

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ПРИЛОЖЕНИЕ «DocFinder: поиск врачей
и мгновенная запись онлайн»**

Студент М.К. 28.05.2025 Козлова Кристина Сергеевна
(подпись, дата) (ФИО полностью)

Руководитель В. 30.05.2025 Гурин Е.И.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 30.05.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025г. № 16)

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от
2 июня 2025г. № 2)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Козлова Кристина Сергеевна гр. 21ВВП1 факультета вычислительной техники

2. Тема работы Приложение «DocFinder: поиск врачей и мгновенная запись онлайн».

Тема утверждена приказом ПГУ № 601/о от "6" мая 2025г.

3. Руководитель бакалаврской работы профессор кафедры ВТ ПГУ Гурин Е.И.

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

Разработать веб-приложение «DocFinder» для поиска врачей и онлайн-записи на прием. Реализовать архитектуру клиент-серверного взаимодействия. Язык программирования — JavaScript. Клиентская часть должна быть реализована с использованием библиотеки React, серверная — на платформе Node.js с фреймворком Express. В качестве системы управления базами данных использовать PostgreSQL. Приложение должно поддерживать аутентификацию и авторизацию с помощью JWT, обработку записей, управление личными кабинетами пользователей различных ролей (пациент, врач, администратор), а также интеграцию с Яндекс.Картами. Размещение приложения выполнить на VPS-сервере с использованием Nginx и SSL-сертификатов для обеспечения безопасности соединения.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить существующие аналоги сервисов записи к врачам, определить требования к функционалу, выбрать стек технологий. Спроектировать архитектуру приложения, реализовать серверную и клиентскую части, настроить взаимодействие с базой данных, реализовать безопасную авторизацию, интеграцию с картографическим сервисом и развертывание на удаленном сервере.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	
2	Анализ требований	8%	07.05.2025-08.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	8%	09.05.2025-10.05.2025	
4	Проектирование архитектуры	12%	11.05.2025-13.05.2025	
5	Проектирование и Разработка ПО	44%	14.05.2025-24.05.2025	
6	Тестирование и отладка	12%	25.05.2025-27.05.2025	
7	Оформление выпускной работы	12%	28.05.2025-30.05.2025	

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Турин Е.И.
(подпись, дата)

Задание к исполнению принял студент  06.05.25 Козлова К.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  30.05.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 126 листов, 56 рисунка, 6 таблиц, 8 источников, 4 приложения

КЛИЕНТ, СЕРВЕР, КАРТА, БЕЗОПАСНОСТЬ, РОЛЬ, ВРАЧ, ПАЦИЕНТ, API, БАЗА ДАННЫХ, ПОЧТА, REACT, NODE.JS, JAVASCRIPT

Цель работы — разработать клиент-серверное веб-приложение, обеспечивающее удобный поиск врачей и запись на прием с функциями управления расписанием и отзывами.

Разработанное веб-приложение «DocFinder» предназначено для пациентов, которые хотят быстро и удобно найти подходящего врача и записаться на прием онлайн, а также для медицинских специалистов, которым требуется эффективный инструмент для управления расписанием, взаимодействия с пациентами и мониторинга качества оказываемых услуг.

В ходе разработки создано клиент-серверное приложение с использованием JavaScript, React для клиентской части и Node.js с фреймворком Express для серверной, а также базы данных PostgreSQL. Реализованы функции регистрации и авторизации пользователей с различными ролями (пациент, врач, администратор), защищённый обмен данными с помощью JWT и интеграция с внешними сервисами, включая Яндекс.Карты. Приложение обеспечивает удобный и интуитивный интерфейс, соответствует функциональным требованиям и предназначено для пациентов, медицинских специалистов и администраторов медицинских учреждений.

					ПГУ1.090301.11.001 ПЗ				
ам.	Лист	№ докум.	Подпись	Дата	Приложение «DocFinder: поиск врачей и мгновенная запись онлайн» Пояснительная записка	Лист	Лист	Листов	
израб.		Козлова К.С.		30.05/3					
ровер.		Гурин Е.И.		30.05/3			2	126	
Контр		Бычков А.С.		30.05/3		ФВТ, каф. ВТ гр. 21ВВП1			
твсрд.		Митрохин М.А.		30.05/3					

Содержание

Перечень сокращений и обозначений.....	6
Введение.....	7
1 Анализ предметной области	9
1.1 Анализ актуальности темы	9
1.2 Анализ существующих аналогов.....	10
1.2.1 ПроДокторов	10
1.2.2 СберЗдоровье	12
1.2.3 Вывод	14
1.3 Постановка задачи	15
2 Выбор технологий и средств разработки приложения	18
2.1 Выбор языка программирования.....	18
2.1.1 JavaScript.....	18
2.1.2 PHP	19
2.1.3 Python	19
2.1.4 Вывод	20
2.2 Выбор фреймворка для клиента	20
2.2.1 Vue.js	20
2.2.2 Angular	20
2.2.3 React	21
2.2.4 Вывод	21
2.3 Выбор фреймворка для сервера.....	21
2.3.1 Express.js	21
2.3.2 Koa.js	22
2.3.3 Napi.js	22
2.3.4 Вывод	23
2.4 Выбор СУБД.....	23
2.4.1 PostgreSQL	23
2.4.2 MySQL	23
2.4.3 Вывод	24
2.5 Выбор среды разработки	24

2.5.1 WebStorm	24
2.5.2 Visual Studio Code	24
2.5.3 Вывод	25
2.6 Выбор паттерна проектирования архитектуры.....	25
2.6.1 MVC (Model-View-Controller)	25
2.6.2 MVP (Model-View-Presenter)	26
2.6.3 MVVM (Model-View-ViewModel).....	27
2.6.4 Вывод	28
2.7 Вывод	30
3 Разработка.....	31
3.1 Разработка VPS-сервера	31
3.2 Разработка серверной части приложения.....	34
3.3 Разработка базы данных	41
3.4 Разработка клиентской части приложения.....	42
3.5 Анализ результатов.....	48
3.5.1 Тестирование API	48
3.5.2 Тестирование интерфейса.....	51
4 Руководство пользователя.....	60
4.1 Навигация и пользовательский интерфейс	60
4.2 Начало работы	62
4.3 Функционал для пользователей с ролью пациент	63
4.4 Функционал для пользователей с ролью доктор	68
4.5 Функционал для пользователей с ролью администратор	71
5 Охрана труда.....	74
5.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ.....	74
5.2 Классификация и категорирование помещения, в котором производится разработка	75
5.2.1 Классификация по взрывопожарной и пожарной опасности	75
5.2.2 Классификация по степени опасности поражения электрическим током	76
5.2.3 Рекомендации по обеспечению безопасных условий	77

5.3 Санитарно-гигиеническая характеристика производства	78
5.4 Расчёт и анализ условий визуальной работы в помещении с персональными вычислительными машинами	79
Заключение	82
Список использованных источников	84
Приложение А – UML-диаграммы.....	86
Приложение Б – Листинг программы	91
Приложение В – Презентация.....	117
Приложение Г – Защита информационной системы персональных данных (ИСПДн).....	123

Перечень сокращений и обозначений

В данной работе применяют следующие сокращения и обозначения

ОС – операционная система

API (Application Programming Interface) – программный интерфейс приложения

CSS (Cascading Style Sheets) – каскадные таблицы стилей

DOM (Document Object Model) – объектная модель документа

HTML (HyperText Markup Language) – язык разметки гипертекста

IDE (Integrated Development Environment) – интегрированная среда разработки

JWT (JSON Web Token) – веб-токен для авторизации

JSON (JavaScript Object Notation) – текстовый формат обмена данными

LTS (Long Term Support) – версия с длительной поддержкой

MVC (Model-View-Controller) – модель-представление-контроллер

MVP (Model-View-Presenter) – модель-представление-презентер

MVVM (Model-View-ViewModel) – модель-представление-модель представления

ORM (Object-Relational Mapping) – объектно-реляционное отображение

SFTP (Secure File Transfer Protocol) – безопасный протокол передачи файлов

SMTP (Simple Mail Transfer Protocol) – простой протокол передачи почты

SQL (Structured Query Language) – язык структурированных запросов

SSH (Secure Shell) – защищённый сетевой протокол

VPS (Virtual Private Server) - виртуальный выделенный сервер

Введение

В современном обществе качество медицинского обслуживания напрямую связано с доступностью и удобством взаимодействия между пациентами и врачами. Традиционные методы записи на прием зачастую создают неудобства и задержки, что снижает эффективность и комфорт получения медицинских услуг. Одновременно, медицинские специалисты сталкиваются с необходимостью эффективного управления своим расписанием и коммуникацией с пациентами в условиях растущей конкуренции и цифровизации отрасли.

В связи с этим возникает потребность в создании инновационных цифровых решений, которые упрощают процесс поиска врачей и записи на прием, обеспечивают прозрачность информации и повышают качество обратной связи. Разработка такого приложения позволяет не только оптимизировать время и усилия обеих сторон, но и способствует формированию более доверительных и эффективных отношений между пациентом и врачом.

Целью данной работы является проектирование и реализация веб-приложения, которое обеспечивает двунаправленное удобное взаимодействие: предоставляет врачам инструменты для управления своей практикой и позволяет пациентам быстро находить специалистов и записываться на прием с использованием современных технологий поиска и фильтрации. В процессе разработки будут освоены навыки построения надежной клиент-серверной архитектуры, организации безопасной аутентификации и авторизации пользователей, а также интеграции с внешними сервисами.

Полученные результаты имеют прикладное значение и могут быть использованы в дальнейшем для улучшения цифровой инфраструктуры здравоохранения. Так же результаты работы будут полезны для приобретения практических навыков в области веб-разработки, построения клиент-серверных приложений и работы с базами данных. Кроме того, реализация

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

проекта способствует развитию компетенций в проектировании удобных пользовательских интерфейсов, обеспечении безопасности и управлении доступом, а также интеграции сторонних сервисов, таких как геолокация и почтовые рассылки.

					ПГУ1.090301.11.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

Для начала необходимо определить целевую аудиторию приложения и функциональные требования, исходя из её потребностей. Затем — определить контекст использования сервиса. Важно изучить архитектуру приложений, так как от её правильной реализации зависит стабильность работы и соответствие ожиданиям пользователей. Это также позволит заранее предусмотреть и решить возможные технические проблемы в процессе разработки.

1.1 Анализ актуальности темы

В условиях цифровизации мира появляется огромное количество возможностей для оптимизации повседневных задач. Все больше людей, которые переходят на онлайн-сервисы для оказания услуг, сталкиваются с проблемой поиска нужного им специалиста. Здоровье – это первая и самая главная потребность человека, а найти нужного врача становится непростой задачей в жизни любого жителя. Традиционные способы: через регистратуру или по телефону – часто оказываются неудобными, медленными и неэффективными. Поэтому современному человеку нужен способ быстрого поиска и записи к специалисту, с предоставлением возможности чтения отзывов других пациентов и выбора комфортного расположения относительно своего места жительства, работы или учебы.

Не мало важную роль такие сервисы играют в жизни врачей. Современные реалии медицинской практики предъявляют всё более высокие требования к эффективности взаимодействия врача с пациентами. В условиях высокой конкуренции и плотного рабочего графика для специалистов становится актуальной задача автоматизации процесса записи, а также расширения присутствия в цифровом пространстве. Особенно это важный аспект для частнопрактикующих врачей, которым нужна платформа для управления записью и продвижения своих профессиональных услуг.

Таким образом, разработка сервиса для поиска и записи к врачу актуальна для запросов обеих сторон и способствует повышению доступности, удобства и качества медицинских услуг в цифровую эпоху

1.2 Анализ существующих аналогов

В данном пункте рассматриваются одни из самых популярных сервисов, предоставляющие возможность найти специалиста в сфере медицины. У каждого сервиса есть свои плюсы и минусы, которые необходимо рассмотреть для дальнейшей разработки.

1.2.1 ПроДокторов

Данный сервис является крупнейшим сайтом с отзывами о врачах, которая так же позволяет записываться онлайн к нужному специалисту. Сервис обеспечивает большое покрытие городов России, включая региональные города.

На рисунке 1.1 представлен интерфейс главной страницы сервиса «ПроДокторов»

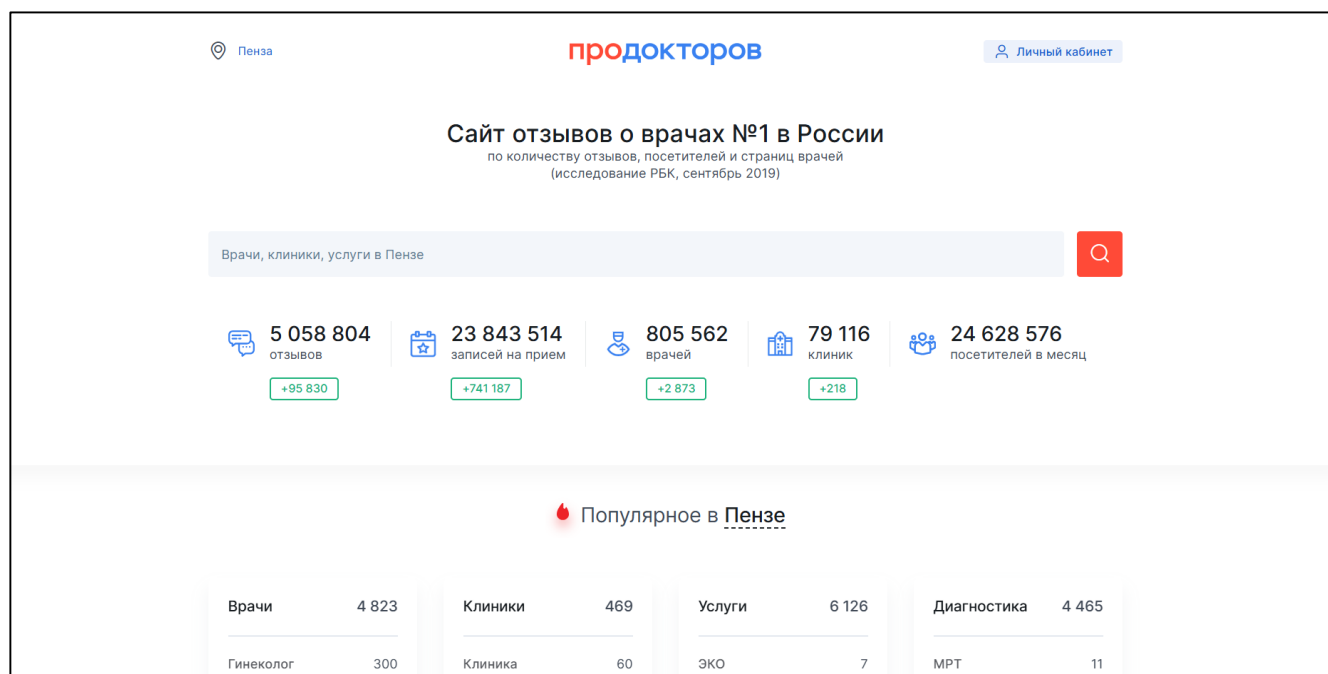


Рисунок 1.1 – Главная страница сервиса «ПроДокторов»

На рисунках 1.2 и 1.3 представлена страница поиска и фильтрации по специалистам.

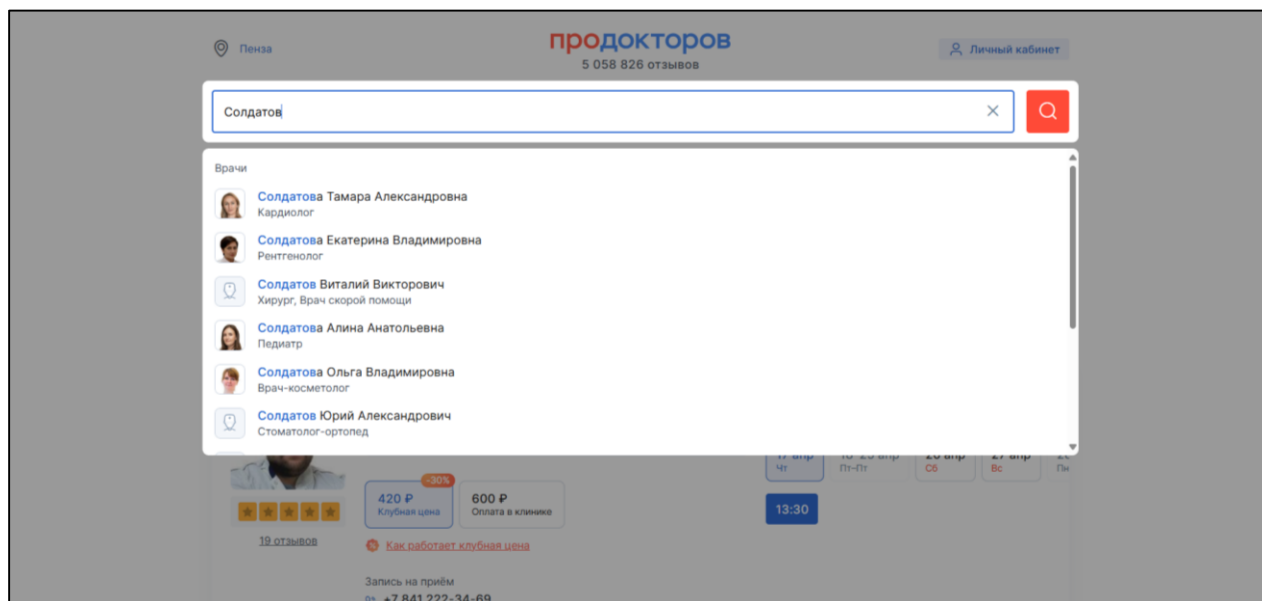


Рисунок 1.2 – Результат поиска по специалистам

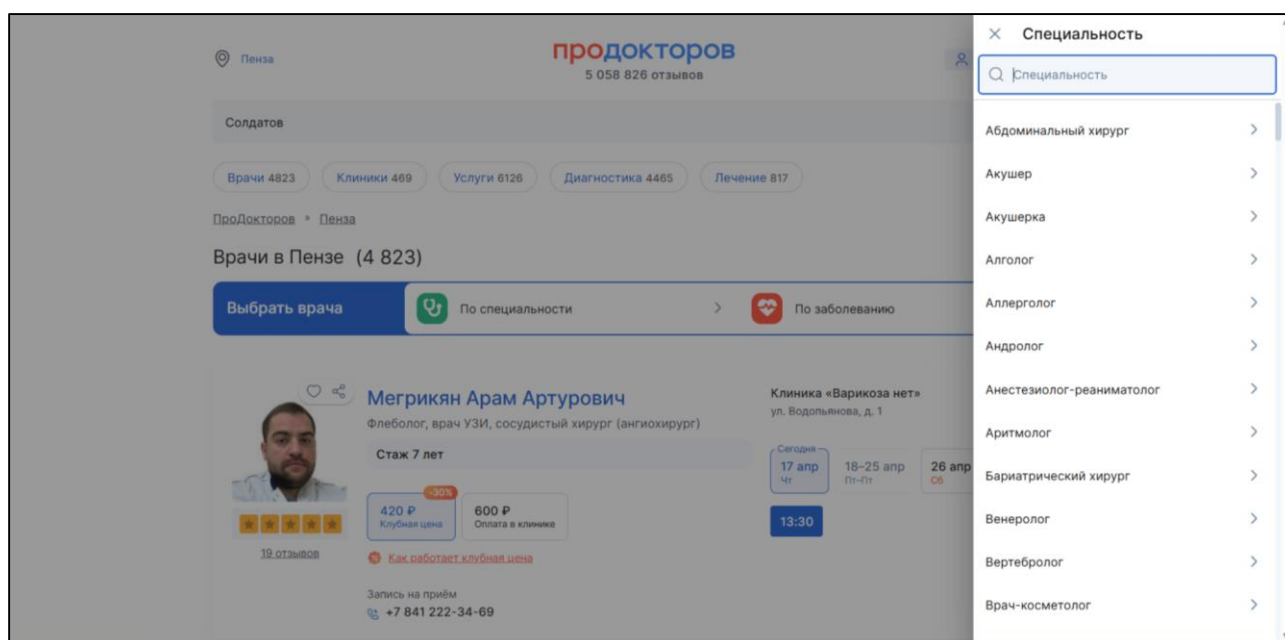


Рисунок 1.3 – Фильтрация по специальностям

Личный кабинет в сервисе представлен интеграцией платформы МедТочка (рисунок 1.4).

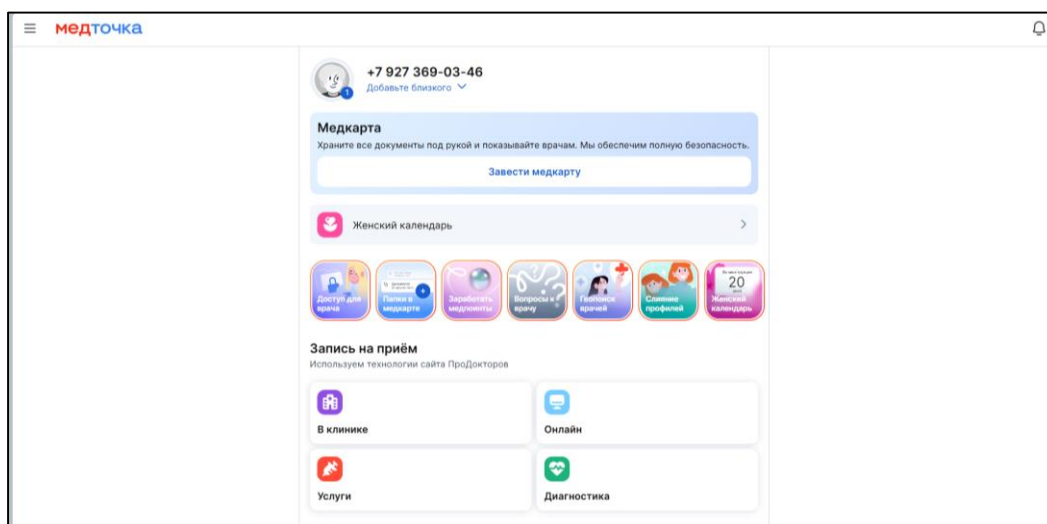


Рисунок 1.4 – Личный кабинет ПроДокторов

Достоинства сервиса:

- большое количество специалистов из разных городов России;
- возможность читать отзывы на специалиста;
- возможность сортировки по рейтингу, опыту, цене;
- можно записаться на удобное время прямо на сайте.

Недостатки сервиса:

- возможность платного продвижения, что влияет на объективность платформы;
- не всегда достоверные отзывы. Многие пользователи жалуются, что негативные отзывы не публикуются, а у положительных не проверяется посещение врача;
- даже при хороших отзывах врачи наблюдали падение рейтинга.

1.2.2 СберЗдоровье

Сервис «СберЗдоровье» предлагает пользователям широкий спектр медицинских услуг, включая онлайн-консультации, запись к врачам

На рисунке 1.5 представлен интерфейс главной страницы сервиса «СберЗдоровье».

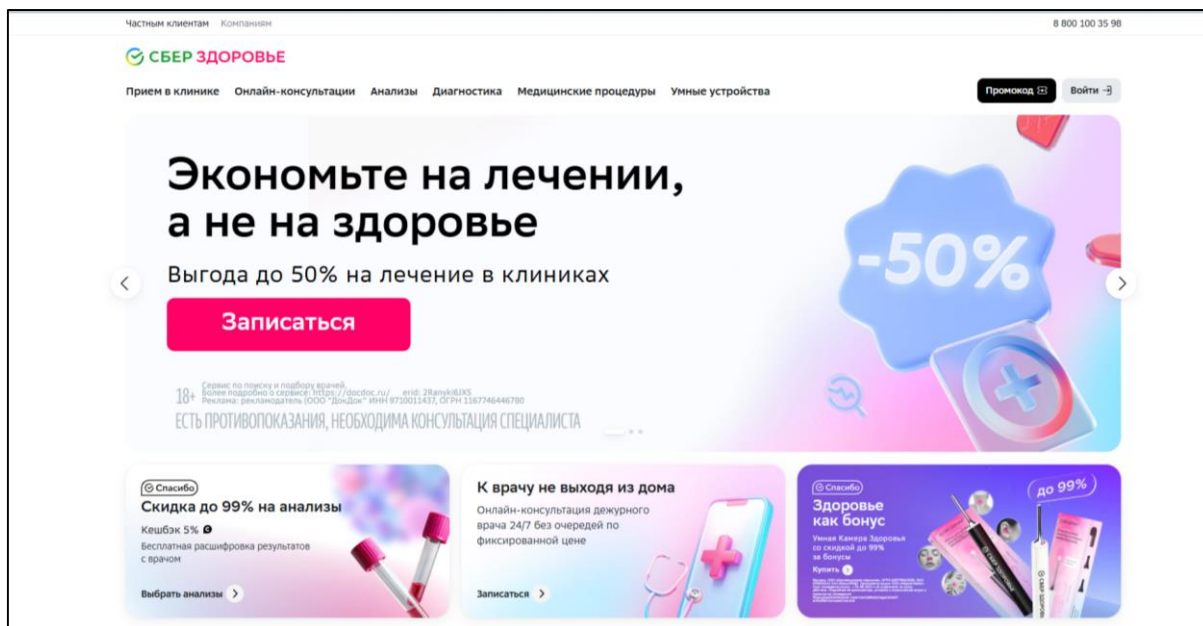


Рисунок 1.5 – Главная страница «СберЗдоровье»

Сайт предоставляет возможность как отфильтровать специалистов через поисковую строку (рисунок 1.6), так и воспользоваться настройками фильтрации (рисунок 1.7).

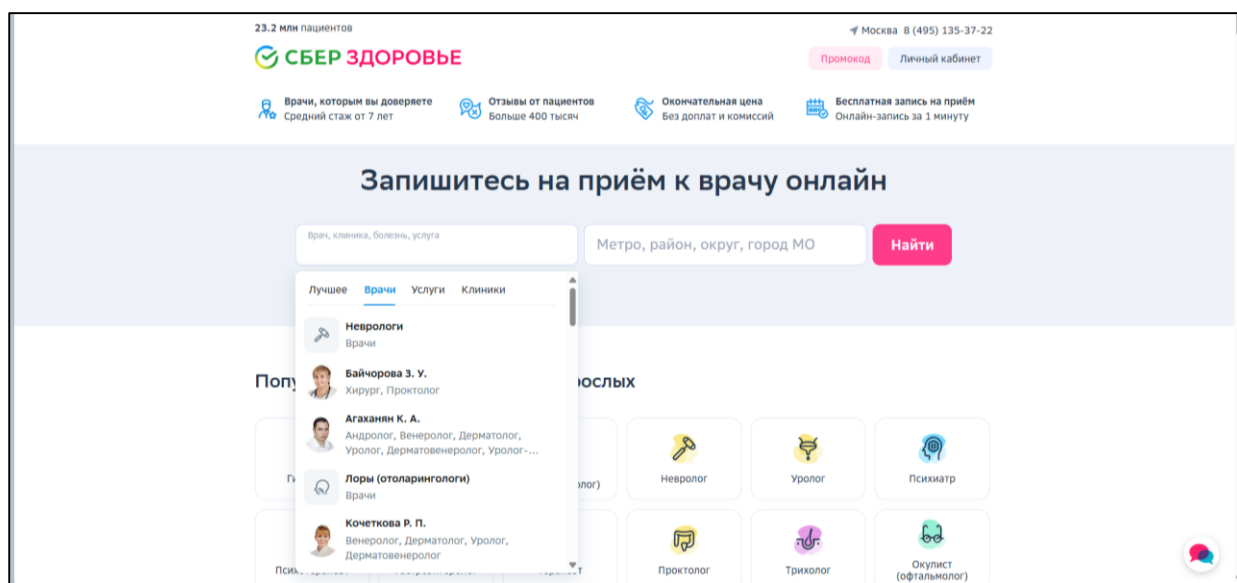


Рисунок 1.6 – Поиск по специалистам «СберЗдоровье»

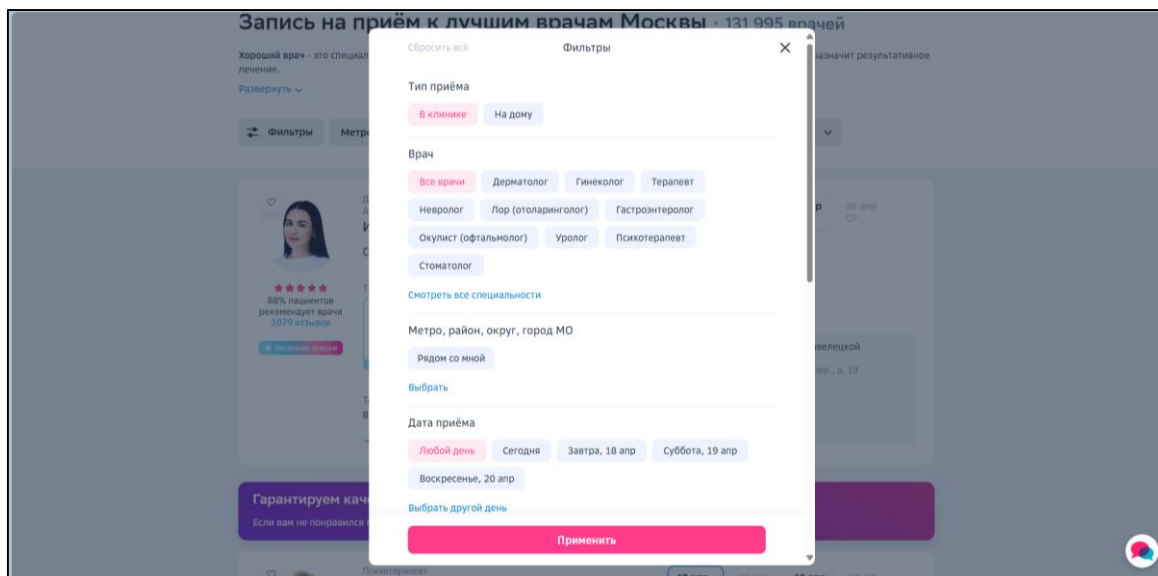


Рисунок 1.7 – Поиск с фильтром «СберЗдоровье»

Достоинства:

- возможность записаться к врачу и получить онлайн консультацию;
- простой и понятный интерфейс;
- наличие скидок и бонусных программ.

Недостатки:

- навязывание сервисов Сбербанка и дополнительных услуг;
- не ориентирован на отзывы, поэтому их мало, а у некоторых специалистов вообще нет;
- вовремя онлайн консультации нельзя выбрать врача.

1.2.3 Вывод

На основе анализа аналогов было принято решение сделать клиент-серверное веб-приложение, что является стандартом для современных веб-приложений и идеально подходит для обеспечения разделения между интерфейсом пользователя и бизнес-логикой приложения, «DocFinder: поиск врачей и мгновенная запись онлайн». Главной задачей будет являться интуитивно понятный интерфейс, поиск с фильтрами, запись и история записей, а также возможность врачу составлять график приема.

Изм.	Лист	№ докум.	Подпись	Дата

1.3 Постановка задачи

Необходимо разработать клиент-серверное веб-приложение для поиска и записи к врачу. Сервис должен обладать следующим функционалом:

Личный кабинет врача с возможностью заполнять информацию о себе, которая будет отображаться в карточке врача: фото, ФИО, адрес, по которому осуществляется прием, дополнительная информация, сертификаты. Редактирование таблицы для составления удобного расписания приема пациентов. Возможность посмотреть отзывы, которые оставили пациенты.

Личный кабинет с возможностью для пациента просматривать историю прошедших и предстоящих записей, просмотр оставленных отзывов.

Обработка поисковых запросов пользователя, фильтры поиска.

Возможность поиска врачей с фильтрами по картам.

Обработка сервером процесса записи к врачу, внесение соответствующих изменений в БД и отображение клиентом изменившейся информации о наличии и доступности записи.

Минимальные системные требования:

- Google Chrome, Mozilla Firefox, Microsoft Edge (с поддержкой HTML5, CSS3);
- Linux (Ubuntu или CentOS), macOS или Windows 10/11;
- Node.js минимальная версия: 16.x.x (рекомендуется использовать LTS-версию);
- PsqI минимальная версия: 12.x ;
- скорость интернета 2-3Мбит/с (рекомендуется от 5 Мбит/с и выше);
- процессор: 2 ядра, предпочтительно с поддержкой x64;
- минимальный объем оперативной памяти: 4ГБ;
- минимум 10 GB свободного места для установки зависимостей, базы данных и логов.

Функционал для авторизованных пользователей больше, чем для не авторизованных.

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

Незарегистрированный пользователь может только искать и просматривать карточки специалистов, без возможности записаться.

Для зарегистрированных пользователей с ролью «Пациент» более широкий спектр возможностей. Они могут не только просматривать врачей, но и записываться на прием, а также оставлять отзывы после посещения врача. Доступен личный кабинет, с возможностью просматривать и редактировать информацию о пользователе. Там же можно посмотреть свою историю записи и оставленные отзывы.

Пользователи, имеющие роль «Доктор», помимо просмотра списка врачей, может просматривать и редактировать информацию о себе в личном кабинете, составлять таблицу записи на прием, загружать сертификаты, а также просматривать оставленные о нем отзывы.

Пользователи, имеющие роль «Администратор» могут просматривать списки специалистов, их карточки, удалять некорректные отзывы, присваивать пользователям роль «Доктор», заполнять описание доктора, стаж и специальность.

Диаграмма вариантов использования приложения для неавторизированных пользователей и пользователей с разными ролями представлена на рисунке 1.8 .

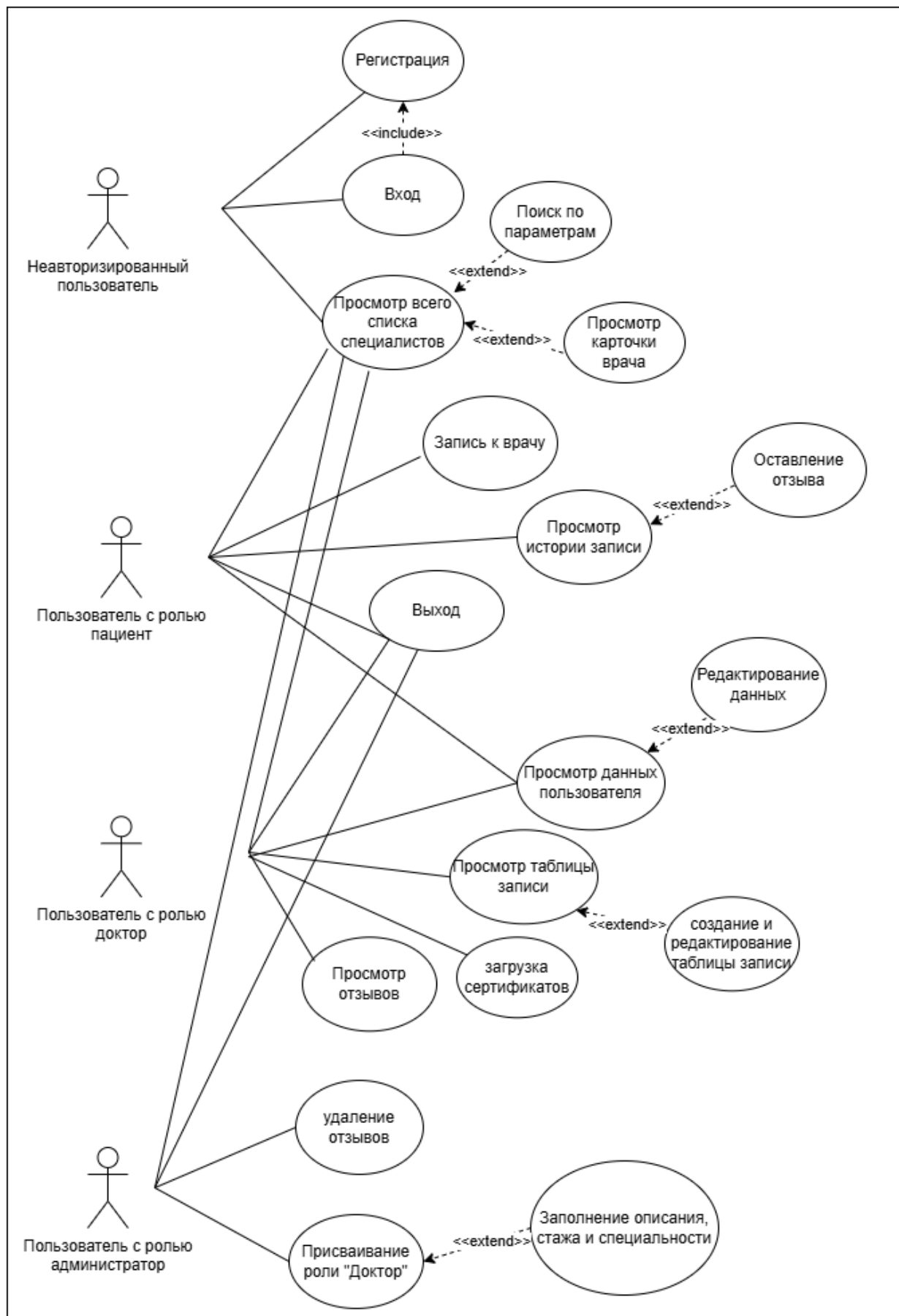


Рисунок 1.8 – UML диаграмма вариантов использования

2 Выбор технологий и средств разработки приложения

В данном разделе будет проведен анализ средств разработки, которые могут быть использованы для разработки клиент-серверного web-приложения. Исходя из анализа были выбраны технологии, которые наиболее удобны.

2.1 Выбор языка программирования

2.1.1 JavaScript

JavaScript – один из самых популярных языков программирования, используемых для разработки веб-страниц. Изначально он был создан для добавления динамичности и интерактивности в веб-страницы, он работает непосредственно в браузере и может взаимодействовать с DOM, позволяя обновлять контент без перезагрузки страницы, а с развитием технологий так же стал активно использоваться и для разработки серверных приложений.

Благодаря стремительно развивающимся фреймворкам разработка становится более быстрой и эффективной. Они предоставляют возможность динамического обновления контента и управления состоянием, маршрутизацией. JavaScript является интерпретируемым языком, что позволяет клиенту исполняться с помощью браузера, а серверной части благодаря Node.js. Одной из главных особенностей является поддержка асинхронности, что очень важно для работы с веб-приложениями, которым нужно часто взаимодействовать с сервером.

Благодаря развитию Node.js появилась возможность запускать JavaScript на сервере, что позволило разработчикам использовать единый язык для разработки как на клиентской, так и на серверной стороне.

Так же данный язык имеет большое сообщество, из-за чего доступно огромное количество ресурсов, документации и библиотек, что ускоряет процесс разработки.

2.1.2 PHP

PHP – это серверный язык программирования, созданный для разработки динамических веб-страниц. На данный момент PHP является одним из самых популярных языков для создания серверной части веб-приложений, особенно в контексте CMS, таких как WordPress, Joomla и Drupal.

Код PHP выполняется на сервере, а не в браузере, генерируя HTML, который после отправляет на клиент. PHP является интерпретируемым языком, код выполняется с помощью интерпретатора, благодаря чему не требуется компиляция. Он имеет простой синтаксис, легко интегрируется с базами данных, особенно с MySQL. Несмотря на улучшения, язык имеет меньшую производительность, если сравнивать с Java или C++.

В отличие от других современных языков PHP не может предоставить продвинутых инструментов для параллельного выполнения и многозадачности.

2.1.3 Python

Python – это высокоуровневый язык программирования с простым синтаксисом, большим количеством инструментов для разработки и библиотек, используется в различных областях, включая веб-разработку. Python широко применяется для создания серверных приложений и является одним из самых популярных языков разработки в области искусственного интеллекта.

Python поддерживает несколько парадигм программирования: процедурное, объектно-ориентированное и функциональное. Это позволяет разработчику выбирать подходящий стиль программирования для решения задач.

Одним из главных преимуществ Python является наличие огромного количества библиотек и фреймворков для разных областей: от веб-разработки до научных вычислений. Python имеет встроенную поддержку многозадачности, хотя стандартная реализация (CPython) имеет ограничения из-за GIL (Global Interpreter Lock).

2.1.4 Вывод

В качестве языка для разработки был выбран JavaScript, так как позволяет разрабатывать как фронтенд, так и бэкенд, что существенно упрощает разработку.

2.2 Выбор фреймворка для клиента

2.2.1 Vue.js

Vue.js – Это фреймворк для создания пользовательских интерфейсов. Он был создан Эваном Ю в 2014 году, и с тех пор популярен благодаря своей простоте, гибкости и доступности для разработчиков всех уровней.

Vue использует компонентный подход, что позволяет организовывать код и повторно использовать компоненты. Он имеет встроенную систему реактивности, которая отслеживает изменение данных и автоматически обновляет DOM.

Vue идеально подходит для проектов, где требуется простота, быстрота разработки.

2.2.2 Angular

Angular – полноценный фреймворк, который уже включает все необходимые средства для разработки, такие как маршрутизация, управление состоянием, взаимодействие с сервером и т.д.

Особенностью Angular является использование двусторонней привязки данных, то есть изменения в модели данных автоматически отражаются в представлении, и наоборот, что упрощает синхронизацию данных в сложных приложениях. Angular разработан с использованием TypeScript, что добавляет статическую типизацию в JavaScript, улучшая поддержку в процессе разработки и предотвращая ошибки на ранних стадиях.

При все этом Angular относительно тяжелый фреймворк, что может стать помехой в проектах, которые нуждаются в быстро загрузке. Так же имеет более строгую структуру и меньше гибкости в сравнении с React и Vue.

2.2.3 React

React – одна из самых популярных библиотек для создания пользовательских интерфейсов. Он использует компонентный подход, который позволяет разбивать интерфейс на компоненты для повторного использования. Это существенно облегчает разработку, так как каждый компонент можно разрабатывать независимо, а затем собирать вместе.

Virtual DOM – главная особенность React, обеспечивающая высокую производительность при изменении в интерфейсе. Вместо того, чтобы обновлять весь DOM, React сначала обновляет Virtual DOM, затем происходит сравнение и обновляются только те элементы, которые были изменены, что уменьшает время рендеринга.

React не навязывает разработчику структуры для всего приложения, что делает его очень гибким. Он может быть интегрирован с любыми другими библиотеками и фреймворками.

2.2.4 Вывод

В качестве фреймворка для разработки клиентской части приложения выбран React, это обусловлено его производительностью, гибкостью, простотой разработки и широкой экосистемой инструментов и библиотек.

2.3 Выбор фреймворка для сервера

2.3.1 Express.js

Express.js – это минималистичный и гибкий фреймворк для Node.js, который облегчает создания серверных приложений и API. Он предоставляет базовый набор для маршрутизации, обработки запросов и работы с HTTP-методами.

Express легковесный тем, что предоставляет разработчикам только основные возможности, что позволяет избежать лишней сложности и фрагментации кода. Он поддерживает использование middleware — функций, которые выполняются на каждом шаге обработки запроса. Express не

навязывает строгое разделение приложений или архитектуры, но при этом легко интегрируется с другими фреймворками и библиотеками для расширения функциональности. Его гибкость позволяет использование внешних модулей для работы с базами данных, сессиями, кэшированием и т.д.

2.3.2 Koa.js

Koa.js – фреймворк, разработанный как более современная и улучшенная альтернатива Express, фокусирующаяся на минимализме и простоте. Он предоставляет только самые основные функции для создания серверных приложений, оставляя выбор инструментов для маршрутизации, обработки тел запросов и т.д на разработчика.

Одной из ключевых особенностей является асинхронность. Koa использует `async/await` на базе Promise, что позволяет обрабатывать асинхронные операции более удобно и понятно, чем с использованием колбэков. Требуется больше настроек и усилий для реализации таких функций, как маршрутизация или обработка тела запросов, по сравнению с Express.

2.3.3 Napi.js

Napi.js – это мощный и функциональный фреймворк для Node.js. Он был разработан с фокусировкой на безопасность и расширяемость приложения, поэтому отлично подходит для больших, сложных приложений.

В отличие от Koa и Express, Napi предоставляет широкий набор встроенных возможностей для создания веб-приложений, таких как маршрутизация, валидация данных, аутентификация, обработка ошибок, логирование и другие функции. Napi предоставляет сильную систему маршрутизацию, с гибкой настройкой под нужды разработки. Этот фреймворк позволяет легко организовать функциональность через плагины.

Для небольших проектов слишком тяжеловесный, так как требуется только простая маршрутизация, поэтому более предпочтителен для сложных приложений.

2.3.4 Вывод

Выбор пал на Express.js, это обусловлено тем, что он позволяет эффективно и быстро разработать решение с возможностью масштабирования и простоты поддержки.

2.4 Выбор СУБД

2.4.1 PostgreSQL

PostgreSQL – это объектно-реляционная система управления базами данных с открытым исходным кодом, которая поддерживает большинство стандартов SQL и предоставляет расширенные возможности для работы с данными.

Помимо стандартных типов данных (целые числа, строки, даты), PostgreSQL поддерживает расширенные типы данных, такие как JSON, XML, hstore (для хранения пар ключ-значение), а также пользовательские типы данных. PostgreSQL использует MVCC (Multiversion Concurrency Control), что позволяет обрабатывать несколько транзакций одновременно без блокировок, обеспечивая высокую производительность при конкурентных запросах.

pgAdmin – это популярный графический интерфейс для управления базами данных PostgreSQL, который значительно упрощает администрирование, настройку и выполнение SQL-запросов.

2.4.2 MySQL

MySQL – это также популярная реляционная система управления базами данных с открытым исходным кодом, ориентированная на быстроту выполнения запросов и простоту настройки.

MySQL демонстрирует отличную производительность при высоких нагрузках на чтение и часто используется в веб-приложениях с высоким трафиком (например, в WordPress). MySQL полностью поддерживает стандарт SQL, но имеет некоторые отличия в реализации транзакций, индексов и других механизмов.

2.4.3 Вывод

PostgreSQL был выбран, потому что он предоставляет мощные возможности для работы с данными. Наличие pgAdmin значительно улучшает опыт работы с PostgreSQL, предоставляя удобный графический интерфейс для администрирования базы данных и упрощая процессы разработки и мониторинга данных.

2.5 Выбор среды разработки

2.5.1 WebStorm

WebStorm – это полнофункциональная интегрированная среда разработки (IDE) от JetBrains, специально разработанная для работы с JavaScript, TypeScript, Node.js, React, Vue.js и другими современными веб-технологиями. WebStorm является одним из самых популярных решений для веб-разработки благодаря множеству встроенных функций и высокому уровню интеграции с фреймворками.

В отличие от более легковесных редакторов, таких как VS Code, WebStorm предоставляет глубокое интеллектуальное автодополнение и рефакторинг кода. Он помогает разработчикам быстро писать чистый и оптимизированный код, автоматически предлагая улучшения.

Основным минусом этой среды является доступность. WebStorm является платным продуктом, и для использования всех его функций необходимо приобрести лицензию.

2.5.2 Visual Studio Code

Visual Studio Code (VS Code) – это легкий, но мощный редактор исходного кода с открытым исходным кодом, разработанный Microsoft. Он поддерживает работу с множеством языков программирования, таких как JavaScript, TypeScript, Python, PHP, C++ и многие другие.

VS Code очень легковесен и быстро запускается, что делает его отличным выбором для проектов с различными требованиями по производительности. Он

не перегружает систему и имеет малый размер по сравнению с полноценными IDE.

VS Code имеет огромную коллекцию расширений для множества технологий и инструментов. Вы можете легко добавить поддержку для новых языков, фреймворков, линтеров и других инструментов разработки.

2.5.3 Вывод

VS Code был выбран благодаря своей доступности, легковесности, скорости и широкой кастомизации через расширения, что позволяет настроить его под любые нужды и технологии. Он обладает мощными возможностями для работы с JavaScript и другими языками, при этом не перегружает систему.

2.6 Выбор паттерна проектирования архитектуры

Для разработки любого приложения необходимо выбрать подходящий архитектурный паттерн. Выбор паттерна связан с выбором фреймворка и технологий, которые будут использоваться в разработке. MVC, MVP и MVVM – это архитектурные паттерны, которые помогают организовать приложение и разделить ответственность между различными слоями.

2.6.1 MVC (Model-View-Controller)

MVC – один из самых популярных архитектурных паттернов, которые используются для взаимодействия между пользователем и системой. В составные части этого паттерна входят:

- Model – слой, который отвечает за управление данными, она не зависит от интерфейса и отображает данные, с которыми будет работать приложение. В Node.js представлена системой баз данных;

- View – слой, который отвечает за отображение данных для пользователя, отвечает за визуальные изменения. В React представлением являются компоненты, которые получают данные и отображают их на экране. Оно лишь отображает данные и реагирует на действия пользователя;

– Controller – слой-посредник между моделью и представлением. Он принимает запросы от пользователя, обрабатывает их, при необходимости взаимодействует с моделью и обновляет представление. В Node.js контроллеры обрабатывают HTTP-запросы и отправляют данные на клиентскую сторону. Они обращаются к модели, получают или изменяют данные и передают их в представление. В React контроллер может быть компонентом, который управляет состоянием приложения, выполняет запросы к серверу и обновляет интерфейс.

Преимущества MVC:

- разделение ответственности между компонентами;
- упрощает тестирование и поддержку кода;
- четкое разграничение логики, данных и отображения.

2.6.2 MVP (Model-View-Presenter)

MVP – это паттерн схожий с MVC, но с ключевым отличием: вместо контроллера логикой управляет Presenter. В составные части паттерна входят:

– Model – слой, который отвечает за управление данными, она не зависит от интерфейса и отображает данные, с которыми будет работать приложение. В Node.js представлена системой баз данных;

– View – слой, который отвечает за отображение данных для пользователя, отвечает за визуальные изменения. В React представлением являются компоненты, которые получают данные и отображают их на экране. Оно лишь отображает данные и реагирует на действия пользователя;

– Presenter – слой, который управляет логикой и обеспечивает связь между двумя предыдущими слоями. Он не только управляет запросами, но и обрабатывает их, обновляет модель, обрабатывает данные и передает их для отображения. В React это может быть компонент, который получает данные от модели (через API) и передает их во View.

Преимущества MVP:

- хорошо разделяет логику и представление;
- легче тестировать презентер, так как он не зависит от представления;
- упрощает расширение логики приложения.

2.6.3 MVVM (Model-View-ViewModel)

MVVM – паттерн, созданный для улучшения разделения ответственности в приложениях с богатым пользовательским интерфейсом. Это паттерн, который используется во многих фреймворках, таких как Angular и KnockoutJS, а также в приложениях с React. В составные части паттерна входят:

- Model - слой, который отвечает за управление данными, она не зависит от интерфейса и отображает данные, с которыми будет работать приложение. В Node.js представлена системой баз данных;

- View - слой, который отвечает за отображение данных для пользователя, отвечает за визуальные изменения. В React представлением являются компоненты. Представление подписано на изменения ViewModel, и при изменении данных, оно автоматически обновляется;

- ViewModel – слой, который связывает модель и представление. Он управляет состоянием, обрабатывает логику и передает данные в представление. В React это может быть компонент, который управляет состоянием с помощью useState или useReducer, или использование Redux для централизованного состояния.

Преимущества MVVM:

- отлично подходит для приложений с динамическим интерфейсом и частыми изменениями данных;
- хорошо работает с двусторонним связыванием данных (например, через React State или Redux);
- упрощает тестирование логики и состояния интерфейса.

Наглядное представление взаимодействия компонентов друг с другом в каждом шаблоне проектирования отражено на рисунке 2.1.

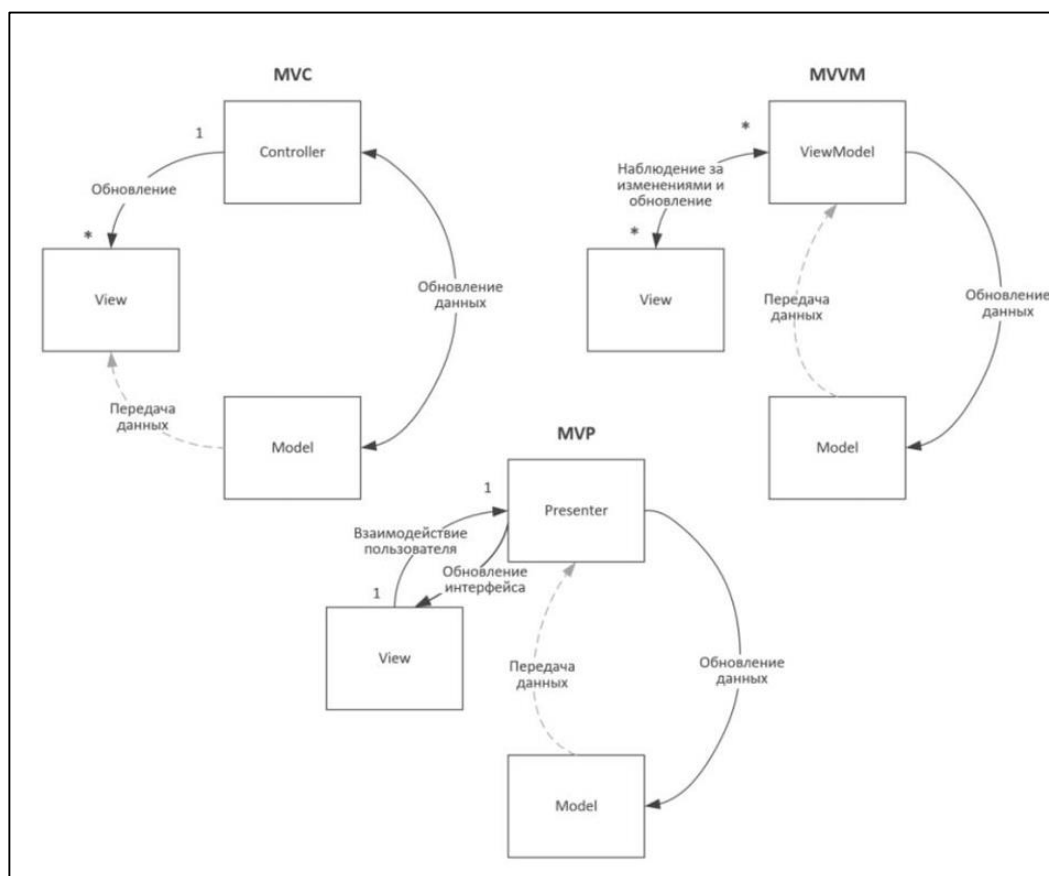


Рисунок 2.1 - Схема взаимодействия компонентов в архитектурных шаблонах проектирования

2.6.4 Вывод

Выбор паттерна MVC (Model-View-Controller) для разработки приложения с использованием React на клиентской стороне и Node.js + Express.js на серверной - это обоснованный и эффективный подход, который предоставляет несколько важных преимуществ для организации кода и упрощения процесса разработки.

В серверной части Node.js модель отвечает за работу с базой данных, бизнес-логику и обработку данных. Взаимодействие с базой данных выполняется через Sequelize, обеспечивающий доступ и модификацию данных.

На клиентской стороне React компоненты выполняют роль представления. Они рендерят данные, полученные от сервера, и отображают их пользователю.

В Node.js + Express.js контроллеры принимают запросы от клиента, обрабатывают их, выполняют бизнес-логику и взаимодействуют с моделью для получения или изменения данных. После этого контроллеры передают данные в представление (через JSON-ответы на клиентскую сторону), что позволяет динамично обновлять UI в React.

UML-диаграмма компонентов приложения отражена на рисунке 2.2

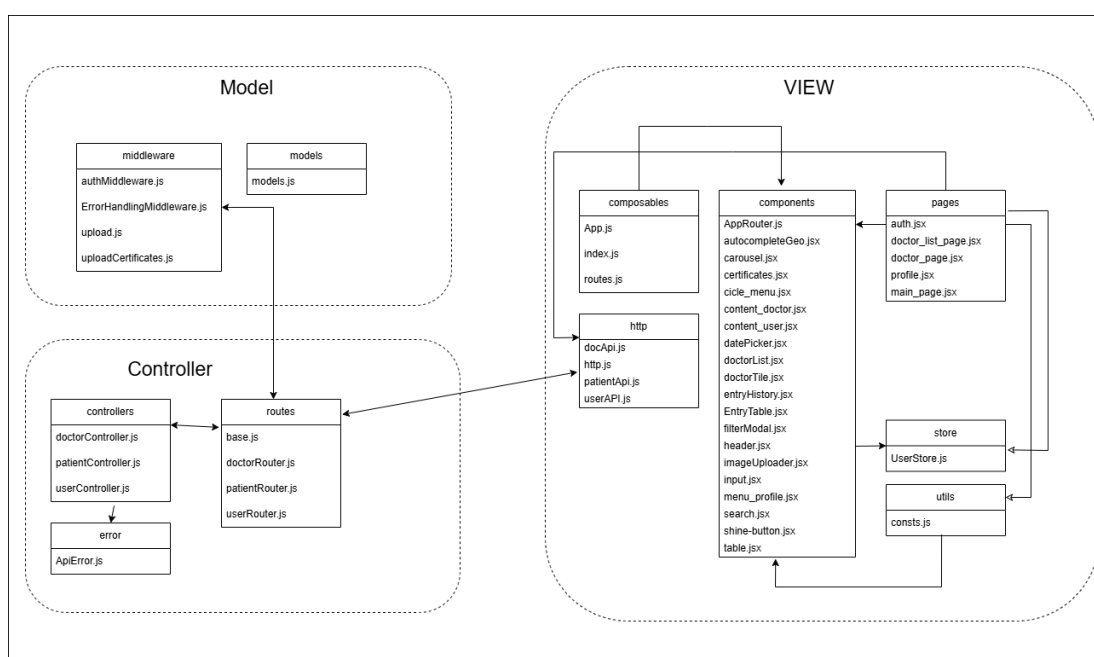


Рисунок 2.2 - UML-диаграмма компонентов приложения

2.7 Вывод

Таким образом в ходе анализа были выбраны следующие технологии:

- язык программирования JavaScript;
- фреймворк React;
- фреймворк Express на платформе Node.js;
- СУБД PostgreSQL;
- среда разработки Visual Studio Code;
- паттерн MVC;

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

3 Разработка

DocFinder состоит из серверного и клиентского приложения. Серверная часть обрабатывает запросы от клиента, выполняет работу с базой данных. Работу клиента выполняет браузер.

Приложение позволяет пользователю: завести аккаунт на сайте, просматривать карточки врачей, производить поиск с фильтрами по списку врачей или по карте, записываться на прием, оставлять отзывы. Пользователи с ролью «врач» могут формировать таблицы приема, загружать сертификаты, просматривать отзывы пациентов. Пользователи с ролью «администратор» могут удалять отзывы, менять роль пользователю, присваивать врачам специализации, стаж и описание.

3.1 Разработка VPS-сервера

Для развертывания серверной части приложения был выбран виртуальный частный сервер (VPS) как наиболее подходящее решение с точки зрения гибкости, контроля и масштабируемости. В качестве хостинг-провайдера использован сервис Jino, предоставляющий стабильную инфраструктуру и удобную панель управления.

На основании анализа требований приложения к ресурсам вычислительной среды был выбран тариф «Бета», который включает: 2 виртуальных процессорных ядра, 4 ГБ оперативной памяти, 20 ГБ дискового пространства (рисунок 3.1).

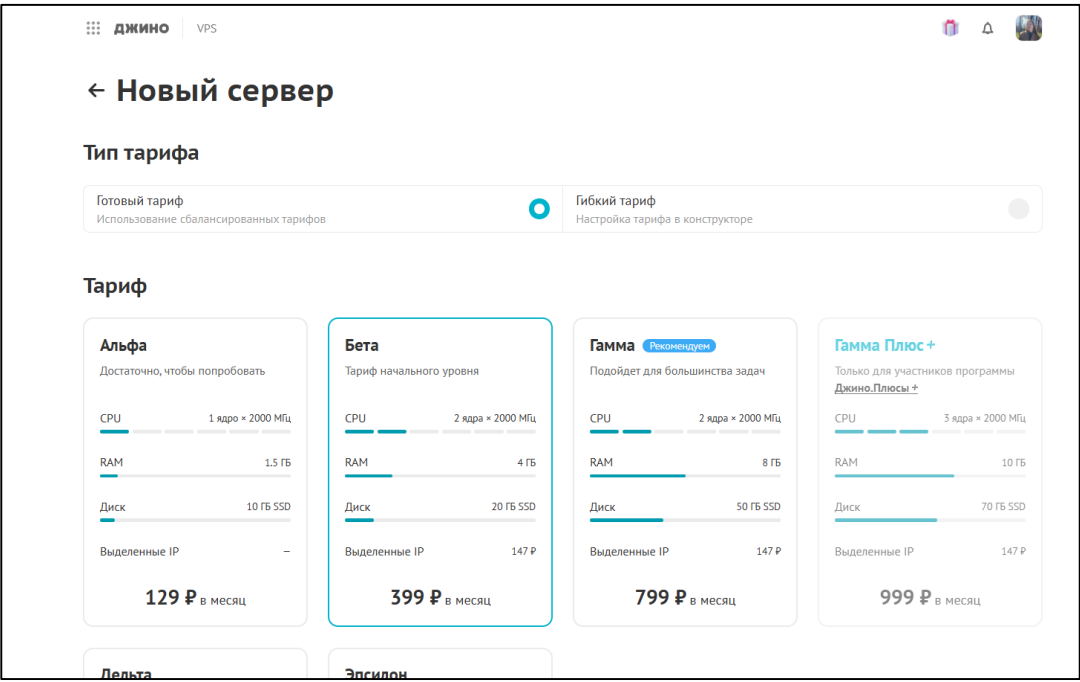


Рисунок 3.1 – Тарифы VPS джино

Данный тариф обеспечивает достаточный запас по ресурсам как для разработки и тестирования, так и для запуска приложения в условиях умеренной нагрузки.

Для развёртывания серверной инфраструктуры использовался следующий стек программного обеспечения:

- ОС: Ubuntu 24.04 LTS;
- сервер приложений: Node.js 22.x;
- СУБД: PostgreSQL 16;
- веб-сервер: Nginx (stable-версия).

Выбранный стек тарифа представлен на рисунке 3.2.

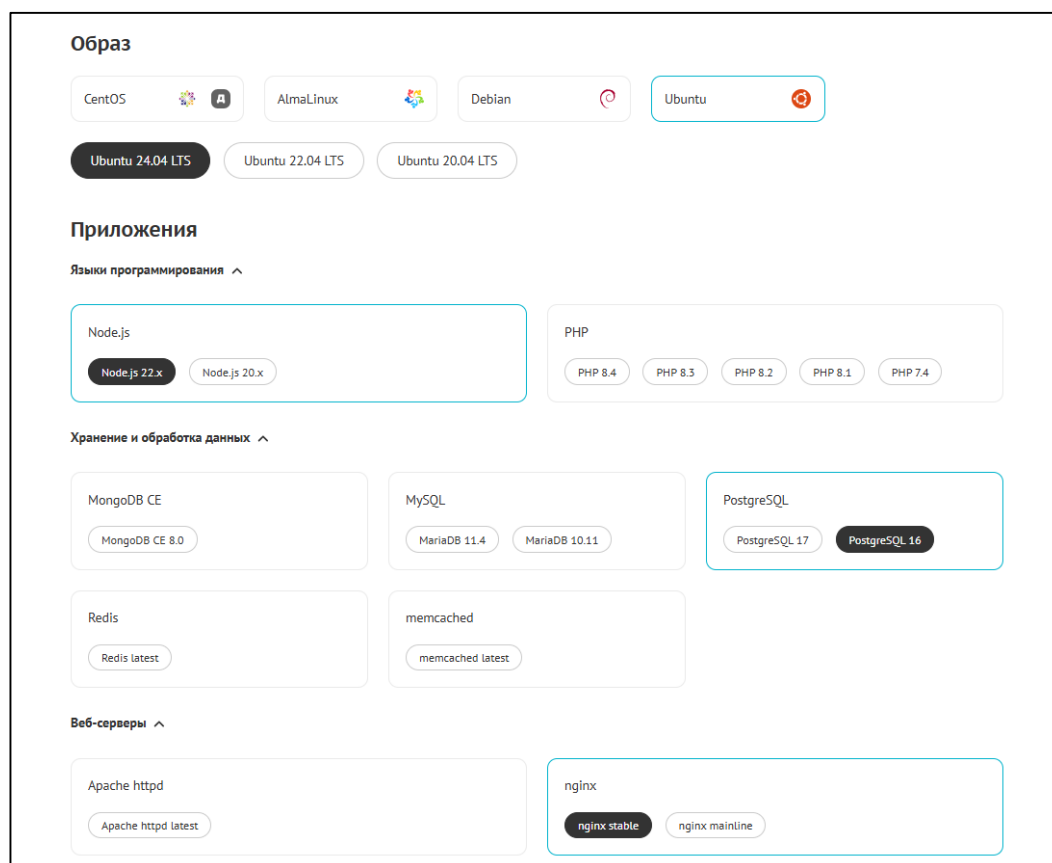


Рисунок 3.2 – Стек программного обеспечения

После установки официального образа Ubuntu была выполнена настройка необходимых компонентов. Веб-сервер Nginx настроен в качестве обратного прокси, маршрутизирующего HTTP-запросы от клиента к серверному приложению на Node.js и к статическим ресурсам. PostgreSQL используется в качестве основной СУБД, обеспечивая надёжное хранение и обработку данных.

Взаимодействие с сервером осуществляется с помощью двух инструментов:

- PuTTY – для подключения по SSH и работы с командной строкой;
- WinSCP – для безопасной передачи файлов по протоколу SFTP.

Выбранная конфигурация обеспечивает стабильную и масштабируемую среду для функционирования веб-приложения в рамках дипломного проекта.

3.2 Разработка серверной части приложения

Серверная часть приложения разрабатывается на Node.js с использованием фреймворка Express.js. Была создана папка для проекта, который проинициализируем командой «`npm init -y`». Для разработки нам потребуется установить фреймворк «`npm install express`», он поможет в описании обработчиков для запросов и установки общих параметров веб-приложения.

После этого добавим несколько библиотек, которые помогут нам в взаимодействии с базой данных и данными:

- `bcrypt` - библиотека для хеширования паролей с использованием алгоритма `bcrypt`. Это помогает безопасно хранить пароли в базе данных, предотвращая их утечку, даже если база данных будет скомпрометирована;

- `nodemailer` - библиотека для отправки электронных писем с использованием различных сервисов почты (например, SMTP);

- `pg` и `pg-hstore` - библиотеки для работы с PostgreSQL в Node.js, сериализации и десериализации объектов в формат JSON для PostgreSQL;

- `sequelize` - ORM (Object-Relational Mapping) для Node.js, который поддерживает работу с базами данных, такими как PostgreSQL, MySQL и SQLite.

Все настройки приложения задаются в конфигурационном файле «`.env`». Ниже приведена часть файла `.env`:

```
PORT = 5000
DB_NAME = health-point
DB_USER = postgres
DB_PASSWORD = 8698
DB_HOST = localhost
DB_PORT = 5432
SECRET_KEY = *****
MAIL_PSW = *****
```

DB_NAME, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT используются для работы с приложения с базой данных. SECRET_KEY – ключ, используемый для хэширования пароля. MAIL_PSW пароль от почты, с которой отправляет письма сервер.

В Node.js с Express.js структура проекта может быть следующей:

- Model – представляет собой файлы моделей, которые взаимодействуют с базой данных (например, используя Sequelize, Mongoose для MongoDB, или стандартные SQL-запросы);

- Controller – содержит логику маршрутов, обработки запросов и взаимодействия с моделями;

- View – в Express.js обычно не реализуется напрямую, так как сервер предоставляет данные клиенту. В случае использования React для интерфейса, представление (View) обрабатывается на клиентской стороне.

В приложении на Node.js с Express.js сервер выполняет роль API, предоставляющего данные в виде JSON, и управляет бизнес-логикой и взаимодействием с базой данных. Клиентская часть является отдельным приложением и реализована с использованием React, занимается рендерингом данных, полученных от сервера, и отображением их пользователю.

Входной точкой в backend приложение являются запросы от frontend приложения. Запросы обрабатываются механизмом маршрутизации Express.js.

Для определения маршрутов к API служат несколько файлов: routes/base.js, routes/userRouter.js, routes/doctorRouter.js, routes/patientRouter.js. Для того, чтобы задать маршрут импортируем Express в файл, создания маршрута используется метод Route с указанием HTTP-метода (get, post, put, delete и т.д.) и пути к маршруту. Например, router.post('/registration', userController.registration) обрабатывает post-запрос и задает путь к контроллеру userController и его методу registration.

Контроллеры в Express являются функциями, которые обрабатывают HTTP-запросы (например, GET, POST, PUT, DELETE). Они получают данные

из запроса, выполняют необходимую логику (например, взаимодействие с базой данных через модели) и отправляют ответ обратно пользователю.

Контроллеры обычно определяются в отдельных модулях и экспортируются для использования в других частях приложения и могут содержать:

- обработчики запросов – это функции, которые принимают запросы, обрабатывают их и отправляют ответы;
- взаимодействие с моделями – контроллеры часто взаимодействуют с моделями для получения данных из базы данных или их модификации;
- ответы – контроллеры формируют ответы, которые отправляются пользователю. Это может быть HTML-страница, JSON-данные или другой тип данных.

Как пример обработки POST-запроса к серверу является контроллер `UserController`, в котором находятся функции, реализующие логику первичной двухфакторной верификации. Алгоритм работы функции проверки временного кода состоит из нескольких шагов, начиная от генерации кода и заканчивая его валидацией при регистрации.

В начале процесса, после получения данных о пользователе, система генерирует шестизначный код подтверждения. Этот код уникален для каждого пользователя и привязывается к его адресу электронной почты. После этого устанавливается время, в течение которого код будет действителен – обычно это 10 минут. Такая временная привязка предотвращает использование устаревших или поддельных кодов.

```
function generateVerificationCode() {  
  return Math.floor(100000 + Math.random() * 900000); //  
  Генерирует случайный код из 6 цифр  
}  
const verificationCode = generateVerificationCode();  
  
// Устанавливаем дату истечения срока действия кода (например,  
10 минут)  
const expirationDate = new Date();
```

```

        expirationDate.setMinutes(expirationDate.getMinutes() + 10);
// Код действителен 10 минут

// Сохраняем код в таблице VerificationCode
await VerificationCode.create({
    email,
    verificationCode,
    expirationDate
});

```

Затем с помощью SMTP-сервера отправляется письмо на указанный пользователем адрес электронной почты. В письме содержится сгенерированный код подтверждения, который пользователь должен ввести на следующем шаге. Это подтверждает, что владелец указанного адреса действительно начал процесс регистрации.

Если введенный пользователем код действителен и его срок действия не истек, система приступает к хешированию пароля пользователя, после чего создается запись в базе данных с указанным email, хешированным паролем и ролью пользователя. После успешной регистрации генерируется JWT-токен, который предоставляет пользователю доступ к защищенным ресурсам системы. Этот токен отправляется обратно на клиент и процесс регистрации считается завершенным.

```

const hashPassword = await bcrypt.hash(password, 5);
const user = await User.create({ email, role, password:
hashPassword });
const patient = await Patient.create({ userId: user.id });
// Создаем JWT токен
const token = generateJwt(user.id, user.email, user.role);

// Удаляем код из временной таблицы
await VerificationCode.destroy({ where: { email,
verificationCode } });

```

Для обеспечения безопасности и предотвращения повторного использования того же кода, временный код удаляется из базы данных. Это гарантирует, что каждый код может быть использован только один раз в процессе регистрации.

Для дополнительной обработки запроса до того, как он достигнет контроллера используется middleware. Они выполняются в цепочке и могут изменять запрос, ответ, завершить обработку запроса или передавать управление следующей функции в цепочке.

Middleware в Express – это функции, которые имеют доступ к объектам запроса (req), ответа (res) и следующей функции в цикле обработки запроса (next). Эти функции могут выполнять различные задачи, такие как:

- обработка ошибок;
- логирование запросов;
- аутентификация и авторизация;
- разбор данных (JSON или формы);
- валидация данных;
- подключение к базе данных.

Одним из примеров использования является middleware upload.js, он используется для обработки файлов, отправляемых на сервер, в частности – для загрузки фотографий.

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    const uploadPath = path.resolve(__dirname, '..', 'static',
'doctor_photos');
    if (!fs.existsSync(uploadPath)) {
      fs.mkdirSync(uploadPath, { recursive: true });
    }
    cb(null, uploadPath); },
  filename: function (req, file, cb) {
    const ext = path.extname(file.originalname); // сохраняем
оригинальное расширение
    const userId = req.user.id;
    cb(null, `${userId}${ext}`); // например: 42.jpg
  }
});
const upload = multer({ storage });
```

Алгоритм работы этой функции начинается с конфигурации хранилища, которое отвечает за путь и имя файлов, сохраняемых на сервере. Первым шагом

проверяется наличие необходимой директории для хранения файлов. Если папка для фотографий еще не создана, система автоматически создает её с использованием рекурсивного подхода, чтобы гарантировать наличие всех необходимых подкаталогов. Когда файл поступает на сервер, middleware извлекает расширение загружаемого файла. Это позволяет сохранить файл с его оригинальным расширением, что важно для корректной работы с файлами различных типов (например, .jpg, .png и т. д.). После того как файл сохранен в соответствующей директории, middleware передает управление следующему этапу обработки запроса или завершает его, в зависимости от дальнейшей логики приложения.

На рисунке 3.3 можно просмотреть различные запросы, их маршрутизацию, и какие контроллеры и модели взаимодействуют.

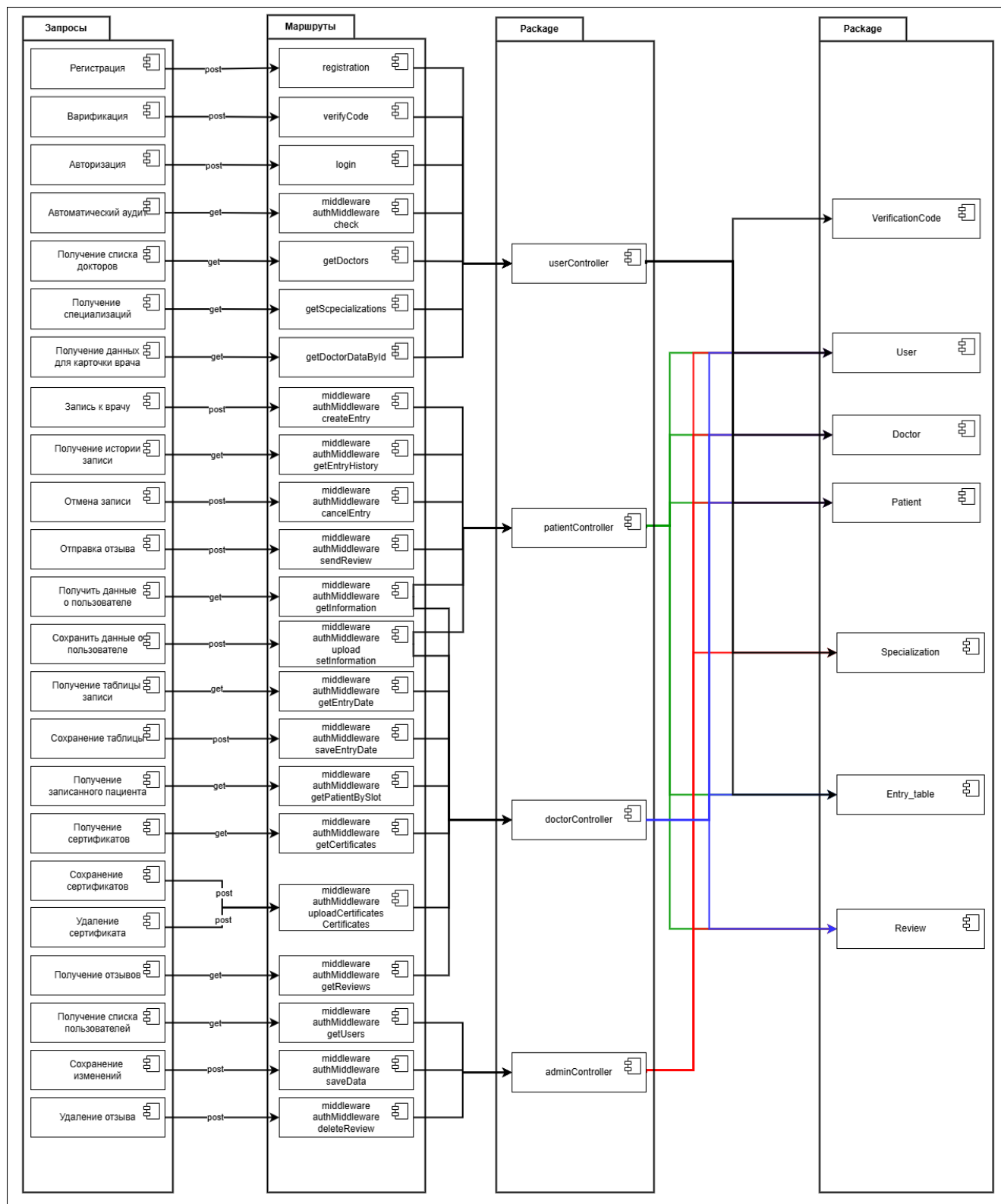


Рисунок 3.3 – UML-диаграмма компонентов

3.3 Разработка базы данных

Благодаря библиотеке `sequelize`, которая предоставляет способы взаимодействия с реляционными базами данных через JavaScript, мы можем работать с базой данных используя JavaScript-объекты и методы, а не писать SQL-запросы вручную. `Sequelize` позволяет определять модели, которые представляют таблицы базы данных. Каждая модель связана с конкретной таблицей и имеет свои атрибуты (поля). Библиотека поддерживает ассоциации между моделями, такие как один к одному, один ко многим и многие ко многим. Модели определяются с помощью метода `define()` или с использованием классов. Вот пример модели `User`:

```
const sequelize = require("../db");
const { DataTypes } = require("sequelize");

const User = sequelize.define("user", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  email: { type: DataTypes.STRING, unique: true },
  password: { type: DataTypes.STRING },
  role: { type: DataTypes.STRING, defaultValue: "USER" },
});
```

Описываются поля сущности, относящиеся к модели `User`, при запуске вызываем метод `sequelize.sync()`, он проверяет, существуют ли таблицы для всех определенных моделей, и если нет – создает их.

Миграции позволяют управлять изменениями в базе данных. С помощью `Sequelize CLI` можно создавать миграции, которые будут автоматически применяться, для этого нужно применить команду в терминале `npm run sequelize-cli migration:generate --name create-users`. В файле миграции нужно описать метод `up`, используемый для применения миграции, и метод `down` для отката миграции. Пример такого файла ниже:

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('Users', {
      id: {
```

```

    type: Sequelize.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  name: {
    type: Sequelize.STRING
  },
  email: {
    type: Sequelize.STRING,
    unique: true
  }
});
},
down: async (queryInterface, Sequelize) => {
  await queryInterface.dropTable('Users');
}
};

```

На рисунке 3.4 представлена физическая модель базы данных.

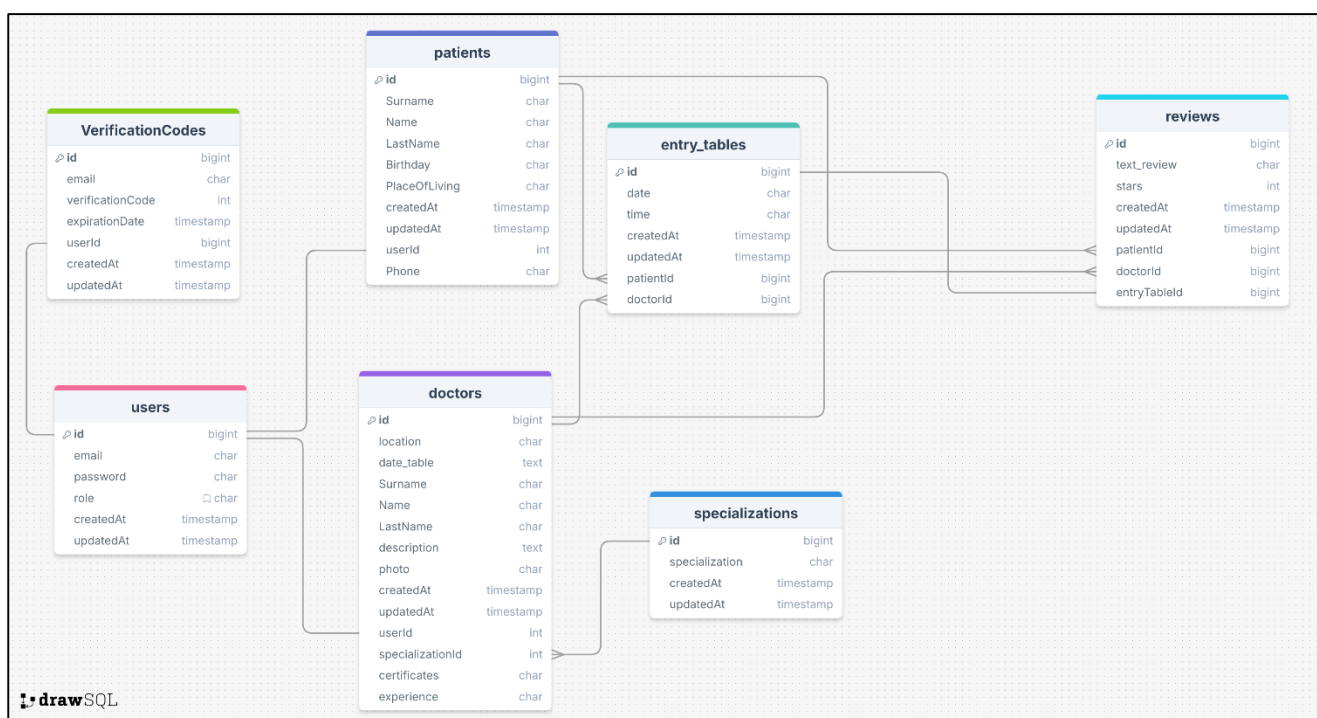
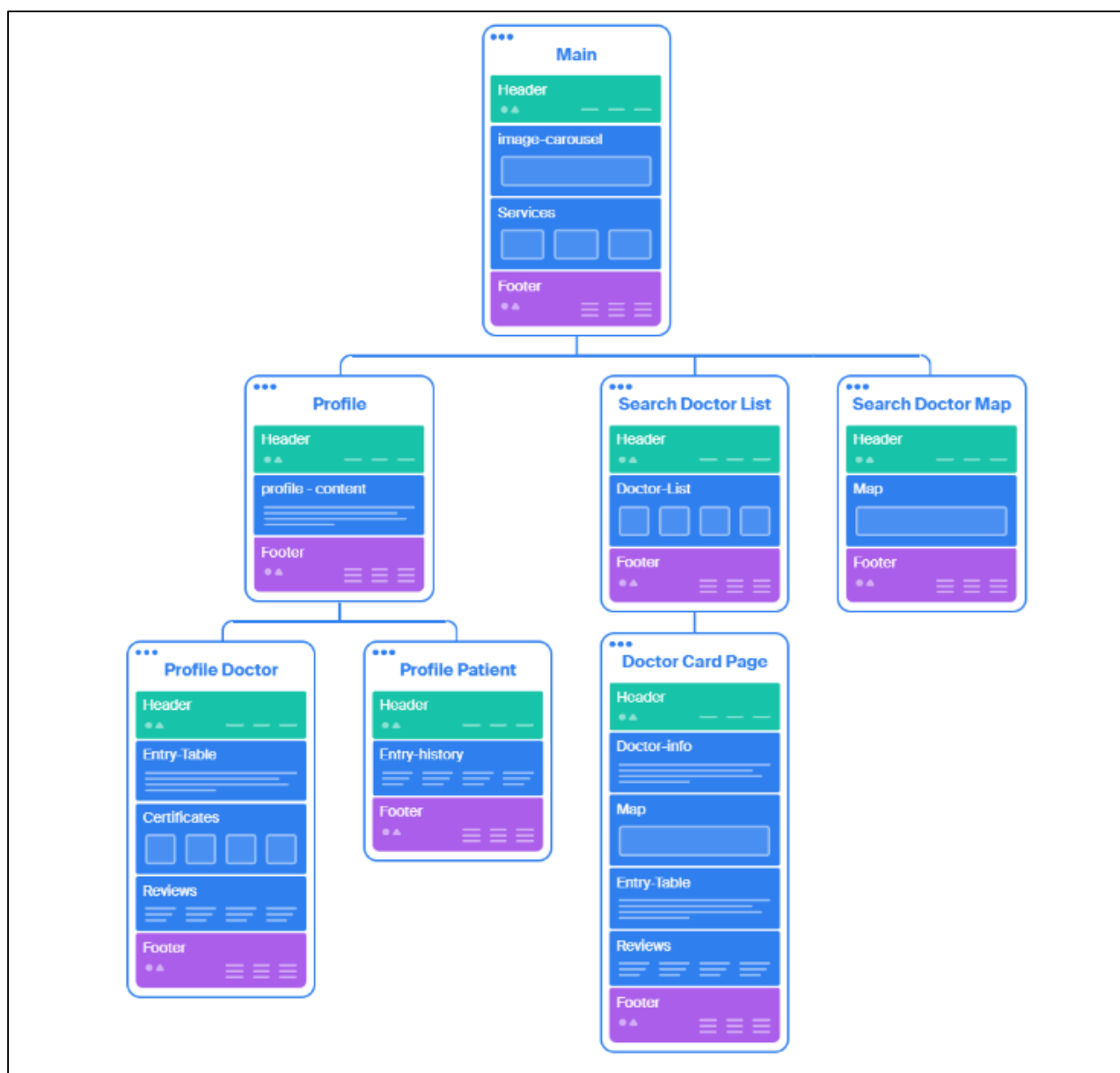


Рисунок 3.4 - Физическая модель базы данных

3.4 Разработка клиентской части приложения

Разработку приложения начнем с планирования структуры и прототипирования сайта. Для разметки структуры приложения воспользуемся сервисом octorpus.do, он поможет сделать набросок структуры будущего сайта

перед тем, как приступить к прототипированию и разработке (рисунок 3.5). Было решено сделать такие страницы: основная страница, демонстрирующая общий функционал сайта и навигацию по другим страницам, страница авторизации, профиль пользователя (имеет разную структуру в зависимости от роли пользователя), страница с списком докторов, поиском и фильтрацией, страница с поиском по карте, а также карточка доктора с информацией и таблицей записи.



3.5 - Структура клиентской части приложения

Для создания прототипа нашего сайта воспользуемся приложением Axure RP (рисунок 3.6). Axure RP 9 – это мощный инструмент для прототипирования

Изм.	Лист	№ докум.	Подпись	Дата

и создания интерактивных веб и мобильных интерфейсов. Он предназначен для дизайнеров и разработчиков, чтобы быстро и эффективно разрабатывать прототипы, которые могут быть использованы как для тестирования, так и для демонстрации клиентам.

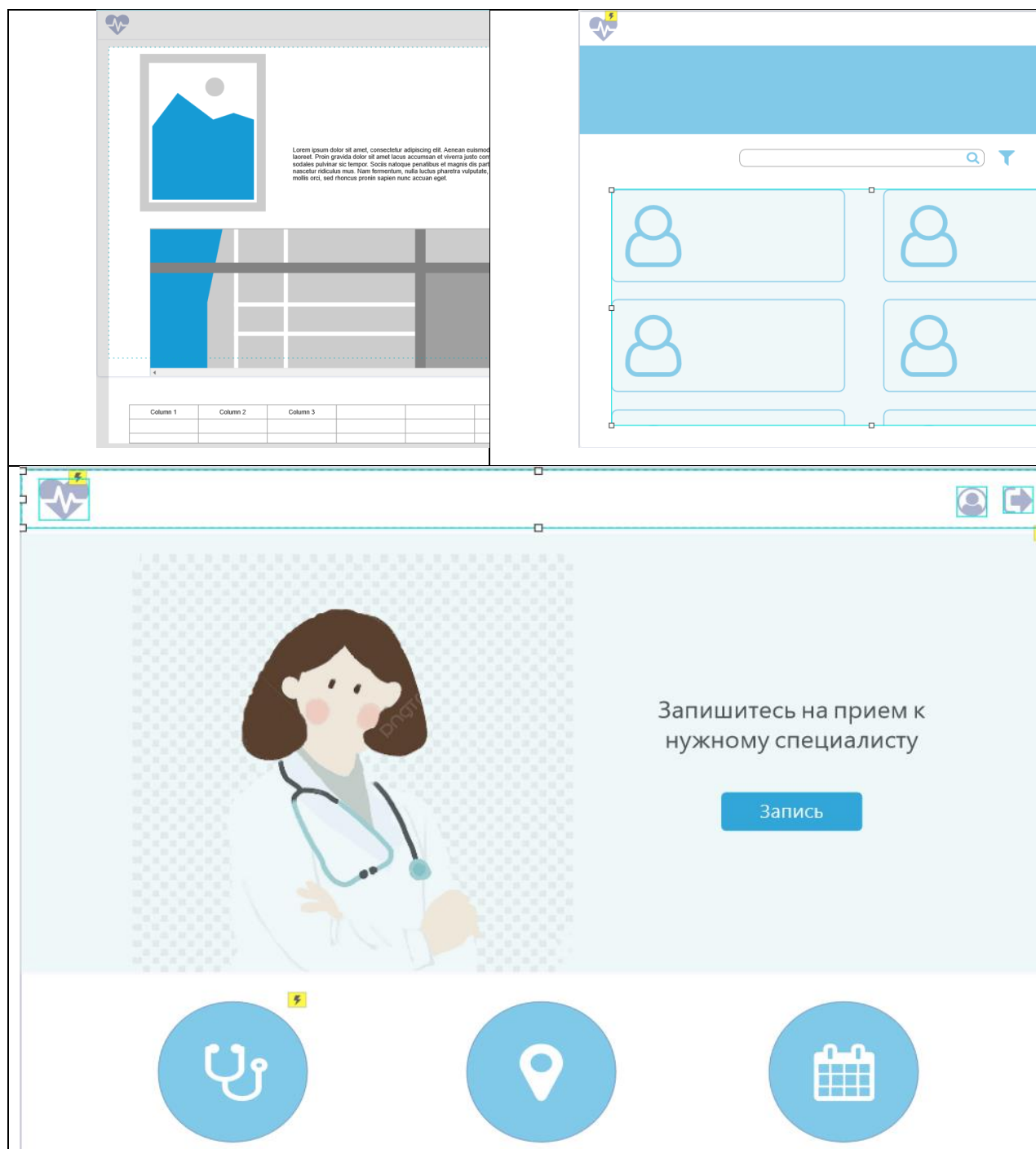


Рисунок 3.6 – Прототип приложения в Axure RP

Изм.	Лист	№ докум.	Подпись	Дата

Приступая к разработке, потребуется создать базовое приложение React с помощью команды create-react-app. Базовая страница приложения приведена на рисунке 3.7.

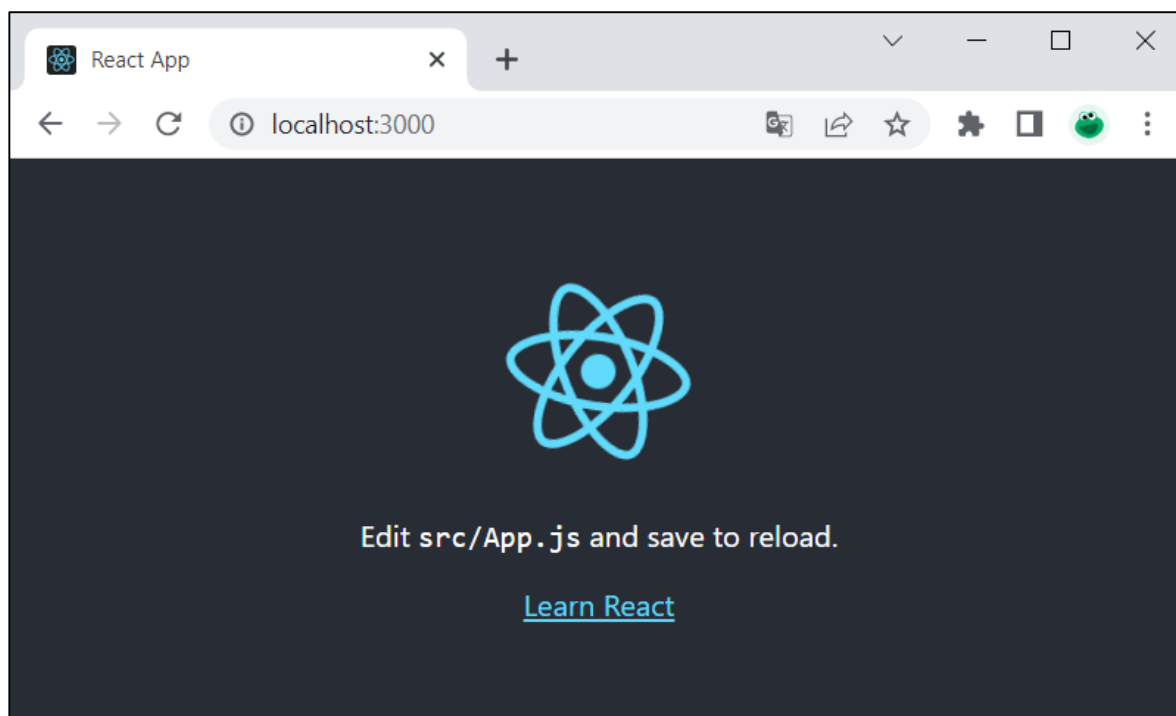


Рисунок 3.7 – Базовая страница React.

Так же потребуется установить несколько библиотек, которые потребуются при разработке этого приложения:

- Axios – это популярная библиотека для выполнения HTTP-запросов. Она поддерживает асинхронные запросы, работу с промисами и имеет удобный API для выполнения запросов к серверу (например, GET, POST, PUT, DELETE);

- MobX – это библиотека для управления состоянием в приложениях React. MobX использует концепцию "реактивных данных", автоматически отслеживая изменения в состоянии и обновляя компоненты, которые используют это состояние;

- React Router DOM – это библиотека для навигации между различными страницами (или компонентами) в приложении React. Она предоставляет компоненты и хук для управления маршрутами и переходами;

– @pbe/react-yandex-maps – это обертка для использования Yandex Maps API в React-приложениях. Библиотека позволяет интегрировать карты Яндекса в компоненты React и предоставляет удобные интерфейсы для работы с картами.

С помощью библиотеки React Router реализована маршрутизация, она позволяет определить маршруты, компоненты, которые будут отображаться при переходе по этому маршруту. Для удобства сначала заводим файл с константами, которые будем использовать при построении маршрута.

Маршрутизация для авторизованных и неавторизованных пользователей:

```
<BrowserRouter>
  <Routes>
    {user.isAuthenticated && authRoutes.map(({path, Component})=>
      <Route key={path} path={path} element={}<Component/>
    exact/>
    )}
    {publicRoutes.map(({path, Component})=>
      <Route key={path} path={path} element={}<Component/>
    exact/>
    )}
  </Routes>
</BrowserRouter>
```

Компоненты BrowserRouter и Routes являются основными для создания маршрутизации в приложениях React. Эти компоненты позволяют управлять навигацией между различными страницами или компонентами в вашем приложении. BrowserRouter позволяет браузеру изменять URL без перезагрузки страницы, сохраняя историю навигации и синхронизируя ее с состоянием приложения. Routes - это компонент, который определяет набор маршрутов приложения. Каждый маршрут внутри Routes связывает путь URL с конкретным компонентом, который должен быть отображен.

На главной странице используются компоненты: шапка страницы, карусель изображений, меню быстрого доступа к страницам, футер. На

начальном этапе пользователь проходит процесс регистрации или входа в систему. При успешной аутентификации сервер генерирует JSON Web Token (JWT), который содержит информацию о пользователе (ID, роль и email).

```
const generateJwt = (id, email, role) => {  
  return jwt.sign({ id: id, email, role }, process.env.SECRET_KEY,  
    {  
      expiresIn: "24h",  
    });  
};
```

Этот токен подписывается с использованием секретного ключа, что позволяет подтвердить его подлинность. Токен имеет срок действия, что добавляет дополнительный уровень безопасности.

После успешного получения токена, клиент сохраняет его в локальном хранилище (localStorage). Это позволяет сохранять информацию о пользователе даже после перезагрузки страницы. Когда клиент совершает запросы к защищенным ресурсам, он включает токен в заголовок Authorization каждого запроса. Сервер проверяет токен, расшифровывая его с использованием секретного ключа и проверяя его действительность. Если токен не просрочен и является подлинным, сервер предоставляет доступ к защищенным данным. В противном случае пользователь будет перенаправлен на страницу авторизации.

```
try{  
  const token = req.headers.authorization.split(' ')[1]  
  if(!token){  
    res.status(401).json({message:"Пользователь не  
авторизован"})  
  }  
  const decoded = jwt.verify(token, process.env.SECRET_KEY)  
  req.user = decoded  
  next()  
}catch(e){  
  res.status(401).json({message:"Пользователь не  
авторизован"})  
}
```

Для защиты от несанкционированного доступа используется промежуточное ПО (middleware), которое автоматически проверяет наличие и

валидность токена в заголовке каждого запроса. В случае его отсутствия или неверности сервер может отправить ошибку с кодом 401, уведомляя клиента о том, что доступ запрещен.

Для удобства и улучшения пользовательского опыта, в приложении реализована функция автоматической проверки наличия, действующего токена при загрузке страницы. Если токен существует и действителен, приложение автоматически авторизует пользователя, подставляя его данные и предоставляя доступ к личному кабинету. В случае истечения срока действия токена или его отсутствия, пользователь должен будет перейти на страницу входа, чтобы пройти повторную аутентификацию.

3.5 Анализ результатов

3.5.1 Тестирование API

Для тестирования работы API будет использоваться приложение Postman. Postman – одно из самых популярных и удобных приложений для тестирования API. Его удобство и эффективность подтверждаются несколькими яркими преимуществами, делающими работу с API максимально простой и продуктивной:

- Postman предлагает удобный графический интерфейс, который позволяет быстро создавать, отправлять и анализировать HTTP-запросы без необходимости писать сложный код;

- приложение поддерживает все популярные методы HTTP (GET, POST, PUT, DELETE и другие), позволяет работать с различными типами данных (JSON, XML, формы), поддерживает авторизацию, настройку заголовков и параметров;

- Postman позволяет группировать запросы в коллекции, что удобно для систематизации тестов по проектам или модулям.

На рисунке 3.8 представлен стартовый экран приложения.

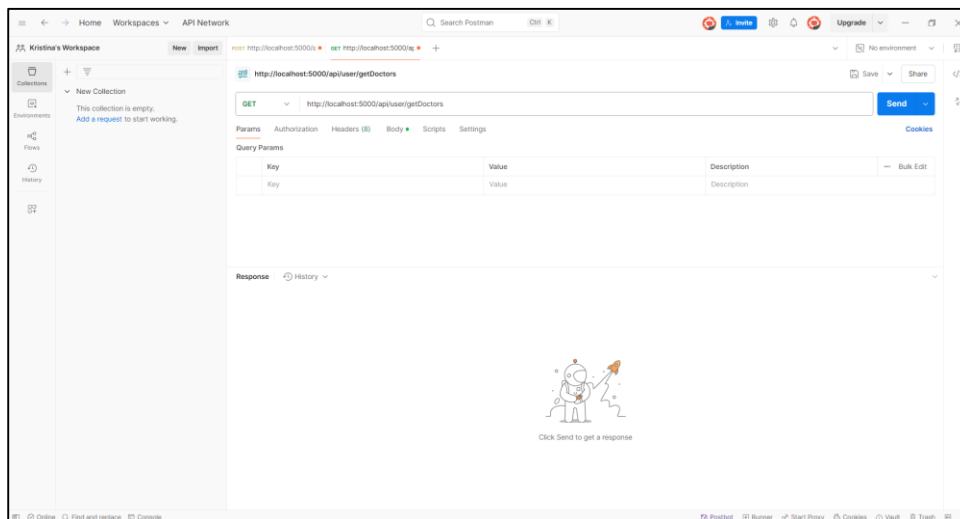


Рисунок 3.8 – Стартовый экран Postman

Для начала проведем тестирование запроса на получение списков врачей с информацией для страницы поиска и фильтрации. На рисунке 3.9 приведены результаты запроса.

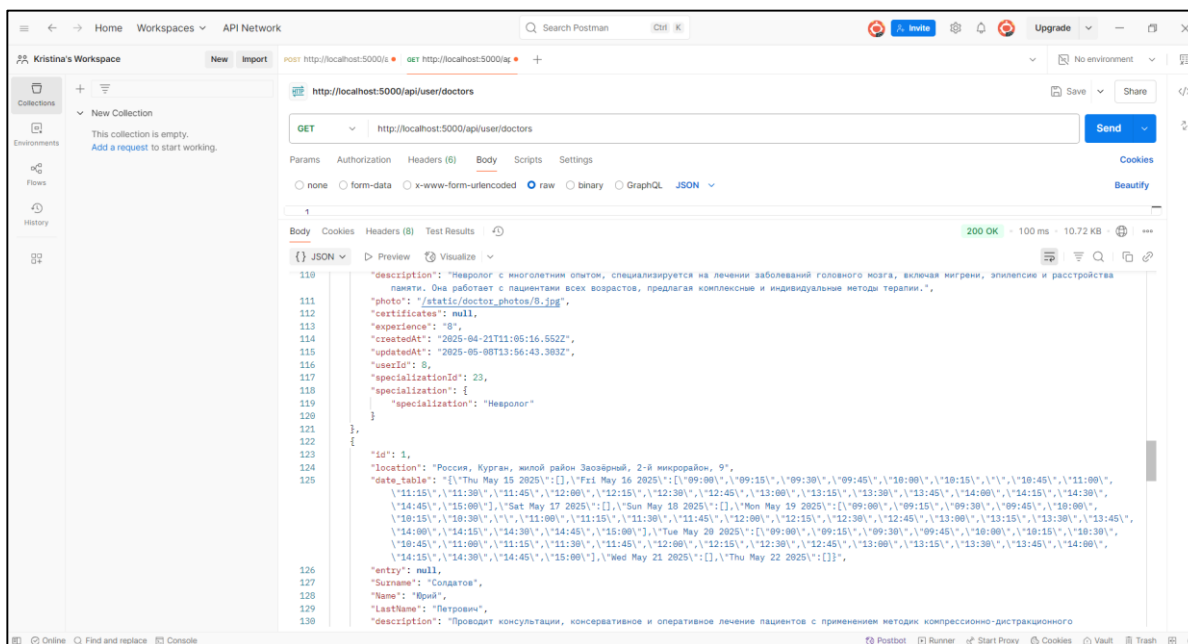


Рисунок 3.9 – Результат запроса на получение списка докторов

Теперь следует проверить работу входа в аккаунт. Для этого требуется прописать нужный маршрут и заполнить форму запроса (email, password). На рисунке 3.10 представлен результат запроса, возвращающий токен.

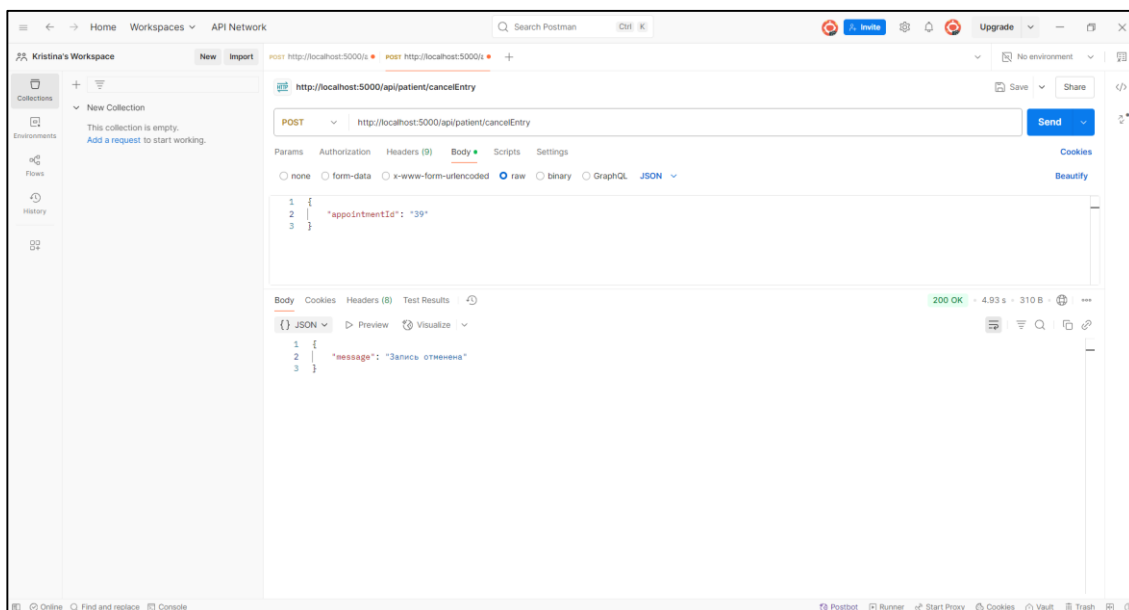


Рисунок 3.12 – Результат отмены записи Postman

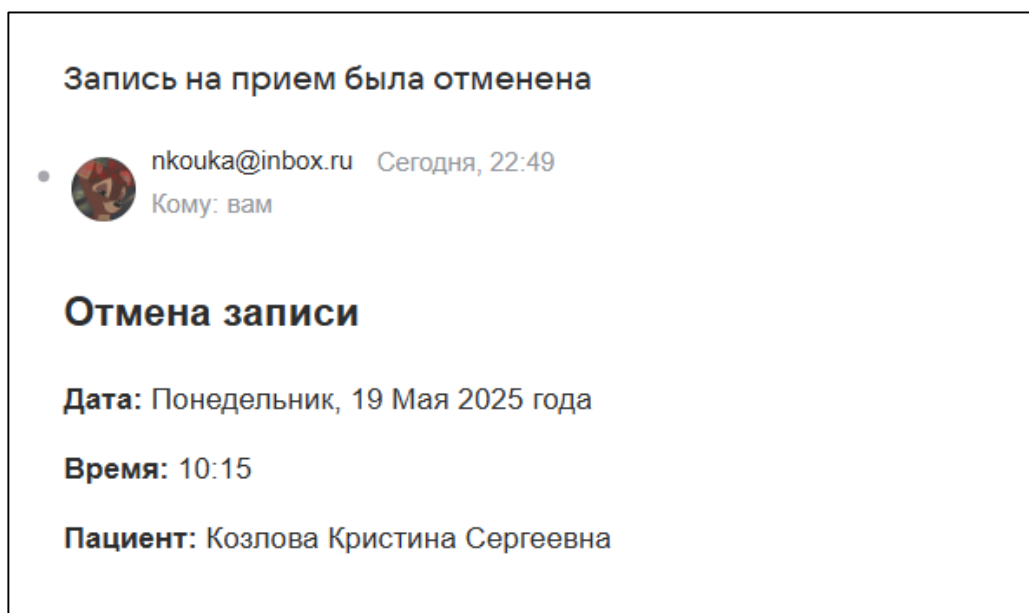


Рисунок 3.13 – Результат отмены записи на почте

3.5.2 Тестирование интерфейса

Цель данного этапа – обеспечить удобство, корректность и надежность работы всех элементов интерфейса, а также проверить их соответствие функциональным требованиям.

Для оценки корректности и стабильности работы ключевых функций приложения были проведены структурированные тесты, оформленные в виде таблиц. Такой формат обеспечивает наглядность – каждый шаг тестового

Изм.	Лист	№ докум.	Подпись	Дата

сценария и ожидаемый результат представлены чётко и однозначно. Это позволяет быстро определить, на каком этапе мог произойти сбой.

В таблице 1 содержатся тестовые сценарии, направленные на проверку корректности работы механизма авторизации пользователей. В тестах рассматриваются различные варианты ввода данных, обработка ошибок и успешный вход в систему.

Таблица 1 - Тестирование авторизации

Проверяется: авторизация					
№ теста	Суть теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ 1.1	Проверить ввод логина незарегистрированного пользователя	vgyugbi@mail.ru	Ввести данные в текстовое поле, нажать кнопку «Войти»	Выведется сообщение о том, что пользователь не был найден	Совпадает с ожидаемым
ПФ 1.2	Проверить неверный ввод пароля для зарегистрированного пользователя	doctor4@почта.ru doctor6	Ввести данные в текстовое поле, нажать кнопку «Войти»	Выведется сообщение о том, что пароль неверный	Совпадает с ожидаемым
ПФ 1.3	Проверка защиты входа от SQL-инъекции	' OR '1'='1	Ввести данные в текстовое поле, нажать кнопку «Войти»	Выведется сообщение о неверных данных	Совпадает с ожидаемым
ПФ 1.4	Проверить ввод корректных значений в оба поля	doctor4@почта.ru doctor4	Ввести данные в текстовое поле, нажать кнопку «Войти»	Переход в профиль пользователя	Совпадает с ожидаемым

Результаты тестирования можно увидеть на рисунках 3.14-3.16.

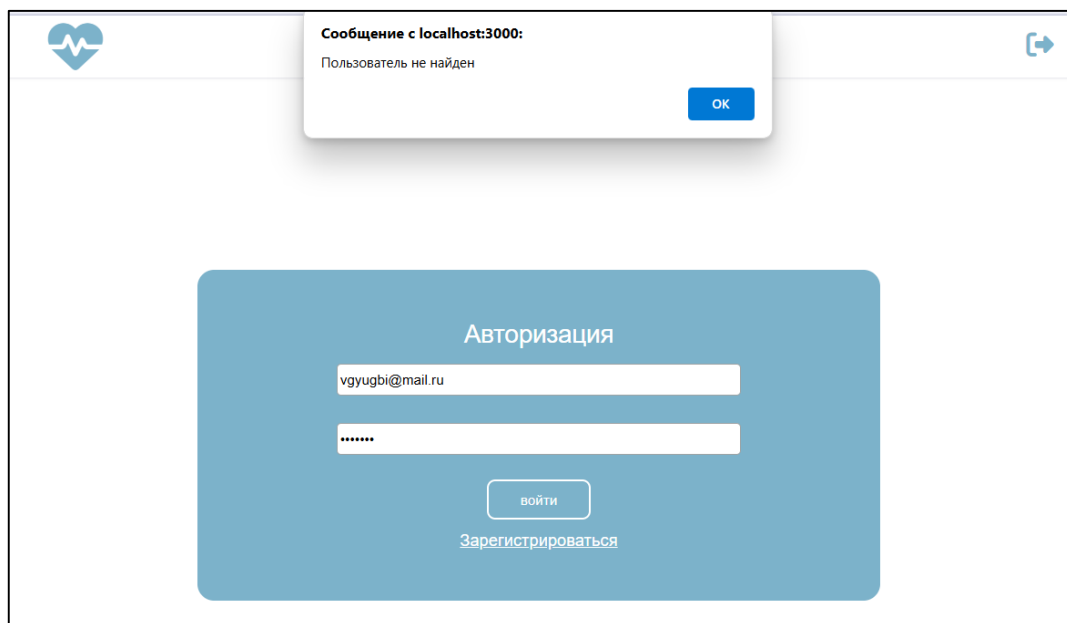


Рисунок 3.14 – ПФ 1.1

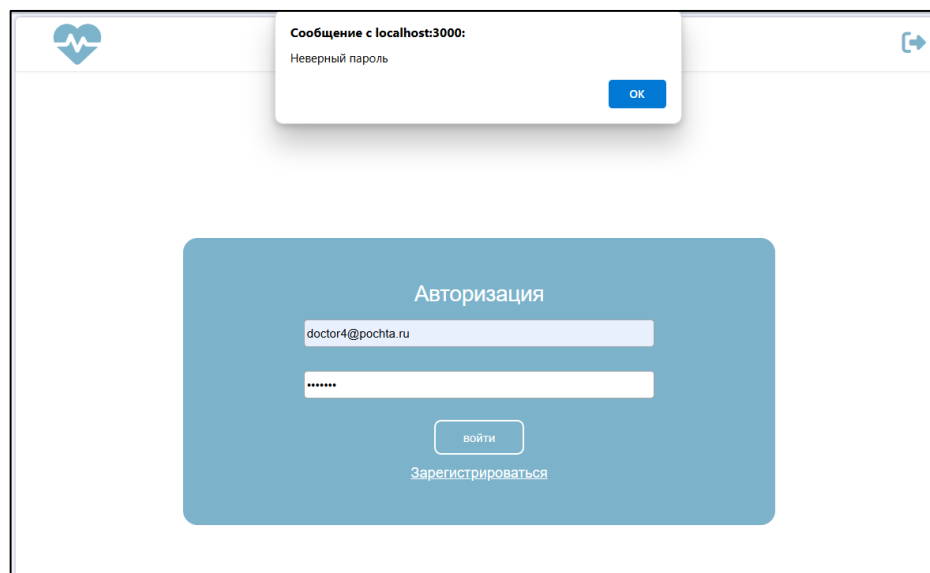


Рисунок 3.15 – ПФ 1.2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.11.001 ПЗ

Лист

53

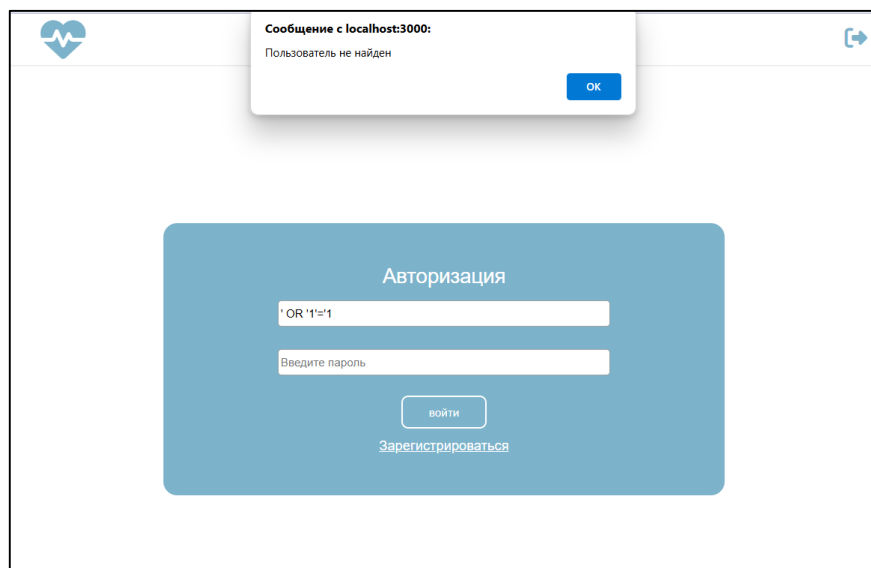


Рисунок 3.16 – ПФ 1.3

В таблице 2 представлены тесты, проверяющие процесс регистрации новых пользователей.

Таблица 2 - Тестирование регистрации

Проверяется: регистрация					
№ теста	Суть теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ 2.1	Проверить ввод значения, не содержащего знак @, в поле «Email»	doctor	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Выведется сообщение о некорректности почты	Совпадает с ожидаемым
ПФ 2.2	Проверить ввод неодинаковых значений в поля «Введите пароль» и «Повторите пароль»	kseroff180203@gmail.com 1111 1112	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Выведется сообщение о том, что пароли не совпадают	Совпадает с ожидаемым
ПФ 2.3	Проверить поведение программы при корректно заполненных полях	kseroff180203@gmail.com 1111 1111	Ввести данные в текстовое поле, нажать кнопку «Зарегистрироваться»	Выведется сообщение, что на почту был отправлен код подтверждения	Совпадает с ожидаемым

Проверяется: регистрация					
№ теста	Суть теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ 2.4	Проверить ввод неверного кода подтверждения почты	1111	Ввести данные в текстовое поле, нажать кнопку «Подтвердить»	Выведется сообщение, что код неверный	Совпадает с ожидаемым
ПФ 2.5	Проверить ввод верного кода подтверждения почты	455853	Ввести данные в текстовое поле, нажать кнопку «Подтвердить»	Выведется сообщение об успешной регистрации, переход в профиль пользователя	Совпадает с ожидаемым

Результаты тестирования можно увидеть на рисунках 3.17-3.21.

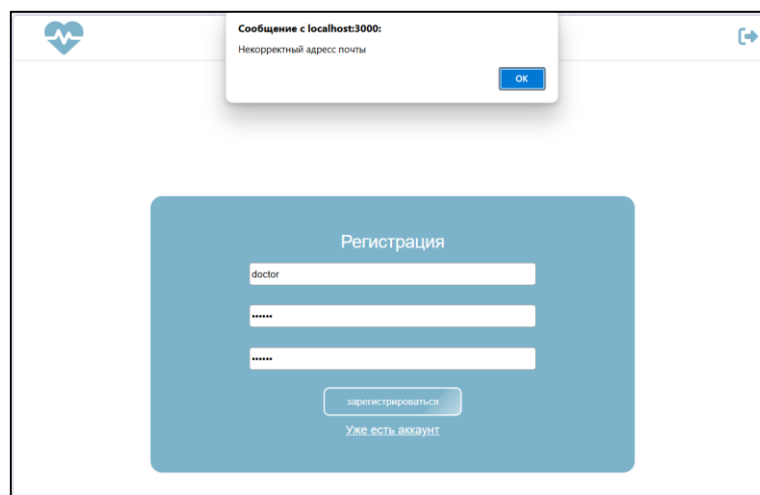


Рисунок 3.17 – ПФ 2.1

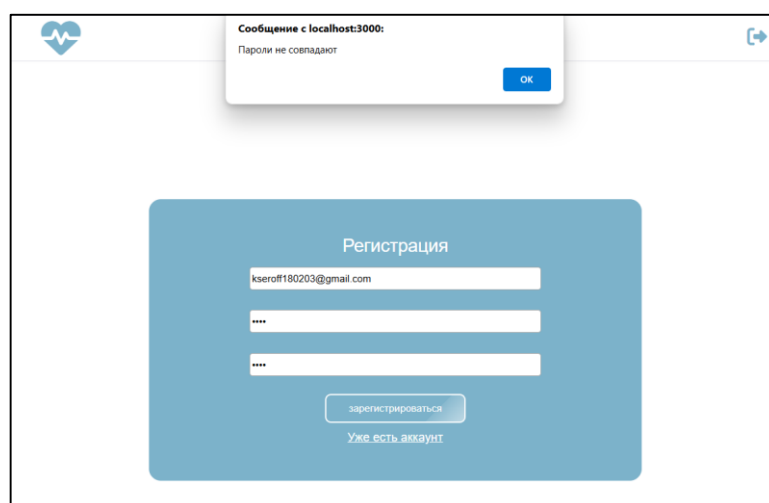


Рисунок 3.18 – ПФ 2.2

Изм.	Лист	№ докум.	Подпись	Дата

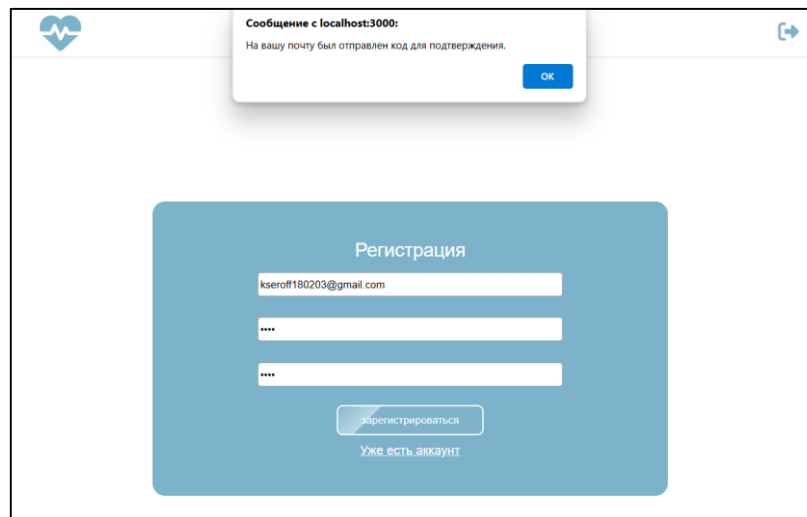


Рисунок 3.19 – ПФ 2.3

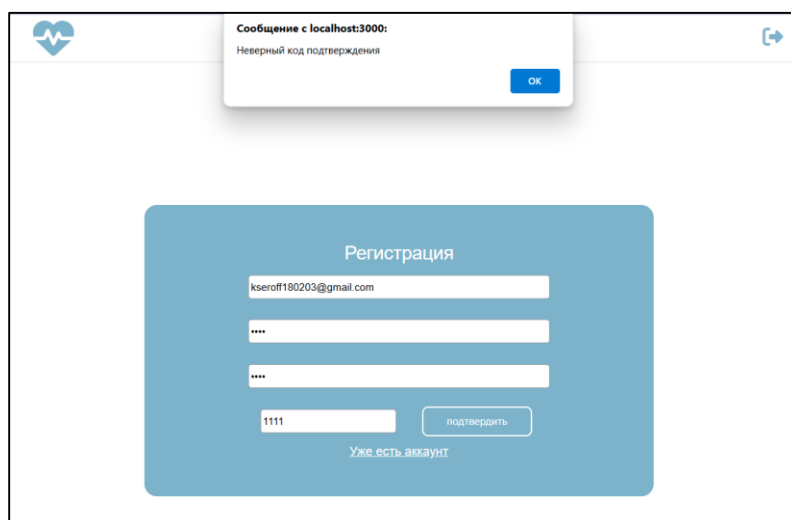


Рисунок 3.20 – ПФ 2.4

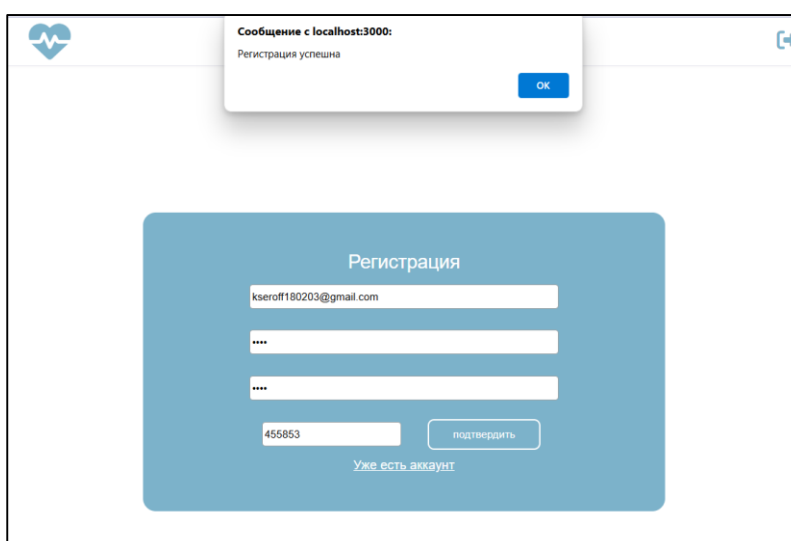


Рисунок 3.21 – ПФ 2.5

Изм.	Лист	№ докум.	Подпись	Дата

Таблица 3 включает тестовые сценарии, направленные на проверку функционала записи пациентов на прием к врачу и возможности оставления отзывов после посещения.

Таблица 3 - Тестирование системы записи на прием и оставления отзывов

Проверяется: система записи на прием и оставления отзывов					
№ теста	Суть теста	Входные данные	Процедура выполнения	Ожидаемый результат	Полученный результат
ПФ 3.1	Проверить недоступность записи для неавторизованного пользователя		Неавторизованным пользователем выбрать врача и попробовать выбрать время для записи	Выведется сообщение о том, что пользователь не авторизован	Совпадает с ожидаемым
ПФ 3.2	Проверить возможность выбора времени для авторизованного пользователя		Авторизованным пользователем выбрать врача и попробовать выбрать время для записи	Откроется окно с информацией и кнопкой записи	Совпадает с ожидаемым
ПФ 3.3	Проверить функционал записи на прием		После выбора времени нажать на кнопку записи	Выйдет сообщение об успешной записи, на почту пользователя придет сообщение и заблокируется время записи	Совпадает с ожидаемым
ПФ 3.4	Проверить, что запись появилась в истории		Перейти в личный кабинет, открыть раздел «Запись», найти запись в предстоящих	В предстоящих записях должна отображаться новая запись с возможностью отменить	Совпадает с ожидаемым
ПФ 3.5	Проверить, что прошедшие записи имеют функционал оставления отзыва		Перейти в личный кабинет, открыть раздел «Запись», найти запись в прошедших	В прошедших записях должны отображаться старые записи с возможностью оставления отзыва	Совпадает с ожидаемым

Результаты тестирования можно увидеть на рисунках 3.22-3.26.

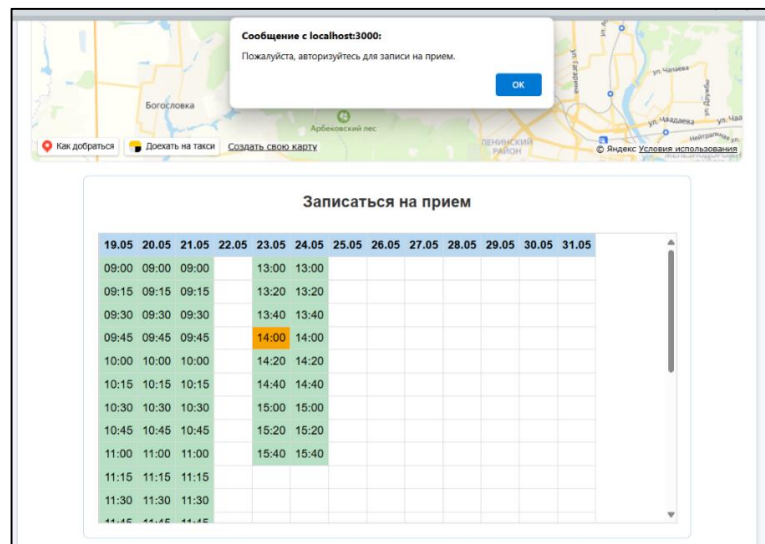


Рисунок 3.22 – ПФ 3.1

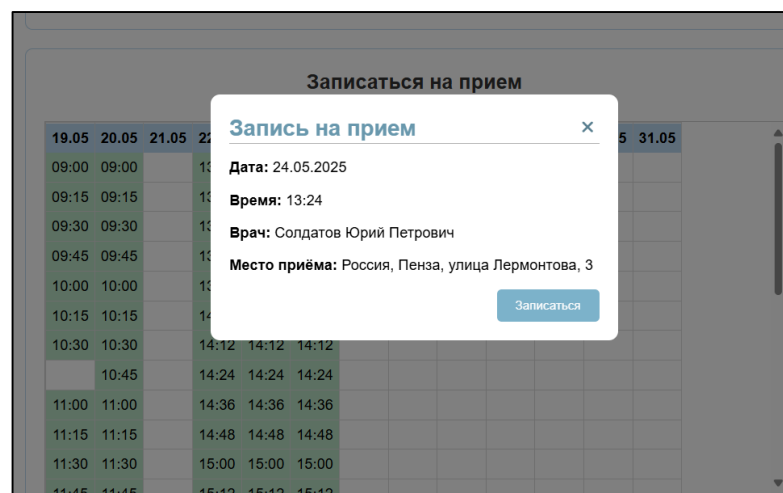


Рисунок 3.23 – ПФ 3.2

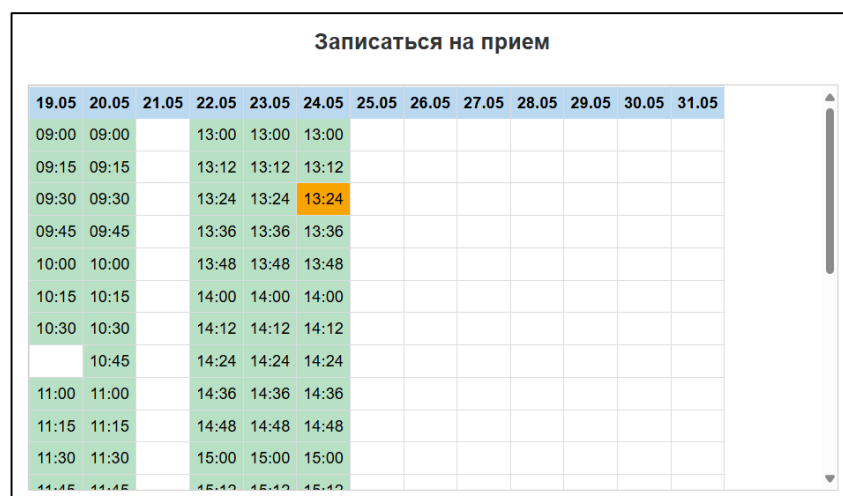


Рисунок 3.24 – ПФ 3.3

▼ Показать предстоящие записи

24.05.2025 13:24

Врач: [Солдатов Юрий Петрович](#)

Адрес: Россия, Пенза, улица Лермонтова, 3

Отменить запись

Рисунок 3.25 – ПФ 3.4

▼ Показать прошедшие записи

25.04.2025 11:12

Врач: [Шуринов Алексей Романович](#)

Адрес: Россия, Пенза, проспект Строителей, 9

▶ Посмотреть отзыв

29.04.2025 15:00

Врач: [Солдатов Юрий Петрович](#)

Адрес: Россия, Пенза, улица Лермонтова, 3

Оставить отзыв

Рисунок 3.26 – ПФ 3.5

4 Руководство пользователя

4.1 Навигация и пользовательский интерфейс

Основные элементы интерфейса расположены таким образом, чтобы обеспечить удобство и быстроту доступа к функциям приложения. В верхней части каждой страницы расположена панель навигации (хедер), включающая логотип, являющийся кнопкой перехода на главную страницу, а также ссылки на личный кабинет и функции авторизации. Главная страница приложения содержит основные разделы: поиск врачей, личный кабинет, карта (рисунок 4.1). Для удобства навигации используются иконки.

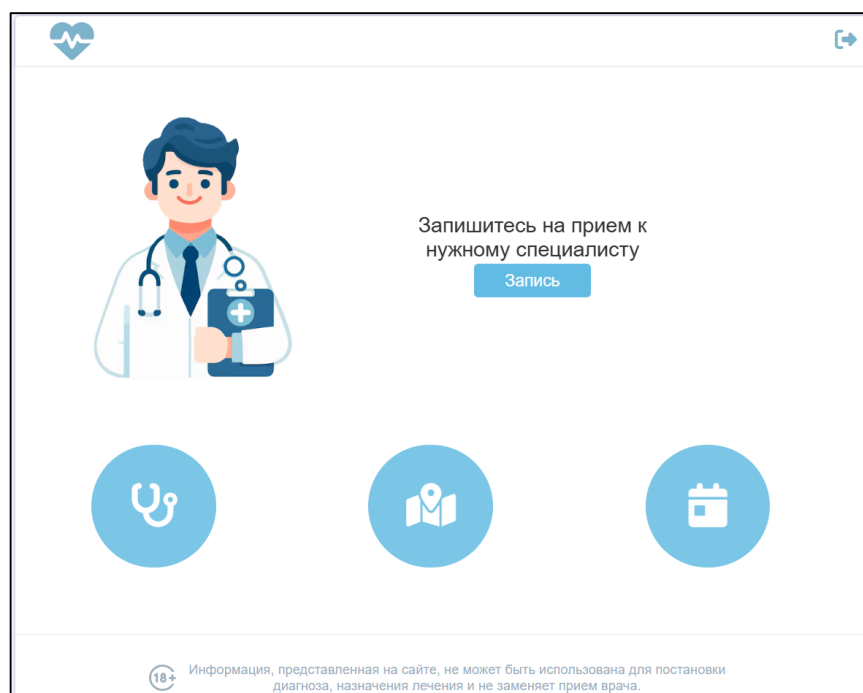


Рисунок 4.1 – Главная страница приложения

На странице поиска присутствует удобный набор фильтров по специальности, опыту и расположению врача, что позволяет быстро сузить результаты. (рисунки 4.2, 4.3) Фильтры поддерживают множественный выбор и динамически обновляют результаты без перезагрузки страницы.

Кроме того, реализована интеграция с интерактивной картой, на которой отображаются места приема врачей, а кликая на метки врача, пользователь открывает краткую информацию с возможностью перейти к полной карточке.

Основные элементы интерфейса, такие как кнопки записи на прием, редактирования профиля и управления расписанием, имеют понятный и интуитивный дизайн.

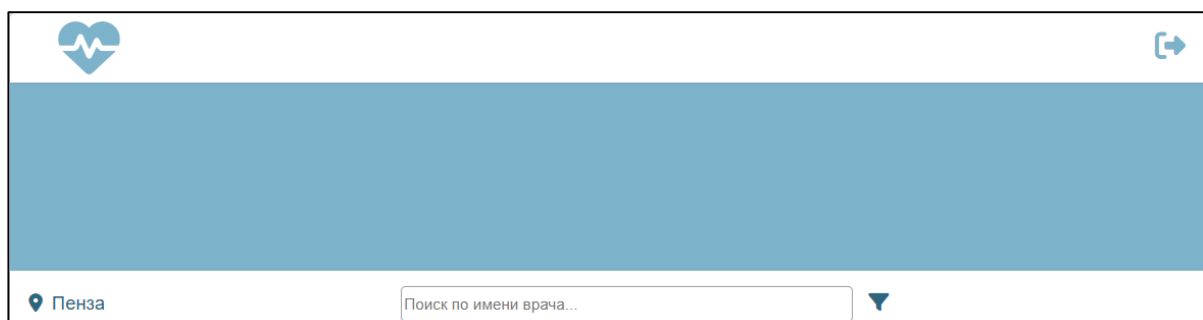


Рисунок 4.2 – Набор фильтров для поиска

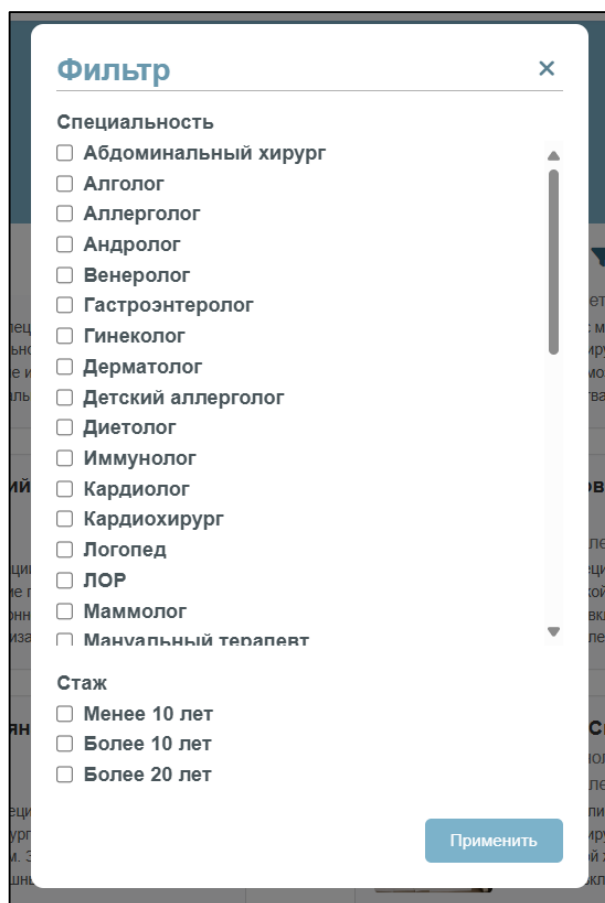


Рисунок 4.3 – Окно фильтрации

4.2 Начало работы

Для начала использования приложения необходимо зарегистрироваться. Процесс регистрации требует указания электронной почты и пароля (рисунок 4.4), после чего на указанную почту будет выслан код для подтверждения регистрации (рисунок 4.5), который будет необходимо ввести в форме для завершения создания аккаунта (рисунок 4.6).

Рисунок 4.4 – Форма регистрации

Рисунок 4.5 – Письмо с кодом-подтверждением

Рисунок 4.6 – Подтверждение почты

После регистрации пользователь может войти в систему, используя свои учетные данные (рисунок 4.7). При успешной авторизации открывается доступ к функционалу, соответствующему роли пользователя.

Рисунок 4.7 – Форма авторизации

4.3 Функционал для пользователей с ролью пациент

Пациенты могут искать врачей несколькими способами. Во-первых, на странице поиска отображается список специалистов с возможностью фильтрации по различным критериям (рисунок 4.8).

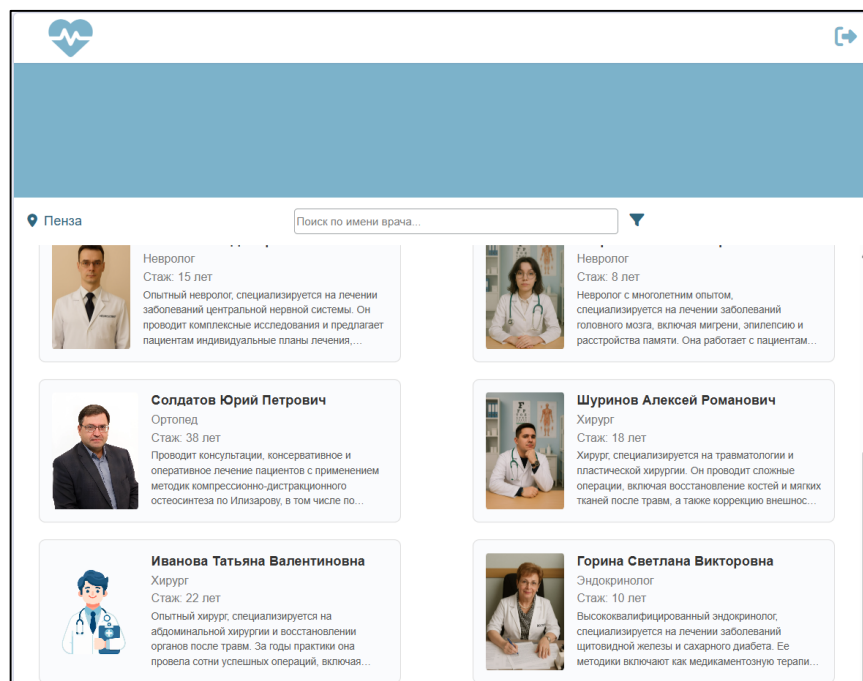


Рисунок 4.8 – Страница поиска списком

Альтернативный способ – это использование карты, на которой отображены врачи синими метками, место жительства (если указано в профиле) красной меткой и текущее местоположение с возможностью выбора ближайшего места приема (рисунок 4.9).

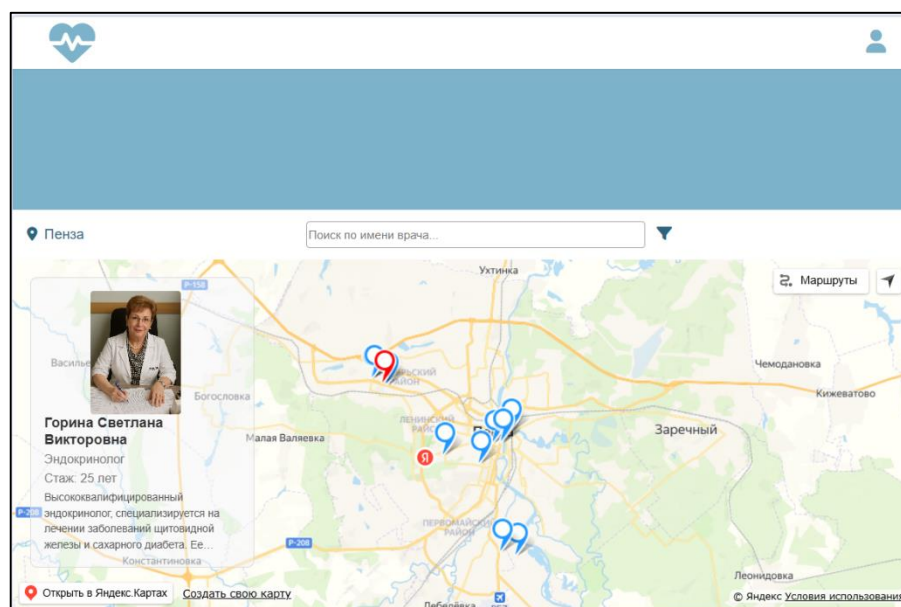


Рисунок 4.9 – Страница поиска с помощью карты

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.11.001 ПЗ

Лист

64

Выбрав специалиста, пациент открывает карточку врача, где представлена подробная информация, включая фото, специализацию, место приема и отзывы других пациентов (рисунок 4.10).



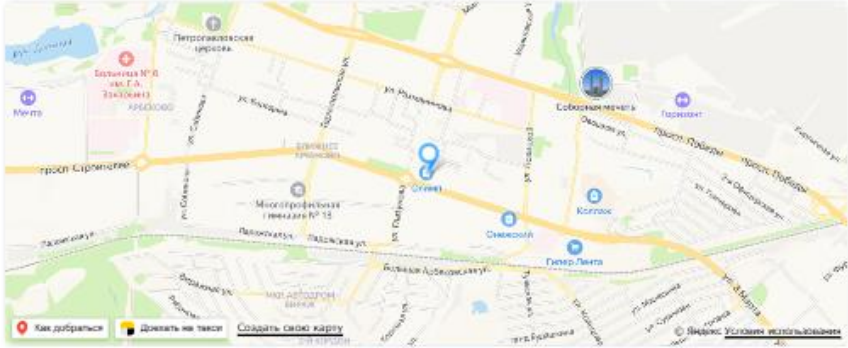



Шуринов Алексей Романович

Специальность: Хирург

Стаж: 18 лет

Описание: Хирург, специализируется на травматологии и пластической хирургии. Он проводит сложные операции, включая восстановление костей и мягких тканей после травм, а также коррекцию внешности с использованием новейших технологий.



Записаться на прием

17.05	18.05	19.05	20.05	21.05	22.05	23.05	24.05	25.05	26.05	27.05	28.05	29.05	30.05
		09:00	09:00	09:00		13:00	13:00						
		09:15	09:15	09:15		13:20	13:20						
		09:30	09:30	09:30		13:40	13:40						
		09:45	09:45	09:45		14:00	14:00						
		10:00	10:00	10:00		14:20	14:20						
		10:15	10:15	10:15		14:40	14:40						
		10:30	10:30	10:30		15:00	15:00						
		10:45	10:45	10:45		15:20	15:20						
		11:00	11:00	11:00		15:40	15:40						
		11:15	11:15	11:15									

Отзывы

Пациент: Козлова Кристина Сергеевна

Дата приема: 25.04.2025

★★★★★

Доктор очень внимателен, подробно объяснил все этапы предстоящей операции и ответил на все мои вопросы. Операция прошла успешно, и я быстро восстановился.

Рисунок 4.10 – Карточка врача

Для записи на прием достаточно выбрать доступное время в расписании врача и подтвердить заявку (рисунок 4.11). На почту будет выслана информация о приеме (рисунок 4.12), а время заблокируется для записи (рисунок 4.13). После записи пациент может просмотреть свои предстоящие и прошедшие визиты в личном кабинете (рисунок 4.14), а также оставить отзыв о прошедшем приеме (рисунок 4.15) или отменить предстоящий.

Рисунок 4.11 – Окно записи на прием

Рисунок 4.12 – Письмо с информацией о записи на прием

Записаться на прием

17.05	18.05	19.05	20.05	21.05	22.05	23.05	24.05	25.05	26.05	27.05	28.05	29.05	30.05	31.05
		09:00	09:00	09:00		13:00	13:00							
		09:15	09:15	09:15		13:20	13:20							
		09:30	09:30	09:30		13:40	13:40							
		09:45	09:45	09:45		14:00	14:00							
		10:00	10:00	10:00		14:20	14:20							
		10:15	10:15	10:15		14:40	14:40							
		10:30	10:30	10:30		15:00	15:00							
		10:45	10:45	10:45		15:20	15:20							
		11:00	11:00	11:00		15:40	15:40							
		11:15	11:15	11:15										
		11:30	11:30	11:30										
		11:45	11:45	11:45										

Рисунок 4.13 – Занятое время в таблице записи

Профиль

Запись

Показать предстоящие записи

23.05.2025 14:00

Врач: [Солдатов Юрий Петрович](#)

Адрес: Россия, Пенза, улица Лермонтова, 3

Отменить запись

23.05.2025 14:00

Врач: [Шуринов Алексей Романович](#)

Адрес: Россия, Пенза, проспект Строителей, 9

Отменить запись

Показать прошедшие записи

25.04.2025 11:12

Врач: [Шуринов Алексей Романович](#)

Адрес: Россия, Пенза, проспект Строителей, 9

Посмотреть отзыв

★★★★★

Доктор очень внимателен, подробно объяснил все этапы предстоящей операции и ответил на все мои вопросы. Операция прошла успешно, и я быстро восстановился.

29.04.2025 15:00

Врач: [Солдатов Юрий Петрович](#)

Адрес: Россия, Пенза, улица Лермонтова, 3

Оставить отзыв

Рисунок 4.14 – История записи пациента

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.11.001 ПЗ

Лист

67

Рисунок 4.15 – Модальное окно создания отзыва

4.4 Функционал для пользователей с ролью доктор

Врачи имеют возможность заполнять и редактировать свою профильную информацию, включая фото, контактные данные, адрес места приема (рисунок 4.16) и загружать сертификаты, подтверждающие квалификацию (рисунок 4.17).

Рисунок 4.16 – Профиль личного кабинета

Изм.	Лист	№ докум.	Подпись	Дата

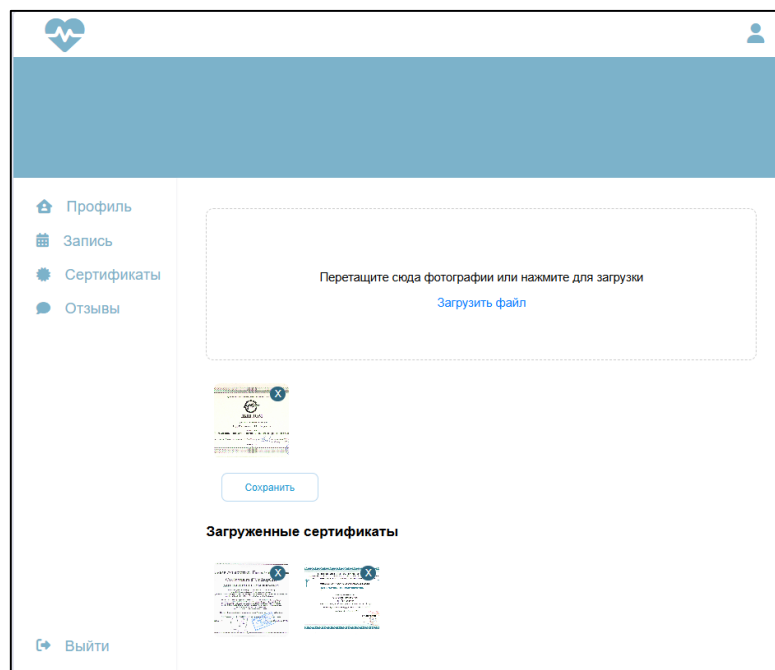


Рисунок 4.17 – Страница загрузки сертификатов

Для организации приема реализован удобный интерфейс составления расписания, где можно создавать таблицы с доступными временными слотами и управлять ими, занятые пациентами слоты окрашены оранжевым, при клике по нему появится окно отмены записи, пациенту в свою очередь придет письмо с причиной отмены (рисунки 4.18, 4.19, 4.20).

Выбор дат
18.05.2025 01.06.2025

Построить таблицу

18.05	19.05	20.05	21.05	22.05	23.05	24.05	25.05	26.05	27.05	28.05	29.05	30.05	31.05
		09:00	09:00		13:15		09:00		13:15				
		09:15	09:15		13:30		09:15		13:30				
		09:30	09:30		13:45		09:30		13:45				
		09:45	09:45		14:00		09:45		14:00				
		10:00	10:00		14:15		10:00		14:15				
		10:15	10:15		14:30		10:15		14:30				
		10:30	10:30		14:45		10:30		14:45				
		10:45	10:45		15:00		10:45		15:00				
		11:00	11:00		15:15		11:00		15:15				
		11:15	11:15		15:30		11:15		15:30				
		11:30	11:30		15:45		11:30		15:45				
		11:45	11:45		16:00		11:45		16:00				

Очистить время для выбранных дат

Время: 13:15 18:45 Длительность (в минутах): 15

Применить

Рисунок 4.18 – Построение таблицы приема

Построить таблицу

Прием
×

Дата: 23.05.2025

Время: 14:00

Пациент: Козлова Кристина Сергеевна

Телефон: +79273690346

Отменить запись

Рисунок 4.19 – Окно с информацией о записи

Прием
×

Дата: 23.05.2025

Время: 14:00

Пациент: Козлова Кристина Сергеевна

Телефон: +79273690346

Введите причину отмены

Отправить

Рисунок 4.20 – Отмена записи со стороны врача

Врачам доступен просмотр отзывов пациентов, что позволяет контролировать качество оказываемых услуг и взаимодействовать с пользователями (рисунок 4.21).

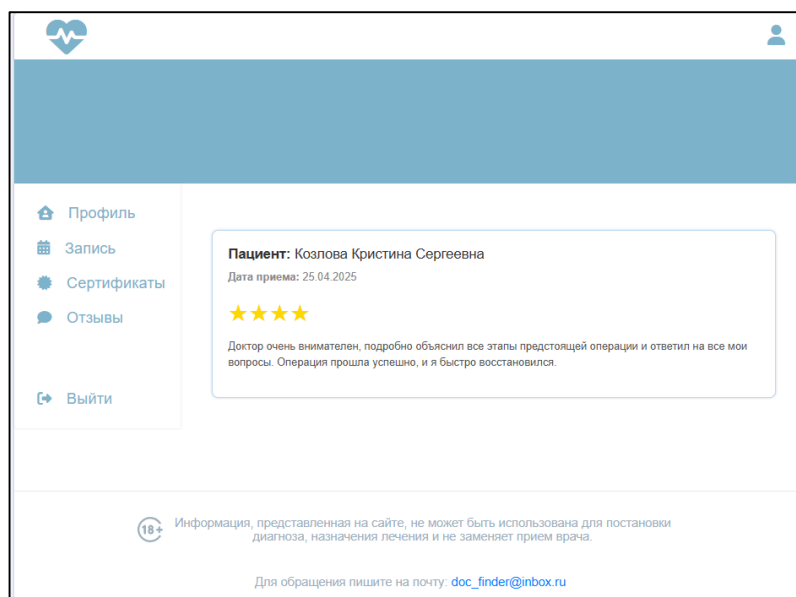


Рисунок 4.21 – Отзывы в личном кабинете врача

4.5 Функционал для пользователей с ролью администратор

Администраторы имеют расширенные права доступа и могут управлять пользователями, назначая им соответствующие роли (рисунки 4.22, 4.23).

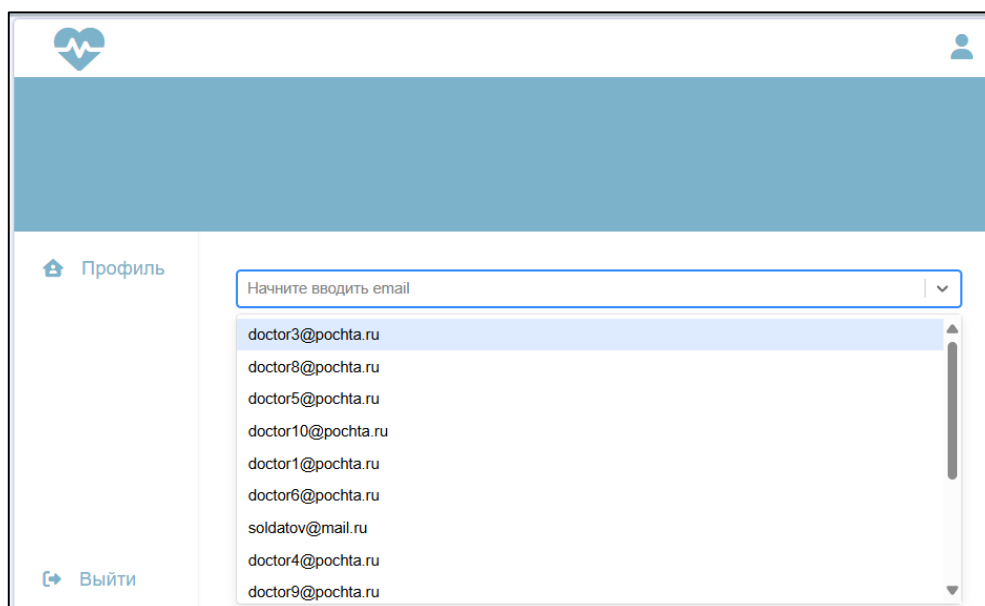


Рисунок 4.22 – Выбор пользователя в кабинете администратора

Профиль

kornev@mail.ru

Выберите роль:

Пациент

Доктор

Пациент

Выйти

Рисунок 4.23 – Смена роли для пользователя

Так же администратор может устанавливать пользователям с ролью врач специальность, стаж и описание с целью достоверности информации (рисунок 4.24).

Профиль

doctor7@pochta.ru

Выберите роль:

Доктор

Дополнительная информация о докторе

Специализация

Эндокринолог

Описание

Опытный эндокринолог, специализируется на лечении заболеваний щитовидной железы и нарушений обмена веществ. В своей практике использует новейшие методики диагностики, включая гормональные исследования и тесты на чувствительность к инсулину.

Стаж

14

Сохранить

Выйти

Рисунок 4.24 – Заполнение информации о враче

Изм.	Лист	№ докум.	Подпись	Дата

В системе предусмотрена возможность модерации отзывов администратором, чтобы исключить некорректные отзывы (рисунок 4.25).

Отзывы

Пациент: Козлова Кристина Сергеевна

Дата приема: 25.04.2025

★★★★★

Доктор очень внимателен, подробно объяснил все этапы предстоящей операции и ответил на все мои вопросы. Операция прошла успешно, и я быстро восстановился.

Удалить отзыв

Рисунок 4.25 – Возможность удаления отзывов администратором

5 Охрана труда

В данном разделе изложены основные аспекты, направленные на создание и поддержание безопасных условий труда для пользователей персональных компьютеров, а также меры, обеспечивающие сохранение их здоровья в процессе работы с программным обеспечением. Особое внимание уделяется выявлению и снижению негативного воздействия факторов, способных вызвать ухудшение физического и психического состояния работника. Рассматриваются методы, которые помогают обеспечить комфорт и безопасность рабочего пространства, предотвращая возможные профессиональные заболевания и несчастные случаи.

5.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Персональные компьютеры и другие электронные устройства широко используются для решения разнообразных задач, однако их эксплуатация связана с определёнными рисками. Среди возможных опасностей выделяют воздействие вредных физических факторов - таких, как напряжение зрения из-за длительной работы с экраном, негативное влияние не эргономичного рабочего места, а также неблагоприятные параметры микроклимата и освещения. Несоблюдение установленных нормативов может привести к ухудшению здоровья пользователя, развитию хронических заболеваний и снижению работоспособности. Для минимизации таких рисков необходимо придерживаться рекомендаций по организации рабочего места, нормировать время работы за компьютером, обеспечивать оптимальный уровень освещённости и поддерживать комфортную шумовую среду.

5.2 Классификация и категорирование помещения, в котором производится разработка

Рабочие помещения, в которых происходит эксплуатация вычислительной техники и ведётся разработка программного обеспечения, подлежат обязательной классификации с учётом возможных рисков для безопасности персонала и имущества. Основные критерии оценки включают взрывопожарную и пожарную опасность, а также вероятность поражения электрическим током.

5.2.1 Классификация по взрывопожарной и пожарной опасности

Помещения распределяются по категориям в зависимости от присутствия и свойств горючих материалов, которые могут стать причиной возгорания или взрыва. Категории определяются с учётом следующих факторов: химический состав веществ, температура вспышки, их количество и условия хранения, а также специфика производственных процессов:

– категория «А» включает помещения с высоким риском возникновения взрывов и пожаров. В таких зонах содержатся горючие газы или легковоспламеняющиеся жидкости с низкой температурой воспламенения (не выше 28 °С). В этом случае возможно образование взрывоопасных парогазовоздушных смесей, способных привести к взрывному давлению, превышающему 5 кПа. Также к этой категории относятся материалы, способные взрываться или гореть при контакте с кислородом, водой или при взаимодействии друг с другом;

– категория «Б» относится к помещениям, где есть взрывопожароопасность, обусловленная наличием горючих пылей и волокон, а также легковоспламеняющихся жидкостей с температурой вспышки выше 28 °С. В таких зонах возможна концентрация пылевоздушных или паровоздушных смесей, при возгорании которых давление взрыва также может превысить 5 кПа;

					ПГУ1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

– категории «В1» – «В4» классифицируются как пожароопасные помещения, в которых присутствуют горючие и трудногорючие жидкости, а также вещества и материалы, способные воспламеняться при взаимодействии с водой, кислородом воздуха или другими веществами;

– категория «Г» включает помещения с умеренной пожароопасностью. В таких зонах находятся негорючие материалы, но которые могут находиться в горячем или раскалённом состоянии, а также выделяют лучистое тепло, искры или открытое пламя в процессе обработки;

– категория «Д» охватывает помещения с пониженной пожароопасностью, где содержатся холодные негорючие материалы и вещества;

Категоризация зданий и помещений необходима для правильного выбора систем пожарной безопасности и организации мер по предупреждению чрезвычайных ситуаций.

5.2.2 Классификация по степени опасности поражения электрическим током

Особое внимание уделяется условиям, которые могут повысить риск поражения электрическим током. Такие условия принято разделять на обычные и повышенной опасности.

К факторам, создающим повышенный риск, относятся:

– Влажность воздуха: особенно опасна среда с высокой относительной влажностью, превышающей 75%. В таких условиях усиливается проводимость электричества, что увеличивает вероятность поражения током.

– Проводящая пыль: оседающая на электропроводах и оборудовании, она ухудшает изоляцию и способствует возникновению коротких замыканий и пробоев, что создает небезопасные условия.

– Конструктивные материалы основания: металлические, железобетонные или кирпичные основания обладают высокой

					ПГУ1.090301.11.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

электропроводностью, что повышает вероятность передачи тока и увеличивает риск поражения.

– Температурный режим: длительное воздействие высоких температур (более 35 °С), а также кратковременные повышения до 40 °С создают особые условия, при которых риск поражения током возрастает.

Помимо вышеперечисленных факторов, существуют дополнительные условия, существенно повышающие опасность:

– наличие сырости, включая осадки, конденсат, частое орошение и влажное покрытие стен и потолков;

– присутствие химически активных веществ, способных оказывать разрушительное воздействие на изоляционные материалы и оборудование;

– совместное воздействие нескольких факторов, например, высокая влажность при высокой температуре и наличии проводящей пыли.

5.2.3 Рекомендации по обеспечению безопасных условий

Для снижения рисков и защиты работников рекомендуется соблюдать следующие требования к помещению:

– температура воздуха должна поддерживаться в диапазоне от 18 до 20 °С, при этом не допускается превышение температуры ниже 30 °С, чтобы предотвратить ухудшение условий работы и опасность поражения током;

– полы рекомендуется выполнять из диэлектрических материалов, что значительно снижает вероятность возникновения электрического заряда и выхода из строя изоляции;

– вентиляция помещений должна обеспечивать регулярное обновление воздуха для поддержания оптимального уровня кислорода и предотвращения образования опасных концентраций газов и паров;

– контроль влажности необходим для поддержания уровня не выше 60%, что позволяет уменьшить образование конденсата на электрических приборах и тем самым снизить вероятность поражения током.

Применение перечисленных мер способствует созданию безопасной и комфортной рабочей среды, предотвращая травматизм и профессиональные заболевания, связанные с электротравмами и пожарными рисками.

5.3 Санитарно-гигиеническая характеристика производства

Для создания безопасных и комфортных условий труда большое значение имеют санитарно-гигиенические параметры рабочего пространства. К ним относятся показатели влажности, температуры, объёма помещения и освещённости, которые непосредственно влияют на самочувствие и продуктивность работников.

Каждое рабочее место должно иметь минимальный объём не менее 15 кубических метров, что обеспечивает достаточное пространство для свободного перемещения и предотвращает перенаселённость. Высота потолков рекомендуется не ниже 3,2 метра, а ширина дверных проёмов должна составлять минимум 0,8 метра для обеспечения удобного доступа и эвакуации в случае необходимости. Коридоры и проходы должны иметь ширину не менее 1,4 метра, что также способствует безопасной организации движения внутри помещения.

Особое внимание уделяется обеспечению естественного освещения. Окна должны располагаться таким образом, чтобы максимально обеспечивать попадание дневного света на рабочие места, способствуя снижению утомляемости глаз и улучшению общего состояния работников. При недостатке естественного освещения необходимо устанавливать искусственные осветительные приборы, которые должны соответствовать установленным стандартам по яркости и распределению света. Для предотвращения бликов и рассеивания света светильники рекомендуется изготавливать из армированного стекла с защитой в виде металлических сеток.

Оптимальные параметры микроклимата включают поддержание температуры воздуха в пределах от 18 до 21 градуса Цельсия при влажности от

					ПГУ1.090301.11.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

40 до 60 процентов. Такой режим способствует сохранению комфортного теплового баланса и предотвращает развитие заболеваний, связанных с неблагоприятными климатическими условиями.

Кроме того, руководству производства необходимо обеспечить сотрудников чистой питьевой водой, которую следует регулярно обновлять. В помещениях с повышенным уровнем шума необходимо устанавливать предупреждающие знаки, информирующие работников о потенциальных рисках. Все сотрудники, работающие в таких условиях, должны проходить инструктаж по безопасности, знать правила использования средств индивидуальной защиты, например, наушников или специальных защитных вкладышей и регулярно контролировать состояние этих средств для эффективной защиты слуха.

Таким образом, соблюдение перечисленных санитарно-гигиенических требований является важной составляющей комплексной системы безопасности труда, способствующей сохранению здоровья и повышению эффективности персонала.

5.4 Расчёт и анализ условий визуальной работы в помещении с персональными вычислительными машинами

Важной составляющей обеспечения безопасности и комфорта при работе с персональными электронными вычислительными машинами (ПЭВМ) является правильное освещение и соответствующие зрительные условия. Это напрямую влияет на уровень усталости глаз, продуктивность труда и общее состояние здоровья сотрудников.

Для оценки условий визуальной работы в помещении необходимо определить ряд ключевых показателей, включая нормативный уровень освещённости, коэффициенты естественного и искусственного освещения, а также параметры, влияющие на качество восприятия изображения на экране.

					ПГУ1.090301.11.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

В рамках исследования были взяты следующие исходные данные: площадь производственного помещения составляет 100 квадратных метров, количество ПЭВМ — 10 единиц, а уровень освещённости — 500 люкс. На основе этих параметров рассчитываются показатели, позволяющие оценить соответствие освещения установленным стандартам.

Для проведения расчета была выбрана методика, основанная на рекомендациях ГОСТ 12.2.003-91 "Эргономические требования к работам на персональных электронных вычислительных машинах". Таблицы из ГОСТ приведены ниже.

Таблица 4 - Уровень освещенности

Категория сложности работы	Уровень освещения, лк	Предельно допустимое значение, лк
Легкая	400-500	750
Средняя	300-400	500
Тяжелая	200-300	300

Таблица 5- Коэффициент тени

Категория сложности работы	Коэффициент тени, %	Предельно допустимое значение, %
Легкая	30	50
Средняя	20	30
Тяжелая	10	20

Таблица 6 - Коэффициент блеска

Категория сложности работы	Коэффициент блеска, %	Предельно допустимое значение, %
Легкая	15-20	40
Средняя	10-15	30
Тяжелая	5-10	20

Среди основных характеристик выделяются:

– нормативный уровень освещённости, который для производственных помещений с использованием ПЭВМ должен составлять от 300 до 500 люкс. При выбранном значении в 500 люкс условия работы считаются оптимальными для предотвращения переутомления зрения;

– коэффициент естественного освещения (КЕО), отражающий долю дневного света в общем освещении помещения. В рассматриваемом случае он равен нулю, что связано с преобладанием искусственного освещения;

– коэффициент использования освещения (КИО) показывает эффективность распределения светового потока на рабочей поверхности. При применении современных люминесцентных или светодиодных светильников и правильном размещении светильников на высоте 2–2,5 метра этот показатель достигает 40%;

– коэффициент тени (КТ) характеризует долю теней, образующихся на рабочих местах. Для обеспечения равномерного освещения компьютеры располагаются с учётом минимизации перекрытия светового потока, что позволяет поддерживать значение коэффициента тени около 10%;

– коэффициент блеска (КБ) показывает влияние отражённых от поверхностей световых лучей на освещённость рабочего места. Использование матовых покрытий и соответствующих материалов позволяет ограничить этот показатель примерно до 5%;

– коэффициент контрастности (КК) оценивает разницу между яркими и тёмными участками на экране и рабочей поверхности. Оптимальным считается соотношение 10:1, что способствует комфортной работе с визуальной информацией;

Проведённый анализ показывает, что при правильной организации освещения и соблюдении нормативов условия зрительной работы в рассматриваемом помещении соответствуют требованиям безопасности и эргономики.

Заключение

В ходе выполнения дипломной работы был разработан веб-сервис, обеспечивающий функционал для врачей и пациентов. В процессе разработки были применены современные технологии веб-разработки: React для клиентской части, Express.js и Node.js для серверной логики, PostgreSQL в качестве СУБД. Особое внимание уделялось организации безопасной аутентификации с использованием JWT, обработке загрузки файлов и интеграции с внешними API, что позволило сформировать комплексное и удобное решение.

Реализованы ключевые возможности: регистрация пользователей с различными ролями, функционал личных кабинетов для врачей и пациентов, система записи на приемы с фильтрацией и отображением по карте, а также управление отзывами и расписанием.

Тестирование с помощью Postman и пользовательский интерфейс подтвердили работоспособность и соответствие функциональным требованиям.

Приложение успешно развернуто на виртуальном частном сервере (VPS) под управлением ОС Ubuntu. Для маршрутизации запросов и обслуживания статики настроен обратный прокси Nginx.

Внедрена поддержка SSL-сертификатов, что обеспечивает защищённое HTTPS-соединение и безопасность передачи данных.

Возможны дальнейшие доработки сервиса:

- расширение возможностей для администраторов,
- улучшение масштабируемости сервиса и оптимизация работы с большим объемом данных.

В числе перспективных направлений развития приложения рассматривается дальнейшая интеграция с сервисами Яндекс, что позволит расширить возможности поиска и отображения врачей, а также повысить точность геолокации и удобство навигации для пользователей.

В будущем возможна интеграция с платформой 2ГИС для автоматического подтягивания отзывов с их сервиса, что позволит обогатить информацию о специалистах и повысить доверие пользователей за счет увеличения объема и объективности отзывов.

Полученные в ходе разработки результаты являются минимально жизнеспособным продуктом (MVP) готовым для внедрения в частные медицинские организации.

					ПГУ1.090301.11.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Документация к библиотеке React. [Электронный ресурс] / Режим доступа: <https://ru.react.js.org/docs/getting-started.html> свободный. (дата обращения 27.05.2025).
2. Документация к библиотеке @pbe/react-yandex-maps. [Электронный ресурс] / Режим доступа: <https://pbe-react-yandex-maps.vercel.app/> свободный. (дата обращения 27.05.2025).
3. Документация по VPS серверам джино. [Электронный ресурс] / Режим доступа: <https://jino.ru/spravka/vps/#vps-articles> свободный. (дата обращения 27.05.2025).
4. Статья Express/Node introduction. [Электронный ресурс] / Режим доступа: https://developer.mozilla.org/ru/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction свободный. (дата обращения 27.05.2025)
5. Статья ИСПДн и уровни защищенности персональных данных. [Электронный ресурс] / Режим доступа: <https://selectel.ru/blog/personal-info-is/#:~:text=%D0%A7%D1%82%D0%BE%D0%B1%D1%8B%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D1%8C%20%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C%20%D0%B7%D0%B0%D1%89%D0%B8%D1%89%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%98%D0%A1%D0%9F%D0%94%D0%BD%2C%20%D0%BD%D1%83%D0%B6%D0%BD%D0%BE%20%D1%80%D0%B0%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D1%8C%D1%81%D1%8F%20%D0%B2,%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D1%8B%20%D0%B8%20%D0%B2%D0%BE%D0%B7%D1%8C%D0%BC%D0%B5%D0%BC>

%20%D0%B2%D1%81%D0%B5%20%D1%82%D1%80%D1%83%D0%B4
%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%BC%D0%B8%
D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%B8%20%D0%BD%D0
%B0%20%D1%81%D0%B5%D0%B1%D1%8F.свободный.(дата
обращения 27.05.2025).

6. Документация по фреймворку Express.js: [Электронный ресурс] /
Режим доступа: <https://expressjs.com/ru/guide/routing.html> свободный.
(дата обращения 27.05.2025).

7. Документация по веб серверу Nginx: [Электронный ресурс] / Режим
доступа:

https://nginx.org/ru/docs/beginners_guide.html#:~:text=%D0%92%20%D1%8D%D1%82%D0%BE%D0%BC%20%D1%80%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B5%20%D0%BE%D0%BF%D0%B8%D1%81%D1%8B%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F%2C%20%D0%BA%D0%B0%D0%BA%20%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D1%82%D0%B8%D1%82%D1%8C%20%D0%B8%20%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%82%D1%8C,nginx%2C%20%D0%B8%20%D0%BA%D0%B0%D0%BA%20%D1%81%D0%B2%D1%8F%D0%B7%D0%B0%D1%82%D1%8C%20nginx%20%D1%81%20%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC%20FastCGI. свободный. (дата обращения 27.05.2025).

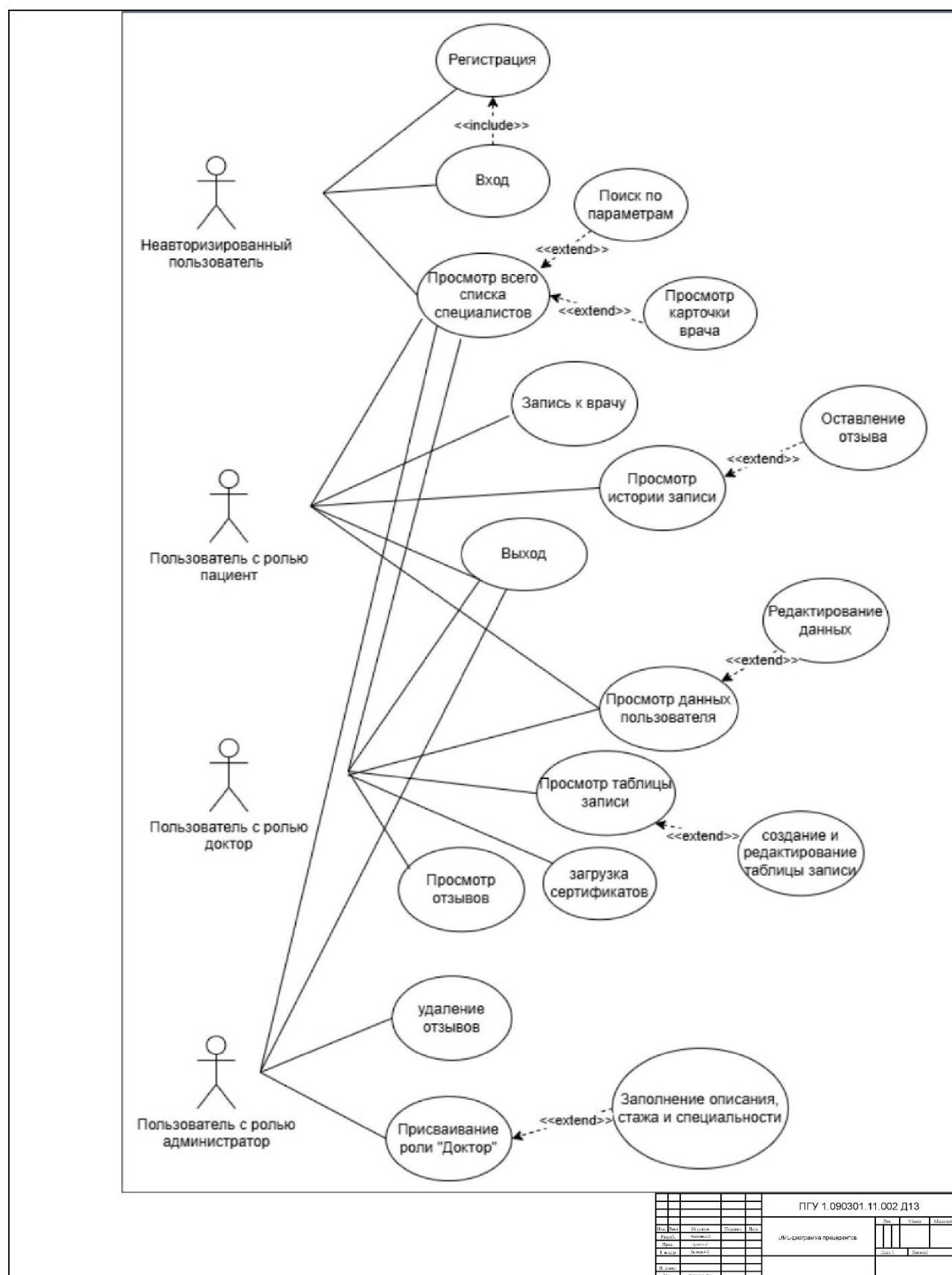
8. Документация Node.js: [Электронный ресурс] / Режим доступа:
<https://nodejs.org/docs/latest/api/> свободный. (дата обращения 27.05.2025).

Приложение А
(Обязательное)

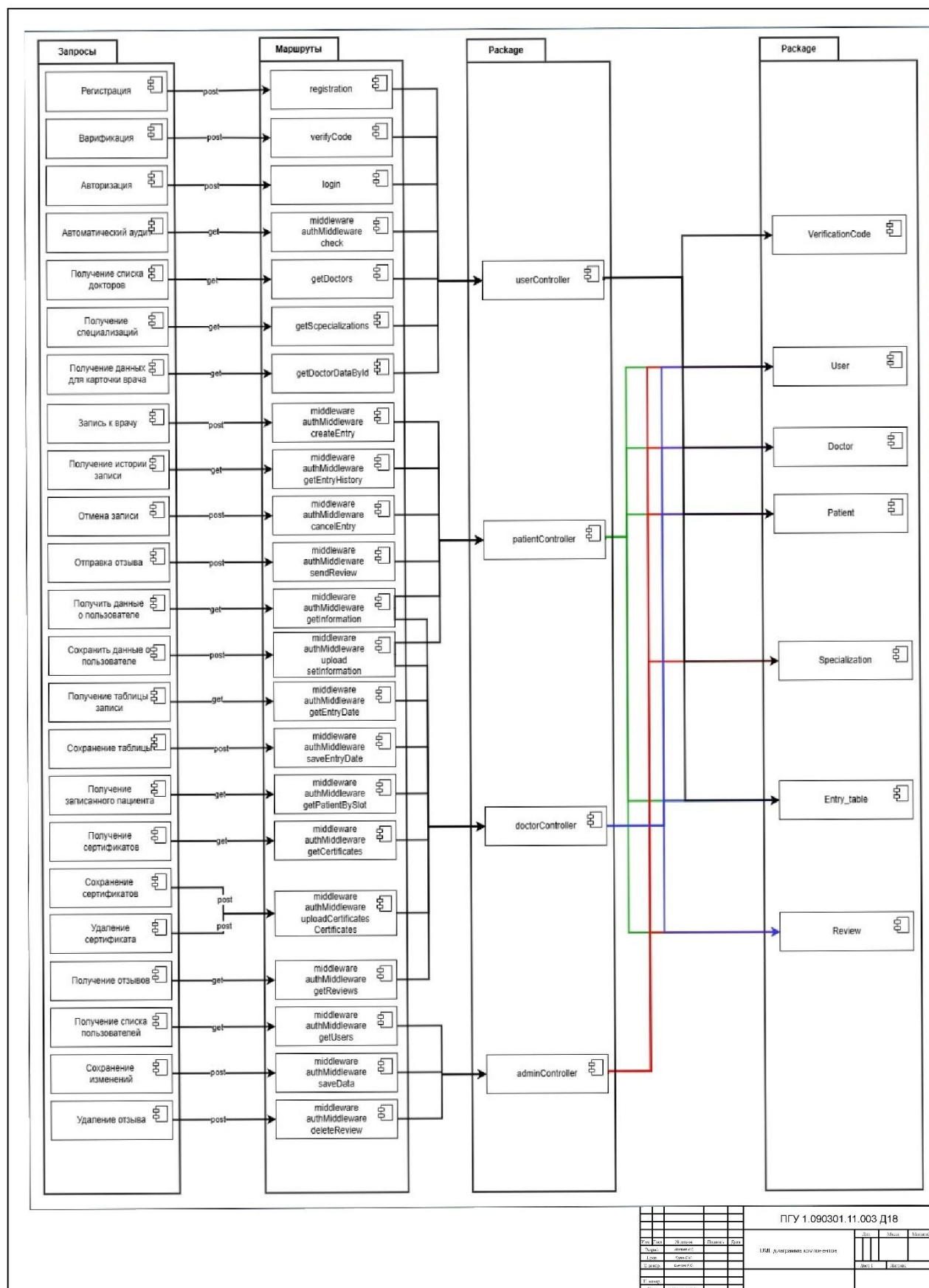
Приложение «DocFinder: поиск врачей и мгновенная запись онлайн»
UML-диаграммы

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

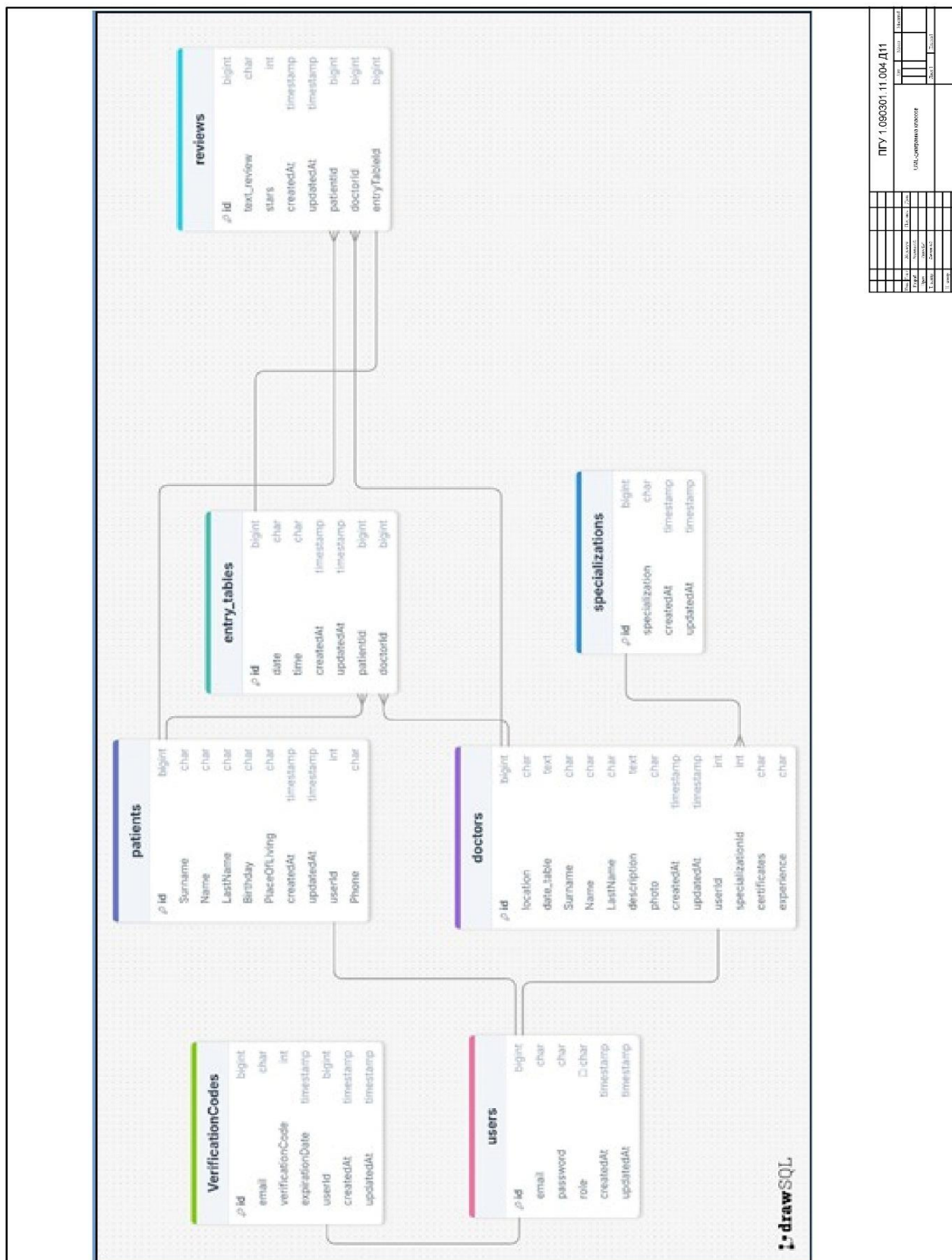
Приложение А.1 – UML – диаграмма прецедентов



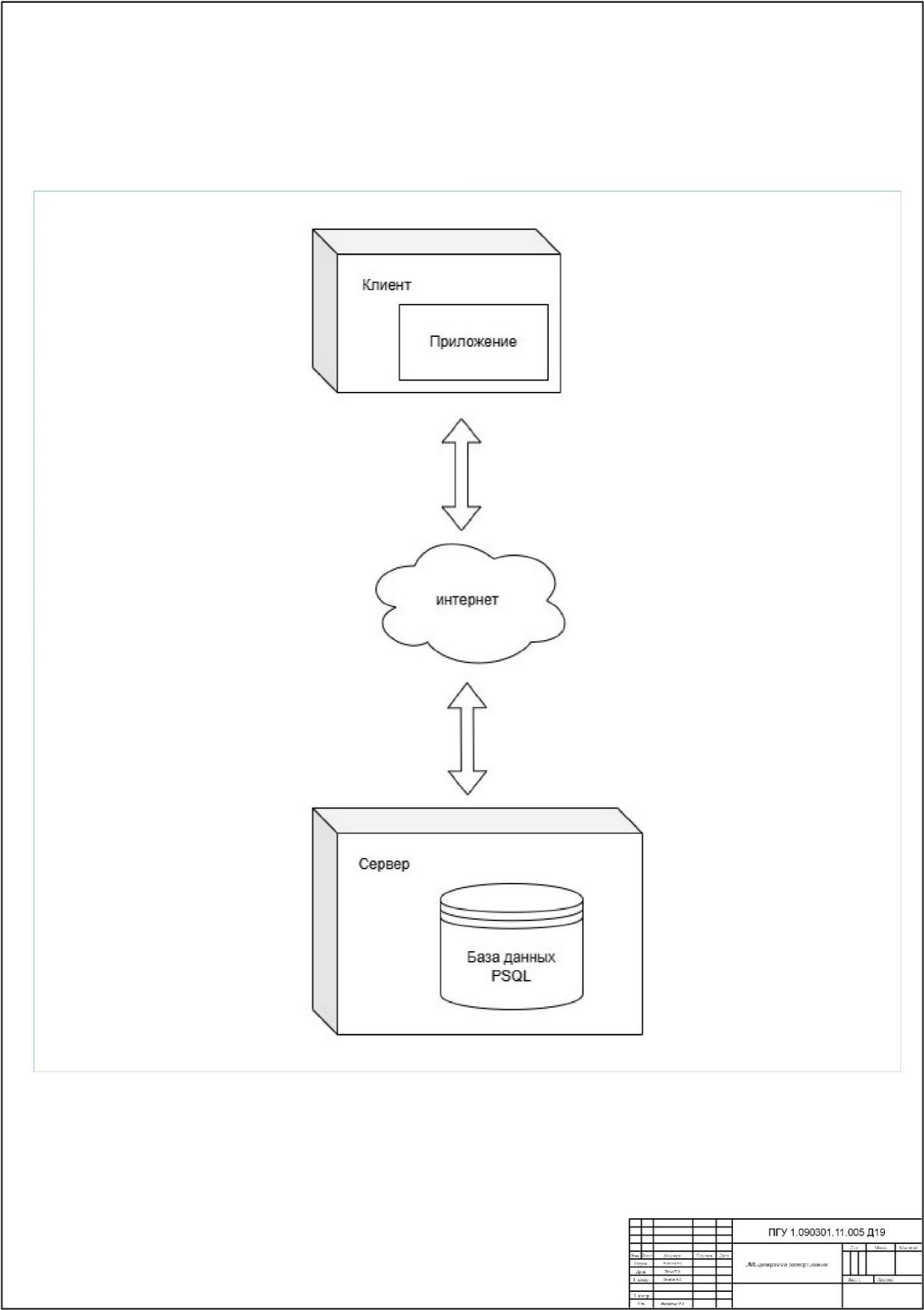
Приложение А.2 – UML – диаграмма компонентов



Приложение А.3 – UML – диаграмма классов



Приложение А.3 – UML – диаграмма развертывания



Приложение Б
(Обязательное)

Приложение «DocFinder: поиск врачей и мгновенная запись онлайн»

Листинг программы

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

Файл server.js

```
require('dotenv').config();
const express = require('express');
const sequelize = require('./db');
const models = require('./models/models');
const path = require('path');
const app = express();
const router = require('./routes/base');
const errorHandler =
require('./middleware/ErrorHandlerMiddleware');
const PORT = process.env.PORT;

app.use(express.json())
app.use('/api', router)
app.use('/static', express.static(path.resolve(__dirname,
'static')));
app.use(errorHandler)
app.get('/api/test', (req, res) => {
  res.send('API is reachable!');
});
const start = async () => {
  try {
    await sequelize.authenticate();
    await sequelize.sync();
    app.listen(PORT, () => console.log('Сервер запущен' +
PORT));
  } catch (e) {
    console.log(e);
  }
};
start();
```

Файл userController.js

```
const bcrypt = require("bcrypt");
const {
  User,
  Doctor,
  Patient,
  Specialization,
  Entry_table,
  VerificationCode,
} = require("../models/models");
const ApiError = require("../error/ApiError");
const jwt = require("jsonwebtoken");
const path = require("path");
const nodemailer = require("nodemailer");
```

```

const fs = require("fs");
const generateJwt = (id, email, role) => {
  return jwt.sign({ id: id, email, role }, process.env.SECRET_KEY,
    {
      expiresIn: "24h",
    });
};
function generateVerificationCode() {
  return Math.floor(100000 + Math.random() * 900000); //
  Генерирует случайный код из 6 цифр
}
class UserController {
  async registration(req, res, next) {
    try {
      const { email, password } = req.body;
      if (!email || !password) {
        return next(ApiError.badRequest("Некорректный email или
пароль"));
      }
      const candidate = await User.findOne({ where: { email } });
      if (candidate) {
        return next(ApiError.badRequest("Пользователь уже
существует"));
      }

      // Генерация кода подтверждения
      const verificationCode = generateVerificationCode();

      // Устанавливаем дату истечения срока действия кода
      (например, 10 минут)
      const expirationDate = new Date();
      expirationDate.setMinutes(expirationDate.getMinutes() + 10);
      // Код действителен 10 минут

      // Сохраняем код в таблице VerificationCode
      await VerificationCode.create({
        email,
        verificationCode,
        expirationDate,
      });

      // Настройка транспорта для отправки email
      let transporter = nodemailer.createTransport({
        host: "smtp.mail.ru",
        port: "465",
        secure: true,
        auth: {
          user: "doc_finder@inbox.ru",
          pass: "z7EJ5ra4DGeCdAPT4rYQ",
        },
      });
    } catch (error) {
      return next(ApiError.badRequest(error.message));
    }
  }
}

```

```

    },
  });

  // Отправка письма с кодом подтверждения
  await transporter.sendMail({
    from: "doc_finder@inbox.ru",
    to: email,
    subject: "Подтверждение регистрации",
    text: `Ваш код подтверждения: ${verificationCode}`,
  });

  return res.json({
    message: "На вашу почту был отправлен код для
подтверждения.",
  });
} catch (e) {
  console.error(e);
}
}
async verifyCode(req, res, next) {
  const { email, verificationCode, role, password } = req.body;

  const codeRecord = await VerificationCode.findOne({
    where: { email, verificationCode },
  });

  if (!codeRecord) {
    return res.json({ message: "Неверный код подтверждения" });
  }

  // Проверяем, не истек ли срок действия кода
  if (new Date() > new Date(codeRecord.expirationDate)) {
    return res.json({ message: "Срок действия кода подтверждения
истек" });
  }

  const hashPassword = await bcrypt.hash(password, 5);
  const user = await User.create({ email, role, password:
hashPassword });
  const patient = await Patient.create({ userId: user.id });
  // Создаем JWT токен
  const token = generateJwt(user.id, user.email, user.role);

  // Удаляем код из временной таблицы
  await VerificationCode.destroy({ where: { email,
verificationCode } });

  return res.json({ message: "Регистрация успешна", token });
}
async login(req, res, next) {

```

```

const { email, password } = req.body;
const user = await User.findOne({ where: { email } });
if (!user) {
    return next(ApiError.internal("Пользователь не найден"));
}
let comparePassword = bcrypt.compareSync(password,
user.password);
if (!comparePassword) {
    return next(ApiError.internal("Неверный пароль"));
}
const token = generateJwt(user.id, user.email, user.role);
return res.json({ token });
}

async check(req, res) {
    const token = generateJwt(req.user.id, req.user.email,
req.user.role);
    return res.json({ token });
}

async getDoctors(req, res, next) {
    try {
        const doctors = await Doctor.findAll({
            include: [
                { model: Specialization, attributes: ["specialization"]
            }, // Добавляем специализацию в ответ
            ],
        });

        // Проходим по всем врачам и фильтруем их date_table
        const today = new Date();
        today.setHours(0, 0, 0, 0); // Устанавливаем начало дня
        (чтобы сравнивать только по датам)

        for (let doctor of doctors) {
            if (doctor.date_table) {
                const table = JSON.parse(doctor.date_table); // Парсим
строку с таблицей

                // Фильтруем записи, оставляя только те, которые равны
или позже текущей даты
                const filteredTable = Object.fromEntries(
                    Object.entries(table).filter(
                        ([dateStr]) => new Date(dateStr) >= today
                    )
                );

                // Если таблица изменилась, сохраняем ее
                if (JSON.stringify(filteredTable) !== doctor.date_table)
            }

```

```

        doctor.date_table = JSON.stringify(filteredTable);
        await doctor.save(); // Сохраняем изменения
    }
}

return res.json(doctors); // Отправляем список врачей с
обновленной date_table
} catch (error) {
    next(error); // Обработка ошибок
}
}

async getSpecializations(req, res, next) {
    try {
        const specializations = await Specialization.findAll(); //
Получаем все специализации
        return res.json(specializations); // Отправляем их в ответе
    } catch (error) {
        next(error); // Обрабатываем ошибки
    }
}

async doctorCertificates(req, res) {
    try {
        const { filePath } = req.body;
        const Path = filePath.replace(/^\\/, "");
        // Путь к папке с сертификатами
        const certificatesFolderPath = path.resolve(__dirname, "..",
Path);

        // Получаем список файлов в папке сертификатов
        const files =
fs.readdirSync(certificatesFolderPath).map((file) => {
            return `${filePath}/${file}`;
        });

        return res.json({ certificates: files });
    } catch (error) {
        console.error("Ошибка при получении сертификатов:", error);
        res.status(500).json({ message: "Ошибка сервера" });
    }
}

async getDoctorSchedule(req, res) {
    try {
        const doctorId = req.params.doctorId;

        // Получаем все записи для этого врача
        const entries = await Entry_table.findAll({

```

```

        where: { doctorId: doctorId },
        attributes: ["date", "time"], // Получаем только дату и
        время
    });

    // Преобразуем данные в формат [дата- время]
    const occupiedSlots = entries.map(
        (entry) => `${entry.date}-${entry.time}`
    );

    return res.json(occupiedSlots); // Отправляем список занятых
    слотов
    } catch (e) {
        console.error("Ошибка при получении расписания врача:", e);
        return res
            .status(500)
            .json({ message: "Ошибка сервера", error: e.message });
    }
}

async getDoctorDataById(req, res) {
    try {
        const doctorId = req.params.doctorId;
        const doctor = await Doctor.findOne({
            where: { id: doctorId },
            include: [
                { model: Specialization, attributes: ["specialization"]
            }, // Добавляем специализацию в ответ
            ],
        });
        return res.json(doctor);
    } catch (e) {
        return res.status(404).json({ message: "Доктор не найден"
    });
    }
}

module.exports = new UserController();

```

Файл App.js

```

import React, { useContext, useEffect, useState } from "react";
import { BrowserRouter, data } from "react-router-dom";
import "./styles/styles.css";
import AppRouter from "./components/AppRouter";
import { observer } from "mobx-react-lite";
import { Context } from "./index";
import { check } from "./http/userAPI";

```

```

const App = observer(() => {
  const { user } = useContext(Context);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    check().then(data => {
      user.setUser(data); // желательно сохранять реальные данные
      пользователя
      user.setIsAuth(true);
      if(data.role === "DOCTOR") {
        user.setIsDoctor(true);
      }
      if(data.role === "ADMIN") {
        user.setIsAdmin(true);
      }
    }).catch(() => {
      user.setUser({});
      user.setIsAuth(false);
      user.setIsDoctor(false);
      localStorage.removeItem('token');
    }).finally(() => {
      setLoading(false);
    });
  }, []);
  return (
    <BrowserRouter>
      <AppRouter />
    </BrowserRouter>
  );
});

export default App;

```

Файл userAPI.js

```

import { $host, $authHost } from "../http";
import { jwtDecode } from "jwt-decode";

export const registration = async (email, password) => {
  const { data } = await $host.post("api/user/registration", {
    email,
    password,
    role: "USER",
  });
  return data;
};

export const login = async (email, password) => {
  const { data } = await $host.post("api/user/login", { email,
    password });

```

```

    localStorage.setItem("token", data.token);
    return jwtDecode(data.token);
  };
export const verifyCode = async (email, verificationCode,
password) => {
  const { data } = await $host.post("api/user/verify-code", {
    email,
    verificationCode,
    password,
    role: "USER",
  });
  if (data.token) {
    localStorage.setItem("token", data.token);
  }
  return data.message
};
export const check = async () => {
  const { data } = await $authHost.get("api/user/auth");
  localStorage.setItem("token", data.token);
  return jwtDecode(data.token);
};

export const fetchDoctors = async () => {
  const { data } = await $host.get("api/user/doctors");
  return data;
};

export const fetchSpecializations = async () => {
  const { data } = await $host.get("api/user/specializations"); //
Предполагаем, что эндпоинт для получения специализаций –
/api/specializations
  return data;
};

export const doctorCertificates = async (filePath) => {
  try {
    const { data } = await
$host.post("api/user/doctorCertificates", {
      filePath,
    });
    return data;
  } catch (error) {
    throw new Error("Ошибка при получении сертификатов");
  }
};
export const getDoctorSchedule = async (doctorId) => {
  const { data } = await
$host.get(`api/user/schedule/${doctorId}`);
  return data;
};

```

```

};

export const getDoctorDataById = async (doctorId) => {
  const { data } = await
$host.get(`api/user/getDoctor/${doctorId}`);
  return data;
};

export const getDocReviews = async (doctorId) => {
  const { data } = await
$host.get(`api/user/getReviews/${doctorId}`);
  return data;
};

```

Файл docAPI.js

```

import { $host, $authHost } from "../http";

export const saveInfo = async (info) => {
  const { surname, name, lastname, location, image } = info;
  const formData = new FormData();
  formData.append("surname", surname);
  formData.append("name", name);
  formData.append("lastname", lastname);
  formData.append("location", location);
  if (image) {
    formData.append("image", image);
  }

  const { data } = await $authHost.post("api/doc/setInfo",
formData, {
    headers: {
      "Content-Type": "multipart/form-data",
    },
  });

  return data;
};

export const getInfo = async () => {
  const { data } = await $authHost.get("api/doc/getInfo");
  return data;
};

export const saveDataTable = async (date) => {
  const { data } = await $authHost.post("api/doc/saveDataTable", {
date });
  return data;
};

```

					ПГУ1.090301.11.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

```

export const getDateTable = async () => {
  const { data } = await $authHost.get("api/doc/getDateTable");
  return data;
};

export const saveCertificates = async (formData) => {
  try {
    const { data } = await $authHost.post(
      "api/doc/uploadCertificates",
      formData,
      {
        headers: {
          "Content-Type": "multipart/form-data",
        },
      }
    );
    return data;
  } catch (error) {
    throw new Error("Ошибка при загрузке сертификатов");
  }
};

export const getCertificates = async () => {
  try {
    const { data } = await
    $authHost.get("api/doc/getCertificates");
    return data;
  } catch (error) {
    throw new Error("Ошибка при получении сертификатов");
  }
};

export const deleteCertificate = async (filePath) => {
  try {
    const { data } = await $host.post("api/doc/deleteCertificate",
    {
      filePath,
    });
    return data;
  } catch (error) {
    throw new Error("Ошибка при удалении сертификата");
  }
};

export const getReviews= async()=>{
  const {data} = await $authHost.get("api/doc/getReviews")
  return data;
}

```

```

export const getPatientBySlot = async( time, dateKey)=>{
  const {data} = await
  $authHost.get(`api/doc/getPatientBySlot/${dateKey}/${time}`)
  return data;
}
export const cancelAppointment = async
(appointmentId, cancelReason) =>{
  const {data} = await $authHost.post("api/doc/cancelEntry",
{appointmentId, cancelReason})
}

```

Файл doctor_list_page.jsx

```

import React, { useState, useEffect } from "react";
import Search from "../components/search";
import Header from "../components/header";
import DoctorList from "../components/doctorList";
import DocMap from "../components/doc_map";
import { fetchDoctors } from "../http/userAPI"; // Импортируем
функцию для запроса врачей
import "../styles/search.css";
import { useLocation } from "react-router-dom";

const DoctorListPage = () => {
  const [doctors, setDoctors] = useState([]); // Хранение списка
врачей
  const [filteredDoctors, setFilteredDoctors] = useState([]); //
Хранение отфильтрованных врачей
  const [loading, setLoading] = useState(true); // Статус загрузки
  const location = useLocation();
  const [userLocation, setUserLocation] = useState(null);
  const [filters, setFilters] = useState({ specializations: [],
experience: [] });
  useEffect(() => {
    const loadDoctors = async () => {
      try {
        const fetchedDoctors = await fetchDoctors(); // Запрос
списка врачей
        setDoctors(fetchedDoctors); // Обновляем состояние
        setFilteredDoctors(fetchedDoctors); // Устанавливаем
изначально все врачи как отфильтрованные
      } catch (error) {
        console.error("Ошибка при загрузке врачей:", error);
      } finally {
        setLoading(false); // Завершаем загрузку
      }
    };
  });
}

```

```

    loadDoctors(); // Вызов функции загрузки врачей
}, []);

const handleSearch = (query) => {
    const filteredByQuery = doctors.filter((doctor) =>
        (doctor.Surname + " " + doctor.Name + " " + doctor.LastName)
            .toLowerCase()
            .includes(query.toLowerCase())
    );

    // Применяем фильтрацию через фильтры и потом поиск
    const finalFilteredDoctors = filteredByQuery.filter((doctor)
=> {
        const { specializations, experience } = filters; // Получаем
        примененные фильтры

        const matchesSpecialization = specializations.length
            ? specializations.some((specialization) =>
                doctor.specialization.specialization
                    .toLowerCase()
                    .includes(specialization.toLowerCase())
            )
            : true;

        const matchesExperience = experience.includes("lessThan10")
            ? doctor.experience < 10
            : experience.includes("moreThan10")
            ? doctor.experience >= 10 && doctor.experience < 20
            : experience.includes("moreThan20")
            ? doctor.experience >= 20
            : true;

        return matchesSpecialization && matchesExperience;
    });

    setFilteredDoctors(finalFilteredDoctors);
};

const handleFilter = (filters) => {
    setFilters(filters); // Обновляем фильтры
    const { specializations, experience } = filters;
    const filtered = doctors.filter(doctor => {
        const matchesSpecialization = specializations.length
            ? specializations.some(specialization =>
                doctor.specialization.specialization.toLowerCase().includes(specialization.toLowerCase()))
            : true;
    });
};

```

```

        const matchesExperience =
            experience.includes('lessThan10') ? doctor.experience < 10
:
            experience.includes('moreThan10') ? doctor.experience >=
10 && doctor.experience < 20 :
            experience.includes('moreThan20') ? doctor.experience >=
20 : true;

        return matchesSpecialization && matchesExperience;
    });
    setFilteredDoctors(filtered);
};

if (loading) {
    return <div>Загрузка...</div>; // Показываем текст загрузки
}

return (
    <div>
        <Header />
        <div id="main">
            <div
                style={{ width: "100%", height: "10em", backgroundColor:
"#7cb2ca" }}
            ></div>
            <Search
                onSearch={handleSearch}
                onFilter={handleFilter}
                setUserLocation={setUserLocation}
            />{" "}
            {/* Передаем функцию поиска и фильтрации в компонент */}
            {location.pathname === "/map" ? (
                <DocMap doctors={filteredDoctors}
userLocation={userLocation} /> // Если путь "/map", рендерим
компонент с картой
            ) : (
                <DoctorList doctors={filteredDoctors} /> // Если не
"/map", показываем список врачей
            )}
        </div>
    </div>
);
};

export default DoctorListPage;

```

Файл doctor_page.jsx

```

import React, { useState, useEffect } from "react";
import { useLocation } from "react-router-dom";
import Header from "../components/header";
import TimeTable from "../components/EntryTable";
import "../styles/doctor-page.css";
import {
  doctorCertificates,
  getDoctorSchedule,
  getDoctorDataById,
} from "../http/userAPI";
import { YMaps, Map, Placemark } from "@pbe/react-yandex-maps";
import { useContext } from "react";
import { Context } from "../index";
import "../styles/filter-modal.css"; // Импортируем стили для
модального окна
import { saveEntry } from "../http/patientApi"; // Импортируем
функцию для сохранения записи
import Reviews from "../components/reviews";
import Footer from "../components/Footer";

const DoctorPage = () => {
  const location = useLocation();
  const { doctor } = location.state || {}; // Получаем данные
врача

  const { user } = useContext(Context); // Получаем информацию о
пользователе
  const [doctorData, setDoctorData] = useState(doctor || null); //
Состояние для данных врача
  const Admin = user.isAdmin;
  const [dates, setDates] = useState(null);
  const [scheduledTimes, setScheduledTimes] = useState({}); // Все
записи
  const [certificates, setCertificates] = useState([]);
  const [modalData, setModalData] = useState(null); // Данные для
модального окна
  const [showModal, setShowModal] = useState(false); // Состояние
для отображения модального окна
  const [coordinates, setCoordinates] = useState(null);
  const [isBookingSuccess, setIsBookingSuccess] = useState(false);
  // Состояние для успешности записи
  const [occupiedSlots, setOccupiedSlots] = useState([]); //
Занятые слоты
  const [currentPage, setCurrentPage] = useState(1); // Текущая
страница
  const [itemsPerPage] = useState(3); // Количество сертификатов
на странице
  useEffect(() => {
    const { id_doctor } = location.state || {};

```

```

    if (id_doctor && !doctorData) {
        const fetchDoctorData = async () => {
            try {
                const data = await getDoctorDataById(id_doctor); //
Запрос на получение данных врача по ID
                setDoctorData(data);
            } catch (error) {
                console.error("Ошибка при получении данных врача:",
error);
            }
        };

        fetchDoctorData();
    }
    if (doctorData) {
        // Если у нас есть данные врача, то извлекаем и геокодируем
адрес
        if (doctorData.location) {
            fetchCoordinatesFromAddress(doctorData.location);
        }
    }
    if (doctorData && doctorData.date_table) {
        const data = JSON.parse(doctorData.date_table);
        if (Object.keys(data).length) {
            const datesArr = Object.keys(data)
                .map((d) => new Date(d)) // Преобразуем строки в объекты
Date
                .sort((a, b) => a - b);
            setDates({ startDate: datesArr[0], endDate: datesArr.at(-
1) });
            setScheduledTimes(data);
        }
    }

    const fetchCertificates = async () => {
        try {
            const certificatesData = await doctorCertificates(
                doctorData.certificates
            );
            setCertificates(certificatesData.certificates);
        } catch (error) {
            console.error("Ошибка при получении сертификатов:",
error);
        }
    };

    if (doctorData) {
        fetchCertificates();
    }

```

```

// Получаем занятые слоты с сервера
const fetchOccupiedSlots = async () => {
  try {
    const rawSlots = await getDoctorSchedule(doctorData.id);
    // ["2025-04-23T21:00:00.000Z-13:00", ...]

    const formatted = rawSlots.map((slotString) => {
      // Разделяем строку на дату и смещение
      const [date, time] = slotString.split("-");

      return { date: date, time: time };
    });

    setOccupiedSlots(formatted); // Сохраняем
отформатированные занятые слоты
  } catch (error) {
    console.error("Ошибка при получении занятых слотов:",
error);
  }
};

fetchOccupiedSlots(); // Изначально получаем данные

// Обновляем занятые слоты каждые 5 секунд
const intervalId = setInterval(fetchOccupiedSlots, 5000);

// Очищаем интервал при размонтировании компонента
return () => clearInterval(intervalId);
}, [location.state, doctorData]);
// Функция для геокодирования адреса и получения координат
const fetchCoordinatesFromAddress = async (address) => {
  try {
    const response = await fetch(
      `https://geocode-maps.yandex.ru/1.x/?apikey=803a134d-e7f7-
4fbf-92d1-e95182a65a4e&format=json&geocode=${encodeURIComponent(
        address
      )}`
    );
    const data = await response.json();
    const coords =
      data.response.GeObjectCollection.featureMember[0]?.GeoObject.Point.pos
        .split(" ")
        .map(Number);
    if (coords) {
      setCoordinates([coords[1], coords[0]]); // Сохраняем
координаты
    }
  }
}

```

					ПГУ1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		107

```

    } catch (error) {
      console.error("Ошибка при геокодировании адреса:", error);
    }
  };

const handleCellClick = (date, time) => {
  if (!user.isAuth) {
    alert("Пожалуйста, авторизуйтесь для записи на прием.");
    return;
  }

  // Проверяем, занята ли ячейка
  const slotKey = `${date}-${time}`;
  if (occupiedSlots.includes(slotKey)) {
    alert("Этот слот уже занят.");
    return;
  }

  // Заполняем данные для модального окна
  setModalData({
    doctorName: `${doctorData.Surname} ${doctorData.Name} ${doctorData.LastName}`,
    date: date,
    time: time,
    location: doctorData.location,
  });
  setShowModal(true); // Показываем модальное окно
  setIsBookingSuccess(false); // Сброс состояния успешной записи
};

const closeModal = () => {
  setShowModal(false); // Закрытие модального окна
};

const handleSubmit = async () => {
  // Отправляем данные на сервер
  const response = await saveEntry({
    doctorId: doctorData.id, // ID врача
    date: modalData.date, // Дата приема
    time: modalData.time, // Время приема
  });

  if (response) {
    setIsBookingSuccess(true); // Успешная запись
  } else {
    alert("Произошла ошибка при записи.");
  }
};

const indexOfLastCertificate = currentPage * itemsPerPage;

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    const indexOfFirstCertificate = indexOfLastCertificate -
itemsPerPage;
    const currentCertificates = certificates.slice(
        indexOfFirstCertificate,
        indexOfLastCertificate
    );
    const paginate = (pageNumber) => setCurrentPage(pageNumber);

    return (
        <div>
            <Header />
            <div className="doctor-page">
                <div className="doctor-info">
                    <img
                        src={
                            doctorData && doctorData.photo
                                ? doctorData.photo.replace('/static/', '/media/')
                                : "img/doc.png"
                        }
                        alt={
                            doctorData
                                ? doctorData.Surname +
                                  " " +
                                  doctorData.Name +
                                  " " +
                                  doctorData.LastName
                                : "Доктор"
                        }
                        className="doctor-photo"
                    />
                    <div className="doctor-details">
                        <h1>
                            {doctorData ? doctorData.Surname : ""}{ " "}
                            {doctorData ? doctorData.Name : ""}{ " "}
                            {doctorData ? doctorData.LastName : ""}
                        </h1>
                        <p>
                            <strong>Специальность:</strong>{ " "}
                            {doctorData ?
doctorData.specialization.specialization : ""}
                        </p>
                        <p>
                            <strong>Стаж:</strong> {doctorData ?
doctorData.experience : ""}{ " "}
                            лет
                        </p>
                        <p>
                            <strong>Описание:</strong>{ " "}
                            {doctorData ? doctorData.description : ""}

```

```

        </p>
      </div>
    </div>
    {coordinates && (
      <div className="doctor-map">
        <YMaps query={{ apiKey: "803a134d-e7f7-4fbf-92d1-
e95182a65a4e" }}>
          <Map
            defaultState={{ center: coordinates, zoom: 14 }}
            width="100%"
            height="400px"
          >
            <Placemark geometry={coordinates} />
          </Map>
        </YMaps>
      </div>
    )}

    {certificates.length > 0 && (
      <div className="doctor-certificates">
        <h3>Сертификаты</h3>
        <div className="certificate-images">
          {currentCertificates.map((certificate, index) => {
            // Если сертификат является изображением
            const isImage =
              certificate.endsWith(".jpg") ||
              certificate.endsWith(".png") ||
              certificate.endsWith(".jpeg");

            return (
              <div key={index} className="certificate-item">
                {isImage ? (
                  <img
                    src={`_${certificate}_`}
                    alt={`Сертификат ${index + 1}`}
                    className="certificate-img"
                  />
                ) : (
                  <a
                    href={`_${certificate}_`}
                    target="_blank"
                    rel="noopener noreferrer"
                  >
                    {certificate.split("/").pop()}
                  </a>
                )}
              </div>
            );
          })}
        </div>
      </div>
    )}
  </div>
) }

```

```

    </div>

    {/* Пейджинг */}
    <div className="pagination">
      {Array.from({
        length: Math.ceil(certificates.length /
itemsPerPage),
      }).map((_, index) => (
        <button
          key={index}
          onClick={() => paginate(index + 1)}
          className={`page-btn ${
            currentPage === index + 1 ? "active" : ""
          }`}
        >
          {index + 1}
        </button>
      ))}
    </div>
  </div>
)}
{dates && (
  <div className="doctor-entry">
    <h2>Записаться на прием</h2>
    <TimeTable
      startDate={dates.startDate}
      endDate={dates.endDate}
      scheduledTimes={scheduledTimes}
      isEditable={false} // Отключаем редактирование
      onClick={handleCellClick}
      occupiedSlots={occupiedSlots} // Передаем занятые
СЛОТЫ
    />
  </div>
)}
<div className="doctor-reviews">
  <h2>Отзывы</h2>
  <Reviews doctorId={doctorData.id} isAdmin={Admin} />
</div>
</div>

{/* Модальное окно для записи */}
{showModal && modalData && (
  <div className="filter-modal-overlay">
    <div className="filter-modal-content">
      <div className="filter-header">
        <h2>Запись на прием</h2>
        <button className="close-btn" onClick={closeModal}>
          <i className="fa-solid fa-xmark"></i>
        </button>
      </div>
    </div>
  </div>
)}

```

Файл doc_map.jsx

					<p style="text-align: center;"><i>ПГУ1.090301.11.001 ПЗ</i></p>	Лист
						112
Изм.	Лист	№ докум.	Подпись	Дата		

```

const DocMap = ({ doctors, userLocation }) => {
  const [coordinates, setCoordinates] = useState([]);
  const [selectedDoctor, setSelectedDoctor] = useState(null); //
Храним выбранного врача
  const [selectedCoords, setSelectedCoords] = useState(null); //
Координаты выбранного Placemark
  const mapRef = useRef(null);
  const [StoredLocation, setStoredLocation] = useState();
  const routePanelRef = useRef(null);
  // Функция для получения координат для каждого врача
  useEffect(() => {
    const storedLocation = localStorage.getItem("userLocation");
    if (storedLocation) {
      const parsedLocation = JSON.parse(storedLocation); //
Преобразуем строку в массив
      setStoredLocation([parsedLocation[1], parsedLocation[0]]);
// Устанавливаем местоположение из LocalStorage
    }
  }, []);
  const fetchCoordinates = (address) => {
    if (address) {
      return fetch(
        `https://geocode-maps.yandex.ru/1.x/?apikey=803a134d-e7f7-4fbf-92d1-e95182a65a4e&format=json&geocode=${encodeURIComponent(
          address
        )}`
      )
        .then((response) => response.json())
        .then((data) => {
          const coords =
            data.response.GeoObjectCollection.featureMember[0]?.GeoObject.Point.pos
              .split(" ")
              .map(Number);
          return coords ? [coords[1], coords[0]] : null; //
[широта, долгота]
        })
        .catch((error) => {
          console.error("Ошибка при получении координат:", error);
        });
    }
    return null;
  };

  // Используем useEffect для получения координат
  useEffect(() => {
    const getDoctorCoordinates = async () => {
      const coords = await Promise.all(
        doctors.map(async (doctor) => {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        const doctorCoords = await
fetchCoordinates(doctor.location);
        return doctorCoords
            ? { id: doctor.id, coordinates: doctorCoords }
            : null;
    })
    );
    setCoordinates(coords.filter((coord) => coord !== null));
};

getDoctorCoordinates();
}, [doctors]);

// Функция для обработки клика по метке
const handlePlacemarkClick = (coordinates, doctor) => {
    setSelectedDoctor(doctor); // Устанавливаем выбранного врача
    setSelectedCoords(coordinates); // Устанавливаем координаты
выбранной метки
};

useEffect(() => {
    if (userLocation && mapRef.current) {
        mapRef.current.setCenter(
            [userLocation.latitude, userLocation.longitude],
            10
        ); // Центрируем карту на новых координатах
    }
}, [userLocation]); // Запускать при изменении местоположения
пользователя

return (
    <div
        style={{
            width: "100%",
            height: "400px",
            position: "relative",
            display: "flex",
        }}
    >
        <YMaps query={{ apikey: "803a134d-e7f7-4fbf-92d1-
e95182a65a4e" }}>
            <Map
                instanceRef={mapRef}
                defaultState={{
                    center: userLocation
                        ? [userLocation.latitude, userLocation.longitude]
                        : [55.7558, 37.6173],
                    zoom: 10,
                }}
            >

```

```

width="100%"
height="100%"
>
{coordinates.map((doctor) => (
  <Placemark
    key={doctor.id}
    geometry={doctor.coordinates}
    onClick={() =>
      handlePlacemarkClick(
        doctor.coordinates,
        doctors.find((doc) => doc.id === doctor.id)
      )
    }
  )
})}
<Placemark
  geometry={StoredLocation} // Координаты второй метки
  options={{
    iconColor: "red", // Изменяем цвет иконки
  }}
  properties={{
    balloonContentHeader: "Здесь вы", // Заголовок
    balloonContentBody: "по информации указанной в
профиле",
  }}
/>
<GeolocationControl
  options={{float:"right"}}
/>
{/* Добавление RouterPanel для маршрута */}

<RouteButton
  options={{
    float: "right",
  }}
/>
</Map>
</YMaps>

{/* Информация о врачах поверх карты, рядом с меткой */}
{selectedDoctor && selectedCoords && (
  <div
    style={{
      position: "absolute",
      backgroundColor: "transparent",
      padding: "10px",
      borderRadius: "8px",
      zIndex: 10,

```

```

        }}
    >
    <DoctorTile
        doctor={selectedDoctor}
        style={{ flexDirection: "column", width: "14em" }}
    />
</div>
)}
</div>
);
};

export default DocMap;

```

					ПГУ1.090301.11.001 ПЗ	Лист
						116
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В
(Обязательное)

Приложение «DocFinder: поиск врачей и мгновенная запись онлайн»
Презентация

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						117
Изм.	Лист	№ докум.	Подпись	Дата		



Приложение «DocFinder: Поиск врачей и мгновенная запись онлайн»

Цифровое решение для удобного взаимодействия пациентов и врачей.

Автор работы:

студент гр. 21ВВП1 Козлова Кристина Сергеевна

Руководитель:

д.т.н., профессор Гурин Евгений Иванович

Цель и задачи

DocFinder стремится значительно улучшить доступность и удобство получения квалифицированной медицинской помощи для каждого пациента, предоставляя цифровое решение для эффективного взаимодействия.

Улучшение доступности медицинской помощи

Предоставление пациентам удобного и быстрого доступа к квалифицированным врачам.

Автоматизация процесса записи

Внедрение системы мгновенной онлайн-записи для сокращения времени ожидания и административных издержек.

Оптимизация поиска врачей

Разработка интуитивно понятного интерфейса для эффективного подбора специалистов по различным критериям.

Повышение качества обслуживания

Создание платформы с отзывами и рейтингами для обеспечения прозрачности и надежности выбора врача.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.11.001 ПЗ

Лист

118

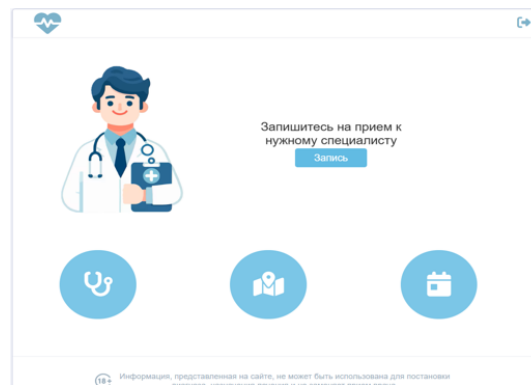
Анализ предметной области и актуальность

Проблемы традиционных методов

Запись по телефону неудобна и медленна, снижает качество услуг.
Пациентам нужен быстрый поиск и отзывы о врачах.

Потребности врачей

Автоматизация записи и цифровое присутствие важны для частных практик.
Сервис повышает доступность и качество медицинских услуг.



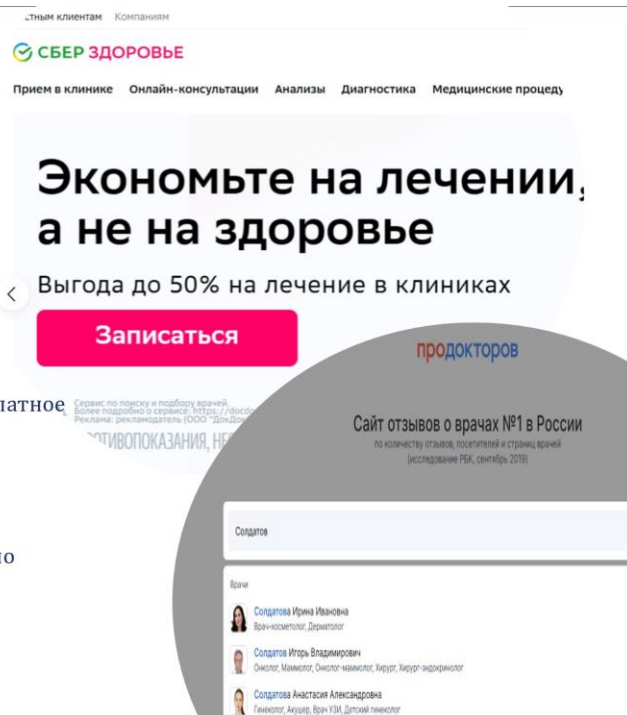
Обзор аналогов: ПроДокторов и СберЗдоровье

ПроДокторов

Большой охват, отзывы, запись онлайн, но есть платное продвижение и сомнительные отзывы.

СберЗдоровье

Онлайн консультации, простой интерфейс, но мало отзывов и навязывание услуг банка.



Изм.	Лист	№ докум.	Подпись	Дата

Технический выбор: Язык и фреймворки

JavaScript

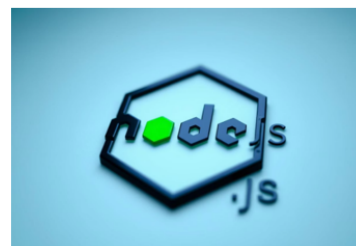
JavaScript

Единый язык для фронтенда и бэкенда, поддержка асинхронности.



React

React — гибкий и
производительный фронтенд.

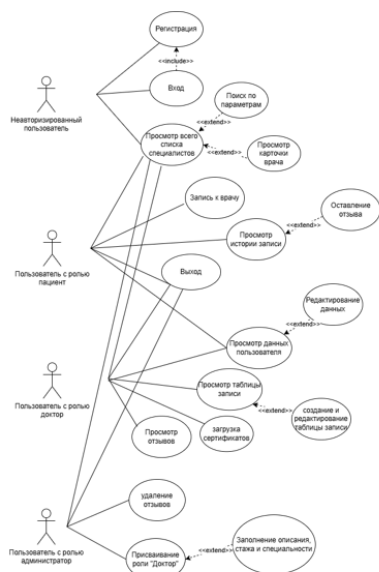


Node.js и Express.js

Node.js — кроссплатформенная среда, позволяющая разрабатывать серверные приложения.

Express — легковесный и масштабируемый серверный фреймворк.

Диаграмма прецедентов DocFinder



- Пациент:** Ищет врачей, просматривает профили, читает отзывы, записывается на прием и управляет своими записями.

- Врач:** Управляет своим расписанием, подтверждает или отклоняет записи, обновляет свой профиль и просматривает отзывы.

- Администратор:** Отвечает за управление пользователями, модерацию отзывов, управление справочниками (специальности, клиники) и общей конфигурацией системы.

Frontend приложение



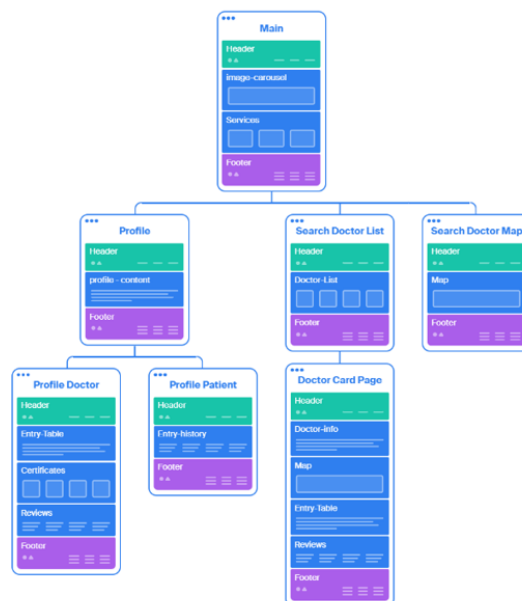
Octopus

Для разметки структуры приложения



Axure RP

Для создания прототипа



Функционал приложения

1

Поиск врачей

Фильтры по специализации, опыту и карте.

2

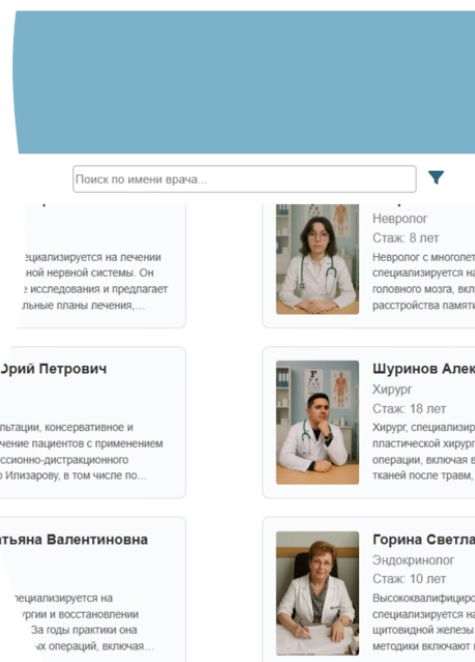
Запись на прием

Выбор времени и подтверждение с уведомлением на почту.

3

История и отзывы

Просмотр записей и возможность оставить отзыв.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.11.001 ПЗ

Лист

121



Заключение и перспективы развития

Реализация

Клиент-серверное приложение с безопасной аутентификацией и интеграцией API.

Дальнейшее развитие

Расширение функций администраторов, интеграция с Яндекс и 2ГИС.

Усиление безопасности и масштабируемости сервиса.

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Г
(справочное)

Приложение «DocFinder: поиск врачей и мгновенная запись онлайн»

Защита информационной системы персональных данных (ИСПДн)

					<i>ПГУ1.090301.11.001 ПЗ</i>	Лист
						123
Изм.	Лист	№ докум.	Подпись	Дата		

Описание объекта защиты

Информационная система предназначена для автоматизации процесса приема пациентов.

Компоненты системы:

– серверная часть:

а) размещается в отдельной серверной комнате;

б) включает сервер с базой данных (БД), где хранятся персональные данные пациентов (ФИО, полис, диагнозы, история обращений).

– клиентская часть:

а) Установлена на компьютере в комнате приема пациентов.

б) Обеспечивает взаимодействие медперсонала с сервером (поиск пациентов, запись на прием, внесение данных).

– связь с внешними системами:

а) Обмен данными осуществляется через защищенную систему ПРОМЕД (предполагается использование внутреннего сетевого протокола).

Категория обрабатываемых данных: персональные данные пациентов (ст. 3 152-ФЗ) — относятся к категории «общие», так как не включают биометрию или специальные категории.

Угрозы безопасности ИСПДн

Проведен анализ угроз в соответствии с Приказом ФСТЭК №21 и Приказом ФСТЭК №17.

Тип угрозы	Источник	Риск для конфиденциальности
Несанкционированный доступ	Внешние злоумышленники	Утечка данных через систему ПРОМЕД
Утечка данных по каналам связи	Сетевые атаки (перехват трафика)	Компрометация персональных данных
Нарушение целостности БД	Сбои оборудования/ПО	Искажение информации о пациентах

Тип угрозы	Источник	Риск для конфиденциальности
Доступ сотрудников без полномочий	Внутренние нарушители	Просмотр/изменение данных

Меры защиты ИСПДн

Технические меры:

– шифрование данных:

а) Использование протокола TLS для передачи данных между клиентом и сервером через ПРОМЕД;

б) Шифрование баз данных (например, AES-256) для защиты от несанкционированного доступа.

– аутентификация и разграничение прав:

а) двухфакторная аутентификация для доступа к серверу (логин/пароль + электронный ключ);

б) ролевая модель доступа к программе (врач, регистратор, администратор).

– защита серверной инфраструктуры:

а) установка межсетевого экрана (МЭ) между сервером и системой ПРОМЕД;

б) регулярное обновление ПО и ОС сервера (патчи безопасности).

– резервное копирование:

а) ежедневное резервирование БД на внешний носитель, хранящийся в сейфе.

Организационные меры:

– инструктаж сотрудников:

а) обучение правилам работы с персональными данными (ст. 18.1 152-ФЗ);

б) запрет на использование внешних носителей в рабочей зоне.

– контроль физического доступа:

					ПГУ1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		125

а) Серверная комната оборудована системой контроля доступа (электронные ключи);

б) Видеонаблюдение в помещениях с доступом к ИСПДн.

Документационное обеспечение:

– Разработаны:

а) Политика обработки персональных данных;

б) Регламент резервного копирования и восстановления БД;

в) Журнал учета действий пользователей (логирование).

Соответствие требованиям законодательства

– Система соответствует 152-ФЗ «О персональных данных» и Постановлению Правительства №1119;

– Проведена аттестация ИСПДн по 4 уровню защищенности (Приказ ФСТЭК №21):

а) Установлены СЗИ: антивирус (Kaspersky Endpoint Security), DLP-система;

б) Реализованы меры по контролю целостности данных (хеширование).