МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»

Специализация: «Эксплуатация вычислительных машин, комплексов, систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРИЛОЖЕНИЕ ДЛЯ СОСТАВЛЕНИЯ РАСПИСАНИЯ В ВОЕННОМ УЧЕБНОМ ЗАВЕДЕНИИ

Студент (подпись, дата) Жихарева Алёна Владимировна (ФИО полностью)

Руководитель (подпись, дата) 01.06.2026 Слепцов Н.В. (фамилия, инициалы)

Консультанты: по (наименование раздела) (подпись, дата) (фамилия, инициалы)

Нормоконтролёр (подпись, дата) 01.06.2026 А.С.Бычков (фамилия, инициалы)

Рецензент доцент ПензГТУ (должность, место работы) (подпись, дата) 04.06.26 С.В.Пегерский (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

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

Работа защищена с отметкой отлично (протокол заседания ГЭК от 10 июня 2026 г. № 1)

Секретарь ГЭК Е.И. Гурин

Пенза 2026

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

Факультет вычислительной техники

Кафедра «Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____ М.А. Митрохин
«11» 02 2026 г.

ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Жихарева Алёна Владимировна гр. 21ВОЭ1 факультета Вычислительной техники

2.Тема работы «Приложение для составления расписания в военном учебном заведении»

Тема утверждена приказом ПГУ № 122/о от "11.02.2026" 2026 г.

3. Руководитель работы к.т.н доцент кафедры ВТ Слепцов Н.В.

4. Задание на работу (назначение разработки, исходные данные и т.п)
Назначением разработки является создание веб-приложения для автоматизации процессов формирования, ведения и публикации расписания учебного заведения. Система должна обеспечивать централизованное хранение данных об учебных группах, преподавателях, дисциплинах, аудиториях и временных слотах, а также предоставлять удобный интерфейс для трёх категорий пользователей: администратора (сотрудника учебного отдела), преподавателя и студента. Исходными данными для разработки служат типовые аналоги учебного заведения по составлению расписания, требования к разграничению прав доступа, форматы входных документов (.xlsx, .docx), а также необходимость интеграции с внешним API для интеллектуальной обработки документов.

5. Перечень подлежащих разработке вопросов
В ходе выполнения дипломного проекта необходимо провести анализ предметной области и существующих аналогов, обосновать актуальность разработки и сформулировать техническое задание. Следует выполнить выбор технологического стека для реализации серверной и клиентской частей, а также системы управления базами данных. Необходимо спроектировать архитектуру приложения, включая ER-модель базы данных, UML-диаграммы вариантов использования, последовательностей, классов и состояний. Требуется разработать серверную часть на базе FastAPI с реализацией REST API, аутентификации и авторизации на основе JWT-токенов, а также алгоритма автоматической генерации расписания с матричной моделью состояния и системой штрафов. Клиентская часть должна быть реализована с использованием React с компонентной архитектурой, адаптивным интерфейсом и поддержкой трёх ролей пользователей. Кроме того, необходимо реализовать импорт данных из файлов .xlsx и .docx с применением YandexGPT API, а также выполнить тестирование разработанной системы. В составе пояснительной записки должны быть представлены разделы по безопасности жизнедеятельности и экономическому обоснованию проекта.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области и анализ существующих аналогов	40	11.02.2026-16.02.2026	
2.	Анализ требований и постановка задачи	20	17.02.2026-20.02.2026	
3.	Выбор технологий, применяемых при проектировании и разработке (FastAPI, PostgreSQL, React)	30	21.02.2026-25.02.2026	
4.	Проектирование архитектуры системы, базы данных и пользовательского интерфейса	60	26.02.2026-06.03.2026	
5.	Разработка серверной части (FastAPI, PostgreSQL, генерация расписания)	80	07.03.2026-27.03.2026	
6.	Разработка клиентской части (React, компоненты, API-взаимодействие)	80	01.05.2026-11.05.2026	
7.	Интеграция компонентов, импорт данных через YandexGPT API, тестирование	40	12.05.2026-17.05.2026	
8.	Оформление пояснительной записки (включая БЖД и экономику)	20	18.05.2026-21.05.2026	

Дата выдачи задания "11" 02 2026 г.

Руководитель работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета А.Р. Фалова
04.06.2026 *(подпись, дата)*

Реферат

Пояснительная записка содержит 140 листов, 23 рисунка, 1 таблица, 10 источников и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, РАСПИСАНИЕ, АВТОМАТИЗАЦИЯ, FASTAPI, REACT, POSTGRESQL

Основная цель исследования заключается в создании веб-приложения, предназначенного для автоматизации процессов формирования и обновления расписания учебного заведения. Разрабатываемая система обеспечивает централизованное хранение данных о занятиях, преподавателях, учебных группах и аудиториях, а также предоставляет удобные инструменты для их редактирования и отображения. Приложение ориентировано на повышение точности расписания, снижение количества ошибок и упрощение взаимодействия пользователей с информацией.

В ходе работы проведено изучение предметной области, рассмотрены существующие решения и выявлены их ограничения. На основе анализа определены требования к функциональности и архитектуре будущей системы. Разработана структура базы данных, описаны механизмы обработки данных и реализованы ключевые компоненты: модуль авторизации, административная панель для управления расписанием, интерфейс визуализации занятий и подсистема работы с данными учебного процесса.

Техническая реализация выполнена с использованием современных инструментов: FastAPI применён для разработки серверной части, PostgreSQL - в качестве системы управления базами данных, React - для построения интерактивного пользовательского интерфейса. Проведённое тестирование подтвердило корректность функционирования приложения и соответствие поставленным требованиям.

					ПГУ 1.090501.05.001 ПЗ	
Изм.	Лист	№ докум.	Подпись	Дата		
Разраб.		Жихарева А.В.		01.06.26	Приложение для составления расписания в военном учебном заведении Пояснительная записка	Лит. / Лист: 2 / Листов: 143
Провер.		Слепцов И.В.		01.06.26		
Н. Контр.		Бычков А.С.		01.06.26		ФВТ, каф. ВТ гр. 21ВОЭ1
Утверд.		Митрохин М.А.		01.06.26		

Содержание

Перечень сокращений и обозначений 5

Введение 6

1 Анализ предметной области 8

1.1 Общая характеристика учреждения здравоохранения 8

1.1.1 Анализ существующих аналогов 10

1.1.2 Обоснование актуальности разработки программы 14

1.1.3 Постановка задачи 16

1.1.4 Выбор технологий и инструментов 19

1.1.5 Выбор серверной платформы 24

1.1.6 Выбор системы управления базами данных 26

1.1.7 Выбор технологий клиентской части 28

1.1.8 Формат обмена данными и взаимодействие компонентов 30

1.2 Проектирование и разработка 32

1.2.1 Проектирование базы данных 34

1.2.2 Проектирование пользовательского интерфейса 38

1.2.3 Разработка серверной части 45

1.2.4 Разработка клиентской части 49

1.2.5 Интеграция и взаимодействие компонентов 52

1.2.6 Обеспечение безопасности 55

1.2.7 Динамика работы приложения 57

1.2.8 Тестирование приложения 60

2 Безопасность жизнедеятельности 62

2.1 Характеристика рабочего места разработчика 62

2.2 Электробезопасность при работе с компьютерным оборудованием 63

2.3 Профилактика утомления при работе с компьютером 64

3 Экономика 65

Заключение 68

Список использованных источников 69

Приложение А – UML-диаграммы 70

Приложение Б – Листинг программы 75

Приложение В – Презентация 115

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

API (Application Programming Interface) – программный интерфейс для взаимодействия между клиентской и серверной частями системы.

CRUD (Create, Read, Update, Delete) – базовые операции над данными в информационных системах.

DB (DataBase) – база данных.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, который используется в качестве сохранения игровых настроек и прогресса.

DTO (Data Transfer Object) – объект передачи данных между слоями приложения.

HTTP (HyperText Transfer Protocol) – протокол передачи данных между клиентом и сервером.

REST (Representational State Transfer) – архитектурный стиль построения веб-сервисов.

React – библиотека JavaScript для создания пользовательских интерфейсов.

UML (Unified Modeling Language) – унифицированный язык моделирования.

FastAPI – фреймворк для разработки серверной части на Python.

SSL (Secure Sockets Layer) – технология шифрования данных при передаче.

TLS (Transport Layer Security) – протокол защиты данных, используемый в HTTPS.

IP (Internet Protocol) – сетевой протокол, используемый для адресации устройств.

UX (User Experience) – пользовательский опыт, удобство взаимодействия с системой.

Введение

Современное образование работает в условиях постоянного роста объёма информации и всё более сложной организации учебного процесса. Учебным заведениям приходится одновременно управлять данными о расписании занятий, занятости преподавателей, использовании аудиторий и соблюдении учебных планов. Когда увеличивается количество учебных групп и расширяется список образовательных программ, привычные способы составления расписания - ручное редактирование таблиц или работа в нескольких несвязанных между собой сервисах - уже не справляются с нагрузкой. Из-за этого появляются ошибки, информация обновляется с задержками, а взаимодействие между участниками учебного процесса становится менее удобным.

Цифровая трансформация образования усиливает спрос на системы, которые объединяют все данные в едином информационном пространстве и дают быстрый доступ к актуальному расписанию. Информационные системы для автоматизации учебного процесса помогают сократить объём рутинной работы, сделать процессы более прозрачными и быстрее вносить изменения. Такие решения упрощают работу учебных подразделений и улучшают взаимодействие студентов, преподавателей и администрации.

При этом многие существующие программные продукты не учитывают особенности конкретных учебных заведений. Одни системы оказываются слишком сложными в освоении и перегруженными функциями, рассчитанными на крупные организации. Другие, наоборот, предлагают слишком ограниченные возможности и не закрывают реальные потребности пользователей. Поэтому возникает необходимость в специализированном веб-приложении, которое сочетает удобный интерфейс, гибкую настройку и быстрое обновление данных.

Веб-ориентированная система управления расписанием даёт пользователям доступ к информации с любого устройства, позволяет хранить данные в едином центре и делает работу с системой более удобной. Разработка такого приложения включает анализ предметной области, изучение существующих решений и формирование требований к системе. После этого проектируется архитектура и создаются функциональные модули, которые обеспечивают стабильную и корректную работу приложения.

1 Анализ предметной области

1.1 Общая характеристика учебного заведения

Учебное заведение представляет собой сложную организационную систему, где образовательные, административные и методические процессы тесно связаны между собой. Для стабильной работы необходимо координировать действия преподавателей, студентов, сотрудников учебных отделов и технических служб. У каждого участника свои задачи, но все они направлены на одно - обеспечение непрерывного и качественного учебного процесса.

Одной из центральных задач образовательной организации остаётся управление учебным расписанием. В этот процесс входит распределение дисциплин по семестрам, формирование потоков, назначение преподавателей, подбор аудиторий и проверка расписания на отсутствие пересечений и конфликтов. При составлении расписания учитываются учебные планы, последовательность изучения дисциплин, занятость преподавателей и доступность помещений. Любые изменения - перенос занятия, замена преподавателя или аудитории - требуют быстрого обновления информации и своевременного уведомления всех участников учебного процесса.

Отдельное место занимает работа с учебной документацией. Сюда входят данные о дисциплинах, учебных планах, преподавательской нагрузке, распределении часов и отчётности по выполнению учебного процесса. Информация должна храниться в структурированном виде, быть доступной для анализа и редактирования, а также защищённой от несанкционированного доступа. Ошибки в документах или расхождения между данными разных подразделений могут нарушить учебный процесс и усложнить планирование.

Административные подразделения учебного заведения решают широкий круг задач: учитывают учебную нагрузку, контролируют выполнение учебных планов, формируют отчётность и координируют взаимодействие между кафедрами и учебными группами. Для такой работы нужны инструменты, которые позволяют быстро получать актуальные данные, анализировать информацию и вносить изменения без риска нарушить согласованность расписания.

Учебное заведение представляет собой многокомпонентную систему, где точное ведение расписания и учебной документации напрямую влияет на стабильность учебного процесса. С увеличением объёма данных, ростом числа учебных групп и усложнением образовательных программ традиционные способы управления расписанием перестают справляться с нагрузкой. Из-за этого возникает потребность в специализированных информационных системах, которые автоматизируют рутинные операции, обеспечивают точность данных и ускоряют работу сотрудников.

1.1.1 Анализ существующих аналогов

Сегодня для автоматизации учебного процесса используется множество программных решений, поддерживающих работу с расписанием. Наиболее распространены комплексные образовательные платформы с модулями планирования занятий, управления учебными планами и организации взаимодействия между участниками образовательной среды. При этом их функциональность и удобство работы заметно отличаются [2].

Одним из наиболее известных решений считается система "1С:Колледж" и её модификации, которые применяются в средних и высших учебных заведениях. Платформа включает инструменты для ведения учебной документации, расчёта нагрузки преподавателей и составления расписания. При широких возможностях система остаётся сложной в настройке, требует специализированного сопровождения и не всегда позволяет быстро вносить изменения в расписание.

Интерфейс ориентирован преимущественно на административных сотрудников, что ограничивает удобство использования для студентов и преподавателей.

Внешний вид приложения представлен на рисунке 1.

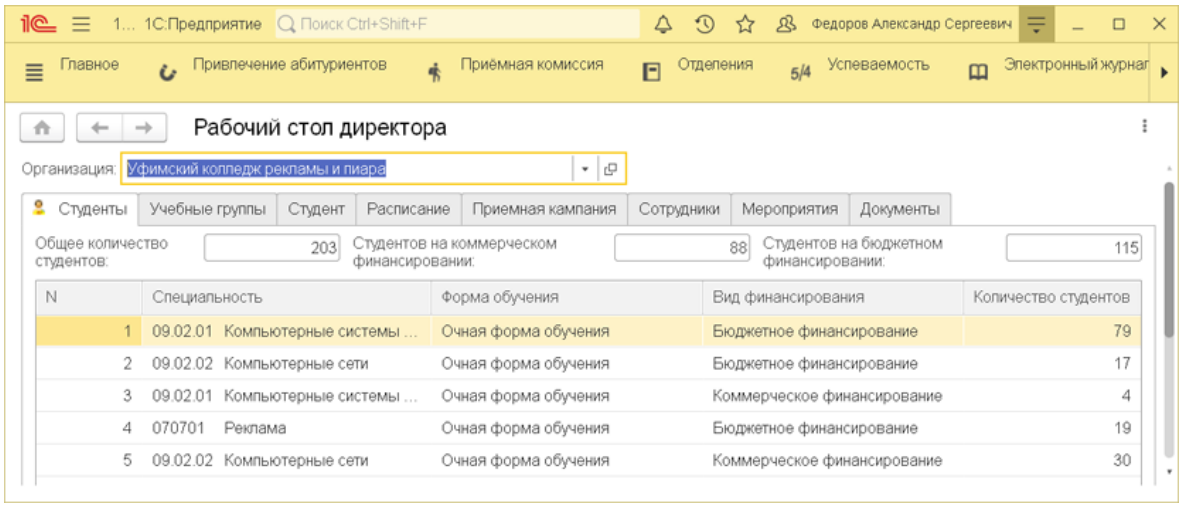


Рисунок 1 – Внешний вид «1С:Колледж»

Другим распространённым аналогом является сервис «АСУ РСО», применяемый в ряде университетов. Он поддерживает публикацию расписания, отображение занятий по группам и преподавателям, а также предоставляет базовые инструменты для редактирования данных. Однако система имеет закрытую архитектуру, ограниченные возможности кастомизации и не позволяет гибко адаптировать функционал под особенности конкретного учебного заведения. Кроме того, интерфейс нередко воспринимается пользователями как перегруженный и устаревший.

Внешний вид приложения представлен на рисунке 2.

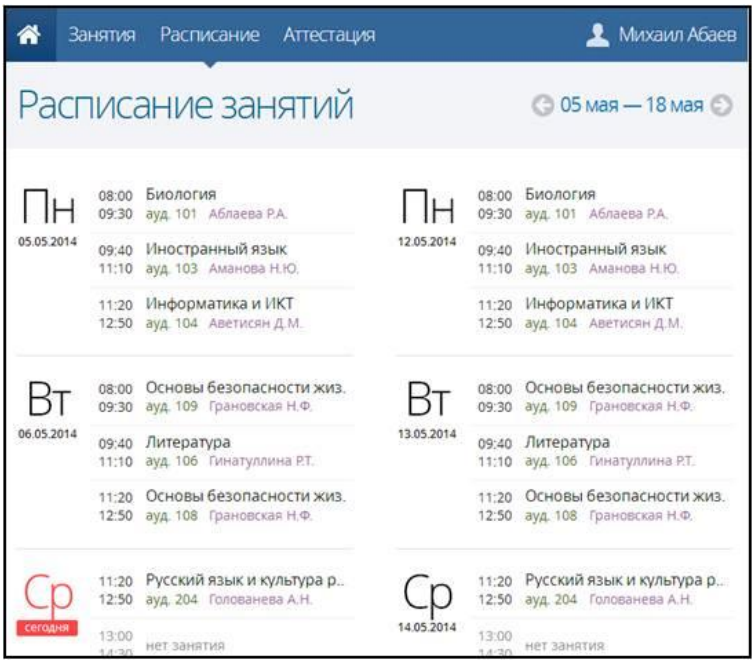


Рисунок 2 – Внешний вид «АСУ РСО»

Сервис «EduPlan» относится к более современным веб-решениям и ориентирован на визуальное представление расписания. Он позволяет отображать занятия по группам, преподавателям и аудиториям, но его функциональность ограничена просмотром данных. Возможности редактирования расписания минимальны, отсутствуют инструменты для учёта учебных планов и автоматического распределения нагрузки. Таким образом, сервис подходит лишь для публикации уже сформированного расписания [10].

Внешний вид приложения представлен на рисунке 3

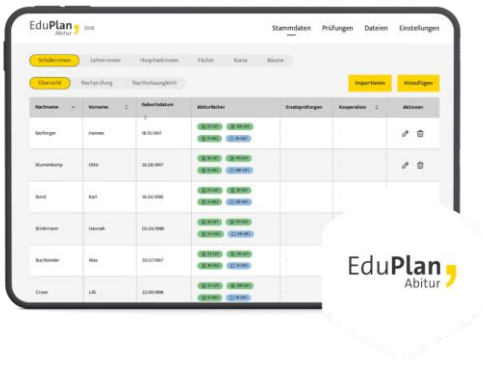


Рисунок 3 – Внешний вид «EduPlan»

В таблице 1 представлен анализ аналогов.

Таблица 1 – Анализ аналогов

Название системы	Основные возможности	Преимущества	Недостатки
1С:Колледж	Управление учебными планами, нагрузкой преподавателей, формирование расписания, отчётность	Широкий функционал, интеграция с другими модулями 1С, распространённость	Сложная настройка, высокая стоимость, перегруженный интерфейс
АСУ РСО	Публикация расписания, отображение занятий по группам и преподавателям, базовое редактирование	Удобный просмотр, стабильная работа, используется в вузах	Ограниченная кастомизация, устаревший интерфейс, закрытая архитектура
EduPlan	Веб-интерфейс для визуального отображения расписания, удобный просмотр	Простота использования, современный интерфейс	Нет полноценного редактирования, отсутствует учёт учебных планов
EduSched	Поддержка создания и редактирования расписания, работа с данными групп.	Гибкая архитектура, удобство использования, возможность адаптации под конкретные процессы	Функциональность пока ограничена базовыми модулями

1.1.2 Обоснование актуальности разработки программы

Организация учебного процесса в современных образовательных учреждениях становится всё сложнее. Количество учебных групп растёт, образовательные программы расширяются, появляются гибкие форматы обучения, а расписание приходится регулярно корректировать. Всё это требует быстрой работы с большим объёмом данных. Привычные способы составления расписания - электронные таблицы и набор несвязанных между собой инструментов - уже не обеспечивают нужную точность, скорость обновления информации и согласованность между участниками учебного процесса.

Ситуацию усложняет необходимость одновременно учитывать множество факторов: занятость преподавателей, доступность аудиторий, требования учебных планов, специфику отдельных дисциплин и возможные пересечения занятий. При ручном составлении расписания возрастает вероятность ошибок и задержек. Из-за несогласованности данных страдает организация учебного процесса, а студенты и преподаватели сталкиваются с постоянными неудобствами.

Развитие цифровой образовательной среды повышает спрос на специализированные программные решения, которые способны автоматизировать рутинные операции и хранить данные в единой системе. Веб-ориентированные платформы дают доступ к актуальной информации с любого устройства, упрощают внесение изменений и делают взаимодействие между подразделениями учебного заведения более прозрачным.

При этом многие существующие программные продукты либо слишком сложны и дороги во внедрении, либо недостаточно гибки и плохо адаптируются под особенности конкретного образовательного учреждения.

Создание веб-приложения для управления расписанием остаётся актуальной задачей, поскольку позволяет повысить эффективность организации учебного процесса, сократить количество ошибок и упростить взаимодействие между всеми участниками образовательной среды.

1.1.3 Постановка задачи

Анализ текущей организации учебного процесса и существующих проблем показывает необходимость разработки программного обеспечения, которое обеспечит более удобное и эффективное управление расписанием в образовательном учреждении. Сегодня учебным заведениям приходится работать с большим объёмом данных: учебными группами, преподавателями, дисциплинами и аудиториями. Информация постоянно меняется, поэтому расписание нужно оперативно обновлять и своевременно доводить изменения до всех участников образовательного процесса.

Использование электронных таблиц и отдельных локальных программ часто приводит к ошибкам, усложняет взаимодействие между подразделениями и снижает прозрачность работы с данными.

Поэтому возникает необходимость в создании веб-системы с централизованным хранением информации, удобными инструментами редактирования расписания и доступом к актуальным данным в режиме реального времени. Разрабатываемое приложение должно поддерживать формирование структуры учебного процесса, учитывать особенности учебных планов, распределять занятия по аудиториям и преподавателям, а также предотвращать конфликты при составлении расписания. Отдельное внимание необходимо уделить интерфейсу системы. Он должен быть понятным и удобным, чтобы сотрудники учебного отдела могли быстро ориентироваться в данных и вносить изменения без риска нарушить согласованность расписания.

Система также должна поддерживать авторизацию и разграничение прав доступа. Это позволит обеспечить безопасную работу с данными и учитывать различия между ролями пользователей. Например, сотрудники учебного отдела смогут редактировать расписание, преподаватели - просматривать изменения по своим дисциплинам, а студенты - получать доступ только к актуальной информации о занятиях.

Архитектуру приложения необходимо сделать гибкой и расширяемой, чтобы в дальнейшем можно было подключать дополнительные модули: автоматическое распределение учебной нагрузки, анализ посещаемости или интеграцию с внутренними информационными системами учебного заведения.

Основная задача разработки заключается в создании современного веб-приложения, которое сочетает удобный интерфейс, надёжность, адаптивность и возможность дальнейшего развития. Система должна повысить эффективность организации учебного процесса и упростить взаимодействие между всеми его участниками.

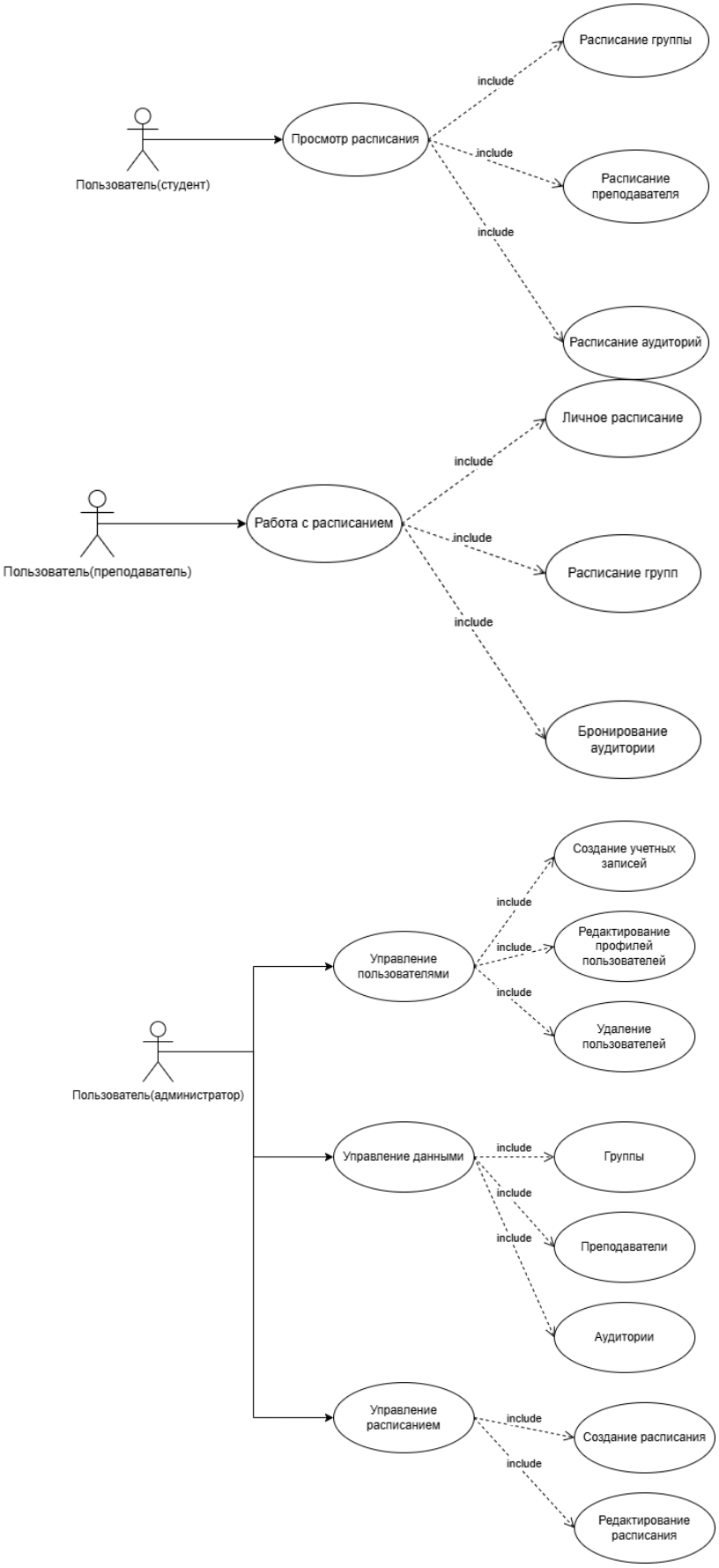


Рисунок 4 – UML-диаграмма вариантов использования

Пользовательские сценарии.

Сценарий 1. Вход в систему.

Администратор открывает страницу входа, вводит свой логин и пароль, после чего нажимает кнопку "Войти". Система проверяет учётные данные, генерирует JWT-токен и перенаправляет пользователя на страницу "Дашборд". При вводе неверных данных система отображает сообщение об ошибке и не предоставляет доступ.

Сценарий 2. Просмотр сводной статистики (дашборд).

После успешной авторизации администратор переходит на главную страницу, где система отображает ключевые показатели: количество занятий на сегодня, общее число студентов, количество преподавателей и число аудиторий. Это позволяет администратору быстро оценить текущую загруженность системы.

Сценарий 3. Создание нового пользователя.

На странице "Пользователи" администратор нажимает кнопку "Добавить пользователя", заполняет форму (ФИО, email, роль - студент или преподаватель, группа для студента), после чего сохраняет данные. Система создаёт учётную запись, генерирует пароль и отправляет его на указанный email. Пользователь получает возможность войти в систему.

Сценарий 4. Редактирование и удаление пользователя.

Администратор выбирает пользователя из списка, редактирует его данные (ФИО, группа, статус активности) и сохраняет изменения. При необходимости администратор может удалить пользователя, после чего система блокирует доступ к учётной записи.

Сценарий 5. Управление аудиториями.

Администратор переходит на страницу "Аудитории", где видит список всех помещений. Он может добавить новую аудиторию (указать номер, корпус, вместимость), отредактировать существующую или удалить её. Все изменения мгновенно учитываются при формировании расписания.

Сценарий 6. Импорт расписания из файла.

На странице "Импорт" администратор загружает файл формата .xlsx или .docx, содержащий данные о дисциплинах, преподавателях и учебных группах. Система с использованием YandexGPT API извлекает из документа структурированную информацию и сохраняет её в базе данных. После завершения обработки администратор получает уведомление: об успешном импорте либо о возникших ошибках с указанием причин.

Сценарий 7. Генерация расписания.

Администратор запускает автоматическую генерацию расписания. Система использует матричную модель состояния и механизм штрафов, чтобы распределить занятия по дням, временным слотам, преподавателям и аудиториям с учётом всех заданных ограничений. Готовое расписание сохраняется в базе данных и становится доступным для просмотра студентам и преподавателям.

Сценарий 8. Просмотр и редактирование расписания.

На странице "Расписание" администратор видит полную сетку занятий с фильтрацией по учебным группам и типу недели. Система позволяет вручную добавить новое занятие: выбрать группу, преподавателя, аудиторию, день и время проведения. При необходимости администратор может изменить или удалить уже существующую запись. Перед сохранением система проверяет занятость преподавателя, группы и аудитории, поэтому конфликты в расписании не допускаются.

Сценарий 9. Обработка заявок на аудиторию.

На странице "Заявки" администратор просматривает обращения преподавателей на бронирование аудиторий. Он может подтвердить заявку - тогда занятие автоматически добавляется в расписание - либо отклонить её с указанием причины. После изменения статуса преподаватель получает уведомление.

Сценарии для преподавателя.

Сценарий 10. Вход в систему.

Преподаватель вводит свои логин и пароль на странице входа. Система проверяет данные, определяет роль "преподаватель" и перенаправляет на персонализированный дашборд.

Сценарий 11. Просмотр личного расписания.

Преподаватель открывает страницу "Расписание", где система автоматически отображает занятия, привязанные к его идентификатору. Расписание представлено в виде таблицы с группировкой по дням недели и временным слотам с учётом типа недели (числитель/знаменатель).

Сценарий 12. Просмотр расписания по группам.

Преподаватель может выбрать конкретную группу из выпадающего списка и увидеть расписание занятий этой группы. Это удобно для отслеживания нагрузки по разным потокам.

Сценарий 13. Создание заявки на аудиторию.

Преподаватель переходит на страницу "Свободные аудитории", выбирает диапазон дат и времени. Система отображает список свободных помещений. Преподаватель выбирает подходящую аудиторию и создаёт заявку на бронирование, указав группу и дисциплину. Заявка отправляется администратору на рассмотрение.

Сценарий 14. Просмотр состояния заявки.

Преподаватель может отслеживать статус своих заявок (в ожидании, одобрена, отклонена) на соответствующей странице.

Сценарии для студента.

Сценарий 15. Вход в систему.

Студент вводит логин и пароль, система определяет роль "студент" и перенаправляет на страницу с расписанием его группы.

Сценарий 16. Просмотр расписания своей группы.

На главной странице студент видит расписание своей учебной группы. Система позволяет переключаться между типами недель - числителем и знаменателем. Занятия отображаются в формате календарной сетки с указанием дисциплины, преподавателя, аудитории и времени проведения.

Сценарий 17. Просмотр расписания на выбранную неделю.

Студент может выбрать нужную неделю по дате или номеру и просмотреть расписание только за этот период. После выбора система автоматически загружает соответствующие данные с сервера.

Сценарий 18. Обновление расписания.

Если администратор вносит изменения в расписание - например, переносит занятие или меняет аудиторию, - студент сразу видит актуальную информацию. Обновление интерфейса происходит динамически, без перезагрузки страницы.

Общий сценарий для всех ролей.

Сценарий 19. Смена пароля.

Любой авторизованный пользователь может открыть страницу смены пароля, указать текущий пароль, затем ввести новый и подтвердить его. Система проверяет правильность старого пароля и совпадение новых данных, после чего сохраняет изменения. При успешной смене пароля пользователь получает уведомление.

1.1.4 Выбор технологий и инструментов.

При разработке веб-системы управления расписанием одним из ключевых этапов стал выбор технологического стека, который способен обеспечить стабильную работу при высокой нагрузке, одновременном подключении большого числа пользователей и постоянном обновлении данных. Система должна работать в режиме реального времени и предоставлять удобный интерфейс сотрудникам учебного отдела, преподавателям и студентам. Поэтому при выборе технологий основное внимание уделялось производительности, гибкости архитектуры и удобству дальнейшего сопровождения [3].

В качестве серверной платформы выбран Python. Этот язык благодаря выразительному синтаксису, развитой экосистеме и поддержке асинхронного программирования хорошо подходит для реализации сложной логики генерации расписания и обработки большого количества параллельных запросов. В роли веб-фреймворка используется FastAPI. Он обеспечивает высокую скорость отклика, автоматически формирует документацию API и поддерживает удобную типизацию данных, что упрощает разработку и делает код более надёжным.

Для хранения данных выбрана PostgreSQL. СУБД позволяет эффективно работать с большим количеством связанных сущностей, поддерживать целостность данных и выполнять сложные запросы без заметного снижения производительности. Взаимодействие с базой данных реализовано через SQLAlchemy, что помогает поддерживать чистую архитектуру приложения, управлять моделями и адаптировать структуру данных под новые требования [7].

Отдельное внимание уделено инструментам автоматической генерации расписания. Алгоритмическое ядро разработано на Python и построено на матричной модели состояния. Такой подход позволяет отслеживать занятость преподавателей, учебных групп и аудиторий с учётом дней недели, временных слотов и типов недель. Система контролирует конфликты, минимизирует количество окон и помогает равномерно распределять нагрузку.

Для повышения качества генерации используется penalty-based система штрафов. Она подбирает оптимальные временные слоты с учётом сразу нескольких факторов: предпочтений преподавателей, временных ограничений, лимитов нагрузки и общей статистики использования расписания. Благодаря этому алгоритм остаётся гибким, адаптивным и легко расширяется при появлении новых требований.

Для автоматического разбора учебных документов и извлечения структурированных данных применяется Yandex GPT API.

1.1.5 Выбор серверной платформы.

При выборе серверной платформы для системы управления расписанием основное внимание уделялось производительности, удобству разработки и способности эффективно работать с асинхронными операциями. В качестве основы серверной части выбран FastAPI - современный Python-фреймворк, предназначенный для создания высоконагруженных веб-сервисов. Благодаря асинхронному стеку на базе Uvicorn и Starlette система может обрабатывать большое количество одновременных подключений без заметных задержек.

Одним из важных преимуществ FastAPI стала строгая типизация данных и автоматическая генерация документации API. Это упрощает разработку, ускоряет тестирование и делает дальнейшее сопровождение проекта более удобным.

Использование Python в качестве основного языка разработки открывает доступ к широкому набору библиотек для обработки данных, реализации алгоритмов генерации расписания и интеграции с внешними сервисами. Дополнительно платформа позволяет использовать инструменты машинного обучения и интеллектуального анализа данных.

FastAPI был выбран после сравнения с другими популярными серверными платформами. Такие фреймворки, как Django и Flask, обладают развитой экосистемой, однако в большей степени ориентированы на синхронную обработку запросов. Django предоставляет большой набор встроенных инструментов, но его монолитная архитектура усложняет создание лёгких и производительных API-сервисов, особенно в проектах с высокой долей асинхронных операций. Flask, наоборот, отличается гибкостью и минимализмом, однако требует больше времени на организацию архитектуры, настройку безопасности и документирование API. Это увеличивает сроки разработки и повышает вероятность ошибок [7].

Сравнение с платформами на других языках также подтвердило, что FastAPI подходит для проекта лучше всего. Node.js благодаря событийно-ориентированной модели показывает высокую производительность, однако его экосистема требует строгого контроля архитектуры и организации кода. Кроме того, работа с типами данных в Python с использованием Pydantic выглядит надёжнее и удобнее.

Java-фреймворки, включая Spring Boot, обеспечивают высокий уровень стабильности и хорошо масштабируются, но их внедрение связано с более сложной разработкой и повышенными требованиями к ресурсам. Для проекта, где важны гибкость и быстрые итерации, такой подход оказался избыточным. Go-фреймворки, например Gin или Fiber, отличаются высокой производительностью, однако требуют более низкоуровневой разработки. Это усложняет реализацию сложной бизнес-логики и интеграцию алгоритмических модулей.

В результате FastAPI стал наиболее подходящим решением для серверной части системы управления расписанием. Фреймворк сочетает высокую скорость работы, современную архитектуру, удобство разработки и возможности экосистемы Python. Такой подход позволяет эффективно реализовать как базовые API-функции, так и сложные алгоритмы генерации расписания. Платформа создаёт надёжную основу для дальнейшего развития системы, подключения новых модулей и адаптации под изменяющиеся требования образовательного процесса.

1.1.6 Выбор системы управления базами данных.

Для хранения данных системы управления расписанием выбрана PostgreSQL. Эта СУБД сочетает высокую производительность, надёжность и широкие возможности для работы со сложными структурами данных. В системе расписания необходимо хранить большое количество взаимосвязанных сущностей: учебные группы, преподавателей, аудитории, типы недель, занятия и историю изменений. В таких условиях особенно важны корректная обработка транзакций, поддержание целостности данных и эффективное выполнение запросов с несколькими уровнями связей.

PostgreSQL давно зарекомендовала себя как стабильная СУБД, способная выдерживать высокие нагрузки и сохранять предсказуемое время отклика даже при большом количестве параллельных операций. Система поддерживает разные типы индексов, работу с JSONB, пользовательские функции и расширения, что позволяет гибко адаптировать структуру данных под особенности алгоритмов генерации расписания.

Выбор PostgreSQL был сделан после анализа альтернативных решений. MySQL, несмотря на широкое распространение, уступает PostgreSQL в поддержке сложных транзакций и строгом соблюдении модели ACID. Для системы управления расписанием это критично, поскольку при одновременном редактировании данных несколькими пользователями нельзя допускать конфликтов или потери информации.

Дополнительно PostgreSQL показывает более стабильную работу со сложными запросами и большим количеством взаимосвязанных сущностей. Это особенно важно для образовательных систем, где структура данных со временем расширяется и усложняется. SQLite, хотя и проста в использовании, плохо подходит для многопользовательских веб-приложений. Она не рассчитана на высокий уровень параллелизма и теряет производительность при большом числе одновременных запросов.

Коммерческие решения, такие как Oracle Database и Microsoft SQL Server, предлагают широкий набор возможностей, однако требуют серьёзных затрат на лицензирование и администрирование. Для проекта, ориентированного на гибкость и экономичность, такой подход оказался неоправданным.

Сравнение с NoSQL-решениями также подтвердило целесообразность выбора реляционной модели. Документоориентированные базы данных, например MongoDB, хорошо подходят для слабоструктурированных данных. Однако в системе расписания ключевое значение имеют строгие связи между сущностями и выполнение сложных выборок. В таких условиях реляционная модель оказывается более логичной и надёжной.

PostgreSQL стала оптимальным решением для серверной части системы управления расписанием благодаря стабильной работе, строгой поддержке транзакций и высокой производительности. СУБД эффективно справляется с большим количеством взаимосвязанных сущностей и создаёт надёжную основу для дальнейшего расширения функциональности, масштабирования системы и интеграции с внешними сервисами. Для проекта такого класса это оказалось наиболее подходящим вариантом.

1.1.7 Выбор технологий клиентской части.

Для разработки клиентской части системы управления расписанием выбрана технология React - один из самых распространённых и зрелых инструментов для создания веб-интерфейсов. React позволяет строить интерфейс в виде набора независимых компонентов. Такой подход упрощает разработку, тестирование отдельных модулей и дальнейшую поддержку проекта.

Система работает с большим объёмом динамически изменяющихся данных: расписанием занятий, состоянием аудиторий, занятостью преподавателей. В этих условиях важным преимуществом React становится использование виртуального DOM. Технология обновляет только те элементы интерфейса, которые действительно изменились, благодаря чему приложение сохраняет высокую скорость работы и плавность взаимодействия даже при большой нагрузке [6].

React был выбран после сравнения с другими популярными решениями. Angular предлагает широкий набор встроенных инструментов, однако отличается более сложной архитектурой и жёсткой структурой проекта. Это делает его менее удобным в условиях, где требуется гибкость и возможность быстро менять архитектурные решения. Vue.js, наоборот, проще в освоении и использовании, но его экосистема уступает React по масштабу и количеству готовых решений.

Сравнение с альтернативными решениями вне экосистемы JavaScript также подтвердило оправданность выбора React. Использование серверной генерации интерфейсов или классических шаблонизаторов заметно ограничило бы интерактивность приложения и усложнило реализацию функций, связанных с мгновенным обновлением данных. Мобильные и гибридные технологии, например Flutter Web, пока не обеспечивают достаточной стабильности и производительности для сложных веб-интерфейсов, особенно при работе с таблицами, календарями и большим количеством интерактивных элементов.

React также удобно интегрируется с современными библиотеками визуализации расписания, включая FullCalendar. Это позволяет создавать наглядные и удобные представления учебного процесса. Дополнительным преимуществом стала поддержка современных инструментов управления состоянием - Redux, Zustand и React Query. С их помощью клиентская часть может эффективно синхронизироваться с сервером и корректно отображать изменения в режиме реального времени. Для учебных систем это особенно важно: расписание обновляется часто, и пользователи ожидают, что интерфейс будет реагировать на изменения сразу.

В результате React стал оптимальным решением для клиентской части системы. Технология сочетает гибкость, высокую производительность, развитую экосистему и удобство разработки. Она позволяет построить современный динамический интерфейс, который легко адаптируется под разные категории пользователей и поддерживает дальнейшее развитие проекта.

1.1.8 Формат обмена данными и взаимодействие компонентов.

Взаимодействие между клиентской и серверной частями системы управления расписанием реализовано с использованием формата JSON и протокола HTTP. Такой подход обеспечивает удобный обмен данными, упрощает интеграцию и сохраняет совместимость с внешними сервисами, которые могут быть подключены к проекту в дальнейшем.

JSON стал фактическим стандартом обмена данными в современных веб-приложениях благодаря компактному формату, удобству сериализации и понятной структуре. Он позволяет эффективно передавать данные о занятиях, преподавателях, аудиториях и параметрах расписания. Использование HTTP в качестве транспортного протокола обеспечивает стабильное и предсказуемое взаимодействие между компонентами системы, а также упрощает масштабирование и интеграцию с другими информационными платформами учебного заведения.

Клиентская часть приложения собирается с использованием Node.js, который применяется для управления зависимостями, сборки модулей и оптимизации фронтенда. Экосистема npm и поддержка современных инструментов сборки, таких как Webpack и Vite, позволяют ускорить разработку, удобно подключать сторонние библиотеки и гибко настраивать окружение проекта. Это особенно важно для систем, где интерфейс должен быстро реагировать на изменения данных, поступающих с сервера.

Для оформления интерфейса используются современные CSS-технологии, включая адаптивные библиотеки и препроцессоры. Такой подход помогает создать единый и понятный интерфейс, который корректно отображается как на компьютерах, так и на мобильных устройствах. Благодаря этому система остаётся доступной и удобной для студентов и преподавателей. Таблицы, интерактивные календари и панели управления могут обновляться динамически без перезагрузки страницы, что делает работу с расписанием быстрее и комфортнее.

Выбор формата обмена данными и инструментов взаимодействия компонентов связан с необходимостью обеспечить высокую производительность, надёжность и гибкость архитектуры. JSON и HTTP упрощают расширение функциональности системы, подключение новых модулей и интеграцию с внешними сервисами - электронными журналами, системами аутентификации и корпоративными порталами.

1.2 Проектирование и разработка

Проектирование системы управления расписанием направлено на создание надёжного, масштабируемого и удобного программного решения, способного поддерживать одновременную работу большого числа пользователей и корректно обрабатывать постоянно изменяющиеся данные учебного процесса. На этом этапе определяется архитектура приложения, выделяются основные компоненты системы, способы их взаимодействия и требования к структуре данных. Отдельное внимание уделяется согласованности информации. Расписание напрямую влияет на организацию учебного процесса, поэтому любые ошибки могут привести к сбоям в работе учебного заведения.

Архитектура проектируется с расчётом на дальнейшее развитие системы. Это позволяет расширять функциональность без переработки уже существующих модулей и адаптировать приложение под изменяющиеся требования образовательного учреждения.

Система построена по клиент-серверной архитектуре, что обеспечивает чёткое разделение ответственности между компонентами. Серверная часть отвечает за обработку запросов, генерацию расписания и взаимодействие с базой данных. Здесь выполняется проверка корректности данных, контроль конфликтов, применение ограничений для преподавателей и учебных групп, а также формирование итогового расписания.

Клиентская часть предоставляет интерфейс для сотрудников учебного отдела, преподавателей и студентов. Она отвечает за отображение расписания, визуализацию изменений в реальном времени и взаимодействие пользователя с системой. Такое разделение позволяет независимо развивать интерфейс и серверную логику, ускоряет внедрение новых функций и упрощает сопровождение проекта.

Взаимодействие между компонентами системы реализовано через REST-интерфейс. Такой подход обеспечивает стандартизированный обмен данными и упрощает интеграцию с внешними сервисами. Для образовательных учреждений это особенно важно, поскольку в их инфраструктуре обычно используются электронные журналы, системы учёта посещаемости, корпоративные порталы и сервисы аутентификации. REST-архитектура позволяет подключать такие решения без серьёзных изменений в структуре приложения, сохраняя гибкость и возможность дальнейшего расширения системы.

Проектирование также включает разработку модели данных, которая отражает структуру учебного процесса: учебные группы, преподавателей, дисциплины, аудитории, типы недель, временные слоты и правила формирования расписания. Модель создаётся с учётом требований к целостности данных и предотвращению конфликтов. Одновременно она должна обеспечивать высокую скорость выполнения запросов и стабильную работу системы.

Отдельное внимание уделяется алгоритмам генерации расписания. Система учитывает большое количество ограничений и предпочтений: занятость преподавателей, доступность аудиторий, распределение нагрузки и отсутствие пересечений занятий. При этом алгоритмы должны сохранять производительность даже при большом объёме данных и постоянных изменениях расписания.

Этап проектирования и разработки направлен на создание устойчивой и масштабируемой архитектуры, способной поддерживать сложные процессы формирования расписания и обеспечивать удобную работу для всех категорий пользователей. Такой подход позволяет системе развиваться, адаптироваться к новым требованиям и интегрироваться с внешними сервисами без потери надёжности и стабильности работы.

1.2.1 Проектирование базы данных.

Проектирование базы данных для системы управления расписанием направлено на обеспечение целостности, непротиворечивости и удобной структуры хранения учебной информации. На этапе разработки была создана ER-модель, которая отражает основные сущности образовательного процесса: учебные группы, преподавателей, дисциплины, аудитории, временные слоты, типы недель и занятия.

Между сущностями определены связи, соответствующие реальной логике формирования расписания. Одна учебная группа может иметь множество занятий, каждое занятие связано с конкретным преподавателем и аудиторией, а временные параметры задаются фиксированными слотами и типами недель. Использование реляционной модели позволяет контролировать целостность данных, применять механизмы валидации, внешние ключи и ограничения. Это особенно важно для предотвращения конфликтов в расписании и исключения дублирующейся информации.

Отдельное внимание уделялось оптимизации структуры таблиц и запросов. Система должна быстро обрабатывать данные даже при большом объёме информации и частых изменениях расписания. При проектировании учитывались требования алгоритма генерации расписания, который постоянно обращается к базе данных при работе с преподавателями, учебными группами, аудиториями и существующими занятиями. Поэтому структура была построена так, чтобы сократить количество сложных соединений и обеспечить эффективную индексацию наиболее используемых полей.

Точка входа серверной части расположена в основном модуле приложения. Он запускается первым и отвечает за инициализацию системы: настройку FastAPI-сервера, подключение маршрутов REST API, создание соединения с базой данных через SQLAlchemy и автоматическую генерацию таблиц при необходимости.

После инициализации сервер подготавливает сервисный слой и репозитории, обеспечивая им доступ к базе данных и механизмам выполнения SQL-операций. Таким образом, взаимодействие основных компонентов WEB-приложения начинается этого модуля, а база данных становится главным местом хранения, к которому обращаются остальные части системы — REST-запросы, внутренние сервисы и генератор расписания.

Ниже приведён пример структуры таблицы lessons, которая является одной из ключевых в системе, поскольку хранит информацию о конкретных занятиях:

```
lessons (
    id SERIAL PRIMARY KEY,
    group_id INTEGER REFERENCES groups(id),
    teacher_id INTEGER REFERENCES users(id),
    room_id INTEGER REFERENCES rooms(id),
    subject_raw VARCHAR(255) NOT NULL,
    weekday VARCHAR(20) NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    week_type INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
)
```

Эта таблица отражает связь между группой, преподавателем, аудиторией и временным интервалом, что позволяет однозначно определить каждое занятие и исключить пересечения.

Для демонстрации взаимодействия серверной части с базой данных ниже приведены примеры реальных SQL-запросов, которые формируются в процессе работы системы. После запуска основного модуля приложение инициализирует веб-сервер, подключает маршруты REST-API и устанавливает соединение с PostgreSQL. Каждый REST-запрос, поступающий от клиента, проходит через сервисный слой и приводит к выполнению конкретной SQL-операции.

Примеры SQL-запросов и взаимодействия с базой данных.

1. Получение списка занятий группы на выбранную неделю:

```
SELECT *
FROM lessons
WHERE group_id = :group_id
AND week_type = :week_type
ORDER BY weekday, start_time;
```

2. Создание нового занятия (используется при генерации расписания):

```
INSERT INTO lessons (group_id, teacher_id, room_id, subject_raw
weekday, start_time, end_time, week_type)
 VALUES (:group_id, :teacher_id, :room_id, :subject_raw,
:weekday, :start_time, :end_time, :week_type);
```

3. Проверка занятости преподавателя в конкретный слот:

```
SELECT id
FROM lessons
WHERE teacher_id = :teacher_id
AND weekday = :weekday
AND start_time = :start_time
AND week_type = :week_type
LIMIT 1;
```

4. Получение всех занятий преподавателя для отображения в личном кабинете:

```
SELECT *
FROM lessons
WHERE teacher_id = :teacher_id
ORDER BY weekday, start_time;
```

На рисунке 5 изображена ER-диаграмма для БД

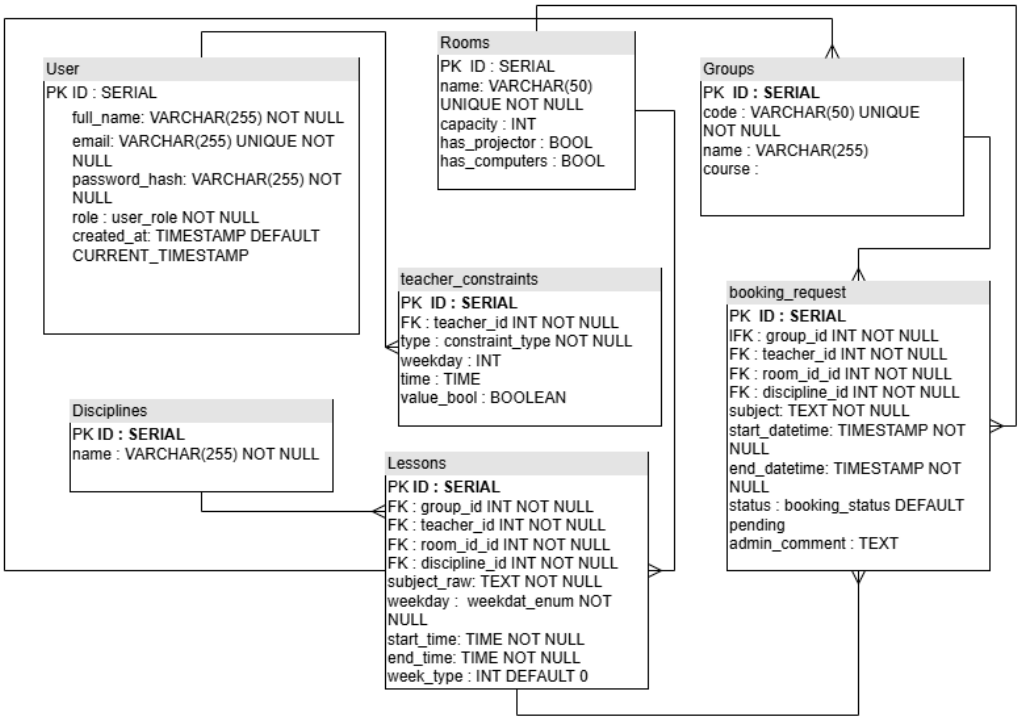


Рисунок 5 – ER-диаграмма БД

ER-диаграмма отражает структуру базы данных системы автоматизации расписания и показывает связи между её основными сущностями. В модели представлены таблицы пользователей, дисциплин, аудиторий, учебных групп, ограничений преподавателей, занятий и заявок на аудитории.

Каждая сущность содержит набор атрибутов с указанием типов данных и ключевых ограничений. Связи между таблицами реализованы через внешние ключи, что позволяет поддерживать целостность данных и предотвращать появление несогласованных записей.

Диаграмма показывает, как пользователи, группы, аудитории и дисциплины объединяются в структуре занятий, каким образом формируются заявки на аудитории и как система учитывает ограничения преподавателей при составлении расписания. Такая организация данных обеспечивает понятную структуру базы и поддерживает стабильную работу всех функций системы.

Проект уделяет большое внимание визуальному представлению расписания. Занятия отображаются в формате календарной сетки - такой подход помогает быстрее ориентироваться в информации и воспринимать её привычным способом. Дисциплины выделены разными цветами, временные слоты структурированы понятно и без перегруженности, а адаптивная верстка позволяет одинаково удобно работать как с компьютера, так и со смартфона. Интерфейс поддерживает динамическое обновление данных: если сотрудники учебного отдела вносят изменения, преподаватели и студенты видят их сразу. Это снижает риск ошибок и помогает поддерживать расписание в актуальном состоянии.

Компонентный подход на базе React сделал интерфейс модульным. Таблицы, карточки занятий, календарные блоки и формы редактирования работают как независимые компоненты, которые можно развивать отдельно друг от друга. За счёт этого интерфейс выглядит единообразно, тестирование занимает меньше времени, а добавление новых функций не требует переработки уже существующих модулей. Такая архитектура упрощает масштабирование системы: можно без серьёзных изменений внедрять новые типы занятий, расширять инструменты генерации расписания или подключать внешние сервисы.

Проектирование интерфейса ориентировано на создание удобной и гибкой среды для всех участников учебного процесса. Продуманная визуальная организация данных, модульная структура и учёт потребностей разных категорий пользователей делают систему надёжным инструментом для работы с расписанием и помогают повысить эффективность образовательного процесса.

1.2.2 Проектирование пользовательского интерфейса.

Проектирование интерфейса системы управления расписанием было ориентировано на создание удобной и понятной среды, в которой разные категории пользователей могут быстро решать свои задачи. При разработке особое внимание уделили доступности, визуальной простоте и сокращению количества лишних действий. Это особенно важно для системы, с которой ежедневно работают сотрудники учебного отдела, преподаватели и студенты. Основная цель заключалась в том, чтобы представить сложную структуру учебного процесса в наглядном формате: пользователь должен быстро находить нужную информацию, ориентироваться в расписании и выполнять необходимые операции без постоянных переходов между разделами.

Интерфейс разделён на функциональные модули с учётом ролей пользователей. Сотрудники учебного отдела получают доступ к инструментам импорта и генерации расписания, управлению дисциплинами, аудиториями, преподавателями и учебными группами. Для них предусмотрены расширенные панели управления, где можно отслеживать конфликты в расписании, анализировать нагрузку и вносить изменения в режиме реального времени.

Преподаватели работают со своим расписанием: просматривают занятия, аудитории, типы недель, а также получают доступ к расписанию учебных групп, с которыми взаимодействуют. Студентам доступен отдельный интерфейс для просмотра расписания с фильтрацией по дням, неделям и типам занятий. Такой подход делает доступ к данным более персонализированным и помогает снизить когнитивную нагрузку, поскольку каждый пользователь видит только те инструменты и сведения, которые нужны ему для работы.

При разработке интерфейса особое внимание уделили визуальному представлению расписания. Занятия отображаются в формате календарной сетки, поэтому пользователи воспринимают информацию привычным и понятным способом. Цветовое разделение дисциплин помогает быстрее ориентироваться в расписании, а чёткая структура временных слотов снижает вероятность ошибок при просмотре. Интерфейс корректно работает как на настольных компьютерах, так и на мобильных устройствах благодаря адаптивной верстке.

Система поддерживает динамическое обновление данных. Если сотрудники учебного отдела вносят изменения в расписание, преподаватели и студенты видят их сразу после обновления. Это помогает поддерживать информацию актуальной и уменьшает количество несоответствий в расписании.

Компонентный подход на базе React позволил выстроить модульную архитектуру интерфейса. Таблицы, карточки занятий, календарные блоки и формы редактирования реализованы как независимые компоненты. Такой подход упрощает тестирование, поддерживает единый стиль отображения и позволяет быстрее расширять функциональность системы без переработки уже существующих модулей.

За счёт модульной структуры интерфейс остаётся гибким и легко адаптируется под новые требования. В систему можно добавлять новые типы занятий, расширять возможности генерации расписания или подключать внешние сервисы без серьёзных изменений базовой архитектуры.

Проектирование интерфейса было направлено на создание удобной и функциональной среды для всех участников учебного процесса. Продуманная визуальная организация данных, адаптивность и ориентация на разные пользовательские роли делают систему надёжным инструментом для работы с расписанием и помогают повысить эффективность образовательного процесса.

На рисунке 6 представлена страница со входом в аккаунт.

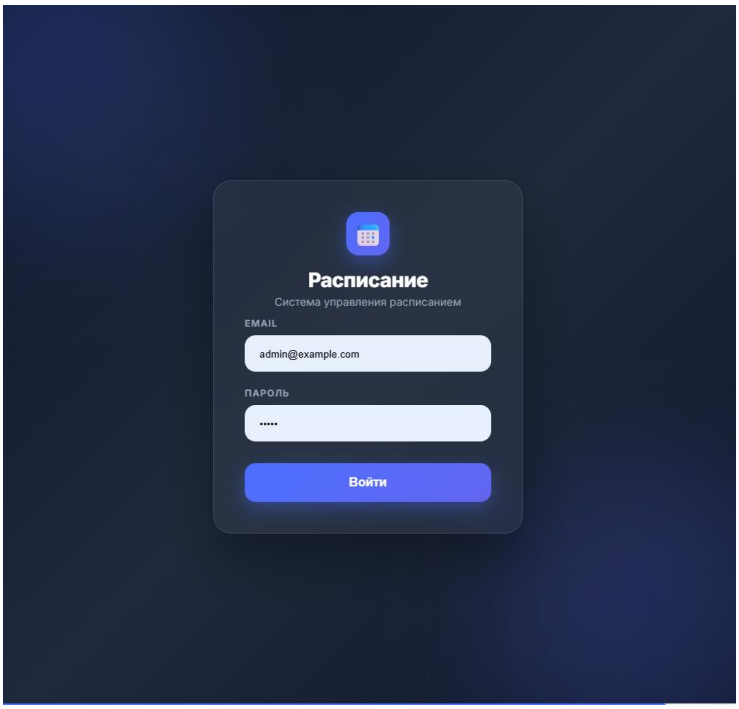


Рисунок 6 – Страница входа

После ввода данных пользователь попадает на страницу «Дашборд», на который выведена основная статистика платформы. Страница изображена на картине 7.

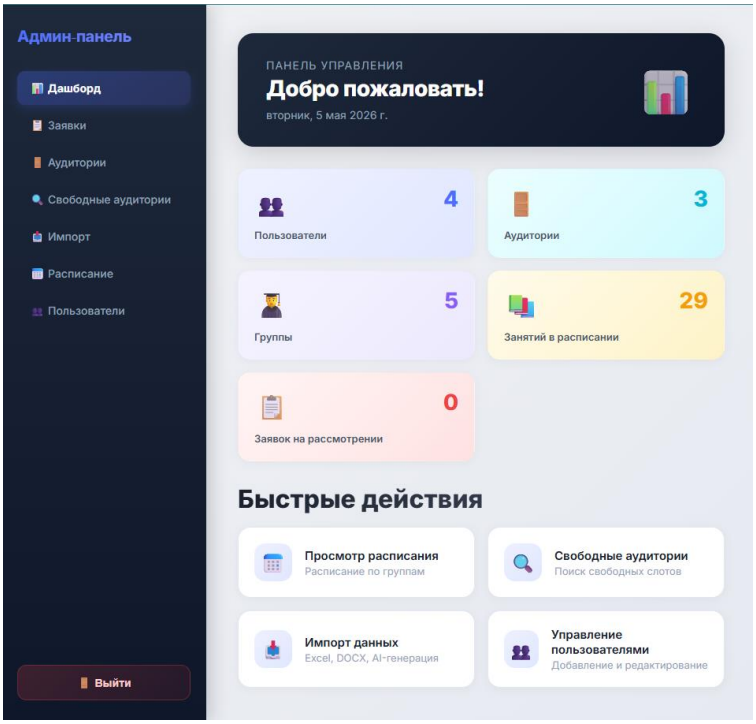


Рисунок 7 – страница «Дашборд»

Перейдя на страницу «Заявки» администратор видит список всех заявок на бронирование аудитории от преподавателей. Внешний вид представлен на рисунке 8. На странице «Аудитории» можно посмотреть список всех аудиторий, а для администратора добавить новые или отредактировать созданные. На рисунках 9 и 10 можно посмотреть внешний вид страницы.

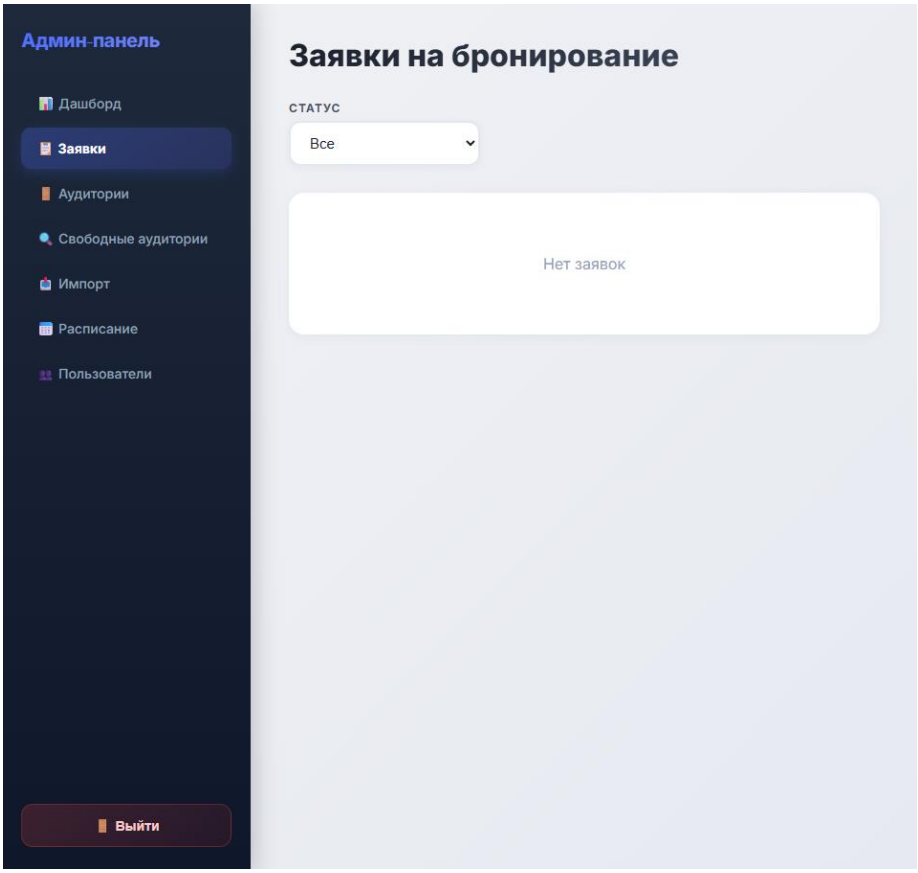


Рисунок 8 – страница «Заявки»

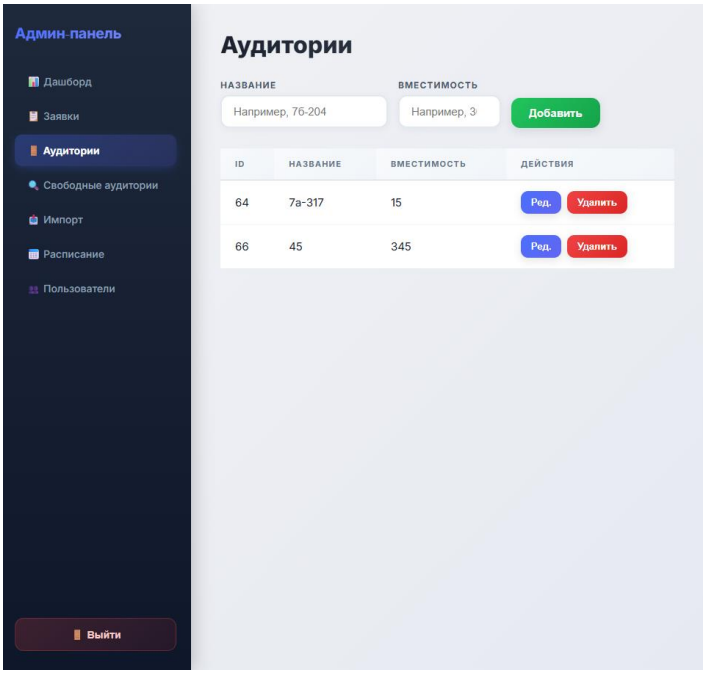


Рисунок 9 – страница «Аудитории»

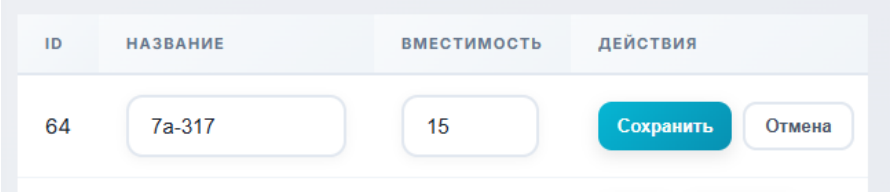


Рисунок 10– редактирование аудитории

На странице «Свободные аудитории» преподаватель и администратор могут ознакомиться с количеством свободных аудиторий в выбранном диапазоне времени. На рисунке 11 представлен внешний вид страницы.

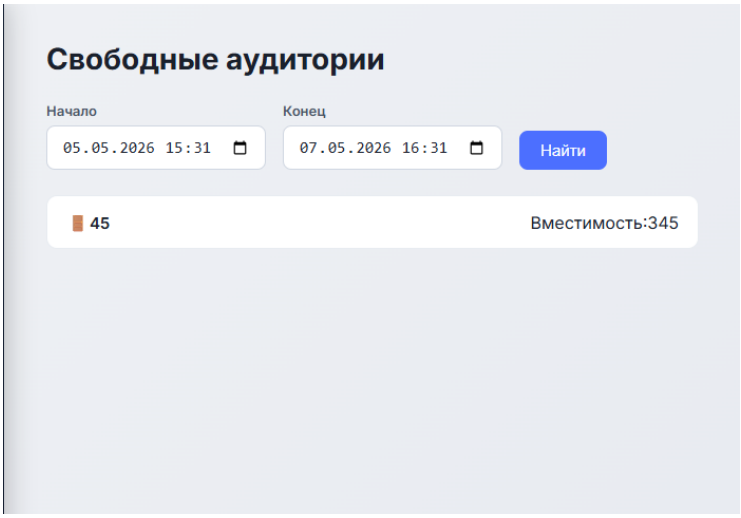


Рисунок 11– Свободные аудитории

На странице импорта администратор может импортировать расписание из .xlsx файла, из .docx файла назначений и сгенерировать расписание с помощью YandexGPT. На рисунке 12 представлена данная вкладка.

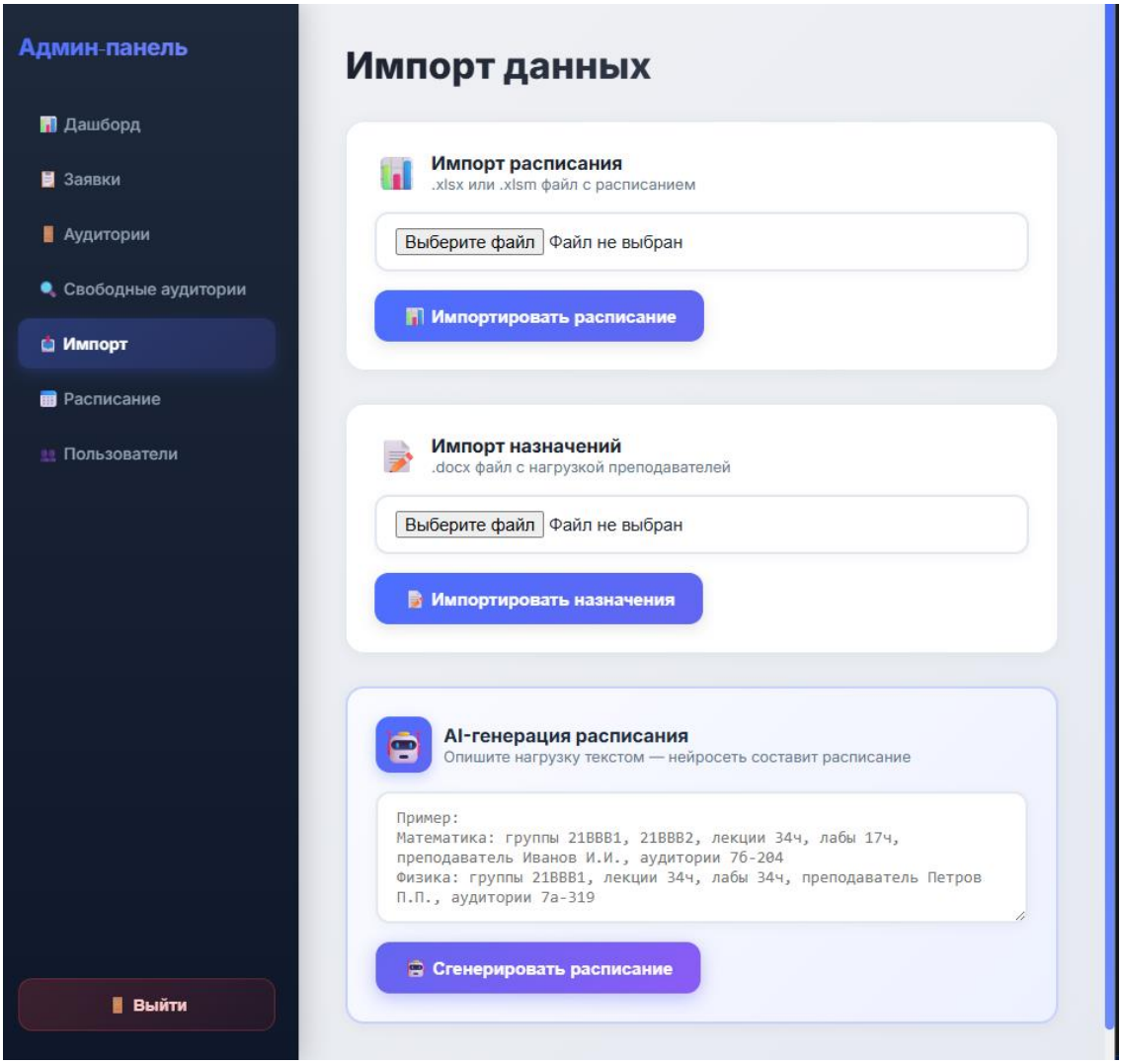


Рисунок 12– Вкладка «Импорт»

Страница расписание представляет собой таблицу, которую можно отсортировать по группе и недели. Администратор видит полное расписание, студент только для своей группы, а преподаватель свое расписание по группам.

На рисунке 13 внешний вид «Расписание»

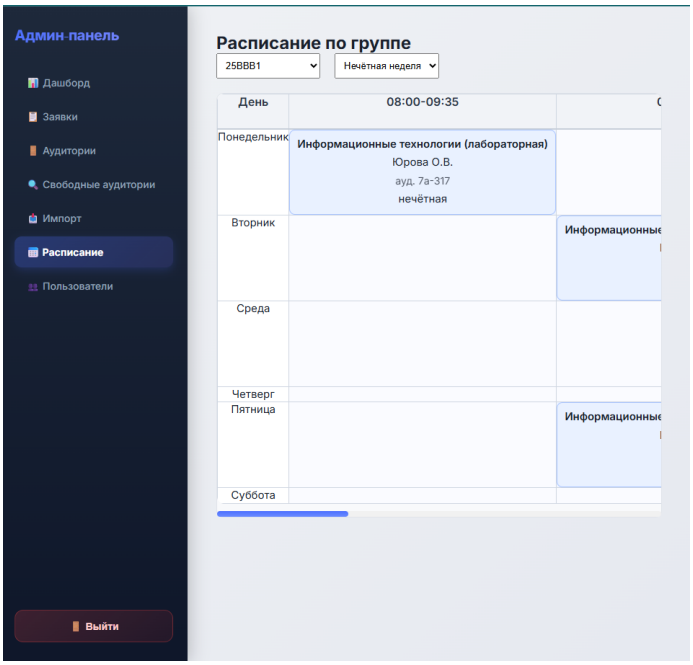


Рисунок 13– Вкладка «Расписание»

Страница «Пользователи» нужна администратору чтобы создавать аккаунты преподавателям и студентам, а также редактировать созданные. Внешний вид на рисунке 14.

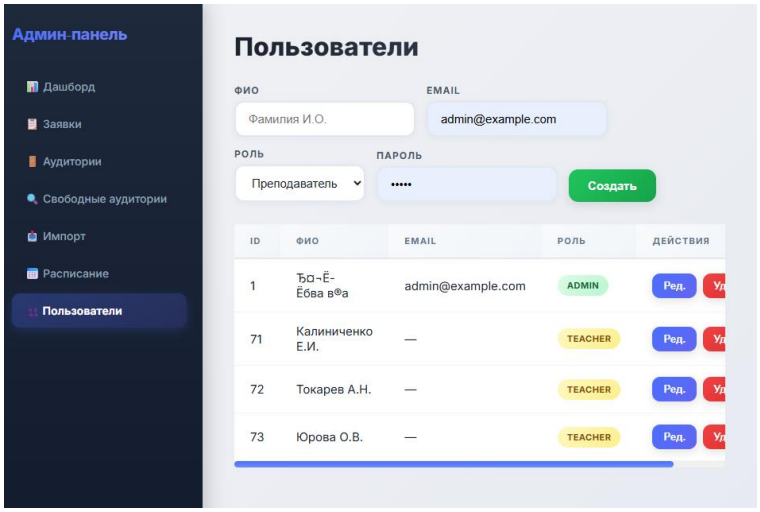


Рисунок 14– Вкладка «Пользователи»

1.2.3 Разработка серверной части.

Серверная часть системы управления расписанием разработана на базе FastAPI. Этот фреймворк позволил создать производительный и удобный для масштабирования API с чёткой структурой. За счёт асинхронной архитектуры сервер способен одновременно обрабатывать большое количество запросов, что особенно важно для системы, где параллельно работают сотрудники учебного отдела, преподаватели и студенты.

На серверной стороне сосредоточена основная логика приложения. Система обрабатывает запросы пользователей, управляет учётными записями, формирует расписание, проверяет конфликты в базе данных и работает с дисциплинами, аудиториями и учебными группами. Каждый функциональный блок вынесен в отдельный модуль. Это упрощает поддержку проекта, делает архитектуру более понятной и позволяет развивать систему без серьёзной переработки существующего кода.

Для взаимодействия с базой данных используется ORM-слой на основе SQLAlchemy. Такой подход позволяет работать с данными через Python-объекты вместо ручного написания SQL-запросов. ORM автоматически формирует запросы, поддерживает строгую типизацию и делает код более читаемым. В модели данных отражены связи между преподавателями, группами, аудиториями и занятиями, поэтому сервер может эффективно выполнять выборку, обновление и удаление информации.

При этом система не ограничивается только ORM. В отдельных сценариях используются "сырые" SQL-запросы - например, при массовой генерации расписания, где критична производительность и скорость обработки большого объёма данных.

Отдельное внимание уделено аутентификации и авторизации пользователей. Для защиты данных применяется токенная система доступа, основанная на современных криптографических механизмах. Каждый пользователь - сотрудник учебного отдела, преподаватель или студент - получает уникальный токен.

Такой подход обеспечивает надёжный контроль доступа и защищает систему от несанкционированного изменения расписания. Права пользователей разделены по ролям. Сотрудники учебного отдела могут редактировать и генерировать расписание, преподаватели получают доступ к информации о своих занятиях, а студенты просматривают расписание своей группы без возможности внесения изменений.

В серверную часть также входит модуль генерации расписания. Он выполняет вычисления, связанные с распределением занятий по дням, аудиториям и временным слотам. При построении расписания алгоритм учитывает сразу несколько параметров: занятость преподавателей и учебных групп, ограничения аудиторий, типы недель, допустимую нагрузку и внутренние правила формирования расписания. Дополнительно система может учитывать предпочтения преподавателей, что помогает сделать расписание более гибким и удобным для участников учебного процесса.

Для повышения производительности генератор вынесен в отдельный сервисный слой. Благодаря этому вычисления выполняются независимо от обработки пользовательских запросов, а сама система продолжает стабильно работать даже при сложной генерации расписания и высокой нагрузке.

Серверная часть выступает основой всей системы управления расписанием. Она объединяет производительный API, модульную архитектуру, механизмы аутентификации и авторизации, ORM-слой для работы с базой данных и алгоритм автоматической генерации расписания.

Пример серверного кода:

```
@app.get("/teachers/{teacher_id}/busy")
def check_teacher_busy(
  teacher_id: int,
  weekday: str,
  slot_index: int,
  week_type: int,
  db: Session = Depends(get_db)
 ):
```

```
start_time, end_time = TIME_SLOTS[slot_index]

lesson = (
    db.query(Lesson)
    .filter(
        Lesson.teacher_id == teacher_id,
        Lesson.weekday == weekday,
        Lesson.start_time == start_time,
        Lesson.week_type == week_type,
    )
    .first()
)
 return {"busy": lesson is not None}
```

Приведённый фрагмент демонстрирует работу одного из серверных методов, реализованных с использованием фреймворка FastAPI. Данный метод отвечает за проверку занятости преподавателя в конкретный день, временной слот и тип недели. Подобная проверка является ключевым элементом бизнес-логики системы управления расписанием, поскольку позволяет предотвратить конфликты при создании или редактировании занятий.

Маршрут объявлен с помощью декоратора @app.get, что делает его доступным по HTTP-методу GET. Он принимает параметры teacher_id, weekday, slot_index и week_type, которые передаются клиентской частью при обращении к API. Для взаимодействия с базой данных используется зависимость db: Session = Depends(get_db), обеспечивающая автоматическое создание и закрытие сессии SQLAlchemy.

Внутри функции выполняется преобразование индекса временного слота в реальные значения времени начала и окончания занятия. Это достигается за счёт обращения к структуре TIME_SLOTS, содержащей фиксированные интервалы, используемые во всей системе. Далее формируется SQL-запрос через ORM-модель Lesson, который проверяет наличие занятия у указанного преподавателя в заданный день и временной интервал.

На рисунке 15 изображена диаграмма последовательности для сценария бронирования аудитории преподавателем.

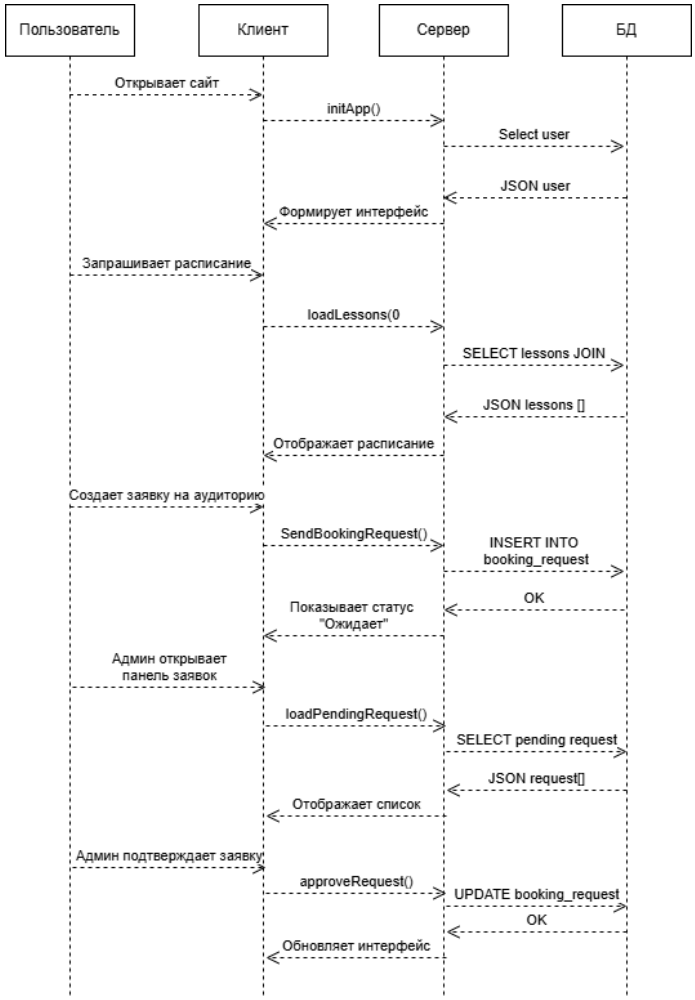


Рисунок 15 – UML-диаграмма последовательности для бронирования аудитории

Представленная диаграмма последовательности отражает процесс взаимодействия пользователя с системой автоматизации расписания. В ней показано, как клиентское приложение инициирует загрузку данных при открытии сайта, запрашивает у сервера информацию о пользователе и расписании, а затем отображает её в интерфейсе. Далее демонстрируется процесс создания заявки на аудиторию: клиент отправляет данные на сервер, который выполняет соответствующие операции в базе данных и возвращает результат. В завершение показано, как администратор обрабатывает поступившие заявки, а система обновляет интерфейс после изменения их статуса.

1.2.4 Разработка клиентской части.

Клиентская часть системы управления расписанием разработана с использованием React. Это позволило создать быстрый и отзывчивый интерфейс, который удобно использовать разным категориям пользователей. Компонентная архитектура React упростила разделение интерфейса на независимые модули, благодаря чему элементы можно повторно использовать и без лишних сложностей расширять функциональность системы. Такой подход особенно важен для проектов, которые постоянно развиваются и со временем получают новые возможности.

В ходе разработки были реализованы отдельные интерфейсные модули для сотрудников учебного отдела, преподавателей и студентов. Каждый из них решает свои задачи. Сотрудники учебного отдела работают с инструментами генерации расписания и анализа нагрузки, преподаватели просматривают занятия и расписание групп, а студенты получают доступ к расписанию по дням и неделям в удобном формате.

Отдельное внимание уделено взаимодействию клиентской части с сервером. Для обмена данными используется REST API, а HTTP-запросы выполняются через библиотеку axios. Это упростило интеграцию между клиентом и сервером и позволило удобно обрабатывать ответы и ошибки. Интерфейс поддерживает динамическое обновление данных: изменения, внесённые на сервере, сразу отображаются у пользователей без необходимости вручную обновлять страницу. Такой подход помогает поддерживать актуальность расписания и снижает вероятность ошибок при работе с системой.

Ниже приведён пример клиентского запроса, который используется для создания нового занятия в расписании. Он показывает, как React-компонент взаимодействует с сервером и обрабатывает полученный результат:

```
axios.post("/api/lessons/create", {
group_id: selectedGroup,
teacher_id: selectedTeacher,
room_id: selectedRoom,
weekday: selectedDay,
slot_index: selectedSlot,
week_type: selectedWeekType
}).then(response => {
setMessage("Занятие успешно добавлено");
}).catch(() => {
setMessage("Ошибка: слот занят или данные некорректны");
});
```

Этот пример отражает основную логику клиентской части. Формирование запроса дальнейшая отправка данных на сервер, обработка ответа с сервера и в конце вывод результата запроса пользователю.

Для визуализации данных используются функциональные компоненты React. Ниже приведён пример компонента, отображающего карточку занятия в расписании:

```
function LessonCard({ lesson }) {
    return (
        <div className="lesson-card">
            <h3>{lesson.subject_raw}</h3>
            <p>Преподаватель: {lesson.teacherName}</p>
            <p>Аудитория: {lesson.roomName}</p>
            <p>{lesson.weekday}, {lesson.start_time} –
{lesson.end_time}</p>
        </div>
    );
}
```

На рисунке 16 представлена диаграмма классов для клиента.

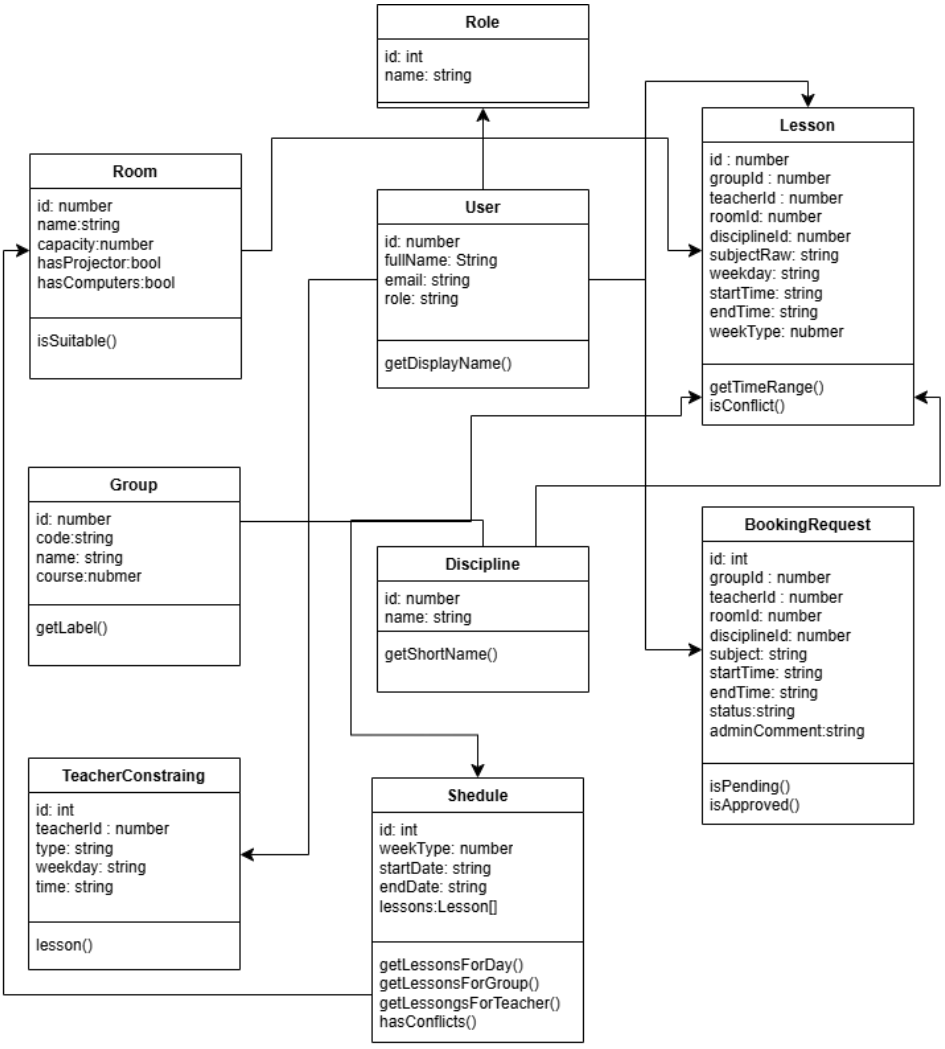


Рисунок 16 – UML-диаграмма классов

Диаграмма классов клиентской части отражает структуру основных сущностей, используемых интерфейсом системы автоматизации расписания. В ней представлены модели данных, такие как User, Lesson, Room, Group, Discipline, BookingRequest, TeacherConstraint и Schedule, каждая из которых содержит набор атрибутов с типами данных и методы для обработки соответствующей логики.

Компонент демонстрирует подход к построению интерфейса: данные, полученные с сервера, передаются в компонент в виде свойств, после чего отображаются в структурированном и визуально понятном виде. Такой подход обеспечивает гибкость, простоту тестирования и возможность повторного использования элементов в различных частях приложения.

Таким образом, клиентская часть системы представляет собой современный, адаптивный и удобный интерфейс, построенный на основе компонентной архитектуры React. Она обеспечивает эффективное взаимодействие с сервером, корректную визуализацию данных и удобство работы для всех категорий пользователей, что делает систему надёжным инструментом для управления учебным расписанием.

1.2.5 Интеграция и взаимодействие компонентов.

Интеграция клиентской и серверной частей системы управления расписанием основана на стандартизированном обмене данными через HTTP-запросы. Такой подход обеспечивает независимость компонентов, позволяет развивать их параллельно и упрощает последующую модернизацию системы. Все данные передаются в формате JSON, который является универсальным, легко читаемым и поддерживается большинством современных веб-технологий. Благодаря этому клиентская часть может быстро обрабатывать ответы сервера, обновлять интерфейс и обеспечивать пользователю актуальную информацию о расписании.

На этапе интеграции проводилась проверка корректности взаимодействия между компонентами, включая тестирование основных сценариев: получение расписания, создание занятий, проверка занятости преподавателей и аудиторий, обновление данных и обработка ошибок. Особое внимание уделено согласованности данных, поскольку расписание является критически важным элементом учебного процесса, и любые несоответствия могут привести к конфликтам или нарушению учебного графика.

Для повышения устойчивости системы реализованы механизмы обработки исключительных ситуаций: сервер возвращает структурированные сообщения об ошибках, а клиентская часть корректно отображает их пользователю, предотвращая некорректные действия и обеспечивая стабильность работы даже при временных сбоях.

Ниже приведён пример JSON-ответа, который сервер возвращает после успешного создания нового занятия. Такой формат используется для подтверждения выполнения операции и передачи клиенту идентификатора созданной записи:


```
{
  "status": "success",
  "message": "Занятие успешно создано",
  "lesson": {
    "id": 128,
    "group": {
      "id": 12,
      "name": "ИС-31"
    },
    "teacher": {
      "id": 45,
      "full_name": "Иванов Пётр Сергеевич"
    },
    "room": {
      "id": 7,
      "name": "Аудитория 204"
    },
    "subject_raw": "Программирование",
    "weekday": "Понедельник",
    "start_time": "10:30",
    "end_time": "12:00",
    "week_type": 1
  },
```

```
  "meta": {
      "created_at": "2024-02-15T14:22:10",
      "server": "schedule-api-v1.4",
      "processing_time_ms": 18
    }
  }
```

Этот ответ позволяет клиентской части обновить интерфейс, отобразить уведомление пользователю и при необходимости выполнить дополнительные действия, например перезагрузить расписание или подсветить новое занятие в календаре.

Таким образом, взаимодействие компонентов системы построено на принципах надёжности, предсказуемости и стандартизации. Использование JSON и HTTP обеспечивает гибкость интеграции, а продуманная обработка ошибок делает систему устойчивой к некорректным данным и внешним сбоям. Такой подход гарантирует согласованность работы всех модулей и создаёт основу для дальнейшего расширения функциональности без нарушения существующей архитектуры.

На рисунке 17 представлена диаграмма последовательности для создания расписания.

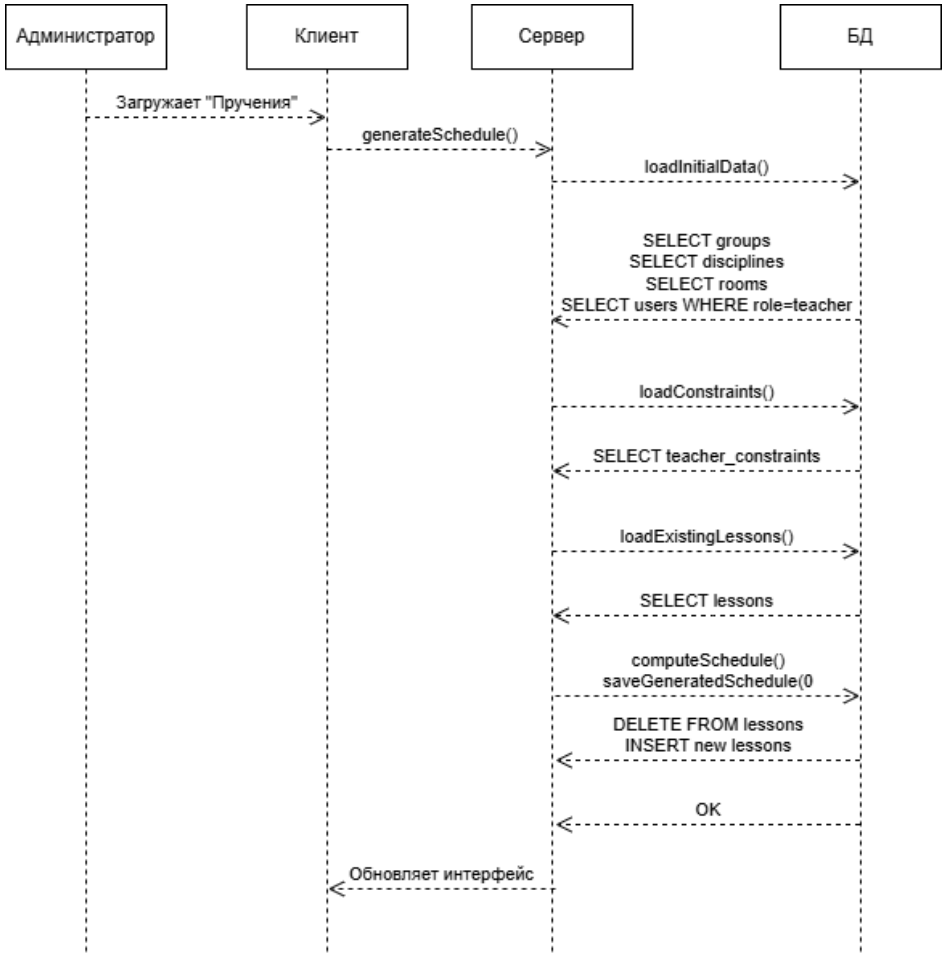


Рисунок 17 – UML-диаграмма последовательности для создания расписания

1.2.6 Обеспечение безопасности.

Безопасность является одним из ключевых аспектов разработки системы управления расписанием, поскольку приложение работает с персональными данными пользователей и внутренней информацией образовательного учреждения. На этапе проектирования были внедрены механизмы защиты, направленные на предотвращение несанкционированного доступа, обеспечение целостности данных и защиту от распространённых атак. Важным элементом архитектуры является разграничение прав доступа: сотрудники учебного отдела, преподаватели и студенты имеют различные уровни привилегий.

Для аутентификации используется механизм JWT-токенов, позволяющий безопасно идентифицировать пользователя при каждом запросе. Токены подписываются секретным ключом и содержат минимальный набор данных, необходимый для проверки прав доступа. Пароли хранятся в базе данных только в зашифрованном виде с использованием современных криптографических алгоритмов, что исключает возможность их восстановления даже при компрометации хранилища. Все входящие данные проходят строгую валидацию на уровне серверной части, что предотвращает SQL-инъекции, XSS-атаки и другие попытки некорректного воздействия на систему.

Особое внимание уделено обработке ошибок и исключительных ситуаций. Сервер возвращает структурированные сообщения об ошибках, позволяющие клиентской части корректно реагировать на сбои, а также предотвращающие раскрытие внутренней информации о системе. Все операции, связанные с изменением расписания, фиксируются в журнале действий, что обеспечивает возможность аудита и контроля за корректностью работы пользователей. Передача данных между клиентом и сервером осуществляется по защищённым каналам, что исключает возможность перехвата или подмены информации.

Ниже приведён пример генерации JWT-токена, используемого в системе:

```
def create_access_token(payload: dict) -> str:
payload["exp"] = datetime.utcnow() + timedelta(hours=6)
return jwt.encode(payload, SECRET_KEY, algorithm="HS256")
```

В данном примере токен содержит срок действия, что предотвращает использование устаревших или украденных токенов.

Также приведён пример middleware-проверки прав доступа, адаптированный под роли нашего проекта:

```
    def check_access(user, required_role: str):
        if user.role != required_role:
            raise HTTPException(status_code=403,
detail="Недостаточно прав для выполнения операции")
```

Этот фрагмент демонстрирует базовый принцип ролевой модели: преподаватель не может изменять расписание, студент не может редактировать данные групп, а доступ к административным функциям ограничен только сотрудниками учебного отдела.

Таким образом, система безопасности включает комплекс мер, направленных на защиту данных, контроль доступа, предотвращение атак и обеспечение стабильной работы приложения.

1.2.7 Динамика работы приложения.

Динамика работы приложения отражает последовательность действий, которые выполняются системой при взаимодействии пользователей с различными функциональными модулями. Работа строится на клиент-серверной архитектуре, где клиентская часть инициирует запросы, а серверная часть обрабатывает их, взаимодействует с базой данных и возвращает структурированные ответы. Такое разделение обеспечивает высокую производительность, чёткую организацию логики и возможность независимого масштабирования каждого из уровней.

При обращении пользователя к системе клиентская часть формирует HTTP-запрос, содержащий необходимые параметры: идентификаторы групп, преподавателей, временных слотов или тип недели. Серверная часть принимает запрос, выполняет валидацию входящих данных, проверяет права доступа пользователя и инициирует выполнение соответствующего сценария. После обработки данных сервер формирует ответ в формате JSON, который клиентская часть интерпретирует и отображает в интерфейсе. Благодаря этому пользователь получает актуальную информацию в режиме реального времени, а изменения в расписании становятся видимыми сразу после их внесения.

Процесс авторизации начинается с ввода логина и пароля. Клиентская часть отправляет запрос на сервер, где выполняется проверка корректности данных и соответствие пользователя определённой роли. В случае успешной авторизации сервер генерирует JWT-токен, содержащий информацию о пользователе, его идентификаторе и уровне доступа. Токен передаётся клиенту и используется для всех последующих запросов, что позволяет системе безопасно идентифицировать пользователя без повторного ввода данных.

При создании нового занятия сотрудник учебного отдела выбирает группу, преподавателя, аудиторию, день недели и временной слот. Клиентская часть отправляет сформированный запрос на сервер, где выполняется проверка занятости преподавателя, группы и аудитории. Если выбранный интервал свободен, сервер создаёт запись в базе данных и возвращает подтверждение в виде JSON-ответа. Клиентская часть обновляет интерфейс, отображая новое занятие в сетке расписания.

Преподаватель, открывая своё расписание, инициирует запрос на получение всех занятий, связанных с его идентификатором. Сервер формирует выборку из базы данных, сортирует данные по дням недели и временным слотам и возвращает их в структурированном виде. Клиентская часть отображает расписание в удобной визуальной форме, позволяя преподавателю быстро ориентироваться в своих занятиях.

Студент взаимодействует с системой аналогичным образом: при выборе своей группы клиентская часть отправляет запрос на получение расписания, после чего сервер возвращает данные, соответствующие выбранной неделе. Интерфейс отображает занятия в календарной сетке, обеспечивая удобный доступ к информации.

Таким образом, динамика работы приложения основана на последовательном обмене данными между клиентом и сервером, строгой валидации запросов, контроле доступа и оперативном обновлении интерфейса.

На рисунке 23 изображена диаграмма состояний для записи на прием.

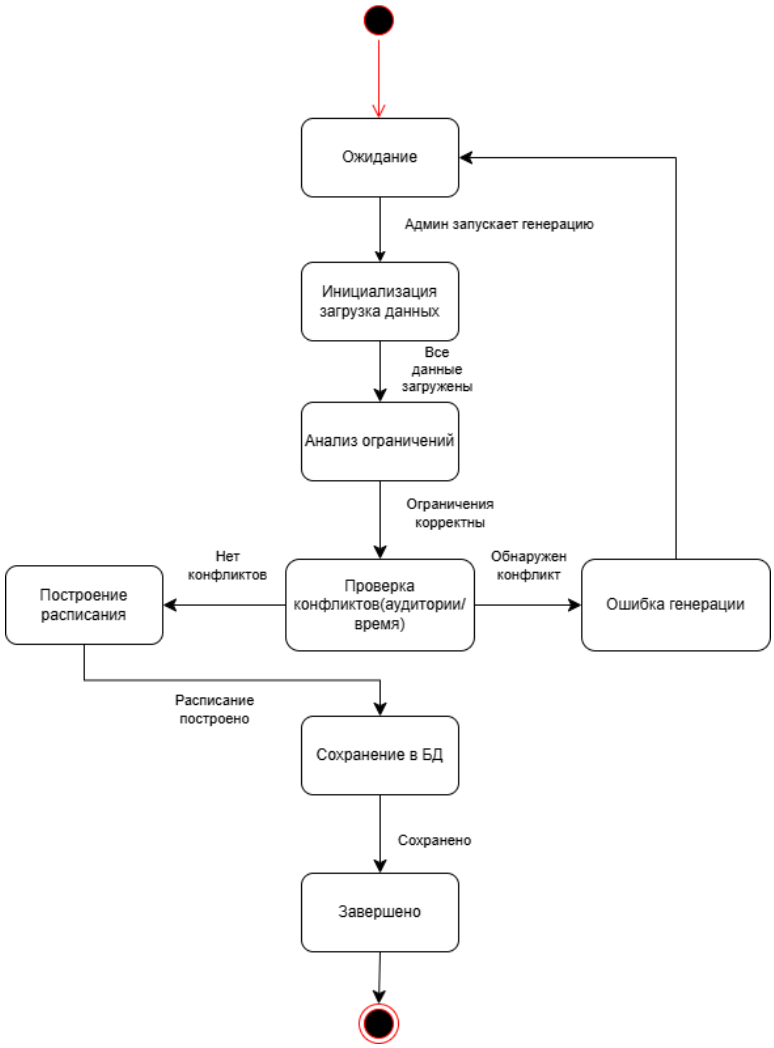


Рисунок 23 – UML-диаграмма состояний

1.2.8 Тестирование приложения.

Проверка работы разработанной системы управления расписанием проводилась с целью убедиться, что все её компоненты функционируют корректно, без ошибок и соответствуют требованиям, которые были определены на этапе проектирования. Тестирование позволило оценить надёжность обработки данных учебного процесса, стабильность работы под нагрузкой и удобство взаимодействия пользователей с интерфейсом. Кроме того, особое внимание уделялось тому, насколько согласованно работают клиентская и серверная части, а также соблюдаются ли правила безопасности и целостности данных.

Для оценки работоспособности системы использовался комплексный подход, включающий несколько уровней тестирования. Модульные тесты были сосредоточены на проверке отдельных функций серверной части: валидации входных данных, обработке запросов, выполнении операций с базой данных и логике определения конфликтов расписания. Такой подход помог выявить ошибки на ранних этапах и проверить корректность работы ключевых компонентов системы.

Интеграционное тестирование позволило убедиться, что разные модули приложения корректно взаимодействуют между собой. Проверялась передача данных через REST API, обработка ответов на стороне клиента и согласованное отображение изменений в расписании. Отдельное внимание уделялось стабильности обмена данными между серверной и клиентской частью.

Функциональное тестирование охватывало основные пользовательские сценарии. Для сотрудников учебного отдела проверялись создание и редактирование занятий, обработка конфликтов в расписании, корректность отображения данных и стабильность работы инструментов генерации. Для преподавателей тестировались просмотр личного расписания, переключение между неделями, отображение аудиторий и временных интервалов. В сценариях для студентов проверялись доступ к расписанию своей группы, фильтрация по неделям и корректное отображение актуальных изменений.

По результатам тестирования удалось улучшить структуру интерфейса, упростить взаимодействие с элементами управления и повысить общую удобочитаемость системы. Проверка показала, что приложение стабильно работает под нагрузкой, корректно обрабатывает данные и соответствует заявленным функциональным требованиям. Обнаруженные в процессе недочёты были устранены до завершения разработки, что повысило надёжность системы и подготовило её к использованию в реальных условиях.

2 Безопасность жизнедеятельности

2.1 Характеристика рабочего места разработчика

Рабочее место разработчика представляет собой специализированную зону, предназначенную для выполнения длительных интеллектуальных операций, связанных с проектированием, программированием и тестированием программных систем. Основу рабочего места составляет персональный компьютер с современным монитором, удобными устройствами ввода и эргономичной мебелью, обеспечивающей комфорт при многочасовой работе. Поскольку деятельность программиста требует высокой концентрации внимания и длительного пребывания в статичной позе, организация рабочего пространства должна учитывать требования охраны труда и санитарных норм.

Ключевым элементом является правильно подобранная мебель. Рабочий стол должен иметь достаточную площадь для размещения оборудования и документов, а также обеспечивать свободное положение рук при работе с клавиатурой и мышью. Кресло должно поддерживать поясничный отдел позвоночника и иметь регулируемые параметры - высоту сиденья, наклон спинки и подлокотники. Это позволяет адаптировать рабочее место под индивидуальные особенности сотрудника и снижает риск развития заболеваний опорно-двигательного аппарата.

Монитор размещается таким образом, чтобы взгляд пользователя был направлен под небольшим углом вниз, а расстояние до экрана соответствовало комфортной зоне восприятия. Это помогает уменьшить нагрузку на глаза и предотвратить развитие зрительного утомления. Для повышения комфорта рекомендуется использовать мониторы с антибликовым покрытием и настраиваемой яркостью.

2.2 Электробезопасность при работе с компьютерным оборудованием

Рабочее место разработчика включает несколько электрических устройств, что требует строгого соблюдения правил электробезопасности. Все элементы оборудования должны быть подключены через исправные розетки с заземлением, а кабели - размещены таким образом, чтобы исключить их повреждение, перегибы и случайное натяжение. Использование сетевых фильтров и стабилизаторов напряжения позволяет защитить технику от скачков напряжения, коротких замыканий и других нарушений в электросети, которые могут привести к выходу оборудования из строя.

Перед началом работы необходимо убедиться в отсутствии повреждений изоляции проводов, трещин на вилках и признаков перегрева. Запрещается подключать устройства мокрыми руками, использовать неисправные удлинители или размещать технику вблизи источников влаги. В случае появления запаха гари, нестабильной работы блока питания или чрезмерного нагрева корпуса оборудование следует немедленно отключить от сети и сообщить ответственному специалисту. Соблюдение этих правил снижает риск поражения электрическим током и предотвращает аварийные ситуации.

Дополнительно рекомендуется проводить регулярные проверки состояния электропроводки и периферийных устройств. Плановые осмотры позволяют своевременно выявлять изношенные кабели, повреждённые разъёмы и другие потенциально опасные элементы. В организациях такие проверки выполняются специалистами по охране труда или техническими сотрудниками, что обеспечивает соответствие рабочего места нормативным требованиям и снижает вероятность возникновения аварийных ситуаций.

2.2 Профилактика утомления при работе с компьютером

Профессиональная деятельность разработчика связана с длительной концентрацией внимания, высокой когнитивной нагрузкой и необходимостью многократно выполнять однотипные действия. Такие условия приводят к быстрому утомлению нервной системы, снижению работоспособности и повышению риска ошибок. Кроме того, работа за компьютером предполагает длительное пребывание в статичной позе, что вызывает напряжение мышц спины, шеи и плечевого пояса. Существенную нагрузку испытывает и зрительный аппарат, поскольку глаза вынуждены постоянно фокусироваться на близко расположенном источнике света.

Для уменьшения психофизиологического напряжения рекомендуется соблюдать рациональный режим труда и отдыха. Оптимальным считается выполнение коротких перерывов каждые 50-60 минут, во время которых следует сменить вид деятельности, выполнить лёгкую разминку, пройтись или выполнить упражнения для расслабления мышц. Особенно полезны гимнастика для глаз, чередование фокусировки на ближних и дальних объектах, а также упражнения для снятия напряжения с шейного отдела позвоночника. Такие меры позволяют снизить утомляемость, поддерживать высокую концентрацию внимания и предотвращать развитие хронических заболеваний.

Важную роль играет и организация рабочего пространства. Использование эргономичной мебели, регулируемого кресла и правильно расположенного монитора помогает снизить статическую нагрузку на опорно-двигательный аппарат. Грамотное освещение - как естественное, так и искусственное - уменьшает зрительное напряжение и предотвращает развитие синдрома сухого глаза. Соблюдение этих требований снижает вероятность появления профессиональных заболеваний, таких как остеохондроз, туннельный синдром, хронические головные боли и нарушения зрения.

3 Экономика

Для расчёта стоимости разработки использовался подход на основе фактических трудозатрат по этапам создания MVP. Разработкой занимался один full-stack разработчик (стек React + FastAPI + PostgreSQL). Исходные данные для расчёта: должность исполнителя - инженер-программист (full-stack), часовая тарифная ставка - 1 200 руб./час (установлена на основании среднерыночной стоимости оплаты труда разработчика соответствующей квалификации в регионе, с учётом страховых взносов и накладных расходов), продолжительность рабочего дня - 8 часов, общий срок выполнения работ - 4 календарных месяца.

В таблице 2 представлены трудозатраты.

Наименование этапа	Трудозатраты, чел.-час	Стоимость, руб.
Анализ предметной области, формирование требований	40	48 00
Проектирование архитектуры системы, структуры базы данных и API	48	57 600

Наименование этапа	Трудозатраты, чел.-час	Стоимость, руб.
Реализация серверной части (FastAPI, SQLAlchemy, PostgreSQL)	160	192 000
Реализация клиентской части (React, интерфейсы для всех ролей)	160	192 000
Интеграция компонентов, развёртывание и отладка	40	48 000
Тестирование (модульное, интеграционное, пользовательское) и оформление документации	48	57 600
Итого	496	595 200

Наибольший объём трудозатрат приходится на реализацию серверной (32 %) и клиентской (32 %) частей, что обусловлено сложностью разработки алгоритма генерации расписания с системой штрафов (penalty-based), интеграции с Yandex GPT API, а также созданием многофункциональных личных кабинетов и интерактивного календаря. При реализации серверной части особое внимание потребовалось уделить проектированию матричной модели состояния занятости преподавателей, групп и аудиторий, а также реализации логики выбора оптимальных временных слотов с учётом множества ограничений. Интеграция с Yandex GPT API потребовала дополнительных трудозатрат на разработку модуля парсинга и структурирования входных документов. Клиентская часть, в свою очередь, включает в себя несколько независимых интерфейсов для разных ролей пользователей, адаптивную вёрстку и динамическое обновление данных без перезагрузки страницы. В совокупности этапы программирования составляют 64 % от общего объёма работ.

Таким образом, расчётная стоимость разработки MVP веб-приложения составила 595 200 рублей. Внедрение разработанной системы позволит достичь следующих эффектов: сокращение временных затрат сотрудников учебного отдела на составление и корректировку расписания примерно на 50-60 % за счёт автоматизированной генерации и импорта данных через Yandex GPT, что особенно актуально в начале семестра при массовом вводе информации о дисциплинах и нагрузке преподавателей; минимизация риска ошибок и конфликтов в расписании (пересечений пар по времени, наложение преподавателей, ошибки в группах и аудиториях) все это возможно благодаря механизмам проверки, которые исключают человеческий фактор при ручном вводе. Повышает быстроту доступа к информации для студентов и преподавателей через современный веб-интерфейс с обновлением данных в реальном времени, позволяющий получать актуальное расписание с любого устройства.

Заключение

В результате выполнения выпускной квалификационной работы была создана программная система, предназначенная для автоматизации процессов, связанных с формированием и управлением расписанием. Исследование предметной области позволило выявить ключевые затруднения, возникающие при использовании традиционных методов организации учебного процесса: высокая трудоёмкость ручного составления расписаний, отсутствие единого хранилища данных и сложности при взаимодействии между участниками процесса. Анализ существующих решений показал, что доступные программные продукты либо не обладают достаточной гибкостью, либо ориентированы на иные масштабы и условия эксплуатации, что подтвердило необходимость разработки собственного программного комплекса.

В ходе работы были сформированы функциональные и технические требования, определены цели и задачи проекта, а также выбран технологический стек, который включает в себя FastAPI, PostgreSQL и React. Такое сочетание технологий обеспечило высокую скорость обработки данных, удобство разработки и возможность дальнейшего расширения системы. На этапе проектирования была сформирована архитектура приложения, которая включает в себя серверную и клиентскую части, а также разработана структура БД, отражающая процессы, связанные с планированием учебной деятельности. Для различных категорий пользователей - сотрудников учебного отдела, преподавателей и студентов - были созданы удобные интерфейсы, обеспечивающие доступ к необходимым функциям.

В процессе разработки реализованы основные модули системы: авторизация и аутентификация, инструменты для управления расписанием, средства проверки занятости преподавателей и аудиторий, интерфейсы просмотра расписания и административные функции.

Список использованных источников

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. - М.: Питер, 2021. - 416 с.

2. Мартин Р. Чистый код. Создание, анализ и рефакторинг. - СПб.: Питер, 2022. - 464 с.

3. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. - СПб.: Питер, 2021. - 352 с.

4. Хант Э., Томас Д. Программист-прагматик. Путь от подмастерья к мастеру. - СПб.: Питер, 2020. - 352 с.

5. Норрис М., Хартл С. Архитектура веб-приложений. - М.: Вильямс, 2021. - 480 с.

6. Mozilla Developer Network. JavaScript Guide. [Электронный ресурс] - URL: https://developer.mozilla.org (дата обращения: 15.02.2026).

7. FastAPI Documentation. [Электронный ресурс] - URL: https://fastapi.tiangolo.com (дата обращения: 10.02.2026).

8. ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE).

9. Wieringa R. Design Science Methodology for Information Systems and Software Engineering. - Springer, 2014. - 336 p.

10. Krug S. Don't Make Me Think. A Common Sense Approach to Web Usability. - New Riders, 2014. - 216 p.

11. DeepSeek-AI. DeepSeek-V3: [Электронный ресурс] - URL: https://www.deepseek.com (дата обращения: 15.05.2026). (В качестве промтов использовались запросы на поиск источников по заданной тематике и суммаризацию информации из найденных источников)

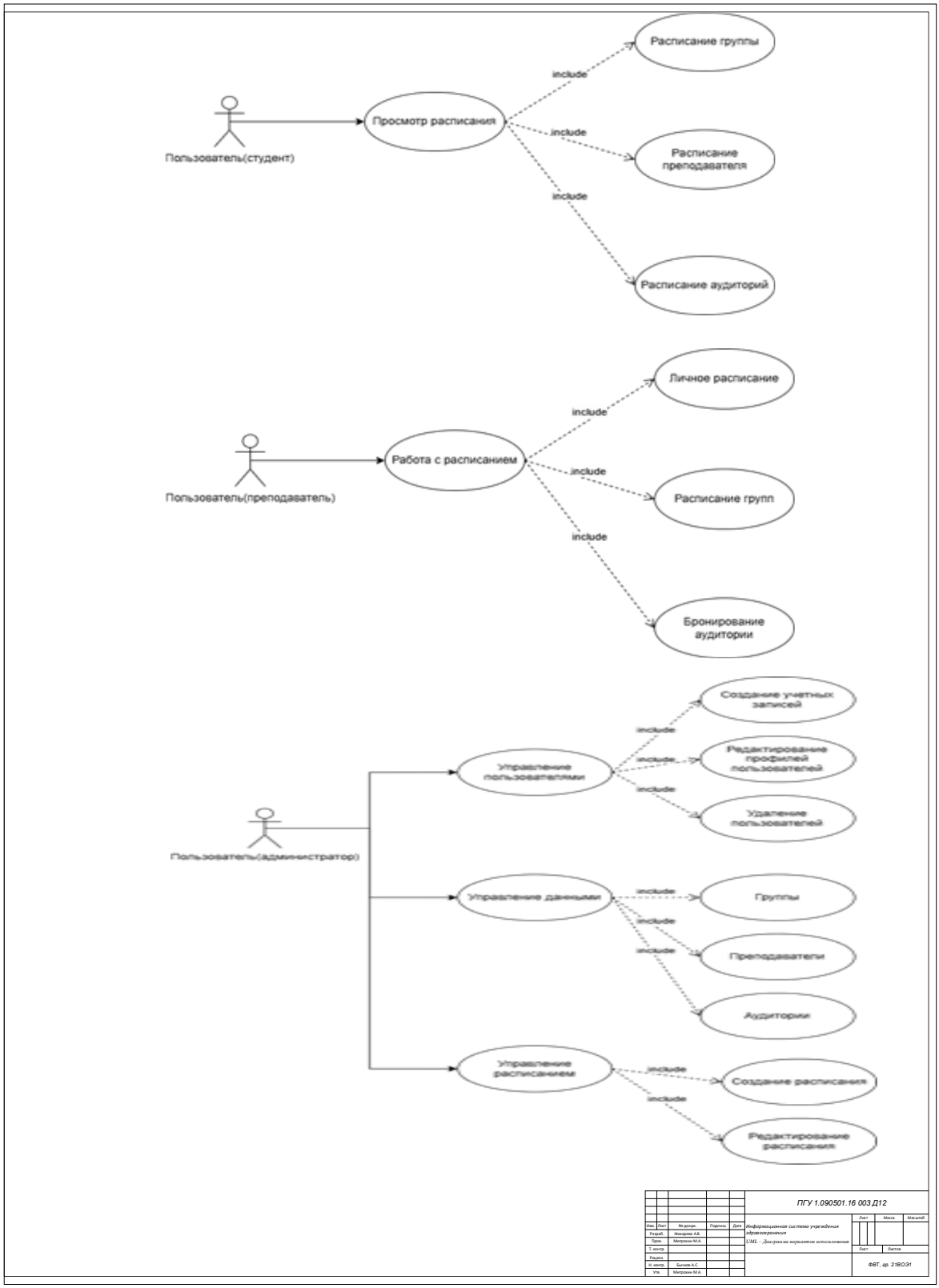
Приложение А

(обязательное)

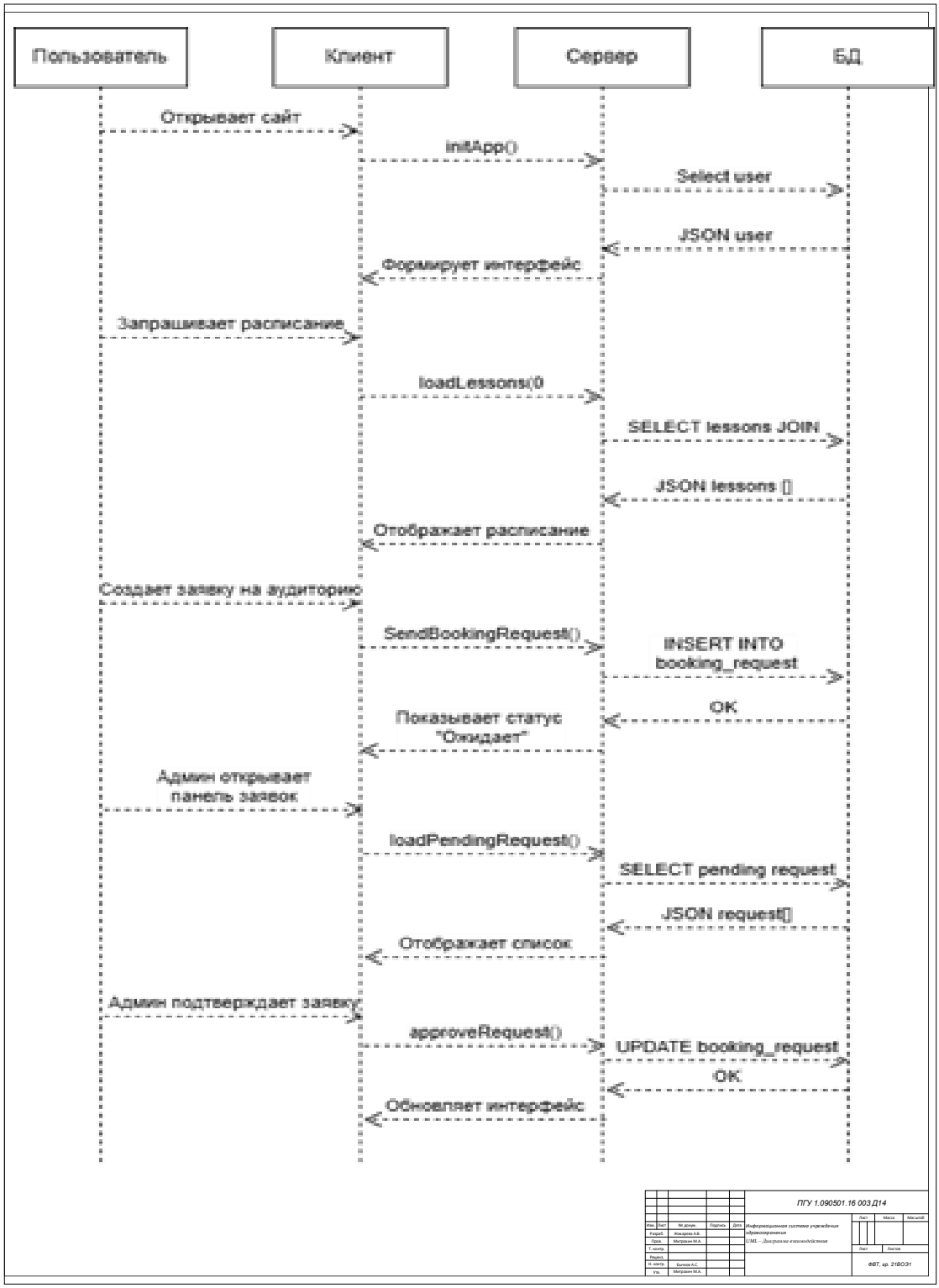
«Приложение для составления расписания в военном учебном заведении»

UML-диаграммы

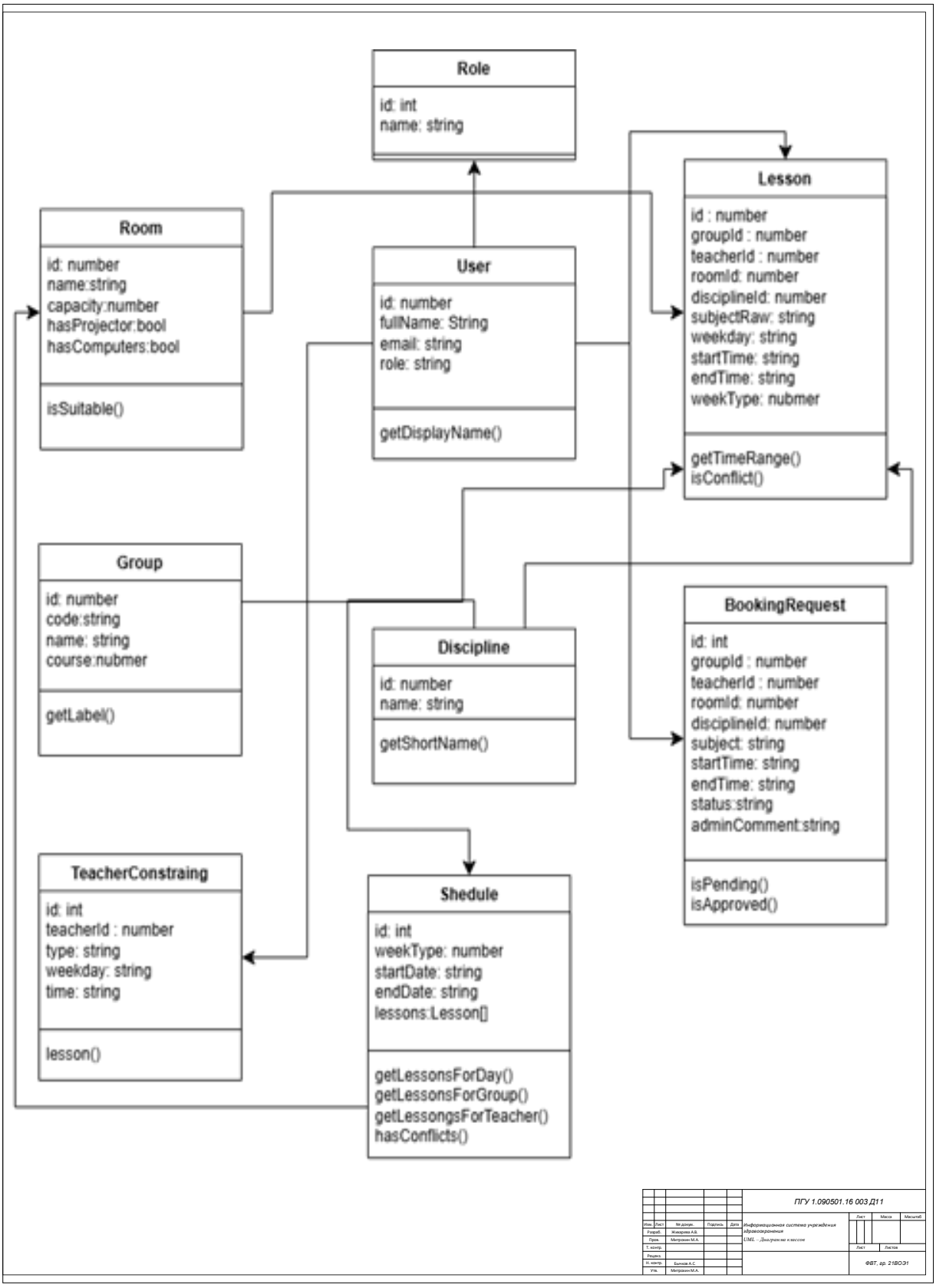
UML – диаграмма вариантов использования



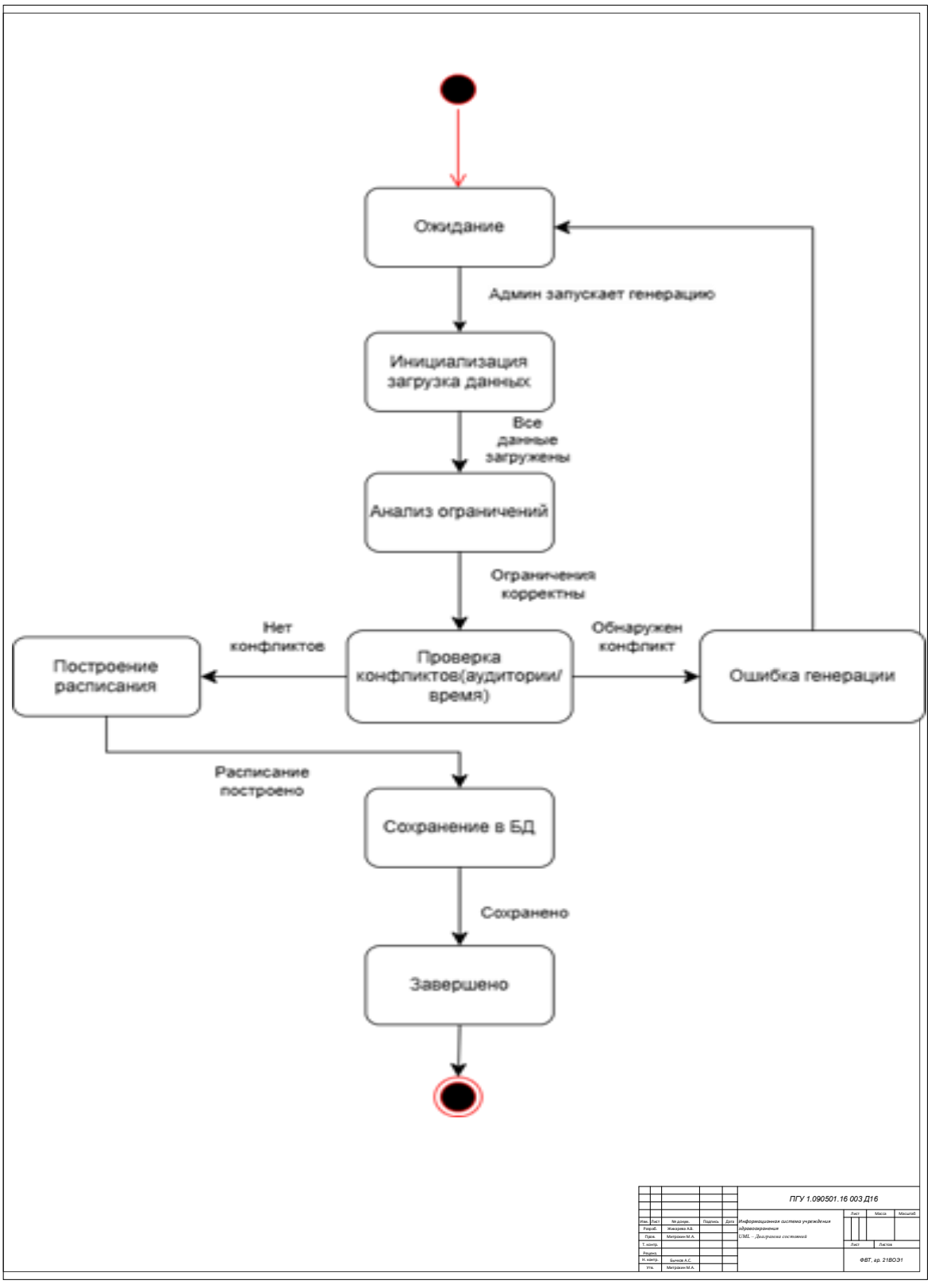
UML – диаграмма последовательности



UML – диаграмма классов



UML – диаграмма состояний



Приложение Б

(обязательное)

«Приложение для составления расписания в военном учебном заведении»

Листинг программы

```
    const handleSubmit = (e) => {
        e.preventDefault();
        if (!validate()) return;
        onCreate(form);
        setForm({ slot_id: "", patient_id: "", complaint: "" });
      };

      return (
        <form className="form" onSubmit={handleSubmit}>
          <div className="form-field">
            <label>Слот</label>
            <select
              className={errors.slot_id ? "input-error" : ""}
              value={form.slot_id}
              onChange={(e) => handleChange("slot_id",
e.target.value)}
            >
              <option value="">-- выберите --</option>
              {slots
                .filter((s) => s.is_available)
                .map((s) => (
                  <option key={s.id} value={s.id}>
                    {new Date(s.start_time).toLocaleString()} —{"
"}
                    {new Date(s.end_time).toLocaleString()}
                  </option>
                ))}
            </select>
            {errors.slot_id && (
              <div className="error-text">{errors.slot_id}</div>
            )}
          </div>
```

```
        <div className="form-field">
            <label>Пациент</label>
            <select
              className={errors.patient_id ? "input-error" : ""}
              value={form.patient_id}
              onChange={(e) => handleChange("patient_id",
e.target.value)}
            >
              <option value="">-- выберите --</option>
              {patients.map((p) => (
                <option key={p.id} value={p.id}>
                  {p.last_name} {p.first_name} {p.middle_name ||
""}
                </option>
              ))}
            </select>
            {errors.patient_id && (
              <div className="error-text">{errors.patient_id}</div>
            )}
          </div>

          <div className="form-field">
            <label>Жалоба</label>
            <textarea
              value={form.complaint}
              onChange={(e) => handleChange("complaint",
e.target.value)}
            />
          </div>

          <Button type="submit" variant="primary">
            Создать запись
          </Button>
        </form>
      );
```

Файл «AppointmentPage.js»

```
    import React, { useEffect, useState } from "react";
import {
  fetchDoctors,
  fetchSlotsForDoctor,
  createAppointment,
  getMyAppointments
} from "../api";

import { Table, Th, Td } from "../components/common/Table";
import Button from "../components/common/Button";
import Loader from "../components/common/Loader";

const AppointmentsPage = () => {
  const [doctors, setDoctors] = useState([]);
  const [slots, setSlots] = useState([]);
  const [myAppointments, setMyAppointments] = useState([]);
  const [selectedDoctor, setSelectedDoctor] = useState("");
  const [loadingSlots, setLoadingSlots] = useState(false);
  const [loadingMy, setLoadingMy] = useState(false);
  const [message, setMessage] = useState("");

  // Загружаем список врачей
  useEffect(() => {
    fetchDoctors().then(setDoctors);
  }, []);
```

```
/ Загружаем записи пациента, если врач не выбран
  useEffect(() => {
    if (!selectedDoctor) {
      loadMyAppointments();
    }
  }, [selectedDoctor]);


  const loadMyAppointments = async () => {
    setLoadingMy(true);
    try {
      const data = await getMyAppointments();
      setMyAppointments(data);
    } finally {
      setLoadingMy(false);
    }
  };
const loadSlots = async (doctorId) => {
    if (!doctorId) {
      setSlots([]);
      return;
    }
```

```
    setLoadingSlots(true);
        try {
          const data = await fetchSlotsForDoctor(doctorId);
          setSlots(data);
        } finally {
          setLoadingSlots(false);
        }
      };

      const handleDoctorChange = (id) => {
        setSelectedDoctor(id);
        if (id) loadSlots(id);
      };

      // Запись пациента
      const handleBook = async (slotId) => {
        try {
          await createAppointment({ slot_id: slotId });
          setMessage("Вы успешно записались на приём");
          loadSlots(selectedDoctor);
        } catch (err) {
          setMessage("Ошибка записи");
        }
      };
    return (
        <div>
          <h1 style={{ marginBottom: 20 }}>Расписание приёмов</h1>

          {message && (
            <div style={{ marginBottom: 20, color: "green",
fontSize: 16 }}>
              {message}
            </div>
          )}
```

```
{/* Выбор врача */}
      <div style={{ marginBottom: 20 }}>
        <label style={{ fontSize: 14 }}>
          Выберите врача:
          <select
            value={selectedDoctor}
            onChange={(e) => handleDoctorChange(e.target.value)}
            style={{
              marginLeft: 10,
              padding: 6,
              borderRadius: 6,
              border: "1px solid #d4d9e5"
            }}
          >
            <option value="">Показать мои записи</option>
            {doctors.map((d) => (
              <option key={d.id} value={d.id}>
                {d.last_name} {d.first_name}
              </option>
            ))}
          </select>
        </label>
      </div>

     {/* ---------------------------- */}
         {/* РЕЖИМ 1: врач НЕ выбран → показываем мои записи */}
         {/* ---------------------------- */}
         {!selectedDoctor && (
          <>
            <h2 style={{ marginTop: 20 }}>Мои записи</h2>

            {loadingMy && <Loader />}

            {!loadingMy && myAppointments.length === 0 && (
              <p>У вас пока нет записей</p>
            )}
```

```
    {!loadingMy &&
                myAppointments.map((a) => (
                  <div
                    key={a.id}
                    style={{
                      padding: 12,
                      border: "1px solid #d4d9e5",
                      borderRadius: 6,
                      marginBottom: 12
                    }}
                  >
                    <div>
                      <b>Врач:</b> {a.slot.doctor.last_name}
{a.slot.doctor.first_name}
                    </div>

                    <div>
                      <b>Начало:</b> {new
Date(a.slot.start_time).toLocaleString()}
                    </div>

                    <div>
                      <b>Конец:</b> {new
Date(a.slot.end_time).toLocaleString()}
                    </div>

                    <div>
                      <b>Статус:</b> {a.status}
                    </div>
```

```
    {a.complaint && (
                      <div>
                        <b>Жалоба:</b> {a.complaint}
                      </div>
                    )}
                  </div>
                ))}
            </>
          )}

          {/* ----------------------------- */}
          {/* РЕЖИМ 2: врач выбран → показываем слоты врача */}
          {/* ----------------------------- */}
          {selectedDoctor && !loadingSlots && (
            <>
     <h2 style={{ marginTop: 20 }}>Слоты врача</h2>

              <Table>
                <thead>
                  <tr>
                    <Th>Начало</Th>
                    <Th>Конец</Th>
                    <Th>Статус</Th>
                    <Th>Пациент</Th>
                    <Th></Th>
                  </tr>
                </thead>
    <tbody>
                  {slots.map((s) => (
                    <tr key={s.id}>
                      <Td>{new
Date(s.start_time).toLocaleString()}</Td>
                      <Td>{new Date(s.end_time).toLocaleString()}</Td>
                      <Td>{s.is_available ? "Свободен" : "Занят"}</Td>
                      <Td>
```

```
      {s.appointment && s.appointment.patient
                          ? `${s.appointment.patient.last_name}
${s.appointment.patient.first_name}`
                          : "—"}
                      </Td>

                      <Td>
                        {s.is_available && (
                          <Button onClick={() => handleBook(s.id)}>
                            Записаться
                          </Button>
     )}
                      </Td>
                    </tr>
                  ))}
                </tbody>
              </Table>
            </>
          )}

          {selectedDoctor && loadingSlots && <Loader />}
        </div>
      );
    };

    export default AppointmentsPage;
```

Файл «BookVisitPage.js»

```
    import React, { useEffect, useState } from "react";
    import { fetchDoctors, fetchSlotsForDoctor, createAppointment }
from "../api";
    import "./BookVisitPage.css";

    const BookVisitPage = () => {
      const [doctors, setDoctors] = useState([]);
      const [slots, setSlots] = useState([]);
      const [selectedDoctorId, setSelectedDoctorId] = useState("");
      const [selectedSlotId, setSelectedSlotId] = useState("");
      const [complaint, setComplaint] = useState("");   // ← жалоба
      const [message, setMessage] = useState(null);
  useEffect(() => {
    loadDoctors();
  }, []);

  const loadDoctors = async () => {
    const data = await fetchDoctors();
    setDoctors(data);
  };

  const loadSlots = async (doctorId) => {
    if (!doctorId) {
      setSlots([]);
      return;
    }

    const data = await fetchSlotsForDoctor(doctorId);

    // Показываем только свободные слоты
    const freeSlots = data.filter(
      (s) => s.is_available && !s.appointment
    );
```

```
    setSlots(freeSlots);
      };

      const handleSubmit = async (e) => {
        e.preventDefault();
        setMessage(null);

        if (!selectedSlotId) {
          setMessage("Выберите время приёма");
          return;
        }

        try {
          await createAppointment({
            slot_id: Number(selectedSlotId),
            complaint: complaint || null   // ← передаём жалобу
          });

          setMessage("Вы успешно записались на приём");
          setSelectedSlotId("");
          setComplaint("");                // ← очищаем поле жалобы
    if (selectedDoctorId) {
            loadSlots(selectedDoctorId);
          }

        } catch (err) {
          console.error(err);
          setMessage("Ошибка записи на приём");
        }
      };
```

```
return (
   <div className="book-container">
     <h1 className="book-title">Записаться к врачу</h1>

     <form onSubmit={handleSubmit} className="book-form">

       {/* ВЫБОР ВРАЧА */}
       <label className="book-label">
         Врач:
         <select
           className="book-select"
           value={selectedDoctorId}
           onChange={(e) => {
             setSelectedDoctorId(e.target.value);
             loadSlots(e.target.value);
           }}
<option value="">— выберите врача —</option>
           {doctors.map((d) => (
             <option key={d.id} value={d.id}>
               {d.last_name} {d.first_name}
             </option>
           ))}
         </select>
       </label>
```

```
    {/* ВЫБОР ВРЕМЕНИ */}
           <label className="book-label">
             Время:
             <select
               className="book-select"
               value={selectedSlotId}
               onChange={(e) => setSelectedSlotId(e.target.value)}
               disabled={!selectedDoctorId}
             >
               <option value="">— выберите время —</option>

               {slots.length === 0 && selectedDoctorId && (
                 <option value="">Свободных слотов нет</option>
               )}

{slots.map((s) => (
                <option key={s.id} value={s.id}>
                  {new Date(s.start_time).toLocaleString()} —{" "}
                  {new Date(s.end_time).toLocaleString()}
                </option>
              ))}
             </select>
           </label>
{/* ЖАЛОБА */}
           <label className="book-label">
             Жалоба:
             <textarea
               className="book-textarea"
               placeholder="Опишите вашу жалобу (необязательно)"
               value={complaint}
               onChange={(e) => setComplaint(e.target.value)}
               rows={3}
             />
           </label>
```

```
 <button type="submit" className="book-btn">
          Записаться
        </button>
      </form>

      {message && <p className="book-message">{message}</p>}
    </div>
  );
};

export default BookVisitPage;
```

Файл «ChangePasswordPage.js»

```
import React, { useState } from "react";
import { changePassword } from "../api";
import "./ChangePasswordPage.css";

const ChangePasswordPage = () => {
  const [oldPassword, setOldPassword] = useState("");
  const [newPassword, setNewPassword] = useState("");
  const [repeatPassword, setRepeatPassword] = useState("");
  const [message, setMessage] = useState(null);

  const handleSubmit = async () => {
    setMessage(null);
```

```
if (newPassword !== repeatPassword) {
      setMessage("Новые пароли не совпадают");
      return;
    }


    try {
      await changePassword({
        old_password: oldPassword,
        new_password: newPassword,
      });
setMessage("Пароль успешно изменён");
        setOldPassword("");
        setNewPassword("");
        setRepeatPassword("");
      } catch (err) {
        console.error(err);
        setMessage(
          err.response?.data?.detail || "Ошибка при смене пароля"
        );
      }
    };
```

```
      return (
         <div className="password-container">
           <h1 className="password-title">Смена пароля</h1>

           {message && <div className="password-
message">{message}</div>}

           <label className="password-label">
             Старый пароль:
             <input
               type="password"
               value={oldPassword}
               onChange={(e) => setOldPassword(e.target.value)}
             />
           </label>
      <label className="password-label">
           Новый пароль:
           <input
             type="password"
             value={newPassword}
             onChange={(e) =>
         setNewPassword(e.target.value)}
             />
           </label>
      label className="password-label">
           Повторите новый пароль:
           <input
             type="password"
             value={repeatPassword}
             onChange={(e) =>
         setRepeatPassword(e.target.value)}
             />
           </label>
```

```
button className="password-btn" onClick={handleSubmit}>
        Изменить пароль
      </button>
    </div>
  );
};

export default ChangePasswordPage;
```

Файл «DashBoard.js»

```
import React, { useEffect, useState } from "react";
import {
  fetchSummaryReport,
  getMyProfile,
  getMyAppointments,
  getMyAppointmentsAsDoctor,
  fetchSlotsForDoctor,
  getDoctorByUserId
} from "../api";

import Loader from "../components/common/Loader";
import ErrorBlock from "../components/common/ErrorBlock";
import Button from "../components/common/Button";
import { useNavigate } from "react-router-dom";

const DashboardPage = () => {
  const navigate = useNavigate();
```

```
  onst [role, setRole] = useState(null);

// patient data
const [patientProfile, setPatientProfile] = useState(null);
const [patientVisits, setPatientVisits] = useState([]);

// admin summary
const [summary, setSummary] = useState(null);

// doctor data
const [doctorId, setDoctorId] = useState(null);
const [doctorAppointments, setDoctorAppointments] = useState([]);
const [todayStats, setTodayStats] = useState({
  total: 0,
  free: 0,
  busy: 0
});
   onst [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    // Универсальный извлекатель пациента
    const extractPatient = (a) =>
      a.patient ||
      a.slot?.patient ||
      a.slot?.appointment?.patient ||
      null;

    const extractPatientId = (patient, a) =>
      patient?.id ||
      patient?.patient_id ||
      a.patient_id ||
      a.slot?.patient_id ||
      null;
```

```
        // 1. Определяем роль
  useEffect(() => {
    const stored = localStorage.getItem("role");
    setRole(stored);
  }, []);

  // 2. Загружаем данные
  useEffect(() => {
    if (!role) return;

// -------------------------
    // ПАЦИЕНТ
    // -------------------------
    if (role === "patient") {
      Promise.all([getMyProfile(), getMyAppointments()])
        .then(([profile, visits]) => {
          setPatientProfile(profile);
          setPatientVisits(visits || []);
          setLoading(false);
        })
        .catch(() => {
          setError("Не удалось загрузить данные пациента");
          setLoading(false);
        });
    }
```

```
    // -------------------------
    // АДМИН
    // -------------------------
    if (role === "admin") {
      const today = new Date().toISOString().slice(0, 10);

      fetchSummaryReport(today)
        .then((data) => {
          setSummary(data);
          setLoading(false);
        })
        .catch(() => {
          setError("Не удалось загрузить данные отчёта");
          setLoading(false);
        });
    }
// -------------------------
    // ВРАЧ
    // -------------------------
    if (role === "doctor") {
      const user = JSON.parse(localStorage.getItem("user"));

      getDoctorByUserId(user.id)
        .then((doctor) => {
          if (doctor) {
            setDoctorId(doctor.id);
// Приёмы врача
            getMyAppointmentsAsDoctor().then((apps) => {
              setDoctorAppointments(apps || []);
            });
```

```
        // Статистика слотов
                fetchSlotsForDoctor(doctor.id, true).then((slots) => {
                  const today = new Date().toISOString().split("T")[0];
                  const todaySlots = slots.filter((s) =>
                    s.start_time.startsWith(today)
                  );

                  setTodayStats({
                    total: todaySlots.length,
                    free: todaySlots.filter((s) =>
s.is_available).length,
                    busy: todaySlots.filter((s) =>
!s.is_available).length
                  });
                });
    }
              setLoading(false);
            })
            .catch(() => {
              setError("Не удалось загрузить данные врача");
              setLoading(false);
            });
        }
      }, [role]);
     if (loading) return <Loader />;
      if (error) return <ErrorBlock message={error} />;

      // ----------------------------------------------------
      // PATIENT DASHBOARD
      // ----------------------------------------------------
      if (role === "patient") {
        const futureVisits = patientVisits.filter(
          (v) => new Date(v.slot.start_time) > new Date()
        );
```

```
    const nextVisit =
          futureVisits.length > 0
            ? futureVisits.sort(
                (a, b) =>
                  new Date(a.slot.start_time) - new
Date(b.slot.start_time)
              )[0]
            : null;

        const formatDate = (value) => new Date(value).toLocaleString();
        const statusMap = {
        pending: "Ожидает",
       confirmed: "Подтверждён",
        cancelled: "Отменён",
        completed: "Завершён",
        in_progress: "В процессе",
        };
     return (
          <div>
            <h1 style={{ marginBottom: 20 }}>
              Добро пожаловать, {patientProfile.first_name}
            </h1>

            {/* Ближайший визит */}
            <div className="next-visit-block" style={{ marginBottom: 20
}}>
              <div className="next-visit-title">Ближайший визит</div>

              {nextVisit ? (
                <div className="next-visit-info">
                  <div>
                    <strong>Дата:</strong>
{formatDate(nextVisit.slot.start_time)}
                  </div>
                  <div>
```

```
<strong>Врач:</strong>{" "}
          {nextVisit.slot.doctor
           ? `${nextVisit.slot.doctor.last_name} ${nextVisit.slot.doctor.first_name}`
           : "Не указан"}
        </div>
        <div>
         <strong>Статус:</strong> {statusMap[nextVisit.status] || nextVisit.status}

        </div>
       </div>
      ) : (

              <div style={{ marginTop: 10 }}>У вас нет предстоящих визитов</div>
                   )}
                 </div>

                 {/* Быстрая статистика */}
                 <div className="dashboard-grid" style={{ marginBottom: 20 }}>
                  <div className="dashboard-card">
                   <div className="dashboard-label">Будущих визитов</div>
                   <div className="dashboard-value">{futureVisits.length}</div>
                  </div>

                  <div className="dashboard-card">
                   <div className="dashboard-label">Полис</div>
                   <div className="dashboard-value">
                    {patientProfile.insurance_number || "—"}
                   </div>
                  </div>
            <div className="dashboard-card">
                   <div className="dashboard-label">Телефон</div>
                   <div className="dashboard-value">
                    {patientProfile.phone || "—"}
                   </div>
                  </div>
                 </div>
               </div>
              );
             }
            / ------------------------------------------------
             // DOCTOR DASHBOARD
             // ------------------------------------------------
             if (role === "doctor") {
              const today = new Date().toISOString().split("T")[0];

              const todaysAppointments = doctorAppointments.filter(
               (a) =>
                extractPatient(a) &&
                a.slot.start_time.startsWith(today)
              );
```

```
const upcomingAppointments = doctorAppointments
      .filter((a) => extractPatient(a))
      .filter((a) => new Date(a.slot.start_time) > new Date())
      .sort(
        (a, b) =>
          new Date(a.slot.start_time) - new Date(b.slot.start_time)
      );

    return (
      <div>
        <h1>Панель врача</h1>
      {/* ======== Статистика
      ======== */}
           <div style={{
      display: "flex", gap: 20,
      marginTop: 20 }}>
            <div
      className="dashboard-
      card">
             <div
      className="dashboard-
      label">Приёмов сегодня</div>
             <div
      className="dashboard-
      value">{todaysAppointments
      .length}</div>
            </div>
      <div className="dashboard-
      card">
             <div
      className="dashboard-
      label">Свободных
      слотов</div>
             <div
      className="dashboard-
      value">{todayStats.free}</
      div>
```

```
</div>

     <div className="dashboard-card">
       <div className="dashboard-label">Всего слотов</div>
       <div className="dashboard-value">{todayStats.total}</div>
     </div>
    </div>
      {/* ======== Быстрые действия ======== */}
              <h2 style={{ marginTop: 30 }}>Быстрые действия</h2>
              <div style={{ display: "flex", gap: 20 }}>
                <Button onClick={() => navigate("/schedule")}>Моё
расписание</Button>
                <Button onClick={() => navigate("/doctor/appointments")}>
                  Мои приёмы
                </Button>
                <Button onClick={() => navigate("/patients")}>Мои
пациенты</Button>
                <Button onClick={() => navigate("/records")}>Мед.
записи</Button>
              </div>

              {/* ======== Ближайшие приёмы ======== */}
              <h2 style={{ marginTop: 40 }}>Ближайшие приёмы</h2>
     {upcomingAppointments.length === 0 && (
                <div>Нет предстоящих приёмов</div>
              )}

              <div style={{ display: "flex", flexDirection: "column",
gap: 10 }}>
                {upcomingAppointments.slice(0, 5).map((a) => {
                  const slot = a.slot;
                  const patient = extractPatient(a);
```

```
const patientName =
              patient?.last_name && patient?.first_name
                ? `${patient.last_name} ${patient.first_name}`
                : patient?.full_name
                ? patient.full_name
                : patient?.insurance_number
                ? patient.insurance_number
                : "Пациент";
```

onst patientId = extractPatientId(patient, a);

```
          return (
            <div
              key={a.id}
              style={{
                padding: 12,
                border: "1px solid #d4d9e5",
                borderRadius: 6,
                display: "flex",
                justifyContent: "space-between"
              }}
            >
```

```
      <div>
                          <b>
                            {new Date(slot.start_time).toLocaleString("ru-
RU", {
                              year: "numeric",
                              month: "2-digit",
                              day: "2-digit",
                              hour: "2-digit",
                              minute: "2-digit"
                            })}
                          </b>{" "}
                          — {patientName}
                          <div style={{ fontSize: 13, color: "#666" }}>
                            {a.complaint || "Без жалобы"}
                          </div>

        </div>


                          <Button
                            onClick={() => {
                              if (!patientId) {
                                alert("У этого приёма нет
      пациента");
                                return;
                              }
                              navigate(`/doctor/patient/${patient
      Id}`);
                            }}
                          >
```

```
    Открыть
                    </Button>
                  </div>
                );
              })}
            </div>
          </div>
        );
      }
    / ------------------------------------------------------
      // ADMIN DASHBOARD
      // ------------------------------------------------------
      if (role === "admin") {
        return (
          <div>
            <h1 style={{ marginBottom: 20 }}>Панель управления</h1>

            {summary && (
              <div className="dashboard-grid">
                <div className="dashboard-card">
                  <div className="dashboard-label">Приёмов сегодня</div>
                  <div className="dashboard-
value">{summary.appointments}</div>
                </div>
```

```
    div className="dashboard-card">
                    <div className="dashboard-label">Пациентов</div>
                    <div className="dashboard-
value">{summary.patients}</div>
                  </div>

                  <div className="dashboard-card">
                    <div className="dashboard-label">Врачей</div>
                    <div className="dashboard-
value">{summary.doctors}</div>
                  </div>
                </div>
    )}

          </div>

        );

      }


      return null;

    };


    export default DashboardPage;
```

Файл «__init__.py»

```
from fastapi import APIRouter
from . import auth_routes, patient_routes, doctor_routes, appointment_routes
from . import medical_record_routes, drug_routes, report_routes
from .specialty_routes import router as specialty_router
from .documents_routes import router as documents_router

api_router = APIRouter(prefix="/api")

api_router.include_router(auth_routes.router)
api_router.include_router(patient_routes.router)
api_router.include_router(doctor_routes.router)
api_router.include_router(appointment_routes.router)
api_router.include_router(medical_record_routes.router)
api_router.include_router(drug_routes.router)
api_router.include_router(report_routes.router)
api_router.include_router(specialty_router)
api_router.include_router(documents_router)  #
```

Файл «appointment_router.py»

```
    # app/api/appointment_routes.py

    from datetime import datetime
    from fastapi import APIRouter, Depends, Query, status,
HTTPException
    from sqlalchemy.orm import Session, joinedload

    from ..db import get_db
    from ..schemas import (
        AppointmentSlotCreate,
        AppointmentSlotRead,
        AppointmentCreate,
        AppointmentUpdateStatus,
        AppointmentRead,
        ScheduleGenerationRequest,
        AppointmentCancelRequest,
        AppointmentRescheduleRequest,
    )
    from ..services.appointment_service import AppointmentService
    from .deps import get_current_user

    from app.models.appointment import Appointment, AppointmentSlot
    from app.models.patient import Patient
    from app.models.doctor import Doctor

    from app.utils.email_sender import send_email

    router = APIRouter(prefix="/appointments", tags=["appointments"])
```

```
    # ----------------------------
    # СОЗДАНИЕ СЛОТА
    # ----------------------------
    @router.post("/slots", response_model=AppointmentSlotRead,
status_code=201)
    def create_slot(
        data: AppointmentSlotCreate,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.create_slot(db, data)

    # ----------------------------
    # СПИСОК СЛОТОВ ВРАЧА
    # ----------------------------
    @router.get("/slots/{doctor_id}",
response_model=list[AppointmentSlotRead])
    def list_slots_for_doctor(
        doctor_id: int,
        start_from: datetime | None = Query(None),
        with_appointments: bool = Query(False),
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.list_slots_for_doctor(
            db, doctor_id, start_from, with_appointments
        )
```

```
    # ----------------------------
    # ГЕНЕРАЦИЯ РАСПИСАНИЯ
    # ----------------------------
    @router.post("/slots/generate/{doctor_id}",
response_model=list[AppointmentSlotRead], status_code=201)
    def generate_schedule_for_doctor(
        doctor_id: int,
        data: ScheduleGenerationRequest,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.generate_schedule_for_doctor(db,
doctor_id, data)

    # ----------------------------
    # СОЗДАНИЕ ПРИЁМА
    # ----------------------------
    @router.post("/", response_model=AppointmentRead, status_code=201)
    def create_appointment_for_patient(
        data: AppointmentCreate,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
     appointment =
AppointmentService.create_appointment_for_patient(db, user.id, data)

        # Email пациенту
        if user.email:
            slot = appointment.slot
            doctor = slot.doctor
```

```
    text = (
                f"Вы успешно записались на приём.\n\n"
                f"Врач: {doctor.last_name} {doctor.first_name}\n"
                f"Дата: {slot.start_time}\n"
                f"Статус: {appointment.status}\n\n"
                f"Спасибо, что выбрали MedClinicIS!"
            )

            send_email(
                to_email=user.email,
                subject="Запись на приём — подтверждение",
                text=text,
            )

        return appointment

    # ----------------------------
    # ОБНОВЛЕНИЕ СТАТУСА
    # ----------------------------
    @router.patch("/{appointment_id}/status",
response_model=AppointmentRead)
    def update_status(
        appointment_id: int,
        data: AppointmentUpdateStatus,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.update_status(db, appointment_id,
data)
    @router.get("/patient/{patient_id}",
response_model=list[AppointmentRead])
```

```
    def list_appointments_for_patient(
        patient_id: int,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        return AppointmentService.list_for_patient(db, patient_id)

    # ----------------------------
    # МОИ ПРИЁМЫ (ПАЦИЕНТ)
    # ----------------------------
    @router.get("/my", response_model=list[AppointmentRead])
    def list_my_appointments(
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        patient = db.query(Patient).filter(Patient.user_id ==
user.id).first()
        if not patient:
            raise HTTPException(status_code=404, detail="Пациент не
найден")

        return AppointmentService.list_for_patient(db, patient.id)

    # ----------------------------
    # МОИ ПРИЁМЫ (ВРАЧ) — ВСЕ ПРИЁМЫ
    # ----------------------------
    @router.get("/doctor/my", response_model=list[AppointmentRead])
    def get_my_appointments_as_doctor(
        db: Session = Depends(get_db),
        current_user=Depends(get_current_user),
```

```
    # 1. Находим врача, привязанного к пользователю
    doctor = db.query(Doctor).filter(Doctor.user_id == current_user.id).first()
    if not doctor:
        return []

    # 2. Возвращаем ВСЕ приёмы врача + подгружаем patient и slot
    appointments = (
        db.query(Appointment)
        .options(
            joinedload(Appointment.patient),
            joinedload(Appointment.slot).joinedload(AppointmentSlot.doctor)
        )
    .join(AppointmentSlot)
        .filter(AppointmentSlot.doctor_id == doctor.id)
        .order_by(AppointmentSlot.start_time.asc())
        .all()
    )

    return appointments
# ----------------------------
# ОТМЕНА ПРИЁМА
# ----------------------------
@router.post("/{appointment_id}/cancel", response_model=AppointmentRead)
def cancel_appointment(
    appointment_id: int,
    data: AppointmentCancelRequest,
    db: Session = Depends(get_db),
    user=Depends(get_current_user),
):
```

```
    appointment = AppointmentService.cancel_appointment(db,
appointment_id, data.reason)

        if user.email:
            send_email(
                to_email=user.email,
                subject="Отмена записи",
                text=f"Ваша запись была отменена.\nПричина:
{data.reason or 'не указана'}"
            )

        return appointment
    # ------------------------------
    # ПЕРЕНОС ПРИЁМА
    # ------------------------------
    @router.post("/{appointment_id}/reschedule",
response_model=AppointmentRead)
    def reschedule_appointment(
        appointment_id: int,
        data: AppointmentRescheduleRequest,
        db: Session = Depends(get_db),
        user=Depends(get_current_user),
    ):
        appointment = AppointmentService.reschedule_appointment(
            db,
            appointment_id,
            data.new_slot_id,
            data.complaint,
        )
```

```
    if user.email:
            slot = appointment.slot
            doctor = slot.doctor

            send_email(
                to_email=user.email,
                subject="Перенос записи",
                text=(
                    f"Ваша запись была перенесена.\n\n"
                    f"Новый врач: {doctor.last_name}
{doctor.first_name}\n"
                    f"Дата: {slot.start_time}\n"
                )
            )

        return appointment
```

Файл «auth_routes.py»

```
    # app/api/auth_routes.py
    from fastapi import APIRouter, Depends, HTTPException
    from sqlalchemy.orm import Session

    from app.db import get_db
    from app.schemas.auth import LoginRequest, RegisterRequest, Token,
ChangePasswordRequest
    from app.services.auth_service import AuthService
    from .deps import get_current_user
    from app.models import User
```

Приложение В

(обязательное)

«Приложение для составления расписания в военном учебном заведении»

Презентация