МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет вычислительной техники

Кафедра МО и ПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Автоматизированная система планирования и анализа расписания занятий в ВУЗе

Студент _____ Луконин Максим Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Шибанов С.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«20» декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Луконин Максим Алексеевич гр. 21ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Автоматизированная система планирования и анализа расписания занятий в ВУЗе

Тема утверждена приказом ПГУ №_____от "____"_____2024 г.

3. Руководитель работы доцент кафедры «МО и ПЭВМ» Шибанов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

иложение ручного и автоматизированного составления
расписания занятий ВУЗа.
а разработки:
Языковые средства разработки: Python3, TypeScript, React, HTML, CSS, UML.
Инструментальные средства: Visual Studio Code 2022, Visual Paradigm,
Технологии разработки: ООП, шаблоны проектирования, Web API
клиент-серверная архитектура.
ирования: браузерная, кроссплатформенная
ции приложения:
а) добавление и просмотр первичных данных;
б) просмотр расписания;
в) редактирование расписания;
г) автоматическое составление расписания;
д) автоматический поиск коллизий в расписании.
озможность автоматического построения расписания
занятий как на основе частично созданного, так и на основе пустого расписания.
/БД и модели данных: реляционные, работа в БД через
ORM
ирование с использованием браузеров: Google Chrome, Opera,
Microsoft Edge

5. Перечень подлежащих разработке вопросов:

1) *Анализ предметной области и постановка задачи;*
2) *Проектирование;*
3) *Реализация;*
4) *Контроль качества программного обеспечения;*
5) *План разработки бюджета;*
6) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «22» ____декабря____2024 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 180 листов, 24 рисунка, 13 таблиц, 21 использованный источник и 2 приложения.

БАЗА ДАННЫХ, ИНФОРМАЦИОННАЯ СИСТЕМА, REST-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, СУБД, ПРЕДМЕТНАЯ ОБЛАСТЬ, РАСПИСАНИЕ ЗАНЯТИЙ ВУЗА.

Объект исследования – расписание занятий ВУЗа, существующие алгоритмы автоматического построения расписания занятий ВУЗа.

Цель выпускной квалификационной работы – разработать сервис с возможностью создания расписания занятий ВУЗа в ручном и автоматическом режимах.

Технологии разработки – разработка сервера приложений проводилась на языке программирования Python3 с использованием фреймворка FastAPI и ORM SQLModel, СУБД SQLite. Для серверной части сервиса, библиотека React Typescript для клиентской части. В качестве среды разработки использовался редактор программного кода Visual Studio Code.

Среда исполнения – Linux.

Результаты работы – в ходе выполнения выпускной квалификационной работы была создана REST-серверная часть сервиса для построения и анализа расписаний занятий в ВУЗе и клиентская часть на основе React.

Оглавление

Введение 7

1 Анализ предметной области и планирование разработки ПО 9

1.1 Процесс построения расписания занятий ВУЗа 9

1.1.1 Структура ВУЗа 9

1.1.2 Анализ внутренних документов отдела планирования 10

1.1.3 Порядок составления расписания занятий ВУЗа 10

1.2 Анализ бизнес-процессов автоматизированной системы составления расписания занятий ВУЗа 11

1.3 Обзор и анализ программного обеспечения аналогичного назначения 14

1.3.1 Анализ программы «1С: Автоматизированное составление расписания. Университет» 14

1.3.2 Анализ системы «Расписание учебных занятий» (РУЗ) СПбПУ 15

1.3.3 Анализ системы «АВТОРасписание» (AVTOR) 16

1.3.4 Сравнительный анализ систем составления расписания занятий ВУЗа 17

1.4 Планирование и анализ процесса разработки 19

2 Анализ требований и постановка задачи на разработку 23

2.1 Анализ функциональных требований 23

2.2 Анализ нефункциональных требований 26

2.3 Анализ требований к данным 28

2.4 Анализ требований к пользовательскому интерфейсу 30

2.5 Обоснование выбора технологий и средств разработки 32

3 Разработка программных средств 33

3.1 Разработка архитектуры 33

3.2 Разработка клиентского приложения 35

3.2.1 Разработка архитектуры пользовательского интерфейса 35

3.2.2 Разработка визуальных прототипов 36

3.2.3 Разработка программного обеспечения клиентского приложения с применением библиотеки React 40

3.3 Разработка серверной части приложения 41

3.3.1 Разработка архитектуры серверной части приложения 41

3.3.2 Разработка ORM-модели базы данных 42

3.3.3 Разработка методов запросов к базе данных и представлений для реализации аналитических бизнес-процессов при помощи ORM 49

3.3.4 Разработка алгоритма автоматического составления расписания .. 51

3.3.5 Разработка алгоритма автоматического поиска коллизий 55

3.3.6 Разработка Web-API приложения 56

4 Оценка качества программных средств 59

4.1 Мероприятия по оценке качества программных средств 59

4.2 Тестирование приложения 60

4.3 Оценка метрик кода 62

Заключение 64

Список использованных источников 65

Приложение А (справочное) Глоссарий 68

Приложение Б (обязательное) Листинг программы 70

Введение

Не смотря на активное развитие информационных технологий, систем машинного обучения и искусственного интеллекта, во многих ВУЗах, в том числе и в ПГУ, расписание занятий на семестр составляется вручную. Этот процесс занимает много времени и требует большого количества усилий, а в результате из-за человеческого фактора в расписании могут быть допущены ошибки [1].

Использование автоматизированных систем составления расписания занятий ВУЗа может сократить время на построение расписания, но качество полученного расписания не соответствует всем требованиям к нему – специалистам отдела планирования необходимо вручную корректировать и дорабатывать расписание. Помимо этого, в построенных с помощью специализированных программ системах теряется устойчивая структура расписания. К примеру один и тот же предмет годами проводится в одной и той же аудитории; занятия по физкультуре всегда проводятся в первой половине дня и т. д.

Ручное планирование позволяет учитывать многие нюансы, которые сложно формализовать и добавить в алгоритм. Отдельно стоит упомянуть пожелания кафедр, которые не всегда возможно добавить в программу составления расписания.

Актуальность онлайн-сервиса составления и анализа расписания занятий ВУЗа обусловлена тем, что сервисы-аналоги либо являются слишком дорогими, либо содержат несовершенные алгоритмы автоматического составления расписания занятий. Трудности также состоят в интеграции ПО для работы с расписание с электронными системами учебных заведений. Каждый ВУЗ имеет свои требования к расписанию и к исходным данным. Все эти требования должны учитываться в алгоритме составления расписания, что зачастую не представляется возможным.

Целью выпускной квалификационной работы является разработка клиент-серверного веб-приложения для автоматизированного составления расписания занятий ВУЗа.

Для достижения поставленной цели были поставлены следующие задачи:

1) провести анализ предметной области, аналогичного ПО;
2) определить требования к разрабатываемому ПО;
3) разработать клиентское и серверное приложения для составления расписания занятий ВУЗа;
4) провести тестирование и оценку качества разработанного ПО.

Проведен анализ предметной области, определены бизнес-процессы в информационной системе ВУЗа, проведен обзор аналогичного ПО и спланирована разработка собственного ПО. После этого был проведен анализ функциональных и нефункциональных требований, требований к данным и интерфейсу. Было разработано клиентское

В ходе выполнения ВКР была исследована предметная область составления расписания занятий ВУЗа, изучено аналогичное ПО, спроектировано приложение составления расписания занятий ВУЗа с функциями автоматического составления расписания и поиска коллизий.

В настоящий момент создан прототип веб-приложения, которое позволяет специалисту отдела планирования добавлять первичные данные, работать с расписанием и пользоваться автоматическими функциями поиска коллизий и составления расписания.

В дальнейшем ПО будет доработано: будет внедрен более совершенный алгоритм составления расписания, расширен функционал для работы с первичными данными, создана система подсказок для специалиста отдела планирования, которая будет работать при ручном составлении и изменении расписания.

1 Анализ предметной области и планирование разработки ПО

1.1 Процесс построения расписания занятий ВУЗа

1.1.1 Структура ВУЗа

ВУЗ представляет собой иерархическую структуру, состоящую из университетов, факультетов, кафедр, специальностей и групп. К примеру, в Пензенском Государственном Университете существует несколько институтов (Политехнический институт, Медицинский институт) и Военно-Учебный Центр (ВУЦ). Каждый из университетов состоит из одного или нескольких факультетов, например Политехнический Университет ПГУ включает в себя факультеты вычислительной техники, информационных технологий и электроники, промышленных технологий, электроэнергетики и транспорта.

Каждый факультет состоит из одной или нескольких кафедр. Так, Факультет вычислительной техники состоит из кафедр Вычислительной техники (ВТ), Математического обеспечения и применение ЭВМ (МОиПЭВМ), Информационно-вычислительных систем (ИВС) и других. У каждой кафедры есть полное и сокращенное название.

На кафедрах готовят студентов по специальностям (направлениям подготовки) на очной, очно-заочной и заочной формах обучения по направлениям бакалавриат (четырёхгодичная форма обучения), специалитет (пятилетняя) и магистратура (дополнительная двухлетняя форма обучения). Так, на кафедре МОиПЭВМ готовят студентов по направлениям подготовки 09.03.04 «Программная инженерия» (бакалавриат) и 09.03.02 «Информационные системы и технологии» (бакалавриат). Каждая специальность имеет свой номер (код), название, сокращенное название и принадлежность к бакалавриату, специалитету или магистратуре.

Студенты обучаются в группах по специальностям. Каждая группа имеет свой номер (например 20ВИ1, 21ВИ1, 21ВП1). Число студентов в

каждой группе ограничено правилами ВУЗа. Для удобства планирования в учебном плане группы неформально объединяются в потоки.

1.1.2 Анализ внутренних документов отдела планирования

Учебный план – список предметов, которые должны быть изучены студентами определённой специальности за семестр.

Нагрузка – количество лекционных и лабораторных часов на изучение каждого предмета за семестр.

На основе учебного плана, нагрузки, списка групп кафедры и профессорско-преподавательского состава кафедры перед началом семестра на каждой кафедре разрабатывается набор документов, определяющих список предметов для каждой группы с количеством часов (пар) за семестр, необходимым для проведения предмета, оборудованием и преподавателями, которые будут проводить данный предмет. Вместе с учебной нагрузкой указываются рекомендации кафедры: аудитории, в которых преподавателю будет удобнее проводить его предмет [2].

Данный документ передаётся в отдел планирования. Отдел планирования вручную составляет расписание для каждого факультета (а иногда и в отдельности для некоторых кафедр), после чего отдельные расписания сопоставляются, устраняются коллизии и накладки.

1.1.3 Порядок составления расписания занятий ВУЗа

Ручное составления расписания занятий ВУЗа в отделе планирования можно представить в виде следующего алгоритма:

1) Планирование работы ВУЦ (Военно-Учебного Центра), составление расписания ВУЦ.

2) Планирование общих для всех специальностей предметов (физкультура, русский язык, история и т. д.)

3) Планирование общих для каждого факультета предметов (МАИС, УТП и т. д.)

4) Планирование предметов кафедр, как правило уникальных для каждой специальности.

5) Ручная корректировка получившегося расписания.

6) Сопоставление расписаний факультетов, устранение коллизий и накладок.

Пункты 5 и 6 могут повторяться несколько раз, пока не будет достигнуто приемлемое качество получившегося расписания.

Составление расписания занятий с использованием специализированного ПО тоже включает в себя несколько этапов ручной корректировки:

1. Ручное составления «каркаса» расписания – выставление тех предметов, проведение которых в определённое время обязательно.
2. Использование алгоритма для автоматического заполнения оставшихся ячеек расписания.
3. Ручная корректировка получившегося расписания.

Пункт 3 может повторяться несколько раз после совещаний с обсуждением получившегося расписания.

Получившееся в результате выполнения всех пунктов расписание публикуют в качестве предварительного, получают обратную связь от кафедр и при необходимости вносят незначительные корректировки.

1.2 Анализ бизнес-процессов автоматизированной системы составления расписания занятий ВУЗа

Пользователем приложения будет являться специалист отдела планирования ВУЗа. Приложение должно обеспечить его следующими возможностями:

- добавление исходных данных о структуре ВУЗа;
- добавление учебного плана;
- ручное составление расписания;
- автоматическое составление расписания;

– автоматический поиск коллизий.

Таким образом, основными бизнес-процессами являются ручное составление расписания, автоматическое составление расписания и автоматический поиск коллизий. Диаграмма деятельности бизнес-процесса ручного составления расписания представлена на рисунке 1.

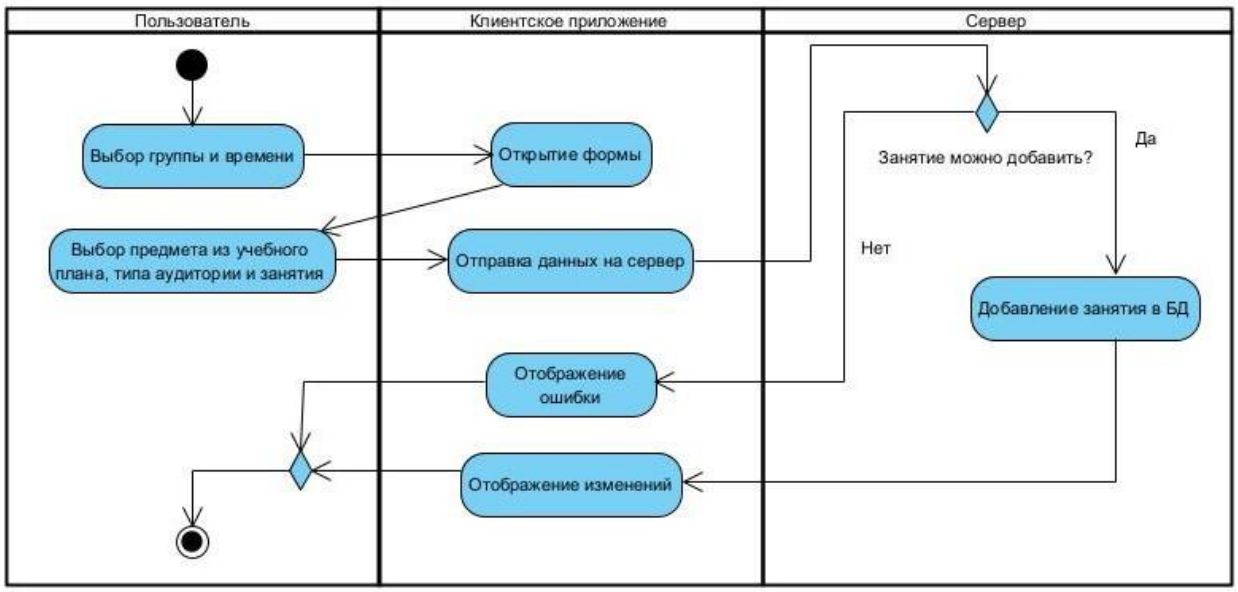

Рисунок 1 – Диаграмма деятельности бизнес-процесса ручного добавления элемента расписания

Данный бизнес-процесс позволяет вручную создать либо изменить элемент расписания. Система проверяет, возможно ли добавить элемент в базу данных, если да – добавляет его и отображает изменения на клиенте, если нет – отображает ошибку.

Диаграмма деятельности бизнес-процесса автоматического составления расписания занятий представлена на рисунке 2.

Как видно из диаграммы, при использовании алгоритма автоматического составления расписания от пользователя требуется только инициировать сам процесс – все остальное делает система. Аналогично работает бизнес-процесс поиска коллизий – пользователь только инициирует процесс, все действия выполняет система.

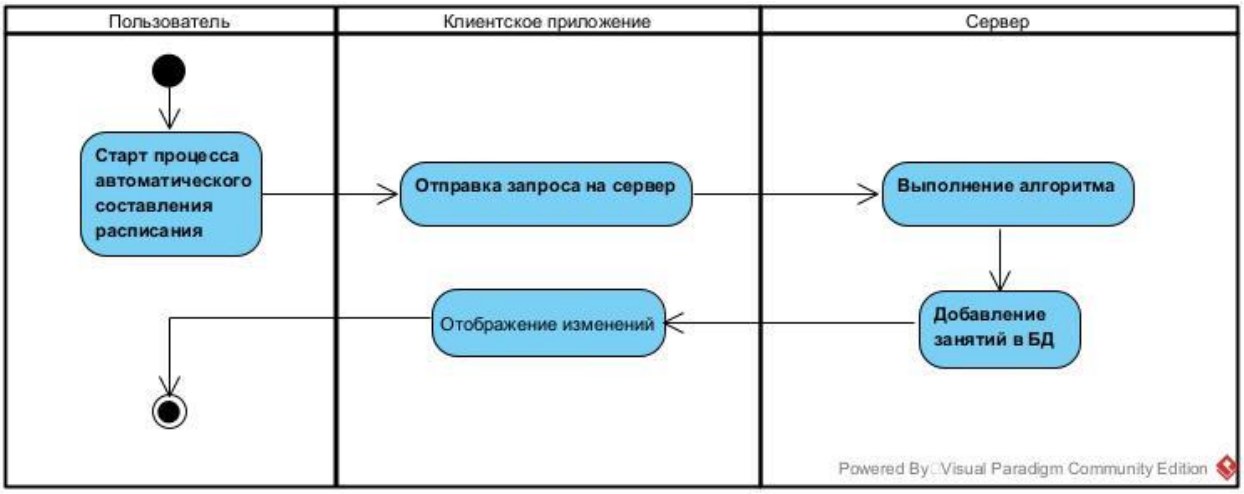

Рисунок 2 – Диаграмма деятельности бизнес-процесса автоматического составления расписания

Диаграмма деятельности бизнес-процесса автоматического поиска коллизий в готовом расписании представлена на рисунке 3.

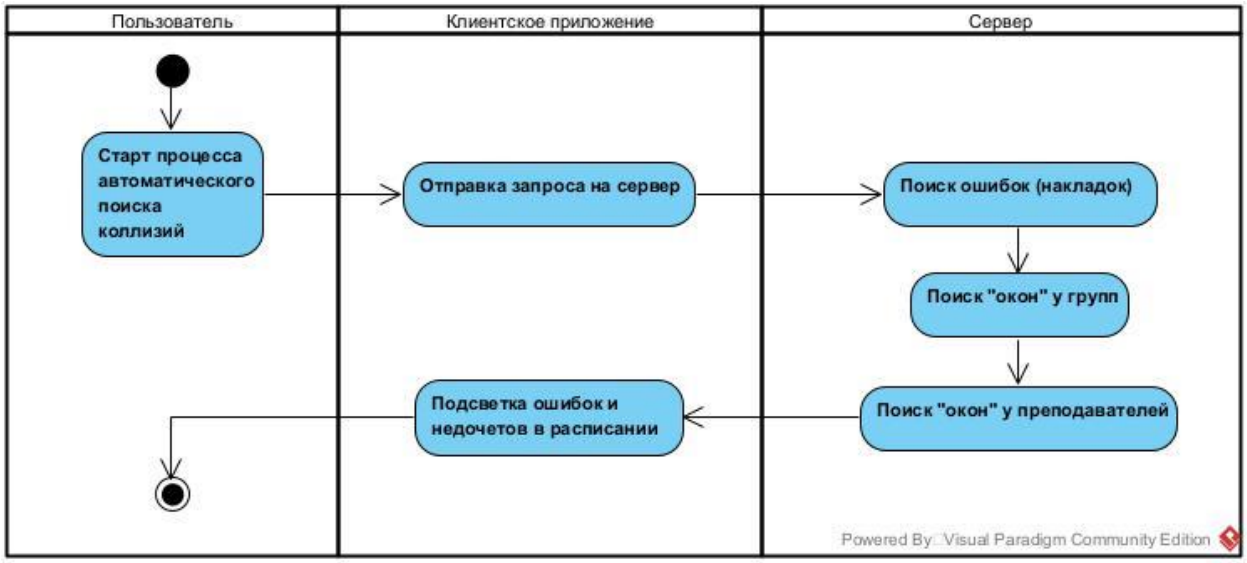

Рисунок 3 – Диаграмма деятельности бизнес-процесса автоматического поиска коллизий

В данном бизнес-процессе в расписание не вносится никаких изменений, но найденные ошибки подсвечиваются в расписании разными цветами. «Накладки» (например случаи, когда один преподаватель должен одновременно проводить два разных предмета) подсвечиваются красным цветом, «окна» у групп – оранжевым, «окна» у преподавателей – желтым.

Наличие «окон» у групп не является серьезной ошибкой, но влияет на качество расписания. «Окна» у преподавателей являются недочетом, но могут быть и заранее запланированы (например для проведения консультаций).

1.3 Обзор и анализ программного обеспечения аналогичного назначения

1.3.1 Анализ программы «1С: Автоматизированное составление расписания. Университет»

Частью экосистемы от компании 1С для ВУЗов является их программа составления расписания, которая может поставляться как отдельно (и в дальнейшем встраиваться в электронную систему ВУЗа), так и в комплекте с другими решениями компании. Система позволяет сотрудникам отдела планирования работать как вручную, так и использовать алгоритмы автоматического построения расписания [4]. На рисунке 4 приведён интерфейс программы 1С: Расписание.

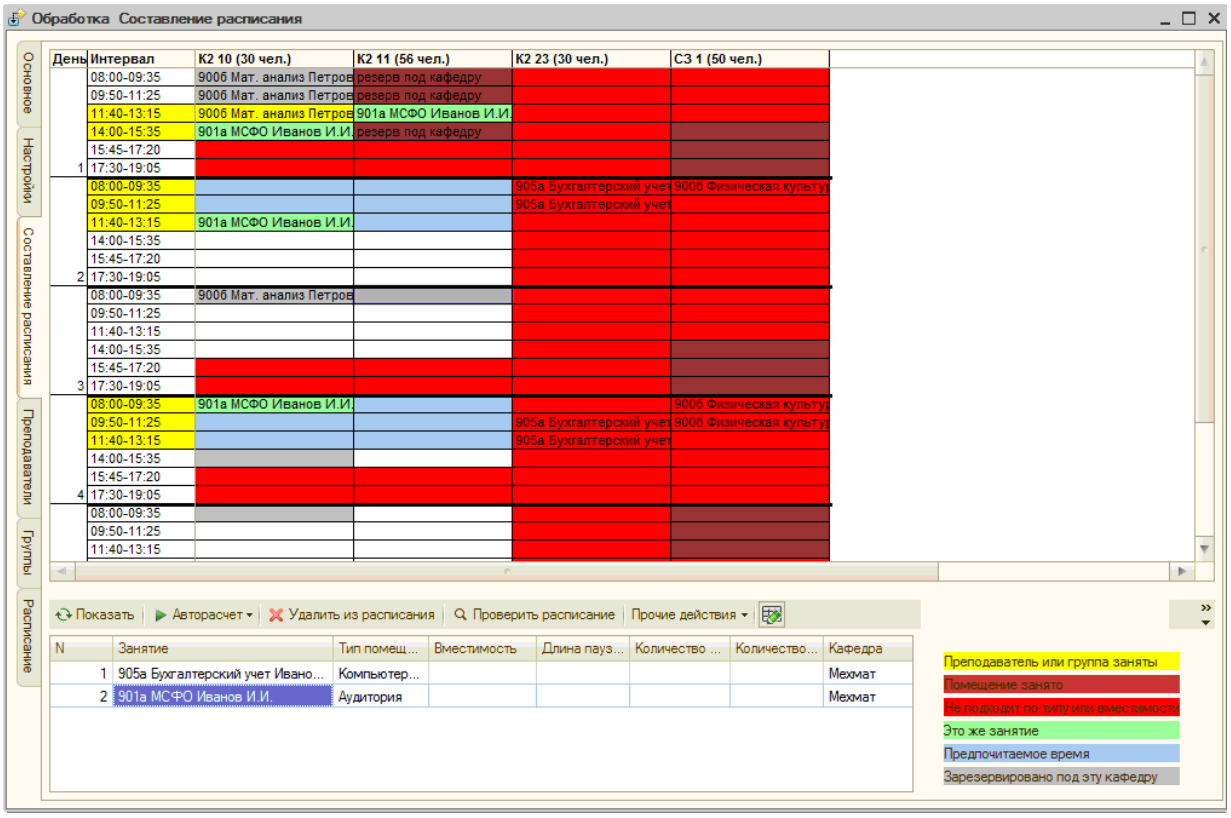

Рисунок 4 – Главное окно работы с расписанием в программе «1С: Автоматизированное составление расписания»

Достоинствами ПО от компании 1С являются: кроссплатформенность (ПО работает на основе платформы «1С: Предприятие»), возможность задать свои ограничения для расписания, поддержка всех форм обучения (очной, очно-заочной, заочной). Программа позволяет загружать исходные данные в различных форматах (XML, Excel). Помимо автоматического построения расписания, в программе реализован автоматический поиск коллизий.

Главным недостатком является высокая стоимость ПО: 119 700 рублей на момент написания работы без учета платы за интеграцию системы. Однако для заинтересованных клиентов доступен небольшой пробный период.

1.3.2 Анализ системы «Расписание учебных занятий» (РУЗ) СПбПУ

Автоматизированная система «Расписание учебных занятий» (РУЗ) СПбПУ регламентирует процедуру составления расписания учебных занятий по основным образовательным программам высшего образования в СПбПУ [4]. Интерфейс программы представлен на рисунке 5.

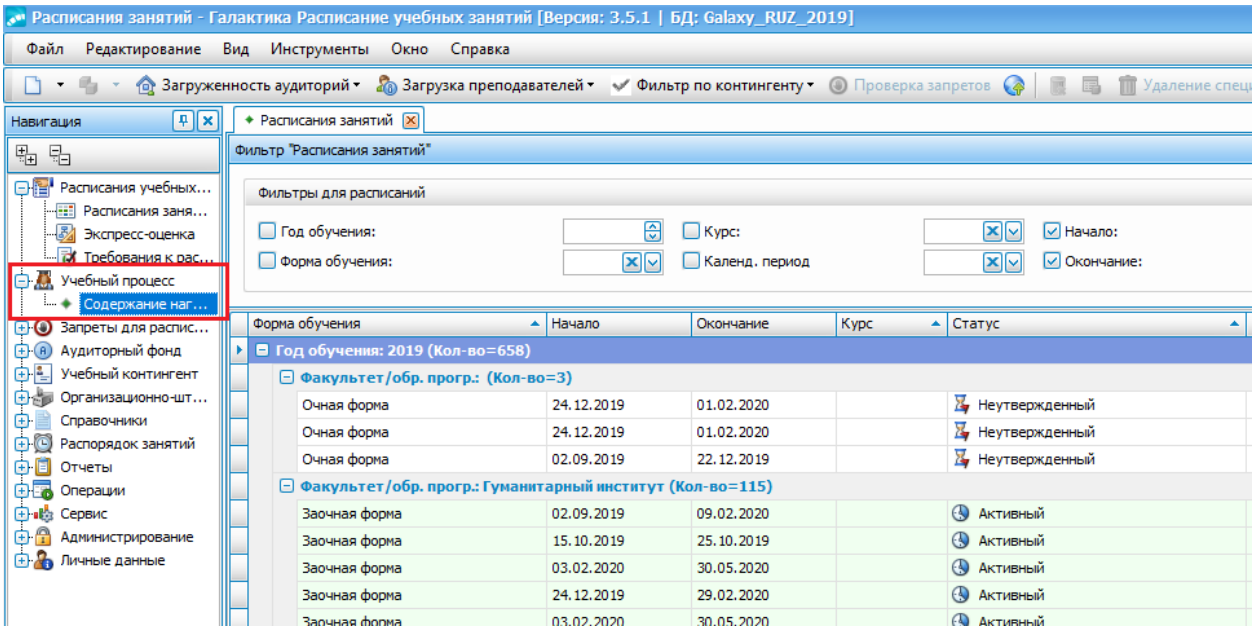

Рисунок 5 – Интерфейс РУЗ

Программное решение создана для составления расписания занятий в СПбПУ и отвечает всем требованиям, существующим в этом ВУЗе. Среди них общие требования Министерства Обращения (шестидневная учебная неделя),

базовые требования (соответствие числа мест в аудитории числу студентов в группе, отсутствие наладок), особые требования (не больше 2-х занятий в день по дисциплинам, изучающим иностранные языки).

Система РУЗ является частью электронной среды СПбПУ, готовое расписание студенты могут увидеть на официальном сайте ВУЗа.

К недостаткам стоит отнести сложный интерфейс со множеством деталей, что осложняет работу специалистов отдела планирования.

1.3.3 Анализ системы «АВТОРасписание» (AVTOR)

АВТОРасписание – одна из первых программ для составления расписаний занятий ВУЗа. Прототип ПО был создан еще в 1993 году для школ и лицеев, в 1996 в неё был добавлен алгоритм автоматического построения и оптимизации расписания, была создана версия для колледжей и ВУЗа [5]. На рисунке 6 приведен интерфейс программы «АВТОРасписание».

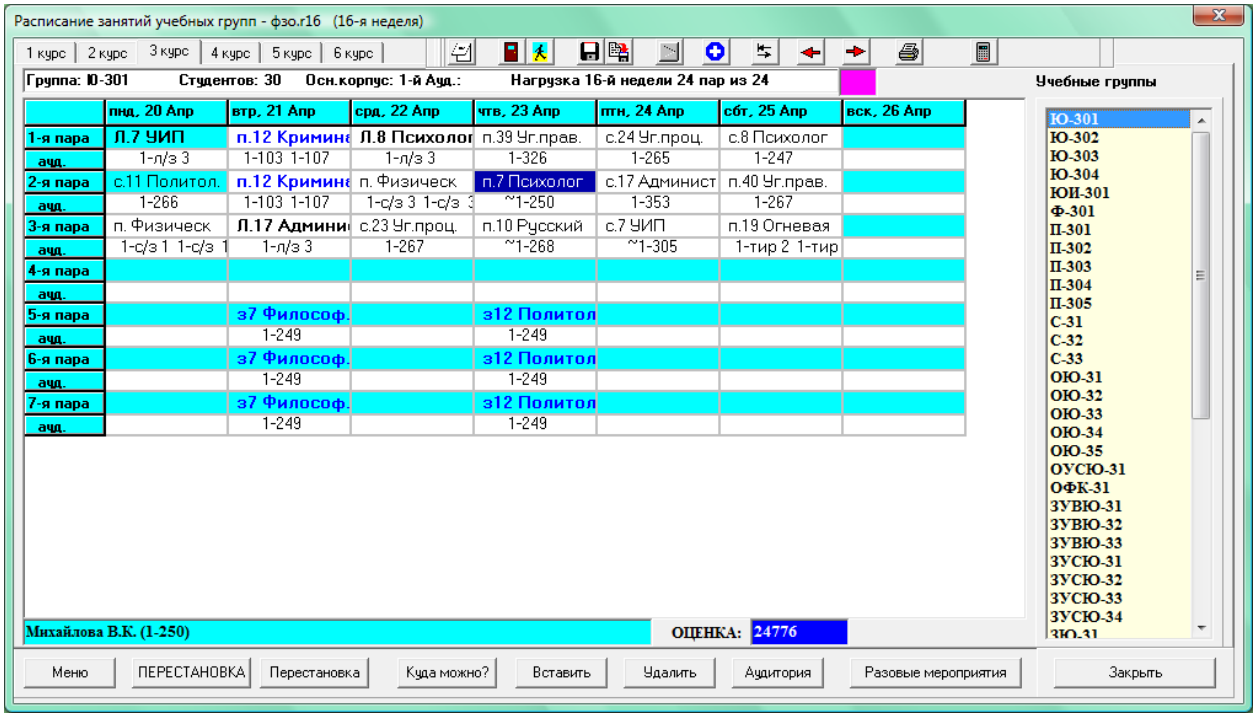

Рисунок 6 – Интерфейс «АВТОРасписание»

«АВТОРасписание» можно индивидуально настроить под требования учебного заведения. Сейчас программу используют более трехсот учебных заведений в нескольких странах. Доступна бесплатная пробная версия.

По утверждению авторов программы «AVTOR», скорость работы алгоритма в 10 – 20 раз превышает скорость работы аналогичного ПО, в результате работы оставляет в 2 – 3 раза меньше «окон». Программа имеет систему подсказок, оценки качества расписания и работы со статистикой.

Программа «AVTOR» платная, в сравнении с аналогами имеет менее удобный интерфейс.

1.3.4 Сравнительный анализ систем составления расписания занятий ВУЗа

Было выполнено качественное сравнение рассмотренных ранее аналогичных программных средств.

«1С: Автоматизированное составление расписания. Университет» - популярное решение, созданное на платформе «1С: Предприятие», благодаря чему работает на многих устройствах и может быть интегрировано с другим ПО, созданным на базе 1С. Позволяет составлять расписание как в автоматическом, так и в ручном режиме, имеет множество настраиваемых ограничений, работает с разными форматами данных. ПО позволяет строить расписание в разрезе групп, аудиторий и преподавателей, что позволяет визуально наблюдать за равномерностью нагрузки. К минусам ПО можно отнести крайне высокую цену – в базовой комплектации от 119 700 до 143 000 рублей, без учёта возможный затрат на интеграцию с существующей в учебном заведении системой. Цена на версии для разного количества рабочих мест различается кратно: клиентская лицензия на 1 рабочее место стоит от 8400 рублей до 15 300 рублей, на 100 рабочих мест – от 658 400 рублей до 1 197 000 рублей. Для ознакомления с возможностями системы клиентам по запросу предоставляют бесплатную демонстрационную онлайн-версию программы.

«Расписание учебных занятий» является собственной системой построения расписания занятий СПбПУ и примером того, что иногда дешевле и надёжнее разработать собственную систему под все свои требования и

условности, чем покупать готовое решение. ПО предоставляет возможности для создания расписания занятий, промежуточных аттестаций, консультаций и экзаменационных сессий. Система построения расписаний плотно интегрирована в систему ВУЗа, поэтому для изменения утверждённого расписания необходимо согласование руководителей ВУЗов или кафедр, повторное утверждение расписания. Готовое и утверждённое расписание сразу отображается на официальном сайте учебного заведения. Среди минусов решения – перегруженный интерфейс, который можно было бы упростить, добавить возможно скрыть лишнюю информацию.

АВТОРасписание – одно из старейших решений в области составления расписания занятий. Программа позволяет не только строить расписание с нуля, но и оптимизировать уже существующее расписание, также имеет систему подсказок при ручном редактировании расписания. Данное ПО имеет множество возможностей, которые были добавлены в него за годы эксплуатации и доработок, что позволяет ему подстраиваться под индивидуальные особенности учебных заведений. Программа составляет допустимое расписание по исходным данным, после чего специалист отдела планирования проводит его ручную доработку. Если в исходных данных есть ошибки, из-за которых нельзя построить расписание, то программа их находит и не строит расписание. Готовый вариант расписания можно экспортировать в Excel, Word и как HTML-документ. К минусам можно отнести несколько устаревший интерфейс программы.

Из вышеописанных программ наиболее удачным решением системы автоматического построения расписания является АВТОРасписание. ПО имеет алгоритм как построения, так и улучшения расписания, учитывает широкий спектр требований, которые можно задать самостоятельно. Однако и оно имеет свои недостатки, такие как устаревший интерфейс и относительно немаленькая стоимость.

1.4 Планирование и анализ процесса разработки

Первым этапом разработки ПО является планирование разработки, определение этапов разработки, задач каждого из этапов, сроков и средств для их выполнения. На этом этапе определяются бюджет и время выполнения проекта, а также основные ресурсы (в том числе трудовые) для создания проекта. Этот этап является одним из важнейших при разработке ПО, так как позволяет заранее определить все сроки и траты при реализации проекта, оценить целесообразность его выполнения.

В таблице 1 представлено описание основных задач при разработке ПО, описан их тип (веха, фаза либо задача), длительность и последовательность выполнения. В качестве основы для трудовой деятельности выбрана пятидневная рабочая неделя.

Таблица 1 – Задачи проекта

№	Задача	Тип	Длительность (дни)	Предшественники
1	Начало проекта	Веха	0	-
2	Анализ предметной области и постановка задачи на разработку	Фаза	11	-
3	Изучение проектов-аналогов	Задача	3	1
4	Анализ структуры ВУЗа	Задача	2	3
5	Анализ процесса составления расписания занятий ВУЗа	Задача	4	4
6	Планирование процесса разработки	Задача	2	5
7	Анализ предметной области и постановка задачи завершены	Веха	0	6
8	Разработка требований	Фаза	8	7
9	Разработка функциональных требований	Задача	2	8
10	Разработка нефункциональных требований	Задача	2	9
11	Разработка требований к данным	Задача	2	10
12	Разработка требований к пользовательскому интерфейсу	Задача	2	11
13	Разработка требований завершена	Веха	0	12
14	Разработка программных средств	Фаза	45	13
15	Разработка серверных ПС	Задача	25	14
16	Разработка клиентских ПС	Задача	20	15
17	Разработка программных средств завершена	Веха	0	16
18	Контроль качества серверных программных средств	Фаза	10	17

Продолжение таблицы 1

19	Планирование процесса контроля качества	Задача	2	18
20	Тестирование серверных ПС	Задача	4	19
21	Тестирование клиентских ПС	Задача	4	20
22	Оценка качества программного кода	Задача	2	21
23	Контроль качества завершен	Веха	0	22
24	Завершение работы над проектом	Фаза	5	23
25	Оформление документации	Задача	4	24
26	Сдача проекта	Задача	1	25
27	Проект завершен	Веха	0	26
28	Конец проекта	Веха	0	27

После определения основных задач была проведена оценка ресурсов. Ресурсы включают в себя все необходимое для завершения процесса разработки: трудовые и финансовые затраты, оборудование и материалы. Перечень ресурсов представлен в таблице 2.

Таблица 2 – Ресурсы проекта

Название ресурса	Тип ресурса	Затраты		Начисление
		Стандартная ставка	Ставка сверхурочных	
Программист	Трудовой	300,00 р/ч	400,00 р/ч	Пропорциональное
Аналитик	Трудовой	300,00 р/ч	400,00 р/ч	Пропорциональное
Тестировщик	Трудовой	200,00 р/ч	250,00 р/ч	Пропорциональное
Интернет	Затраты	-	-	Пропорциональное

Информация об распределении ресурсов по каждой задаче представлена в таблице 3.

Таблица 3 – Распределение ресурсов по задачам

№	Задача	Ресурсы	Единицы (затраты)
1	Начало проекта	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Изучение проектов-аналогов	Аналитик Интернет	100% 400р
4	Анализ структуры ВУЗа	Аналитик	100%
5	Анализ процесса составления расписания занятий ВУЗа	Аналитик	100%
6	Планирование процесса разработки	Аналитик	100%
7	Анализ предметной области и постановка задачи завершены	-	-

Продолжение таблицы 3

8	Разработка требований	-	-
9	Разработка функциональных требований	Программист	100%
10	Разработка нефункциональных требований	Программист	100%
11	Разработка требований к данным	Программист	100%
12	Разработка требований к пользовательскому интерфейсу	Программист	100%
13	Разработка требований завершена	-	-
14	Разработка программных средств	-	-
15	Разработка серверных ПС	Программист	100%
16	Разработка клиентских ПС	Программист	100%
17	Разработка программных средств завершена	-	-
18	Контроль качества серверных программных средств	-	-
19	Планирование процесса контроля качества	Тестировщик	100%
20	Тестирование серверных ПС	Тестировщик	100%
21	Тестирование клиентских ПС	Тестировщик	100%
22	Оценка качества программного кода	Тестировщик	100%
23	Контроль качества завершен	-	-
24	Завершение работы над проектом	-	-
25	Оформление документации	Программист Интернет	100% 400р
26	Сдача проекта	Программист	100%
27	Проект завершен	-	-
28	Конец проекта	-	-

Диаграмма Ганта [6], которая получена в результате планирования процесса разработки, представлена на рисунке 7.

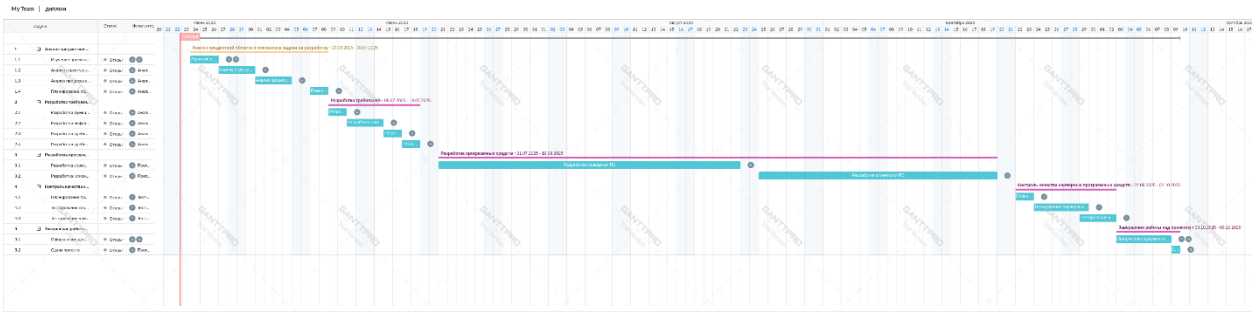

Рисунок 7 – Диаграмма Ганта

На данной диаграмме отображены все этапы разработки и задачи, а также требуемые ресурсы. Выполнение задач имеет последовательный характер.

Общие затраты на разработку проекта составили 182 400 рублей. Разработка проекта заняла 79 рабочих дней или 632 часа.

2 Анализ требований и постановка задачи на разработку

2.1 Анализ функциональных требований

Для анализа функциональных требований к ПО была создана диаграмма вариантов использования проекта, которая отражает основной функционал, доступный пользователю приложения. Диаграмма вариантов использования представлена на рисунке 8.

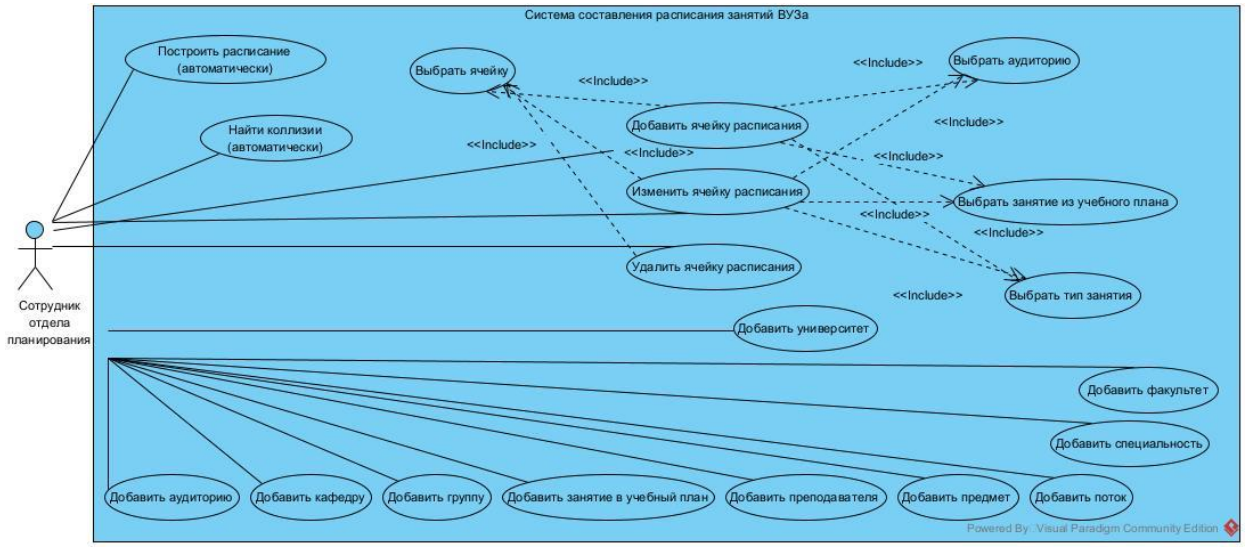

Рисунок 8 – Диаграмма вариантов использования

Составлена спецификация для основных прецедентов из диаграммы вариантов использования.

В таблице 4 приведено описание основного порядка выполнения варианта использования «Добавить ячейку расписания».

Таблица 4 – Спецификация прецедента «Добавить ячейку расписания»

Прецедент: добавить ячейку расписания
ID: 1
Краткое описание: Добавление элемента расписания
Главные актеры: Сотрудник отдела планирования

Продолжение таблицы 4

Основной поток: 1. Сотрудник выбирает ячейку расписания 2. Сотрудник выбирает занятие из учебного плана 3. Сотрудник выбирает тип занятия (лекционное/лабораторное) 4. Сотрудник выбирает аудиторию 5. Система сохраняет ячейку в базе данных

В таблице 5 приведено описание основного порядка выполнения варианта использования «Построить расписание (автоматически)». В разделе «Основной поток» приведено обобщенное описание работы алгоритма составления расписания занятий.

Таблица 5 – Спецификация прецедента «Построить расписание (автоматически)»

Прецедент: построить расписание (автоматически)
ID: 2
Краткое описание: Сотрудник отдела планирования инициирует процесс автоматического составления расписания. Составление расписания проводится системой.
Главные актеры: Сотрудник отдела планирования
Основной поток: 1. Специалист инициирует процесс составления расписания 2. Система автоматически составляет расписание 2.1. Система перебирает невыставленные занятия из учебного плана, находит удовлетворяющее наибольшему числу ограничений 2.2. Система повторяет процесс пока не будут выставлены все занятия из учебного плана либо пока не останется занятий, которые возможно выставить без нарушения главных ограничений

Продолжение таблицы 5

3. Система сохраняет расписание в БД 4. Система отображает расписание на клиентской части ПО

В таблице 6 приведено описание основного порядка выполнения варианта использования «Найти коллизии (автоматически)». Вариант использования должен выполняться только в том случае, если все занятия из учебного плана уже выставлены в расписании.

Таблица 6 – Спецификация прецедента «Найти коллизии (автоматически)»

Прецедент: найти коллизии (автоматически)
ID: 3
Краткое описание: Сотрудник отдела планирование инициирует процесс автоматического поиска коллизий и ошибок в уже составленном расписании
Главные актеры: Сотрудник отдела планирования
Предусловие: Все занятия из учебного плана присутствуют в расписании

На основе диаграммы вариантов использования и анализа аналогичных программных решений были сформированы следующие функциональные требования к разрабатываемому приложению:

– просмотр и добавление исходных данных;

– обеспечение создания и хранения расписания в ручном режиме;

– просмотр расписания для кафедр, групп, аудиторий и преподавателей;

– ручное редактирование созданного листа персонажа;

– автоматический поиск ошибок и коллизий;

– автоматическое составление расписания занятий.

Добавление исходных данных должно включать в себя удобный интерфейс для просмотра добавления данных обо всей иерархической структуре ВУЗа, предметах, преподавателях, аудиториях, группах и потоках, а также добавление и просмотр учебного плана.

Для редактирования ячейки расписания в ручном режиме специалисту отдела планирования необходимо будет выбрать ячейку (ячейка имеет параметры неделя, день, номер пары и группа), указать занятие группы из учебного плана, тип занятия (лекционное либо лабораторное) и аудиторию.

Автоматическое построение расписания может проводиться как на основе заполненного «каркаса» расписания, так и на пустом расписании. Для каждого занятия из учебного плана будет выбираться наиболее предпочтительное время проведения пары так, чтобы у групп и преподавателей не было «окон» (либо, если это невозможно, то свести их количество к минимуму). Аудитория для проведения занятия должна иметь достаточную вместимость и относиться к кафедре либо факультету, на котором проводится занятие.

Поиск коллизий также является автоматическим и будет срабатывать при выборе соответствующего пункта в меню. При поиске коллизий будет проводиться поиск «накладок» в расписании – например случаев, при которых в одной аудитории в одно время проводится несколько разных занятий. Такие «накладки» в расписании будут подсвечиваться красным, «окна» у групп – оранжевым, «окна» у преподавателей и несоответствие аудитории кафедре проводимого занятия – желтым.

2.2 Анализ нефункциональных требований

Нефункциональные требования описывают такие характеристики системы, которые не связаны с ее функционалом. В ходе разработки системы автоматического построения расписания занятий ВУЗа были определены следующие нефункциональные требования:

– доступность – система должна быть доступна с минимальным временем простоя (менее 10 минут в рабочее время, техническое обслуживание должно проводиться автономно в нерабочее время);

– поддержка работы нескольких пользователей – система должна позволять нескольким специалистам одновременно редактировать одно и то же расписание;

– производительность – система должна обеспечивать быстрое время отклика (менее одной секунды для не автоматических операций, менее 10 минут для автоматических операций на расписаниях с числом групп более 200);

– масштабируемость – система должна поддерживать увеличение количества данных (до 500 групп) без значительной потери производительности;

– удобство и простота использования – интерфейс системы должен быть интуитивно понятен и прост в использовании.

Требование к доступности системы особенно критично в периоды активного формирования расписания (за 1 – 2 недели до начала семестра). В эти сроки отдел планирования работает в интенсивном режиме, и даже кратковременный простой может сорвать утверждение расписания. Система должна обеспечивать бесперебойную работу с плановым обслуживанием исключительно в ночные часы, минимизировать влияние на рабочий процесс.

Производительность и масштабируемость являются ключевыми для практического внедрения системы. Время автоматического построения расписания для ВУЗа не должно превышать 20 минут, что сопоставимо с показателями аналогичных систем. Отклик интерфейса при добавлении данных вручную должен быть практически мгновенным (менее 1 секунды).

Программа должна быть удобна и интуитивно понятна для сотрудников отдела планирования. Интерфейс не должен быть перегружен деталями. В приложении должна быть удобная и понятная навигация (переход между первичными данными и расписанием), отображение расписание, учебного

плана и структуры ВУЗа. Дизайн приложения должен быть адаптивен для комфортной работы на различных разрешениях экрана и в основных браузерах.

2.3 Анализ требований к данным

Исходные данные и готовое расписание будут храниться в реляционной базе данных. Обращение к ним из программного кода будет проходить посредством технологии ORM (Object-Relational Mapping), которая позволяет представлять сущности базы данных в виде объектов и классов в языках программирования [7].

Концептуальная модель представляет собой описание основных сущностей (таблиц) и связей между ними без учета принятой модели БД и синтаксиса целевой СУБД [8].

Графическое представление концептуальной модели базы данных представлено на рисунке 9. Концептуальная модель построена в среде Visual Paradigm в виде обобщенной диаграммы классов, так как в приложении используется объектно-ориентированный подход и БД будет строиться не на основе SQL-кода, а на основе ORM-модели в языке Python. Данная диаграмма отражает основные сущности информационной системы, которые также являются Python-классами в ORM-представлении базы данных.

Структурные подразделения ВУЗа (иерархия Университет – Факультет – Кафедры) были представлены в виде одной сущности «Структурное подразделение» с иерархической связью, за которую отвечает атрибут «Родитель». Далее будет подразумеваться, что в базе данных таблица «Структурное подразделение» заполнена верно, без ошибок в иерархии. Кафедры имеют сокращенные названия для удобства восприятия, у остальных подразделений сокращения указывать не обязательно.

Между группами и потоками существует связь многие-ко-многим, так как одна группа может содержаться в нескольких потоках, а каждый поток содержит в себе одну или несколько групп.

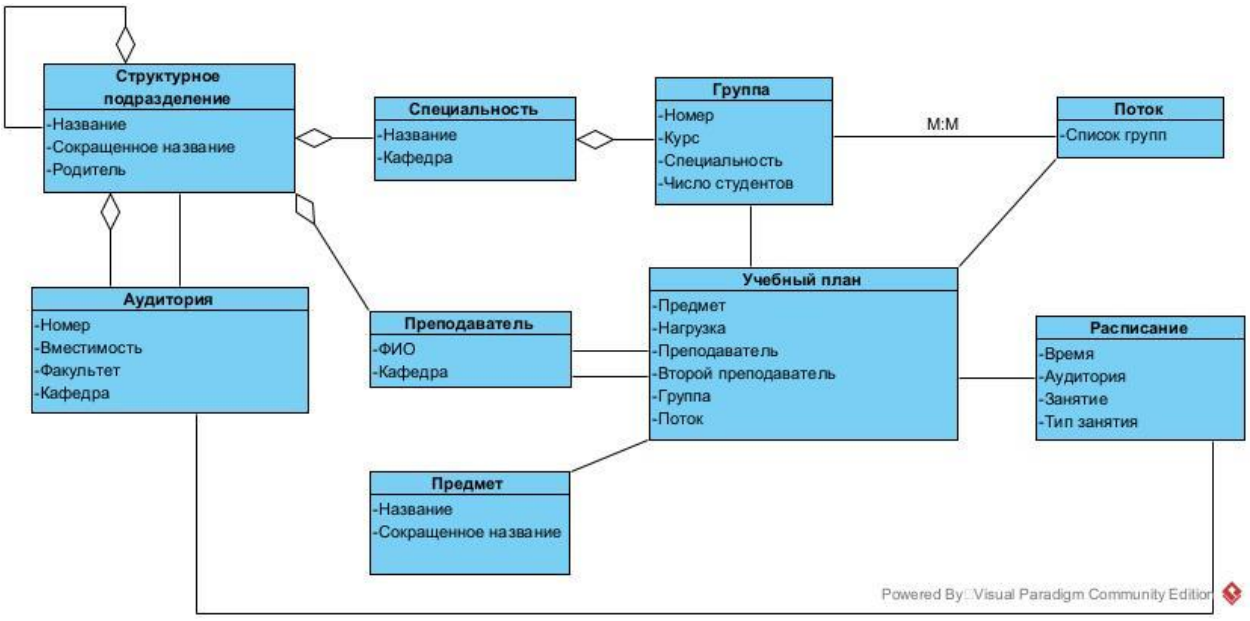

Рисунок 9 – Графическое представление концептуальной модели БД в виде диаграммы классов

Аудитории могут относиться как к отдельной кафедре, так и ко всему факультету в целом. Во втором случае атрибут «Кафедра» сущности «Аудитория» будет пустым (null).

Каждый предмет помимо полного названия имеет сокращенное, которое может совпадать с полным (например для предмета «Физика» сокращение будет «Физика»).

Учебный план представлен в виде соответствующей сущности. Учебный план имеет атрибуты предмета, преподавателей, групп и числа часов на проведение предмета за семестр. У предмета всегда есть главный преподаватель, второго преподавателя может не быть. Для занятия в учебном плане указывается либо одна группа, либо поток (список групп). Указать и группу, и поток нельзя.

Итоговое расписание является сочетанием информации о занятии, аудитории его проведения и времени проведения. Время является составным атрибутом и содержит информацию о номере недели (1 либо 2), дне (понедельник – суббота) и номере пары (1 – 8).

2.4 Анализ требований к пользовательскому интерфейсу

Пользовательский интерфейс системы проектируется с ориентацией на сотрудников отдела планирования. Обучение использованию программы должно занимать у специалиста минимум времени. В пользовательском интерфейсе должны использоваться термины, употребляемые сотрудниками отдела планирования.

В интерфейсе должно быть две отдельные вкладки: для ввода первичных данных и для составления расписания. При переходе между вкладками открытые в них данные не должны сбрасываться.

На вкладке ввода первичных данных должны быть панели для отображения структуры ВУЗа (университетов, факультетов, кафедр, специальностей в групп), потоков, предметов, преподавателей, аудиторий и учебного плана. На каждой панели должны быть кнопки для добавления новых данных. При нажатии на кнопку должна появляться форма ввода необходимых данных с кнопками «Сохранить» и «Отменить». При нажатии на кнопку «Отменить» форма должна закрываться без внесения изменений в БД. При нажатии на кнопку «Сохранить» должен отправляться запрос к серверной части проекта на добавление записи в БД. Если запрос успешен, то форма должна закрываться, а данные отображаться в таблице. Если запрос не успешен, то должно выводиться сообщение об ошибке. Запрос к серверу должен осуществляться только после корректного ввода всех данных.

На вкладке расписания должна быть представлена таблица расписания. Строками таблицы будет являться время проведения пары (день и номер пары), столбцами – номера групп. Переключение между 1 и 2 неделей должно осуществляться по нажатию на кнопки вверху экрана. В ячейках таблицы должны быть указаны предметы (с типом занятия), преподаватели и аудитории их проведения. Заполненную ячейку можно будет изменить либо удалить. Если ячейка пуста, то в ней будет кнопка со знаком «+», при нажатии на которую открывается форма добавления ячейки расписания.

Форма добавления и изменения ячейки расписания включает в себя неизменяемую информацию о неделе, дне, паре и группе, а также инструменты выбора занятия из учебного плана, типа занятия и аудитории. Если занятие не имеет отношение к группе, то оно не отображается к окне выбора. Если занятие уже выставлено в расписание, то оно также не отображается. Если выставлены все доступные лекционные занятия, то в качестве типа занятия можно указать только лабораторное занятие.

Расписание отображается для всех групп выбранной кафедры либо для всех групп всех кафедр выбранного факультета. Выбор отображаемых кафедр проводится посредством нажатия кнопок выбора факультета, университета и кафедры (либо кнопки «все» для отображения информации по всем кафедрам выбранного факультета. После выбора одного из университетов на форме выбора остаются только кнопки с названиями факультетов этого университета. Аналогично для выбранного факультета остаются только принадлежащие ему кафедры.

Кнопка автоматического составления расписания занятий должна находиться на вкладке составления расписания. При нажатии на кнопку запускается процесс составления расписания занятий. До завершения процесса вносить изменения в расписание и первичные данные запрещено. После завершения процесса составления расписания занятий новое расписание отображается в таблице. Если процесс завершился с ошибкой, то ошибка отображается на экране.

Кнопка автоматического поиска коллизий должна находиться рядом с кнопкой автоматического составления расписания. При нажатии на нее должен начинаться процесс поиска коллизий. Накладки в расписании должны подсвечиваться красным, «окна» у групп – оранжевым, «окна» у преподавателей и несоответствие аудитории кафедре проводимого занятия – желтым.

2.5 Обоснование выбора технологий и средств разработки

Проект будет состоять из клиентской и серверной частей.

Серверная часть проекта будет разработана на языке Python 3 с использованием фреймворка FastAPI для реализации веб-API и фреймворка SQLModel, который интегрируется с FastAPI.

FastAPI является одним из самых используемых Python-фреймворков для веб-приложений. Он имеет высокую производительность и безопасность, активно использует декораторы, аннотации типов и интроспекцию кода, что позволяет уменьшить количество шаблонного кода в веб-приложении. FastAPI автоматически генерирует и отображает документацию согласно спецификации OpenAPI [10].

В качестве СУБД выбрана SQLite, так как она является наиболее быстрой и простой в использовании СУБД. База данных в SQLite хранится в виде файла в папке проекта, что упрощает работу с ней. Выбор данной СУБД обусловлен тем, что в проекте используется ORM, поэтому предпочтение при выборе СУБД отдавалось не функционалу, а простоте.

Для клиентской части проекта была выбрана технология React и язык программирования Typescript. React – популярная библиотека для создания пользовательских интерфейсов, значительно упрощающая разработку веб-приложений на основе HTML, CSS и JavaScript. TypeScript – это расширенная версия языка JavaScript с добавлением статической типизации, что позволяет разрабатывать более надёжные программные продукты.

React позволяет создавать динамические web-приложения и удобно обмениваться данными с серверной частью проекта. TypeScript позволяет сделать разработку более строгой и безопасной, упрощает понимание кода благодаря использованию статической типизации.

В качестве среды разработки выбрана программа Microsoft Visual Studio Code, так как она позволяет работать со множеством различных языков программирования и технологий, в том числе Python, React и Typescript.

3 Разработка программных средств

3.1 Разработка архитектуры

Разрабатываемая информационная система составления расписания занятий ВУЗа будет построена на основе клиент-серверной архитектуры с разделением фронтенда и бекенда (клиентской и серверной части). Серверная часть будет построена на основе многослойно-монолитной архитектуры. Это означает, что система будет представлять собой единое приложение, разделенное на логические слои, каждый из которых отвечает за определенные функции. Такая архитектура сочетает в себе простоту монолитного подхода (единая кодовая база) и гибкость многослойной структуры, что упрощает разработку и поддержку системы [9].

Клиентская часть (frontend) будет отвечать за взаимодействие с пользователем. Она включает веб-интерфейс, разработанный с использованием HTML, CSS и TypeScript на технологии React. Предоставляет сотрудникам отдела планирования ВУЗа удобный доступ к функциям системы через браузер.

Серверная часть (backend) реализует бизнес-логику системы и состоит из трех слоев: слоя приложения (Application Layer), слоя доступа к данным и слоя базы данных.

Слой приложения включает контроллеры и сервисы, которые обрабатывают запросы от клиентской части и взаимодействуют с базой данных. Отвечает за выполнение операций, таких как добавление, редактирование и удаление записей об автомобилях

Слой доступа к данным обеспечивает взаимодействие с базой данных и отвечает за хранение и извлечение данных о расписании и о ВУЗе.

Слой базы данных хранит в себе все первичные данные о ВУЗе и данные о расписании, а также информацию о структуре данных. Для их отображения использует ORM SQLModel, которая взаимодействует с базой данных SQLite.

Была разработана диаграмма компонентов, описывающая архитектуру разрабатываемого приложения. Диаграмма представлена на рисунке 10.

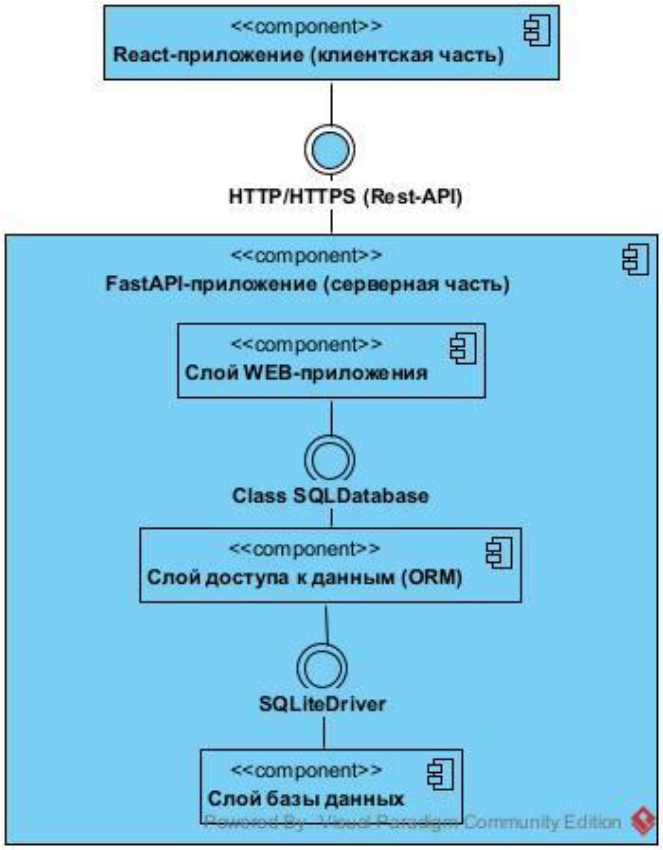

Рисунок 10 – Архитектура приложения, представленная в виде диаграммы компонентов

Диаграмма компонентов отображает разделение проекта на клиентскую и серверную части и показывает способ их взаимодействия – через REST API по протоколу HTTP. Клиентская часть является монолитным React-приложением, а серверная состоит из трех слоев, взаимодействующих друг с другом. Слой web-приложения взаимодействует со слоем доступа к данным через класс работы с БД SQLDatabase, а слой доступа к данным взаимодействует со слоем базы данных через ORM-модели и драйвер взаимодействия с СУБД SQLite.

3.2 Разработка клиентского приложения

3.2.1 Разработка архитектуры пользовательского интерфейса

Клиентская часть приложения представляет собой пользовательский интерфейс, который обеспечивает взаимодействие пользователя с системой. Она отвечает за отображение данных, полученных от серверной части, а также за отправку запросов для выполнения операций на добавление, редактирование и удаление данных. Клиентская часть разработана как веб-приложение, которое взаимодействует с серверной частью через REST-API, что обеспечивает гибкость и удобство в использовании. Пользовательский интерфейс разрабатывался с использованием технологии React, языков HTML, CSS и TypeScript.

Структура основных панелей и форм клиентского приложения показана на рисунке 11.

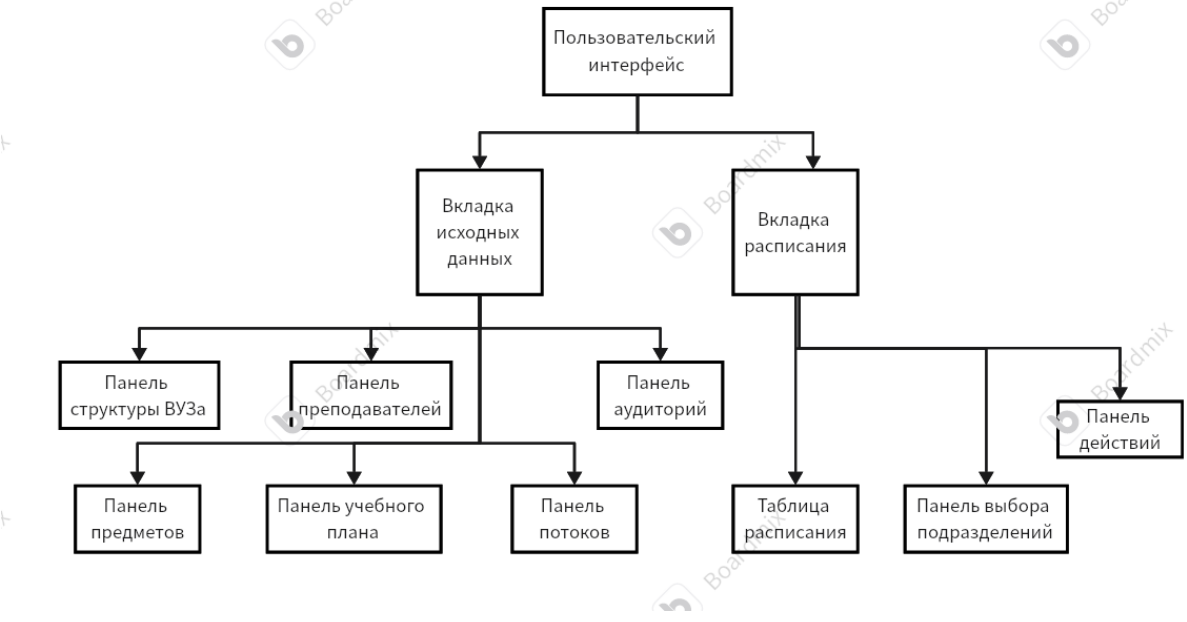

Рисунок 11 – Структура интерфейса клиентского приложения

Основными компонентами клиентской части являются вкладка исходных данных и вкладка расписания.

На вкладке исходных данных находятся:

– Панель структурных подразделений: отображает данные об университетах, факультетах, кафедрах, специальностях и группах (номер группы, число студентов, курс и направление обучения). Позволяет добавить структурные подразделения, специальности и кафедры.

– Панель профессорско-преподавательского состава – отображает данные о преподавателях в ВУЗе (на кафедрах, факультетах и в университетах). Позволяет добавить преподавателей.

– Панель аудиторий: содержит данные о номерах аудиторий факультетов и кафедр. Позволяет добавить аудитории.

– Панель предметов: содержит данные о полных и сокращённых названиях предметов. Позволяет добавить предметы.

– Панель потоков: содержит данные о потоках (списках групп). Позволяет добавить новый поток.

– Панель учебного плана: содержит данные о проведении занятий у групп, числе часов и преподавателях. Позволяет добавить занятие в учебный план.

На вкладке расписания находится панель выбора подразделений, которая включает в себя кнопки выбора кафедр и факультетов, для которых отображать расписание, панель действий, которая состоит из кнопки выбора недели, кнопок автоматического построения расписания и автоматического поиска коллизий, а также таблица расписания занятий. В таблице отображаются данные для выбранной недели по группам выбранных кафедр. В пустые ячейки таблицы расписания занятий можно добавить предмет, в заполненных есть кнопки изменения и удаления ячейки.

Добавление и изменение всех данных производится через формы на всплывающих окнах.

3.2.2 Разработка визуальных прототипов

При исследовании аналогичного ПО был выбран дизайн приложения: минималистичный, без лишних деталей и цветового выделения, без мелких

элементов. Такой интерфейс поможет сотруднику отдела планирования качественно выполнять свою работу и не путаться во множестве вкладок.

На рисунке 12 показана панель структурных подразделений ВУЗа.

Рисунок 12 – Панель структурных подразделений ВУЗа

На рисунке 13 показана панель профессорско-преподавательского состава ВУЗа.

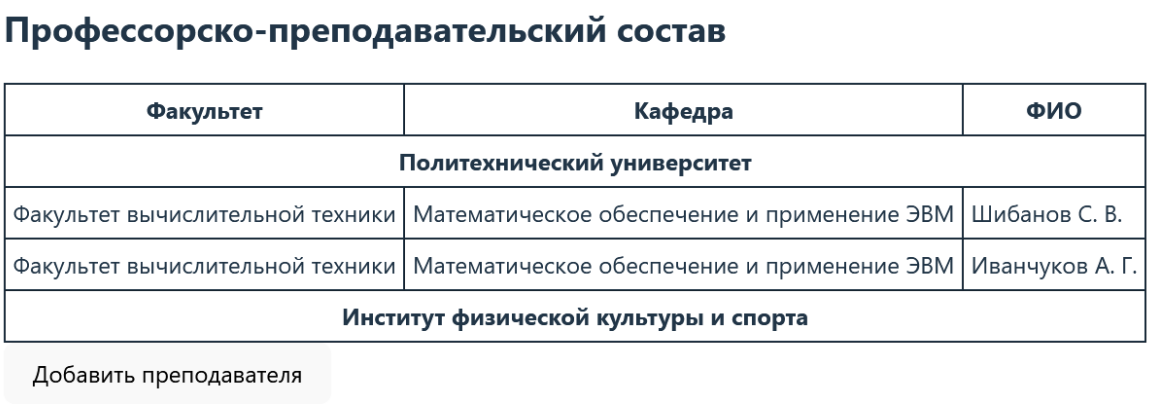

Рисунок 13 – Панель профессорско-преподавательского состава

На рисунке 14 показана панель потоков.

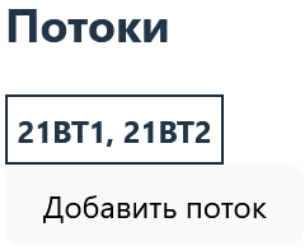

Рисунок 14 – Панель потоков

На рисунке 15 показана форма добавления группы. Остальные формы выглядят аналогично.

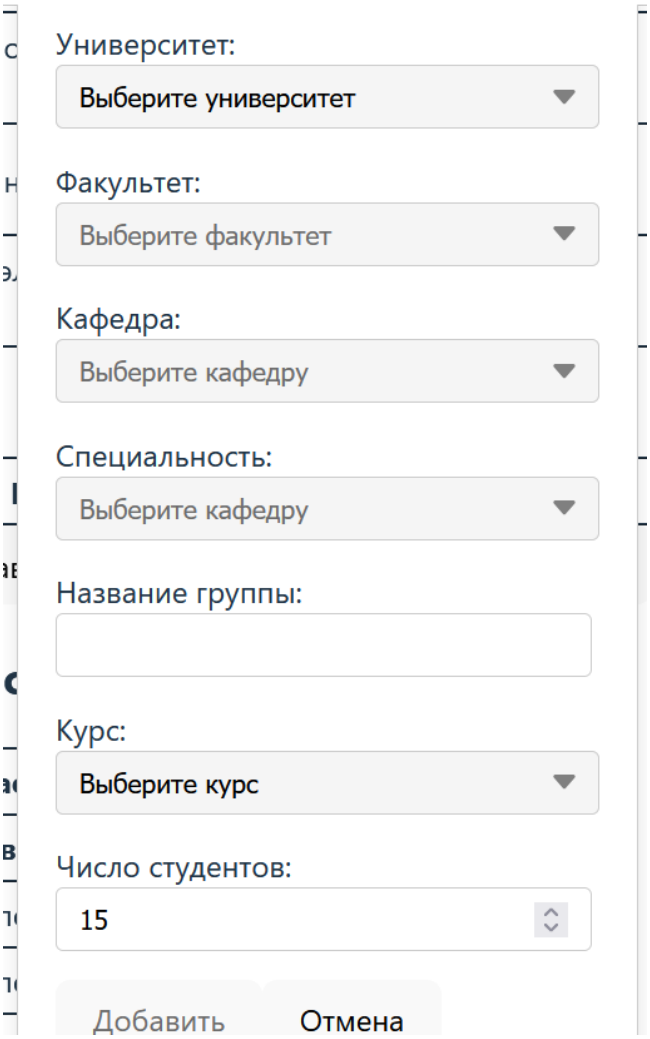

Рисунок 15 – Форма добавления группы

На рисунке 16 показана панель аудиторий.

Аудитории

Номер	Факультет	Кафедра
Политехнический университет		
7а-206	Факультет вычислительной техники	
7б-324	Факультет вычислительной техники	Математическое обеспечение и применение ЭВМ
Институт физической культуры и спорта		

Добавить аудиторию

Рисунок 16 – Панель аудиторий

На рисунке 17 показана панель предметов.

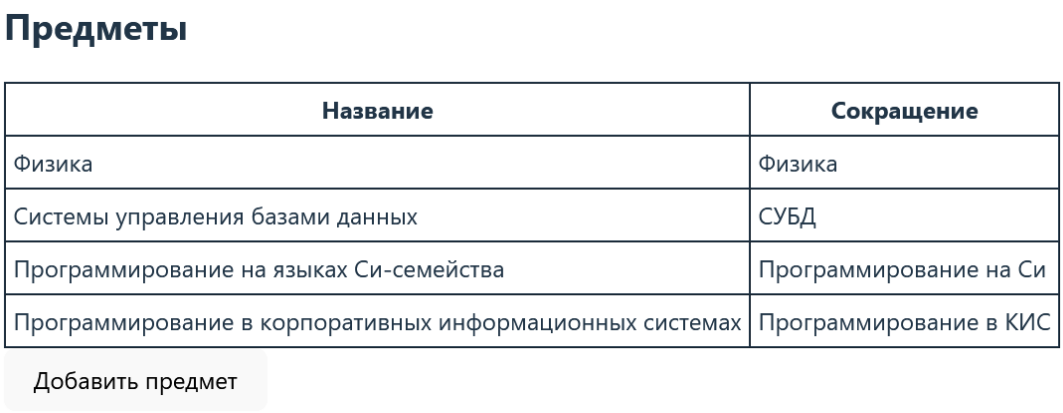

Рисунок 17 – Панель предметов

На рисунке 18 показана панель учебного плана.

Рисунок 18 – Панель учебного плана

На рисунке 19 отображена вкладка «Расписание».

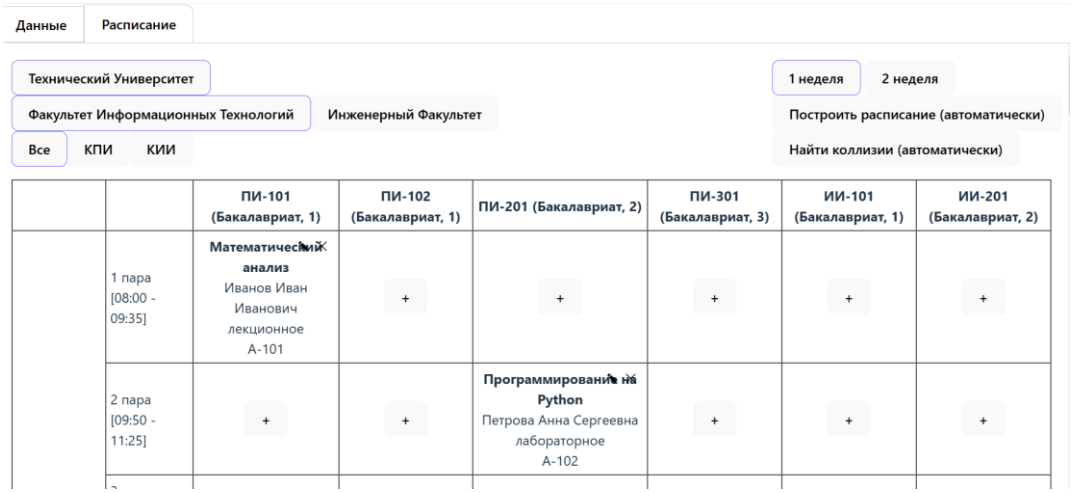

Рисунок 19 – Вкладка «Расписание»

Стоит более подробно рассмотреть вкладку «Расписание». В верхней части вкладки находится панель для работы с расписанием: слева – кнопки выбора отображаемых кафедр, справа – кнопки выбора недели, автоматического составления расписания занятий и автоматического поиска коллизий.

3.2.3 Разработка программного обеспечения клиентского приложения с применением библиотеки React

Для каждого из элементов пользовательского интерфейса был создан свой React-компонент. Диаграмма компонентов клиентской части приложения представлена на рисунке 20.

Рисунок 20 – Компоненты клиентской части приложения

Основной задачей клиентского приложения является обеспечение взаимодействия пользователя с сервером через удобный интерфейс.

Клиентская часть отправляет на сервер HTTP-запросы на получение данных (GET), добавление данных (POST), обновление расписания (PUT) и удаление элемента расписания (DELETE). Серверная часть обрабатывает эти запросы, выполняет необходимые операции с базой данных и возвращает результат в формате JSON. Это взаимодействие обеспечивает синхронизацию данных между клиентской и серверной частями, что позволяет пользователю работать с актуальной информацией [12].

3.3 Разработка серверной части приложения

3.3.1 Разработка архитектуры серверной части приложения

Код серверной части приложения разбит на отдельные файлы, которые организованы в соответствии с архитектурой приложения. Расположение файлов проекта описывается в диаграмме развертывания, представленной на рисунке 21.

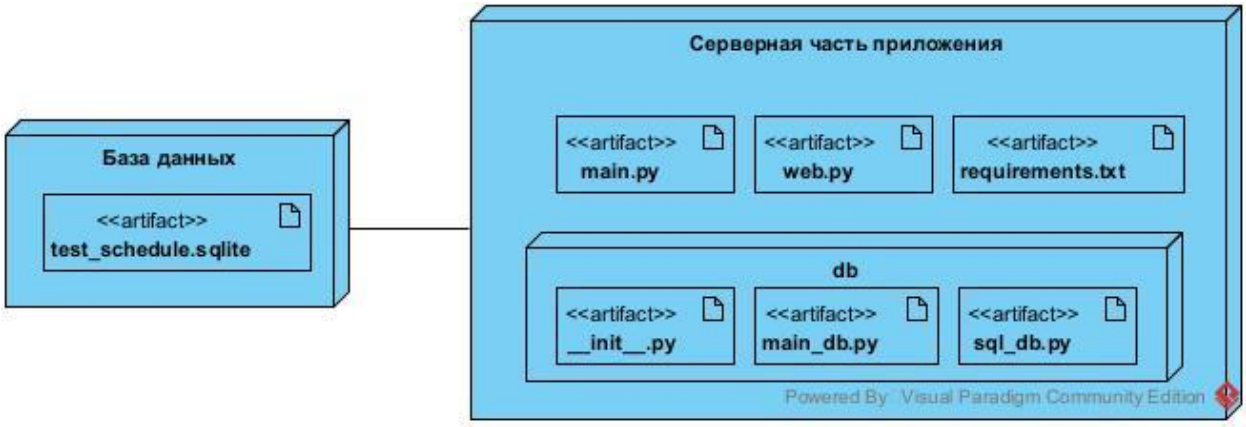

Рисунок 21 – Диаграмма развертывания серверной части приложения

Диаграмма показывает, что серверная часть состоит из нескольких ключевых компонентов: точки входа в проект (main.py), модуля с классом веб-приложения (web.py), модуля со списком зависимостей (requirements.txt), модулей для доступа к базе данных: main_db.py, определяющего общие правила взаимодействия с базами данных и sql_db.py, определяющего взаимодействие с реляционными базами данных. Файл __init__.py является вспомогательным и необходим для инициализации каталога db/ модуля. В

основном каталоге проекта находится файл test_schedule.sqlite, в котором хранится база данных проекта.

3.3.2 Разработка ORM-модели базы данных

Логическая модель базы данных, полученная в результате развертывания концептуальной модели, представлена на рисунке 22. Сущности и атрибуты предметной области были переименованы в соответствии с нотацией языка Python, была добавлена типизация атрибутов.

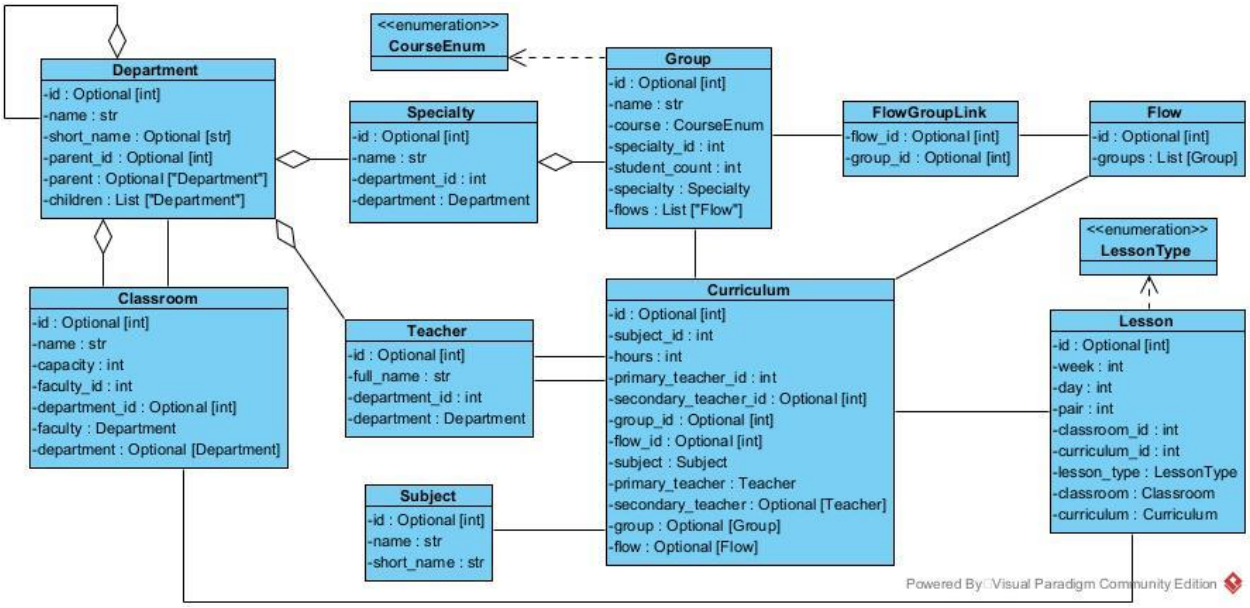

Рисунок 22 – Логическая модель базы данных

В результате построения логической модели связь многие-ко-многим между группами и потокам раскрылась в новый атрибут FlowGroupLink.

Для хранения данных о типа занятий и курсах были созданы перечисления CourseEnum и LessonType.

Для хранения связей между сущностями в каждой сущности были созданы атрибуты типа Relationship, которые хранят ссылку на объект класса, с которым установлена связь.

Данные о структурных подразделениях ВУЗа хранятся в классе Department. Идентификатором является id, по стандарту SQLModel его тип указан как Optional[int]. В классе реализована рекурсивная связь. На родителя указывает атрибут parent_id – у университетов он равен None (Null). Сама

связь реализована атрибутами parent (указатель на родителя) и children (указатель на потомков). Каждое структурное подразделение имеет уникальное имя name, у кафедр указываются сокращения short_name. Исходный код класса представлен в листинге 1.

Листинг 1 – Исходный код класса Department

```
class Department(SQLModel, table=True):
    """Структурное подразделение
    Подразлеления без родителя - университет
    Дочерние университету подразделения - факультеты
    Дочерние факультетам - кафедры"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    short_name: Optional[str]
    parent_id: Optional[int] = Field(default=None,
foreign_key="department.id")

    # Связь на самого себя для иерархии
    parent: Optional["Department"] = Relationship(
        back_populates="children",
        sa_relationship_kwargs={
            "remote_side": "Department.id",
            "foreign_keys": "Department.parent_id"
        }
    )
    children: List["Department"] =
Relationship(back_populates="parent")
```

Данные о специальностях хранятся в классе Specialty, который связан с кафедрой через атрибут department_id и связь department. Исходный код класса Specialty представлен в листинге 2.

Листинг 2 – Исходный код класса Specialty

```
class Specialty(SQLModel, table=True):
    """Специальность"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    department_id: int = Field(foreign_key="department.id")

    department: Department = Relationship()
```

Данные о курсах хранятся в перечислении CourseEnum. Исходный код перечисления представлен в листинге 3.

Листинг 3 – Исходный код перечисления CourseEnum

```
class CourseEnum(str, Enum):
    """Возможные значения курса"""
```

```
    BACHELOR_1 = "Бакалавриат, 1"
    BACHELOR_2 = "Бакалавриат, 2"
    BACHELOR_3 = "Бакалавриат, 3"
    BACHELOR_4 = "Бакалавриат, 4"
    MASTER_1 = "Магистратура, 1"
    MASTER_2 = "Магистратура, 2"
    SPECIALIST_1 = "Специалитет, 1"
    SPECIALIST_2 = "Специалитет, 2"
    SPECIALIST_3 = "Специалитет, 3"
    SPECIALIST_4 = "Специалитет, 4"
    SPECIALIST_5 = "Специалитет, 5"
```

Данные о типах занятий хранятся в перечислении LessonType. Исходный код перечисления представлен в листинге 4.

Листинг 4 – Исходный код перечисления LessonType

```
class LessonType(str, Enum):
    """Возможные типы занятий"""
    LECTURE = "лекционное"
    LAB = "лабораторное"
```

Данные о группах хранятся в классе Group. Класс содержит данные о номере группы (атрибут name), курсе обучения (атрибут course), специальности (атрибут specialty_id и связь specialty), числе студентов в группе (атрибут student_count). Атрибут flows необходим для реализации связи многие-ко-многим с потоками. Исходный код класса представлен в листинге 5.

Листинг 5 – Исходный код класса Group

```
class Group(SQLModel, table=True):
    """Группа"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    course: CourseEnum
    specialty_id: int = Field(foreign_key="specialty.id")
    student_count: int

    specialty: Specialty = Relationship()
    # Обратная связь для связи многие-ко-многим с Flow
    flows: List["Flow"] = Relationship(back_populates="groups",
link_model=FlowGroupLink)
```

Данные о потоках (списках групп) хранятся в классе Flow. Исходный код класса представлен в листинге 6.

Листинг 6 – Исходный код класса Flow

```
class Flow(SQLModel, table=True):
```

```
    """Поток"""
    id: Optional[int] = Field(default=None, primary_key=True)

    groups: List[Group] = Relationship(back_populates="flows",
link_model=FlowGroupLink)
```

Связь многие-ко-многим между группами и потоками реализуется благодаря вспомогательному классу FlowGroupLink. Он позволяет сопоставить идентификатор группы (flow_id) и идентификатор потока (group_id). Класс FlowGroupLink указан в Relationship flows и groups в классах Group и Flow в качестве link_model, что позволяет ему определять связь многие-ко-многим. Исходный код класса представлен в листинге 7.

Листинг 7 – Исходный код класса FlowGroupLink

```
class FlowGroupLink(SQLModel, table=True):
    """Связь многие-ко-многим между Потоком и Группами"""
    flow_id: Optional[int] = Field(default=None,
foreign_key="flow.id", primary_key=True)
    group_id: Optional[int] = Field(default=None,
foreign_key="group.id", primary_key=True)
```

Данные об аудиториях хранятся в классе Classroom. Каждая аудитория имеет номер (name), вместимость (capacity), относится к кафедре либо факультету в целом. Если аудитория относится к кафедре, то в атрибуте department_id указывается id родительской кафедре. Если аудитория относится ко всему факультету, то department_id не указывается. Связь с родительским факультетом (атрибут faculty_id) указывается всегда для удобства работы с аудиториями в коде. Исходный код класса представлен в листинге 8.

Листинг 8 – Исходный код класса Classroom

```
class Classroom(SQLModel, table=True):
    """Аудитория"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    capacity: int
    faculty_id: int = Field(foreign_key="department.id")
    department_id: Optional[int] =
Field(foreign_key="department.id")

    faculty: Department = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Classroom.faculty_id"}
```

```
    )
    department: Optional[Department] = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Classroom.department_id"}
    )
```

Данные о предметах хранятся в классе Subject. У каждого предмета есть полное (name) и сокращенное (short_name) название. У некоторых предметах они могут быть раны (например у предмета «Физика»). Исходный код класса представлен в листинге 9.

Листинг 9 – Исходный код класса Subject

```
class Subject(SQLModel, table=True):
    """Предмет"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    short_name: str
```

Данные о профессорско-преподавательском составе представлены в классе Teacher. В нем указано ФИО каждого преподавателя (full_name) и кафедра, на которой он преподает. Исходный код класса представлен в листинге 10.

Листинг 10 – Исходный код класса Teacher

```
class Teacher(SQLModel, table=True):
    """Преподаватель"""
    id: Optional[int] = Field(default=None, primary_key=True)
    full_name: str = Field(index=True, unique=True)
    department_id: int = Field(foreign_key="department.id")

    department: Department = Relationship()
```

Данные об учебном плане содержатся в классе Curriculum. Учебный план является совокупностью предметов, преподавателей, которые их проводят, групп, у которых эти предметы проводятся и нагрузки – количестве часов проведения предмета за семестр. Основной преподаватель (primary_teacher_id) у предмета есть всегда, а дополнительный (secondary_teacher_id) указывается только в определенных случаях. Также у объекта учебного плана должна быть указана либо группа, либо поток. Указать и то, и другое одновременно нельзя. За реализацию этого ограничения отвечает метод добавления данных в учебный план, в самом классе ограничение не реализовано, но есть ограничение на число часов – оно должно

соответствовать ограничению «hours IN (72, 108, 144)». Данные функционал также можно было реализовать путем создания нового перечисления, но для упрощения кода был выбран этот способ. Исходный код класса представлен в листинге 11.

Листинг 11 – Исходный код класса Curriculum

```
class Curriculum(SQLModel, table=True):
    """Учебный план"""
    __table_args__ = (
        CheckConstraint("hours IN (72, 108, 144)",
name="hours_check"),
    )

    id: Optional[int] = Field(default=None, primary_key=True)
    subject_id: int = Field(foreign_key="subject.id")
    hours: int
    primary_teacher_id: int = Field(foreign_key="teacher.id")
    secondary_teacher_id: Optional[int] =
Field(foreign_key="teacher.id")
    group_id: Optional[int] = Field(default=None,
foreign_key="group.id")
    flow_id: Optional[int] = Field(default=None,
foreign_key="flow.id")

    subject: Subject = Relationship()
    primary_teacher: Teacher = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Curriculum.primary_teacher_id"}
    )
    secondary_teacher: Optional[Teacher] = Relationship(
        sa_relationship_kwargs={
            "foreign_keys": "Curriculum.secondary_teacher_id"
        }
    )
    group: Optional[Group] = Relationship()
    flow: Optional[Flow] = Relationship()
```

Расписание занятий представлено в виде объектов класса Lesson. Класс Lesson содержит данные о занятии из учебного плана (curriculum_id), аудитории проведения занятия (classroom_id) и типе занятия (lesson_type). Время проведения занятия является сочетанием значений трех атрибутов: week для номера недели (ограничение «week IN (1, 2)»), day для дня недели (понедельник – суббота, ограничение «day BETWEEN 1 AND 6») и pair для номера пары (число от 1 до 8, ограничение «pair BETWEEN 1 AND 8»). Исходный код класса представлен в листинге 12.

Листинг 12 – Исходный код класса Curriculum

```
class Lesson(SQLModel, table=True):
    """Занятие в расписании"""
    __table_args__ = (
        CheckConstraint("week IN (1, 2)", name="week_check"),
        CheckConstraint("day BETWEEN 1 AND 6",
name="day_check"),
        CheckConstraint("pair BETWEEN 1 AND 8",
name="pair_check"),
    )

    id: Optional[int] = Field(default=None, primary_key=True)
    week: int
    day: int
    pair: int
    classroom_id: int = Field(foreign_key="classroom.id")
    curriculum_id: int = Field(foreign_key="curriculum.id")
    lesson_type: LessonType

    classroom: Classroom = Relationship()
    curriculum: Curriculum = Relationship()
```

Работа с базой данных осуществляется через класс SQLDatabase. Фрагмент кода инициализации класса представлен в листинге 13.

Листинг 13 – Фрагмент кода класса SQLDatabase

```
class SQLDatabase(Database):
    '''Класс для работы с реляционными БД через ORM'''
    def __init__(self, db_url: str) -> None:
        self.engine = create_engine(db_url)
        SQLModel.metadata.create_all(self.engine)
```

Метод __init__ является одним из системных методов в Python и определяет действия, которые выполняются при создании нового объекта класса [11]. В качестве аргумента при создании класса он принимает строку db_url, которая определяет механизм подключения к базе данных. В программе используется «db_url = 'sqlite:///test_schedule.sqlite'» – это значит, что происходит подключение к СУБД SQLite к базе данных test_schedule, которая располагается в файле test_schedule.sqlite в текущей директории. Логин и пароль для доступа не требуются.

Метод create_engine является одним из методов ORM-фреймворка SQLModel и отвечает за подключение к базе данных и представлении ее в виде созданных ранее классов. Метод SQLModel.metadata.create_all(self.engine) отвечает за создание в базе данных тех таблиц, которых там нет.

3.3.3 Разработка методов запросов к базе данных и представлений для реализации аналитических бизнес-процессов при помощи ORM

Основные методы взаимодействия с БД были определены в файле main_db.py в абстрактном классе Database и делятся на несколько основных групп: методы добавления данных, принимающие в качестве аргументов данные из формы в пользовательском интерфейсе и возвращающие id новой записи, методы получения данных, методы ручной работы с расписанием и методы автоматической работы с расписанием.

Спецификация методов добавления данных представлена в таблице 7.

Таблица 7 – Спецификация методов добавления данных

Метод	Описание	Аргументы	Возвращает
add_structural_divizion	Добавление структурного подразделения	parent_id: Optional[int] name: str	Идентификатор добавленного подразделения
add_speciality	Добавление специальности	department_id: int name: str	Идентификатор добавленной специальности
add_group	Добавление группы	speciality_id: int name: str course: CourseEnum student_count: int	Идентификатор добавленной группы
add_teacher	Добавление преподавателя	department_id: int name: str	Идентификатор добавленного преподавателя
add_classroom	Добавление аудитории	faculty_id: int department_id: Optional[int] name: str capacity: int	Идентификатор добавленной аудитории
add_subject	Добавление предмета	name: str short_name: str	Идентификатор добавленного предмета
add_flow	Добавление потока	groups: list[str]	Идентификатор добавленного потока
add_lesson_to_plan	Добавление занятия в учебный план	subject_id: int hours: int, primary_teacher_id: int secondary_teacher_id: int group_id: Optional[int] flow_id: Optional[int]	Идентификатор добавленного занятия

Спецификация методов получения данных представлена в таблице 8.

Таблица 8 – Спецификация методов получения данных

Метод	Описание	Аргументы	Возвращает
get_university_data	Получение данных о полной структуре университета	-	Данные о структурных подразделениях, специальностях, группах, преподавателях и аудиториях
get_subjects	Получение предметов	-	Предметы
get_flows	Получение потоков	-	Потоки
get_curriculum	Получение учебного плана	-	Учебный план
get_schedule	Получение расписание	-	Расписание занятий

Для более строгой типизации методов структуры данных ответов были разработаны на основе классов библиотеки pydantic, которая позволят отображать структуру JSON-файлов в виде классов языка Python. Полный код методов и их ответов представлен в приложении Б.

Спецификация методов ручной работы с расписание представлена в таблице 9.

Таблица 9 – Спецификация методов ручной работы с расписанием

Метод	Описание	Аргументы	Возвращает
add_lesson_to_schedule	Добавление элемента расписания	week: int day: int pair: int classroom_id: int curriculum_id: int, lesson_type: LessonType	Идентификатор добавленного элемента
edit_schedule_cell	Редактирование элемента расписания с указанным id	id: int classroom_id: int curriculum_id: int lesson_type: LessonType	-
remove_schedule_cell	Удаление элемента расписания с указанным id	id: int	-

Спецификация методов автоматической работы с расписание представлена в таблице 10.

Таблица 10 – Спецификация методов автоматической работы с расписанием

Метод	Описание	Аргументы	Возвращает
auto_schedule	Автоматического построение расписания	-	-
find_collisions	Автоматический поиск коллизий	-	Структура данных с информацией о критических ошибках (накладки, в клиентской части подсвечиваются красным), некритических ошибках («окна» у студентов между пар) и недочетах («окна» у преподавателей)

Реализацией абстрактного класса Database является класс SQLDatabse из файла sql_db.py. SQLDatabse реализует все абстрактные методы класса Database, а также содержит метод генерации тестовых данных. В методах добавления данных проверяются все ограничения (например при добавлении аудитории проверяет, существует ли аудитория с таким названием, а также существуют ли указанные кафедра и факультет). В случае ошибки метод инициирует исключение, которое позже интерпретируется в качестве ошибки HTTP-запроса с соответствующим кодом и текстом.

3.3.4 Разработка алгоритма автоматического составления расписания

Алгоритм автоматического составления расписания университетских занятий работает с учётом семестровой нагрузки и различных ограничений ресурсов [13]. Сначала система собирает информацию о текущей занятости аудиторий, преподавателей и групп в течение двухнедельного цикла. Затем анализируются учебные планы, где для каждого плана определяется количество и тип необходимых занятий: для 72 часов требуется одна лекция

за две недели, для 108 часов – одна лекция и одна лабораторная работа, а для 144 часов – одна лекция и две лабораторные работы.

Обобщенная блок-схема алгоритма представлена на рисунке 23.

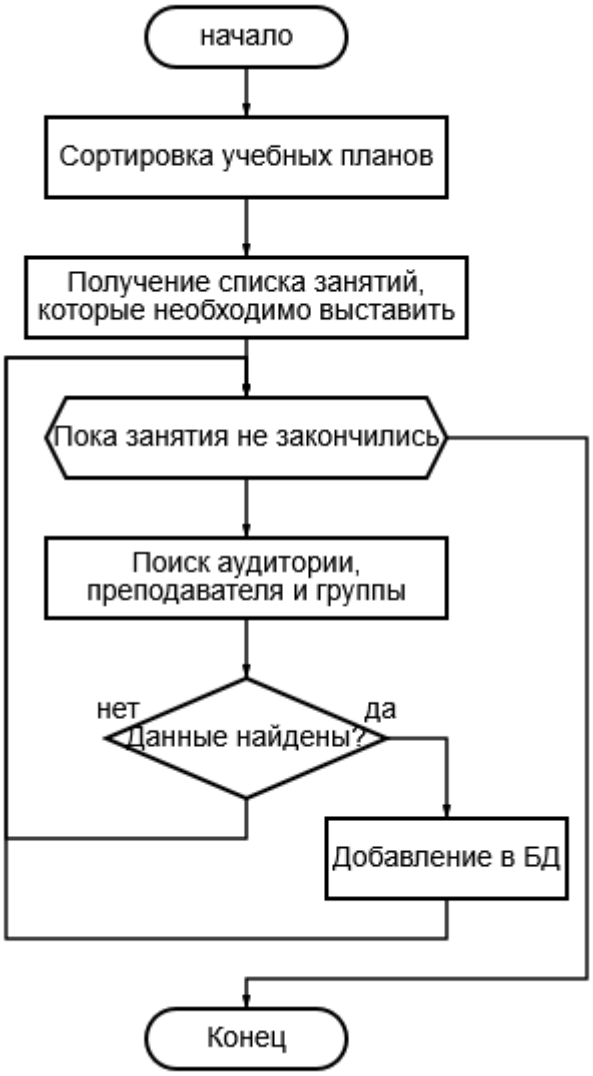

Рисунок 23 – Обобщенная блок-схема алгоритма автоматического составления расписания

Алгоритм сортирует учебные планы по сложности размещения, отдавая приоритет занятиям с большими группами студентов и занятиям, требующим вместительных аудиторий. Для каждого типа занятия в учебном плане система последовательно перебирает доступные временные ячейки (неделя 1 или 2, дни с понедельника по субботу, пары с 1 по 8). На каждой «временной ячейке» проверяются три ключевых ограничения: доступность аудитории

подходящего размера, отсутствие временных конфликтов у преподавателей и свободное расписание у всех задействованных групп студентов.

Когда находится подходящее время и аудитория для проведения занятия, система фиксирует его в расписании и обновляет информацию о занятости ресурсов. Если для какого-либо занятия не удаётся найти подходящее время, система сохраняет эту информацию для последующего анализа. По завершении процесса все изменения сохраняются в базу данных.

Математическое описание алгоритма представлено ниже.

Исходными данными для составления алгоритма являются множество учебных планов C, (занятия из учебного плана $c \in C$), множество групп студентов G, множество преподавателей для занятия $P(c)$, множество подходящих аудиторий $R(c)$ для занятия c, где $\text{вместимость}(r) \geq \sum_{g \in G(c)} \text{размер}(g)$, количество часов в занятии $H(c)$ из учебного плана с, где $H(c) \in \{72, 108, 144\}$, функция $T(c)$, которая возвращает список пар для занятия c:

$$T(c) = \begin{cases} [\textbf{LECTURE}] \text{ если } \mathbf{H(c) = 72} \\ [\textbf{LECTURE}, \textbf{LAB}] \text{ если } \mathbf{H(c) = 108} \\ [\textbf{LECTURE}, \textbf{LAB}, \textbf{LAB}] \text{ если } \mathbf{H(c) = 108} \end{cases}$$

Результатом работы алгоритма является набор множеств:

– $B_{classroom} = \{(w, d, p, r) | r \text{ занята в неделю } w, \text{день } d, \text{пару } p\}$;

– $B_{teacher} = \{(w, d, p, t) | t \text{ занят в неделю } w, \text{день } d, \text{пару } p\}$;

– $B_{group} = \{(w, d, p, g) | g \text{ занята в неделю } w, \text{день } d, \text{пару } p\}$.

Перед началом работы алгоритм сортирует множество C, по убыванию аудиторий, в которых может проводиться занятие. Аудитории рассчитываются на основе числа студентов в группах и размером аудитории. Размер аудитории определяет функция $\text{вместимость}(c)$,

Для каждого занятия $\tau \in T(c)$ алгоритм пытается найти такое (w, d, p, r) такое, что $(w, d, p, r) \notin B_{classroom}$, $(w, d, p, t) \notin B_{teacher} \; \forall \; t \in P(c)$, $(w, d, p, g) \notin B_{group} \; \forall \; g \in G(c)$ и $r \in R(c)$.

Если решение найдено, то алгоритм добавляет данные в расписание: $B_{classroom} \leftarrow B_{classroom} \cup \{(w, d, p, r)\}$, $B_{teacher} \leftarrow B_{teacher} \cup \{(w, d, p, t) | t \in P(c)\}$, $B_{group} \leftarrow B_{group} \cup \{(w, d, p, g) | g \in G(c)\}$.

Если решение не найдено, то алгоритм регистрирует невозможность размещения занятия.

Алгоритм использует эвристику «Наиболее ограниченная переменная» (Most Constrained Variable, MCV) является фундаментальным принципом в решении задач удовлетворения ограничений (Constraint Satisfaction Problems, CSP). Её суть заключается в приоритезации переменных с наименьшим количеством допустимых значений или находящихся под максимальными ограничениями. В контексте указанного класса задач это означает, что алгоритм сначала работает с переменными, на которые наложено наименьшее количество ограничений, что позволяет быстро выявлять потенциальные конфликты и сокращать пространство перебора. Например, в задачах автоматического составления расписаний сначала рассматриваются занятия, требующие специализированных аудиторий или затрагивающие множество групп, – такие элементы создают наибольшее число взаимозависимостей. Данная стратегия основана на идее, что «узкие места» системы следует разрешать в первую очередь, чтобы предотвратить каскадные сбои на поздних этапах поиска решения [14].

Ключевое преимущество MCV – сокращение вычислительных затрат (времени, потраченного на получение результата). Выбор наиболее ограниченной переменной минимизирует ветвление дерева поиска, поскольку её неудачное назначение сразу приводит к отсечению нежизнеспособных вариантов. Это особенно критично в NP-трудных задачах, к которым относится задача составления расписания. В этом классе задач полный перебор невозможен из-за кратно возрастающего времени работы алгоритма [15].

3.3.5 Разработка алгоритма автоматического поиска коллизий

Алгоритм анализа расписания и поиска в нем ошибок и коллизий базируется на принципах верификации временных ограничений, широко применяемых в системах автоматизированного планирования. Основная задача – выявление двух типов проблемных ситуаций: прямых конфликтов ресурсов и неоптимальных временных промежутков. Методология восходит к классическим работам по анализу целостности временных данных, где ключевым критерием является проверка непересекаемости интервалов занятости ресурсов [16].

Обобщенная блок-схема работы алгоритма представлена на рисунке 24.

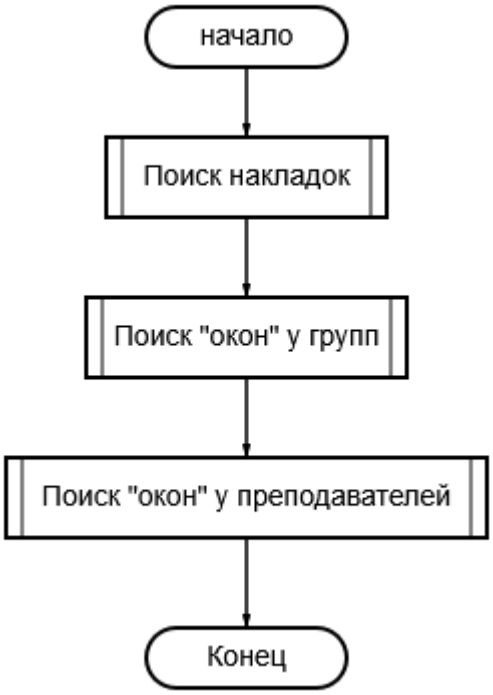

Рисунок 24 – Блок-схема алгоритма поиска коллизий

«Окна» – свободные промежутки между занятиями – выявляются через анализ последовательности занятости ресурса в течение учебного дня. После упорядочивания пар по времени выполнения алгоритм вычисляет разрывы между соседними занятиями. Промежуток считается «окном», если между окончанием одного занятия и началом следующего есть хотя бы одна

свободная пара. Для групп и преподавателей эти данные собираются раздельно, что позволяет группировать данные о них в разных частях кода и возвращать отдельный результат. Важно отметить, что «окна» не являются ошибкой при составлении расписания, но их наличие указывает на потенциальную возможность оптимизации расписания [17].

Специалист отдела планирования может проводить оптимизацию в ручном режиме, так как алгоритм не вносит изменения в данные, а лишь находит несоответствия в них.

3.3.6 Разработка Web-API приложения

Для обеспечения взаимодействия базы данных с клиентской частью приложения было создано API приложения, которое предоставляет доступ к методам класса SQLDatabase через HTTP-запросы. За это отвечает класс WebApp из файла web.py. Код инициализации класса WebApp приведен в листинге 14.

Листинг 14 – Метод __init__ класса WebApp

```
class WebApp:
    '''Класс для настройки веб-API'''
    def __init__(self, db: Database, listen_params:
ListenParams) -> None:
        self.db = db
        self.listen_params = listen_params
        self.app = FastAPI()

        origins = [
            'http://localhost:5173',
            'localhost:5173'
        ]

        self.app.add_middleware(
            CORSMiddleware,
            allow_origins=origins,
            allow_credentials=True,
            allow_methods=['*'],
            allow_headers=['*']
        )

        self.app.add_api_route(
            '/structural_divizion',
            self.add_structural_divizion,
```

```
            methods=["POST"]
        )
        self.app.add_api_route(
            '/speciality', self.add_speciality, methods=["POST"]
        )
        self.app.add_api_route('/group', self.add_group,
methods=["POST"])
        self.app.add_api_route('/teacher', self.add_teacher,
methods=["POST"])
        self.app.add_api_route(
            '/classroom', self.add_classroom, methods=["POST"]
        )
        self.app.add_api_route('/subject', self.add_subject,
methods=["POST"])
        self.app.add_api_route(
            '/subject', self.db.get_subjects, methods=["GET"]
        )
        self.app.add_api_route('/flow', self.add_flow,
methods=["POST"])
        self.app.add_api_route('/flow', self.db.get_flows,
methods=["GET"])
        self.app.add_api_route(
            '/curriculum', self.add_lesson_to_plan,
methods=["POST"]
        )
        self.app.add_api_route(
            '/curriculum', self.db.get_curriculum,
methods=["GET"]
        )
        self.app.add_api_route(
            '/schedule', self.add_lesson_to_schedule,
methods=["POST"]
        )
        self.app.add_api_route(
            '/schedule', self.db.get_schedule, methods=["GET"]
        )
        self.app.add_api_route(
            '/schedule', self.edit_schedule_cell,
methods=["PUT"]
        )
        self.app.add_api_route(
            '/schedule', self.remove_schedule_cell,
methods=["DELETE"]
        )
        self.app.add_api_route('/university_data',
self.db.get_university_data)
        self.app.add_api_route(
            '/auto_create_schedule', self.db.auto_schedule,
methods=["POST"]
        )
        self.app.add_api_route(
            '/find_collisions', self.db.find_collisions,
methods=["POST"]
```

```
        )
```

Данный код вызывается при создании класса WebApp. В качестве аргументов принимает объект базы данных (на практике это SQLDatabase) и кортеж из двух элементов: хост и порт для запуска web-приложения. По умолчанию это «0.0.0.0» и 8000. IP-адрес «0.0.0.0» – это немаршрутизируемый адрес IPv4 [18], который позволяет обращаться к приложению как из локальной сети, так и из глобальной.

Далее код создает экземпляр класса FastAPI и разрешает подключение к web-приложению из клиенткой части, запущенной по адресу «http://localhost:5173».

Также код регистрирует все методы API и их функции-обработчики. Для методов получения данных установлен тип HTTP-запроса GET, для методов добавления новых данных, составления расписания и поиска коллизий – POST, для модификации данных – PUT, для удаления данных – DELETE [19].

Для запуска web-приложения используется метод serve класса WebApp. Код метода приведен в листинге 15.

Листинг 15 – Метод serve класса WebApp

```
def serve(self) -> None:
        '''Начинает обслуживание сервера-API'''
        host, port = self.listen_params
        uvicorn.run(self.app, host=host, port=port)
```

Данный код запускает веб-приложение с помощью ASGI-сервера (Asynchronous Server Gateway Interface) Uvicorn, что позволяет эффективно обрабатывать запросы к серверу в асинхронной Python-программе [20].

4 Оценка качества программных средств

4.1 Мероприятия по оценке качества программных средств

Тестирование является важным этапом разработки программного обеспечения, которое позволяет убедиться в корректности работы приложения и соответствии его функциональным требованиям. Основные способы тестирования включают функциональное, интеграционное, системное, регрессионное и тестирование пользовательского интерфейса.

Функциональное тестирование: направлено на проверку того, что приложение выполняет свои функции в соответствии с требованиями. Оно проверяет, как система работает с точки зрения пользователя.

Интеграционное тестирование: проверяет взаимодействие между различными модулями или компонентами системы. Оно направлено на выявление проблем, которые могут возникнуть при интеграции отдельных частей системы.

Системное тестирование: проверяет работу системы в целом, включая взаимодействие с внешними системами (например, базами данных, API сторонних сервисов). Оно направлено на проверку соответствия системы требованиям.

Тестирование пользовательского интерфейса (UI-тестирование): проверяет корректность отображения и взаимодействия с элементами интерфейса. Оно направлено на обеспечение удобства использования системы.

Тестирование производительности: проверяет, как система ведет себя под нагрузкой. Оно включает проверку скорости работы, стабильности и масштабируемости системы.

Регрессионное тестирование: проверяет, что новые изменения в коде не повлияли на существующую функциональность. Оно направлено на выявление регрессий (ошибок, которые появляются после внесения изменений). [21]

Проверка работоспособности программы была осуществлена методом функционального тестирования. Такой выбор обусловлен тем, что к ПО были предъявлены в основном функциональные требования, выполнение которых нужно проверить.

4.2 Тестирование приложения

Было проведено функциональное тестирование всех возможностей приложения. Тестирование проводилось в клиентской части приложения, так как каждая функция из клиентской части соответствует методу из серверной части приложения. Таким образом, тестирование охватило и клиентскую, и серверную части приложения составления расписания занятий ВУЗа. Для основных вариантов использования были выполнены функциональные тесты, представленные в таблице 11.

Таблица 11 – Функциональные тесты приложения

№	Назначение теста	Входные данные	Результат теста
1	Добавление университета	Название: Новый университет	Тест пройден успешно, университет создан
2	Добавление факультета	Название: Новый факультет Университет: Новый университет	Тест пройден успешно, факультет создан
3	Добавление кафедры	Название: Новая кафедра Университет: Новый факультет	Тест пройден успешно, кафедра создана
4	Добавление специальности	Название: Новая специальность Родительская кафедра: Новая кафедра	Тест пройден успешно, специальность создана
5	Добавление группы	Название: НГ1 Специальность: Новая специальность	Тест пройден успешно, группа создана
6	Добавление преподавателя	Кафедра: Новая кафедра ФИО: Иванов И.И.	Тест пройден успешно, преподаватель создан

Продолжение таблицы 11

7	Добавление аудитории	Факультет: Новый факультета Кафедра: Новая кафедра Номер: «107-1»	Тест пройден успешно, аудитория создана
8	Добавление потока	Группы: НГ1	Тест пройден успешно, поток создан
9	Добавление предмета	Название: Физика	Тест пройден успешно, предмет создан
10	Добавление занятия в учебный план	Предмет: Физика Преподаватель: Иванов И.И. Второй преподаватель: отсутствует Группа: НГ1 Поток: отсутствует Число часов: 144	Тест пройден успешно, занятие в учебном плане создано
11	Добавление ячейки расписания	Время: 1 неделя, понедельник, 1 пара Группа: НГ1 Занятие из учебного плана: Физика Аудитория: 107-1 Тип занятия: лекционное	Тест пройден успешно, занятие в расписании создано
12	Изменение ячейки расписания	Время: 1 неделя, понедельник, 1 пара Группа: НГ1 Тип занятия: изменено на лабораторное	Тест пройден успешно, занятие в расписании изменено
13	Удаление ячейки расписания	Время: 1 неделя, понедельник, 1 пара Группа: НГ1	Тест пройден успешно, занятие в расписании удалено
14	Автоматическое построение расписания	Исходные данные сгенерированы функцией create_test_data класса SQLDatabase	Тест пройден успешно, в расписание добавлены все занятия из учебного плана
15	Автоматический поиск коллизий	Исходные данные сформированы после выполнения теста 14	Тест пройден успешно. Ошибок не найдено. Найдено одно «окно» у группы и одно – у преподавателя.
16	Получение и отображение данных	Приложение было перезагружено	Тест пройден успешно, после перезагрузки все изменения сохранились.

Таким образом, тестирование приложения было проведено успешно, ошибок в работке программы не найдено.

4.3 Оценка метрик кода

Была оценена сложность функции автоматического составления расписания по метрике Холстеда. Данная метрика позволяет количественно оценить сложность программы без анализа кода, только на основе анализа операторов и операндов. Ключевыми показателями при расчете метрики являются число уникальных операторов и операндов, а также их общее количество. Таблица операторов метода автоматического составления расписания представлена в таблице 12.

Таблица 12 – Операторы метода составления расписания

Оператор	Число вхождений
= (присваивание)	35
. (доступ)	55
(	70
)	70
:	40
,	65
[	25
]	25
If	12
For	15
In	22
Def	1
->	1
With	1
As	1
Not	6
And	3
Or	1
Is	2
continue	1
break	1
return	1
{	1
}	1

Общее число операторов равно 418, из них уникальных – 24. В таблице 13 представлены операнды метода автоматического составления расписания.

Таблица 13 – Операнды метода составления расписания

Операнд	Число вхождений
Self	20
session	18
lesson	16
curriculum	22
group_ids	10
Curr	28
Week	18
Day	12
Pair	12
lesson_type	10
room_id	8
Lesson	9
group	8
classroom	5
None	5
1	15
2	12
72	1
108	1
144	1
LessonType.LECTURE	4
LessonType.LAB	4
Строки (сообщения)	2

Общее число операндов равно 300, из них уникальных – 23. Длина N равна сумме числа операндов и операторов:

$$N = 418 + 300 = 718.$$

Словарь n рассчитывается аналогично длине N, как сумма уникальных операндов и операторов:

$$n = 24 + 23 = 47.$$

Объем кода V рассчитывается по формуле:

$$V = N * log_n(n) \approx 3985.$$

Высокое значение объема позволяет считать, что функция составления расписания занятий переусложнена. Упростить алгоритм не представляется возможным (в ВКР был выбран один из самых простых алгоритмов составления расписания), но при усовершенствовании ПО имеет смысл разбить функцию составления расписания на несколько отдельных методов.

Заключение

Целью выпускной квалификационной работы была разработка автоматизированной системы составления расписания занятий ВУЗа. Цель выполнена полностью, созданный при выполнении ВКР прототип приложения отвечает всем заявленным требованиям:

- имеет простой, интуитивно понятный интерфейс;
- позволяет просматривать и добавлять первичные данные;
- позволяет просматривать и изменять расписание в ручном режиме;
- позволяет автоматически строить расписание занятий;
- позволяет автоматически находить ошибки и недочеты в расписании;
- имеет полную документацию всех функций и методов API.

Главными достоинствами созданного в результате выполнения ВКР прототипа системы являются простота и лаконичность интерфейса, что значительно упрощает работу специалиста отдела планирования даже в ручном режиме. В дальнейшем прототип будет дополняться, в него будут внедрены более совершенные алгоритмы составления и корректировки расписания, добавлены подсказки для специалистов отдела планирования.

Список использованных источников

1. Аганина, Д.А. Проблемы автоматизированного расписания образовательного процесса [Текст] / Д.А. Аганина // Молодой ученый. – 2018. - № 42 (228). – С. 42-43.

2. Инструкция для составления расписания учебных занятий и экзаменационных сессий в ФГБОУ ВО Орловский ГАУ. 2017. [Электронный ресурс]. URL: https://www.orelsau.ru/upload/iblock/968/9687c56767b148ea44cce48d59cab7c5.pdf (дата обращения: 29.04.2025).

3. Автоматизированное составление расписания | Университет. [Электронный ресурс]. – URL: https://largenumbers.ru/software/1c-avtomatizirovannoe-sostavlenie-raspisaniya-universitet (дата обращения: 29.04.2025).

4. Инструкция по составлению расписания учебных занятий, промежуточной аттестации по дисциплинам (модулям) в автоматизированной системе «Расписание учебных занятий» СПбПУ. [Электронный ресурс]. – URL: https://open.spbstu.ru/wp-content/uploads/2020/04/instruction_schedule.pdf (дата обращения: 01.06.2025).

5. Губенко И. О. Авторасписание. [Электронный ресурс]. – URL: https://www.mmis.ru/programs/avtor (дата обращения: 01.06.2025).

6. Как диаграммы Ганта упрощают работу с проектами [Электронный ресурс]. URL: https://habr.com/ru/company/hygger/blog/415271/ (дата обращения 01.06.2025).

7. ORM – Википедия. [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/ORM (дата обращения 02.06.2025)

8. Уровни моделирования ИС [Электронный ресурс]. URL: https://dl.nure.ua/pluginfile.php/550/mod_resource/content/2/content/content4.html (дата обращения 02.06.2025).

9. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения: пер. с англ. / Р. Мартин. – СПб.: Питер, 2018. – 352 с.

10. FastAPI | Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/FastAPI (дата обращения 02.06.2025).

11. Что такое __init__ в Python? Онлайн справочник на itProger [Электронный ресурс]. URL: https://itproger.com/spravka/python/init_ (дата обращения 22.06.2025).

12. Дакетт, Дж. HTML и CSS. Разработка и создание веб-сайтов: пер. с англ. / Дж. Дакетт. — М.: Эксмо, 2021. – 480 с.

13. Петросян А. С., Сурмачевская А. А., Сафонова Е. А., Кузнецова Э. М., Гареева Г. А. Автоматизация процесса составления учебного расписания // IJAS. 2023. №4. (дата обращения: 22.06.2025).

14. Составление расписаний с ресурсными ограничениями – Искусственный интеллект [Электронный ресурс]. URL: http://www.rriai.org.ru/sostavlenie-raspisaniy-s-resursnyimi-ogranicheniyami-3.html (дата обращения 23.06.2025).

15. Воробович Н. П., Лопатеева О. Н. О NP-полноте задач формирования расписания в вузе // Вестник КрасГАУ. 2006. №11. (дата обращения: 23.06.2025).

16. Клеванский Н.Н. Алгоритмы форммирования расписания занятий высших учебных заведений // Фундаментальные исследования. 2017. № 10 (часть 3) С. 454-458. (дата обращения: 23.06.2025).

17. Что такое: Эвристическая оценка: подробный обзор [Электронный ресурс]. URL: https://ru.statisticseasily.com/glossario/what-is-heuristic-evaluation-detailed-overview/ (дата обращения 23.06.2025).

18. Ликбез 19. Что такое IP-адрес 0.0.0.0. | Telecom & IT [Электронный ресурс]. URL: https://shalaginov.com/2021/09/21/0000-ip-address/ (дата обращения 23.06.2025).

19. Типы HTTP-запросов и философия REST / Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/50147/ (дата обращения 23.06.2025).

20. Введение в ASGI: становление асинхронной веб-экосистемы Python / Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/482936/ (дата обращения 23.06.2025).

21. Куликов С. Тестирование программного обеспечения. Базовый курс / С. Куликов. — 3-е изд. — Минск: Четыре четверти, 2020. — 312 с. (дата обращения 23.06.2025).

Приложение А

Глоссарий

(справочное)

СУБД – система управления базами данных.

REST – Representation State Transfer (передача репрезентативного состояния), архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

API – Application Programming Interface (интерфейс программирования приложений), совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

SQLite – реляционная система управления базами данных.

FastAPI – библиотека для языка программирования Python, предназначенная для создания веб-приложений с REST API.

SQLModel – библиотека для языка программирования Python, предназначенная для создания отображения объектов базы данных в виде классов языка Python.

SQL – язык программирования, применяемый для управления базами данных.

JSON – JavaScript Object Notation, текстовый формат, полностью независимый от языка реализации.

UML – Unified Modeling Language (унифицированный язык моделирования).

Приложение Б
Листинг программы
(обязательное)

Листинг Б.1 – Файл backend/main.py

```
# Основной модуль приложения

import logging

from web import WebApp
from db.sql_db import get_database


LISTEN_HOST = '0.0.0.0'
DEFAULT_LISTEN_PORT = 8000


logging.basicConfig(
    level=logging.INFO, filename='log.log', filemode='a',
    format='%(asctime)s %(levelname)s %(message)s'
)


def main() -> int:
    '''Метод для запуска веб-API'''
    db = get_database('sqlite:///test_schedule.sqlite')
    #db.create_test_data()
    web_app = WebApp(db, (LISTEN_HOST, DEFAULT_LISTEN_PORT))
    web_app.serve()

if __name__ == '__main__':
    main()
```

Листинг Б.2 – Файл backend/web.py

```
#
#
#

'''Модуль отвечает за определение методов веб-API'''

from typing import Tuple, Optional

import uvicorn

from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware

from db.main_db import Database, CourseEnum, LessonType


ListenParams = Tuple[str, int]


class WebApp:
    '''Класс для настройки веб-API'''
```

```
    def __init__(self, db: Database, listen_params:
ListenParams) -> None:
        self.db = db
        self.listen_params = listen_params
        self.app = FastAPI()

        origins = [
            'http://localhost:5173',
            'localhost:5173'
        ]

        self.app.add_middleware(
            CORSMiddleware,
            allow_origins=origins,
            allow_credentials=True,
            allow_methods=['*'],
            allow_headers=['*']
        )

        self.app.add_api_route(
            '/structural_divizion',
            self.add_structural_divizion,
            methods=["POST"]
        )
        self.app.add_api_route(
            '/speciality', self.add_speciality, methods=["POST"]
        )
        self.app.add_api_route('/group', self.add_group,
methods=["POST"])
        self.app.add_api_route('/teacher', self.add_teacher,
methods=["POST"])
        self.app.add_api_route(
            '/classroom', self.add_classroom, methods=["POST"]
        )
        self.app.add_api_route('/subject', self.add_subject,
methods=["POST"])
        self.app.add_api_route(
            '/subject', self.db.get_subjects, methods=["GET"]
        )
        self.app.add_api_route('/flow', self.add_flow,
methods=["POST"])
        self.app.add_api_route('/flow', self.db.get_flows,
methods=["GET"])
        self.app.add_api_route(
            '/curriculum', self.add_lesson_to_plan,
methods=["POST"]
        )
        self.app.add_api_route(
            '/curriculum', self.db.get_curriculum,
methods=["GET"]
        )
        self.app.add_api_route(
```

```
            '/schedule', self.add_lesson_to_schedule,
methods=["POST"]
        )
        self.app.add_api_route(
            '/schedule', self.db.get_schedule, methods=["GET"]
        )
        self.app.add_api_route(
            '/schedule', self.edit_schedule_cell,
methods=["PUT"]
        )
        self.app.add_api_route(
            '/schedule', self.remove_schedule_cell,
methods=["DELETE"]
        )
        self.app.add_api_route('/university_data',
self.db.get_university_data)
        self.app.add_api_route(
            '/auto_create_schedule', self.db.auto_schedule,
methods=["POST"]
        )
        self.app.add_api_route(
            '/find_collisions', self.db.find_collisions,
methods=["POST"]
        )

    def add_structural_divizion(
        self,
        name: str,
        short_name: Optional[str] = None,
        parent_id: Optional[int] = None
    ) -> int:
        '''Добавляет структурное подразделение. Если
родительского подразделения
        не существует или подразделеие с таким названием уже
есть -
        возвращает ошибку 404'''
        try:
            return self.db.add_structural_divizion(parent_id,
name, short_name)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_speciality(
        self,
        department_id: int,
        name: str
    ) -> int:
        '''Добавляет специальность. Если родительского
подразделения
        не существует или специальность с таким названием уже
есть -
        возвращает ошибку 404'''
```

```
        try:
            return self.db.add_speciality(department_id, name)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_group(
        self,
        speciality_id: int,
        name: str,
        course: CourseEnum,
        student_count: int
    ) -> int:
        '''Добавляет группу. Если специальности не существует
или группа с таким
        именем уже есть - возвращает ошибку 404'''
        try:
            return self.db.add_group(speciality_id, name,
course, student_count)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_teacher(
        self,
        department_id: int,
        name: str
    ) -> int:
        '''Добавляет преподавателя. Если родительской кафедры не
существует или
        преподаватель с таким ФИО уже есть - возвращает ошибку
404'''
        try:
            return self.db.add_teacher(department_id, name)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_classroom(
        self,
        faculty_id: int,
        name: str,
        capacity: int,
        department_id: Optional[int] = None
    ) -> int:
        '''Добавляет аудиторию. Если родительского факультета,
кафедры не
        существует или аудитория с таким номером уже есть -
        возвращает ошибку 404'''
        try:
            return self.db.add_classroom(
                faculty_id, department_id, name, capacity
            )
```

```
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_subject(
        self,
        name: str,
        short_name: str
    ) -> int:
        '''Добавляет предмет. Если предмет с таким названием уже
есть -
        возвращает ошибку 404. Сокращённое название может быть
неуникально'''
        try:
            return self.db.add_subject(name, short_name)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_flow(self, groups: list[str]) -> int:
        '''Добавляет поток (список групп). Если в потоке есть
несуществующие
        группы - возвращает ошибку 404'''
        try:
            return self.db.add_flow(groups)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_lesson_to_plan(
        self,
        subject_id: int,
        hours: int,
        primary_teacher_id: int,
        secondary_teacher_id: Optional[int] = None,
        group_id: Optional[int] = None,
        flow_id: Optional[int] = None
    ) -> int:
        '''Добавляет занятие в учебный план.
        Возвращает ошибку 404 если:

        - Предмета, одного из преподавателей, группы или потока
нет в БД

        - Указаны и группа, и поток, либо не указано ни то, ни
другое

        - У группы/потока уже есть такой предмет'''
        try:
            return self.db.add_lesson_to_plan(
                subject_id, hours, primary_teacher_id,
                secondary_teacher_id, group_id, flow_id
            )
```

```
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def add_lesson_to_schedule(
        self,
        week: int,
        day: int,
        pair: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> int:
        '''Добавляет ячейку расписания. Если аудитории или
занятия из учебного
        плана не существует либо параметр не корректен -
возвращает ошибку 404.

        Параметры:

        - week (неделя) - 1 либо 2

        - day (день) - от 1 до 6 (понедельник - 1, суббота - 6)

        - pair (номер пары) - от 1 до 8'''
        try:
            return self.db.add_lesson_to_schedule(
                week, day, pair, classroom_id, curriculum_id,
lesson_type
            )
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def edit_schedule_cell(
        self,
        id: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> None:
        '''Изменяет ячейку расписания с заданным id'''
        try:
            return self.db.edit_schedule_cell(
                id, classroom_id, curriculum_id, lesson_type
            )
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def remove_schedule_cell(self, id: int) -> None:
        '''Удаляет ячейку расписания с заданным id'''
        try:
```

```
            return self.db.remove_schedule_cell(id)
        except ValueError as exc:
            raise HTTPException(status_code=404,
detail=exc.message) from exc

    def serve(self) -> None:
        '''Начинает обслуживание сервера-API'''
        host, port = self.listen_params
        uvicorn.run(self.app, host=host, port=port)
```

Листинг Б.3 – Файл backend/db/main_db.py

```
#
#
#

'''Модуль определяет абстракцию для взаимодействия с любыми
БД'''

import abc
from enum import Enum
from typing import Optional

from pydantic import BaseModel


class CourseEnum(str, Enum):
    """Возможные значения курса"""
    BACHELOR_1 = "Бакалавриат, 1"
    BACHELOR_2 = "Бакалавриат, 2"
    BACHELOR_3 = "Бакалавриат, 3"
    BACHELOR_4 = "Бакалавриат, 4"
    MASTER_1 = "Магистратура, 1"
    MASTER_2 = "Магистратура, 2"
    SPECIALIST_1 = "Специалитет, 1"
    SPECIALIST_2 = "Специалитет, 2"
    SPECIALIST_3 = "Специалитет, 3"
    SPECIALIST_4 = "Специалитет, 4"
    SPECIALIST_5 = "Специалитет, 5"


class LessonType(str, Enum):
    """Возможные типы занятий"""
    LECTURE = "лекционное"
    LAB = "лабораторное"


class GroupData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные о группах'''
    id: int
    name: str
    course: CourseEnum
    students_count: int
```

```
class SpecialityData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные о специальностях'''
    id: int
    name: str
    groups: list[GroupData]


class LecturerData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные о преподавателях'''
    id: int
    full_name: str


class ClassroomData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные об аудиториях'''
    id: int
    number: str
    capacity: int


class DepartmentData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные о кафедрах'''
    id: int
    name: str
    short_name: str
    specialities: list[SpecialityData]
    lecturers: list[LecturerData]
    classrooms: list[ClassroomData]


class FacultyData(BaseModel):
    '''Вспомогательный тип данных для структуры университета
    для get_university_data. Содержит данные о факультетах.
    departments - кафедры
    classrooms - аудитории факультета, не относящиеся ни к одной
кафедре'''
    id: int
    name: str
    departments: list[DepartmentData]
    classrooms: list[ClassroomData]


class UniversityData(BaseModel):
    '''Тип данных структуры университета для функции
get_university_data'''
    id: int
    name: str
```

```
    faculties: list[FacultyData]


class SubjectsData(BaseModel):
    '''Данные о предметах для метода get_subjects'''
    id: int
    name: str
    short_name: str


class FlowsData(BaseModel):
    '''Данные о потоках для метода get_flows. groups - список
названий групп'''
    id: int
    groups: list[str]


class CurriculumData(BaseModel):
    '''Данные об учебном плане для метода get_curriculum.
    subject - сокращенное название предмета
    groups - названия групп, у которых проводится занятие. Если
в БД указана
    группа, то указывается список из одной этой группы. Если
указан поток,
    то список всех групп потока
    primary_teacher - ФИО основного преподавателя
    secondary_teacher - ФИО второго преподавателя (если он
есть)'''
    id: int
    subject: str
    groups: list[str]
    hours: int
    primary_teacher: str
    secondary_teacher: Optional[str]


class ScheduleCellData(BaseModel):
    '''Данные о ячейке расписания'''
    id: int
    lesson_type: LessonType
    subject: str    # Название предмета
    teachers: str   # Преподватели
    classroom: str  # Номер аудитории


class ScheduleData(BaseModel):
    '''Данные о расписании'''
    data: dict[
        int,  # Уровень 0: Неделя (1 либо 2)
        dict[
            int,  # Уровень 1: День недели (1-6, понедельник-
суббота)
            dict[
```

```
                int,  # Уровень 3: номер пары (1-8)
                dict[
                    str,  # Уровень 4: Группа (название)
                    ScheduleCellData
                ]
            ]
        ]
    ]


class Database(abc.ABC):
    '''Абстрактный класс для взаимодействия с любыми БД'''
    @abc.abstractmethod
    def add_structural_divizion(
        self,
        parent_id: Optional[int],
        name: str
    ) -> int:
        '''Добавляет структурное подразделение (университет,
факультет, кафедру)
        Возвращает id добавленной записи'''

    @abc.abstractmethod
    def add_speciality(
        self,
        department_id: int,
        name: str
    ) -> int:
        '''Добавляет специальность, возвращает id добавленной
записи'''

    @abc.abstractmethod
    def add_group(
        self,
        speciality_id: int,
        name: str,
        course: CourseEnum,
        student_count: int
    ) -> int:
        '''Добавляет группу, возвращает id добавленной записи'''

    @abc.abstractmethod
    def add_teacher(
        self,
        department_id: int,
        name: str
    ) -> int:
        '''Добавляет преподавателя, возвращает id добавленной
записи'''

    @abc.abstractmethod
    def add_classroom(
        self,
```

```
        faculty_id: int,
        department_id: Optional[int],
        name: str,
        capacity: int
    ) -> int:
        '''Добавляет аудиторию, возвращает id добавленной
записи'''

    @abc.abstractmethod
    def add_subject(
        self,
        name: str,
        short_name: str
    ) -> int:
        '''Добавляет аудиторию, возвращает id добавленной
записи'''

    @abc.abstractmethod
    def add_flow(self, groups: list[str]) -> int:
        '''Добавляет поток, возвращает id добавленной записи'''

    @abc.abstractmethod
    def add_lesson_to_plan(
        self,
        subject_id: int,
        hours: int,
        primary_teacher_id: int,
        secondary_teacher_id: int,
        group_id: Optional[int],
        flow_id: Optional[int]
    ) -> int:
        '''Добавляет занятие в учебный план, возвращает id
добавленной записи'''

    @abc.abstractmethod
    def add_lesson_to_schedule(
        self,
        week: int,
        day: int,
        pair: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> int:
        '''Добавляет занятие в расписание, возвращает id
добавленной записи'''

    @abc.abstractmethod
    def get_university_data(self) -> list[UniversityData]:
        '''Возвращает данные о структуре университета, которые
выключат в себя:
        - Университеты
        - Факультеты
```

```
        - Кафедры
        - Специальности
        - Группы
        - Аудитории кафедр
        - Аудитории факультетов (те, у которых не указан
параметр кафедры)
        - Преподаватели'''

    @abc.abstractmethod
    def get_subjects(self) -> list[SubjectsData]:
        '''Возвращает данные о предметах'''

    @abc.abstractmethod
    def get_flows(self) -> list[FlowsData]:
        '''Возвращает данные о потоках (списках групп)'''

    @abc.abstractmethod
    def get_curriculum(self) -> list[CurriculumData]:
        '''Возвращает учебный план'''

    @abc.abstractmethod
    def get_schedule(self) -> ScheduleData:
        '''Возвращает расписание занятий'''

    @abc.abstractmethod
    def edit_schedule_cell(
        self,
        id: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> None:
        '''Изменяет ячейку расписания занятий'''

    @abc.abstractmethod
    def remove_schedule_cell(self, id: int) -> None:
        '''Удаляет ячейку расписания занятий'''

    @abc.abstractmethod
    def auto_schedule(self) -> None:
        '''Автоматическое составление расписания с учётом
семестровой нагрузки'''

    @abc.abstractmethod
    def find_collisions(self) -> dict:
        '''Поиск коллизий и окон в расписании'''
```

Листинг Б.4 – Файл backend/db/sql_db.py

```
#
#
#

'''Модуль определяет взаимодействие с реляционными БД'''
```

```
from typing import List, Optional
from collections import defaultdict

from sqlmodel import (
  Field, SQLModel, create_engine, Session, select, Relationship,
CheckConstraint
)
from sqlalchemy import func
from sqlalchemy.orm import aliased

from db.main_db import (
    Database, LessonType, CourseEnum, UniversityData,
DepartmentData, GroupData,
    FacultyData, LecturerData, ClassroomData, SpecialityData,
SubjectsData,
    FlowsData, CurriculumData, ScheduleData, ScheduleCellData
)


class Department(SQLModel, table=True):
    """Структурное подразделение
    Подразлеления без родителя - университет
    Дочерние университету подразделения - факультеты
    Дочерние факультетам - кафедры"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    short_name: Optional[str]
    parent_id: Optional[int] = Field(default=None,
foreign_key="department.id")

    # Связь на самого себя для иерархии
    parent: Optional["Department"] = Relationship(
        back_populates="children",
        sa_relationship_kwargs={
            "remote_side": "Department.id",
            "foreign_keys": "Department.parent_id"
        }
    )
    children: List["Department"] =
Relationship(back_populates="parent")


class Specialty(SQLModel, table=True):
    """Специальность"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    department_id: int = Field(foreign_key="department.id")

    department: Department = Relationship()


class FlowGroupLink(SQLModel, table=True):
```

```
    """Связь многие-ко-многим между Потоком и Группами"""
    flow_id: Optional[int] = Field(default=None,
foreign_key="flow.id", primary_key=True)
    group_id: Optional[int] = Field(default=None,
foreign_key="group.id", primary_key=True)


class Group(SQLModel, table=True):
    """Группа"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    course: CourseEnum
    specialty_id: int = Field(foreign_key="specialty.id")
    student_count: int

    specialty: Specialty = Relationship()
    # Обратная связь для связи многие-ко-многим с Flow
    flows: List["Flow"] = Relationship(back_populates="groups",
link_model=FlowGroupLink)


class Flow(SQLModel, table=True):
    """Поток"""
    id: Optional[int] = Field(default=None, primary_key=True)

    groups: List[Group] = Relationship(back_populates="flows",
link_model=FlowGroupLink)


class Classroom(SQLModel, table=True):
    """Аудитория"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    capacity: int
    faculty_id: int = Field(foreign_key="department.id")
    department_id: Optional[int] =
Field(foreign_key="department.id")

    faculty: Department = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Classroom.faculty_id"}
    )
    department: Optional[Department] = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Classroom.department_id"}
    )


class Subject(SQLModel, table=True):
    """Предмет"""
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str = Field(index=True, unique=True)
    short_name: str
```

```
class Teacher(SQLModel, table=True):
    """Преподаватель"""
    id: Optional[int] = Field(default=None, primary_key=True)
    full_name: str = Field(index=True, unique=True)
    department_id: int = Field(foreign_key="department.id")

    department: Department = Relationship()


class Curriculum(SQLModel, table=True):
    """Учебный план"""
    __table_args__ = (
        CheckConstraint("hours IN (72, 108, 144)",
name="hours_check"),
    )

    id: Optional[int] = Field(default=None, primary_key=True)
    subject_id: int = Field(foreign_key="subject.id")
    hours: int
    primary_teacher_id: int = Field(foreign_key="teacher.id")
    secondary_teacher_id: Optional[int] =
Field(foreign_key="teacher.id")
    group_id: Optional[int] = Field(default=None,
foreign_key="group.id")
    flow_id: Optional[int] = Field(default=None,
foreign_key="flow.id")

    subject: Subject = Relationship()
    primary_teacher: Teacher = Relationship(
        sa_relationship_kwargs={"foreign_keys":
"Curriculum.primary_teacher_id"}
    )
    secondary_teacher: Optional[Teacher] = Relationship(
        sa_relationship_kwargs={
            "foreign_keys": "Curriculum.secondary_teacher_id"
        }
    )
    group: Optional[Group] = Relationship()
    flow: Optional[Flow] = Relationship()


class Lesson(SQLModel, table=True):
    """Занятие в расписании"""
    __table_args__ = (
        CheckConstraint("week IN (1, 2)", name="week_check"),
        CheckConstraint("day BETWEEN 1 AND 6",
name="day_check"),
        CheckConstraint("pair BETWEEN 1 AND 8",
name="pair_check"),
    )
```

```
    id: Optional[int] = Field(default=None, primary_key=True)
    week: int
    day: int
    pair: int
    classroom_id: int = Field(foreign_key="classroom.id")
    curriculum_id: int = Field(foreign_key="curriculum.id")
    lesson_type: LessonType

    classroom: Classroom = Relationship()
    curriculum: Curriculum = Relationship()


class SQLDatabase(Database):
    '''Класс для работы с реляционными БД через ORM'''
    def __init__(self, db_url: str) -> None:
        self.engine = create_engine(db_url)
        SQLModel.metadata.create_all(self.engine)

    def add_structural_divizion(
        self,
        parent_id: Optional[int],
        name: str,
        short_name: Optional[str]
    ) -> int:
        new_division = Department(
            name=name, parent_id=parent_id,
short_name=short_name
        )
        with Session(self.engine) as session:
            # Проверка существования родительского подразделения
            if parent_id is not None:
                parent = session.get(Department, parent_id)
                if not parent:
                    raise ValueError(
                        f"Родительское подразделение с ID
{parent_id} не найдено"
                    )
            # Проверка уникальности названия
            existing = session.exec(
                select(Department).where(Department.name ==
name)
            ).first()
            if existing:
                raise ValueError(
                    f"Подразделение с именем '{name}' уже
существует"
                )

            session.add(new_division)
            session.commit()
            session.refresh(new_division)
        return new_division.id
```

```
    def add_speciality(
        self,
        department_id: int,
        name: str
    ) -> int:
        new_speciality = Specialty(name=name,
department_id=department_id)
        with Session(self.engine) as session:
            # Проверка существования "родительской" кафедры
            parent = session.get(Department, department_id)
            if not parent:
                raise ValueError(f"Кафедра с ID {department_id}
не найдена")
            # Проверка уникальности названия
            existing = session.exec(
                select(Specialty).where(Specialty.name == name)
            ).first()
            if existing:
                raise ValueError(f"Специальность '{name}' уже
существует")

            session.add(new_speciality)
            session.commit()
            session.refresh(new_speciality)
        return new_speciality.id

    def add_group(
        self,
        speciality_id: int,
        name: str,
        course: CourseEnum,
        student_count: int
    ) -> int:
        new_group = Group(
            name=name,
            course=course,
            specialty_id=speciality_id,
            student_count=student_count
        )
        with Session(self.engine) as session:
            # Проверка существования "родительской"
специальности
            parent = session.get(Specialty, speciality_id)
            if not parent:
                raise ValueError(f"Специальность с ID
{speciality_id} не найдена")
            # Проверка уникальности названия
            existing = session.exec(
                select(Group).where(Group.name == name)
            ).first()
            if existing:
                raise ValueError(f"Группа '{name}' уже
существует")
```

```
            # Сюда можно добавить проверку наличия в группе
минимально необходимого числа студентов
            session.add(new_group)
            session.commit()
            session.refresh(new_group)
        return new_group.id

    def add_teacher(
        self,
        department_id: int,
        name: str
    ) -> int:
        new_teacher = Teacher(full_name=name,
department_id=department_id)
        with Session(self.engine) as session:
            # Проверка существования "родительской" кафедры
            parent = session.get(Department, department_id)
            if not parent:
                raise ValueError(f"Кафедра с ID {department_id}
не найдена")
            # Проверка уникальности ФИО
            existing = session.exec(
                select(Teacher).where(Teacher.full_name == name)
            ).first()
            if existing:
                raise ValueError(f"Преподаватель '{name}' уже
существует")

            session.add(new_teacher)
            session.commit()
            session.refresh(new_teacher)
        return new_teacher.id

    def add_classroom(
        self,
        faculty_id: int,
        department_id: Optional[int],
        name: str,
        capacity: int
    ) -> int:
        new_classroom = Classroom(
            name=name,
            capacity=capacity,
            faculty_id=faculty_id,
            department_id=department_id
        )
        with Session(self.engine) as session:
            # Проверка существования "родительского" факультета
            parent_faculty = session.get(Department, faculty_id)
            if not parent_faculty:
                raise ValueError(f"Факультет с ID {faculty_id}
не найден")
            # Проверка существования "родительской" кафедры
```

```
            if department_id:
                parent_department = session.get(Department,
department_id)
                if not parent_department:
                    raise ValueError(f"Кафедра с ID
{department_id} не найдена")
            # Проверка уникальности номера аудитории
            existing = session.exec(
                select(Classroom).where(Classroom.name == name)
            ).first()
            if existing:
                raise ValueError(f"Аудитория '{name}' уже
существует")

            session.add(new_classroom)
            session.commit()
            session.refresh(new_classroom)
        return new_classroom.id

    def add_subject(
        self,
        name: str,
        short_name: str
    ) -> int:
        new_subject = Subject(name=name, short_name=short_name)
        with Session(self.engine) as session:
            # Проверка уникальности полного названия
            existing = session.exec(
                select(Subject).where(Subject.name == name)
            ).first()
            if existing:
                raise ValueError(f"Предмет '{name}' уже
существует")

            session.add(new_subject)
            session.commit()
            session.refresh(new_subject)
        return new_subject.id

    def add_flow(self, groups: list[str]) -> int:
        with Session(self.engine) as session:
            # Проверка существования всех групп
            found_groups = session.exec(
                select(Group).where(Group.name.in_(groups))
            ).all()
            found_names = {g.name for g in found_groups}
            missing_groups = [name for name in groups if name
not in found_names]
            if missing_groups:
                raise ValueError(f"Группы {',
'.join(missing_groups)} не найдены")

            new_flow = Flow()
```

```
            session.add(new_flow)
            session.flush() # Для получения ID потока
            # Создание связи между потоком и группами
            for group in found_groups:
                session.add(FlowGroupLink(flow_id=new_flow.id,
group_id=group.id))
            session.commit()
            return new_flow.id

    def add_lesson_to_plan(
        self,
        subject_id: int,
        hours: int,
        primary_teacher_id: int,
        secondary_teacher_id: Optional[int],
        group_id: Optional[int],
        flow_id: Optional[int]
    ) -> int:
        new_lesson = Curriculum(
            subject_id=subject_id,
            hours=hours,
            primary_teacher_id=primary_teacher_id,
            secondary_teacher_id=secondary_teacher_id,
            group_id=group_id,
            flow_id=flow_id
        )
        with Session(self.engine) as session:
            # Проверка существования предмета
            subject = session.get(Subject, subject_id)
            if not subject:
                raise ValueError(f"Предмет с ID {subject_id} не
найден")
            # Проверка существования основного преподавателя
            primary_teacher = session.get(Teacher,
primary_teacher_id)
            if not primary_teacher:
                raise ValueError(f"Преподаватель с ID
{primary_teacher_id} не найден")
            # Проверка существования второго преподавателя
            if secondary_teacher_id:
                secondary_teacher = session.get(Teacher,
secondary_teacher_id)
                if not secondary_teacher:
                    raise ValueError(f"Преподаватель с ID
{secondary_teacher_id} не найден")
            # Проверка существования группы
            if group_id:
                group = session.get(Group, group_id)
                if not group:
                    raise ValueError(f"Группа с ID {group_id} не
найдена")
            # Проверка существования потока
            if flow_id:
```

```
                flow = session.get(Flow, flow_id)
                if not flow:
                    raise ValueError(f"Поток с ID {flow_id} не
найден")
            # Должна быть ЛИБО группа, ЛИБО поток
            if (group_id and flow_id)  or ((not group_id) and
(not flow_id)):
                raise ValueError(f"Укажите ЛИБО группу, ЛИБО
поток")
            # Проверка уникальности предмета и группы/потока
            existing = session.exec(
                select(Curriculum).where(
                    Curriculum.subject_id == subject_id,
                    Curriculum.group_id == group_id,
                    Curriculum.flow_id == flow_id
                )
            ).first()
            if existing:
                raise ValueError(f"Этот предмет у группы/потока
уже есть в учебном плане")

            session.add(new_lesson)
            session.commit()
            session.refresh(new_lesson)
        return new_lesson.id

    def add_lesson_to_schedule(
        self,
        week: int,
        day: int,
        pair: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> int:
        new_lesson = Lesson(
            week=week,
            day=day,
            pair=pair,
            classroom_id=classroom_id,
            curriculum_id=curriculum_id,
            lesson_type=lesson_type
        )
        with Session(self.engine) as session:
            # Проверка существования аудитории
            classroom = session.get(Classroom, classroom_id)
            if not classroom:
                raise ValueError(f"Аудитория с ID {classroom_id}
не найдена")
            # Проверка существования занятия в учебном плане
            lesson_in_curriculum = session.get(Curriculum,
curriculum_id)
            if not lesson_in_curriculum:
```

```
                raise ValueError(f"Занятия с ID {curriculum_id}
нет в учебном плане")
            # Проверка корректности ввода недели/дня/номера пары
            if not week in (1, 2):
                raise ValueError(f"Параметр week может быть 1
либо 2")
            if not 1 <= day <= 6:
                raise ValueError(f"Параметр day должен быть
числом от 1 до 6")
            if not 1 <= pair <= 8:
                raise ValueError(f"Параметр pair должен быть
числом от 1 до 8")

            session.add(new_lesson)
            session.commit()
            session.refresh(new_lesson)
        return new_lesson.id

    def get_university_data(self) -> list[UniversityData]:
        with Session(self.engine) as session:
            # Получение всех подразделений и построение иерархии
ВУЗа
            departments = session.exec(select(Department)).all()
            children_map = defaultdict(list)
            for dept in departments:
                children_map[dept.parent_id].append(dept)

            # Загрузка доп. данных
            specialties = session.exec(select(Specialty)).all()
            groups = session.exec(select(Group)).all()
            teachers = session.exec(select(Teacher)).all()
            classrooms = session.exec(select(Classroom)).all()

            # Группировка доп. данных
            specialties_by_dept = defaultdict(list)
            for spec in specialties:

specialties_by_dept[spec.department_id].append(spec)

            groups_by_specialty = defaultdict(list)
            for group in groups:

groups_by_specialty[group.specialty_id].append(group)

            teachers_by_dept = defaultdict(list)
            for teacher in teachers:

teachers_by_dept[teacher.department_id].append(teacher)

            classrooms_by_dept = defaultdict(list)
            classrooms_by_faculty = defaultdict(list)
            for room in classrooms:
                if room.department_id:
```

```
classrooms_by_dept[room.department_id].append(room)
                else:

classrooms_by_faculty[room.faculty_id].append(room)

            # Создание структуры ВУЗа
            result = []
            for university in children_map[None]:  #
Университеты (без родителя)
                faculties = []
                for faculty in children_map.get(university.id,
[]):  # Факультеты
                    departments_data = []
                    for department in
children_map.get(faculty.id, []):  # Кафедры
                        # Сбор данных для кафедры
                        specs_data = []
                        for spec in
specialties_by_dept.get(department.id, []):
                            groups_data = [
                                GroupData(
                                    id=g.id,
                                    name=g.name,
                                    course=g.course,

students_count=g.student_count
                                ) for g in
groups_by_specialty.get(spec.id, [])
                            ]
                            specs_data.append(
                                SpecialityData(
                                    id=spec.id,
                                    name=spec.name,
                                    groups=groups_data
                                )
                            )

                        departments_data.append(
                            DepartmentData(
                                id=department.id,
                                name=department.name,

short_name=department.short_name,
                                specialities=specs_data,
                                lecturers=[
                                    LecturerData(id=t.id,
full_name=t.full_name)
                                    for t in
teachers_by_dept.get(department.id, [])
                                ],
                                classrooms=[
```

```
                                    ClassroomData(id=r.id,
number=r.name, capacity=r.capacity)
                                    for r in
classrooms_by_dept.get(department.id, [])
                                ]
                            )
                        )

                    faculties.append(
                        FacultyData(
                            id=faculty.id,
                            name=faculty.name,
                            departments=departments_data,
                            classrooms=[
                                ClassroomData(id=r.id,
number=r.name, capacity=r.capacity)
                                for r in
classrooms_by_faculty.get(faculty.id, [])
                                ]
                            )
                        )

                result.append(
                    UniversityData(
                        id=university.id,
                        name=university.name,
                        faculties=faculties
                    )
                )

            return result

    def get_subjects(self) -> list[SubjectsData]:
        with Session(self.engine) as session:
            subjects = session.exec(select(Subject)).all()
            return [
                SubjectsData(
                    id=subject.id,
                    name=subject.name,
                    short_name=subject.short_name
                )
                for subject in subjects
            ]

    def get_flows(self) -> list[FlowsData]:
        with Session(self.engine) as session:
            # Загрузка потоков с группами
            flows = session.exec(
                select(Flow, Group)
                .join(FlowGroupLink, Flow.id ==
FlowGroupLink.flow_id)
                .join(Group, FlowGroupLink.group_id == Group.id)
            ).all()
```

```
            # Группировка групп по потокам
            flow_groups = defaultdict(list)
            for flow, group in flows:
                flow_groups[flow.id].append(group.name)

            return [
                FlowsData(id=flow_id, groups=groups)
                for flow_id, groups in flow_groups.items()
            ]

    def get_curriculum(self) -> list[CurriculumData]:
        with Session(self.engine) as session:
            TeacherPrimary = aliased(Teacher)
            TeacherSecondary = aliased(Teacher)

            # Загрузка учебных планов со связанными данными
            curriculum_query = (
                select(
                    Curriculum,
                    Subject.short_name,
                    TeacherPrimary.full_name,
                    TeacherSecondary.full_name,
                    Group.name,
                    Flow.id
                )
                .join(Subject, Curriculum.subject_id ==
Subject.id)
                .join(TeacherPrimary,
Curriculum.primary_teacher_id == TeacherPrimary.id)
                .join(TeacherSecondary,
Curriculum.secondary_teacher_id == TeacherSecondary.id,
isouter=True)
                .join(Group, Curriculum.group_id == Group.id,
isouter=True)
                .join(Flow, Curriculum.flow_id == Flow.id,
isouter=True)
            )
            results = session.exec(curriculum_query).all()

            # Сбор ID всех потоков
            flow_ids = set()
            for res in results:
                if res[5]:  # Flow.id (индекс 5 в результате)
                    flow_ids.add(res[5])

            # Загрузка групп для потоков
            flow_groups_map = {}
            if flow_ids:
                flow_groups = session.exec(
                    select(FlowGroupLink.flow_id, Group.name)
                    .join(Group, FlowGroupLink.group_id ==
Group.id)
                    .where(FlowGroupLink.flow_id.in_(flow_ids))
```

```
                ).all()

                for flow_id, group_name in flow_groups:
                    if flow_id not in flow_groups_map:
                        flow_groups_map[flow_id] = []
                    flow_groups_map[flow_id].append(group_name)

            # Формарование результата
            curriculum_list = []
            for row in results:
                curriculum, subject_short, primary_teacher,
secondary_teacher, group_name, flow_id = row

                groups = []
                if group_name:
                    groups = [group_name]
                elif flow_id:
                    groups = flow_groups_map.get(flow_id, [])

                curriculum_list.append(CurriculumData(
                    id=curriculum.id,
                    subject=subject_short,
                    groups=groups,
                    hours=curriculum.hours,
                    primary_teacher=primary_teacher,
                    secondary_teacher=secondary_teacher
                ))

            return curriculum_list

    def get_schedule(self) -> ScheduleData:
        with Session(self.engine) as session:
            PrimaryTeacher = aliased(Teacher)
            SecondaryTeacher = aliased(Teacher)

            schedule_query = (
                select(
                    Lesson,
                    Curriculum,
                    Subject,
                    Classroom,
                    PrimaryTeacher,
                    SecondaryTeacher,
                    Group
                )
                .join(Curriculum, Lesson.curriculum_id ==
Curriculum.id)
                .join(Subject, Curriculum.subject_id ==
Subject.id)
                .join(Classroom, Lesson.classroom_id ==
Classroom.id)
                .join(PrimaryTeacher,
Curriculum.primary_teacher_id == PrimaryTeacher.id)
```

```
                .outerjoin(SecondaryTeacher,
Curriculum.secondary_teacher_id == SecondaryTeacher.id)
                .outerjoin(Group, Curriculum.group_id ==
Group.id)
            )
            results = session.exec(schedule_query).all()

            flow_ids = set()
            for row in results:
                _, curriculum, *_, group = row
                if curriculum.flow_id is not None:
                    flow_ids.add(curriculum.flow_id)

            flow_groups = defaultdict(list)
            if flow_ids:
                flow_group_stmt = select(FlowGroupLink,
Group).join(Group).where(FlowGroupLink.flow_id.in_(flow_ids))
                flow_group_results =
session.exec(flow_group_stmt).all()
                for link, group in flow_group_results:
                    flow_groups[link.flow_id].append(group)

            schedule_dict = {
                week: {
                    day: {pair: {} for pair in range(1,9)} for
day in range(1,7)
                } for week in range(1,3)
            }
            for row in results:
                lesson, curriculum, subject, classroom,
primary_teacher, secondary_teacher, group = row

                groups = []
                if curriculum.group_id and group is not None:
                    groups = [group]
                elif curriculum.flow_id:
                    groups = flow_groups.get(curriculum.flow_id,
[])

                teachers = primary_teacher.full_name
                if secondary_teacher:
                    teachers += f",
{secondary_teacher.full_name}"

                cell_data = ScheduleCellData(
                    id=lesson.id,
                    lesson_type=lesson.lesson_type,
                    subject=subject.name,
                    teachers=teachers,
                    classroom=classroom.name
                )

                week = int(lesson.week)
```

```
                day = int(lesson.day)
                pair = int(lesson.pair)
                for group_obj in groups:
                    group_name = group_obj.name

                    schedule_dict[week][day][pair][group_name] =
cell_data

            # Преобразование defaultdict в обычный dict
            return ScheduleData(data=schedule_dict)

    def edit_schedule_cell(
        self,
        id: int,
        classroom_id: int,
        curriculum_id: int,
        lesson_type: LessonType
    ) -> None:
        with Session(self.engine) as session:
            # Получение занятия
            lesson = session.get(Lesson, id)
            if not lesson:
                raise ValueError(f"Ячейка расписания с ID {id}
не найдена")
            # Проверка существования аудитории
            classroom = session.get(Classroom, classroom_id)
            if not classroom:
                raise ValueError(f"Аудитория с ID {classroom_id}
не найдена")
            # Проверка существования занятия в учебном плане
            lesson_in_curriculum = session.get(Curriculum,
curriculum_id)
            if not lesson_in_curriculum:
                raise ValueError(f"Занятия с ID {curriculum_id}
нет в учебном плане")

            lesson.classroom_id = classroom_id
            lesson.curriculum_id = curriculum_id
            lesson.lesson_type = lesson_type

            session.add(lesson)
            session.commit()

    def remove_schedule_cell(self, id: int) -> None:
        with Session(self.engine) as session:
            lesson = session.get(Lesson, id)
            if not lesson:
                raise ValueError(f"Ячейка расписания с ID {id}
не найдена")
            session.delete(lesson)
            session.commit()

    def create_test_data(self) -> None:
```

```
        '''Создание тестовых данных'''
        with Session(self.engine) as session:
            # Университет
            university = Department(name="Технический
Университет", short_name="ТУ")
            session.add(university)
            session.commit()
            session.refresh(university)

            # Факультеты
            faculty1 = Department(
                name="Факультет Информационных Технологий",
                short_name="ФИТ",
                parent_id=university.id
            )
            faculty2 = Department(
                name="Инженерный Факультет",
                short_name="ИФ",
                parent_id=university.id
            )
            session.add_all([faculty1, faculty2])
            session.commit()
            session.refresh(faculty1)
            session.refresh(faculty2)

            # Кафедры
            department1 = Department(
                name="Кафедра Программной Инженерии",
                short_name="КПИ",
                parent_id=faculty1.id
            )
            department2 = Department(
                name="Кафедра Искусственного Интеллекта",
                short_name="КИИ",
                parent_id=faculty1.id
            )
            department3 = Department(
                name="Кафедра Машиностроения",
                short_name="КМ",
                parent_id=faculty2.id
            )
            session.add_all([department1, department2,
department3])
            session.commit()
            session.refresh(department1)
            session.refresh(department2)
            session.refresh(department3)

            # Специальности
            spec1 = Specialty(name="Программная инженерия",
department_id=department1.id)
            spec2 = Specialty(name="Искусственный интеллект",
department_id=department2.id)
```

```
            spec3 = Specialty(name="Машиностроение",
department_id=department3.id)
            session.add_all([spec1, spec2, spec3])
            session.commit()
            session.refresh(spec1)
            session.refresh(spec2)
            session.refresh(spec3)

            # Группы
            groups = [
                # ФИТ
                Group(name="ПИ-101",
course=CourseEnum.BACHELOR_1, specialty_id=spec1.id,
student_count=20),
                Group(name="ПИ-102",
course=CourseEnum.BACHELOR_1, specialty_id=spec1.id,
student_count=25),
                Group(name="ПИ-201",
course=CourseEnum.BACHELOR_2, specialty_id=spec1.id,
student_count=30),
                Group(name="ПИ-301",
course=CourseEnum.BACHELOR_3, specialty_id=spec1.id,
student_count=28),

                Group(name="ИИ-101",
course=CourseEnum.BACHELOR_1, specialty_id=spec2.id,
student_count=22),
                Group(name="ИИ-201",
course=CourseEnum.BACHELOR_2, specialty_id=spec2.id,
student_count=26),

                # ИФ
                Group(name="МС-101",
course=CourseEnum.BACHELOR_1, specialty_id=spec3.id,
student_count=35),
                Group(name="МС-102",
course=CourseEnum.BACHELOR_1, specialty_id=spec3.id,
student_count=40),
                Group(name="МС-301",
course=CourseEnum.BACHELOR_3, specialty_id=spec3.id,
student_count=32),
            ]
            session.add_all(groups)
            session.commit()

            # Потоки
            flow1 = Flow()  # Поток из ПИ-101 и ПИ-102
            flow2 = Flow()  # Поток из МС-101 и МС-102
            session.add_all([flow1, flow2])
            session.flush()

            # Добавление групп в потоки
```

```
            for group in [groups[0], groups[1]]:  # ПИ-101, ПИ-
102
                session.add(FlowGroupLink(flow_id=flow1.id,
group_id=group.id))

            for group in [groups[6], groups[7]]:  # МС-101, МС-
102
                session.add(FlowGroupLink(flow_id=flow2.id,
group_id=group.id))

            # Аудитории
            classrooms = [
                # Аудитории кафедр
                Classroom(name="А-101", capacity=30,
faculty_id=faculty1.id, department_id=department1.id),
                Classroom(name="А-102", capacity=40,
faculty_id=faculty1.id, department_id=department1.id),
                Classroom(name="Б-201", capacity=25,
faculty_id=faculty1.id, department_id=department2.id),
                Classroom(name="К-105", capacity=35,
faculty_id=faculty2.id, department_id=department3.id),

                # Аудитории факультетов
                Classroom(name="Лекторий-1", capacity=100,
faculty_id=faculty1.id),
                Classroom(name="Лекторий-2", capacity=80,
faculty_id=faculty2.id),
                Classroom(name="Компьютерный класс-1",
capacity=20, faculty_id=faculty1.id),
                Classroom(name="Лаборатория-1", capacity=15,
faculty_id=faculty2.id),
            ]
            session.add_all(classrooms)

            # Преподаватели
            teachers = [
                Teacher(full_name="Иванов Иван Иванович",
department_id=department1.id),
                Teacher(full_name="Петрова Анна Сергеевна",
department_id=department1.id),
                Teacher(full_name="Сидоров Алексей
Владимирович", department_id=department2.id),
                Teacher(full_name="Козлова Мария Дмитриевна",
department_id=department3.id),
                Teacher(full_name="Николаев Дмитрий Петрович",
department_id=department3.id),
            ]
            session.add_all(teachers)

            # Предметы
            subjects = [
                Subject(name="Математический анализ",
short_name="МатАн"),
```

```
                Subject(name="Программирование на Python",
short_name="Python"),
                Subject(name="Базы данных", short_name="БД"),
                Subject(name="Машинное обучение",
short_name="МО"),
                Subject(name="Теоретическая механика",
short_name="ТерМех"),
                Subject(name="Сопротивление материалов",
short_name="СопрМат"),
            ]
            session.add_all(subjects)
            session.commit()

            # Учебные планы
            curriculum = [
                # Для отдельных групп
                Curriculum(
                    subject_id=subjects[0].id,
                    hours=72,
                    primary_teacher_id=teachers[0].id,
                    group_id=groups[0].id  # ПИ-101
                ),
                Curriculum(
                    subject_id=subjects[1].id,
                    hours=108,
                    primary_teacher_id=teachers[1].id,
                    group_id=groups[2].id  # ПИ-201
                ),
                Curriculum(
                    subject_id=subjects[3].id,
                    hours=72,
                    primary_teacher_id=teachers[2].id,
                    group_id=groups[4].id  # ИИ-101
                ),

                # Для потоков
                Curriculum(
                    subject_id=subjects[0].id,
                    hours=144,
                    primary_teacher_id=teachers[0].id,
                    secondary_teacher_id=teachers[1].id,
                    flow_id=flow1.id  # Поток из ПИ-101 и ПИ-102
                ),
                Curriculum(
                    subject_id=subjects[4].id,
                    hours=108,
                    primary_teacher_id=teachers[3].id,
                    flow_id=flow2.id  # Поток из МС-101 и МС-102
                ),

                # Сложный случай (большой поток)
                Curriculum(
                    subject_id=subjects[5].id,
```

```
                    hours=72,
                    primary_teacher_id=teachers[4].id,
                    group_id=groups[8].id  # МС-301
                )
            ]
            session.add_all(curriculum)

            # Несколько существующих занятий в расписании
            existing_lessons = [
                Lesson(
                    week=1,
                    day=1,  # Понедельник
                    pair=1,
                    classroom_id=classrooms[0].id,
                    curriculum_id=curriculum[0].id,
                    lesson_type=LessonType.LECTURE
                ),
                Lesson(
                    week=1,
                    day=1,
                    pair=2,
                    classroom_id=classrooms[1].id,
                    curriculum_id=curriculum[1].id,
                    lesson_type=LessonType.LAB
                )
            ]
            session.add_all(existing_lessons)

            session.commit()

    def auto_schedule(self) -> None:
        '''Автоматическое составление расписания с учётом
семестровой нагрузки'''
        with Session(self.engine) as session:
            # Сбор существующей занятости для недель 1 и 2
            busy_rooms = set()
            busy_teachers = set()
            busy_groups = set()

            # Загрузка существующих занятий в неделях 1 и 2
            existing_lessons = session.exec(
                select(Lesson)
                .where(Lesson.week.in_([1, 2]))
            ).all()

            for lesson in existing_lessons:
                curriculum = session.get(Curriculum,
lesson.curriculum_id)
                if not curriculum:
                    continue

                # Определение групп
                group_ids = []
```

```
                if curriculum.group_id:
                    group_ids = [curriculum.group_id]
                elif curriculum.flow_id:
                    flow_groups = session.exec(
                        select(FlowGroupLink.group_id)
                        .where(FlowGroupLink.flow_id ==
curriculum.flow_id)
                    ).all()
                    group_ids = [g[0] for g in flow_groups]

                # Преподаватели
                teacher_ids = [curriculum.primary_teacher_id]
                if curriculum.secondary_teacher_id:

teacher_ids.append(curriculum.secondary_teacher_id)

                # Обновление занятости
                for gid in group_ids:
                    busy_groups.add((lesson.week, lesson.day,
lesson.pair, gid))
                for tid in teacher_ids:
                    busy_teachers.add((lesson.week, lesson.day,
lesson.pair, tid))
                busy_rooms.add((lesson.week, lesson.day,
lesson.pair, lesson.classroom_id))

            # Подготовка учебных планов
            curriculum_data = []
            all_curriculum =
session.exec(select(Curriculum)).all()

            # Шаблоны занятий на двухнедельный цикл
            curriculum_templates = {
                72: [LessonType.LECTURE],  # 1 лекция за 2
недели
                108: [LessonType.LECTURE, LessonType.LAB],  # 1
лекция + 1 лаба
                144: [LessonType.LECTURE, LessonType.LAB,
LessonType.LAB]  # 1 лекция + 2 лабы
            }

            for curr in all_curriculum:
                # Проверка допустимости часов
                if curr.hours not in curriculum_templates:
                    print(f"Недопустимое количество часов:
{curr.hours} для curriculum_id={curr.id}")
                    continue

                # Получаем шаблон занятий для данного количества
часов
                template = curriculum_templates[curr.hours]
```

```
                # Подсчёт уже запланированных занятий каждого
типа в цикле (недели 1 и 2)
                type_counter = {LessonType.LECTURE: 0,
LessonType.LAB: 0}
                for lesson_type in template:
                    existing_count = session.scalar(
                        select(func.count(Lesson.id))
                        .where(Lesson.curriculum_id == curr.id)
                        .where(Lesson.week.in_([1, 2]))
                        .where(Lesson.lesson_type ==
lesson_type)
                    ) or 0
                    type_counter[lesson_type] = existing_count

                # Определяем, сколько занятий каждого типа нужно
добавить
                needed_lessons = []
                for lesson_type in template:
                    # Сколько должно быть занятий этого типа
                    required_count = sum(1 for t in template if
t == lesson_type)
                    # Сколько уже есть
                    current_count = type_counter[lesson_type]
                    # Сколько нужно добавить
                    to_add = required_count - current_count

                    # Добавляем в список необходимых занятий
                    needed_lessons.extend([lesson_type] *
to_add)

                if not needed_lessons:
                    continue

                # Определение групп и вместимости
                group_ids = []
                capacity_required = 0
                if curr.group_id:
                    group = session.get(Group, curr.group_id)
                    if group:
                        group_ids = [group.id]
                        capacity_required = group.student_count
                elif curr.flow_id:
                    flow_groups = session.exec(
                        select(Group)
                        .join(FlowGroupLink, Group.id ==
FlowGroupLink.group_id)
                        .where(FlowGroupLink.flow_id ==
curr.flow_id)
                    ).all()
                    group_ids = [g.id for g in flow_groups]
                    capacity_required = sum(g.student_count for
g in flow_groups)
```

```
                if not group_ids:
                    continue

                # Подходящие аудитории
                classrooms = session.exec(
                    select(Classroom)
                    .where(Classroom.capacity >=
capacity_required)
                ).all()
                classroom_ids = [c.id for c in classrooms]

                curriculum_data.append({
                    'obj': curr,
                    'needed_lessons': needed_lessons,
                    'groups': group_ids,
                    'teachers': [
                        t for t in [curr.primary_teacher_id,
curr.secondary_teacher_id]
                        if t is not None
                    ],
                    'classrooms': classroom_ids,
                    'capacity_req': capacity_required
                })

            # Сортировка по сложности (вместимость аудитории и
количество групп)
            curriculum_data.sort(key=lambda x: (-
x['capacity_req'], -len(x['groups'])))

            # Распределение занятий
            for item in curriculum_data:
                curr = item['obj']
                for lesson_type in item['needed_lessons']:
                    placed = False
                    for week in [1, 2]:
                        for day in range(1, 7):  # Пн-Сб
                            for pair in range(1, 9):  # 1-8 пары
                                # Проверка групп
                                if any(
                                    (week, day, pair, gid) in
busy_groups
                                    for gid in item['groups']
                                ):
                                    continue

                                # Проверка преподавателей
                                if any(
                                    (week, day, pair, tid) in
busy_teachers
                                    for tid in item['teachers']
                                ):
                                    continue
```

```
                                    # Поиск свободной аудитории
                                    for room_id in
item['classrooms']:
                                        if (week, day, pair,
room_id) in busy_rooms:
                                            continue

                                        # Создание занятия
                                        new_lesson = Lesson(
                                            week=week,
                                            day=day,
                                            pair=pair,
                                            classroom_id=room_id,
                                            curriculum_id=curr.id,
                                            lesson_type=lesson_type
                                        )
                                        session.add(new_lesson)

                                        # Обновление занятости
                                        busy_rooms.add((week, day,
pair, room_id))
                                        for gid in item['groups']:
                                            busy_groups.add((week,
day, pair, gid))
                                        for tid in item['teachers']:
                                            busy_teachers.add((week,
day, pair, tid))

                                        placed = True
                                        break
                                    if placed:
                                        break
                                if placed:
                                    break
                            if placed:
                                break
                        if not placed:
                            print(f"Не удалось разместить
curriculum_id={curr.id} (тип: {lesson_type})")

            session.commit()

    def find_collisions(self) -> dict:
        '''Поиск коллизий и окон в расписании'''
        with Session(self.engine) as session:
            # Загрузка всех занятий
            lessons = session.exec(select(Lesson)).all()

            # Структуры для анализа
            group_schedule = defaultdict(lambda:
defaultdict(lambda: defaultdict(list)))
            teacher_schedule = defaultdict(lambda:
defaultdict(lambda: defaultdict(list)))
```

```
            room_schedule = defaultdict(lambda:
defaultdict(lambda: defaultdict(list)))

            # Заполнение структур
            for lesson in lessons:
                curriculum = session.get(Curriculum,
lesson.curriculum_id)
                if not curriculum:
                    continue

                # Определение групп
                group_ids = []
                if curriculum.group_id:
                    group_ids = [curriculum.group_id]
                elif curriculum.flow_id:
                    # Исправление: получение простого списка ID
групп
                    flow_groups = session.exec(
                        select(FlowGroupLink.group_id)
                        .where(FlowGroupLink.flow_id ==
curriculum.flow_id)
                    ).all()
                    group_ids = [g for g, in flow_groups] if
flow_groups and isinstance(flow_groups[0], tuple) else
flow_groups

                # Преподаватели
                teacher_ids = [curriculum.primary_teacher_id]
                if curriculum.secondary_teacher_id:

teacher_ids.append(curriculum.secondary_teacher_id)

                # Заполнение данных
                for group_id in group_ids:

group_schedule[group_id][lesson.week][lesson.day].append((lesson
.pair, lesson.id))

                for teacher_id in teacher_ids:

teacher_schedule[teacher_id][lesson.week][lesson.day].append((le
sson.pair, lesson.id))


room_schedule[lesson.classroom_id][lesson.week][lesson.day].appe
nd((lesson.pair, lesson.id))

            # Поиск ошибок (наложение занятий)
            errors = []

            # Для групп
            for group_id, weeks in group_schedule.items():
                for week, days in weeks.items():
```

```
                    for day, pairs in days.items():
                        pairs.sort()
                        for i in range(1, len(pairs)):
                            if pairs[i][0] == pairs[i-1][0]:
                                errors.append({
                                    "type": "group",
                                    "id": group_id,
                                    "week": week,
                                    "day": day,
                                    "pair": pairs[i][0],
                                    "lesson_ids": [pairs[i-
1][1], pairs[i][1]]
                                })

            # Для преподавателей
            for teacher_id, weeks in teacher_schedule.items():
                for week, days in weeks.items():
                    for day, pairs in days.items():
                        pairs.sort()
                        for i in range(1, len(pairs)):
                            if pairs[i][0] == pairs[i-1][0]:
                                errors.append({
                                    "type": "teacher",
                                    "id": teacher_id,
                                    "week": week,
                                    "day": day,
                                    "pair": pairs[i][0],
                                    "lesson_ids": [pairs[i-
1][1], pairs[i][1]]
                                })

            # Для аудиторий
            for room_id, weeks in room_schedule.items():
                for week, days in weeks.items():
                    for day, pairs in days.items():
                        pairs.sort()
                        for i in range(1, len(pairs)):
                            if pairs[i][0] == pairs[i-1][0]:
                                errors.append({
                                    "type": "classroom",
                                    "id": room_id,
                                    "week": week,
                                    "day": day,
                                    "pair": pairs[i][0],
                                    "lesson_ids": [pairs[i-
1][1], pairs[i][1]]
                                })

            # Поиск окон
            group_windows = []
            teacher_windows = []

            # Для групп
```

```
            for group_id, weeks in group_schedule.items():
                for week, days in weeks.items():
                    for day, pairs in days.items():
                        pairs.sort()
                        for i in range(1, len(pairs)):
                            gap = pairs[i][0] - pairs[i-1][0] -
1
                            if gap > 0:
                                group_windows.append({
                                    "group_id": group_id,
                                    "week": week,
                                    "day": day,
                                    "window_start_pair":
pairs[i-1][0],
                                    "window_end_pair":
pairs[i][0],
                                    "window_size": gap
                                })

            # Для преподавателей
            for teacher_id, weeks in teacher_schedule.items():
                for week, days in weeks.items():
                    for day, pairs in days.items():
                        pairs.sort()
                        for i in range(1, len(pairs)):
                            gap = pairs[i][0] - pairs[i-1][0] -
1
                            if gap > 0:
                                teacher_windows.append({
                                    "teacher_id": teacher_id,
                                    "week": week,
                                    "day": day,
                                    "window_start_pair":
pairs[i-1][0],
                                    "window_end_pair":
pairs[i][0],
                                    "window_size": gap
                                })

            return {
                "errors": errors,
                "group_windows": group_windows,
                "teacher_windows": teacher_windows
            }


def get_database(db_string: str) -> Database:
    '''Проверяет, корректно ли введён параметр базы данных,
    возвращает объект БД'''
    try:
        return SQLDatabase(db_string)
    except Exception as e:
        raise TypeError(
```

```
            f'The database address is incorrect. Error: {e}'
        ) from e
```

Листинг Б.5 – Файл frontend/index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Vite + React + TS</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Листинг Б.6 – Файл frontend/src/main.tsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Листинг Б.7 – Файл frontend/src/index.css

```
:root {
  font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
  font-weight: 400;

  color-scheme: light dark;
  color: rgba(255, 255, 255, 0.87);
  background-color: #242424;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

a {
  font-weight: 500;
  color: #646cff;
  text-decoration: inherit;
}
```

```
a:hover {
  color: #535bf2;
}

body {
  margin: 0;
  display: flex;
  place-items: center;
  min-width: 320px;
  min-height: 100vh;
}

h1 {
  font-size: 3.2em;
  line-height: 1.1;
}

button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #1a1a1a;
  cursor: pointer;
  transition: border-color 0.25s;
}
button:hover {
  border-color: #646cff;
}
button.active {
  border-color: #646cff;
}
button:focus,
button:focus-visible {
  outline: 4px auto -webkit-focus-ring-color;
}

@media (prefers-color-scheme: light) {
  :root {
    color: #213547;
    background-color: #ffffff;
  }
  a:hover {
    color: #747bff;
  }
  button {
    background-color: #f9f9f9;
  }
}
```

Листинг Б.8 – Файл frontend/src/App.tsx

```
import { useState, useEffect, useMemo } from 'react'
import './App.css'
import {
  UniversityType, SubjectType, FlowType, CurriculumType,
GroupType,
  ScheduleData, ClassroomType
} from './types';
import {
  fetchUniversityData, fetchSubjects, fetchFlows,
fetchCurriculum, fetchSchedule
} from './api';

import { ScheduleComponent } from
'./components/ScheduleComponent.tsx';
import { DataComponent } from './components/DataComponent.tsx';

function App() {
  const [universityData, setUniversityData] =
useState<UniversityType[]>([]);
  const [subjects, setSubjects] = useState<SubjectType[]>([]);
  const [flows, setFlows] = useState<FlowType[]>([]);
  const [curriculum, setCurriculum] =
useState<CurriculumType[]>([]);
  const [schedule, setSchedule] = useState<ScheduleData>({data:
{}});

  // Для вкладки "Расписание"
  const [activeUniversity, setActiveUniversity] =
useState<string | null>(null);
  const [activeFaculty, setActiveFaculty] = useState<string |
null>(null);
  const [activeDepartment, setActiveDepartment] =
useState<string | null>(null);
  const [groupsForSchedule, setGroupsForSchedule] =
useState<GroupType[]>([]);
  const [currentWeek, setCurrentWeek] = useState<number>(1);

  const allClassrooms = useMemo<ClassroomType[]>(() => {
    const getAllGroups = (data: UniversityType[]):
ClassroomType[] => {
      let facultiesClassrooms = data.flatMap(university =>
        university.faculties.flatMap(faculty =>
faculty.classrooms)
      );
      let departmentsClassrooms = data.flatMap(university =>
        university.faculties.flatMap(faculty =>
          faculty.departments.flatMap(department =>
department.classrooms)
        )
      );
      return facultiesClassrooms.concat(departmentsClassrooms);
    };
    return Array.from(new Set(getAllGroups(universityData)));
```

```
}, [universityData]);

const [activeTab, setActiveTab] = useState('Данные');

useEffect(() => {
  const loadData = async () => {
    const universetyData = await fetchUniversityData();
    setUniversityData(universetyData);
    const subjectsData = await fetchSubjects();
    setSubjects(subjectsData);
    const flowsData = await fetchFlows();
    setFlows(flowsData);
    const curriculumData = await fetchCurriculum();
    setCurriculum(curriculumData);
    const scheduleData = await fetchSchedule();
    setSchedule(scheduleData);
  };

  loadData();
}, []);

const handleTabClick = (tab: string) => {
  setActiveTab(tab);
};

return (
  <div className="tabs-container">
    <div className="tabs">
      <button
        className={activeTab === 'Данные' ? 'active' : ''}
        onClick={() => handleTabClick('Данные')}
        style={{ borderTopLeftRadius: 10, marginLeft: 5 }}
      >
        Данные
      </button>
      <button
        className={activeTab === 'Расписание' ? 'active' : ''}
        onClick={() => handleTabClick('Расписание')}
        style={{ borderTopRightRadius: 10 }}
      >
        Расписание
      </button>
    </div>
    <div className="tab-content">
      {activeTab === 'Данные' &&
        <DataComponent
          universityData={universityData}
          setUniversityData={setUniversityData}
          subjects={subjects}
          setSubjects={setSubjects}
          flows={flows}
          setFlows={setFlows}
          CurriculumData={curriculum}
```

```
            setCurriculumData={setCurriculum}
          />
        }
        {activeTab === 'Расписание' &&
          <ScheduleComponent
            universityData={universityData}
            setUniversityData={setUniversityData}
            subjects={subjects}
            setSubjects={setSubjects}
            flows={flows}
            setFlows={setFlows}
            CurriculumData={curriculum}
            setCurriculumData={setCurriculum}
            activeUniversity={activeUniversity}
            setActiveUniversity={setActiveUniversity}
            activeFaculty={activeFaculty}
            setActiveFaculty={setActiveFaculty}
            activeDepartment={activeDepartment}
            setActiveDepartment={setActiveDepartment}
            groups={groupsForSchedule}
            setGroups={setGroupsForSchedule}
            currentWeek={currentWeek}
            setCurrentWeek={setCurrentWeek}
            schedule={schedule}
            setSchedule={setSchedule}
            allClassrooms={allClassrooms}
          />
        }
      </div>
    </div>
  );
}

export default App;
```

Листинг Б.9 – Файл frontend/src/App.css

```
/* Стили для контейнера вкладок */
.tabs-container {
  display: flex;
  flex-direction: column;
  height: 100vh;
  width: 100vw;
  margin: 0;
}

/* Стили для блока вкладок */
.tabs {
  display: flex;
  position: sticky;
  top: 0;
  background-color: #fff;
  z-index: 1;
}
```

```
/* Стили для кнопок вкладок */
.tabs button {
  padding: 10px 20px;
  margin-top: 3px;
  border-radius: 0;
  border: 1px solid #ccc;
  background-color: inherit;
  cursor: pointer;
}

/* Стиль активной вкладки */
.tabs button.active {
  margin-top: 1px;
  background-color: #fff;
  border-bottom: none;
  position: relative;
  z-index: 1;
}

/* Стили для содержимого вкладок */
.tab-content {
  flex: 1;
  border: 1px solid #ccc;
  padding: 20px;
  overflow-y: auto;
}

.modal {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  padding: 20px;
  background: white;
  border: 1px solid #ccc;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

.styled-select {
  width: 100%;
  padding: 8px 12px;
  margin-bottom: 15px;
  background: #f5f5f5;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
  appearance: none; /* Убираем стандартную стрелку */
  background-image: url('data:image/svg+xml;utf8,<svg
xmlns="http://www.w3.org/2000/svg" width="16" height="16"
fill="gray" viewBox="0 0 16 16"><path d="M7.247 11.14L2.451
5.658C1.885 5.013 2.345 4 3.204 4h9.592a1 1 0 0 1 .753 1.657l-
4.796 5.489a1 1 0 0 1-1.504 0z"/></svg>');
```

```
  background-position: right 10px center;
  background-repeat: no-repeat;
}

.input-field {
  width: 90%;
  padding: 8px 12px;
  margin-bottom: 15px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
}

.suggestion {
  padding: 8px;
  cursor: pointer;
  border-bottom: 1px solid #eee;
}

.suggestion:hover {
  background-color: #f0f0f0;
}

.add-button {
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
}

.add-button.disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

.form-group label {
  display: block;
}

.form-control {
  width: 100%;
  box-sizing: border-box;
}
```

Листинг Б.10 – Файл frontend/src/api.ts

```
// Модуль определяет взаимодействие с методами API

import {
  UniversityType, SubjectType, FlowType, CurriculumType,
ScheduleData
} from './types';
```

```
// Получение данных о структурных подразделения университета
export const fetchUniversityData = async ():
Promise<UniversityType[]> => {
    try {
      const response = await
fetch("http://localhost:8000/university_data");
      if (!response.ok) {
        throw new Error(`HTTP error! Status:
${response.status}`);
      }
      return await response.json();
    } catch (error) {
      console.error("Failed to fetch university data:", error);
      return [];
    }
};

// Получение предметов
export const fetchSubjects = async (): Promise<SubjectType[]> =>
{
    try {
      const response = await
fetch("http://localhost:8000/subject");
      if (!response.ok) {
        throw new Error(`HTTP error! Status:
${response.status}`);
      }
      return await response.json();
    } catch (error) {
      console.error("Failed to fetch university data:", error);
      return [];
    }
};

// Получение потоков
export const fetchFlows = async (): Promise<FlowType[]> => {
    try {
      const response = await
fetch("http://localhost:8000/flow");
      if (!response.ok) {
        throw new Error(`HTTP error! Status:
${response.status}`);
      }
      return await response.json();
    } catch (error) {
      console.error("Failed to fetch university data:", error);
      return [];
    }
};

// Получение учебного плана
export const fetchCurriculum = async ():
Promise<CurriculumType[]> => {
```

```
    try {
      const response = await
fetch("http://localhost:8000/curriculum");
      if (!response.ok) {
        throw new Error(`HTTP error! Status:
${response.status}`);
      }
      return await response.json();
    } catch (error) {
      console.error("Failed to fetch university data:", error);
      return [];
    }
};

// Получение расписания
export const fetchSchedule = async (): Promise<ScheduleData> =>
{
  try {
    const response = await
fetch("http://localhost:8000/schedule");
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return await response.json();
  } catch (error) {
    console.error("Failed to fetch university data:", error);
    return {data: {}};
  }
};

// Удаление ячейки расписания
export const fetchRemoveSchedule = async (id: number):
Promise<null> => {
  try {
    const url = new URL('http://localhost:8000/schedule');
    url.searchParams.append('id', id.toString());
    const response = await fetch(url.toString(), {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(
        `Ошибка ${response.status}: ${errorData.detail ||
'Неизвестная ошибка'}`
      );
    }

    return await response.json();
  } catch (error) {
```

```
    console.error('Ошибка создания специальности:', error);
    throw error;
  }
};

// Изменение ячейки расписания
export const fetchEditSchedule = async (
  id: number,
  classroom_id: number,
  curriculum_id: number,
  lesson_type: string
): Promise<null> => {
  try {
    const url = new URL('http://localhost:8000/schedule');
    url.searchParams.append('id', id.toString());
    url.searchParams.append('classroom_id',
classroom_id.toString());
    url.searchParams.append('curriculum_id',
curriculum_id.toString());
    url.searchParams.append('lesson_type', lesson_type);

    const response = await fetch(url.toString(), {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(
        `Ошибка ${response.status}: ${errorData.detail ||
'Неизвестная ошибка'}`
      );
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания специальности:', error);
    throw error;
  }
};

// Добавление структурного подразделения: университета,
факультета либо кафедры
export const createStructuralDivision = async (
  name: string,
  parent_id?: number,
  short_name?: string
): Promise<number> => {
  try {
    const url = new
URL('http://localhost:8000/structural_divizion');
```

```
    url.searchParams.append('name', name);
    if (short_name) url.searchParams.append('short_name',
short_name);
    if (parent_id !== undefined)
url.searchParams.append('parent_id', parent_id.toString());

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(
        `Ошибка ${response.status}: ${errorData.detail ||
'Неизвестная ошибка'}`
      );
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания подразделения:', error);
    throw error;
  }
};

// Добавление специальности
export const createSpeciality = async (
  department_id: number,
  name: string
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/speciality');
    url.searchParams.append('department_id',
department_id.toString());
    url.searchParams.append('name', name);

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(
        `Ошибка ${response.status}: ${errorData.detail ||
'Неизвестная ошибка'}`
      );
    }
```

```
    return await response.json();
  } catch (error) {
    console.error('Ошибка создания специальности:', error);
    throw error;
  }
};

// Добавление группы
export const createGroup = async (
  speciality_id: number,
  name: string,
  course: string, // Предполагается, что тип CourseEnum
определен
  student_count: number
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/group');
    url.searchParams.append('speciality_id',
speciality_id.toString());
    url.searchParams.append('name', name);
    url.searchParams.append('course', course);
    url.searchParams.append('student_count',
student_count.toString());

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания группы:', error);
    throw error;
  }
};

// Добавление преподавателя
export const createTeacher = async (
  department_id: number,
  name: string
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/teacher');
    url.searchParams.append('department_id',
department_id.toString());
    url.searchParams.append('name', name);
```

```
    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания преподавателя:', error);
    throw error;
  }
};

// Добавление аудитории
export const createClassroom = async (
  faculty_id: number,
  name: string,
  capacity: number,
  department_id?: number
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/classroom');
    url.searchParams.append('faculty_id',
faculty_id.toString());
    if (department_id) {
      url.searchParams.append('department_id',
department_id.toString());
    }
    url.searchParams.append('name', name);
    url.searchParams.append('capacity', capacity.toString());

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания аудитории:', error);
    throw error;
  }
```

```
};

// Добавление предмета
export const createSubject = async (
  name: string,
  short_name: string
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/subject');
    url.searchParams.append('name', name);
    url.searchParams.append('short_name', short_name);

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания предмета:', error);
    throw error;
  }
};

// Добавление потока (с передачей данных в теле запроса)
export const createFlow = async (
  groups: string[] // Массив имен групп
): Promise<number> => {
  try {
    const response = await fetch('http://localhost:8000/flow', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(groups),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка создания потока:', error);
    throw error;
  }
};
```

```
// Добавление занятия в учебный план
export const addCurriculumLesson = async (
  subject_id: number,
  hours: number,
  primary_teacher_id: number,
  secondary_teacher_id?: number,
  group_id?: number,
  flow_id?: number
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/curriculum');
    url.searchParams.append('subject_id',
subject_id.toString());
    url.searchParams.append('hours', hours.toString());
    url.searchParams.append('primary_teacher_id',
primary_teacher_id.toString());

    if (secondary_teacher_id !== undefined) {
      url.searchParams.append('secondary_teacher_id',
secondary_teacher_id.toString());
    }
    if (group_id !== undefined) {
      url.searchParams.append('group_id', group_id.toString());
    }
    if (flow_id !== undefined) {
      url.searchParams.append('flow_id', flow_id.toString());
    }

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка добавления занятия в учебный план:',
error);
    throw error;
  }
};

// Добавление занятия в расписание
export const addScheduleLesson = async (
  week: number,
  day: number,
  pair: number,
```

```
  classroom_id: number,
  curriculum_id: number,
  lesson_type: string
): Promise<number> => {
  try {
    const url = new URL('http://localhost:8000/schedule');
    url.searchParams.append('week', week.toString());
    url.searchParams.append('day', day.toString());
    url.searchParams.append('pair', pair.toString());
    url.searchParams.append('classroom_id',
classroom_id.toString());
    url.searchParams.append('curriculum_id',
curriculum_id.toString());
    url.searchParams.append('lesson_type', lesson_type);

    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }

    return await response.json();
  } catch (error) {
    console.error('Ошибка добавления занятия в расписание:',
error);
    throw error;
  }
};

// Автоматическое построение расписания
export const autoShedule = async () => {
  try {
    const url = new
URL('http://localhost:8000/auto_create_schedule');
    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }
  } catch (error) {
    console.error('Ошибка автоматического построения
расписания:', error);
    throw error;
```

```
  }
};

// Автоматическое нахождение коллизий расписания
export const findCollisions = async () => {
  try {
    const url = new
URL('http://localhost:8000/find_collisions');
    const response = await fetch(url.toString(), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`Ошибка ${response.status}:
${errorData.detail || 'Неизвестная ошибка'}`);
    }
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Ошибка автоматического построения
расписания:', error);
    throw error;
  }
};
```

Листинг Б.11 – Файл frontend/src/types.ts

```
export type ClassroomType = {
    id: number;
    number: string;
};

export type LecturerType = {
    id: number;
    full_name: string;
};

export type GroupType = {
    id: number;
    name: string;
    course: string;
    studentsCount: number;
};

/// Специальности
type SpecialitiesType = {
    id: number;
    name: string;
    groups: GroupType[];
}

type DepartmentType = {
```

```
    id: number;
    name: string;
    short_name: string;
    specialities: SpecialitiesType[];
    lecturers: LecturerType[];
    classrooms: ClassroomType[];
};

type FacultyType = {
    id: number;
    name: string;
    departments: DepartmentType[];
    classrooms: ClassroomType[];
};

export type UniversityType = {
    id: number;
    name: string;
    faculties: FacultyType[];
};

/// Интерфейс для входных данных компонентов
export interface UniversityTableProps {
    data: UniversityType[];
    setData:
React.Dispatch<React.SetStateAction<UniversityType[]>>;
}

export type SubjectType = {
    id: number;
    name: string;
    short_name: string;  // Сокращение
};

// Потоки
export type FlowType = {
    id: number;
    groups: string[];
};

// Учебный план
export type CurriculumType = {
    id: number;
    subject: string;    // Идентификация предмета по полному
имени
    groups: string[];   // Список групп
    hours: number;  // Число часов за семестр
    primary_teacher: string; // ФИО преподавателя
    secondary_teacher?: string; // ФИО второго преподавателя
};

// Типы для расписания занятий (получение из API)
```

```
export type LessonType = "лекционное" | "лабораторное";

// Данные ячейки расписания
export interface ScheduleCellData {
  id: number;
  lesson_type: LessonType;
  subject: string;     // Название предмета
  teachers: string;    // Преподаватели
  classroom: string;   // Номер аудитории
}

export type WeekNumber = 1 | 2; // Номер недели
export type DayNumber = 1 | 2 | 3 | 4 | 5 | 6; // День недели
(Пн-Сб)
export type PairNumber = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8; // Номер
пары

// Основная структура расписания
export interface ScheduleData {
    data: {
      [week in WeekNumber]?: {                      // Неделя (1
или 2)
        [day in DayNumber]?: {                     // День
недели (1-6)
          [classNum in PairNumber]?: {            // Номер пары
(1-8)
            [group: string]: ScheduleCellData;     // Группа
(название)
          }
        }
      }
    }
  }
```

Листинг Б.12 – Файл frontend/src/components/ClassroomsTable.tsx

```
import React, { useState, useEffect } from 'react';
import { UniversityTableProps } from '../types';
import { createClassroom } from '../api';

export const ClassroomsTable = ({ data, setData }:
UniversityTableProps) => {
  const [IsAddClassroomModalOpen, setIsAddClassroomModalOpen] =
useState(false);
  const [selectedUniversityIdToClassroom,
setSelectedUniversityIdToClassroom] =
useState<number|null>(null);
  const [selectedFacultyIdToClassroom,
setSelectedFacultyIdToClassroom] = useState<number|null>(null);
  const [selectedDepartmentIdToClassroom,
setSelectedDepartmentIdToClassroom] =
useState<number|null>(null);
  const [newClassroomNumber, setNewClassroomNumber] =
useState("");
```

```
  const [newClassroomCapacity, setNewClassroomCapacity] =
useState(15);

  useEffect(() => {
    setSelectedFacultyIdToClassroom(null);
    setSelectedDepartmentIdToClassroom(null);
  }, [selectedUniversityIdToClassroom]);

  useEffect(() => {
    setSelectedDepartmentIdToClassroom(null);
  }, [selectedFacultyIdToClassroom]);

  const addClassroom = async () => {
    if (
      !selectedUniversityIdToClassroom ||
      !selectedFacultyIdToClassroom ||
      !newClassroomNumber
    ) return;

    const selectedUniversity = data.find(u => u.id ===
selectedUniversityIdToClassroom);
    const selectedFaculty = selectedUniversity?.faculties.find(f
=> f.id === selectedFacultyIdToClassroom);

    if (selectedDepartmentIdToClassroom === null) {
      // Если кафедра не выбрана - аудитория принадлежит всему
факультету
      if (!selectedFaculty) return;
      const newId = await createClassroom(
        selectedFacultyIdToClassroom,
        newClassroomNumber,
        newClassroomCapacity
      );
      setData(prevData => prevData.map(university => {
        if (university.id === selectedUniversityIdToClassroom) {
          return {
            ...university,
            faculties: university.faculties.map(faculty => {
              if (faculty.id === selectedFacultyIdToClassroom) {
                return {
                  ...faculty,
                  classrooms: [
                    ...faculty.classrooms,
                    { id: newId, number: newClassroomNumber }
                  ]
                };
              }
              return faculty;
            })
          };
        }
        return university;
      }));
```

```
    } else {
      // Если кафедра выбрана - добавляем аудиторию к кафедре
      const selectedDepartment =
selectedFaculty?.departments.find(d => d.id ===
selectedDepartmentIdToClassroom);
      if (!selectedDepartment) return;
      const newId = await createClassroom(
        selectedFacultyIdToClassroom,
        newClassroomNumber,
        newClassroomCapacity,
        selectedDepartmentIdToClassroom
      );
        setData(prevData => prevData.map(university => {
          if (university.id === selectedUniversityIdToClassroom)
{
            return {
              ...university,
              faculties: university.faculties.map(faculty => {
                if (faculty.id === selectedFacultyIdToClassroom)
{
                  return {
                    ...faculty,
                    departments:
faculty.departments.map(department => {
                      if (department.id ===
selectedDepartmentIdToClassroom) {
                        return {
                          ...department,
                          classrooms: [
                            ...department.classrooms,
                            { id: newId, number:
newClassroomNumber }
                          ]
                        };
                      }
                      return department;
                    })
                  };
                }
                return faculty;
              })
            };
          }
          return university;
        }));
    }

    setIsAddClassroomModalOpen(false);
    setSelectedUniversityIdToClassroom(null);
    setSelectedFacultyIdToClassroom(null);
    setSelectedDepartmentIdToClassroom(null);
    setNewClassroomNumber("");
    setNewClassroomCapacity(15);
```

```
  };

  return (
    <>
      <h2>Аудитории</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
        <thead>
          <tr>
            <th>Номер</th>
            <th>Факультет</th>
            <th>Кафедра</th>
          </tr>
        </thead>
        <tbody>
          {data.map((university) => (
            <React.Fragment key={university.id}>
              <tr>
                <th colSpan={3}>{university.name}</th>
              </tr>
              {university.faculties.map((faculty) => (
                <React.Fragment key={faculty.id}>
                  {faculty.classrooms.map((classroom) => (
                    <tr key={`{faculty.id}-{classroom.id}`}>
                      <td>{classroom.number}</td>
                      <td>{faculty.name}</td>
                      <td></td>
                    </tr>
                  ))}
                  {faculty.departments.map((department) => (
                    <React.Fragment key={department.id}>
                      {department.classrooms.map((classroom) =>
(
                        <tr key={`{faculty.id}-{department.id}-
{classroom.id}`}>
                          <td>{classroom.number}</td>
                          <td>{faculty.name}</td>
                          <td>{department.name}</td>
                        </tr>
                      ))}
                    </React.Fragment>
                  ))}
                </React.Fragment>
              ))}
            </React.Fragment>
          ))}
        </tbody>
      </table>

      <button onClick={() => setIsAddClassroomModalOpen(true)}>
        Добавить аудиторию
      </button>

      {IsAddClassroomModalOpen && (
```

```
        <div className="modal">
          <h3>Добавить аудиторию</h3>

          <div className="form-group">
            <label>Университет:</label>
            <select
              className="styled-select"
              value={selectedUniversityIdToClassroom || ''}
              onChange={(e) =>
setSelectedUniversityIdToClassroom(Number(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Факультет:</label>
            <select
              className="styled-select"
              disabled={!selectedUniversityIdToClassroom}
              value={selectedFacultyIdToClassroom || ''}
              onChange={(e) =>
setSelectedFacultyIdToClassroom(Number(e.target.value))}
            >
              <option value="">Выберите факультет</option>
              {selectedUniversityIdToClassroom && (
                data.find(u => u.id ===
selectedUniversityIdToClassroom)?.faculties.map(faculty => (
                  <option key={faculty.id} value={faculty.id}>
                    {faculty.name}
                  </option>
                )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Кафедра (не обязательное):</label>
            <select
              className="styled-select"
              disabled={!selectedFacultyIdToClassroom}
              value={selectedDepartmentIdToClassroom || ''}
              onChange={(e) =>
setSelectedDepartmentIdToClassroom(Number(e.target.value))}
            >
              <option value="">Выберите кафедру</option>
              {selectedFacultyIdToClassroom && (
```

```
                data.find(u => u.id ===
selectedUniversityIdToClassroom)
                  ?.faculties.find(f => f.id ===
selectedFacultyIdToClassroom)
                  ?.departments.map(department => (
                    <option key={department.id}
value={department.id}>
                      {department.name}
                    </option>
                  )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Номер:</label>
            <input
              type="text"
              className="input-field"
              value={newClassroomNumber}
              onChange={(e) =>
setNewClassroomNumber(e.target.value.trim())}
            />
          </div>

          <div className="form-group">
            <label>Вместимость:</label>
            <input
              type="number"
              className="input-field"
              value={newClassroomCapacity}
              min="15"
              onChange={(e) =>
setNewClassroomCapacity(parseInt(e.target.value))}
            />
          </div>

          <div className="button-container">
            <button
              onClick={addClassroom}
              disabled={
                !selectedUniversityIdToClassroom ||
                !selectedFacultyIdToClassroom ||
                !newClassroomNumber
              }
            >
              Добавить
            </button>
            <button
              onClick={() => setIsAddClassroomModalOpen(false)}
            >
              Отмена
            </button>
```

```
          </div>
        </div>
      )}
    </>
  )
}
```

Листинг Б.13 – Файл frontend/src/components/Curriculum.tsx

```
// Curriculum - учебный план
import React, { useState } from 'react';
import { CurriculumType, SubjectType, FlowType, LecturerType,
GroupType } from '../types';
import { addCurriculumLesson } from '../api';

// Количество часов за семестр
const HOURS = [72, 108, 144];

interface CurriculumProps {
  data: CurriculumType[];
  setData:
React.Dispatch<React.SetStateAction<CurriculumType[]>>;
  subjects: SubjectType[],
  allGroups: GroupType[],
  flows: FlowType[],
  lecturers: LecturerType[],
}

export const Curriculum = ({ data, setData, subjects, allGroups,
flows, lecturers }: CurriculumProps) => {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [selectedSubject, setSelectedSubject] = useState("");
  const [selectedGroup, setSelectedGroup] = useState("");
  const [selectedHours, setSelectedHours] = useState(0);
  const [selectedAttestation, setSelectedAttestation] =
useState("");
  const [selectedLecturer, setSelectedLecturer] =
useState<string>("");
  const [selectedSecondLecturer, setSelectedSecondLecturer] =
useState<string | null>(null);
  const [selectedFlow, setSelectedFlow] = useState<number |
null>(null);

  const handleAddCurriculum = async () => {
    const selectedSubjectData = subjects.find(u => u.name ===
selectedSubject);
    if (!selectedSubjectData) return;
    const primaryTeacher = lecturers.find(u => u.full_name ===
selectedLecturer);
    if (!primaryTeacher) return;
    const secondaryTeacher = lecturers.find(u => u.full_name ===
selectedSecondLecturer);
    const groupData = allGroups.find(u => u.name ===
selectedGroup);
```

```
    const newId = await addCurriculumLesson(
      selectedSubjectData.id,
      selectedHours,
      primaryTeacher.id,
      secondaryTeacher?.id,
      groupData ? groupData.id : undefined,
      selectedFlow ? selectedFlow : undefined
    );
    let selectedGroups: string[] = [];
    if (groupData) selectedGroups = [groupData.name];
    else {
      const flow = flows.find(u => u.id == selectedFlow);
      if (flow)
        selectedGroups = flow.groups;
    }
    setData([
      ...data,
      {
        id: newId,
        subject: selectedSubject,
        groups: selectedGroups,
        hours: selectedHours,
        primary_teacher: selectedLecturer,
        secondary_teacher: selectedSecondLecturer?
selectedSecondLecturer : undefined,
      }
    ]);
    setIsModalOpen(false);
    setSelectedSubject("");
    setSelectedHours(0);
    setSelectedAttestation("");
    setSelectedFlow(null);
    setSelectedLecturer("");
    setSelectedSecondLecturer(null);
  };

  return (
    <>
      <h2>Учебный план</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
      <thead>
        <tr>
          <th>Предмет</th>
          <th>Группы</th>
          <th>Часы</th>
          <th>Преподаватели</th>
        </tr>
      </thead>
      <tbody>
        {data.map((Curriculum) => (
          <React.Fragment key={Curriculum.id}>
            <tr>
              <td>{Curriculum.subject}</td>
```

```
              <td>{Curriculum.groups.join(", ")}</td>
              <td>{Curriculum.hours}</td>
              <td>{Curriculum.secondary_teacher?
Curriculum.primary_teacher + ', ' +
Curriculum.secondary_teacher : Curriculum.primary_teacher}</td>
            </tr>
          </React.Fragment>
        ))}
      </tbody>
      </table>

      <div>
        <button onClick={() => setIsModalOpen(true)}>
          Добавить
        </button>
      </div>

      {isModalOpen && (
        <div className="modal">
          <select
            className="styled-select"
            value={selectedSubject || ""}
            onChange={(e) => setSelectedSubject(e.target.value)}
          >
            <option value="">Выберите предмет</option>
            {subjects.map(subject => (
              <option key={subject.name} value={subject.name}>
                {subject.name}
              </option>
            ))}
          </select>

          <div>
            <select
              className="styled-select"
              value={selectedGroup || ""}
              onChange={(e) =>
{setSelectedGroup(e.target.value); setSelectedFlow(null);}}
            >
              <option value="">Выберите группу</option>
              {allGroups.map(group => (
                <option key={group.name} value={group.name}>
                  {group.name}
                </option>
              ))}
            </select>

          </div>

          {/* Выбор потока для добавления всех групп */}
          <div>
            <select
              className="styled-select"
```

```
              value={selectedFlow ?? ''}
              onChange={(e) =>
{setSelectedFlow(Number(e.target.value) || null);
setSelectedGroup("");}}
            >
              <option value="">Выберите поток</option>
              {flows.map(flow => (
                <option key={flow.id} value={flow.id}>
                  Поток {flow.id} (Группы: {flow.groups.join(",
")})
                </option>
              ))}
            </select>
          </div>

          <select
            className="styled-select"
            value={selectedHours}
            onChange={(e) =>
setSelectedHours(parseInt(e.target.value))}
          >
            <option value="0">Выберите число часов (за
семестр)</option>
            {HOURS.map(hour => (
              <option key={hour} value={hour}>
                {hour}
              </option>
            ))}
          </select>

          <div className="form-group">
            <label
htmlFor="firstLecturer">Преподаватель:</label>
            <input
              id="firstLecturer"
              type="text"
              value={selectedLecturer}
              onChange={(e) =>
setSelectedLecturer(e.target.value)}
              list="lecturers"
              placeholder="Выберите преподавателя"
              style={{
                padding: '8px',
                width: '300px',
                marginRight: '8px',
                border: '1px solid #ccc'
              }}
              className="form-control"
            />
            <datalist id="lecturers">
              {lecturers.map(lecturer => (
                <option key={lecturer.full_name}
value={lecturer.full_name} />
```

```
              ))}
            </datalist>
          </div>

          <div className="form-group" style={{ marginTop:
'1rem', marginBottom: '1rem' }}>
            <label htmlFor="firstLecturer">
              Второй преподаватель (необязательно):
            </label>
            <input
              type="text"
              value={selectedSecondLecturer || ''}
              onChange={(e) => {
                const value = e.target.value;
                setSelectedSecondLecturer(value === '' ? null :
value);
              }}
              list="secondLecturers"
              placeholder="Выберите преподавателя"
              style={{
                padding: '8px',
                width: '300px',
                marginRight: '8px',
                border: '1px solid #ccc'
              }}
              className="form-control"
            />
            <datalist id="secondLecturers">
              {lecturers.map(lecturer => (
                <option key={lecturer.full_name}
value={lecturer.full_name} />
              ))}
            </datalist>
          </div>
          <button
            onClick={handleAddCurriculum}
            disabled={
              !selectedSubject ||
              !selectedHours ||
              !selectedAttestation ||
              !lecturers.map(l =>
l.full_name).includes(selectedLecturer) ||
                (selectedSecondLecturer !== null
&& !lecturers.map(l =>
l.full_name).includes(selectedSecondLecturer)) ||
                selectedSecondLecturer === selectedLecturer
            }
          >
            Добавить
          </button>
          <button onClick={() => setIsModalOpen(false)}>
            Отмена
          </button>
```

```
        </div>
      ))}
    </>
  )
}
```

Листинг Б.14 – Файл frontend/src/components/DataComponent.tsx

```
import { useMemo } from 'react';
import { UniversityTable } from './UniversityTable.tsx';
import { LecturersTable } from './LecturersTable.tsx';
import { ClassroomsTable } from './ClassroomsTable.tsx';
import { SubjectsTable } from './SubjectsTable.tsx';
import { FlowsTable } from './flowsTable.tsx';
import { Curriculum } from './Curriculum.tsx';
import { UniversityType, SubjectType, FlowType, CurriculumType,
LecturerType, GroupType } from '../types';

export interface DataProps {
  universityData: UniversityType[];
  setUniversityData:
React.Dispatch<React.SetStateAction<UniversityType[]>>;
  subjects: SubjectType[];
  setSubjects:
React.Dispatch<React.SetStateAction<SubjectType[]>>;
  flows: FlowType[];
  setFlows: React.Dispatch<React.SetStateAction<FlowType[]>>;
  CurriculumData: CurriculumType[];
  setCurriculumData:
React.Dispatch<React.SetStateAction<CurriculumType[]>>;
}

/// В компоненте описаны все элементы на вкладке "Данные"
export const DataComponent = ({
  universityData, setUniversityData,
  subjects, setSubjects,
  flows, setFlows,
  CurriculumData, setCurriculumData,
}: DataProps) => {
  // Получение все названия групп из университетских данных
  const groupNames = useMemo<string[]>(() => {
    const getAllGroups = (data: UniversityType[]): string[] => {
      return data.flatMap(university =>
        university.faculties.flatMap(faculty =>
          faculty.departments.flatMap(department =>
            department.specialities.flatMap(speciality =>
              speciality.groups.map(group => group.name)
            )
          )
        )
      );
    };
    return Array.from(new Set(getAllGroups(universityData)));
  }, [universityData]);
```

```
  //
  const allGroups = useMemo<GroupType[]>(() => {
    const getAllGroups = (data: UniversityType[]): GroupType[]
=> {
      return data.flatMap(university =>
        university.faculties.flatMap(faculty =>
          faculty.departments.flatMap(department =>
            department.specialities.flatMap(speciality =>
speciality.groups)
          )
        )
      );
    };
    return Array.from(new Set(getAllGroups(universityData)));
  }, [universityData]);
  // Все преподаватели
  const allLecturers = useMemo<LecturerType[] >(() => {
    const getAllLecturers = (data: UniversityType[]):
LecturerType[]  => {
      return data.flatMap(university =>
        university.faculties.flatMap(faculty =>
          faculty.departments.flatMap(department =>
department.lecturers)
        )
      );
    };
    return Array.from(new Set(getAllLecturers(universityData)));
  }, [universityData]);

  return (
    <>
      <UniversityTable
        data={universityData}
        setData={setUniversityData}
      />
      <LecturersTable
        data={universityData}
        setData={setUniversityData}
      />
      <ClassroomsTable
        data={universityData}
        setData={setUniversityData}
      />
      <SubjectsTable
        data={subjects}
        setData={setSubjects}
      />
      <FlowsTable
        allGroups={groupNames}
        data={flows}
        setData={setFlows}
      />
      <Curriculum
```

```
        data={CurriculumData}
        setData={setCurriculumData}
        subjects={subjects}
        allGroups={allGroups}
        flows={flows}
        lecturers={allLecturers}
      />
    </>
  );
};
```

Листинг Б.15 – Файл frontend/src/components/flowsTable.tsx

```
import React, { useState, useEffect } from 'react';
import { FlowType } from '../types';
import { createFlow } from '../api';

export interface FlowsTableProps {
  allGroups: string[];
  data: FlowType[];
  setData: React.Dispatch<React.SetStateAction<FlowType[]>>;
}

export const FlowsTable = ({ allGroups, data, setData }:
FlowsTableProps) => {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [flows, setFlows] = useState<string[]>([]);
  // Вспомогательные значения
  const [inputValue, setInputValue] = useState('');
  const [filteredGroups, setFilteredGroups] =
useState<string[]>([]);
  const [showSuggestions, setShowSuggestions] = useState(false);

  // Фильтрация подсказок при вводе
  useEffect(() => {
    const filtered = allGroups.filter(name =>
      name.toLowerCase().includes(inputValue.toLowerCase())
    );
    setFilteredGroups(filtered);
  }, [inputValue, allGroups]);

  // Добавление группы в flows
  const handleAdd = () => {
    if (inputValue && allGroups.includes(inputValue)) {
      setFlows(prevFlows => [...prevFlows, inputValue]);
      setInputValue('');
      setShowSuggestions(false);
    }
  };

  // Удаление группы из flows
  const handleDelete = (index: number) => {
    setFlows(prevFlows => prevFlows.filter((_, i) => i !==
index));
```

```
  };

  const addFlow = async () => {
    const newId = await createFlow(flows);
    setData([...data, { id: newId, groups: flows }]);
    setFlows([]);
    setInputValue('');
    setFilteredGroups([]);
    setShowSuggestions(false);
    setIsModalOpen(false);
  };

  return (
    <>
      <h2>Потоки</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
      <thead>
      </thead>
      <tbody>
        {data.map((flow) => (
          <React.Fragment key={flow.id}>
            <tr>
              <th>{flow.groups.join(", ")}</th>
            </tr>
          </React.Fragment>
        ))}
      </tbody>
      </table>

      <div>
        <button onClick={() => setIsModalOpen(true)}>
          Добавить поток
        </button>
      </div>

      {isModalOpen && (
        <div className="modal">
          <div style={{ position: 'relative', marginBottom:
'1rem' }}>
            <input
              type="text"
              value={inputValue}
              onChange={(e) => {
                setInputValue(e.target.value);
                setShowSuggestions(true);
              }}
              placeholder="Выберите группу"
              style={{
                padding: '8px',
                width: '300px',
                marginBottom: '5px',
                border: '1px solid #ccc'
              }}
```

```
            />

            {showSuggestions && filteredGroups.length > 0 && (
              <div
                style={{
                  position: 'absolute',
                  width: '100%',
                  border: '1px solid #ccc',
                  backgroundColor: 'white',
                  maxHeight: '150px',
                  overflowY: 'auto',
                  marginTop: '2px',
                  borderRadius: '4px'
                }}
              >
                {filteredGroups.map((group, index) => (
                  <div
                    key={index}
                    onClick={() => {
                      setInputValue(group);
                      setFilteredGroups([]);
                      setShowSuggestions(false);
                    }}
                    style={{
                      padding: '8px',
                      cursor: 'pointer',
                      borderBottom: '1px solid #eee',
                    }}
                    className="suggestion"
                  >
                    {group}
                  </div>
                ))}
              </div>
            )}

            <button
              onClick={handleAdd}
              disabled={!inputValue
|| !allGroups.includes(inputValue)}
              style={{
                padding: '8px 16px',
                marginLeft: '8px',
                backgroundColor: '#007bff',
                color: 'white',
                border: 'none',
                cursor: 'pointer',
              }}
              className={`add-button ${!inputValue
|| !allGroups.includes(inputValue) ? 'disabled' : ''}`}
            >
              Добавить
            </button>
```

```
            </div>

            <ul style={{ listStyleType: 'none', padding: 0 }}>
              {flows.map((flow, index) => (
                <li
                  key={index}
                  style={{
                    display: 'flex',
                    justifyContent: 'space-between',
                    alignItems: 'center',
                    padding: '8px',
                    marginBottom: '4px',
                    border: '1px solid #eee',
                    borderRadius: '4px'
                  }}
                >
                  <span>{flow}</span>
                  <button
                    onClick={() => handleDelete(index)}
                    style={{
                      border: 'none',
                      background: 'transparent',
                      color: 'red',
                      cursor: 'pointer',
                      marginLeft: '8px'
                    }}
                  >
                    ×
                  </button>
                </li>
              ))}
            </ul>
            <button
              onClick={addFlow}
              disabled={flows.length === 0}
            >
              Добавить
            </button>
            <button onClick={() => setIsModalOpen(false)}>
              Отмена
            </button>
          </div>
        )}
      </>
    )
}
```

Листинг Б.16 – Файл frontend/src/components/LecturersTable.tsx

```
import React, { useState, useEffect } from 'react';
import { UniversityTableProps } from '../types';
import { createTeacher } from '../api';
```

```
export const LecturersTable = ({ data, setData }:
UniversityTableProps) => {
  const [isAddLecturerModalOpen, setIsAddLecturerModalOpen] =
useState(false);
  const [selectedUniversityIdToLecturer,
setSelectedUniversityIdToLecturer] =
useState<number|null>(null);
  const [selectedFacultyIdToLecturer,
setSelectedFacultyIdToLecturer] = useState<number|null>(null);
  const [selectedDepartmentIdToLecturer,
setSelectedDepartmentIdToLecturer] =
useState<number|null>(null);
  const [newLecturerName, setNewLecturerName] = useState("");

  useEffect(() => {
    setSelectedFacultyIdToLecturer(null);
    setSelectedDepartmentIdToLecturer(null);
  }, [selectedUniversityIdToLecturer]);

  useEffect(() => {
    setSelectedDepartmentIdToLecturer(null);
  }, [selectedFacultyIdToLecturer]);

  const addLecturer = async () => {
    if (
      !selectedUniversityIdToLecturer ||
      !selectedFacultyIdToLecturer ||
      !selectedDepartmentIdToLecturer ||
      !newLecturerName
    ) return;

    const selectedUniversity = data.find(u => u.id ===
selectedUniversityIdToLecturer);
    const selectedFaculty = selectedUniversity?.faculties.find(f
=> f.id === selectedFacultyIdToLecturer);
    const selectedDepartment =
selectedFaculty?.departments.find(d => d.id ===
selectedDepartmentIdToLecturer);
    if (!selectedDepartment) return;

    const newId = await createTeacher(selectedDepartment.id,
newLecturerName);
    setData(prevData => prevData.map(university => {
      if (university.id === selectedUniversityIdToLecturer) {
        return {
          ...university,
          faculties: university.faculties.map(faculty => {
            if (faculty.id === selectedFacultyIdToLecturer) {
              return {
                ...faculty,
                departments: faculty.departments.map(department
=> {
```

```
                  if (department.id ===
selectedDepartmentIdToLecturer) {
                    return {
                      ...department,
                      lecturers: [
                        ...department.lecturers,
                        { id: newId, full_name:
newLecturerName }
                      ]
                    };
                  }
                  return department;
                })
              };
            }
            return faculty;
          })
        };
      }
      return university;
    }));

    setIsAddLecturerModalOpen(false);
    setSelectedUniversityIdToLecturer(null);
    setSelectedFacultyIdToLecturer(null);
    setSelectedDepartmentIdToLecturer(null);
    setNewLecturerName("");
  };

  return (
    <>
      <h2>Профессорско-преподавательский состав</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
        <thead>
          <tr>
            <th>Факультет</th>
            <th>Кафедра</th>
            <th>ФИО</th>
          </tr>
        </thead>
        <tbody>
          {data.map((university) => (
            <React.Fragment key={university.id}>
              <tr>
                <th colSpan={3}>{university.name}</th>
              </tr>
              {university.faculties.map((faculty) => (
                <React.Fragment key={faculty.id}>
                  {faculty.departments.map((department) => (
                    <React.Fragment key={department.id}>
                      {department.lecturers.map((lecturer) => (
                        <tr key={lecturer.id}>
                          <td>{faculty.name}</td>
```

```
                          <td>{department.name}</td>
                          <td>{lecturer.full_name}</td>
                        </tr>
                      ))}
                    </React.Fragment>
                  ))}
                </React.Fragment>
              ))}
            </React.Fragment>
          ))}
        </tbody>
      </table>

      <button onClick={() => setIsAddLecturerModalOpen(true)}>
        Добавить преподавателя
      </button>

      {isAddLecturerModalOpen && (
        <div className="modal">
          <h3>Добавить преподавателя</h3>

          <div className="form-group">
            <label>Университет:</label>
            <select
              className="styled-select"
              value={selectedUniversityIdToLecturer || ''}
              onChange={(e) =>
setSelectedUniversityIdToLecturer(Number(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Факультет:</label>
            <select
              className="styled-select"
              disabled={!selectedUniversityIdToLecturer}
              value={selectedFacultyIdToLecturer || ''}
              onChange={(e) =>
setSelectedFacultyIdToLecturer(Number(e.target.value))}
            >
              <option value="">Выберите факультет</option>
              {selectedUniversityIdToLecturer && (
                data.find(u => u.id ===
selectedUniversityIdToLecturer)?.faculties.map(faculty => (
                  <option key={faculty.id} value={faculty.id}>
```

```
                    {faculty.name}
                  </option>
                )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Кафедра:</label>
            <select
              className="styled-select"
              disabled={!selectedFacultyIdToLecturer}
              value={selectedDepartmentIdToLecturer || ''}
              onChange={(e) =>
setSelectedDepartmentIdToLecturer(Number(e.target.value))}
            >
              <option value="">Выберите кафедру</option>
              {selectedFacultyIdToLecturer && (
                data.find(u => u.id ===
selectedUniversityIdToLecturer)
                  ?.faculties.find(f => f.id ===
selectedFacultyIdToLecturer)
                  ?.departments.map(department => (
                    <option key={department.id}
value={department.id}>
                      {department.name}
                    </option>
                  )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>ФИО:</label>
            <input
              type="text"
              className="input-field"
              value={newLecturerName}
              onChange={(e) =>
setNewLecturerName(e.target.value)}
            />
          </div>

          <div className="button-container">
            <button
              onClick={addLecturer}
              disabled={
                !selectedUniversityIdToLecturer ||
                !selectedFacultyIdToLecturer ||
                !selectedDepartmentIdToLecturer ||
                !newLecturerName.trim()
              }
            >
```

```
              Добавить
            </button>
            <button
              onClick={() => setIsAddLecturerModalOpen(false)}
            >
              Отмена
            </button>
          </div>
        </div>
      )}
    </>
  )
}
```

Листинг Б.17 – Файл frontend/src/components/ScheduleComponent.tsx

```
import React, { useState } from 'react';
import {
  UniversityType, SubjectType, FlowType, CurriculumType,
GroupType,
  ScheduleData, DayNumber, PairNumber, WeekNumber,
ClassroomType, LessonType
} from '../types';
import {
  addScheduleLesson, fetchRemoveSchedule, fetchEditSchedule,
  autoShedule, findCollisions
} from '../api';


export interface DataProps {
  universityData: UniversityType[];
  setUniversityData:
React.Dispatch<React.SetStateAction<UniversityType[]>>;
  subjects: SubjectType[];
  setSubjects:
React.Dispatch<React.SetStateAction<SubjectType[]>>;
  flows: FlowType[];
  setFlows: React.Dispatch<React.SetStateAction<FlowType[]>>;
  CurriculumData: CurriculumType[];
  setCurriculumData:
React.Dispatch<React.SetStateAction<CurriculumType[]>>;
  activeUniversity: string | null;
  setActiveUniversity:
React.Dispatch<React.SetStateAction<string | null>>;
  activeFaculty: string | null;
  setActiveFaculty: React.Dispatch<React.SetStateAction<string |
null>>;
  activeDepartment: string | null;
  setActiveDepartment:
React.Dispatch<React.SetStateAction<string | null>>;
  groups: GroupType[];  // Текущие группы для выбранной кафедры
  setGroups: React.Dispatch<React.SetStateAction<GroupType[]>>;
  currentWeek: number;
  setCurrentWeek: React.Dispatch<React.SetStateAction<number>>;
```

```
  schedule: ScheduleData;
  setSchedule:
React.Dispatch<React.SetStateAction<ScheduleData>>;
  allClassrooms: ClassroomType[];
}

const daysOfWeek = ['Понедельник', 'Вторник', 'Среда',
'Четверг', 'Пятница', 'Суббота'];
const pairs = [
  '1 пара [08:00 - 09:35]',
  '2 пара [09:50 - 11:25]',
  '3 пара [11:40 - 13:15]',
  '4 пара [13:45 - 15:20]',
  '5 пара [15:35 - 17:10]',
  '6 пара [17:25 - 19:00]',
  '7 пара [19:15 - 20:50]',
  '8 пара [21:05 - 22:35]'
];

type rowType = {
  day: string;
  pair: string;
}

export const ScheduleComponent = ({
  universityData, activeUniversity, setActiveUniversity,
activeFaculty,
  setActiveFaculty, activeDepartment, setActiveDepartment,
groups, setGroups,
  currentWeek, setCurrentWeek, schedule, setSchedule,
allClassrooms, CurriculumData
}: DataProps) => {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [currentDay, setCurrentDay] = useState(1);
  const [currentPair, setCurrentPair] = useState(1);
  const [selectedClassroom, setSelectedClassroom] = useState(0);
  const [selectedType, setSelectedType] =
useState<LessonType>("лабораторное");
  const [selectedCurriculum, setSelectedCurriculum] =
useState(0);
  const [selectedGroup, setSelectedGroup] = useState("");
  // Добавление или изменение ячейки расписание. true -
добавление, false - изменение
  const [changeOrNew, setChangeOrNew] = useState(true);
  const [changedId, setChangedId] = useState(0);  // ID какой
ячейки меняем

  const rows: rowType[] = daysOfWeek.flatMap(day =>
    pairs.map(pair => ({ day, pair }))
  );

  const handleTabClick = (university: string) => {
    setActiveUniversity(university);
```

```
    setActiveFaculty(null);
    setActiveDepartment(null);
  };
  const handleSecondTabClick = (faculty: string) => {
    setActiveFaculty(faculty);
    setActiveDepartment(null);
  };
  const handleThirdTabClick = (department: string) => {
    setActiveDepartment(department);
    let currentGroups =
getGroupsForActiveDepartment(department);
    setGroups(currentGroups);
  };

  // Функция для получения групп по выбранной кафедре
  const getGroupsForActiveDepartment = (departmentName: string):
GroupType[] => {
    if (!activeUniversity || !activeFaculty) return [];

    const university = universityData.find(u => u.name ===
activeUniversity);
    if (!university) return [];
    const faculty = university.faculties.find(f => f.name ===
activeFaculty);
    if (!faculty) return [];
    if (departmentName === 'all') {
      return faculty.departments.flatMap(department =>
        department.specialities.flatMap(spec => spec.groups)
      );
    }
    const department = faculty.departments.find(d =>
d.short_name === departmentName);
    if (!department) return [];

    return department.specialities.flatMap(spec => spec.groups);
  };

  const handleAddSchedule = async () => {
    const newId = await addScheduleLesson(
      currentWeek,
      currentDay,
      currentPair,
      selectedClassroom,
      selectedCurriculum,
      selectedType
    );
    setSchedule(prev => ({
      data: {
        ...prev.data,
        [currentWeek]: {
          ...prev.data[currentWeek as WeekNumber],
          [currentDay]: {
```

```
            ...prev.data[currentWeek as WeekNumber]?.[currentDay
as DayNumber],
            [currentPair]: {
              ...prev.data[currentWeek as
WeekNumber]?.[currentDay as DayNumber]?.[currentPair as
PairNumber],
              [selectedGroup]: {
                id: newId,
                lesson_type: selectedType,
                subject: CurriculumData.find(u => u.id ===
selectedCurriculum)?.subject,
                teachers: "Иванова Т.П.",
                classroom: allClassrooms.find(u => u.id ===
selectedClassroom)?.number
              }
            }
          }
        }
      }
    }));
    setIsModalOpen(false);
  };

  // Изменение ячейки расписания
  const handleChangechedule = async () => {
    await fetchEditSchedule(changedId, selectedClassroom,
selectedCurriculum, selectedType);
    setSchedule(prev => ({
      data: {
        ...prev.data,
        ...([1, 2] as WeekNumber[]).reduce((weeks, week) => {
          const weekData = prev.data[week];
          if (!weekData) return weeks;

          return {
            ...weeks,
            [week]: {
              ...weekData,
              ...([1, 2, 3, 4, 5, 6] as
DayNumber[]).reduce((days, day) => {
                const dayData = weekData[day];
                if (!dayData) return days;

                return {
                  ...days,
                  [day]: {
                    ...dayData,
                    ...([1, 2, 3, 4, 5, 6, 7, 8] as
PairNumber[]).reduce((pairs, pair) => {
                      const pairData = dayData[pair];
                      if (!pairData) return pairs;

                      let updated = false;
```

```
                        const updatedGroups =
Object.entries(pairData).reduce((groups, [group, cell]) => {
                          if (cell.id === changedId) {
                            updated = true;
                            return {
                              ...groups,
                              [group]: {
                                ...cell,
                                lesson_type: selectedType,
                                subject: CurriculumData.find(u =>
u.id === selectedCurriculum)?.subject || cell.subject,
                                classroom: allClassrooms.find(u =>
u.id === selectedClassroom)?.number || cell.classroom
                              }
                            };
                          }
                          return {...groups, [group]: cell};
                        }, {});

                        return updated ? {...pairs, [pair]:
updatedGroups} : pairs;
                      }, {})
                    }
                  };
                }, {})
              }
            };
          }, {})
        }
      }));
      setIsModalOpen(false);
    };

    // Удаление ячейки расписания
    const handleRemoveCell = async (removedCellId: number) => {
      await fetchRemoveSchedule(removedCellId);
      setSchedule(prev => ({
        data: {
            ...prev.data,
            ...([1, 2] as WeekNumber[]).reduce((weeksAcc, week) =>
({
                ...weeksAcc,
                [week]: prev.data[week] && {
                    ...prev.data[week],
                    ...([1, 2, 3, 4, 5, 6] as
DayNumber[]).reduce((daysAcc, day) => ({
                        ...daysAcc,
                        [day]: prev.data[week]?.[day] && {
                            ...prev.data[week]?.[day],
                            ...([1, 2, 3, 4, 5, 6, 7, 8] as
PairNumber[]).reduce((pairsAcc, pair) => ({
                                ...pairsAcc,
```

```
                                    [pair]:
Object.entries(prev.data[week]?.[day]?.[pair] || {}).reduce(
                                        (groupsAcc, [group, cellData])
=>
                                            cellData.id ===
removedCellId
                                                ? groupsAcc
                                                : { ...groupsAcc,
[group]: cellData },
                                        {}
                                    )
                                }), {})
                            }
                        }), {})
                }
            }), {})
        }
    }));
  };

  return (
    <>
    <div>
      <div style={{ position: 'absolute', right: '25px' }}>
        <div>
          <button
            className={currentWeek === 1 ? 'active' : ''}
            onClick={() => setCurrentWeek(1)}
            style={{ marginRight: 5 }}
          >
            1 неделя
          </button>
          <button
            className={currentWeek === 2 ? 'active' : ''}
            onClick={() => setCurrentWeek(2)}
          >
            2 неделя
          </button>
        </div>

        <div>
          <button onClick={autoShedule}>
            Построить расписание (автоматически)
          </button>
        </div>
        <div>
          <button onClick={findCollisions}>
            Найти коллизии (автоматически)
          </button>
        </div>
      </div>
      <div>
        {universityData.map((university) => (
```

```
          <button
            className={activeUniversity === university.name ?
'active' : ''}
            onClick={() => handleTabClick(university.name)}
          >
            {university.name}
          </button>
        ))}
      </div>
      <div>
        <div>
          {universityData.map((university) => (
            (university.name === activeUniversity) &&
            <React.Fragment key={university.id}>
              <div>
                {university.faculties.map((faculty) => (
                  <button
                    className={activeFaculty === faculty.name ?
'active' : ''}
                    onClick={() =>
handleSecondTabClick(faculty.name)}
                  >
                    {faculty.name}
                  </button>
                  ))}
              </div>
              {university.faculties.map((faculty) => (
                <div
                  style={{
                    visibility: activeFaculty === faculty.name ?
'visible' : 'hidden',
                    height: activeFaculty === faculty.name ?
'auto' : '0'
                  }}
                >
                  <div>
                    {university.faculties.map((faculty) => (
                      (activeFaculty === faculty.name) ? (
                        <React.Fragment key={faculty.id}>
                          <button
                            className={activeDepartment ===
'all' ? 'active' : ''}
                            onClick={() =>
handleThirdTabClick('all')}
                          >
                            Все
                          </button>
                          {faculty.departments.map((department)
=> (
                            <button
                              className={activeDepartment ===
department.short_name ? 'active' : ''}
```

```
                                onClick={() =>
handleThirdTabClick(department.short_name)}
                              >
                                {department.short_name}
                              </button>
                            ))}
                          </React.Fragment>
                        ) : null
                        ))}
                    </div>
                  </div>
                ))}
              </React.Fragment>
            ))}
          </div>
        </div>
      </div>
      {(activeDepartment) &&
        <table border={1} cellPadding="5" cellSpacing="0"
style={{ marginTop: '15px' }}>
          <thead>
            <tr>
              <th></th>
              <th></th>
              {groups.map(group => (
                <th key={group.id}>
                  {`${group.name} (${group.course})`}
                </th>
              ))}
            </tr>
          </thead>
          <tbody>
            {rows.map(row => (
              <tr key={`${row.day}-${row.pair}`}>
                {(row.pair === pairs[0]) &&
                  <td rowSpan={pairs.length}>
                    {row.day}
                  </td>
                }
                <td>{row.pair}</td>
                {groups.map(group => {
                  const dayNumber = (daysOfWeek.indexOf(row.day) +
1) as DayNumber;
                  const pairNumber = (pairs.indexOf(row.pair) + 1)
as PairNumber;
                  const cellData = schedule.data?.[currentWeek as
WeekNumber]?.[dayNumber]?.[pairNumber]?.[group.name];

                  return (
                    <td key={group.id} style={{ textAlign:
'center', verticalAlign: 'middle' }}>
                      {cellData ? (
                        <div style={{ position: 'relative' }}>
```

```
                        <div style={{
                          position: 'absolute',
                          right: '4px',
                          display: 'flex',
                          zIndex: 10
                        }}>
                          <button
                            onClick={ () => {
                              setIsModalOpen(true);
                              setCurrentDay(dayNumber);
                              setCurrentPair(pairNumber);
                              setSelectedGroup(group.name);
                              const classroomData =
allClassrooms.find(u => u.number === cellData.classroom)?.id;
                              if (classroomData)
setSelectedClassroom(classroomData);
                              const curriculumData =
CurriculumData.find(u => u.subject === cellData.subject)?.id;
                              if(curriculumData)
setSelectedCurriculum(curriculumData);

setSelectedType(cellData.lesson_type);
                              setChangeOrNew(false);  //
Изменение
                              setChangedId(cellData.id);
                            } }
                            style={{ width: '20px', height:
'20px', background: 'transparent', padding: 0, paddingRight:
'15px' }}
                          >✎</button>
                          <button
                            onClick={ () =>
handleRemoveCell(cellData.id) }
                            style={{ width: '20px', height:
'20px', background: 'transparent', padding: 0 }}
                          >✕</button>
                        </div>

<div><strong>{cellData.subject}</strong></div>
                        <div>{cellData.teachers}</div>
                        <div>{cellData.lesson_type}</div>
                        <div>{cellData.classroom}</div>
                      </div>
                    ) : (
                      <button onClick={
                        () => {
                          setIsModalOpen(true);
                          setCurrentDay(dayNumber);
                          setCurrentPair(pairNumber);
                          setSelectedGroup(group.name);
                          setChangeOrNew(true); // Добавление
                        }
                      }>+</button>
```

```
                )}
              </td>
            );
          })}
        </tr>
      ))}
    </tbody>
  </table>
}
{isModalOpen && (
  <div className="modal">
    <div className="form-group">
      <label htmlFor="week">Неделя:</label>
      <input
          id="week"
          type="text"
          value={currentWeek}
          style={{
            padding: '8px',
            width: '300px',
            marginRight: '8px',
            border: '1px solid #ccc'
          }}
          className="form-control"
          disabled
      />
    </div>
    <div className="form-group">
      <label htmlFor="day">День:</label>
      <input
          id="day"
          type="text"
          value={daysOfWeek[currentDay-1]}
          style={{
            padding: '8px',
            width: '300px',
            marginRight: '8px',
            border: '1px solid #ccc'
          }}
          className="form-control"
          disabled
      />
    </div>
    <div className="form-group">
      <label htmlFor="pair">Пара:</label>
      <input
          id="pair"
          type="text"
          value={pairs[currentPair-1]}
          style={{
            padding: '8px',
            width: '300px',
            marginRight: '8px',
```

```
                  border: '1px solid #ccc'
                }}
                className="form-control"
                disabled
              />
            </div>
            <div className="form-group">
              <label htmlFor="classroom">Аудитория:</label>
              <select
                  id="classroom"
                  value={selectedClassroom}
                  className="styled-select"
                  onChange={(e) =>
setSelectedClassroom(parseInt(e.target.value))}
              >
                <option value="0">Выберите аудиторию</option>
                {allClassrooms.map(classroom => (
                  <option key={classroom.id} value={classroom.id}>
                    {classroom.number}
                  </option>
                ))}
              </select>
            </div>
            <div className="form-group">
              <label htmlFor="classroom">Занятие:</label>
              <select
                  id="classroom"
                  value={selectedCurriculum}
                  className="styled-select"
                  onChange={(e) =>
setSelectedCurriculum(parseInt(e.target.value))}
              >
                <option value="0">Выберите занятие</option>
                {CurriculumData.filter(u =>
u.groups.includes(selectedGroup)).map(curriculum => (
                  <option key={curriculum.id} value={curriculum.id}>
                    {curriculum.subject}
{curriculum.primary_teacher} {curriculum.secondary_teacher}
                  </option>
                ))}
              </select>
            </div>
            <div className="form-group">
              <label htmlFor="classroom">Тип занятия:</label>
              <select
                  id="classroom"
                  value={selectedType}
                  className="styled-select"
                  onChange={(e) => setSelectedType(e.target.value as
LessonType)}
              >
                <option key="лекционное"
value="лекционное">Лекционное</option>
```

```
            <option key="лабораторное"
value="лабораторное">Лабораторное</option>
          </select>
        </div>

        <button
            onClick={() => {
              if (changeOrNew) handleAddSchedule();
              else handleChangechedule();
            }}
            disabled={selectedClassroom == 0 ||
selectedCurriculum == 0}
        >
          {changeOrNew ? 'Добавить' : 'Изменить'}
        </button>
        <button onClick={() => {
          setIsModalOpen(false);
          setSelectedType("лабораторное");
          setSelectedClassroom(0);
          setSelectedCurriculum(0);
        }}>
          Отмена
        </button>
      </div>
    )}
    </>
  );
};
```

Листинг Б.18 – Файл frontend/src/components/SubjectsTable.tsx

```
import React, { useState } from 'react';
import { SubjectType } from '../types';
import { createSubject } from '../api';

export interface SubjectsTableProps {
  data: SubjectType[];
  setData: React.Dispatch<React.SetStateAction<SubjectType[]>>;
}

export const SubjectsTable = ({ data, setData }:
SubjectsTableProps) => {
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [newSubjectName, setNewSubjectName] = useState("");
  const [newSubjectShortName, setNewSubjectShortName] =
useState("");

  const addSubject = async () => {
    const newId = await createSubject(newSubjectName,
newSubjectShortName);
    setData(prevData => [
      ...prevData,
      {
        id: newId,
```

```
        name: newSubjectName,
        short_name: newSubjectShortName,
      }
    ]);
    setIsModalOpen(false);
    setNewSubjectName("");
    setNewSubjectShortName("");
  };

  return (
    <>
      <h2>Предметы</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
      <thead>
        <tr>
          <th>Название</th>
          <th>Сокращение</th>
        </tr>
      </thead>
      <tbody>
        {data.map((subject) => (
          <React.Fragment key={subject.id}>
            <tr>
              <td>{subject.name}</td>
              <td>{subject.short_name}</td>
            </tr>
          </React.Fragment>
        ))}
      </tbody>
      </table>

      <div>
        <button onClick={() => setIsModalOpen(true)}>
          Добавить предмет
        </button>
      </div>

      {isModalOpen && (
        <div className="modal">
          <input
            type="text"
            placeholder="Название предмета"
            value={newSubjectName}
            onChange={(e) => setNewSubjectName(e.target.value)}
            className="input-field"
          />
          <input
            type="text"
            placeholder="Сокращение"
            value={newSubjectShortName}
            onChange={(e) =>
setNewSubjectShortName(e.target.value)}
            className="input-field"
```

```
          />
          <button
            onClick={addSubject}
            disabled={!newSubjectName || !newSubjectShortName}
          >
            Добавить
          </button>
          <button onClick={() => setIsModalOpen(false)}>
            Отмена
          </button>
        </div>
      )}
    </>
  )
}
```

Листинг Б.19 – Файл frontend/src/components/UniversityTable.tsx

```
import React, { useState, useEffect } from 'react';
import { UniversityTableProps } from '../types';
import { createStructuralDivision, createSpeciality,
createGroup } from '../api';

const COURSES = [
  { value: 'Бакалавриат, 1', label: 'Бакалавриат, 1' },
  { value: 'Бакалавриат, 2', label: 'Бакалавриат, 2' },
  { value: 'Бакалавриат, 3', label: 'Бакалавриат, 3' },
  { value: 'Бакалавриат, 4', label: 'Бакалавриат, 4' },
  { value: 'Магистратура, 1', label: 'Магистратура, 1' },
  { value: 'Магистратура, 2', label: 'Магистратура, 2' },
  { value: 'Специалитет, 1', label: 'Специалитет, 1' },
  { value: 'Специалитет, 2', label: 'Специалитет, 2' },
  { value: 'Специалитет, 3', label: 'Специалитет, 3' },
  { value: 'Специалитет, 4', label: 'Специалитет, 4' },
  { value: 'Специалитет, 5', label: 'Специалитет, 5' }
];

export const UniversityTable = ({ data, setData }:
UniversityTableProps) => {
  // Упрощает добавление университетов/факультетов/кафдер/групп
  // Добавление университета
  const [isAddUniversityModalOpen, setIsAddUniversityModalOpen]
= useState(false);
  const [newUniversityName, setNewUniversityName] =
useState("");
  // Добавление факультета
  const [isAddFacultyModalOpen, setIsAddFacultyModalOpen] =
useState(false);
  const [selectedUniversityIdToFaculty,
setSelectedUniversityIdToFaculty] = useState<number|null>(null);
  const [newFacultyName, setNewFacultyName] = useState("");
  // Добавление кафедры
  const [isAddDepartmentModalOpen, setIsAddDepartmentModalOpen]
= useState(false);
```

```
  const [selectedUniversityIdToDepartment,
setSelectedUniversityIdToDepartment] =
useState<number|null>(null);
  const [selectedFacultyIdToDepartment,
setSelectedFacultyIdToDepartment] = useState<number|null>(null);
  const [newDepartmentName, setNewDepartmentName] =
useState('');
  const [newDepartmentShortName, setNewDepartmentShortName] =
useState('');
  // Добавление специальности
  const [isAddSpecialityModalOpen, setIsAddSpecialityModalOpen]
= useState(false);
  const [selectedUniversityIdToSpeciality,
setSelectedUniversityIdToSpeciality] =
useState<number|null>(null);
  const [selectedFacultyIdToSpeciality,
setSelectedFacultyIdToSpeciality] = useState<number|null>(null);
  const [selectedDepartmentIdToSpeciality,
setSelectedDepartmentIdToSpeciality] =
useState<number|null>(null);
  const [newSpecialityName, setNewSpecialityName] =
useState('');
  // Добавление группы
  const [isAddGroupModalOpen, setIsAddGroupModalOpen] =
useState(false);
  const [selectedUniversityIdToGroup,
setSelectedUniversityIdToGroup] = useState<number|null>(null);
  const [selectedFacultyIdToGroup, setSelectedFacultyIdToGroup]
= useState<number|null>(null);
  const [selectedDepartmentIdToGroup,
setSelectedDepartmentIdToGroup] = useState<number|null>(null);
  const [selectedSpecialityIdToGroup,
setSelectedSpecialityIdToGroup] = useState<number|null>(null);
  const [newGroupName, setNewGroupName] = useState('');
  const [selectedCourse, setSelectedCourse] = useState('');
  const [studentsCount, setStudentsCount] = useState(15);

  // Сброс факультета при выборе нового университета
  useEffect(() => {
    setSelectedFacultyIdToDepartment(null);
  }, [selectedUniversityIdToDepartment]);

  useEffect(() => {
    setSelectedFacultyIdToSpeciality(null);
    setSelectedDepartmentIdToSpeciality(null);
  }, [selectedUniversityIdToSpeciality]);

  useEffect(() => {
    setSelectedDepartmentIdToSpeciality(null);
  }, [selectedFacultyIdToSpeciality]);

  useEffect(() => {
    setSelectedFacultyIdToGroup(null);
```

```
    setSelectedDepartmentIdToGroup(null);
    setSelectedSpecialityIdToGroup(null);
  }, [selectedUniversityIdToGroup]);

  useEffect(() => {
    setSelectedDepartmentIdToGroup(null);
    setSelectedSpecialityIdToGroup(null);
  }, [selectedFacultyIdToGroup]);

  useEffect(() => {
    setSelectedSpecialityIdToGroup(null);
  }, [selectedDepartmentIdToGroup]);

  const addUniversity = async () => {
    const newId = await
createStructuralDivision(newUniversityName);
    setData(prevData => [
      ...prevData,
      {
        id: newId,
        name: newUniversityName,
        faculties: []
      }
    ]);
    setIsAddUniversityModalOpen(false);
    setNewUniversityName("");
  };

  const addFaculty = async () => {
    // Находим выбранный университет
    const selectedUniversity = data.find(
      (university) => university.id ===
selectedUniversityIdToFaculty
    );
    if (!selectedUniversity) return;
    // Находим максимальный ID факультета в университете. Если
их нет, то 0
    const newId = await createStructuralDivision(newFacultyName,
selectedUniversity.id);
    // Добавляем факультет
    setData(prevData =>
      prevData.map(university => {
        if (university.id === selectedUniversityIdToFaculty) {
          return {
            ...university,
            faculties: [
              ...university.faculties,
              {
                id: newId,
                name: newFacultyName,
                departments: [],
                classrooms: []
              }
```

```
            ]
          };
        }
        return university;
      })
    );
    setIsAddFacultyModalOpen(false);
    setSelectedUniversityIdToFaculty(null);
    setNewFacultyName("");
  };

  const addDepartment = async () => {
    if (!selectedUniversityIdToDepartment
|| !selectedFacultyIdToDepartment || !newDepartmentName
|| !newDepartmentShortName) return;

    // Находим выбранный факультет
    const selectedUniversity = data.find(u => u.id ===
selectedUniversityIdToDepartment);
    if (!selectedUniversity) return;
    const selectedFaculty = selectedUniversity.faculties.find(f
=> f.id === selectedFacultyIdToDepartment);
    if (!selectedFaculty) return;

    // Добавляем кафедру
    const newId = await
createStructuralDivision(newDepartmentName, selectedFaculty.id,
newDepartmentShortName);
    setData(prevData => prevData.map(university => {
      if (university.id === selectedUniversityIdToDepartment) {
        return {
          ...university,
          faculties: university.faculties.map(faculty => {
            if (faculty.id === selectedFacultyIdToDepartment) {
              return {
                ...faculty,
                departments: [
                  ...faculty.departments,
                  {
                    id: newId,
                    name: newDepartmentName,
                    shortName: newDepartmentShortName,
                    specialities: [],
                    lecturers: [],
                    classrooms: []
                  }
                ]
              };
            }
            return faculty;
          })
        };
      }
```

```
      return university;
    }));

    // Сбрасываем состояние
    setIsAddDepartmentModalOpen(false);
    setSelectedUniversityIdToDepartment(null);
    setSelectedUniversityIdToDepartment(null);
    setNewDepartmentName('');
    setNewDepartmentShortName('');
  };

  const addSpeciality = async () => {
    if (
      !selectedUniversityIdToSpeciality ||
      !selectedFacultyIdToSpeciality ||
      !selectedDepartmentIdToSpeciality ||
      !newSpecialityName
    ) return;

    // Находим выбранную кафедру
    const selectedUniversity = data.find(u => u.id ===
selectedUniversityIdToSpeciality);
    if (!selectedUniversity) return;
    const selectedFaculty = selectedUniversity.faculties.find(f
=> f.id === selectedFacultyIdToSpeciality);
    if (!selectedFaculty) return;
    const selectedDepartment =
selectedFaculty.departments.find(d => d.id ===
selectedDepartmentIdToSpeciality);
    if (!selectedDepartment) return;

    // Добавляем специальность
    const newId = await createSpeciality(selectedDepartment.id,
newSpecialityName);
    setData(prevData => prevData.map(university => {
      if (university.id === selectedUniversityIdToSpeciality) {
        return {
          ...university,
          faculties: university.faculties.map(faculty => {
            if (faculty.id === selectedFacultyIdToSpeciality) {
              return {
                ...faculty,
                departments: faculty.departments.map(department
=> {
                  if (department.id ===
selectedDepartmentIdToSpeciality) {
                    return {
                      ...department,
                      specialities: [
                        ...department.specialities,
                        {
                          id: newId,
                          name: newSpecialityName,
```

```
                        groups: []
                      }
                    ]
                  };
                }
                return department;
              })
            };
          }
          return faculty;
        })
      };
    }
    return university;
  }));

  // Сбрасываем состояние
  setIsAddSpecialityModalOpen(false);
  setSelectedUniversityIdToSpeciality(null);
  setSelectedFacultyIdToSpeciality(null);
  setSelectedDepartmentIdToSpeciality(null);
  setNewSpecialityName('');
};

const addGroup = async () => {
  if (
    !selectedUniversityIdToGroup ||
    !selectedFacultyIdToGroup ||
    !selectedDepartmentIdToGroup ||
    !selectedSpecialityIdToGroup ||
    !newGroupName ||
    !selectedCourse
  ) return;

  // Находим выбранную специальность
  const selectedUniversity = data.find(u => u.id ===
selectedUniversityIdToGroup);
  if (!selectedUniversity) return;
  const selectedFaculty = selectedUniversity.faculties.find(f
=> f.id === selectedFacultyIdToGroup);
  if (!selectedFaculty) return;
  const selectedDepartment =
selectedFaculty.departments.find(d => d.id ===
selectedDepartmentIdToGroup);
  if (!selectedDepartment) return;
  const selectedSpeciality =
selectedDepartment.specialities.find(s => s.id ===
selectedSpecialityIdToGroup);
  if (!selectedSpeciality) return;

  // Добавляем группу
  const newId = await createGroup(selectedSpeciality.id,
newGroupName, selectedCourse, studentsCount)
```

```
    setData(prevData => prevData.map(university => {
      if (university.id === selectedUniversityIdToGroup) {
        return {
          ...university,
          faculties: university.faculties.map(faculty => {
            if (faculty.id === selectedFacultyIdToGroup) {
              return {
                ...faculty,
                departments: faculty.departments.map(department
=> {
                  if (department.id ===
selectedDepartmentIdToGroup) {
                    return {
                      ...department,
                      specialities:
department.specialities.map(speciality => {
                        if (speciality.id ===
selectedSpecialityIdToGroup) {
                          return {
                            ...speciality,
                            groups: [
                              ...speciality.groups,
                              {
                                id: newId,
                                name: newGroupName,
                                course: selectedCourse,
                                studentsCount: studentsCount
                              }
                            ]
                          };
                        }
                        return speciality;
                      })
                    };
                  }
                  return department;
                })
              };
            }
            return faculty;
          })
        };
      }
      return university;
    }));

    // Сбрасываем состояние
    setIsAddGroupModalOpen(false);
    setSelectedUniversityIdToGroup(null);
    setSelectedFacultyIdToGroup(null);
    setSelectedDepartmentIdToGroup(null);
    setSelectedSpecialityIdToGroup(null);
    setNewGroupName('');
```

```
    setSelectedCourse('');
    setStudentsCount(15);
  };

  return (
    <>
      <h2>Структурные подразделения</h2>
      <table border={1} cellPadding="5" cellSpacing="0">
      <thead>
        <tr>
          <th>Факультет</th>
          <th>Кафедра</th>
          <th>Специальность</th>
          <th>Курс</th>
          <th>Группа</th>
          <th>Число студентов</th>
        </tr>
      </thead>
      <tbody>
        {data.map((university) => (
          <React.Fragment key={university.id}>
            <tr>
              <th colSpan={7}>{university.name}</th>
            </tr>

            {university.faculties.map((faculty) => (
              <React.Fragment key={faculty.id}>
                {/* Строка факультета, если нет кафедр */}
                {faculty.departments.length === 0 && (
                  <tr>
                    <td>{faculty.name}</td>
                    <td></td>
                    <td></td>
                    <td></td>
                    <td></td>
                    <td></td>
                  </tr>
                )}

                {faculty.departments.map((department) => (
                  <React.Fragment key={department.id}>
                    {/* Строка кафедры, если нет специальностей
*/}
                    {department.specialities.length === 0 && (
                      <tr>
                        <td>{faculty.name}</td>
                        <td>{department.name}</td>
                        <td></td>
                        <td></td>
                        <td></td>
                        <td></td>
                      </tr>
                    )}
```

```
                    {department.specialities.map((speciality) =>
(
                      <React.Fragment key={speciality.id}>
                        {speciality.groups.length === 0 && (
                          <tr>
                            <td>{faculty.name}</td>
                            <td>{department.name}</td>
                            <td>{speciality.name}</td>
                            <td></td>
                            <td></td>
                            <td></td>
                          </tr>
                        )}

                        {speciality.groups.map((group) => (
                          <tr key={group.id}>
                            <td>{faculty.name}</td>
                            <td>{department.name}</td>
                            <td>{speciality.name}</td>
                            <td>{group.course}</td>
                            <td>{group.name}</td>
                            <td>{group.studentsCount}</td>
                          </tr>
                        ))}
                      </React.Fragment>
                    ))}
                  </React.Fragment>
                ))}
              </React.Fragment>
            ))}
          </tbody>
          </table>

          <div>
            <button onClick={() =>
setIsAddUniversityModalOpen(true)}>
              Добавить университет
            </button>

            <button onClick={() => setIsAddFacultyModalOpen(true)}>
              Добавить факультет
            </button>

            <button onClick={() =>
setIsAddDepartmentModalOpen(true)}>
              Добавить кафедру
            </button>

            <button onClick={() =>
setIsAddSpecialityModalOpen(true)}>
```

```
          Добавить специальность
        </button>

        <button onClick={() => setIsAddGroupModalOpen(true)}>
          Добавить группу
        </button>
      </div>

      {isAddUniversityModalOpen && (
        <div className="modal">
          <input
            type="text"
            placeholder="Название университета"
            value={newUniversityName}
            onChange={(e) =>
setNewUniversityName(e.target.value)}
            className="input-field"
          />
          <button
            onClick={addUniversity}
            disabled={!newUniversityName}
          >
            Добавить
          </button>
          <button onClick={() =>
setIsAddUniversityModalOpen(false)}>
            Отмена
          </button>
        </div>
      )}

      {isAddFacultyModalOpen && (
        <div className="modal">
          <h3>Добавить факультет</h3>

          <div style={{ marginBottom: "15px" }}>
            <select
              className="styled-select"
              value={selectedUniversityIdToFaculty || ""}
              onChange={(e) =>
setSelectedUniversityIdToFaculty(parseInt(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>

            <input
              type="text"
```

```
              placeholder="Название факультета"
              value={newFacultyName}
              onChange={(e) =>
setNewFacultyName(e.target.value)}
              className="input-field"
            />
          </div>

          <div className="button-container">
            <button
              onClick={addFaculty}
              disabled={!selectedUniversityIdToFaculty
|| !newFacultyName}
            >
              Добавить
            </button>
            <button
              onClick={() => setIsAddFacultyModalOpen(false)}
            >
              Отмена
            </button>
          </div>
        </div>
      )}

      {isAddDepartmentModalOpen && (
        <div className="modal">
          <h3>Добавить кафедру</h3>

          <div className="form-group">
            <label>Университет:</label>
            <select
              className="styled-select"
              value={selectedUniversityIdToDepartment || ''}
              onChange={(e) =>
setSelectedUniversityIdToDepartment(parseInt(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Факультет:</label>
            <select
              className="styled-select"
              disabled={!selectedUniversityIdToDepartment}
              value={selectedFacultyIdToDepartment || ''}
```

```
              onChange={(e) =>
setSelectedFacultyIdToDepartment(parseInt(e.target.value))}
            >
              <option value="">Выберите факультет</option>
              {selectedUniversityIdToDepartment && (
                data.find(u => u.id ===
selectedUniversityIdToDepartment)?.faculties.map(faculty => (
                  <option key={faculty.id} value={faculty.id}>
                    {faculty.name}
                  </option>
                )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Название кафедры:</label>
            <input
              type="text"
              className="input-field"
              value={newDepartmentName}
              onChange={(e) =>
setNewDepartmentName(e.target.value)}
            />
          </div>

          <div className="form-group">
            <label>Сокращённое название:</label>
            <input
              type="text"
              className="input-field"
              value={newDepartmentShortName}
              onChange={(e) =>
setNewDepartmentShortName(e.target.value)}
            />
          </div>

          <div className="button-container">
            <button
              onClick={addDepartment}
              disabled={
                !selectedUniversityIdToDepartment ||
                !selectedFacultyIdToDepartment ||
                !newDepartmentName ||
                !newDepartmentShortName
              }
            >
              Добавить
            </button>
            <button
              onClick={() => setIsAddDepartmentModalOpen(false)}
            >
              Отмена
```

```
            </button>
          </div>
        </div>
      )}

      {isAddSpecialityModalOpen && (
        <div className="modal">
          <h3>Добавить специальность</h3>

          <div className="form-group">
            <label>Университет:</label>
            <select
              className="styled-select"
              value={selectedUniversityIdToSpeciality || ''}
              onChange={(e) =>
setSelectedUniversityIdToSpeciality(parseInt(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Факультет:</label>
            <select
              className="styled-select"
              disabled={!selectedUniversityIdToSpeciality}
              value={selectedFacultyIdToSpeciality || ''}
              onChange={(e) =>
setSelectedFacultyIdToSpeciality(parseInt(e.target.value))}
            >
              <option value="">Выберите факультет</option>
              {selectedUniversityIdToSpeciality && (
                data.find(u => u.id ===
selectedUniversityIdToSpeciality)?.faculties.map(faculty => (
                  <option key={faculty.id} value={faculty.id}>
                    {faculty.name}
                  </option>
                )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Кафедра:</label>
            <select
              className="styled-select"
              disabled={!selectedFacultyIdToSpeciality}
```

```
              value={selectedDepartmentIdToSpeciality || ''}
              onChange={(e) =>
setSelectedDepartmentIdToSpeciality(parseInt(e.target.value))}
            >
              <option value="">Выберите кафедру</option>
              {selectedFacultyIdToSpeciality && (
                data.find(u => u.id ===
selectedUniversityIdToSpeciality)
                  ?.faculties.find(f => f.id ===
selectedFacultyIdToSpeciality)
                  ?.departments.map(department => (
                    <option key={department.id}
value={department.id}>
                      {department.name}
                    </option>
                  )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Название специальности:</label>
            <input
              type="text"
              className="input-field"
              value={newSpecialityName}
              onChange={(e) =>
setNewSpecialityName(e.target.value.trim())}
            />
          </div>

          <div className="button-container">
            <button
              onClick={addSpeciality}
              disabled={
                !selectedUniversityIdToSpeciality ||
                !selectedFacultyIdToSpeciality ||
                !selectedDepartmentIdToSpeciality ||
                !newSpecialityName
              }
            >
              Добавить
            </button>
            <button
              onClick={() => setIsAddSpecialityModalOpen(false)}
            >
              Отмена
            </button>
          </div>
        </div>
      )}

      {isAddGroupModalOpen && (
```

```
        <div className="modal">
          <h3>Добавить группу</h3>

          <div className="form-group">
            <label>Университет:</label>
            <select
              className="styled-select"
              value={selectedUniversityIdToGroup || ''}
              onChange={(e) =>
setSelectedUniversityIdToGroup(parseInt(e.target.value))}
            >
              <option value="">Выберите университет</option>
              {data.map(university => (
                <option key={university.id}
value={university.id}>
                  {university.name}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Факультет:</label>
            <select
              className="styled-select"
              disabled={!selectedUniversityIdToGroup}
              value={selectedFacultyIdToGroup || ''}
              onChange={(e) =>
setSelectedFacultyIdToGroup(parseInt(e.target.value))}
            >
              <option value="">Выберите факультет</option>
              {selectedUniversityIdToGroup && (
                data.find(u => u.id ===
selectedUniversityIdToGroup)?.faculties.map(faculty => (
                  <option key={faculty.id} value={faculty.id}>
                    {faculty.name}
                  </option>
                )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Кафедра:</label>
            <select
              className="styled-select"
              disabled={!selectedFacultyIdToGroup}
              value={selectedDepartmentIdToGroup || ''}
              onChange={(e) =>
setSelectedDepartmentIdToGroup(parseInt(e.target.value))}
            >
              <option value="">Выберите кафедру</option>
              {selectedFacultyIdToGroup && (
```

```
                data.find(u => u.id ===
selectedUniversityIdToGroup)
                  ?.faculties.find(f => f.id ===
selectedFacultyIdToGroup)
                  ?.departments.map(department => (
                    <option key={department.id}
value={department.id}>
                      {department.name}
                    </option>
                  )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Специальность:</label>
            <select
              className="styled-select"
              disabled={!selectedDepartmentIdToGroup}
              value={selectedSpecialityIdToGroup || ''}
              onChange={(e) =>
setSelectedSpecialityIdToGroup(parseInt(e.target.value))}
            >
              <option value="">Выберите кафедру</option>
              {selectedDepartmentIdToGroup && (
                data.find(u => u.id ===
selectedUniversityIdToGroup)
                  ?.faculties.find(f => f.id ===
selectedFacultyIdToGroup)
                  ?.departments.find(f => f.id ===
selectedDepartmentIdToGroup)
                  ?.specialities.map(speciality => (
                    <option key={speciality.id}
value={speciality.id}>
                      {speciality.name}
                    </option>
                  )) || []
              )}
            </select>
          </div>

          <div className="form-group">
            <label>Название группы:</label>
            <input
              type="text"
              className="input-field"
              value={newGroupName}
              onChange={(e) =>
setNewGroupName(e.target.value.trim())}
            />
          </div>

          <div className="form-group">
```

```
            <label>Курс:</label>
            <select
              className="styled-select"
              value={selectedCourse}
              onChange={(e) =>
setSelectedCourse(e.target.value)}
            >
              <option value="">Выберите курс</option>
              {COURSES.map(course => (
                <option key={course.value} value={course.value}>
                  {course.label}
                </option>
              ))}
            </select>
          </div>

          <div className="form-group">
            <label>Число студентов:</label>
            <input
              type="number"
              className="input-field"
              value={studentsCount}
              min="15"
              onChange={(e) =>
setStudentsCount(parseInt(e.target.value))}
            />
          </div>

          <div className="button-container">
            <button
              onClick={addGroup}
              disabled={
                !selectedUniversityIdToGroup ||
                !selectedFacultyIdToGroup ||
                !selectedDepartmentIdToGroup ||
                !selectedSpecialityIdToGroup ||
                !newGroupName ||
                !selectedCourse
              }
            >
              Добавить
            </button>
            <button
              onClick={() => setIsAddGroupModalOpen(false)}
            >
              Отмена
            </button>
          </div>
        </div>
      )}
    </>
  );
};
```