Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (МОСКОВСКИЙ ПОЛИТЕХ)

КУРСОВОЙ ПРОЕКТ

По курсу «Разработка веб-приложений»

ТЕМА

«Разработка веб-приложения для логопедического центра с системой абонементов и личными кабинетами пользователей»

Выполнил Самохвалов Вячеслав Дмитриевич

Группа 231-3210

Проверил Кружалов А.С.

  Москва, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  заведующая кафедрой  «Инфокогнитивные технологии»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Е. А. Пухова /  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. |

**ЗАДАНИЕ**

**на выполнение курсовой работы (проекта)**

Самохвалову Вячеславу Дмитриевичу,

(ФИО обучающегося)

обучающемуся группы 231-3210,

направления подготовки 09.03.01 «Информатика и вычислительная техника»

по дисциплине «Разработка веб-приложений»

на тему «Разработка веб-приложения для логопедического центра с системой абонементов и личными кабинетами пользователей»

1. Исходные данные к работе (проекту): информационные ресурсы в сети интернет, научные публикации в открытой печати.

2. Содержание задания по курсовой работе (проекту) – перечень вопросов, подлежащих разработке:

|  |  |  |  |
| --- | --- | --- | --- |
| **Разрабатываемый вопрос** | **Объем от всего задания, %** | **Срок выполнения** | **Примечание** |
| Раздел 1. Анализ предметной области |  |  |  |
| Задача 1.1. Обзор существующих программных продуктов по теме работы | 10 | 22.03.2025 |  |
| Задача 1.2. Анализ программных инструментов разработки веб-приложений | 7 | 26.03.2025 |  |
| Задача 1.3. Формулировка цели и задач работы | 8 | 30.03.2025 |  |
| Раздел 2. Проектирование веб-приложения |  |  |  |
| Задача 2.1. Анализ целевой аудитории | 5 | 02.04.2025 |  |
| Задача 2.2. Описание функциональности приложения (диаграмма вариантов использования, user story и т. д.) | 7 | 06.04.2025 |  |
| Задача 2.3. Проектирование модели данных (ER-диаграмма, логическая и физическая схемы БД) | 8 | 10.04.2025 |  |
| Задача 2.4. Разработка макетов страниц (Wireframe) | 5 | 14.04.2025 |  |
| Раздел 3. Разработка веб-приложения |  |  |  |
| Задача 3.1. Разработка базовой структуры приложения и вёрстка шаблонов страниц | 8 | 21.04.2025 |  |
| Задача 3.2. Реализация аутентификации пользователей | 7 | 27.04.2025 |  |
| Задача 3.3. Реализация CRUD-интерфейса для пользователей, абонементов и домашних заданий | 7 | 03.05.2025 |  |
| Задача 3.4. Реализация системы управления абонементами | 12 | 09.05.2025 |  |
| Задача 3.5. Разработка системы домашних заданий | 6 | 15.05.2025 |  |
| Раздел 4. Оформление итогов работы |  |  |  |
| Задача 4.1. Создание Git-репозитория с кодом проекта | 3 | 22.03.2025 |  |
| Задача 4.2. Деплой приложения на хостинг | 4 | 26.05.2025 |  |
| Задача 4.3. Оформление отчёта о проделанной работе | 3 | 26.05.2025 |  |

Руководитель курсовой работы (проекта): ст. преподаватель кафедры «Инфокогнитивные технологии»

|  |  |  |  |
| --- | --- | --- | --- |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | А. С. Кружалов | |
| Дата выдачи задания | | | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. |
| Дата сдачи выполненной работы (проекта) | | | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. |

Задание принял к исполнению

|  |  |  |
| --- | --- | --- |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (И. О. Фамилия) |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc199620412)

[ВВЕДЕНИЕ 5](#_Toc199620413)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc199620414)

[1.1 Обзор существующих программных продуктов 6](#_Toc199620415)

[1.2 Анализ программных инструментов разработки веб-приложений 8](#_Toc199620416)

[1.3 Формулировка цели и задач работы 10](#_Toc199620417)

[2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ 12](#_Toc199620418)

[2.1 Описание работы системы 12](#_Toc199620419)

[2.2 Анализ целевой аудитории 13](#_Toc199620420)

[2.3 Построение диаграммы вариантов использования (Use Case Diagram) 14](#_Toc199620421)

[2.4 Диаграмма пользовательских историй (User Story Diagram) 16](#_Toc199620422)

[2.5 Разработка Wireframe-моделей интерфейса 18](#_Toc199620423)

[3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ПРИЛОЖЕНИЯ 20](#_Toc199620424)

[3.1 Построение ER-модели базы данных 20](#_Toc199620425)

[3.2 Логическая модель базы данных 22](#_Toc199620426)

[3.3 Физическая модель базы данных 25](#_Toc199620427)

[4 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ 26](#_Toc199620428)

[4.1 Структура проекта 26](#_Toc199620429)

[4.2 Используемые паттерны проектирования 27](#_Toc199620430)

[4.3 Файл конфигурации 28](#_Toc199620431)

[4.4 Подключение к базе данных 29](#_Toc199620432)

[4.5 Моделирование данных (ORM-модели) 29](#_Toc199620433)

[4.6 Реализация CRUD API 31](#_Toc199620434)

[5 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ 33](#_Toc199620435)

[5.1 Архитектура клиентского приложения 33](#_Toc199620436)

[5.2 Навигация и маршрутизация 33](#_Toc199620437)

[5.3 Взаимодействие с API 34](#_Toc199620438)

[6 ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ И ЗАПУСК НА СЕРВЕРЕ 36](#_Toc199620439)

[6.1 Размещение кода сервереной части на GitHub и развертывание на сервере 36](#_Toc199620440)

[6.2 Размещение кода клиентской части на GitHub и развертывание на сервере 36](#_Toc199620441)

[6.3 Выбор и настройка доменного имени 37](#_Toc199620442)

[ЗАКЛЮЧЕНИЕ 38](#_Toc199620443)

[СПИСОК ЛИТЕРАТУРЫ 39](#_Toc199620444)

[ПРИЛОЖЕНИЯ 41](#_Toc199620445)

# ВВЕДЕНИЕ

Современные информационные технологии предоставляют широкие возможности для автоматизации процессов в сфере образования и коррекционной педагогики. Особенно актуальным становится внедрение информационных систем в логопедическую практику, где важно учитывать индивидуальные особенности учеников, гибкое расписание занятий и абонементную форму обучения.

Данная работа посвящена разработке информационной системы для логопедического центра. Система ориентирована на три роли пользователей: ученик, преподаватель и администратор. Она предусматривает прикрепление учеников к преподавателям по специальному коду либо через администратора, оформление абонементов на занятия, ведение индивидуального расписания, учет отмен с учетом 24-часового правила, архивирование завершенных абонементов и отслеживание активности по каждому из них.

Отдельное внимание уделяется функциональности управления расписанием и системе домашних заданий. Преподаватель может прикреплять материалы к занятию, а ученик – загружать выполненные задания. Администратор при этом имеет доступ к общему расписанию центра и может контролировать организацию учебного процесса в целом.

В рамках работы были спроектированы логическая и физическая модели базы данных, разработан макет интерфейса и продумана структура системы с учетом реальных требований логопедического центра. Реализация проекта позволит оптимизировать взаимодействие между всеми участниками образовательного процесса, повысить эффективность ведения занятий и улучшить контроль за выполнением заданий.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор существующих программных продуктов

В настоящее время на рынке представлено множество программных решений для организации образовательного процесса и управления учебными центрами. Однако существующие продукты не в полной мере удовлетворяют требованиям логопедического центра, поскольку не предоставляют удобных инструментов для работы с абонементами и интеграции логопедических заданий.

Одной из популярных CRM-систем для учебных центров является Битрикс24. Данная система предоставляет функционал для управления клиентами, ведения расписания и коммуникации с пользователями. Однако в ней отсутствует встроенная поддержка абонементной системы, что делает учет посещений менее удобным. Также в CRM-системах, ориентированных на образовательные учреждения, нет специализированных инструментов для работы с логопедическими заданиями и их проверки.

Существуют также LMS-системы (Learning Management System), предназначенные для организации образовательного процесса. Примеры таких систем – Moodle и Mirapolis. Они позволяют создавать онлайн-курсы, загружать учебные материалы и тестировать учеников. Однако эти платформы ориентированы именно на дистанционное обучение и не содержат функционала для управления расписанием занятий, организации системы абонементов и учета оплат.

Сравнение существующих программных продуктов, а именно CRM-систем и LMS-систем представлено в таблице 1.

Таблица 1 – Сравнение существующих программных продуктов

|  |  |  |  |
| --- | --- | --- | --- |
| Функционал | CRM-системы (Битрикс24 и аналоги) | LMS-системы (Moodle, Mirapolis) | Требуемое веб-приложение |
| Управление расписанием | Есть инструменты для планирования встреч и событий, но нет гибкой настройки расписания для учеников и преподавателей | Отсутствует. Можно загружать курсы и лекции, но нет системы календаря занятий | Полноценная система расписания с возможностью корректировки преподавателем |
| Привязка учеников к преподавателям | В CRM можно вручную назначать клиентов сотрудникам, но нет автоматизированной системы привязки учеников к преподавателям по коду или администратором | Отсутствует. LMS предполагает самостоятельное обучение, поэтому логика привязки учеников к преподавателям не предусмотрена | Ученики могут привязываться к преподавателям через код или администратора |
| Система абонементов | CRM-системы работают по модели разовых платежей или подписок, но нет встроенной системы отслеживания оставшихся занятий | Отсутствует. LMS ориентированы на доступ к курсам, а не на модель посещений по абонементу | Полноценный учет абонементов: количество оставшихся занятий, списание при посещении, сохранение при своевременной отмене |
| Домашние задания и логопедические материалы | В CRM можно прикреплять файлы к задачам, но нет системы проверки и отслеживания выполнения домашнего задания | LMS-системы позволяют загружать задания и проверять их, но они не адаптированы под логопедические занятия | Преподаватель может загружать материалы, а ученик — прикреплять выполненные задания, отслеживать статус проверки |
| Гибкость настройки под логопедический центр | CRM можно адаптировать, но требуется сложная настройка. Основной функционал ориентирован на бизнес-задачи | LMS ориентированы на курсы и тестирование, нет инструментов управления очными занятиями | Полная адаптация под специфику логопедического центра: учет посещаемости, контроль выполнения заданий, удобное расписание |

Таким образом, существующие решения не обеспечивают комплексного подхода к организации логопедического центра. Это подтверждает актуальность разработки специализированного веб-приложения, которое объединит удобный учет абонементов, управление расписанием занятий и систему домашних заданий.

## 1.2 Анализ программных инструментов разработки веб-приложений

Для разработки веб-приложения логопедического центра было рассмотрено несколько стеков технологий. В итоге был выбран стек Flask (REST API) + React.js (Frontend) + MySQL (база данных). Однако существуют и альтернативные решения, которые также широко применяются в веб-разработке.

Для разработки серверной части веб-приложения можно использовать различные фреймворки и платформы. Сравнение наиболее популярных фреймворков представлено в таблице 2.

Таблица 2 – Сравнение популярных серверных фреймворков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Flask (Python) [1] | Django (Python) [2] | Node.js (JavaScript) [3] | Spring Boot (Java) [4] |
| Архитектура | Микрофреймворк, минималистичный | Монолитный фреймворк | Легковесный, событийно-ориентированный | Полноценный фреймворк |
| Гибкость | Позволяет выбирать нужные библиотеки | Встроенные решения (ORM) | Позволяет использовать любые модули | Предлагает мощные встроенные инструменты |
| Поддержка REST API | Отличная поддержка | Встроенная поддержка API | Отличная поддержка, нативная работа с JSON | Хорошая поддержка, но сложная настройка |

Flask был выбран за его гибкость, простоту и удобство интеграции с REST API. Django – хороший вариант, но он более строгий и ориентирован на монолитные приложения. Express (Node.js) подходит для высоконагруженных и асинхронных задач, но уступает Flask в удобстве работы с Python-экосистемой. Spring Boot – мощное решение, но требует больше ресурсов и сложнее в освоении [5].

Фронтенд-приложения могут быть реализованы с помощью различных фреймворков и библиотек, приведенных в таблице 3.

Таблица 3 – Сравнение наиболее популярных фронтенд-фреймворков

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | React.js [6] | Angular [7] | Vue.js [8] |
| Архитектура | Компонентная библиотека | Полноценный фреймворк | Компонентный фреймворк |
| Производительность | Высокая за счет Virtual DOM | Производительность ниже из-за двустороннего связывания | Высокая, но уступает React |
| Гибкость | Позволяет выбрать любые библиотеки | Включает все необходимые модули | Баланс между гибкостью и функционалом |
| Кривая обучения | Средняя, нужно знать JSX и концепции состояния | Средняя, требует знания TypeScript | Легкая, понятный синтаксис |
| Экосистема | Огромное количество готовых решений | Встроенные модули для работы с формами и HTTP | Хорошо подходит для небольших проектов |

React.js был выбран за его гибкость, производительность и широкую экосистему. Angular – мощное решение, но сложнее в освоении и менее гибкое. Vue.js – отличный вариант для небольших проектов, но для масштабируемых решений уступает React [9].

Выбор системы управления базами данных (СУБД) – критически важный этап при разработке веб-приложения. От него зависят скорость работы системы, удобство хранения и обработки данных, поддержка транзакций и возможность масштабирования. Для логопедического центра необходимо хранить информацию о пользователях (учениках, преподавателях, администраторах), абонементах, занятиях, расписании и домашних заданиях. Сравниние популярных СУБД представлено в таблице 4.

Таблица 4 – Сравнение популярных СУБД

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | MySQL [10] | PostgreSQL [11] | MongoDB [12] |
| Тип | Реляционная (SQL) | Реляционная (SQL) | Документо-ориентированная (NoSQL) |
| Производительность | Высокая, но уступает PostgreSQL в сложных запросах | Лучшая производительность для сложных транзакций | Отлично подходит для работы с JSON |
| Гибкость | Хорошая, поддержка сложных запросов | Поддержка JSON, расширенные возможности SQL | Гибкая схема данных, нет строгих связей |
| Масштабируемость | Хорошая, поддерживает кластеризацию | Высокая, поддерживает репликацию | Отличная, горизонтальное масштабирование |

MySQL был выбран за его стабильность, производительность и широкую поддержку. PostgreSQL – мощный вариант, но сложнее в настройке [13].

## Формулировка цели и задач работы

Цель работы: разработка веб-приложения для логопедического центра, включающего систему абонементов и личные кабинеты пользователей (учеников, преподавателей и администраторов). Приложение должно обеспечивать удобную организацию занятий, ведение расписания, управление абонементами и интеграцию логопедических заданий.

Задачи работы:

1. Обзор существующих CRM и LMS систем для учебных центров.
2. Анализ инструментов для разработки веб-приложений (фреймворки, базы данных, библиотеки).
3. Формулировка цели и задач работы.
4. Определение целевой аудитории приложения (ученики, преподаватели, администраторы).
5. Описание функциональности приложения (абонементы, расписания, домашние задания).
6. Проектирование модели данных (ER-диаграмма, схемы базы данных).
7. Разработка wireframe для страниц приложения.
8. Создание структуры приложения и верстка шаблонов страниц.
9. Реализация системы аутентификации пользователей.
10. Реализация CRUD-интерфейса для управления пользователями и абонементами.
11. Разработка системы управления абонементами (создание, отправка в архив, отмена занятий по абонементам).
12. Реализация системы домашних заданий (загрузка и выполнение заданий).
13. Создание Git-репозитория с кодом проекта.
14. Деплой приложения на хостинг.

# 2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

# 2.1 Описание работы системы

Разрабатываемое веб-приложение представляет собой информационную систему для автоматизации процессов логопедического центра. Система ориентирована на упрощение взаимодействия между участниками образовательного процесса, управление абонементами, расписанием и домашними заданиями. В системе предусмотрено три типа пользователей: ученик, преподаватель и администратор, каждый из которых имеет определённый набор функциональных возможностей.

Ролевая структура:

1. Ученик (или родитель ученика) регистрируется в системе и может либо самостоятельно прикрепиться к преподавателю по специальному коду, либо быть добавлен в список преподавателя администратором.
2. Преподаватель управляет своими учениками, создаёт для них расписание занятий, прикрепляет домашние задания, отслеживает прогресс.
3. Администратор обладает расширенными полномочиями: он управляет пользователями, назначает преподавателей ученикам, контролирует глобальное расписание центра и следит за функционированием всей системы.

Ключевая особенность проекта – гибкое управление абонементами. Ученики приобретают абонементы на определённое количество занятий (не менее пяти), при этом согласовывают с преподавателем удобное время. Занятие считается использованным, если не было отменено заранее — за 24 часа до его начала. Это условие необходимо для соблюдения дисциплины и учёта времени преподавателя. В системе реализовано отображение:

* активных абонементов;
* количества оставшихся, использованных и "сгоревших" занятий;
* архива истёкших абонементов.

У каждого ученика и преподавателя формируется собственное расписание. Преподаватель может редактировать расписание индивидуально под каждого ученика. Администратор, в свою очередь, имеет доступ к общему расписанию центра и может видеть все занятия, включая информацию о времени, ученике и преподавателе. Это позволяет избежать пересечений, лучше планировать нагрузку и использовать помещения центра более эффективно.

Преподаватель может прикреплять к будущим занятиям домашние задания в виде логопедических или дидактических материалов. Ученик, в свою очередь, имеет возможность загрузить выполненное задание обратно в систему, обеспечивая обратную связь и повышая качество обучения.

Ключевые преимущества системы:

1. Централизованное управление расписанием и абонементами;
2. Чёткое распределение ролей и полномочий;
3. Прозрачный механизм переноса/отмены занятий;
4. Интеграция домашних заданий в процесс обучения;
5. Возможность масштабирования и подключения новых преподавателей и учеников.

Таким образом, проект направлен на повышение эффективности организации учебного процесса в логопедическом центре, улучшение коммуникации между участниками и автоматизацию ключевых процессов, таких как расписание, учет занятий и абонементов.

## 2.2 Анализ целевой аудитории

Целевую аудиторию веб-приложения логопедического центра можно разделить на три основные группы: ученики (или их родители), преподаватели и администраторы. У каждой группы — свои цели, задачи и сценарии использования системы. Информация, описывающая особенности каждой из групп, представлена в таблице 5.

Таблица 5 – Особенности каждой из категорий пользователей

|  |  |  |
| --- | --- | --- |
| Категории пользователей | Основные задачи | Особенности |
| Ученики / Родители | Просмотр расписания занятий, получение домашних заданий, загрузка выполненных заданий | Интерфейс должен быть простым, понятным, с акцентом на удобство и информативность. |
| Преподаватели | Управление расписанием, назначение и проверка домашних заданий, прикрепление учеников, просмотр информации об абонементах | Пользуются системой регулярно. Необходима удобная навигация по ученикам и занятиям. Поддержка загрузки и прикрепления файлов. |
| Администраторы | Создание и редактирование учетных записей, просмотр глобального расписания центра, прикрепление учеников к преподавателям, контроль абонементов и активности пользователей | Нужен доступ ко всей информации. Интерфейс более функциональный, с возможностями фильтрации и управления пользователями. |

При проектировании интерфейса и функционала приложения важно учитывать потребности всех категорий пользователей и обеспечивать удобный доступ к информации в соответствии с их ролью.

## 2.3 Построение диаграммы вариантов использования (Use Case Diagram)

Диаграмма вариантов использования отражает ключевые действия, доступные пользователям системы в зависимости от их ролей: ученик, преподаватель и администратор. Она позволяет визуализировать поведение системы, очертить границы её функциональности и понять, как пользователи взаимодействуют с основными компонентами веб-приложения.

Основные варианты использования для ученика:

* просмотр текущих и завершённых абонементов;
* просмотр расписания предстоящих занятий;
* получение домашних заданий от преподавателя.

Основные варианты использования для преподавателя:

* назначение занятий для прикреплённых учеников;
* просмотр личного расписания;
* прикрепление новых учеников по коду или вручную;
* прикрепление домашних заданий к занятиям;
* управление списком абонементов учеников.

Основные варианты использования для администратора:

* регистрация и управление пользователями всех ролей;
* прикрепление учеников к преподавателям;
* управление абонементами (создание, продление, удаление);
* просмотр и анализ глобального расписания центра;
* контроль корректности выполнения домашних заданий.

Таким образом, диаграмма вариантов использования демонстрирует взаимодействие трёх ключевых ролей с системой и охватывает все основные сценарии, необходимые для эффективной работы логопедического центра. Этот этап проектирования позволяет обеспечить полноту и непротиворечивость требований при дальнейшем развитии архитектуры и интерфейса веб-приложения.

Диаграмма вариантов использования (Use Case Diagram) представлена на рисунке 1.

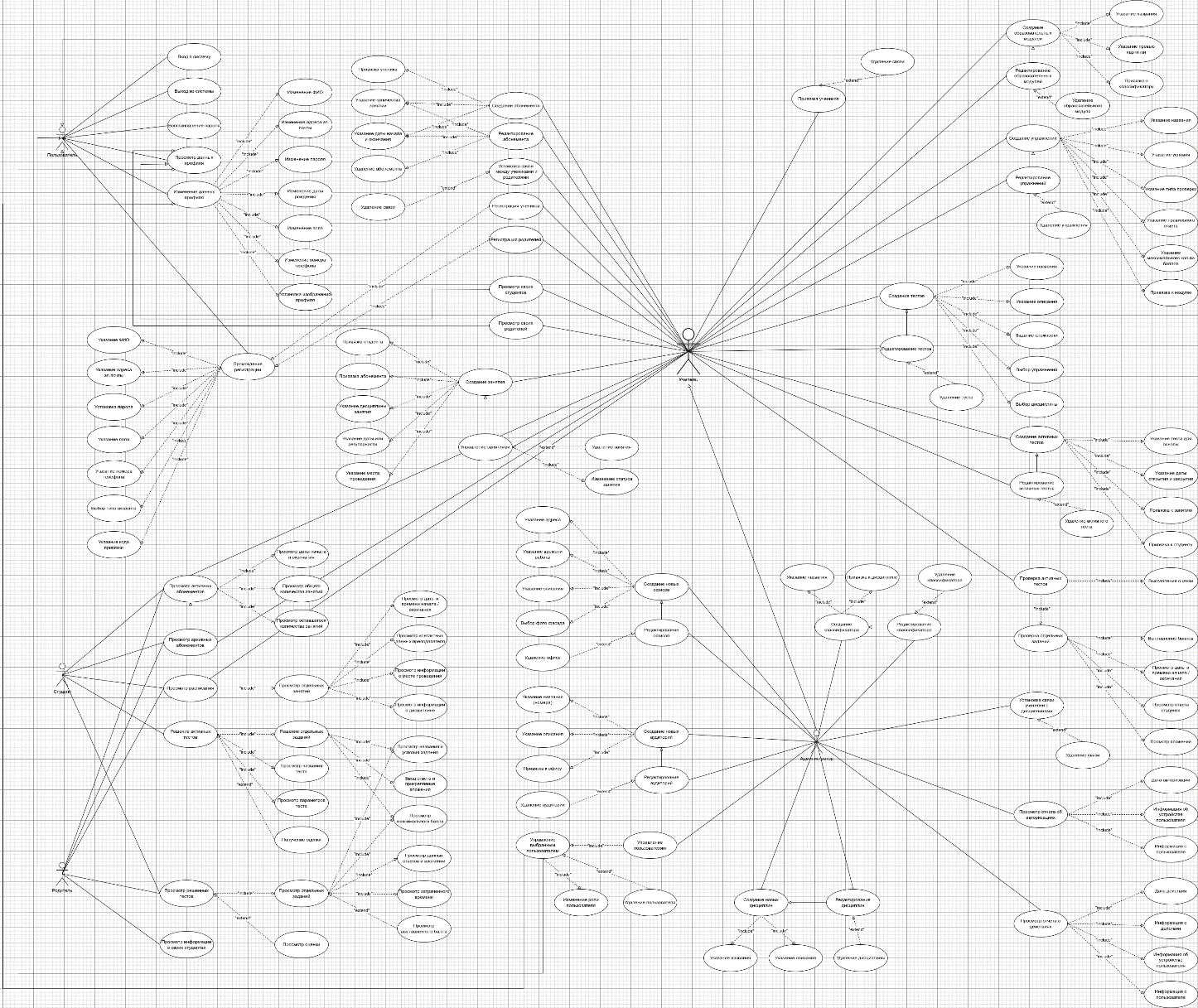


Рисунок 1 – Диаграмма вариантов использования

## 2.4 Диаграмма пользовательских историй (User Story Diagram)

Диаграмма пользовательских историй представляет взаимодействие пользователей с системой с точки зрения их конкретных целей. В отличие от диаграммы вариантов использования, которая описывает доступные действия, User Story диаграмма сосредотачивается на мотивации и ожидаемом результате этих действий, что особенно важно при проектировании ориентированных на пользователя решений.

Пользовательские истории формулируются по шаблону: «Я как [роль], хочу [цель и результат].»

Такой подход позволяет чётко зафиксировать потребности каждой роли в системе и служит основой для реализации функциональности, соответствующей реальным ожиданиям пользователей. Это важно как для формирования технических требований, так и для дальнейшего UX-дизайна и разработки интерфейсов.

Некоторые из основных пользовательских историй для каждой из групп приведены в таблице 6. Полная диаграмма пользовательских историй приведена в Приложении 1.

Таблица 6 – Основные пользовательские истории

|  |  |
| --- | --- |
| Роль | Пользовательская история |
| Пользователь | Я, как обычный пользователь, хочу выполнять вход в систему и выход из нее |
| Пользователь | Я, как обычный пользователь, хочу иметь возможность самостоятельно зарегистрироваться в системе |
| Студент | Я, как обучающийся, хочу просматривать свои активные и архивные абонементы |
| Студент | Я, как обучающийся, хочу просматривать расписание моих занятий |
| Учитель | Я, как учитель, хочу иметь возможность просматривать своих учеников и их родителей |
| Учитель | Я, как учитель, хочу, при создании абонемента, указывать количество приобретаемых занятий, дату начала и окончания срока действия абонемента |
| Администратор | Я, как администратор, хочу просматривать созданные мной офисы (места проведения занятий) |
| Администратор | Я, как администратор, хочу, при создании классификатора, указать его название и привязать его к дисциплине |
| Родитель | Я, как родитель, хочу просматривать все абонементы и информацию по ним для каждого закрепленного за мной обучающегося |
| Родитель | Я, как родитель, хочу видеть дату и время начала и окончания, контактные данные преподавателя, информацию о месте проведения, информацию о дисциплине по каждому занятию в расписании |

Диаграмма пользовательских историй является важным шагом в проектировании приложения, направленного на реальное удовлетворение потребностей участников логопедического центра – учеников, преподавателей и администраторов.

## 2.6 Разработка Wireframe-моделей интерфейса

Для визуализации структуры веб-приложения на начальном этапе проектирования были разработаны Wireframe-модели основных интерфейсов. В качестве инструмента проектирования была выбрана Figma – современный облачный сервис для создания интерфейсов, позволяющий эффективно работать над дизайном в команде. С помощью Figma были созданы «low-fidelity» прототипы ключевых страниц приложения, включая личный кабинет пользователя, систему записи на занятия, раздел с расписанием и интерфейс управления абонементами. Особое внимание уделялось проектированию удобного и интуитивно понятного интерфейса для работы с логопедическими заданиями. Разработанные wireframes послужили основой для дальнейшей детализации дизайн-системы и создания полноценных UI-макетов. Полотно всех созданных Wireframe-моделей приведено на рисунке 2. Отдельно макеты представлены в Приложении 2.

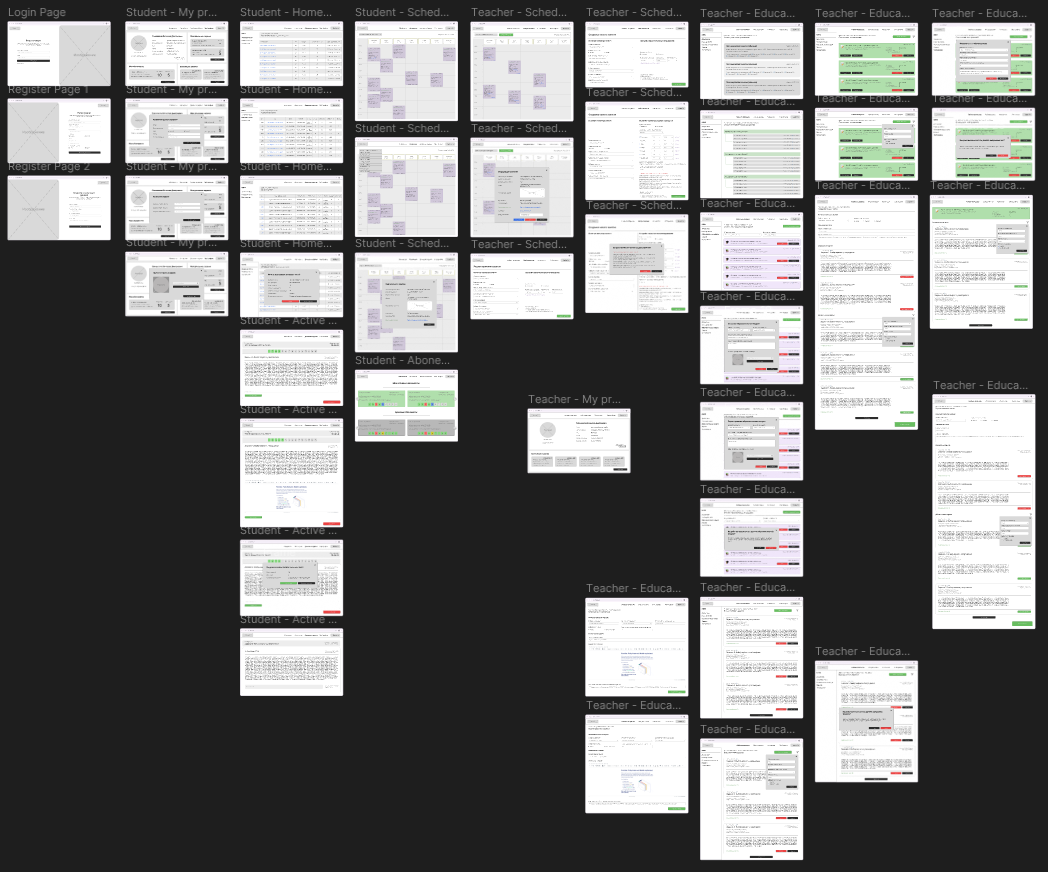


Рисунок 2 – Полотно макетов в Figma

# 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ПРИЛОЖЕНИЯ

## 3.1 Построение ER-модели базы данных

Для проектирования базы данных была разработана ER-диаграмма (Entity-Relationship Diagram), отражающая взаимосвязи между ключевыми сущностями образовательной платформы. Данная база данных обеспечивает хранение и управление информацией о пользователях, преподавателях, учениках, администраторах, занятиях, подписках, образовательных модулях и других объектах системы.

Основные сущности базы данных:

1. Пользователи: главная сущность, содержащая общую информацию обо всех типах пользователей: ФИО, email, хэш пароля, дата рождения, пол, город, телефон, ссылка на фото и уникальный код.
2. Ученики: является подмножеством пользователей. Содержит дополнительную информацию: номер класса и название школы.
3. Преподаватели: также являются пользователями, но с полями: стаж и основное место работы.
4. Администраторы: представляют пользователей с правами управления. Атрибут «уровень доступа» определяет параметры доступа (например, только логи или полный доступ).
5. Филиалы: отражают местоположения, где могут проходить занятия. Содержат информацию об адресе, режиме работы, описании и ссылке на фото. Привязаны к администратору.
6. Кабинеты: местоположения внутри филиалов, в которых проводятся очные занятия. Содержат название, описание, ссылку на филиал и администратора.
7. Дисциплины: представляют собой учебные направления (например, математика, информатика). Содержат название, описание и связаны с администратором.
8. Подписки: связывают учеников и преподавателей в рамках обучения. Включают общее количество занятий, даты начала и окончания подписки, флаг архивации.
9. Занятия: конкретные учебные сессии, привязанные ко времени, ученику, преподавателю, дисциплине и, при необходимости, к кабинету или онлайн-ссылке. Имеют статус (запланировано, завершено, отменено и т.д.).
10. Классификаторы: используются для группировки дисциплин по направлениям. Содержат название и дату создания. Связаны с дисциплиной и администратором.
11. Образовательные модули: представляют собой учебные блоки с описанием и превью. Привязаны к классификатору и преподавателю.

ER-модель базы данных веб-приложения в нотации Питера Чена приведена на рисунке 3.

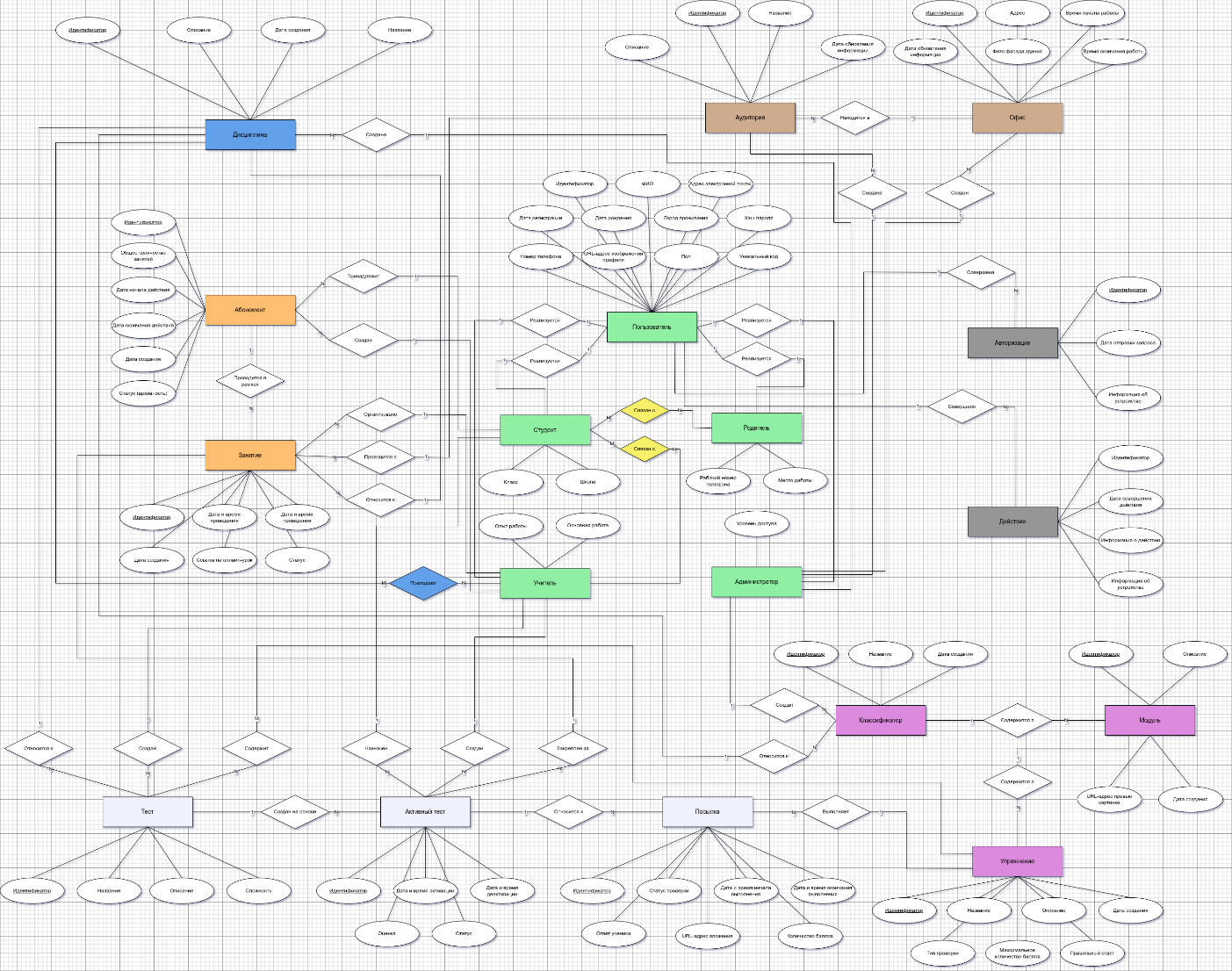


Рисунок 3 – ER-модель базы данных

## 3.2 Логическая модель базы данных

Логическая модель базы данных отражает структуру данных и взаимосвязи между сущностями, не привязанную к конкретной СУБД. Для построения модели использовалась методология IDEF1X (Integration Definition for Information Modeling) – стандарт, широко применяемый для проектирования реляционных баз данных.

Модель обеспечивает представление сущностей, их атрибутов, а также логических связей между ними. Все отношения между сущностями определяются в терминах ключевых зависимостей и правил целостности.

В качестве примера рассмотрим реализацию ролевой модели пользователей. Описание основных сущностей и их атрибутов приведено в таблицах 7-11.

Таблица 7 – Сущность «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Параметры поля | Хранимая информация |
| user\_id | INT | PRIMARY KEY, AUTO\_INCREMENT, NOT NULL | уникальный идентификатор пользователя |
| full\_name | VARCHAR(255) | NOT NULL | полное имя пользователя |
| email | VARCHAR(255) | UNIQUE, NOT NULL | адрес электронной почты |
| password\_hash | VARCHAR(255) | NOT NULL | хеш пароля пользователя |
| birthday | DATE | NULL | дата рождения пользователя |
| gender | ENUM('male','female') | NOT NULL | пол пользователя |
| city | VARCHAR(100) | NULL | город проживания |
| phone\_number | VARCHAR(20) | NULL | контактный телефон |
| profile\_picture\_url | TEXT | NULL | URL аватарки профиля |
| unique\_code | VARCHAR(32) | UNIQUE, NOT NULL | уникальный код пользователя |

Таблица 8 – Сущность «Students»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Параметры поля | Хранимая информация |
| user\_id | INT | PRIMARY KEY, FOREIGN KEY (Users.user\_id), NOT NULL | ссылка на пользователя |
| class\_number | INT | NOT NULL | номер класса ученика |
| school\_name | VARCHAR(255) | NOT NULL | название учебного заведения |

Таблица 9 – Сущность «Teachers»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Параметры поля | Хранимая информация |
| user\_id | INT | PRIMARY KEY, FOREIGN KEY (Users.user\_id), NOT NULL | ссылка на пользователя |
| experience | INT | NULL | стаж работы в годах |
| main\_work | VARCHAR(255) | NULL | основное место работы |

Таблица 10 – Сущность «Administrators»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Параметры поля | Хранимая информация |
| user\_id | INT | PRIMARY KEY, FOREIGN KEY (Users.user\_id), NOT NULL | ссылка на пользователя |
| access\_level | ENUM('low','medium','high') | DEFAULT 'low', NOT NULL | уровень доступа администратора |

Таблица 11 – Сущность «Parents»

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип данных | Параметры поля | Хранимая информация |
| user\_id | INT | PRIMARY KEY, FOREIGN KEY (Users.user\_id), NOT NULL | ссылка на пользователя (родителя) |
| work\_name | VARCHAR(100) | NULL | место работы родителя |
| work\_phone | VARCHAR(20) | NULL | рабочий телефон родителя |

Логическая модель базы данных в нотации IDEF1X приведена на рисунке 4.

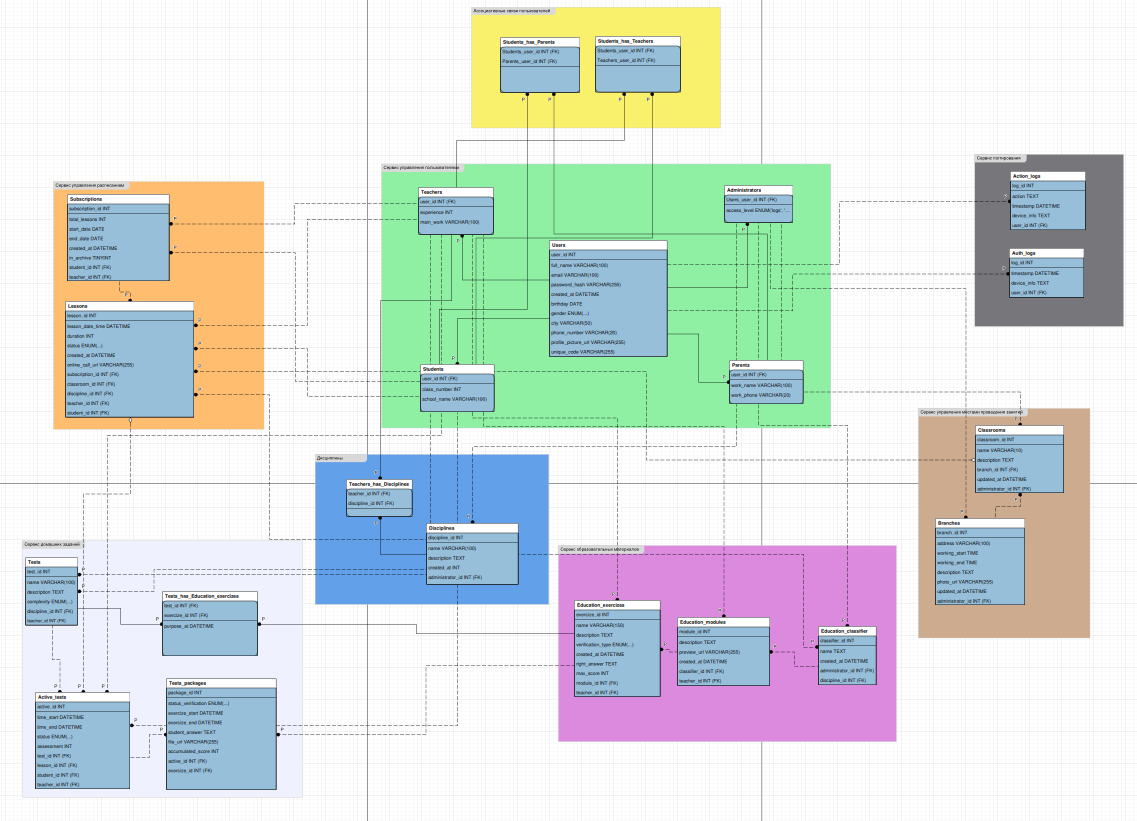


Рисунок 4 – Логическая модель базы данных в нотации IDEF1X

## 3.3 Физическая модель базы данных

Физическая модель представляет собой реализацию логической модели базы данных на конкретной СУБД, в нашем случае – MySQL. Она оформляется в виде SQL-скрипта, который содержит инструкции для создания всех таблиц, определения типов данных, ограничений (ключей, связей, уникальности, NOT NULL и др.), а также возможных индексов и дефолтных значений. Эта модель обеспечивает физическое хранение данных, соблюдение целостности и поддержку производительности при выполнении запросов. SQL-скрипт для создания базы данных приведен в Приложении 3.

# 4 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

## 4.1 Структура проекта

Проект организован в виде набора файлов и папок, что способствует удобству разработки, тестирования и поддержки. Основные компоненты системы разделены по функциональному признаку, что позволяет ясно разграничить области ответственности.

В корне проекта находятся файлы конфигурации и инициализации, а также несколько ключевых директорий:

* «repositories» — содержит модули для работы с данными. Здесь реализованы классы и функции, отвечающие за взаимодействие с базой данных, включая выполнение запросов и логику доступа к сущностям. Такой подход позволяет изолировать бизнес-логику от деталей хранения данных;
* «routes» — содержит описание маршрутов (endpoint-ов) веб-приложения. Здесь прописываются URL-адреса и методы HTTP-запросов (GET, POST и др.), а также связанная с ними логика обработки запросов. Это центральное место для реализации API и взаимодействия с клиентом;
* «utils» — папка с вспомогательными утилитами и общими функциями, которые используются в разных частях проекта. Это могут быть инструменты для валидации данных, генерации ответов, обработки ошибок и другие полезные функции, не относящиеся напрямую к бизнес-логике;
* «migrations» — директория, предназначенная для хранения файлов миграций базы данных. Миграции позволяют последовательно изменять структуру базы данных (создавать таблицы, добавлять колонки и т.п.) без потери данных и с возможностью отката.

Такое разделение кода упрощает масштабирование приложения, облегчает тестирование и позволяет разным разработчикам работать параллельно над разными аспектами проекта.

## 4.2 Используемые паттерны проектирования

При разработке серверной части приложения были использованы несколько проверенных паттернов проектирования, способствующих модульности, читаемости и масштабируемости кода.

Одним из ключевых решений стало применение паттерна Repository. Он позволил вынести всю логику работы с базой данных в отдельный слой, изолировав бизнес-логику от деталей хранения данных. Например, класс «UserRepository» инкапсулирует операции по созданию, поиску и проверке пользователей, предоставляя единый интерфейс для взаимодействия с сущностями пользователей. Такой подход облегчает тестирование и последующую модификацию кода, не затрагивая остальные части приложения.

Также в архитектуре был применён паттерн Flask Blueprint. Это стандартный механизм Flask, позволяющий разделить маршруты на логические модули. В нашем случае, например, маршруты, связанные с аутентификацией, были вынесены в отдельный blueprint «auth\_bp». Такой подход упрощает организацию кода и делает проект более структурированным и удобным для поддержки и расширения.

Для реализации системы авторизации и защиты маршрутов использовалась токен-базированная аутентификация с помощью JWT (JSON Web Token). Эта модель обеспечивает безопасную работу с пользователями, позволяя хранить «access» и «refresh» токены в cookies. Таким образом, безопасность приложения и логика обработки токенов были отделены от основной бизнес-логики, что также соответствует принципам хорошего проектирования.

Кроме того, вся бизнес-логика, связанная с регистрацией, входом в систему и проверками, была вынесена в отдельные методы и классы. Это позволило организовать архитектуру по принципу слоёв, где каждый слой решает строго определённую задачу.

В результате применения этих паттернов код получился модульным, удобным для сопровождения и масштабирования, с чётким разграничением ответственности между различными компонентами системы.

## 4.3 Файл конфигурации

Конфигурация приложения вынесена в отдельный модуль «config.py». В нем определён базовый класс «Config», где задаётся строка подключения к базе данных MySQL через драйвер «mysqlconnector» и отключено отслеживание изменений для оптимизации производительности («SQLALCHEMY\_TRACK\_MODIFICATIONS = False»).

Для разных режимов работы приложения (разработка и продакшн) созданы наследуемые классы «DevelopmentConfig» и «ProductionConfig», которые отличаются флагом «DEBUG». В режиме разработки включён режим отладки, что облегчает поиск ошибок, а в продакшн — отключён для безопасности и производительности. Исходный код файла конфигурации приведен в листинге 1.

Листинг 1 – Файл конфигурации «config.py»

class Config:

SQLALCHEMY\_DATABASE\_URI = 'mysql+mysqlconnector://root:password@localhost:3306/speechtherapistsoffice'

SQLALCHEMY\_TRACK\_MODIFICATIONS = False

class DevelopmentConfig(Config):

DEBUG = True

class ProductionConfig(Config):

DEBUG = False

## 4.4 Подключение к базе данных

Инициализация подключения к базе данных реализована в модуле «db.py». Для работы с базой используется расширение «SQLAlchemy», которое упрощает работу с базой через ORM и позволяет описывать модели в виде классов.

Для соблюдения единого стиля именования индексов, ограничений и внешних ключей используется специальная схема именования, заданная в метаданных «MetaData» через параметр «naming\_convention». Это улучшает читаемость и сопровождение базы данных. Кроме того, объявляется базовый класс моделей «Base», от которого будут наследоваться все сущности, что упрощает управление метаданными и миграциями. Исходный код модуля «db.py» приведен в листинге 2.

Листинг 2 – Модуль для работы с базой данных «db.py»

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.orm import DeclarativeBase

from sqlalchemy import MetaData

class Base(DeclarativeBase):

metadata = MetaData(naming\_convention={

"ix": "ix\_%(column\_0\_label)s",

"uq": "uq\_%(table\_name)s\_%(column\_0\_name)s",

"ck": "ck\_%(table\_name)s\_%(constraint\_name)s",

"fk": "fk\_%(table\_name)s\_%(column\_0\_name)s\_%(referred\_table\_name)s",

"pk": "pk\_%(table\_name)s"

})

db = SQLAlchemy(model\_class=Base)

## 4.5 Моделирование данных (ORM-модели)

Для работы с базой данных в проекте используется библиотека «SQLAlchemy», которая позволяет использовать объектно-реляционное отображение (ORM) для моделирования сущностей. Такой подход обеспечивает более чистую архитектуру, упрощает взаимодействие с базой данных и повышает читаемость и сопровождаемость кода.

Были реализованы модели для всех основных сущностей системы: пользователей, ролей, заданий, абонементов, филиалов и офисов, а также вспомогательных таблиц. Все модели определены в едином модуле «models.py». Для примера рассмотрим структуру моделей пользователей и ролей, которые демонстрируют взаимосвязь между таблицами и применение декларативного стиля:

* + 1. Модель «User» содержит основные данные, необходимые для регистрации и аутентификации пользователей: полное имя, электронную почту, пароль, дату рождения, пол, город проживания, номер телефона и уникальный код. Также в модели предусмотрены связи с таблицами ролей: студент, родитель, учитель и администратор. Эти связи позволяют быстро определить тип пользователя и получить доступ к дополнительной информации, специфичной для его роли. Фрагмент кода, описывающего модель «User» представлен в листинге 3.

Листинг 3 – Фрагмент кода для модели «User»

class User(Base):

\_\_tablename\_\_ = "Users"

user\_id: Mapped[int] = mapped\_column(primary\_key=True, autoincrement=True)

full\_name: Mapped[str] = mapped\_column(String(100))

email: Mapped[str] = mapped\_column(String(100), unique=True)

...

student: Mapped["Student"] = relationship(back\_populates="user", uselist=False)

teacher: Mapped["Teacher"] = relationship(back\_populates="user", uselist=False)

parent: Mapped["Parent"] = relationship(back\_populates="user", uselist=False)

administrator: Mapped["Administrator"] = relationship(back\_populates="user", uselist=False)

1. Модели ролей «Student», «Teacher», «Administrator», «Parent»: каждая из ролей хранится в отдельной таблице и имеет внешний ключ, ссылающийся на таблицу пользователей. Таким образом обеспечивается единый центр хранения информации о пользователях при возможности расширения функциональности под каждую отдельную роль. Например, модель «Student», приведенная в листинге 4, хранит сведения о классе и школе.

Листинг 4 – Фрагмент кода для модели «Student»

class Student(Base):

\_\_tablename\_\_ = "Students"

user\_id: Mapped[int] = mapped\_column(ForeignKey("Users.user\_id"), primary\_key=True)

class\_number: Mapped[int | None] = mapped\_column(Integer, nullable=True)

school\_name: Mapped[str | None] = mapped\_column(String(100), nullable=True)

user: Mapped["User"] = relationship(back\_populates="student")

Для доступа к данным используются специальные классы-репозитории. В «UserRepository» реализованы основные методы для работы с пользователями: получение всех пользователей, поиск по email или ID, обновление данных, удаление и аутентификация. Рассмотрим метод «create\_user\_with\_role», приведенный в листинге 5. Данный метод автоматически создает запись в таблице соответствующей роли в зависимости от значения поля «selectedRole».

Листинг 5 – Метод «create\_user\_with\_role»

def create\_user\_with\_role(self, user\_data: dict) -> User:

...

if role == 'student':

self.session.add(Student(...))

elif role == 'teacher':

self.session.add(Teacher(...))

...

## 4.6 Реализация CRUD API

Для взаимодействия клиентской части с сервером в проекте был реализован REST API [14]. Всего реализовано множество эндпоинтов: регистрация, авторизация, выход из системы, обновление токена, проверка email и другие. В качестве примера рассмотрим один из них — эндпоинт регистрации пользователя («/register»). Фрагмент кода данного эндпоинта приведен в листинге 6.

Листинг 6 – Фрагмент кода эндпоинта «/register»

@auth\_bp.route('/register', methods=['POST'])

def register():

try:

data = request.get\_json()

...

data['unique\_code'] = generate\_unique\_code()

user = repo.create\_user\_with\_role(data)

access\_token = create\_access\_token(identity=str(user.user\_id))

refresh\_token = create\_refresh\_token(identity=str(user.user\_id))

response = jsonify({

"msg": "Пользователь успешно зарегистрирован",

"user\_id": user.user\_id,

"role": data['selectedRole']

})

set\_access\_cookies(response, access\_token)

set\_refresh\_cookies(response, refresh\_token)

return response

...

Данный обработчик выполняет проверку обязательных полей, преобразует дату рождения в нужный формат, генерирует уникальный код пользователя, сохраняет данные в базе через репозиторий и возвращает JWT-токены в cookies. Для генерации токенов используется библиотека «flask\_jwt\_extended» [15]. Такой подход обеспечивает безопасную авторизацию пользователей и удобную работу с токенами на клиенте.

# 5 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

## Архитектура клиентского приложения

Клиент организован по модульному принципу. Все запросы к серверу находятся в папке «src/api/», разделённой на логические блоки, например: авторизация («auth»), регистрация («register»), работа с пользователем («users»). Это упрощает поддержку сетевой логики.

Папка «components/» содержит переиспользуемые элементы интерфейса, например: кнопки, хедер, футер. Стили хранятся рядом с компонентами в виде модульных CSS-файлов.

Папка «pages/» включает основные страницы приложения, например: «home», «sign-in», «sign-up», «profile». Каждая папка содержит JSX-файл страницы и файл стилей.

В «assets/» размещены изображения, сгруппированные по назначению, а в «utils/» — вспомогательные функции.

## Навигация и маршрутизация

Для управления переходами между страницами в клиентской части используется библиотека «react-router-dom». Основная настройка маршрутов происходит в компоненте «App», где определяются пути и соответствующие компоненты страниц.

Каждый путь отображает определённую страницу, примеры наиболее важных маршрутов приведены ниже:

* «/» — главная страница
* «/sign-up» — регистрация
* «/sign-in» — вход
* «/profile» — профиль пользователя

Также вне зависимости от текущего маршрута отображаются компоненты шапки и подвала сайта.

Пример реализации маршрутизации в компоненте «App» приведен в листинге 7.

Листинг 7 – Маршрутизация в компоненте «App»

function App() {

return (

<div className="app">

<AppHeader />

<main>

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/sign-up" element={<SignUpPage />} />

<Route path="/sign-in" element={<SignInPage />} />

<Route path="/profile" element={<ProfilePage />} />

</Routes>

</main>

<AppFooter />

</div>

);

}

## Взаимодействие с API

Клиентская часть приложения обменивается данными с сервером через HTTP-запросы к REST API. Для удобства взаимодействия и логического разделения, функции для работы с API сгруппированы по директориям: «auth», «register», «users» и др., каждая из которых отвечает за свою область.

Запросы выполняются с помощью стандартного fetch. Все функции настроены на отправку необходимых заголовков и данных в формате JSON, а также обрабатывают полученные ответы.

Например, функция «checkEmail», которая приведена в листинге 8, используется для проверки, зарегистрирован ли уже указанный email.

Листинг 8 – Функция «checkEmail»

export async function checkEmail(data) {

try {

const response = await fetch('http://localhost:5000/auth/check\_email', {

method: 'POST',

credentials: 'include',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(data)

});

const responseData = await response.json();

return responseData.msg;

} catch (error) {

console.log('Request failed:', error);

}

}

# 6 ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ И ЗАПУСК НА СЕРВЕРЕ

## 6.1 Размещение кода сервереной части на GitHub и развертывание на сервере

По завершении разработки серверной части система была размещена в публичном репозитории на GitHub. Ссылка на репозиторий: https://github.com/qelio/SpeechTherapistOffice-Backend. В репозитории были четко организованы папки по логическим модулям – маршруты, модели, репозитории, утилиты и другие части приложения.

Для развертывания проекта использовался сервер с операционной системой Linux. На сервере были предварительно установлены необходимые компоненты: интерпретатор Python, система управления базами данных MySQL, а также инструменты для запуска веб-приложений в продакшене.

После переноса кода на сервер была выполнена настройка среды: установлены зависимости проекта, создано виртуальное окружение, настроены переменные окружения с доступами к базе данных и ключам безопасности.

## 6.2 Размещение кода клиентской части на GitHub и развертывание на сервере

По завершении разработки клиентской части приложение было размещено в отдельном публичном репозитории на GitHub. Ссылка на репозиторий: https://github.com/qelio/SpeechTherapistOffice-Frontend. В репозитории структура была организована по стандартным папкам — компоненты, страницы, стили, сервисы и прочие элементы фронтенд-приложения.

Для развертывания использовался тот же сервер с операционной системой Linux. На сервере была предварительно настроена среда для запуска фронтенд-приложения: установлен Node.js и менеджер пакетов npm, а также настроен веб-сервер для обслуживания статических файлов и проксирования запросов к серверной части.

После копирования исходного кода на сервер была выполнена сборка проекта с помощью команды «npm run build». Готовые статические файлы были размещены в директории, доступной веб-серверу. Для обеспечения корректной работы приложения были настроены переменные окружения и параметры веб-сервера, обеспечивающие взаимодействие с серверной частью и безопасность передачи данных.

## 6.3 Выбор и настройка доменного имени

Для обеспечения удобного доступа пользователей к приложению было зарегистрировано доменное имя «cabinet-logopeda.ru». Выбор домена основан на простоте запоминания, отражении сути сервиса и ориентированности на целевую аудиторию.

Регистрация доменного имени была выполнена через одного из официальных регистраторов доменов, поддерживающих зону .ru. После регистрации домена была проведена настройка DNS-записей для связывания доменного имени с IP-адресом сервера, на котором размещены клиентская и серверная части приложения.

В DNS-конфигурации были добавлены записи типа A, указывающие на основной IP-адрес сервера, а также записи CNAME и MX для обеспечения работы почтовых сервисов и поддоменов. Для повышения безопасности и надежности доступа к приложению была настроена поддержка HTTPS с помощью сертификатов, полученных через бесплатный сервис «Let's Encrypt».

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было разработан веб-приложение для логопедического центра с системой абонементов и личными кабинетами пользователей, учитывающее специфику работы центра и потребности его клиентов. Реализованы основные функциональные возможности, такие как регистрация и разграничение ролей (ученик, преподаватель, администратор), управление абонементами с учетом правил посещения и отмен занятий, а также система домашних заданий.

Созданная информационная система позволяет автоматизировать процессы взаимодействия между преподавателями и учениками, облегчить работу администратора и повысить качество обслуживания клиентов. Использование абонементной системы с учетом возможности переноса занятий при своевременном уведомлении обеспечивает гибкость и удобство для пользователей.

Веб-приложение разработано с применением современных технологий и методов, что обеспечивает его расширяемость и дальнейшее развитие. Реализация личных кабинетов и прозрачной системы учёта занятий способствует улучшению коммуникации и повышению эффективности работы логопедического центра.

Таким образом, поставленные цели работы были достигнуты, и созданное приложение может быть внедрено в деятельность логопедического центра для оптимизации рабочих процессов и повышения уровня сервиса.

СПИСОК ЛИТЕРАТУРЫ

1. Flask: веб-фреймворк на Python [Электронный ресурс]. — 2024. — URL: https://flask.palletsprojects.com/ (дата обращения: 01.03.2024).
2. Django: документация фреймворка [Электронный ресурс]. — 2024. — URL: https://docs.djangoproject.com/ (дата обращения: 10.03.2024).
3. Express.js: фреймворк для Node.js [Электронный ресурс]. — 2024. — URL: https://expressjs.com/ (дата обращения: 15.03.2024).
4. Spring Boot: руководство разработчика [Электронный ресурс]. — 2024. — URL: https://spring.io/projects/spring-boot (дата обращения: 15.03.2024).
5. Обзорный анализ фреймворков Python [Электронный ресурс] / Tproger. — 2023. — URL: https://tproger.ru/articles/obzornyj-analiz-python-veb-frejmvorkov (дата обращения: 15.03.2024).
6. React: официальная документация [Электронный ресурс]. — 2024. — URL: https://react.dev/ (дата обращения: 20.03.2024).
7. Angular: руководство разработчика [Электронный ресурс]. — 2024. — URL: https://angular.io/docs (дата обращения: 15.03.2024).
8. Vue.js: документация [Электронный ресурс]. — 2024. — URL: https://vuejs.org/guide/ (дата обращения: 21.03.2024).
9. Что нужно знать о популярных JS-фреймворках [Электронный ресурс] / Habr. — 2024. — URL: https://habr.com/ru/companies/yandex\_praktikum/articles/533702/ (дата обращения: 30.03.2024).
10. MySQL: официальная документация [Электронный ресурс]. — 2024. — URL: https://dev.mysql.com/doc/ (дата обращения: 30.03.2024).
11. PostgreSQL: руководство администратора [Электронный ресурс]. — 2024. — URL: https://www.postgresql.org/docs/ (дата обращения: 30.03.2024).
12. MongoDB: документация [Электронный ресурс]. — 2024. — URL: https://www.mongodb.com/docs/ (дата обращения: 11.03.2024).
13. Сравнение SQL- и NoSQL-баз данных [Электронный ресурс] / Habr. — 2024. — URL: https://habr.com/ru/companies/ruvds/articles/727474/ (дата обращения: 15.03.2024).
14. Лучшие практики REST API [Электронный ресурс] / Microsoft Docs. — 2024. — URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата обращения: 15.03.2024).
15. Flask-JWT-Extended: документация по аутентификации [Электронный ресурс]. - 2024. - URL: https://flask-jwt-extended.readthedocs.io/en/stable/ (дата обращения: 20.04.2024).

# ПРИЛОЖЕНИЯ

**Приложение 1 – Диаграмма вариантов использования**



1. Я, как обычный пользователь, хочу выполнять вход в систему и выход из нее
2. Я, как обычный пользователь, хочу иметь возможность самостоятельно зарегистрироваться в системе
3. Я, как обычный пользователь, хочу иметь возможность восстановить свой забытый пароль
4. Я, как обычный пользователь, хочу иметь возможность просматривать и редактировать данные своего профиля



1. Я, как обучающийся, хочу иметь весь функционал, который имеет обычный пользователь
2. Я, как обучающийся, хочу просматривать свои активные и архивные абонементы
3. Я, как обучающийся, хочу видеть дату начала и окончания, оставшееся количество занятий и общее количество занятий по каждому абонементу
4. Я, как обучающийся, хочу просматривать расписание моих занятий
5. Я, как обучающийся, хочу видеть дату и время начала и окончания, контактные данные преподавателя, информацию о месте проведения, информацию о дисциплине по каждому занятию
6. Я, как обучающийся, хочу иметь возможность решать активные тесты, назначенные преподавателем
7. Я, как обучающийся, хочу просматривать информацию о каждом активном тесте
8. Я, как обучающийся, хочу в каждом упражнении из активного теста видеть название и условие задания, а также его параметры
9. Я, как обучающийся, хочу в каждом упражнении из активного теста иметь возможность дать либо краткий, либо развернутый ответ, а также прикрепить вложение
10. Я, как обучающийся, хочу иметь возможность получить оценку за выполненный активный тест
11. Я, как обучающийся, хочу просматривать решенные мной активные тесты, а также видеть все упражнения из этого теста, правильный ответ к каждому упражнению, мои ответы к упражнению, набранные мной баллы и максимальные баллы



1. Я, как учитель, хочу иметь весь функционал, который имеет обычный пользователь
2. Я, как учитель, хочу иметь возможность регистрировать учеников и родителей в системе
3. Я, как учитель, хочу иметь возможность привязывать учеников к своему профилю
4. Я, как учитель, хочу иметь возможность отвязывать учеников от своего профиля
5. Я, как учитель, хочу иметь возможность устанавливать и разрывать связь между учеником и родителем
6. Я, как учитель, хочу иметь возможность просматривать своих учеников и их родителей
7. Я, как учитель, хочу видеть созданные мной абонементы
8. Я, как учитель, хочу создавать новые абонементы для посещения занятий
9. Я, как учитель, хочу, при создании абонемента, привязывать его к одному из своих учеников
10. Я, как учитель, хочу, при создании абонемента, указывать количество приобретаемых занятий, дату начала и окончания срока действия абонемента
11. Я, как учитель, хочу иметь возможность редактировать или удалять каждый абонемент
12. Я, как учитель, хочу видеть расписание своих занятий
13. Я, как учитель, хочу добавлять новые занятия в расписание
14. Я, как учитель, хочу, при создании занятия, иметь возможность привязать его к ученику и абонементу, указать название дисциплины, указать дату и регулярность проведения занятия, указать место проведения занятия
15. Я, как учитель, хочу иметь возможность редактировать или удалять отдельные занятия
16. Я, как учитель, хочу видеть все образовательные модули
17. Я, как учитель, хочу создавать свои образовательные модули
18. Я, как учитель, хочу, при создании образовательного модуля, указать его название и загрузить картинку, а также привязать его к классификатору
19. Я, как учитель, хочу, иметь возможность редактировать или удалять образовательный модуль
20. Я, как учитель, хочу видеть все упражнения
21. Я, как учитель, хочу, создавать свои упражнения
22. Я, как учитель, хочу, указывать название, условие (текст + картинки), тип проверки, правильный ответ, максимальное количество баллов для каждого упражнения
23. Я, как учитель, хочу, привязать определенное упражнение к образовательному модулю
24. Я, как учитель, хочу, иметь возможность редактировать или удалить определенное упражнение
25. Я, как учитель, хочу видеть все созданные тесты
26. Я, как учитель, хочу, создавать тесты, используя, созданные мной и не только мной, упражнения
27. Я, как учитель, хочу, при создании теста, указывать его название, описание, уровень сложности
28. Я, как учитель, хочу иметь возможность закреплять тест за определенной дисциплиной
29. Я, как учитель, хочу иметь возможность редактировать или удалить созданный мной тест, и понимаю, что удаление одного теста, повлечет за собой удаление всех активных тестов
30. Я, как учитель, хочу назначать созданные мной тесты ученикам (создавать активные тесты), а также привязывать тест к занятиям в расписании
31. Я, как учитель, хочу, при назначении теста ученику, указать дату и время открытия и закрытия теста
32. Я, как учитель, хочу видеть все назначенные мной тесты, прогресс их выполнения и ответы обучающегося
33. Я, как учитель, хочу проверять назначенные мной тесты, и выставлять оценку
34. Я, как учитель, хочу иметь возможность редактировать или удалять назначенные мной тесты

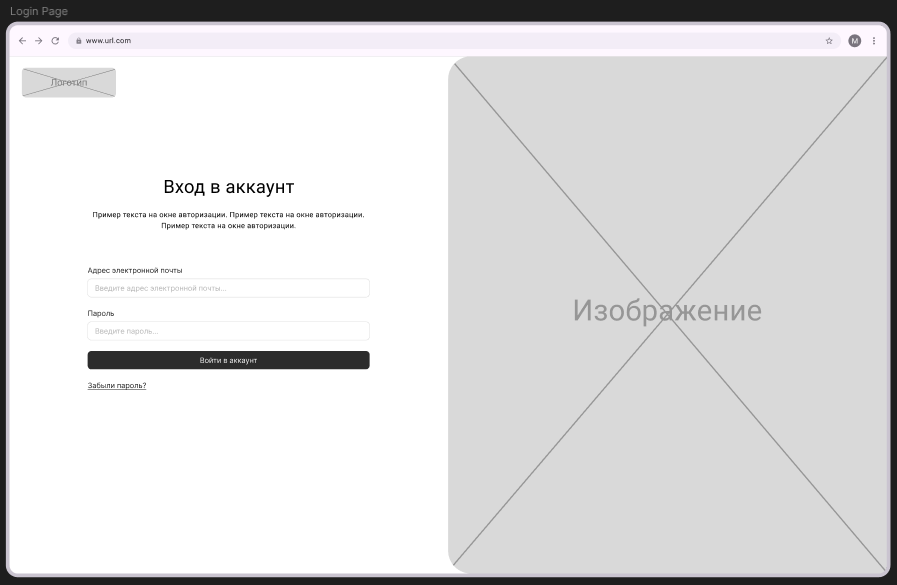


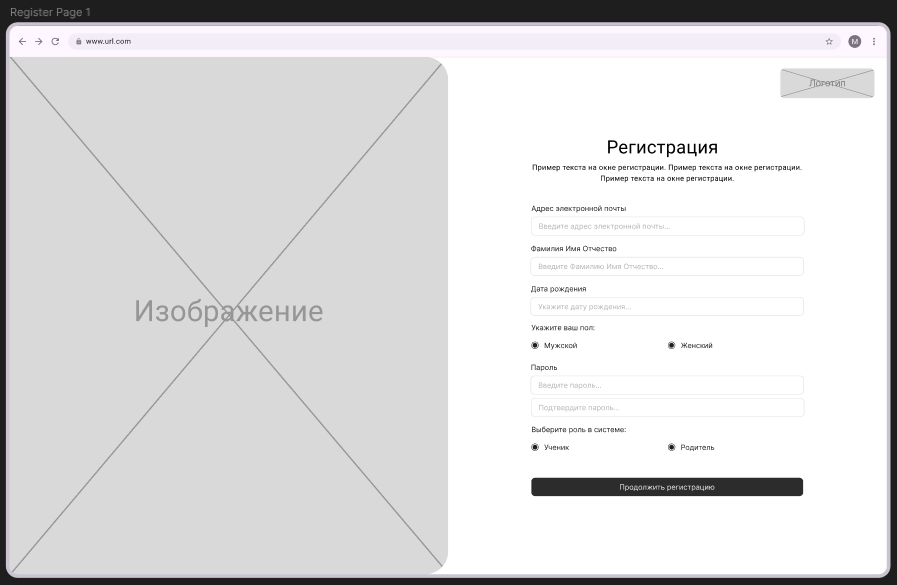
1. Я, как администратор, хочу иметь весь функционал, который имеет обычный пользователь
2. Я, как администратор, хочу иметь весь функционал, который имеет учитель, но при этом я должен иметь полный доступ (просмотр, создание, редактирование и удаление) ко всем компонентам системы, таким, как: образовательные модули, созданные упражнения, созданные или назначенные тесты, абонементная система, расписание и занятия и т.д.
3. Я, как администратор, хочу просматривать созданные мной офисы (места проведения занятий)
4. Я, как администратор, хочу создавать новые офисы (места проведения занятий)
5. Я, как администратор, хочу, при создании офиса, иметь возможность указать его адрес, время работы, описание, загрузить фото фасада здания
6. Я, как администратор, хочу иметь возможность редактировать и удалять созданные мной офисы
7. Я, как администратор, хочу видеть все созданные мной аудитории
8. Я, как администратор, хочу создавать новые аудитории и закреплять их за определенным офисом
9. Я, как администратор, хочу, при создании аудитории, указать его название или номер, и описание
10. Я, как администратор, хочу иметь возможность редактировать или удалять созданные мной аудитории
11. Я, как администратор, хочу видеть все созданные классификаторы
12. Я, как администратор, хочу создавать новые классификаторы
13. Я, как администратор, хочу, при создании классификатора, указать его название и привязать его к дисциплине
14. Я, как администратор, хочу иметь возможность редактировать и удалять созданные мной классификаторы
15. Я, как администратор, хочу видеть все созданные дисциплины
16. Я, как администратор, хочу создавать новые дисциплины
17. Я, как администратор, хочу, при создании дисциплины, указать ее название и описание
18. Я, как администратор, хочу иметь возможность редактировать созданные ранее дисциплины
19. Я, как администратор, хочу привязывать определенных преподавателей к определенным дисциплинам
20. Я, как администратор, хочу отвязывать определенных преподавателей от определенных дисциплин
21. Я, как администратор, хочу иметь полный контроль над всеми пользователями: изменять различные параметры их профилей, удалять профили пользователей
22. Я, как администратор, хочу видеть историю определенных действий и авторизаций всех пользователей в системе

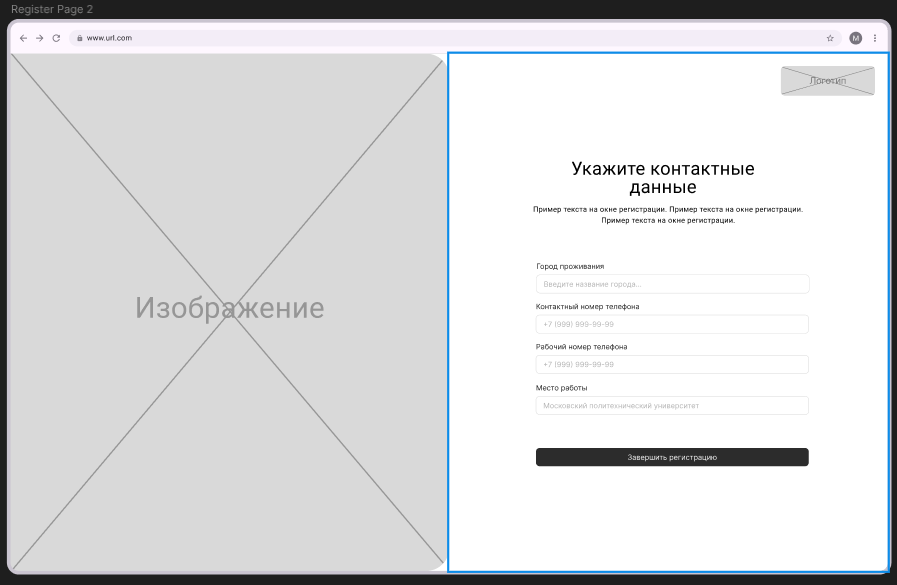


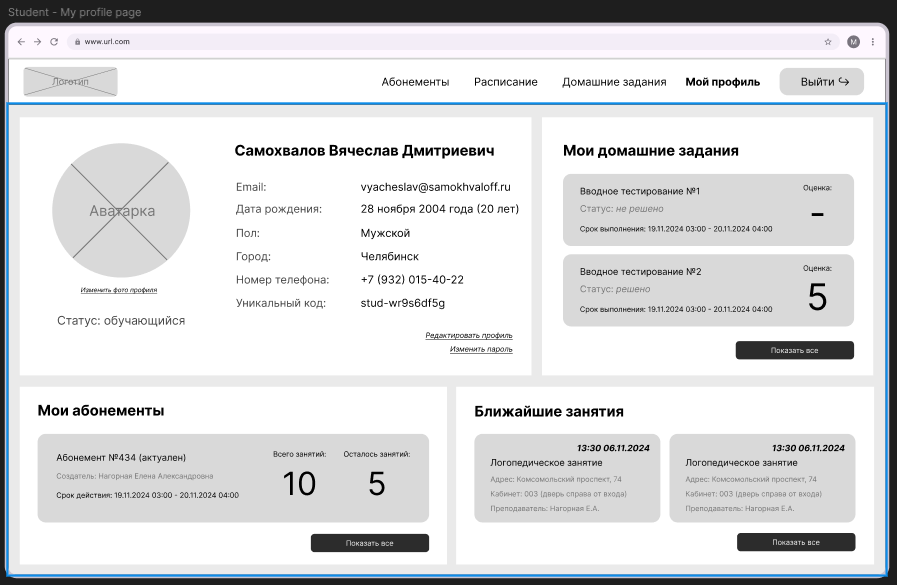
1. Я, как родитель, хочу иметь весь функционал, который имеет обычный пользователь
2. Я, как родитель, хочу видеть всех закрепленных за мной обучающихся
3. Я, как родитель, хочу просматривать все абонементы и информацию по ним для каждого закрепленного за мной обучающегося
4. Я, как родитель, хочу видеть общее расписание по всем моим обучающимся
5. Я, как родитель, хочу видеть дату и время начала и окончания, контактные данные преподавателя, информацию о месте проведения, информацию о дисциплине по каждому занятию в расписании
6. Я, как родитель, хочу видеть сводку по решенным и назначенным тестам моих обучающихся

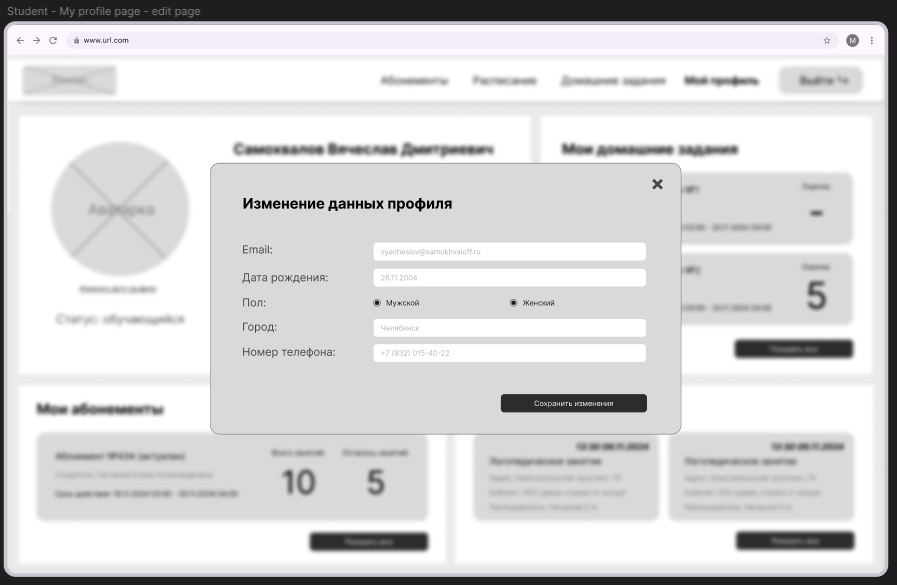
**Приложение 2 – Wireframe-макеты веб-приложения**

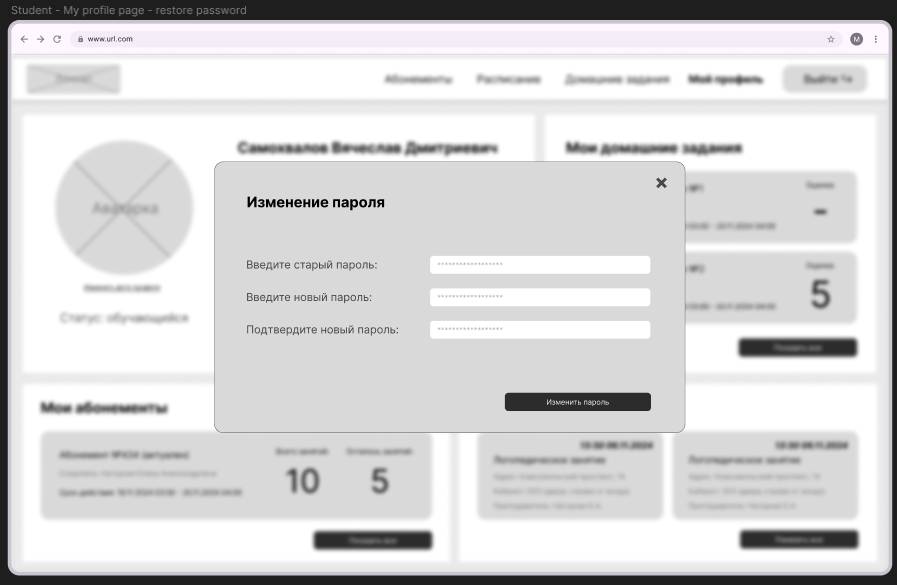
****

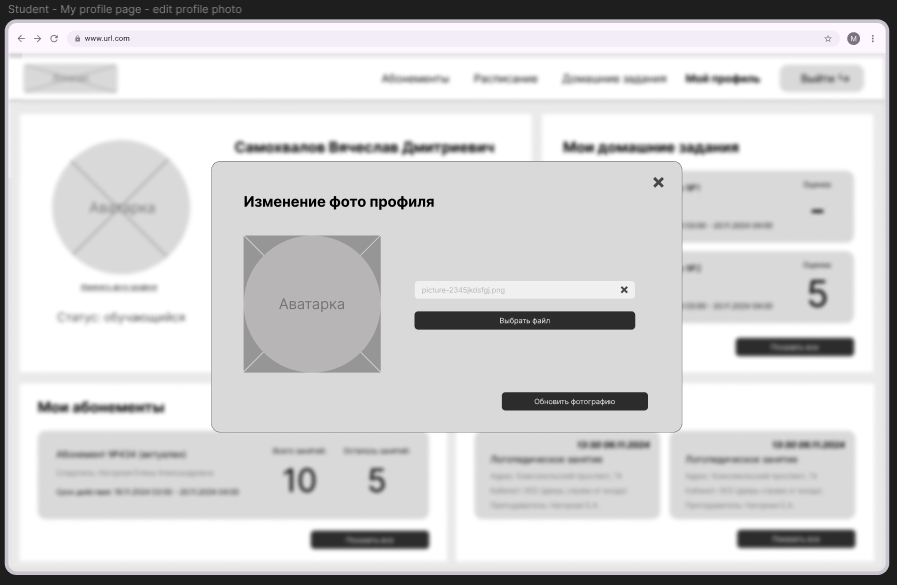
****

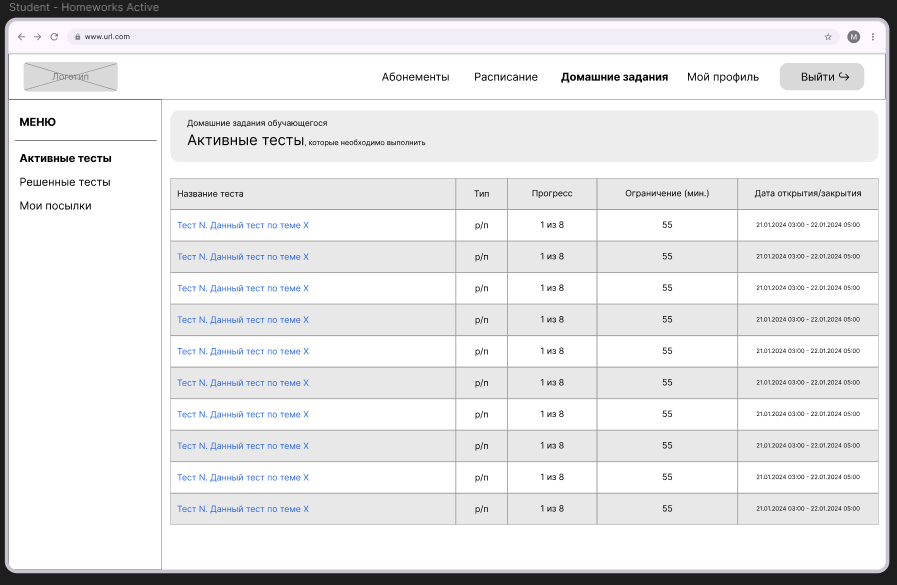
****

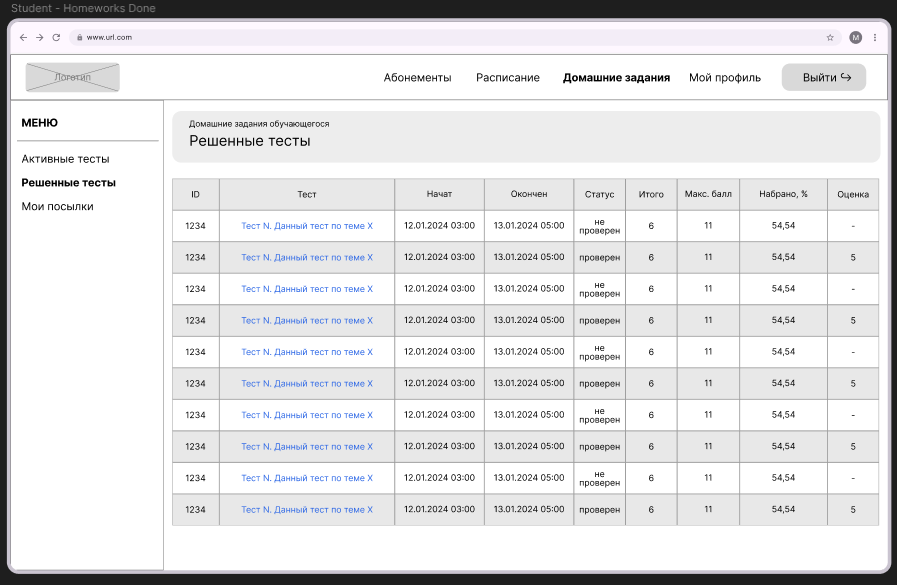
****

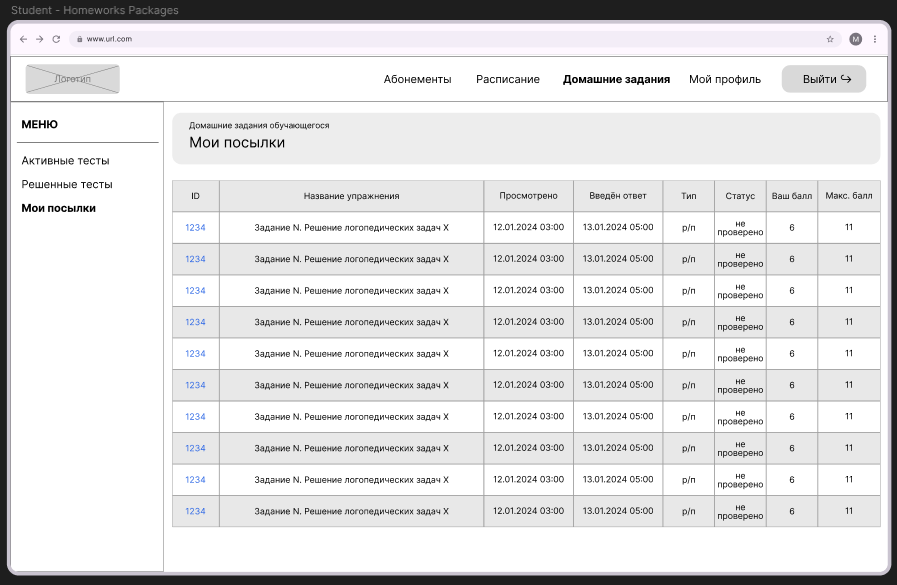
****

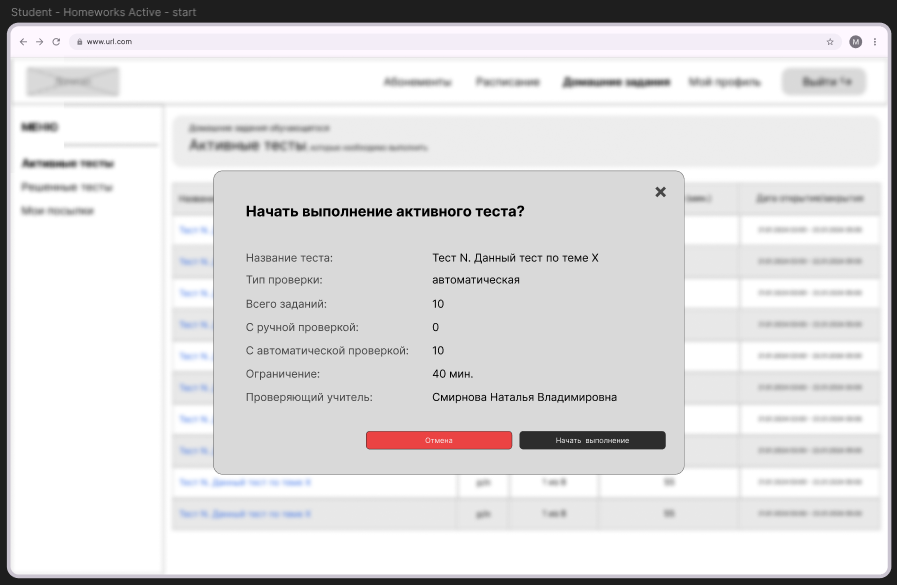
****

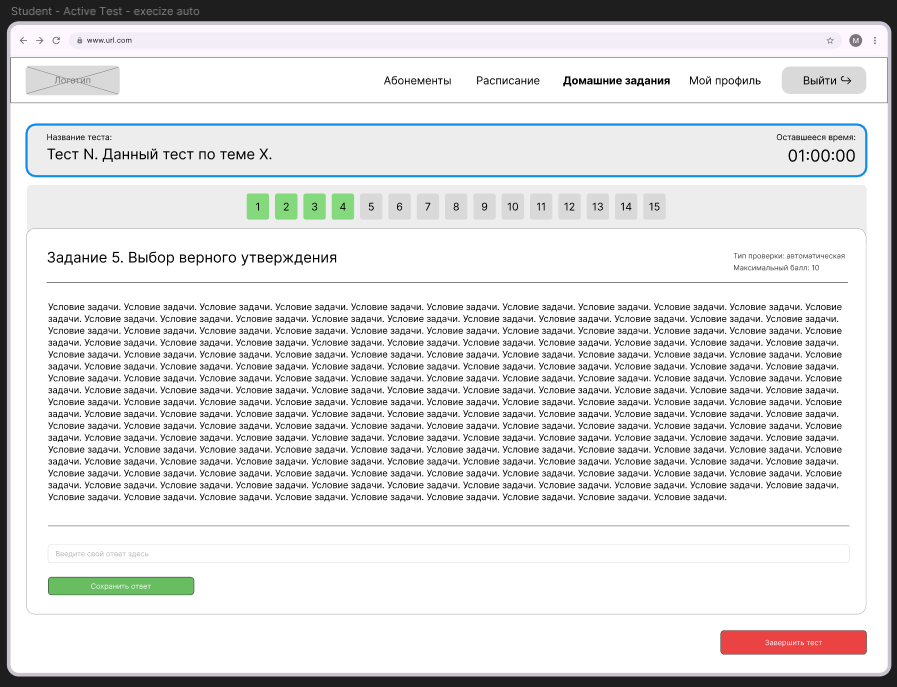
****

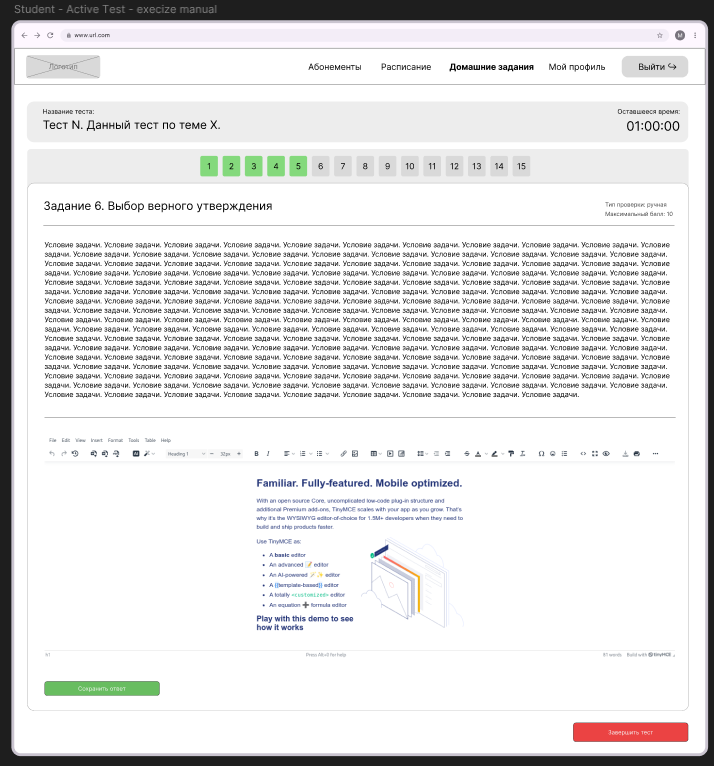
****

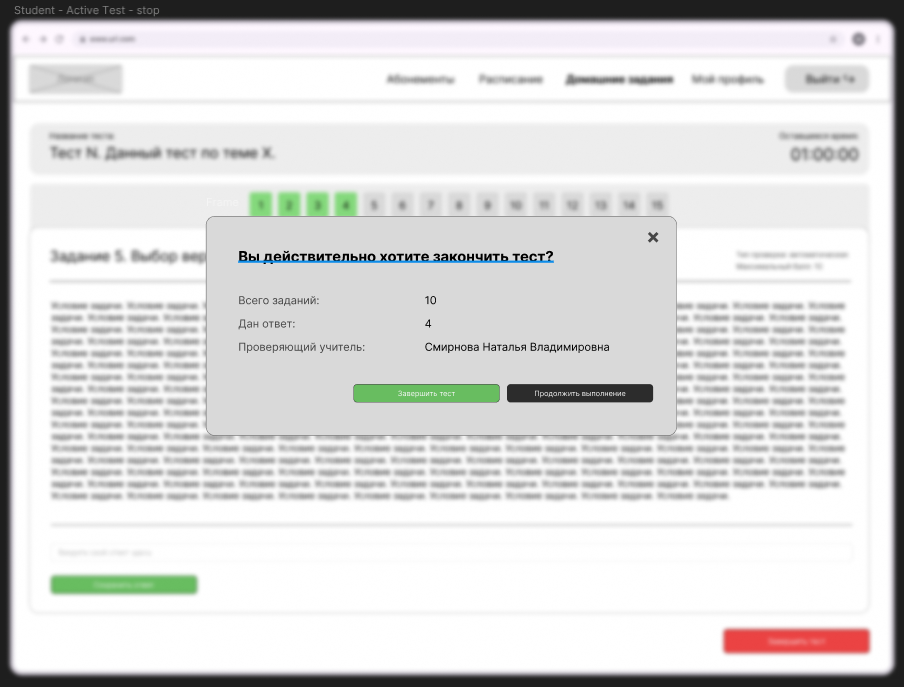
****

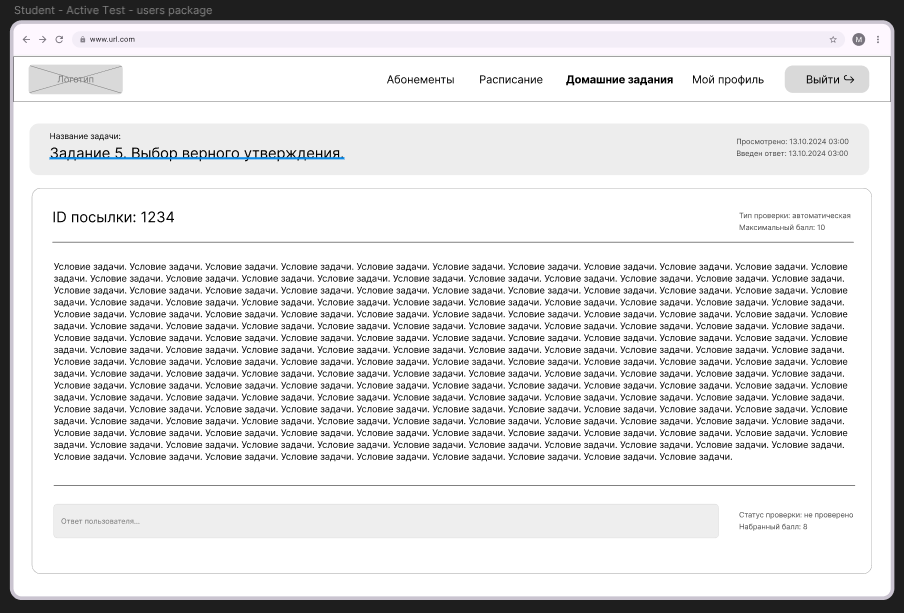
****

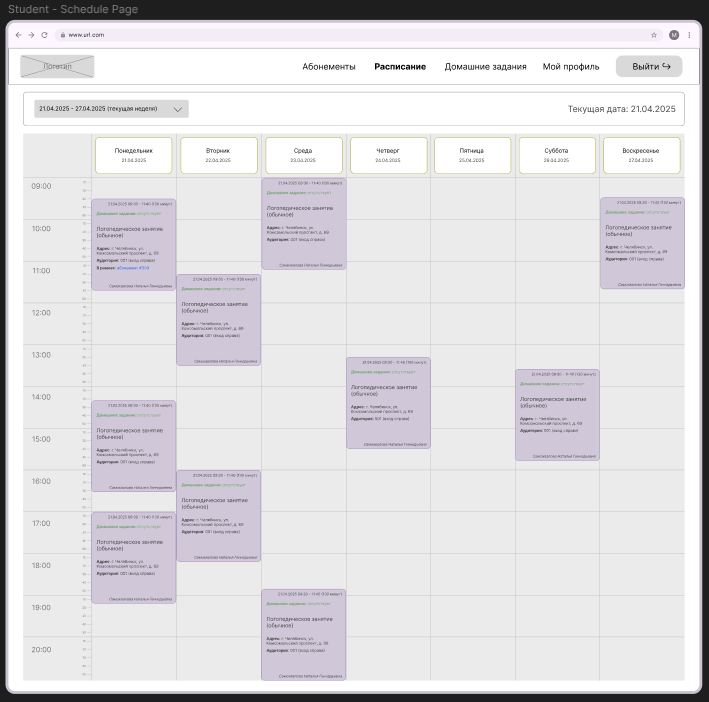
****

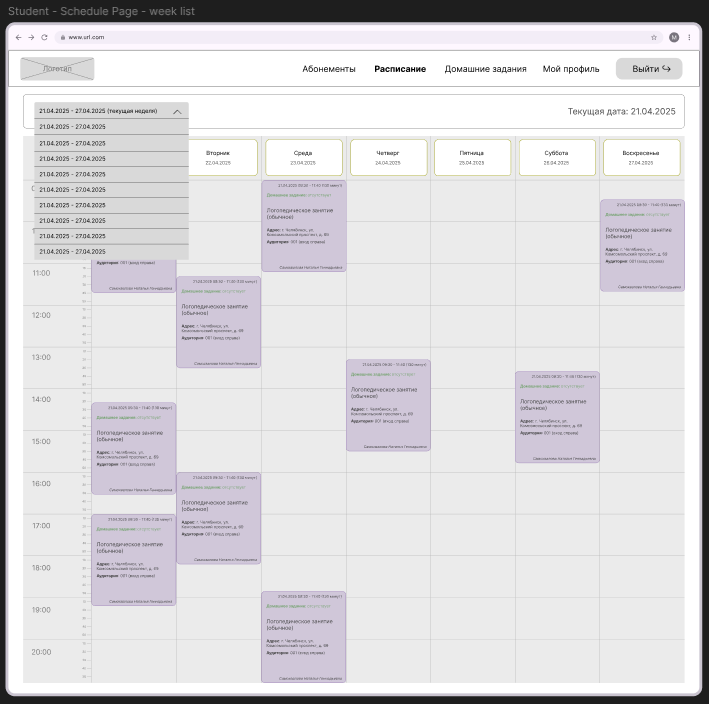
****

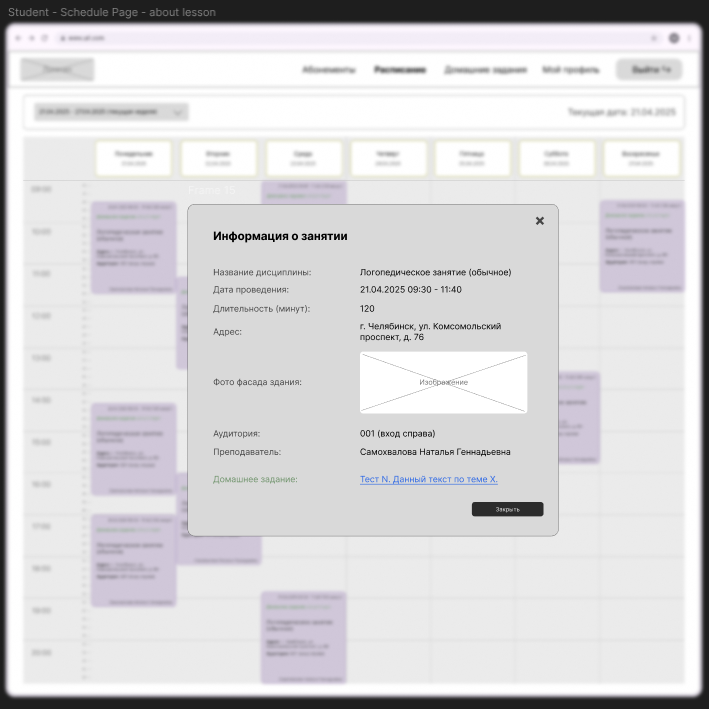


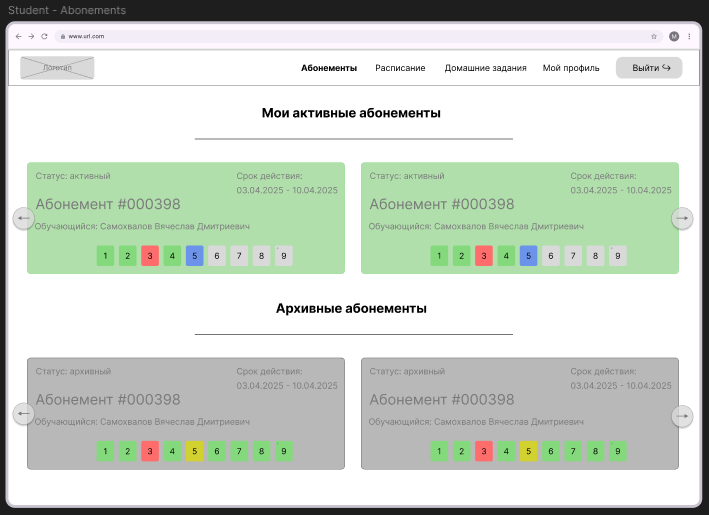


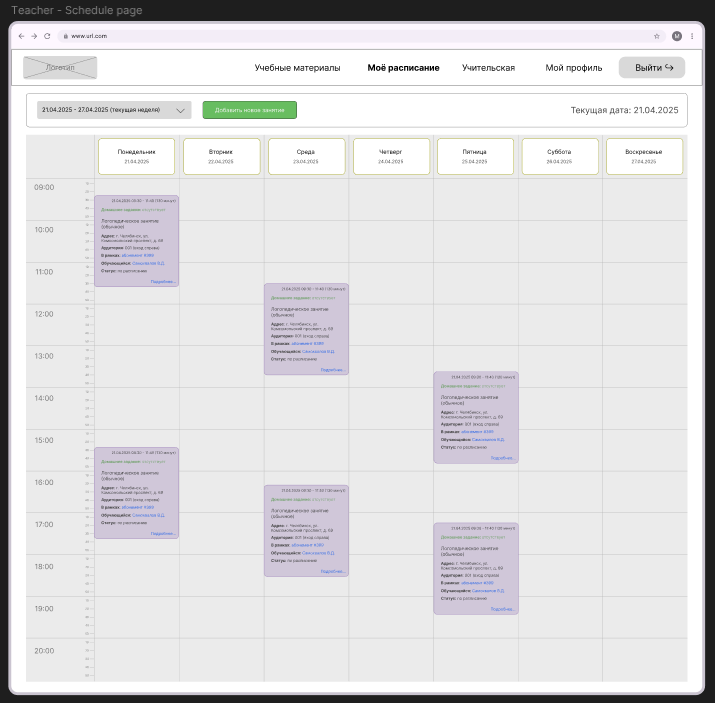


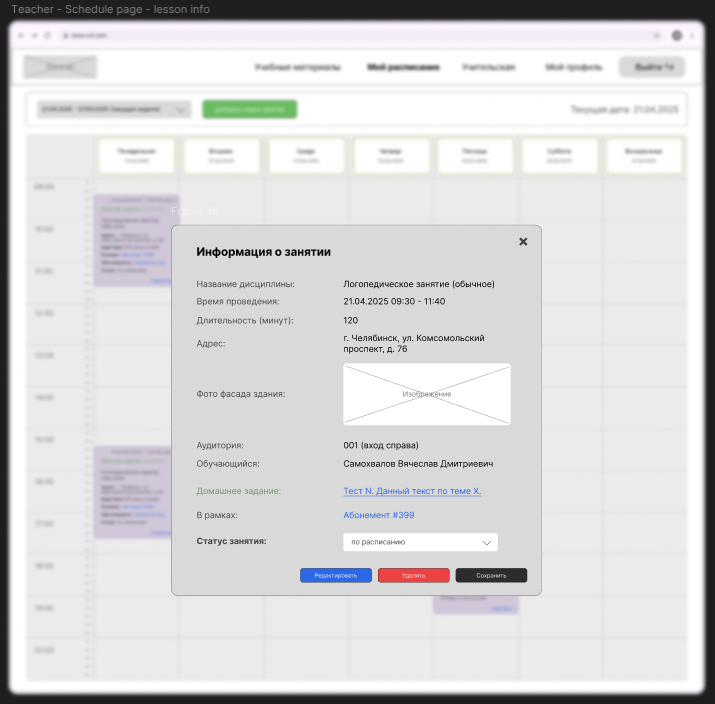


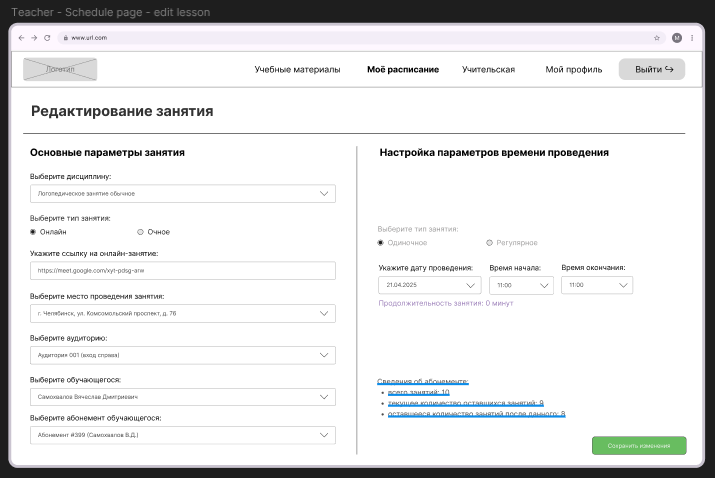


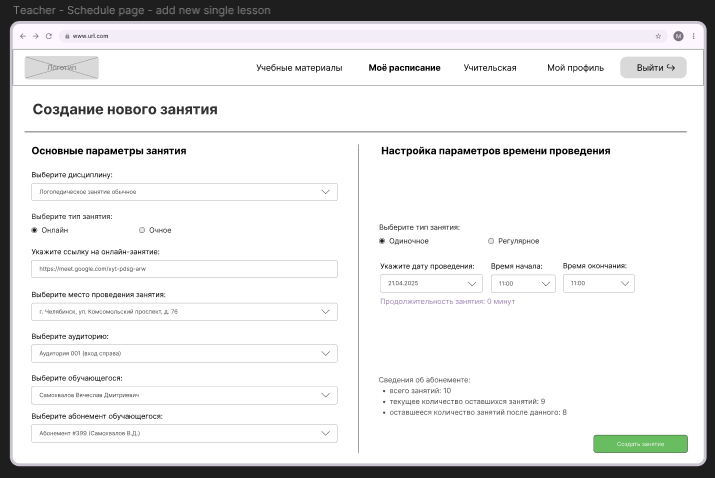


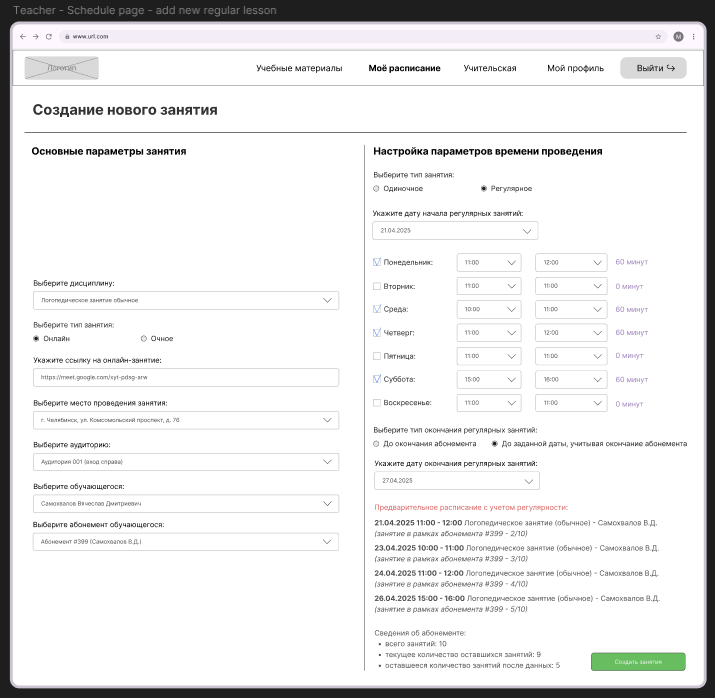


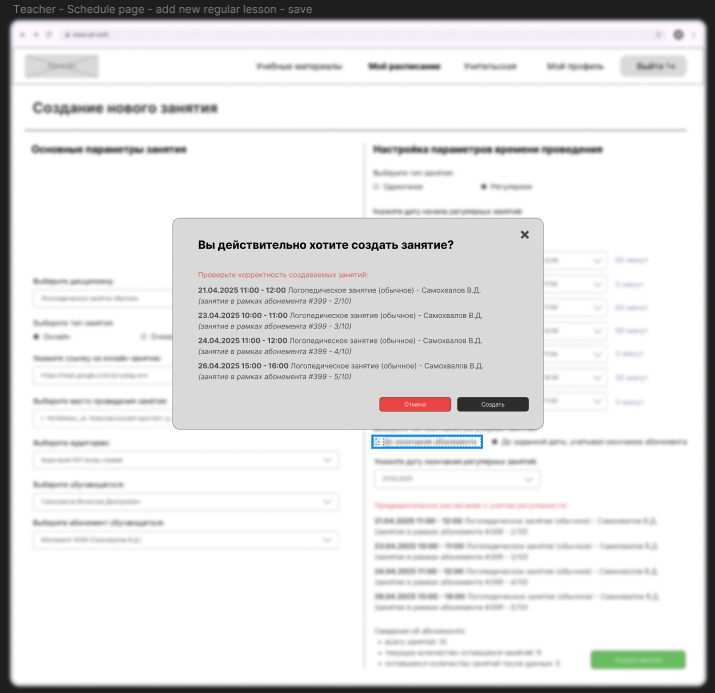


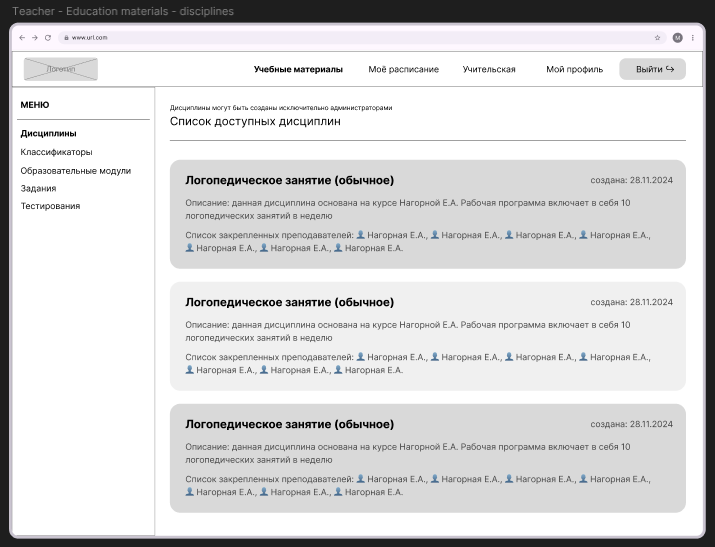


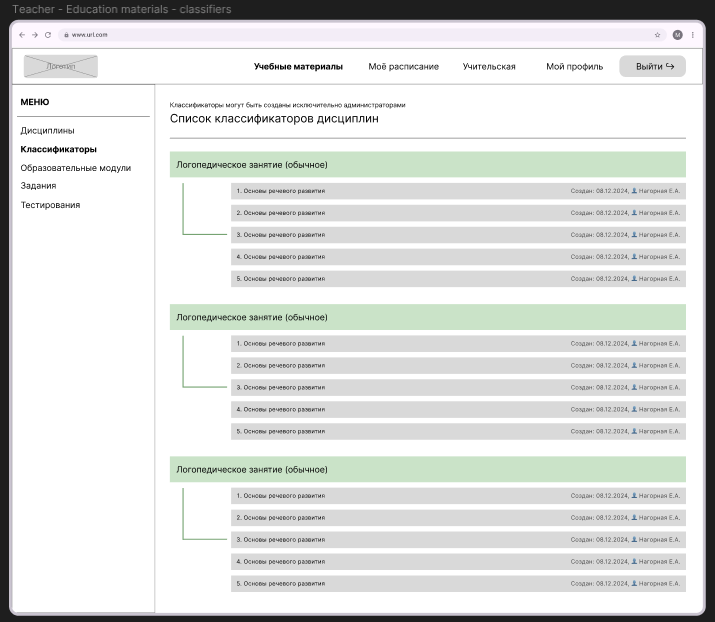


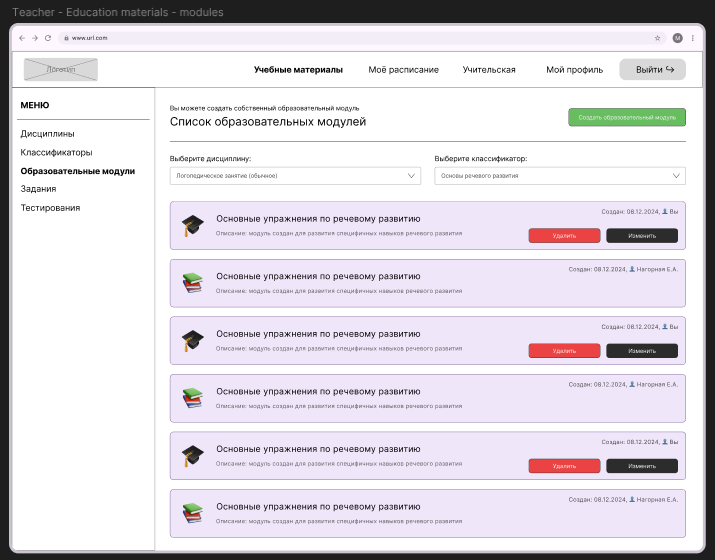


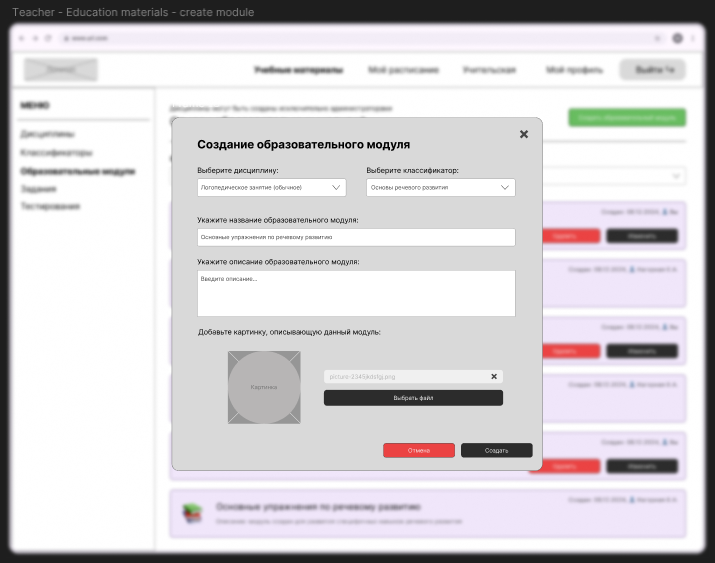


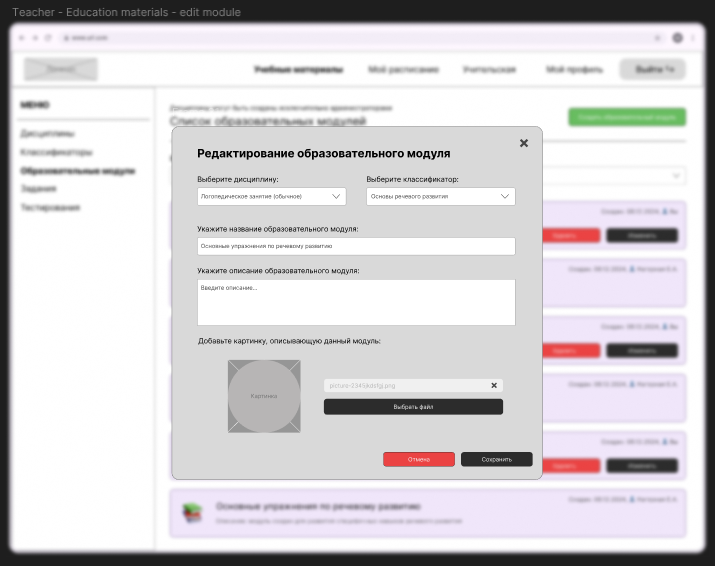


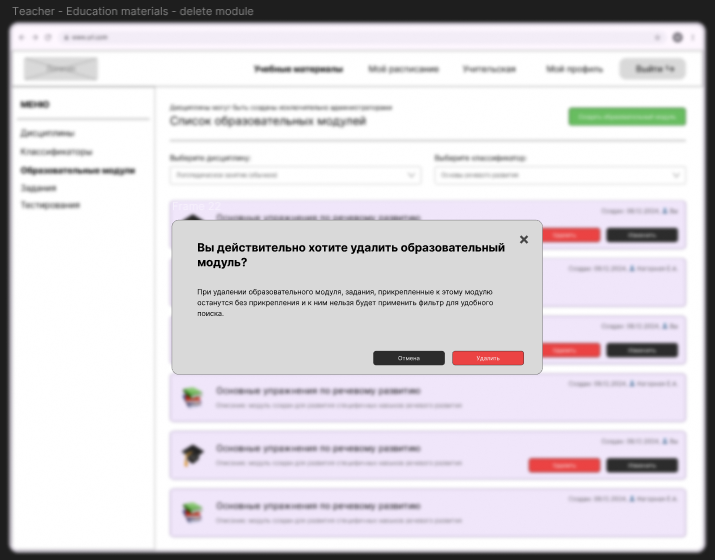


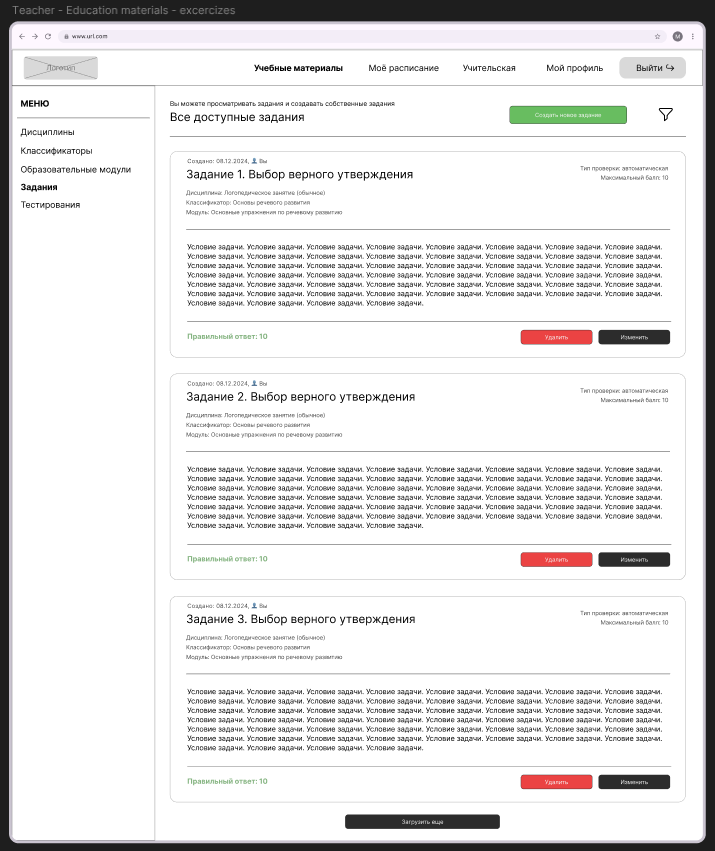


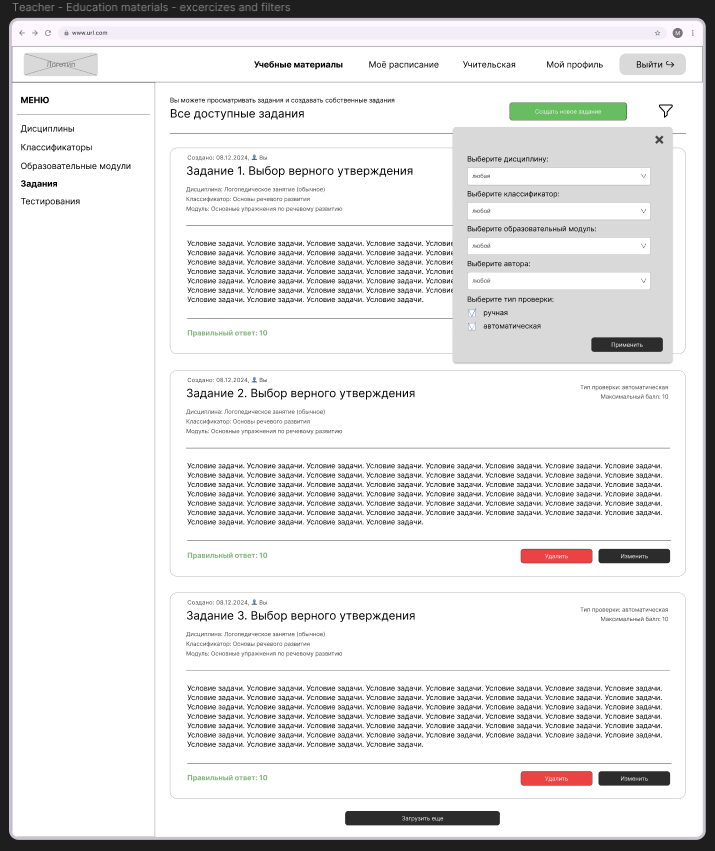




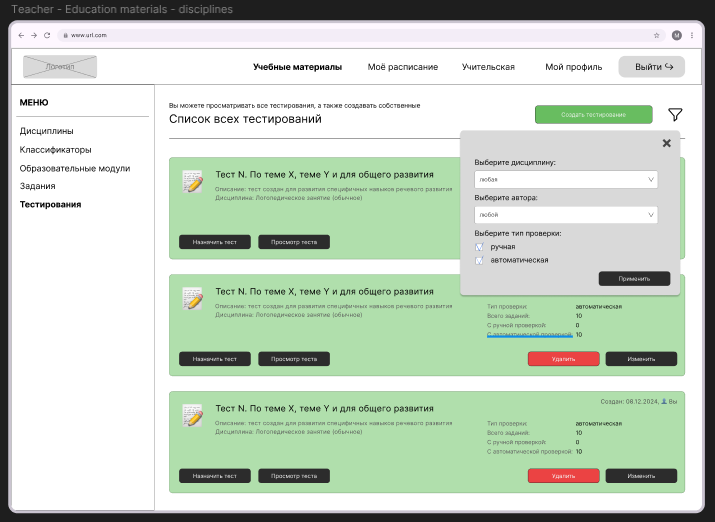


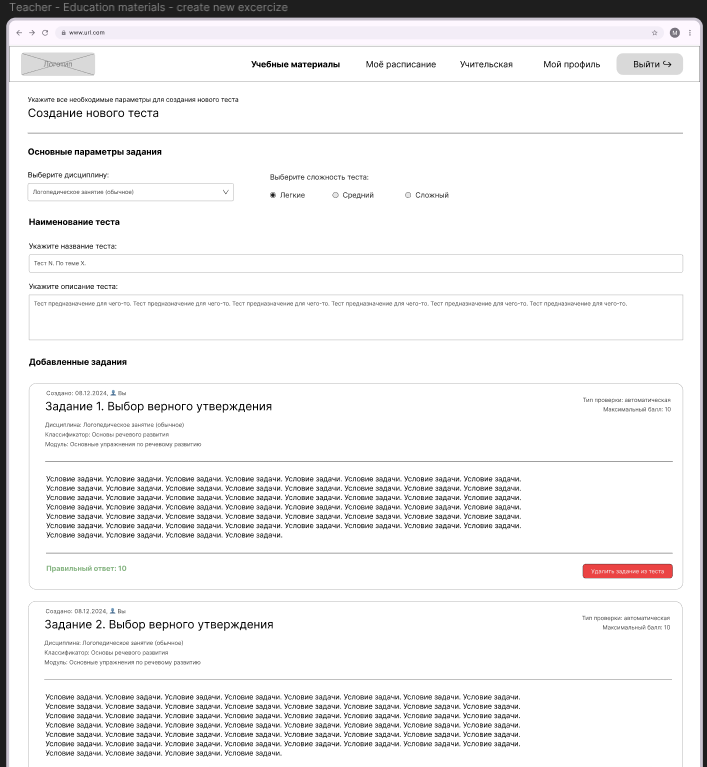


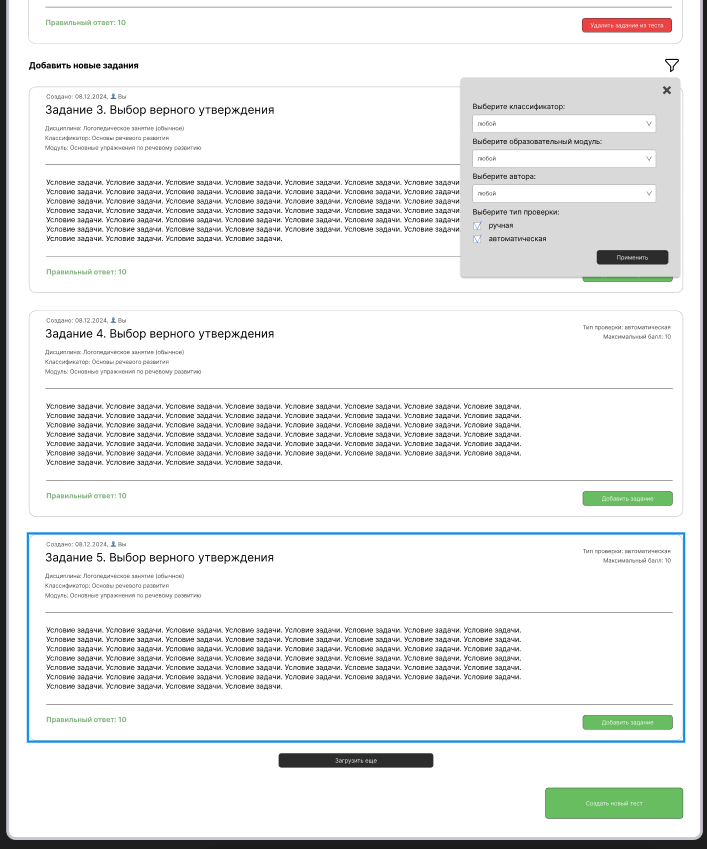


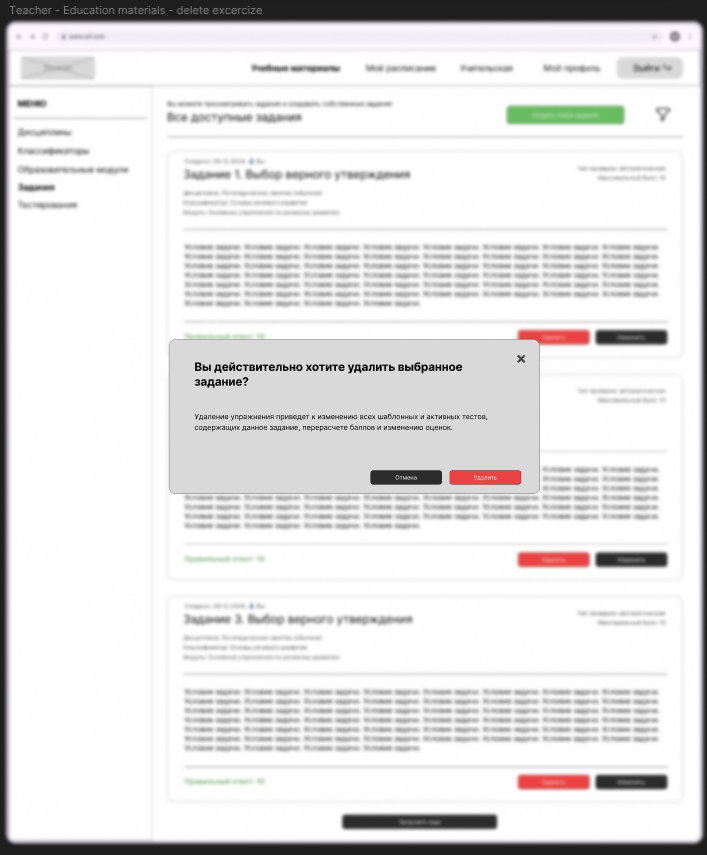


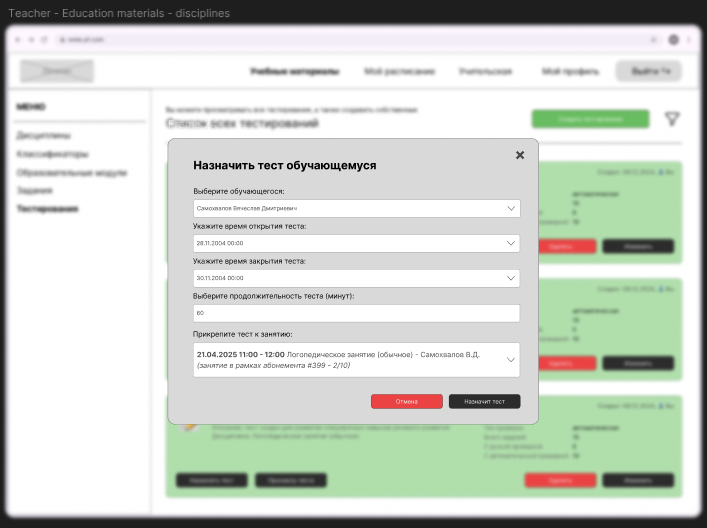


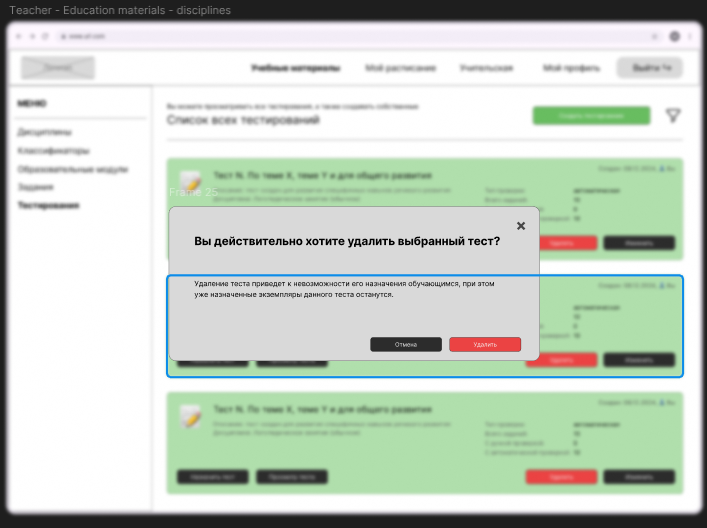


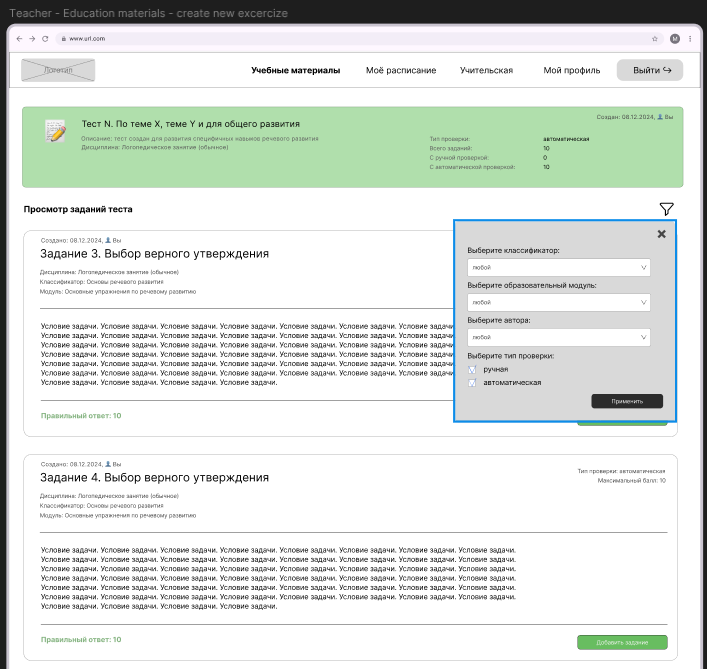


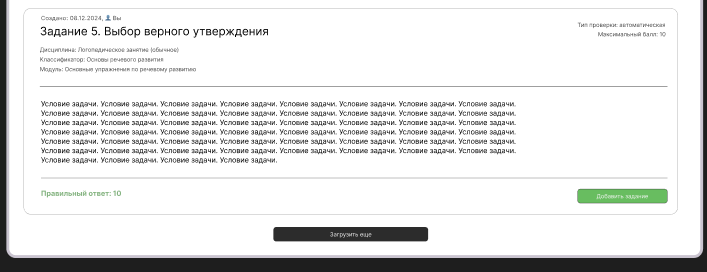


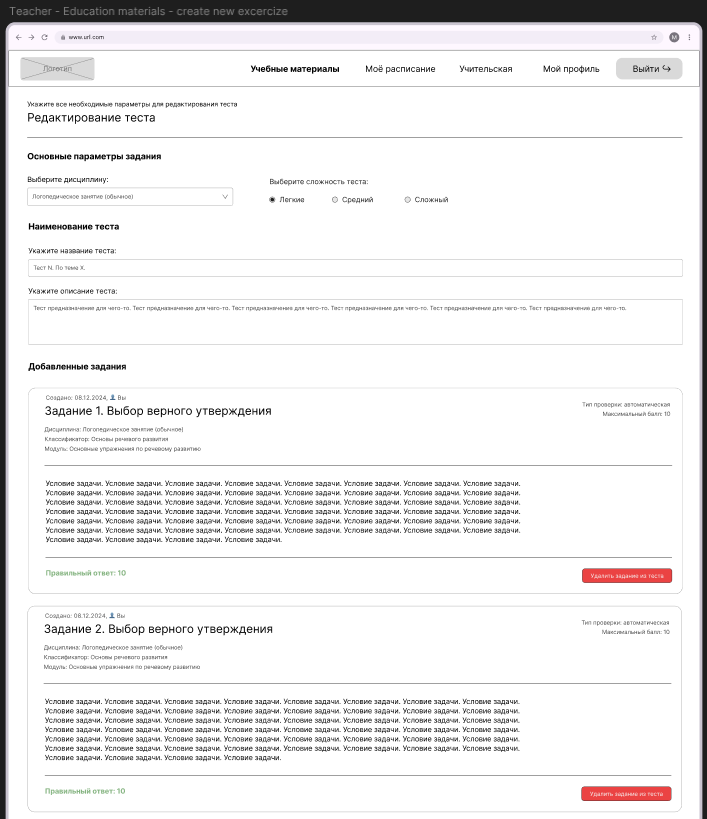


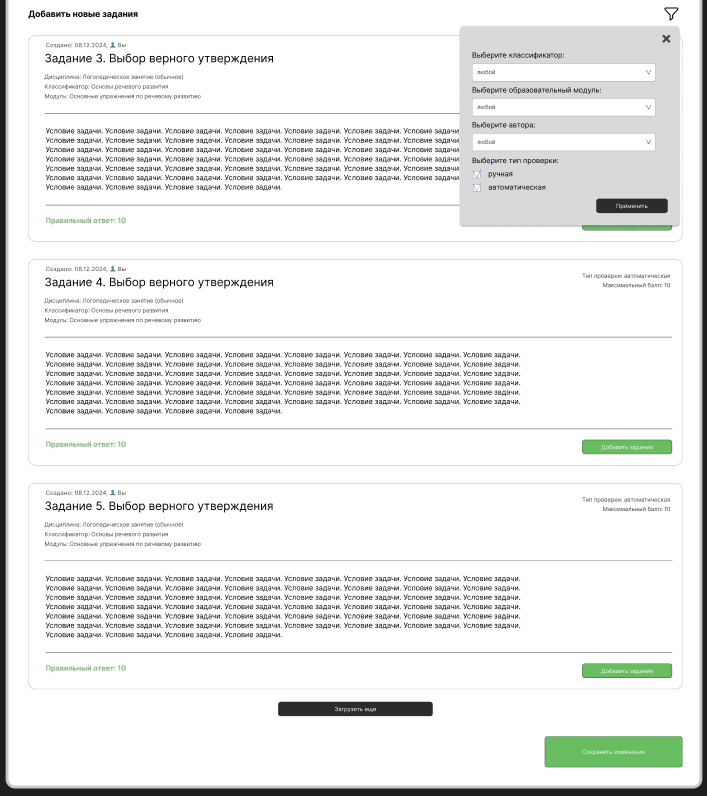
****

****

****

****

****

****

**Приложение 3 – SQL-скрипт для создания базы данных**

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;

USE `mydb` ;

-- -----------------------------------------------------

-- Table `mydb`.`Users`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Users` (

`user\_id` INT NOT NULL AUTO\_INCREMENT,

`full\_name` VARCHAR(100) NOT NULL,

`email` VARCHAR(100) NOT NULL,

`password\_hash` VARCHAR(255) NOT NULL,

`created\_at` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`birthday` DATE NOT NULL,

`gender` ENUM('Male', 'Female') NOT NULL,

`city` VARCHAR(50) NOT NULL,

`phone\_number` VARCHAR(20) NOT NULL,

`profile\_picture\_url` VARCHAR(255) NULL,

`unique\_code` VARCHAR(255) NOT NULL,

PRIMARY KEY (`user\_id`),

UNIQUE INDEX `email\_UNIQUE` (`email` ASC) VISIBLE,

UNIQUE INDEX `unique\_code\_UNIQUE` (`unique\_code` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Students`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Students` (

`user\_id` INT NOT NULL,

`class\_number` INT NULL,

`school\_name` VARCHAR(100) NULL,

PRIMARY KEY (`user\_id`),

CONSTRAINT `fk\_Students\_Users1`

FOREIGN KEY (`user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Teachers`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Teachers` (

`user\_id` INT NOT NULL,

`experience` INT NOT NULL,

`main\_work` VARCHAR(100) NOT NULL,

PRIMARY KEY (`user\_id`),

CONSTRAINT `fk\_Teachers\_Users1`

FOREIGN KEY (`user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Subscriptions`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Subscriptions` (

`subscription\_id` INT NOT NULL AUTO\_INCREMENT,

`total\_lessons` INT NOT NULL,

`start\_date` DATE NOT NULL,

`end\_date` DATE NOT NULL,

`created\_at` DATETIME NOT NULL,

`in\_archive` TINYINT NOT NULL,

`student\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`subscription\_id`),

INDEX `fk\_Subscriptions\_Students1\_idx` (`student\_id` ASC) VISIBLE,

INDEX `fk\_Subscriptions\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Subscriptions\_Students1`

FOREIGN KEY (`student\_id`)

REFERENCES `mydb`.`Students` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Subscriptions\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Administrators`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Administrators` (

`Users\_user\_id` INT NOT NULL,

`access\_level` ENUM('logs', 'full') NULL,

PRIMARY KEY (`Users\_user\_id`),

CONSTRAINT `fk\_Administrators\_Users1`

FOREIGN KEY (`Users\_user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Branches`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Branches` (

`branch\_id` INT NOT NULL AUTO\_INCREMENT,

`address` VARCHAR(100) NULL,

`working\_start` TIME NOT NULL,

`working\_end` TIME NOT NULL,

`description` TEXT NULL,

`photo\_url` VARCHAR(255) NULL,

`updated\_at` DATETIME NOT NULL,

`administrator\_id` INT NOT NULL,

PRIMARY KEY (`branch\_id`),

INDEX `fk\_Branches\_Administrators1\_idx` (`administrator\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Branches\_Administrators1`

FOREIGN KEY (`administrator\_id`)

REFERENCES `mydb`.`Administrators` (`Users\_user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Classrooms`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Classrooms` (

`classroom\_id` INT NOT NULL,

`name` VARCHAR(10) NOT NULL,

`description` TEXT NULL,

`branch\_id` INT NOT NULL,

`updated\_at` DATETIME NOT NULL,

`administrator\_id` INT NOT NULL,

PRIMARY KEY (`classroom\_id`),

INDEX `fk\_Classrooms\_Branches1\_idx` (`branch\_id` ASC) VISIBLE,

INDEX `fk\_Classrooms\_Administrators1\_idx` (`administrator\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Classrooms\_Branches1`

FOREIGN KEY (`branch\_id`)

REFERENCES `mydb`.`Branches` (`branch\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Classrooms\_Administrators1`

FOREIGN KEY (`administrator\_id`)

REFERENCES `mydb`.`Administrators` (`Users\_user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Disciplines`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Disciplines` (

`discipline\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(100) NOT NULL,

`description` TEXT NOT NULL,

`created\_at` INT NOT NULL,

`administrator\_id` INT NOT NULL,

PRIMARY KEY (`discipline\_id`),

INDEX `fk\_Disciplines\_Administrators1\_idx` (`administrator\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Disciplines\_Administrators1`

FOREIGN KEY (`administrator\_id`)

REFERENCES `mydb`.`Administrators` (`Users\_user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Lessons`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Lessons` (

`lesson\_id` INT NOT NULL,

`lesson\_date\_time` DATETIME NOT NULL,

`duration` INT NOT NULL,

`status` ENUM('scheduled', 'completed', 'cancelled\_in\_time', 'missed') NOT NULL,

`created\_at` DATETIME NOT NULL,

`online\_call\_url` VARCHAR(255) NULL,

`subscription\_id` INT NULL,

`classroom\_id` INT NULL,

`discipline\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

`student\_id` INT NOT NULL,

PRIMARY KEY (`lesson\_id`),

INDEX `fk\_Lessons\_Subscriptions1\_idx` (`subscription\_id` ASC) VISIBLE,

INDEX `fk\_Lessons\_Classrooms1\_idx` (`classroom\_id` ASC) VISIBLE,

INDEX `fk\_Lessons\_Disciplines1\_idx` (`discipline\_id` ASC) VISIBLE,

INDEX `fk\_Lessons\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

INDEX `fk\_Lessons\_Students1\_idx` (`student\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Lessons\_Subscriptions1`

FOREIGN KEY (`subscription\_id`)

REFERENCES `mydb`.`Subscriptions` (`subscription\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Lessons\_Classrooms1`

FOREIGN KEY (`classroom\_id`)

REFERENCES `mydb`.`Classrooms` (`classroom\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Lessons\_Disciplines1`

FOREIGN KEY (`discipline\_id`)

REFERENCES `mydb`.`Disciplines` (`discipline\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Lessons\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Lessons\_Students1`

FOREIGN KEY (`student\_id`)

REFERENCES `mydb`.`Students` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Education\_classifier`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Education\_classifier` (

`classifier\_id` INT NOT NULL,

`name` TEXT NOT NULL,

`created\_at` DATETIME NULL,

`administrator\_id` INT NOT NULL,

`discipline\_id` INT NOT NULL,

PRIMARY KEY (`classifier\_id`),

INDEX `fk\_Education\_classifier\_Administrators1\_idx` (`administrator\_id` ASC) VISIBLE,

INDEX `fk\_Education\_classifier\_Disciplines1\_idx` (`discipline\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Education\_classifier\_Administrators1`

FOREIGN KEY (`administrator\_id`)

REFERENCES `mydb`.`Administrators` (`Users\_user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Education\_classifier\_Disciplines1`

FOREIGN KEY (`discipline\_id`)

REFERENCES `mydb`.`Disciplines` (`discipline\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Education\_modules`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Education\_modules` (

`module\_id` INT NOT NULL AUTO\_INCREMENT,

`description` TEXT NOT NULL,

`preview\_url` VARCHAR(255) NULL,

`created\_at` DATETIME NULL,

`classifier\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`module\_id`),

INDEX `fk\_Education\_modules\_Education\_classifier1\_idx` (`classifier\_id` ASC) VISIBLE,

INDEX `fk\_Education\_modules\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Education\_modules\_Education\_classifier1`

FOREIGN KEY (`classifier\_id`)

REFERENCES `mydb`.`Education\_classifier` (`classifier\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Education\_modules\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Education\_exerсizes`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Education\_exerсizes` (

`exerсize\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(150) NOT NULL,

`description` TEXT NOT NULL,

`verification\_type` ENUM('auto', 'manual') NOT NULL,

`created\_at` DATETIME NOT NULL,

`right\_answer` TEXT NULL,

`max\_score` INT NOT NULL,

`module\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`exerсize\_id`),

INDEX `fk\_Education\_exerсizes\_Education\_modules1\_idx` (`module\_id` ASC) VISIBLE,

INDEX `fk\_Education\_exerсizes\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Education\_exerсizes\_Education\_modules1`

FOREIGN KEY (`module\_id`)

REFERENCES `mydb`.`Education\_modules` (`module\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Education\_exerсizes\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Tests`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Tests` (

`test\_id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(100) NOT NULL,

`description` TEXT NULL,

`complexity` ENUM('easy', 'medium', 'difficult') NOT NULL,

`discipline\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`test\_id`),

INDEX `fk\_Homeworks\_Disciplines1\_idx` (`discipline\_id` ASC) VISIBLE,

INDEX `fk\_Homeworks\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Homeworks\_Disciplines1`

FOREIGN KEY (`discipline\_id`)

REFERENCES `mydb`.`Disciplines` (`discipline\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homeworks\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Tests\_has\_Education\_exerсizes`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Tests\_has\_Education\_exerсizes` (

`test\_id` INT NOT NULL,

`exerсize\_id` INT NOT NULL,

`purpose\_at` DATETIME NOT NULL,

PRIMARY KEY (`test\_id`, `exerсize\_id`),

INDEX `fk\_Homeworks\_has\_Education\_exerсizes\_Education\_exerсizes1\_idx` (`exerсize\_id` ASC) VISIBLE,

INDEX `fk\_Homeworks\_has\_Education\_exerсizes\_Homeworks1\_idx` (`test\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Homeworks\_has\_Education\_exerсizes\_Homeworks1`

FOREIGN KEY (`test\_id`)

REFERENCES `mydb`.`Tests` (`test\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homeworks\_has\_Education\_exerсizes\_Education\_exerсizes1`

FOREIGN KEY (`exerсize\_id`)

REFERENCES `mydb`.`Education\_exerсizes` (`exerсize\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Active\_tests`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Active\_tests` (

`active\_id` INT NOT NULL AUTO\_INCREMENT,

`time\_start` DATETIME NULL,

`time\_end` DATETIME NULL,

`status` ENUM('active\_sent', 'active\_not\_sent') NOT NULL,

`assessment` INT NULL,

`test\_id` INT NOT NULL,

`lesson\_id` INT NULL,

`student\_id` INT NOT NULL,

`teacher\_id` INT NOT NULL,

PRIMARY KEY (`active\_id`),

INDEX `fk\_Homeworks\_active\_Homeworks1\_idx` (`test\_id` ASC) VISIBLE,

INDEX `fk\_Homeworks\_active\_Lessons1\_idx` (`lesson\_id` ASC) VISIBLE,

INDEX `fk\_Homeworks\_active\_Students1\_idx` (`student\_id` ASC) VISIBLE,

INDEX `fk\_Homeworks\_active\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Homeworks\_active\_Homeworks1`

FOREIGN KEY (`test\_id`)

REFERENCES `mydb`.`Tests` (`test\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homeworks\_active\_Lessons1`

FOREIGN KEY (`lesson\_id`)

REFERENCES `mydb`.`Lessons` (`lesson\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homeworks\_active\_Students1`

FOREIGN KEY (`student\_id`)

REFERENCES `mydb`.`Students` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homeworks\_active\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Tests\_packages`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Tests\_packages` (

`package\_id` INT NOT NULL AUTO\_INCREMENT,

`status\_verification` ENUM('verified', 'not\_verified') NOT NULL,

`exerсize\_start` DATETIME NULL,

`exerсize\_end` DATETIME NULL,

`student\_answer` TEXT NULL,

`file\_url` VARCHAR(255) NULL,

`accumulated\_score` INT NULL,

`active\_id` INT NOT NULL,

`exerсize\_id` INT NOT NULL,

PRIMARY KEY (`package\_id`),

INDEX `fk\_Homework\_submissions\_Homeworks\_active1\_idx` (`active\_id` ASC) VISIBLE,

INDEX `fk\_Homework\_submissions\_Education\_exerсizes1\_idx` (`exerсize\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Homework\_submissions\_Homeworks\_active1`

FOREIGN KEY (`active\_id`)

REFERENCES `mydb`.`Active\_tests` (`active\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Homework\_submissions\_Education\_exerсizes1`

FOREIGN KEY (`exerсize\_id`)

REFERENCES `mydb`.`Education\_exerсizes` (`exerсize\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Action\_logs`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Action\_logs` (

`log\_id` INT NOT NULL AUTO\_INCREMENT,

`action` TEXT NOT NULL,

`timestamp` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`device\_info` TEXT NOT NULL,

`user\_id` INT NOT NULL,

PRIMARY KEY (`log\_id`),

INDEX `fk\_Action\_logs\_Users1\_idx` (`user\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Action\_logs\_Users1`

FOREIGN KEY (`user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Auth\_logs`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Auth\_logs` (

`log\_id` INT NOT NULL AUTO\_INCREMENT,

`timestamp` DATETIME NULL,

`device\_info` TEXT NULL,

`user\_id` INT NOT NULL,

PRIMARY KEY (`log\_id`),

INDEX `fk\_Auth\_logs\_Users1\_idx` (`user\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Auth\_logs\_Users1`

FOREIGN KEY (`user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Parents`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Parents` (

`user\_id` INT NOT NULL,

`work\_name` VARCHAR(100) NULL,

`work\_phone` VARCHAR(20) NULL,

PRIMARY KEY (`user\_id`),

CONSTRAINT `fk\_Parents\_Users1`

FOREIGN KEY (`user\_id`)

REFERENCES `mydb`.`Users` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Teachers\_has\_Disciplines`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Teachers\_has\_Disciplines` (

`teacher\_id` INT NOT NULL,

`discipline\_id` INT NOT NULL,

PRIMARY KEY (`teacher\_id`, `discipline\_id`),

INDEX `fk\_Teachers\_has\_Disciplines\_Disciplines1\_idx` (`discipline\_id` ASC) VISIBLE,

INDEX `fk\_Teachers\_has\_Disciplines\_Teachers1\_idx` (`teacher\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Teachers\_has\_Disciplines\_Teachers1`

FOREIGN KEY (`teacher\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Teachers\_has\_Disciplines\_Disciplines1`

FOREIGN KEY (`discipline\_id`)

REFERENCES `mydb`.`Disciplines` (`discipline\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Students\_has\_Parents`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Students\_has\_Parents` (

`Students\_user\_id` INT NOT NULL,

`Parents\_user\_id` INT NOT NULL,

PRIMARY KEY (`Students\_user\_id`, `Parents\_user\_id`),

INDEX `fk\_Students\_has\_Parents\_Parents1\_idx` (`Parents\_user\_id` ASC) VISIBLE,

INDEX `fk\_Students\_has\_Parents\_Students1\_idx` (`Students\_user\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Students\_has\_Parents\_Students1`

FOREIGN KEY (`Students\_user\_id`)

REFERENCES `mydb`.`Students` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Students\_has\_Parents\_Parents1`

FOREIGN KEY (`Parents\_user\_id`)

REFERENCES `mydb`.`Parents` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Students\_has\_Teachers`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`Students\_has\_Teachers` (

`Students\_user\_id` INT NOT NULL,

`Teachers\_user\_id` INT NOT NULL,

PRIMARY KEY (`Students\_user\_id`, `Teachers\_user\_id`),

INDEX `fk\_Students\_has\_Teachers\_Teachers1\_idx` (`Teachers\_user\_id` ASC) VISIBLE,

INDEX `fk\_Students\_has\_Teachers\_Students1\_idx` (`Students\_user\_id` ASC) VISIBLE,

CONSTRAINT `fk\_Students\_has\_Teachers\_Students1`

FOREIGN KEY (`Students\_user\_id`)

REFERENCES `mydb`.`Students` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Students\_has\_Teachers\_Teachers1`

FOREIGN KEY (`Teachers\_user\_id`)

REFERENCES `mydb`.`Teachers` (`user\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;