

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства для планирования и
контроля студентом учебных задач и закрытия
академических задолженностей**

Студент

(подпись, дата)

Саксонов Денис Витальевич
(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой

(протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Саксонов Денис Витальевич гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства для планирования и контроля студентом
учебных задач и закрытия академических задолженностей

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

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

1) *Назначение: программные средства для планирования и контроля студентом*
учебных задач и закрытия академических задолженностей.

2) *Среда выполнения: Windows 10/Android.*

Языковые средства разработки: Python, TypeScript, SQL, UML,

Инструментальные средства: Visual Studio Code, PyCharm, Docker, npm,
PostgreSQL, Draw.io, Swagger, Visual Paradigm.

Технологии разработки: ООП, слоистая архитектура.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) добавление и управление учебными задачами;

в) классификация и фильтрация задач;

г) планирование учебной активности по календарю;

д) система мотивации и достижений;

е) система уведомлений и напоминаний.

5. Перечень подлежащих разработке вопросов

1) <i>Анализ предметной области и постановка задачи;</i>
2) <i>Проектирование;</i>
3) <i>Реализация;</i>
4) <i>Контроль качества программного обеспечения;</i>
5) <i>Оформление пояснительной записки.</i>

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 «20» 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 95 листов, 46 рисунков, 15 таблиц, 20 использованных источников и 3 приложения.

ПЛАНИРОВАНИЕ УЧЕБНЫХ ЗАДАЧ, АКАДЕМИЧЕСКИЕ ЗАДОЛЖЕННОСТИ, ВЕБ-ПРИЛОЖЕНИЕ, PWA, СЛОИСТАЯ АРХИТЕКТУРА, POSTGRESQL, PYTHON, TYPESCRIPT, UML, БАЗА ДАННЫХ.

Объект исследования – программные средства для планирования и контроля студентом учебных задач и закрытия академических задолженностей.

Цель выпускной квалификационной работы – разработать программные средства для планирования и контроля учебных задач и закрытия академических задолженностей.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования TypeScript с использованием фреймворка Next.js 14. Разработка серверной части приложения производилась на языке программирования Python с использованием фреймворка FastAPI. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – современные веб-браузеры, мобильные устройства (iOS, Android).

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть программных средств для планирования учебных задач и закрытия академических задолженностей. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08BP211.21 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства для планирования и контроля студентом учебных задач и закрытия академических задолженностей. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Саксонов Д.В.					4	95
Пров.		Балашова И.Ю.						
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.				Группа 21ВП1		

Содержание

Введение.....	7
1 Анализ программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей	8
1.1 Исследование предметной области	8
1.2 Обзор программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей	9
1.2.1 Веб-приложение «MyStudyLife».....	9
1.2.2 Веб-приложение «Power Planner»	10
1.2.3 Веб-приложение «Notion»	11
1.2.4 Мобильное приложение «School Planner»	12
1.2.5 Сравнение программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей	13
1.3 Анализ функциональных требований	15
1.4 Анализ нефункциональных требований	19
1.5 Планирование разработки и оценка бюджета	21
2 Проектирование программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей..	26
2.1 Архитектура приложения	26
2.2 Проектирование структур данных.....	27
2.2.1 Концептуальная модель базы данных	27
2.2.2 Логическая модель базы данных	29
2.3 Проектирование пользовательского интерфейса.....	30
2.4 Проектирование классов программных средств для планирования учебных задач студентом и закрытия академических задолженностей	31

3	Реализация программных средств для планирования учебных задач студентом и закрытия академических задолженностей.....	35
3.1	Реализация базы данных.....	35
3.2	Реализация серверной части приложения.....	38
3.3	Реализация клиентской части приложения	40
3.4	Реализация интерфейса приложения.....	41
4	Контроль качества программного обеспечения.....	55
4.1	Выбор режима тестирования и метрик кода	55
4.2	Тестирование программных средств для планирования учебных задач студентом и закрытия академических задолженностей	57
4.2.1	Функциональное тестирование	57
4.2.2	Тестирование на совместимость системы.....	58
4.3	Оценка качества программного кода	59
4.3.1	Метрика Холстеда	59
4.3.2	Метрика Чепина.....	61
	Заключение	64
	Список использованных источников	66
	Приложение А (справочное) Глоссарий	68
	Приложение Б (обязательное) Исходный код приложения.....	71
	Приложение В (рекомендуемое) Функциональное тестирование	90

Введение

Разработка программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей актуальна из-за высокой учебной нагрузки, сложностей в организации работы и необходимости расстановки приоритетов. Грамотное управление задачами и своевременное выполнение обязательств существенно влияют на успех в обучении.

Прогрессивное веб-приложение (PWA) обеспечивает удобство использования благодаря офлайн-доступу, возможности установки на устройства и адаптивному интерфейсу для разных платформ.

Целью выпускной квалификационной работы является разработка программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей с возможностями анализа эффективности обучения и мотивации через систему достижений.

Исходя из цели данной работы, были поставлены следующие задачи:

- изучить предметную область, аналоги и сформулировать требования на разработку;
- спроектировать программные средства и их структуры данных;
- разработать клиентскую и серверную части приложения;
- проверить качество программного обеспечения.

Реализация поставленных задач позволит разработать инструмент для повышения продуктивности студентов и эффективного закрытия академических задолженностей.

1 Анализ программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей

1.1 Исследование предметной области

Программные средства для планирования и контроля учебных задач студентом и закрытия академических задолженностей – это специальные инструменты, предназначенные для помощи обучающимся в эффективной организации учебного процесса. Они позволяют фиксировать сроки сдачи лабораторных заданий, экзаменов, курсовых работ, контролировать процесс выполнения учебных задач и задолженностей, а также отслеживать сроки сдачи заданий. Они включают функции, такие как создание задач, напоминания о дедлайнах и календарное представление.

Академическая деятельность студента включает в себя регулярное выполнение текущих заданий, участие в промежуточных аттестациях и закрытие академических задолженностей, при их возникновении. При этом студенту важно не только помнить о предстоящих событиях, но и уметь расставлять приоритеты, планировать свою нагрузку, учитывать сроки пересдач и вовремя закрывать задолженности. В условиях дистанционного или смешанного обучения потребность в таких инструментах особенно возрастает.

Программные решения для управления учебной нагрузкой позволяют:

- централизованно хранить информацию о заданиях и дедлайнах;
- получать напоминания и уведомления;
- визуализировать прогресс выполнения заданий;
- классифицировать задачи по предметам и типу (лабораторная работа, курсовой проект, зачет, экзамен и т. п.);
- отслеживать статус академической задолженности.

Данная предметная область охватывает технологии и программные инструменты, помогающие студентам организовать учебную деятельность,

систематизировать задачи, минимизировать несдачи и успешно закрывать академические задолженности.

1.2 Обзор программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей

1.2.1 Веб-приложение «MyStudyLife»

Приложение «MyStudyLife» представляет собой бесплатный кроссплатформенный учебный планировщик, разработанный для студентов и школьников [1]. Интерфейс данного приложения изображен на рисунке 1.

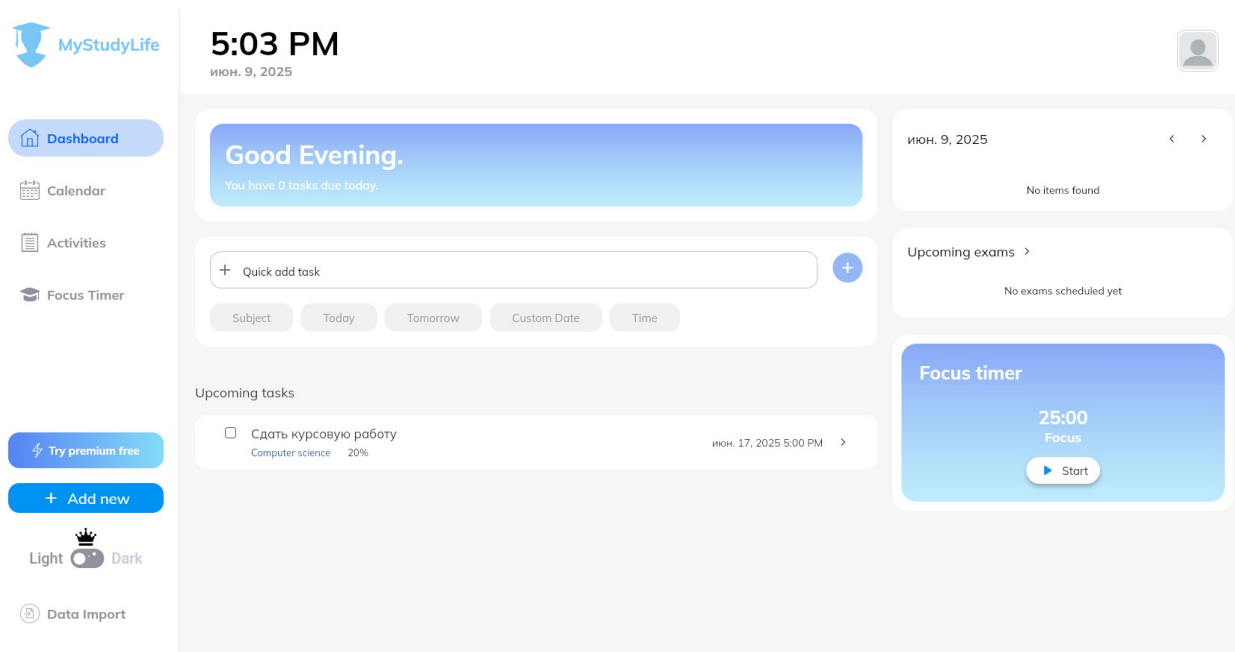


Рисунок 1 – Интерфейс приложения «MyStudyLife»

Интерфейс данного приложения интуитивно понятен и адаптирован под задачи планирования занятий, есть возможность создавать задачи, добавлять свои учебные предметы, и отслеживать их выполнение в удобном календарном формате.

Главным недостатком, с точки зрения пользователя, можно считать отсутствие функции фильтрации задач по приоритету, а также системы мотивации, стимулирующей их выполнение.

1.2.2 Веб-приложение «Power Planner»

Приложение «Power Planner» – учебный планировщик с возможностью отслеживания расписания. Поддерживает интеграцию со сторонними календарями. Предназначен в основном для школьников и студентов. Интерфейс приложения представлен на рисунке 2.

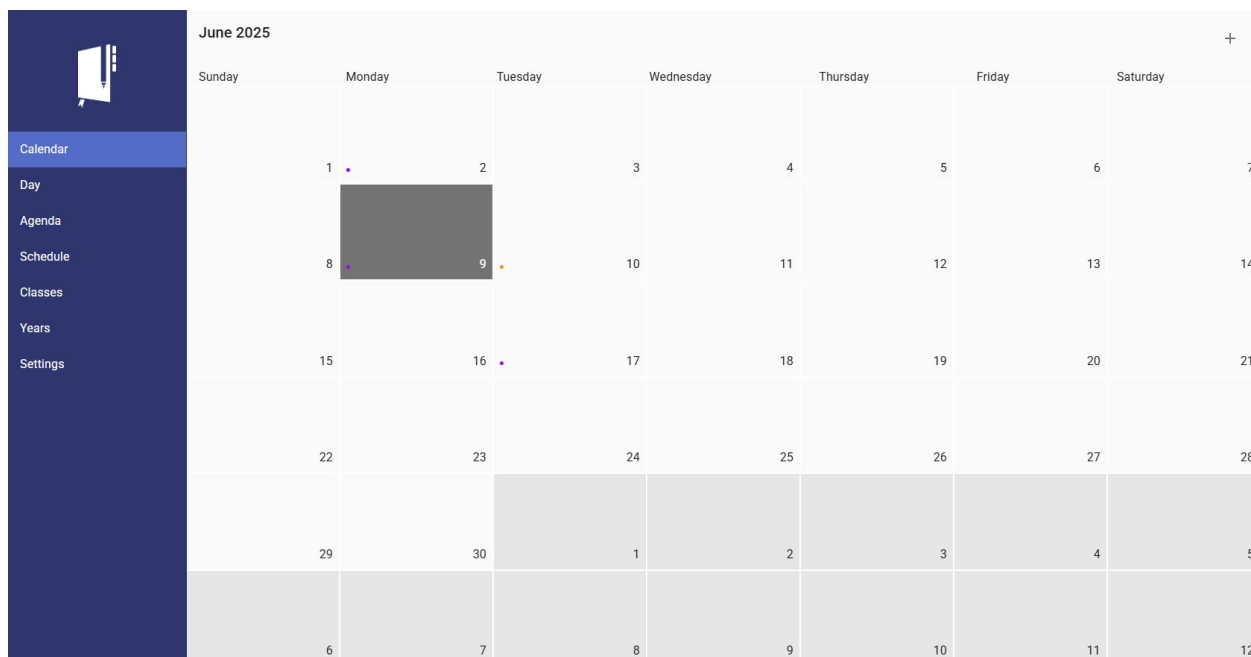


Рисунок 2 – Интерфейс приложения «Power Planner»

Среди достоинств этого приложения можно выделить то, что оно предоставляет пользователю возможность отслеживать задания, экзамены и оценки по предметам. «Power Planner» интегрируется с календарями и предоставляет напоминания, что помогает студенту контролировать сроки и оценивать влияние текущих оценок на итоговые результаты [2].

Дополнительным преимуществом является функция прогноза итоговых оценок, учитывая введенные текущие баллы.

К недостаткам можно отнести не всегда интуитивно понятный интерфейс, который может затруднять использование, а также то, что приложение ориентировано преимущественно на индивидуальное использование, без поддержки совместной работы над учебными проектами.

1.2.3 Веб-приложение «Notion»

Приложение «Notion» является универсальным инструментом для ведения заметок, баз данных и управления задачами, который активно используется студентами для планирования и систематизации учебной деятельности, интерфейс представлен на рисунке 3.

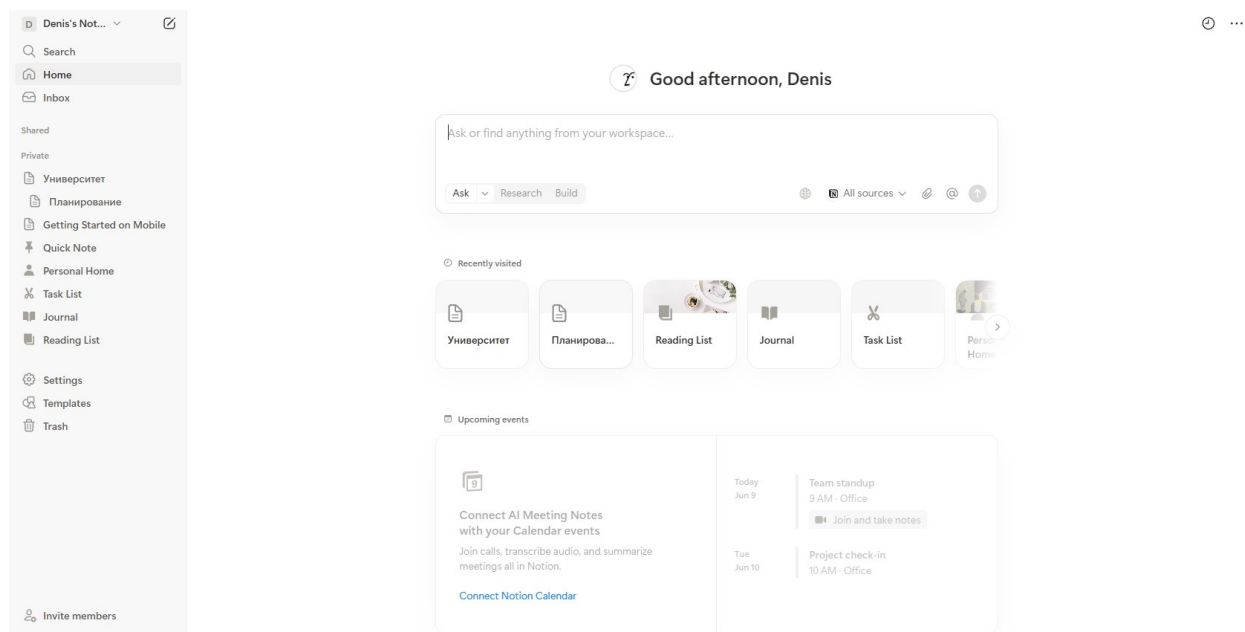


Рисунок 3 – Интерфейс приложения «Notion»

Среди достоинств можно выделить его мощные возможности для создания структурированных баз данных, которые позволяют связывать задачи с учебными предметами, организовывать таблицы, чек-листы и использовать готовые шаблоны. Платформа предоставляет гибкость в организации информации по курсам, создании персональных трекеров задолженностей и адаптации рабочего пространства под индивидуальный стиль пользователя. «Notion» подходит не только для учебы, но и для управления проектами, личного планирования и других целей благодаря своей универсальности.

К недостаткам «Notion» относится отсутствие встроенной системы уведомлений и напоминаний, что требует интеграции со сторонними сервисами для реализации этой функции. Для полноценного ведения календаря также необходимо использовать отдельное приложение или дополнительные

интеграции. Кроме того, освоение платформы и настройка собственной структуры могут занять значительное время, что может быть сложным для новых пользователей [3].

1.2.4 Мобильное приложение «School Planner»

Приложение «School Planner» – это мобильный органайзер для устройств на базе Android, разработанный для учащихся всех возрастов. Оно предоставляет возможность вести расписание занятий, отслеживать задания, устанавливать напоминания, контролировать посещаемость и сохранять информацию о преподавателях.

Ключевые особенности данного приложения:

- автоматизация учебного процесса;
- управление заданиями;
- возможность аналитики успеваемости;
- контроль посещаемости учащегося;
- уведомления и напоминания.

Интерфейс приложения «School Planner» представлен на рисунке 4.

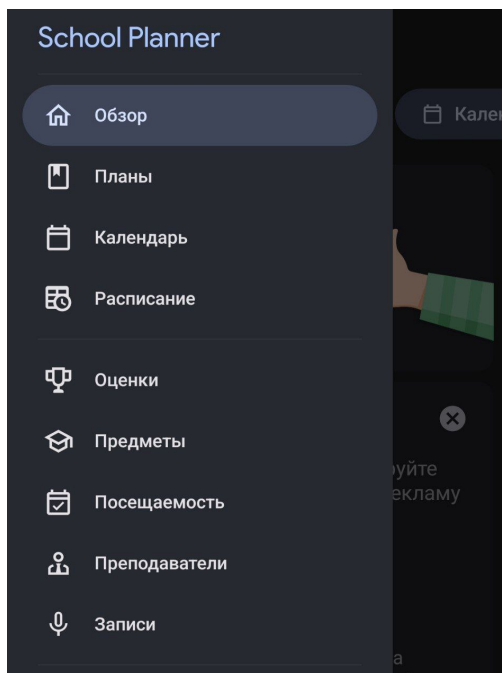


Рисунок 4 – Интерфейс приложения «School Planner»

Одним из ключевых преимуществ приложения является возможность создания детализированного расписания занятий, добавления заданий, экзаменов и событий. «School Planner» позволяет настраивать напоминания, вести базу данных преподавателей и автоматически группировать задачи по типу и предмету, что упрощает контроль учебной нагрузки и помогает эффективно организовать учебный процесс.

Недостатком этого приложения является отсутствие веб-версии и ограниченные возможности по синхронизации с другими платформами. Также «School Planner» не поддерживает визуализацию статистики выполнения задач, что снижает его функциональность по сравнению с более универсальными решениями [4].

1.2.5 Сравнение программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей

Проведем сравнительный анализ рассмотренных выше программных средств: «MyStudyLife», «Power Planner», «Notion», «School Planner», по ключевым функциям, значимым для студента. Для оценки будет использована трехбалльная шкала, характеристики оценки представлены в таблице 1.

Таблица 1 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) полностью отсутствует
1	существует частичная реализация функции (качества)
2	функция (качество) полностью реализована

На основе указанной шкалы проведена сравнительная оценка ключевых функций работы в программных средствах, рассматриваемых выше. Каждый критерий оценивался независимо на основе тестирования актуальных на данный момент версий приложений.

Результаты сравнения программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей представлены в таблице 2.

Таблица 2 – Сравнение программных средств аналогичного назначения

Критерий \ Приложение	«MyStudy Life»	«Power Planner»	«Notion»	«School Planner»
1. Возможность добавления задач	2	2	2	2
2. Возможность распределять задачи по типу	2	2	2	2
3. Возможность распределять задачи по приоритету выполнения	1	2	2	1
4. Возможность получения напоминаний и уведомлений	2	2	1	2
5. Возможность просмотра в календарном виде	2	2	1	2
6. Возможность разбивать задачи на этапы	0	1	1	0
7. Статистика и аналитика	1	2	1	1
8. Система мотивации и достижений	1	0	1	0
ИТОГО	11	13	12	10

Анализ показывает, что среди рассмотренных решений «Power Planner» обладает наибольшим набором функций, полезных для студента, включая учет оценок, аналитику, интеграцию с календарем и классификацию заданий. «Notion» демонстрирует высокую гибкость и возможность использования готовых шаблонов, но требует больше времени на освоение. «MyStudyLife» – хороший выбор для простого планирования с акцентом на расписание и напоминания. «School Planner» минималистичное мобильное приложение, подходящее для школьников, но уступающее в аналитике и системе мотивации.

Для наглядности, полученные результаты отображены на диаграмме, приведенной на рисунке 5.

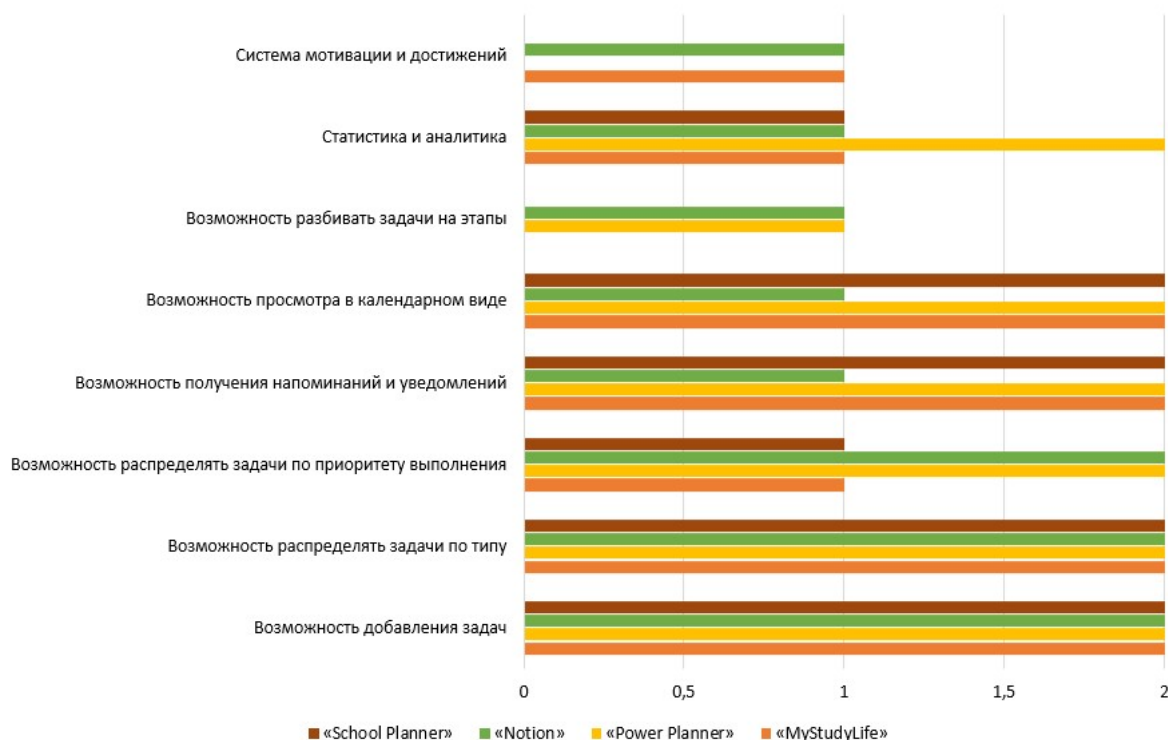


Рисунок 5 – Результаты сравнения приложений-аналогов

Таким образом, ни одно из решений не реализует в полной мере все необходимые функции, включая возможность разбивать задачи на этапы, систему достижений и мотивации. Это указывает на актуальность разработки нового программного продукта, ориентированного на нужды студентов высших учебных заведений с ориентированием на контроль академических задолженностей и задач текущего учебного плана.

1.3 Анализ функциональных требований

На основе анализа предметной области и существующих программных решений были определены функциональные требования к разрабатываемому программному средству, предназначенному для планирования и контроля учебных задач студентом, а также для своевременного закрытия академических задолженностей.

Программное средство должно предоставлять пользователю следующие основные функции:

- регистрация и авторизация пользователя;

- редактирование профиля пользователя;
- добавление и управление учебными задачами;
- отслеживание выполнения задач;
- планирование учебной активности по календарю;
- контроль и учет академических задолженностей;
- система уведомлений и напоминаний;
- классификация и фильтрация задач;
- система мотивации в виде достижений.

Для наглядного представления функциональных требований системы была разработана диаграмма вариантов использования. Эта модель визуализирует взаимодействие пользователей с приложением, четко определяя доступные сценарии работы.

Диаграмма вариантов использования (use case diagram) относится к стандарту UML и выполняет три важные функции. Во-первых, она структурирует возможности системы по смысловым блокам, например, группируя функции планирования и аналитики. Во-вторых, определяет права доступа для разных категорий пользователей, таких как студенты или преподаватели. В-третьих, демонстрирует взаимосвязи между действиями пользователей и ответами программного обеспечения. Все это обеспечивает более четкое понимание функционала приложения.

В соответствии с выявленными функциональными требованиями была разработана диаграмма вариантов использования UML, представленная на рисунке 6.

Построенная диаграмма описывает функционал программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей, она охватывает все главные действия пользователя, которым в большинстве случаев является студентом высшего учебного заведения.

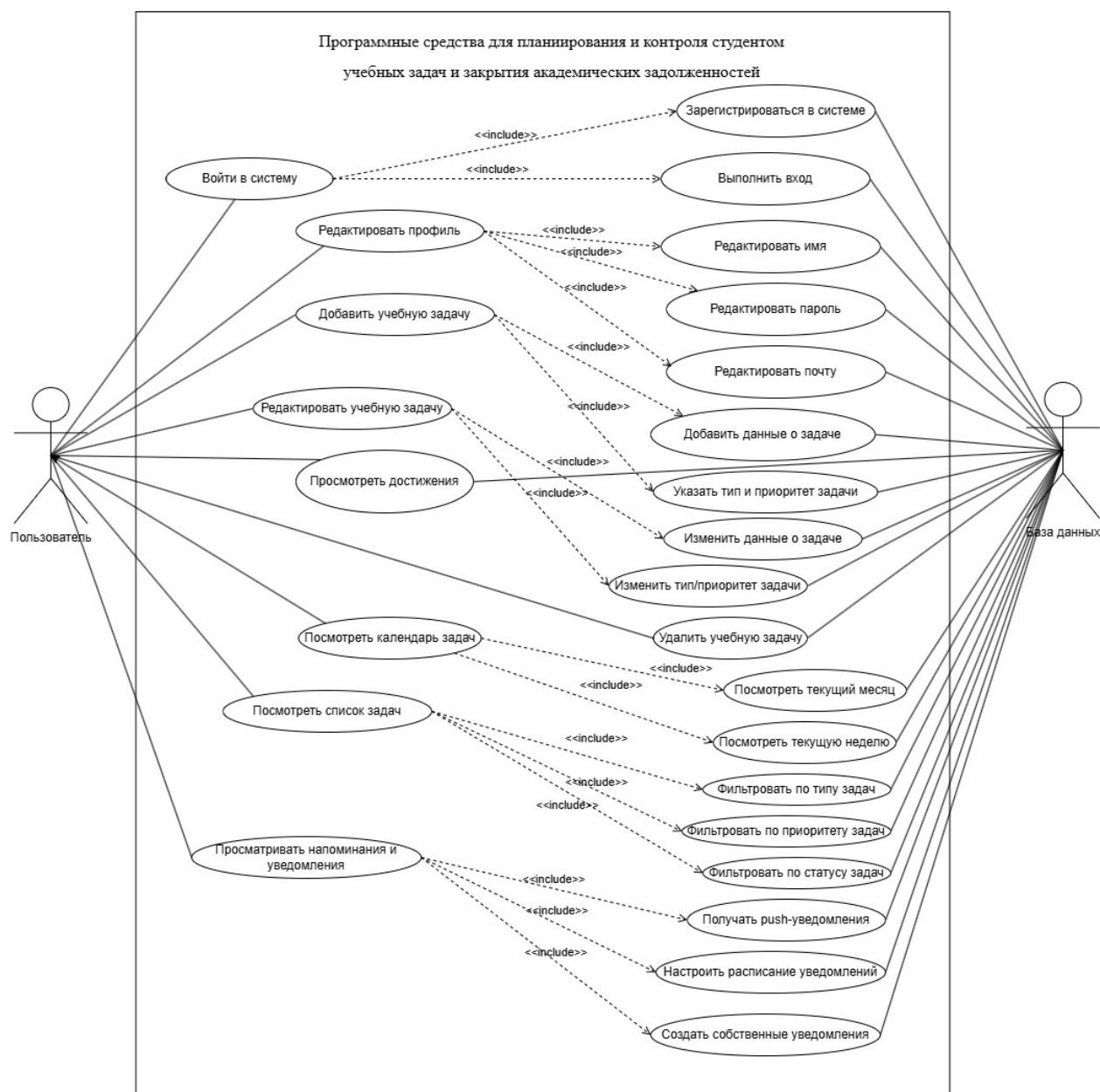


Рисунок 6 – Диаграмма вариантов использования

Для детализации функциональных требований и более четкого описания взаимодействия пользователя с разрабатываемой системой ниже были приведены спецификации наиболее важных бизнес-процессов, таких как вход в систему, добавление учебной задачи и просмотр существующих задач. Каждая спецификация представлена в виде структурированной таблицы, имеет свой идентификатор и название, а также включает следующие элементы: участников бизнес-процесса, условия начала выполнения, последовательность шагов и ожидаемое завершение процесса.

Сценарий варианта использования «Войти в систему» представлен на рисунке 7.

Прецендент: Войти в систему
ID: 1
Краткое описание: Пользователь регистрируется или авторизуется в системе
Главные актёры: Пользователь
Второстепенные актёры: База данных
Предусловие: 1. Пользователь открыл приложение 2. Пользователь не был зарегистрирован/авторизован
Основной поток: 1. Пользователь переходит на страницу входа в приложение 2. Пользователь вводит данные для входа в существующий аккаунт 3. Пользователь получает доступ к функционалу приложения
Альтернативный поток: 2.а. Пользователь вводит данные для регистрации нового аккаунта
Постусловие: 1. Пользователь авторизован в приложении и ему доступен функционал

Рисунок 7 – Сценарий варианта использования «Войти в систему»

Сценарий варианта использования «Добавить учебную задачу» изображен на рисунке 8.

Прецендент: Добавить учебную задачу
ID: 2
Краткое описание: Пользователь добавляет новую задачу в список своих задач
Главные актёры: Пользователь
Второстепенные актёры: База данных
Предусловие: 1. Пользователь прошёл авторизацию
Основной поток: 1. Пользователь нажимает на кнопку «Добавить задачу» 2. Пользователь указывает необходимую информацию о задаче 3. Пользователь указывает тип и приоритет задачи
Постусловие: 2. Учебная задача создана и добавлена в систему

Рисунок 8 – Сценарий варианта использования «Добавить учебную задачу»

Сценарий варианта использования «Посмотреть задачи» представлен на рисунке 9.

Прецедент: Посмотреть задачи
ID: 3
Краткое описание: Пользователь просматривает информацию о своих учебных задачах
Главные актёры: Пользователь
Второстепенные актёры: База данных
Предусловие: 1. Пользователь прошёл авторизацию
Основной поток: 1. Пользователь просматривает учебные задачи в виде списка 2. Пользователь настраивает фильтрацию задач Альтернативный поток: 1.а. Пользователь просматривает календарь учебной активности 2.а. Пользователь настраивает вид календаря
Постусловие: 1. Пользователем просмотрена информация

Рисунок 9 – Сценарий варианта использования «Посмотреть задачи»

В результате анализа функциональных требований были выделены варианты использования программы. Для некоторых вариантов использования, которые являются основными для работы разрабатываемого программного средства, были представлены их спецификации, позволяющие отобразить последовательность действий пользователя и системы.

1.4 Анализ нефункциональных требований

В программном средстве для планирования и контроля учебных задач студентом и закрытия академических задолженностей предусматривается разработка единой базы данных. Разработка базы данных будет проводиться с использованием PostgreSQL на языке SQL с применением SQLAlchemy для ORM и Alembic для управления миграциями, что обеспечит надежное хранение данных и их целостность [5].

Клиентская часть приложения будет реализована в формате прогрессивного веб-приложения (PWA) на языке TypeScript с использованием фреймворка Next.js 14 в среде разработки Visual Studio Code. PWA-функциональность позволит обеспечить поддержку офлайн-режима, установку на мобильные устройства и высокую скорость загрузки.

Серверная часть приложения будет реализована на основе слоистой MVC-архитектуры с использованием фреймворка FastAPI на языке Python. Все операции взаимодействуют с единой базой данных через REST API.

Развертывание приложения осуществляется в продуктивной среде с использованием Docker и Docker Compose. Клиентская и серверная части упакованы в отдельные Docker-контейнеры, определенные в Dockerfile для каждой части. Контейнеризация упрощает масштабирование, развертывание и обслуживание системы.

Для обеспечения безопасности передачи данных между клиентской и серверной частями приложения будет использоваться система авторизации и аутентификации на основе JWT-токенов.

Использование PostgreSQL обеспечит высокую производительность, надежность хранения данных и соответствие стандартам реляционного моделирования. Применение технологии PWA позволит реализовать удобный, кроссплатформенный интерфейс, поддерживающий как онлайн-, так и офлайн-режимы работы [6].

Построение диаграмм будет осуществляться с использованием языка UML 2.0 в онлайн-среде draw.io, а также в , которая позволяет создавать диаграммы вариантов использования, классов, последовательностей, развертывания и другие типы визуальных моделей.

Разрабатываемое веб-приложение должно быть оптимизировано для работы в современных браузерах, таких как Google Chrome (версии 80 и выше), Mozilla Firefox (версии 75 и выше), Apple Safari (версии 13 и выше), Microsoft Edge (версии 80 и выше).

1.5 Планирование разработки и оценка бюджета

Для оценки временных затрат и структурированного планирования процесса разработки была создана диаграмма Ганта с использованием инструмента ProjectLibre.

Диаграмма Ганта – это инструмент для управления проектами по методике Agile. Она наглядно отображает временную шкалу выполнения работ, включая начало и конец работ, взаимосвязи между задачами и распределение нагрузки среди ресурсов, диаграмма Ганта помогает планировать прогресс всех задач и действий в рамках проекта.

Процесс разработки программного средства включает четыре ключевых этапа, каждый из которых состоит из подзадач, необходимых для достижения следующей вехи. Для создания приложения определены следующие этапы:

- анализ предметной области и постановка требований к программным средствам для планирования и контроля учебных задач студентом и закрытия академических задолженностей;
- проектирование программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей;
- реализация программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей;
- контроль качества и тестирование программного обеспечения.

Визуализацией разработанного плана работ, соответствующего описанным этапам, является диаграмма Ганта

Для ее построения необходимо составить описание всех задач с четким указанием их продолжительности, даты начала и окончания, а также назначить ресурсы для их выполнения, грамотно распределяя нагрузку.

Применение этой диаграммы также позволяет вовремя корректировать сроки выполнения при возникновении задержек.

Составление перечня всех этапов разработки выполнено в среде ProjectLibre и представлено на рисунке 10.

Название	Продолжи...	Начало	Окончание	
Начало проекта	0 дней?	08.04.25 8:00	08.04.25 8:00	
Анализ предметной области и постановка требований к прог	5 дней	08.04.25 8:00	14.04.25 17:00	1
Анализ предметной области	0,5 дней	08.04.25 8:00	08.04.25 13:00	
Изучение аналогов	0,5 дней	08.04.25 13:00	08.04.25 17:00	3
Постановка задачи	1 день	09.04.25 8:00	09.04.25 17:00	4
Анализ функциональных требований	1 день	10.04.25 8:00	10.04.25 17:00	5
Анализ нефункциональных требований	1 день	11.04.25 8:00	11.04.25 17:00	6
Планирование разработки и оценка бюджета	1 день	14.04.25 8:00	14.04.25 17:00	7
Проектирование программных средств для планирования и	9 дней	15.04.25 8:00	25.04.25 17:00	8
Архитектура приложения	1 день	15.04.25 8:00	15.04.25 17:00	
Проектирование структур данных	2 дней	16.04.25 8:00	17.04.25 17:00	10
Проектирование базы данных	2 дней	18.04.25 8:00	21.04.25 17:00	11
Проектирование пользовательского интерфейса	2 дней	22.04.25 8:00	23.04.25 17:00	12
Проектирование классов программных средств	2 дней	24.04.25 8:00	25.04.25 17:00	13
Реализация программных средств для планирования и конт	11 дней	28.04.25 8:00	12.05.25 17:00	14
Реализация базы данных	2 дней	28.04.25 8:00	29.04.25 17:00	
Реализация серверной части	4 дней	30.04.25 8:00	05.05.25 17:00	16
Реализация клиентской части	3 дней	06.05.25 8:00	08.05.25 17:00	17
Реализация интерфейса	2 дней	09.05.25 8:00	12.05.25 17:00	18
Контроль качества и тестирование программного обеспечен	8 дней	13.05.25 8:00	22.05.25 17:00	19
Выбор режима тестирования	1 день	13.05.25 8:00	13.05.25 17:00	
Тестирование программных средств	4 дней	14.05.25 8:00	19.05.25 17:00	21
Оценка качества программного кода	3 дней	20.05.25 8:00	22.05.25 17:00	22
Оформление отчета	8 дней	23.05.25 8:00	03.06.25 17:00	23
Оформление пояснительной записки	8 дней	23.05.25 8:00	03.06.25 17:00	
Конец проекта	0 дней	03.06.25 17:00	03.06.25 17:00	24

Рисунок 10 – Описание этапов разработки приложения

Для оценки финансовых затрат определены стоимости ресурсов, задействованных в разработке. Основную часть расходов составляют человеческие ресурсы: два разработчика, аналитик и тестировщик. Для каждого из них указаны стандартная и сверхурочная ставки почасовой оплаты труда. Стоимость всех ресурсов представлена на рисунке 11.

	Название	Стандартная ставка	Ставка сверхурочных	Затраты на одно испол...
1	Аналитик	200/час	240/час	0 руб.
2	Frontend-разработчик	250/час	300/час	0 руб.
3	Backend-разработчик	260/час	310/час	0 руб.
4	Тестировщик	210/час	250/час	0 руб.
5	Компьютер	0		60000 руб.
6	Интернет	1 000/месяц	0/час	0 руб.

Рисунок 11 – Стоимость используемых ресурсов

Для визуализации временных характеристик разработки была построена диаграмма Ганта, приведенная на рисунке 12.

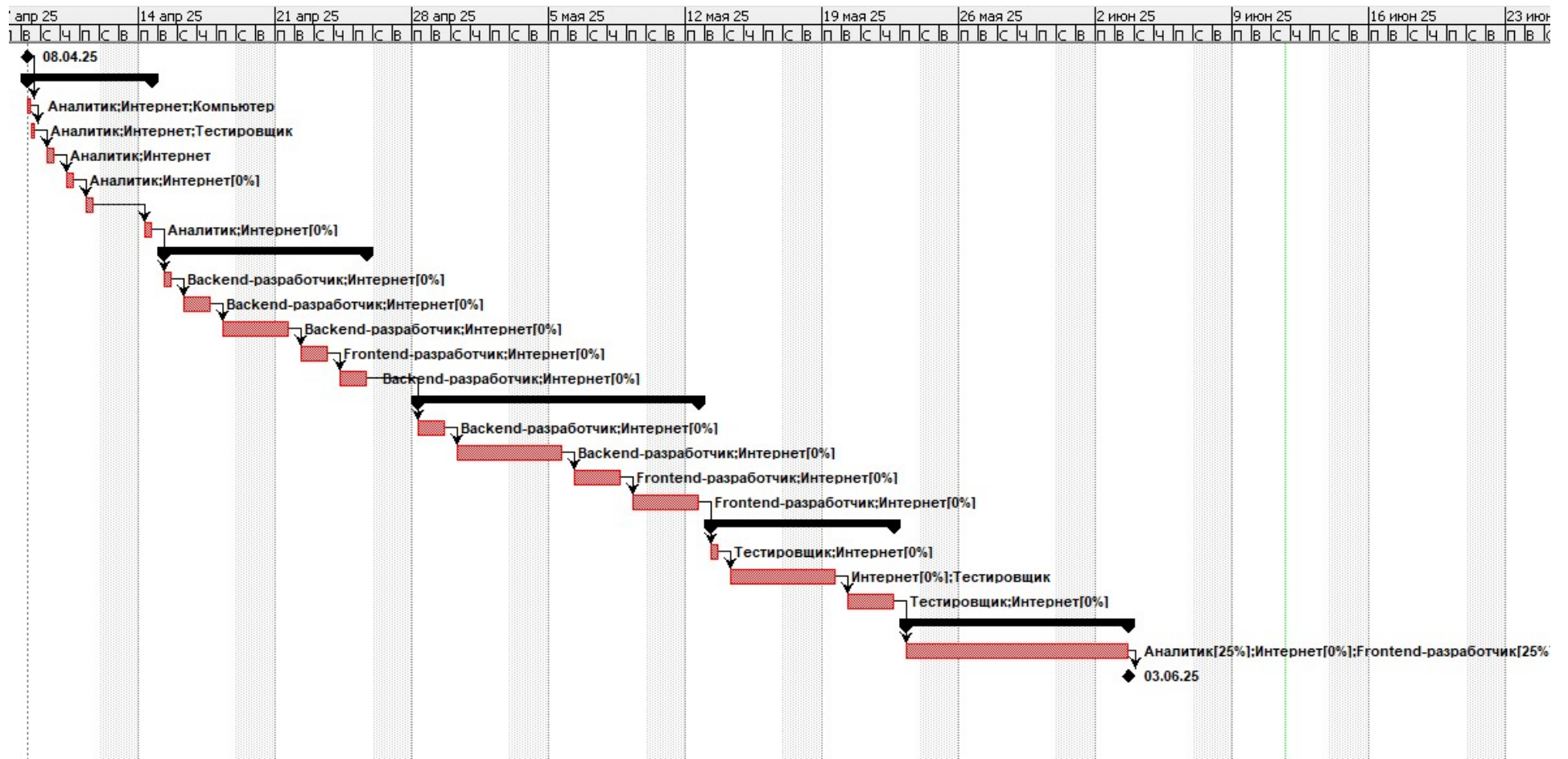


Рисунок 12 – Этапы разработки на диаграмме Ганта

На первом этапе выполняется анализ предметной области, постановка задачи на разработку, изучение аналогов и формирование функциональных и нефункциональных требований. Второй этап включает проектирование архитектуры, разработку UML-диаграмм, проектирование базы данных и пользовательского интерфейса. Третий этап охватывает реализацию клиентской части и серверной части, базы данных и интерфейса, а также выбор способа развертывания. Четвертый этап предполагает проведение тестирования и проверку приложения на тестовых сценариях для выявления ошибок.

Для общей оценки бюджета был сгенерирован отчет по общим затратам на проект, который приведен на рисунке 13.

Dates			
Start	08.04.25 8:00	Finish	03.06.25 17:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	41 дней	Remaining	41 дней
Baseline	0 дней	Actual	0 дней
		Percent Complete	0%

Work			
Scheduled	313,71 часов	Remaining	313,71 часов
Baseline	0 часов	Actual	0 часов

Costs			
Scheduled	130958,34 руб.	Remaining	130958,34 руб.
Baseline	0 руб.	Actual	0 руб.
		Variance	0 руб.

Рисунок 13 – Оценка финансовых затрат проекта

Таким образом, планирование разработки с использованием диаграммы Ганта позволило структурировать процесс, а оценка бюджета показала, что

затраты на разработку программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей составляют 130 958,34 рублей при продолжительности проекта 41 день.

2 Проектирование программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей

2.1 Архитектура приложения

Изучаемое программное средство разработано на основе слоистой архитектуры с использованием шаблона проектирования MVC, который может обеспечить разделение логики данных, бизнес-логики и пользовательского интерфейса. Архитектура включает в себя клиентскую и серверную части, которые взаимодействуют между собой через REST API. Общая структура системы изображена на диаграмме развертывания, которая представлена на рисунке 14.

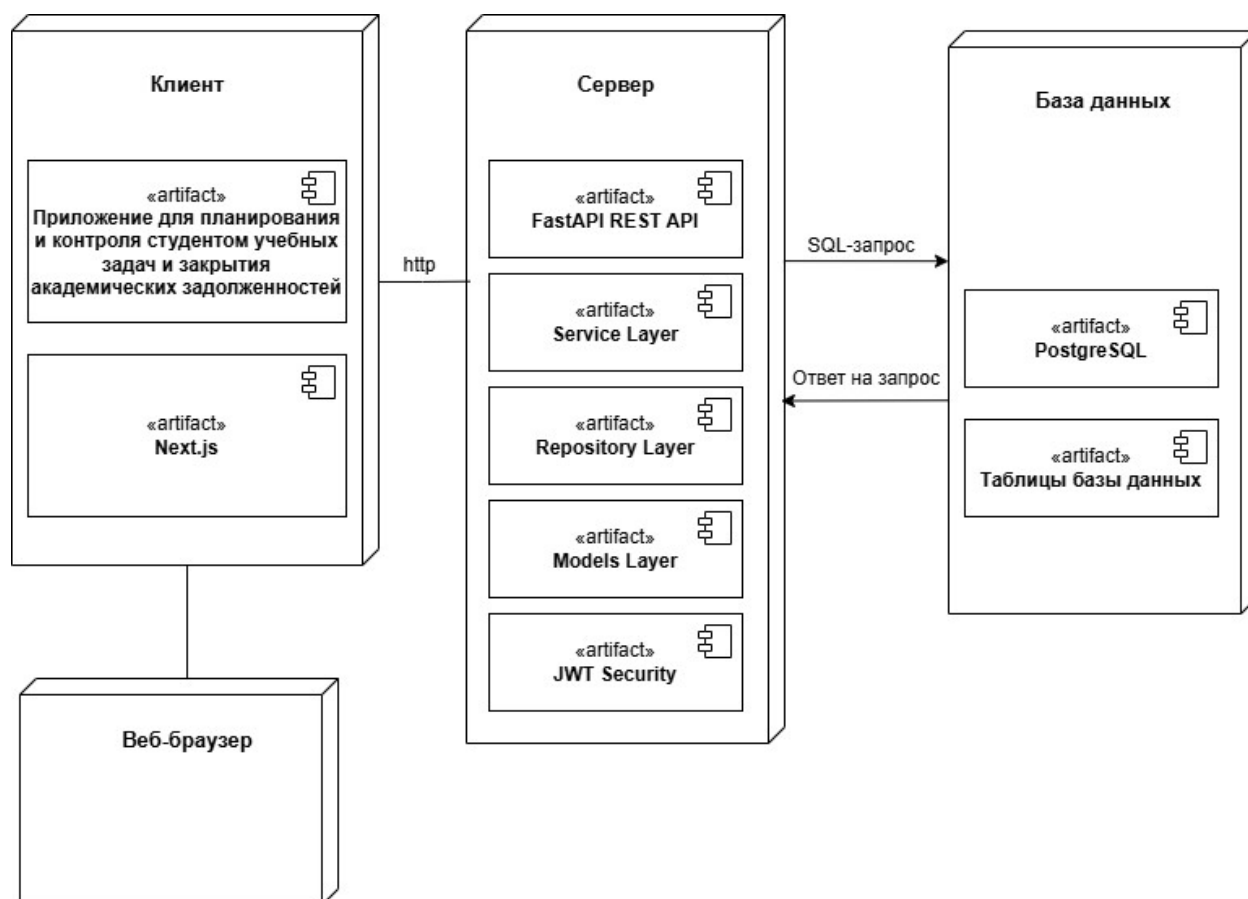


Рисунок 14 – Архитектура программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей

Клиентская часть реализована как прогрессивное веб-приложение, она взаимодействует с сервером через REST API. PWA-функциональность обеспечивает офлайн-доступ.

Серверная часть включает в себя модель, контроллер и бизнес-логику, взаимодействует с единой базой данных, которая хранит информацию о пользователях, учебных задачах, задолженностях, достижениях.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель базы данных

Концептуальная модель данных представляет собой общую структуру данных, необходимых для поддержки бизнес-требований, независимо от какого-либо программного обеспечения или структуры хранения данных [7].

IDEF1X является одним из методов для разработки реляционных баз данных, в этой нотации используется условный синтаксис, разработанный специально для удобного построения концептуальной модели.

Данная диаграмма иллюстрирует основные сущности, их атрибуты и взаимосвязи, дает общее представление о том, что должно входить в состав модели базы данных.

В рамках данного проекта была разработана концептуальная модель базы данных, ключевыми сущностями которой являются: пользователь, задача, этап задачи, цель, уведомление, достижение, полученное достижение, пуш-подписка.

Каждая сущность обладает определенным набором атрибутов, отражающих все нужные данные для правильного функционирования системы. Также важной деталью концептуальной модели являются связи между всеми этими сущностями. Связи обеспечивают целостность данных и правильную реализацию бизнес-логики веб-приложения.

На рисунке 15 представлена диаграмма типа «сущность-связь», построенная в нотации IDEF1X.

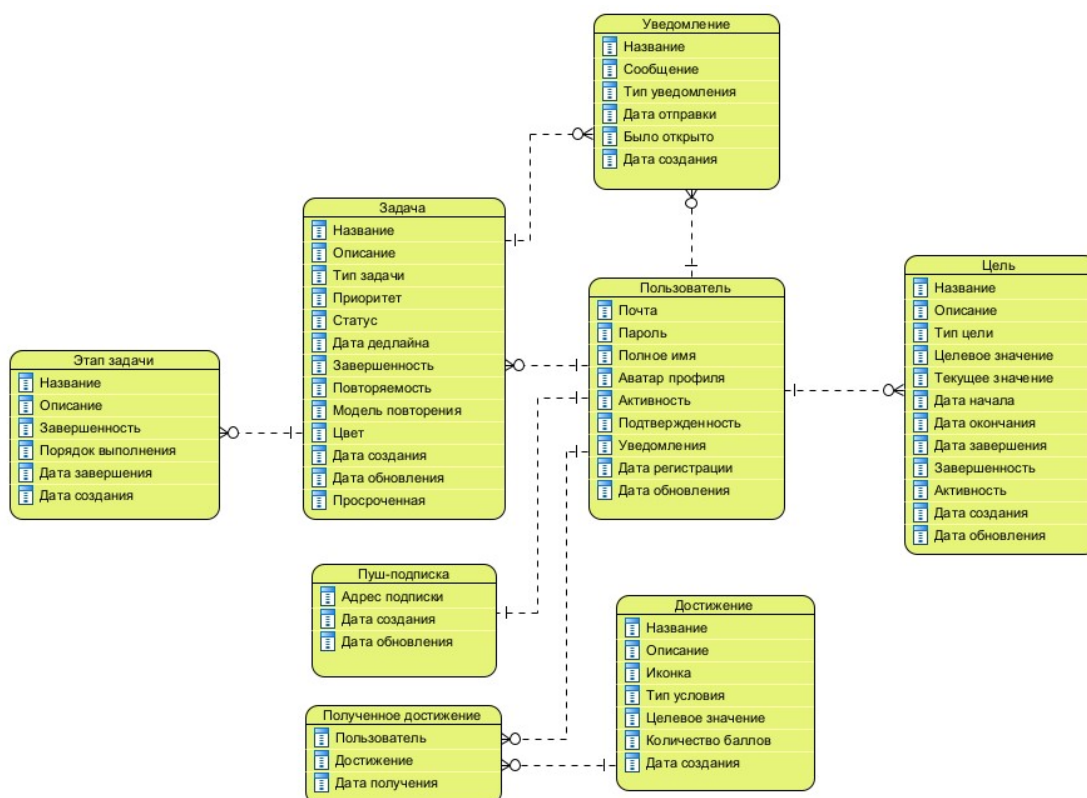


Рисунок 15 – Концептуальная модель базы данных приложения

Диаграмма наглядно показывает хранимые данные приложения и связи между ними. В таблице 3 приведены основные сущности, выделенные в рамках анализа бизнес-процессов:

Таблица 3 – Описание сущностей

Наименование сущности	Идентификатор сущности	Назначение сущности
Пользователь	users	Представляет зарегистрированного студента. Содержит основные учетные данные
Задача	tasks	Содержит данные об учебной задаче с ее типом, приоритетом и статусом
Этап задачи	task_steps	Содержит данные, с помощью которых можно разбить задачу на шаги
Цель	goals	Содержит данные, с помощью которых пользователь может отследить свой прогресс
Достижение	achievements	Содержит условия награды студента за выполнение определенных действий

Продолжение таблицы 3

Наименование сущности	Идентификатор сущности	Назначение сущности
Полученное достижение	user_achievements	Отображает, какие достижения были получены конкретным студентом
Уведомление	notifications	Информирует пользователя о событиях
Пуш-подписка	push_subscriptions	Содержит параметры подписки для отправки уведомлений

В таблице 4 представлено описание связей между сущностями.

Таблица 4 – Описание связей между сущностями

Наименование связи	Описание	Мощность
Пользователь-Задача	Определяет принадлежность пользователей к задачам	1:M
Задача-Этап задачи	Определяет принадлежность задач к этапам задачи	1:M
Пользователь-Цель	Определяет принадлежность пользователей к целям	1:M
Пользователь-Уведомление	Определяет принадлежность пользователей к уведомлениям	1:M
Задача-Уведомление	Определяет принадлежность задач к уведомлениям	1:M
Пользователь-Пуш-подписка	Определяет принадлежность пользователей к пуш-подпискам	1:1
Пользователь-Полученное достижение	Определяет принадлежность пользователей к полученным достижениям	1:M
Достижение-Полученное достижение	Определяет принадлежность достижений к полученным достижениям	1:M

Данное представление концептуальной модели построено в среде Visual Paradigm 17.0 в нотации IDEF1X.

2.2.2 Логическая модель базы данных

Логической моделью данных называют усовершенствованную версию концептуальной модели. В ней схематически представлены ограничения данных, имена сущностей и связи, которые нужно будет реализовать независимо от платформы [8].

На основе концептуальной модели базы данных была построена логическая модель, которая представлена на рисунке 16.

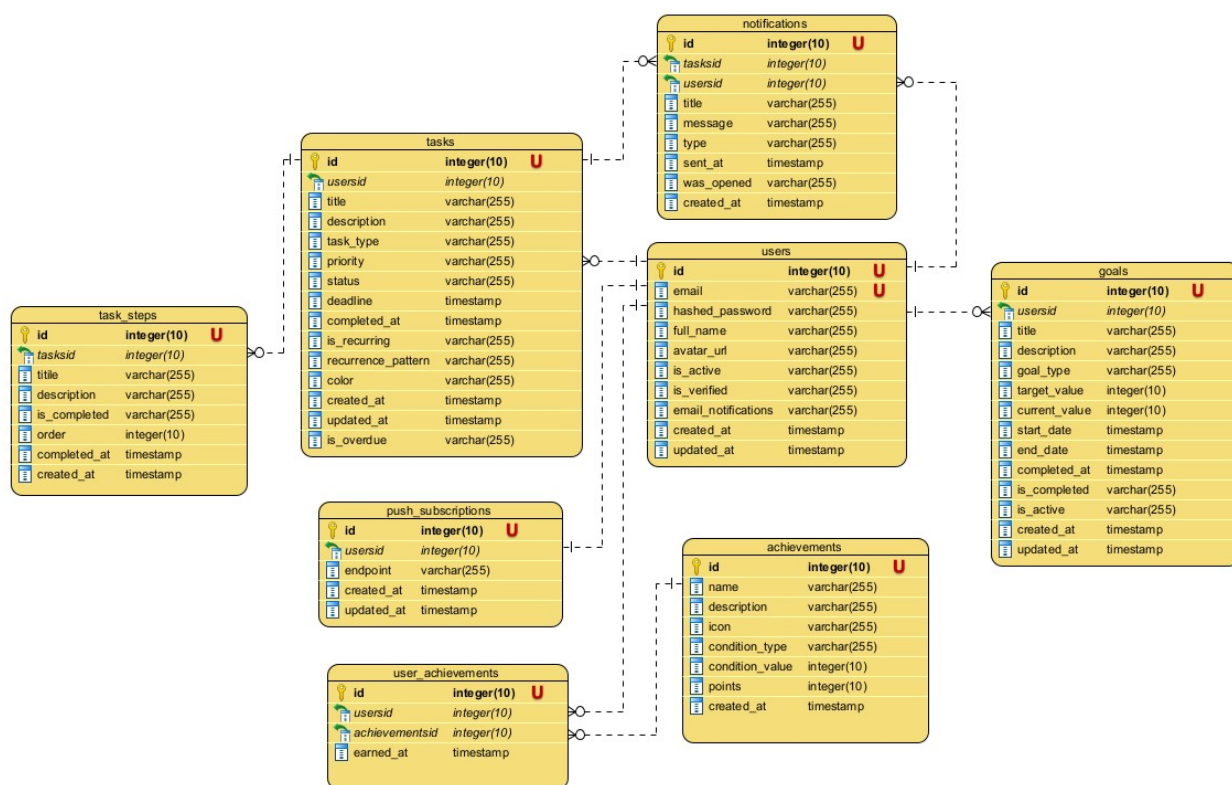


Рисунок 16 – Логическая модель базы данных приложения

Логическая модель была построена в среде Visual Paradigm 17.0 в нотации IDEF1X.

Таким образом, построенная логическая модель позволяет детализировать структуру хранения информации, обеспечить полноту и целостность данных, подготовить основу для построения физической структуры базы данных.

2.3 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса является важнейшим этапом разработки веб-приложения. Интерфейс должен соответствовать как функциональным требованиям системы, так и принципам удобства и доступности, включая логичную навигацию, минимальное количество действий для выполнения задач и наглядное отображение данных [9].

Разрабатываемый интерфейс ориентирован на студентов и предусматривает простое и быстрое добавление учебных задач, а также их просмотр, отслеживание прогресса, напоминания о дедлайнах и

задолженностях, наглядное представление учебной активности в виде календаря. Он включает в себя страницу входа и регистрации, главную страницу с общей актуальной статистикой, ближайшими и просроченными задачами, страницу со списком задач и возможностью фильтрации, календарное представление, страница личного профиля с различными настройками и отдельная страница достижений, где представлены текущие награды пользователя и прогресс.

Интерфейс реализован с использованием фреймворка Next.js и основан на архитектуре компонентов. Каждый элемент представляет собой отдельный компонент, контролирующий свое состояние и поведение. Для стилизации используется Tailwind CSS, что обеспечивает гибкость, адаптивность и быструю настройку внешнего вида.

2.4 Проектирование классов программных средств для планирования учебных задач студентом и закрытия академических задолженностей

Серверная часть разрабатываемого приложения построена по принципам многоуровневой архитектуры, организованной согласно шаблону проектирования MVC с логическим разделением на модули и слои.

MVC расшифровывается как «модель-представление-контроллер». Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения [10].

Модель – этот компонент отвечает за данные, а также определяет структуру приложения, представление – отвечает за взаимодействие с пользователем, и контроллер – за связь между моделью и представлением.

Такое разделение обеспечивает высокую поддерживаемость кода и упрощает масштабирование системы.

На рисунке 17 представлена схема архитектуры MVC.

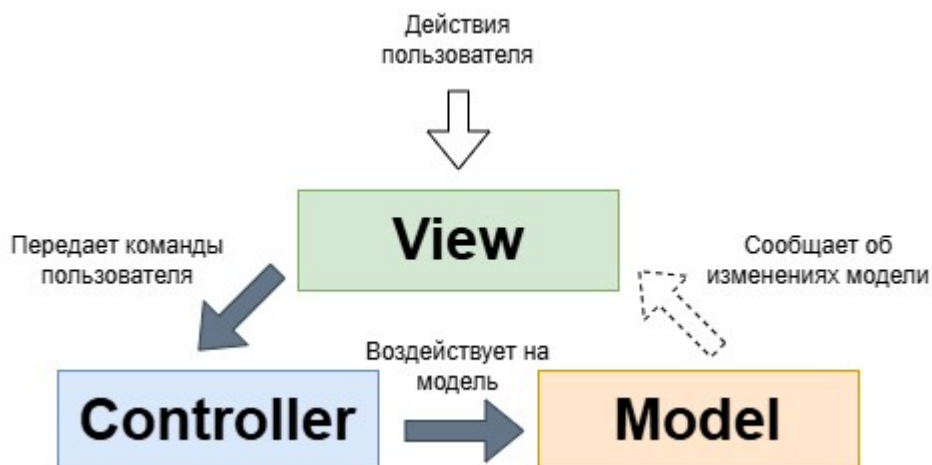


Рисунок 17 – Общая схема MVC

В ходе проектирования была разработана диаграмма классов для архитектуры разрабатываемого приложения.

Диаграмма классов описывает типы объектов системы и статические отношения, которые существуют между ними. На диаграммах классов отображаются атрибуты классов, операции и методы классов, а также ограничения, которые накладываются на связи между объектами [11].

Разработанная диаграмма отражает главные компоненты системы, а также их взаимодействие между собой. Она включает в себя модели предметной области, сервисные классы, компоненты для работы с данными и API-маршрутизаторы для работы с запросами.

Диаграмма показывает разделение ответственности между всеми компонентами системы. Модели данных определяют структуру сущностей, классы CRUD отвечают за взаимодействие с базой данных, сервисы реализуют сложную бизнес-логику, а маршрутизаторы обеспечивают API для клиентской части. Использование такой архитектуры обеспечивает разделение системы по модулям.

Диаграмма классов программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей представлена на рисунке 18.

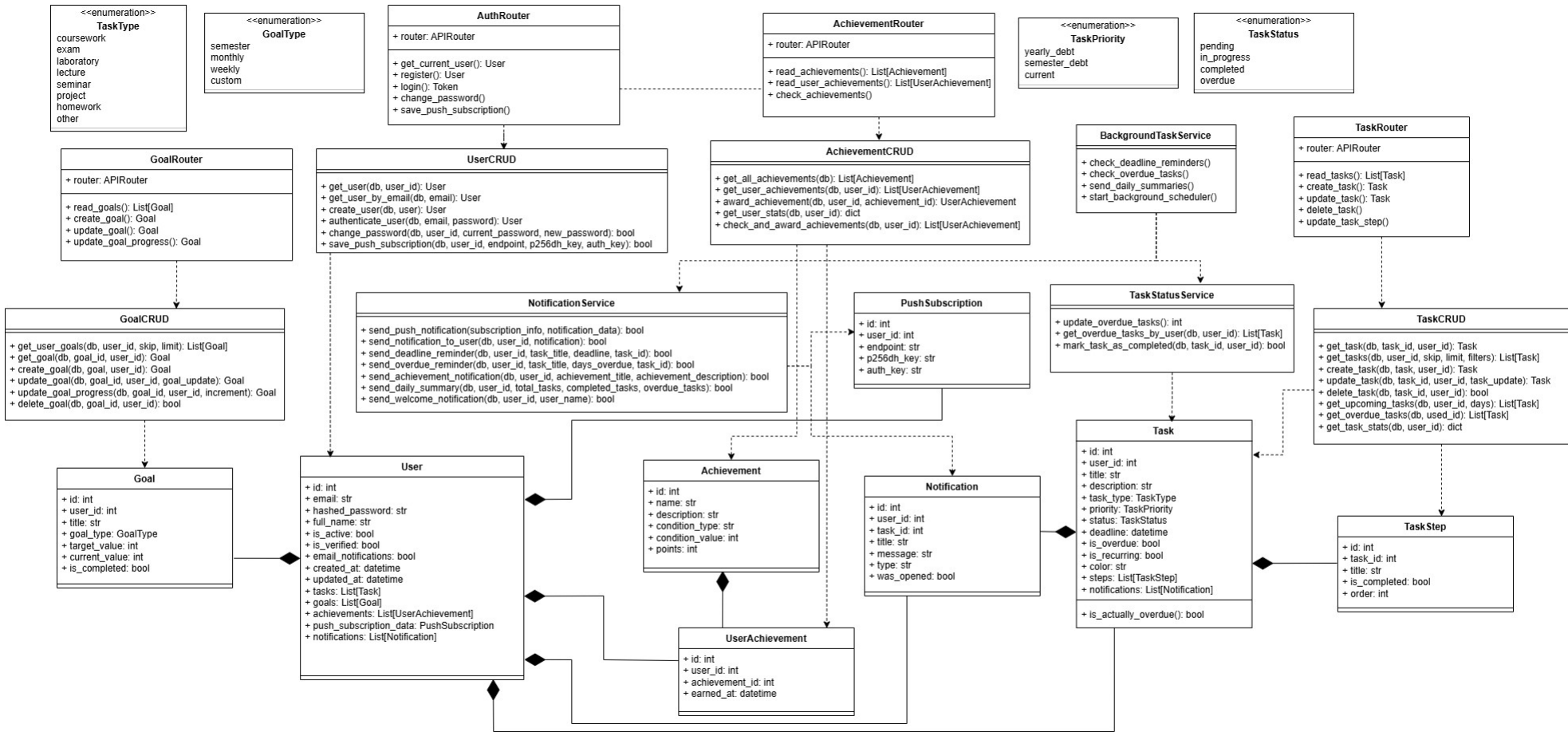


Рисунок 18 – Диаграмма классов разрабатываемого приложения

Описание функционального назначения наиболее значимых классов приложения приведено в таблице 5.

Таблица 5 – Компоненты диаграммы классов

Компонент	Назначение
User	Модель, представляющая пользователя системы, в ней содержатся основные данные учетной записи и настройки
Task	Модель учебной задачи, в ней содержатся: тип задачи, приоритет, статус, срок выполнения
TaskStep	Модель этапа учебной задачи
Goal	Модель учебной цели, в которой отображаются показатели прогресса
Achievement	Модель достижения, содержащая условия для его получения
UserAchievement	Модель достижения, которое было получено пользователем
PushSubscription	Модель подписки пользователя на пуш-уведомления
Notification	Модель уведомления
AuthRouter	Маршрутизатор, который обрабатывает запросы авторизации и аутентификации
TaskRouter	Маршрутизатор, который обрабатывает все запросы, связанные с учебными задачами
AchievementRouter	Маршрутизатор, который обрабатывает все запросы, связанные с достижениями
NotificationService	Сервис, предназначенный для отправки уведомлений пользователю
BackgroundTaskService	Сервис, предназначенный для фоновых задач
TaskStatusService	Сервис, который отвечает за управление статусом учебных задач

Построенная диаграмма классов разрабатываемого приложения дает возможность организовать структуру приложения и уточнить детали его реализации.

3 Реализация программных средств для планирования учебных задач студентом и закрытия академических задолженностей

3.1 Реализация базы данных

Физическая модель базы данных — это модель данных, которая определяет, каким образом представляются данные, и содержит все детали, необходимые СУБД для создания базы данных. В рамках разрабатываемого приложения используется СУБД PostgreSQL [12].

Физические модели данных предоставляют подробные сведения, на основе которых администраторы и разработчики баз данных воспроизводят бизнес-логику в физической базе данных. Эти модели содержат дополнительные атрибуты, которых не было в логической модели данных [13].

Согласно концептуальным и логическим моделям, была разработана физическая модель базы данных, представленная на рисунке 19.

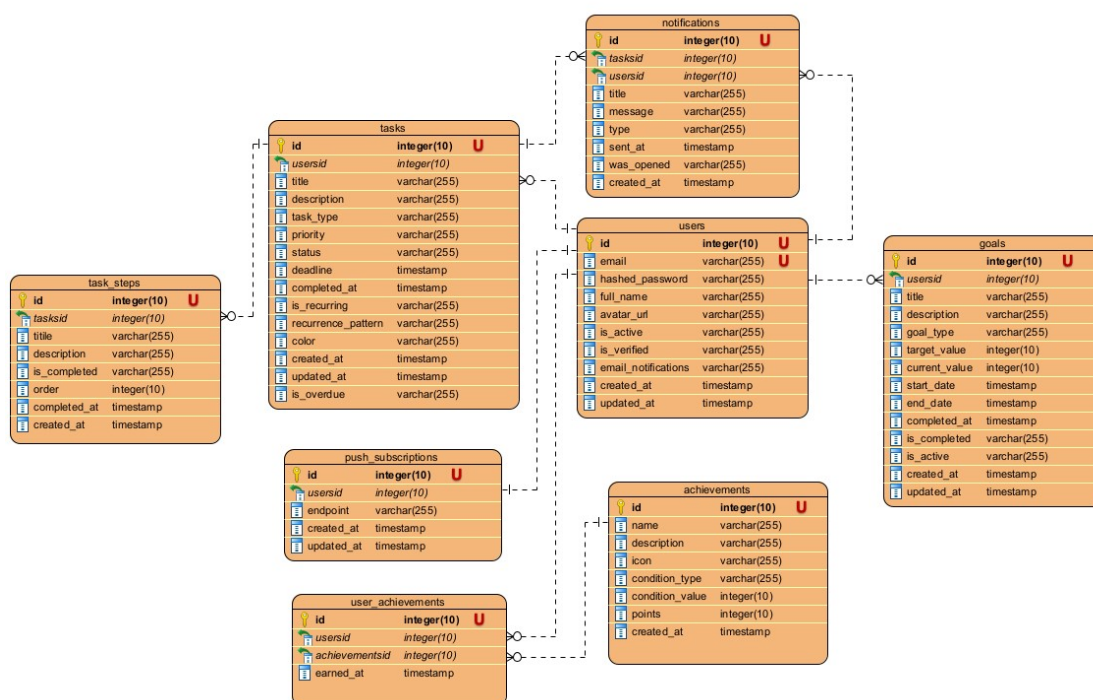


Рисунок 19 – Физическая модель базы данных приложения

Благодаря построенной физической модели, была разработана база данных в СУБД PostgreSQL. Спецификация сущности «Учебная задача» представлена в таблице 6.

Таблица 6 – Спецификация сущности «Учебная задача»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор задачи	INTEGER	Нет	> 0	—	PK
Usersid	Идентификатор пользователя	INTEGER	Нет	> 0	—	FK
Title	Название задачи	VARCHAR(255)	Нет	Любое	—	—
Description	Описание задачи	VARCHAR(255)	Да	Любое	—	—
task_type	Тип задачи	VARCHAR(255)	Нет	Любое	—	—
Priority	Приоритет задачи	VARCHAR(255)	Нет	Любое	—	—
Status	Статус задачи	VARCHAR(255)	Нет	Любое	—	—
Deadline	Срок выполнения	TIMESTAMP	Нет	Любое	—	—
completed_at	Время завершения	TIMESTAMP	Да	Любое	—	—
is_reccuring	Повторяема ли задача	BOOLEAN	Нет	True/False	—	—
recurrence_pattern	Шаблон повторения	VARCHAR(255)	Да	Любое	—	—
Color	Цвет задачи	VARCHAR(255)	Нет	Любое	—	—
created_at	Дата создания	TIMESTAMP	Нет	Любое	—	—

Продолжение таблицы 6

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
updated_at	Дата обновления	TIMESTAMP	Нет	Любое	—	—
is_overdue	Просрочена ли задача	BOOLEAN	Нет	True/False	—	—

В таблице 7 представлена спецификация сущности «Пользователь».

Таблица 7 – Спецификация сущности «Пользователь»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор пользователя	INTEGER	Нет	> 0	—	ПК
Email	Электронная почта	VARCHAR(255)	Нет	Любое	—	—
hashed_password	Пароль	VARCHAR(255)	Нет	Любое	—	—
full_name	Полное имя	VARCHAR(255)	Нет	Любое	—	—
avatar_url	Ссылка на аватар	VARCHAR(255)	Да	Любое	—	—
is_active	Активен ли пользователь	BOOLEAN	Нет	True/False	—	—
is_verified	Подтвержденность аккаунта	BOOLEAN	Нет	True/False	—	—
email_notifications	Включены ли уведомления	BOOLEAN	Нет	True/False	—	—
created_at	Дата создания	TIMESTAMP	Нет	Любое	—	—
updated_at	Дата обновления	TIMESTAMP	Нет	Любое	—	—

В таблице 8 представлена спецификация сущности «Достижение».

Таблица 8 – Спецификация сущности «Достижение»

Наименование	Описание	Тип данных	Возможность значения NULL	ОДЗ	Значение по умолчанию	Роль
Id	Идентификатор достижения	INTEGER	Нет	> 0	–	PK
Name	Название пользователя	VARCHAR(255)	Нет	Любое	–	–
Description	Описание достижения	VARCHAR(255)	Нет	Любое	–	–
Icon	Иконка	VARCHAR(255)	Нет	Любое	–	–
condition_type	Тип условия	VARCHAR(255)	Нет	Любое	–	–
condition_value	Значение условия	INTEGER	Нет	> 0	–	–
Points	Количество очков	INTEGER	Нет	> 0	–	–
created_at	Дата создания	TIMESTAMP	Нет	Любое	–	–

Таким образом, реализация физической модели базы данных позволяет хранить все сведения об учебных задачах, задолженностях, пользователях, достижениях и уведомлениях.

3.2 Реализация серверной части приложения

Серверная часть PWA-приложения была разработана с использованием языка программирования Python и фреймворка FastAPI, который обеспечивает простоту в создании RESTful API. Для взаимодействия с базой данных в СУБД PostgreSQL использовалась библиотека SQLAlchemy. Разработка велась в среде программирования JetBrains PyCharm. Код фрагмента серверной части программы приведен в приложении Б. Полный исходный код проекта доступен в репозитории на GitHub [14].

Серверная часть включает в себя следующие элементы:

- модели данных, которые отражают структуру таблиц базы данных и включают в себя сущности, которые служат основой для проектирования базы данных;
- схемы, обеспечивающие валидацию данных, сериализацию при обмене между клиентом и сервером;
- CRUD-операции, которые реализованы для каждой из сущностей, они выполняются через соответствующие модули;
- API-методы, которые дают доступ к функционалу серверной части через REST API.

Примеры основных API-методов разработанного приложения приведены в таблице 9.

Таблица 9 – Методы API программных средств для планирования и контроля учебных задач студентом и закрытия академических задолженностей

URL запроса	HTTP метод	Описание
/api/v1/auth/register	POST	Регистрация пользователя
/api/v1/auth/login	POST	Авторизация пользователя
/api/v1/tasks	GET	Получение списка учебных задач
/api/v1/tasks	POST	Создание новой учебной задачи
/api/v1/tasks/{task_id}	PUT	Обновление учебной задачи
/api/v1/tasks/{task_id}	DELETE	Удаление учебной задачи
/api/v1/achievements/user	GET	Получение достижений пользователя
/api/v1/achievements/check	POST	Присвоение новых достижений

Для обеспечения безопасности при использовании приложения реализована аутентификация через JSON Web Tokens, хеширование паролей, а также валидация входных данных.

Приложение разворачивается в Docker-контейнерах. Это упрощает его перенос и дает возможность запуска на различных платформах, docker-контейнеры обеспечивают легковесную виртуализацию приложения, упаковывая код, зависимости и библиотеки.

3.3 Реализация клиентской части приложения

Клиентская часть программных средств планирования и контроля учебных задач студентом и закрытия академических задолженностей разработана с использованием языка программирования TypeScript и фреймворка Next.js, для стилизации использовался фреймворк Tailwind CSS. Сборка клиентской части осуществлялась с помощью встроенной конфигурации WebPack.

Диаграмма компонентов – это структурная диаграмма языка унифицированного моделирования, которая описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами [15].

На рисунке 20 представлена диаграмма компонентов клиентской части разрабатываемых программных средств.

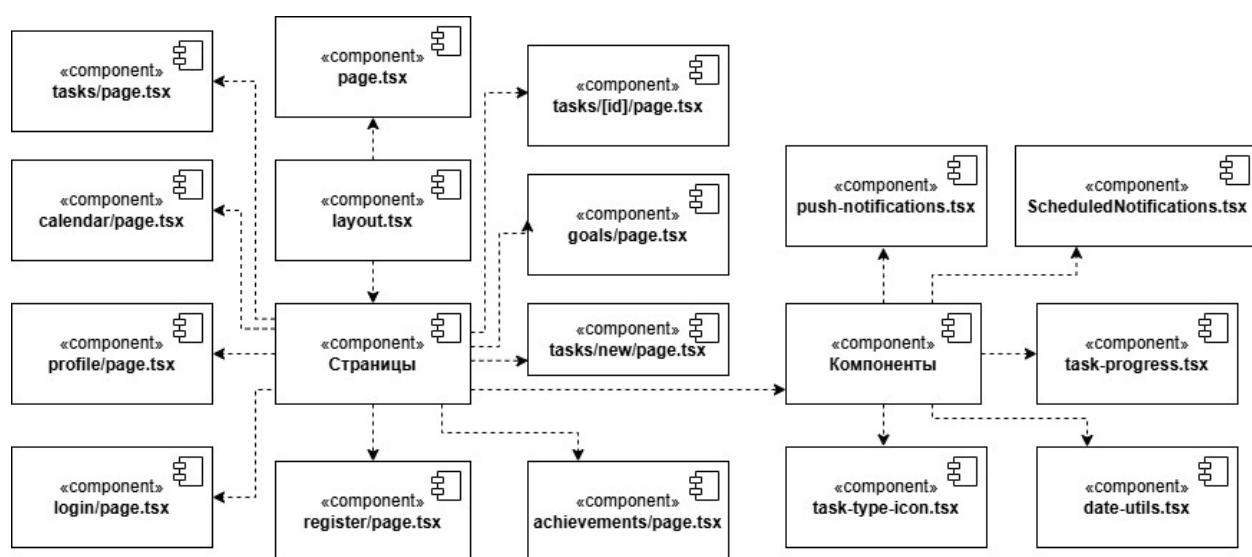


Рисунок 20 – Диаграмма компонентов клиентской части

Разработанная диаграмма демонстрирует модульную архитектуру, где каждый из компонентов управляет своей функциональной областью. Компоненты страниц отвечают за отображение контента, а другие компоненты определяют конкретную бизнес-логику.

Более подробное описание компонентов, показанных на диаграмме, представлено в таблице 10.

Таблица 10 – Компоненты клиентской части приложения

Компонент	Назначение
layout.tsx	Корневой компонент, содержащий в себе маршрутизацию.
page.tsx	Главная страница, отображающая статистику и ближайшие задачи.
tasks/page.tsx	Компонент, реализующий страницу со списком всех задач, а также фильтрацию.
tasks/{id}/page.tsx	Компонент, реализующий детальный просмотр и редактирование учебной задачи.
tasks/new/page.tsx	Компонент, реализующий функционал создания новой учебной задачи
calendar/page.tsx	Компонент, реализующий функционал календаря задач
goals/page.tsx	Компонент управления учебными целями
achievement/page.tsx	Компоненты, связанные с достижениями
profile/page.tsx	Компоненты профиля пользователя и настройки
login/page.tsx	Компоненты авторизации пользователя
register/page.tsx	Компоненты регистрации пользователя
PushNotifications.tsx	Компонент, который отвечает за управление подпиской на пуш-уведомления
ScheduledNotifications.tsx	Компонент, который служит для настройки запланированных уведомлений
TaskProgress.tsx	Компонент, который отображает прогресс выполнения учебной задачи

Для взаимодействия клиентской части с серверной использовался HTTP-клиент Axios. Выбор этого инструмента обусловлен простотой использования, автоматическим преобразованием данных и поддержкой перехватчиков для запросов [16].

3.4 Реализация интерфейса приложения

В соответствии с предъявляемыми требованиями к пользовательскому интерфейсу, описанными в пункте 2.3, в рамках разработки программных

средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей были разработаны следующие страницы:

- страница авторизации;
- страница регистрации;
- главная страница;
- страница со списком задач;
- страница создания новой задачи;
- страница календаря задач;
- страница профиля пользователя;
- страница достижений.

При открытии веб-приложения пользователь сначала попадает на страницу авторизации. Интерфейс этой страницы отображен на рисунке 21.

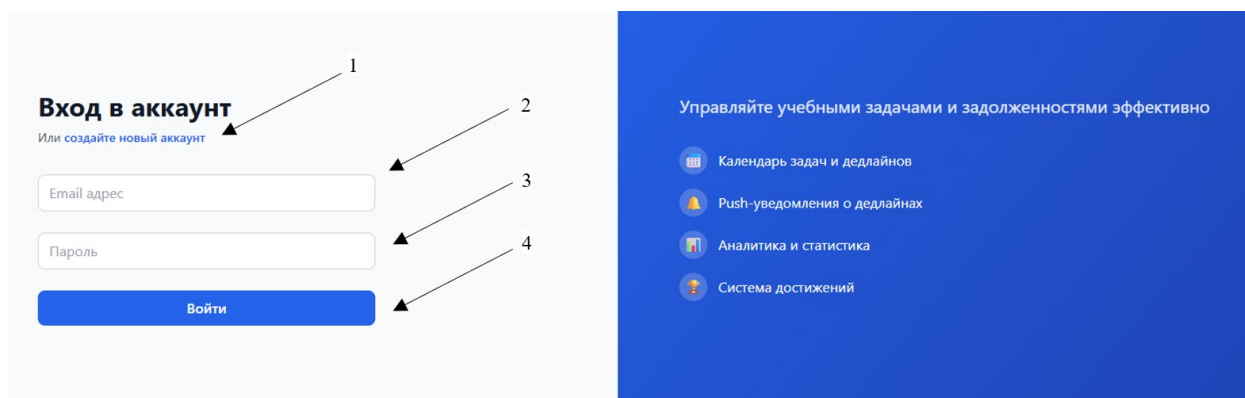


Рисунок 21 – Страница авторизации

Основные элементы страницы авторизации:

- 1 – кнопка для перехода на страницу регистрации;
- 2 – поле для ввода электронной почты;
- 3 – поле для ввода пароля;
- 4 – кнопка «Войти».

При нажатии на кнопку «Войти» система проверяет введенные пользователем данные электронной почты и пароля на правильность, а также на наличие пользователя с такими данными в базе данных. При нахождении совпадения, пользователь попадает на главную страницу приложения.

Справа от формы для авторизации расположена краткая информация о функциональности приложения. При нажатии на кнопку для перехода на страницу регистрации, пользователь попадает на соответствующую страницу. Справа от формы для регистрации нового пользователя расположено приветствие и краткое описание функционала приложения.

Интерфейс страницы регистрации изображен на рисунке 22.

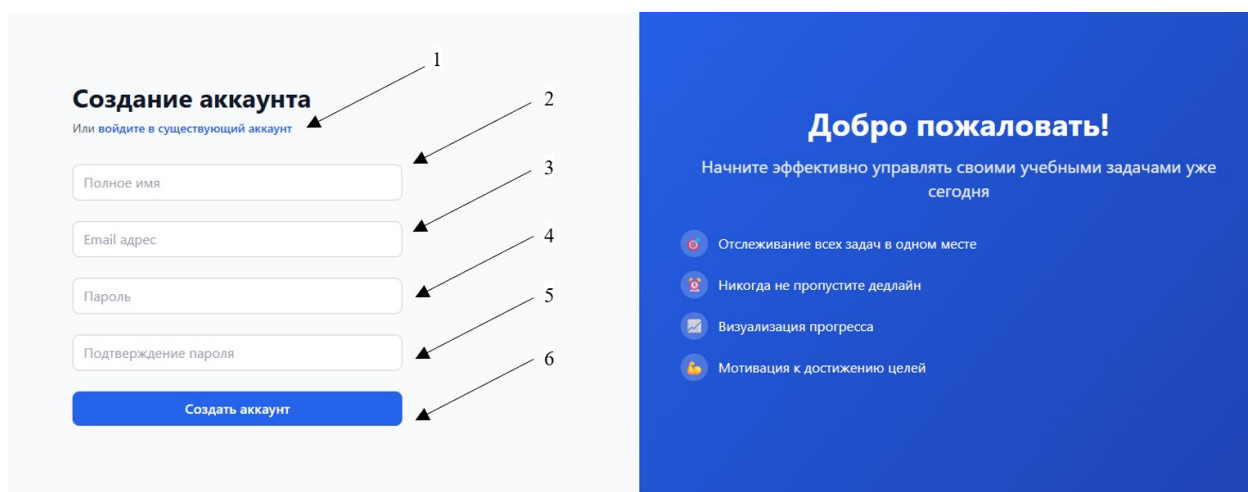


Рисунок 22 – Страница регистрации

Основные элементы страницы регистрации:

- 1 – кнопка для перехода на страницу авторизации;
- 2 – поле для ввода полного имени пользователя;
- 3 – поле для ввода электронной почты;
- 4 – поле для ввода пароля;
- 5 – поле для подтверждения пароля;
- 6 – кнопка для создания аккаунта.

При нажатии на кнопку «Зарегистрироваться» система проверяет введенные пользователем данные на правильность, а также на наличие пользователя с такими данными в базе данных. При нахождении совпадения электронной почты, пользователю выдается предупреждение, что такой пользователь уже существует в системе. Если такое совпадение не нашлось, пользователь успешно регистрируется в системе, введенные им данные заносятся в базу данных, и он попадает на главную страницу приложения. В

случае некорректного ввода одного из полей, пользователь получает предупреждение.

Главная страница приложения представлена на рисунке 23.

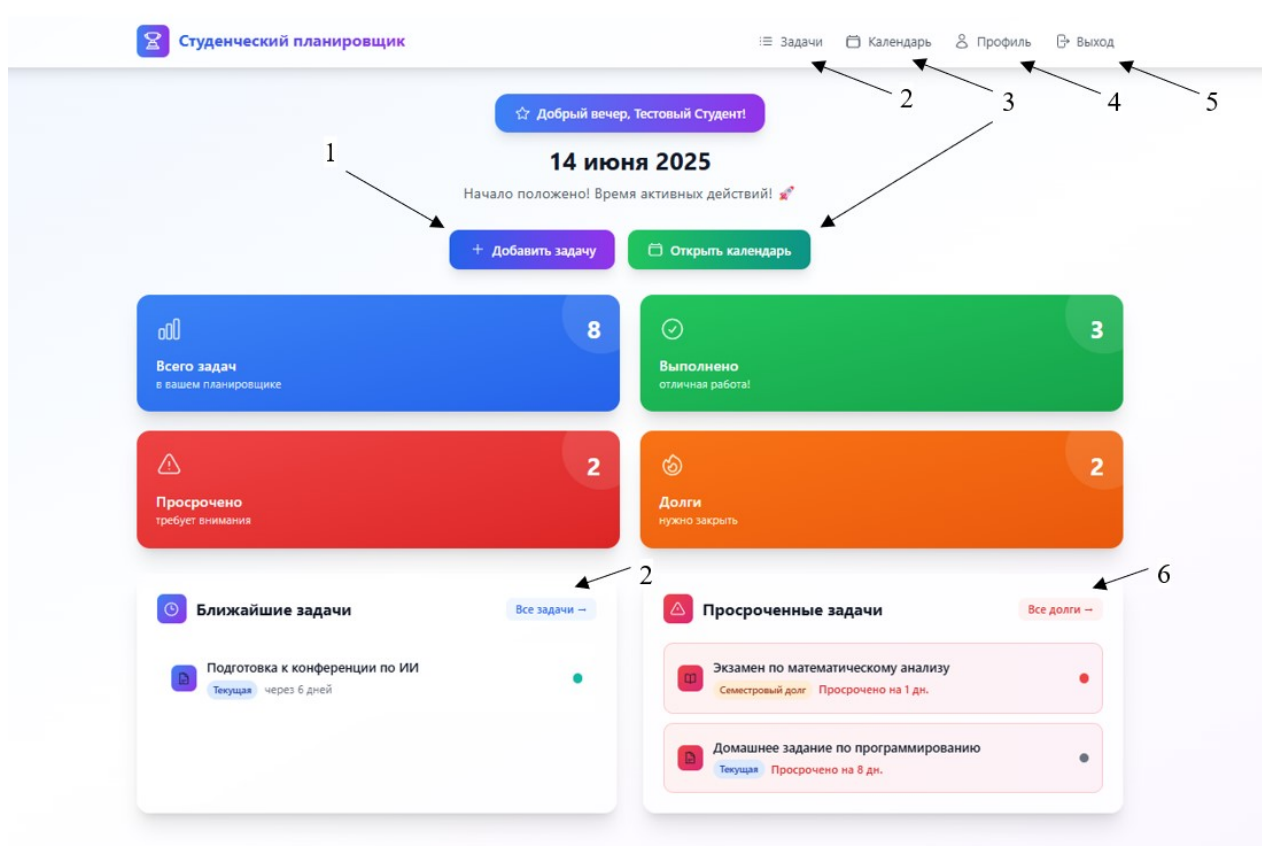


Рисунок 23 – Главная страница приложения

Основные элементы главной страницы приложения:

- 1 – кнопка для перехода на страницу с созданием новой задачи;
- 2 – кнопка для перехода на страницу со списком задач;
- 3 – кнопка для перехода на страницу календаря;
- 4 – кнопка для перехода на страницу личного профиля;
- 5 – кнопка для выхода из аккаунта;
- 6 – кнопка для перехода на страницу с долгами (задачи с фильтром).

На данной странице можно увидеть общую статистику по учебным задачам и задолженностям авторизованного пользователя, также здесь представлена навигация по всему приложению. При нажатии на соответствующие кнопки, пользователь может перейти на другие страницы

программного средства для планирования и контроля студентом учебных задач и закрытия академических задолженностей.

На рисунке 24 представлен интерфейс страницы со списком задач.

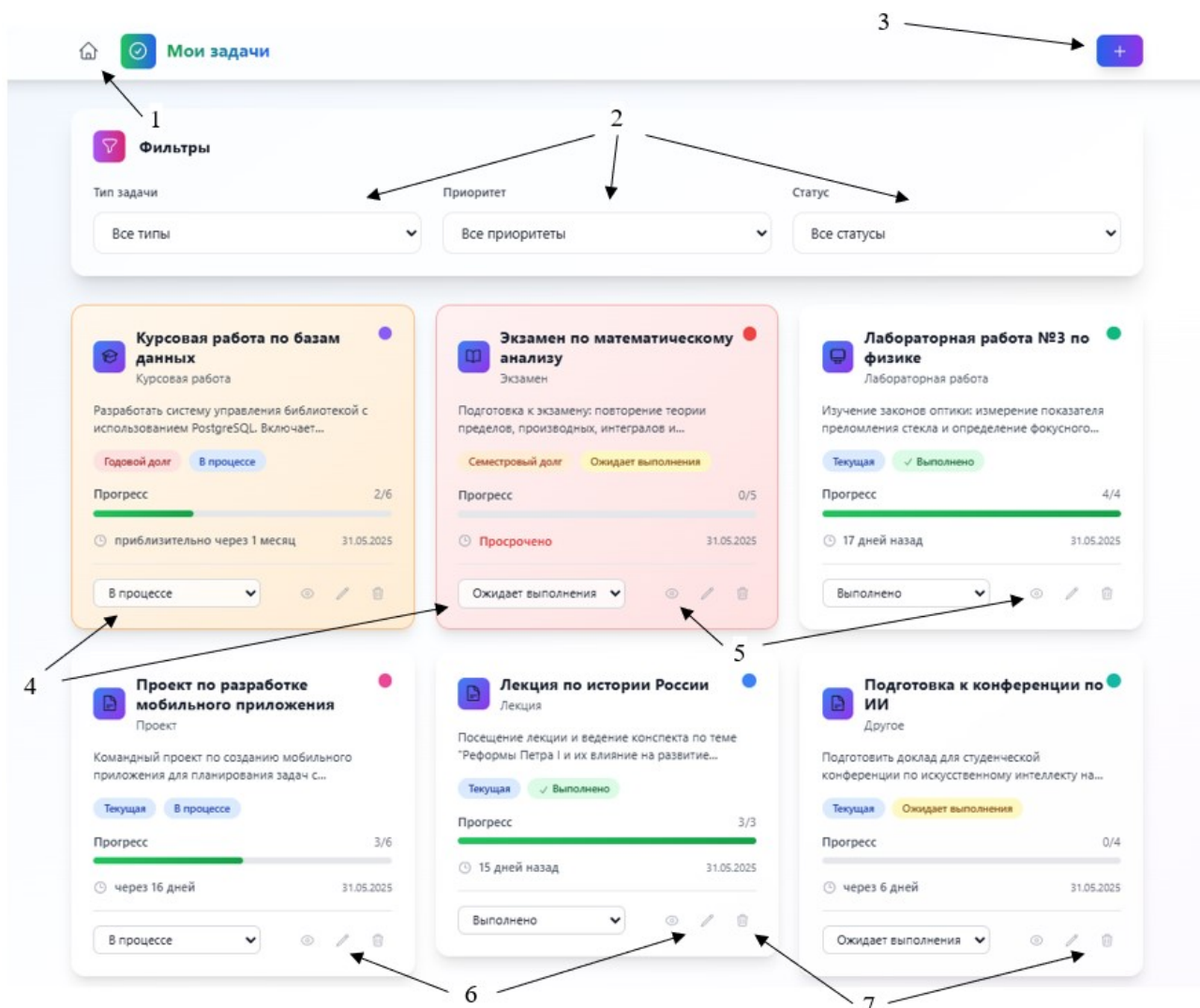


Рисунок 24 – Страница со списком учебных задач

Основные элементы страницы со списком задач:

- 1 – кнопка для перехода на главную страницу;
- 2 – выпадающие списки для фильтрации учебных задач;
- 3 – кнопка для перехода на страницу создания новой учебной задачи;
- 4 – выпадающий список для изменения статуса у конкретной задачи;
- 5 – кнопка для просмотра детальной информации конкретной задачи;
- 6 – кнопка для редактирования конкретной учебной задачи;
- 7 – кнопка для удаления учебной задачи.

На этой странице пользователь может наглядно увидеть все задачи, которые были добавлены им. Имеется возможность фильтрации задач по типу, приоритету и статусу задач. Каждый объект может редактироваться, быть удален или просмотрен, что позволяет гибко планировать свои учебные задачи.

Интерфейс детального просмотра задачи представлен на рисунке 25.

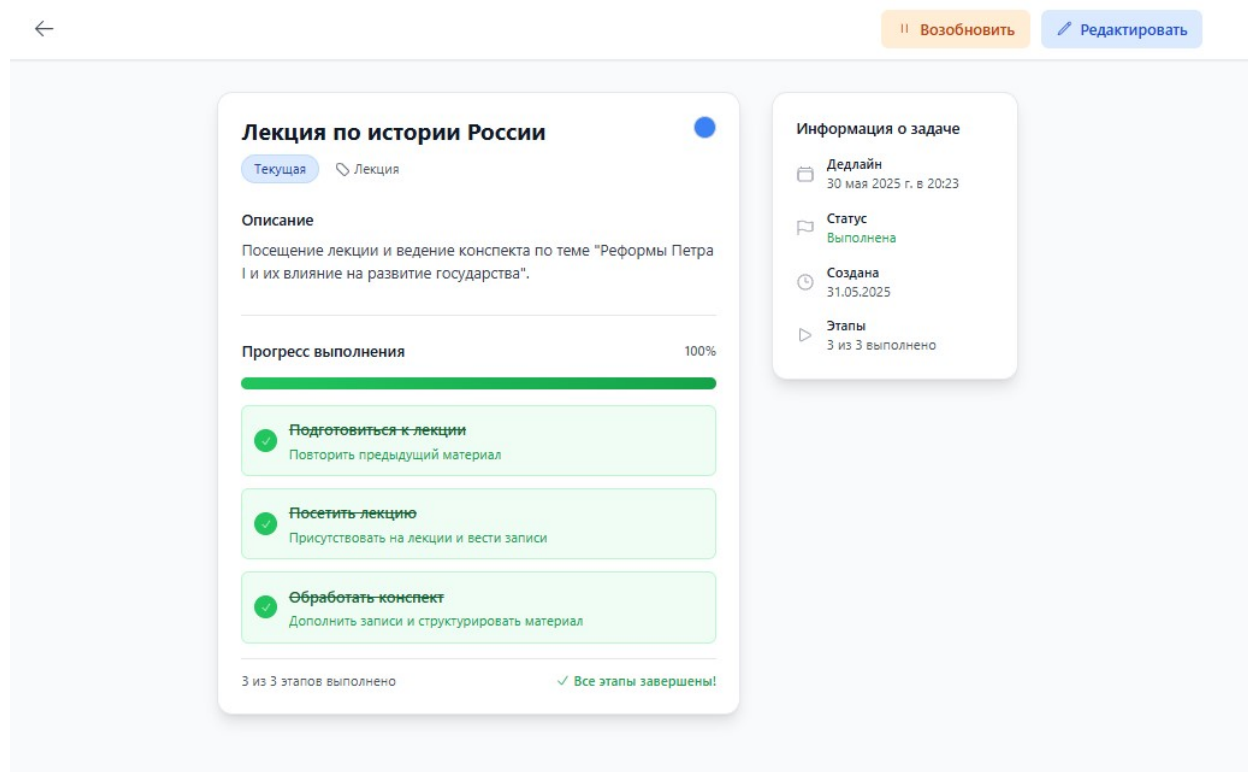


Рисунок 25 – Детальный просмотр задачи

Страница с детальным просмотром демонстрирует максимально подробную информацию об учебной задаче, включая прогресс выполнения, этапы задачи, срок выполнения, статус, тип и приоритет задачи. Также у пользователя есть возможность редактировать данную задачу, нажав соответствующую кнопку.

Интерфейс веб-страницы редактирования задачи выглядит достаточно просто и в то же время предоставляет пользователю все возможности для редактирования задачи, которые включают в себя изменение названия задачи, описания, типа задачи, приоритета, дедлайна и цвета. Форма составлена

интуитивно понятно, слева расположены поля для редактирования, а справа некоторая информация о задаче.

Веб-страница с редактированием учебной задачи представлена на рисунке 26.

Рисунок 26 – Редактирование задачи

Интерфейс страницы для создания новой учебной задачи представлен на рисунке 27. Он состоит из нескольких логических элементов, которые необходимы для заполнения данными создаваемой учебной задачи.

Рисунок 27 – Страница создания учебной задачи

Основные элементы страницы создания учебной задачи:

- 1 – поля и списки для ввода данных о задаче;
- 2 – кнопка для добавления этапа учебной задачи;
- 3 – кнопка для отмены создания учебной задачи;
- 4 – кнопка «Создать задачу».

После нажатия на кнопку «Добавить этап» в интерфейсе появляется новый блок, который необходимо заполнить данными: указать название и описание этапа.

После заполнения всех необходимых данных о задаче и нажатии на кнопку «Создать задачу», система проверяет правильность ввода и наличие обязательных данных. Если проверка пройдена успешно, новая учебная задача сохраняется в базе данных.

Помимо просмотра страницы со списком задач, приложение имеет возможность предоставить информацию об учебных задачах в удобном календарном варианте, что позволяет эффективнее планировать и контролировать учебную деятельность. Интерфейс страницы календаря в недельном представлении представлен на рисунке 28.

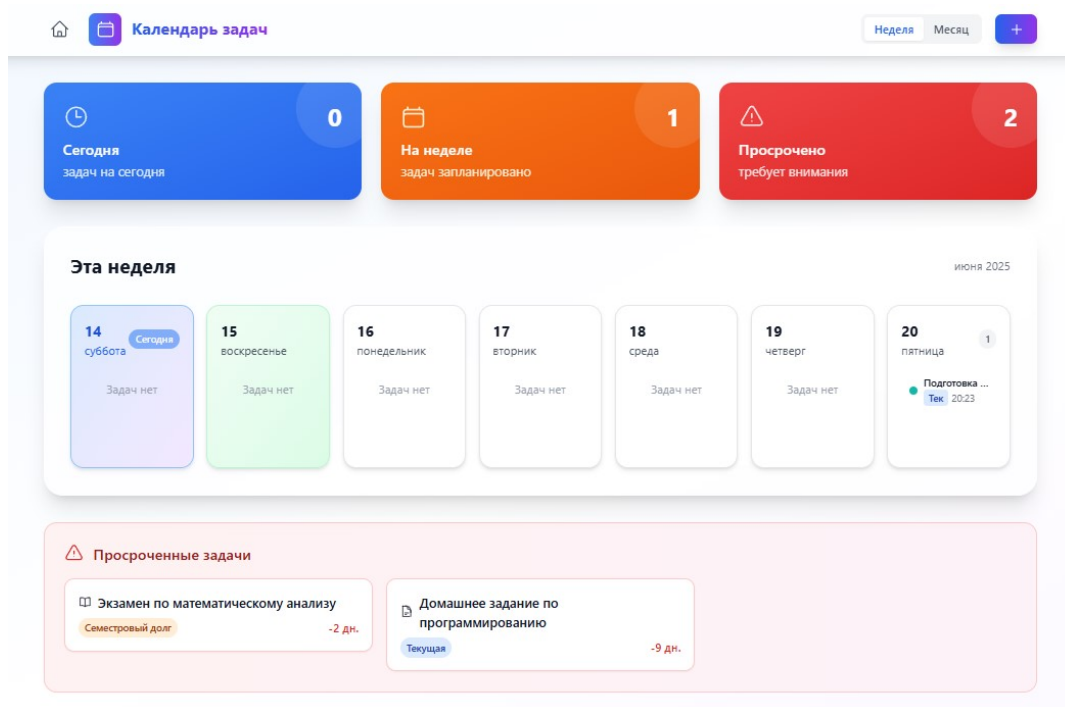


Рисунок 28 – Календарь задач с недельным представлением

Помимо недельного представления существует календарь с отображением всего месяца, интерфейс этой страницы представлен на рисунке 29.

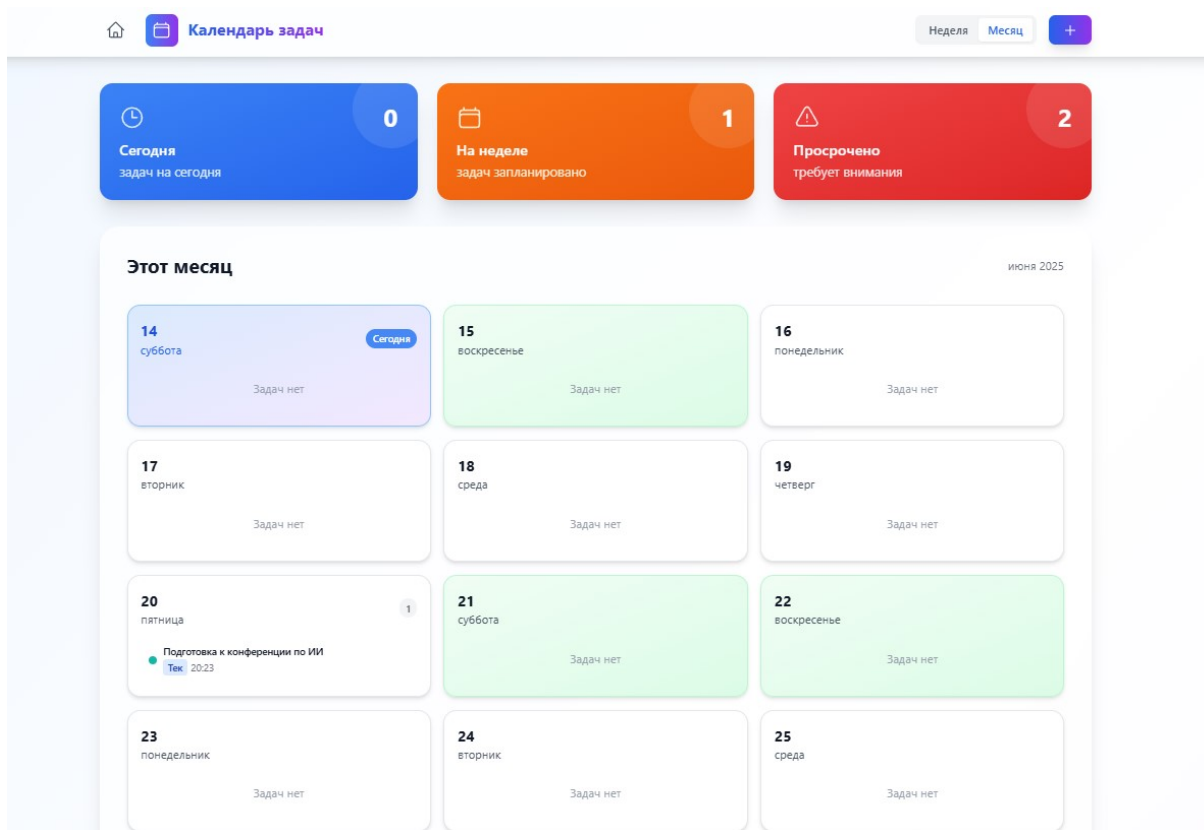


Рисунок 29 – Календарь задач с месячным представлением

Переключить представление календаря можно на верхней панели справа. На этой странице присутствует статистика по запланированным задачам, а также их календарное представление. При нажатии на название задачи, откроется страница ее детального просмотра.

Следующей рассматриваемой страницей является личная страница профиля пользователя.

Основные элементы страницы профиля пользователя:

- 1 – кнопка для перехода на страницу достижений;
- 2 – кнопка для отображения окна с обзором активности;
- 3 – кнопка для отображения окна с достижениями пользователя;
- 4 – кнопка для отображения окна с возможностью изменения личной информации пользователя;

5 – кнопка для отображения окна безопасности;

6 – кнопка для отображения окна для настройки уведомлений.

Интерфейс страницы с личным профилем пользователя представлен на рисунке 30.

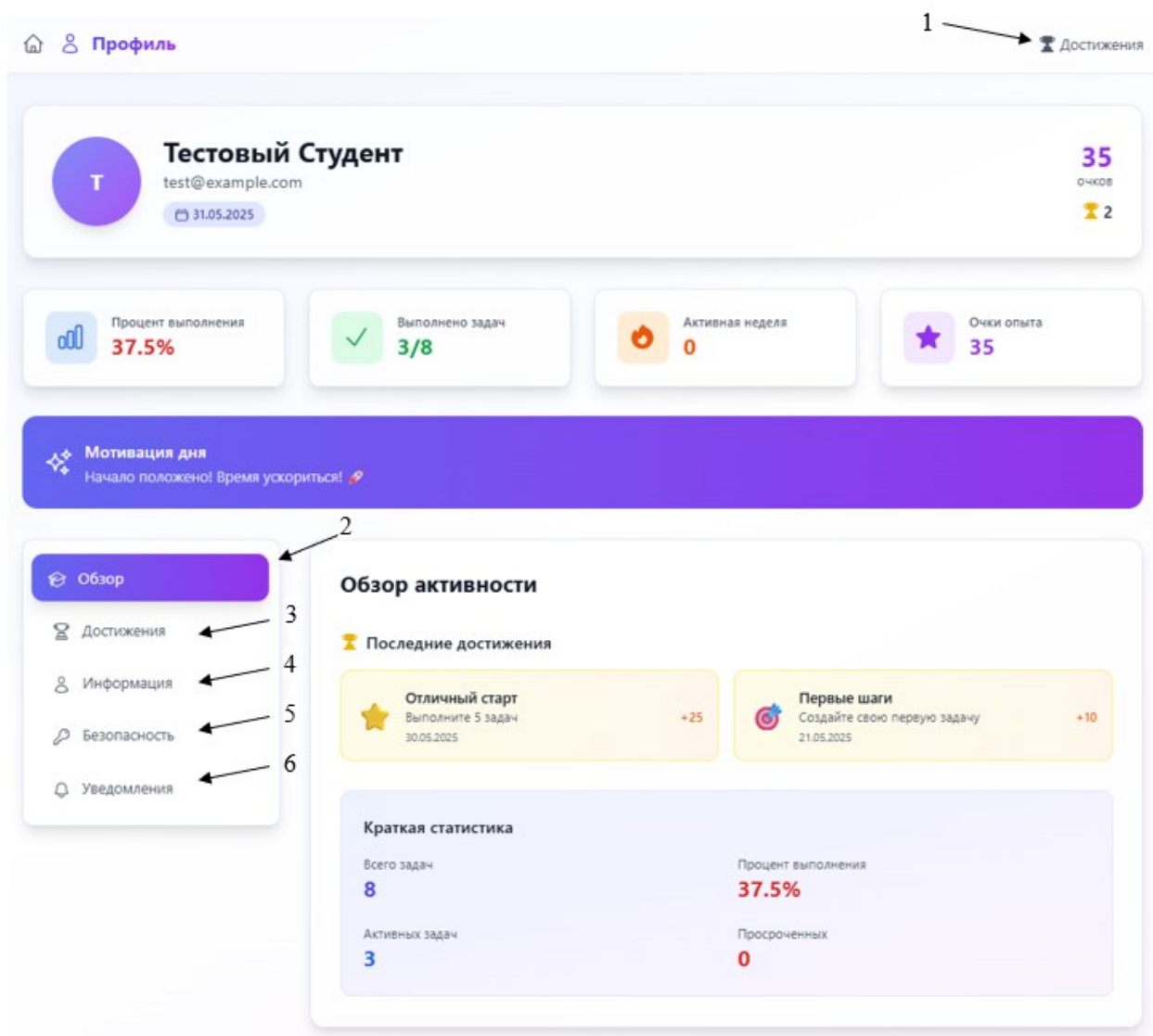


Рисунок 30 – Страница профиля пользователя

На данной странице можно увидеть всю основную информацию о профиле пользователя, а также отредактировать ее при необходимости.

Во вкладке «Обзор» можно увидеть краткую статистику пользователя по учебной активности, а также последние полученные достижения. Данная часть страницы является переключаемой, поэтому есть несколько вариантов вкладок.

Во вкладке достижения, которая представлена на рисунке 31, показаны все достижения, которые получил пользователь, и общее количество заработанных очков.

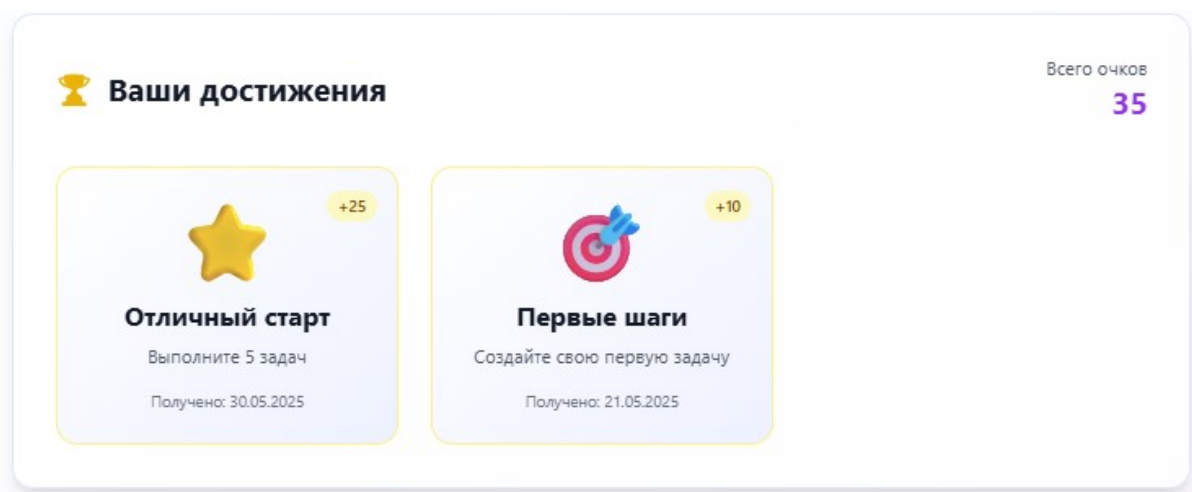


Рисунок 31 – Вкладка с достижениями

Вкладка с информацией о пользователе, представленная на рисунке 32, позволяет пользователю отредактировать свои идентификационные данные, такие как полное имя и электронная почта.

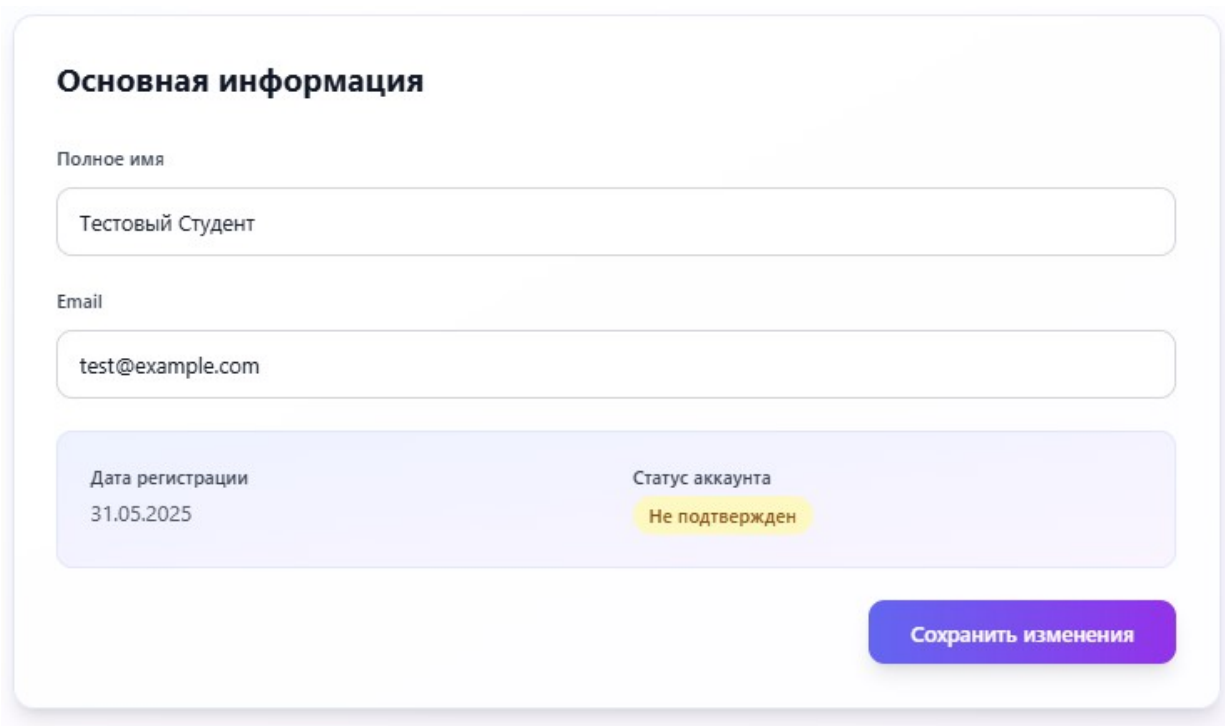
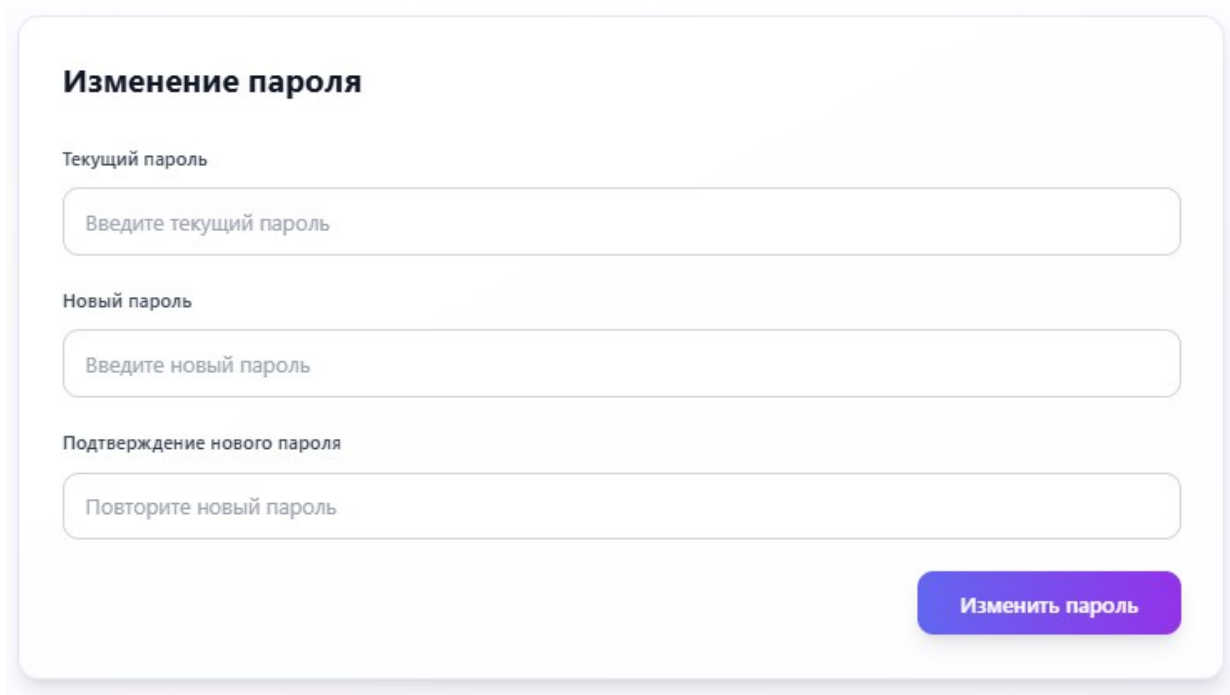


Рисунок 32 – Вкладка с основной информацией

Следующим компонентом является вкладка безопасности, с помощью которой можно изменить пароль аккаунта. Вкладка «Безопасность» изображена на рисунке 33.



The image shows a web form titled "Изменение пароля" (Change Password). It contains three input fields: "Текущий пароль" (Current password) with placeholder text "Введите текущий пароль", "Новый пароль" (New password) with placeholder text "Введите новый пароль", and "Подтверждение нового пароля" (Confirm new password) with placeholder text "Повторите новый пароль". A purple button labeled "Изменить пароль" (Change password) is located at the bottom right of the form.

Рисунок 33 – Вкладка безопасности профиля

Окно настроек уведомлений представлено на рисунке 34.

Интерфейс данной страницы включает три основные секции:

- push-уведомления;
- расписание уведомлений;
- пользовательские напоминания.

Push-уведомления предназначены для отправки напоминаний пользователю о приближении сроков выполнения учебных задач, а также предоставления ежедневных сводок. В верхней части находится кнопка для включения либо отключения уведомлений, а также краткое описание их функционала.

Пользователь может настраивать параметры получения таких уведомлений, выбирая удобное время и дни недели в средней секции страницы.

В нижней части страницы предусмотрена расширенная возможность создания пользовательских напоминаний по собственным предпочтениям.

Пользователь может добавить персонализированное уведомление со своим текстом, описанием, временем и периодичностью.

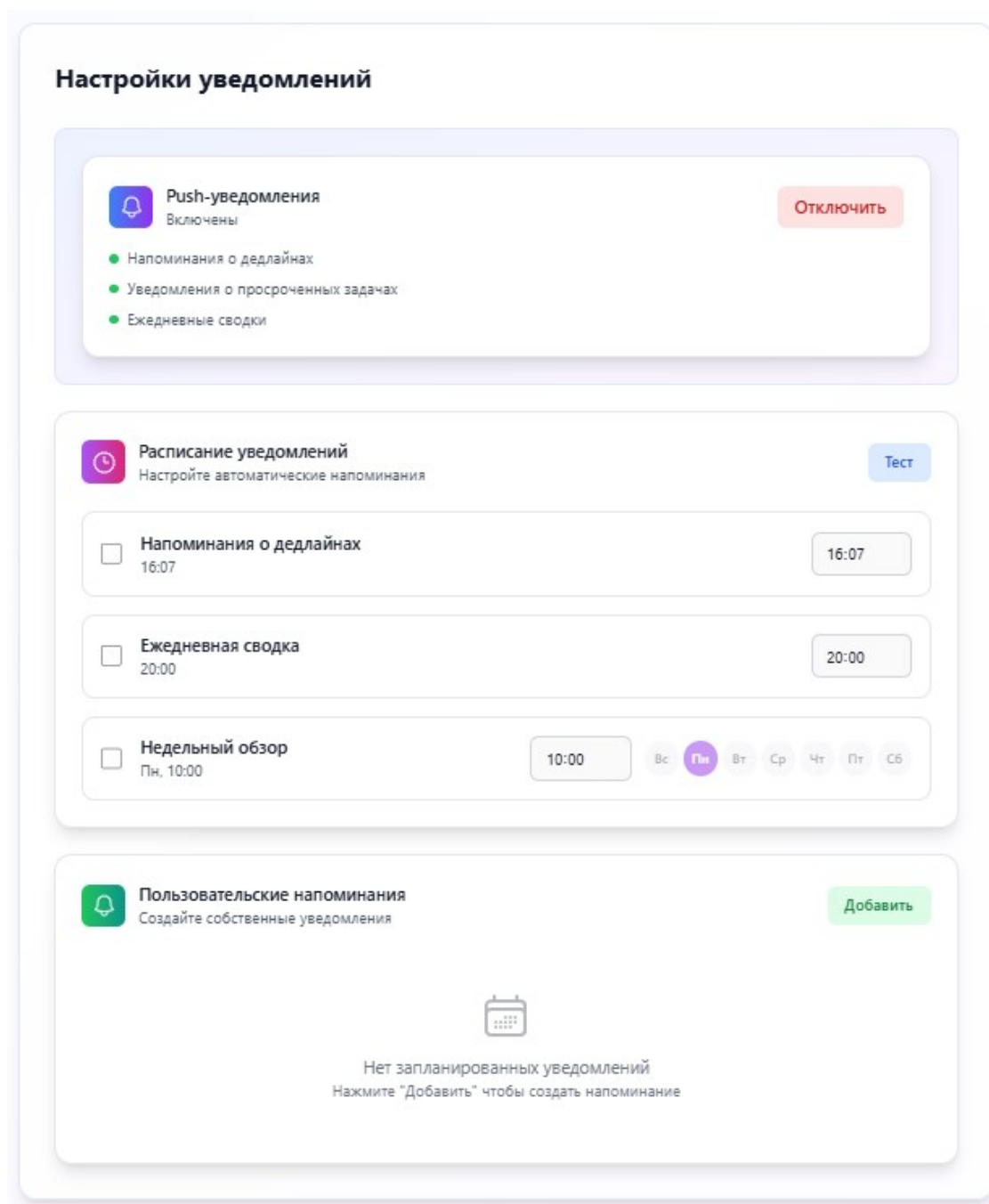


Рисунок 34 – Настройки уведомлений

Интерфейс страницы с достижениями представлен на рисунке 35. Страница разделена на несколько блоков, которые отображают информацию о достижениях пользователя и личной статистике. Также показывается количество очков опыта и достижения, которые еще предстоит получить. В верхней части страницы небольшая инструкция по системе достижений.

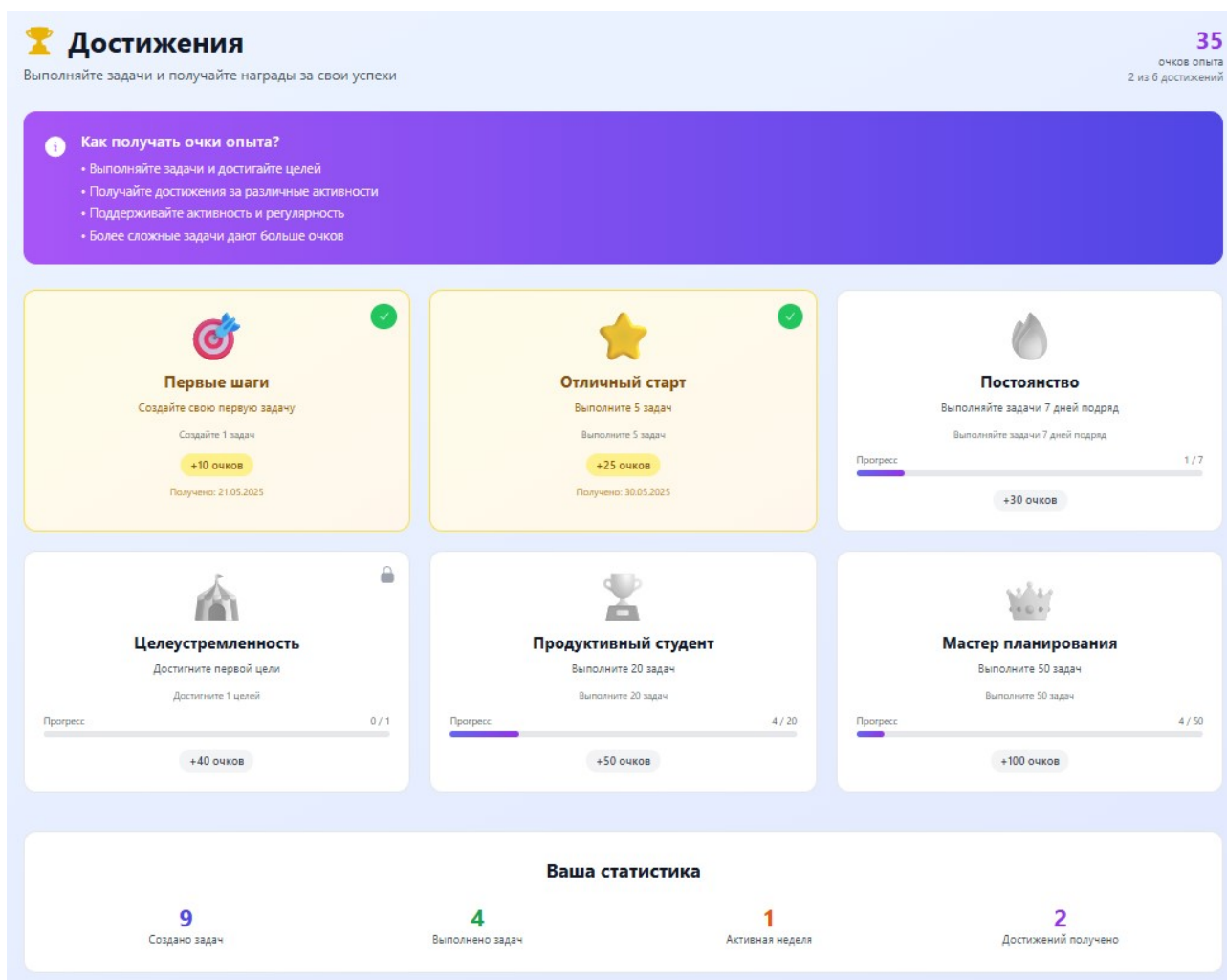


Рисунок 35 – Страница достижений

Данный интерфейс интуитивно понятен и мотивирует пользователя к выполнению поставленных учебных задач для получения достижений.

Разработка интерфейса клиентской части приложения осуществлялась с использованием современных веб-технологий, включая гипертекстовую разметку HTML5 и каскадные таблицы стилей CSS3. HTML5 применялся для формирования структуры веб-страниц, обеспечивая логичное расположение элементов интерфейса. CSS3 использовался для визуальной настройки, включая адаптивную верстку, анимации, цветовую палитру, что позволило создать единый стиль для всей системы. Для упрощения работы со стилями была задействована библиотека Tailwind CSS, которая ускорила разработку за счет готовых классов и поддержки тем оформления.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования и метрик кода

В современной практике разработки программного обеспечения существует множество видов тестирования. Среди них наиболее распространены:

- функциональное тестирование;
- модульное тестирование;
- интеграционное тестирование;
- системное тестирование;
- приемочное тестирование.

Функциональное тестирование направлено на проверку соответствия программного продукта бизнес-требованиям путем анализа конечных результатов операций без детального рассмотрения промежуточных состояний системы.

Модульное тестирование (юнит-тестирование) выполняется на уровне исходного кода и предполагает проверку отдельных методов, функций или модулей приложения. Благодаря узкой направленности такие тесты отличаются высокой скоростью выполнения и не требуют сложной автоматизации.

Для комплексной оценки взаимодействия пользователя с приложением разрабатываются тест-кейсы, детально описывающие последовательность действий для проверки конкретных функций на основе реальных пользовательских сценариев.

Интеграционное тестирование сосредоточено на проверке корректного взаимодействия и совместной работы отдельных модулей, компонентов или сервисов приложения после их объединения. Для его проведения требуется запуск нескольких взаимосвязанных частей системы одновременно.

Системное тестирование представляет собой уровень проверки полностью собранного программного продукта как единого целого. Оно

направлено на оценку соответствия системы всем заявленным функциональным и нефункциональным требованиям. Тестирование проводится с точки зрения пользователя, моделируя реальные сценарии работы [17].

Заключительной фазой в тестировании программного обеспечения является приемочное тестирование. Его цель – определить, готов ли продукт к передаче конечному пользователю. Именно поэтому оно часто проводится на стороне ключевых пользователей и фокусируется на проверке соответствия реальным бизнес-процессам, чтобы подтвердить пригодность системы для использования по назначению.

Выбор функционального тестирования в качестве основного метода обусловлен ключевой важностью проверки соответствия программы заявленным функциональным требованиям.

Кроме того, в рамках нефункционального тестирования, был выбран метод тестирования на совместимость. Тестирование совместимости оценивает способность программного обеспечения корректно функционировать в различных средах, включая операционные системы, браузеры, устройства и сетевые условия. Данный вид тестирования играет важную роль при тестировании веб-приложения, поскольку такие приложения должны обеспечивать работу на различных устройствах и браузерах [18].

Метрики кода – это числовые показатели, с помощью которых можно измерить и проанализировать свойства кода, ход разработки и общее качество программного продукта. Они дают объективную оценку, помогают выявлять слабые места и оптимизировать процесс создания ПО, что делает их полезным инструментом в работе программистов [19].

В рамках данного проекта для оценки качества кода были выбраны две наиболее популярные метрики:

- метрика Холстеда, основанная на подсчете операторов и операндов в исходном коде, а также анализе общего объема кода;
- метрика Чепина, она ориентирована на анализ логической структуры модулей.

4.2 Тестирование программных средств для планирования учебных задач студентом и закрытия академических задолженностей

4.2.1 Функциональное тестирование

После завершения разработки программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей было проведено функциональное тестирование всей системы. Результаты тестирования приведены в приложении В.

План тестирования представлен в таблице 11.

Таблица 11 – План функционального тестирования

Назначение теста	Описание теста	Результаты теста
Проверка отображения задач	Пользователь переходит на вкладку «Задачи», где отображаются добавленные задачи	Все задачи пользователя корректно отображаются (рисунок В.1)
Проверка создания задачи	Пользователь создает новую задачу с обязательными полями	Задача успешно сохраняется и отображается в списке задач (рисунок В.2)
Проверка редактирования задачи	Пользователь изменяет данные существующей задачи	Изменения сохраняются и отображаются в списке задач (рисунок В.3)
Проверка удаления задач	Пользователь удаляет задачу из списка	Задача пропадает из списка и больше не отображается (рисунок В.4)
Проверка отображения календаря	Пользователь переходит на страницу «Календарь», где задачи отображаются по датам	Задачи корректно отображаются в соответствии с датами (рисунок В.5)
Проверка фильтрации задач	Пользователь применяет фильтры (по типу, приоритету, статусу)	Список задач обновляется в соответствии с выбранными фильтрами (рисунок В.6)
Проверка отображения достижений	Пользователь переходит на страницу «Достижения»	Отображаются все существующие достижения и прогресс выполнения (рисунок В.7)
Проверка авторизации с неверным паролем	Пользователь вводит неверный пароль при входе	Система выводит сообщение «Неверный email или пароль» (рисунок В.8)

Продолжение таблицы 11

Назначение теста	Описание теста	Результаты теста
Проверка регистрации с некорректной почтой	Пользователь пытается зарегистрироваться в системе с существующим email	Система выводит сообщение «Пользователь с таким email уже существует» (рисунок В.9)
Проверка регистрации с недостающими данными	Пользователь пытается зарегистрироваться в системе с пустыми полями	Система выводит предупреждения над каждым полем (рисунок В.10)
Проверка включения уведомлений	Пользователь включает push-уведомления в настройках профиля	Система включает уведомления (рисунок В.11)

Программа успешно прошла все функциональные тесты, показав правильность работы программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей и полное соответствие функциональным требованиям.

4.2.2 Тестирование на совместимость системы

Тестирование на совместимость представляет собой важный этап проверки программного обеспечения, направленный на обеспечение его корректной работы в различных средах.

Большую значимость такое тестирование приобретает для веб-приложений, так как пользователи могут обращаться к системе из самых разных окружений. Особое внимание уделяется кросс-браузерной совместимости, где ключевым условием является гарантия одного и того же поведения интерфейса и в различных браузерах, и на разных операционных системах.

Проверялась не только визуальная составляющая интерфейса, но и корректность выполнения главных сценариев.

В таблице 12 приведены результаты тестирования на совместимость программных средств для планирования и контроля студентом учебных задач и закрытия академических задолженностей.

Таблица 12 – Результаты тестирования совместимости

Устройство/Браузер	Сценарий	Результат	Тест пройден/не пройден
Windows 10, Yandex 25.4	Регистрация в системе, создание задачи, просмотр календаря, просмотр достижений	Регистрация прошла успешно, задача создана, календарь отобразился корректно, достижения отображаются корректно	Тест пройден
Windows 11, Google Chrome 137.0	Авторизация, создание задачи, работа с фильтрами задач	Авторизация прошла успешно, задача создана, фильтры применены успешно	Тест пройден
Windows 11, Microsoft Edge 110.0	Авторизация, редактирование задачи, удаление задачи	Авторизация прошла успешно, задача отредактирована, а затем удалена	Тест пройден
Ubuntu 22.04, Firefox 115	Регистрация в системе, просмотр задач, изменение статуса задачи	Регистрация прошла успешно, задачи корректно отображены, статус задачи изменен	Тест пройден
macOS 15.5, Safari 18	Просмотр календаря, проверка уведомлений, просмотр задач	Календарь отобразился корректно, уведомления приходят, задачи отображаются корректно	Тест пройден
Android 15.0, Chrome 126	Регистрация в системе, просмотр задачи, редактирование задачи	Регистрация прошла успешно, задачи отображаются корректно и успешно редактируются	Тест пройден

Результаты тестирования подтвердили полную функциональную и визуальную совместимость приложения со всеми заявленными браузерами и операционными системами.

4.3 Оценка качества программного кода

4.3.1 Метрика Холстеда

Для расчета метрик Холстеда рассматривается функция `register`. Данная функция отвечает за процесс регистрации новых пользователей в системе, она проверяет, есть ли пользователь с такой электронной почтой в базе данных. Код данной функции представлен в листинге 1.

Листинг 1 – Код функции регистрации пользователей

```
@router.post("/register", response_model=User)
def register(user: UserCreate, db: Session = Depends(get_db)) ->
Any:
    """Регистрация нового пользователя"""
    db_user = crud_user.get_user_by_email(db, email=user.email)
    if db_user:
        raise HTTPException(
            status_code=400,
            detail="Пользователь с таким email уже существует"
        )
    db_user = crud_user.create_user(db=db, user=user)
    return db_user
```

Операторы функции register и число их вхождений приведены ниже в таблице 13.

Таблица 13 – Операторы функции register

№	Операторы	Число вхождений
1	=	5
2	if	2
3	raise	1
4	return	1
5	==	1
6	.	4
7	()	3
8	:	2
Итого	n2 = 8	N2 = 19

Операнды функции register, а также число их вхождений приведены в таблице 14.

Таблица 14 – Операнды функции register

№	Операнды	Число вхождений
1	register	1
2	user	3
3	db	4
4	db user	3
5	email	2
6	status_code	1
7	detail	1
8	create_user	1
9	user.email	1
10	n2 = 10	N2 = 17

Данные, полученные в ходе обработки программного кода функции

регистрации пользователя, были использованы для расчета характеристик программы по формулам из метрики Холстеда. Этот метод позволяет количественно оценить сложность программного кода, анализируя используемые операторы и операнды.

Результаты вычислений представлены в таблице 15.

Таблица 15 – Результаты вычислений по метрике Холстеда

Значение	Результат
Длина реализации программы	36
Словарь программы	18
Длина программы	57.2
Объем программы	150.12
Потенциальный объем программы	42.96
Теоретический уровень программы	0.286
Сложность программы	3.5
Работа программирования	42.89

Все полученные показатели части программы находятся в пределах нормы и удовлетворяют требованиям по метрикам программного кода: длина реализации меньше длины программы, следовательно, функция реализована без лишних операций; теоретический уровень программы весьма низкий, что подтверждает простую и понятную реализацию; возможности языка программирования использованы на достаточном уровне [20].

4.3.2 Метрика Чепина

Расчет метрики Чепина произведен над функцией, который отвечает за авторизацию пользователя. Исходный код данного компонента представлен в листинге 2.

Листинг 2 – Функция авторизации пользователя

```
@router.post("/login", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends(), db:
Session = Depends(get_db)) -> Any:
```

```

    """Авторизация пользователя"""
    user = crud_user.authenticate_user(db,
    email=form_data.username, password=form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверный email или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )
    elif not user.is_active:
        raise HTTPException(status_code=400, detail="Аккаунт не
активен")

    access_token_expires =
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = security.create_access_token(
        data={"sub": user.email},
    expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

```

Методика расчета метрики Чепина предполагает комплексный анализ всех переменных, участвующих в работе программного модуля, с последующим их распределением по четырем функциональным категориям. К первой группе (Р) относятся входные переменные, используемые для вычислений и формирования результата. Вторая категория (М) включает переменные, значения которых изменяются в процессе выполнения программы. Третья группа (С) содержит элементы, управляющие потоком выполнения кода. Четвертый тип (Т) объединяет неиспользуемые («паразитные») переменные, присутствующие в коде, но не влияющие на его работу.

Формула расчета имеет вид:

$$Q = a1 \times P + a2 \times M + a3 \times C + a4 \times T,$$

где весовые коэффициенты отражают степень влияния каждой категории на общую сложность кода. Наибольший вес присвоен управляющим переменным, так как они непосредственно определяют логику выполнения программы. Для модифицируемых переменных установлен коэффициент $a2=2$, входные данные учитываются с весом $a1=1$. Даже неиспользуемые переменные имеют ненулевой коэффициент ($a4=0.5$), поскольку их наличие может затруднять понимание кода.

Применительно к анализируемому модулю авторизации было выявлено пять входных переменных, три модифицируемых параметра, а также два управляющих элемента. Неиспользуемые переменные отсутствуют.

Подстановка этих значений в формулу дает результат:

$$Q = 5 + 2 \times 3 + 3 \times 2 = 17,$$

что свидетельствует о среднем уровне сложности, характерном для надежной системы аутентификации, сочетающей необходимые проверки безопасности с достаточно простой и понятной структурой реализации [20].

Заключение

В рамках выпускной квалификационной работы спроектированы и разработаны программные средства для планирования и контроля студентом учебных задач и закрытия академических задолженностей. Клиентская часть приложения реализована с использованием языка программирования TypeScript и фреймворка Next.js 14, что обеспечило высокую производительность и удобство взаимодействия с пользователем. Серверная часть разработана на языке Python с применением фреймворка FastAPI, что позволило создать надежное и масштабируемое REST API. Для хранения данных использовалась СУБД PostgreSQL, обеспечивающая целостность и безопасность информации.

На этапе анализа требований проведено исследование предметной области, выполнен сравнительный анализ существующих аналогов, таких как «MyStudyLife», «Power Planner», «Notion» и «School Planner», что позволило выделить ключевые функциональные и нефункциональные требования к разрабатываемому приложению. Особое внимание уделено потребностям студентов в организации учебного процесса, контроле дедлайнов и мотивации через систему достижений.

На стадии проектирования разработана архитектура приложения, основанная на слоистой (многоуровневой) MVC-структуре, спроектированы диаграммы классов и вариантов использования, а также созданы концептуальная, логическая и физическая модели базы данных. Пользовательский интерфейс был разработан с учетом принципов удобства и доступности, включая адаптивный дизайн для различных устройств.

Реализация включала создание клиентской и серверной частей приложения, интеграцию с базой данных, а также настройку системы уведомлений и достижений. Контроль качества проведен через функциональное тестирование и тестирование на совместимость, что подтвердило корректность работы приложения в различных браузерах и

операционных системах. Метрики Холстеда и Чепина показали высокое качество кода и его соответствие современным стандартам разработки.

Планирование проекта и оценка бюджета позволили эффективно распределить ресурсы и завершить разработку в поставленные сроки. Затраты на проект составили 130 958,34 рублей, а продолжительность работ – 41 день.

Таким образом, в результате выполнения выпускной квалификационной работы создано программное средство, которое помогает студентам эффективно организовывать учебный процесс, контролировать выполнение задач и закрывать академические задолженности. Приложение соответствует функциональным и нефункциональным требованиям, а его функционал может быть расширен в будущем для поддержки дополнительных возможностей.

Список использованных источников

1. Web App – MyStudyLife [Электронный ресурс]. URL: <https://web.mystudylife.com/> (дата обращения: 19.05.2025).
2. Power Planner [Электронный ресурс]. URL: <https://app.powerplanner.net/calendar/> (дата обращения: 19.05.2025).
3. Notion [Электронный ресурс]. URL: <https://www.notion.com/> (дата обращения: 19.05.2025).
4. Google Play – School Planner [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=daldev.android.gradehelper> (дата обращения: 19.05.2025).
5. Работа с базами данных на языке Python с использованием библиотеки SQLAlchemy 2.0: учеб. пособие / А. И. Елисеев, Д. В. Поляков. – Тамбов: Изд-во ТГТУ, 2024. – 81 с.
6. PWA или классические приложения: новые возможности для банков [Электронный ресурс]. URL: <https://surf.ru/pwa/> (дата обращения: 19.05.2025).
7. Концептуальная модель базы данных // Пензенский Государственный Университет Архитектуры и Строительства. [Электронный ресурс]. URL: <https://studfile.net/preview/9959196/page:2/> (дата обращения 20.05.2025).
8. Моделирование данных: обзор. [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения 20.05.2025).
9. Проектирование пользовательского интерфейса. [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/windows/designing-a-user-interface/> (дата обращения 20.05.2025).
10. Что такое модель MVC: теория. [Электронный ресурс]. URL: <https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami> (дата обращения 20.05.2025).
11. UML: обзор основных типов диаграмм, диаграмма классов.

Часть 1. [Электронный ресурс]. URL: <https://habr.com/ru/articles/738428/> (дата обращения 20.05.2025).

12. PostgreSQL Documentation. [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения 20.05.2025).

13. В чем разница между логической моделью данных и физической моделью данных? [Электронный ресурс]. URL: <https://aws.amazon.com/ru/compare/> (дата обращения 20.05.2025).

14. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2. [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения 20.05.2025).

15. Student-Planner. GitHub-репозиторий. [Электронный ресурс]. URL: <https://github.com/clody8/student-planner/> (дата обращения 15.06.2025).

16. HTTP-клиент для браузера и node.js на основе Promise. [Электронный ресурс]. URL: <https://axios-http.com/ru/docs/intro> (дата обращения 20.05.2025).

17. Основные виды тестирования ПО. [Электронный ресурс]. URL: <https://tquality.ru/blog/vidy-testirovaniya-programmnogo-obespecheniya/> (дата обращения 21.05.2025).

18. Тестирование совместимости: все, что вам нужно знать. [Электронный ресурс]. URL: <https://habr.com/ru/articles/716862/> (дата обращения 21.05.2025).

19. Программный код и его метрики. [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 22.05.2025).

20. Метрики. Саратовский Государственный технический университет им. Ю. А. Гагарина [Электронный ресурс]. URL: <https://studfile.net/preview/1568115/page:2/> (дата обращения 24.05.2025).

Приложение А
(справочное)
Глоссарий

БД (База данных) – это имеющая название совокупность данных, которая отражает состояние объектов и их отношений в рассматриваемой предметной области.

UML (Unified Modeling Language) – язык графического моделирования, применяемый для проектирования архитектуры приложения.

СУБД (Система управления базами данных) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

REST (Representational State Transfer) – это архитектурная концепция, определяющая набор ограничений для построения масштабируемых распределенных систем. Основными принципами REST является ресурсно-ориентированная архитектура, использование стандартных HTTP-методов для манипуляции ресурсами, передача данных без сохранения состояния на сервере, унифицированный интерфейс доступа.

API (Application Programming Interface) – набор готовых классов, функций, процедур и структур, предоставляемых приложением для взаимодействия с другими программами. В контексте работы используется REST API для связи клиентской и серверной частей приложения.

PostgreSQL – это открытая и мощная реляционная система управления данными, основанная на объектно-ориентированной парадигме, обеспечивающая высокую производительность, надежность и масштабируемость.

Фреймворк – программная платформа, определяющая структуру приложения и предоставляющая готовые решения для типовых задач.

Веб-браузер – прикладное программное обеспечение для поиска и просмотра веб-страниц, а также управления веб-приложениями;

Веб-приложение – это клиент-серверное программное решение, доступ к которому осуществляется через сетевой браузер посредством передачи данных по протоколу HTTP, размещается на удаленном сервере

PWA (Progressive Web App) – веб-приложение с возможностью офлайн-работы, пуш-уведомления и установки на устройство.

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

MVC – это архитектурный паттерн, предписывающий разделение программного кода на три взаимосвязанных компонента: модель данных, представление для визуализации информации и контроллер для обработки пользовательских запросов.

Дедлайн – крайний срок выполнения учебной задачи, после которого она считается просроченной.

Приложение Б
(обязательное)
Исходный код приложения

Листинг Б.1 – Файл Endpoints/Achievements.py

```

from typing import List, Any
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ....crud import achievement as crud_achievement
from ....db.session import get_db
from ....schemas.user import User
from ....schemas.achievement import Achievement, UserAchievement,
UserStats
from .auth import get_current_user

router = APIRouter()

@router.get("/", response_model=List[Achievement])
def read_achievements(
    db: Session = Depends(get_db)
) -> Any:
    """
    Получить все доступные достижения
    """
    return crud_achievement.get_all_achievements(db)

@router.get("/user", response_model=List[UserAchievement])
def read_user_achievements(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить достижения текущего пользователя
    """
    return crud_achievement.get_user_achievements(db,
current_user.id)

@router.get("/stats", response_model=UserStats)
def read_user_stats(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить статистику пользователя
    """
    stats = crud_achievement.get_user_stats(db, current_user.id)
    return UserStats(**stats)

@router.post("/check")
def check_achievements(
    db: Session = Depends(get_db),

```

```

        current_user: User = Depends(get_current_user)
    ) -> Any:
        """
        Проверить и присвоить новые достижения
        """
        new_achievements =
crud_achievement.check_and_award_achievements(db, current_user.id)
        return {
            "message": f"Проверка завершена. Получено новых
достижений: {len(new_achievements)}",
            "new_achievements": new_achievements
        }

```

Листинг Б.2 – Файл Endpoints/Auth.py

```

from datetime import timedelta
from typing import Any
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer,
OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from ....core import security
from ....core.config import settings
from ....crud import user as crud_user
from ....db.session import get_db
from ....schemas.user import (
    User, UserCreate, Token, UserLogin, PasswordChange,
    PushSubscription,
    PushNotification
)
from ....services.notifications import NotificationService

router = APIRouter()
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="api/v1/auth/login")

def get_current_user(db: Session = Depends(get_db), token: str =
Depends(oauth2_scheme)) -> User:
    payload = security.verify_token(token)
    if payload is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Could not validate credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )

    email: str = payload.get("sub")
    if email is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Could not validate credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )

```

```

user = crud_user.get_user_by_email(db, email=email)
if user is None:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="User not found",
        headers={"WWW-Authenticate": "Bearer"},
    )
return user

@router.post("/register", response_model=User)
def register(user: UserCreate, db: Session = Depends(get_db)) ->
Any:
    """
    Регистрация нового пользователя
    """
    db_user = crud_user.get_user_by_email(db, email=user.email)
    if db_user:
        raise HTTPException(
            status_code=400,
            detail="Пользователь с таким email уже существует"
        )
    db_user = crud_user.create_user(db=db, user=user)
    return db_user

@router.post("/login", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends(), db:
Session = Depends(get_db)) -> Any:
    """
    Авторизация пользователя
    """
    user = crud_user.authenticate_user(db,
email=form_data.username, password=form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверный email или пароль",
            headers={"WWW-Authenticate": "Bearer"},
        )
    elif not user.is_active:
        raise HTTPException(status_code=400, detail="Аккаунт не
активен")

    access_token_expires =
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = security.create_access_token(
        data={"sub": user.email},
        expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

```

```

@router.post("/login-json", response_model=Token)
def login_json(user_login: UserLogin, db: Session =
Depends(get_db)) -> Any:
    """
    Авторизация пользователя через JSON
    """
    user = crud_user.authenticate_user(db, email=user_login.email,
password=user_login.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Неверный email или пароль",
        )
    elif not user.is_active:
        raise HTTPException(status_code=400, detail="Аккаунт не
активен")

    access_token_expires =
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = security.create_access_token(
        data={"sub": user.email},
        expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

@router.get("/me", response_model=User)
def read_users_me(current_user: User = Depends(get_current_user))
-> Any:
    """
    Получить данные текущего пользователя
    """
    return current_user

@router.post("/change-password")
def change_password(
    password_data: PasswordChange,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Изменить пароль пользователя
    """
    success = crud_user.change_password(
        db, current_user.id, password_data.current_password,
password_data.new_password
    )
    if not success:
        raise HTTPException(status_code=400, detail="Неверный
текущий пароль")
    return {"message": "Пароль успешно изменен"}

```

```

@router.post("/push-subscription")
def save_push_subscription(
    subscription: PushSubscription,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Сохранить push-подписку для уведомлений
    """

    success = crud_user.save_push_subscription(
        db, current_user.id, subscription.endpoint,
        subscription.keys['p256dh'], subscription.keys['auth']
    )
    if not success:
        raise HTTPException(status_code=400, detail="Ошибка
        сохранения подписки")

    # Отправляем тестовое уведомление
    test_notification = PushNotification(
        title="🔔 Уведомления включены!",
        body="Теперь вы будете получать напоминания о дедлайнах",
        url="/",
        tag="push-enabled"
    )
    NotificationService.send_notification_to_user(db,
        current_user.id, test_notification)

    return {"message": "Push-подписка сохранена"}

@router.post("/test-notification")
def send_test_notification(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Отправить тестовое уведомление (для разработки)
    """

    if not settings.DEBUG:
        raise HTTPException(status_code=404, detail="Не найдено")

    test_notification = PushNotification(
        title="🔧 Тестовое уведомление",
        body="Это тестовое push-уведомление для проверки работы
системы",
        url="/",
        tag="test"
    )

    success = NotificationService.send_notification_to_user(
        db, current_user.id, test_notification

```

```

    )

    if success:
        return {"message": "Тестовое уведомление отправлено"}
    else:
        raise HTTPException(
            status_code=400,
            detail="Не удалось отправить уведомление. Проверьте
подписку."
        )

```

Листинг Б.3 – Файл Endpoints/Tasks.py

```

from typing import List, Optional, Any
from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.orm import Session
from ....crud import task as crud_task
from ....db.session import get_db
from ....schemas.user import User
from ....schemas.task import Task, TaskCreate, TaskUpdate,
TaskFilter, TaskStats, TaskStep
from .auth import get_current_user

router = APIRouter()

@router.get("/", response_model=List[Task])
def read_tasks(
    skip: int = 0,
    limit: int = 100,
    task_type: Optional[str] = Query(None),
    priority: Optional[str] = Query(None),
    status: Optional[str] = Query(None),
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить список задач пользователя с фильтрацией
    """
    filters = TaskFilter()
    if task_type:
        filters.task_type = task_type
    if priority:
        filters.priority = priority
    if status:
        filters.status = status

    tasks = crud_task.get_tasks(
        db=db, user_id=current_user.id, skip=skip, limit=limit,
filters=filters
    )
    return tasks

```

```

@router.post("/", response_model=Task)
def create_task(
    task: TaskCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Создать новую задачу
    """
    return crud_task.create_task(db=db, task=task,
user_id=current_user.id)

@router.get("/{task_id}", response_model=Task)
def read_task(
    task_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить задачу по ID
    """
    task = crud_task.get_task(db=db, task_id=task_id,
user_id=current_user.id)
    if not task:
        raise HTTPException(status_code=404, detail="Задача не
найдена")
    return task

@router.put("/{task_id}", response_model=Task)
def update_task(
    task_id: int,
    task_update: TaskUpdate,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Обновить задачу
    """
    task = crud_task.update_task(
        db=db, task_id=task_id, user_id=current_user.id,
task_update=task_update
    )
    if not task:
        raise HTTPException(status_code=404, detail="Задача не
найдена")
    return task

@router.delete("/{task_id}")

```

```

def delete_task(
    task_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Удалить задачу
    """
    success = crud_task.delete_task(db=db, task_id=task_id,
user_id=current_user.id)
    if not success:
        raise HTTPException(status_code=404, detail="Задача не
найденa")
    return {"message": "Задача удалена"}

@router.get("/upcoming/list", response_model=List[Task])
def read_upcoming_tasks(
    days: int = Query(7, description="Количество дней вперед"),
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить ближайшие задачи
    """
    return crud_task.get_upcoming_tasks(db=db,
user_id=current_user.id, days=days)

@router.get("/overdue/list", response_model=List[Task])
def read_overdue_tasks(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить просроченные задачи
    """
    return crud_task.get_overdue_tasks(db=db,
user_id=current_user.id)

@router.get("/stats/summary", response_model=TaskStats)
def read_task_stats(
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Получить статистику по задачам
    """
    stats = crud_task.get_task_stats(db=db,
user_id=current_user.id)
    return TaskStats(**stats)

```

```

@router.put("/steps/{step_id}/complete")
def complete_task_step(
    step_id: int,
    is_completed: bool,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
) -> Any:
    """
    Отметить этап задачи как выполненный/невыполненный
    """
    step = crud_task.update_task_step(db=db, step_id=step_id,
is_completed=is_completed)
    if not step:
        raise HTTPException(status_code=404, detail="Этап задачи
не найден")
    return {"message": "Статус этапа обновлен"}

```

Листинг Б.4 – Файл Crud/Achievement.py

```

from typing import List, Optional, Dict, Any
from sqlalchemy.orm import Session
from sqlalchemy import and_, desc, func, distinct
from ..db.models.goal import Achievement, UserAchievement, Goal
from ..db.models.task import Task, TaskStatus
from ..db.models.user import User
from datetime import datetime, timedelta

def get_all_achievements(db: Session) -> List[Achievement]:
    """Получить все доступные достижения"""
    return
db.query(Achievement).order_by(Achievement.points).all()

def get_user_achievements(db: Session, user_id: int) ->
List[UserAchievement]:
    """Получить достижения пользователя"""
    return db.query(UserAchievement).filter(
        UserAchievement.user_id == user_id
    ).order_by(desc(UserAchievement.earned_at)).all()

def award_achievement(db: Session, user_id: int, achievement_id:
int) -> UserAchievement:
    """Присвоить достижение пользователю"""
    # Проверяем, есть ли уже это достижение у пользователя
    existing = db.query(UserAchievement).filter(
        and_(
            UserAchievement.user_id == user_id,
            UserAchievement.achievement_id == achievement_id
        )
    ).first()

```

```

if existing:
    return existing

user_achievement = UserAchievement(
    user_id=user_id,
    achievement_id=achievement_id,
    earned_at=datetime.now()
)
db.add(user_achievement)
db.commit()
db.refresh(user_achievement)
return user_achievement

def get_user_stats(db: Session, user_id: int) -> Dict[str, Any]:
    """Получить статистику пользователя для достижений"""
    # Базовая статистика задач
    total_tasks = db.query(Task).filter(Task.user_id ==
user_id).count()
    completed_tasks = db.query(Task).filter(
        and_(Task.user_id == user_id, Task.status ==
TaskStatus.completed)
    ).count()
    pending_tasks = db.query(Task).filter(
        and_(Task.user_id == user_id, Task.status ==
TaskStatus.pending)
    ).count()
    overdue_tasks = db.query(Task).filter(
        and_(Task.user_id == user_id, Task.status ==
TaskStatus.overdue)
    ).count()

    # Статистика целей
    completed_goals = db.query(Goal).filter(
        and_(Goal.user_id == user_id, Goal.is_completed == True)
    ).count()

    # Подсчет очков
    user_achievements = db.query(UserAchievement).filter(
        UserAchievement.user_id == user_id
    ).all()

    total_points = 0
    for ua in user_achievements:
        achievement = db.query(Achievement).filter(Achievement.id
== ua.achievement_id).first()
        if achievement:
            total_points += achievement.points

    # Расчет completion_rate
    completion_rate = (completed_tasks / total_tasks * 100) if
total_tasks > 0 else 0

```

```

# Подсчет streak (упрощенная версия - количество выполненных
задач за последние 7 дней)
week_ago = datetime.now() - timedelta(days=7)
recent_completed = db.query(Task).filter(
    and_(
        Task.user_id == user_id,
        Task.status == TaskStatus.completed,
        Task.completed_at >= week_ago
    )
).count()

return {
    "total_tasks": total_tasks,
    "completed_tasks": completed_tasks,
    "pending_tasks": pending_tasks,
    "overdue_tasks": overdue_tasks,
    "completion_rate": round(completion_rate, 1),
    "current_streak": recent_completed, # Упрощенная версия
streak
    "total_points": total_points,
    "achievements_count": len(user_achievements),
    "completed_goals": completed_goals
}

def check_and_award_achievements(db: Session, user_id: int) ->
List[UserAchievement]:
    """Проверить и присвоить новые достижения пользователю"""
    new_achievements = []
    stats = get_user_stats(db, user_id)

    # Получаем все достижения
    all_achievements = get_all_achievements(db)

    # Получаем уже полученные достижения
    earned_achievement_ids = {ua.achievement_id for ua in
get_user_achievements(db, user_id)}

    for achievement in all_achievements:
        if achievement.id in earned_achievement_ids:
            continue

        should_award = False

        # Проверяем условия для каждого типа достижения
        if achievement.condition_type == "tasks_completed" and
stats["completed_tasks"] >= achievement.condition_value:
            should_award = True
        elif achievement.condition_type == "tasks_created" and
stats["total_tasks"] >= achievement.condition_value:
            should_award = True

```

```

        elif achievement.condition_type == "streak_days" and
stats["current_streak"] >= achievement.condition_value:
            should_award = True
        elif achievement.condition_type == "goals_completed" and
stats["completed_goals"] >= achievement.condition_value:
            should_award = True

        if should_award:
            new_achievement = award_achievement(db, user_id,
achievement.id)
            new_achievements.append(new_achievement)

    return new_achievements

```

Листинг Б.5 – Файл User.py

```

User.py
import logging
from typing import Optional
from sqlalchemy.orm import Session
from ..core.security import get_password_hash, verify_password
from ..db.models.user import User
from ..db.models.push_subscription import PushSubscription
from ..schemas.user import UserCreate, UserUpdate

logger = logging.getLogger(__name__)

def get_user(db: Session, user_id: int) -> Optional[User]:
    return db.query(User).filter(User.id == user_id).first()

def get_user_by_email(db: Session, email: str) -> Optional[User]:
    return db.query(User).filter(User.email == email).first()

# OAuth methods removed as per PRD requirements

def create_user(db: Session, user: UserCreate) -> User:
    hashed_password = get_password_hash(user.password)
    db_user = User(
        email=user.email,
        hashed_password=hashed_password,
        full_name=user.full_name,
        is_active=user.is_active
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

```

```
# OAuth user creation methods removed as per PRD requirements
```

```
def update_user(db: Session, user_id: int, user_update:
UserUpdate) -> Optional[User]:
    db_user = get_user(db, user_id)
    if not db_user:
        return None

    update_data = user_update.dict(exclude_unset=True)
    for field, value in update_data.items():
        setattr(db_user, field, value)

    db.commit()
    db.refresh(db_user)
    return db_user

def authenticate_user(db: Session, email: str, password: str) ->
Optional[User]:
    user = get_user_by_email(db, email)
    if not user:
        return None
    if not verify_password(password, user.hashed_password):
        return None
    return user

def change_password(db: Session, user_id: int, current_password:
str, new_password: str) -> bool:
    user = get_user(db, user_id)
    if not user:
        return False

    if not verify_password(current_password,
user.hashed_password):
        return False

    user.hashed_password = get_password_hash(new_password)
    db.commit()
    return True

def save_push_subscription(db: Session, user_id: int, endpoint:
str, p256dh_key: str, auth_key: str) -> bool:
    """Сохранить push-подписку пользователя"""
    try:
        # Удаляем старую подписку если есть
        old_subscription =
db.query(PushSubscription).filter(PushSubscription.user_id ==
user_id).first()
        if old_subscription:
            db.delete(old_subscription)
```

```

        db.commit() # Важно: коммитим удаление перед
созданием новой

        # Создаем новую подписку
        new_subscription = PushSubscription(
            user_id=user_id,
            endpoint=endpoint,
            p256dh_key=p256dh_key,
            auth_key=auth_key
        )
        db.add(new_subscription)
        db.commit()
        return True
    except Exception as e:
        db.rollback()
        logger.error(f"Ошибка сохранения push-подписки для
пользователя {user_id}: {e}")
        return False

def get_push_subscription(db: Session, user_id: int) ->
Optional[PushSubscription]:
    """Получить push-подписку пользователя"""
    return
db.query(PushSubscription).filter(PushSubscription.user_id ==
user_id).first()

```

Листинг Б.6 – Файл Model/Notification.py

```

from sqlalchemy import Column, Integer, String, Text, ForeignKey,
DateTime, Boolean
from sqlalchemy.sql import func
from sqlalchemy.orm import relationship
from ..base import Base

class Notification(Base):
    __tablename__ = "notifications"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"),
nullable=False)
    task_id = Column(Integer, ForeignKey("tasks.id"),
nullable=True) # Может быть системное уведомление

    title = Column(String, nullable=False)
    message = Column(Text, nullable=False)
    type = Column(String, nullable=False) # "deadline",
"overdue", "achievement", "system"

    sent_at = Column(DateTime(timezone=True),
server_default=func.now())

```

```

was_opened = Column(Boolean, default=False)

created_at = Column(DateTime(timezone=True),
server_default=func.now())

# Relationships
user = relationship("User", back_populates="notifications")
task = relationship("Task", back_populates="notifications")

```

Листинг Б.7 – Файл `Services/Task_status.py`

```

from typing import List
from sqlalchemy.orm import Session
from datetime import datetime, timezone
from ..db.models.task import Task, TaskStatus
from ..db.session import SessionLocal

class TaskStatusService:
    """Сервис для управления статусами задач"""

    @staticmethod
    def update_overdue_tasks() -> int:
        """
        Обновляет статусы просроченных задач.
        Возвращает количество обновленных задач.
        """
        db = SessionLocal()
        try:
            now = datetime.now(timezone.utc)

            # Находим задачи, которые просрочены, но еще не
помечены как просроченные
            overdue_tasks = db.query(Task).filter(
                Task.deadline < now,
                Task.status.in_([TaskStatus.pending,
TaskStatus.in_progress]),
                Task.is_overdue == False
            ).all()

            updated_count = 0
            for task in overdue_tasks:
                task.is_overdue = True
                task.status = TaskStatus.overdue
                updated_count += 1

            if updated_count > 0:
                db.commit()

            return updated_count

        except Exception as e:
            db.rollback()

```

```

        raise e
    finally:
        db.close()

    @staticmethod
    def get_overdue_tasks_by_user(db: Session, user_id: int) ->
List[Task]:
    """Получить просроченные задачи пользователя"""
    return db.query(Task).filter(
        Task.user_id == user_id,
        Task.is_overdue == True,
        Task.status == TaskStatus.overdue
    ).all()

    @staticmethod
    def mark_task_as_completed(db: Session, task_id: int, user_id:
int) -> bool:
    """Отметить задачу как выполненную"""
    task = db.query(Task).filter(
        Task.id == task_id,
        Task.user_id == user_id
    ).first()

    if not task:
        return False

    task.status = TaskStatus.completed
    task.completed_at = datetime.now(timezone.utc)
    task.is_overdue = False # Сбрасываем флаг просрочки

    db.commit()
    return True

```

Листинг Б.9 – Файл app/achievements/page.tsx

```

'use client';

import { useState, useEffect } from 'react';
import { useRouter } from 'next/navigation';
import Link from 'next/link';
import { achievementsAPI, Achievement, UserAchievement, UserStats
} from '@lib/api';
import {
    TrophyIcon as TrophyIconSolid,
    CheckIcon,
    LockClosedIcon,
    InformationCircleIcon,
    HomeIcon
} from '@heroicons/react/24/solid';

export default function AchievementsPage() {
    const router = useRouter();

```

```

    const [allAchievements, setAllAchievements] =
useState<Achievement[]>([]);
    const [userAchievements, setUserAchievements] =
useState<UserAchievement[]>([]);
    const [stats, setStats] = useState<UserStats | null>(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState<string | null>(null);

    useEffect(() => {
        loadData();
    }, []);

    const loadData = async () => {
        try {
            setLoading(true);
            const [achievements, userAch, userStats] = await
Promise.all([
                achievementsAPI.getAllAchievements(),
                achievementsAPI.getUserAchievements(),
                achievementsAPI.getUserStats(),
            ]);
            setAllAchievements(achievements);
            setUserAchievements(userAch);
            setStats(userStats);
        } catch (err: any) {
            setError('Ошибка загрузки достижений');
        } finally {
            setLoading(false);
        }
    };

    const getProgressValue = (achievement: Achievement): number => {
        if (!stats) return 0;

        switch (achievement.condition_type) {
            case 'tasks_completed':
                return stats.completed_tasks;
            case 'tasks_created':
                return stats.total_tasks;
            case 'streak_days':
                return stats.current_streak;
            case 'goals_completed':
                return stats.completed_goals;
            default:
                return 0;
        }
    };

    const getConditionDescription = (achievement: Achievement):
string => {
        switch (achievement.condition_type) {
            case 'tasks_completed':
                return `Выполните ${achievement.condition_value} задач`;

```

```

    case 'tasks_created':
      return `Создайте ${achievement.condition_value} задач`;
    case 'streak_days':
      return `Выполняйте задачи ${achievement.condition_value}
дней подряд`;
    case 'goals_completed':
      return `Достигните ${achievement.condition_value} целей`;
    default:
      return achievement.description;
  }
};

const isEarned = (achievementId: number): boolean => {
  return userAchievements.some(ua => ua.achievement.id ===
achievementId);
};

const getProgressPercentage = (achievement: Achievement): number
=> {
  const current = getProgressValue(achievement);
  const target = achievement.condition_value;
  return Math.min(100, Math.round((current / target) * 100));
};

if (loading) {
  return (
    <div className="min-h-screen bg-gradient-to-br from-blue-50
to-indigo-100">
      {/* Header */}
      <header className="bg-white/80 backdrop-blur-sm border-b
border-indigo-100 sticky top-0 z-10">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <div className="flex items-center h-16">
            <Link href="/" className="mr-4 text-gray-600
hover:text-indigo-600 transition-colors">
              <HomeIcon className="h-6 w-6" />
            </Link>
            <TrophyIconSolid className="h-6 w-6 text-yellow-500
mr-3" />
            <h1 className="text-xl font-bold bg-gradient-to-r
from-indigo-600 to-purple-600 bg-clip-text text-transparent">
              Достижения
            </h1>
          </div>
        </div>
      </header>
      <div className="flex items-center justify-center min-h-
[calc(100vh-80px)]">

```

Приложение В
(рекомендуемое)
Функциональное тестирование

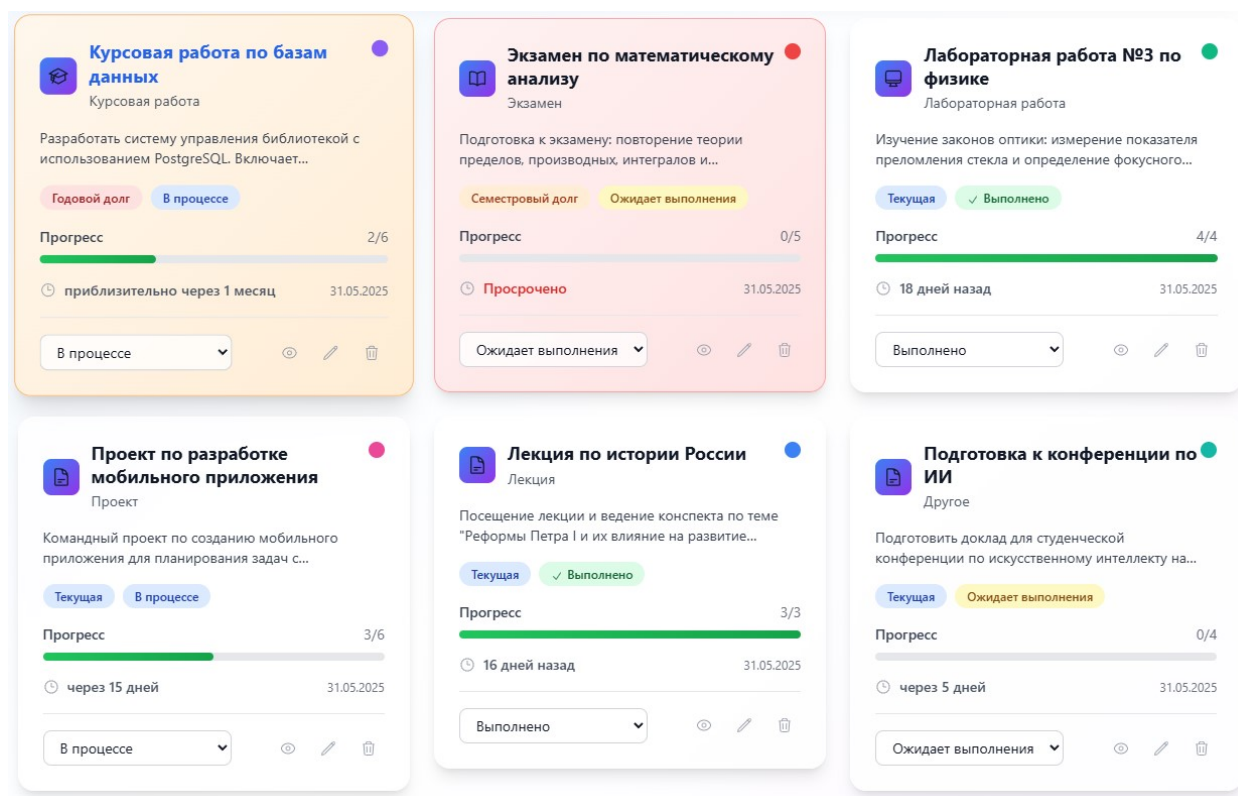


Рисунок В.1 – Проверка отображения задач

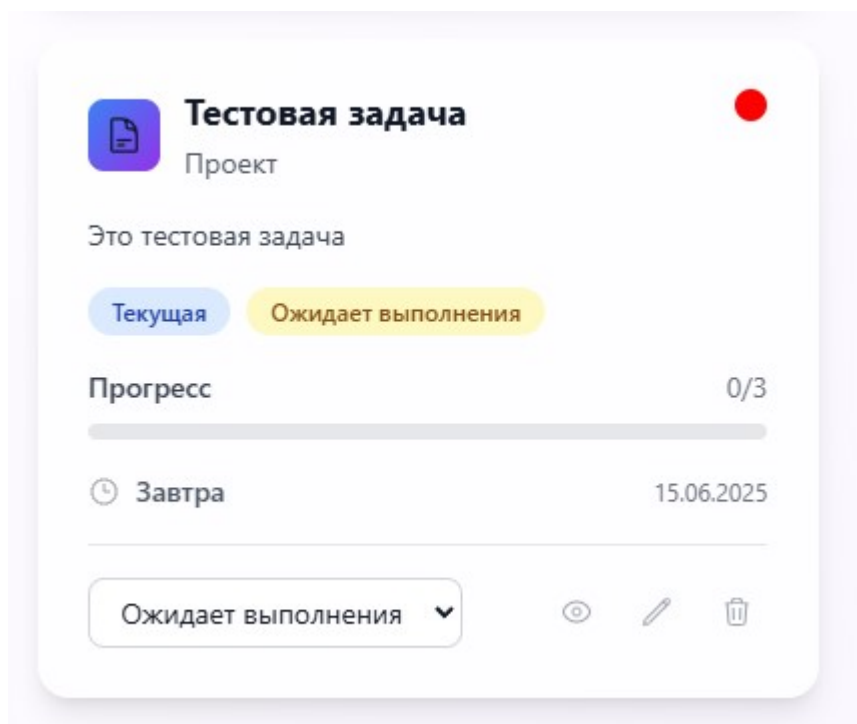


Рисунок В.2 – Создание новой задачи

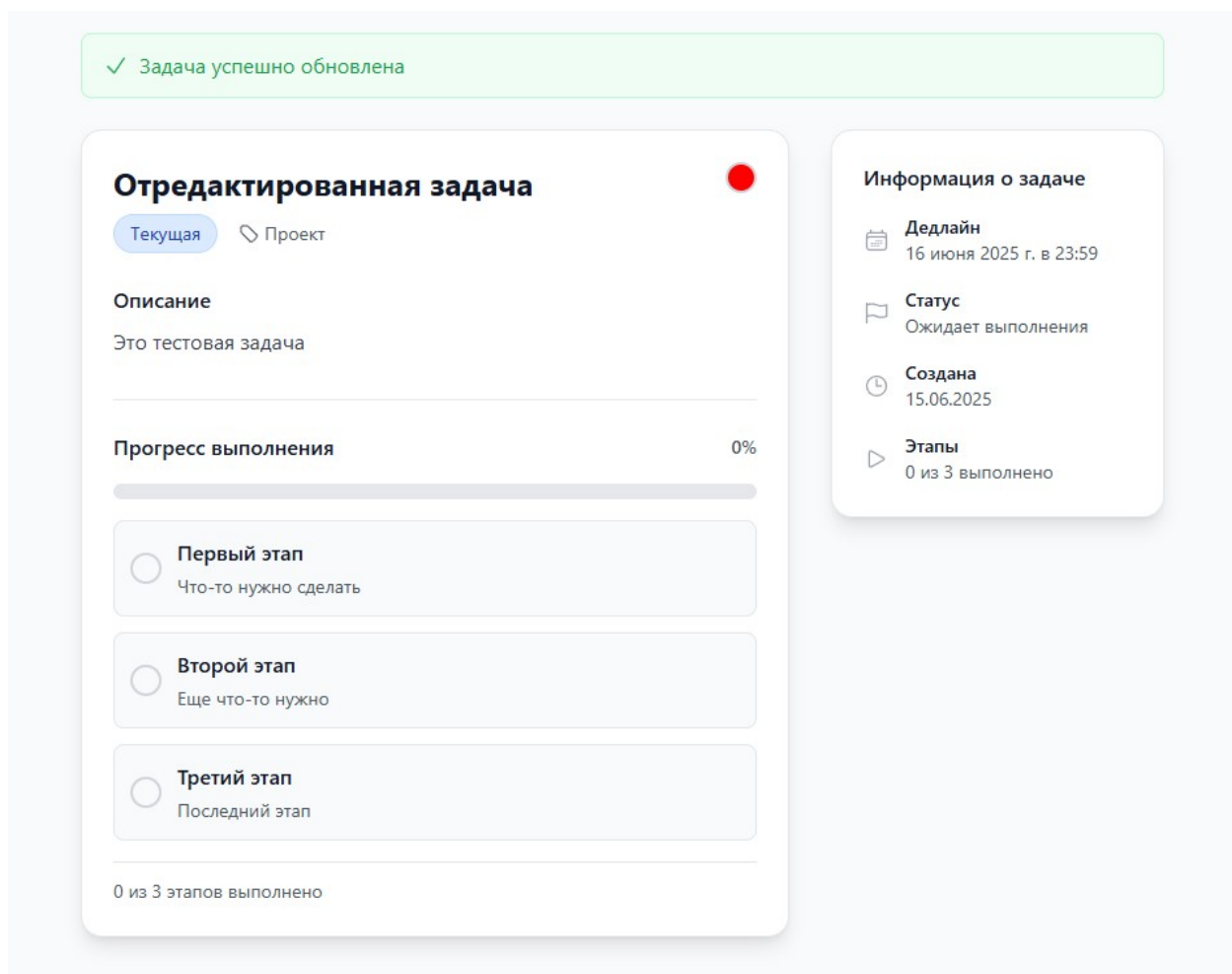


Рисунок В.3 – Редактирование задачи

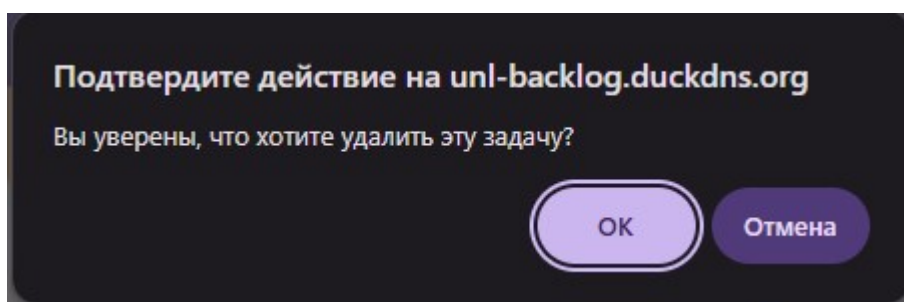


Рисунок В.4 – Удаление задачи

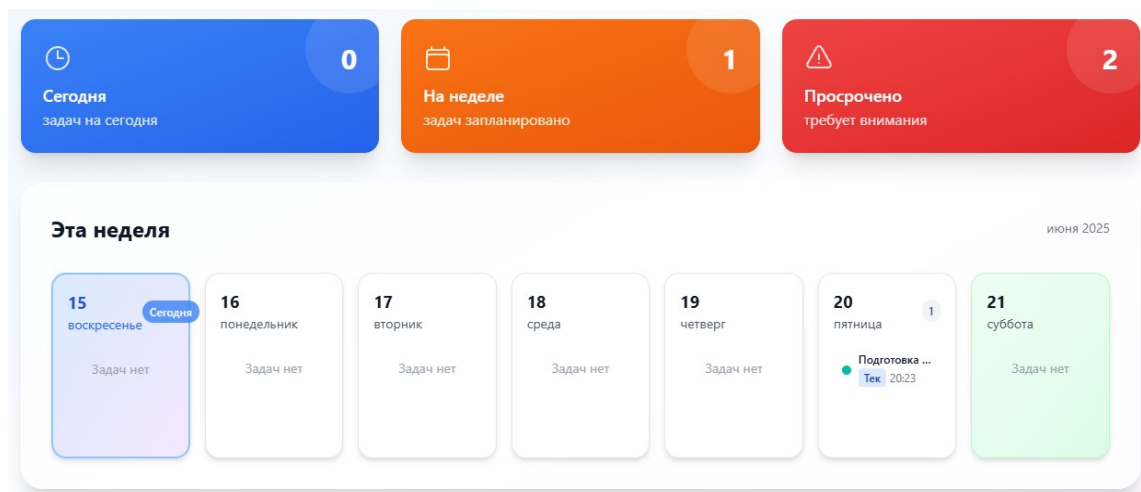


Рисунок В.5 – Отображение календаря

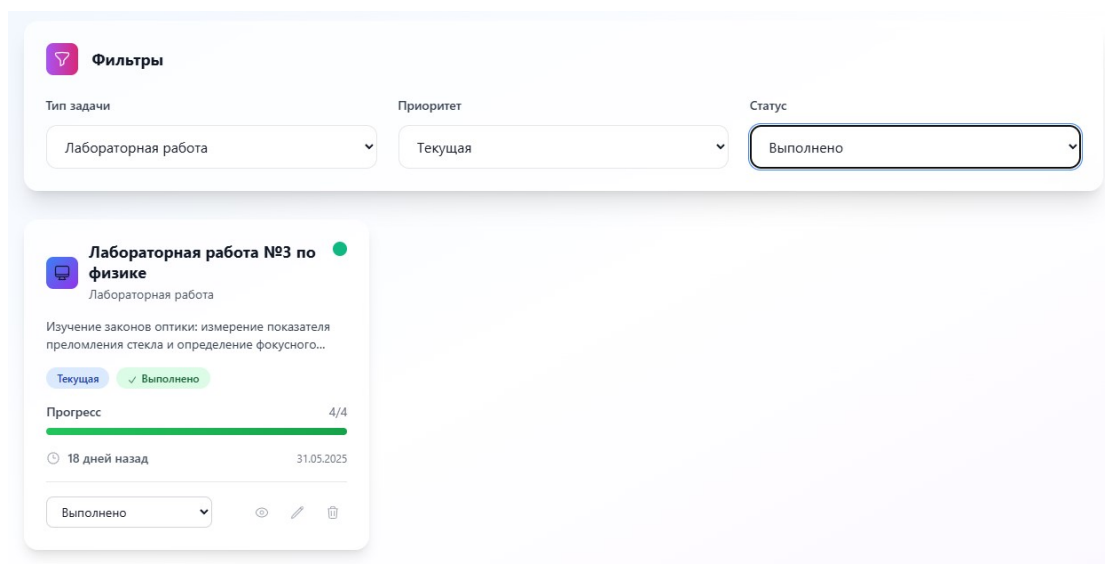


Рисунок В.6 – Применение фильтрации

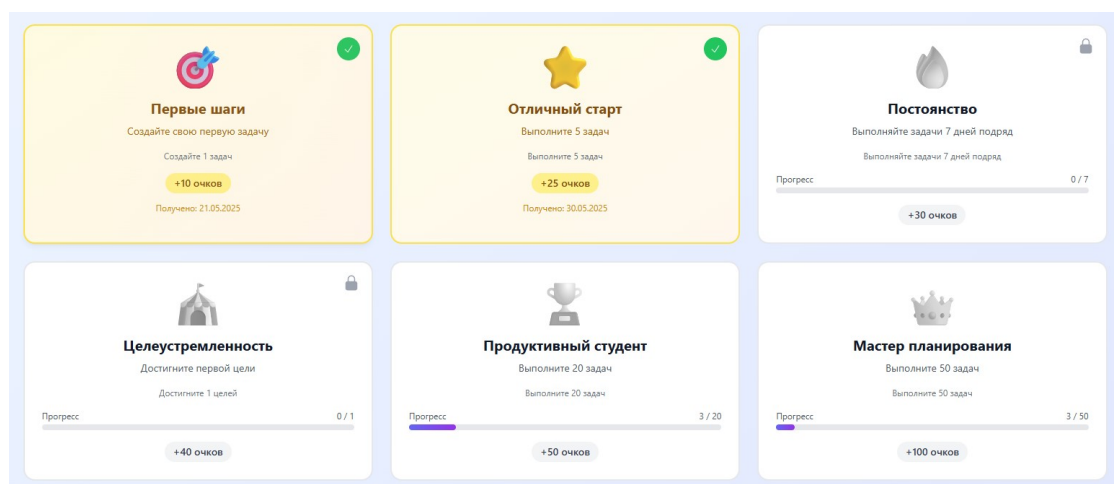
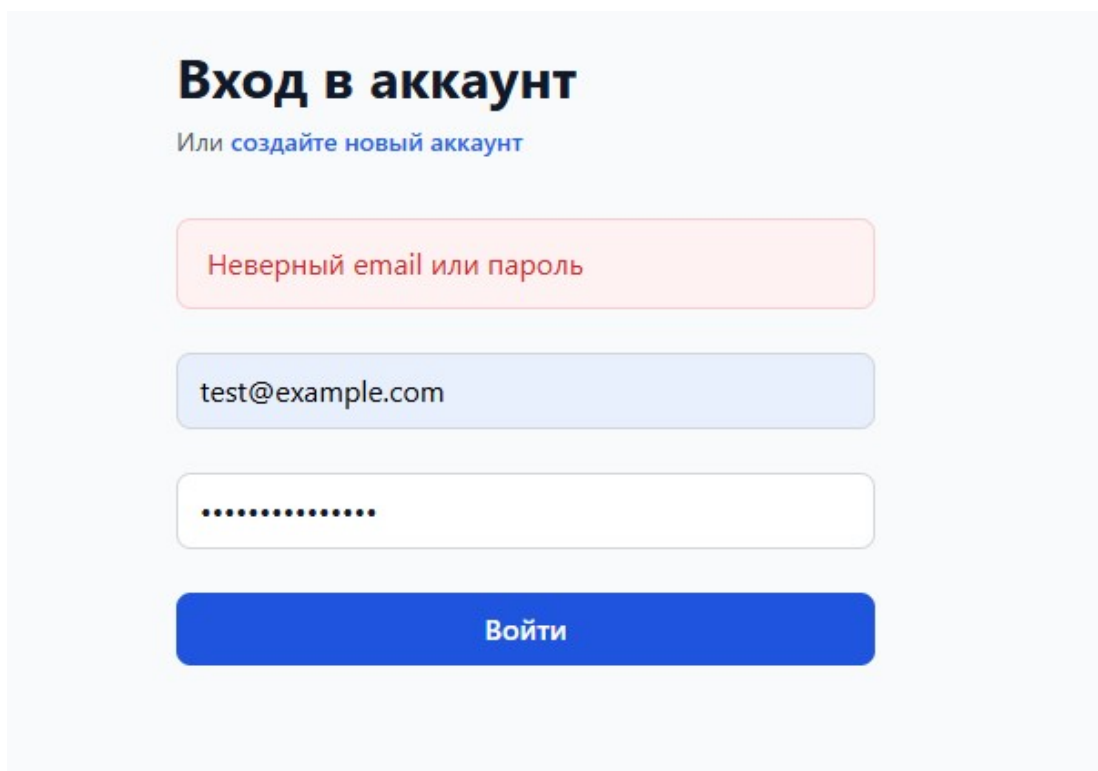


Рисунок В.7 – Отображение страницы достижений



Вход в аккаунт
Или [создайте новый аккаунт](#)

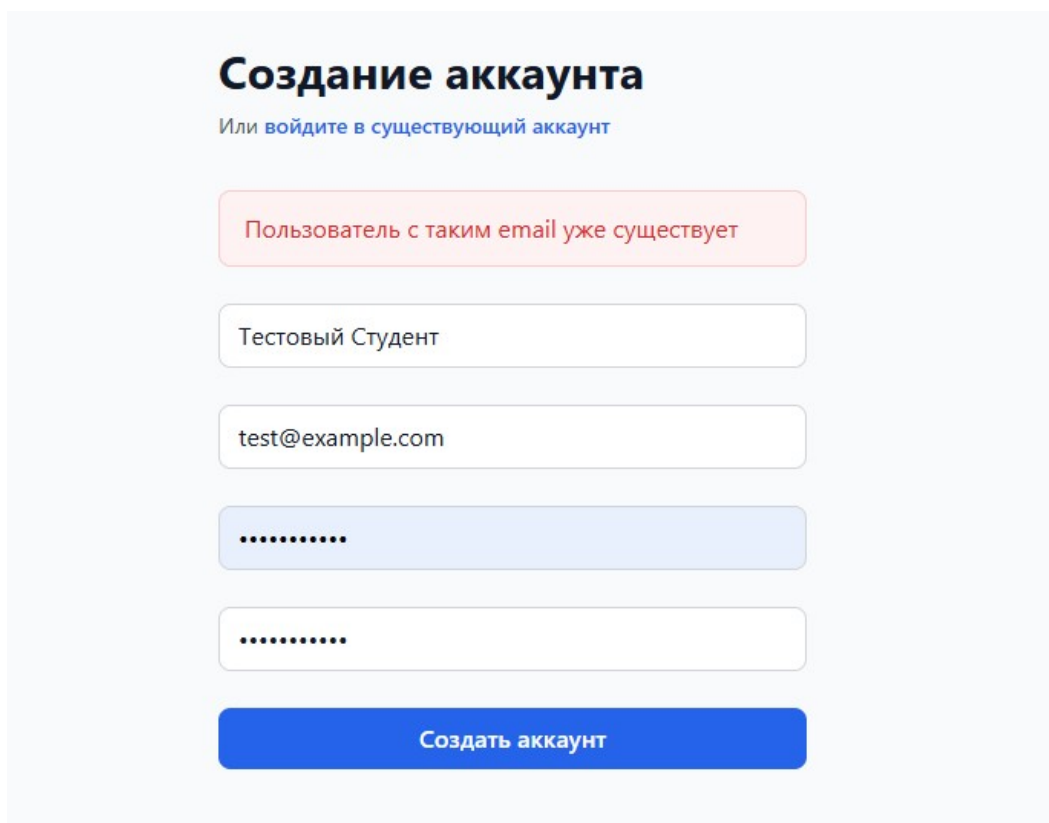
Неверный email или пароль

test@example.com

.....

Войти

Рисунок В.8 – Проверка авторизации с неверным паролем



Создание аккаунта
Или [войдите в существующий аккаунт](#)

Пользователь с таким email уже существует

Тестовый Студент

test@example.com

.....

.....

Создать аккаунт

Рисунок В.9 – Проверка регистрации с существующей почтой

Создание аккаунта

Или [войдите](#) в существующий аккаунт

Имя обязательно

Email обязателен

Пароль обязателен

Подтверждение пароля обязательно

[Создать аккаунт](#)

Рисунок В.10 – Проверка регистрации с недостающими данными



Push-уведомления
Включены

[Отключить](#)

- Напоминания о дедлайнах
- Уведомления о просроченных задачах
- Ежедневные сводки

Рисунок В.11 – Включение уведомлений