

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

**Разработка веб-приложения для поиска и
бронирования лекарственных средств в
аптечной сети**

Студент _____ Костригин Артём Олегович _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Пензенский государственный университет
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Костригин Артём Олегович гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка веб-приложения для поиска и бронирования
лекарственных средств в аптечной сети

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025г.

3. Руководитель работы д.т.н., профессор кафедры «МО и ПЭВМ» Кольчугина Е.А.

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

1) Назначение: программные средства для поиска и бронирования лекарственных средств в аптечной сети.

2) Среда выполнения: Windows 11, macOS.

Языковые средства разработки: TypeScript, SQL, UML.

Инструментальные средства: Visual Studio Code, PostgreSQL, Prisma ORM, Visual Paradigm Online.

Технологии разработки: клиент-серверная архитектура, Rest API, Express

3) Основные функции приложения:

а) регистрация и авторизация пользователя;

б) поиск и просмотр лекарственных средств;

в) просмотр аптечных предложений с ценой и остатком;

г) создание и отмена бронирований;

д) обработка бронирований сотрудником аптеки;

е) административное управление системой;

5. Перечень подлежащих разработке вопросов

- 1) *Анализ предметной области и требований к системе;*
- 2) *Проектирование программных средств;*
- 3) *Реализация программных средств;*
- 4) *Контроль качества программного обеспечения;*
- 5) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

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

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 66 листов, 27 рисунков, 18 таблиц, 23 использованных источника и 2 приложения.

ЛЕКАРСТВЕННОЕ СРЕДСТВО, АПТЕКА, БРОНИРОВАНИЕ, ВЕБ-ПРИЛОЖЕНИЕ, REACT, TYPESCRIPT, NODE.JS, EXPRESS, POSTGRESQL, PRISMA ORM, БАЗА ДАННЫХ.

Объект исследования – программные средства для поиска лекарственных средств и оформления предварительного бронирования в аптечной сети.

Цель выпускной квалификационной работы – спроектировать и разработать веб-приложение, позволяющее пользователю находить препараты, просматривать аптечные предложения и создавать заявки на бронирование.

Технологии разработки – разработка клиентской части приложения проводилась с использованием TypeScript, React и Vite. Серверная часть реализована на Node.js с применением Express. Для хранения данных используется PostgreSQL, взаимодействие с базой данных выполняется через Prisma ORM.

Среда исполнения – персональный компьютер или сервер с установленными Node.js, npm и PostgreSQL, а так же современный браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были спроектированы и разработаны программные средства для поиска лекарственных средств, отображения аптечных предложений, создания и обработки бронирований. Реализованы необходимые для функционирования приложения функции. Основные требования к разрабатываемому приложению выполнены.

					ПГУ 09.03.04 – 08BP222.16 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Костригин А.О.			Разработка веб-приложения для поиска и бронирования лекарственных средств в аптечной сети	Лит.	Лист	Листов	
Пров.		Кольчугина Е.А.					4	66	
Реценз.						Группа 22ВП2			
Н. Контр.		Такташкин Д.В.							
Утв.		Козлов А.Ю.							

Содержание

Введение	7
1 Анализ предметной области и постановка требований к приложению для поиска и бронирования лекарственных средств	9
1.1 Назначение и условия применения приложения для поиска и бронирования лекарственных средств	9
1.2 Сравнительный анализ продуктов-аналогов	9
1.3 Анализ функциональных требований	13
1.4 Анализ нефункциональных требований	17
1.5 Оценка бюджета и планирование разработки	17
2 Проектирование веб-приложения для поиска и бронирования лекарственных средств	20
2.1 Архитектура приложения	20
2.2 Проектирование структур данных	21
2.2.1 Концептуальная модель базы данных	21
2.2.2 Логическая модель базы данных	23
2.3 Проектирование программных компонентов	24
2.4 Проектирование структуры пользовательского интерфейса	26
3 Реализация веб-приложения для поиска и бронирования лекарственных средств в аптечной сети	28
3.1 Реализация серверной части приложения	28
3.2 Реализация структуры хранения данных	30
3.3 Реализация пользовательского интерфейса	34
4 Тестирование веб-приложения для поиска и бронирования лекарств в аптечной сети	39
4.1 Тестирование методом «белого ящика»	39
4.1.1 Покрытие операторов	41
4.1.2 Покрытие решений	42
4.1.3 Комбинаторное покрытие условий	43
4.2 Модульное тестирование	44

	6
4.3 Оценка качества кода	46
4.3.1 Метрика Чепина	46
4.3.2 Метрики Холстеда	47
4.3.3 SLOC тестирование	49
Заключение	51
Список использованных источников	52
Приложение А (справочное) Глоссарий	55
Приложение Б (обязательное) Листинг приложения	57

Введение

Покупка лекарств зачастую начинается не с посещения аптеки, а с попытки заранее узнать, в какой точке есть нужный препарат, какова его цена и получится ли забрать его без ожидания. Зачастую лекарства требуются быстро, а обход нескольких аптек или серия телефонных звонков отнимают время и не всегда дают точный результат. Если сведения о наличии, цене и адресах аптек хранятся разрозненно, пользователь не может быстро сравнить варианты и выбрать наиболее подходящую точку для получения.

Для аптечной сети такая задача тоже остается значимой. Недостаточно просто показать пользователю список препаратов: Система должна учитывать конкретные аптеки, остатки на местах, уже зарезервированное количество товара и состояние каждой заявки. Иначе бронирование превращается в формальную запись, которая не связана с реальным наличием препарата.

Актуальность выпускной квалификационной работы связана с необходимостью создания удобного цифрового инструмента для поиска и предварительного бронирования лекарственных средств в аптечной сети. Такое приложение позволяет пользователю быстрее найти подходящий препарат в нужном районе, увидеть доступное количество и оформить бронь, а сотрудникам аптеки и администратору – вести заявки, справочники и остатки в едином информационном пространстве.

Цель работы заключается в проектировании и реализации клиент-серверного веб-приложения, с помощью которого пользователь может найти лекарственный препарат в аптечной сети и оформить бронирование.

Для достижения цели необходимо решить следующие задачи:

- проанализировать предметную область, связанную с поиском лекарств, учётом аптек, складских остатков и обработкой бронирований;
- рассмотреть существующие сервисы поиска, заказа и бронирования аптечных товаров, определить их особенности и ограничения;
- сформулировать функциональные и нефункциональные требования

к разрабатываемому приложению;

- спроектировать архитектуру приложения, структуру базы данных, основные программные сущности и пользовательский интерфейс;
- реализовать клиентскую часть, серверное API, базу данных, авторизацию пользователей и ролевую модель доступа;
- выполнить проверку работоспособности приложения и оценить качество программного кода.

1 Анализ предметной области и постановка требований к приложению для поиска и бронирования лекарственных средств

1.1 Назначение и условия применения приложения для поиска и бронирования лекарственных средств

Перед посещением аптеки пользователю необходимо узнать, есть ли нужный лекарственный препарат в наличии, сколько он стоит и в какой точке его можно приобрести. Для этого используются веб-сервисы поиска аптечных товаров, которые позволяют заранее получить сведения о препаратах и выбрать удобную аптеку без ручного обзвона нескольких точек.

Приложение для поиска и бронирования лекарственных средств – это программное средство, предназначенное для просмотра каталога препаратов, отображения аптечных предложений и оформления предварительной брони выбранного товара. С его помощью пользователь может сократить время поиска лекарства, сравнить доступные варианты и получить подтверждение о брони до посещения аптеки.

Многие существующие аптечные сервисы ориентированы на крупные маркетплейсы, федеральные агрегаторы или крупные аптечные сети, из-за чего пользователь часто сталкивается с избыточным количеством разделов, разными условиями получения и необходимостью проверять сведения в нескольких источниках. В связи с этим актуальной задачей является разработка приложения, предоставляющего пользователю сведения о наличии лекарств в аптеках и возможность предварительного бронирования в рамках единой системы.

1.2 Сравнительный анализ продуктов-аналогов

Для сравнительного анализа были выбраны сервисы Apteka.ru, Здравсити и Мегаптека. Они связаны с поиском лекарственных средств, заказом аптечных товаров и выбором аптеки для получения. Анализ направлен на выявление сильных и слабых сторон существующих решений, а также на определение

функций, которые должны быть учтены при разработке приложения для поиска и бронирования лекарственных средств.

«Apteka.ru» является крупным сервисом заказа аптечных препаратов с получением в выбранной аптеке [1]. Пользователь может найти препарат, ознакомиться с карточкой товара и оформить заказ. Преимуществом сервиса является широкий каталог и привычный сценарий оформления. Недостатком для рассматриваемой задачи является то, что пользователь вынужден проходить именно долгую процедуру заказа, а не короткий сценарий предварительного бронирования выбранного средства в удобной аптеке. Интерфейс сервиса представлен на рисунке 1.

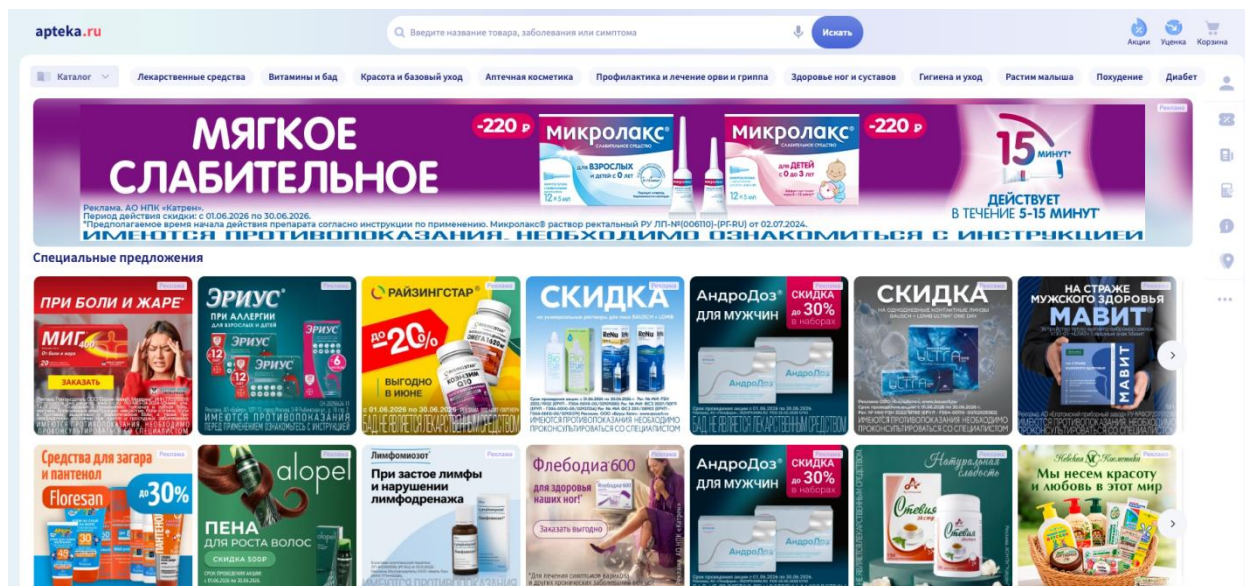


Рисунок 1 – Интерфейс сервиса «Apteka.ru»

«Здравсити» так же относится к крупным сервисам заказа товаров для здоровья [2]. В нём представлены лекарственные препараты, товары медицинского назначения, косметика, БАДы и другие категории. Широкий ассортимент делает сервис удобным для массового пользователя, однако задачи быстрого поиска конкретного лекарства такая насыщенность интерфейса может усложнять восприятие. Пользователю приходится работать с большим каталогом, где поиск лекарства является только одним из нескольких направлений. Интерфейс сервиса представлен на рисунке 2.

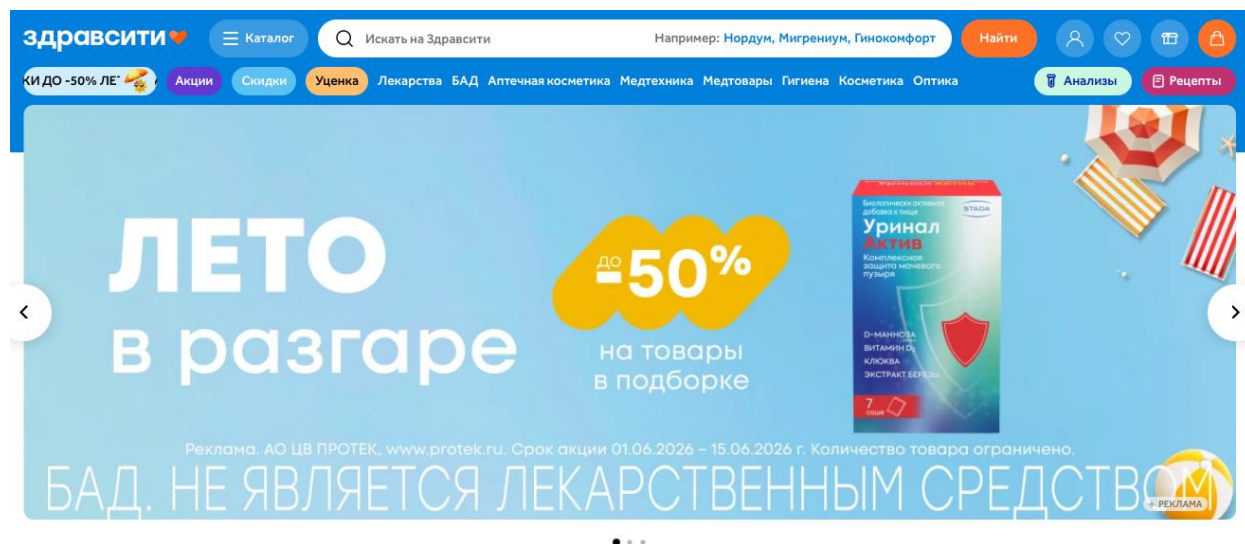


Рисунок 2 – Интерфейс сервиса «Здравсити»

«Мегаптека» наиболее приближена к теме работы, так как делает акцент на поиске аптечных предложений и бронировании товара [3]. Пользователь может сравнить цены, выбрать аптеку и оформить заявку. недостатком является то, что точные условия бронирования пользователь узнает отдельно после оформления заказа. Интерфейс сервиса представлен на рисунке 3.

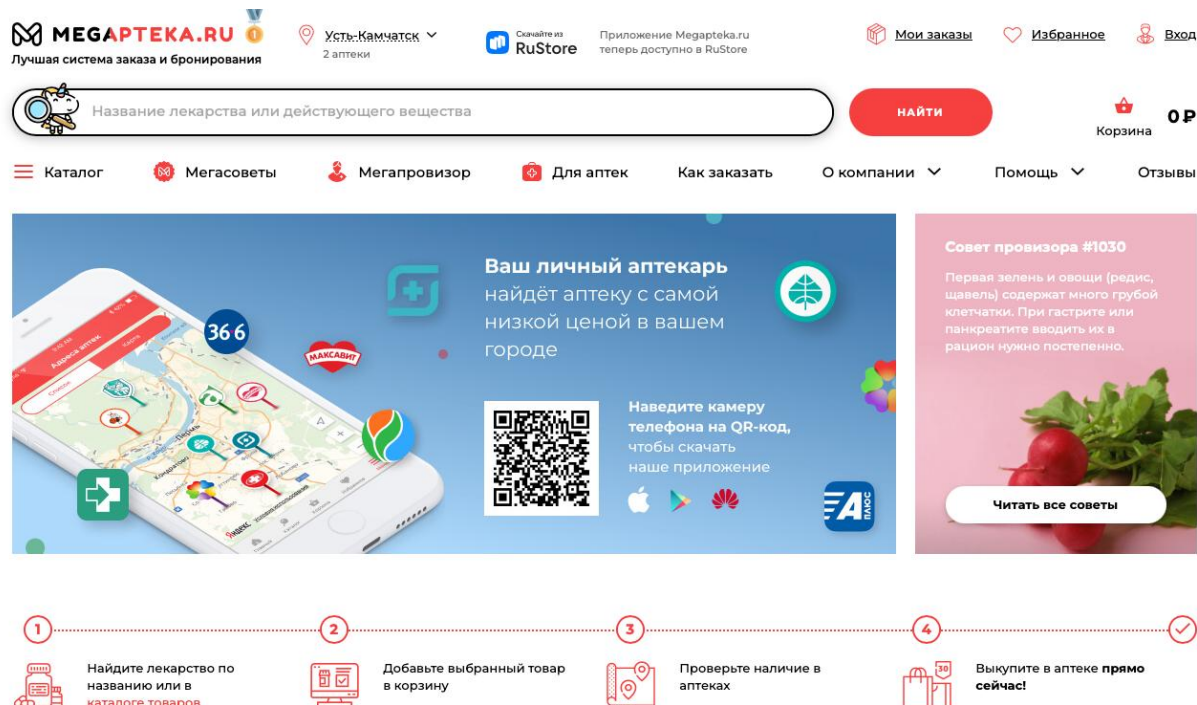


Рисунок 3 – Интерфейс сервиса «Мегаптека»

Для сравнения рассмотренных решений выделены критерии с балльной системой оценки:

- 0 – отсутствие реализации;
- 1 – реализовано частично;
- 2 – реализовано в полной мере.

Результаты сравнения продуктов-аналогов по критериям, важным для разрабатываемого приложения, приведены в таблице 1.

Таблица 1 – Результаты сравнения продуктов-аналогов

Критерий оценивания	Arteka.ru	Здравсити	Мегаптека
Интуитивно понятный интерфейс	2	2	2
Поиск лекарственного средства по названию	2	2	2
Выбор аптеки для получения товара	2	2	2
Фильтрация по расположению аптеки	1	1	2
Отображение цены в выбранной аптеке	1	1	2
Отображение точного доступного количества товара	0	0	0
Предварительное бронирование без длинного оформления заказа	0	0	1
Просмотр статуса созданной заявки	1	1	1
Короткий пользовательский путь от поиска до брони	0	0	1
Отсутствие перегрузки лишними критериями товаров	1	0	1
Итого	10	9	14

Анализируя сравнительную таблицу, можно сделать вывод, что рассматриваемые сервисы предоставляют пользователю широкий набор возможностей, однако не полностью закрывают задачу простого предварительного бронирования лекарства. Решения перегружены широким

каталогом, ориентированы на оформление заказа, а не на быструю предварительную бронь. Поэтому необходимо разработать приложение, которое будет предоставлять пользователям следующий функционал:

- поиск лекарственного средства по названию;
- просмотр карточки товара и аптечных предложений;
- отображение цены и доступного количества в конкретной аптеке;
- фильтрацию предложений по аптеке, городу, району, цене и наличию;
- предварительное бронирование выбранного количества товара;
- удобное отображение статуса бронирования для пользователя;
- простой интерфейс без лишних товарных и рекламных разделов.

Исходя из проведённого анализа, принято решение разработать приложение для поиска и бронирования лекарственных средств, ориентированное на работу с конкретной аптечной сетью. такое приложение должно объединять каталог препаратов, сведения об аптеках, складские остатки и бронирование в едином пользовательском сценарии.

1.3 Анализ функциональных требований

Функциональные требования определяют действия, которые система должна предоставлять пользователям. В приложении есть несколько ролей, поэтому требования будет удобно разделить по участникам процесса.

Неавторизованный посетитель должен иметь доступ к публичной части системы. Он может просматривать каталог лекарств, искать препарат по названию, действующему веществу или производителю, а так же фильтровать предложения по аптеке, городу, району, цене и наличию. Кроме того, посетитель может открыть список аптек и посмотреть адреса, телефоны и график работы. Создание бронирования для него недоступно до входа в систему.

Клиент после авторизации получает возможность оформить бронь. Для этого он выбирает препарат, указывает количество и отправляет заявку. Если

данные корректны, пользователю показывается код брони и срок действия заявки. В личном кабинете пользователь должен видеть только свои заявки и иметь возможность отменить активную бронь.

Сотрудник аптеки работает с заявками, относящимися к его аптеке. Он должен видеть список таких бронирований, подтверждать заявку или отменять её при невозможности выдачи, а так же завершать при передаче заказа клиенту. Сотрудник не должен получать доступ к броням других аптек.

Администратор отвечает за сопровождение данных системы. Ему нужны функции просмотра статистики, управления пользователями и ролями, привязки сотрудника к аптеке, добавления лекарств, добавления аптек и обновления складских остатков. Эти функции позволят поддерживать каталог и бронирования без прямого редактирования базы данных.

В соответствии с требованиями была разработана представленная на рисунке 4 диаграмма вариантов использования UML.



Рисунок 4 – Диаграмма вариантов использования

Диаграмма вариантов использования в UML – визуальная схема, которая показывает, кто взаимодействует с разрабатываемой системой и какие именно функции (варианты использования) она должна для них выполнять [4].

Диаграмма даёт общее представление о функционировании системы. Ниже приведены сценарии на примере основных вариантов использования, которые подразумевают работу с поиском и бронированием медикаментов пользователем.

Сценарий варианта использования «Найти лекарство в каталоге» представлен на рисунке 5.

Прецедент: Найти лекарство в каталоге
ID: 1
Краткое описание: Пользователь ищет лекарственный препарат по названию, аптеке, городу, району, цене или наличию
Главные акторы: Посетитель, Клиент
Второстепенные акторы: Нет
Предусловие: Пользователь открыл веб-приложение
Основной поток: 1. Пользователь открывает каталог лекарств 2. Пользователь вводит параметры поиска и/или выбирает фильтры 3. Система отправляет запрос к базе данных 4. Система отображает список найденных препаратов 5. Пользователь открывает подробную карточку товара
Постусловие: Пользователь получает информацию о наличии лекарства и может перейти к бронированию

Рисунок 5 – Сценарий варианта использования «Найти лекарство в каталоге»

Сценарий варианта использования «Создать бронирование» представлен на рисунке 6.

Прецедент: Создать бронирование
ID: 2
Краткое описание: Клиент выбирает лекарство в конкретной аптеке и создаёт бронь на нужное количество
Главные акторы: Клиент
Второстепенные акторы: Нет
Предусловие: Клиент авторизован, лекарство есть в наличии
Основной поток: 1. Клиент выбирает препарат и аптеку 2. Клиент указывает нужное количество лекарства 3. Клиент нажимает кнопку бронирования 4. Система проверяет доступный остаток 5. Система создаёт запись бронирования 6. Система увеличивает количество зарезервированного товара
Постусловие: Бронирование сделано, клиент видит код брони

Рисунок 6 – Сценарий варианта использования «Создать бронирование»

Сценарий варианта использования «Отменить свою бронь» представлен на рисунке 7

Прецедент: Отменить свою бронь
ID: 3
Краткое описание: Клиент просматривает список бронирований и отменяет активную бронь
Главные акторы: Клиент
Второстепенные акторы: Нет
Предусловие: Клиент авторизован и имеет хотя бы одно бронирование
Основной поток: 1. Клиент открывает раздел бронирований 2. Система загружает список броней текущего пользователя 3. Клиент нажимает кнопку отмены бронирования 4. Система меняет статус брони 5. Система уменьшает количество зарезервированного товара
Постусловие: Бронь отменена, товар снова доступен для бронирования

Рисунок 7 – Сценарий варианта использования «Отменить свою бронь»

Выделенные требования покрывают полный путь работы приложения: от публичного просмотра каталога, до внутренней обработки заявок аптекой.

1.4 Анализ нефункциональных требований

Нефункциональные требования описывают свойства приложения, которые не сводятся к отдельным кнопкам или операциям, но напрямую влияют на качество продукта. Для системы поиска и бронирования лекарств важны в первую очередь удобство, безопасность и надёжность.

Разработка веб-приложения должна быть выполнена в среде разработки Visual Studio Code с использованием клиент-серверной архитектуры [5, 6]. Клиентская часть приложения должна быть реализована на языке TypeScript с применением библиотеки React и сборщика Vite [7]. Серверная часть должна быть разработана на платформе Node.js с использованием фреймворка Express и языка TypeScript.

В качестве системы управления базой данных должна использоваться PostgreSQL [8]. Взаимодействие серверной части с базой данных необходимо выполнить с помощью Prisma ORM, что позволит описать структуру таблиц в схеме проекта и выполнять операции с данными через программный интерфейс.

Пользовательская часть должна быть оптимизирована для работы в большинстве современных браузеров, таких как Яндекс Браузер, Apple Safari, Google Chrome, Microsoft Edge.

1.5 Оценка бюджета и планирование разработки

Разработка приложения была разделена на несколько этапов: анализ предметной области, проектирование, настройка окружения и базы данных, реализация серверной и клиентской части, интеграция, тестирование и подготовка пояснительной записки. Такое разделение позволяет последовательно перейти от требований к рабочему программному продукту и не смешивать аналитические, проектные и реализационные задачи.

Для эффективного управления процессом разработки проекта

необходимо установить задачи, установить сроки и приоритеты а так же рассчитать смету. Для удобного управления проектом была выбрана программа ProjectLibre, в которую были внесены все данные процесса разработки приложения. Список выделенных задач, а так же сроков реализации представлен на рисунке 8.

	Ⓢ	Название	Продолжи...	Начало	Окончание	Предшествующие	Название ресурса
1		Анализ и требования	8 дней	03.03.2026, 08:00	12.03.2026, 17:00		
2		Анализ предметной области	4 дней	03.03.2026, 08:00	06.03.2026, 17:00		Аналитик
3		Анализ продуктов-аналогов	2 дней	09.03.2026, 08:00	10.03.2026, 17:00	2	Аналитик; Интернет; Электричество
4		Описание ролей, сценариев и треб	2 дней	11.03.2026, 08:00	12.03.2026, 17:00	3	Аналитик
5		Проектирование	10 дней	13.03.2026, 08:00	26.03.2026, 17:00		
6		Проектирование архитектуры прил	1 день	13.03.2026, 08:00	13.03.2026, 17:00	4	Backend-разработчик; Аналитик
7		Проектирование базы данных	2 дней	16.03.2026, 08:00	17.03.2026, 17:00	6	Аналитик; Backend-разработчик
8		Проектирование API, интерфейса и	5 дней	18.03.2026, 08:00	24.03.2026, 17:00	6,7	Аналитик; Backend-разработчик
9		Подготовка UML и ER-диаграмм	2 дней	25.03.2026, 08:00	26.03.2026, 17:00	8	Аналитик; Backend-разработчик; Интернет; Электричество
10		Окружение и база данных	4 дней	27.03.2026, 08:00	01.04.2026, 17:00		
11		Создание структуры проекта	1 день	27.03.2026, 08:00	27.03.2026, 17:00	9	Backend-разработчик; Frontend-разработчик; ЭВМ; Интерн...
12		Настройка PostgreSQL, Prisma и тест	3 дней	30.03.2026, 08:00	01.04.2026, 17:00	11	Backend-разработчик; ЭВМ; Интернет; Электричество
13		Backend-разработка	11 дней	02.04.2026, 08:00	16.04.2026, 17:00		
14		Реализация базового Express-прилс	2 дней	02.04.2026, 08:00	03.04.2026, 17:00	12	Backend-разработчик; ЭВМ; Интернет; Электричество
15		Реализация авторизации и ролей	2 дней	06.04.2026, 08:00	07.04.2026, 17:00	14	Backend-разработчик; ЭВМ; Интернет; Электричество
16		Реализация API каталога и аптек	2 дней	08.04.2026, 08:00	09.04.2026, 17:00	15	Backend-разработчик; ЭВМ; Интернет; Электричество
17		Реализация API бронирования	3 дней	10.04.2026, 08:00	14.04.2026, 17:00	16	Backend-разработчик; ЭВМ; Интернет; Электричество
18		Реализация API администрирования	2 дней	15.04.2026, 08:00	16.04.2026, 17:00	17	Backend-разработчик; ЭВМ; Интернет; Электричество
19		Frontend-разработка	11 дней	17.04.2026, 08:00	01.05.2026, 17:00		
20		Настройка React-приложения и авт	2 дней	17.04.2026, 08:00	20.04.2026, 17:00	18	Frontend-разработчик; ЭВМ; Интернет; Электричество
21		Реализация навигации и страниц в	1 день	21.04.2026, 08:00	21.04.2026, 17:00	20	Frontend-разработчик; ЭВМ; Интернет; Электричество
22		Реализация каталога, фильтров и б	4 дней	22.04.2026, 08:00	27.04.2026, 17:00	21	Frontend-разработчик; Интернет; Электричество; ЭВМ
23		Реализация страницы аптек и остат	1 день	28.04.2026, 08:00	28.04.2026, 17:00	22	Frontend-разработчик; ЭВМ; Интернет; Электричество
24		Реализация страницы бронирования	2 дней	29.04.2026, 08:00	30.04.2026, 17:00	23	Frontend-разработчик; ЭВМ; Интернет; Электричество
25		Реализация админ-панели	1 день	01.05.2026, 08:00	01.05.2026, 17:00	24	Frontend-разработчик; ЭВМ; Интернет; Электричество
26		Интеграция и тестирование	4,5 дней	04.05.2026, 08:00	08.05.2026, 13:00		
27		Интеграция frontend и backend	1,5 дней	04.05.2026, 08:00	05.05.2026, 13:00	25	Backend-разработчик; Frontend-разработчик; ЭВМ; Интерн...
28		Функциональное тестирование	2 дней	05.05.2026, 13:00	07.05.2026, 13:00	27	Тестировщик; ЭВМ; Интернет; Электричество
29		Исправление ошибок и проверка сб	1 день	07.05.2026, 13:00	08.05.2026, 13:00	28	Backend-разработчик; Frontend-разработчик; Тестировщи...
30		Документирование	3 дней	08.05.2026, 13:00	13.05.2026, 13:00		
31		Составление пояснительной записк	3 дней	08.05.2026, 13:00	13.05.2026, 13:00	29	Аналитик; Backend-разработчик; Frontend-разработчик; Т...

Рисунок 8 – Список этапов и задач проекта

Диаграмма Ганта является одной из форм календарного плана, необходимого для определения длительности как всего проекта, так и его фаз [5]. Диаграмма Ганта проекта представлена на рисунке 9.

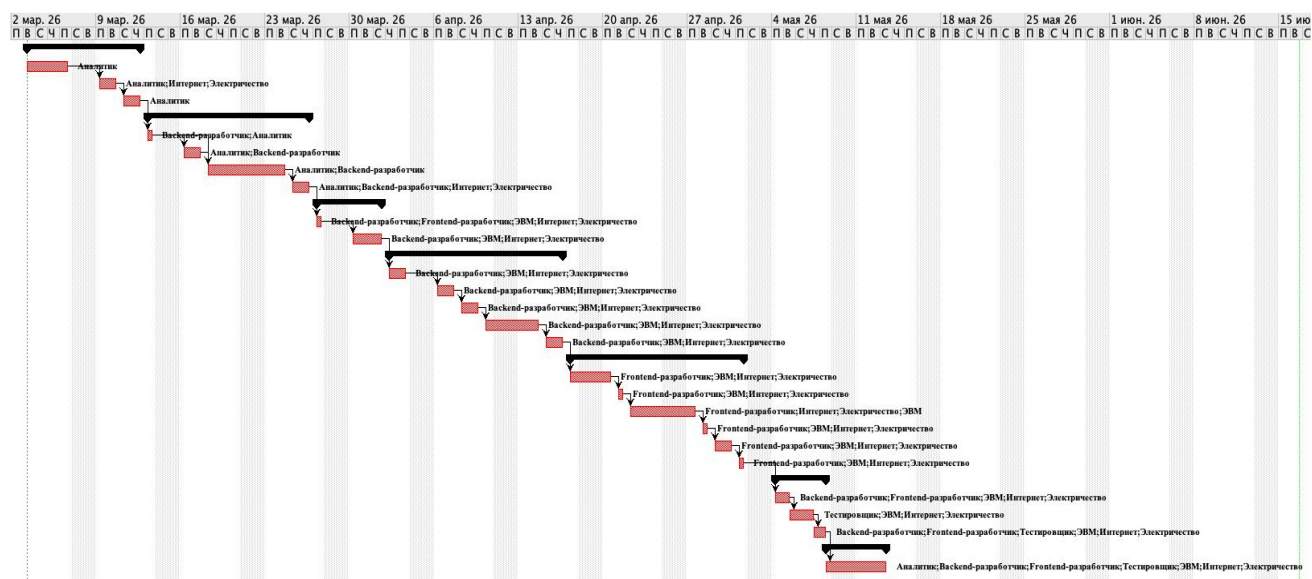


Рисунок 9 – Диаграмма Ганта

Для оценки бюджета использована расчётная модель проекта. В плане выделены роли аналитика, frontend и backend разработчика, а так же затраты и материальные ресурсы. Оценка затрат каждого ресурса представлена на рисунке 10.

	🔗	Название	Тип	Максимальные единицы	Стандартная ставка	Ставка сверхурочных
1	🔗	Аналитик	Работа	100 %	300,00 Р/час	0,00 Р/час
2	🔗	Backend-разработчик	Работа	100 %	400,00 Р/час	0,00 Р/час
3	🔗	Frontend-разработчик	Работа	100 %	400,00 Р/час	0,00 Р/час
4	🔗	Тестирующий	Работа	100 %	300,00 Р/час	0,00 Р/час
5	🔗	ЭВМ	Материал		200,00 Р	
6	🔗	Интернет	Материал		80,00 Р	
7	🔗	Электричество	Материал		50,00 Р	

Рисунок 10 – Ресурсы проекта

По расчётной оценке трудозатраты составили 356 часов. С учётом ставок трудовая часть оценивается в 464200 рублей. Дополнительные затраты на оборудование, интернет и электроэнергию оцениваются в дополнительные 11000 рублей.

Планирование показывает, что проект имеет понятную последовательность работ. Выбранная структура позволяет отдельно контролировать готовность требований, базы данных, серверной части, интерфейса и сопроводительной документации.

2 Проектирование веб-приложения для поиска и бронирования лекарственных средств

2.1 Архитектура приложения

Приложение спроектировано как веб-система с разделением на клиентскую часть, серверную часть и базу данных. Такое разделение выбрано потому, что пользовательский интерфейс, бизнес логика и система хранения имеют разные зоны ответственности.

Клиентская часть реализуется как SPA-приложение на React и TypeScript [9]. Она содержит страницы каталога лекарств, аптек, бронирований, авторизации и административной панели. Для взаимодействия с сервером используется общий модуль API-запросов, который добавляет токен автоматизации и обрабатывает ответы.

Серверная часть построена на Node.js, Express и TypeScript. API разделено по предметным областям: авторизация, лекарства, аптеки, бронирование, администрирование и проверка состояния сервера. Доступ к административным и служебным функциям ограничивается ролью пользователя.

Согласно представленным выше требованиям была разработана диаграмма развёртывания, в полной мере отражающая архитектуру приложения. Диаграмма развёртывания представлена на рисунке 11.

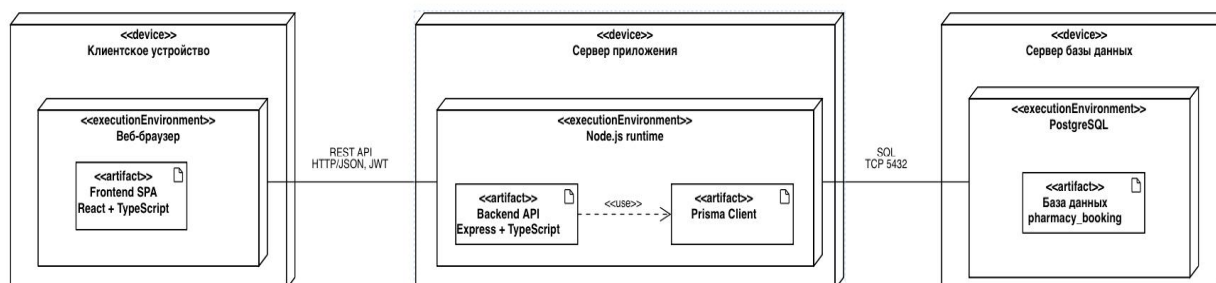


Рисунок 11 – Диаграмма развёртывания

В качестве базы данных применяется PostgreSQL. Для доступа к данным

используется Prisma ORM, которая позволяет описать модели в единой схеме и работать с таблицами через типизированный клиент [10]. Критичные операции, связанные с бронированием и изменением остатков, выполняются в транзакциях.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель отражает основные сущности предметной области без привязки к конкретным типам данных СУБД. Для приложения выделены пользователи, города, аптеки, районы, лекарства, складские позиции и бронирования. Центральной частью модели является связь между лекарством и аптекой через складскую позицию, так как именно в этой записи хранится цена, количество и зарезервированный остаток. Описание всех сущностей базы данных приложения представлено в таблице 2.

Таблица 2 – Описание основных сущностей базы данных

Сущность	Назначение	Основные атрибуты
Пользователь	Хранит учётные записи клиентов, сотрудников и администраторов	Email, пароль, ФИО, телефон, роль
Город	Определяет город расположения аптек	Название города
Район	Уточняет территориальную привязку аптеки в городе	Название района, город
Аптека	Описывает точку аптечной сети	Название, адрес, телефон, график работы
Лекарство	Хранит справочную информацию о препарате	Название, действующее вещество, описание, дозировка, форма, категория
Складская позиция	Связывает лекарство и аптеку, хранит цену и остаток	Лекарство, аптека, цена количество, зарезервированное количество
Бронирование	Фиксирует заявку клиента на получение товара	Код, клиент, складская позиция, количество, статус, срок действия

Диаграмма сущность-связь концептуальной модели – это верхнеуровневый визуальный план [11]. Она показывает сущности предметной области, их свойства и взаимодействие друг с другом без привязки к конкретной реализации в базе данных. Диаграмма сущность-связь разрабатываемой базы данных представлена на рисунке 12.

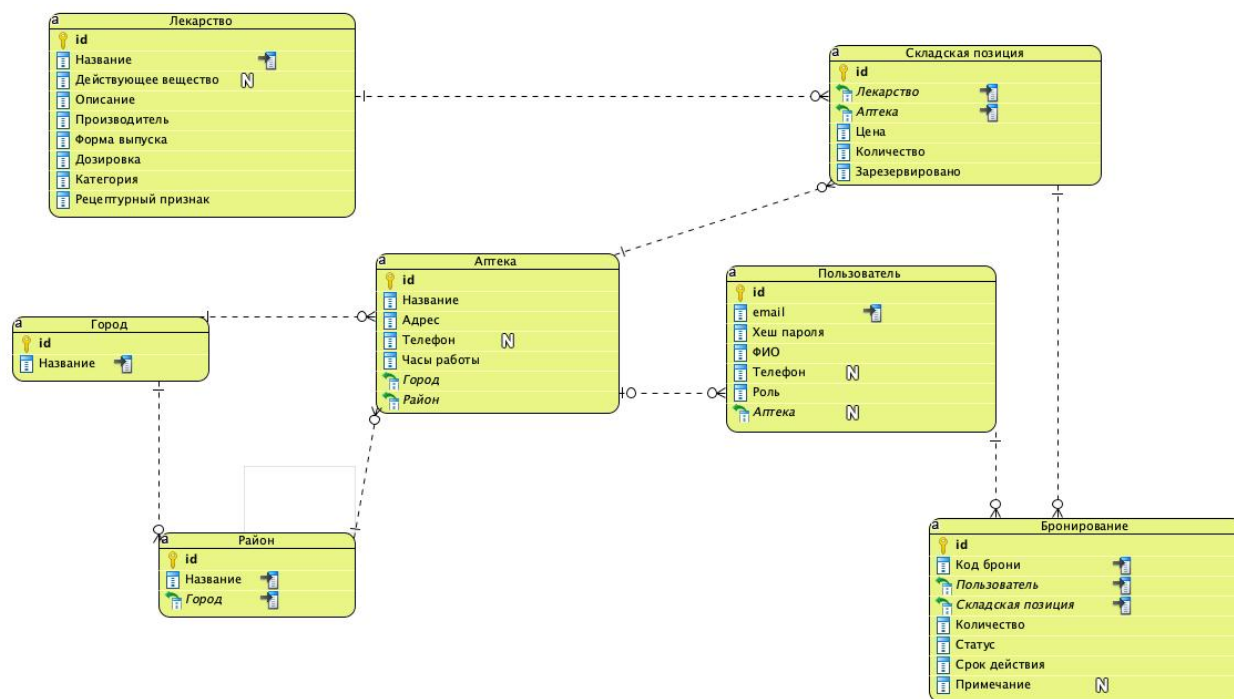


Рисунок 12 – Диаграмма сущность-связь концептуальной модели

Табличное описание связей между сущностями базы данных представлено в таблице 3.

Таблица 3 – Сущности концептуальной модели базы данных

Связь	Кратность	Смысл связи
Город – Район	1:M	Один город содержит несколько районов
Город – Аптека	1:M	В одном городе может быть несколько аптек
Район – Аптека	1:M	В одном районе может быть несколько аптек
Аптека – Пользователь	1:M	К аптеке могут быть привязаны сотрудники
Аптека – Складская позиция	1:M	В аптеке хранится много лекарственных позиций

Продолжение таблицы 3

Связь	Кратность	Смысл связи
Лекарство – Складская позиция	1:M	Одно лекарство может быть доступно в разных аптеках
Пользователь – Бронирование	1:M	Клиент может создать несколько бронирований
Складская позиция – Бронирование	1:M	По одной позиции может быть несколько заявок

Концептуальная модель показывает, что бронирование не хранит лекарство и аптеку напрямую. Оно связано со складской позицией, а через неё получает сведения о препарате, цене и конкретной аптеке. Это уменьшает дублирование и позволяет корректно пересчитать остатки при изменении статуса заявки.

2.2.2 Логическая модель базы данных

Логическая модель данных – это детальная схема, которая систематизирует элементы данных, задаёт связи между ними и устанавливает бизнес-правила. Она является независимым от конкретной СУБД промежуточным этапом между абстрактной концептуальной моделью и физической реализацией [12].

При построении логической модели сущности предметной области преобразуются в логические связи. Для каждой сущности определяется набор атрибутов, необходимых для выполнения пользовательских сценариев. Например, для пользователя должны храниться сведения, позволяющие выполнить вход и определить его роль, а для бронирования – сведения о выбранном товаре, количестве, статусе и сроке действия заявки.

Особое значение в логической модели имеет связь лекарственного средства с аптекой. Один и тот же препарат может находиться в разных аптеках по разной цене и иметь разное количество на остатке. Поэтому между лекарством и аптекой выделяется отдельная логическая сущность, описывающая наличие конкретного препарата в конкретной аптеке. Через неё

бронирование связывается не с абстрактным препаратом, а с реальным аптечным предложением.

На основе концептуальной модели была построена логическая модель данных, включающая основные сущности предметной области и связи между ними. Такая модель позволяет показать, какие данные участвуют в поиске препарата, выборе аптеки и оформлении брони. Диаграмма сущность-связь логической модели представлена на рисунке 13.

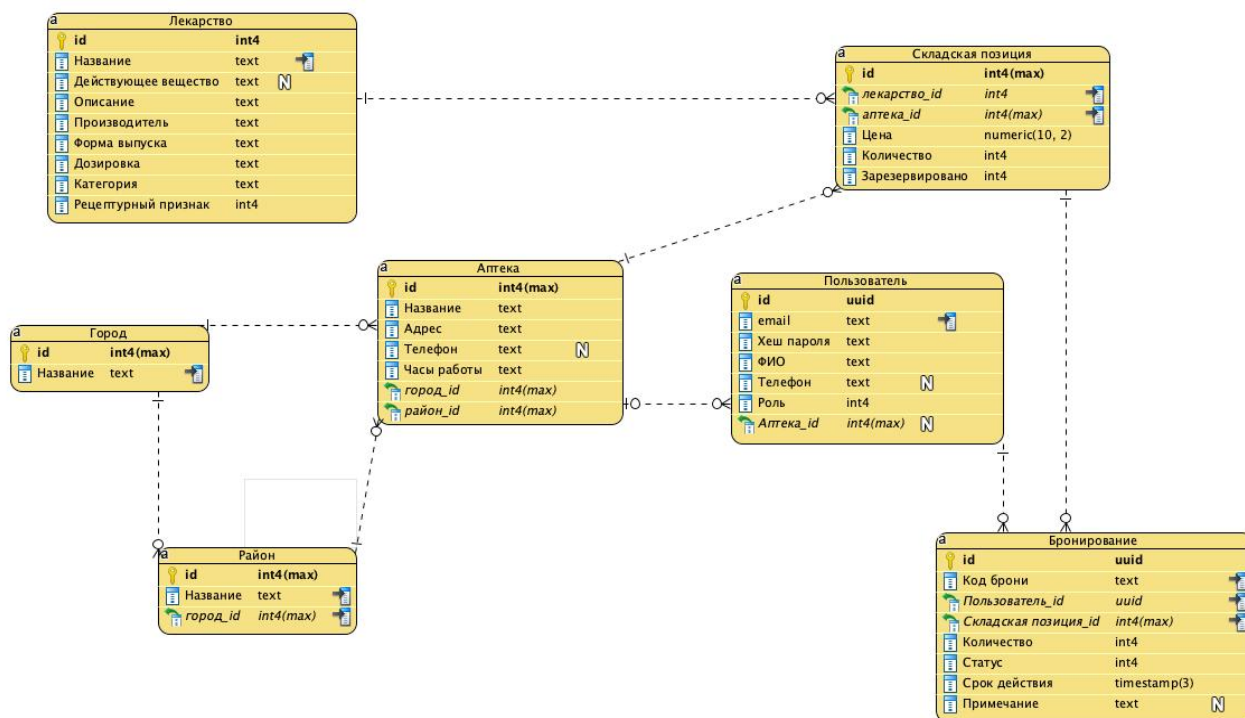


Рисунок 13 – Диаграмма сущность-связь логической модели

Разработанная логическая модель задаёт основу для построения физической структуры базы данных и реализации программной части приложения. Благодаря выделению сущностей, их атрибутов и связей, становится понятно, какие данные должны храниться в системе, как они связаны между собой и какие операции необходимо поддерживать при разработке поиска, отображения аптечных предложений и оформления бронирований.

2.3 Проектирование программных компонентов

Диаграмма программных компонентов – это структурная схема из семейства UML, которая используется для наглядного представления

структуры приложения, основных модулей системы и связей между ними. С её помощью можно показать, какие части отвечают за пользовательский интерфейс, обработку запросов, проверку доступа и работу с данными [13].

Диаграмма программных компонентов, спроектированная в соответствии с архитектурой приложения, представлена на рисунке 14.

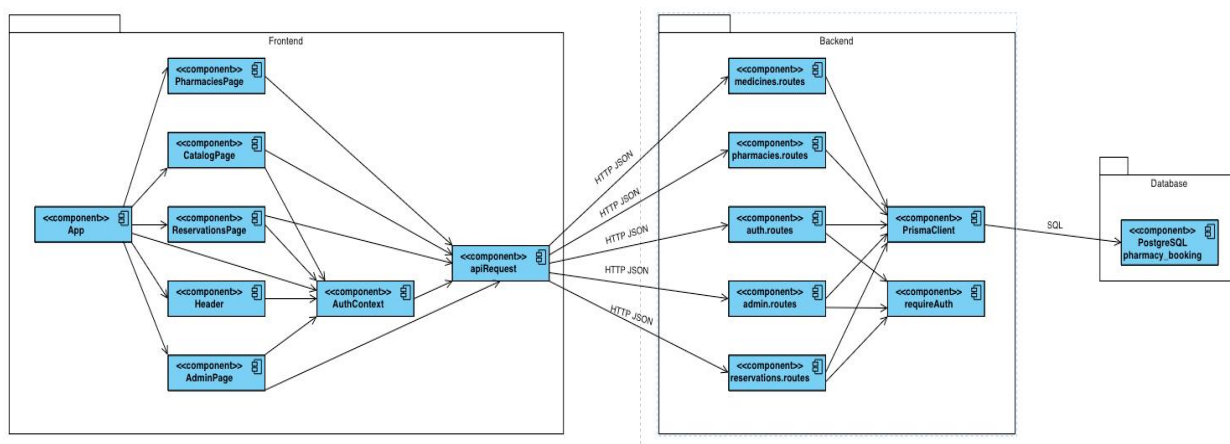


Рисунок 14 – Диаграмма программных компонентов

Основные программные компоненты приложения и их роль в работе системы представлены в таблице 4.

Таблица 4 – Основные компоненты приложения

Компонент	Назначение компонента
App	Определяет структуру SPA и подключает страницы
Header	Отображает навигацию и состояние пользователя
AuthContext	Хранит пользователя, токен и операции входа/выхода
CapalogPage	Показывает лекарства, фильтры и действие бронирования
PharmaciesPage	Показывает список аптек и сведения об остатках
ReservationsPage	Отображает бронирования клиента, сотрудника или администратора
AdminPage	Предоставляет управление справочниками и пользователями
apiRequest	Выполняет HTTP запросы к backend и добавляет токен

Продолжение таблицы 4

Компонент	Назначение компонента
auth.routes	Реализует реализацию, вход и получение текущего пользователя
medicines.routes	Возвращает каталог лекарств и предложения аптек
pharmacies.routes	Возвращает аптеки и связанные данные
reservations.routes	Создаёт и обрабатывает бронирования
admin.routes	Управляет пользователями, лекарствами, аптеками и остатками
requireAuth	Проверяет JWT-токен и добавляет пользователя в запрос
PrismaClient	Выполняет операции чтения и изменения данных

Построенная диаграмма фиксирует состав приложения на уровне основных модулей и показывает, как они взаимодействуют между собой в пользовательских сценариях. Это помогает согласовать структуру клиентской и серверной частей перед переходом к реализации.

2.4 Проектирование структуры пользовательского интерфейса

Пользовательский интерфейс проектируется как рабочая веб-система, а не как рекламная страница. Первый экран приложения должен сразу вести пользователя к каталогу лекарств, поиску и фильтрам. Поэтому в структуре интерфейса выделены основные разделы: каталог, бронирования, аптеки, вход или регистрация, админ-панель для пользователя с ролью администратора.

Главный сценарий клиента состоит из нескольких шагов: открыть каталог, найти препарат, выбрать аптеку, указать количество и создать бронь. Для этого на странице каталога нужны строка поиска, необходимые фильтры, а так же карточки лекарств с понятной и удобной для пользователя информацией [14]. Действие бронирования должно отображаться только там, где это уместно, и сопровождаться сообщением об успехе или ошибке.

Страница аптек нужна для просмотра точек сети. В карточке аптеки должны быть название, адрес, район, телефон и график работы. Это помогает

пользователю выбрать наиболее удобную аптеку.

Страница бронирования имеет разное поведение для ролей. Клиент видит собственные заявки и может отменять активные. Сотрудник аптеки видит заявки своей аптеки и может подтверждать, отменять или завершать их. Администратор получает более широкий доступ.

Задача административной панели – постоянное управление данными. В ней нужны блок статистики, управление пользователями, формы добавления лекарств, аптек и складских остатков. При этом формы не должны смешиваться с пользовательским каталогом, что бы разные роли не путали рабочие сценарии.

Цветовая схема выбрана спокойной – светлый фон, мягкие зелёные оттенки и аккуратные карточки. Такой стиль не перегружает экран и не путает пользователя излишне нагруженным интерфейсом.

3 Реализация веб-приложения для поиска и бронирования лекарственных средств в аптечной сети

3.1 Реализация серверной части приложения

Серверная часть приложения реализована на платформе Node.js, с использованием фреймворка Express и языка TypeScript [15]. Она отвечает за обработку HTTP-запросов, проверку входных данных, авторизацию пользователей, выполнение бизнес-правил и обращение к базе данных через Prisma Client. Такой подход позволяет отделить пользовательский интерфейс от логики работы с данными и бронированиями.

Маршруты API разделены по назначению. Файл `auth.routes.ts` отвечает за регистрацию, вход и получение сведений о текущем пользователе. при регистрации пароль сохраняется в виде хеша, а при успешном входе сервер формирует JWT-токен.

Работа с каталогом лекарств реализована в маршрутах `medicines.routes.ts`. Сервер получает параметры поиска и фильтрации, после чего возвращает препараты вместе с аптечными предложениями. В ответ включаются сведения о цене, аптеке, районе, городе и доступном количестве. Это позволяет клиентской части отображать подготовленные данные без самостоятельной сборки сложных связей.

Маршруты `pharmacies.routes.ts` предназначены для получения списка аптек и информации о связанных с ними остатках. Эти данные используются на страницах аптек и каталога. Административные операции вынесены в `admin.routes.ts`, через них выполняется управление пользователями, лекарствами, аптеками и складскими позициями.

Ключевой частью серверной логики является работа с заявками бронирования. В файле `reservations.routes.ts` реализовано создание брони, получение списка заявок и изменение статуса. При создании бронирования маршрут `POST/api/reservations` получает идентификатор складской позиции и количество товара, а так же проверяет корректность данных, находит остаток и

рассчитывает доступное количество. Если товара недостаточно, сервер возвращает ошибку. Если операция допустима, в транзакции увеличивается зарезервированное количество и создаётся запись бронирования со статусом ожидания.

Изменение статуса брони так же выполняется на сервере. Клиент может отменить только свою активную заявку, сотрудник работает только с бронированиями своей аптеки, а администратор имеет доступ ко всем заявкам. Для завершённых и отменённых бронирований повторное изменение статуса запрещено. При отмене резерв освобождается, а при завершении заявки уменьшается и зарезервированное количество, и общий остаток товара.

Ключевым серверным сценарием является создание бронирования, так как в нём одновременно выполняется проверка пользователя, проверка доступного остатка и изменение данных о складской позиции. Последовательность обработки такого запроса представлена на рисунке 15.

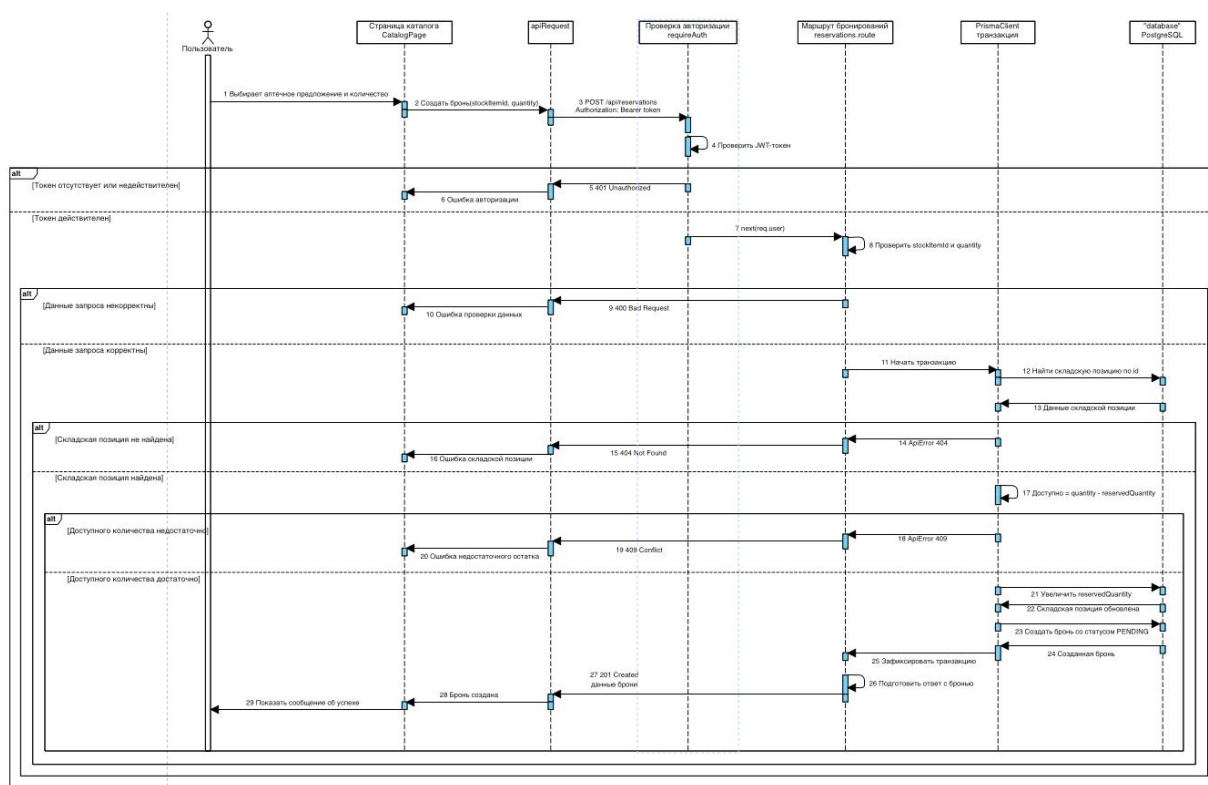


Рисунок 15 – Диаграмма последовательности запроса на создание бронирования

Реализованная серверная часть обеспечивает связь между

пользовательским интерфейсом и базой данных. Она принимает запросы от клиентской части, проверяет права пользователя, выполняет операции с лекарствами, аптеками и бронированиями. Разделение маршрутов по предметным областям позволяет изолировать отдельные группы функций и упростить дальнейшее изменение серверной логики.

3.2 Реализация структуры хранения данных

Физическая модель базы данных представляет собой структуру хранения данных, подготовленную для реализации в выбранной СУБД. На данном этапе логическая модель преобразуется в конкретные таблицы, поля, типы данных, ключи, а так же ограничения, необходимые для корректной работы приложения [17].

В проекте используется база данных PostgreSQL. Структура таблиц описана в Prisma Schema, после чего может быть развёрнута в базу данных средствами Prisma. Физическая модель включает сущности пользователей, городов, районов, аптек, лекарственных средств, складских позиций и бронирований. разработанная физическая модель представлена на рисунке 16.

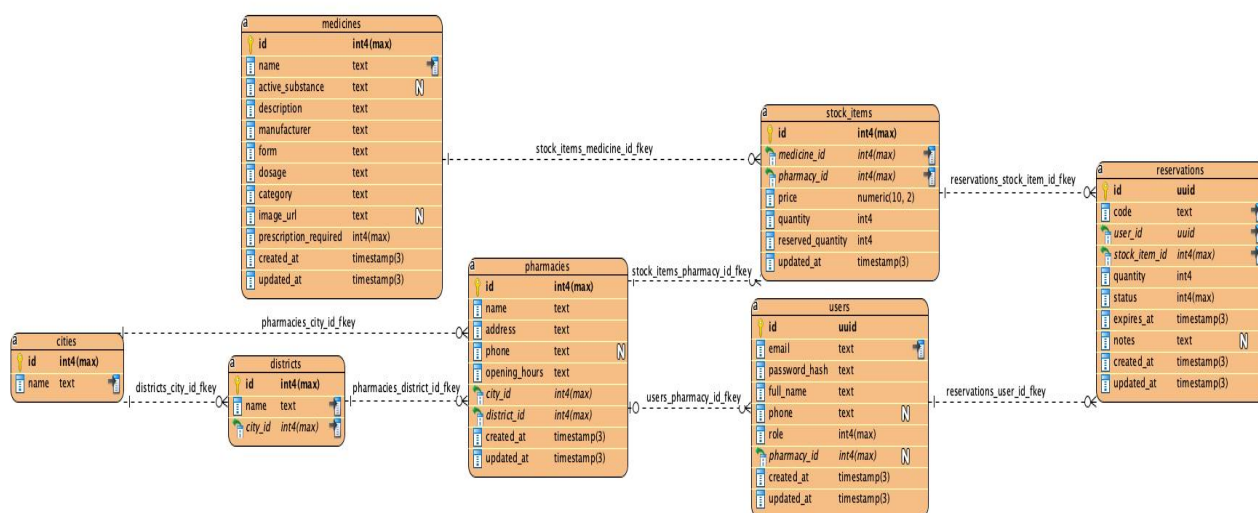


Рисунок 16 – Физическая модель базы данных

Структура таблицы пользователей приведена в таблице 5.

Таблица 5 – Таблица пользователей

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор пользователя	id	uuid	PK	36	Да
Адрес электронной почты	email	text	–	255	Да
Хеш пароля	password_hash	text	–	255	Да
ФИО пользователя	full_name	text	–	150	Да
Номер телефона	phone	text	–	20	Нет
Роль	role	enum Role	–	20	Да
Идентификатор аптеки сотрудника	pharmacy_id	integer	FK	4	Нет
Дата создания записи	created_at	timestamp	–	8	Да
Дата обновления записи	updated_at	timestamp	–	8	Да

Структура таблицы городов приведена в таблице 6.

Таблица 6 – Таблица городов

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор города	id	integer	PK	4	Да
Название города	name	text	–	100	Да

Структура таблицы районов приведена в таблице 7.

Таблица 7 – Таблица районов

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор района	id	integer	PK	4	Да
Название района	name	text	–	100	Да
Идентификатор города	city_id	integer	FK	4	Да

Структура таблицы аптек приведена в таблице 8.

Таблица 8 – Таблица аптек

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор аптеки	id	integer	PK	4	Да
название аптеки	name	text	–	150	Да
Адрес аптеки	adress	text	–	255	Да
Контактный телефон	phone	text	–	20	Нет
График работы	opening_hours	text	–	100	Да
Идентификатор города	city_id	integer	FK	4	Да
Идентификатор района	district_id	integer	FK	4	Да
Дата создания записи	created_at	timestamp	–	8	Да
Дата обновления записи	updated_at	timestamp	–	8	Да

Структура таблицы лекарственных средств приведена в таблице 9.

Таблица 9 – Таблица лекарственных средств

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор лекарства	id	integer	PK	4	Да
Название лекарства	name	text	–	200	Да
Действующее вещество	active_substance	text	–	200	Нет
Описание препарата	description	text	–	1000	Да
Производитель	manufacturer	text	–	150	Да
Форма выпуска	form	text	–	100	Да
Дозировка	dosage	text	–	50	Да
Категория препарата	category	text	–	100	Да

Продолжение таблицы 9

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Ссылка на изображение	image_url	text	–	500	Нет
Признак рецептурного препарата	prescription_required	boolean	–	1	Да
Дата создания записи	created_at	timestamp	–	8	Да
Дата обновления записи	updated_at	timestamp	–	8	Да

Структура таблицы аптечных предложений приведена в таблице 10.

Таблица 10 – Таблица аптечных предложений

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор складской позиции	id	integer	PK	4	Да
Идентификатор лекарства	medicine_id	integer	FK	4	Да
Идентификатор аптеки	pharmacy_id	integer	FK	4	Да
Цена препарата в аптеке	price	decimal	–	10,2	Да
Общее количество товара	quantity	integer	–	4	Да
Зарезервированное количество	reserved_quantity	integer	–	4	Да
Дата обновления остатка	uploaded_at	timestamp	–	8	Да

Структура таблицы бронирований приведена в таблице 11.

Таблица 11 – Таблица бронирований

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор бронирования	id	uuid	PK	36	Да
Код бронирования	code	text	–	20	Да

Продолжение таблицы 11

Содержание	Наименование	Тип	Ключ	Размер	Обязательность
Идентификатор пользователя	user_id	uuid	FK	36	Да
Идентификатор складской позиции	stock_item_id	integer	FK	4	Да
Количество товара в заявке	quantity	integer	–	4	Да
Статус бронирования	status	enum ReservationStatus	–	20	Да
Срок действия бронирования	expires_at	timestamp	–	8	Да
Примечание к бронированию	notes	text	–	500	Нет
Дата создания записи	created_at	timestamp	–	8	Да
Дата обновления записи	updated_at	timestamp	–	8	Да
Идентификатор пользователя	user_id	uuid	FK	36	Да

Реализованная физическая модель базы данных соответствует структуре приложения и поддерживает основные операции, а использование ключей и обязательных полей позволяет сохранить связность данных и снизить вероятность некорректных записей.

3.3 Реализация пользовательского интерфейса

Интерфейс приложения спроектирован как набор страниц React. Основная навигация расположена в шапке и включает в себя переходы в каталог, списки аптек и заявок, форму входа, а так же переход на административную панель при наличии соответствующей роли.

Главная рабочая страница – каталог лекарств. На ней размещены строка поиска, фильтры и карточки препаратов. В карточке отображаются названия, форма выпуска, дозировка, производитель, признак рецептурности и доступные предложения аптек. Пользователь сразу получает всю нужную информацию. Страница каталога лекарств представлена на рисунке 17.

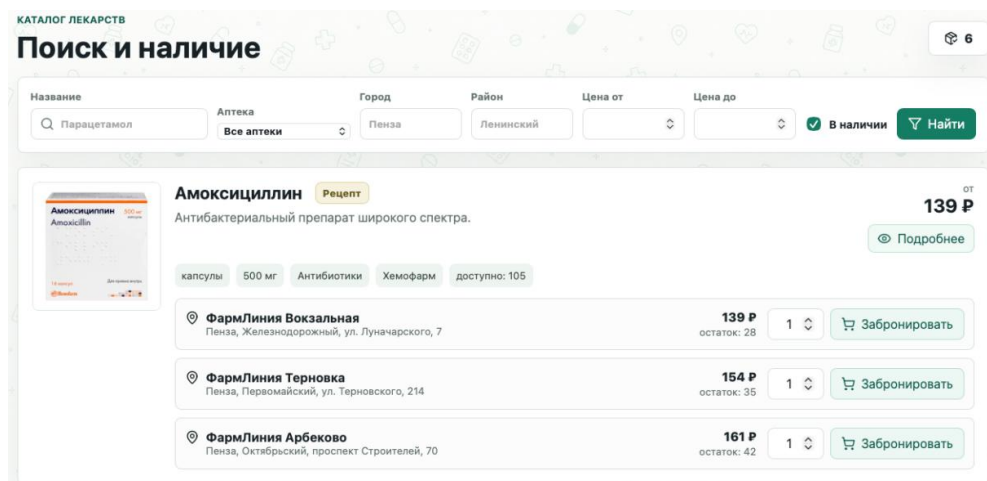


Рисунок 17 – Скриншот страницы каталога лекарств

При выборе препарата пользователь переходит к действию бронирования. Интерфейс показывает выбранную аптеку, цену, доступное количество и поле ввода количества. Интерфейс бронирования представлен на рисунке 18.

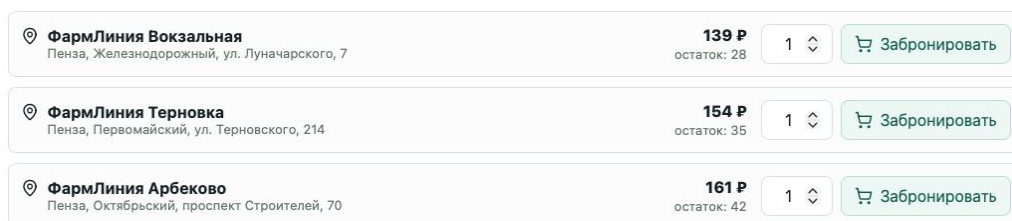


Рисунок 18 – Скриншот интерфейса бронирования

Если пользователь не авторизован, приложение открывает окно входа в аккаунт, так как бронь должна быть связана с клиентом и его списком заявок. Окно входа представлено на рисунке 19.

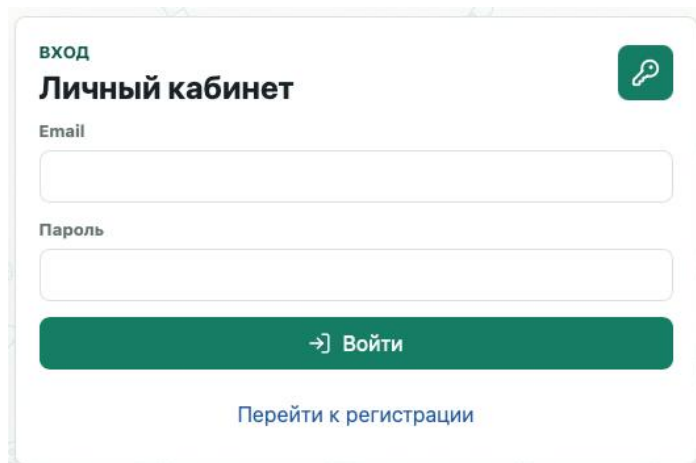


Рисунок 19 – Скриншот окна входа

Страница аптек предназначена для просмотра точек сети. Для каждой аптеки отображается адрес, название, район, телефон, график работы и связанные складские позиции. Эта страница полезна, когда пользователь сначала выбирает удобную аптеку, а уже потом смотрит доступные препараты. Страница аптек представлена на рисунке 20.

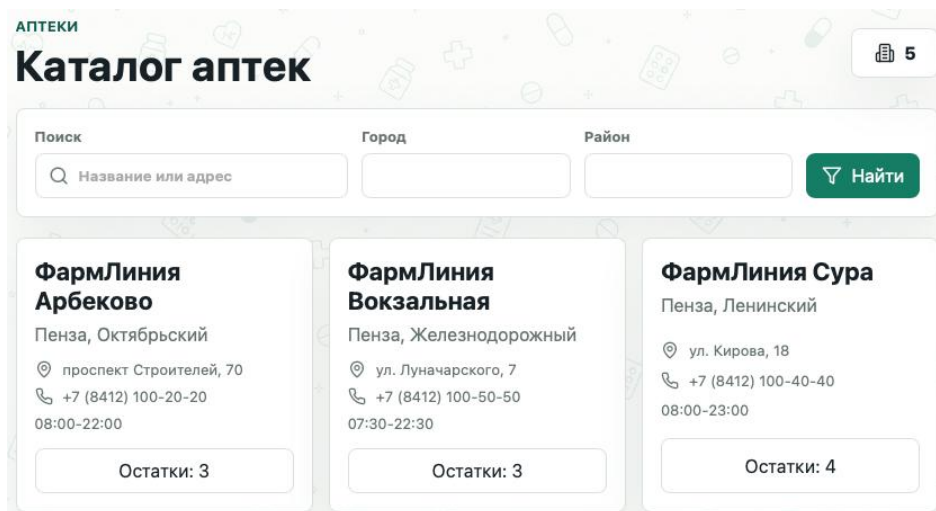


Рисунок 20 – Скриншот страницы каталога аптек

Регистрация выполнена отдельной формой. Пользователь вводит ФИО, email, телефон и пароль. После успешного создания учётной записи можно перейти к входу и продолжить работу с бронированиями. Страница регистрации представлена на рисунке 21.

РЕГИСТРАЦИЯ

Новый клиент

ФИО

Телефон

Email

Пароль

Создать аккаунт

Уже есть аккаунт

Рисунок 21 – Скриншот страницы регистрации

После создания бронирования пользователь получает сообщение об успешной операции. Страница бронирований клиента показывает список его заявок. Для каждой записи выводятся препарат, адрес аптеки, количество, статус и другие параметры. Активную бронь можно отменить, если пользователь передумал забирать товар. Страница списка бронирований представлена на рисунке 22.



Рисунок 22 – Скриншот страницы бронирований

Административная панель начинается со статистики. Она отображает количество пользователей, аптек, лекарств, бронирований и ожидающих заявок. Страница статистики изображена на рисунке 23.

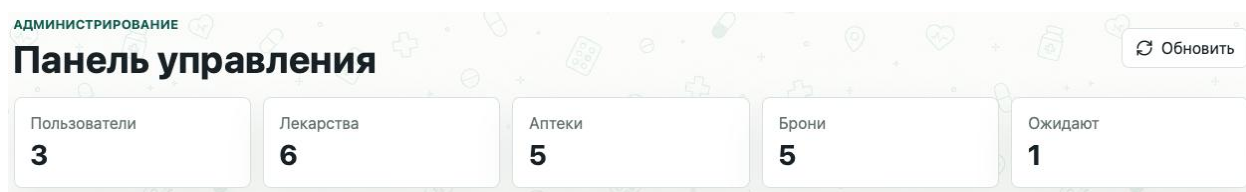


Рисунок 23 – Скриншот раздела статистики

В разделе пользователей администратор может просматривать учётные записи, менять роль и привязывать сотрудника к аптеке. Это позволяет настраивать доступ сотрудников и клиентов без прямого обращения к базе данных. Раздел пользователей изображён на рисунке 24.

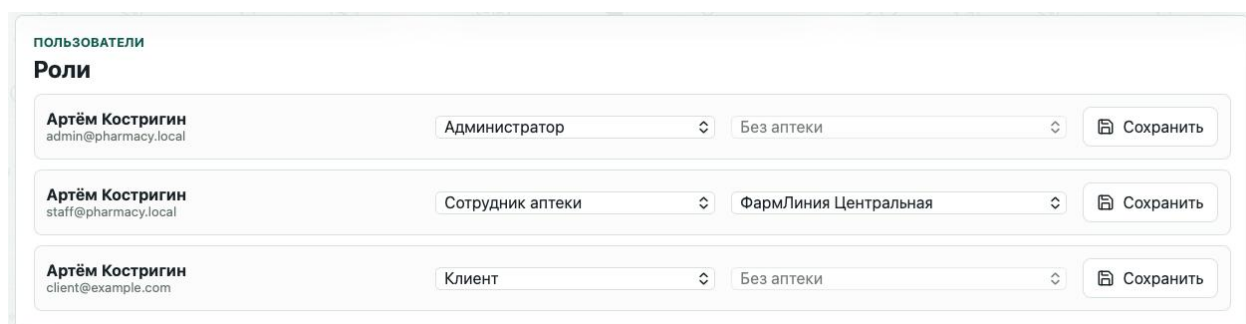
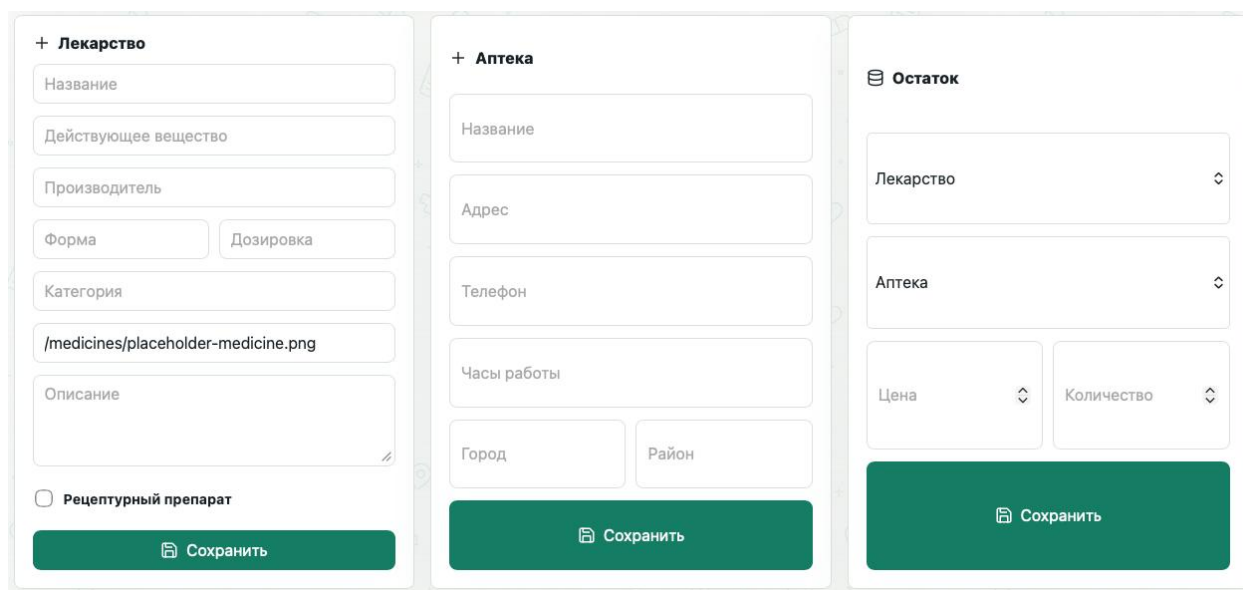


Рисунок 24 – Скриншот раздела пользователей

Форма добавления лекарства содержит название, действующее вещество, а так же другие необходимые характеристики препарата. После сохранения лекарство становится доступно для складских остатков. Форма добавления лекарства представлена на рисунке 25.

Форма добавления аптеки используется для расширения аптечной сети. Администратор указывает характеристики новой аптеки, при необходимости создавая новый город и район. Форма добавления аптеки представлена на рисунке 25.

Отдельная форма отвечает за складские остатки. Администратор выбирает препарат и аптеку, задаёт цену и количество. Если позиция уже существует, система обновляет её, а не создаёт дубль. Эта форма представлена на рисунке 25.



The image displays three side-by-side web forms for a pharmacy management system. Each form has a green 'Сохранить' (Save) button at the bottom.

- Form 1: + Лекарство (Add Medicine)**
 - Fields: Название (Name), Действующее вещество (Active ingredient), Производитель (Manufacturer), Форма (Form), Дозировка (Dosage), Категория (Category), Image upload (/medicines/placeholder-medicine.png), and Описание (Description).
 - Checkbox: ☐ Рецепттурный препарат (Prescription drug).
- Form 2: + Аптека (Add Pharmacy)**
 - Fields: Название (Name), Адрес (Address), Телефон (Phone), Часы работы (Hours of operation), Город (City), and Район (District).
- Form 3: Остаток (Inventory)**
 - Dropdowns: Лекарство (Medicine) and Аптека (Pharmacy).
 - Input fields: Цена (Price) and Количество (Quantity).

Рисунок 25 – Скриншот форм добавления лекарств, добавления аптек и редактирования остатков

Реализованный интерфейс покрывает все роли системы и сохраняет единый визуальный стиль, а так же содержит понятные формы. Основное внимание остаётся на поиске лекарства, а не на дополнительных коммерческих разделах.

4 Тестирование веб-приложения для поиска и бронирования лекарств в аптечной сети

4.1 Тестирование методом «белого ящика»

После реализации приложения была выполнена проверка серверной логики методом «белого ящика». Данный метод был выбран потому, что для маршрута создания бронирования необходимо учитывать не только внешний результат запроса, но и внутреннюю последовательность проверок, выполняемых в коде [18].

Тестирование методом «белого ящика» предполагает, что структура программы известна проверяющему. В данном случае тестовые данные подбираются так, что бы пройти значимые операторы, условия и маршруты выполнения. такой подход позволяет обнаружить ошибки, которые могут не проявиться при обычной проверке интерфейса, например некорректную обработку граничного количества товара или преждевременное изменение количества товара в базе данных.

Для тестирования был выбран алгоритм создания бронирования лекарственного препарата. Он получает из тела запроса `stockItemId` и `quantity`, проверяет корректность этих значений, ищет складскую позицию в базе данных, рассчитывает доступный остаток, после чего либо возвращает ошибку, либо создаёт бронирование и увеличивает зарезервированное количество товара. Текст алгоритма приведён в листинге 1.

Листинг 1 – Исходный код алгоритма создания бронирования

```
router.post(  
  "/",  
  requireAuth,  
  asyncHandler(async (req, res) => {  
    const stockItemId = Number(req.body.stockItemId);  
    const quantity = Number(req.body.quantity ?? 1);  
  
    if (!Number.isInteger(stockItemId)  
    || !Number.isInteger(quantity) || quantity <= 0) {  
      throw new ApiError(400, "Valid stock item and quantity are  
required");  
    }  
  })  
);
```

```

    const reservation = await prisma.$transaction(async (tx) => {
      const stockItem = await tx.stockItem.findUnique({ where:
{ id: stockItemId } });

      if (!stockItem) {
        throw new ApiError(404, "Stock item was not found");
      }

      const availableQuantity = stockItem.quantity -
stockItem.reservedQuantity;
      if (availableQuantity < quantity) {
        throw new ApiError(409, "Not enough medicine quantity
available");
      }

      await tx.stockItem.update({
        where: { id: stockItemId },
        data: { reservedQuantity: { increment: quantity } }
      });

      return tx.reservation.create({
        data: {
          code: reservationCode(),
          userId: req.user!.id,
          stockItemId,
          quantity,
          status: ReservationStatus.PENDING,
          expiresAt: new Date(Date.now() + 24 * 60 * 60 * 1000)
        },
        include: reservationInclude
      });
    });

    res.status(201).json({ data: reservationView(reservation) });
  })
);

```

Для дальнейшего анализа операторы алгоритма обозначены буквами. Оператор a получает stockItemId, оператор b получает quantity, оператор c выполняет проверку входных данных, оператор d формирует ошибку 400. Оператор e запускает транзакцию, оператор f выполняет поиск складской позиции, g проверяет наличие найденной записи, h формирует ошибку 404. Оператор i рассчитывает доступное количество, j проверяет достаточность остатка, k формирует ошибку 409, l увеличивает reservedQuantity, m создаёт бронирование, а n возвращает успешный ответ клиенту.

Схема переходов алгоритма показана на рисунке 26.

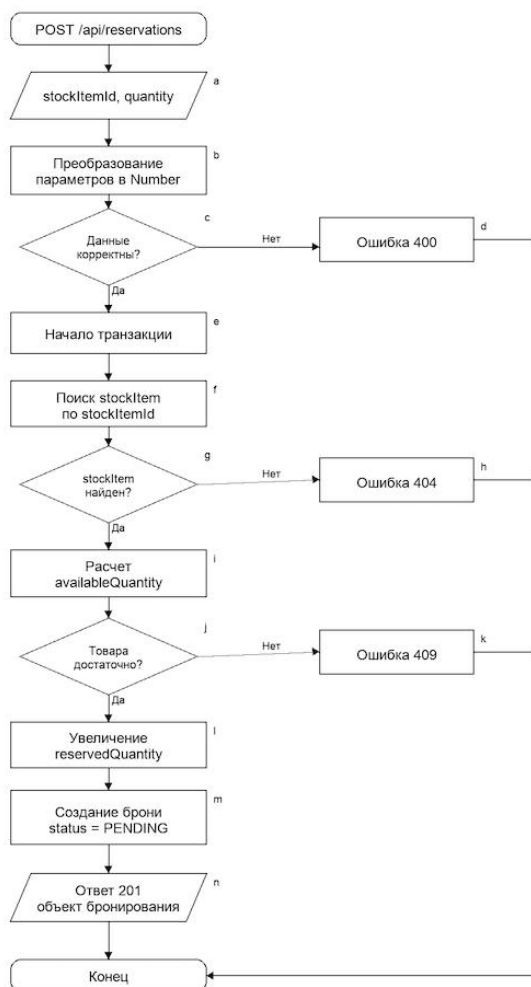


Рисунок 26 – Граф алгоритма создания бронирования

Граф показывает основные этапы выполнения метода создания бронирования и возможные варианты завершения алгоритма. На его основе были определены тестовые случаи для проверки операторов, решений и комбинаций условий.

4.1.1 Покрытие операторов

При проверке покрытия операторов необходимо добиться, что бы каждый значимый оператор выбранного фрагмента был выполнен хотя бы один раз. Тестовые данные подбирались так, что бы пройти как ошибочные ветви рассматриваемого метода, так и успешное создание заявки. Итог проверки операторов приведён в таблице 12.

Таблица 12 – Результаты проверки операторов

№	Проверяемые решения	Данные запроса	Ожидаемый результат	Фактический результат
1	a, b, c, d	stockItemId = «abc»; quantity = 1	HTTP 200; сообщение о некорректных данных; бронь не создана	Тест пройден
2	a, b, c, e, f, g, h	stockItemId = 999999; quantity = 1	HTTP 404; складская позиция не найдена	Тест пройден
3	a, b, c, e, f, g, i, j, k	stockItemId = 1; quantity = 1000	HTTP 409; недостаточно доступного количества	Тест пройден
4	a, b, c, e, f, g, I, j, l, m, n	stockItemId = 1; quantity = 1	HTTP 201; создана бронь со статусом «Ожидание»	Тест пройден

В результате выполнения покрытия были проверены все значимые операторы алгоритма. Ошибочные сценарии позволяют выполнить операторы формирования ответов 400, 404 и 409, а успешный сценарий проходит через обновление резерва, создание бронирования и возврат результата пользователю.

Итоговое покрытие операторов для рассматриваемого метода составляет 100%.

4.1.2 Покрытие решений

Для покрытия решений нужно проверить, что каждое условие выбранного алгоритма срабатывает в обе стороны, а именно приводит к выполнению ветви ошибки и пропускает выполнение дальше при корректных данных. В методе создания бронирования выделено 3 основные развилки: корректность ввода данных, наличие складской позиции и достаточное количество товара.

Проверка решений представлена в таблице 13.

Таблица 13 – Результаты проверки решений

№	Проверяемые решения	Данные запроса	Ожидаемый результат	Фактический результат
1	c – да	stockItemId = «abc»; quantity = 1	HTTP 400; выполнение завершается транзакции до	Тест пройден
2	c – нет, g – да	stockItemId = 999999; quantity = 1	HTTP 404; складская позиция отсутствует	Тест пройден
3	c – нет, g – нет, j – да	stockItemId = 1; quantity = 1000	HTTP 409; товара недостаточно	Тест пройден
4	c – нет, g – нет, j – нет	stockItemId = 1; quantity = 1	HTTP 201; бронирование создано	Тест пройден

В результате прохождения данного этапа тестирования были проверены все ветви условий выбранного метода. Первая строка таблицы подтверждает выполнение истинной ветви проверки входных данных. Последующие строки проходят ложную ветвь первого условия и проверяют остальные решения алгоритма.

Исходя из этого, каждое решение было проверено в необходимых вариантах, а покрытие решений составляет 100%.

4.1.3 Комбинаторное покрытие условий

Комбинаторное покрытие используется для проверки сочетаний условий, от которых зависит дальнейшая работа программы. В рассматриваемом методе часть сочетаний невозможно выполнить, так как алгоритм завершается сразу после обнаружения ошибки. Например, если входные данные некорректны, поиск складской позиции уже не выполняется.

Для проверки были выделены условия, которые определяют дальнейший маршрут выполнения.

Набор подходящих для тестирования сочетаний и результаты тестирования представлены в таблице 14.

Таблица 14 – Результат выполнения комбинаторного покрытия условий

№	Проверяемые решения	Данные запроса	Ожидаемый результат	Фактический результат
1	c – да; stockItemId = «abc»	stockItemId = «abc»; quantity = 1	HTTP 400	Тест пройден
2	c – да; quantity не является числом	stockItemId = 999999; quantity = «two»	HTTP 400	Тест пройден
3	c – да; quantity ≤ 0	stockItemId = 1; quantity = 0	HTTP 400	Тест пройден
4	c – нет, g – да	stockItemId = 9999; quantity = 1	HTTP 404	Тест пройден
5	c – нет, g – нет, j – да	stockItemId = 1; quantity = 1000	HTTP 409	Тест пройден
6	c – нет, g – нет, j – нет	stockItemId = 1; quantity = 1	HTTP 201; бронь создана	Тест пройден

В результате выполнения тестирования были проверены все достижимые комбинации условий выбранного метода. Отдельно проверены разные причины некорректности ввода данных. Это важно, так как все эти ситуации приводят к одному HTTP-коду, но возникают по разным причинам.

Проведённая проверка показала, что функция корректно работает при разных последовательностях выполнения операторов и условий. Ошибочные сценарии не изменяют данные, а успешный сценарий приводит к созданию бронирования и увеличению зарезервированного остатка.

4.2 Модульное тестирование

Модульное тестирование направлено на проверку небольших самостоятельных частей программы. Такой подход применяется к логике создания бронирования, так как она содержит расчёт доступного количества товара и несколько вариантов завершения работы [19].

Основная цель модульного тестирования заключается в том, чтобы убедиться, что функция корректно реагирует на разные входные данные и не создаёт бронирование в ошибочных ситуациях. В отличие от пользовательской проверки через интерфейс, модульный тест позволяет проверить отдельный фрагмент логики без ручного повторения действий в браузере.

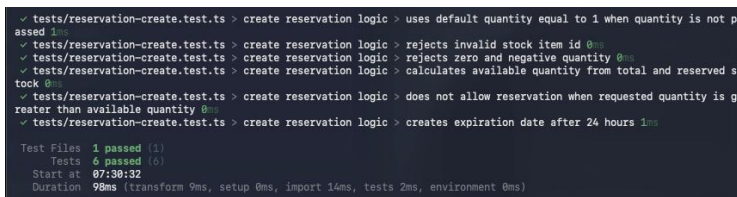
Для тестирования метода создания бронирования были составлены следующие тесты:

- если параметр `quantity` не передан, метод должен использовать значение по умолчанию, равное 1;
- некорректный `stockItemId` должен приводить к ошибке 400;
- нечисловое значение `quantity` также должно отклоняться с ошибкой 400;
- нулевое или отрицательное количество не должно приводить к созданию бронирования;
- при отсутствии складской позиции в базе данных сервер должен возвращать ошибку 404;
- недостаточное доступное количество товара должно завершать запрос ошибкой 409;
- успешное бронирование должно увеличивать значение `reservedQuantity` на указанное количество;
- созданная запись должна получать статус «В процессе» и срок действия на 24 часа;
- запрос без JWT-токена должен завершаться ошибкой авторизации до запуска основного алгоритма создания брони.

Дополнительно проверялась расчётная часть алгоритма. Доступное количество определяется как разность общего количества товара и уже зарегистрированного количества. Если полученное значение меньше количества, которое запрашивает пользователь, бронирование должно быть отключено.

Данная проверка является важной, потому что именно она предотвращает

резервирование товара сверх фактического остатка. Результат проверки приведён на рисунке 27.



```

✓ tests/reservation-create.test.ts > create reservation logic > uses default quantity equal to 1 when quantity is not passed 1ms
✓ tests/reservation-create.test.ts > create reservation logic > rejects invalid stock item id 0ms
✓ tests/reservation-create.test.ts > create reservation logic > rejects zero and negative quantity 0ms
✓ tests/reservation-create.test.ts > create reservation logic > calculates available quantity from total and reserved stock 0ms
✓ tests/reservation-create.test.ts > create reservation logic > does not allow reservation when requested quantity is greater than available quantity 0ms
✓ tests/reservation-create.test.ts > create reservation logic > creates expiration date after 24 hours 1ms

Test Files  1 passed (1)
Tests       6 passed (6)
Start at    07:30:32
Duration    98ms (transform 9ms, setup 0ms, import 14ms, tests 2ms, environment 0ms)
  
```

Рисунок 27 – Результат модульного тестирования алгоритма

По итогам модульного тестирования подтверждена корректность работы метода создания бронирования. Проверенные сценарии показывают, что приложение правильно обрабатывает как успешное создание брони, так и основные ошибочные ситуации.

4.3 Оценка качества кода

После проверки корректности работы метода была выполнена оценка качества программного кода. Для анализа выбран тот же фрагмент, который использовался при тестировании методом «белого ящика». Этот метод содержит не только последовательные операции, но и несколько условий, работу с базой данных и изменение состояния складского остатка. Поэтому по нему будет удобно показать размер кода, объём используемых операторов и операндов, а так же информационную сложность переменных.

4.3.1 Метрика Чепина

Метрика Чепина оценивает информационную сложность программы через переменные, используемые в выбранном фрагменте [20]. Для корректного применения данной метрики необходимо рассмотреть переменные не только по факту их наличия в коде, но и по их влиянию на выполнение алгоритма. Одни переменные передают входные данные, другие изменяют состояние системы, а часть переменных определяет, по какой ветви будет продолжено выполнение метода. Поэтому перед расчётом метрики все переменные были разделены по функциональному назначению и представлены в таблице 15.

Таблица 15 – Распределение метрик по группам Чепина

Группа	Переменные	Количество
P	req.user.id, reservation	2
M	reservedQuantity, status, expiresAt	3
C	stockItemId, quantity, stockItem, availableQuantity	4
T	Отсутствуют	0

Расчёт метрик Чепина выполняется по следующей формуле:

$$Q = 1 \times P + 2 \times M + 3 \times C + 0.5 \times T.$$

Подставляем значения в формулу:

$$Q = 1 \times 2 + 2 \times 3 + 3 \times 4 = 2 + 6 + 12 = 20.$$

Значение метрики Чепина для метода создания бронирования составило 20. Основную часть сложности формируют управляющие переменные, так как именно они определяют дальнейший сценарий выполнения метода.

Полученное значение можно считать допустимым для рассматриваемого метода. Алгоритм выполняет несколько связанных действий, однако его логика остаётся читаемой.

4.3.2 Метрики Холстеда

Метрики Холстеда применяются для оценки сложности программы через операторы и операнды [21]. Операторами считаются конструкции и действия языка программирования, например объявление переменной, условный оператор, вызов функции, арифметическая операция или обращение к методу. Операндами считаются данные, с которыми работают операторы, к примеру значения и свойства объектов.

Результат подсчёта операторов алгоритма представлен в таблице 16.

Таблица 16 – Подсчёт операторов алгоритма

№	Операторы	Число вхождений
1	Объявление маршрута router.post()	1

Продолжение таблицы 16

№	Операторы	Число вхождений
2	Проверка авторизации requireAuth	1
3	Асинхронный обработчик asyncHandler	3
4	Объявление переменных const	5
5	Преобразование данных Number()	2
6	Оператор значения по умолчанию ??	1
7	Условные операторы if	3
8	Проверки корректности и логические операции	7
9	Генерации ошибок throw new ApiError()	3
10	Асинхронное ожидание await	3
11	Транзакция \$transaction()	1
12	Поиск записи findUnique()	1
13	Изменение записи update()	1
14	Создание записи create()	1
15	Арифметические операции	5
16	Формирование ответа res.status().json()	1
	Итого	39

Операнды рассматриваемого метода и число их вхождений приведены в таблице 17.

Таблица 17 – Подсчёта операндов алгоритма

№	Операнды	Число вхождений
1	req.body.stockItemId	1
2	req.body.quantity	1
3	stockItemId	6
4	quantity	7
5	Значения 1 и 0	2
6	Коды ошибок 400, 404, 409	3
7	Текстовые сообщения ошибок	3
8	reservation	2
9	tx	5
10	stockItem	5

Продолжение таблицы 17

№	Операнды	Число вхождений
11	availableQuantity	2
12	stockItem.quantity	1
13	stockItem.reservedQuantity	1
14	reservedQuantity	3
15	ReservationStatus.PENDING	1
16	req.user.id	1
17	Значения срока действия бронирования	7
	Итого	51

На основе полученных данных можно произвести расчёт метрик. Результаты представлены в таблице.

Таблица 18 – Расчёт метрик Холстеда

Значение	Результат
Словарь программы	33
Длина программы	90
Объём программы	454,31
Сложность программы	24
Усилие на понимание	10903,44

По результатам расчёта можно сделать вывод, что объём программы является небольшим для анализируемого фрагмента. Значение сложности можно считать умеренным, так как метод содержит несколько проверок и выполняет транзакционное изменение данных, но при этом не имеет сложной внутренней структуры. Усилие на понимание показывает, что код требует внимательного чтения, однако последовательность действий остаётся понятной для пользователя.

4.3.3 SLOC тестирование

Метрика SLOC используется для определения количества строк исходного кода. Для примера, подсчёт выполнен для фрагмента кода, реализующего алгоритм создания бронирования [22].

Общее количество строк выбранного фрагмента – 45. При этом, пустых

строк – 8, а строк с комментариями – 0. Количество физических строк кода определяется как разность между общим количеством строк, пустыми строками и строками комментариев: $45 - 8 - 0 = 37$.

Следовательно, физический размер анализируемого фрагмента составляет 37 строк кода. Пустые строки составляют приблизительно 18% и используются для разделения смысловых частей метода. Это делает код более читаемым, при этом сам алгоритм остаётся компактным.

Заключение

В ходе выполнения выпускной квалификационной работы разработано веб-приложения для поиска и бронирования лекарственных средств в аптечной сети. Пользователь может просматривать каталог препаратов, искать лекарства по основным параметрам, видеть аптечные предложения, создавать бронирования и отслеживать их состояние.

На этапе анализа была проанализирована предметная область и существующие продукты-аналоги. Сравнение показало, что для данного приложения не требуется повторять крупный аптечный маркетплейс. Проект ориентирован на локальную аптечную сеть.

В проектной части определены архитектура приложения, структура базы данных, логика программных компонентов и подход к разработке пользовательского интерфейса. Приложение разделено на клиентскую часть на React, серверную часть на Express и базу данных на PostgreSQL. Доступ к данным выполняется через Prisma ORM, а права пользователей разграничиваются соответствующими ролями.

Контроль качества выполнен через описание тестирования пользовательской и серверной части, расчёт цикломатической сложности, метрики Холстеда и SLOC. Результаты тестирования позволяют говорить о достаточном уровне разработанного приложения, а так же о корректности работы программы.

Поставленная цель достигнута – разработано и описано приложение, которое поддерживает путь от поиска лекарства до создания и обработки бронирования. Проект является основой для дальнейшего развития, в том числе добавления уведомлений, автоматического контроля срока бронирования, интеграции с внешними аптечными системами.

Список использованных источников

1. Интернет-аптека Apteka.ru, поиск лекарств // Сервис Apteka.ru [Электронный ресурс]. URL: <https://apteka.ru/> (Дата обращения: 18.04.2026)
2. Здравсити – интернет-аптека, поиск лекарств онлайн // Сервис Здравсити [Электронный ресурс]. URL: <https://zdravcity.ru/> (Дата обращения: 18.04.2026)
3. Мегаптека – Наличие и цены на лекарства в аптеках // Сервис Мегаптека [Электронный ресурс]. URL: <https://megapteka.ru/> (Дата обращения: 18.04.2026)
4. Леоненков, А.В. Самоучитель UML / А. В. Леоненков – 2-е изд. – СПб. : БХВ-Петербург, 2004. – 432 с.
5. Балашова И.Ю. Современные информационные технологии в проектировании программных систем и комплексов : учеб. пособие /И. Ю. Балашова, Д. В. Такташкин ; под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2019. 106 с.
6. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл – 9-е изд. – М. : Вильямс, 2012. – 864с.
7. Бэнкс, А. React и Redux. Функциональная разработка / А. Бэнкс, Е. Порселло. – СПб. : Питер, 2019 – 336с.
8. Документация PostgreSQL // Официальная документация PostgreSQL [Электронный ресурс]. URL: <https://www.postgresql.org/docs/current/> (Дата обращения: 01.05.2026)
9. Введение в TypeScript Handbook // Официальная документация TypeScript [Электронный ресурс]. URL: <https://www.typescriptlang.org/docs/handbook/> (Дата обращения: 02.05.2026)
10. Prisma ORM // Официальная документация Prisma [Электронный ресурс]. URL: <https://www.prisma.io/docs/orm> (Дата обращения: 03.05.2026)
11. Глава 14. Проектирование и разработка баз данных. Концептуальная, логическая и физическая модели данных // Официальная документация Visual Paradigm [Электронный ресурс]. URL: <https://circle.visual->

- paradigm.com/docs/database-design-engineering/database-designers-guide/conceptual-logical-and-physical-data-model/ (Дата обращения: 03.05.2026)
12. Карпова, Т. С. Базы данных: модели, разработка, реализация : учебное пособие / Т. С. Карпова. – 2-е изд. – Москва : ИНТУИТ, 2016. – 403 с.
 13. Фаулер, М. UML. Основы. / М. Фаулер. – 3-е изд. – СПб. : Символ-Плюс, 2004. – 192с.
 14. Флэнаган, Д. JavaScript. Подробное руководство / Д. Флэнаган – 7-е изд. – СПб. : Диалектика, 2021. – 720 с.
 15. Маршрутизация и middleware в Express // Официальная документация Express.js [Электронный ресурс]. URL: (Дата обращения: 05.05.2026)
 16. Начало работы с Vite // Официальная документация Vite [Электронный ресурс]. URL: <https://vite.dev/guide/> (Дата обращения: 06.05.2026)
 17. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – 8-е изд. – М. : Вильямс, 2005. – 1328с.
 18. Куликов, С. Тестирование программного обеспечения. Базовый курс / С. Куликов. – Минск : EPAM Systems, 2016. – 300с.
 19. Учебное пособие по тестированию программного обеспечения // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-testing/software-testing-tutorial/> (Дата обращения: 08.05.2026)
 20. Значения метрик кода // Официальная документация Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2022> (Дата обращения: 10.05.2026)
 21. Метрики программного обеспечения Холстеда // Обучающий портал «GeeksforGeeks» [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-engineering/software-engineering-halsteads-software-metrics/> (Дата обращения: 10.05.2026)
 22. Строки кода для SLOC: раскрываем показатели качества // Metrudev [Электронный ресурс]. URL: <https://www.metrudev.com/en/metrics/sloc-lines-of-code-unveiling-the-hidden-quality-metrics/> (Дата обращения: 10.05.2026)

23. Веб-приложение для поиска и бронирования лекарственных средств в аптечной сети // GitHub репозиторий [Электронный ресурс]. URL: <https://github.com/kostriginartem82-blip/Diplom-Apteka.git> (Дата обращения: 10.05.2026)

Приложение А
(справочное)
Глоссарий

REST API – программный интерфейс взаимодействия между клиентской и серверной частями приложения, основанный на HTTP-запросах и ресурсной структуре маршрутов.

Клиент-серверная архитектура – архитектура программной системы, при которой пользовательский интерфейс и серверная логика разделены на самостоятельные части, взаимодействующие между собой через сетевые запросы.

База данных – организованное хранилище данных, предназначенное для долговременного сохранения, поиска, изменения и связывания информации в приложении.

ORM – технология сопоставления объектов программы с таблицами базы данных, позволяющая выполнять операции с данными через программные модели.

ER-модель – модель данных, описывающая сущности предметной области, их атрибуты и связи между сущностями.

Транзакция – последовательность операций с базой данных, выполняемая как единое целое. При ошибке изменения отменяются, при успешном выполнении сохраняются.

Приложение Б
(обязательное)
Листинг приложения

Листинг Б.1 – Файл Schema.Prisma

```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url       = env("DATABASE_URL")
}

enum Role {
  CLIENT
  PHARMACY_STAFF
  ADMIN
}

enum ReservationStatus {
  PENDING
  CONFIRMED
  CANCELLED
  COMPLETED
}

model User {
  id          String      @id @default(uuid()) @db.Uuid
  email       String      @unique
  passwordHash String      @map("password_hash")
  fullName    String      @map("full_name")
  phone       String?
  role        Role         @default(CLIENT)
  pharmacyId  Int?         @map("pharmacy_id")
  pharmacy    Pharmacy?    @relation(fields: [pharmacyId],
references: [id], onDelete: SetNull)
  reservations Reservation[]
  createdAt   DateTime     @default(now()) @map("created_at")
  updatedAt   DateTime     @updatedAt @map("updated_at")

  @@map("users")
}

model City {
  id          Int          @id @default(autoincrement())
  name        String       @unique
  districts    District[]
  pharmacies  Pharmacy[]

  @@map("cities")
}

model District {
  id          Int          @id @default(autoincrement())
  name        String

```

```

        cityId      Int          @map("city_id")
        city        City          @relation(fields: [cityId], references:
[id], onDelete: Cascade)
        pharmacies Pharmacy[]

        @@unique([name, cityId])
        @@map("districts")
    }

model Pharmacy {
    id              Int              @id @default(autoincrement())
    name            String
    address         String
    phone           String?
    openingHours    String           @map("opening_hours")
    cityId          Int              @map("city_id")
    districtId      Int              @map("district_id")
    city            City             @relation(fields: [cityId], references:
[id], onDelete: Restrict)
    district        District         @relation(fields: [districtId],
references: [id], onDelete: Restrict)
    stockItems      StockItem[]
    staff           User[]
    createdAt       DateTime         @default(now()) @map("created_at")
    updatedAt       DateTime         @updatedAt @map("updated_at")

    @@map("pharmacies")
}

model Medicine {
    id              Int              @id @default(autoincrement())
    name            String
    activeSubstance String?          @map("active_substance")
    description      String
    manufacturer     String
    form             String
    dosage           String
    category         String
    imageUrl         String?         @map("image_url")
    prescriptionRequired Boolean      @default(false)
    @map("prescription_required")
    stockItems       StockItem[]
    createdAt        DateTime         @default(now())
    @map("created_at")
    updatedAt        DateTime         @updatedAt @map("updated_at")

    @@index([name])
    @@map("medicines")
}

model StockItem {
    id              Int              @id @default(autoincrement())
    medicineId      Int              @map("medicine_id")

```

```

    pharmacyId      Int                @map("pharmacy_id")
    price            Decimal            @db.Decimal(10, 2)
    quantity         Int
    reservedQuantity Int                @default(0)
  @map("reserved_quantity")
  medicine          Medicine           @relation(fields: [medicineId],
references: [id], onDelete: Cascade)
  pharmacy          Pharmacy           @relation(fields: [pharmacyId],
references: [id], onDelete: Cascade)
  reservations      Reservation[]
  updatedAt         DateTime           @updatedAt @map("updated_at")

  @@unique([medicineId, pharmacyId])
  @@index([pharmacyId])
  @@map("stock_items")
}

model Reservation {
  id          String                @id @default(uuid()) @db.Uuid
  code        String                @unique
  userId      String                @map("user_id") @db.Uuid
  stockItemId Int                  @map("stock_item_id")
  quantity    Int
  status       ReservationStatus    @default(PENDING)
  expiresAt   DateTime             @map("expires_at")
  notes       String?
  user        User                 @relation(fields: [userId],
references: [id], onDelete: Cascade)
  stockItem   StockItem            @relation(fields: [stockItemId],
references: [id], onDelete: Restrict)
  createdAt   DateTime             @default(now()) @map("created_at")
  updatedAt   DateTime             @updatedAt @map("updated_at")

  @@index([userId])
  @@index([stockItemId])
  @@map("reservations")
}

```

Листинг Б.2 – Файл reservation.routes.ts

```

import { Router } from "express";
import { ReservationStatus, Role, type PrismaClient } from
"@prisma/client";
import { prisma } from "../lib/prisma.js";
import { requireAuth } from "../middleware/auth.js";
import { ApiError } from "../middleware/errorHandler.js";
import { asyncHandler } from "../utils/asyncHandler.js";
import { reservationView } from "../utils/serialization.js";

const router = Router();

const reservationInclude = {

```

```

    user: { select: { id: true, email: true, fullName: true, phone:
true } },
    stockItem: {
      include: {
        medicine: true,
        pharmacy: { include: { city: true, district: true } }
      }
    }
  } as const;

function reservationCode() {
  return `BR-${Math.random().toString(36).slice(2,
6).toUpperCase()}-${Date.now()
.toString()
.slice(-5)}`;
}

function canSeeReservation(reqUser: Express.UserPayload,
reservation: { userId: string; stockItem: { pharmacyId: number } })
{
  if (reqUser.role === Role.ADMIN) {
    return true;
  }

  if (reqUser.role === Role.PHARMACY_STAFF) {
    return reqUser.pharmacyId === reservation.stockItem.pharmacyId;
  }

  return reqUser.id === reservation.userId;
}

function allowedStatusesForRole(role: Role): ReservationStatus[] {
  if (role === Role.CLIENT) {
    return [ReservationStatus.CANCELLED];
  }

  return [ReservationStatus.CONFIRMED, ReservationStatus.CANCELLED,
ReservationStatus.COMPLETED];
}

async function updateStockForStatus(
  tx: Omit<PrismaClient, "$connect" | "$disconnect" | "$on" |
"$transaction" | "$use" | "$extends">,
  stockItemId: number,
  quantity: number,
  fromStatus: ReservationStatus,

```

```

    toStatus: ReservationStatus
  ) {
    const activeStatuses: ReservationStatus[] =
[ReservationStatus.PENDING, ReservationStatus.CONFIRMED];

    if (fromStatus === toStatus) {
      return;
    }

    const releasesReservation =
      activeStatuses.includes(fromStatus) && toStatus ===
ReservationStatus.CANCELLED;

    const completesReservation =
      activeStatuses.includes(fromStatus) && toStatus ===
ReservationStatus.COMPLETED;

    if (releasesReservation) {
      await tx.stockItem.update({
        where: { id: stockItemId },
        data: { reservedQuantity: { decrement: quantity } }
      });
    }

    if (completesReservation) {
      await tx.stockItem.update({
        where: { id: stockItemId },
        data: {
          reservedQuantity: { decrement: quantity },
          quantity: { decrement: quantity }
        }
      });
    }
  }
}

router.post(
  "/",
  requireAuth,
  asyncHandler(async (req, res) => {
    const stockItemId = Number(req.body.stockItemId);
    const quantity = Number(req.body.quantity ?? 1);

    if (!Number.isInteger(stockItemId)
|| !Number.isInteger(quantity) || quantity <= 0) {
      throw new ApiError(400, "Valid stock item and quantity are
required");
    }
  })
);

```

```

    }

    const reservation = await prisma.$transaction(async (tx) => {
      const stockItem = await tx.stockItem.findUnique({ where:
{ id: stockItemId } });

      if (!stockItem) {
        throw new ApiError(404, "Stock item was not found");
      }

      const availableQuantity = stockItem.quantity -
stockItem.reservedQuantity;
      if (availableQuantity < quantity) {
        throw new ApiError(409, "Not enough medicine quantity
available");
      }

      await tx.stockItem.update({
        where: { id: stockItemId },
        data: { reservedQuantity: { increment: quantity } }
      });

      return tx.reservation.create({
        data: {
          code: reservationCode(),
          userId: req.user!.id,
          stockItemId,
          quantity,
          status: ReservationStatus.PENDING,
          expiresAt: new Date(Date.now() + 24 * 60 * 60 * 1000)
        },
        include: reservationInclude
      });
    });

    res.status(201).json({ data: reservationView(reservation) });
  })
);

router.get(
  "/",
  requireAuth,
  asyncHandler(async (req, res) => {
    const where =
      req.user!.role === Role.ADMIN
        ? {}

```

```

        : req.user!.role === Role.PHARMACY_STAFF
        ? { stockItem: { pharmacyId: req.user!.pharmacyId ?? -
1 } }
        : { userId: req.user!.id };

const reservations = await prisma.reservation.findMany({
  where,
  include: reservationInclude,
  orderBy: { createdAt: "desc" }
});

res.json({ data: reservations.map(reservationView) });
})
);

router.patch(
  "("/:id/status",
  requireAuth,
  asyncHandler(async (req, res) => {
    const nextStatus = req.body.status as ReservationStatus |
undefined;

    if (!nextStatus
|| !Object.values(ReservationStatus).includes(nextStatus)) {
      throw new ApiError(400, "Valid reservation status is
required");
    }

    if
(!allowedStatusesForRole(req.user!.role).includes(nextStatus)) {
      throw new ApiError(403, "This role cannot set the selected
status");
    }

    const current = await prisma.reservation.findUnique({
      where: { id: req.params.id },
      include: { stockItem: true }
    });

    if (!current) {
      throw new ApiError(404, "Reservation was not found");
    }

    if (!canSeeReservation(req.user!, current)) {
      throw new ApiError(403, "Access denied");
    }

```

```

    const finalStatuses: ReservationStatus[] =
[ReservationStatus.CANCELLED, ReservationStatus.COMPLETED];

    if (finalStatuses.includes(current.status)) {
        throw new ApiError(409, "Final reservation status cannot be
changed");
    }

    const reservation = await prisma.$transaction(async (tx) => {
        await updateStockForStatus(tx, current.stockItemId,
current.quantity, current.status, nextStatus);

        return tx.reservation.update({
            where: { id: current.id },
            data: { status: nextStatus },
            include: reservationInclude
        });
    });

    res.json({ data: reservationView(reservation) });
})
);

export default router;

```

Листинг Б.3– Файл auth.ts

```

import type { NextFunction, Request, Response } from "express";
import jwt from "jsonwebtoken";
import { Role } from "@prisma/client";
import { env } from "../config/env.js";
import { ApiError } from "../errorHandler.js";

type TokenPayload = {
    id: string;
    email: string;
    role: Role;
    pharmacyId: number | null;
};

export function requireAuth(req: Request, _res: Response, next:
NextFunction) {
    const authHeader = req.headers.authorization;
    const token = authHeader?.startsWith("Bearer ") ?
authHeader.slice(7) : null;

    if (!token) {
        return next(new ApiError(401, "Authorization token is
required"));
    }

```

```

    }

    try {
      const payload = jwt.verify(token, env.jwtSecret) as
TokenPayload;
      req.user = payload;
      return next();
    } catch {
      return next(new ApiError(401, "Invalid or expired token"));
    }
  }

export function requireRoles(...roles: Role[]) {
  return (req: Request, _res: Response, next: NextFunction) => {
    if (!req.user) {
      return next(new ApiError(401, "Authorization token is
required"));
    }

    if (!roles.includes(req.user.role)) {
      return next(new ApiError(403, "Access denied"));
    }

    return next();
  };
}

```