

Содержание

Введение	5
1 Постановка задачи и обзор аналогичных решений	6
1.1 Постановка задачи	6
1.2 Обзор аналогов	6
1.2.1 Интернет-ресурс «ЛОДЭ»	6
1.2.2 Интернет-ресурс «Мед Авеню»	8
1.2.3 Интернет-ресурс «MedCenter»	10
1.3 Выводы по разделу	11
2 Проектирование web-приложения	12
2.1 Функциональные возможности web-приложения	12
2.2 Проектирование базы данных	15
2.3 Архитектура web-приложения	21
2.4 Выводы по разделу	22
3 Реализация web-приложения	23
3.1 Программная платформа Node.js	23
3.2 Система управления базами данных PostgreSQL	23
3.3 Реализация базы данных	23
3.4 Программные библиотеки	29
3.5 Описание серверной части	30
3.6 Реализация функций пользователя с ролью «Гость»	32
3.6.1 Аутентификация	32
3.6.2 Регистрация	33
3.6.3 Просмотр информации о врачах	34
3.7 Реализация функций пользователя с ролью «Клиент»	34
3.7.1 Запись на приём к врачу	34
3.7.2 Отмена записи на приём к врачу	35
3.7.3 Редактирование собственного профиля	35
3.7.4 Удаление собственного профиля	36
3.8 Реализация функций пользователя с ролью «Врач»	37
3.8.1 Просмотр результатов обследований	37
3.8.2 Запись результатов обследований в медицинскую карту	38
3.8.3 Запись диагноза в медицинскую карту	38
3.8.4 Запись назначенных лекарств в медицинскую карту	39
3.9 Реализация функций пользователя с ролью «Администратор»	39
3.9.1 Добавление врача	39
3.9.2 Удаление врача	40
3.9.3 Изменение информации о враче и его графика работы	41
3.9.4 Добавление лекарств	41
3.9.5 Удаление лекарств	42
3.9.6 Изменение статуса доступности лекарств	42
3.9.7 Изменение цены лекарств	43
3.10 Структура клиентской части	43
3.11 Настройка конфигурации Docker	46

3.12	Настройка конфигурации nginx.....	49
3.13	Выводы по разделу	50
4	Тестирование web-приложения	51
4.1	Функциональное тестирование.....	51
4.2	Выводы по разделу	56
5	Руководство пользователя.....	57
5.1	Руководство для роли «Гость»	57
5.1.1	Аутентификация.....	57
5.1.2	Регистрация.....	58
5.1.3	Просмотр информации о врачах. Фильтрация	59
5.2	Руководство для роли «Клиент».....	60
5.2.1	Запись на приём к врачу	61
5.2.2	Отмена записи на приём к врачу	62
5.2.3	Редактирование собственного профиля.....	63
5.2.4	Удаление собственного профиля.....	64
5.3	Руководство для роли «Врач».....	64
5.3.1	Запись результатов медицинского обследования, диагностированных заболеваний, назначенных лекарств.....	65
5.3.2	Просмотр результатов обследований пациента	67
5.4	Руководство для роли «Администратор»	68
5.4.1	Добавление врача	69
5.4.2	Изменение врача и его графика работы.....	71
5.4.3	Удаление врача	71
5.4.4	Добавление лекарства	71
5.4.5	Изменение статуса доступности и цены лекарства	72
5.4.6	Удаление лекарства.....	73
5.5	Выводы по разделу	73
	Заключение.....	74
	Список используемых источников	75
	Приложение А.....	77
	Приложение Б	81

Введение

Медицинские центры — это учреждения, где люди могут получить медицинскую помощь и консультации. В них работают врачи различных специальностей, проводятся обследования и лечение.

Web-приложения для медицинских центров служат важным инструментом для информирования пациентов, предоставляя подробную информацию о предлагаемых услугах, расписании приема врачей и ценах, а также для записи на прием и получения результатов анализов.

Web-приложение призвано трансформировать процесс взаимодействия между пациентами и медицинским персоналом, а также оптимизировать внутренние процессы управления. Внедрение web-приложения позволяет значительно повысить уровень автоматизации процессов. Автоматизация управления расписанием и медицинскими картами способствует более эффективному использованию ресурсов медцентра.

Целью курсового проекта является упростить процесс обслуживания пациентов, ведения медкарт с помощью web-приложения медцентра. Web-приложение должно поддерживать несколько ролей: гость, клиент, врач и администратор. Гостям предоставляется возможность зарегистрироваться и ознакомиться с доступными врачами, в то время как клиенты могут записываться на прием и управлять своим профилем. Врачи, в свою очередь, получают доступ к медицинским картам пациентов и смогут вносить необходимые данные, такие как диагнозы и результаты обследований. Администраторы смогут эффективно управлять расписанием, изменять информацию о врачах и следить за актуальностью данных.

Целевая аудитория web-приложения для медицинского центра включает пациентов, врачей и администраторов. Основной группой пользователей являются пациенты, которые ищут доступные и качественные медицинские услуги. Врачи, работающие в медцентре, используют web-приложение для документирования результатов обследований и взаимодействия с пациентами. Для администраторов web-приложение должно предоставлять инструменты для добавления и редактирования информации о врачах, расписаниях и лекарствах.

Для достижения указанной цели поставлены следующие задачи.

1. проанализировать аналоги (раздел 1);
2. спроектировать web-приложение (раздел 2);
3. разработать приложение по спроектированной архитектуре (раздел 3);
4. организовать тестирование и отладку приложения (раздел 4);
5. написать руководство пользователя (раздел 5).

В качестве программной платформы выбрана Node.js 20.16.0 [1] для серверной части, React.js 18 [2] для клиентской части. Для хранения данных выбрана база данных PostgreSQL 17.2 [3].

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

Web-приложение медцентра должно обеспечить автоматизацию процесса записи пациентов на приём и ведения медкарт. Пациенты, не прошедшие аутентификацию, могут посмотреть список врачей, работающих в медцентре. Запись на приём и её отмену могут осуществлять только зарегистрированные и аутентифицированные пользователи. Они так же могут выполнять изменение и удаление своего профиля. Врачи смогут просматривать записи к себе, информацию о пациенте и историю болезней, если пациент уже посещал медцентр. Так же врачи смогут записывать в медкарту пациента результаты обследований, поставленный диагноз и прописанные лекарства. Администратор будет выполнять управление базой врачей и лекарств: добавлять, удалять или изменять информацию о враче, менять их график работы, изменять доступность лекарств и их цены.

1.2 Обзор аналогов

1.2.1 Интернет-ресурс «ЛОДЭ»

Одним из самых популярных на данный момент аналогов является web-сайт «ЛОДЭ» [4], главная страница которого представлена на рисунке 1.1.

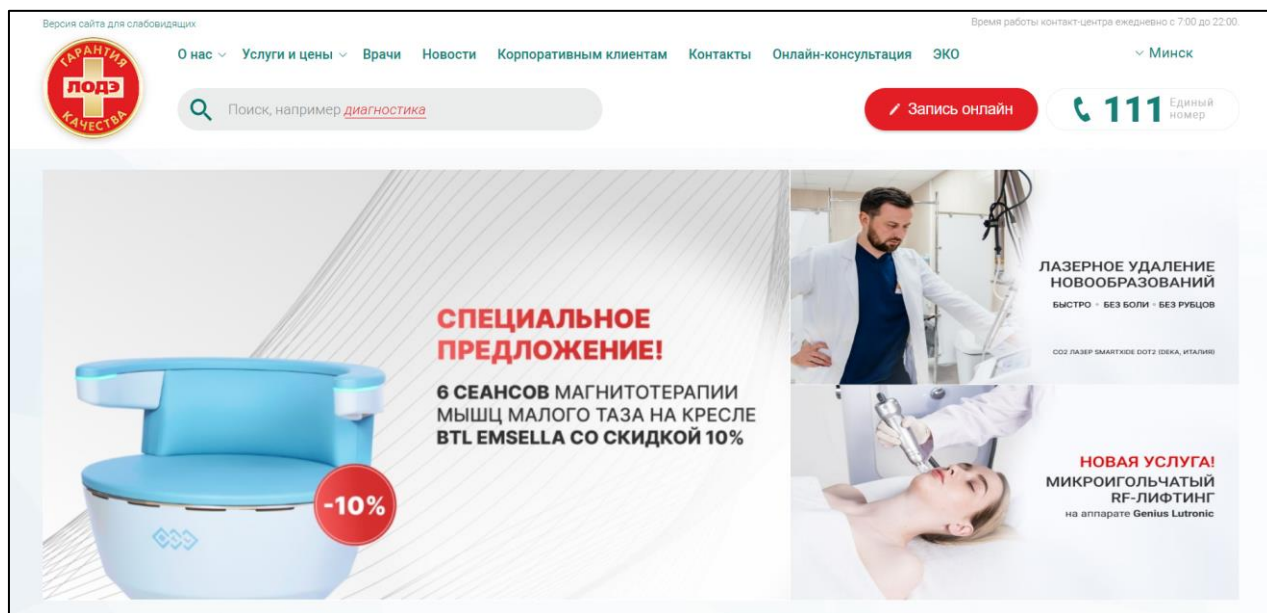


Рисунок 1.1 – Главная страница web-сайта «ЛОДЭ»

Пациент может посмотреть информацию о врачах и выбрать подходящего, используя фильтрацию списка по специализации врача и его типу. Пример страницы со списком информации о врачах представлен на рисунке 1.2.

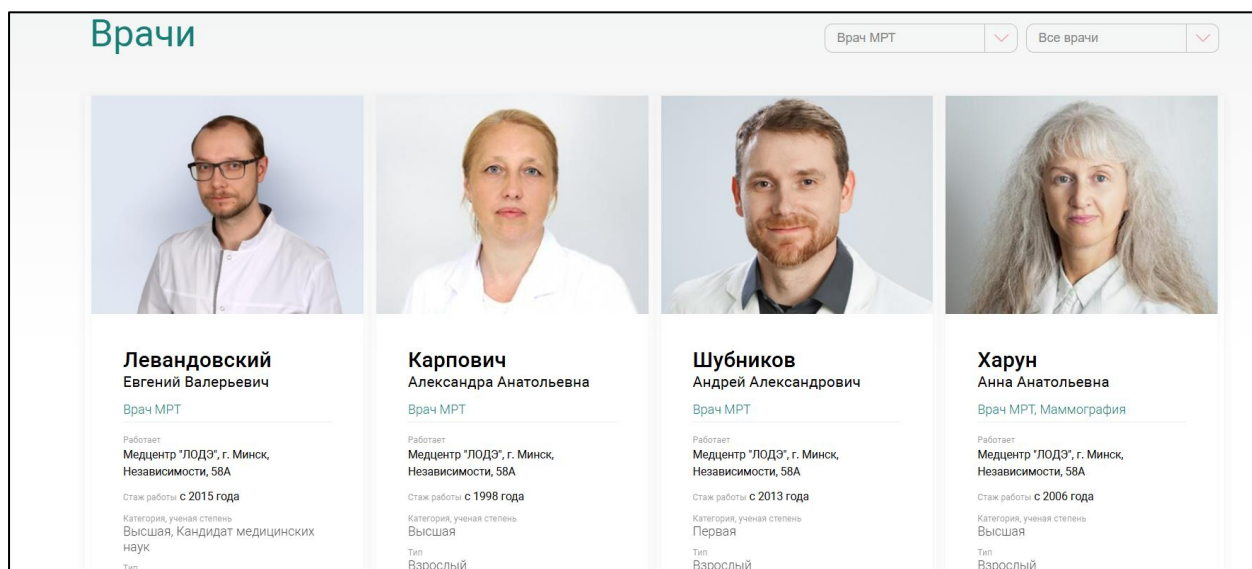


Рисунок 1.2 – Страница с информацией о врачах медцентра

На главной странице пациент может записаться на приём к определённому врачу, выбрав нужную информацию. Пример записи представлен на рисунке 1.3.

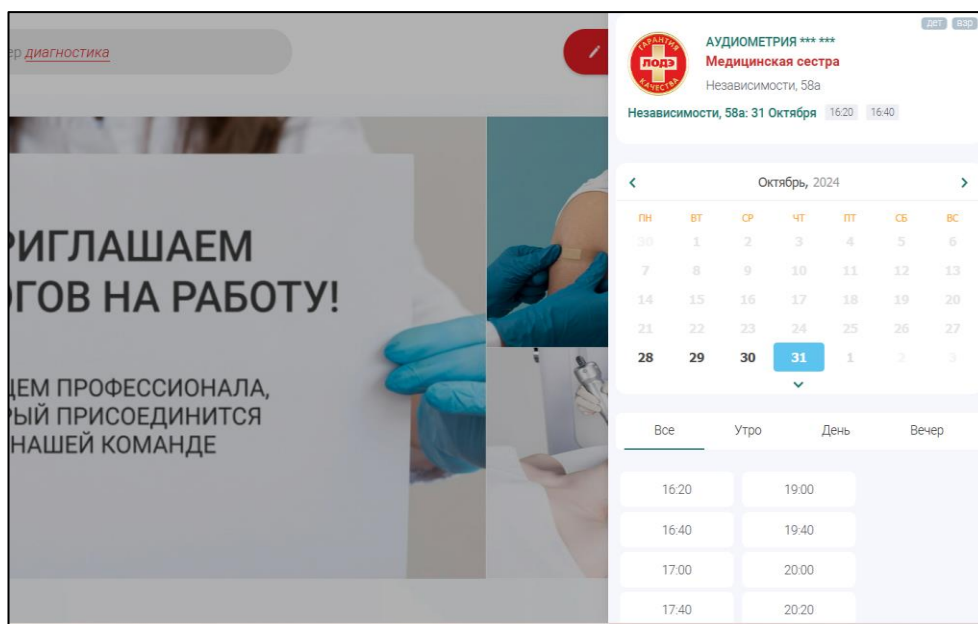


Рисунок 1.3 – Запись на прием к врачу

Сайт медицинского центра «ЛОДЭ» использует CMS 1С-Битрикс для управления контентом. Web-сервер Nginx [5] обеспечивает высокую производительность, а язык программирования PHP отвечает за динамическую генерацию страниц. Фронтенд реализован с помощью библиотеки jQuery, что позволяет создавать интерактивные элементы. Для выявления используемых технологий при создании данного аналога и последующих рассмотренных был использован специальный сервис PR-CY [6]. При запросе несуществующей страницы сервер возвращает ошибку 404, пример страницы представлен на рисунке 1.4.

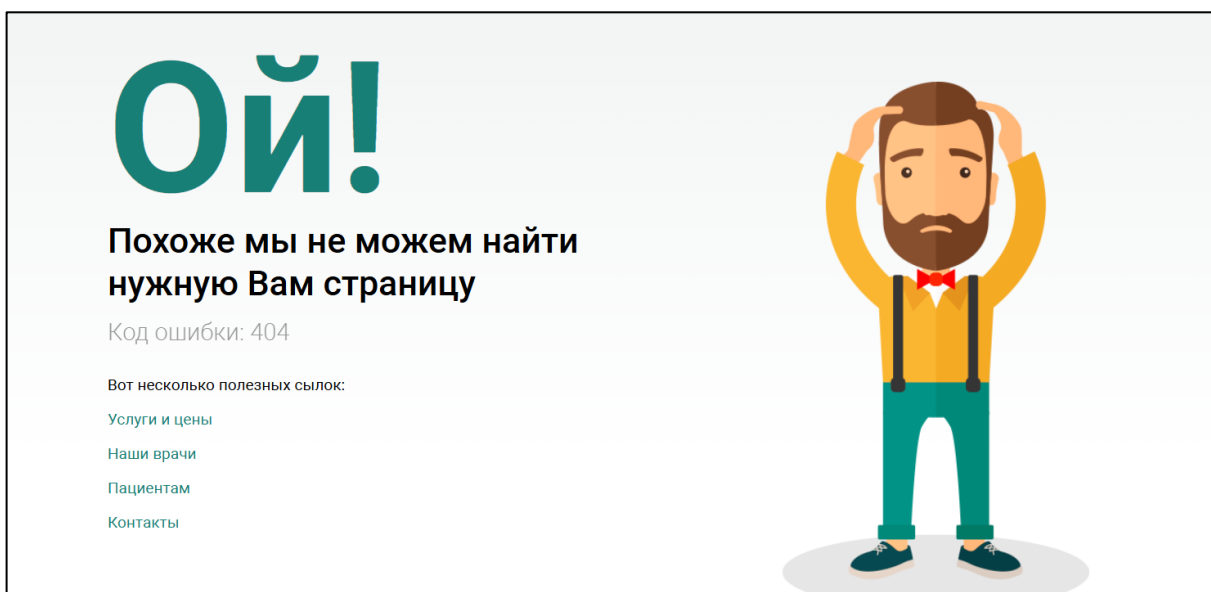


Рисунок 1.4 – Страница ошибки 404

Из недостатков этого аналога можно отметить отсутствие возможности создать личный аккаунт пользователя и, следовательно, неосуществимость отмены записи к врачу через ресурс.

1.2.2 Интернет-ресурс «Мед Авеню»

Следующим аналогом был выбран web-сайт медцентра «Мед Авеню» [7], который также является довольно посещаемым. Пример интерфейса главной страницы представлен на рисунке 1.5

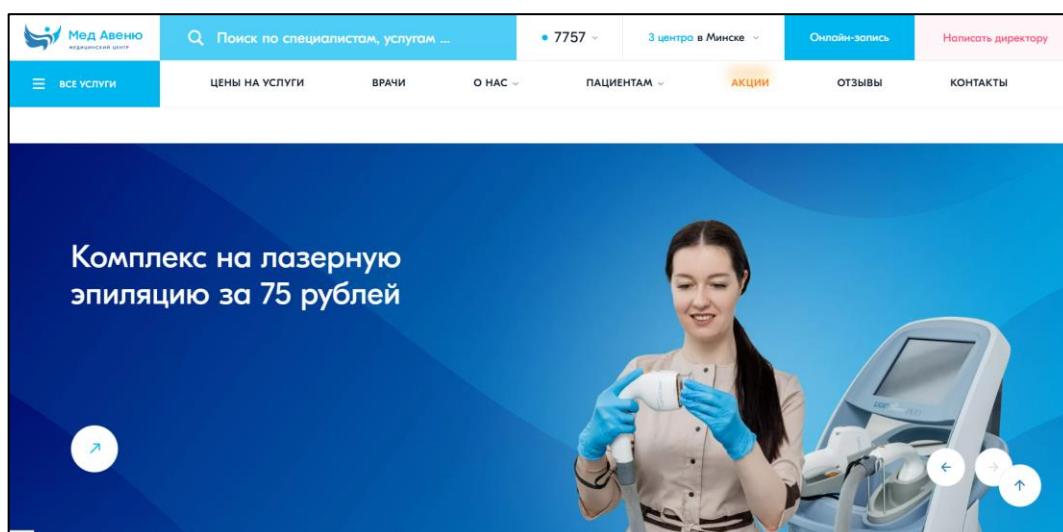


Рисунок 1.5 – Главная страница web-сайта «Мед Авеню»

Простота интерфейса позволяет пользователям быстро находить необходимую информацию. Включение функции поиска и фильтрации услуг и врачей делает процесс более интуитивным и удобным, что, в свою очередь, повысит уровень удовлетворенности клиентов. Система онлайн-записи на прием значительно

упрощает процесс взаимодействия с медицинским центром. Пример записи представлен на рисунке 1.6.

The screenshot shows a web form for booking a medical appointment. At the top, the title 'Запись на прием' is centered. Below it are two dropdown menus: the first is set to 'Хирург' and the second to 'Любой врач'. Under these is a section titled 'Доступное время' (Available time) with a dropdown for 'Медицинский центр'. This section contains three boxes: 'Сегодня Талонов нет', 'Завтра Талонов нет', and 'На 7 дней Талонов Более 100'. Below this is a section 'Ближайшие талоны' (Upcoming tickets) showing a specific appointment: '16:20 3 Апреля' and 'Гаврилик Андрей Минск. Грибоедова 11'. At the bottom are two blue buttons: 'Выбрать время' and 'Любое время', followed by a link 'Позвонить' with an upward arrow icon.

Рисунок 1.6 – Запись на прием к врачу

Сайт использует CMS WordPress, что позволяет легко управлять контентом и обновлять страницы. В качестве базы данных применяется MySQL. Для видео-контента интегрирован YouTube. Разработка страниц осуществляется с помощью конструктора uKit и плагина WP Rocket, что улучшает скорость загрузки. На фронтенде задействованы библиотеки JavaScript, такие как GSAP для анимации и Swiper для создания слайдеров. Web-сервер Nginx обеспечивает надежную работу сайта. При запросе несуществующей страницы сервер также возвращает ошибку 404, пример страницы представлен на рисунке 1.7.

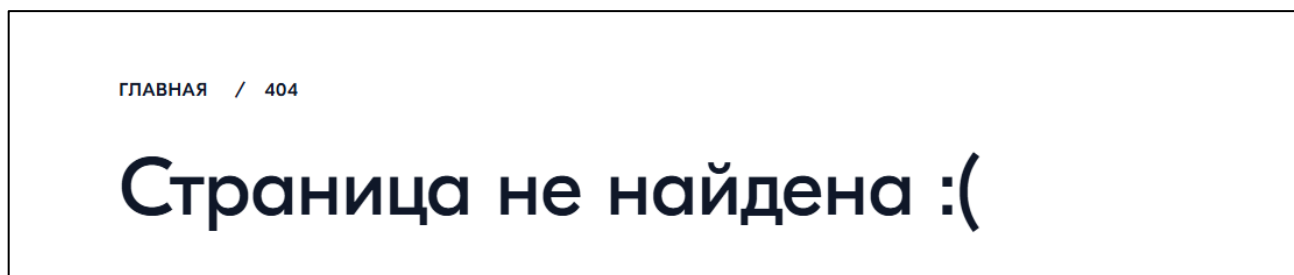


Рисунок 1.7 – Страница ошибки 404

Из недостатков можно выделить то, что статические ресурсы не оптимизированы. Во время записи на приём к врачу проверка существующих записей

для рендера свободных талонов длилась довольно длительное время. Размер ресурса можно уменьшить, удалив ненужные элементы страницы, например, лишние пробелы, переносы строки и отступы.

1.2.3 Интернет-ресурс «MedCenter»

Из популярных зарубежных аналогов был выбран web-сайт «MedCenter» [8], интерфейс главной страницы представлен на рисунке 1.8.



Рисунок 1.8 – Главная страница web-сайта «MedCenter»

На главной странице клиенту предоставлена возможность записаться на приём, связавшись с нужным медицинским центром. Для записи нужно подождать своей очереди в списке ожидающих, после чего клиента свяжут с центром. Это может быть как преимуществом, так и недостатком. С одной стороны, упрощение процесса записи позволяет избежать долгих поисков и дает возможность быстро получить помощь. С другой стороны, ожидание своей очереди может вызвать неудобства и продлить процесс обращения за медицинской помощью. В отличие от некоторых других медицинских центров, где предлагаются специальные предложения и разнообразные услуги, web-сайт «MedCenter» сосредоточен на более ограниченном наборе возможностей.

Сайт построен на CMS WordPress, что обеспечивает гибкость и удобство управления контентом. В качестве базы данных используется MySQL, а для интеграции карт применяются Google Maps. Язык программирования PHP отвечает за серверную логику. На фронтенде применяются AngularJS для динамического контента и различные библиотеки JavaScript, такие как jQuery и Lightbox, для улучшения пользовательского интерфейса. Web-сервер Nginx обеспечивает стабильную работу сайта. При запросе несуществующей страницы, web-сайт перенаправляет пользователя на главную. Страница ошибки 404 отсутствует.

1.3 Выводы по разделу

1. В течение разработки web-приложения для медцентра будет создана эффективная система, обеспечивающая автоматизацию процессов записи клиентов на приём к врачу и ведения медкарт пациентов. Web-приложение будет реализовывать следующий функционал.

Функции пользователя с ролью «Гость»:

- регистрация;
- аутентификация;
- просмотр информации о врачах и их поиск по направлению.

Функции пользователя с ролью «Клиент»:

- запись на приём и отмена записи к врачу;
- редактирование собственного профиля и его удаление;
- просмотр информации о врачах и их поиск по направлению.

Функции пользователя с ролью «Администратор»:

- управление врачами (добавление, удаление, изменение);
- изменение графика работы врача;
- просмотр информации о врачах;
- изменение статуса доступности лекарств и их цен.

Функции пользователя с ролью «Врач»:

- просмотр результатов обследований клиента;
- запись назначенного лекарства в медицинскую карту клиента;
- запись диагноза в медицинскую карту клиента;
- запись результатов обследований в медицинскую карту.

2. Были проанализированы аналоги Интернет-ресурсов, существующих медцентров, и выявлены как сильные, так и слабые стороны web-сайтов «ЛЮДЭ», «Мед Авеню» и «MedCenter». Среди ключевых недостатков отмечается отсутствие личного кабинета пользователя.

2 Проектирование web-приложения

2.1 Функциональные возможности web-приложения

Функциональные возможности web-приложения представлены в диаграмме вариантов использования, представленной на рисунке 2.1.

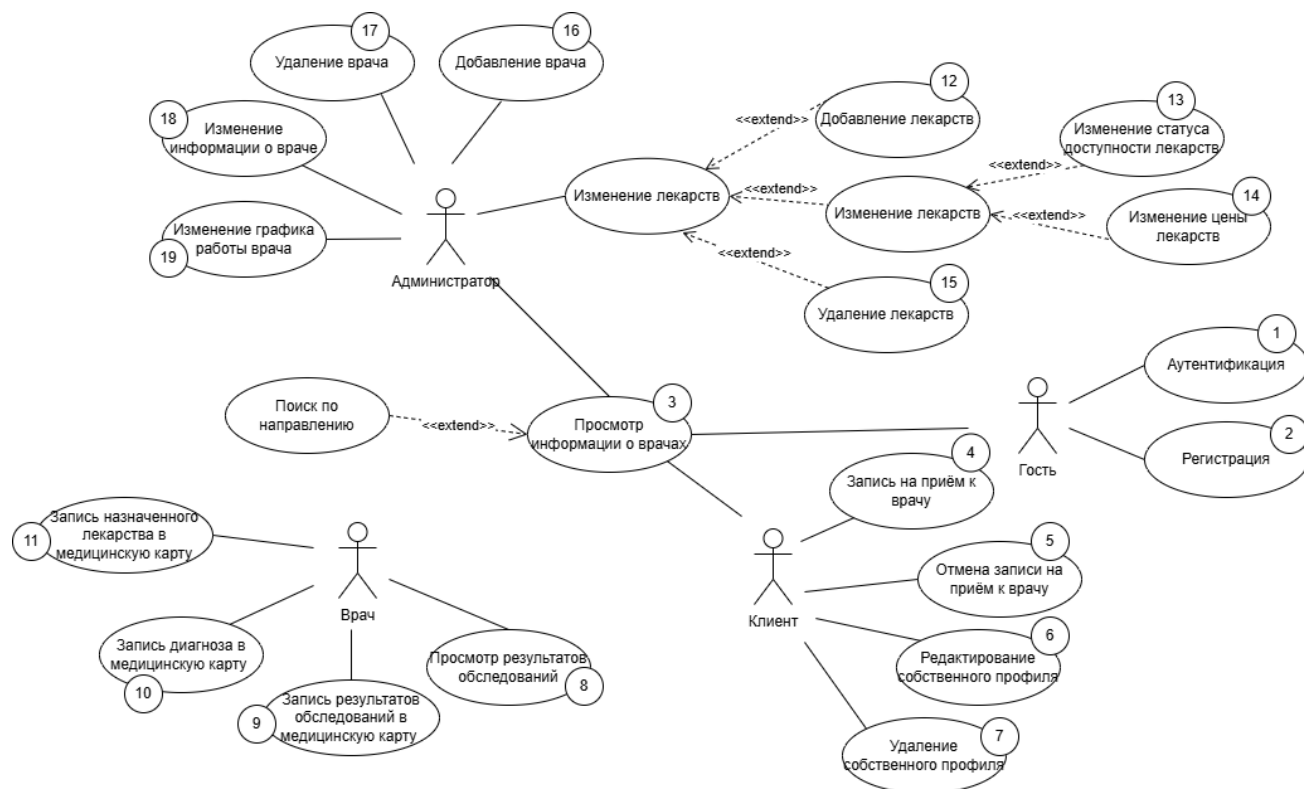


Рисунок 2.1 – Диаграмма вариантов использования web-приложения

На диаграмме вариантов использования представлен функционал web-приложения для различных ролей: гость, клиент, врач и администратор. Описание назначения ролей представлено в таблице 2.1.

Таблица 2.1 – Назначение ролей пользователей в веб-приложении

Роль	Назначение
Гость	Пользователь, не зарегистрированный в системе. Имеет доступ к базовой информации о приложении, а также может просматривать страницы без необходимости входа в систему.
Клиент	Зарегистрированный пользователь, который может записываться на прием, отменять свои записи, просматривать, управлять личными данными.
Врач	Специалист, имеющий доступ к информации о своих пациентах, может вести медицинскую документацию, назначать лечение.
Администратор	Пользователь с расширенными правами. Обеспечивает настройку пользователей, управление контентом.

Функциональные возможности пользователя с ролью «Гость» представлены в таблице 2.2.

Таблица 2.2 – Функциональные возможности

№	Вариант использования	Пояснение
1	Аутентификация	Процесс предоставления пользователям прав на выполнение определенных действий с использованием учетных данных, таких как логин и пароль.
2	Регистрация	Гость может создать учетную запись, заполнив необходимые данные, чтобы получить доступ к основному функционалу web-приложения.
3	Просмотр информации о врачах	Гость может просматривать информацию о врачах, работающих в медцентре, с возможностью поиска по их направлению.

Функциональные возможности пользователя с ролью «Клиент» представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности

№	Вариант использования	Пояснение
3	Просмотр информации о врачах	Гость может просматривать информацию о врачах, работающих в медцентре, с возможностью поиска по их направлению.
4	Запись на приём к врачу	Клиент может выбрать врача и удобное время для записи на приём. Система отображает доступные временные слоты и позволяет подтвердить запись.
5	Отмена записи на приём к врачу	Клиент может отменить ранее запланированный приём. Это действие может быть выполнено через личный кабинет.
6	Редактирование собственного профиля	Клиент может обновлять свои личные данные, такие как логин и пароль.
7	Удаление собственного профиля	Клиент может удалить свою учетную запись, если больше не желает использовать web-приложение. Данные пациента и его история болезней остаются в базе данных.

Функциональные возможности пользователя с ролью «Врач» представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности

№	Вариант использования	Пояснение
8	Просмотр результатов обследований клиента	Врач может получить доступ к результатам медицинских обследований пациента.

Продолжение таблицы 2.4

№	Вариант использования	Пояснение
9	Запись результатов обследований в медицинскую карту клиента	Врач может заносить результаты различных обследований (анализов, исследований) в медицинскую карту пациента
10	Запись диагноза в медицинскую карту клиента	Врач может фиксировать диагнозы, поставленные пациенту, в его медицинской карте.
11	Запись назначенного лекарства в медицинскую карту клиента	Врач имеет возможность добавлять информацию о назначенных лекарствах в медицинскую карту пациента.

Функциональные возможности пользователя с ролью «Администратор» представлены в таблице 2.5.

Таблица 2.5 – Функциональные возможности

№	Вариант использования	Пояснение
12	Добавление лекарств	Администратор может добавлять новую информацию о лекарственных средствах, включая их названия, фармацевтическую подгруппу и цены.
13	Изменение статуса доступности лекарств	Администратор можешь изменять статус доступности лекарства.
14	Изменение цены лекарств	Администратор можешь изменять цену лекарства.
15	Удаление лекарств	Администратор можешь удалять лекарства.
16	Добавление врача	Администратор может вносить новых врачей в систему, заполняя необходимые данные, такие как ФИО, специальность, контактная информация, день рождения и категория.
17	Удаление врача	Администратор может удалять записи о врачах из системы, если они больше не работают в учреждении. Данные о враче остаются в базе данных.
18	Изменение информации о враче	Администратор имеет возможность редактировать данные врачей.
19	Изменение графика работы врача	Администратор может корректировать расписание работы врачей, устанавливая доступные часы для записи и длительность приёма.

Таким образом для каждой роли определен набор доступных действий и возможностей. У актера «Клиент» есть смежные функции с актером «Гость», это значит, что оба пользователя имеют к ним доступ.

2.2 Проектирование базы данных

Была спроектирована база данных «medcenter», которая обеспечивает хранение и управление всей необходимой информацией для функционирования web-приложения. Структура базы данных представлена на рисунке 2.2.

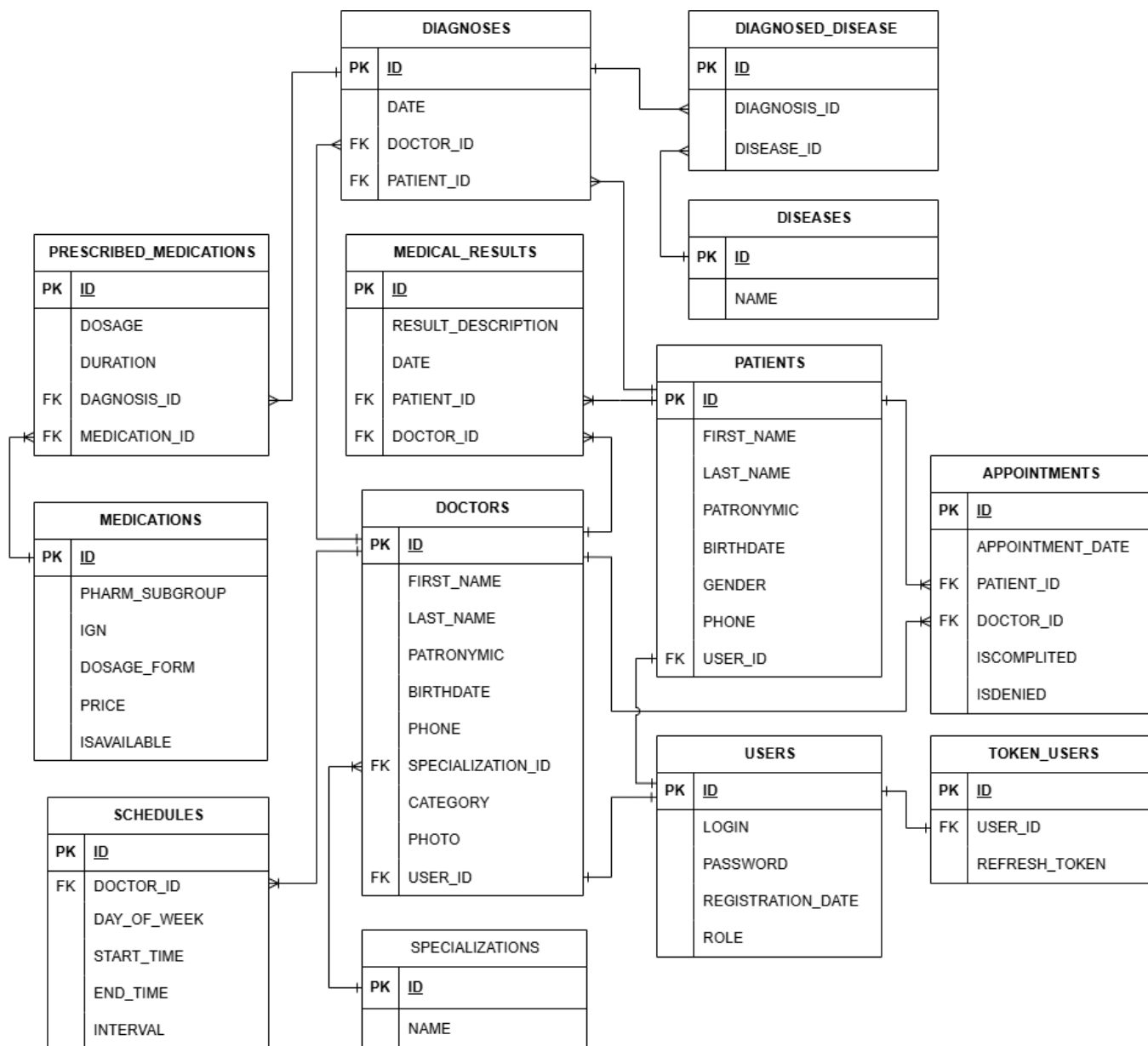


Рисунок 2.2 – Логическая схема базы данных web-приложения

База данных содержит тринадцать таблиц, назначение каждой из которых описано в таблице 2.6

Таблица 2.6 – Назначение таблиц базы данных

Таблица	Назначение
Users	Таблица, содержащая информацию о всех пользователях системы, включая логины, пароли и роли (клиент, врач, администратор). Она обеспечивает управление доступом к функционалу приложения.

Продолжение таблицы 2.6

Таблица	Назначение
Patients	Таблица, хранящая данные о пациентах. Эта таблица позволяет вести учет пациентов медцентра.
Doctors	Таблица, содержащая информацию о врачах. Она помогает организовать данные о медицинских специалистах и их квалификации.
Specializations	Таблица, хранящая названия всех специализаций врачей. Позволяет реализовать добавление новых направлений в медцентре.
Appointments	Таблица, фиксирующая записи на приём к врачам, включая дату, время и идентификаторы пациентов и врачей. Она позволяет отслеживать запланированные визиты и управлять расписанием.
Medical_Results	Таблица, в которой хранятся результаты медицинских обследований пациентов. Она обеспечивает доступ к информации, необходимой для диагностики и лечения.
Diagnoses	Таблица, фиксирующая диагнозы, поставленные врачами пациентам. Она помогает в поддержании медицинской документации и отслеживании состояния здоровья пациентов.
Diseases	Таблица, в которой содержится список заболеваний с кодом классификации.
Diagnosed_ Diseases	Таблица, которая используется для связи заболеваний и диагнозов.
Medications	Таблица, содержащая информацию о лекарствах, включая их названия и цены. Она служит справочной основой для назначения медикаментов.
Prescribed_Medications	Таблица, связывающая пациентов с назначенными им лекарствами, включая информацию о дозировках и длительности назначения. Она позволяет отслеживать лечение и лекарственные назначения.
Schedules	Таблица, фиксирующая графики работы врачей, включая дни и время приёма. Она помогает управлять расписанием и обеспечивать доступность врачей для пациентов.
Token_Users	Таблица предназначена для хранения информации о токенах, связанных с пользователями системы.

В таблице Users хранится информация о пользователях web-приложения. Содержащиеся в ней поля описаны в таблице 2.7.

Таблица 2.7 – Структура таблицы Users

Название	Тип данных	Описание
ID	uuid	Идентификатор пользователя
Login	varchar	Логин пользователя

Продолжение таблицы 2.7

Password	varchar	Пароль пользователя
Registration_Date	timestamp with time zone	Дата регистрации пользователя в системе заполняется автоматически при создании аккаунта
Role	varchar	Роль пользователя (клиент, врач, администратор)

В таблице Patients хранится информация о пациентах медцентра. Содержащиеся в ней поля описаны в таблице 2.8.

Таблица 2.8 – Структура таблицы Patients

Название	Тип данных	Описание
ID	uuid	Идентификатор пациента
First_Name	varchar	Имя пациента
Last_Name	varchar	Фамилия пациента
Patronymic	varchar	Отчество пациента
Birthday	date	Дата рождения пациента
Gender	character	Пол пациента
Phone	varchar	Телефон пациента
User_ID	uuid	Идентификатор пользователя, соответствующего пациента, вторичный ключ

В таблице Doctors хранится информация о врачах, работающих в медцентре. Содержащиеся в ней поля описаны в таблице 2.9.

Таблица 2.9 – Структура таблицы Doctors

Название	Тип данных	Описание
ID	uuid	Идентификатор врача
First_Name	varchar	Имя врача
Last_Name	varchar	Фамилия врача
Patronymic	varchar	Отчество врача
Birthday	date	Дата рождения врача
Gender	character	Пол врача
Phone	varchar	Телефон врача
Specialization_ID	uuid	Идентификатор направления врача, вторичный ключ
Category	varchar	Категория врача
Photo	varchar	Ссылка на фотографию в файловом хранилище
User_ID	uuid	Идентификатор пользователя, соответствующего врачу, вторичный ключ

В таблице Specializations хранятся названия допустимых направлений.

Содержащиеся в ней поля описаны в таблице 2.10.

Таблица 2.10 – Структура таблицы Specializations

Название	Тип данных	Описание
ID	uuid	Идентификатор специализации
Name	varchar	Название специализации

В таблице Appointments хранится информация о записях пациентов на приём к врачу. Содержащиеся в ней поля описаны в таблице 2.11.

Таблица 2.11 – Структура таблицы Appointments

Название	Тип данных	Описание
ID	uuid	Идентификатор записи на приём к врачу
Patient_ID	uuid	Идентификатор пациента, вторичный ключ
Doctor_ID	uuid	Идентификатор врача, вторичный ключ
Appointment_Date	timestamp with time zone	Дата и время записи
isDenied	boolean	Флаг, показывающий отменена запись или нет

В таблице Medical_Results хранится информация о медицинских обследованиях пациентов. Содержащиеся в ней поля описаны в таблице 2.12.

Таблица 2.12 – Структура таблицы Medical_Results

Название	Тип данных	Описание
ID	uuid	Идентификатор записи результата обследования
Patient_ID	uuid	Идентификатор пациента, вторичный ключ
Doctor_ID	uuid	Идентификатор врача, вторичный ключ
Result_Description	text	Описание результата обследования
Date	date	Время проведения обследования

В таблице Diagnoses хранится информация о диагнозах пациентов. Содержащиеся в ней поля описаны в таблице 2.13.

Таблица 2.13 – Структура таблицы Diagnoses

Название	Тип данных	Описание
ID	uuid	Идентификатор поставленного диагноза
Patient_ID	uuid	Идентификатор пациента, вторичный ключ
Doctor_ID	uuid	Идентификатор врача, выставившего диагноз, вторичный ключ
Date	date	Время выставления диагноза

В таблице Diseases хранится информация о базе заболеваний. Содержащиеся в ней поля описаны в таблице 2.14.

Таблица 2.14 – Структура таблицы Diseases

Название	Тип данных	Описание
ID	uuid	Идентификатор заболевания
Name	uuid	Название заболевания с кодом классификации

В таблице Diagnosed_Diseases хранится информация о диагностированных заболеваниях. Содержащиеся в ней поля описаны в таблице 2.15.

Таблица 2.15 – Структура таблицы Diagnosed_Diseases

Название	Тип данных	Описание
ID	uuid	Идентификатор диагностированного заболевания
Diagnosis_ID	uuid	Идентификатор диагноза, вторичный ключ
Disease_ID	uuid	Идентификатор заболевания, вторичный ключ

В таблице Medications хранится информация о лекарствах. Содержащиеся в ней поля описаны в таблице 2.16.

Таблица 2.16 – Структура таблицы Medications

Название	Тип данных	Описание
ID	uuid	Идентификатор лекарства
Pharm_Subgroup	varchar	Название фармацевтической подгруппы
IGN	varchar	Международное название лекарства
Dosage_Form	varchar	Форма выпуска лекарства
Price	numeric	Цена лекарства
isAvailable	boolean	Флаг, показывающий доступно лекарство или нет

В таблице Prescribed_Medications хранится информация о прописанных лекарствах. Содержащиеся в ней поля описаны в таблице 2.17.

Таблица 2.17 – Структура таблицы Prescribed_Medications

Название	Тип данных	Описание
ID	uuid	Идентификатор лекарства
Diagnosis_ID	uuid	Идентификатор диагноза, вторичный ключ
Medication_ID	uuid	Идентификатор лекарства, вторичный ключ
Dosage	varchar	Доза лекарства
Duration	integer	Длительность приёма лекарства

В таблице Schedules хранится информация о расписаниях работы врачей. Содержащиеся в ней поля описаны в таблице 2.18.

Таблица 2.18 – Структура таблицы Schedules

Название	Тип данных	Описание
ID	uuid	Идентификатор лекарства
Doctor_ID	uuid	Идентификатор доктора, вторичный ключ
Day_of_week	varchar	День недели
Start_time	time without time zone	Время начала смены врача
End_time	time without time zone	Время конца смены врача
Interval	time without time zone	Интервал приёма пациента

В таблице Token_Users хранится информация о токенах пользователей нужных для JWT [9] авторизации. Содержащиеся в ней поля описаны в таблице 2.19.

Таблица 2.19 – Структура таблицы Token_Users

Название	Тип данных	Описание
ID	uuid	Идентификатор лекарства
User_ID	uuid	Идентификатор пользователя, вторичный ключ
RefreshToken	varchar	Refresh токен

Связи между всеми таблицами базы данных медцентра предоставлены в таблице 2.20.

Таблица 2.20 – Назначение связей между таблицами

Таблица источник	Связанная таблица	Тип связи	Описание
Users	Patients	Один к одному	К одному пациенту привязан один аккаунт пользователя
Users	Doctors	Один к одному	К одному доктору привязан один аккаунт пользователя
Users	Token_Users	Один к одному	К одному пользователю привязана одна запись с токеном
Patients	Appointments	Один ко многим	У одного пациента может быть несколько записей к врачу
Patients	Diagnoses	Один ко многим	У одного пациента может быть несколько диагнозов
Patients	Medical_Results	Один ко многим	У пациента может быть множество результатов обследований
Doctors	Appointments	Один ко многим	К доктору может быть несколько записей на обследование

Продолжение таблицы 2.20

Таблица источник	Связанная таблица	Тип связи	Описание
Doctors	Medical_Results	Один ко многим	Доктор может ставить записывать в медкарту результатов медицинских обследований
Doctors	Diagnoses	Один ко многим	Доктор может ставить несколько диагнозов
Doctors	Schedules	Один ко многим	У доктора есть множество записей с его расписанием
Doctors	Specializations	Многие к одному	У одного доктора одна специализация, но может быть множество врачей с одной специализацией
Diagnoses	Prescribed_Medications	Один ко многим	Для одного диагноза может быть выписано множество лекарств
Diagnoses	Diagnosed_Disease	Один ко многим	По одному диагнозу может быть записано несколько заболеваний
Medications	Prescribed_Medications	Один ко многим	Лекарство может быть предписано много раз
Diseases	Diagnosed_Disease	Один ко многим	Заболевание может быть записано в медкарту много раз

Каждая таблица в базе данных имеет чётко определённые поля, которые отражают конкретные аспекты работы медцентра. Эти поля служат для структурированного хранения и управления информацией, обеспечивая эффективное функционирование различных процессов сервиса.

2.3 Архитектура web-приложения

Клиент-серверная архитектура является основой данного приложения. Web-приложение состоит из двух основных компонентов: клиентской и серверной частей. В качестве сервера будет выступать приложение, написанное с использованием платформы Node.js 20.16.0 с использованием фреймворка Express 4.21.2. Для взаимодействия с базой данных и выполнения операций над данными используется ORM Sequelize 6.37.6 [10]. В качестве базы данных выбран PostgreSQL 17.2.

Клиентская часть приложения будет разработана с использованием библиотеки React.js 18 и Redux Toolkit Query 5.0.1. Схема развертывания данного приложения

представлена на рисунке 2.3.

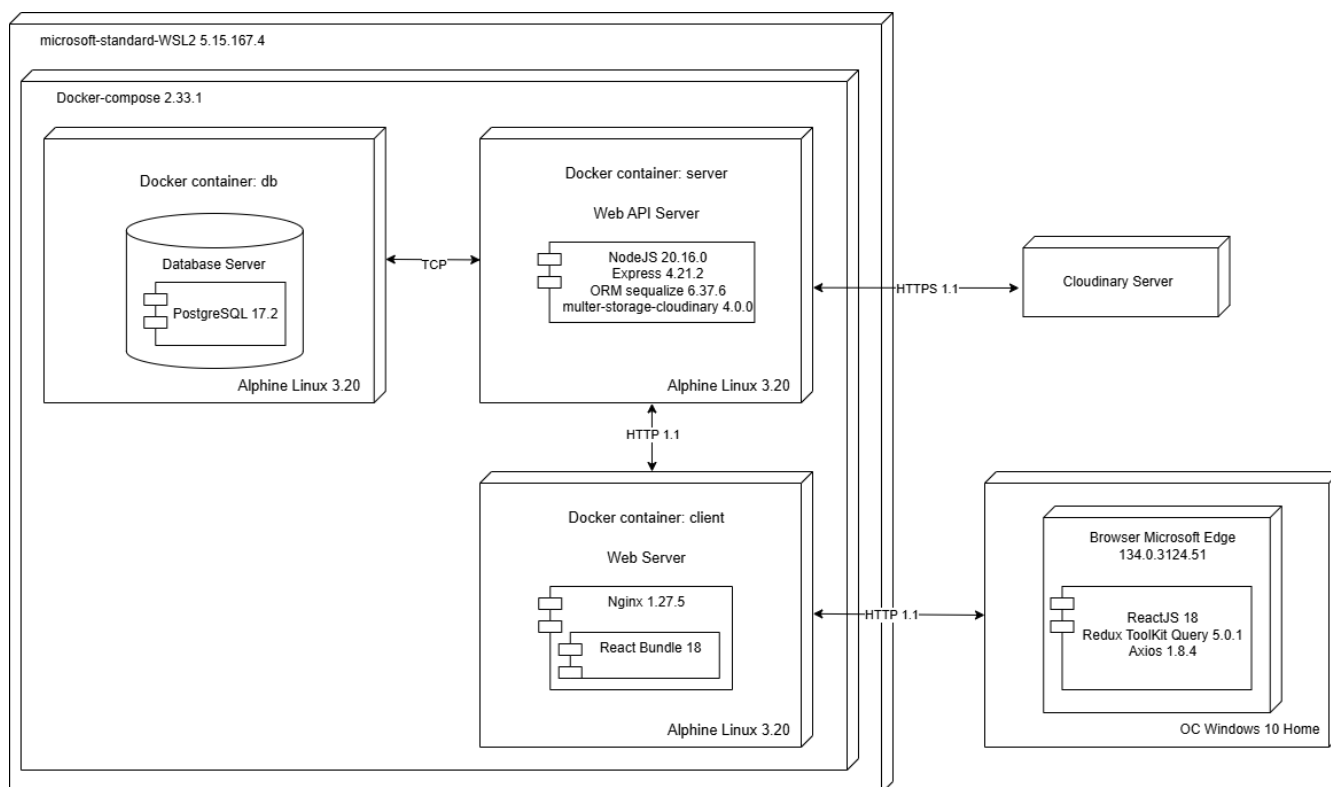


Рисунок 2.3 – Схема развёртывания

Протокол HTTP 1.1 [11] будет использоваться для передачи данных между клиентом и сервером. Протокол TCP [12] будет применяться для связи между бэкенд-сервером и сервером базы данных, обеспечивая надежную передачу данных.

Для хранения фотографий врачей будет использоваться сервис Cloudinary [13], который является облачным хранилищем.

2.4 Выводы по разделу

1. Была определена основная функциональность web-приложения медцентра, выделены роли (гость, клиент, врач, администратор) и разграничены варианты использования по ролям.

2. Была рассмотрена логическая схема базы данных web-приложения, которая включает тринадцать таблиц Diagnoses, Diagnosed_Disease, Disease Prescribed_Medications, Medical_Results, Patients, Doctors, Specializations, Appointments, Users, Token_Users, Medications, Schedules.

3. Для серверной части будет использовано приложение, разработанное на платформе Node.js 20.16.0 с использованием фреймворка Express 4.21.2. База данных будет реализована с помощью PostgreSQL 17.2. Для взаимодействия с базой данных будет применяться ORM Sequelize 6.37.6. Клиентская часть приложения будет разработана с использованием библиотеки React.js 18.

3 Реализация web-приложения

3.1 Программная платформа Node.js

Для серверной части проекта была выбрана платформа Node.js 20.16.0.

В рамках проекта Node.js 20.16.0 используется для создания REST API [14], обеспечивая удобное взаимодействие между клиентской и серверной частью. Для работы с базой данных применяется ORM Sequelize 6.37.6, который предоставляет интерфейс для выполнения запросов, валидации данных и управления моделями. Node.js 20.16.0 в сочетании с Express.js 4.21.2 предлагает инструменты для маршрутизации, обработки запросов и настройки middleware.

3.2 Система управления базами данных PostgreSQL

Для работы web-приложения используется PostgreSQL 17.4 – надежная и масштабируемая система управления базами данных. Она поддерживает сложные SQL-запросы, транзакции с соответствием стандарту ACID, а также широкий спектр типов данных, включая JSON.

3.3 Реализация базы данных

Для обеспечения корректной работы веб-приложения была разработана продуманная структура базы данных с четко определенными моделями и связями между ними. В процессе проектирования использовался ORM-инструмент Sequelize 6.37.6, который предоставляет возможность работы с базой данных через JavaScript-модели, значительно упрощая управление схемой данных и процесс миграций.

Разработка велась по методологии Code First [15], когда структура базы данных формируется на основе программных моделей. В результате было создано 13 взаимосвязанных моделей, каждая из которых представляет отдельную таблицу в базе данных с соответствующими полями и связями.

Код модели User представлен в листинге 3.1.

```
User.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
    primaryKey: true},
  Login: {type: DataTypes.STRING(50), unique: true, allowNull:
    false},
  Password: {type: DataTypes.STRING(100), allowNull: false},
  Role: {type:DataTypes.STRING(30),allowNull: false,
    validate: {isIn: [['admin', 'doctor', 'client']],}},
  Registration_Date: {type: DataTypes.DATE, allowNull: false,
    defaultValue: DataTypes.NOW}}, {
  sequelize,
  modelName: 'User',
  tableName: 'Users',
  timestamps: false,});
```

Листинг 3.1 – Код модели User

Код модели Patient представлен в листинге 3.2.

```
Patient.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4,
primaryKey: true},
  First_Name: {type: DataTypes.STRING(50), allowNull: false},
  Last_Name: {type: DataTypes.STRING(50), allowNull: false},
  Patronymic: {type: DataTypes.STRING(50), allowNull: false},
  Birthdate: {type: DataTypes.DATEONLY, allowNull: true,
  validate: { isDate: true}},
  Gender: {type:DataTypes.CHAR(1), allowNull:false, defaultValue:'M',
  validate: {isIn: [['M', 'Ж']],}},
  Phone: {type: DataTypes.STRING(20), unique: true, allowNull: false},
  User_ID: {type: DataTypes.UUID, allowNull: true,
  references: {model: 'Users', key: 'ID',}},
}, {
  sequelize,
  modelName: 'Patient',
  tableName: 'Patients',
  timestamps: false,
});
```

Листинг 3.2 – Код модели Patient

Код модели Doctor представлен в листинге 3.3.

```
Doctor.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  First_Name: {type: DataTypes.STRING(50), allowNull: false,},
  Last_Name: {type: DataTypes.STRING(50), allowNull: false,},
  Patronymic: {type: DataTypes.STRING(50), allowNull: false},
  Birthdate: {type: DataTypes.DATEONLY(50), allowNull: false},
  Phone: {type: DataTypes.STRING(20),unique: true,allowNull: false},
  Specialization_ID: {type: DataTypes.UUID, allowNull: true,
  references: {model: 'Specializations', key: 'ID',}},
}, {
  Category: {type: DataTypes.STRING(100), allowNull: false,},
  Photo: {type: DataTypes.STRING(255), allowNull: false, defaultValue:
null},
  User_ID: {type: DataTypes.UUID, allowNull: true,
  references: {model: 'Users', key: 'ID',}},
}, {
  sequelize,
  modelName: 'Doctor',
  tableName: 'Doctors',
  timestamps: false,
});
```

Листинг 3.3 – Код модели Doctor

Код модели Specialization представлен в листинге 3.4.

```
Specialization.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Name: {type: DataTypes.TEXT, unique: true, allowNull: false,},
}, {
  sequelize,
  modelName: 'Specialization',
  tableName: 'Specializations',
  timestamps: false,
});
```

Листинг 3.4 – Код модели Specialization

Код модели Appointment представлен в листинге 3.5.

```
Appointment.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Patient_ID: {type: DataTypes.UUID, allowNull: false,
  references: {model: 'Patients', key: 'ID',}},
},
  Doctor_ID: {type: DataTypes.UUID, allowNull: false,
  references: {model: 'Doctors', key: 'ID',}},
},
  Appointment_Date: {type: DataTypes.DATE, allowNull: false,},
  isComplited: {type: DataTypes.BOOLEAN, allowNull: false,},
  isDenied: {type: DataTypes.BOOLEAN, allowNull: false,},
}, {
  sequelize,
  modelName: 'Appointment',
  tableName: 'Appointments',
  timestamps: false,
});
```

Листинг 3.5 – Код модели Appointment

Код модели MedicalResult представлен в листинге 3.6.

```
MedicalResult.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Patient_ID: {type: DataTypes.UUID, allowNull: false,
  references: {model: 'Patients', key: 'ID',}},
},
  Doctor_ID: {type: DataTypes.UUID, allowNull: false,
  references: {model: 'Doctors', key: 'ID',}},
},
  Result_Description: {type: DataTypes.TEXT, allowNull: false,
},
  Date: {type: DataTypes.DATE, allowNull: false, },
```

```

}, {
  sequelize,
  modelName: 'MedicalResult',
  tableName: 'Medical_Results',
  timestamps: false,
});

```

Листинг 3.6 – Код модели MedicalResult

Код модели Diagnosis представлен в листинге 3.7.

```

Diagnosis.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Patient_ID: {type: DataTypes.UUID, allowNull: false,
    references: { model: 'Patients', key: 'ID',}},
  },
  Doctor_ID: {type: DataTypes.UUID,allowNull: false,
    references: {model: 'Doctors', key: 'ID',}},
  },
  Date: {type: DataTypes.DATE, allowNull: false,}},
}, {
  sequelize,
  modelName: 'Diagnosis',
  tableName: 'Diagnoses',
  timestamps: false,
});

```

Листинг 3.7 – Код модели Diagnosis

Код модели Disease представлен в листинге 3.8.

```

Diseases.init({
  ID: {type: DataTypes.UUID,defaultValue: DataTypes.UUIDV4(),
primaryKey: true },
  Name: { type: DataTypes.TEXT, unique: true, allowNull: false,}},
}, {
  sequelize,
  modelName: 'Disease',
  tableName: 'Diseases',
  timestamps: false,
});

```

Листинг 3.8 – Код модели Disease

Код модели DiagnosedDisease представлен в листинге 3.9.

```

DiagnosedDisease.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Diagnosis_ID: { type: DataTypes.UUID,allowNull: false,
    references: {model: 'Diagnoses', key: 'ID',}},

```

```

    },
    Disease_ID: { type: DataTypes.UUID, allowNull: false,
      references: {model: 'Diseases', key: 'ID'}, },
    }, {
    sequelize,
    modelName: 'DiagnosedDisease',
    tableName: 'DiagnosedDiseases',
    timestamps: false,
  });

```

Листинг 3.9 – Код модели Disease

Код модели Medication представлен в листинге 3.10.

```

Medication.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
    primaryKey: true},
  Pharm_Subgroup: { type: DataTypes.STRING(250), unique: false,
    allowNull: false, },
  IGN: {type: DataTypes.STRING(200), unique: false, allowNull: false,},
  Dosage_Form: {type: DataTypes.STRING(200), unique: false, allowNull:
    false, },
  Price: { type: DataTypes.DECIMAL(10, 2), allowNull: false,},
  isAvailable: { type: DataTypes.BOOLEAN, allowNull: false,},
}, {
  sequelize,
  modelName: 'Medication',
  tableName: 'Medications',
  timestamps: false,
});

```

Листинг 3.10 – Код модели Medication

Код модели PrescribedMedication представлен в листинге 3.11.

```

PrescribedMedication.init({
  ID: {type: DataTypes.UUID, defaultValue: DataTypes.UUIDV4(),
    primaryKey: true },
  Diagnosis_ID: { type: DataTypes.UUID, allowNull: false,
    references: { model: 'Diagnoses', key: 'ID', },
  },
  Medication_ID: {type: DataTypes.UUID, allowNull: false,
    references: { model: 'Medications', key: 'ID',},
  },
  Dosage: { type: DataTypes.STRING(100), allowNull: false,},
  Duration: { type: DataTypes.INTEGER, allowNull: false,},
}, {
  sequelize,
  modelName: 'PrescribedMedication',
  tableName: 'Prescribed_Medications',
  timestamps: false,
});

```

Листинг 3.11 – Код модели PrescribedMedication

Код модели Schedule представлен в листинге 3.12.

```
Schedule.init({
  ID: {type: DataTypes.UUID,defaultValue: DataTypes.UUIDV4(),
primaryKey: true},
  Doctor_ID: { type: DataTypes.UUID, allowNull: false,
    references: { model: 'Doctors', key: 'ID', },},
  Day_of_week: { type: DataTypes.STRING(20),allowNull: false,
    validate: { isIn: [['Понедельник', 'Вторник', 'Среда', 'Четверг',
'Пятница', 'Суббота', 'Воскресенье']],}},
  Start_time: { type: DataTypes.TIME, allowNull: false},
  End_time: {type: DataTypes.TIME, allowNull: false},
  Interval: { type: DataTypes.TIME, allowNull: false},}, {
  sequelize,
  modelName: 'Schedule',
  tableName: 'Schedules',
  timestamps: false,
});
```

Листинг 3.12 – Код модели Schedule

Код модели TokenUser представлен в листинге 3.13.

```
TokenUser.init({
  ID: {type: DataTypes.UUID,defaultValue: DataTypes.UUIDV4(),
primaryKey: true },
  User_ID: { type: DataTypes.UUID, allowNull: false,
    references: { model: 'Users', key: 'ID', },},
  RefreshToken: {type: DataTypes.STRING(300), allowNull: false}}, {
  sequelize,
  modelName: 'TokenUser',
  tableName: 'TokenUsers',
  timestamps: false,
});
```

Листинг 3.13 – Код модели TokenUser

Контекст базы данных в Sequelize настраивается через экземпляр Sequelize, который управляет подключением к базе данных database.js. Код для создания экземпляра Sequelize представлен в листинге 3.14.

```
const sequelize = new Sequelize(process.env.DB_NAME,
process.env.DB_USER, process.env.DB_PASSWORD, {
  host: process.env.DB_HOST,
  dialect: 'postgres',
  logging: console.log,});
```

Листинг 3.14 – Код создания экземпляра Sequelize

SQL-запросы для создания базы данных представлен в Приложении А.

3.4 Программные библиотеки

В процессе разработки серверной части веб-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.1

Таблица 3.1 – Программные библиотеки серверной части

Библиотека	Версия	Назначение
express	4.21.2	Основной фреймворк для создания web-приложений на Node.js.
bcrypt [16]	5.1.1	Используется для хеширования паролей и обеспечения безопасного хранения пользовательских данных.
cors [17]	2.8.5	Используется для включения поддержки CORS (Cross-Origin Resource Sharing) в приложениях Express.
cookie-parser [18]	1.4.7	Позволяет разбирать куки из HTTP-запросов и работать с ними в приложении.
dotenv [19]	16.4.7	Позволяет загружать переменные окружения из файла .env [20], что упрощает конфигурацию приложения.
express-validator [21]	7.2.1	Библиотека для валидации и обработки данных входящих запросов в приложении Express.
jsonwebtoken [22]	9.0.2	Используется для создания и проверки JSON Web Token (JWT) для аутентификации пользователей.
multer [23]	1.4.5-lts.2	Middleware для обработки multipart/form-data, используемого для загрузки файлов.
multer-storage-cloudinary [24]	4.0.0	Библиотека для работы с облачным хранилищем Cloudinary
pg [25]	8.14.1	Библиотека для работы с PostgreSQL из Node.js, обеспечивает взаимодействие с базой данных.
pg-hstore [26]	2.3.4	Библиотека для сериализации и десериализации объектов в формате hstore для PostgreSQL.
sequelize	6.37.6	ORM для Node.js, позволяющая взаимодействовать с различными базами данных, включая PostgreSQL.
uuid [27]	11.1.0	Генератор уникальных идентификаторов (UUID) для использования в приложениях.

В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки клиентской части

Библиотека	Версия	Назначение
ant-design/icons [28]	6.0.0	Иконки для Ant Design (UI-библиотеки)
fluentui/react [29]	8.122.14	UI-компоненты от Microsoft (Fluent Design)
reduxjs/toolkit [30]	2.6.1	Официальный инструментарий Redux. Упрощает настройку хранилища, предоставляя инструменты и утилиты.
antd [31]	5.24.6	Комплексная UI-библиотека Ant Design. Готовые профессиональные компоненты, темы, локализация
Axios [32]	1.8.4	HTTP-клиент для API запросов. Поддержка промисов, перехватчики запросов, отмена запросов
Moment [33]	2.30.1	Работа с датами и временем. Форматирование, парсинг, локализация дат
React [34]	18.3.1	Основная библиотека для создания пользовательских интерфейсов с использованием компонентов.
react-dom [35]	18.3.1	Позволяет взаимодействовать с DOM и рендерить React-компоненты в браузере.
react-router-dom [36]	7.4.1	Библиотека для маршрутизации в React-приложениях, позволяющая управлять навигацией между страницами.
react-redux [37]	9.2.0	Связывает React с Redux, позволяя использовать состояние Redux внутри компонентов.
swiper [38]	11.2.6	Современный слайдер. Поддержка touch-жестов, адаптивность, эффекты переходов

Все используемые библиотеки обеспечивают необходимую функциональность, позволяя разработать эффективное и современное web-приложение.

3.5 Описание серверной части

Серверная часть приложения, реализованная на платформе Node.js 20.16.0, включает в себя ключевые элементы, которые обеспечивают корректную работу и расширяемость приложения. В таблице 3.3 приведён список директорий проекта и назначение файлов, хранящихся в этих директориях.

Таблица 3.3 – Директории серверной части проекта

Директория	Назначение
config	Хранит контекст для подключения в базе данных, связи моделей и определение подключения к облачному хранилищу.
controllers	Содержит контроллеры, которые обрабатывают HTTP-запросы и выполняют соответствующую логику.

Продолжение таблицы 3.3

dtos	Содержит объекты передачи данных, используемые для обработки входящих и исходящих данных.
exceptions	Хранит обработчики исключений для централизованной обработки ошибок в приложении.
middlewares	Содержит промежуточные обработчики для выполнения задач, таких проверка доступности данных.
models	Определяет сущности базы данных, представляя структуру данных приложения.
photos	Содержит фотографии врачей.
routers	Определяет маршруты приложения и связывает их с соответствующими контроллерами.
services	Реализует бизнес-логику приложения и обрабатывает данные, взаимодействуя с моделями и репозиториями.
utils	Содержит дополнительные функции для работы с изображениями и генерации логина и пароля.

Таблица соответствия маршрутов контроллерам и функциям в исходном коде представлена в таблице 3.4.

Таблица 3.4 – Контроллеры и функции маршрутов

Метод	Маршрут	Контроллер	Метод контроллера
POST	auth/registration	UserController	registration
POST	auth/login	UserController	login
POST	auth/logout	UserController	logout
POST	auth/refresh	UserController	refresh
GET	auth/me	UserController	getUserData
DELETE	auth/me	UserController	deleteUser
PATCH	auth/me/username	UserController	updateUsername
GET	auth/:id	UserController	getUserById
DELETE	auth/:id	UserController	deleteUser
GET	auth/doctors	AdminController	getDoctors
GET	auth/doctors/:id	AdminController	getDoctor
POST	patients/empty	PatientController	createEmpty
PUT	patients/:id	PatientController	update
GET	patients/user/:userId	PatientController	getUserById
POST	admin/doctors	AdminController	addDoctor
PUT	admin/doctors/:id	AdminController	updateDoctor
DELETE	admin/doctors/:id	AdminController	deleteDoctor
GET	admin/doctors/specialization	AdminController	getSpecializations
POST	admin/doctors/specialization	AdminController	addSpecialization
GET	admin/medicines	AdminController	getMedicines
POST	admin/medicines	AdminController	addMedicine
PUT	admin/medicines/:id	AdminController	updateMedicineStatus

Продолжение таблицы 3.4

PUT	admin/medicines/:id/price	AdminController	updateMedicinePrice
DELETE	admin/medicines/:id	AdminController	deleteMedicine
GET	doctors/doctor/:id	DoctorController	getDoctorByUserId
GET	doctors/appointments/:id	DoctorController	getDoctorAppointments
GET	doctors/patient/:id	DoctorController	getPatient
POST	appointments/	PatientController	createAppointment
PUT	appointments/:id/cancel	PatientController	cancelAppointment
GET	appointments/patient/:id	PatientController	getPatientAppointments
GET	appointments/doctor/:id	DoctorController	getDoctorAppointments
GET	appointments/appointments	AdminController	getAppointments
GET	medical/diseases	MedicalController	getDiseases
GET	medical/medications	MedicalController	getMedications
GET	medical/result/patient/:id	MedicalController	getPatientMedicalResults
GET	medical/result/doctor/:id	MedicalController	getDoctorMedicalResults
POST	medical/results	MedicalController	createMedicalResult
POST	medical/diagnoses	MedicalController	createDiagnoses
POST	medical/prescriptions	MedicalController	prescribeMedication
POST	medical/diagnoses/patient/:id	MedicalController	getPatientDiagnoses
GET	medical/patients/:id/diagnoses-data	MedicalController	getPatientDiagnosesData

При передаче данных между клиентом и сервером используется формат JSON (JavaScript Object Notation) [39]. В контроллерах осуществляются взаимодействия с сервисами, которые инкапсулируют бизнес-логику приложения. Реализации основных сервисов можно найти в Приложении Б.

3.6 Реализация функций пользователя с ролью «Гость»

3.6.1 Аутентификация

Гость web-приложения может пройти авторизацию для получения прав на выполнение действий. Процесс авторизации реализован в методе login контроллера UserController. Реализация метода представлена в листинге 3.15.

```

async login(req, res, next) {
    try{
        const {login, password, role} = req.body;
        const userData = await userServices.login(login, password,
role);
        if (!userData.refreshToken) {

```

```

        return res.json(userData);
    }
    res.cookie('refreshToken', userData.refreshToken, {maxAge:
30 * 24 * 60 * 60 * 1000, httpOnly: true});
    return res.json(userData);
} catch (e){
    next(e);
}
}
}

```

Листинг 3.15 – Реализация метода login

Метод login принимает HTTP-запрос с данными из тела, включая логин, пароль и роль. После получения данных метод вызывает `userServices.login`, который проверяет учетные данные пользователя и возвращает информацию о пользователе. Затем устанавливается `refreshToken` в cookie и возвращаются данные пользователя в формате JSON в ответе.

3.6.2 Регистрация

Гость web-приложения может зарегистрироваться в системе для создания своего аккаунта и получения прав на выполнение действий. Процесс регистрации реализован в методе `registration` контроллера `UserController`. Реализация метода представлена в листинге 3.16.

```

async registration(req, res, next) {
    try{
        const errors = validationResult(req);
        if(!errors.isEmpty()){
            return next(ApiError.BadRequest('Ошибка валидации',
errors.array()));
        }
        const {login, password} = req.body;
        const userData = await userServices.registration(login,
password);
        res.cookie('refreshToken', userData.refreshToken, {maxAge:
30 * 24 * 60 * 60 * 1000, httpOnly: true});
        return res.json(userData);
    } catch (e){
        next(e);
    }
}
}

```

Листинг 3.16 – Реализация метода registration

Метод `registration` принимает HTTP-запрос с данными из тела, включая логин и пароль. После получения данных метод вызывает `userServices.registration`, который проверяет уникальность пользователя, хэширует пароль и создает учетную запись нового пользователя. Затем устанавливается `refreshToken` в cookie и возвращаются данные пользователя в формате JSON в ответе.

3.6.3 Просмотр информации о врачах

Гость web-приложения может видеть список врачей, существующих в базе данных. Просмотр информации о врачах реализован в методе `getDoctors` контроллера `AdminController`. Реализация метода представлена в листинге 3.17.

```
async getDoctors(req, res, next) {
  try {
    const doctors = await doctorServices.getAllDoctors();
    res.json(doctors);
  } catch (e) {
    next(e);
  }
}
```

Листинг 3.17 – Реализация метода `getDoctors`

Метод `getDoctors` принимает HTTP-запрос. Метод вызывает `doctorServices.getAllDoctors`, который возвращает список всех врачей в виде `doctorDto`, состоящий из информации о враче, его расписание и специализацию. Затем данные пользователя возвращаются в формате JSON в ответе.

3.7 Реализация функций пользователя с ролью «Клиент»

Клиент так же, как и гость, может просматривать информацию о врачах включая их расписание и специализацию. Дополнительный функционал представляет из себя: запись на приём к врачу, отмена собственной записи, редактирования собственного профиля и его удаление. Реализация этого функционала описана ниже.

3.7.1 Запись на приём к врачу

Клиент web-приложения может записаться на приём к врачу, существующему в базе данных. Запись реализована в методе `createAppointment` контроллера `PatientController`. Реализация метода представлена в листинге 3.18.

```
async createAppointment(req, res, next) {
  try {
    const { date } = req.body;
    const token = req.headers.authorization;
    const appointment = await appointmentService.createAppointment(
      date, token
    );
    res.status(201).json(appointment);
  } catch (error) {
    console.error(error);
    res.status(400).json({ message: error.message });
  }
}
```

Листинг 3.18 – Реализация метода `createAppointment`

Метод `createAppointment` принимает HTTP-запрос. В теле запроса принимается информация для создания записи к врачу, а именно идентификатор врача, пациента и время записи. Метод вызывает `appointmentServices.createAppointment`, который возвращает `appointmentDto`, состоящий из идентификаторов врача и пациента, времени записи и два булевых поля для определения отменена или же завершена запись. Затем данные возвращаются в формате JSON в ответе.

3.7.2 Отмена записи на приём к врачу

Клиент web-приложения может отменить свою запись на приём к врачу, существующему в базе данных. Отмена записи реализована в методе `cancelAppointment` контроллера `PatientController`. Реализация метода представлена в листинге 3.19.

```
async cancelAppointment(req, res, next) {
  try {
    const { id } = req.params;
    const token = req.headers.authorization;
    const appointment = await appointmentService.cancelAppointment(
      id, token
    );
    res.json(appointment);
  } catch (error) {
    console.error(error);
    res.status(400).json({ message: error.message });
  }
}
```

Листинг 3.19 – Реализация метода `cancelAppointment`

Метод `cancelAppointment` принимает HTTP-запрос. В параметрах запроса принимается идентификатор записи к врачу, которую нужно отменить. Метод вызывает `appointmentServices.cancelAppointment`, который возвращает `appointmentDto` с полем `isDenied` равным `true`. Затем данные возвращаются в формате JSON в ответе.

3.7.3 Редактирование собственного профиля

Клиент web-приложения может изменять информацию на странице своего профиля. Редактирование профиля реализовано в методе `update` контроллера `PatientController`. Реализация метода представлена в листинге 3.20.

```
async update(req, res, next) {
  try {
    const { id } = req.params;
    const patientData = req.body;
    if (!id) {
      throw ApiError.BadRequest('Не указан ID пациента');
    }
  }
}
```

```

const updatedPatient = await Patient.update(patientData, {
  where: { ID: id },
  returning: true,
  plain: true
});
if (!updatedPatient[1]) {
  throw ApiError.NotFound('Пациент не найден');
}
res.json({
  success: true,
  data: updatedPatient[1]
});
} catch (error) {
  next(error);
}
}

```

Листинг 3.20 – Реализация метода update

Метод update принимает HTTP-запрос. В параметрах запроса принимается идентификатор пациента, а в теле новые данные пациента. В случае отсутствия идентификатора или же ошибке обновления данных возвращается ошибка. Метод возвращает объект обновлённого пациента. Затем данные возвращаются в формате JSON в ответе.

3.7.4 Удаление собственного профиля

Клиент web-приложения может удалить свой профиль. Удаление профиля реализовано в методе deleteUser контроллера UserController. Реализация метода представлена в листинге 3.21.

```

async deleteUser(req, res) {
  try{
    const { id } = req.user.userDto;
    await userServices.deleteUser(id);
    res.clearCookie('refreshToken');
    return res.status(204).send();
  } catch (e){
    next(e);
  }
}

```

Листинг 3.21 – Реализация метода deleteUser

Метод deleteUser принимает HTTP-запрос. В параметрах запроса принимается идентификатор пользователя. Метод вызывает userServices.deleteUser, который удаляет из базы данных пользователя с переданным идентификатором. Потом очищаются cookie от refreshToken. Метод возвращает код успешного удаления 204.

3.8 Реализация функций пользователя с ролью «Врач»

Врач обладает преимущественно функциональностью для работы с медкартой пациента: просмотр результатов обследований пациента, запись результатов обследования в медицинскую карту, запись диагноза и назначенных лекарств в медицинскую карту. Реализация этого функционала описана ниже.

3.8.1 Просмотр результатов обследований

Врач web-приложения может перед тем как записать полученные результаты посмотреть историю результатов медицинских обследований в случае, если пациент уже посещал медцентр. Просмотр результатов реализован в двух методах `getPatientMedicalResults`, `getPatientDiagnosisData` контроллера `MedicalController`. Реализация методов представлена в листинге 3.22.

```

async getPatientMedicalResults(req, res, next) {
    try {
        const { id } = req.params;
        const results = await
medicalService.getPatientMedicalResults(id);
        res.json({
            success: true,
            data: results
        });
    } catch (e) {
        next(e);
    }
}

async getPatientDiagnosisData(req, res, next) {
    try {
        const { id } = req.params;
        const data = await medicalService.getPatientDiagnosisData(id);
        res.json({
            success: true,
            data
        });
    } catch (e) {
        next(e);
    }
}

```

Листинг 3.22 – Реализация методов `getPatientMedicalResults`, `getPatientDiagnosisData`

Методы `getPatientMedicalResults`, `getPatientDiagnosisData` принимают HTTP-запросы. В параметрах передаётся идентификатор. Методы вызывают `medicalService.getPatientMedicalResults` и `medicalService.getPatientDiagnosisData` соответственно, которые возвращают список из результатов медицинских обследований, поставленных диагнозов и назначенных лекарств. Затем данные возвращаются в формате JSON в ответе.

3.8.2 Запись результатов обследований в медицинскую карту

Врач web-приложения может записать полученные результаты в медицинскую карту пациента. Запись результатов реализована в методе `createMedicalResult` контроллера `MedicalController`. Реализация методов представлена в листинге 3.23.

```
async createMedicalResult(req, res, next) {
  try {
    const { patientId, description, diseases, medications,
appointmentId } = req.body;
    const doctor = await
doctorService.getDoctorByUserId(req.user.userDto.id);
    const result = await medicalService.createMedicalResult(
patientId, description, diseases, medications, doctor.id,
appointmentId);
    res.status(201).json({
      success: true, data: result
    });
  } catch (e) {next(e);}
}
```

Листинг 3.23 – Реализация метода `createMedicalResult`

Метода `createMedicalResult` принимает HTTP-запрос. В теле передаётся информация для создания результата медицинского обследования, а именно идентификатор пациента и записи, результат обследования, заболевания и назначенные лекарства. Метод вызывает `doctorService.getDoctorByUserId` для получения объекта врача по идентификатору и `medicalService.createMedicalResult`, который возвращает объект добавленных в базу данных результатов. Затем данные возвращаются в формате JSON в ответе.

3.8.3 Запись диагноза в медицинскую карту

Врач web-приложения может записать диагноз в медицинскую карту пациента. Запись диагноза реализована в методе `createDiagnosis` контроллера `MedicalController`. Реализация методов представлена в листинге 3.24.

```
async createDiagnosis(req, res, next) {
  try {
    const { patientId, diseaseIds } = req.body;
    const diagnosis = await medicalService.createDiagnosis(
patientId, diseaseIds
    );
    res.status(201).json({
      success: true, data: diagnosis
    });
  } catch (e) {next(e);}
}
```

Листинг 3.24 – Реализация метода `createDiagnosis`

Метод `createDiagnosis` принимает HTTP-запрос. В теле передаётся информация для добавления заболеваний в диагностированные: идентификаторы пациента и заболеваний. Метод вызывает `medicalService.createDiagnosis`, который возвращает список из заболеваний. Затем данные возвращаются в формате JSON в ответе.

3.8.4 Запись назначенных лекарств в медицинскую карту

Врач web-приложения может записать лекарства с дозировкой и длительностью приёма в медицинскую карту пациента. Запись лекарств реализована в методе `prescribeMedication` контроллера `MedicalController`. Реализация метода представлена в листинге 3.25.

```

    async prescribeMedication(req, res, next) {
      try {
        const { diagnosisId, medicationId, dosage, duration } =
req.body;
        const prescription = await
medicalService.prescribeMedication(diagnosisId, medicationId, dosage,
duration);
        res.status(201).json({
          success: true, data: prescription
        });
      } catch (e) {
        next(e);
      }
    }
  }
}

```

Листинг 3.25 – Реализация метода `prescribeMedication`

Метод `prescribeMedication` принимает HTTP-запрос. В теле передаются идентификаторы записи диагноза, лекарства, доза лекарства и продолжительность приёма. Метод вызывает `medicalService.prescribeMedication`, который возвращает список из назначенных лекарств. Затем данные возвращаются в формате JSON в ответе.

3.9 Реализация функций пользователя с ролью «Администратор»

Администратор обладает возможностью добавления врачей, изменение их информации, их удаление. Он так же может изменять расписание врачей и работать со списком лекарств (удаление, изменение). Реализация этого функционала описана ниже.

3.9.1 Добавление врача

Администратор web-приложения может добавлять новых врачей. Добавление реализовано в методе `addDoctor` контроллера `AdminController`. Реализация методов представлена в листинге 3.26.

```

async addDoctor(req, res, next) {
    try {
        const doctorData = {
            ...req.body,
            Photo: req.file?.path,
            Schedule: req.body.Schedule
        };
        JSON.parse(req.body.Schedule) : []

        const result = await doctorServices.addDoctor(doctorData);
        res.json(result);
    } catch (e) {
        next(e);
    }
}

```

Листинг 3.26 – Реализация метода addDoctor

Метод addDoctor принимает HTTP-запрос. В теле запроса передаётся информация о враче включая его расписание и файлом передаётся фотография врача. Метод вызывает doctorServices.addDoctor, который возвращают объект DoctorDto. Объект включает основную информацию врача (ФИО, день рождения и т.д.), а так же объекты с расписанием и специализацией врача. Затем данные возвращаются в формате JSON в ответе.

3.9.2 Удаление врача

Администратор web-приложения может удалять данные врачей из базы данных. Удаление реализовано в методе deleteDoctor контроллера AdminController. Реализация методов представлена в листинге 3.27.

```

async deleteDoctor(req, res, next) {
    try {
        const result = await
        doctorServices.deleteDoctor(req.params.id);
        res.json(result);
    } catch (e) {
        next(e);
    }
}

```

Листинг 3.27 – Реализация метода deleteDoctor

Метод deleteDoctor принимает HTTP-запрос. В параметрах запроса передаётся идентификатор врача, который нужно удалить. Метод вызывает doctorServices.deleteDoctor, который в случае успешного выполнения возвращает объект с ключом success в true. Затем данные возвращаются в формате JSON в ответе.

3.9.3 Изменение информации о враче и его графика работы

Администратор web-приложения может изменять данные врача в базы данных в том числе и его расписание. Изменение реализовано в методе `updateDoctor` контроллера `AdminController`. Реализация методов представлена в листинге 3.28.

```
async updateDoctor(req, res, next) {
  try {
    const doctorData = {
      ...req.body,
      Photo: req.file?.path
    };
    const doctor = await doctorServices.updateDoctor(
      req.params.id,
      doctorData,
      req.file?.path
    );
    res.json(doctor);
  } catch (e) {
    next(e);
  }
}
```

Листинг 3.28 – Реализация метода `updateDoctor`

Метод `updateDoctor` принимает HTTP-запрос. В теле запроса передаётся новая информация, в том числе и файл с фотографией врача. Метод вызывает `doctorServices.updateDoctor`, который в случае успешного выполнения возвращает объект врача с обновлёнными данными. Затем данные возвращаются в формате JSON в ответе.

3.9.4 Добавление лекарств

Администратор web-приложения может добавлять новые записи лекарств в базу данных. Добавление реализовано в методе `addMedicine` контроллера `AdminController`. Реализация методов представлена в листинге 3.29.

```
async addMedicine(req, res, next) {
  try {
    const data = req.body;
    const medicine = await medicineService.addMedicine(data);
    return res.json(medicine);
  } catch (e) {
    next(e);
  }
}
```

Листинг 3.29 – Реализация метода `addMedicine`

Метод `addMedicine` принимает HTTP-запрос. В теле запроса передаётся новая

информация для создания нового лекарства. Метод вызывает `medicineService.addMedicine`, который в случае успешного выполнения возвращает объект `MedicationDto` с новым лекарством. Затем данные возвращаются в формате JSON в ответе.

3.9.5 Удаление лекарств

Администратор web-приложения может удалять записи лекарств из базы данных. Удаление реализовано в методе `deleteDoctor` контроллера `AdminController`. Реализация методов представлена в листинге 3.30.

```
async deleteMedicine(req, res, next) {
  try {
    const { id } = req.params;
    const result = await medicineService.deleteMedicine(id);
    return res.json(result);
  } catch (e) {
    next(e);
  }
}
```

Листинг 3.30 – Реализация метода `deleteDoctor`

Метод `deleteDoctor` принимает HTTP-запрос. В параметрах запроса передаётся идентификатор лекарства, который нужно удалить. Метод вызывает `medicineService.deleteMedicine`, который в случае успешного выполнения возвращает сообщение об успешном удалении. Затем данные возвращаются в формате JSON в ответе.

3.9.6 Изменение статуса доступности лекарств

Администратор web-приложения может изменить статус доступности лекарств в базе данных. И в случае, если лекарства будут по статусу недоступны, назначить эти лекарства не получится. Изменение реализовано в методе `updateMedicineStatus` контроллера `AdminController`. Реализация методов представлена в листинге 3.31.

```
async updateMedicineStatus(req, res, next) {
  try {
    const { id } = req.params;
    const { isAvailable } = req.body;
    const medicine = await
medicineService.updateMedicineStatus(id, isAvailable);
    return res.json(medicine);
  } catch (e) {
    next(e);
  }
}
```

Листинг 3.31 – Реализация метода `updateMedicineStatus`

Метод `updateMedicineStatus` принимает HTTP-запрос. В параметрах запроса передаётся идентификатор лекарства, статус которого нужно изменить, а в теле передаётся новый статус. Метод вызывает `medicineService.updateMedicineStatus`, который в случае успешного выполнения возвращает `MedicationDto` с обновлённым лекарством. Если же лекарство не было найдено, будет вызвана ошибка. Затем данные возвращаются в формате JSON в ответе.

3.9.7 Изменение цены лекарств

Администратор web-приложения может изменить цену лекарств в базе данных. Изменение реализовано в методе `updateMedicinePrice` контроллера `AdminController`. Реализация методов представлена в листинге 3.32.

```

async updateMedicinePrice(req, res, next) {
  try {
    const { id } = req.params;
    const { price } = req.body;
    const medicine = await
medicineService.updateMedicinePrice(id, price);
    return res.json(medicine);
  } catch (e) {
    next(e);
  }
}

```

Листинг 3.32 – Реализация метода `updateMedicinePrice`

Метод `updateMedicinePrice` принимает HTTP-запрос. В параметрах запроса передаётся идентификатор лекарства, цену которого нужно изменить, а в теле передаётся новая цена. Метод вызывает `medicineService.updateMedicinePrice`, который в случае успешного выполнения возвращает `MedicationDto` с обновлённым лекарством. Если же лекарство не было найдено, будет вызвана ошибка. Затем данные возвращаются в формате JSON в ответе.

3.10 Структура клиентской части

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории `src`. Директории представлены в таблице 3.5.

Таблица 3.5 – Основные директории проекта в папке `src` и их назначение

Директория	Назначение
<code>components</code>	Хранит React-компоненты, предназначенные для создания элементов пользовательского интерфейса web-приложения.
<code>pages</code>	Содержит страницы web-приложения
<code>redux</code>	В этой папке находятся файлы, связанные с управлением состоянием приложения, включая настройки Redux

Продолжение таблицы 3.5

Директория	Назначение
services	Содержит настройки для работы с HTTP-запросами, включая конфигурацию экземпляров Axios.
styles	Хранит стили для страниц и компонентов

Таблица соответствия маршрутов и компонентов страниц для пользователя с ролью «Гость» представлена в таблице 3.6.

Таблица 3.6 – Маршруты и компоненты страниц для роли «Гость»

Компонент страницы	Маршрут	Назначение компонента
LoginForm	/auth/login	Форма для авторизации.
RegistrationForm	/auth/register	Форма для регистрации.
HomePage	/	Главная страница web-приложения.
DoctorsListPage	/doctors_list	Отображает список врачей, работающих в медцентре.
DoctorDetailsPage	/doctors_list/:id	Отображает карту врача.
CompleteProfile	/complete-profile	Страница для заполнения профиля пациента после регистрации.

Таблица соответствия маршрутов и компонентов страниц для пользователя с ролью «Клиент» представлена в таблице 3.7.

Таблица 3.7 – Маршруты и компоненты страниц для роли «Клиент»

Компонент страницы	Маршрут	Назначение компонента
HomePage	/	Главная страница.
DoctorsListPage	/doctors_list	Отображает список врачей, работающих в медцентре.
DoctorDetailsPage	/doctors_list/:id	Отображает карту врача.
UserProfile	/me	Содержит страницу с профилем пользователя.
AppointmentFormPage	/book-appointment	Страница для записи на приём к врачу
PatientAppointmentsPage	/appointments	Содержит все записи на приём к врачу пользователя
PatientMedicalResultPage	/medical-results	Содержит результаты медицинского обследования пациента

Таблица соответствия маршрутов и компонентов страниц для пользователя с ролью «Врач» представлена в таблице 3.8.

Таблица 3.8 – Маршруты и компоненты страниц для роли «Врач»

Компонент страницы	Маршрут	Назначение компонента
DoctorDashboard	/doctor	Содержит карту врача с основной информацией и расписанием.
DoctorAppointments	/doctor/appointments	Содержит отсортированные записи к врачу.
MedicalCardPage	/doctor/medical-card/:patientId	Хранит медкарту пациента с отображением истории его результатов обследований, нужен для записи в медкарту результатов.

Таблица соответствия маршрутов и компонентов страниц для пользователя с ролью «Администратор» представлена в таблице 3.9.

Таблица 3.9 – Маршруты и компоненты страниц для роли «Администратор»

Компонент страницы	Маршрут	Назначение компонента
AdminDashboard	/admin	Панель управления с возможностью добавления новых специализаций врачей.
AdminDoctors	/admin/doctors	Содержит список врачей, требуется для создания, изменения и удаления врачей.
AdminMedicines	/admin/medicines	Содержит список лекарств, требуется для создания, изменения и удаления лекарств.

Другие компоненты, которые используются на клиентской части представлены в таблице 3.10.

Таблица 3.10 – Вспомогательные компоненты

Компонента	Назначение компонента
Layout	Является главным слоем для вставки страниц для клиента.
DoctorLayout	Является главным слоем для вставки страниц для доктора.
AdminLayout	Является главным слоем для вставки страниц для администратора.

Продолжение таблицы 3.10

Компонента	Назначение компонента
ProtectedRoute	Компонента, которая служит дополнительной защитой функционала с доступом клиенту.
ProtectedDoctorRoute	Компонента, которая служит дополнительной защитой функционала с доступом доктору.
ProtectedAdminRoute	Компонента, которая служит дополнительной защитой функционала с доступом администратору.
DoctorsSlider	Компонента для отображения врачей на главной странице в виде слайдера.
AppointmentForm	Компонента для перехода на страницу записи на приём к врачу.

Маршрут, соответствующий корневому пути приложения, отображает в зависимости от роли либо компонента HomePage, либо DoctorDashboard, либо AdminDashboard, которые служат основными стартовыми страницами приложения. Если пользователь попытается перейти на несуществующий адрес, будет осуществлено перенаправление на главную страницу в зависимости от роли.

3.11 Настройка конфигурации Docker

Разрабатываемое web-приложение медицинского центра использует Docker 28.0.1 [39] для контейнеризации, обеспечивая изолированную и воспроизводимую среду как для разработки, так и для эксплуатации. Docker позволяет упаковать приложение со всеми зависимостями в единый контейнер, значительно упрощая процессы развертывания и масштабирования.

Для клиентской части реализована оптимизированная многоэтапная сборка: на этапе сборки используется образ node:20-alpine3.20 с установкой необходимых системных зависимостей, после чего копируются исходный код и зависимости npm, выполняется сборка проекта через npm run build. Финальный образ основан на облегченном nginx:alpine (версия 1.27.5), куда переносятся только собранные статические файлы, а веб-сервер настраивается через конфигурационный файл default.conf для обслуживания клиента на 80 порту. Такой подход минимизирует размер итогового контейнера, исключая инструменты сборки из финального образа, сохраняя при этом все функциональные возможности. Полная конфигурация Dockerfile для клиентской части представлена в листинге 3.37, демонстрируя эффективное сочетание современных практик контейнеризации и оптимизации фронтенд-сборки.

```
FROM node:20-alpine3.20 AS build
RUN apk add --no-cache python3 make g++

RUN addgroup -S appgroup && adduser -S appuser -G appgroup -u 1001
WORKDIR /app

COPY --chown=appuser:appgroup package*.json ./
RUN npm cache clean --force && \
```



```

    npm install --no-optional --legacy-peer-deps
RUN npm install @mui/system@latest --force
COPY --chown=appuser:appgroup . .
RUN npm run build && \
    mkdir -p /app/dist && \
    mv /app/build/* /app/dist/
FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Листинг 3.33 – Реализация Dockerfile для клиентской части

Серверная часть приложения контейнеризуется на базе образа `node:20-alpine3.20`, что обеспечивает минимальный размер контейнера. На первом этапе копируются файлы зависимостей, после чего выполняется их установка командой `npm install`. Далее копируется весь исходный код сервера и открывается порт 8080 для доступа к API. Запуск приложения осуществляется командой `node index.js`, которая инициализирует сервер. Такой подход гарантирует изолированность среды выполнения и воспроизводимость развертывания. Реализация Dockerfile для серверной части представлен в листинге 3.34.

```

FROM node:20-alpine3.20
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD ["node", "app.js"]

```

Листинг 3.34 – Реализация Dockerfile для серверной части

Docker Compose [40] используется для определения взаимосвязей контейнеров. Конфигурация включает три сервиса: `client`, `server`, `db`.

Сервис `client` собирается из Dockerfile в директории `client`, где происходит многоэтапная сборка фронтенда. Контейнеру присваивается имя `client`, и он становится доступен на порту 3000 хоста, который перенаправляется на порт 80 внутри контейнера с Nginx 1.27.5. Сервис подключается к внутренней сети `network` и ожидает готовности серверной части, что гарантирует корректный порядок запуска.

Сервис `server` создается из Dockerfile в папке `server`, где разворачивается Node.js-приложение. Контейнер получает имя `server`, использует переменные окружения из файла `.env` и открывает порт 8080 для API. Сервис подключен к сети `network` и зависит от готовности базы данных, что обеспечивает стабильное подключение к PostgreSQL.

Сервис `db` развертывается из официального образа `postgres:12-alpine` с предустановленными параметрами. Для мониторинга состояния СУБД используется

healthcheck с интервалом проверки 5 секунд. Данные сохраняются в томе postgres_data, что исключает их потерю при перезапуске контейнера. Порт 5432 пробрасывается на хост для внешнего подключения. Реализация Docker Compose конфигурации представлен в листинге 3.35.

```
services:
  client:
    build:
      context: ./client
      dockerfile: Dockerfile
    container_name: client
    ports:
      - "3000:80"
    networks:
      - network
    depends_on:
      - server

  server:
    build:
      context: ./server
      dockerfile: Dockerfile
    container_name: server
    env_file:
      - ./server/.env
    ports:
      - "8080:8080"
    networks:
      - network
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres:17-alpine
    container_name: db
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: Qwerty12345
      POSTGRES_DB: medcenter
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - network

networks:
```

```

network:

volumes:
  postgres_data:

```

Листинг 3.35 – Реализация Docker Compose конфигурации

Сеть `network` объединяет все сервисы. Том `postgres_data` гарантирует сохранность данных базы между перезапусками контейнеров.

Для управление контейнерами используется следующие команды:

- `docker-compose up -d --build` – собирает образы и запускает контейнеры в фоновом режиме;
- `docker-compose up` – запуск контейнеров;
- `docker-compose stop` – остановка контейнера.

Клиентское приложение будет доступно по адресу <http://localhost:3000>. Серверное API будет доступно по адресу <http://localhost:8080>. База данных PostgreSQL будет доступна на порту 5432.

3.12 Настройка конфигурации nginx

Для проксирования запросов к API используется Nginx 1.27.5. Конфигурация Nginx представлена в листинге 3.36.

```

server {
    listen 80;
    server_name localhost;

    location / {
        root /usr/share/nginx/html;
        index index.html;
        try_files $uri $uri/ /index.html;

        add_header Cache-Control "no-cache, no-store, must-revalidate";
        add_header Pragma "no-cache";
        add_header Expires 0;
    }
    location /api/ {
        proxy_pass http://server:5000;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
}

```

```

        proxy_buffering off;}
location = /api {
    if ($request_method = 'OPTIONS') {
        add_header 'Access-Control-Allow-Origin'
'http://localhost:3000';
        add_header 'Access-Control-Allow-Methods' 'GET, POST,
OPTIONS, PUT, DELETE';
        add_header 'Access-Control-Allow-Headers' 'DNT,User-
Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-
Type,Range,Authorization';
        add_header 'Access-Control-Max-Age' 1728000;
        add_header 'Content-Type' 'text/plain; charset=utf-8';
        add_header 'Content-Length' 0;
        return 204;}}}

```

Листинг 3.36 – Реализация конфигурации Nginx

Использование Nginx 1.27.5 в проекте обеспечивает эффективную раздачу статического контента и проксирование запросов к server-сервису.

3.13 Выводы по разделу

1. В web-приложении используется платформа Node.js 20.16.0 с применением фреймворка Express.js 4.21.2.
2. Для хранения данных использовалась СУБД PostgreSQL 17.2. Для упрощения взаимодействия с базой данных применялась ORM Sequelize 6.37.6.
3. Рассмотрена структура серверной части проекта, которое использует современные программные библиотеки. Общее количество маршрутов сервера вышло 41, которые принимают 5 методов (GET, POST, PUT, PATCH, DELETE). Общее количество контроллеров вышло 8.
4. Реализованы все функции для всех ролей web-приложения: гостя, клиента, врача и администратора. Количество функций web-приложения составило 19.
5. Рассмотрена структура клиентской части проекта, которое использует современные программные библиотеки. Общее количество маршрутов клиентской части вышло 19. Общее количество компонентов необходимых для отображения страниц для пользователей составило 27.
6. Реализована контейнеризация с использованием Docker 28.0.1. Оркестрация контейнеров выполнена через Docker Compose 2.35.1, а также был настроен Nginx 1.27.5.

4 Тестирование web-приложения

4.1 Функциональное тестирование

Для проверки корректности работы всех функций разработанного web-приложения было проведено ручное тестирование, описание и итоги которого представлены в таблице 4.1.

Таблица 4.1 – Описание тестирования функций web-приложения

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
1	Аутентификация	Под ролью «Гость» перейти на страницу с формой для входа (/auth/login) нажав на кнопку «Войти» на шапке страницы. После открытия формы ввести логин и пароль. Нажать на кнопку «Войти».	Пользователю отобразится главная страница (/) с его логином с иконкой профиля на месте скрытых кнопок «Войти» и «Регистрация».
2	Регистрация	Под ролью «Гость» перейти на страницу с формой для регистрации (/auth/register) нажав на кнопку «Регистрация» на шапке страницы. После открытия формы ввести логин и пароль. Нажать на кнопку «Зарегистрироваться».	Пользователю отобразится страница с формой для заполнения профиля основной информацией пациента (/complete-profile)
3	Просмотр информации о врачах	Под ролью «Гость» или «Клиент» перейти на страницу с врачами(/doctors_list), нажав в навигационной панели на ссылку «Врачи», либо на главной странице нажать на красную кнопку «Наши врачи».	Пользователю отобразится страница с информацией о всех врачах.
4	Запись на приём к врачу	Под ролью «Клиент» внизу стартовой страницы нажать на кнопку «Записаться», чтобы перейти на страницу записи (/book-appointment). На странице выбрать специализацию из доступных. Выбрать врача из этого направления. В отображенной таблице талонов нажать на окошко для выбора (стало	Пользователю отобразится страница с таблицей талонов. Выбранное окно на прошлом шаге станет недоступным, а в записях пациента (/appointments) отобразится новая запись.

Продолжение таблицы 4.1

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
		тёмно-зелёным), нажать на кнопку «Записаться» внизу страницы. В появившемся модальном окне нажать на кнопку «Подтвердить».	
5	Отмена записи на приём к врачу	Под ролью «Клиент» перейти на страницу своих записей, нажав на иконку профиля пользователя, (/appointments). При условии наличия предстоящих записей на странице, нажать на кнопку «Отменить» под нужной записью.	На странице под вкладкой «Предстоящие» запись пропадёт. Перейдя кликом на вкладку «Отменённые», нужная запись отобразится с красным контуром.
6	Редактирование собственного профиля	Под ролью «Клиент» навести на иконку профиля рядом с логином пользователя. Нажать на ссылку «Мой профиль» на появившемся окне. На странице профиля (/me) изменить данные в любом поле. Нажать на кнопку «Сохранить изменения»	Пользователя перекинет на стартовую страницу (/) и вверху страницы отобразится уведомление об успешном изменении профиля.
7	Удаление собственного профиля	Под ролью «Клиент» навести на иконку профиля рядом с логином пользователя. Нажать на ссылку «Мой профиль» на появившемся окне. На странице профиля (/me) нажать на кнопку «Удалить аккаунт». На всплывшем модельном окне нажать на кнопку «Удалить».	Пользователя перекинет на стартовую страницу (/) и вверху страницы отобразится уведомление об успешном удалении профиля.
8	Просмотр результатов обследований	Под ролью «Врач» нажать на ссылку «Мои приёмы» на боковой панели, чтобы перейти на страницу расписания приёмом к врачу (/doctor/appointments). При условии, что к врачу существуют записи на приём, нажать на запись пользователя, чтобы перейти на страницу с	Пользователю отобразится страница с данными пациента и его историей результатов обследований.

Продолжение таблицы 4.1

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
		медкартой пользователя (/doctor/medical-card/:patientId).	
9	Запись результатов обследований в медицинскую карту	Под ролью «Врач» нажать на ссылку «Мои приёмы» на боковой панели, чтобы перейти на страницу расписания приёмов к врачу (/doctor/appointments). При условии, что к врачу существуют записи на приём, нажать на запись пользователя, чтобы перейти на страницу с медкартой пользователя (/doctor/medical-card/:patientId). Внизу страницы заполнить поля с результатом медицинского обследования (обязательно), заболевания и лекарства (опционально). Нажать на кнопку «Сохранить запись»	Пользователя перенесёт на страницу с расписанием записей приёмов (/doctor/appointments). Запись, в которой сохранили данные, пропадёт из общего расписания.
10	Запись диагноза в медицинскую карту	Под ролью «Врач» нажать на ссылку «Мои приёмы» на боковой панели, чтобы перейти на страницу расписания приёмов к врачу (/doctor/appointments). При условии, что к врачу существуют записи на приём, нажать на запись пользователя, чтобы перейти на страницу с медкартой пользователя (/doctor/medical-card/:patientId). Внизу страницы заполнить поля с результатом медицинского обследования, выбрать одно или более заболеваний. Нажать на кнопку «Сохранить запись»	Пользователя перенесёт на страницу с расписанием записей приёмов (/doctor/appointments). Запись, в которой сохранили данные, пропадёт из общего расписания
11	Запись назначенных лекарств в	Под ролью «Врач» нажать на ссылку «Мои приёмы» на боковой панели, чтобы перейти	Пользователя перенесёт на страницу с расписанием записей

Продолжение таблицы 4.1

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
	медицинскую карту	на страницу расписания приёмом к врачу (/doctor/appointments). При условии, что к врачу существуют записи на приём, нажать на запись пользователя, чтобы перейти на страницу с медкартой пользователя (/doctor/medical-card/:patientId). Внизу страницы заполнить поля с результатом медицинского обследования, выбрать одно или более лекарств. В ниже отобразившихся полях заполнить данные о дозировке и продолжительности приёма лекарств. Нажать на кнопку «Сохранить запись»	приёмов (/doctor/appointments). Запись, в которой сохранили данные, пропадёт из общего расписания
12	Добавление врача	Под ролью «Администратор» нажать на ссылку на боковой панели «Врачи», чтобы перейти на страницу для редактирования списка врачей (/admin/doctors). На странице над таблицей нажать на кнопку «Добавить врача». В появившемся модальном окне заполнить поля, добавить фотографию, выбрать с помощью чекбоксов дни работы врача, а в появившихся под ними меню выбрать параметры расписания для каждого дня отдельно. Нажать на кнопку «ОК».	Пользователю отобразится таблица врачей. Новый врач появится в таблице.
13	Удаление врача	Под ролью «Администратор» нажать на ссылку на боковой панели «Врачи», чтобы перейти на страницу для редактирования списка врачей (/admin/doctors). В столбце	Пользователю отобразится таблица врачей. Вверху страницы отобразится уведомление об успешном удалении врача. Врач перестанет

Продолжение таблицы 4.1

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
		«Действия» таблицы врачей напротив нужного врача нажать на корзину. В появившемся модальном окне нажать на «ОК».	отображаться в таблице.
15	Изменение информации о враче и его графика работы	Под ролью «Администратор» нажать на ссылку на боковой панели «Врачи», чтобы перейти на страницу для редактирования списка врачей (/admin/doctors). В столбце «Действия» таблицы врачей напротив нужного врача нажать на «ручку». В появившемся модальном окне изменить нужные данные и нажать на «ОК».	Пользователю отобразится таблица врачей. Изменения подгрузятся в таблице.
16	Добавление лекарств	Под ролью «Администратор» нажать на ссылку на боковой панели «Лекарства», чтобы перейти на страницу для редактирования списка лекарств (/admin/medicines). На странице над таблицей нажать на кнопку «Добавить лекарство». В появившемся модальном окне заполнить поля, выбрать доступность лекарства. Нажать на кнопку «ОК».	Пользователю отобразится таблица лекарств. Вверху страницы отобразится уведомление об успешном добавлении лекарства. Новое лекарство появится в таблице.
17	Удаление лекарства	Под ролью «Администратор» нажать на ссылку на боковой панели «Лекарства», чтобы перейти на страницу для редактирования списка лекарств (/admin/medicines). В столбце «Действия» таблицы лекарства напротив нужного лекарства нажать на корзину. В появившемся модальном окне нажать на «ОК».	Пользователю отобразится таблица лекарств. Вверху страницы отобразится уведомление об успешном удалении лекарства. Лекарство перестанет отображаться в таблице.
18	Изменение	Под ролью «Администратор»	Пользователю

Продолжение таблицы 4.1

№	Функция web-приложения	Описание тестирования	Ожидаемый результат
	статуса доступности лекарства	нажать на ссылку на боковой панели «Лекарства», чтобы перейти на страницу для редактирования списка лекарств (/admin/ medicines). В столбце «Статус» таблицы лекарств напротив нужного лекарства нажать на переключатель. В появившемся модальном окне нажать на «Подтвердить».	отобразится таблица лекарств. Вверху страницы отобразится уведомление о доступности/недоступности лекарства. В столбце «Статус» переключатель изменится на «недоступен» (серого цвета), если до этого лекарство было доступно. Если наоборот — «доступен» (голубого цвета).
19	Изменение цены лекарства	Под ролью «Администратор» нажать на ссылку на боковой панели «Лекарства», чтобы перейти на страницу для редактирования списка лекарств (/admin/ medicines). В столбце «Цена» таблицы лекарств напротив нужного лекарства нажать на цену. В появившемся модальном окне выбрать другую цену нажать на «ОК».	Пользователю отобразится таблица лекарств. Вверху страницы отобразится уведомление о успешном изменении цены лекарства. В столбце «Цена» отобразятся изменения.

Таким образом, были протестированы все ключевые функции web-приложения.

4.2 Выводы по разделу

Проведено ручное тестирование всех ключевых функций web-приложения. Корректность работы системы подтверждена соответствием фактических результатов тестирования ожидаемым. Количество тестов составило 19, покрытие тестами 100%.

5 Руководство пользователя

5.1 Руководство для роли «Гость»

При открытии web-приложения гость автоматически попадает на стартовую страницу, где может просмотреть информацию о медцентре и врачах. Предоставляется возможность перейти на страницу врачей. Гость так же может войти в свой аккаунт, если он есть, или зарегистрироваться. Интерфейс главной страницы представлен на рисунке 5.1.

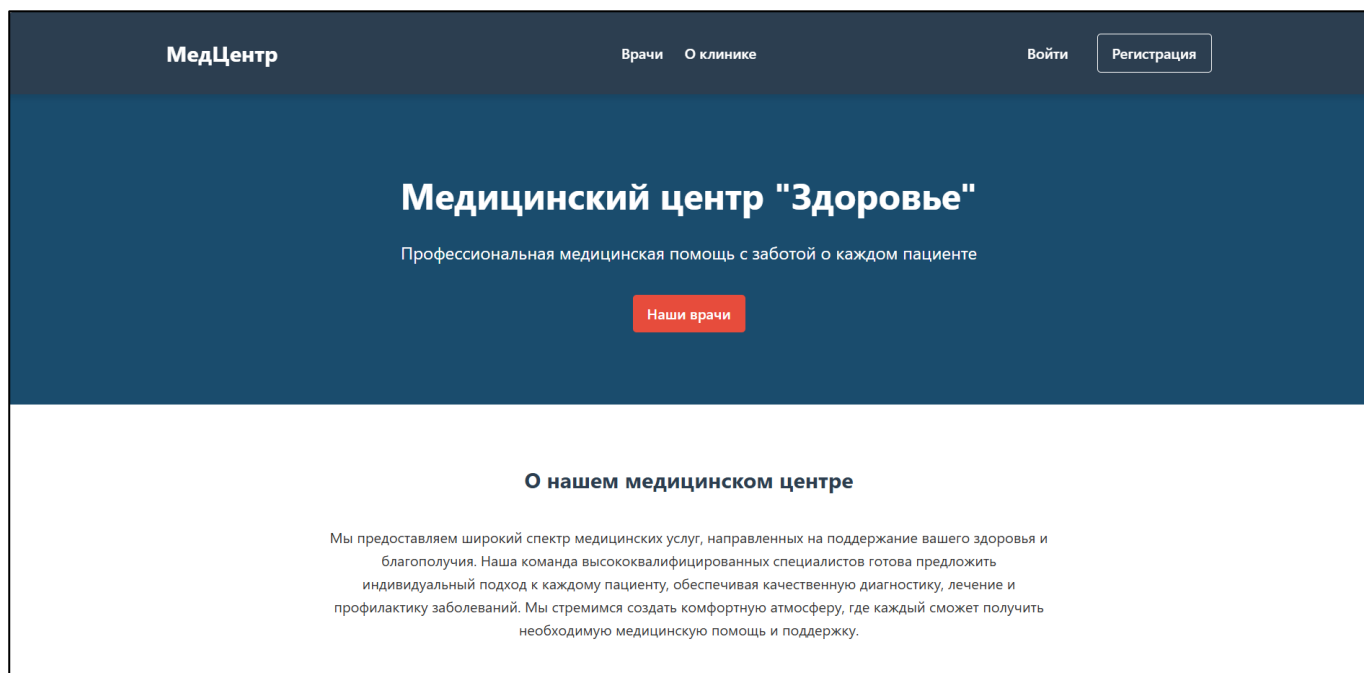
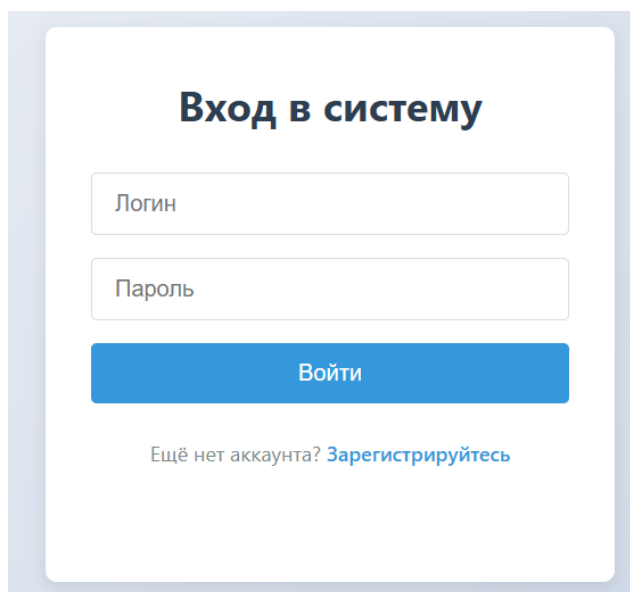


Рисунок 5.1 – Главная страница

Ссылки для перехода между страницами располагаются на главной странице и в шапке. Главная страница информативная, на ней так же располагается слайдер со списком врачей.

5.1.1 Аутентификация

Аутентификация реализуется с помощью формы, на которую можно перейти, нажав на кнопку «Войти» в шапке страницы. Форма входа представлена на рисунке 5.2. Перейдя на форму пользователю будет предложено ввести логин и пароль.



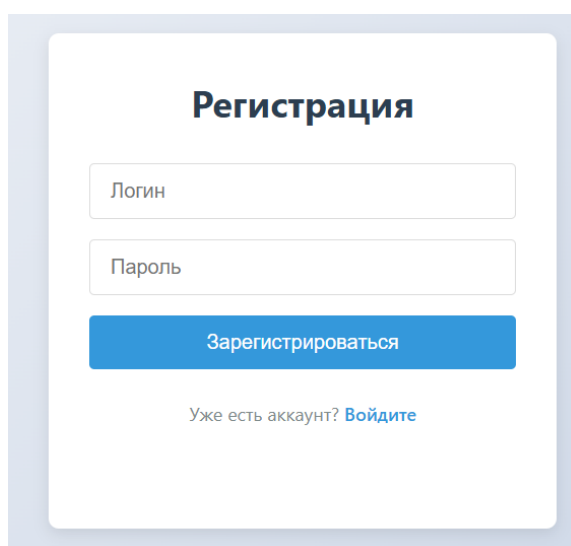
The image shows a login form titled "Вход в систему" (Login to the system). It contains two input fields: "Логин" (Login) and "Пароль" (Password). Below these fields is a blue button labeled "Войти" (Login). At the bottom, there is a link that says "Ещё нет аккаунта? [Зарегистрируйтесь](#)" (Don't have an account? [Register](#)).

Рисунок 5.2 – Форма аутентификации

После ввода данных пользователю требуется нажать на кнопку «Войти». После авторизации гостю выдаётся роль и в зависимости от роли пользователя перенаправляет на разные страницы. В случае если гость — клиент, то его перенаправляет на главную страницу, представленную на рисунке 5.1. Если гость — врач или администратор, то его перенаправляет соответственно на рабочую панель врача или админ-панель.

5.1.2 Регистрация

С формы аутентификации можно по ссылке перейти на форму регистрации. На этой форме пользователю так же предлагается придумать логин и пароль. После ввода данных гость должен нажать на кнопку «Зарегистрироваться». После этого гость приобретает роль клиента. Форма регистрации представлена на рисунке 5.3.



The image shows a registration form titled "Регистрация" (Registration). It contains two input fields: "Логин" (Login) and "Пароль" (Password). Below these fields is a blue button labeled "Зарегистрироваться" (Register). At the bottom, there is a link that says "Уже есть аккаунт? [Войдите](#)" (Already have an account? [Login](#)).

Рисунок 5.3 – Форма регистрации

После регистрации пользователю будет предложено заполнить форму для создания его медкарты. Форма для заполнения представлена на рисунке 5.4.

Завершите регистрацию

* Имя

* Фамилия

* Отчество

* Дата рождения

2007-05-26

* Пол

Мужской

* Телефон

Сохранить профиль

Рисунок 5.4 – Форма данных пациента

После регистрации и заполнения формы пользователь считается зарегистрированным и авторизованным. Позже ему становится доступным функционал клиента.

5.1.3 Просмотр информации о врачах. Фильтрация

Пользователь с ролями гость или клиент можешь видеть список врачей, работающих в медцентре. Перейти на страницу врачей можно через ссылку в шапки или через кнопку на главной странице «Наши врачи». Страница врачей представлена на рисунке 5.5.

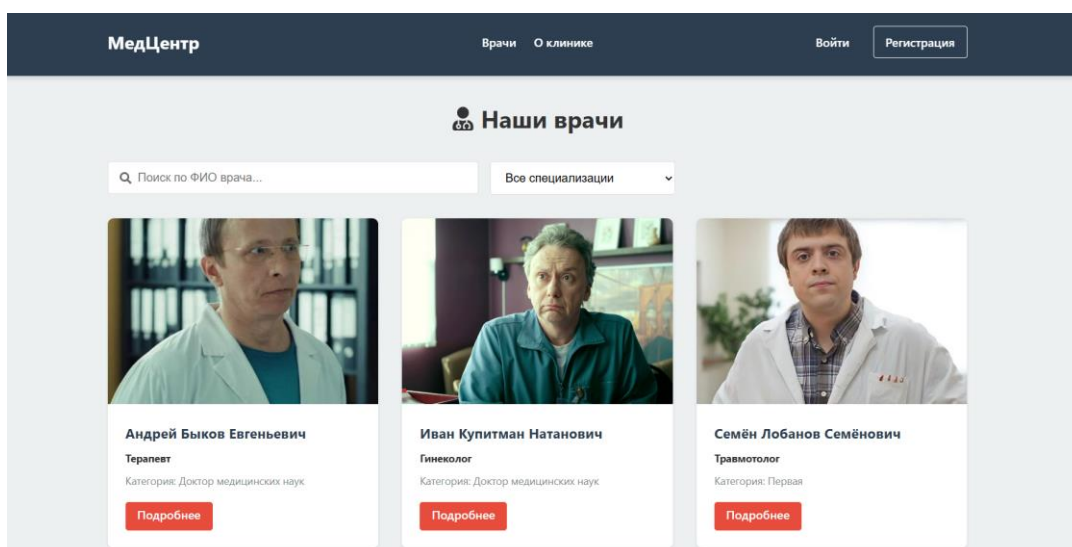


Рисунок 5.5 – Страница со списком врачей

На странице доступна фильтрация по направлениям и ФИО врачей. Так же если пользователь выберет специализацию врача и решит ввести в поисковую строку ФИО искомого врача, то поиск будет производиться по направлению. Нажав на кнопку «Подробнее» пользователя перейдёт на страницу с картой врача. Страница с картой врача представлена на рисунке 5.6.

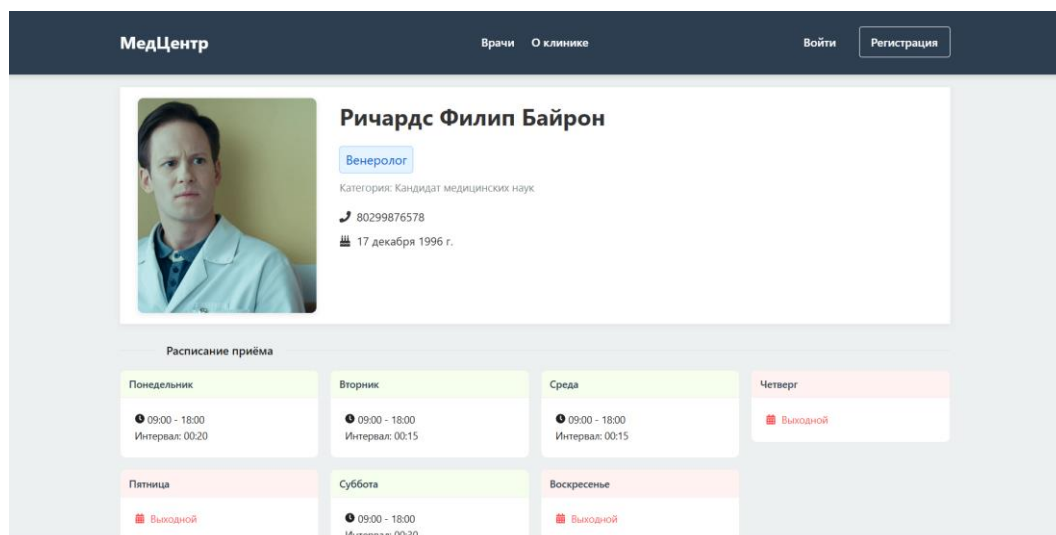


Рисунок 5.6 – Карточка врача

На карте врача располагается информация о нём и его расписание. Внизу страницы есть кнопка «Записаться на приём», которая перенесёт пользователя на страницу записи определённого врача.

5.2 Руководство для роли «Клиент»

После аутентификации и авторизации клиент будет перенаправлен на главную страницу. Теперь вместо кнопок «Войти» и «Регистрация» отображается логин пользователя и иконка профиля. Наведя на неё откроется выбор перехода между страницами профиля пользователя, его записями и результатами медицинских обследований. Профиль пользователя представлен на рисунке 5.7

Рисунок 5.7 – Профиль пользователя

У пользователя есть такие функции как изменение данных в профиле, изменение логина, выход из аккаунта, удаление аккаунта.

5.2.1 Запись на приём к врачу

Пользователь, перейдя на страницу записи, должен выбрать нужное ему направление и наиболее понравившегося врача. После этого на странице отобразится таблица из доступных талонов. Страница записи представлена на рисунке 5.8.

Время	пн 26	вт 27	ср 28	чт 29	пт 30	сб 31	вс 1
09:00			09:00	09:00	09:00	09:00	09:00
09:30			09:30	09:30	09:30	09:30	09:30
10:00			10:00	10:00	10:00	10:00	10:00

Рисунок 5.8 – Страница записи на приём к врачу

Чтобы выбрать талон и записаться, пользователь должен нажать на прямоугольник с подходящим ему временем, он станет тёмно-зелёным. Внизу страницы находится кнопка «Записаться». После того, как пользователь на неё нажмёт, выбранный талон пропадёт из общего списка. Талоны отсортированы по неделе, есть возможность выбирать неделю на стрелки.

После того, как врач запишет результаты обследований в медкарту пациента, у клиента на странице с его историей записей появится результаты обследования. Пример представлен на рисунке 5.9

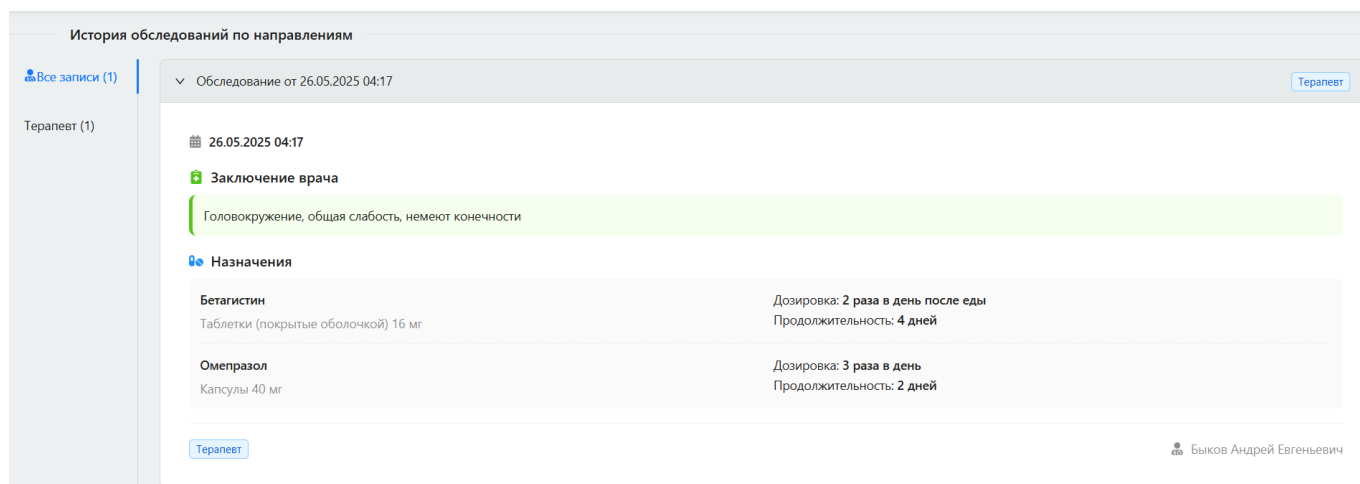


Рисунок 5.9 – Страница результатов обследований

На странице осуществляется сортировка результатов обследований по специальностям.

5.2.2 Отмена записи на приём к врачу

После того, как пользователь записался на определённое время, пользователь может посмотреть свои записи, кликнув на иконку профиля. Страница записей представлена на рисунке 5.10.

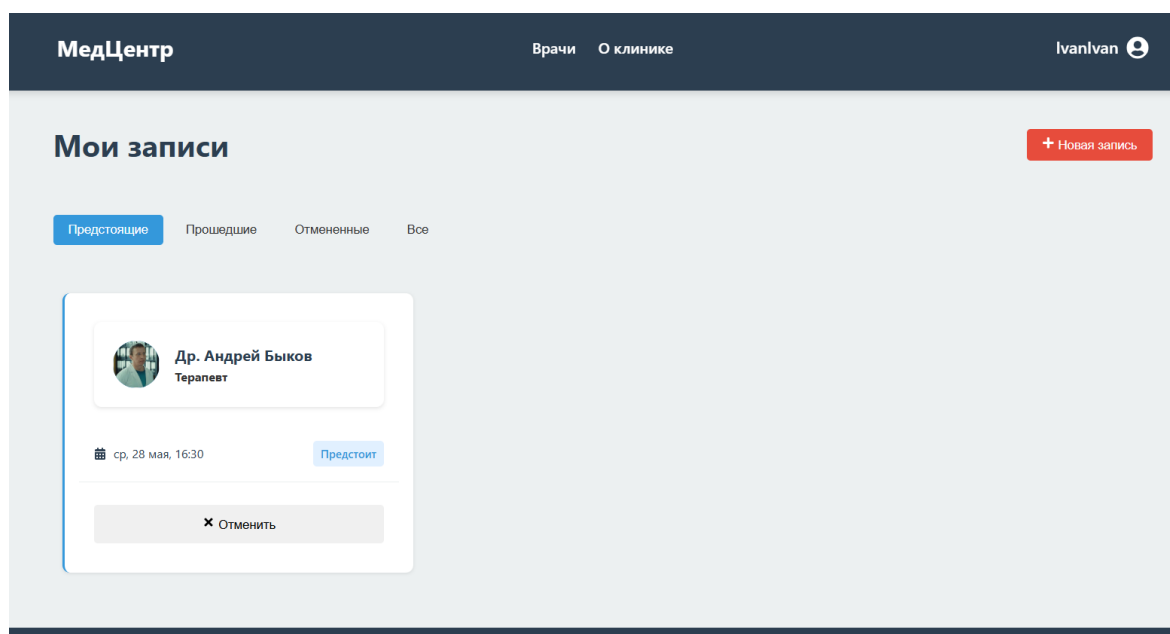


Рисунок 5.10 – Страница записей пользователя

На странице записи сортируются по вкладкам «Предстоящие», «Прошедшие»,

«Отменённые». Для того, чтобы отменить запись на приём клиент должен нажать на кнопку «Отменить» в карточке записи. После этого данная запись будет отображаться во вкладке «Отменённые» с красной окантовкой. Пример представлен на рисунке 5.11

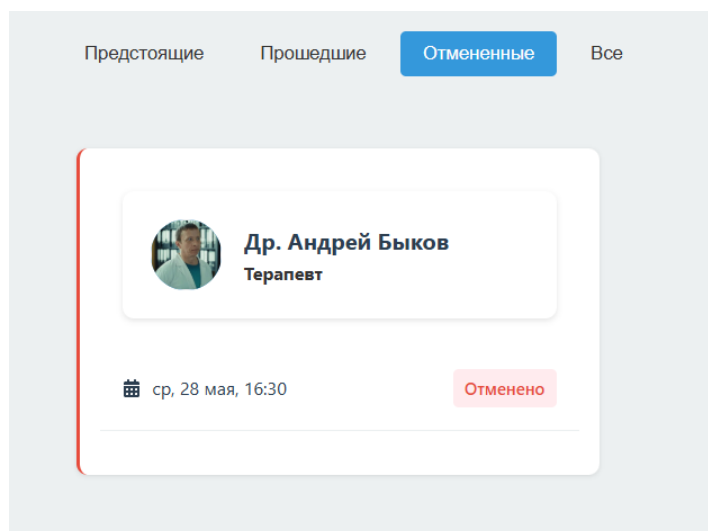


Рисунок 5.11 – Вкладка «Отменённые»

На странице так же присутствует возможно перехода обратно на страницу записи с помощью кнопки «Новая запись».

5.2.3 Редактирование собственного профиля

На странице профиля, клиент может изменять свои данные по необходимости. Компонента данных пациента представлена на рисунке 5.12.

Персональные данные

* Имя

* Фамилия

* Отчество

Иван

Пономаренко

Викторович

* Дата рождения

* Пол

2003-05-01

Мужской

* Телефон

80293749201

Сохранить изменения

Рисунок 5.12 – Компонента с данными пациента

Соответственно на кнопку «Сохранить изменения», перед эти изменив поля,

можно изменять данные.

5.2.4 Удаление собственного профиля

Удаление пользователя происходит с помощью кнопки «Удалить аккаунт». После того, как клиент нажмёт на кнопку, ему покажется модальное окно для подтверждения. Модельное окно представлено на рисунке 5.13

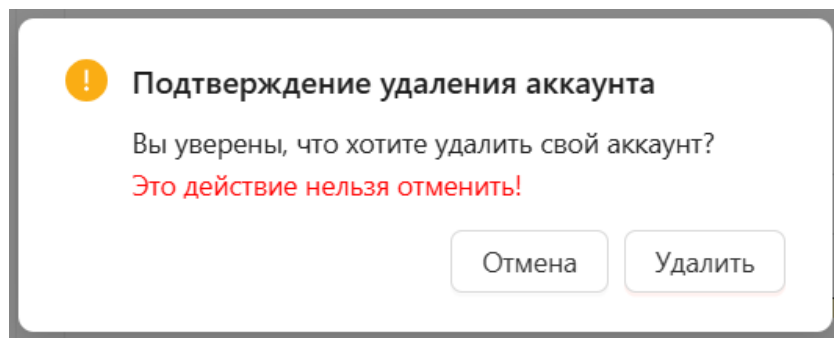


Рисунок 5.13 – Модальное окно подтверждения

После подтверждения пользователя перенаправляет на главную страницу и вверху экрана появляется уведомление об успешном удалении аккаунта.

5.3 Руководство для роли «Врач»

Если пользователь после аутентификации и авторизации определяется с ролью «Врач», то его перенаправляет на страницу панели врача. На главной странице отображается информация о вошедшем враче и его расписании. Карта врача представлена на рисунке 5.14

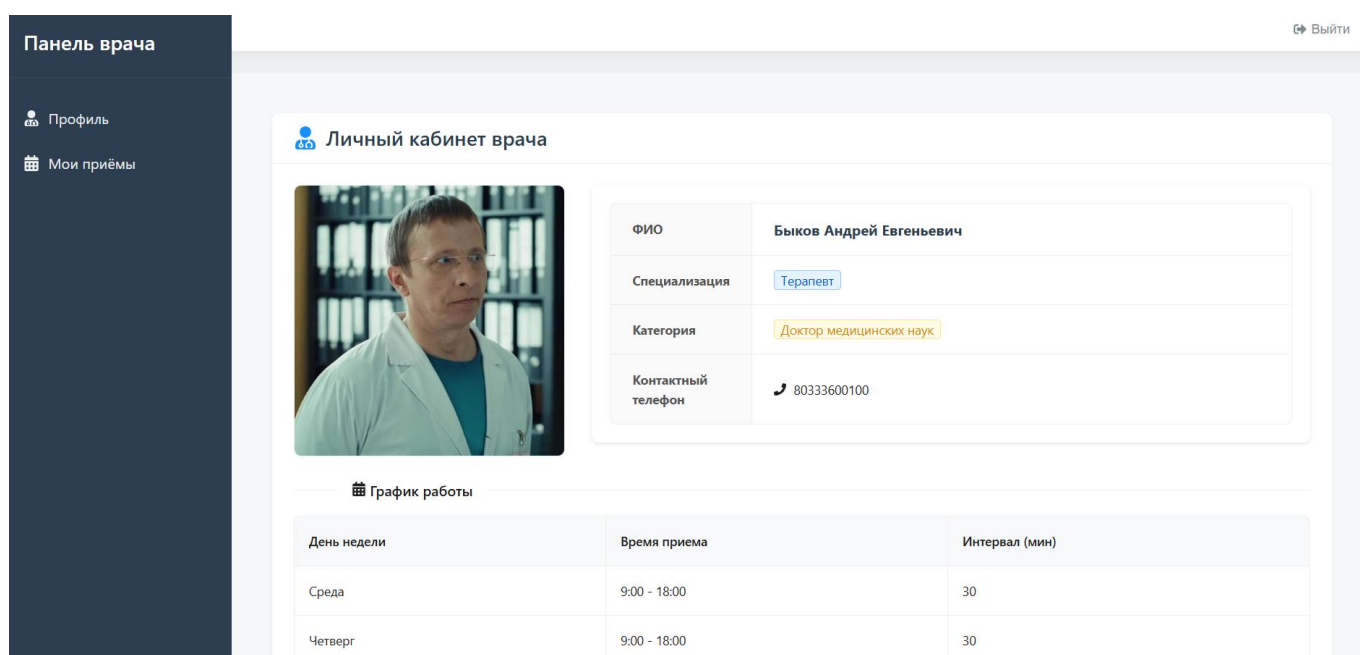


Рисунок 5.14 – Главная страница в панели врача

На боковой панели располагаются ссылки на главный экран и расписание

приёмов врача

5.3.1 Запись результатов медицинского обследования, диагностированных заболеваний, назначенных лекарств

После перехода на страницу «Мои приёмы» пользователь может ознакомиться со своим расписанием и загруженностью по датам. Страница «Мои приёмы» представлена на рисунке 5.15.

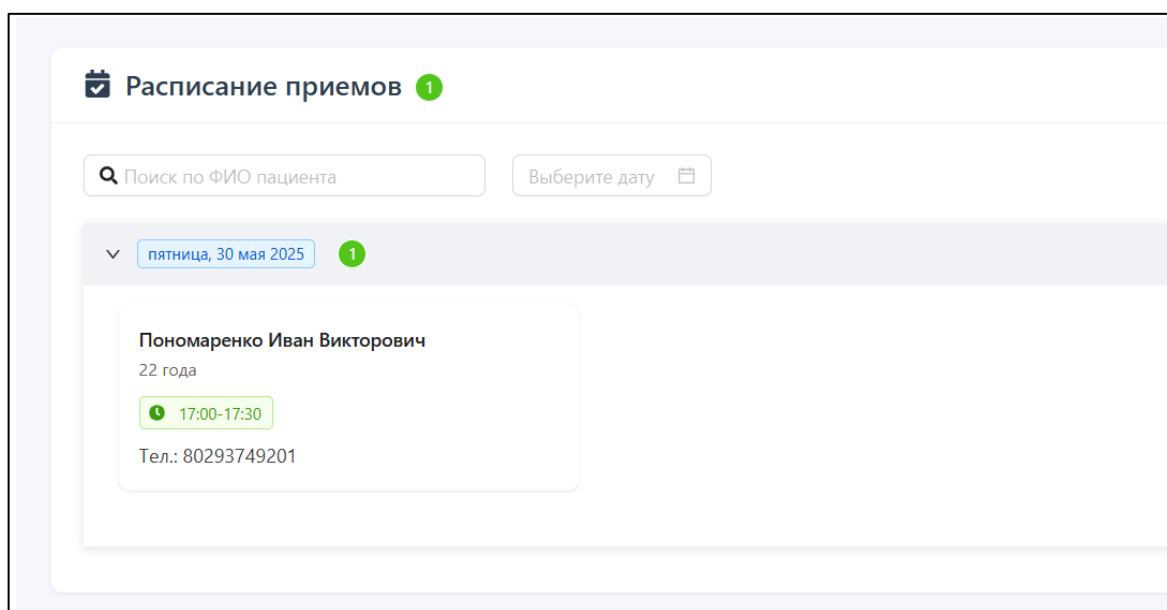


Рисунок 5.15 – Страница «Мои приёмы»

На странице отображаются все предстоящие записи к врачу. Врач может увидеть, кто к нему записался, во сколько, а также имеет возможно отсортировать свои записи по ФИО пациента, либо зайти записи в определённый день, выбранный в календаре.

Для того, чтобы оставить свою запись в медкарте пациента врач должен нажать на запись. Пользователя перенаправит на страницу с медкартой пациента. Страница с медкартой представлена на рисунке 5.16.

Медицинская карта пациента

Назад к расписанию

ФИО	Пономаренко Иван Викторович
Дата рождения	01.05.2003
Возраст	22 лет
Телефон	80293749201

История обследований по направлениям

Нет данных о медицинских обследованиях

Рисунок 5.16 – Страница с медкартой пациента

Тут будут отображаться записи других врачей и основная информация о пациента. Форма для заполнения результата находится ниже на странице и представлена на рисунке 5.17

Новое медицинское обследование

* Заключение врача

Опишите результаты осмотра, жалобы пациента, проведенные процедуры и рекомендации

Диагнозы

Выберите диагнозы

Назначения

Выберите лекарственные препараты

Сохранить запись

Рисунок 5.17 – Форма для заполнения результата обследования

В обязательном порядке должно быть заполнено поле с заключением врача. Оно представляет из себя краткую сводку того, что врач говорит на приёме пациенту о его состоянии, требуемых мероприятиях для поддержания здоровья.

Поля «Диагнозы» и «Назначения» заполняются опционально. В диагнозах помещаются заболевания с классификатором, а лекарства с фармакологической подгруппой согласно реестру Минздрава. Примеры заполнения представлены на рисунке 5.18.

Диагнозы

A88.1 Эпидемическое головокружение ×

A08 Вирусные и другие уточненные кишечные инфекции ×

Назначения

Бетагистин (Таблетки (покрытые оболочкой) 16 мг) ×

Омепразол (Капсулы 40 мг) ×

Укажите дозировки и продолжительность приема

Бетагистин

Дозировка

2 раза в день после еды

Например: 1 таблетка 2 раза в день

Продолжительность (дней)

4

Омепразол

Дозировка

3 раза в день

Например: 1 таблетка 2 раза в день

Продолжительность (дней)

2

Сохранить запись

Рисунок 5.18 – Пример заполнения полей заболеваний и лекарств

После врач должен сохранить свои записи с помощью кнопки «Сохранить запись». Данная запись на приём пропадёт из расписания врача и будет помечена как «Выполненная»

5.3.2 Просмотр результатов обследований пациента

Если пациент в медцентре не впервые, то его записи будут сохраняться в его медкарте. Позже эти записи смогут просматривать другие врачи для того, чтобы выставить более точные диагнозы или назначения. Как отображаются истории записей представлено на рисунке 5.19.

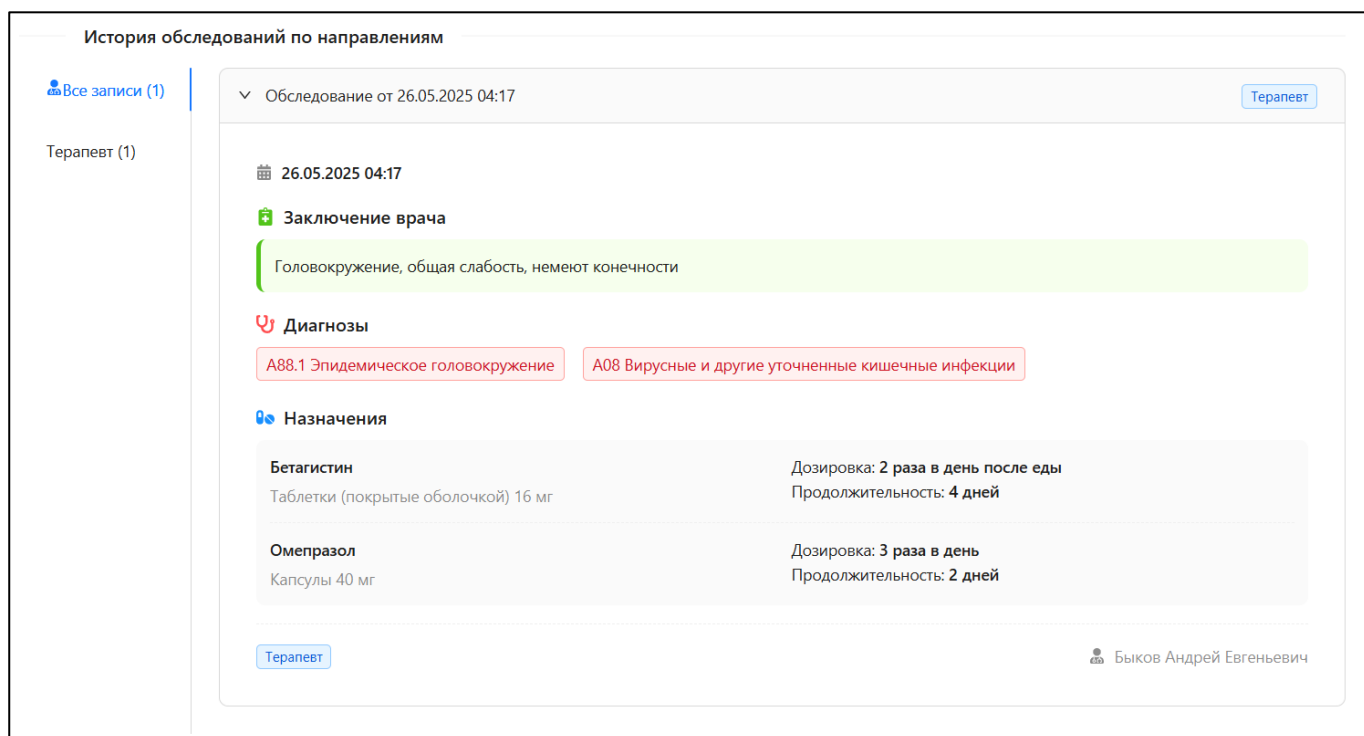


Рисунок 5.19 – Истории результатов обследований

Как видно по рисунку для каждой записи определяется врач, который создал эту запись, а также направление. Записи одной категории будут сортироваться в виде вкладок на панели слева.

5.4 Руководство для роли «Администратор»

Если пользователь после аутентификации и авторизации определяется с ролью «Администратор», то его перенаправляет на страницу админ-панели. На главной странице отображается вычислительная информация о медцентре: кол-во врачей, кол-во лекарств, кол-во записей на сегодняшний день, кол-во различных специализаций. Главная страница админ-панели представлена на рисунке 5.20

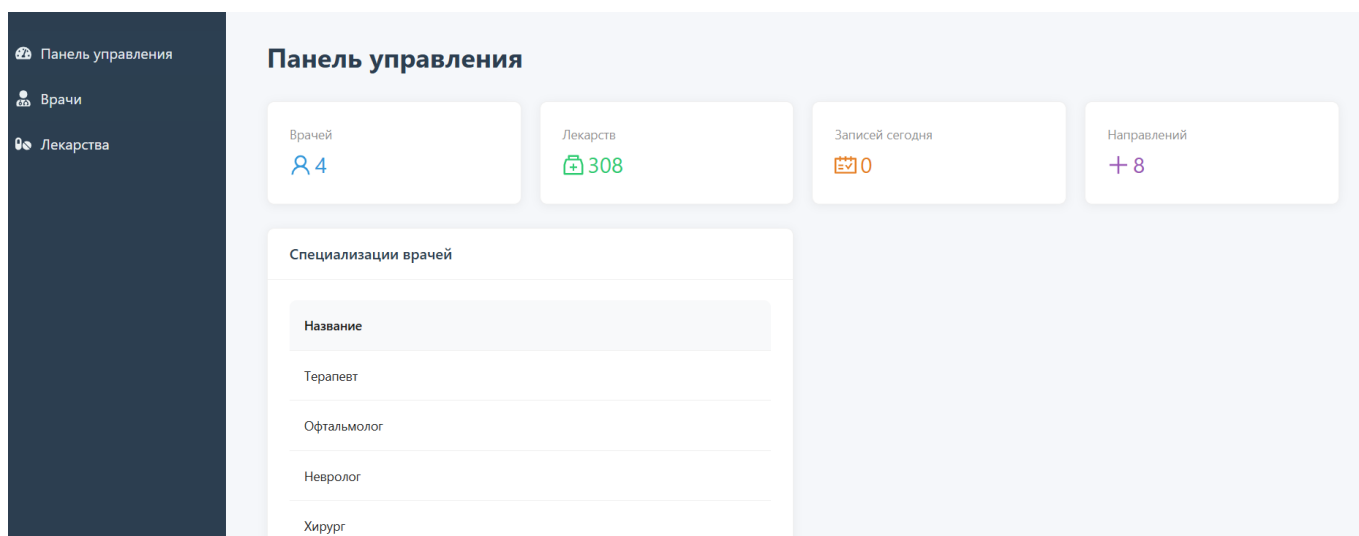


Рисунок 5.20 – Главная страница админ-панели

Также в панели управления есть возможно добавить новую специализацию.

5.4.1 Добавление врача





В боковой панели указаны ссылки для перехода на страницы «Врачи» и «Лекарства». Если администратор зайдёт по ссылке «Врачи», то отобразится страница таблица с существующими врачами. Страница врачей представлена на рисунке 5.21.

Управление врачами

+ Добавить врача

Поиск по ФИО

Поиск по специализации

Фамилия	Имя	Отчество	Дата рождения	Телефон	Специализация	Категория	Фото	Действия
Быков	Андрей	Евгеньевич	14.11.1975	80333600100	Терапевт	Доктор медицинских наук		<div><div></div><div></div></div>
Купитман	Иван	Натанович	24.01.1977	80334563754	Гинеколог	Доктор медицинских наук		<div><div></div><div></div></div>
Лобанов	Семён	Семёнович	10.01.1994	80296782367	Травмотолог	Первая		<div><div></div><div></div></div>
Ричард	Филип	Байрен	13.11.1997	80291237643	Венеролог	Вторая		<div><div></div><div></div></div>

Всего врачей: 4

< 1 >

10 / page

Рисунок 5.21 – Страница «Врачи»

На странице есть возможность фильтрации списка врачей по ФИО и специализации. Для добавления нового врача администратору нужно нажать на кнопку «Добавить врача». Тогда откроется модально окно с формой для заполнения. Модальное окно создания врача представлено на рисунке 5.22.

Добавить врача

* Фамилия: Скрябина

* Имя: Любовь

* Отчество: Михайловна

* Дата рождения: 1989-02-12

* Телефон: 80293456173

* Специализация: Психолог

* Категория: Кандидат медицинских наук

Фото врача

Загрузить фото

Люба.jpg

Расписание

Рисунок 5.22 – Окно добавления врача

При создании врача так же нужно указать расписание его работы. Так как нужно реализовать гибкое управление, то для каждого дня настройка расписания уникальна. Пример настройки рабочего дня представлен на рисунке 5.23.

Расписание

Понедельник

Начало работы: 09:00

Конец работы: 18:00

Интервал приема: 30 минут

Вторник

Среда

Четверг

Пятница

Рисунок 5.23 – Пример заполнения расписания работы врача

После заполнения полей пользователь должен нажать на кнопку «ОК». После создания врача для него создаётся соответствующий пользователь. Данные для входа

нигде не сохраняются, поэтому после создания врача появляется модальное окно с логином и паролем для добавленного врача. Пример модального окна представлен на рисунке 5.24

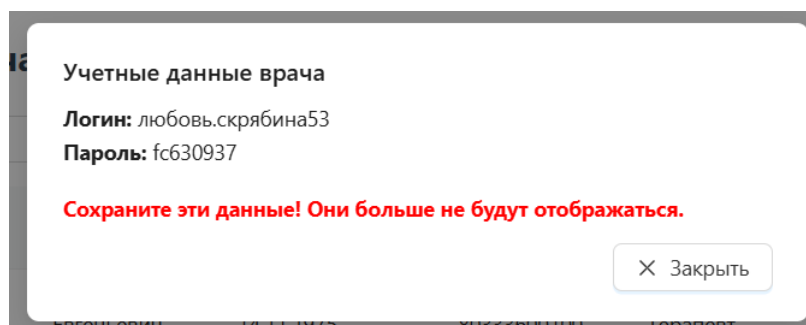


Рисунок 5.24 – Пример модального окна с данными для входа

После добавления новый врач появляется в таблице.

5.4.2 Изменение врача и его графика работы

Для изменения данных в таблице в столбце «Действия» есть кнопка «ручка». При нажатии на которую открывается модальное окно в данным, которые можно изменить. Столбец «Действия» представлен на рисунке 5.25.

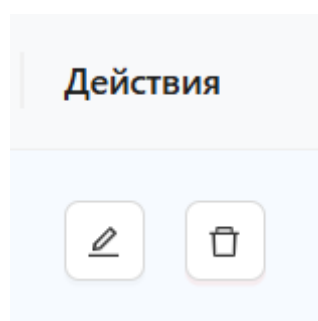


Рисунок 5.25 – Столбец «Действия»

Открытое модальное окно сохраняется таким же образом, как и при создании нового врача. Новые данные отобразятся в таблице

5.4.3 Удаление врача

Для удаления врача в том же столбце «Действия» располагается корзина, при нажатии на которую появляется модальное окно для подтверждения удаления. После подтверждения врач больше не отображается в таблице

5.4.4 Добавление лекарства

При переходе на страницу «Лекарства» отображается страница с таблицей лекарств. По аналогии в таблице врачей на этой странице так же есть фильтрация и поиск. Страница лекарств представлена на рисунке 5.26.

Управление лекарствами						+ Добавить лекарство
<input type="text" value="Поиск по МНН"/>		<input type="text" value="Поиск по фарм. подгруппе"/>				
Фарм. подгруппа	МНН	Форма выпуска	Цена	Статус	Действия	
A02A Антацидные средства	Алюминия фосфат	Суспензия	22.55 BYN	Доступен		
A02A Антацидные средства	Соединения, содержащие гидроокись алюминия и магния	Суспензия для внутреннего применения, таблетки	20.50 BYN	Доступен		
A02B Противоязвенные средства и средства, применяемые при гастроэзофагеальной рефлюксной болезни	Ранитидин	Раствор для инъекций 25 мг/мл 2 мл	10.50 BYN	Доступен		
A02B Противоязвенные средства и средства, применяемые при гастроэзофагеальной рефлюксной болезни	Фамотидин	Таблетки, покрытые оболочкой, 20 мг	2.94 BYN	Доступен		

Рисунок 5.26 – Страница «Лекарства»

Для добавления нового лекарства администратор должен нажать на кнопку «Добавить лекарство». После этого откроется модальное окно с формой, представленной на рисунке 5.27.

Добавить лекарство

* Фармацевтическая подгруппа

Выберите фарм. подгруппу

* Международное непатентованное название (МНН)

* Форма выпуска

* Цена

BYN

Статус

☐ Недоступен

Cancel

OK

Рисунок 5.27 – Форма для добавления лекарства

После добавления новое лекарство отображается в таблице с остальными.

5.4.5 Изменение статуса доступности и цены лекарства

Для изменения статуса лекарств пользователю хватит нажать на переключатель и подтвердить изменение в модальном окне. Для изменения цены нужно нажать на цену лекарства и в появившемся окне изменить. Все изменения сразу отображаются

в таблице лекарств.

5.4.6 Удаление лекарства

По аналогии в таблице врачей в таблице лекарств тоже есть столбец «Действия». Для удаления требуется нажать на корзину и подтвердить удаление в модальном окне.

5.5 Выводы по разделу

1. Описано подробно руководство, в котором изложены действия, выполняющие пользователи разных ролей в системе: «Гость», «Клиент», «Врач» и «Администратор».

2. Гости могут просматривать информацию о врачах, а также регистрироваться и аутентифицироваться.

3. Клиенты могут записываться на приёмы к врачу, просматривать свои записи, и отменять их. Так же, как и гость, клиенты могут просматривать информацию о врачах, а также могут фильтровать их список по направлению.

4. Врачи могут читать историю обследований пациента, записывать в медкарты пациентов результаты обследования, диагностированные заболевания, а также назначенные лекарства с дозировкой и длительностью приёма.

5. Администраторы могут добавлять, редактировать, удалять объекты из списка врачей или лекарств. Также они могут добавлять новые специальности для врачей.

Заключение

В ходе выполнения курсового проекта была определена основная функциональность web-приложения медцентра, выделены роли (гость, клиент, врач, администратор) и разграничены варианты использования по ролям.

Была рассмотрена логическая схема базы данных web-приложения, которая включает тринадцать таблиц Diagnoses, Diagnosed_Disease, Disease Prescribed_Medications, Medical_Results, Patients, Doctors, Specializations, Appointments, Users, Token_Users, Medications, Schedules.

Для серверной части было использовано приложение, разработанное на платформе Node.js 20.16.0 с использованием фреймворка Express 4.21.2. База данных была реализована с помощью PostgreSQL 17.2. Для взаимодействия с базой данных применялся ORM Sequelize 6.37.6. Клиентская часть приложения была разработана с использованием библиотеки React.js 18.

Общее количество маршрутов сервера вышло 41, которые принимаю 5 методов (GET, POST, PUT, PATCH, DELETE). Общее количество контроллеров вышло 8.

Реализованы все функции для всех ролей web-приложения: гостя, клиента, врача и администратора. Количество функций web-приложения составило 19.

Рассмотрена структура клиентской части проекта, которое использует современные программные библиотеки. Общее количество маршрутов клиентской части вышло 19. Общее количество компоненты необходимых для отображения страниц для пользователей составило 27.

Реализована контейнеризация с использованием Docker 28.0.1. Оркестрация контейнеров выполнена через Docker Compose 2.35.1, а также был настроен Nginx 1.27.5.

Было выполнено ручное тестирование всех ключевых функций web-приложения. Корректность работы системы подтверждена соответствием фактических результатов тестирования ожидаемым. Количество тестов составило 19, покрытие тестами 100%.

Список используемых источников

- 1 Node.js Documentation [Электронный ресурс] / Режим доступа: <https://nodejs.org/en/docs/> – Дата доступа: 14.03.2025.
- 2 React Official Documentation [Электронный ресурс] / Режим доступа: <https://react.dev/> – Дата доступа: 14.03.2025.
- 3 PostgreSQL Documentation [Электронный ресурс] / Режим доступа: <https://www.postgresql.org/docs/> – Дата доступа: 14.03.2025.
- 4 Официальный сайт медицинского центра ЛОДЭ [Электронный ресурс] / Режим доступа: <https://lode.by/> – Дата доступа: 14.03.2025.
- 5 Nginx Official Documentation [Электронный ресурс] / Режим доступа: <https://nginx.org/en/docs/> – Дата доступа: 20.03.2025.
- 6 PR-CY Analytics Tool [Электронный ресурс] / Режим доступа: <https://a.pr-cy.ru/> – Дата доступа: 20.03.2025.
- 7 Официальный сайт клиники «Мед Авеню» [Электронный ресурс] / Режим доступа: <https://med-avenue.ru/> – Дата доступа: 23.03.2025.
- 8 Urgent Care [Электронный ресурс] / Режим доступа: <https://medcenterurgentcare.com/> – Дата доступа: 23.03.2025.
- 9 JWT Introduction [Электронный ресурс] / Режим доступа: <https://jwt.io/introduction/> – Дата доступа: 23.03.2025.
- 10 Sequelize ORM Documentation [Электронный ресурс] / Режим доступа: <https://sequelize.org/> – Дата доступа: 23.03.2025.
- 11 HTTP Protocol Specification [Электронный ресурс] / Режим доступа: <https://httpwg.org/specs/> – Дата доступа: 23.03.2025.
- 12 TCP Protocol RFC [Электронный ресурс] / Режим доступа: <https://tools.ietf.org/html/rfc793> – Дата доступа: 23.03.2025.
- 13 Cloudinary Official Documentation [Электронный ресурс] / Режим доступа: <https://cloudinary.com/documentation> – Дата доступа: 23.03.2025.
- 14 REST API Concepts [Электронный ресурс] / Режим доступа: <https://restfulapi.net/> – Дата доступа: 24.03.2025.
- 15 Sequelize Documentation (Code First Approach) [Электронный ресурс]. - Режим доступа: <https://sequelize.org/docs/v6/core-concepts/model-basics/>. - Дата доступа: 24.03.2025.
- 16 Bcrypt.js Documentation [Электронный ресурс] / Режим доступа: <https://github.com/kelektiv/node.bcrypt.js> – Дата доступа: 24.03.2025.
- 17 CORS Middleware [Электронный ресурс] / Режим доступа: <https://github.com/expressjs/cors> – Дата доступа: 24.03.2025.
- 18 Cookie-parser npm [Электронный ресурс] / Режим доступа: <https://www.npmjs.com/package/cookie-parser> – Дата доступа: 24.03.2025.
- 19 Dotenv Package [Электронный ресурс] / Режим доступа: <https://github.com/motdotla/dotenv> – Дата доступа: 24.03.2025.
- 20 .env File Syntax [Электронный ресурс] / Режим доступа: <https://github.com/motdotla/dotenv#rules> – Дата доступа: 24.03.2025.
- 21 Express-validator GitHub [Электронный ресурс] / Режим доступа: <https://github.com/express-validator/express-validator> – Дата доступа: 24.03.2025.

22 Jsonwebtoken Documentation [Электронный ресурс] / Режим доступа: <https://github.com/auth0/node-jsonwebtoken> – Дата доступа: 24.03.2025.

23 Multer Middleware [Электронный ресурс] / Режим доступа: <https://github.com/expressjs/multer> – Дата доступа: 24.03.2025.

24 Multer Storage Cloudinary Documentation [Электронный ресурс] // GitHub Repository. – Режим доступа: <https://github.com/affanshahid/multer-storage-cloudinary>. – Дата обращения: 23.03.2025.

25 Node-postgres [Электронный ресурс] / Режим доступа: <https://node-postgres.com/> – Дата доступа: 26.03.2025.

26 Pg-hstore Package [Электронный ресурс] / Режим доступа: <https://github.com/scarney81/pg-hstore> – Дата доступа: 26.03.2025.

27 Uuid Package [Электронный ресурс] / Режим доступа: <https://github.com/uuidjs/uuid> – Дата доступа: 26.03.2025.

28 Ant Design Icons Documentation [Электронный ресурс] // Ant Financial. - Режим доступа: <https://ant.design/components/icon/>. - Дата доступа: 23.03.2025.

29 Fluent UI React Documentation [Электронный ресурс] // Microsoft Corporation. - Режим доступа: <https://developer.microsoft.com/en-us/fluentui#/>. - Дата доступа: 23.03.2025.

30 Redux Toolkit [Электронный ресурс] / Режим доступа: <https://redux-toolkit.js.org/> – Дата доступа: 23.03.2025.

31 Ant Design Component Library [Электронный ресурс] // Ant Financial. - Режим доступа: <https://ant.design/docs/react/introduce>. - Дата доступа: 23.02.2025.

32 Axios Documentation [Электронный ресурс] / Режим доступа: <https://axios-http.com/docs/intro> – Дата доступа: 22.03.2025.

33 Moment.js Documentation [Электронный ресурс] // JS Foundation. - Режим доступа: <https://momentjs.com/docs/>. - Дата доступа: 23.02.2025.

34 React DOM Docs [Электронный ресурс] / Режим доступа: <https://react.dev/reference/react-dom> – Дата доступа: 17.03.2025.

35 React Router [Электронный ресурс] / Режим доступа: <https://reactrouter.com/> – Дата доступа: 17.03.2025.

36 React-Redux [Электронный ресурс] / Режим доступа: <https://react-redux.js.org/> – Дата доступа: 17.03.2025.

37 Swiper.js Documentation [Электронный ресурс] // Swiper Team. - Режим доступа: <https://swiperjs.com/get-started>. - Дата обращения: 23.02.2025.

38 JSON Specification [Электронный ресурс] // Ecma International. – Режим доступа: <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>. – Дата обращения: 23.02.2025.

39 Docker Official Documentation [Электронный ресурс] / Режим доступа: <https://docs.docker.com/> – Дата доступа: 10.05.2025.

40 Docker Compose Docs [Электронный ресурс] / Режим доступа: <https://docs.docker.com/compose/> – Дата доступа: 10.05.2025.

Приложение А

Скрипт создания базы данных

```

CREATE TABLE IF NOT EXISTS public."Users"
(
    "ID" uuid NOT NULL,
    "Login" character varying(50) COLLATE pg_catalog."default" NOT
NULL,
    "Password" character varying(100) COLLATE pg_catalog."default"
NOT NULL,
    "Role" character varying(30) COLLATE pg_catalog."default" NOT
NULL,
    "Registration_Date" timestamp with time zone NOT NULL,
    CONSTRAINT "Users_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Users_Login_key" UNIQUE ("Login")
)

CREATE TABLE IF NOT EXISTS public."TokenUsers"
(
    "ID" uuid NOT NULL,
    "User_ID" uuid NOT NULL,
    "RefreshToken" character varying(300) COLLATE
pg_catalog."default" NOT NULL,
    CONSTRAINT "TokenUsers_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "TokenUsers_User_ID_fkey" FOREIGN KEY ("User_ID")
REFERENCES public."Users" ("ID") MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE
)

CREATE TABLE IF NOT EXISTS public."Specializations"
(
    "ID" uuid NOT NULL,
    "Name" text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Specializations_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Specializations_Name_key" UNIQUE ("Name")
)

CREATE TABLE IF NOT EXISTS public."Schedules"
(
    "ID" uuid NOT NULL,
    "Doctor_ID" uuid NOT NULL,
    "Day_of_week" character varying(20) COLLATE pg_catalog."default"
NOT NULL,
    "Start_time" time without time zone NOT NULL,
    "End_time" time without time zone NOT NULL,
    "Interval" time without time zone NOT NULL,
    CONSTRAINT "Schedules_pkey" PRIMARY KEY ("ID")
)

CREATE TABLE IF NOT EXISTS public."Prescribed_Medications"
(

```

```

        "ID" uuid NOT NULL,
        "Diagnosis_ID" uuid NOT NULL,
        "Medication_ID" uuid NOT NULL,
        "Dosage" character varying(100) COLLATE pg_catalog."default" NOT
NULL,
        "Duration" integer NOT NULL,
        CONSTRAINT "Prescribed_Medications_pkey" PRIMARY KEY ("ID")
    )
CREATE TABLE IF NOT EXISTS public."Patients"
(
    "ID" uuid NOT NULL,
    "First_Name" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Last_Name" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Patronymic" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Birthdate" date,
    "Gender" character(1) COLLATE pg_catalog."default" NOT NULL
DEFAULT 'M'::bpchar,
    "Phone" character varying(20) COLLATE pg_catalog."default" NOT
NULL,
    "User_ID" uuid,
    CONSTRAINT "Patients_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Patients_Phone_key" UNIQUE ("Phone"),
    CONSTRAINT "Patients_User_ID_fkey" FOREIGN KEY ("User_ID")
        REFERENCES public."Users" ("ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
    )
CREATE TABLE IF NOT EXISTS public."Medications"
(
    "ID" uuid NOT NULL,
    "Pharm_Subgroup" character varying(250) COLLATE
pg_catalog."default" NOT NULL,
    "IGN" character varying(200) COLLATE pg_catalog."default" NOT
NULL,
    "Dosage_Form" character varying(200) COLLATE
pg_catalog."default" NOT NULL,
    "Price" numeric(10,2) NOT NULL,
    "isAvailable" boolean NOT NULL,
    CONSTRAINT "Medications_pkey" PRIMARY KEY ("ID")
    )
CREATE TABLE IF NOT EXISTS public."Medical_Results"
(
    "ID" uuid NOT NULL,
    "Patient_ID" uuid NOT NULL,
    "Doctor_ID" uuid NOT NULL,
    "Result_Description" text COLLATE pg_catalog."default" NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    CONSTRAINT "Medical_Results_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Medical_Results_Patient_ID_fkey" FOREIGN KEY
("Patient_ID")

```



```

        REFERENCES public."Patients" ("ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
    )
CREATE TABLE IF NOT EXISTS public."Doctors"
(
    "ID" uuid NOT NULL,
    "First_Name" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Last_Name" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Patronymic" character varying(50) COLLATE pg_catalog."default"
NOT NULL,
    "Birthdate" date NOT NULL,
    "Phone" character varying(20) COLLATE pg_catalog."default" NOT
NULL,
    "Specialization_ID" uuid,
    "Category" character varying(100) COLLATE pg_catalog."default"
NOT NULL,
    "Photo" character varying(255) COLLATE pg_catalog."default" NOT
NULL DEFAULT NULL::character varying,
    "User_ID" uuid,
    CONSTRAINT "Doctors_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Doctors_Phone_key" UNIQUE ("Phone"),
    CONSTRAINT "Doctors_Specialization_ID_fkey" FOREIGN KEY
("Specialization_ID")
        REFERENCES public."Specializations" ("ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT "Doctors_User_ID_fkey" FOREIGN KEY ("User_ID")
        REFERENCES public."Users" ("ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)
CREATE TABLE IF NOT EXISTS public."Diseases"
(
    "ID" uuid NOT NULL,
    "Name" text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "Diseases_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Diseases_Name_key" UNIQUE ("Name")
)
CREATE TABLE IF NOT EXISTS public."Diagnoses"
(
    "ID" uuid NOT NULL,
    "Patient_ID" uuid NOT NULL,
    "Doctor_ID" uuid NOT NULL,
    "Date" timestamp with time zone NOT NULL,
    CONSTRAINT "Diagnoses_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Diagnoses_Patient_ID_fkey" FOREIGN KEY
("Patient_ID")
        REFERENCES public."Patients" ("ID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE

```

```

)
CREATE TABLE IF NOT EXISTS public."DiagnosedDiseases"
(
    "ID" uuid NOT NULL,
    "Diagnosis_ID" uuid NOT NULL,
    "Disease_ID" uuid NOT NULL,
    CONSTRAINT "DiagnosedDiseases_pkey" PRIMARY KEY ("ID")
)
CREATE TABLE IF NOT EXISTS public."Appointments"
(
    "ID" uuid NOT NULL,
    "Patient_ID" uuid NOT NULL,
    "Doctor_ID" uuid NOT NULL,
    "Appointment_Date" timestamp with time zone NOT NULL,
    "isComplited" boolean NOT NULL,
    "isDenied" boolean NOT NULL,
    CONSTRAINT "Appointments_pkey" PRIMARY KEY ("ID"),
    CONSTRAINT "Appointments_Patient_ID_fkey" FOREIGN KEY
("Patient_ID")
    REFERENCES public."Patients" ("ID") MATCH SIMPLE
    ON UPDATE CASCADE
    ON DELETE CASCADE
)

```

Приложение Б

Скрипт реализации основных сервисов

```

const {connection, User, Patient} = require('../config/database');
const bcrypt = require('bcrypt');
const TokenService = require('../services/tokenService');
const UserDto = require('../dtos/userDto');
const ApiError = require('../exceptions/api-error');
let tokenService = new TokenService();
class UserServices {
  registration = async(login, password) => {
    const candidate = await User.findOne({where: {Login:login}});
    if(candidate){
      throw ApiError.BadRequest(`Пользователь с логином ${login}
уже существует`);
    }
    const hashPassword = await bcrypt.hash(password, 3);
    const user = await User.create({Login:login, Password:
hashPassword, Role: 'client'});
    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({userDto});
    await tokenService.saveToken(userDto.id, tokens.refreshToken);
    return { ...tokens, user: userDto }
  }
  login = async(login, password) => {
    const user = await User.findOne({where: {Login: login}});
    if(!user){
      throw ApiError.BadRequest('Пользователь с таким логином не
найден');
    }
    const isPassEquals = await bcrypt.compare(password,
user.Password);
    if(!isPassEquals){
      throw ApiError.BadRequest('Неверный пароль');
    }
    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({userDto});
    await tokenService.saveToken(userDto.id, tokens.refreshToken);
    return { ...tokens, user: userDto }
  }
  logout = async(refreshToken) => {
    const token = await tokenService.removeToken(refreshToken);
    return token;
  }
  refresh = async(refreshToken) => {
    if(!refreshToken){
      throw ApiError.UnauthorizedError()
    }
    const userData = await
tokenService.validateRefreshToken(refreshToken);
    const tokenFromDb = await tokenService.findToken(refreshToken);

```

```

    if(!userData || !tokenFromDb){
      throw ApiError.UnauthorizedError();
    }
    const user = await User.findByPk(userData.id);
    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({userDto});
    await tokenService.saveToken(userDto.id, tokens.refreshToken);
    return { ...tokens, user: userDto }
  }
  getUserData = async (id) => {
    const user = await User.findByPk(id);
    if (!user) {
      throw ApiError.BadRequest('Пользователь не найден');
    }
    const userDto = new UserDto(user);
    const tokens = tokenService.generateTokens({userDto});
    await tokenService.saveToken(userDto.id, tokens.refreshToken);
    return { ...tokens, user: userDto }
  }
  getUserId = async (id) => {
    const user = await User.findByPk(id);
    if (!user) {
      throw new Error('User not found');
    }
    return new UserDto(user);
  };
  insertUser = async (login, password) => {
    const hashPassword = await bcrypt.hash(password, 3);
    const user = await User.create({Login: login, Password:
hashPassword});
    return new UserDto(user);
  };
  updateUsername = async (id, username) => {
    const user = await User.findByPk(id);
    if (!user) {
      throw new Error('User not found');
    }
    const existingUser = await User.findOne({ where: { Login:
username } });
    if (existingUser && existingUser.ID !== id) {
      throw new Error('Этот логин уже занят');
    }
    const [updated] = await User.update(
      { Login: username },
      { where: { ID: id } }
    );
    if (!updated) {
      throw new Error('Не удалось обновить логин');
    }
    const updatedUser = await User.findByPk(id);
    return new UserDto(updatedUser);
  };
  deleteUser = async (id) => {

```

```

    const user = await User.findByPk(id);
    if (!user) {
        throw new Error('User not found');
    }
    const patient = await Patient.findOne({where: {User_ID: id}});
    if (patient) {
        await patient.update({User_ID: null});
    }
    await User.destroy({where: {ID: id}});
    return new UserDto(user);
};

isAdmin = async (userId) => {
    const user = await User.findByPk(userId);
    return user?.Role === 'admin';
};

isDoctor = async (userId) => {
    const user = await User.findByPk(userId);
    return user?.Role === 'doctor';
};
}

module.exports = UserServices;

const jwt = require('jsonwebtoken');
const {TokenUser} = require('../config/database');

class TokenService{
    generateTokens(payload){
        const accessToken = jwt.sign(payload,
process.env.JWT_ACCESS_SECRET, {expiresIn: '60m'});
        const refreshToken = jwt.sign(payload,
process.env.JWT_REFRESH_SECRET, {expiresIn: '30d'});
        return{
            accessToken,
            refreshToken
        }
    }
    async saveToken(userId, refreshToken){
        const tokenData = await TokenUser.findOne({where: {User_ID:
userId}});
        if(tokenData){
            tokenData.dataValues.RefreshToken = refreshToken;
            return await TokenUser.update({RefreshToken:refreshToken},
{where:{User_ID: userId}})
        }
        const token = await TokenUser.create({User_ID:userId,
RefreshToken:refreshToken});
        return token;
    }
    async removeToken(refreshToken){
        console.log(refreshToken);
        const tokenData = await
TokenUser.destroy({where:{RefreshToken:refreshToken}});
        return tokenData;
    }
}

```

```

    }
    async validateAccessToken(token) {
      try{
        const          userData          =          jwt.verify(token,
process.env.JWT_ACCESS_SECRET);
        return userData;
      }catch(e){
        return null;
      }
    }
    async validateRefreshToken(token) {
      try{
        const          userData          =          jwt.verify(token,
process.env.JWT_REFRESH_SECRET).userDto;
        return userData;
      }catch(e){
        return null;
      }
    }
    async findToken(refreshToken){
      const          tokenData          =          await
TokenUser.findOne({where:{RefreshToken:refreshToken}});
      return tokenData;
    }
  }
}
module.exports = TokenService;
const { connection, Patient } = require('../config/database');
const PatientDto = require('../dtos/patientDto');
class PatientServices {
  getPatients = async () => {
    const patients = await Patient.findAll();
    return patients.map(patient => new PatientDto(patient));
  }
  getPatientById = async (id) => {
    const patient = await Patient.findPk(id);
    if (!patient) {
      throw new Error('Patient not found');
    }
    return new PatientDto(patient);
  };
  insertPatient = async (data) => {
    const patient = await Patient.create(data);
    return new PatientDto(patient);
  };
  updatePatient = async (id, updates) => {
    const patient = await Patient.findPk(id);
    if (!patient) {
      throw new Error('Patient not found');
    }
    await Patient.update(updates, { where: { ID: id } });
    const updatedPatient = await Patient.findPk(id);
    return new PatientDto(updatedPatient);
  };
};

```

```

deletePatient = async (id) => {
  const patient = await Patient.findByPk(id);
  if (!patient) {
    throw new Error('Patient not found');
  }
  await Patient.destroy({ where: { ID: id } });
  return {success: true};
};

getPatientDetails = async (patientId) => {
  const patient = await Patient.findByPk(patientId);
  if (!patient) {
    throw new Error('Patient not found');
  }
  return new PatientDto(patient);
};

updatePatientMedicalInfo = async (patientId, updates) => {
  const patient = await Patient.findByPk(patientId);
  if (!patient) {
    throw new Error('Patient not found');
  }
  await patient.update(updates);
  return new PatientDto(patient);
};
}

module.exports = PatientServices;
const { Medication } = require('../config/database');
const MedicationDto = require('../dtos/medicationDto');
const ApiError = require('../exceptions/api-error');
class MedicineService {
  async getAllMedicines() {
    const medicines = await Medication.findAll();
    return medicines.map(medicine => new MedicationDto(medicine));
  }
  async getMedicineById(id) {
    const medicine = await Medication.findByPk(id);
    if (!medicine) {
      throw ApiError.BadRequest('Лекарство не найдено');
    }
    return new MedicationDto(medicine);
  }
  async addMedicine(data) {
    const medicine = await Medication.create({
      Pharm_Subgroup: data.Pharm_Subgroup,
      IGN: data.IGN,
      Dosage_Form: data.Dosage_Form,
      Price: data.Price,
      isAvailable: data.isAvailable
    });
    return new MedicationDto(medicine);
  }
  async updateMedicineStatus(id, isAvailable) {
    const medicine = await Medication.findByPk(id);
    if (!medicine) {

```

```

        throw ApiError.BadRequest('Лекарство не найдено');
    }
    await medicine.update({isAvailable: isAvailable});
    return new MedicationDto(medicine);
}
async updateMedicinePrice(id, price) {
    const medicine = await Medication.findByPk(id);
    if (!medicine) {
        throw ApiError.BadRequest('Лекарство не найдено');
    }
    await medicine.update({ Price: price });
    return new MedicationDto(medicine);
}
async deleteMedicine(id) {
    const medicine = await Medication.findByPk(id);
    if (!medicine) {
        throw ApiError.BadRequest('Лекарство не найдено');
    }
    await medicine.destroy();
    return { message: 'Лекарство успешно удалено' };
}
}
module.exports = MedicineService;
const { MedicalResult, Diagnosis, Disease, DiagnosedDisease, Medication,
PrescribedMedication, Appointment } = require('../config/database');
const ApiError = require('../exceptions/api-error');
class MedicalService {
    async getDiseases() {
        const diseases = await Disease.findAll({
            attributes: ['ID', 'Name'],
            order: [['Name', 'ASC']]
        });
        return diseases;
    }
    async getMedications() {
        const medications = await Medication.findAll({
            attributes: ['ID', 'IGN', 'Dosage_Form', 'Pharm_Subgroup'],
            where: { isAvailable: true },
            order: [['IGN', 'ASC']]
        });
        return medications;
    }
    async getPatientMedicalResults(patientId) {
        const results = await MedicalResult.findAll({
            where: { Patient_ID: patientId }
        });
        return results;
    }
    async getDoctorMedicalResults(doctorId) {
        const results = await MedicalResult.findAll({
            where: { Doctor_ID: doctorId }
        });
        return results;
    }
}

```



```

    }
    async createMedicalResult(patientId, description, diseases,
medications, doctorId, appointmentId) {
      try {
        const now = new Date();
        const medicalResult = await MedicalResult.create({
          Patient_ID: patientId,
          Doctor_ID: doctorId,
          Result_Description: description,
          Date: now
        });
        const appointment = await
Appointment.findByPk(appointmentId);
        if (!appointment) {
          throw new Error(`Appointment ${appointmentId} not found`);
        }
        appointment.isCompleted = true;
        await appointment.save();
        let diagnosis = null;
        if (diseases && diseases.length > 0) {
          diagnosis = await Diagnosis.create({
            Patient_ID: patientId,
            Doctor_ID: doctorId,
            Date: now
          });
          await DiagnosedDisease.bulkCreate(
            diseases.map(diseaseId => ({
              Diagnosis_ID: diagnosis.ID,
              Disease_ID: diseaseId
            })))
        );
      }
      if (medications && medications.length > 0) {
        if (!diagnosis) {
          diagnosis = await Diagnosis.create({
            Patient_ID: patientId,
            Doctor_ID: doctorId,
            Date: now
          });
        }
        await PrescribedMedication.bulkCreate(
          medications.map(med => ({
            Diagnosis_ID: diagnosis.ID,
            Medication_ID: med.medicationId,
            Dosage: med.dosage,
            Duration: med.duration
          })), {
            fields: ['ID', 'Diagnosis_ID', 'Medication_ID',
'Dosage', 'Duration']
          }
        );
      }
      return medicalResult;
    }
  }
}

```

```

    } catch (error) {
      console.error('Detailed error:', error);
      throw ApiError.BadRequest('Ошибка при создании медицинского
результата: ' + error.message);
    }
  }
  async createDiagnosis(patientId, diseaseIds, notes, doctorId) {
    try {
      const diagnosis = await Diagnosis.create({
        Patient_ID: patientId,
        Doctor_ID: doctorId,
        Date: new Date()
      });
      if (diseaseIds && diseaseIds.length > 0) {
        await DiagnosedDisease.bulkCreate(
          diseaseIds.map(diseaseId => ({
            Diagnosis_ID: diagnosis.ID,
            Disease_ID: diseaseId
          })))
      );
    }
    return diagnosis;
  } catch (error) {
    throw ApiError.BadRequest('Ошибка при создании диагноза: '
+ error.message);
  }
}
  async prescribeMedication(diagnosisId, medicationId, dosage,
duration) {
    try {
      const prescription = await PrescribedMedication.create({
        Diagnosis_ID: diagnosisId,
        Medication_ID: medicationId,
        Dosage: dosage,
        Duration: duration
      });
      return prescription;
    } catch (error) {
      throw ApiError.BadRequest('Ошибка при назначении лекарства:
' + error.message);
    }
  }
  async getPatientDiagnoses(patientId) {
    const diagnoses = await Diagnosis.findAll({
      where: { Patient_ID: patientId },
      include: [{
        model: Disease,
        through: { attributes: [] },
        attributes: ['ID', 'Name']
      }],
      order: [['Date', 'DESC']]
    });
    return diagnoses;
  }
}

```

```

    }
    async getPatientDiagnosisData(patientId) {
      const diagnoses = await Diagnosis.findAll({
        where: { Patient_ID: patientId },
        include: [
          {
            model: DiagnosedDisease,
            include: [Disease]
          },
          {
            model: PrescribedMedication,
            include: [Medication]
          }
        ],
        order: [['Date', 'DESC']]
      });
      return diagnoses.map(diagnosis => ({
        id: diagnosis.ID,
        date: diagnosis.Date,
        doctorId: diagnosis.Doctor_ID,
        diseases: diagnosis.DiagnosedDiseases?.map(dd => ({
          id: dd.Disease?.ID,
          name: dd.Disease?.Name
        })) || [],
        medications: diagnosis.PrescribedMedications?.map(pm => ({
          id: pm.Medication?.ID,
          name: pm.Medication?.IGN,
          form: pm.Medication?.Dosage_Form,
          dosage: pm.Dosage,
          duration: pm.Duration
        })) || []
      }));
    }
  }
}

module.exports = MedicalService;
const { Doctor, Appointment, Patient, User, Schedule, Specialization }
= require('../config/database');
const DoctorDto = require('../dtos/doctorDto');
const PatientDto = require('../dtos/patientDto');
const UserDto = require('../dtos/userDto');
const {generateLogin, generatePassword} = require('../utils/userUtils')
const bcrypt = require('bcrypt');
const cloudinary = require('../config/cloudinary.js');
const SpecializationDto = require('../dtos/specializationDto.js');
class DoctorServices {
  addDoctor = async (data) => {
    try {
      const login = generateLogin(data.First_Name,
data.Last_Name);
      const password = generatePassword();
      const hashPassword = await bcrypt.hash(password, 3);
      const candidate = await User.findOne({
        where: { Login: login }

```

```

    });
    if (candidate) {
        throw Error(`Пользователь с логином ${login} уже
существует`);
    }
    const user = await User.create({
        Login: login,
        Password: hashPassword,
        Role: 'doctor'
    });
    let specialization;
    if (data.Specialization_ID) {
        specialization = await
Specialization.findByIdPk(data.Specialization_ID);
        if (!specialization) {
            throw Error('Указанная специализация не найдена');
        }
    } else if (data.Specialization) {
        [specialization] = await Specialization.findOrCreate({
            where: { Name: data.Specialization },
            defaults: { Name: data.Specialization }
        });
    } else {
        throw Error('Не указана специализация врача');
    }
    const doctor = await Doctor.create({
        First_Name: data.First_Name,
        Last_Name: data.Last_Name,
        Patronymic: data.Patronymic,
        Birthdate: data.Birthdate,
        Phone: data.Phone,
        Specialization_ID: specialization.ID,
        Category: data.Category,
        Photo: data.Photo,
        User_ID: user.ID
    });
    let schedulesToCreate = [];
    console.log(data.Schedule)
    try {
        if (data.Schedule) {
            schedulesToCreate = typeof data.Schedule ===
'string'
                ? JSON.parse(data.Schedule)
                : data.Schedule;
            if (!Array.isArray(schedulesToCreate)) {
                throw new Error('Schedule should be an array');
            }
        }
    } catch (e) {
        console.error('Schedule parsing error:', e);
        throw Error('Неверный формат расписания');
    }
    if (schedulesToCreate.length > 0) {

```

```

        await Promise.all(
            schedulesToCreate.map(schedule => {
                if (!schedule.day_of_week ||
!schedule.start_time || !schedule.end_time || !schedule.interval) {
                    throw Error('Неверный формат данных
расписания');
                }
                return Schedule.create({
                    Doctor_ID: doctor.ID,
                    Day_of_week: schedule.day_of_week,
                    Start_time: schedule.start_time,
                    End_time: schedule.end_time,
                    Interval: schedule.interval
                });
            })
        );
    }
    const doctorWithSchedule = await Doctor.findByPk(doctor.ID,
{
    include: [
        { model: Schedule, as: 'Schedules' },
        { model: Specialization, as: 'Specialization' }
    ]
});
    return {
        doctor: new DoctorDto(doctorWithSchedule),
        credentials: { login, password }
    };
} catch (error) {
    if (data.Photo && data.Photo.includes('cloudinary')) {
        const publicId =
data.Photo.split('/').pop().split('.')[0];
        await
cloudinary.uploader.destroy(`doctors/${publicId}`);
    }
    throw error;
}
}
updateDoctor = async (id, updates, oldPhotoPublicId) => {
    try {
        const doctor = await Doctor.findByPk(id);
        if (!doctor) throw Error('Doctor not found');
        if (updates.Photo && oldPhotoPublicId) {
            try {
                await
cloudinary.uploader.destroy(oldPhotoPublicId);
            } catch (e) {
                console.error('Error deleting old photo:', e);
            }
        }
        await doctor.update(updates);

        await Schedule.destroy({

```

```

        where: { Doctor_ID: id }
    });
    const schedulesToCreate = Array.isArray(updates.Schedule)
    ? updates.Schedule
    : JSON.parse(updates.Schedule || '[]');

    if (schedulesToCreate.length > 0) {
        await Promise.all(
            schedulesToCreate.map(schedule =>
                Schedule.create({
                    Doctor_ID: id,
                    Day_of_week: schedule.day_of_week,
                    Start_time: schedule.start_time,
                    End_time: schedule.end_time,
                    Interval: schedule.interval
                })
            )
        );
    }

    const updatedDoctor = await Doctor.findByPk(id, {
        include: {
            model: Schedule,
            as: 'Schedules'
        }
    });

    return new DoctorDto(updatedDoctor);
} catch (error) {
    if (updates.Photo && updates.Photo.includes('cloudinary'))
    {
        const                publicId                =
updates.Photo.split('/').pop().split('.')[0];
        await
cloudinary.uploader.destroy(`doctors/${publicId}`);
    }
    throw error;
}

deleteDoctor = async (id) => {
    const doctor = await Doctor.findByPk(id);
    if (!doctor) throw new Error('Doctor not found');
    await User.destroy({ where: { ID: doctor.User_ID } });
    await Doctor.destroy({ where: { ID: id } });

    return { success: true };
}

getAllDoctors = async () => {
    const doctors = await Doctor.findAll({
        include: [
            {
                model: Schedule,
                as: 'Schedules'
            }
        ]
    });

```

```

        },
        {
            model: Specialization,
            as: 'Specialization'
        }
    ]
    });
    return doctors.map(doctor => new DoctorDto(doctor));
}

getDoctorById = async (id) => {
    const doctor = await Doctor.findByPk(id, {
        include: [
            {
                model: Schedule,
                as: 'Schedules'
            },
            {
                model: Specialization,
                as: 'Specialization'
            }
        ]
    });
    if (!doctor) throw new Error('Doctor not found');
    return new DoctorDto(doctor);
}

getDoctorByUserId = async (userId) => {
    const doctor = await Doctor.findOne({
        where: { User_ID: userId },
        include: [
            {
                model: Schedule,
                as: 'Schedules'
            },
            {
                model: Specialization,
                as: 'Specialization'
            }
        ]
    });
    if (!doctor) {
        throw new Error('Doctor not found');
    }
    return new DoctorDto(doctor);
};

getDoctorAppointments = async (doctorId) => {
    const appointments = await Appointment.findAll({
        where: { Doctor_ID: doctorId },
        include: [{
            model: Patient,
            as: 'Patient',
        }]
    });
    console.log(doctorId)

```

```

    console.log(appointments)
    return {
      appointments,
      patients: appointments.map(a => new PatientDto(a.Patient))
    };
  };
};

updateDoctorSchedule = async (doctorId, schedules) => {
  const doctor = await Doctor.findByPk(doctorId);
  if (!doctor) {
    throw ApiError.BadRequest('Врач не найден');
  }
  await Schedule.destroy({
    where: { Doctor_ID: doctorId }
  });
  const createdSchedules = await Promise.all(
    schedules.map(schedule =>
      Schedule.create({
        Doctor_ID: doctorId,
        Day_of_week: schedule.day_of_week,
        Start_time: schedule.start_time,
        End_time: schedule.end_time,
        Interval: schedule.interval
      })
    )
  );
  return createdSchedules;
}

updateDoctorInfo = async (doctorId, updates) => {
  const doctor = await Doctor.findByPk(doctorId);
  if (!doctor) {
    throw new Error('Doctor not found');
  }

  await doctor.update(updates);
  return new DoctorDto(doctor);
};

// Категории
async getSpecializations() {
  const specializations = await Specialization.findAll({
    attributes: ['ID', 'Name']
  });
  return specializations.map(specialization => new
SpecializationDto(specialization));
}

async addSpecialization(name) {
  if (!name || typeof name !== 'string') {
    throw Error('Некорректное название специализации');
  }
  const existingSpecialization = await Specialization.findOne({
    where: { Name: name }
  });
  if (existingSpecialization) {
    throw Error('Специализация с таким названием уже существует');
  }
}

```



```

    }
    const newSpecialization = await Specialization.create({
      Name: name.trim()
    });
    return new SpecializationDto(newSpecialization);
  })
}
module.exports = DoctorServices;
const {Appointment, Doctor, Schedule} = require('../config/database');
let DoctorServices = require('../services/doctorService');
const AppointmentDto = require('../dtos/appointmentDto');
const { Op } = require('sequelize');
const doctorServices = new DoctorServices();
class AppointmentService {
  async createAppointment(date, token) {
    try {
      const { patientId, doctorId, appointmentData } = date;
      const newAppointmentTime = new Date(appointmentData);
      const doctor = await doctorServices.getDoctorById(doctorId);
      if (!doctor) throw new Error('Doctor not found');
      const dayNames = ['Воскресенье', 'Понедельник', 'Вторник', 'Среда',
        'Четверг', 'Пятница', 'Суббота'];
      const appointmentDayName = dayNames[newAppointmentTime.getDay()];
      const doctorSchedule = doctor.schedule.find(s => s.day_of_week ===
appointmentDayName);
      if (!doctorSchedule) {
        throw new Error('Врач не принимает в выбранный день');
      }
      const [intervalHours, intervalMinutes] =
doctorSchedule.interval.split(':').map(Number);
      const APPOINTMENT_DURATION = intervalHours * 60 + intervalMinutes;
      const appointmentEnd = new Date(newAppointmentTime.getTime() +
APPOINTMENT_DURATION * 60000);
      const patientAppointments = await Appointment.findAll({
        where: {
          Patient_ID: patientId,
          isComplited: false,
          isDenied: false,
          Appointment_Date: {
            [Op.gte]: new Date(new Date().setHours(0, 0, 0, 0))
          }
        },
        include: [{
          model: Doctor,
          as: 'Doctor',
          include: [{
            model: Schedule,
            as: 'Schedules'
          }]
        }]
      });
    } catch (err) {
      console.log(err);
    }
    for (const existing of patientAppointments) {
      const existingDayName =
Date(existing.Appointment_Date).getDay();
      const dayNames[newAppointmentTime.getDay()] =

```

```

    const existingSchedule = existing.Doctor.Schedules.find(s =>
s.Day_of_week === existingDayName);
    if (existingSchedule) {
        const [existingHours, existingMinutes] =
existingSchedule.Interval.split(':').map(Number);
        const existingDuration = existingHours * 60 + existingMinutes;
        const existingEnd = new Date(new
Date(existing.Appointment_Date).getTime() + existingDuration * 60000);
        const isConflict = (
            (newAppointmentTime >= new Date(existing.Appointment_Date) &&
newAppointmentTime < existingEnd) ||
            (appointmentEnd > new Date(existing.Appointment_Date) &&
appointmentEnd <= existingEnd) ||
            (newAppointmentTime <= new Date(existing.Appointment_Date) &&
appointmentEnd >= existingEnd)
        );
        if (isConflict) {
            const conflictTime = new
Date(existing.Appointment_Date).toLocaleString();
            throw new Error(`У вас уже есть запись на ${conflictTime},
которая пересекается с выбранным временем`);
        }
    }
}
const appointment = await Appointment.create({
    Patient_ID: patientId,
    Doctor_ID: doctorId,
    Appointment_Date: newAppointmentTime,
    isComplited: false,
    isDenied: false
});
return new AppointmentDto(appointment);
} catch (error) {
    console.error('Error in createAppointment:', error);
    throw error;
}
}
async cancelAppointment(appointmentId, token) {
    try {
        const appointment = await Appointment.findByPk(appointmentId);

        if (!appointment) {
            throw new Error('Запись не найдена');
        }
        appointment.isDenied = true;
        await appointment.save();
        return new AppointmentDto(appointment);
    } catch (error) {
        console.error('Error in cancelAppointment:', error);
        throw error;
    }
}
}
async getPatientAppointments(patientId, token) {

```

```

    try {
      const appointments = await Appointment.findAll({
        where: { Patient_ID: patientId},
        order: [['Appointment_Date', 'ASC']]
      });
      return appointments.map(app => new AppointmentDto(app));
    } catch (error) {
      console.error('Error in getPatientAppointments:', error);
      throw error;
    }
  }
  async getDoctorAppointments(doctorId) {
    try {
      const appointments = await Appointment.findAll({
        where: { Doctor_ID: doctorId }
      });
      return appointments.map(app => new AppointmentDto(app));
    } catch (error) {
      console.error('Error in getDoctorAppointments:', error);
      throw error;
    }
  }
  async getAppointments() {
    try {
      const appointments = await Appointment.findAll();
      return appointments.map(app => new AppointmentDto(app));
    } catch (error) {
      console.error('Error in getPatientAppointments:', error);
      throw error;
    }
  }
}
module.exports = AppointmentService;

```