МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Факультет №8
«Компьютерные науки и прикладная математика»
Кафедра №806
«Вычислительная математика и программирование»
Направление подготовки
02.03.02 «Фундаментальная информатика и информационные технологии»

Курсовой проект
по дисциплине «Базы данных»
на тему: «Проектирование схемы БД, разработка API и клиентского приложения для информационной модели “Медицинский центр”»

Студент: Сафин Тимур Айратович
Группа: М8О-311Б-22
Преподаватель: Крылов Сергей Сергеевич
Оценка: _____
Дата: _____

Москва, 2024

Содержание

Введение

В современных условиях автоматизации бизнеса и цифровизации различных сфер деятельности особое значение приобретают системы управления базами данных (СУБД) и клиентские приложения для доступа к ним. Настоящий курсовой проект посвящен разработке информационной системы для медицинского центра, основными функциями которой являются запись пациентов к врачам и оформление заявок на медицинские услуги, предоставляемые учреждением.

Цель проекта – создать комплексное программное решение, обеспечивающее эффективное управление данными медицинского центра. В качестве системы управления базой данных (СУБД) выбрана PostgreSQL, обеспечивающая надежное хранение и обработку данных. Клиентская часть приложения разработана на языке программирования Java с использованием фреймворка Spring, что обеспечивает структурированную и масштабируемую архитектуру серверного приложения. Для реализации пользовательского интерфейса используется библиотека React, позволяющая создать удобное и интуитивно понятное веб-приложение для взаимодействия с пользователем.

В процессе разработки выполнены следующие задачи:

- Проектирование и создание структуры базы данных, состоящей из 7 таблиц, нормализованных до 3NF;
- Разработка клиентского приложения для пользователей различных ролей с возможностью авторизации по логину и паролю;
- Реализация логики обработки данных на стороне сервера;
- Тестирование и отладка приложения для обеспечения корректной обработки ошибок.

Настоящий проект демонстрирует комплексный подход к разработке программного продукта: начиная от проектирования базы данных и заканчивая реализацией клиентского интерфейса. Полученная система может использоваться для оптимизации процессов медицинского центра, упрощения записи на приём к врачам и улучшения качества обслуживания пациентов.

Теоретическая часть

Введение в реляционные базы данных

Технологии баз данных существовали не всегда. Однако и до их внедрения в практику люди также собирали и обрабатывали данные. Одним из способов хранения данных были так называемые плоские файлы (flat files), которые имели очень простую структуру: данные хранились в виде записей, разделенных на поля фиксированной длины. В реальной жизни между элементами данных зачастую возникают сложные связи, которые необходимо перенести и в электронную базу данных. При использовании плоских файлов эти связи организовать сложно, а еще сложнее поддерживать их при изменениях и удалениях отдельных элементов данных.

В ходе эволюции теорий и идей была разработана реляционная модель данных, которая сейчас и является доминирующей. Поэтому в настоящее время преобладают базы данных реляционного типа [1]. Их характерной чертой является тот факт, что данные воспринимаются пользователем как таблицы. В распоряжении пользователя имеются операторы для выборки данных из таблиц, а также для вставки новых данных, обновления и удаления имеющихся данных.

Одним из достоинств реляционной базы данных является ее способность поддерживать связи между элементами данных, избавляя программиста от необходимости заниматься этой рутинной и очень трудоемкой работой. В те времена, когда технологии и SQL реляционных баз данных еще не получили широкого распространения, программистам приходилось на процедурных языках вручную реализовывать такие операции, которые сейчас называются каскадным обновлением внешних ключей или каскадным удалением записей из подчиненных таблиц (файлов).

Работая с реляционными базами данных, программист избавлен от программирования на «атомарном» уровне, потому что современные языки для «общения» с этими базами данных являются декларативными. Это означает, что для получения результата достаточно лишь указать, что нужно получить, но не требуется предписывать способ получения результата, т. е. как его получить.

Система баз данных — это компьютеризированная система, предназначенная для хранения, переработки и выдачи информации по запросу пользователей. Такая система включает в себя программное и аппаратное обеспечение, сами данные, а также пользователей. Современные системы баз данных являются, как правило, многопользовательскими. В таких системах одновременный доступ к базе данных могут получить сразу несколько пользователей. Основным программным обеспечением является система управления базами данных.

Кроме СУБД в систему баз данных могут входить утилиты, средства для разработки приложений (программ), средства проектирования базы данных, генераторы отчетов и др. Пользователи систем с базами данных подразделяются на

ряд категорий. Первая категория — это прикладные программисты. Вторая категория — это конечные пользователи, ради которых и выполняется вся работа. Они могут получить доступ к базе данных, используя прикладные программы или универсальные приложения, которые входят в программное обеспечение самой СУБД.

В большинстве СУБД есть так называемый процессор языка запросов, который позволяет пользователю вводить команды языка высокого уровня (например, языка SQL). Третья категория пользователей — это администраторы базы данных. В их обязанности входят: создание базы данных, выбор оптимальных режимов доступа к ней, разграничение полномочий различных пользователей на доступ к той или иной информации в базе данных, выполнение резервного копирования базы данных и т. д. Систему баз данных можно разделить на два главных компонента: сервер и набор клиентов (или внешних интерфейсов). Сервер — это и есть СУБД. Клиентами являются различные приложения, написанные прикладными программистами, или встроенные приложения, поставляемые вместе с СУБД. Один сервер может обслуживать много клиентов.

В реляционных базах данных пользователь воспринимает данные в виде таблиц. Поэтому термину «файл» соответствует термин «таблица», вместо термина «запись» используется термин «строка», а вместо термина «поле» — термин «столбец» (или «колонка»). Таким образом, таблицы состоят из строк и столбцов, на пересечении которых должны находиться «атомарные» значения, которые нельзя разбить на более мелкие элементы без потери смысла. В формальной теории реляционных баз данных эти таблицы называют отношениями (relations) — поэтому и базы данных называются реляционными. Отношение — это математический термин. При определении свойств таких отношений используется теория множеств. В терминах данной теории строки таблицы будут называться кортежами (tuples), а колонки — атрибутами. Отношение имеет заголовок, который состоит из атрибутов, и тело, состоящее из кортежей. Количество атрибутов называется степенью отношения, а количество кортежей — кардинальным числом. Кроме теории множеств, одним из оснований реляционной теории является такой раздел математической логики, как исчисление предикатов. Таким образом, в теории и практике баз данных существует три группы терминов. Иногда термины из разных групп используют в качестве синонимов, например, запись и строка. Как мы уже сказали выше, в реляционных базах данных пользователь воспринимает данные в виде таблиц.

Для идентификации строк в таблицах и для связи таблиц между собой используются так называемые ключи. Потенциальный ключ — это комбинация атрибутов таблицы, позволяющая уникальным образом идентифицировать строки в ней. Ключ может состоять только лишь из одного атрибута таблицы. Например, в таблице «Студенты» таким идентификатором может быть атрибут «Номер зачетной

книжки». В качестве потенциального ключа данной таблицы могут также служить два ее атрибута, взятые вместе: «Серия документа, удостоверяющего личность» и «Номер документа, удостоверяющего личность». Ни один из них в отдельности не может использоваться в качестве уникального идентификатора. В таком случае ключ будет составным. При этом важным является то, что потенциальный ключ должен быть неизбыточным, т.е. никакое подмножество атрибутов, входящих в него, не должно обладать свойством уникальности. Потенциальный ключ, включающий два упомянутых атрибута, является неизбыточным.

Ключи нужны для адресации на уровне строк (записей). При наличии в таблице более одного потенциального ключа один из них выбирается в качестве так называемого первичного ключа, а остальные будут являться альтернативными ключами. Таблица, содержащая внешний ключ, называется ссылающейся таблицей (referencing table). Таблица, содержащая соответствующий потенциальный ключ, называется ссылочной (целевой) таблицей (referenced table). В таких случаях говорят, что внешний ключ ссылается на потенциальный ключ в ссылочной таблице. Внешний ключ может быть составным, т.е. может включать более одного атрибута. Внешний ключ не обязан быть уникальным. Проблема обеспечения того, чтобы база данных не содержала неверных значений внешних ключей, известна как проблема ссылочной целостности. Ограничение, согласно которому значения внешних ключей должны соответствовать значениям потенциальных ключей, называется ограничением ссылочной целостности (ссылочным ограничением).

Обеспечением выполнения ограничений ссылочной целостности занимается СУБД, а от разработчика требуется лишь указать атрибуты, служащие в качестве внешних ключей. При проектировании баз данных часто предусматривается, что при удалении строки из ссылочной таблицы соответствующие строки из ссылающейся таблицы должны быть также удалены, а при изменении значения столбца, на который ссылается внешний ключ, должны быть изменены значения внешнего ключа в ссылающейся таблице. Этот подход называется каскадным удалением (обновлением). Иногда применяются и другие подходы. Например, вместо удаления строк из ссылающейся таблицы в этих строках просто заменяют значения атрибутов, входящих во внешний ключ, так называемыми NULL-значениями. Это специальные значения, означающие «ничто» или отсутствие значения, они не совпадают со значением «нуль» или «пустая строка». NULL-значение применяется в базах данных и в качестве значения по умолчанию, когда пользователь не ввел никакого конкретного значения. Первичные ключи не могут содержать NULL-значений.

Язык SQL

Язык SQL — это непроцедурный язык, который является стандартным средством работы с данными во всех реляционных СУБД. Операторы (команды), написанные на этом языке, лишь указывают СУБД, какой результат должен быть

получен, но не описывают процедуру получения этого результата. СУБД сама определяет способ выполнения команды пользователя. В языке SQL традиционно выделяются группа операторов определения данных (Data Definition Language — DDL), группа операторов манипулирования данными (Data Manipulation Language — DML) и группа операторов, управляющих привилегиями доступа к объектам базы данных (Data Control Language — DCL).

К операторам языка определения данных (DDL) относятся команды для создания, изменения и удаления таблиц, представлений и других объектов базы данных.

К операторам языка манипулирования данными (DML) относятся команды для выборки строк из таблиц, вставки строк в таблицы, обновления и удаления строк.

PostgreSQL

PostgreSQL — это мощная объектно-реляционная система управления базами данных с открытым исходным кодом, которая поддерживает широкий набор стандартов SQL и имеет расширенные возможности для работы с различными типами данных. Она отличается высокой производительностью, надежностью, масштабируемостью и гибкостью, что делает её одной из наиболее популярных СУБД среди разработчиков, работающих с большими объемами данных и сложными запросами.

PostgreSQL поддерживает весь стандарт SQL, включая все команды из групп DDL и DML, а также включает дополнительные возможности для работы с расширенными типами данных, такими как массивы, JSON, XML и другие, что позволяет разрабатывать более гибкие и сложные приложения. Помимо стандартных типов данных, PostgreSQL поддерживает создание пользовательских типов данных и функций, что дает возможность пользователям адаптировать СУБД под специфические задачи.

Одной из важных особенностей PostgreSQL является поддержка транзакций с использованием принципов ACID (атомарность, согласованность, изолированность, долговечность). Это означает, что операции с данными, выполненные в рамках одной транзакции, либо будут выполнены полностью, либо отменены в случае сбоя, обеспечивая целостность данных. PostgreSQL также поддерживает механизмы блокировок на уровне строк и таблиц, что гарантирует высокую степень изоляции транзакций при параллельном доступе.

Кроме того, PostgreSQL предоставляет механизмы для работы с внешними ключами (ссылочной целостностью), триггерами, индексами и представлениями, что значительно улучшает эффективность работы с данными. Важным моментом является поддержка расширений, которые позволяют добавлять функциональность, недоступную в стандартной конфигурации, таких как полнотекстовый поиск, географические информационные системы (PostGIS) и другие.

СУБД PostgreSQL активно используется для создания как небольших, так и крупных корпоративных приложений благодаря своей открытости, постоянному обновлению и поддержке сообщества. Она поддерживает работу на различных платформах, включая Linux, Windows, macOS и другие, что позволяет применять её в самых различных средах.

Основные возможности PostgreSQL включают:

- Поддержка стандартных SQL команд: PostgreSQL поддерживает как базовые команды DDL и DML, так и сложные запросы с использованием оконных функций, подзапросов, агрегатных функций, операторы JOIN и другие продвинутые возможности.
- Механизмы безопасности: В PostgreSQL реализованы механизмы управления правами доступа к данным с помощью ролей, что позволяет эффективно управлять привилегиями пользователей.
- Поддержка репликации и масштабирования: PostgreSQL предлагает возможности для настройки репликации данных и распределения нагрузки на несколько серверов, что важно для масштабируемых решений.
- Расширенная поддержка индексов: СУБД позволяет создавать индексы на основе различных типов данных и методов их хранения, что способствует улучшению производительности запросов.

Описание проектирования сущностей БД

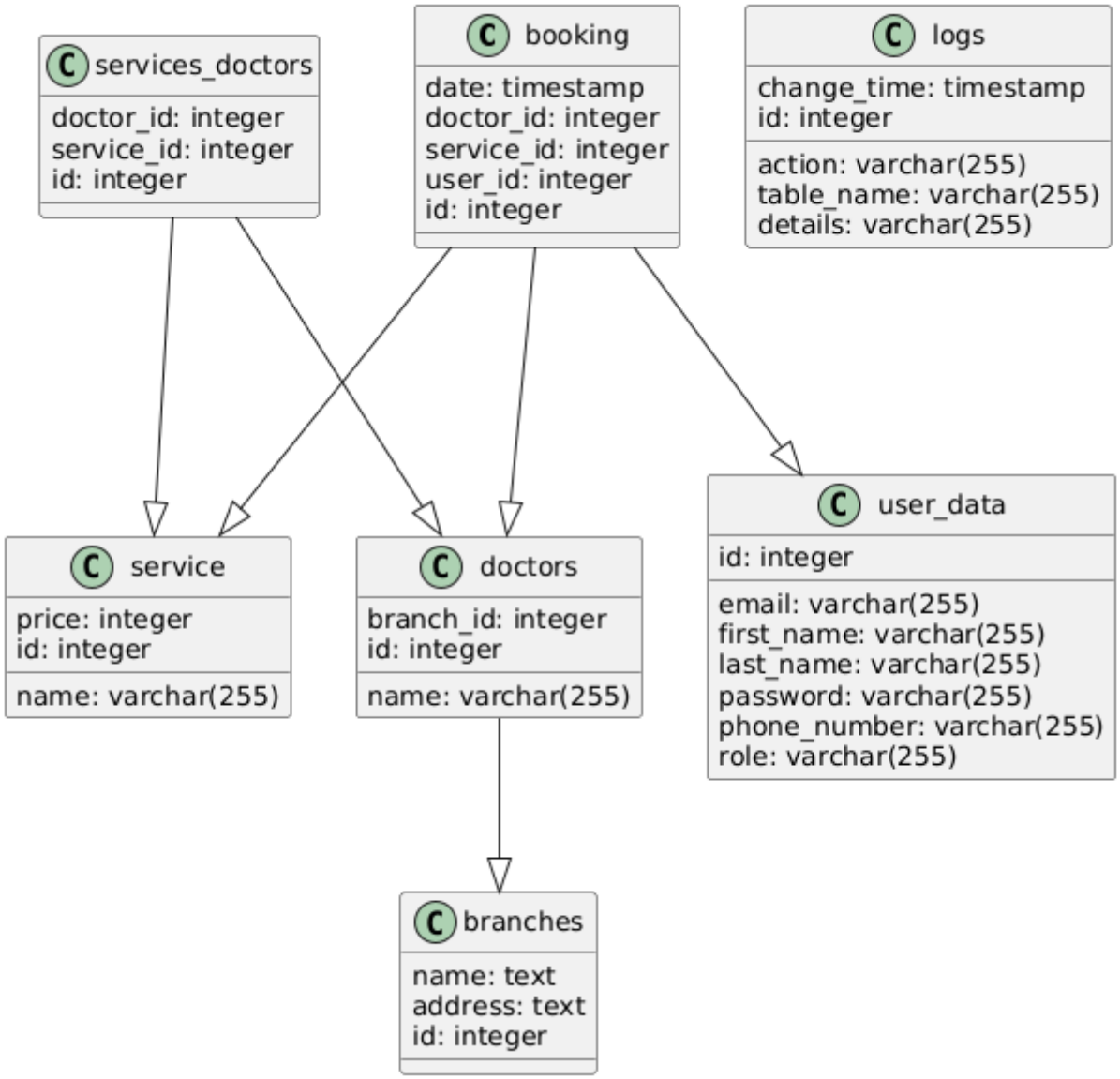


Рисунок 1. Диаграмма базы данных

1. Таблица *booking* (Бронирование):
   - *id* — уникальный идентификатор записи.
   - *date* — дата и время записи.
   - *doctor_id* — ссылка на таблицу врачей (связь с врачом, который назначен на приём).
   - *service_id* — ссылка на таблицу услуг (связь с услугой, которая предоставляется).
   - *user_id* — ссылка на таблицу пользователей (связь с клиентом, который записан на приём).
2. Таблица *branches* (Филиалы):
   - *id* — уникальный идентификатор филиала.
   - *name* — название филиала.
   - *address* — адрес филиала.

3. Таблица *doctors* (Врачи):
   - *id* — уникальный идентификатор врача.
   - *name* — имя врача.
   - *branch_id* — ссылка на филиал, где работает врач.
4. Таблица *logs* (Логирование):
   - *id* — уникальный идентификатор записи в логе.
   - *action* — описание действия (например, создание, обновление, удаление).
   - *table_name* — название таблицы, в которой произошло изменение.
   - *change_time* — время изменения.
   - *details* — дополнительные детали изменения.
5. Таблица *service* (Услуги):
   - *id* — уникальный идентификатор услуги.
   - *name* — название услуги.
   - *price* — стоимость услуги.
6. Таблица *user_data* (Пользовательские данные):
   - *id* — уникальный идентификатор пользователя.
   - *email* — электронная почта пользователя.
   - *first_name* — имя пользователя.
   - *last_name* — фамилия пользователя.
   - *password* — пароль пользователя (для аутентификации).
   - *phone_number* — номер телефона пользователя.
   - *role* — роль пользователя (admin или user).
7. Таблица services_doctors (Связь врачей с услугами):
   - *id* — уникальный идентификатор записи.
   - *doctor_id* — ссылка на врача, который предоставляет услугу.
   - *service_id* — ссылка на услугу, предоставляемую врачом.

   Связи:

1. *booking* связана с *doctors*, *service*, и *user_data* через внешние ключи. Это означает, что запись на приём включает данные о враче, услуге и пользователе.
2. *doctors* связано с *branches*, что позволяет определить, в каком филиале работает врач.
3. *services_doctors* служит связующей таблицей для отношения многие ко многим между врачами и услугами (врачи могут предоставлять несколько услуг, а услуги могут быть предоставлены несколькими врачами).

Эта схема представляет основы для построения базы данных медицинского центра, где можно будет легко организовать запись пациентов, учёт услуг и врачей, а также логи изменений системы.

Данная схема представлена в виде третьей нормальной формы (3NF). Это означает, что:

1. Схема находится в *первой нормальной форме* (*1NF*): каждое поле содержит только атомарные значения, то есть значения, которые не могут быть разделены на более мелкие части.
2. Схема находится во *второй нормальной форме* (*2NF*): все не ключевые атрибуты функционально зависят от всего первичного ключа, а не от его части.
3. Схема находится в *третьей нормальной форме* (*3NF*): все не ключевые атрибуты не имеют транзитивных зависимостей от других не ключевых атрибутов. То есть, не существует ситуации, когда один не ключевой атрибут зависит от другого не ключевого атрибута через первичный ключ.

Таким образом, данная схема обеспечивает минимизацию избыточности данных и улучшает целостность базы данных, что является основным преимуществом использования третьей нормальной формы.

Практическая часть

Применённые технологии

Технологический стек:

1. Backend
   - Java: Основной язык программирования для разработки серверной части приложения.
   - Spring Framework: Использован для создания RESTful API, обработки запросов и управления зависимостями через Spring Boot.
   - PostgreSQL: Реляционная база данных, применяемая для хранения и управления данными.
2. Frontend
   - JavaScript: Язык программирования, используемый для создания динамичного интерфейса пользователя.
   - React: Библиотека для разработки компонентного пользовательского интерфейса и управления состоянием приложения.

Ход выполнения:

1. Проектирование
   - Разработка архитектуры приложения (клиент-сервер).
   - Определение структуры базы данных и создание ER-диаграммы.
2. Реализация Backend
   - Разработка REST API на основе Spring Boot.
   - Настройка подключения к PostgreSQL и реализация CRUD операций.
3. Реализация Frontend
   - Разработка пользовательского интерфейса на основе React.
   - Реализация взаимодействия с API через axios для работы с данными.

Создание и инициализация базы данных

Помимо таблиц, описанных ранее, в базе данных так же были реализованы:

1. Триггеров и функций
   - Функция логирования *log_action*(): реализует запись действий (INSERT, UPDATE, DELETE) в таблицу logs.
   - Триггеры для автоматического логирования: применены к основным таблицам (*user_data*, *branches*, *doctors*, *service*, *booking*, *services_doctors*).
2. Виды и дополнительные возможности
   - Представление *today_bookings*: отображает записи на текущий день.
   - Ограничения
     1. Проверка формата email (*email_regex*).
     2. Проверка формата номера телефона (*phone_regex*).

3. Начальная инициализация данных
   - Услуги (*service*).
   - Врачи (*doctors*).
   - Связь врачей и услуг (*services_doctors*).
   - Филиалы (*branches*).

   Скрипт создания таблиц:

Листинг 1. Создание таблиц

```
create table if not exists user_data
(
    id           integer generated always as identity primary key,
    email        varchar(255) not null unique,
    first_name   varchar(255) not null,
    last_name    varchar(255) not null,
    password     varchar(255) not null,
    phone_number varchar(255) not null,
    role         varchar(255) check (role in ('ADMIN', 'USER'))
);

create table if not exists branches
(
    id      integer generated always as identity primary key,
    name    text not null unique,
    address text not null unique
);

create table if not exists doctors
(
    id        integer generated always as identity primary key,
    name      varchar(255) not null,
    branch_id integer references branches (id) on delete no action
);

create table if not exists service
(
    id    integer generated always as identity primary key,
    name  varchar(255) not null,
    price integer default 0
);

CREATE TABLE IF NOT EXISTS logs
(
    id          integer generated always as identity primary key,
    action      varchar(255) check (action in ('INSERT', 'DELETE',
'UPDATE')),
    table_name  varchar(255),
    change_time timestamp default now(),
    details     varchar(255)
);

create table if not exists booking
```

```
(
    id         integer generated always as identity primary key,
    date       timestamp,
    doctor_id  integer references doctors (id) on delete cascade,
    service_id integer references service (id) on delete cascade,
    user_id    integer references user_data (id) on delete cascade
);

create table if not exists services_doctors
(
    id         integer generated always as identity primary key,
    service_id integer references service (id) on delete cascade,
    doctor_id  integer references doctors (id) on delete cascade
);

alter table user_data add constraint email_regex check ( email ~* '([a-
zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)' );

alter table user_data add constraint phone_regex check (phone_number ~*
'^((8|\+7)[\- ]?)?(\(?\d{3}\)?[\- ]?)?[\d\- ]{7,10}$');
```

Создание процедур и триггеров:

Листинг 2. Создание процедур и триггеров

```
create or replace function log_action() returns trigger as
$log_act$
begin
    insert into logs (action, table_name, change_time, details)
    select TG_OP,
           tg_table_name,
           now(),
           case
               when TG_OP ILIKE 'INSERT' then 'Added' || NEW::TEXT
               when TG_OP ILIKE 'UPDATE' then 'Updated' || NEW::TEXT
               when TG_OP ILIKE 'DELETE' then 'Deleted' || OLD::TEXT
               end;
    return new;
end ;
$log_act$ LANGUAGE plpgsql;

create or replace trigger log_act
    AFTER INSERT OR UPDATE OR DELETE
    on booking
    for each row
execute procedure log_action();

create or replace trigger log_act
    AFTER INSERT OR UPDATE OR DELETE
    on doctors
    for each row
execute procedure log_action();

create or replace trigger log_act
```

```
    AFTER INSERT OR UPDATE OR DELETE
    on service
    for each row
execute procedure log_action();

create or replace trigger log_act
    AFTER INSERT OR UPDATE OR DELETE
    on services_doctors
    for each row
execute procedure log_action();

create or replace trigger log_act
    AFTER INSERT OR UPDATE OR DELETE
    on user_data
    for each row
execute procedure log_action();

create or replace trigger log_act
    AFTER INSERT OR UPDATE OR DELETE
    on branches
    for each row
execute procedure log_action();
```

Инициализация базы данных:

Листинг 3. Создание нового пользователя

```
insert into service (price, name)
VALUES (1000, 'Сбор крови'),
       (4000, 'КТ'),
       (6000, 'МРТ'),
       (700, 'Приём стоматолога'),
       (1000, 'Приём хирурга'),
       (1000, 'Приём дерматолога'),
       (1000, 'Приём офтальмолога'),
       (1000, 'Приём терапевта');

insert into doctors (name)
values ('Михаил Петрович Малышев'),
       ('Богдан Виссарионович Сталин'),
       ('Михаил Петрович Малышев'),
       ('Мистер Бист');

insert into services_doctors (doctor_id, service_id)
VALUES (1, 1),
       (2, 2),
       (3, 3),
       (4, 4);

insert into branches (name, address)
values ('Филиал 1', 'Москва, Факультетский переулок, д. 10'),
       ('Филиал 2', 'Москва, Ленинградский проспект, д. 15к2');
```

Создание представления текущих записей:

Листинг 4. Создание представлений

```
create or replace view today_bookings as
select *
from booking b
where (date_trunc('day', b.date) = date_trunc('day', now()));
```

Эта настройка позволяет разграничить доступ к базе данных, чтобы пользователь имел только необходимые права для выполнения задач. Это улучшает безопасность и защищает данные от несанкционированного доступа или изменений.

Листинг 5. Создание нового пользователя

```
CREATE USER "user"
GRANT SELECT, UPDATE, INSERT, DELETE ON booking TO "user";
GRANT SELECT ON doctors TO "user";
GRANT SELECT ON service TO "user";
GRANT SELECT ON branches TO "user";
GRANT SELECT ON services_doctors TO "user";
GRANT INSERT ON logs TO "user";
GRANT SELECT, UPDATE, INSERT ON user_data TO "user";
```

Создание серверной части приложения

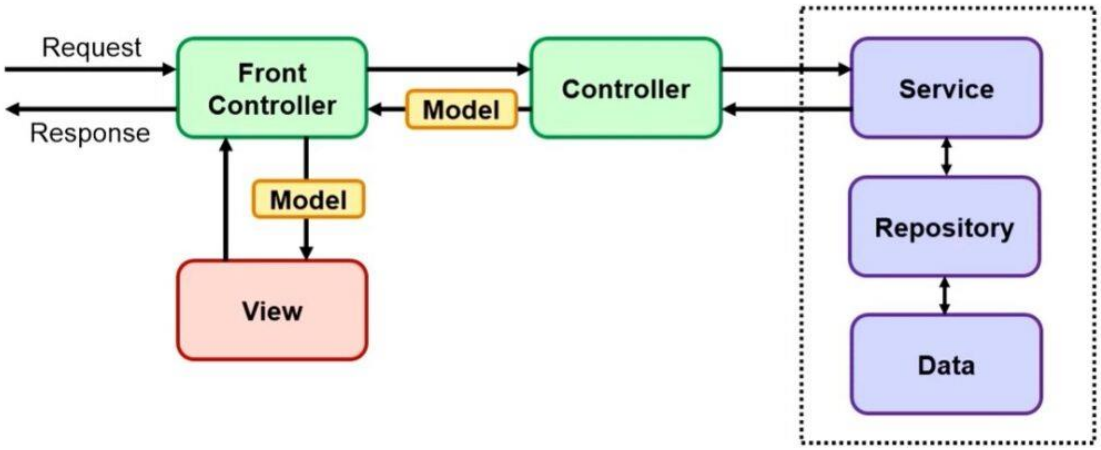


Рисунок 2. Шаблон проектирования MVC. Схема

Серверная часть организована по принципам MVC (Model-View-Controller) [3] и разделена на несколько основных компонентов:

1. Конфигурация (config)

   Содержит классы для настройки приложения:

   - ApplicationConfig — общие настройки приложения.
   - DataSourceConfig — настройка источника данных для подключения к базе.
   - JwtAuthenticationFilter — фильтр для обработки JWT-токенов и аутентификации запросов.
   - SecurityConfig — настройка безопасности приложения с использованием Spring Security и JWT.
   - WebConfig — настройка CORS и дополнительных веб-параметров.

2. Контроллеры (controller)
   Взаимодействуют с клиентской частью и обрабатывают HTTP-запросы:
   - AuthenticationController — отвечает за регистрацию и авторизацию пользователей.
   - AdminController — контроллер для управления данными с правами администратора.
   - BookingController — управляет записями на приём.
   - DoctorController — возвращает данные о врачах.
   - ServiceController — отвечает за обработку данных об услугах.
   - UserController — предоставляет функционал для работы с пользователями.
   - ExceptionController — обработка исключений в приложении.
3. Сущности (entity)
   Определяют структуру данных, используемых в базе данных. Благодаря Hibernate и JPA, они автоматически сопоставляются с таблицами базы данных:
   - Booking — сущность для записи на приём.
   - Branches — филиалы, в которых оказываются услуги.
   - Doctors — данные о врачах.
   - Logs — логирование действий в системе.
   - Role — роли пользователей (например, ADMIN или USER).
   - Service — описание предоставляемых услуг.
   - ServicesDoctors — связь между услугами и врачами.
   - User — данные о пользователях системы.
4. DTO (dto)
   - Для передачи данных между сервером и клиентом используются DTO-классы (Data Transfer Object).
5. Источник данных (data_source)
   - DataSourceContextHolder и DynamicRoutingDataSource — позволяют динамически управлять источниками данных.
6. Сервисы и репозитории
   - Сервисы содержат бизнес-логику приложения и обеспечивают связь между контроллерами и репозиториями.
   - Репозитории используют Spring Data JPA для абстракции работы с базой данных, позволяя выполнять запросы через интерфейсы.

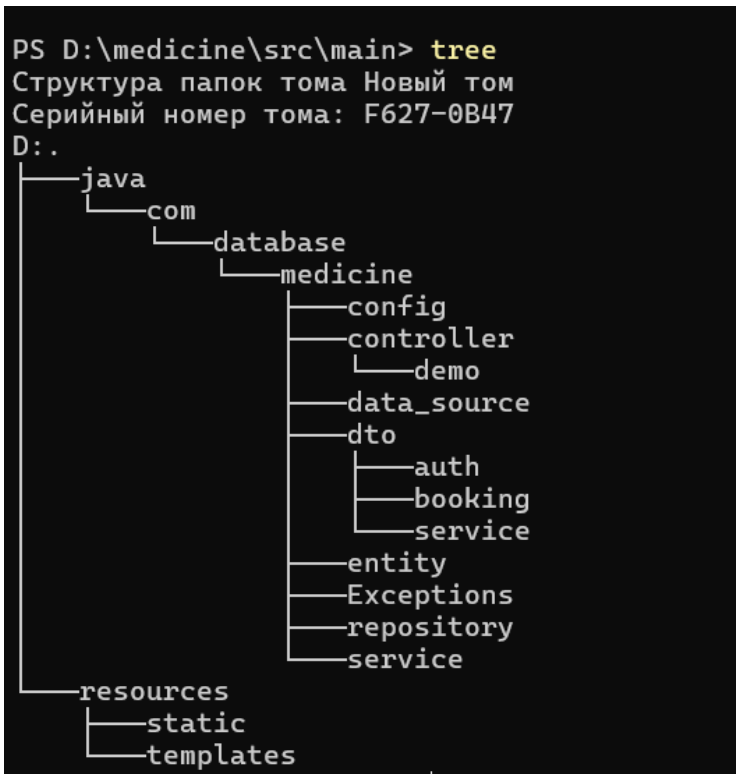


Рисунок 3. Структура серверной части приложения

Аутентификация с использованием JWT (JSON Web Token) реализована следующим образом:

1. Создание токена
   - При успешной аутентификации пользователь получает JWT-токен.
   - Токен содержит зашифрованную информацию о пользователе, включая его ID, роль и срок действия токена.
2. Фильтр JwtAuthenticationFilter
   - Каждый запрос от клиента проходит через этот фильтр.
   - Фильтр извлекает токен из заголовка Authorization и проверяет его валидность.
   - Если токен действителен, он передаёт информацию о пользователе в SecurityContext, чтобы Spring Security мог идентифицировать пользователя.
3. Настройка безопасности (SecurityConfig)
   - Используется Spring Security для защиты маршрутов и управления правами доступа.
   - Доступ к определённым эндпоинтам ограничен в зависимости от роли пользователя (например, ADMIN или USER).
4. Преимущества JWT
   - Токен хранится на клиенте и передаётся в каждом запросе, что исключает необходимость хранения сессий на сервере.
   - Позволяет безопасно идентифицировать пользователей и контролировать доступ к защищённым ресурсам.

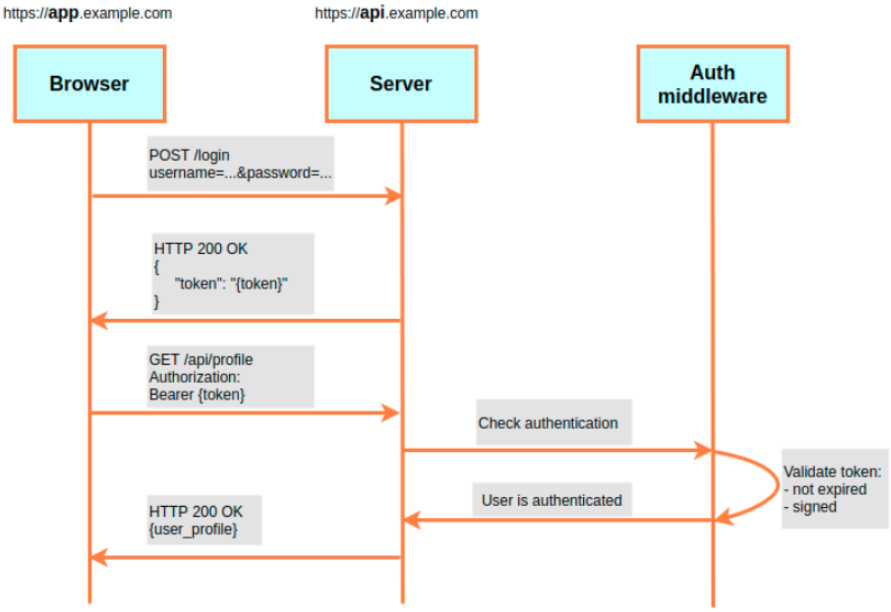


Рисунок 4. Авторизация с помощью JWT. Схема

Создание пользовательского интерфейса

Пользовательский интерфейс разработан на JavaScript с использованием библиотеки React. Приложение построено на компонентном подходе, организовано по модулям и включает в себя маршрутизацию, аутентификацию с JWT и взаимодействие с сервером через REST API. Использование таких технологий обеспечивает масштабируемость, поддержку и гибкость в разработке веб-приложения.

Приложение реализовано по следующим принципам:

1. Компонентный подход
   - Весь интерфейс разбит на переиспользуемые компоненты. Например:
     - AdminPanel.js — отдельный компонент для панели администратора.
     - NavHeader.js — компонент для навигации.
   - Такая структура обеспечивает гибкость и лёгкость поддержки приложения.
2. Маршрутизация
   - Использован React Router для навигации между страницами.
   - В router/PrivateRoute.js реализована защита маршрутов для авторизованных пользователей, что обеспечивает безопасность приложения.
3. Аутентификация
   - Аутентификация реализована с использованием JWT (JSON Web Token).
   - Файл isTokenExpired.js используется для проверки срока действия токена.
   - Кнопка выхода из аккаунта реализована в LogoutButton.js.
4. Работа с сервером
   - Для выполнения HTTP-запросов к серверу используется библиотека Axios.
   - Базовый клиент для API находится в axios_api/apiClient.js.
5. Стилизация
   - Стили для компонентов хранятся в отдельных CSS-файлах, например, NavHeader.css.
   - Приложение использует модульный подход к CSS, где стили привязаны к конкретным компонентам.
6. Хранилище данных
   - В проекте используется состояние компонентов (React State) и хуки (например, useState, useEffect) для управления данными на фронтенде.
7. Организация кода
   - Код структурирован по модулям (например, auth, admin, booking), что улучшает читаемость и позволяет легко расширять функционал.

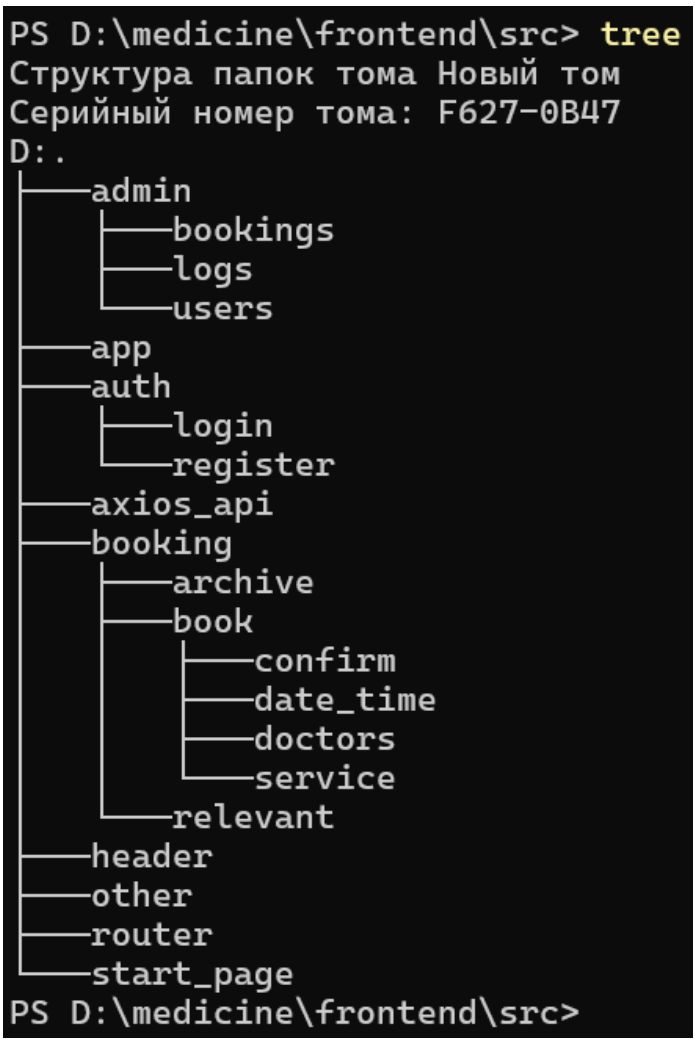


Рисунок 4. Схема React приложения

Инструкция по использованию

При попытке зайти на сайт медицинского сервера вас встретит окно авторизации. Введите Вашу электронную почту или пароль чтобы войти. Если Вы ещё не зарегистрированы на сайте, нажмите на кнопку «Регистрация» снизу кнопки «Логин».

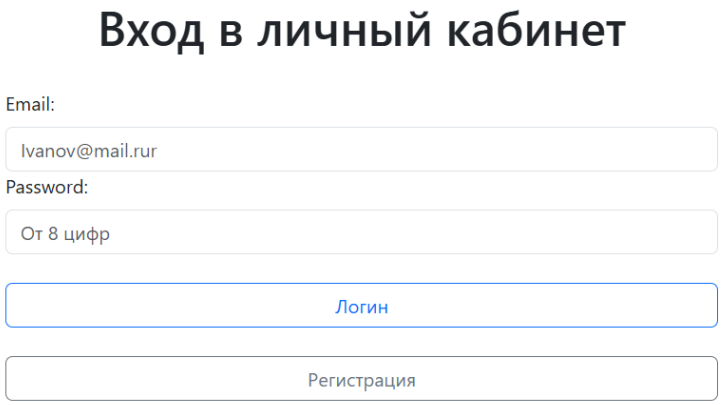


Рисунок 5. Вход в личный кабинет.

Для регистрации необходимо ввести Ваше имя, фамилию, email, номер телефона (российского формата) и пароль, после чего нажать на кнопку «Регистрация». В случае, если Вы перешли в окно регистрации случайно, либо же вспомнили данные от аккаунта – нажмите на кнопку «Логин» под кнопкой «Регистрация» чтобы перейти в окно авторизации.

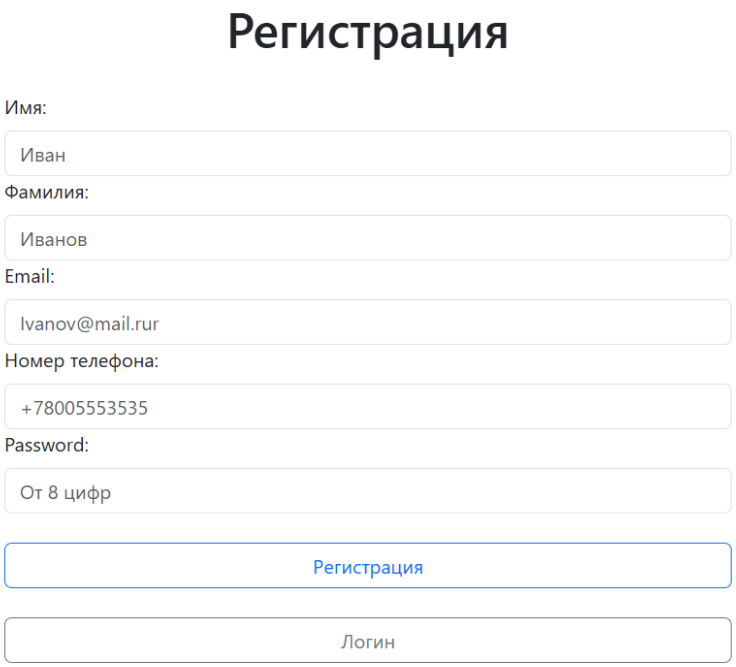
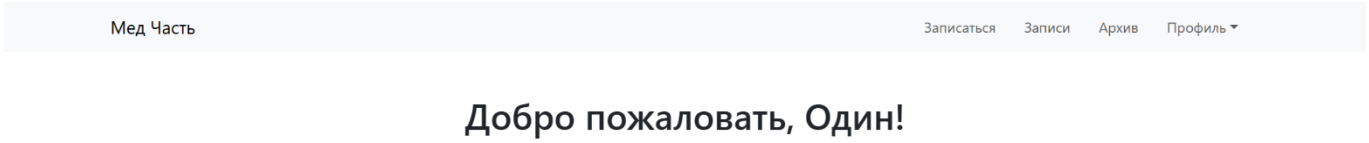


Рисунок 6. Регистрация

После входа в личный кабинет вы попадаете на начальную страницу, где Вас встретит надпись «Добро пожаловать, “*ваше имя*”», а также панель навигации в верху страницы:

1. Записаться – открывает окно записи.
2. Записи – открывает окно для просмотра активных записей.
3. Архив – открывает окно для просмотра прошедших записей.
4. Профиль – раскрывающееся меню с расширенными возможностями



Добро пожаловать, Один!

Рисунок 7. Начальная страница

При нажатии на кнопку «Записаться» открывается окно, где представлены все доступные услуги, на которые вы можете сделать запись. Выбираете на нужную Вам услугу и нажмите «Записаться» для выбора врача.

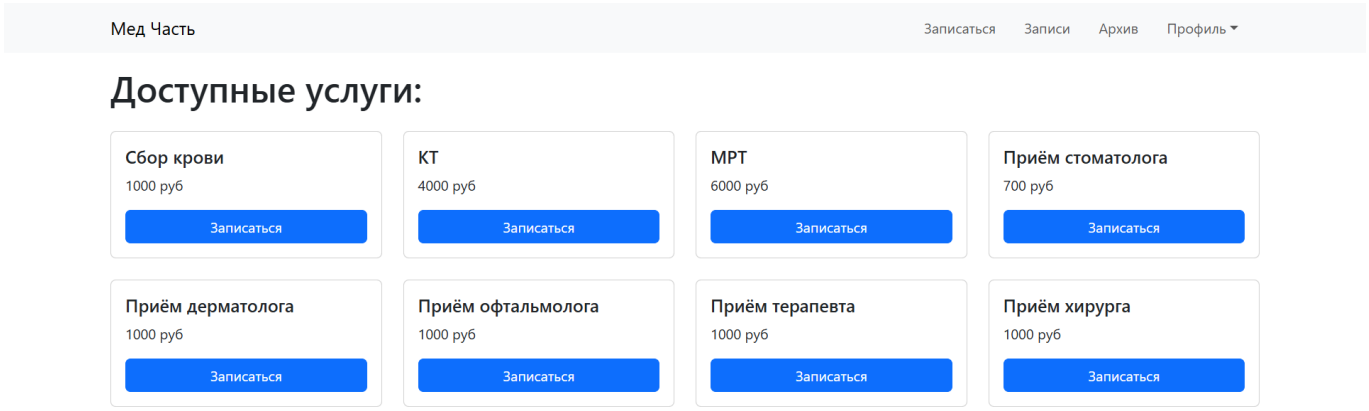


Рисунок 8. Страница записи. Выбор услуги

После выбора услуги, выберите подходящего Вам врача. Обратите внимание на филиал, где работает данный врач. Нажмите на кнопку «Записаться» чтобы выбрать дату и время записи.

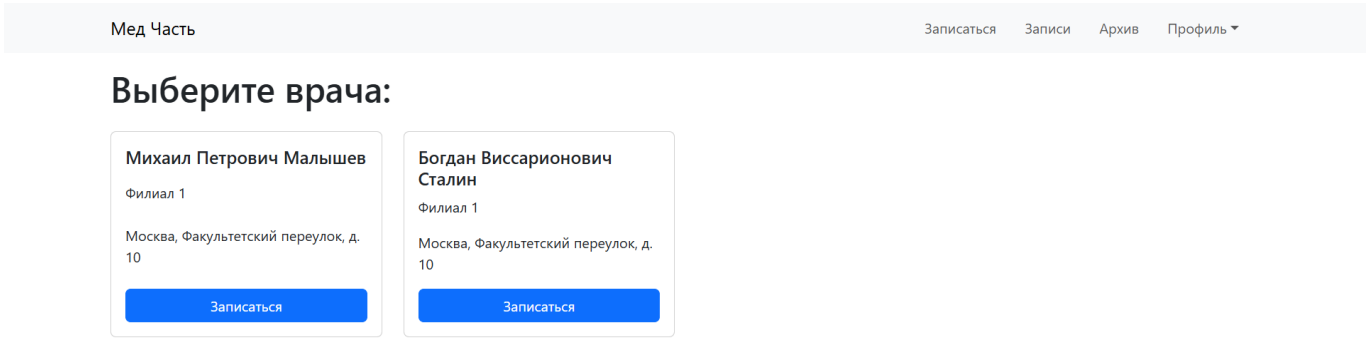


Рисунок 9. Страница записи. Выбор врача

Далее выберите день приёма, нажав на число в списке под надписью «Выберите дату:», после чего нажмите на удобное для вас время записи. Если кнопка со временем подсвечена серым – на данное время невозможно записаться, выберите другое время или дату.

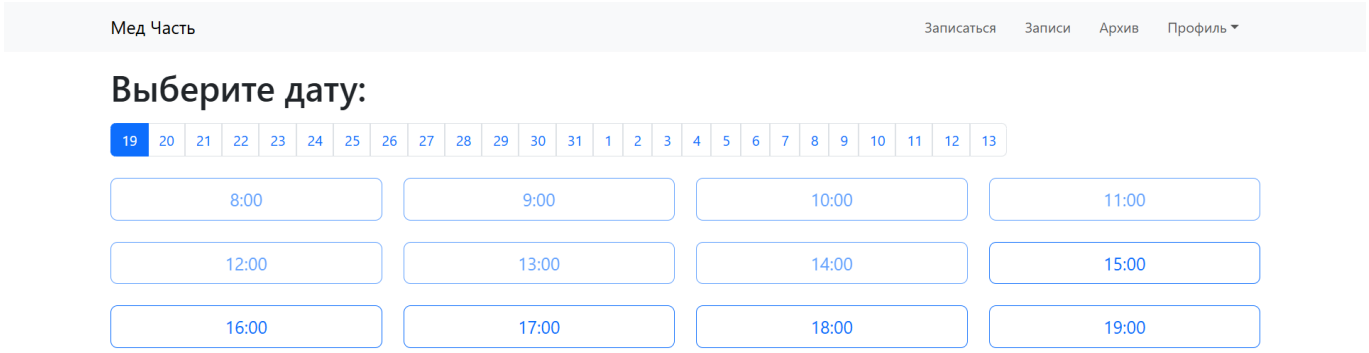


Рисунок 10. Страница записи. Выбор даты

После того, как вы выбрали время – внимательно проверьте выбранные Вами данные для записи, после чего нажмите «Подтвердить запись».

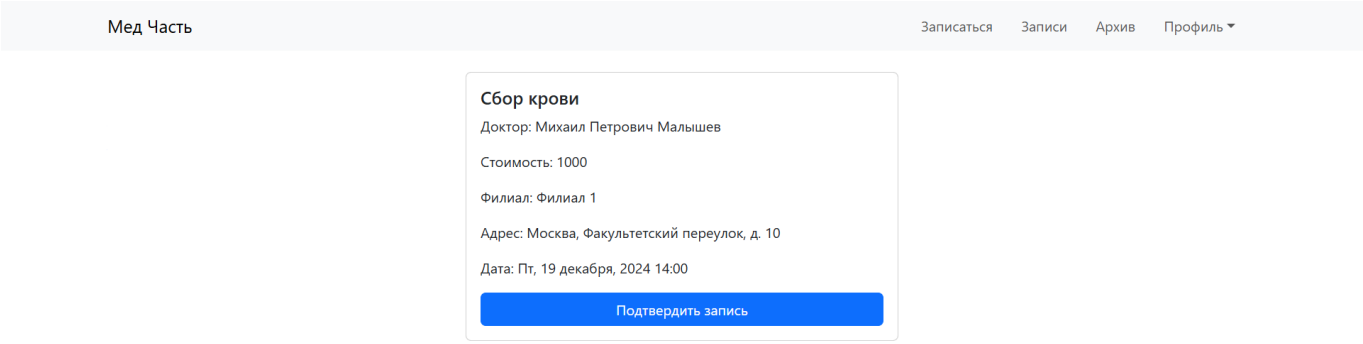


Рисунок 11. Подтверждение записи

После подтверждения записи появится всплывающее окно, в котором будет написан статус записи. Если всё прошло успешно, то будет надпись «Успешно!». Если в ходе записи возникли какие-либо ошибки – окно укажет на них. Нажмите на крестик или на кнопку «На главную» чтобы завершить запись.

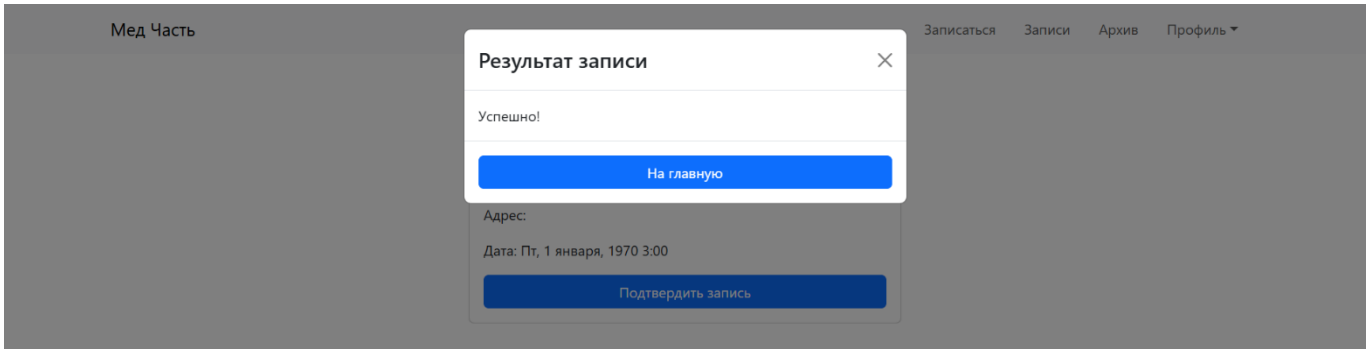


Рисунок 12. Всплывающее окно "Результат записи"

При нажатии кнопки «Записи» в панели навигации откроется страница с текущими записями. Вы можете посмотреть данные о записях, которые вам предстоит посетить, а также отменить запись. Для этого нажмите на кнопку «Удалить запись» в карточке записи.

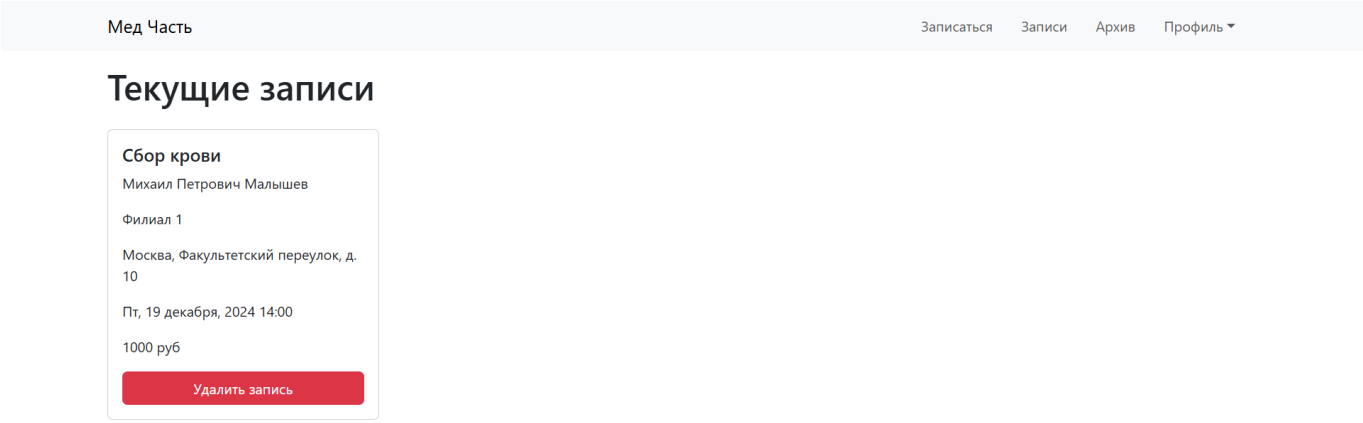


Рисунок 13. Текущие записи

При нажатии кнопки «Архив» в панели навигации откроется страница с прошедшими записями. Вы можете посмотреть данные о записях, которые посетили ранее.

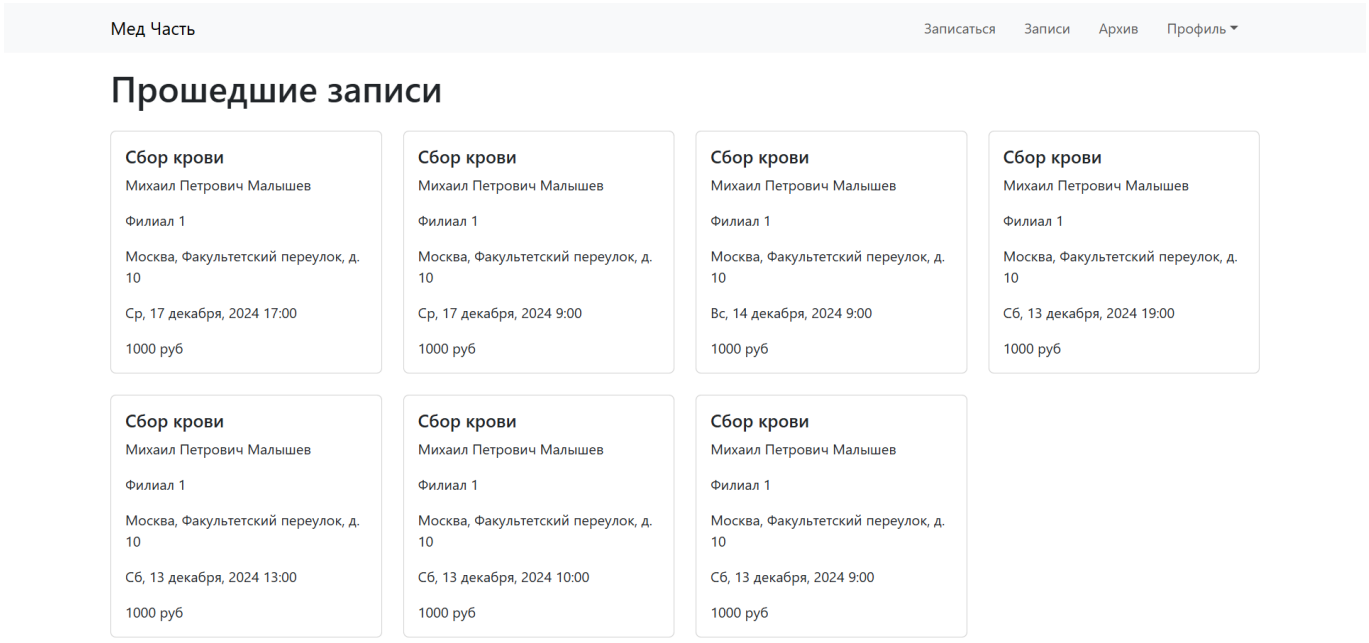


Рисунок 14. Прошедшие записи

Чтобы выйти из профиля, на панели навигации нажмите на раскрывающееся меню «Профиль», после чего на «Выйти».

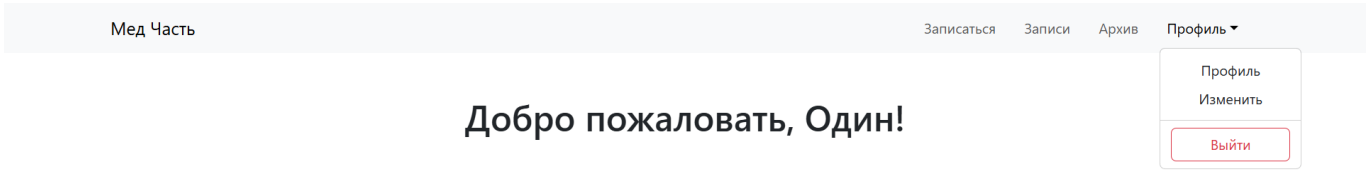


Рисунок 15. Выход из профиля

В случае если вы зашли с аккаунта администратора, в раскрывающемся меню «Профиль» появится дополнительная кнопка «Админ панель» для управления системой.

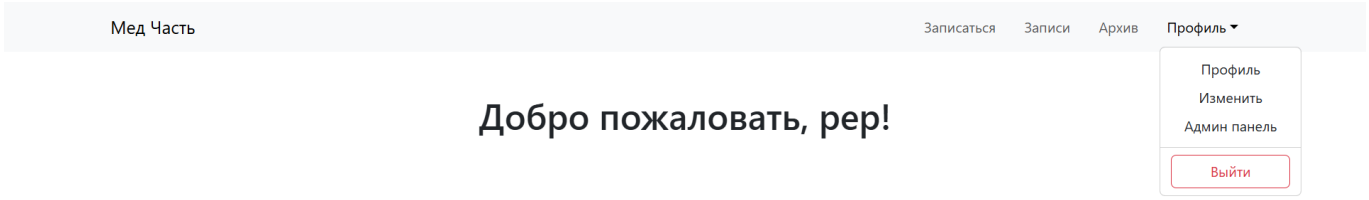


Рисунок 16. Переход в админ панель

При переходе в админ панель вас встречает информация о всех зарегистрированных пользователях. Вы можете увидеть всю их информацию, а также назначить администратором или удалить пользователя, нажав на кнопки «Сделать админом» и «Удалить» соответственно в карточке пользователя. Если пользователь является администратором, вы можете сделать его обычным пользователем, нажав кнопку «Удалить из админов».

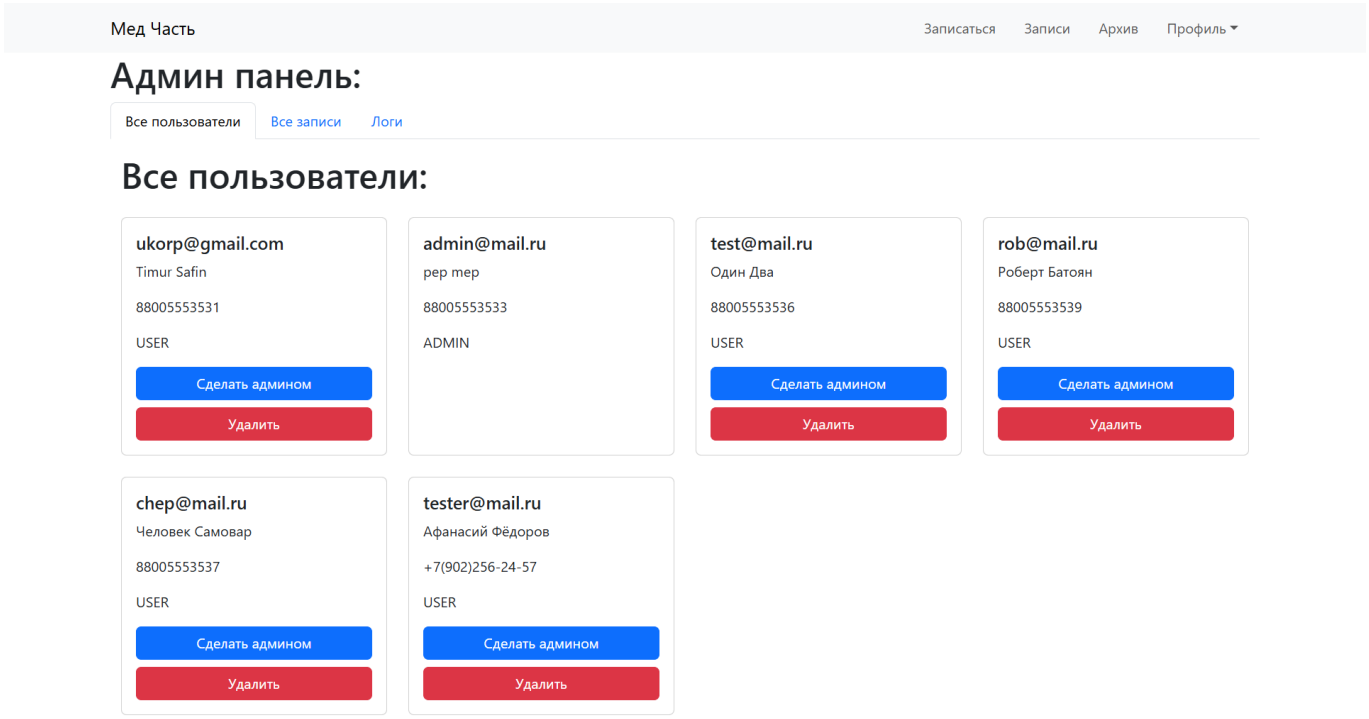


Рисунок 17. Админ панель. Все пользователи

Во вкладке «Все записи» админ панели вы можете посмотреть информацию о абсолютно всех записях каждого пользователя за любое время, а также удалить их, нажав на кнопку «Удалить запись» на карточке нужной записи.

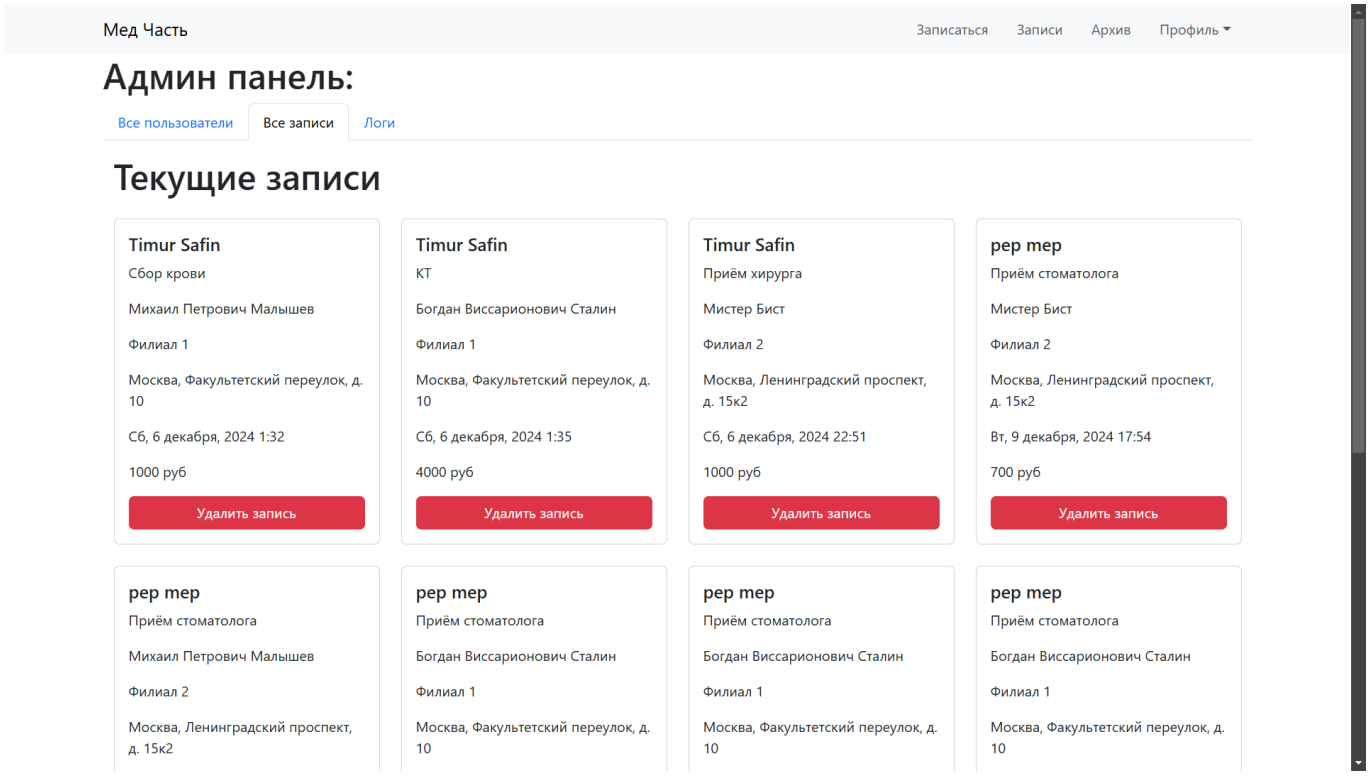


Рисунок 18. Админ панель. Все записи

Во вкладке «Логи» показаны действия вставки, добавления и удаления в базу данных приложения, которые вы можете посмотреть.

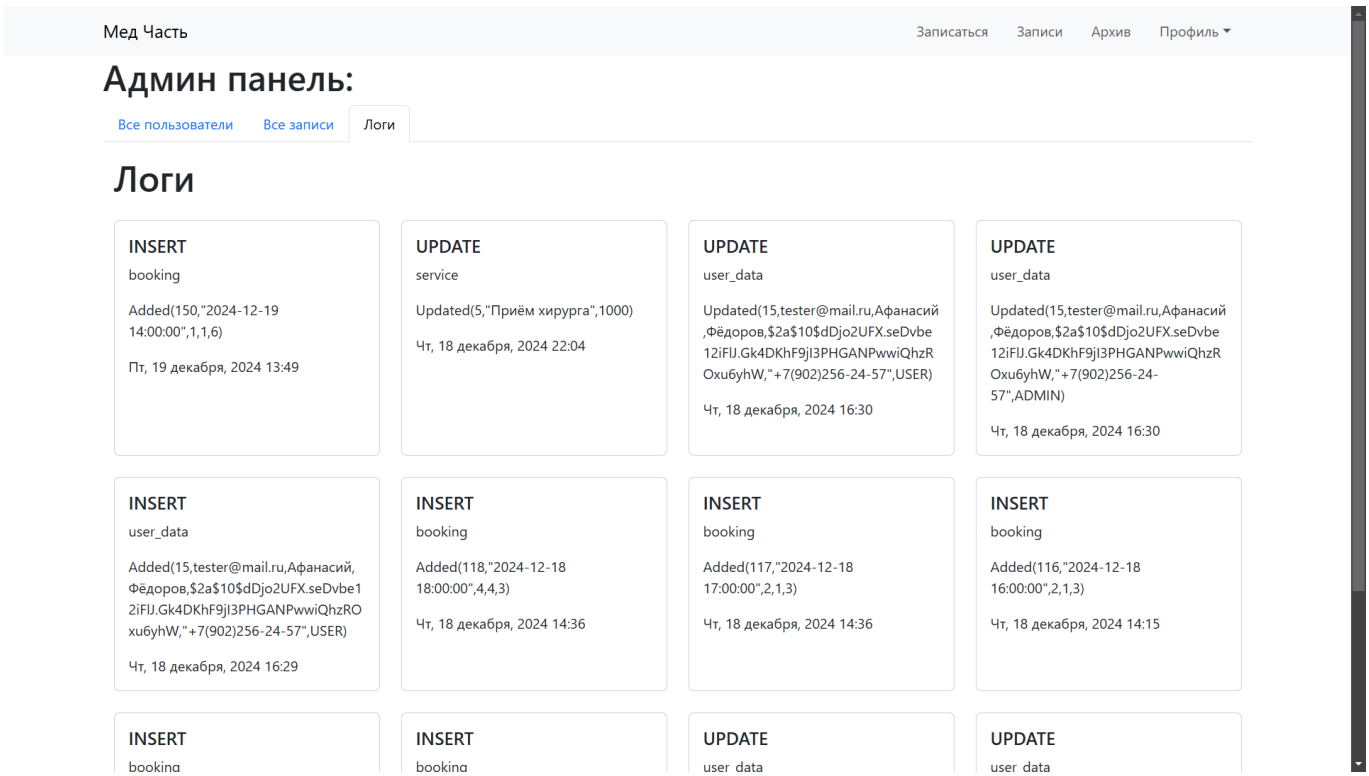


Рисунок 19. Админ панель. Логи

Заключение

В ходе выполнения курсового проекта на тему «Проектирование схемы БД, разработка API и клиентского приложения для информационной модели “Медицинский центр”» была разработана многоуровневая система, предназначенная для автоматизации ключевых процессов медицинского центра.

- Структура базы данных была нормализована до третьей нормальной формы (3NF) для устранения избыточности данных и повышения их целостности.
- Для обеспечения безопасности данных реализованы роли на уровне базы данных (администратор, пациент), которые определяют доступ к таблицам и действиям (чтение, запись, изменение, удаление).
- На стороне сервера реализовано взаимодействие с базой данных через REST API, разработанное на языке Java с использованием фреймворка Spring Boot.
- Для обеспечения безопасности и управления доступом пользователей была внедрена аутентификация и авторизация на основе JWT (JSON Web Token). Это позволило реализовать безопасный доступ к API, включая контроль ролей и прав на выполнение различных действий.
- Пользовательский интерфейс был разработан с использованием библиотеки React, что обеспечило удобство взаимодействия с системой и кроссплатформенность.
- Клиентское приложение поддерживает авторизацию пользователей, представленных в системе (администраторы, врачи, пациенты), с доступом к функционалу в зависимости от их ролей.
- Пациенты могут записываться на приём к врачам и выбирать медицинские услуги, врачи имеют доступ к своему расписанию, а администраторы могут управлять данными о врачах, услугах и расписании.
- Обеспечена обработка ошибок как на уровне клиента, так и сервера, включая информирование пользователей о проблемах (например, неверные данные при входе в систему или попытка выполнения недоступной операции).

В результате реализации проекта была создана надёжная, производительная и безопасная система, автоматизирующая процессы записи на приём и управления медицинским центром. Все этапы разработки соответствовали требованиям технического задания, включая использование PostgreSQL, реализацию аутентификации через JWT, поддержку ролей на уровне базы данных, обработку ошибок и создание клиентского интерфейса.

Система прошла тестирование и продемонстрировала высокую надёжность и удобство использования. Полученные в ходе работы знания в области проектирования баз данных, разработки API и клиентских приложений с распределённым доступом позволили укрепить профессиональные навыки.

Список литературы

[1] Моргунов Е. П. Основы языка SQL. СПб: БХВ-Петербург, 2018. – 336 с.

[2] Гамма Э., Хемл Р., Джонсон Р., Влиссидес Д. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. СПб: Питер, 2001. – 368 с.

[3] Информация по Java, Spring и веб-разработке Baeldung [Электронный ресурс] URL: https://www.baeldung.com/ (дата обращения: 30.11.2024)

[4] Документация фреймворка Spring [Электронный ресурс] URL: https://spring.io/ (дата обращения: 28.11.2024)

[5] Портал со статьями о веб-разработки на Java HowToDoInJava [Электронный ресурс]: Spring-MVC. URL: https://howtodoinjava.com/spring-mvc/ (дата обращения 2.12.2024)

[6] IT-платформа статей Хабр [Электронный ресурс]: сервис аутентификации и авторизации по JWT на основе фильтров SpringSecurity. URL: https://habr.com/ru/articles/781066/ (Дата обращения 2.12.2024)

[7] Документация библиотеки React [Электронный ресурс] URL: https://react.dev/ (дата обращения: 7.12.2024)

Приложение

Сафин Т. А. Приложение кода к проекту на GitHub [Электронный ресурс] URL: https://github.com/Ukorp/MedicineCenterCW