Техническая документации

к проекту системы управления и просмотра расписания учебных заведений

2025 г.

Оглавление

Введение

Данный документ представляет собой сведения о технологиях и библиотеках, архиетктуре и структуре моего проекта. В качестве демонстрации основных структур были использованы UML-диаграммы

Роли и контроль доступа

В системе реализована ролевая модель доступа. Студент — может просматривать только своё расписание. Администратор — имеет доступ к административной панели и операциям управления (Рис 1).

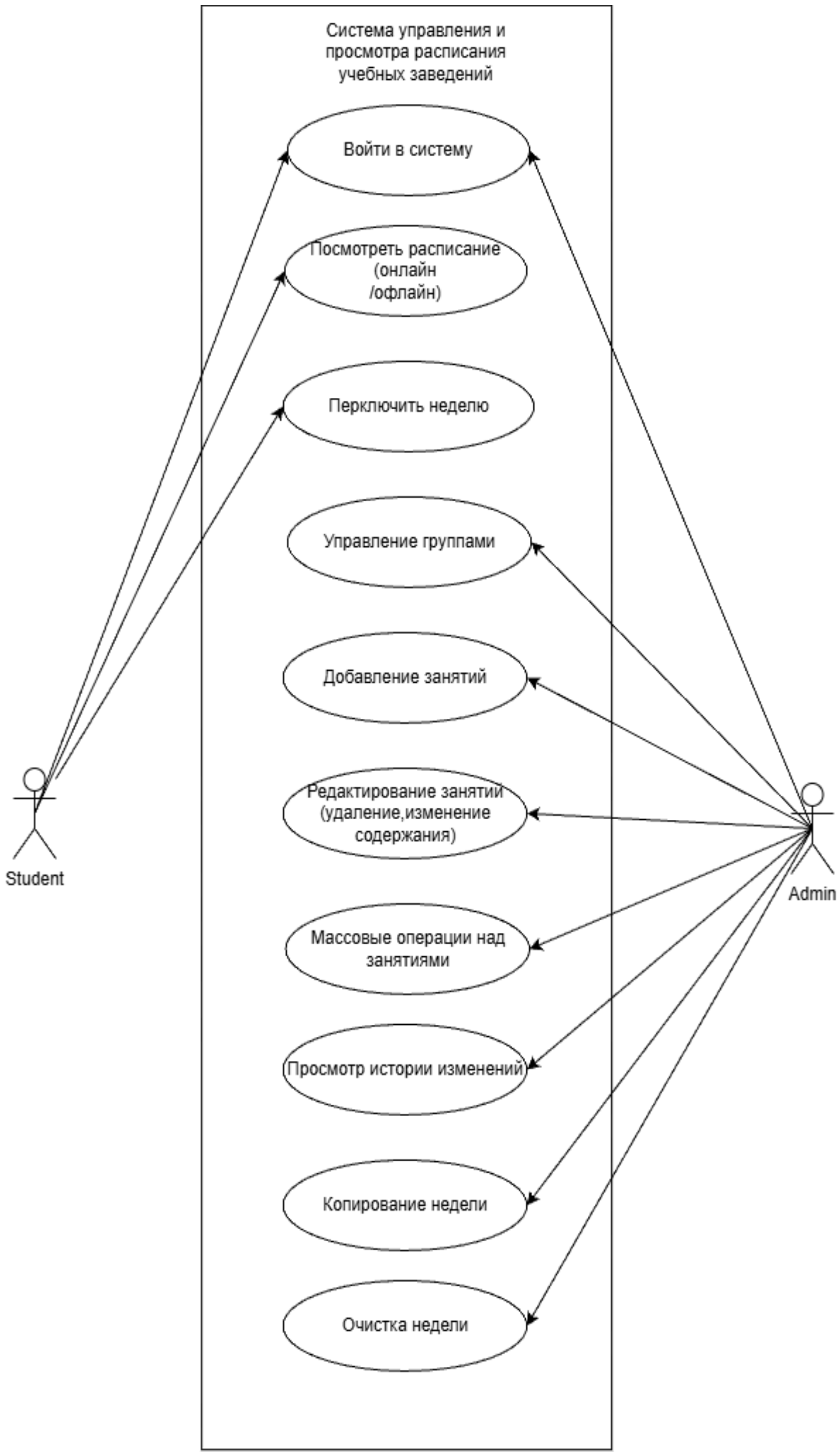

Рис.1 Возможности пользователей (use cases)

Роль пользователя хранится в базе данных и передаётся в JWT.

На сервере используются middleware-функции, которые проверяют:

1. наличие авторизации,
2. валидность токена,
3. роль пользователя перед доступом к защищённым маршрутам.

Используемые технологии и библиотеки

В рамках данного проекта была реализована полнофункциональная информационная система отображения и управления учебным расписанием, построенная на современной клиент-серверной архитектуре(рис.2). При выборе технологического стека основной акцент делался на надёжность, безопасность, масштабируемость и простоту сопровождения без привязки к тяжёлым фреймворкам.

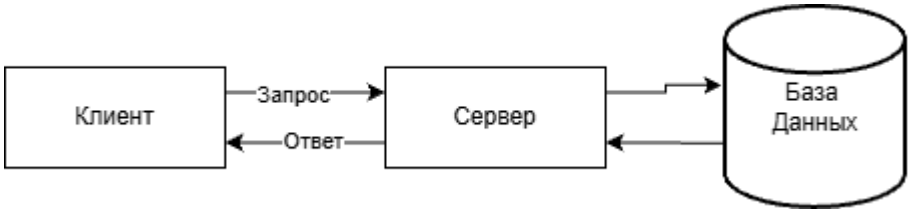

Рис.2 Клиент-серверная архитектура

Общая архитектура

Проект построен по принципу разделения ответственности (Separation of Concerns) и состоит из трёх логических уровней(Рис.3):

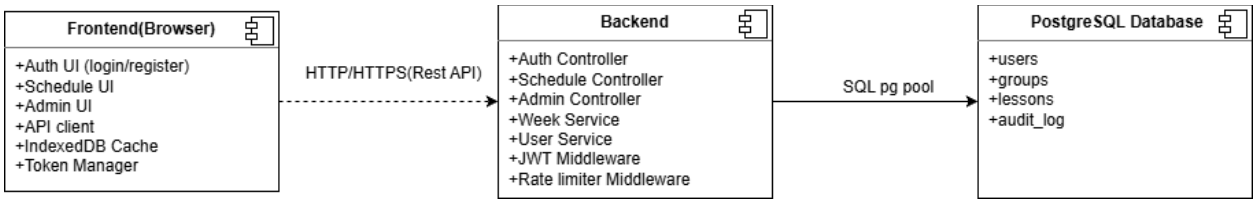

Рис.3 Диаграмма компонентов

- Frontend (клиентская часть) — отвечает за интерфейс пользователя и взаимодействие с API
- Backend (серверная часть) — реализует бизнес-логику, авторизацию и работу с БД
- Database (уровень хранения данных) — централизованное хранение пользователей, расписания и аудита

Каждый уровень может развиваться и масштабироваться независимо.

Backend (серверная часть)

Серверная часть приложения реализована с использованием модульной архитектуры(рис). Код разделён на уровни: маршруты (routes), контроллеры (controllers), сервисы (services) и слой доступа к данным (database layer). Такой подход обеспечивает масштабируемость, читаемость кода и упрощает поддержку системы.

Node.js

В качестве серверной платформы используется Node.js, что позволяет эффективно обрабатывать большое количество параллельных запросов за счёт неблокирующей асинхронной модели ввода-вывода. Это особенно важно для образовательных систем, где возможны пиковые нагрузки (начало учебного дня, сессия, массовые проверки расписания). Данный выбор также обусловлен высокой производительностью Node.js при работе с I/O-операциями, а также простотой построения REST API с помощью Express.

Express.js

Фреймворк Express.js выбран в качестве основы backend-приложения. Он обеспечивает: Простое и прозрачное описание REST-API, поддержку middleware, гибкую маршрутизацию, хорошую читаемость и поддерживаемость кода

Архитектура backend разделена на логические слои (рис.4):

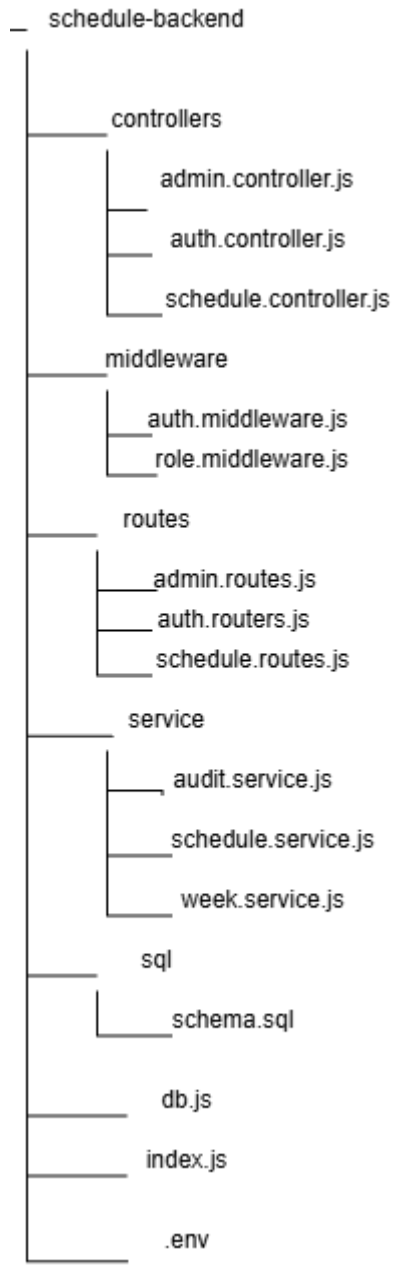

Рис.4 Общая архитектура файлов

- routes — описание HTTP-маршрутов
- controllers — обработка запросов
- services — бизнес-логика
- middleware — авторизация, роли, лимиты
- db — работа с базой данных

PostgreSQL
В качестве основной базы данных используется PostgreSQL — надёжная реляционная СУБД, подходящая для академических и корпоративных систем.Структура базы данных(Рис.5)

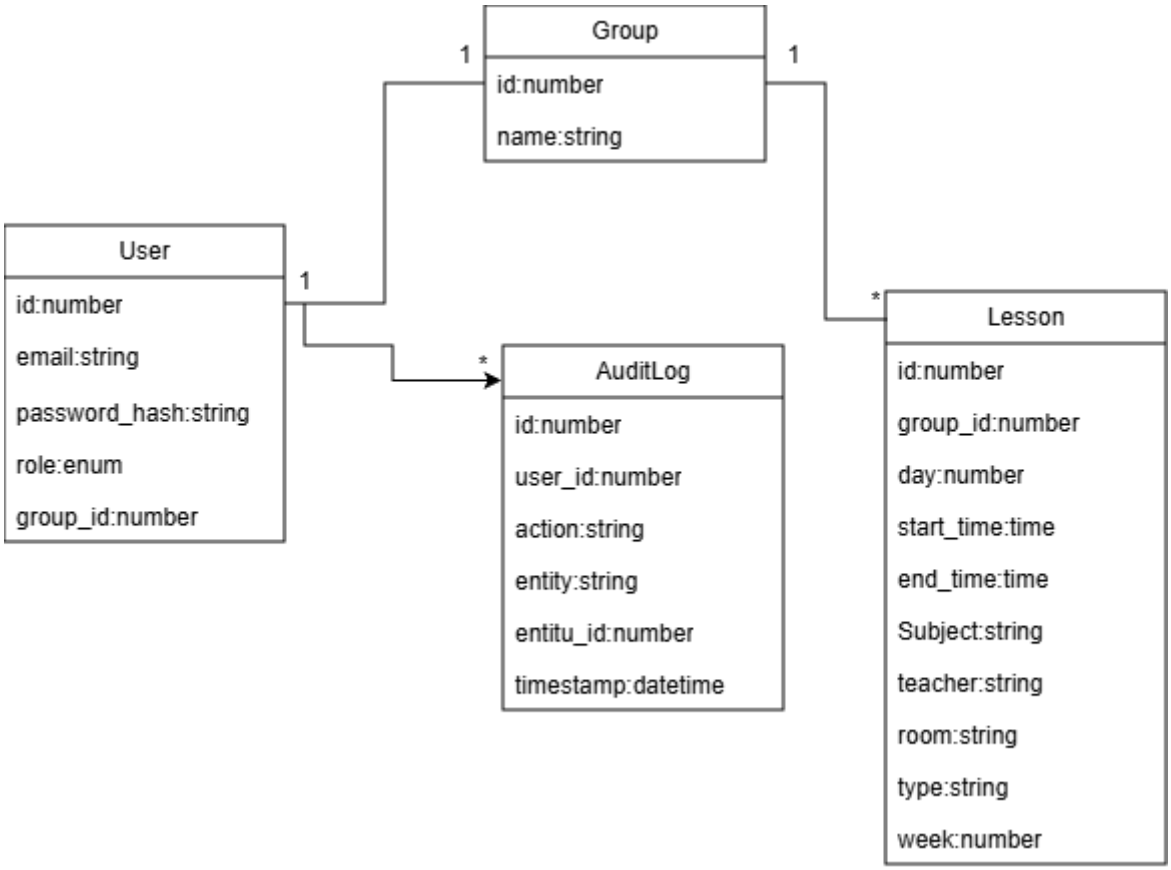

Рис.5 Структура базы данных

PostgreSQL обеспечивает:

- строгую целостность данных
- поддержку сложных связей (foreign keys)
- транзакционность
- расширяемость
- хорошую производительность при росте данных

В базе данных хранятся:

1. пользователи и их роли
2. учебные группы
3. занятия и расписание
4. история изменений (audit log)
5. служебные данные авторизации

pg (node-postgres)

Для взаимодействия с PostgreSQL используется библиотека pg.Она позволяет безопасно выполнять SQL-запросы с параметрами, предотвращая SQL-инъекции.

Работа с БД вынесена в отдельный модуль (db.js), что упрощает сопровождение и облегчает тестирование, а также позволяет в будущем заменить источник данных без переписывания логики

Безопасность

Особое внимание в проекте уделено вопросам безопасности. Реализована аутентификация на основе JWT с разделением на access token и refresh token.
Дополнительно применяются механизмы ограничения частоты запросов и защиты HTTP-заголовков.

JWT (JSON Web Tokens)

Для аутентификации и авторизации используется JWT.Реализована современная двухтокенная схема: Access Token — короткоживущий (≈15 минут), хранится в памяти, Refresh Token — долгоживущий (≈7 дней), хранится в httpOnly cookie (ниже подробнее об этих системах)

Access Token

Используется для доступа к защищённым API-эндпоинтам.Передаётся в заголовке Authorization: Bearer <token>.Имеет короткое время жизни (например, 15 минут).Содержит минимальный набор данных: id, email, role, group.

Refresh Token

Используется исключительно для обновления Access Token.

Хранится в httpOnly cookie, что делает его недоступным из JavaScript и защищает от XSS-атак.Имеет более длительное время жизни (например, 7 дней).Отправляется автоматически браузером при запросе на эндпоинт обновления токена.

Данная схема позволяет:

- повысить безопасность,

- автоматически продлевать сессии пользователя,
- минимизировать риск компрометации учётных данных.

Это решение обеспечивает:
- защиту от XSS-атак
- возможность автоматического обновления сессии
- отсутствие необходимости повторного входа пользователя

bcrypt

Для безопасного хранения паролей используется библиотека bcrypt.Пароли никогда не хранятся в открытом виде — только в виде криптографического хэша.

Middleware безопасности
В проекте используются следующие механизмы защиты:

1. CORS — контроль источников запросов
2. Rate Limiting — защита от брутфорса и DoS:
3. строгий лимит на логин
4. отдельные лимиты для админ-роутов

Role-based access control (RBAC) — разграничение прав студентов и администраторов

HTTP-безопасность (Helmet)
Для усиления защиты HTTP-заголовков используется библиотека Helmet.

Она автоматически:

1. запрещает встраивание сайта в iframe (защита от clickjacking),
2. ограничивает MIME-типы,
3. настраивает Content-Security-Policy (CSP),
4. скрывает информацию о сервере.

Helmet обеспечивает базовый уровень защиты без необходимости ручной настройки заголовков.

Хеширование паролей

Пароли пользователей никогда не хранятся в открытом виде. Для хеширования используется библиотека bcryptjs, которая: применяет адаптивный алгоритм хеширования, устойчива к атакам перебора (brute force),позволяет настраивать сложность (salt rounds).

При регистрации пароль хэшируется и сохраняется в базе данных, а при входе сравнивается с хэшем.

Rate Limiting (ограничение частоты запросов)

Для защиты сервера от злоупотреблений и атак перебором реализован rate limiting с помощью библиотеки express-rate-limit.В проекте применяются разные лимиты:

- общий лимит для всех API-запросов,
- строгий лимит для эндпоинтов логина и регистрации,
- отдельные лимиты для административных маршрутов.

Это позволяет:

- снизить нагрузку на сервер,
- защититься от brute-force атак,
- повысить общую стабильность системы.

Frontend (клиентская часть)
Чистый HTML, CSS и JavaScript

Frontend реализован без использования фреймворков (React/Vue/Angular). Это осознанное архитектурное решение, позволяющее:

- Снизить порог входа для поддержки в вузе
- Упростить деплой
- Избежать сложных сборок
- Повысить прозрачность кода
- Уменьшить требовательнось

Интерфейс полностью контролируется собственным кодом проекта. Стоит добавить, что проект можно легко перенести на React или Vue при желании, но на раннем этапе это не столь нужно.

Fetch API
Для взаимодействия с backend используется стандартный Fetch API:

- асинхронные запросы
- автоматическая обработка 401
- повтор запроса после обновления access token

IndexedDB
Для реализации оффлайн-режима используется IndexedDB.

Функциональность включает:

1. кэширование расписания по неделям
2. чтение данных при отсутствии сети
3. автоматическое обновление при возврате онлайн

Это особенно важно для студентов, которые часто пользуются мобильным интернетом, а также в условиях работы мобильной сети с перебоями

Дополнительные технологии

Адаптивная вёрстка

Интерфейс оптимизирован под мобильные устройства и удобства в применении,при желании проект можно развивать со стороны UI инженерии:

1. media queries
2. горизонтальный скролл расписания
3. крупные элементы управления
4. автоскролл к текущему дню
5. «липкая» шкала времени

Потоки работы

Для наглядного понимания работы ключевых сценариев системы были построены диаграммы последовательностей. Они отражают порядок взаимодействия между клиентом, сервером, системой авторизации и базой данных(Рис 6,7).

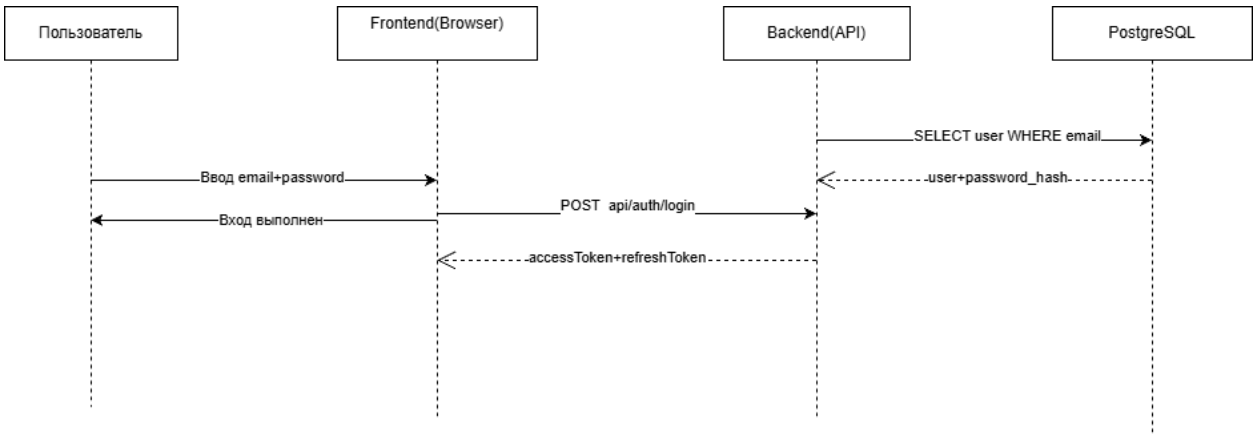

Рис.6 Сценарий регистрации

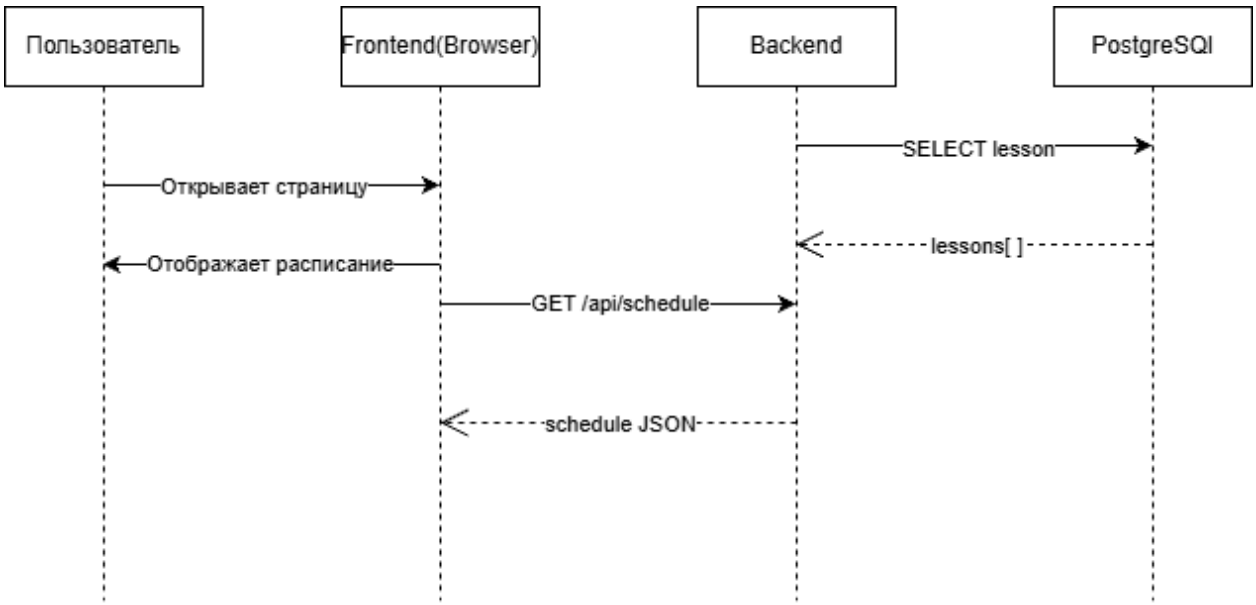

Рис.7 Сценарий работы с расписанием

Выводы по проекту

В рамках данного проекта была разработана полнофункциональная информационная система для отображения и управления расписанием учебных занятий в образовательном учреждении. Основной целью проекта являлось создание удобного, безопасного и масштабируемого решения, ориентированного как на студентов, так и на административный персонал вуза.

Проект успешно решает ключевые проблемы, характерные для традиционных систем расписаний: отсутствие актуальности данных, неудобство использования на мобильных устройствах, высокая нагрузка на персонал при ручном обновлении расписания и отсутствие прозрачности изменений. Реализованная система позволяет централизованно управлять расписанием, обеспечивая при этом мгновенный доступ студентов к актуальной информации.

С архитектурной точки зрения проект построен на современных принципах разделения ответственности и модульности. Backend и frontend полностью разделены и взаимодействуют через REST API, что обеспечивает гибкость, возможность независимого развития компонентов и упрощает сопровождение системы. Использование PostgreSQL в качестве основной базы данных позволило создать надёжное хранилище с чёткой структурой данных и поддержкой сложных запросов.

Особое внимание в проекте уделено вопросам безопасности. Реализована современная схема аутентификации и авторизации с использованием Access Token и Refresh Token, защита от XSS-атак,

ограничение количества запросов (rate limiting), а также базовая защита HTTP-заголовков с помощью Helmet. Это делает систему устойчивой к распространённым типам атак и готовой к эксплуатации в реальной среде.

С точки зрения пользовательского опыта система адаптирована под мобильные устройства, что особенно важно, учитывая, что большинство студентов просматривают расписание со смартфонов. Реализованы такие функции, как автоматическая прокрутка к текущему дню, линия текущего времени, переключение недель и оффлайн-режим с кэшированием данных.

Таким образом, проект можно считать завершённым и готовым к пилотному внедрению в учебном заведении. Он демонстрирует как практическую ценность, так и высокий уровень технической проработки, что делает его сильным примером портфолио-проекта.

Перспективы развития проекта

Несмотря на завершённость текущей версии, архитектура системы изначально проектировалась с учётом дальнейшего развития и масштабирования. В будущем проект может быть расширен и дополнен рядом функциональных и технологических улучшений.

Одним из ключевых направлений развития является расширение пользовательских ролей. Помимо студентов и администраторов, возможно добавление ролей преподавателей, кураторов групп и деканата. Это позволит преподавателям самостоятельно просматривать своё расписание, а ответственным лицам — управлять расписанием отдельных факультетов или курсов.

Следующим логичным шагом является интеграция системы с существующими информационными системами вуза, такими как электронный журнал, система управления обучением (LMS) или корпоративный портал. Это позволит использовать единый источник данных и повысить общую цифровую зрелость образовательного учреждения.

Также перспективным направлением является внедрение системы уведомлений. Студенты могут получать push-уведомления, email или сообщения в мессенджерах при изменении расписания, отмене занятий или переносе пар. Это значительно повысит информированность и снизит количество организационных ошибок.

С технической точки зрения возможно дальнейшее усиление отказоустойчивости системы. Например, добавление Redis для кэширования, внедрение очередей (RabbitMQ / BullMQ) для обработки фоновых задач, а

также контейнеризация приложения с использованием Docker для упрощения развертывания и масштабирования.

Наконец, проект может быть развёрнут как SaaS-решение, позволяющее подключать несколько учебных заведений в рамках одной платформы, что открывает возможности коммерческого использования и дальнейшего развития продукта.

Стоит также отметить, что из данной системы можно создать полноценный профиль студента, позволяющий отслеживать успеваемость, добавлять достижения в учебной и внеучебной деятельности, а также полностью цифр визировать личные дела студентов, добавить взаимодействие с деканатом и кафедрой.