработки программного обеспечения	40
2.5.3 Обоснование выбора языковых средств разработки.....	41
2.5.4 Обоснование выбора средств управления данными	42
3 Разработка программного обеспечения единой информационной системы медицинских организация города	44
3.1 Разработка программного обеспечения REST-сервера.....	44
3.2 Разработка микросервисов	46
3.2.1 Разработка структуры микросервисов.....	46
3.2.2 Разработка программного обеспечения микросервиса пациента	50
3.2.3 Разработка программного обеспечения микросервиса врача	51
3.2.4 Разработка программного обеспечения микросервиса администратора	53
3.3 Разработка программного обеспечения клиентского приложения.....	54
4 Контроль качества программного обеспечения.....	57
4.1 Функциональное тестирование.....	57
4.2 Нефункциональное тестирование интерфейса	60
4.3 Метрики кода.....	65
4.3.1 Метрика SLOC	65
4.3.2 Метрика Холстеда.....	66
4.3.3 Метрика Чепина	69
Заключение	70
Список использованных источников	71
Приложение А (справочное) Глоссарий	73
Приложение Б (рекомендуемое) Спецификации вариантов использования	75
Приложение В (рекомендуемое) Листинг приложения	80

Приложение Г (рекомендуемое) Результаты тестирования	96
--	----

Введение

Использование информационных технологий в сфере медицины становится все более актуальной темой. Их внедрение в данную сферу позволяет улучшить процесс лечения пациентов, в том числе благодаря переносу и сохранению результатов обследований в цифровом варианте. Другим важным прорывом стало создание информационных систем, работающих в сети Интернет, которые позволяют записаться к специалистам в сфере медицины, не выходя из дома.

Существует много разновидностей реализаций подобных информационных систем, которые можно разделить на три основные группы: системы, работающие с конкретной сетью частных медицинских организаций; системы, включающие в себя множество разных сетей частных медицинских организаций; системы, работающие с государственными медицинскими организациями. Во всем этом разнообразии отсутствует информационная система, которая позволила бы централизовать работу со всеми медицинскими организациями города.

Именно описание и разработка подобной системы и является целью данной работы. Для ее решения необходимо выполнить следующие пункты:

- провести анализ предметной области и бизнес-процессов;
- создать концептуальную модель базы данных;
- спроектировать архитектуру разрабатываемого приложения;
- разработать серверную часть приложения;
- разработать клиентскую часть приложения;
- провести тестирование программного обеспечения;
- оформить пояснительную записку.

В процессе проектирования и разработки были использованы методы сравнительного анализа, функционального и нефункционального тестирования.

1 Анализ предметной области и постановка задачи на разработку единой информационной системы медицинских организаций города

1.1 Анализ предметной области информационной системы

1.1.1 Анализ структурной организации предметной области

Медицинские организации различаются по многим характеристикам, определяющим как их основной профиль деятельности, так и другие параметры. В набор утвержденных Минздравом РФ медицинских организаций входят три группы:

- лечебно-профилактические медицинские организации: роддома, больницы, госпитали и диспансеры;
- медицинские организации особого типа: медцентры, лаборатории и бюро судебной экспертизы;
- медицинские организации по надзору: центры гигиены, центры эпидемиологии и дезинфекционные центры [1].

Другой характеристикой медицинских организаций является уровень оказания медико-санитарной помощи:

- первичный уровень – подобные учреждения относятся к группе первичного контакта с населением, к которым относятся фельдшерские пункты, поликлиники, травмпункты, а также аптеки;
- второй уровень – на данном уровне находятся учреждения амбулаторного обслуживания, которые способны оказывать специализированную помощь, и имеющие в составе свои больницы, стационары и госпитали;
- третий уровень – учреждения занимаются оказанием узкоспециализированной и высокотехнологичной медицинской помощи, в числе которых есть центры сердечно-сосудистой хирургии, ожоговые центры и медицинские НИИ [2].

Несмотря на подобное разделение, разные уровни могут взаимодействовать друг с другом при возникновении подобной необходимости.

Существуют и другие структурные особенности, по которым разделяют медицинские организации, такие как системы финансирования и виды оказания медико-санитарной помощи.

1.1.2 Анализ свойств информационных объектов предметной области

В предметной области информационной системы существуют разнообразные объекты с различными свойствами. Данные объекты можно разделить на две основные группы: структура медицинской организации и информация, связанная с пользователем.

Рассмотрим свойства объектов, относящихся к первой группе:

- поликлиника: обладает названием, адресом и статусом активности, который определяет, является ли поликлиника действующей;
- отделение: имеет название и статус активности, а также прикрепляется к конкретной поликлинике и не может существовать самостоятельно.

Данные объекты являются одними из важнейших в предметной области, так как именно они и определяют структуру медицинской организации.

Теперь проанализируем свойства другой группы объектов, относящейся к информации, напрямую связанной с пользователями системы.

Главным объектом данной группы и является сам пользователь. Так как в информационной системе существует несколько видов пользователей, данная сущность может быть администратором, врачом или пациентом. Пользователь имеет логин и пароль для входа в ИС, а также содержит личные данные, такие как имя, пол и дата рождения. Если пользователь является администратором, то в набор его свойств также входит личный рейтинг и отделение, к которому он привязан.

Объект записи к врачу обладает общей информацией о времени записи, а также данными о записавшемся пациенте с обследующим его врачом.

Специальный объект диагноз хранит данные об одном диагнозе пациента, содержит описание данного диагноза, информацию о пациенте, дату заключения диагноза и его текущий статус: является ли он излеченным.

Свойства данных объектов из этих двух групп позволяют полноценно описать основные процессы в разрабатываемой информационной системе.

1.1.3 Анализ бизнес-процессов предметной области

Проведение анализа бизнес-процессов представляет собой важный этап исследовательской и аналитической работы, на основе результатов которого будет построена бизнес-логика информационной системы. На данном этапе можно определить как самые главные процессы (чтобы построить работу программы вокруг них), так и выявить потенциальные проблемы в функциональных возможностях.

Сами бизнес-процессы можно разделить на четыре основные группы:

- бизнес-процессы администратора;
- бизнес-процессы врача;
- бизнес-процессы пациента;
- общие пользовательские бизнес-процессы.

Так, к группе администратора относятся процессы, связанные с управлением жизненным циклом поликлиники, отделения и пользователя, а в группе врача сосредоточены процессы по записи пациентов к врачу и работы с их медицинской картой. Группа пациента основана на добавлении новых персональных записей к врачу и просмотре своей медицинской карты, а группа общих пользовательских бизнес-процессов необходима для работы с личными данными.

Для наглядного представления описанных бизнес-процессов и их групп была построена диаграмма деятельности, показанная на рисунке 1, которая позволит выделить основные этапы этих процессов, их возможные пути развития и используемые ресурсы.

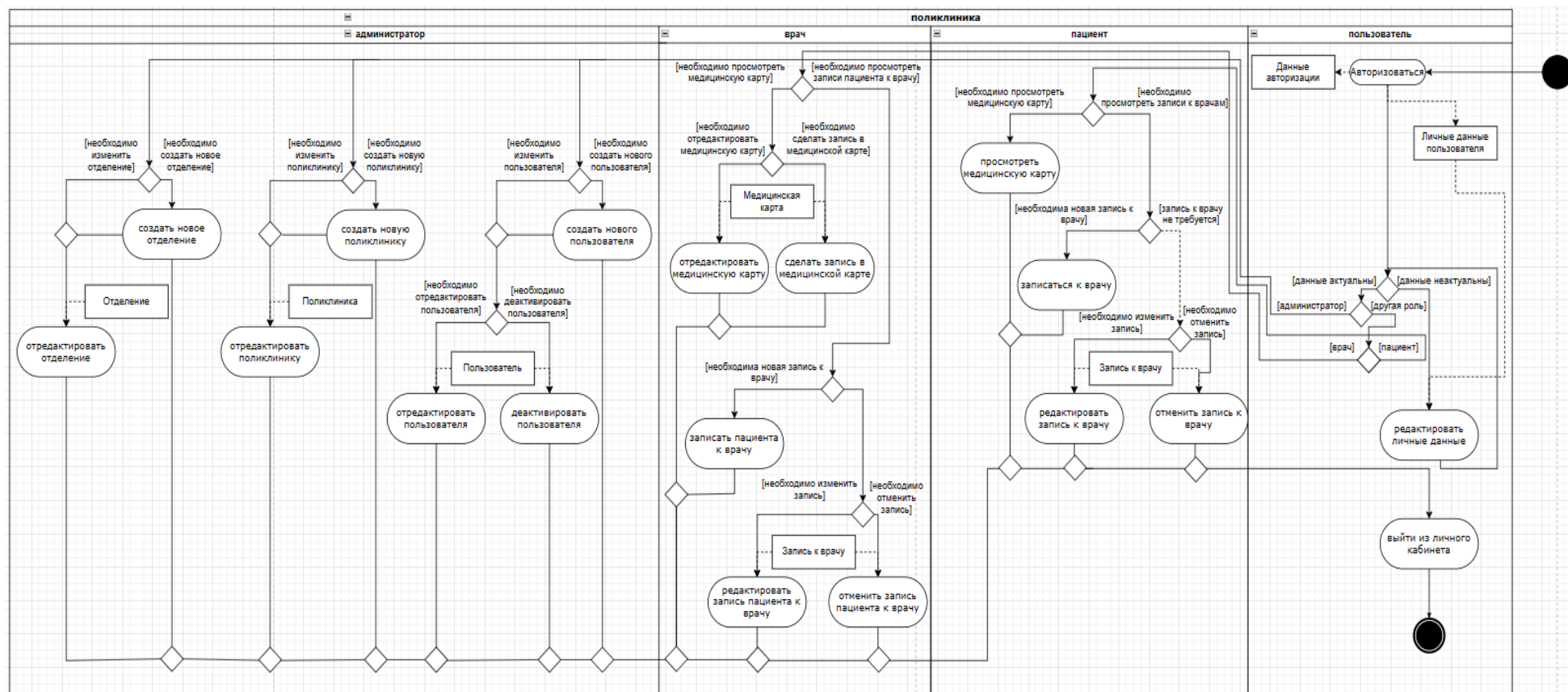


Рисунок 1 – Диаграмма деятельности бизнес-процессов

Спецификация бизнес-процессов предметной области показана в таблице 1.

Таблица 1 – Спецификации бизнес-процессов предметной области

№ п/п	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потребитель	Содержание	
1	Войти в аккаунт	Пользователь	Пользователь	Логин и пароль	Пользователь	Данные о пользователе	Пользователь должен быть зарегистрирован
2	Редактировать личные данные	Пользователь	Пользователь	Новые личные данные	Пользователь	Данные о пользователе	Новые личные данные должны быть валидны
3	Просмотреть свою медицинскую карту	Пациент	Пациент	Идентификатор пациента	Пациент	Данные личной карты	
4	Управлять своими записями к врачам	Пациент	Пациент	Идентификатор пациента	Пациент	Данные о записях к врачам	
5	Управлять медицинской картой пациента	Врач	Врач	Идентификатор пациента	Пациент	Данные личной карты	Пациент должен существовать
6	Управлять записями пациента к врачам	Врач	Врач	Идентификатор пациента	Пациент и врач	Данные о записях к врачам	Пациент должен существовать
7	Создать нового пользователя	Администратор	Администратор	Данные для создания пользователя	Пользователь	Данные о пользователе	Данные пользователя должны быть валидными
8	Управлять пользователями	Администратор	Администратор	Идентификатор пользователя	Пользователь	Данные о пользователе	
9	Управлять поликлиниками	Администратор	Администратор	Идентификатор поликлиники	Поликлиника	Данные о поликлинике	
10	Управлять отделениями	Администратор	Администратор	Идентификатор отделения	Отделение	Данные об отделении	

Представленная спецификация описывает самые основные бизнес-процессы в информационной системе.

1.2 Сравнительный анализ приложений аналогов

1.2.1 Цели и задачи сравнительного анализа

Основной задачей сравнительного анализа информационной системы с приложениями-аналогами является выявление сильных и слабых сторон разрабатываемой программы, а также определение возможностей для ее дальнейшей реализации.

Для решения данной задачи необходимо выполнить следующие пункты:

- рассмотреть в общих чертах существующие приложения и выявить популярные решения;
- выделить основные критерии для проведения сравнительного анализа;
- проанализировать каждое из приложений, чтобы выявить их слабые и сильные стороны;
- провести сравнительный анализ рассматриваемых приложений и подвести его итоги.

В результате проведения сравнительного анализа будут выявлены и рассмотрены особенности приложений, начиная от интерфейса и заканчивая функциональными возможностями.

В качестве основных критериев сравнения будут выступать следующие параметры:

- наличие современного и удобного программного интерфейса;
- наличие встроенной медицинской карты;
- выбор отделения для записи к врачу;
- просмотр расписания своих записей к врачу;
- возможность отмены или переноса записи;
- поддержка государственных медицинских учреждений.

На основе этих критериев будет проведен анализ приложений аналогов.

1.2.2 Выбор приложений-аналогов для анализа

Для проведения сравнительного анализа был рассмотрен набор приложений-аналогов, используемых как для работы с частными клиниками, так и с государственными. Подобные аналоги существуют в разном виде, в том числе как мобильные приложения и сайты. Общее рассмотрение данных программ представлено в таблице 2.

Таблица 2 – Общие сведения о приложениях-аналогах

Название приложения	Организация	Общие сведения
medguard.ru	ООО «Медгард - Пенза»	Одна из крупнейших сетей медицинских центров в Поволжье, имеющая свои клиники в Самаре, Пензе, Тольятти и других городах. Имеет как взрослые, так и детские поликлиники, стационары и лаборатории
mediclinic.ru	ООО КДЦ «МЕДИКЛИНИК»	Многопрофильный медицинский центр, предлагающий полный набор услуг в своей сфере. В клинике есть круглосуточный и дневной стационар
medekspert58.ru	ООО «Консилиум»	Медицинский центр «Персональное здоровье» оснащен современным оборудованием и предлагает услуги высококвалифицированных врачей. На базе центра проводятся исследования в области лабораторно-диагностических, аллергологических и прочих направлений
СберЗдоровье	ООО «ИННОВАЦИОННАЯ МЕДИЦИНА»	Компания, предоставляющая цифровые медицинские сервисы, в том числе возможность записи к врачу, телемедицину и поддержку онлайн аптеки
Здоровье.ру	ООО «ЗДОРОВЬЕ.РУ»	Мобильное приложение для работы с государственными поликлиниками, предназначенное для записи к специалистам в сфере медицины и ведения онлайн-диагностики на основе анализов

Первые три описанных в таблице аналога принадлежат городским частным клиникам и существуют как часть функционала на их официальных сайтах. Последние два аналога преимущественно или целиком используются как мобильные приложения и работают не с конкретной сетью медицинских учреждений, а сразу со многими из них.

1.2.3 Обзор приложений-аналогов

Приложение сети медицинских центров «Медгард» предоставляет набор функциональных возможностей для работы с их учреждениями в разных городах [3], в том числе позволяя записаться к врачу, как показано на рисунке 2.

Главная / Запись на приём

Запись на прием

Лечебно-диагностический комплекс

Гастроэнтеролог

Поиск по врачам

4 мая — 6 мая

СЛЕДУЮЩИЕ 3 ДНЯ

Иванова Ольга Коняхина

врач-гастроэнтеролог, терапевт

гастроэнтеролог

Стоимость приема
1 900 ₽

г. Пенза, ул. Карла Маркса, д. 16

Запишитесь на приём, оператор свяжется с Вами для назначения времени приёма!

ЗАПИСАТЬСЯ НА ПРИЕМ

Корнеева Ирина Валерьевна

Врач-гастроэнтеролог, терапевт, кардиолог, врач высшей категории, кандидат медицинских наук

гастроэнтеролог

Стоимость приема
2 200 ₽

г. Пенза, ул. Карла Маркса, д. 16

6 мая

13:00	13:30	14:00	14:30
15:00	15:30		

Рисунок 2 – Страница записи к врачу в приложении «Медгард»

Преимущества:

- наличие современного и удобного программного интерфейса;
- возможность настроить поиск врача по медицинскому центру, специальности и времени;
- возможность управлять своими существующими записями;
- функция оставления заявки для вызова врача на дом.

Недостатки:

- отсутствует поддержка медицинской карты;

– поддерживает только городские медицинские центры сети «Медгард».

Информационная система медицинских центров «Медиклиник» предлагает набор возможностей для оформления и использования врачебных услуг [4]. Пример одной из страниц системы показан на рисунке 3.

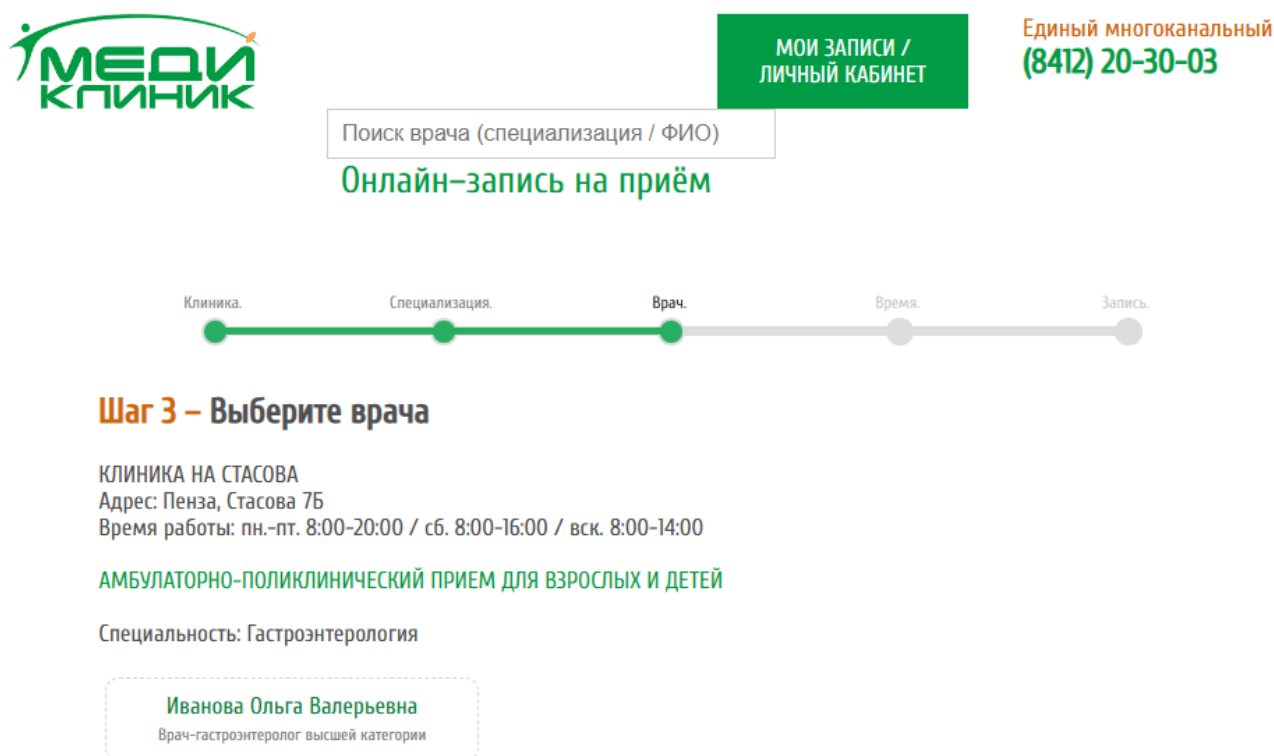


Рисунок 3 – Страница записи к врачу в приложении «Медиклиник»

Преимущества:

- удобный программный интерфейс;
- выбор врача из набора клиник и специальностей;
- просмотр записей к врачам в личном кабинете.

Недостатки:

- нет поддержки медицинской карты;
- поддерживает исключительно городские медицинские центры сети «Медиклиник».

Приложение медицинской сети «Персональное здоровье» предлагает базовый набор функционала в сфере медицинских организаций [5]. Страница записи к врачу в один из медицинских центров сети показана на рисунке 4.

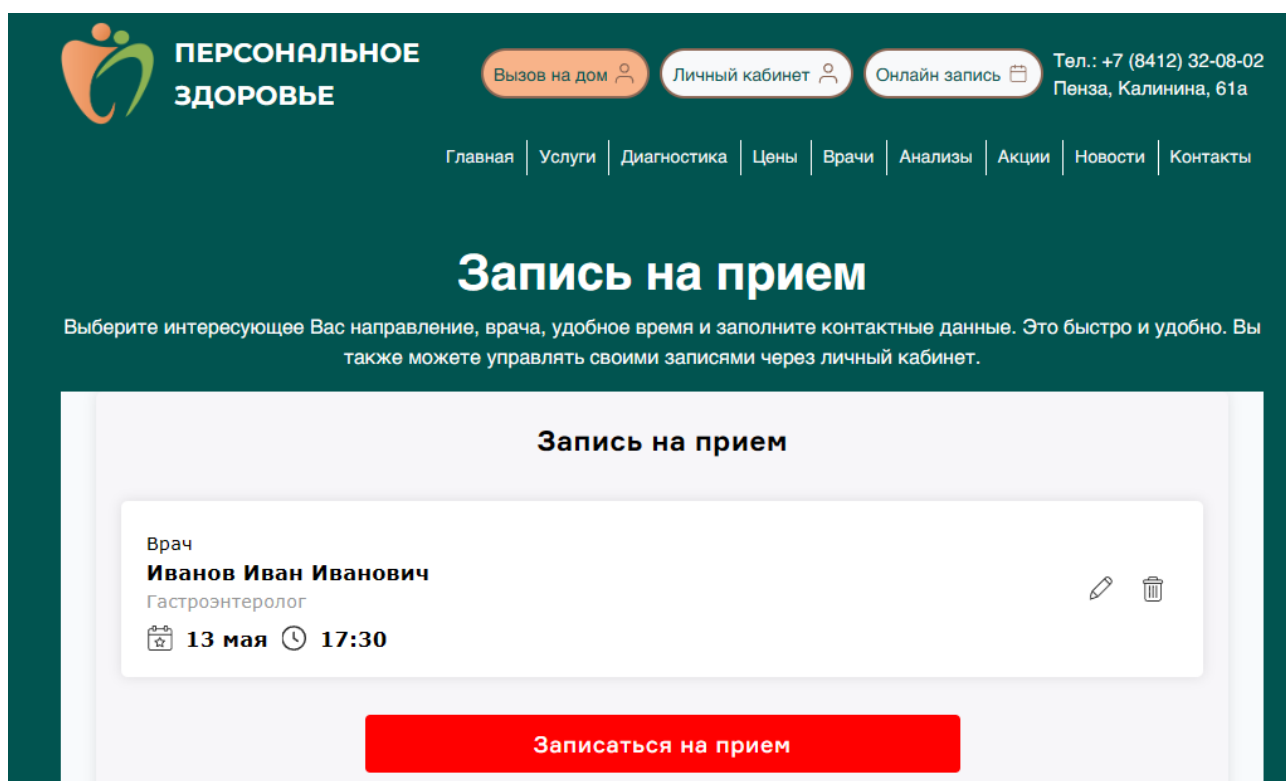


Рисунок 4 – Страница записи к врачу в приложении «Персональное здоровье»

Преимущества:

- возможность вызвать врача на дом;
- выбор врача по требуемой специальности и времени.

Недостатки:

- нет поддержки медицинской карты;
- примитивный интерфейс;
- поддерживает только городские медицинские центры сети

«Персональное здоровье».

Маркетплейс медицинских услуг и продуктов «СберЗдоровье» предоставляет своим клиентам возможность записаться к врачу в частные городские медицинские центры, проведение онлайн-консультаций в формате телемедицины, а также другой обширный функционал [6]. Одна из страниц данной информационной системы, позволяющая записаться к врачу показана на рисунке 5.

Гастроэнтерологи в Пензе · 45 врачей

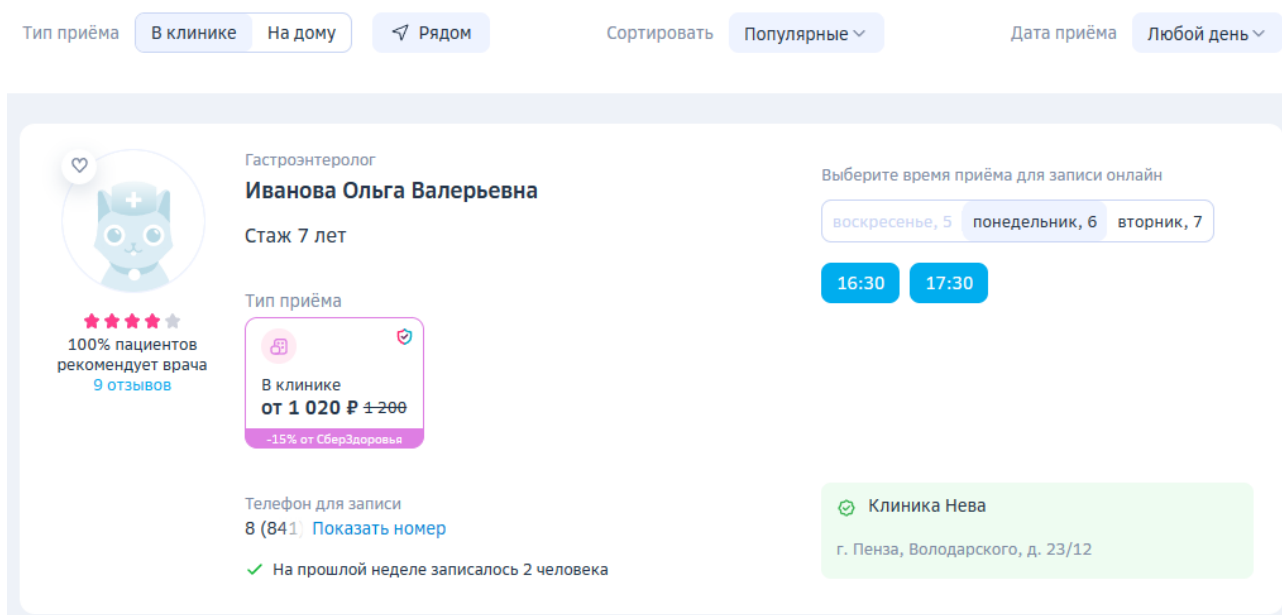


Рисунок 5 – Страница записи к врачу в приложении «СберЗдоровье»

Преимущества:

- современный и удобный интерфейс;
- предоставление услуг телемедицины;
- наличие медицинской карты;
- выбор врача по требуемой специальности и времени.

Недостатки:

- избыточный функционал, если рассматривать систему как информационную систему медицинских организаций города;
- поддерживает только частные городские медицинские центры.

Мобильное приложение «Здоровье.ру» предоставляет возможность для бесплатной записи в государственные поликлиники по полису ОМС, проведение начальной онлайн-диагностики на основе специальных тестов и рекомендаций врачей, а также специальный дневник для контроля и аналитики показателей здоровья [7]. Страница дневника здоровья в информационной системе показана на рисунке 6.

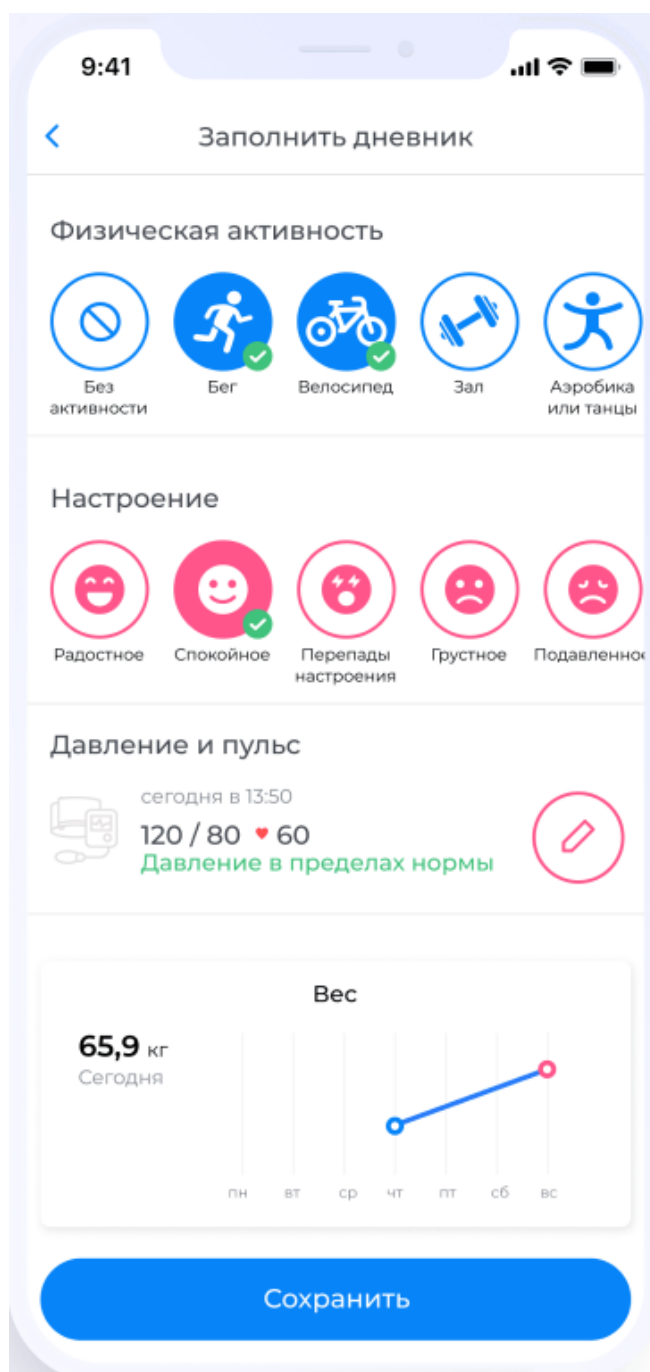


Рисунок 6 – Страница дневника здоровья в приложении «Здоровье.ру»

Преимущества:

- возможность записи к врачу в государственные поликлиники;
- наличие дневника здоровья;
- возможность организации онлайн-консультации с врачами.

Недостатки:

- отсутствует поддержка медицинской карты;
- возможность записи только в государственные медицинские центры.

Далее, рассмотренные в рамках этого пункта приложения-аналоги, будут сравнены между собой и разрабатываемой информационной системой.

1.2.4 Результаты сравнительного анализа

Основываясь на выбранных критериях сравнения, была построена таблица, рассматривающая приложения-аналоги и разрабатываемую информационную систему на соответствие им. Результат сравнения показан в таблице 3.

Таблица 3 – Результат сравнения информационной системы с аналогами

Критерий сравнения	ИС	Медгард	Медиклиник	Персональное здоровье	СберЗдоровье	Здоровье.ру
Поддержка медицинской карты	Да	Нет	Нет	Нет	Да	Нет
Современный программный интерфейс	Да	Да	Да	Нет	Да	Да
Поддержка работы с гос. клиник	Да	Нет	Нет	Нет	Нет	Да
Выбор врача по конкретной специализации	Да	Да	Да	Да	Да	Да
Возможность управлять своими записями к врачу	Да	Да	Да	Да	Да	Да

На основе результатов сравнения можно сделать вывод, что разрабатываемая информационная система медицинских организаций города способна в полной мере исполнять необходимые задачи и поддерживать требуемый функционал, включая наличие медицинских карт и возможность работы как с частными клиниками, так и с государственными.

1.3 Постановка задачи на разработку информационной системы

1.3.1 Цели разработки и назначение информационной системы

Информационная система медицинских организаций города разрабатывается с целью предоставления удобного функционала для работы с медицинскими учреждениями как со стороны пациента, так и со стороны

специалиста. Перечисленные далее возможности ИС позволяют ей реализовать себя в описанном направлении.

Возможность работать как с частными медицинскими организациями, так и с государственными: одна из задач информационной системы заключается в создании единой платформы, позволяющей подключать и работать со всеми городскими медицинскими центрами. Реализация данной задачи позволит отказаться от необходимости использования приложений-аналогов, где одна часть приложений поддерживает только частные клиники, а вторая – только государственные.

Поддержка медицинской карты: данная возможность позволяет обеспечить поддержку единой медицинской карты для всех медицинских организаций города. Это позволит врачам быстро и удобно ознакомиться со всеми обследованиями пациента, даже если они были проведены и записаны при посещении другой клиники;

Возможность управлять записями к врачу: информационная система позволяет записываться к врачам разного профиля и в разные поликлиники, а также предоставляет возможность изменять данные записи. Благодаря этому все записи пациента к врачу находятся в одном месте, что облегчает работу с ними.

Данные возможности информационной системы позволят повысить доступность и улучшить качество работы с медицинскими организациями благодаря предоставлению удобной единой системы.

1.3.2 Классы и роли пользователей информационной системы

В рамках информационной системы представлены три вида пользователей, позволяющих полноценно обеспечивать работу медицинских организаций города на всех требуемых этапах функциональных возможностей. Рассмотрим эти виды пользователей:

- пациент: пользователи ИС с данной ролью обладают возможностью просмотра своей медицинской карты для ознакомления с результатами анализов и с заключениями специалистов, способны редактировать свои личные данные,

а также управлять своими записями к врачу, что подразумевает как создание новых записей в медицинские организации, так и редактирование уже существующих;

- врач: данная роль в ИС дает возможность вносить и редактировать записи в медицинской карте пациента, а также управлять записями пациентов к врачам, позволяя самостоятельно записывать их к требуемому специалисту и вносить изменения в уже существующие записи при необходимости;

- администратор: главная роль в системе, дающая пользователю доступ к функциям для создания и редактирования поликлиник и их отделений, а также созданием, редактированием и деактивацией пользователей.

Подобное распределение возможностей в рамках информационной системы делает возможным ее правильное и полноценное функционирование.

1.3.3 Автоматизируемые функции

Одной из самых основных задач информационной системы является автоматизация и оптимизация ряда бизнес-процессов и функций, протекающих в рамках рассматриваемой предметной области. Это упростит и улучшить процесс работы медицинского персонала, а также позволит пациентам удобнее взаимодействовать с медицинскими организациями города.

В таблице 4 описаны основные автоматизируемые функции в информационной системе.

Таблица 4 – Автоматизируемые функции в информационно системе

Бизнес-процесс	Требуемая роль	Описание
Управление жизненным циклом поликлиники	Администратор	Администратор ИС может создавать новую поликлинику и редактировать уже существующую
Управление жизненным циклом отделения	Администратор	Администратор ИС может создавать новое отделение и редактировать уже существующее
Управление жизненным циклом пользователей	Администратор	Администратор ИС может создавать новых пользователей, редактировать и деактивировать их

Продолжение таблицы 4

Бизнес-процесс	Требуемая роль	Описание
Управление жизненным циклом записи к врачу	Врач/Пациент	Возможность создавать и редактировать записи пациента к врачу. Пользователь с ролью «врач» может работать с записями любого пациента, тогда как пользователь с ролью «пациент» работает только со своими записями
Взаимодействие с медицинской картой пациента	Врач/Пациент	Для пользователя с ролью «пациент» данное взаимодействие ограничивается простым просмотром содержимого медицинской карты, тогда как врач способен вносить в нее новые записи и редактировать уже существующие
Управление личными данными	Любая	Пользователь информационной системы может просматривать свои личные данные и редактировать их
Вход в систему	Любая	Существующий пользователь информационной системы может войти в нее при введении корректных данных для аутентификации

Получившаяся в результате таблица позволяет увидеть как сами автоматизируемые функции в информационной системе, так и роль пользователя, задействующую ее. Эти данные станут одной из основ для реализации приложения.

1.4 Планирование и оценка процесса разработки информационной системы

1.4.1 Выбор нотации и программных средств планирования и оценки процесса разработки информационной системы

Для правильного планирования процесса разработки и оценки используемых ресурсов необходимо выбрать наиболее подходящее программное средство и правильную нотацию. Это позволит понять, сколько времени займет процесс разработки, какие ресурсы будут задействованы и какой потребуется бюджет. Все это необходимо для того, чтобы как можно точнее оценить процесс разработки.

В качестве используемой нотации будет выступать диаграмма Ганта. Данный подход позволяет проиллюстрировать план или график работ внутри проекта. Диаграмма в наглядной форме отображает последовательность задач, а также длительность их исполнения. Важными преимуществами подхода

является простота в понимании, возможность быстро вносить корректировки в план, а также видение текущего прогресса в проекте. Также диаграмма Ганта позволяет удобно распределять ресурсы проекта и выявлять критические пути в нем [8].

Project Libre, выбранный в качестве программного средства для планирования процесса разработки, предоставляет широкий функционал для обеспечения эффективного управления проектом. Данный инструмент поддерживает работу с диаграммами Ганта и дает возможность управлять используемыми ресурсами проекта. Также Project Libre поддерживает и другие виды диаграмм, как, например, PERT, Resource Breakdown Structure и Work Breakdown Structure [9].

Совместное использование инструмента Project Libre и диаграммы Ганта позволит в понятной и удобной форме спланировать процесс разработки проекта. Это будет достигнуто благодаря визуализации всех этапов проекта, ведения учета используемых ресурсов и возможности внести корректировки при необходимости.

1.4.2 Планирование процесса разработки информационной системы

Этап планирования процесса разработки позволяет сформировать представление о том, как именно будет проходить разработка ИС и какие сроки будут заложены для каждого из этапов. Сам план должен включать в себя все основные этапы, начиная с анализа предметной области и заканчивая проведением контроля качества получившегося программного обеспечения. Также в дальнейшем план процесса разработки будет использован при планировании бюджета.

Для планирования процесса разработки в инструменте Project Libre была построена диаграмма Ганта, разделяющая процесс разработки на четыре основных этапа. Описание задач, используемых для построения диаграммы, показано на рисунке 7.

На данном этапе разработки происходит выполнение задач, направленных на проведение анализа поставленной задачи, предметной области, а также приложений аналогов.

Вторая часть диаграммы Ганта, рассматривающая проектирование ИС, показана на рисунке 9.

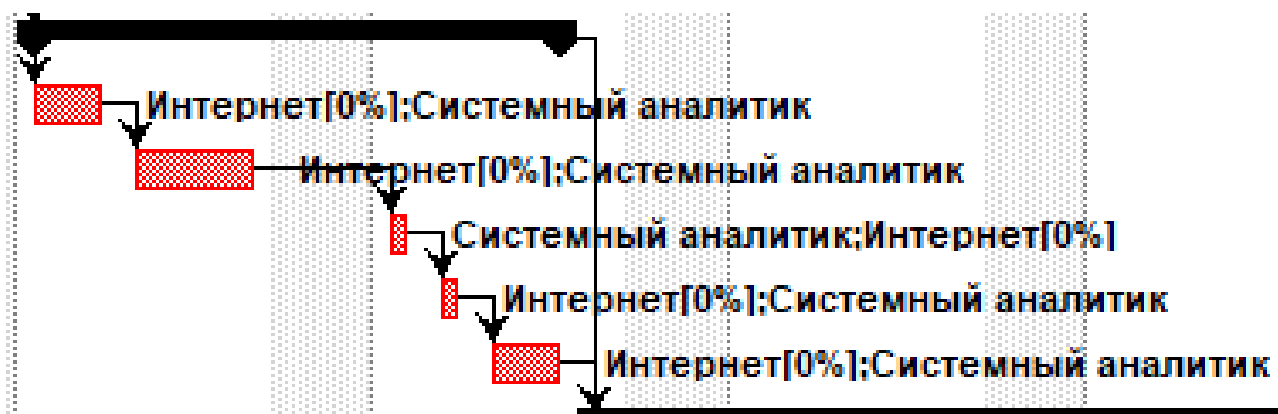


Рисунок 9 – Вторая часть диаграммы Ганта

В рамках проектирования ИС решаются задачи по выявлению функциональных возможностей системы и созданию архитектуры программы. К тому же происходит выбор средств разработки.

Третья часть диаграммы Ганта оценивает задачи, связанные с разработкой информационной системы. Данная часть показана на рисунке 10.

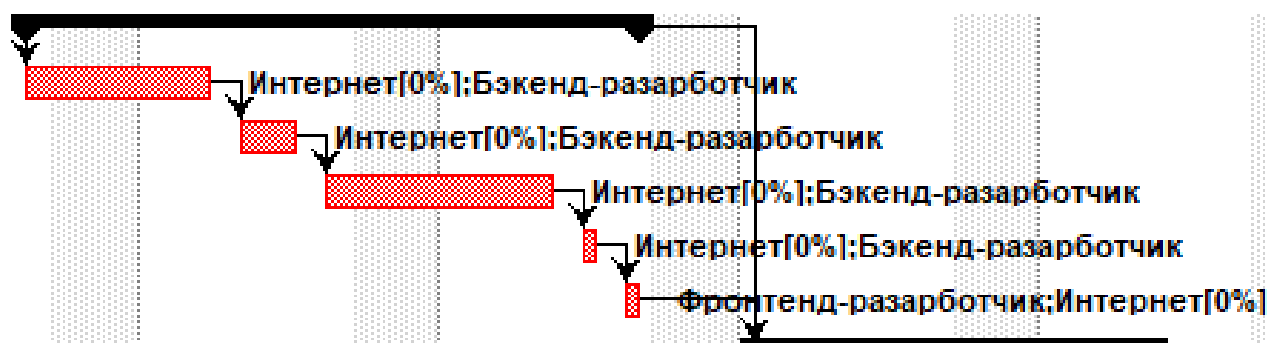


Рисунок 10 – Третья часть диаграммы Ганта

Набор задач этапа разработки включает в себя создание серверной части ИС, разработку микросервисов и клиентского приложения.

Последняя, четвертая часть диаграммы Ганта, показанная на рисунке 11, рассматривает задачи тестирования и оформления пояснительной записки, после которых завершается этап разработки ИС.

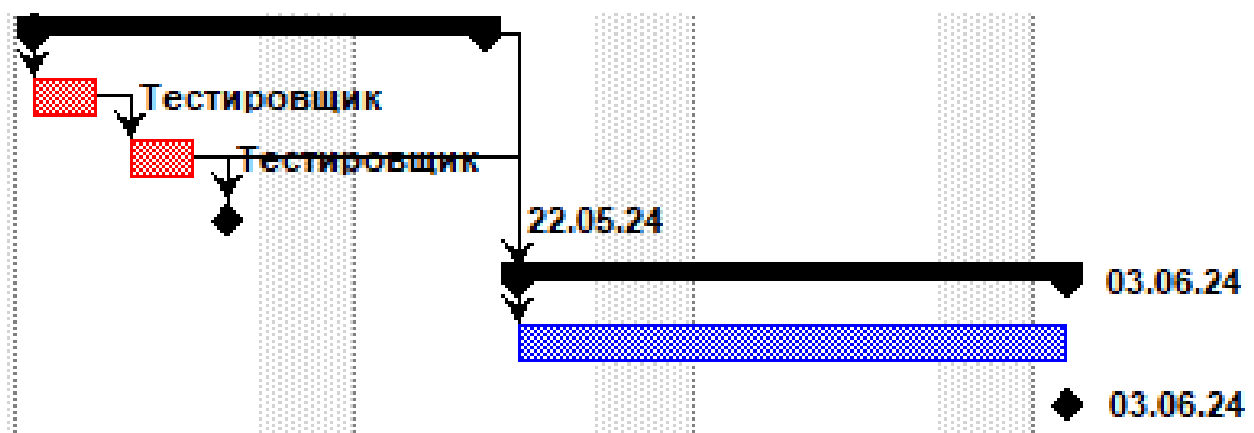


Рисунок 11 – Четвертая часть диаграммы Ганта

Последовательное исполнение данных частей диаграммы Ганта и позволит обеспечить спланированный процесс разработки информационной системы.

1.4.3 Оценка временных и финансовых затрат информационной системы

Оценка времени выполнения поставленных задач на разработку информационной системы позволит точнее определить, какой объем работ предстоит проделать на каждом этапе и какие потребуются сроки, а также будет возможно выделить основные ресурсы, используемые в работе. Это позволит минимизировать многие риски, связанные с превышением запланированного бюджета и установленными сроками. Подобная оценка также проводится для всех этапов, от анализа до тестирования. Все это позволит подсчитать итоговую стоимость процесса разработки, а также узнать временные рамки данного процесса.

Оценка временных затрат процесса разработки показана на рисунке 12.

Название	Начало	Окончание	Продолжительн...
Старт проекта	08.04.24 9:00	08.04.24 9:00	0 дней
Анализ предметной области и постановка зад	08.04.24 9:00	12.04.24 18:00	5 дней
Анализ предметной области информационной систе	08.04.24 9:00	09.04.24 18:00	2 дней
Сравнительный анализ приложений-аналогов	10.04.24 9:00	10.04.24 18:00	1 день
Постановка задачи на разработку информационной	11.04.24 9:00	11.04.24 18:00	1 день
Планирование и оценка процесса разработки инфс	12.04.24 9:00	12.04.24 18:00	1 день
Проектирование единой информационной сист	15.04.24 9:00	25.04.24 18:00	9 дней
Моделирование функциональных возможностей ин	15.04.24 9:00	16.04.24 18:00	2 дней
Разработка концептуальной модели информаци	17.04.24 9:00	19.04.24 18:00	3 дней
Разработка программной архитектуры информаци	22.04.24 9:00	22.04.24 18:00	1 день
Разработка требований к аппаратно-программному	23.04.24 9:00	23.04.24 18:00	1 день
Обоснование выбора средств разработки	24.04.24 9:00	25.04.24 18:00	2 дней
Разработка программного обеспечения	26.04.24 9:00	10.05.24 18:00	11 дней
Разработка программного обеспечения REST-сервер	26.04.24 9:00	30.04.24 18:00	3 дней
Разработка программного обеспечения микросерви	01.05.24 9:00	02.05.24 18:00	2 дней
Разработка программного обеспечения микросерви	03.05.24 9:00	08.05.24 18:00	4 дней
Разработка программного обеспечения микросерви	09.05.24 9:00	09.05.24 18:00	1 день
Разработка программного обеспечения клиентского	10.05.24 9:00	10.05.24 18:00	1 день
Контроль качества программного обеспечения	13.05.24 9:00	22.05.24 18:00	8 дней
Планирование процесса контроля качества програм	13.05.24 9:00	14.05.24 18:00	2 дней
Тестирование программных средств	15.05.24 9:00	16.05.24 18:00	2 дней
Оценка качества программного кода на основе ме	17.05.24 9:00	22.05.24 18:00	4 дней
Оформление пояснительной записки	23.05.24 9:00	03.06.24 18:00	8 дней
Оформление отчета	23.05.24 9:00	03.06.24 18:00	8 дней
Завершение проекта	03.06.24 18:00	03.06.24 18:00	0 дней
			Итого: 41 день

Рисунок 12 – Оценка временных затрат

В результате расчета было выявлено, что общий процесс разработки ИС, начиная с анализа и заканчивая тестированием, займет 41 день.

Также была проведена оценка финансовых затрат на разработку информационной системы, основанная на ресурсах, используемых в процессе реализации. Данные ресурсы показаны на рисунке 13.

Название	Стандартная ставка
Бэкенд-разработчик	400/час
Фронтенд-разработчик	350/час
Системный аналитик	300/час
Тестирующий	220/час
Компьютер	50/час
Интернет	500/месяц

Рисунок 13 – Ресурсы проекта

Оценка финансовых затрат с использованием данных ресурсов показана на рисунке 14.

Название	Оставшаяся стоимость
Старт проекта	0 руб.
Анализ предметной области и постановка задач на разработку	12000,04 руб.
Анализ предметной области информационной системы	4800 руб.
Сравнительный анализ приложений-аналогов	2400,04 руб.
Постановка задачи на разработку информационной системы	2400 руб.
Планирование и оценка процесса разработки информационной системы	2400 руб.
Проектирование единой информационной системы медицинских орга	21600,16 руб.
Моделирование функциональных возможностей информационной системы	4800,06 руб.
Разработка концептуальной модели информационной системы	7200,06 руб.
Разработка программной архитектуры информационной системы	2400,01 руб.
Разработка требований к аппаратно-программному окружению	2400,01 руб.
Обоснование выбора средств разработки	4800,02 руб.
Разработка программного обеспечения	34800,2 руб.
Разработка программного обеспечения REST-сервера	9600,02 руб.
Разработка программного обеспечения микросервиса пациента	6400,06 руб.
Разработка программного обеспечения микросервиса врача	12800,08 руб.
Разработка программного обеспечения микросервиса администратора	3200,01 руб.
Разработка программного обеспечения клиентского приложения	2800,03 руб.
Контроль качества программного обеспечения	14080 руб.
Планирование процесса контроля качества программного обеспечения	3520 руб.
Тестирование программных средств	3520 руб.
Оценка качества программного кода на основе метрик	7040 руб.
Оформление пояснительной записки	0 руб.
Оформление отчета	0 руб.
Завершение проекта	0 руб.
Итого: 84480,4 руб.	

Рисунок 14 – Оценка финансовых затрат

В результате оценки итоговой стоимости проекта была получена таблица, содержащая информацию как о стоимости каждой отдельной задачи, так и общую сумму затрат на разработку. Это позволит более наглядно увидеть самые сложные и затратные задачи, чтобы уделить им особое внимание для достижения успешного результата разработки.

По результатам оценки итоговая стоимость проекта составит 84480 рублей.

2 Проектирование единой информационной системы медицинских организаций города

2.1 Моделирование функциональных возможностей информационной системы

Проведение моделирования функциональных возможностей информационной системы является важным и комплексным процессом, в котором можно выделить следующие этапы:

- выделение и анализ основных функциональных возможностей в информационной системе;
- определения основных частей и групп функциональных возможностей;
- выделение связи между описанными функциональными возможностями.

Данная диаграмма позволяет в наглядной форме представить работу и возможности информационной системы, а также облегчает последующее проектирование архитектуры программного обеспечения, так как уже на этом этапе выделяются основные роли в системе, ее составные части, реализуемый функционал и взаимодействие компонентов.

Продуманная и правильно спроектированная модель является важной частью разрабатываемого программного обеспечения, так как она используется на всех основных этапах его разработки.

В рамках данного раздела были спроектированы диаграммы вариантов использования для всех основных частей разрабатываемой информационной системы, описывающие ее функциональные возможности и составляющие их части.

На рисунке 15 показана диаграмма вариантов использования приложения пациента.

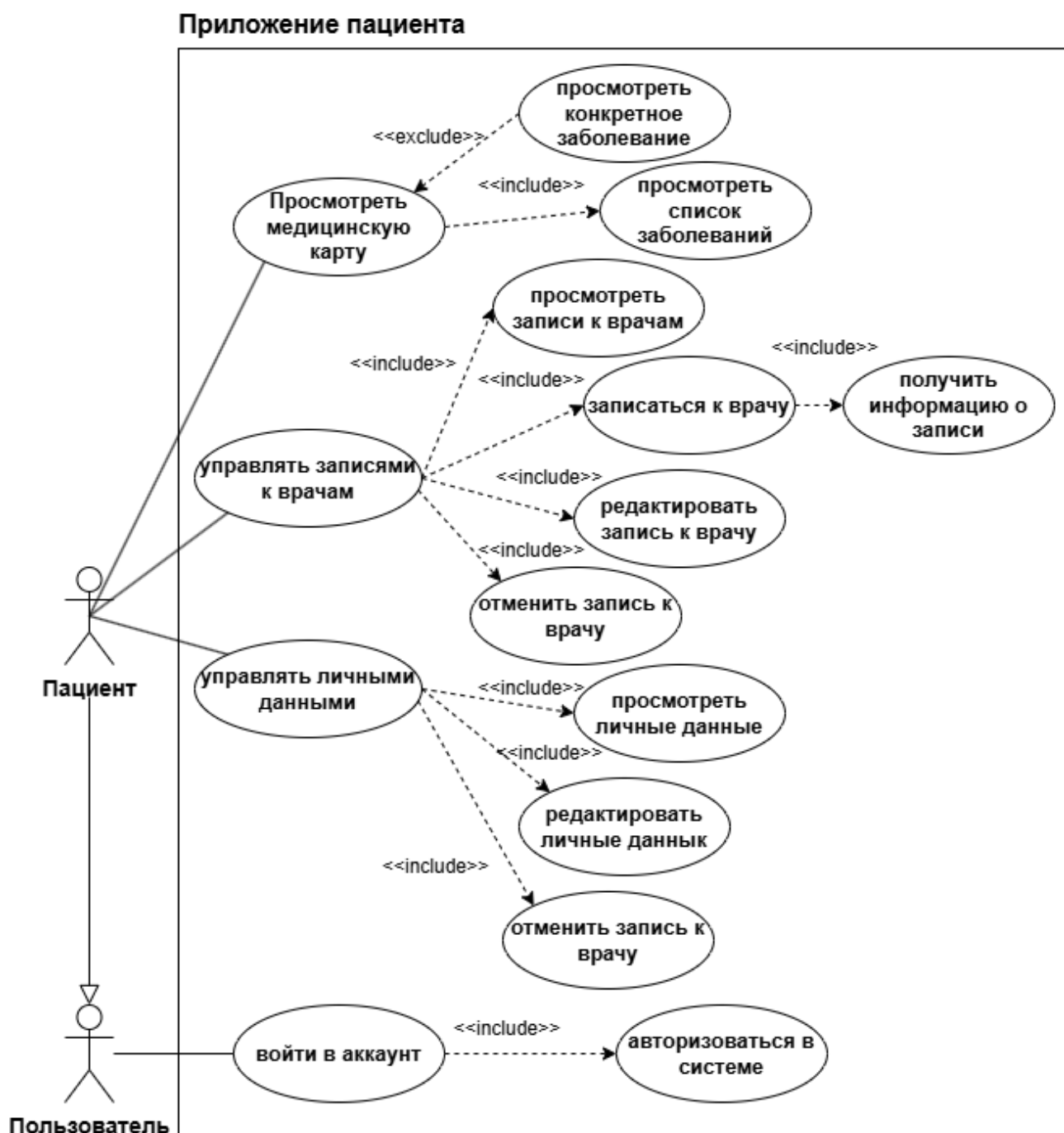


Рисунок 15 – Диаграмма вариантов использования приложения пациента

Основной функционал приложения пациента заключается в управлении своими личными данными, записями к врачу и просмотром личной медицинской карты. Подобный набор возможностей позволяет дать пациенту самый оптимальный набор возможностей для комфортной работы в рамках информационной системы.

На рисунке 16 показана диаграмма вариантов использования приложения врача.

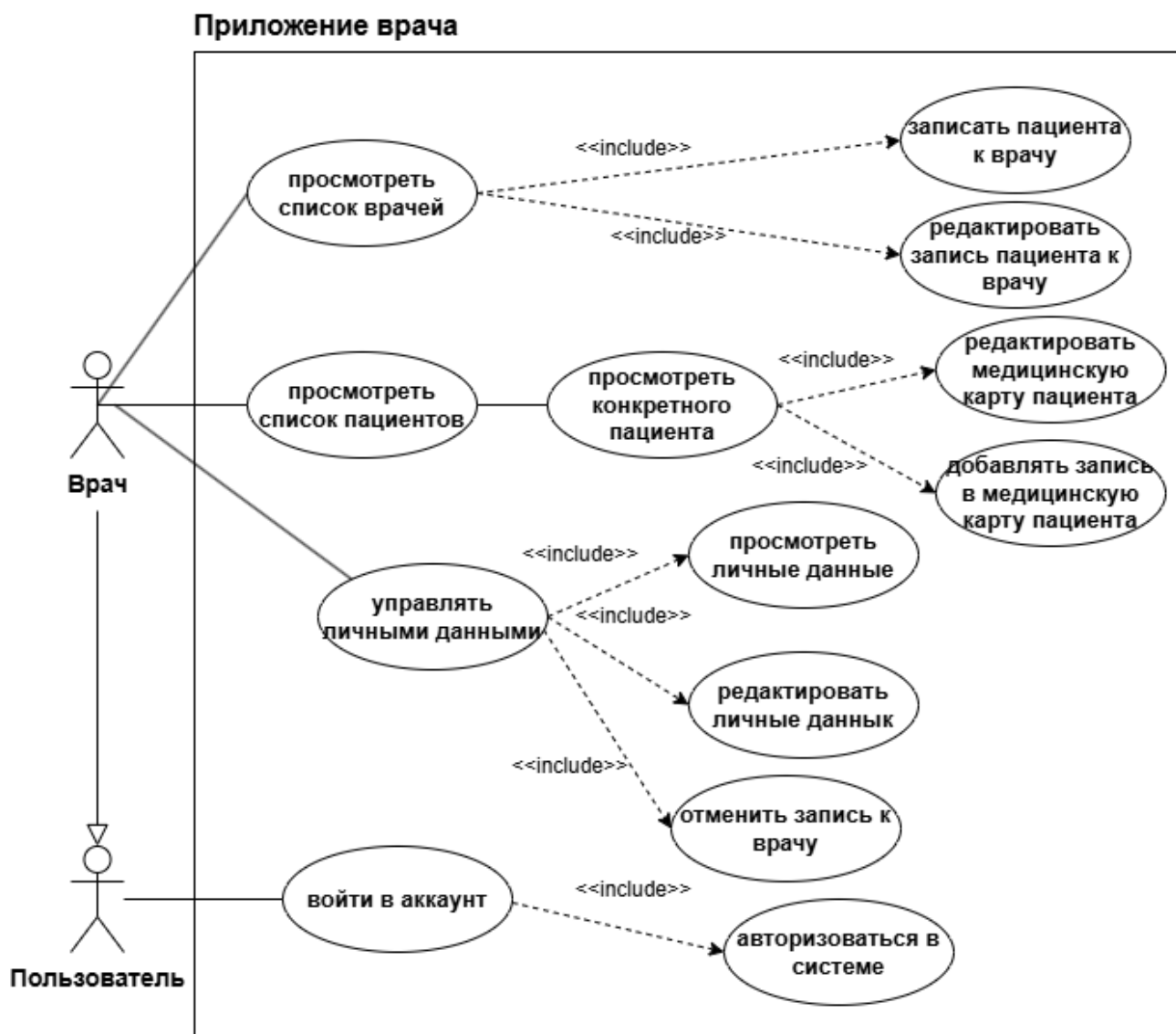


Рисунок 16 – Диаграмма вариантов использования приложения врача

Функционал приложения врача расширяется рядом возможностей, которые позволяют ему работать с пациентами. В этот набор входит управление записями пациента к врачу, что позволяет как создавать новые записи, так и редактировать уже существующие, а также возможность просмотра и изменения медицинских карт пациентов для внесения в нее новых записей и ознакомления с ее содержимым при проведении обследования.

На рисунке 17 показана диаграмма вариантов использования приложения администратора.

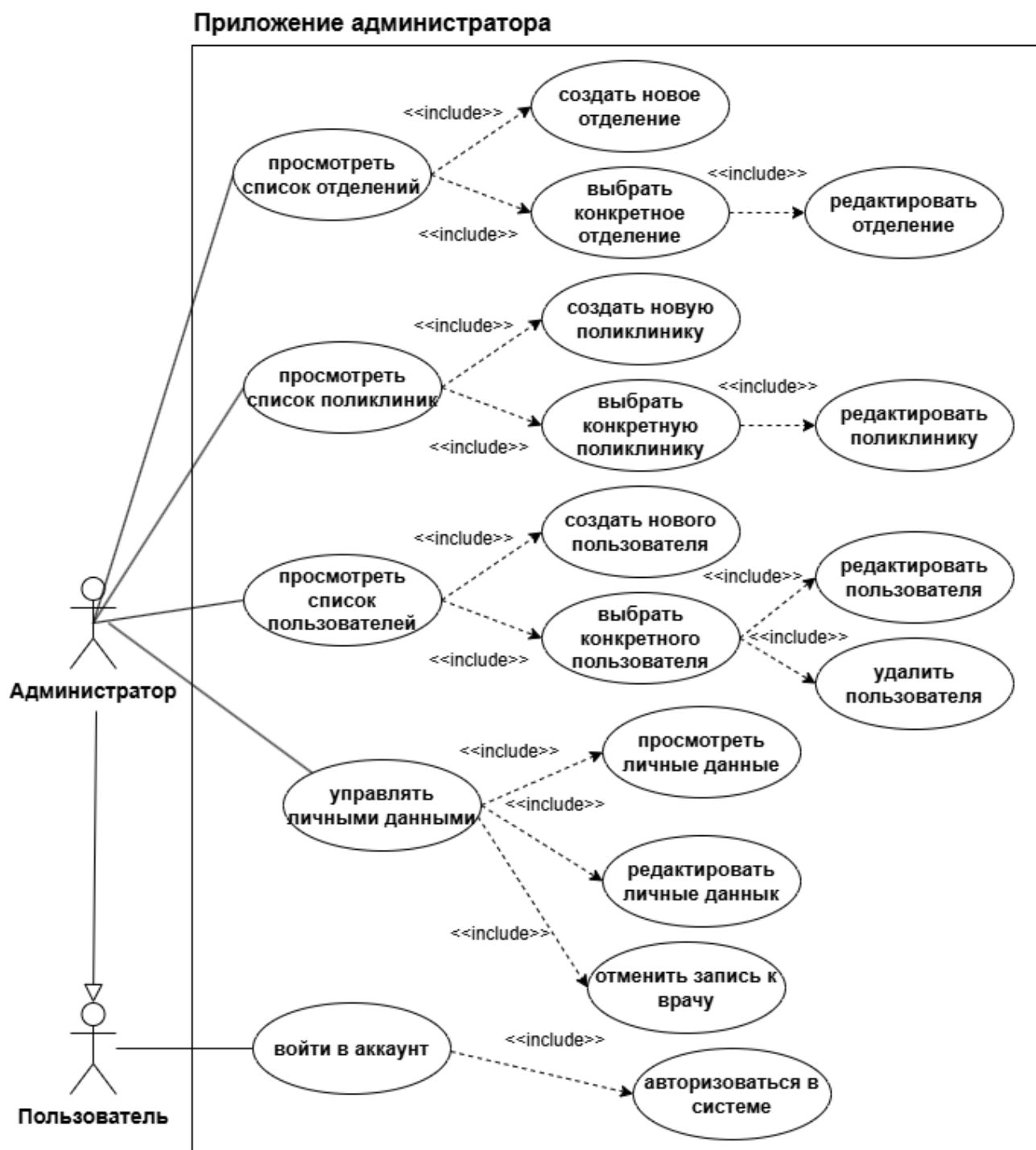


Рисунок 17 – Диаграмма вариантов использования приложения администратора

Администратор в рамках системы занимается задачами добавления и изменения сущностей. Именно пользователь с данной ролью является ответственным за создание и деактивацию пользователей, а также управлением данными об отделениях и поликлиниках.

Для приведенных диаграмм вариантов использования были созданы таблицы Б.1 – Б.3, описывающие их спецификации.

2.2 Разработка концептуальной модели информационной системы

Концептуальная модель базы данных – это абстрактное представление о данных, их структуре и взаимосвязях, которое используется для проектирования базы данных. Она описывает сущности и их атрибуты, а также связи между этими сущностями [10].

Концептуальная модель базы данных обычно создается на начальном этапе проектирования базы данных и служит основой для разработки физической модели. Она помогает определить основные объекты данных, их характеристики и взаимосвязи, что позволяет лучше понять предметную область и оптимизировать структуру базы данных.

Создание концептуальной модели базы данных является важным этапом проектирования базы данных, поскольку она определяет основные принципы организации данных и структуру БД. Сама модель показана на рисунке 18.

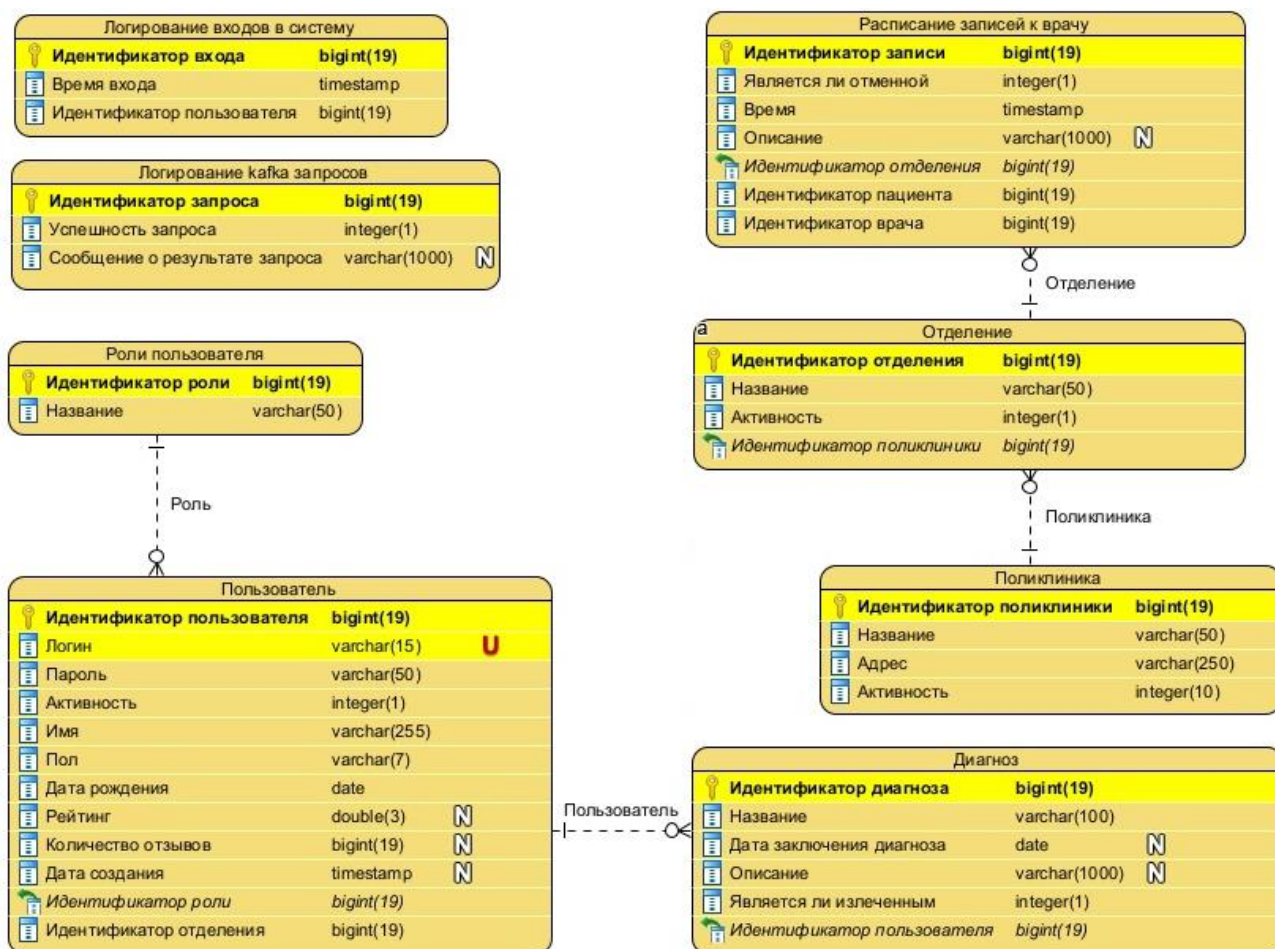


Рисунок 18 – Концептуальная модель БД администратора

Получившаяся в результате модель предусматривает все необходимые поля и таблицы для хранения информации о пользователях, поликлиниках и записях к врачу. Также предусмотрено две таблицы для хранения логирования.

В дальнейшем концептуальная модель базы данных будет использована как основа для реализации физической модели.

2.3 Разработка программной архитектуры информационной системы

Важной частью архитектуры разрабатываемого приложения является выбор микросервисной архитектуры, а также разделения программы на клиентскую и серверную часть.

Так как все микросервисы серверного приложения работают на разных портах сервера, важно упростить взаимодействие клиентского приложения с сервером таким образом, чтобы оно могло посылать все запросы на один порт. Для этого один из существующих микросервисов будет исполнять функцию шлюза, распределяющего запросы в другие микросервисы.

Для взаимодействия приложения клиента с сервером был выбран архитектурный стиль REST, который позволяет спроектировать понятное и масштабируемое взаимодействия в рамках использования протокола HTTP. Также использование REST API позволит удобно реализовать использование и работу с сущностями программы, предоставив все необходимые методы для их получения, модификации, удаления и добавления.

Реализацию взаимодействия микросервисов друг с другом также было решено построить на технологии REST API для простых запросов по получению данных. Для запросов, подразумевающих изменение или добавление данных, была выбрана технология брокера сообщений, чтобы гарантировать доставку запроса и сохранение отправленных сообщений. Также важной частью в реализации взаимодействия между микросервисами в данной информационной системе является обработка результата запроса, чтобы в случае возникновения

ошибки было возможно отобразить правильное сообщение пользователю, описывающее причину или место ошибки.

Для реализуемой ИС была разработана обобщенная архитектурная схема, показанная на рисунке 19.

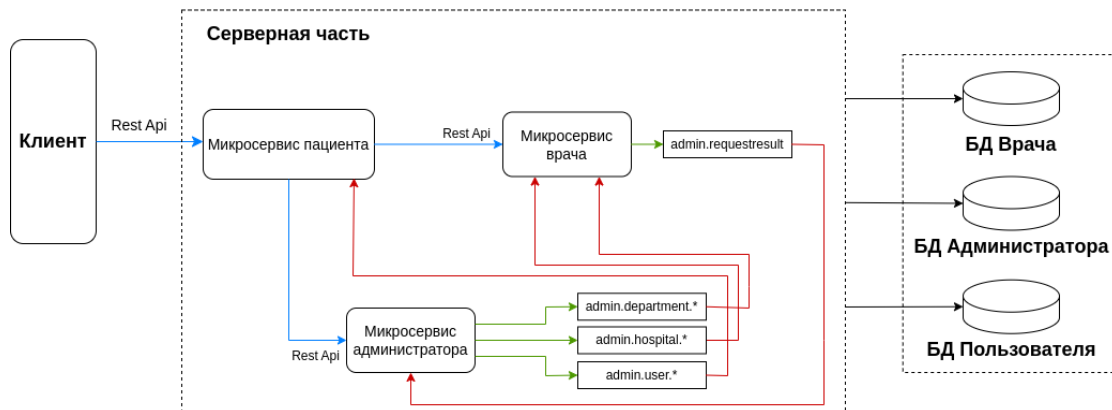


Рисунок 19 – Обобщенная архитектурная схема информационной системы

В рамках данной обобщенной архитектурной схемы приложения можно рассмотреть основные пути взаимодействия между разными её компонентами, включая работу клиентского приложения с сервером, а также взаимодействие микросервисов между собой.

Также была создана микросервисная архитектурная схема для данной информационной системы, показанная на рисунке 20.

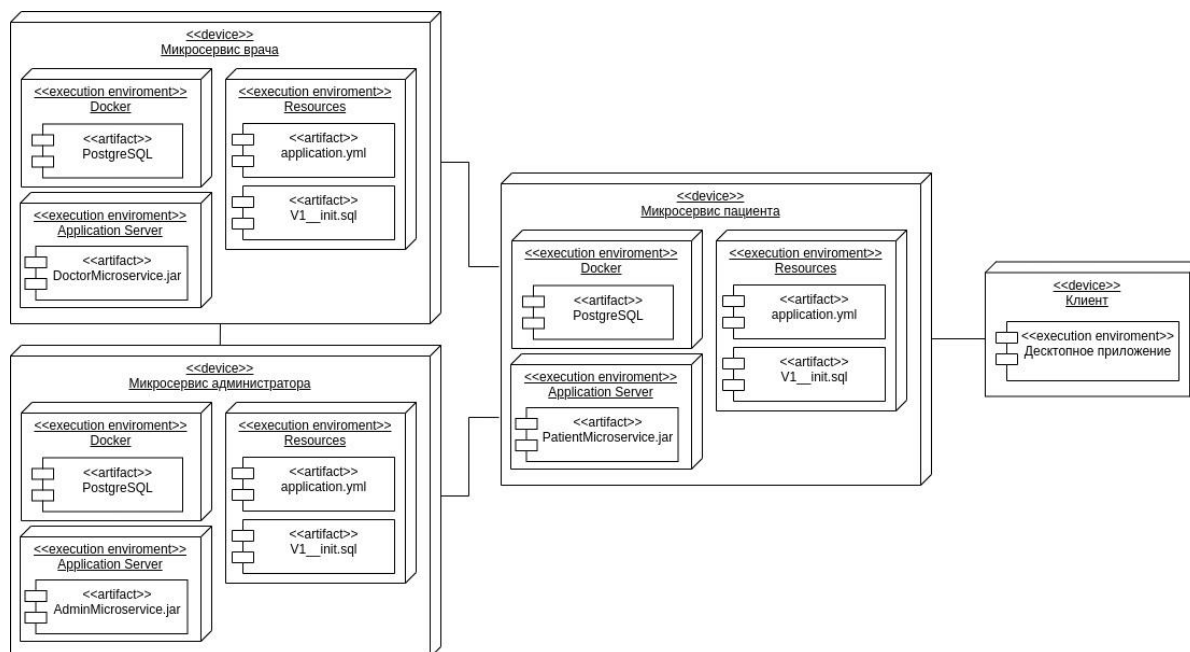


Рисунок 20 – Микросервисная архитектурная схема

На данной схеме отображено, на основе какой структуры строятся микросервисы информационной системы и их связи между собой и клиентской частью.

2.4 Разработка требований к аппаратно-программному обеспечению

Определение и описание требований к аппаратно-программному обеспечению – это один из важных этапов при разработке информационной системы. Данные требования дают оценку, насколько компьютер пользователя совместим с работой программного обеспечения. Определение этих требований позволит обеспечить правильность работы клиентской программы информационной системы. Требования для работы клиентской части приложения показаны в таблице 5.

Таблица 5 – Требования к клиентскому аппаратно-программному обеспечению

Компонент	Минимальные требования	Рекомендуемые требования
Операционная система	Windows 10	Windows 10
Процессор	Intel Core i3-2100	Intel Core i5-8400
Оперативная память	4 Гб	8 Гб
Жесткий диск	20 мегабайт свободного пространства	20 мегабайт свободного пространства
Установленное ПО	Фреймворк .NET	Фреймворк .NET

С точки зрения технических характеристик компьютер должен соответствовать минимальным требованиям, применяемым для работы с современным программным обеспечением. Дополнительно на компьютер пользователя должна быть установлена операционная система Window 10 или выше, а также фреймворк .NET для поддержки работы клиентской части приложения.

Также были составлены требования для работы серверной части приложения, показанные в таблице 6.

Таблица 6 – Требования к серверному аппаратно-программному обеспечению

Компонент	Минимальные требования	Рекомендуемые требования
Операционная система	Ubuntu	Ubuntu
Процессор	Intel Xeon Bronze 3204 OEM	Intel Xeon Silver 4214R OEM
Оперативная память	8 Гб	64 Гб
Жесткий диск	4 гигабайта свободного пространства	256 гигабайт свободного пространства
Установленное ПО	Docker	Docker

Соответствие аппаратно-программного обеспечения данным требованиям позволяет обеспечить эффективную и корректную работу как клиентского, так и серверного программного обеспечения.

2.5 Обоснование выбора средств разработки

2.5.1 Обоснование выбора средств автоматизации

Использование средств автоматизации при разработке программного обеспечения позволяет улучшить и упростить процесс разработки на всех этапах, начиная от анализа и проектирования и заканчивая тестированием и сопровождением. Такие средства позволяют создавать документацию, строить диаграммы, анализировать код и тестировать приложение. Благодаря этому выбор средств автоматизации является важным этапом. В рамках создания информационной системы был рассмотрен ряд подобных средств, в результате чего были выбраны инструменты Visual Paradigm и Postman. Эти программные средства обладают широким функционалом, помогающим совершенствовать процесс на разных этапах разработки.

Программное обеспечение Postman является одним из самых популярных инструментов для отладки API серверных приложений. Он обладает интуитивно понятным графическим интерфейсом, возможностью создавать рабочие пространства для совместного тестирования и поддерживает взаимодействие не только по протоколу HTTP, но и по WebSocket, gRPC и другим. Также присутствует функционал создания автоматизированных тестов. Данные

преимущества делают инструмент Postman очень полезным инструментом на стадии тестирования программного обеспечения [11].

Инструмент Visual Paradigm активно используется на стадии проектирования информационной системы и составления документации. Эта программа предоставляет большой набор функционала для создания самых разных диаграмм, в том числе диаграмму классов, базы данных, архитектурную схему и многие другие. Также данное программное обеспечение поддерживает стандарт UML и SysML. Проектирование диаграмм, отображающих структуру и взаимодействие компонентов программного обеспечения, является важным шагом при разработке информационной системы. Именно благодаря широкому функционалу Visual Paradigm, способному решить эти задачи при разработке информационной системы, данный инструмент является одним из лучших среди доступных решений [12].

2.5.2 Обоснование выбора инструментальных средств разработки программного обеспечения

В процессе работы над ИС был рассмотрен ряд инструментальных средств разработки ПО, в результате чего были выбраны два основных инструмента: IntelliJ IDEA и Visual Studio. Если среда разработки IntelliJ IDEA используется для разработки серверной части приложения на языке Kotlin, Visual Studio необходим для создания клиентской части на языке C#.

IntelliJ IDEA уже много лет считается стандартом для профессиональной разработки приложений на таких языках, как Java и Kotlin. Данная среда разработки предоставляет широкие возможности для персонализации ее внешнего вида и работы с ней, в том числе за счет дополнительных плагинов, которые позволяют добавить дополнительный функционал или поддержку определенных технологий. Помимо этого, IntelliJ IDEA предоставляет инструменты для работы с базами данных, системами версионирования, а также для отладки и тестирования приложений [13]. Процесс написания программы сильно упрощается благодаря технологии обнаружения многих синтаксических

и логических ошибок еще на этапе написания программного кода. Эти и многие другие преимущества делают среду разработки IntelliJ IDEA отличным инструментом для разработки программного обеспечения на языке программирования Kotlin.

В свою очередь Visual Studio является самым популярным инструментом для разработки приложений на языке C#. Данная среда разработки обладает удобным и понятным интерфейсом, а также предоставляет весь необходимый функционал для комфортной разработки мобильных, десктопных, а также веб-приложений. Сильной стороной данного инструмента является наличие конструктора интерфейса разрабатываемого приложения, что является очень важным преимуществом при разработке клиентской части приложения информационной системы [14].

На основании рассмотренных преимуществ данных средств разработки можно сделать вывод, что их использование позволит существенно упростить разработку программного обеспечения.

2.5.3 Обоснование выбора языковых средств разработки

При выборе языковых средств разработки ИС были выбраны два языка программирования: Kotlin и C#. Язык Kotlin необходим для разработки серверной части приложения, где важную роль в этом процессе играет фреймворк Spring. C# используется для реализации клиентской части приложения, в процессе создания которой задействуется фреймворк .NET. Выбор данных языковых средств будет далее обоснован.

Kotlin является объектно-ориентированным языком программирования, обладающим большой популярностью среди разработчиков, особенно в сфере мобильной и бэкенд разработки. Важной особенностью языка является полная совместимость с Java, где за много лет было разработано множество библиотек и фреймворком, многие из которых позволяют создавать масштабируемые и безопасные приложения. Язык Kotlin также поддерживает корутины, позволяя очень легко реализовывать параллельные вычисления.

Фреймворк Spring, популярный при разработке Java и Kotlin программ, предлагает большой набор разнообразных инструментов и библиотек для разработки. Spring берет в свои руки многие низкоуровневые задачи, возникающие при создании приложений, позволяя сконцентрироваться на конкретной бизнес-логике [15]. Сам фреймворк разделен на ряд модулей:

- Data Access/Integration, реализующий поддержку работы с базами данных;
- Web, предоставляющий инструменты для создания веб-сервера;
- Core Container, содержащий базовую логику фреймворка;
- Test, используемый для создания тестового окружения и написания автоматизированных тестов.

Описанные преимущества языка Kotlin и его фреймворка Spring делают использование данных языковых средств наиболее подходящим для разработки серверной части приложения.

Еще один объектно-ориентированный язык программирования C#, используемый для создания клиентской части, обладает такими преимуществами, как функциональное программирование, автоматизированная работа с памятью и динамическая типизация. Важной частью данного языка является наличие ряда библиотек для создания программного интерфейса, таких как Windows Form и WPF. Его фреймворк .NET предоставляет разработчику набор инструментов для создания разнообразных приложений, в том числе давая возможность реализовать взаимодействие с веб-сервером [16]. Также фреймворк предлагает единую среду исполнения и поддерживает кроссплатформенность.

Все эти возможности делают C# и фреймворк .NET оптимальным выбором для создания клиентской части приложения.

2.5.4 Обоснование выбора средств управления данными

В качестве основного средства для управления данными в информационной системе был рассмотрен ряд средств для решения данной

задачи, в результате чего в качестве используемого инструмента была выбрана система управления базами данных PostgreSQL.

Одним из важнейших преимуществ данной СУБД является наличие открытого исходного кода, что позволяет как удостовериться в высокой надежности и безопасности PostgreSQL, так и при необходимости внести какие-либо изменения в данный код. СУБД поддерживает все необходимые функции для управления, создания и обработки данных. В этот набор входят такие механизмы, как поддержка транзакций и настройка их уровня изоляции, использование индексов и процедур.

Другой особенностью PostgreSQL является высокая масштабируемость и эффективная работа с большими объемами данных. Все это достигается благодаря оптимизированной работе с данными и поддержке механизма горизонтального масштабирования. Это делает данную СУБД способной к работе при растущей нагрузке без проявления особых задержек в работе системы [17].

PostgreSQL является бесплатным программным обеспечением и окружает себя большим сообществом активных пользователей участвующих в развитии данного проекта благодаря внесению новой функциональности в СУБД, исправлению программных ошибок и улучшению документации.

Так как данная СУБД поддерживается фреймворком Spring для работы с хранимыми в ней данными через технологию ORM, это позволяет представить каждую таблицу базы данных как отдельный Java класс, что позволяет напрямую работать с хранимыми кортежами через программный код.

Данные особенности средства управления данными PostgreSQL делают ее подходящим инструментом для хранения и управления данными, что достигается благодаря большим возможностям в масштабируемости и надежности.

3 Разработка программного обеспечения единой информационной системы медицинских организация города

3.1 Разработка программного обеспечения REST-сервера

Для реализации обработки REST запросов в программе созданы специальные интерфейсы с описанной спецификой запроса. Реализации данных интерфейсов и занимаются дальнейшей обработкой того, что было получено со стороны клиента. Пример описания обработчика запросов показан в листинге 1.

Листинг 1 – Пример описания обработчика запросов

```
@Tag(name = "Записи о поликлиниках")
@RequestMapping("/api/hospitals")
interface HospitalApi {
    @Operation(
        description = "Получение списка поликлиник"
    )
    @GetMapping
    fun getHospitals(): List<HospitalRsDto>
    @Operation(description = "Получение поликлиники по id")
    @GetMapping("/{id}")
    fun getHospitalById(@PathVariable id: Long):
        HospitalRsDto
}
```

А далее данный интерфейс реализует класс обработчика запросов. Пример реализации показан в листинге 2.

Листинг 2 – Пример реализации обработчика запросов

```
@RestController
class HospitalController(
    private val hospitalService: HospitalService
) : HospitalApi {
    override fun getHospitals():
        List<HospitalRsDto> = hospitalService.getHospitals()
    override fun getHospitalById(id: Long):
        HospitalRsDto = hospitalService.getHospitalById(id)
}
```

Подобное разделение кода программы на блоки с описанием структуры и конкретной реализации позволяет легко заменять итоговый код при необходимости и выделить в отдельное место те аннотации, которые обеспечивают документирование кода.

В результате был реализован REST API для работы с серверной частью приложения. Описание основных методов показано в таблице 7.

Таблица 7 – Основные методы REST API

URL запроса	Http метод	Описание	Query параметры запроса
/api/users/{id}	GET	Получение пользователя по его id	Id – идентификатор пользователя
/api/users/{id}	PUT	Обновление записи о пользователе	Id – идентификатор пользователя
/api/users	POST	Добавление нового пользователя	-
/api/hospitals/{id}	GET	Получение поликлиники по ее id	Id – идентификатор поликлиники
/api/hospitals/{id}	PUT	Обновление записи о поликлинике	Id – идентификатор поликлиники
/api/hospitals	POST	Добавление новой поликлиники	-
/api/departments/{id}	GET	Получение отделения по его id	Id – идентификатор отделения
/api/departments/{id}	PUT	Обновление записи об отделении	Id – идентификатор отделения
/api/departments	POST	Добавление нового отделения	-
/api/diagnoses/{id}	GET	Получение диагноза по его id	Id – идентификатор диагноза
/api/diagnoses/{id}	PUT	Обновление записи о диагнозе	Id – идентификатор диагноза
/api/diagnoses	POST	Добавление нового диагноза	-
/api/appointments/{id}	GET	Получение записи к врачу по ее id	Id – идентификатор записи к врачу
/api/appointment/{id}	PUT	Обновление записи о записи к врачу	Id – идентификатор записи к врачу
/api/appointment	POST	Добавление новой записи к врачу	-

Реализация данных методов позволит обеспечить удобное и понятное взаимодействие с серверной частью приложения.

3.2 Разработка микросервисов

3.2.1 Разработка структуры микросервисов

В процессе работы над серверной частью информационной системы была создана и реализована общая структура микросервисов. Рассмотрим ее на рисунке 21 на примере микросервиса врача.

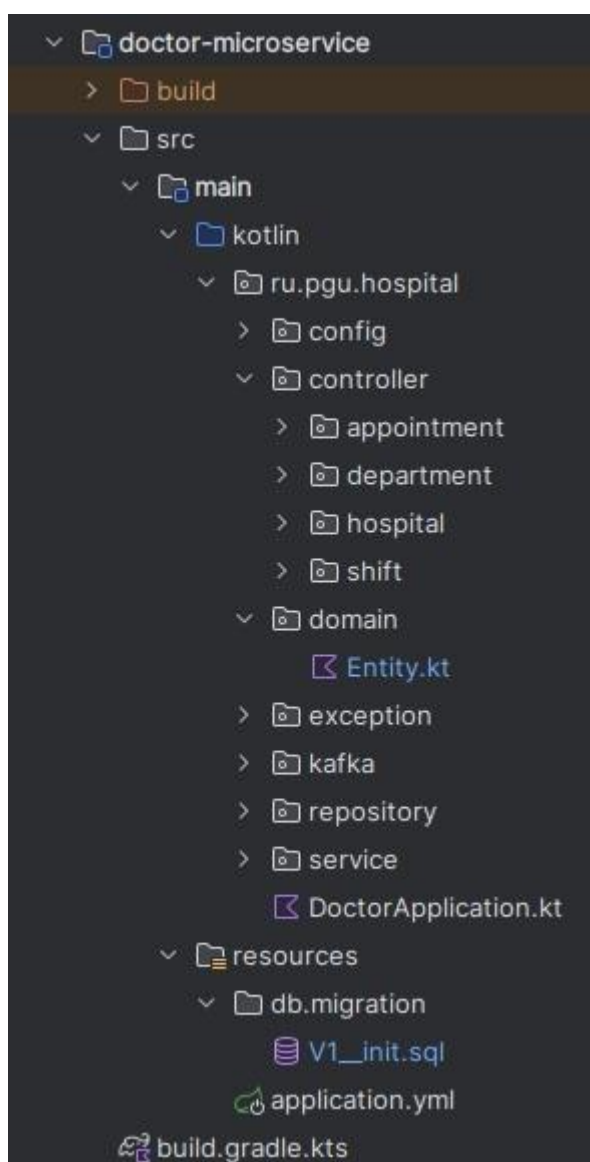


Рисунок 21 – Структура микросервиса врача

В данной структуре можно выделить ее основные части:

- пакет Kotlin, содержащий исходный код микросервиса;

- пакет resources, хранящий код инициализации БД и настройки;
- файл build.gradle.kts.

Рассмотрим подробнее каждую из этих частей.

Пакет kotlin включает в себя множество других пакетов, необходимых для распределения исходного кода в специальные группы, определяющие их назначение. Так, пакет config содержит классы, необходимые для описания конфигураций, как это делает класс KafkaConfig, который определяет основные настройки используемого брокера сообщений. Пакет controller используется для хранения классов, являющихся обработчиками HTTP запросов, а пакет domain хранит представления таблиц базы данных в виде классов. Exception используется для обработки ошибок, а классы в пакте kafka позволяют отправлять и принимать сообщения в одноименном брокере. Пакет service содержит бизнес-логику микросервиса и использует файлы из repository для работы с базой данных. Наконец, имеется главный файл DoctorApplication.kt, необходимый для запуска программы.

Другой пакет, resources, состоит из пакета db.migration и файла application.yml. Представленный пакет необходим для хранения миграций базы данных в виде sql скриптов, что позволяет сохранить историю изменений БД и легко вносить в нее изменения. Файл application.yml хранит самые главные настройки микросервиса, а именно данные для подключения к базе данных и рабочий порт. Содержание данного файла показана на рисунке 22.

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/doctor_db
    username: doctor_db
server:
  port: 8081
```

Рисунок 22 – Содержание файла application.yml

Последней важной частью в структуре микросервиса является файл `build.gradle.kts`, который используется системой сборки Gradle для конфигурации проекта. Это позволяет легко подключать сторонние библиотеки и плагины в проект, а также настраивать его процесс сборки. Часть данного файла, именуемая `dependencies`, используемая для подключения сторонних библиотек, показана на рисунке 23.

```
dependencies { Edit Starters... this: DependencyHandlerScope
    implementation("org.springframework.boot:spring-boot-starter-validation")
    implementation("org.jetbrains.kotlin:kotlin-maven-noarg:1.9.0")
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("javax.persistence:javax.persistence-api:2.2")
    implementation("org.springdoc:springdoc-openapi-starter-webmvc-ui:2.1.0")
    implementation("org.springframework.data:spring-data-envers")
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("org.flywaydb:flyway-core")
    implementation("io.projectreactor.kafka:reactor-kafka:1.3.20")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-reactor")
    runtimeOnly("org.postgresql:postgresql")
    developmentOnly("org.springframework.boot:spring-boot-devtools")
    annotationProcessor("org.springframework.boot:spring-boot-configuration-processor")
    implementation("org.springframework.boot:spring-boot-starter-actuator")
    implementation("org.springframework.boot:spring-boot-starter-validation")
    annotationProcessor("org.springframework.boot:spring-boot-configuration-processor")
}
```

Рисунок 23 – Часть `dependencies` файла `build.gradle.kts`

Также важно отметить, что для работы всех представленных микросервисов, используется технология Docker, позволяющая запускать изолированные контейнеры, в роли которых здесь выступает база данных и брокер сообщений. Благодаря этому значительно упрощается развертывание приложения на системе, так как не требуется установка стороннего программного обеспечения, кроме самого Docker.

Рассмотрим часть используемого в программе Docker файла, показанную на рисунке 24, необходимую для развертывания трех баз данных для каждого из микросервиса.

```

version: "3.3"

services:
  myapp-postgresql:
    image: postgres:14
    volumes:
      - /home/codeinside/IdeaProjects/docker-postgresql-multiple-databases-master:/docker-entrypoint-initdb.d
    environment:
      POSTGRES_MULTIPLE_DATABASES: user_db,doctor_db,admin_db
      POSTGRES_HOST_AUTH_METHOD: trust
    ports:
      - "5432:5432"

```

Рисунок 24 – Docker образ для развертывания баз данных

Описание данного образа позволяет определить, какое название будет у баз данных, способ доступа к ним и их рабочий порт.

На рисунке 25 показано описание образов zookeeper и broker для включения Kafka в работу.

```

zookeeper:
  image: confluentinc/cp-zookeeper
  container_name: zookeeper
  ports:
    - "2181:2181"
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
broker:
  image: confluentinc/cp-kafka
  container_name: broker
  depends_on:
    - zookeeper
  ports:
    - "9092:9092"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:29092,PLAINTEXT_HOST://localhost:9092
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0

```

Рисунок 25 – Docker образы для развертывания брокера Kafka

Конфигурация данных образов позволит настроить как рабочие порты брокера, так и его общие параметры, как, например, настройка задержки ребалансировки.

3.2.2 Разработка программного обеспечения микросервиса пациента

Микросервис пациента разрабатывается с целью обеспечения работы и хранения информации о пользователях системы и диагнозах. Другой важной задачей данного сервиса является выполнения роли API шлюза. Это необходимо, так как все микросервисы системы работают на разных серверных портах и для обеспечения единой точки взаимодействия с информационной системой и был реализован данный механизм. Пример перенаправления запроса в данном микросервисе показан в листинге 3.

Листинг 3 – Пример перенаправления запроса в шлюзе

```
val adminHospitalsApi = "http://localhost:8082/api/hospitals"
val hospitalsApi = "http://localhost:8081/api/hospitals"
override suspend fun updateHospital(id: Long, updateDto:
AdminHospitalUpdateDto) {
    val existRequest = runCatching {
restTemplate.getForEntity("$hospitalsApi/$id", HospitalRsDto::class.java)
    }
    existRequest.getOrElse {
        if (it.message!!.contains("NOT_FOUND")) { throw
EntityNotFoundException("Поликлиника с id=$id не найдена") }
        throw MicroserviceRequestException("Произошла ошибка при запросе
в микросервис администратора")
    }
    restTemplate.exchange("$adminHospitalsApi/$id", HttpMethod.PUT,
HttpEntity<AdminHospitalUpdateDto>(updateDto), String::class.java)
}
```

В данном примере рассматривается метод для обновления данных о поликлинике. Сначала делается HTTP запрос в микросервис врача, где выясняем, существует ли данная поликлиника. Если поликлиника существует, то запрос на вызов операции ее обновления перенаправляется в микросервис администратора. А если требуемая поликлиника не будет найдена, то пользователю, сделавшему запрос на обновление, будет отправлено сообщение об ошибке. Данный подход позволяет предусмотреть все требуемые действия, если необходимая операция не может быть выполнена.

Также была реализована физическая модель базы данных, показанная на рисунке 26.

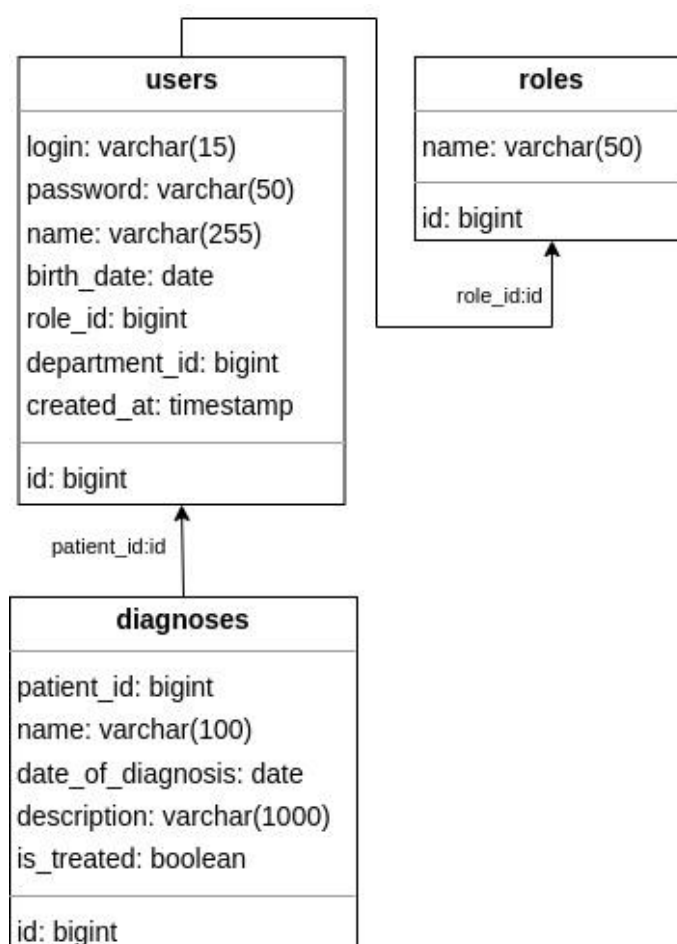


Рисунок 26 – Физическая модель базы данных микросервиса пациента

Другой важной особенностью микросервиса пациента является поддержание взаимодействия с другими микросервисами по Apache Kafka, что позволяет обеспечить гарантию доставки запросов для самых важных операций, вроде операции добавления или удаления.

Если в микросервис пациента по брокеру Kafka приходит запрос от микросервиса администратора на создание нового пользователя, то происходит процесс его формирования на основе пришедших данных. Если пользователь создается успешно, то администратору отправляет ответное сообщение об успехе, а иначе сообщение о неудаче.

3.2.3 Разработка программного обеспечения микросервиса врача

Разработка микросервиса врача осуществляется для поддержания функционала, взаимодействующего с данными о поликлиниках, отделениях и

записях к врачу. Для этого доступен ряд методов, позволяющий получить о них информацию. При этом, как и в случае с микросервисом пациента, часть операций, такие как удаление или обновление некоторых сущностей, осуществляется только посредством получения запроса по Apache Kafka от микросервиса администратора. Пример кода обработчика таких запросов показан в листинге 4.

Листинг 4 – Пример обработчика запросов в брокере Kafka

```
fun receiver(): Disposable =
    hospitalSaveReceiver.receive()
        .doOnNext { receivedMessage ->
            runCatching {
                mapper.readValue<HospitalSavePayload>(receivedMessage.value())
            }.onSuccess { saveRequest ->
                runBlocking {
                    val saveResult =
                        hospitalService.saveHospital(saveRequest)
                    if(saveResult.isSuccess) {
                        kafkaProducer.sendRequestResultPayload(RequestResultPayload(
                            requestId = saveRequest.requestId,
                            isRequestSuccessful = true,
                            message = null
                        ))
                    } else {
                        kafkaProducer.sendRequestResultPayload(RequestResultPayload(
                            requestId = saveRequest.requestId,
                            isRequestSuccessful = false, message = "Произошла ошибка при сохранении
поликлиники"
                        ))
                    }
                }
            }.also {
                receivedMessage.receiverOffset().acknowledge()
            }
        }.subscribe()
```

Данным методом получает запрос и сообщение от брокера, в котором хранятся данные, на основе которых требуется создать новую поликлинику. Эти данные передаются в специальный сервис, в котором происходит попытка сохранения новой поликлиники, после чего в микросервис администратора отправляется запрос, хранящий результат операции.

Для микросервиса также была создана физическая модель ее базы данных, показанная на рисунке 27.

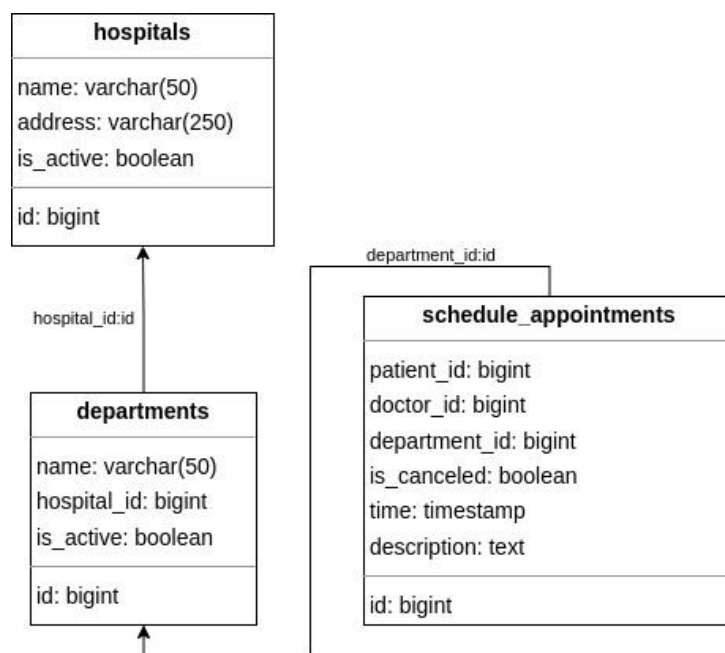


Рисунок 27 – Физическая модель базы данных микросервиса врача

В результате был реализован специальный микросервис, позволяющий работать с информацией об отделениях, поликлиниках и записях к врачу.

3.2.4 Разработка программного обеспечения микросервиса администратора

Разрабатываемый микросервис администратора выполняет задачу управления состоянием таких сущностей, как пользователь, поликлиника и отделение. Выполнение таких операций, как создание, обновление и деактивация некоторых сущностей возможна только через данный сервис.

Так как сами сущности, состояние которых нужно изменять, хранятся в других микросервисах, необходимо реализовать вызов требуемых операций таким образом, чтобы гарантировать доставку запросов, в чем важную роль играет брокер сообщений Apache Kafka. При выполнении запроса на изменение состояния сущности, Kafka отправляет сообщение в нужный микросервис, после чего в коде программы включается режим ожидания ответа от адресата. Если время ожидания истекло или операция завершилась с ошибкой, то происходит выбрасывание исключения. Блок кода, ожидающий получения ответа от микросервиса по брокеру Kafka, показан в листинге 5.

Листинг 5 – Код ожидания ответа от микросервиса по Kafka

```
suspend fun awaitKafkaRequestResult(mills: Long, requestId: Long) {
    var isRequestStatusOld = true
    var responseMessage: String? = null
    var passedMillis = 0
    while (isRequestStatusOld && responseMessage == null && passedMillis
< mills) {
        val requestResult =
kafkaRequestLogRepository.findKafkaRequestLogById(requestId)
        isRequestStatusOld = requestResult?.isRequestSuccessful == false
        responseMessage = requestResult?.message
        passedMillis += 50
        delay(50)
    }
    if (responseMessage != null) {
        throw BadRequestException(responseMessage)
    } else if (passedMillis >= mills) {
        throw TimeoutException("Превышено время ожидания ответа от
сервера")
    }
}
```

Данный подход позволяет как обеспечить надежность запросов в другие микросервисы, так и реализовать механизм по обработке ответов от них.

3.3 Разработка программного обеспечения клиентского приложения

Для разработки клиентского приложения была выбрана технология WPF. Она предоставляет широкий набор инструментов и компонентов для создания функциональных приложений с графическим интерфейсом пользователя, а также позволяет легко настраивать внешний вид и поведение элементов интерфейса [18].

В основе WPF лежит независимый от разрешения векторный модуль визуализации, использующий возможности современного графического оборудования. Возможности этого модуля расширяются с помощью комплексного набора функций разработки приложений, которые включают в себя язык XAML, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции.

Общий принцип клиентского приложения состоит в переключении страниц на главном окне. Это позволяет вести независимую разработку каждой

страницы отдельно, не нагружая при этом пространство логики для основного окна.

Немаловажной особенностью WPF является создание собственных графических элементов для взаимодействия с пользователем. В клиентском приложении используются дополнительные элементы выбора времени врача с возможностью гибкой настройки отображения.

Также в рамках реализации клиентского приложения была создана дополнительная библиотека для отправки запросов на сервер. В ее основе лежит специальный класс, посредством которого осуществляется доступ к методам взаимодействия. Каждый набор методов для взаимодействия с определенным типом информации (сведения о медкартах и сведения о записях) описан отдельным классом, представляющим собой хранилище для реализации методов доступа к API сервера, а также метаданные о конечной точке доступа к определенным методам сервера. Это позволит беспрепятственно вносить незначительные изменения, если те будут происходить на сервере. Каждое из таких хранилищ использует класс `RestClient`, который позволяет в потоковом режиме осуществлять отправки запроса на сервер с определенными заголовками и данными. Это позволяет производить гибкую настройку как конечной точки запроса, так и его содержимого. Пример метода получения записей к определенному врачу представлен в листинге 6.

Листинг 6 – Пример метода получения записей к определенному врачу

```
public IEnumerable<AppointmentDataModel> GetAppointmentsByDoctorId(int
doctorId)
{
    try
    {
        RestClient getAppointmentClient = new
RestClient(EndPoint, HttpVerb.GET);
        string appointmentsData =
getAppointmentClient.MakeRequest($"doctor/{doctorId}");
        var result =
JsonSerializer.Deserialize<List<AppointmentDataModel>>(appointmentsData);
        return result;
    }
    catch
    {
        return null;
    }
}
```

```

    }
}

```

Сама структура клиентского приложения состоит из двух основных частей, а именно части, отвечающей за пользовательский интерфейс приложения, и части бизнес-логики. Организация приложения показана на рисунке 28.

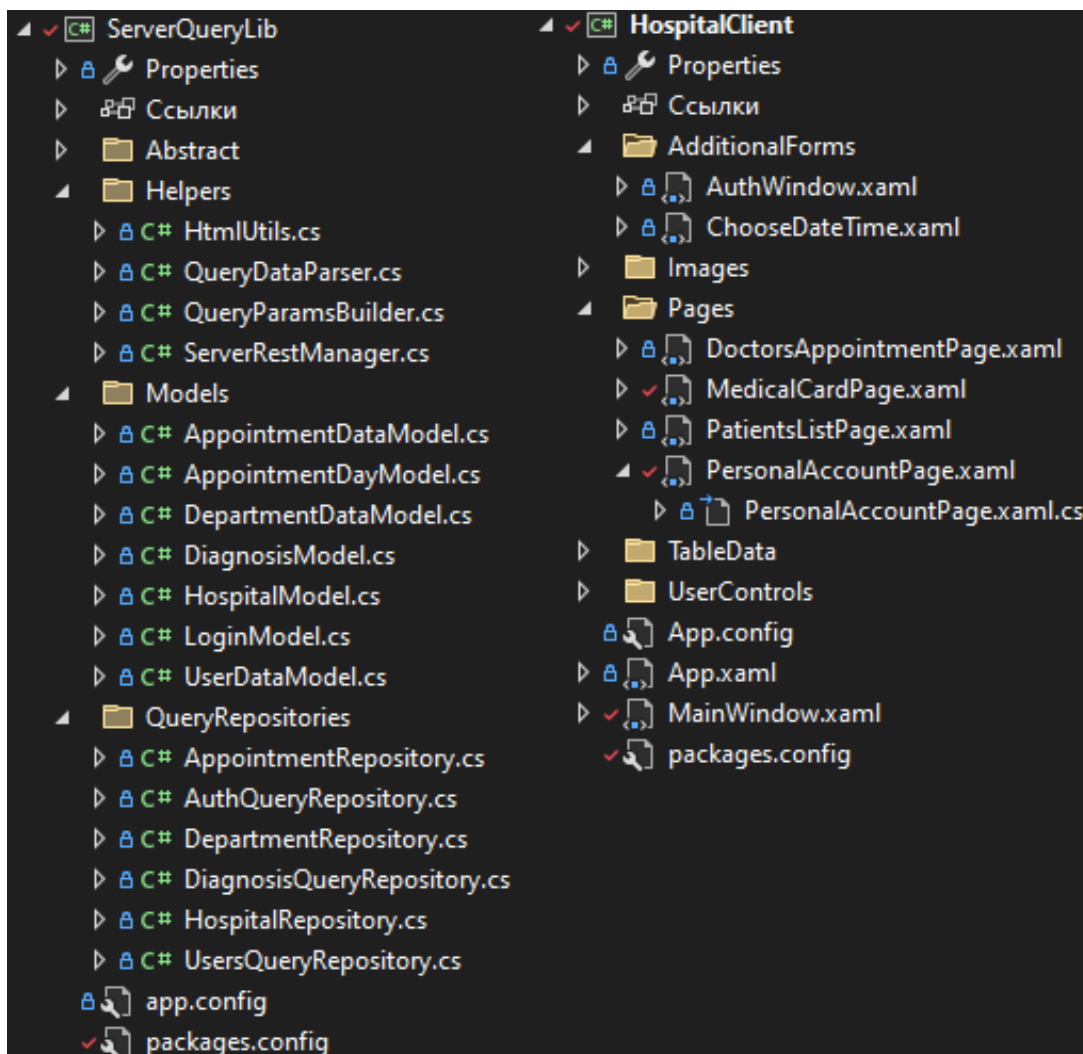


Рисунок 28 – Организация клиентского приложения

Благодаря использованию WPF, можно легко разделить логику взаимодействия с сервером и логику рендеринга пользовательского интерфейса. В общем случае, когда пользователь взаимодействует с пользовательским интерфейсом, приложение с помощью специальных привязок данных Data Bindings производит вызов функций отправки запроса на сервер, и по итогу обработанные данные ответа от сервера отображаются в приложении. Таким образом и осуществляется «общение» от пользователя к серверу.

4 Контроль качества программного обеспечения

4.1 Функциональное тестирование

Проведем функциональное тестирование процесса успешного завершения записи к врачу.

Сначала пациент должен записаться к врачу. Для этого он должен осуществить и подтвердить выбор следующих пунктов при создании записи: подходящая поликлиника, подходящее отделение, подходящий врач и подходящее время. После создания записи к врачу пациент не должен отменить ее. В этом случае после того, как пройдет время записи, она будет считаться успешно завершенной.

Для создания высокоэффективных и результативных тестов можно использовать метод функциональных диаграмм. При построении тестов с помощью этого метода в спецификации определяются причины и следствия. Причина – это входное условие, а следствие – выходные данные системы. Затем спецификация преобразуется в управляющий граф.

Входные данные и выводы системы представлены в таблицах 8 и 9 соответственно.

Таблица 8 – Причины

П1	Выбрана поликлиника
П2	Выбрано отделение
П3	Выбран врач
П4	Выбрано время
П5	Запись не отменена

Таблица 9 – Следствия

С1	Поликлиника не выбрана
С2	Отделение не выбрано
С3	Врач не выбран
С4	Время не выбрано
С5	Запись отменена
С6	Запись успешно завершена

Для процесса была построена функциональная диаграмма, представленная на рисунке 29.

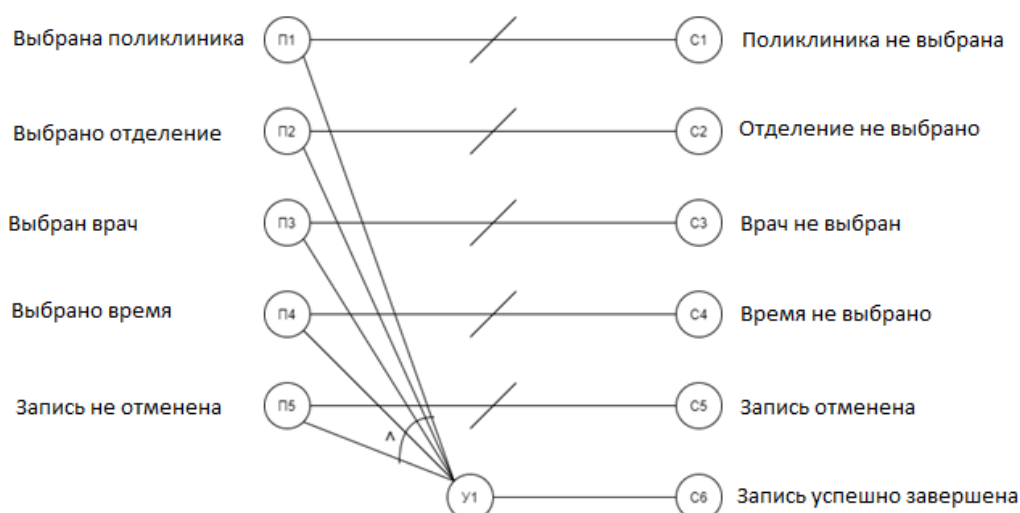


Рисунок 29 – Функциональная диаграмма

На основании функциональной диаграммы была построена таблица решений, представленная в таблице 10.

Таблица 10 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
П5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
С5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

В таблице 11 представлен набор тестов, составленные на основе таблицы решений.

Таблица 11 – Набор функциональных тестов

№	Входные данные	Ожидаемый результат	Результат
1	Не выбрано ничего; Запись не отменена	Запись не создана: поликлиника не выбрана; отделение не выбрано; врач не выбран; время не выбрано;	Тест пройден (рисунок Г.1)

Продолжение таблицы 11

№	Входные данные	Ожидаемый результат	Результат
2	Выбрана поликлиника; Запись не отменена	Запись не создана: отделение не выбрано; врач не выбран; время не выбрано;	Тест пройден
3	Выбрано отделение; Запись не отменена	Запись не создана: поликлиника не выбрана; врач не выбран; время не выбрано;	Тест пройден
4	Выбрана поликлиника и отделение; Запись не отменена	Запись не создана: врач не выбран; время не выбрано;	Тест пройден (рисунок Г.2)
5	Выбран врач; Запись не отменена	Запись не создана: поликлиника не выбрана; отделение не выбрано; время не выбрано;	Тест пройден
6	Выбрана поликлиника и врач; Запись не отменена	Запись не создана: отделение не выбрано; время не выбрано;	Тест пройден
7	Выбрано отделение и врач; Запись не отменена	Запись не создана: поликлиника не выбрана; время не выбрано;	Тест пройден (рисунок Г.3)
8	Выбрана поликлиника, отделение и врач; Запись не отменена	Запись не создана: время не выбрано;	Тест пройден (рисунок Г.4)
9	Выбрано время; Запись не отменена	Запись не создана: поликлиника не выбрана; отделение не выбрано; врач не выбран;	Тест пройден
10	Выбрана поликлиника и время; Запись не отменена	Запись не создана: отделение не выбрано; врач не выбран;	Тест пройден
11	Выбрано отделение и время; Запись не отменена	Запись не создана: поликлиника не выбрана; врач не выбран;	Тест пройден
12	Выбрана поликлиника, отделение и время; Запись не отменена	Запись не создана: врач не выбран;	Тест пройден (рисунок Г.5)
13	Выбран врач и время; Запись не отменена	Запись не создана: поликлиника не выбрана; отделение не выбрано;	Тест пройден
14	Выбрана поликлиника, врач и время; Запись не отменена	Запись не создана: отделение не выбрано;	Тест пройден (рисунок Г.6)
15	Выбрано отделение, врач и время; Запись не отменена	Запись не создана: поликлиника не выбрана;	Тест пройден

Окончание таблицы 11

№	Входные данные	Ожидаемый результат	Результат
16	Выбрана поликлиника, отделение, врач и время; Запись отменена	Запись отменена.	Тест пройден (рисунок Г.7)
17	Выбрана поликлиника, отделение, врач и время; Запись не отменена	Запись успешно завершена.	Тест пройден (рисунок Г.8)

Функциональное тестирование показало, что приложение работает корректно.

4.2 Нефункциональное тестирование интерфейса

В рамках проведения нефункционального тестирования интерфейса программы будет проведена оценка внешнего вида страниц, а именно наличие подсказок, соответствие единому стилю и удобство использования.

Результаты оценки интерфейса представлены в таблице 12.

Таблица 12 – Набор тестов интерфейса

№	Описание теста	Результат и оценка
1	Оценка компонентов страницы на примере страницы личного кабинета.	Страница показана на рисунке 30. Оценка проведена для пунктов 1, 3 – 8.
2	Оценка возможности навигации между страницами интерфейса.	Страница показана на рисунке 30. Оценка проведена для пунктов 2, 9 – 13.
3	Оценка подсказок на странице входа в систему.	Страница показана на рисунках 31–32. Реализованные подсказки отображаются при всех возможных проблемах с входными данными.
4	Оценка страницы медицинской карты.	Страница показана на рисунке 33. Реализованная страница позволяет в форме таблицы наглядно ознакомиться с записями в медицинской карте.
5	Оценка страницы записи к врачу.	Страница показана на рисунках 34–35. Страница позволяет просмотреть как свои записи к врачу для пользователей с ролью пациент, так и записи пациентов для пользователей с ролью врач. Ниже показан функционал для создания новой записи к врачу.
6	Оценка страницы со списком пользователей.	Страница показана на рисунке 36. Данная страница позволяет как найти нужного пользователя, так и провести его редактирование или изучить личные данные

Продолжение таблицы 12

№	Описание теста	Результат и оценка
7	Оценка страницы с расписанием работы врача.	Страница показана на рисунке 37. Реализованная страница реализует просмотра и изменения расписания врача в удобной форме.
8	Оценка единого стиля интерфейса.	Все показанные рисунки и компоненты интерфейса соответствуют единому стилю.
9	Проверка контрастности текста и фона.	Текст на страницах интерфейса читается без затруднений при разном уровне освещения.
10	Общая оценка визуального восприятия.	Реализованный интерфейс не перегружен излишними компонентами, а также состоит из хорошо сочетающихся цветов, благодаря чему приятен для восприятия.

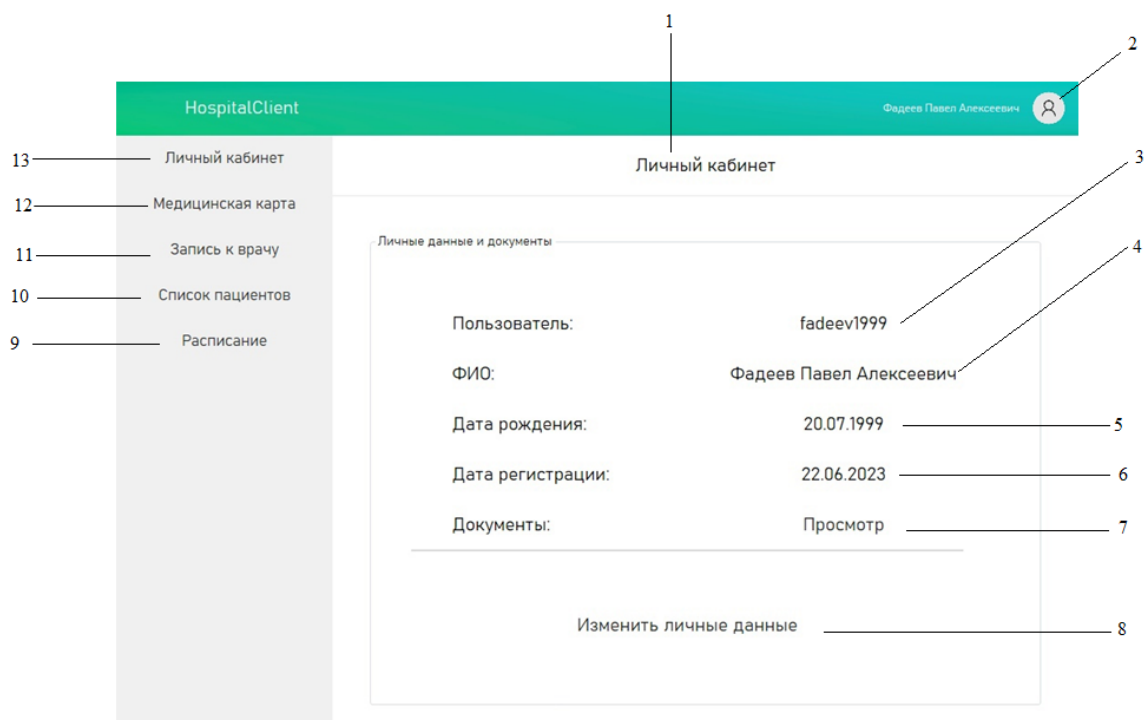


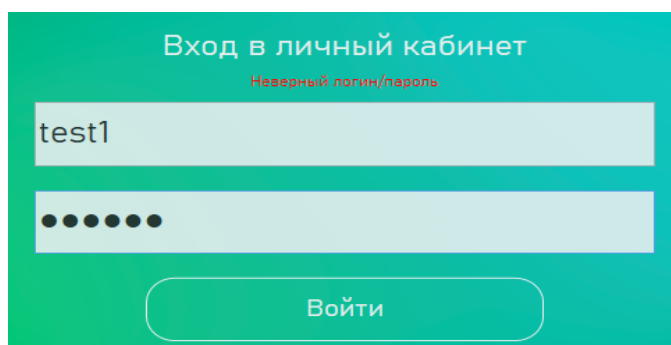
Рисунок 30 – Результат теста 1, 2

Рассмотрим элементы интерфейса, представленные на рисунке 26:

- 1 – заголовок выбранной страницы приложения;
- 2 – кнопка быстрого перехода в личный кабинет с подписью авторизованного пользователя;
- 3 – элемент отображения логина авторизованного пользователя;
- 4 – элемент отображения ФИО авторизованного пользователя;
- 5 – элемент отображения даты рождения авторизованного лица;
- 6 – элемент отображения даты регистрации авторизованного лица;

- 7 – кнопка перехода в раздел документов авторизованного лица;
- 8 – кнопка перехода в раздел изменения личных данных авторизованного лица;
- 9 – кнопка перехода на страницу настраиваемого расписания для врачей;
- 10 – кнопка перехода на страницу списка пользователей информационной системы;
- 11 – кнопка перехода на страницу записи к врачу пользователем;
- 12 – кнопка перехода на страницу просмотра медицинской карты пользователя;
- 13 – кнопка перехода на страницу личного кабинета авторизованного лица.

На рисунках 31 и 32 показана страница входа в систему. Здесь предусмотрено два вида сообщений на тот случай, если пользователь неверно ввел свои данные или не заполнил требуемые поля.



Вход в личный кабинет

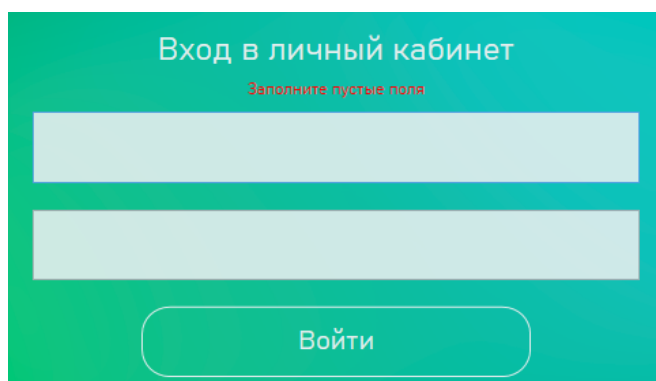
Неверный логин/пароль

test1

•••••

Войти

Рисунок 31 – Результат теста 3



Вход в личный кабинет

Заполните пустые поля

Войти

Рисунок 32 – Результат теста 3

Страница медицинской карты, показанная на рисунке 33, позволяет ознакомиться с записями о состоянии здоровья пациента.

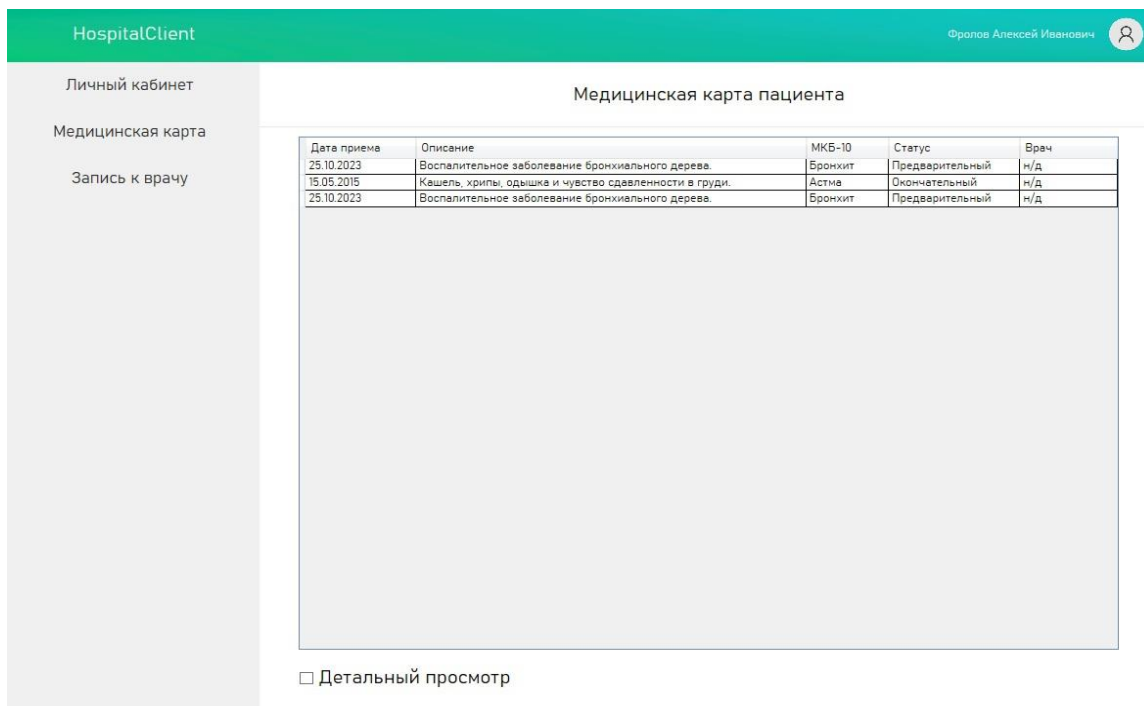


Рисунок 33 – Результат теста 4

Показанная на рисунках 34 и 35 страница записи к врачу предоставляет весь необходимый функционал для данной операции.

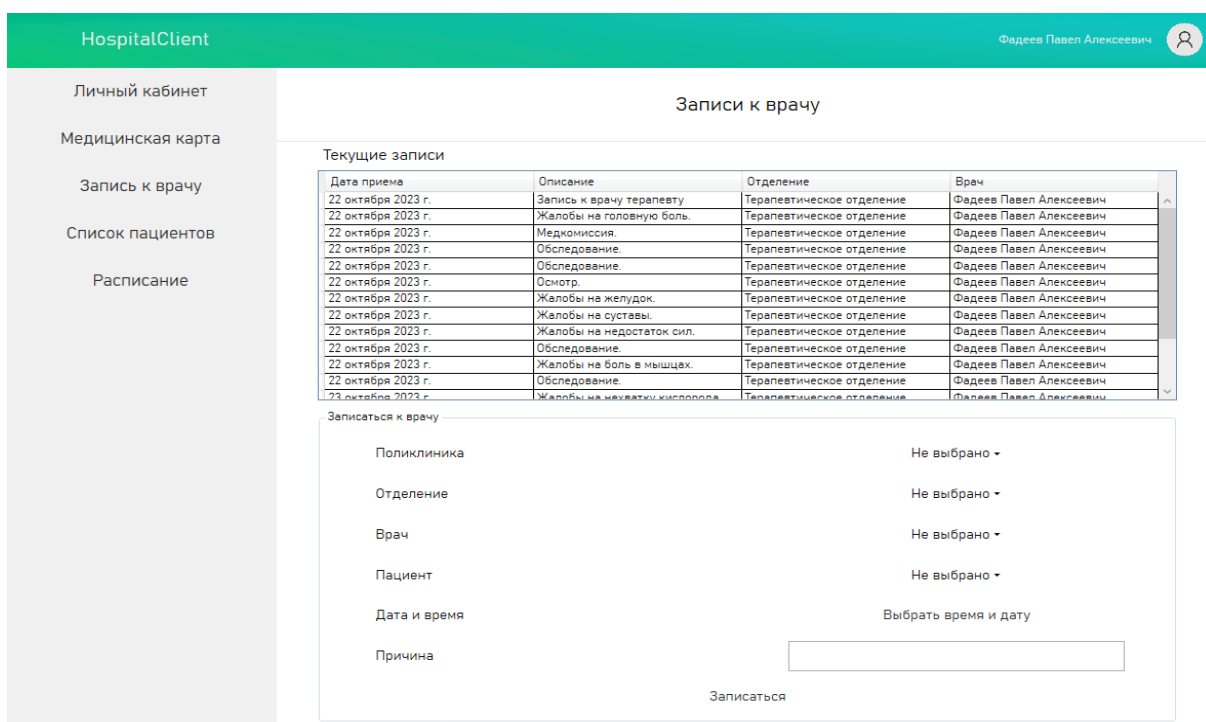


Рисунок 34 – Результат теста 5

Записаться к врачу

Поликлиника Не выбрано ▾

Отделение Не выбрано

Врач Городская Поликлиника № 11

Пациент Поликлиника № 5

Дата и время Городская Поликлиника № 10

Причина Не выбрано ▾

Выбрать время и дату

Записаться

Рисунок 35 – Результат теста 5

На рисунке 36 показана страница списка пациентов, позволяющая найти и ознакомиться с информацией о требуемом человеке.

HospitalClient Фадеев Павел Алексеевич

Личный кабинет

Медицинская карта

Запись к врачу

Список пациентов

Расписание

Список пациентов

Поиск... Фильтр

Дата рождения	ФИО	Пол
27.10.1994	Фролов Алексей Иванович	мужской
14.11.1998	Семенов Игорь Николаевич	мужской
13.05.1991	Алексеев Павел Юрьевич	мужской
21.07.1982	Медведева Анна Антоновна	женский
12.12.2000	Фролов Николай Сергеевич	мужской
22.01.1995	Гаспарян Инна Викторовна	женский
10.03.2001	Макарова Ольга Федоровна	женский
04.12.1965	Коняхин Алексей Максимович	мужской

Редактировать Посмотреть медкарту

Рисунок 36 – Результат теста 6

На рисунке 37 показана страница с расписанием работы врача.

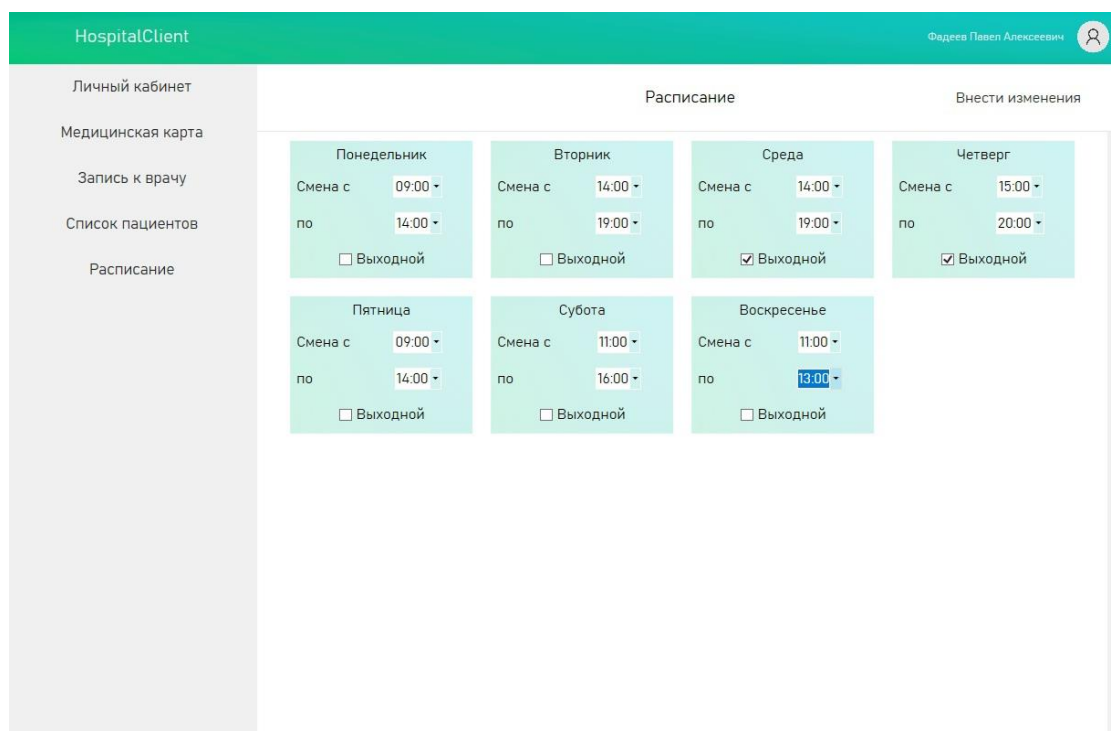


Рисунок 37 – Результат теста 7

Проведенные тесты показали, что интерфейс приложения реализован корректно.

4.3 Метрики кода

4.3.1 Метрика SLOC

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объема с помощью подсчета количества строк в тексте исходного кода [19].

Был проведен подсчет данной метрики в классе AppointmentServiceImpl. Данный класс предназначен для управления записями к врачу. Код приведен в листинге В.5.

Физическими строками считаются все непустые строки текстового файла.

Измеряя логические строки кода, предпринимается попытка посчитать количество представленных операторов в программе, хоть их определение зависит от конкретного языка программирования.

В данной функции имеем:

- 83 физических строк кода;
- 55 логических строк кода;
- 8 строк комментариев.

Значение метрики SLOC равняется 83 физическим строкам кода и 55 логическим.

4.3.2 Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа общего числа строк и синтаксических элементов исходного кода программы [20].

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- число уникальных операндов программы (словарь операндов);
- общее число операторов в программе;
- общее число операндов в программе.

Операторы класса AppointmentServiceImpl приведены в таблице 13.

Таблица 13 – Операторы

Операторы	Число вхождений
appointmentRepository.findAll	1
AppointmentRsDto.fromEntity	5
List.map	3
appointmentRepository.findById	3
?:	9
appointmentRepository.findById	1
appointmentRepository.findById	1
?	1
Standart.let	1
appointmentRepository.save	3
->	1

Продолжение таблицы 13

Операторы	Число вхождений
appointment.copy	2
checkAppointmentTime	2
Standart.takeIf	4
!=	6
if	3
departmentRepository.findDepartment ById	2
for	1
in	1
==	1
appointmentTime.toLocalTime	4
LocalTime.isAfter	2
	2
!	2
&&	1
equals	2
break	1
{}	20
()	52
n1 = 29	N1 = 135

Операнды класса AppointmentServiceImpl приведены в таблице 14.

Таблица 14 – Операнды

Операнды	Число вхождений
it	7
id	3
patientId	3
doctorShiftService	3
departmentRepository	3
appointmentRepository	9
appointment	7
isCanceled	2
saveDto	6
doctorId	9
time	8
departmentId	5
null	7
false	1

Продолжение таблицы 14

Операнды	Число вхождений
true	1
updateDto	9
department	3
description	4
doctorShifts	1
shift	5
appointmentTime	5
startTime	4
«Запись к врачу с id=\$id не найдена»	3
«Отделение с id=\${saveDto.departmentId} не найдено»	1
«Отделение с id=\${updateDto.departmentId} не найдено»	1
«Недопустимое время записи»	1
n2 = 26	N2 = 111

Длина программы рассчитывается по формуле:

$$N = N1 + N2,$$

где N1 – общее количество операторов в коде;

N2 – общее количество операндов в коде.

По результатам расчетов длина программы N равняется 266.

Словарь программы рассчитывается по формуле:

$$n = n1 + n2,$$

где n1 – количество различных операторов;

n2 – количество различных операндов.

По результатам расчетов словарь программы n равняется 65.

Объем программы рассчитывается по формуле:

$$HV = N \cdot \log_2 n.$$

По результатам расчетов объем программы HV равняется 1601.32.

Сложность программы рассчитывается по формуле:

$$HD = (n1/2) \cdot (N2/n2).$$

По результатам расчетов сложность программы HD равняется 61.9.

В результате была проведена оценка программного кода метрикой Холстеда.

4.3.3 Метрика Чепина

Задача метода заключается в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода [21].

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

- вводимые переменные для расчетов и для обеспечения вывода,
- модифицируемые, или создаваемые внутри программы переменные,
- переменные, участвующие в управлении работой программного модуля,
- не используемые в программе («паразитные») переменные.

Каждая переменная может выполнять несколько функций одновременно, поэтому необходимо учитывать ее в каждой соответствующей функциональной группе. Метрика Чепина рассчитывается следующим образом:

$$Q = a_1 \cdot P + a_2 \cdot M + a_3 \cdot C + a_4 \cdot T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты;

P – вводимые переменные для расчетов и для обеспечения вывода;

M – модифицируемые, или создаваемые внутри программы, переменные;

C – переменные, участвующие в управлении работой программного модуля;

T – неиспользуемые в программе.

Для отражения различного влияния на сложность программы каждой группы использованы весовые коэффициенты. Весовые коэффициенты групп распределяются следующим образом: $a_1=1$, $a_2=2$, $a_3=3$, $a_4=0.5$. Неиспользуемые переменные имеют ненулевой коэффициент. В представленной в листинге В.5 функции $P = 6$, $M = 2$, $C = 2$, $T = 0$. Учитывая весовые коэффициенты:

$$Q = 6 + 2 \cdot 2 + 3 \cdot 2 + 0.5 \cdot 0 = 16.$$

Значение метрики Чепина равняется 16.

Заключение

В процессе выполнения данной работы была разработана информационная система медицинских организаций города, позволяющая централизовать их работу.

Первоначально был проведен анализ рассматриваемой предметной области, структурной организации и ее основных объектов. Вместе с рассмотрением ключевых бизнес-процессов это позволило выделить основной функционал и составляющие разрабатываемой программы. Также был проведен сравнительный анализ приложений аналогов, позволивший выделить их основные преимущества и недостатки, которые были учтены при разработке информационной системы. Была проведена и оценка бюджета.

В процессе проектирования системы были смоделированы функциональные возможности информационной системы и спроектирована концептуальная модель базы данных. Также была создана архитектурная схема программы, ставшая основой для создания основных компонентов системы и их взаимодействия.

Реализация информационной системы позволила создать клиентское и серверное приложения, взаимодействующие друг с другом по архитектуре REST. В рамках серверного приложения были созданы три микросервиса, взаимодействующие друг с другом по брокеру Apache Kafka и HTTP протоколу.

Получившаяся в результате система была протестирована методом функционального и нефункционального тестирования, а также проведена оценка метрик кода разными способами.

Программное обеспечение было разработано на языках Kotlin и C# с применением фреймворков Spring и .NET.

Список использованных источников

1. Номенклатура медицинских организаций [Электронный ресурс]. URL: <https://base.garant.ru/70453400/53f89421bbdaf741eb2d1ecc4ddb4c33/?ysclid=lx28cxz3xy777217207> (дата обращения: 19.05.2024).
2. О повышении доступности медицинской помощи и расширении возможностей ее оказания [Электронный ресурс]. URL: <http://government.ru/orders/selection/405/14957/> (дата обращения: 19.05.2024).
3. Клиника «Медгард» [Электронный ресурс]. URL: <https://prodoctorov.ru/penza/lpu/85119-medgard/#information> (дата обращения: 19.05.2024).
4. Многопрофильные клиники «Медиклиник» [Электронный ресурс]. URL: <https://www.mediclinic.ru/> (дата обращения: 19.05.2024).
5. Медицинский центр «Персональное здоровье» [Электронный ресурс]. URL: <https://prodoctorov.ru/penza/lpu/36246-personalnoe-zdorove/#information> (дата обращения: 19.05.2024).
6. Сервис «СберЗдоровье» [Электронный ресурс]. URL: <https://sberhealth.ru/> (дата обращения: 19.05.2024).
7. Сервис «Здоровье.ру» [Электронный ресурс]. URL: <https://zdorovie.ru/> (дата обращения: 19.05.2024).
8. Что такое диаграмма Ганта и как ее построить [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/diagramma-ganta/> (дата обращения: 21.05.2024).
9. Официальный сайт «ProjectLibre» [Электронный ресурс]. URL: <https://www.projectlibre.com/> (дата обращения: 21.05.2024).
10. Моделирование данных в СУБД: от концептуального проектирования к реализации [Электронный ресурс]. URL: <https://appmaster.io/ru/blog/modelirovanie-dannykh-v-rdbms> (дата обращения: 23.05.2024).

11. Официальный сайт программного обеспечения «Postman» [Электронный ресурс]. URL: <https://www.postman.com/> (дата обращения: 24.05.2024).
12. Официальный сайт инструмента «Visual Paradigm» [Электронный ресурс]. URL: <https://www.visual-paradigm.com/> (дата обращения: 24.05.2024).
13. Официальная страница среды разработки «IntelliJ IDEA» [Электронный ресурс]. URL: <https://www.jetbrains.com/ru-ru/idea/> (дата обращения: 25.05.2024).
14. Официальная страница среды разработки «Visual Studio» [Электронный ресурс]. URL: <https://visualstudio.microsoft.com/ru/> (дата обращения: 25.05.2024).
15. Документация фреймворка «Spring» [Электронный ресурс]. URL: <https://spring.io/projects/spring-framework> (дата обращения: 26.05.2024).
16. Официальная страница фреймворка «.NET» [Электронный ресурс]. URL: <https://spring.io/projects/spring-framework> (дата обращения: 26.05.2024).
17. Официальная страница «PostgreSQL» [Электронный ресурс]. URL: <https://www.postgresql.org/> (дата обращения: 27.05.2024).
18. Документация «WPF» [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0> (дата обращения: 28.05.2024).
19. Source lines of code [Электронный ресурс]. URL: https://wiki2.org/en/Source_lines_of_code?ysclid=lwxlbpxya1647836544 (дата обращения: 29.05.2024).
20. Halsted's Software Metrics – Software Engineering [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/> (дата обращения: 30.05.2024).
21. Метрика Чепина Engineering [Электронный ресурс]. URL: https://studopedia.ru/12_94184_metrika-chEpina.html?ysclid=lwxlgnick0268658738 (дата обращения: 30.05.2024).

Приложение А
(справочное)
Глоссарий

База данных (БД) – это организованная коллекция данных, обеспечивающая удобный доступ, обновление и управление данными.

Микросервис – архитектурный стиль разработки программного обеспечения, подразумевающий разделение приложения на автономные сервисы, выполняющие определенную задачу.

Система управления базами данных (СУБД) – программное обеспечение, используемое для взаимодействия, создания и управления базами данных.

Unified Modeling Language (UML) – графический унифицированный язык моделирования, использующийся для визуализации, конструирования и документирования программных систем.

Информационная система (ИС) – программное или аппаратное обеспечение, предназначенное для обработки, хранения и передачи информации.

НИИ (Научно-исследовательский институт) – учреждение, специализирующееся на проведении научных исследований в разных областях знаний.

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня, предназначенный для передачи гипертекстовых документов в сети.

Spring (Spring Framework) – фреймворк, написанный на языке Java, предоставляющий набор инструментов для разработки корпоративных приложений.

.NET – платформа, включающая в себя набор библиотек, используемых при разработке на языках C# и F#.

Kafka (Apache Kafka) – распределенный брокер сообщений, используемый для обработки и передачи данных, обеспечивающий высокую надежность и производительность в работе.

Приложение Б
(рекомендуемое)
Спецификации вариантов использования

Таблица Б.1 – Описание основных спецификаций вариантов использования администратора

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
Просмотреть список отделений	1	Система возвращает список отделений	Администратор	Существуют исходные записи	<p>1. Администратор с помощью переключаемых кнопок на интерфейсе выбирает действие над отделением (создать новое отделение, редактировать существующее или просмотреть информацию) и выполняет операцию</p> <p>2. После завершения выбранной операции администратору возвращается список отделений</p>	Данные возвращены
Просмотреть список пользователей	2	Система возвращает список пользователей	Администратор	Существуют исходные записи	<p>1. Администратор с помощью переключаемых кнопок на интерфейсе выбирает действие над пользователем (создать нового пользователя, редактировать существующего или просмотреть информацию) и выполняет операцию</p>	Данные возвращены

Продолжение таблицы Б.1

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
					2. После завершения выбранной операции администратору возвращается список пользователей	

Таблица Б.2 – Описание основных спецификаций вариантов использования врача

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
Просмотреть список врачей	1	Система возвращает список врачей	Врач	Существуют исходные записи	<p>1. Врач с помощью переключаемых кнопок на интерфейсе выбирает действие над пациентом в разделе списка врачей (записать пациента к врачу или редактировать запись пациента к врачу) и выполняет операцию</p> <p>2. После завершения выбранной операции врачу возвращается список врачей</p>	Данные возвращены

Продолжение таблицы Б.2

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
Просмотреть список пациентов	2	Система возвращает список пациентов	Врач	Существуют исходные записи	<p>1. Врач переходит в профиль конкретного пациента и выбирает действие (редактировать медицинскую карту пациента или добавить в нее новую запись) и выполняет операцию</p> <p>2. После завершения выбранной операции врачу возвращается список пациентов</p>	Данные возвращены

Таблица Б.3 – Описание основных спецификаций вариантов использования пациента

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
Просмотреть медицинскую карту	1	Система возвращает данные медицинской карты пациента	Пациент	Существуют исходные записи	1. Пациент открывает раздел своей медицинской карты и выбирает действие (просмотреть конкретное заболевание или список заболеваний)	Данные возвращены

Продолжение таблицы Б.3

Название	ID	Краткое описание	Действующие лица	Предусловие	Основной поток	Постусловие
					2. После завершения выбранной операции пользователю возвращаются данные медицинской карты	
Управлять записями к врачам	2	Система возвращает список записей к врачам	Пациент	Существуют исходные записи	1. Пациент с помощью переключаемых кнопок на интерфейсе выбирает действие над своими записями к врачам (просмотреть все записи, создать новую запись или редактировать существующую запись) 2. После завершения выбранной операции пациенту возвращается список записей к врачам	Данные возвращены

Приложение В
(рекомендуемое)
Листинг приложения

Листинг В.1 – Файл DiagnoseApi.kt

```

@Tag(name = "Записи о диагнозах")
@RequestMapping("/api/diagnoses")
interface DiagnoseApi {
    @Operation(description = "Получение списка диагнозов")
    @GetMapping
    fun getDiagnoses(): List<DiagnoseRsDto>

    @Operation(description = "Получение списка диагнозов для пациента")
    @GetMapping("/patient/{userId}")
    fun getDiagnosesForUser(@PathVariable userId: Long):
List<DiagnoseRsDto>

    @Operation(description = "Получение диагноза по id")
    @GetMapping("/{id}")
    fun getDiagnoseById(@PathVariable id: Long): DiagnoseRsDto

    @Operation(description = "Добавление диагноза")
    @PostMapping
    fun saveDiagnose(@RequestBody saveDto: DiagnoseSaveDto):
DiagnoseRsDto

    @Operation(description = "Обновление диагноза по id")
    @PutMapping("/{id}")
    fun updateDiagnose(@PathVariable id: Long, @RequestBody updateDto:
DiagnoseUpdateDto): DiagnoseRsDto
}

@RestController
class DiagnoseController(
    private val diagnoseService: DiagnoseService
) : DiagnoseApi {
    override fun getDiagnoses(): List<DiagnoseRsDto> =
diagnoseService.getDiagnoses()

    override fun getDiagnosesForUser(userId: Long): List<DiagnoseRsDto> =
diagnoseService.getDiagnosesForUser(userId)

    override fun getDiagnoseById(id: Long): DiagnoseRsDto =
diagnoseService.getDiagnoseById(id)

    override fun saveDiagnose(saveDto: DiagnoseSaveDto): DiagnoseRsDto =
diagnoseService.saveDiagnose(saveDto)

    override fun updateDiagnose(id: Long, updateDto: DiagnoseUpdateDto):
DiagnoseRsDto = diagnoseService.updateDiagnose(id, updateDto)
}

```

Листинг В.2 – Файл HospitalApi.kt

```

@Tag(name = "Записи о поликлиниках")
@RequestMapping("/api/hospitals")
interface HospitalApi {
    @Operation(description = "Получение списка поликлиник")

```

```

    @GetMapping
    fun getHospitals(): List<HospitalRsDto>

    @Operation(description = "Получение поликлиники по id")
    @GetMapping("/{id}")
    fun getHospitalById(@PathVariable id: Long): HospitalRsDto
}

@RestController
class HospitalController(
    private val restTemplate: RestTemplate
) : HospitalApi {
    val hospitalsApi = "http://localhost:8081/api/hospitals"

    override fun getHospitals(): List<HospitalRsDto> {
        val request = runCatching {
            restTemplate.getForEntity(hospitalsApi,
                emptyArray<HospitalRsDto>()::class.java)
        }
        val result = request.getOrElse { throw
            MicroserviceRequestException("Произошла ошибка при запросе в микросервис
            врача") }
        return result.body!!.asList()
    }

    override fun getHospitalById(id: Long): HospitalRsDto {
        val request = runCatching {
            restTemplate.getForEntity("$hospitalsApi/$id", HospitalRsDto::class.java)
        }
        val result = request.getOrElse {
            if (it.message!!.contains("NOT_FOUND")) throw
                EntityNotFoundException("Поликлиника с id=$id не найдена")
            throw MicroserviceRequestException("Произошла ошибка при
            запросе в микросервис врача")
        }
        return result.body!!
    }
}

```

Листинг В.3 – Файл AppointmentApi.kt

```

@Tag(name = "Записи о записях к врачу")
@RequestMapping("/api/appointments")
interface AppointmentApi {
    @Operation(description = "Получение списка записей к врачу")
    @GetMapping
    fun getAppointments(): List<AppointmentRsDto>

    @Operation(description = "Получение записи к врачу по id")
    @GetMapping("/{id}")
    fun getAppointmentById(@PathVariable id: Long): AppointmentRsDto

    @Operation(description = "Получение записей пациента по id")
    @GetMapping("/patient/{patientId}")
    fun getPatientAppointments(@PathVariable patientId: Long):
List<AppointmentRsDto>

    @Operation(description = "Получение записей врача по id")
    @GetMapping("/doctor/{doctorId}")
    fun getDoctorAppointments(@PathVariable doctorId: Long):
List<AppointmentRsDto>

    @Operation(description = "Отмена записи к врачу по id")
    @PatchMapping("/cancel/{id}")
    fun cancelAppointmentById(@PathVariable id: Long)

    @Operation(description = "Добавление записи к врачу")
    @PostMapping
    fun saveAppointment(@RequestBody saveDto: AppointmentSaveDto)

    @Operation(description = "Обновление записи к врачу по id")
    @PutMapping("/{id}")
    fun updateAppointment(@PathVariable id: Long, @RequestBody updateDto:
AppointmentUpdateDto): AppointmentRsDto
}

@RestController
class AppointmentController(
    private val appointmentService: AppointmentService
) : AppointmentApi {
    override fun getAppointments(): List<AppointmentRsDto> =
appointmentService.getAppointments()

    override fun getAppointmentById(id: Long): AppointmentRsDto =
appointmentService.getAppointmentById(id)

    override fun getPatientAppointments(patientId: Long):
List<AppointmentRsDto> =
appointmentService.getPatientAppointments(patientId)

    override fun getDoctorAppointments(doctorId: Long):
List<AppointmentRsDto> =
appointmentService.getDoctorAppointments(doctorId)

```

```

        override fun cancelAppointmentById(id: Long) =
            appointmentService.cancelAppointmentById(id)

        override fun saveAppointment(saveDto: AppointmentSaveDto) =
            appointmentService.saveAppointment(saveDto)

        override fun updateAppointment(id: Long, updateDto:
            AppointmentUpdateDto): AppointmentRsDto =
            appointmentService.updateAppointment(id, updateDto)
    }

```

Листинг В.4 – Файл AdminDepartmentApi.kt

```

@Tag(name = "Управлять записями об отделениях поликлиники")
@RequestMapping("/api/departments")
interface AdminDepartmentApi {

    @Operation(description = "Добавление отделения")
    @PostMapping
    suspend fun saveDepartment(@RequestBody saveDto:
        AdminDepartmentSaveDto)

    @Operation(description = "Обновление отделения по id")
    @PutMapping("/{id}")
    suspend fun updateDepartment(@PathVariable id: Long, @RequestBody
        updateDto: AdminDepartmentUpdateDto)

    @Operation(description = "Деактивировать отделения по id")
    @PatchMapping("/deactivation/{id}")
    suspend fun deactivateDepartment(@PathVariable id: Long)

    @Operation(description = "Активировать отделения по id")
    @PatchMapping("/activation/{id}")
    suspend fun activateDepartment(@PathVariable id: Long)
}

@RestController
class AdminDepartmentController(
    private val restTemplate: RestTemplate
) : AdminDepartmentApi {
    val adminDepartmentsApi = "http://localhost:8082/api/departments"
    val departmentsApi = "http://localhost:8081/api/departments"
    override suspend fun saveDepartment(saveDto: AdminDepartmentSaveDto)
    {
        val request = runCatching {
            restTemplate.postForEntity(adminDepartmentsApi,
                HttpEntity<AdminDepartmentSaveDto>(saveDto), String::class.java)
        }
        request.getOrElse {
            if (it.message!!.contains("NOT_FOUND")) throw
                EntityNotFoundException("Поликлиника с id=${saveDto.hospitalId} не
                    найдена")
            throw MicroserviceRequestException("Произошла ошибка при
                запросе в микросервис администратора")
        }
    }
}

```

```

        override suspend fun updateDepartment(id: Long, updateDto:
AdminDepartmentUpdateDto) {
            val existRequest = runCatching {
restTemplate.getForEntity("$departmentsApi/$id",
DepartmentRsDto::class.java) }
            existRequest.getOrElse {
                if (it.message!!.contains("NOT_FOUND")) {
                    if (it.message!!.contains("Поликлиника с id=")) throw
EntityNotFoundException("Поликлиника с id=$id не найдена")
                    if (it.message!!.contains("Отделение с id=")) throw
EntityNotFoundException("Отделение с id=$id не найдено")
                }
                throw MicroserviceRequestException("Произошла ошибка при
запросе в микросервис администратора")
            }
            restTemplate.exchange("$adminDepartmentsApi/$id", HttpMethod.PUT,
HttpEntity<AdminDepartmentUpdateDto>(updateDto), String::class.java)
        }

        override suspend fun deactivateDepartment(id: Long) {
            val request = runCatching {
restTemplate.getForEntity("$departmentsApi/$id",
DepartmentRsDto::class.java) }
            request.getOrElse {
                if (it.message!!.contains("NOT_FOUND")) throw
EntityNotFoundException("Отделение с id=$id не найдено")
                throw MicroserviceRequestException("Произошла ошибка при
запросе в микросервис администратора")
            }

restTemplate.patchForObject("$adminDepartmentsApi/deactivation/$id",
null, String::class.java)
        }

        override suspend fun activateDepartment(id: Long) {
            val request = runCatching {
restTemplate.getForEntity("$departmentsApi/$id",
DepartmentRsDto::class.java) }
            request.getOrElse {
                if (it.message!!.contains("NOT_FOUND")) throw
EntityNotFoundException("Отделение с id=$id не найдено")
                throw MicroserviceRequestException("Произошла ошибка при
запросе в микросервис администратора")
            }

restTemplate.patchForObject("$adminDepartmentsApi/activation/$id", null,
String::class.java)
        }
    }
}

```

Листинг В.5 – Файл AppointmentService.kt

```

@Service
class AppointmentServiceImpl(
    private val doctorShiftService: DoctorShiftService,
    private val departmentRepository: DepartmentRepository,
    private val appointmentRepository: AppointmentRepository

```

```

) : AppointmentService {

    //получить список всех записей к врачу
    override fun getAppointments() =
        appointmentRepository.findAll().map {
            AppointmentRsDto.fromEntity(it)
        }

    //получить запись к врачу по id
    override fun getAppointmentById(id: Long) =
        AppointmentRsDto.fromEntity(
            appointmentRepository.findAppointmentById(id)
                ?: throw EntityNotFoundException("Запись к врачу с id=$id
не найдена")
        )

    //получить записи пациента к врачу по его id
    override fun getPatientAppointments(patientId: Long) =
        appointmentRepository.findByPatientId(patientId).map {
            AppointmentRsDto.fromEntity(it)
        }

    //получить записи к врачу по id врача
    override fun getDoctorAppointments(doctorId: Long) =
        appointmentRepository.findByDoctorId(doctorId).map {
            AppointmentRsDto.fromEntity(it)
        }

    //отменить запись к врачу по id записи
    override fun cancelAppointmentById(id: Long) {
        appointmentRepository.findAppointmentById(id)?.let { appointment
->
            appointmentRepository.save(appointment.copy(isCanceled =
true))
        } ?: throw EntityNotFoundException("Запись к врачу с id=$id не
найдена")
    }

    //добавить запись к врачу
    override fun saveAppointment(saveDto: AppointmentSaveDto) {
        checkAppointmentTime(saveDto.doctorId, saveDto.time) todo
uncomment
        appointmentRepository.save(
            ScheduleAppointment(
                patientId = saveDto.patientId,
                doctorId = saveDto.doctorId,
                department =
departmentRepository.findDepartmentById(saveDto.departmentId)
                    ?: throw EntityNotFoundException(
                        "Отделение с id=${saveDto.departmentId} не
найдено"
                    ),
                time = saveDto.time,
                description = null,
                isCanceled = false
            )
        )
    }
}

```

```

    }

    //обновить данные о записи к врачу
    override fun updateAppointment(id: Long, updateDto:
AppointmentUpdatedDto): AppointmentRsDto {
        val appointment = appointmentRepository.findAppointmentById(id)
        ?: throw EntityNotFoundException("Запись к врачу с id=$id не найдена")
        if(updateDto.time != null) {
            checkAppointmentTime(appointment.doctorId, updateDto.time) }
        return AppointmentRsDto.fromEntity(
            appointmentRepository.save(
                appointment.copy(
                    patientId = updateDto.patientId.takeIf { it != null }
                    ?: appointment.patientId,
                    doctorId = updateDto.doctorId.takeIf { it != null }
                    ?: appointment.doctorId,
                    department = if (updateDto.departmentId != null) {
                        departmentRepository.findDepartmentById(updateDto.departmentId)
                            ?: throw EntityNotFoundException("Отделение с
id=${updateDto.departmentId} не найдено")
                    } else appointment.department,
                    description = updateDto.description.takeIf { it !=
null } ?: appointment.description,
                    time = updateDto.time.takeIf { it != null } ?:
appointment.time
                )
            )
        )
    }

    //проверить доступности времени для записи к врачу
    private fun checkAppointmentTime(doctorId: Long, appointmentTime:
LocalDateTime) {
        val doctorShifts = doctorShiftService.getDoctorShifts(doctorId)
        for (shift in doctorShifts) {
            if(shift.dayOfTheWeek == appointmentTime.dayOfTheWeek.name) {

                if((appointmentTime.toLocalTime().isAfter(shift.startTime) ||
!appointmentTime.toLocalTime().equals(shift.startTime))
                    &&
                (appointmentTime.toLocalTime().isAfter(shift.startTime) ||
!appointmentTime.toLocalTime().equals(shift.startTime))) {
                    throw InappropriateTimeException("Недопустимое время
записи")
                }
                break
            }
        }
    }
}

```

Листинг В.6 – Файл DiagnoseService.kt

```

interface DiagnoseService {
    fun getDiagnoses(): List<DiagnoseRsDto>
}

```

```

    fun getDiagnosesForUser(userId: Long): List<DiagnoseRsDto>

    fun getDiagnoseById(id: Long): DiagnoseRsDto

    fun saveDiagnose(saveDto: DiagnoseSaveDto): DiagnoseRsDto

    fun updateDiagnose(id: Long, updateDto: DiagnoseUpdateDto):
DiagnoseRsDto
}

@Service
class DiagnoseServiceImpl(
    private val userRepository: UserRepository,
    private val diagnoseRepository: DiagnoseRepository
) : DiagnoseService {
    override fun getDiagnoses() =
        diagnoseRepository.findAll().map {
            DiagnoseRsDto(
                id = it.id,
                patientName = it.patient.name,
                name = it.name,
                dateOfDiagnosis = it.dateOfDiagnosis,
                description = it.description,
                isTreated = it.isTreated
            )
        }

    override fun getDiagnosesForUser(userId: Long): List<DiagnoseRsDto> =
        diagnoseRepository.findAllDiagnosesForPatient(userId).map {
            DiagnoseRsDto(
                id = it.id,
                patientName = it.patient.name,
                name = it.name,
                dateOfDiagnosis = it.dateOfDiagnosis,
                description = it.description,
                isTreated = it.isTreated
            )
        }

    override fun getDiagnoseById(id: Long) =
        DiagnoseRsDto.fromEntity(
            diagnoseRepository.findDiagnoseById(id) ?: throw
EntityNotFoundException("Диагноз с id=$id не найден")
        )

    override fun saveDiagnose(saveDto: DiagnoseSaveDto) =
        DiagnoseRsDto.fromEntity(
            diagnoseRepository.save(
                Diagnose(
                    patient =
userRepository.findUserById(saveDto.patientId)
                    ?: throw EntityNotFoundException("Диагноз с
id=${saveDto.patientId} не найден"),
                    name = saveDto.name,
                    dateOfDiagnosis = saveDto.dateOfDiagnosis,
                    description = saveDto.description,
                    isTreated = saveDto.isTreated
                )
            )
        )
}

```

```

        )
    )
)

override fun updateDiagnose(id: Long, updateDto: DiagnoseUpdateDto) =
    DiagnoseRsDto.fromEntity(diagnoseRepository.findDiagnoseById(id)?.let {
        diagnose ->
            diagnoseRepository.save(
                diagnose.copy(
                    patient = if(updateDto.patientId != null) {
                        userRepository.findUserById(updateDto.patientId)
                            ?: throw EntityNotFoundException("Пользователь
${updateDto.patientId} не найдена")
                    } else diagnose.patient,
                    name = updateDto.name.takeIf { it != null } ?:
diagnose.name,
                    dateOfDiagnosis = updateDto.dateOfDiagnosis.takeIf {
it != null } ?: diagnose.dateOfDiagnosis,
                    description = updateDto.description.takeIf { it !=
null } ?: diagnose.description,
                    isTreated = updateDto.isTreated.takeIf { it != null }
?: diagnose.isTreated,
                )
            )
        } ?: throw EntityNotFoundException("Диагноз с id=$id не найден"))
    }

```

Листинг В.7 – Файл KafkaProducer.kt

```

@Component
class KafkaProducer(
    private val kafkaSender: KafkaSender<String, String>,
    private val mapper: ObjectMapper
) {

    suspend fun sendDepartmentSavePayload(value: DepartmentSavePayload):
Disposable {
        val producerRecord = ProducerRecord("admin.department.save",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendDepartmentUpdatePayload(value:
DepartmentUpdatePayload): Disposable {
        val producerRecord = ProducerRecord("admin.department.update",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendDepartmentDeactivatePayload(value:
DepartmentDeactivatePayload): Disposable {
        val producerRecord =
ProducerRecord("admin.department.deactivate",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
    }

```

```

        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendDepartmentActivatePayload(value:
DepartmentActivatePayload): Disposable {
        val producerRecord = ProducerRecord("admin.department.activate",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendHospitalSavePayload(value: HospitalSavePayload):
Disposable {
        val producerRecord = ProducerRecord("admin.hospital.save",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendHospitalUpdatePayload(value: HospitalUpdatePayload):
Disposable {
        val producerRecord = ProducerRecord("admin.hospital.update",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendHospitalDeactivatePayload(value:
HospitalDeactivatePayload): Disposable {
        val producerRecord = ProducerRecord("admin.hospital.deactivate",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendHospitalActivatePayload(value:
HospitalActivatePayload): Disposable {
        val producerRecord = ProducerRecord("admin.hospital.activate",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendUserSavePayload(value: UserSavePayload): Disposable {
        val producerRecord = ProducerRecord("admin.user.save",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendUserUpdatePayload(value: UserUpdatePayload):
Disposable {
        val producerRecord = ProducerRecord("admin.user.update",
UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

```

```

    }

    suspend fun sendUserDeactivatePayload(value: UserDeactivatePayload):
    Disposable {
        val producerRecord = ProducerRecord("admin.user.deactivate",
        UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
        producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }

    suspend fun sendUserActivatePayload(value: UserActivatePayload):
    Disposable {
        val producerRecord = ProducerRecord("admin.user.activate",
        UUID.randomUUID().toString(), mapper.writeValueAsString(value))
        return mono { SenderRecord.create(producerRecord,
        producerRecord.key()) }.let { kafkaSender.send(it) }.subscribe()
    }
}

```

Листинг В.8 – Файл UserSaveConsumer.kt

```

@Component
class UserSaveConsumer(
    private val userSaveReceiver: KafkaReceiver<String, String>,
    private val kafkaProducer: KafkaProducer,
    private val userService: UserService,
    private val mapper: ObjectMapper
) {
    private val disposables = Disposables.composite()

    @PostConstruct
    fun connect() {
        disposables.add(receiver())
    }

    @PreDestroy
    fun disconnect() {
        this.disposables.dispose()
    }

    fun receiver(): Disposable =
        userSaveReceiver.receive()
            .doOnNext { receivedMessage ->
                runCatching {
                    mapper.readValue<UserSavePayload>(receivedMessage.value())
                }.onSuccess { saveRequest ->
                    runBlocking {
                        val saveResult =
                            userService.saveUser(saveRequest)
                        if (saveResult.isSuccess) {
                            kafkaProducer.sendRequestResultPayload(
                                RequestResultPayload(
                                    requestId = saveRequest.requestId,
                                    isRequestSuccessful = true,
                                    message = null
                                )
                            )
                        }
                    }
                }
            }
    }
}

```

```

        )
    } else {
        kafkaProducer.sendRequestResultPayload(
            RequestResultPayload(
                requestId = saveRequest.requestId,
                isRequestSuccessful = false,
                message = "Произошла ошибка при
сохранении пользователя"
            )
        )
    }
}
}.also {
    receivedMessage.receiverOffset().acknowledge()
}
}.subscribe()
}

```

Листинг В.9 – Файл KafkaConfig.kt

```

@Configuration
class KafkaConfig {
    @Bean
    fun kafkaSender(): KafkaSender<String, String> {
        return KafkaSender.create(getSenderOptions())
    }

    @Bean
    fun userDeleteReceiver(): KafkaReceiver<String, String> {
        return
KafkaReceiver.create(getReceiverOptions("admin.user.delete"))
    }

    @Bean
    fun userSaveReceiver(): KafkaReceiver<String, String> {
        return
KafkaReceiver.create(getReceiverOptions("admin.user.save"))
    }

    @Bean
    fun userUpdateReceiver(): KafkaReceiver<String, String> {
        return
KafkaReceiver.create(getReceiverOptions("admin.user.update"))
    }

    @Bean
    fun userDeactivateReceiver(): KafkaReceiver<String, String> {
        return
KafkaReceiver.create(getReceiverOptions("admin.user.deactivate"))
    }

    @Bean
    fun userActivateReceiver(): KafkaReceiver<String, String> {
        return
KafkaReceiver.create(getReceiverOptions("admin.user.activate"))
    }
}

```

```

        private fun getSenderOptions(): SenderOptions<String, String> {
            val producerProps = Properties()
            producerProps[ProducerConfig.BOOTSTRAP_SERVERS_CONFIG] =
"localhost:9092"
            producerProps[ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG] =
StringSerializer::class.java
            producerProps[ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG] =
StringSerializer::class.java
            return SenderOptions.create<String,
String>(producerProps).maxInFlight(1024)
        }

        private fun getReceiverOptions(topic: String):
ReceiverOptions<String, String> {
            val consumerProps = Properties()
            consumerProps[ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG] =
"localhost:9092"
            consumerProps[ConsumerConfig.GROUP_ID_CONFIG] = "patient-
receiver"
            consumerProps[ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG] =
StringDeserializer::class.java
            consumerProps[ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG] =
StringDeserializer::class.java
            consumerProps[ConsumerConfig.AUTO_OFFSET_RESET_CONFIG] =
"earliest"
            return ReceiverOptions.create<String,
String>(consumerProps).subscription(listOf(topic))
        }
    }
}

```

Листинг В.10 – Файл MainWindow.xaml.cs

```

namespace HospitalClient
{
    public partial class MainWindow : Window
    {
        private UserDataModel _currentUser;
        public MainWindow()
        {
            ServerRestManager.GetInstance("192.168.68.109", "8080");
            InitializeComponent();
            Authorization();
        }

        private void Authorization()
        {
            this.Hide();
            AuthWindow authWindow = new AuthWindow();
            authWindow.SetCreatingForm = this;
            authWindow.ShowDialog();
            if (authWindow.AuthStatus)
            {
                _currentUser = authWindow.userAuthData;
                this.Show();
            }
            else
            {

```

```

    }
}
private void ShowRolePages()
{
    foreach (var item in ButtonsList.Items)
    {
        var button = item as PanelButton;
        button.Visibility = Visibility.Collapsed;
    }

    PersonalAccountButton.Visibility = Visibility.Visible;

    switch (ServerRestManager.Role)
    {
        case ServerRestManager.UserRole.USER:
        {
            MedicalCardButton.Visibility =
Visibility.Visible;
            DoctorsAppointmentButton.Visibility =
Visibility.Visible;
            break;
        }
        case ServerRestManager.UserRole.DOCTOR:
        {
            PatientsListButton.Visibility =
Visibility.Visible;
            break;
        }
        case ServerRestManager.UserRole.ADMIN:
        {
            AdminPanelButton.Visibility = Visibility.Visible;
            LoginLogsButton.Visibility = Visibility.Visible;
            break;
        }
        default: break;
    }
}

private void PersonalAccountButton_Click(object sender,
MouseButtonEventArgs e)
{
    MainFrame.Content = new
PersonalAccountPage(_currentUser.Login, _currentUser.Name,
_currentUser.BirthDate, _currentUser.CreatedAt);
}

private void MedicalCardButton_Click(object sender,
MouseButtonEventArgs e)
{
    var diagnoses =
(List<DiagnosisModel>)ServerRestManager.Diagnosis.GetAllDiagnosisByUserId
(_currentUser.Id);
    MainFrame.Content = new MedicalCardPage(diagnoses);
}

private void DoctorsAppointmentButton_Click(object sender,
MouseButtonEventArgs e)

```

```

        {
            List<AppointmentDataModel> appointments = new
List<AppointmentDataModel>();
            if (ServerRestManager.Role ==
ServerRestManager.UserRole.USER)
                appointments =
(List<AppointmentDataModel>) ServerRestManager.Appointments.GetAppointment
sByUserId(_currentUser.Id);
            else
            {
                appointments =
(List<AppointmentDataModel>) ServerRestManager.Appointments.GetAppointment
sByDoctorId(_currentUser.Id);
            }
            MainFrame.Content = new DoctorsAppointmentPage(appointments);
        }

        private void PacientsListButton_Click(object sender,
MouseButtonEventArgs e)
        {
            MainFrame.Content = new PatientsListPage();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            ShowRolePages();
            lblUsernameHead.Content = _currentUser.Name;
            PersonalAccountButton_Click(null, null);
        }
    }
}

```

Приложение Г
(рекомендуемое)
Результаты тестирования

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1
  }'
```

Request URL

http://localhost:8080/api/appointments

Server response

Code	Details
400	Undocumented Error: response status is 400

Response body

```
{
  "error": "BAD_REQUEST",
  "description": "Запись не создана: поликлиника не выбрана; отделение не выбрано; врач не выбран; время не выбрано;"
}
```

Рисунок Г.1 – Функциональный тест 1

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1,
    "hospitalId": 1,
    "departmentId": 1
  }'
```

Request URL

http://localhost:8080/api/appointments

Server response

Code	Details
400	Undocumented Error: response status is 400

Response body

```
{
  "error": "BAD_REQUEST",
  "description": "Запись не создана: врач не выбран; время не выбрано;"
}
```

Рисунок Г.2 – Функциональный тест 4

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1,
    "doctorId": 9,
    "departmentId": 1
  }'
```

Request URL

http://localhost:8080/api/appointments

Server response

Code	Details
400	<p><i>Undocumented</i> Error: response status is 400</p> <p>Response body</p> <pre>{ "error": "BAD_REQUEST", "description": "Запись не создана: поликлиника не выбрана; время не выбрано;" }</pre>

Рисунок Г.3 – Функциональный тест 7

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1,
    "doctorId": 9,
    "hospitalId": 1,
    "departmentId": 1
  }'
```

Request URL

http://localhost:8080/api/appointments

Server response

Code	Details
400	<p><i>Undocumented</i> Error: response status is 400</p> <p>Response body</p> <pre>{ "error": "BAD_REQUEST", "description": "Запись не создана: время не выбрано;" }</pre>

Рисунок Г.4 – Функциональный тест 8

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1,
    "hospitalId": 1
    "departmentId": 1
    "time": "2024-06-05T11:00:00.000Z"
  }'
```

Request URL

`http://localhost:8080/api/appointments`

Server response

Code	Details
400	<p><i>Undocumented</i> Error: response status is 400</p> <p>Response body</p> <pre>{ "error": "BAD_REQUEST", "description": "Запись не создана: врач не выбран" }</pre>

Рисунок Г.5 – Функциональный тест 12

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/appointments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "patientId": 1,
    "hospitalId": 1,
    "doctorId": 9,
    "time": "2024-06-05T11:00:00.000Z"
  }'
```

Request URL

`http://localhost:8080/api/appointments`

Server response

Code	Details
400	<p><i>Undocumented</i> Error: response status is 400</p> <p>Response body</p> <pre>{ "error": "BAD_REQUEST", "description": "Запись не создана: отделение не выбрано;" }</pre>

Рисунок Г.6 – Функциональный тест 14

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/appointments/1' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/appointments/1
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "patientId": 1, "doctorId": 9, "hospitalId": 1, "departmentId": 1, "time": "2024-06-05T10:00:00", "description": "Жалобы на головную боль.", "isCanceled": true }</pre>

Рисунок Г.7 – Функциональный тест 16

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/appointments/3' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/appointments/3
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 3, "patientId": 3, "doctorId": 9, "hospitalId": 3, "departmentId": 1, "time": "2024-05-20T12:00:00", "description": "Обследование.", "isCanceled": false }</pre>

Рисунок Г.8 – Функциональный тест 17