МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ШКОЛА КОМПЬЮТЕРНЫХ НАУК

Кафедра программного обеспечения

**ОТЧЕТ ПО ДИСЦИПЛИНЕ**

**«ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ ПРИНЯТИЯ РЕШЕНИЙ»**

Разработка интерактивной образовательной платформы для централизованного хранения, управления и использования методических материалов

|  |  |  |
| --- | --- | --- |
| Выполнил (а) работу  студент (ка) \_\_ курса  *очной* формы обучения | *\_\_\_\_\_\_\_\_\_\_*  *Подпись* | Серов Никита Сергеевич |
| Выполнил (а) работу  студент (ка) \_\_ курса  *очной* формы обучения | *\_\_\_\_\_\_\_\_\_\_*  *Подпись* | Кириченко Денис Дмитриевич |
| Научный руководитель | \_\_\_\_\_\_\_\_\_  *Подпись* | Гаврилова Наталия Михайловна |

Тюмень

2024

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc185545080)

[ГЛАВА 1. ОПИСАНИЕ ПРОЕКТА 4](#_Toc185545081)

[1.1. Идея проекта 4](#_Toc185545082)

[1.2. Требования к проекту 4](#_Toc185545083)

[1.3. Постановка задачи 5](#_Toc185545084)

[1.4. Используемые технологии 6](#_Toc185545085)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА 8](#_Toc185545086)

[2.1. Теоретические разработки 8](#_Toc185545087)

[2.2. Практические решения 13](#_Toc185545088)

[2.3. Процесс разработки 13](#_Toc185545089)

[ГЛАВА 3. ФУНКЦИОНАЛ И ТЕСТИРОВАНИЕ ПРОЕКТА 23](#_Toc185545090)

[3.1. Пользовательский интерфейс 23](#_Toc185545091)

[3.2. Функциональные возможности 24](#_Toc185545092)

[3.3. Тестирование проекта 24](#_Toc185545093)

[ЗАКЛЮЧЕНИЕ 27](#_Toc185545094)

[СПИСОК ЛИТЕРАТУРЫ 29](#_Toc185545095)

[ПРИЛОЖЕНИЯ 1-7 31](#_Toc185545096)

# ВВЕДЕНИЕ

Образовательный процесс сталкивается с отсутствием единого подхода к хранению и доступу к методическим материалам. Это создает сложности для преподавателей и студентов в их использовании. Учебные материалы часто распределены по разным источникам: личные архивы преподавателей на локальных компьютерах, физические носители, разрозненные цифровые хранилища. Такое распределение затрудняет поиск, актуализацию и систематизацию информации. Например, преподаватели тратят много времени на обновление нужных материалов, а студенты сталкиваются с трудностью поиска этой информации в случае, если он размещен разрозненных цифровых хранилищах, в случае же хранения информации в личных хранилищах преподавателей, доступ к информации еще сильнее затрудняется.

Проблема хранения и актуализации материалов становится особенно важной в условиях цифровизации образования. Учебные программы требуют оперативного обновления и адаптации. Так, для дисциплины "Математическое моделирование природных систем" методические материалы представлены в виде книги из двух частей[16]. Однако ее невозможно обновить в текущем формате, что ограничивает возможности преподавателей и студентов. Размещение материалов в едином цифровом хранилище позволит решить проблему: обновление будет происходить оперативно, доступ к актуальной информации станет более быстрым для всех участников образовательного процесса, а структура материалов станет более организованной.

Целью проекта является разработка интерактивной образовательной системы, обеспечивающей централизованное хранение, управление и использование методических материалов.

# ГЛАВА 1. ОПИСАНИЕ ПРОЕКТА

# 1.1. Идея проекта

Создать интерактивную образовательную платформу, которая объединит все методические материалы в единой системе и предоставит доступ к ним через веб-интерфейс. Система должна включать модули для управления материалами, их обновления и использования, а также добавление новых модулей для различных дисциплин по желанию преподавателей, на основе статьи ИСА РАН по разработке образовательной Web-платформы[1].

# 1.2. Требования к проекту

Проект должен учитывать множество требований и ограничений. Они могут быть как функциональными, так и нефункциональными. Система должна предоставлять студентам доступ к методическим материалам, организованным по темам курса, с возможностью навигации по ним. Одной из ключевых функций в рамках проекта станет проведение математических экспериментов для дисциплины “Математическое моделирование природных систем”, где студенты смогут вводить исходные данные и получать результаты в виде графиков и диаграмм. Эти эксперименты помогут лучше понять теоретический материал за счет визуализации и анализа данных.

Преподаватели, в свою очередь, должны иметь возможность создавать и редактировать методические материалы, включая тексты, формулы и графики, а также управлять настройками математических экспериментов.

Система должна быть производительной, обеспечивая доступ одновременно многих пользователей, а также предоставляющая доступ к проведению расчётов, которые нагружают систему. Интерфейс должен быть адаптирован для ПК, а данные пользователей — защищены от несанкционированного доступа. Разработку платформы следует вести с учетом возможности масштабирования для добавления новых функций, материалов и экспериментов.

В разработке необходимо учитывать и санкционные ограничения, которые ограничивают использование зарубежных облачных сервисов, таких как AWS и Google Cloud, а также некоторых библиотек с лицензиями, запрещающими использование в России. Это требует использования отечественных или открытых решений, которые не зависят от геополитических факторов. Также усложнено взаимодействие с зарубежными API и интеграция с международными системами, что накладывает дополнительные ограничения на выбор технологий.

Технические ограничения включают в себя необходимость оптимизации расчётов для снижения нагрузки на сервер и минимизации объемов передаваемых данных. Платформа должна быть готова к работе в условиях нестабильного интернет-соединения у пользователей. Также важно учитывать, что доступ к технической поддержке зарубежных решений может быть затруднен или вовсе невозможен.

# 1.3. Постановка задачи

Разработать интерактивную образовательную платформу, которая будет:

1. Объединять методические материалы различных дисциплин в единой системе.
2. Предоставлять доступ к материалам студентам через веб-интерфейс, организованный по темам курсов.
3. Содержать модуль проведения математических экспериментов для дисциплины "Математическое моделирование природных систем" с возможностью ввода параметров, выполнения расчётов и отображения результатов в виде графиков.
4. Обеспечивать преподавателям функционал для создания, редактирования и управления методическими материалами, включая текстовые блоки, формулы.
5. Поддерживать возможность масштабирования для добавления новых функций, модулей и материалов.

Задачи разработки:

1. Разработать бэкенд с использованием FastAPI и PostgreSQL, обеспечивающий работу с базой данных, содержащий необходимые маршруты для авторизации, добавления, удаления, редактирования данных.
2. Разработать фронтенд на React и TypeScript, поддерживающий работу с бэкендом, с механизмом навигации, возможностью ввода математических формул с помощью виртуальной клавиатуры, возможностью отображения данных.
3. Организовать систему автоматизированного развертывания и тестирования с использованием Docker и GitHub Actions.
4. Разработать и интегрировать в фронтенд модуль для математических экспериментов с визуализацией результатов в виде графиков.

# 1.4. Используемые технологии

В процессе разработки проекта использовались современные технологии для обеспечения производительности, надежности и масштабируемости системы.

Бэкенд реализован с использованием Python[3] и фреймворка FastAPI[4], который обеспечивает быструю и асинхронную обработку запросов, а также удобное создание RESTful API[11]. В качестве базы данных используется PostgreSQL[5], а для управления миграциями применяется Alembic[8]. Для асинхронного взаимодействия с базой данных используется SQLAlchemy[7]. Контейнеризация проекта выполнена с использованием Docker [9] и docker-compose, что упрощает развёртывание и масштабирование системы. Автоматизация процессов деплоя реализована с помощью GitHub Actions. Для тестирования использован pytest.

Фронтенд построен с применением React[12] и TypeScript[13], что обеспечивает типизацию и безопасность кода. Сборка и оптимизация проекта осуществляются с использованием Vite[14]. Маршрутизация во фронтенде организована с помощью React Router, а для работы с математическими выражениями интегрирована библиотека MathLive[15]. Для построения графиков и визуализации данных использованы JavaScript-библиотеки.

Процесс развертывания автоматизирован с помощью CI/CD через GitHub Actions, что позволяет автоматически тестировать и деплоить изменения при пуше в основную ветку репозитория. Это значительно ускоряет цикл разработки и снижает вероятность ошибок при развертывании.

Серверная часть приложения работает на операционной системе Linux. Для обработки HTTP-запросов и проксирования трафика между пользователями и приложением использован Nginx — высокопроизводительный веб-сервер[17].

# ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА

Для достижения цели проекта перед нами были поставлены задачи по разработке бэкенда, фронтенда, а также реализации модуля экспериментов. Данная глава посвящена описанию теоретических и практических подходов, использованных для реализации этих задач.

# 2.1. Теоретические разработки

Так как для создания проекта необходимо было реализовать модуль экспериментов, нужно было разработать эти эксперименты. Они должны были представлять собой решение математических задач с вводом параметров, а также иметь возможность распараллеливания. В рамках проекта были рассмотрены решения задач с использованием численных методов и параллельных вычислений. В частности, мы сосредоточились на решении систем линейных алгебраических уравнений (СЛАУ) и численном интегрировании, которые являются ключевыми для решения большинства типовых задач курса "Математическое моделирование природных систем". Для этого мы выбрали две задачи: решение СЛАУ и вычисление определенных интегралов.

Задача решения СЛАУ представляет собой стандартную задачу в математическом моделировании. В традиционных подходах для решения таких систем применяется метод Гаусса [18], который предполагает преобразование системы уравнений в верхнетреугольную форму с помощью элементарных операций, после чего можно найти решение методом обратной подстановки. Метод Гаусса можно описать как последовательное исключение переменных (рис. 1). Для каждого уравнения в системе выполняется операция вычитания с пропорциональными коэффициентами, пока система не станет верхнетреугольной. Процесс решается по следующему алгоритму:

1. В первом шаге выбирается главный элемент в первой строке и осуществляется вычитание всех последующих строк с учетом пропорциональных коэффициентов.
2. Далее в каждой следующей строке повторяется операция исключения переменной, пока не будет получена верхнетреугольная матрица.
3. Наконец, происходит обратная подстановка, начиная с последнего уравнения, и решение системы восстанавливается по частям.

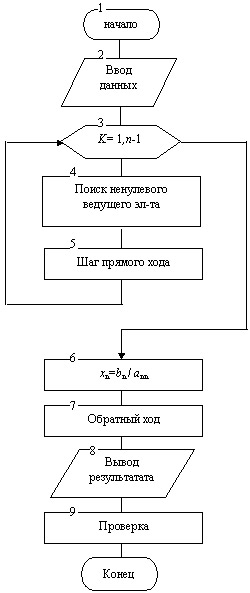


Рисунок 1. Блок-схема метода Гаусса

Однако этот алгоритм является последовательным и может требовать значительных вычислительных ресурсов при больших значениях n. В рамках нашего проекта мы нашли способ распараллелить этот процесс.

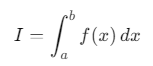
Для ускорения вычислений мы реализовали параллельное решение этой задачи. Мы использовали многозадачность для распараллеливания этапов исключения переменных, при этом разделив работу на несколько потоков. Каждый поток обрабатывает части матрицы, например, вычисляя пропорциональные коэффициенты и выполняя вычитания с определенной подматрицей. Это позволяет значительно ускорить процесс, так как каждый поток работает с подмассивами и вычисляет нужные элементы независимо от других потоков.

Скорость работы зависит от размерности матрицы и кол-ва потоков. При тестировании производительность у параллельной реализации была ниже на малых размерностях и выше на больших размерностях. При тестировании были получены следующие результаты:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Размерность матрицы(n) | Кол-во потоков | Время, мс |
| Последовательный способ | 3 | - | 0 |
| Параллельный способ | 3 | 2 | 14,9 |
| Последовательный способ | 300 | - | 114,2 |
| Параллельный способ | 300 | 2 | 72,5 |
| Последовательный способ | 300 | - | 98,1 |
| Параллельный способ | 300 | 4 | 46 |
| Последовательный способ | 300 | - | 106,1 |
| Параллельный способ | 300 | 6 | 342,2 |
| Последовательный способ | 600 | - | 1186 |
| Параллельный способ | 600 | 2 | 678,4 |

Время работы параллельного способа также зависит и от количества потоков. На 4 потоках получена максимальная эффективность, в то время как на 6 результат становится значительно хуже последовательного способа.

Задача численного интегрирования заключается в нахождении значения определенного интеграла функции f(x) на отрезке [a,b]:



Одним из основных методов численного интегрирования является метод трапеций (рис. 2) [19]. В этом методе интеграл аппроксимируется суммой площади трапеций, образующихся при разбиении отрезка [a,b] на n равных частей.

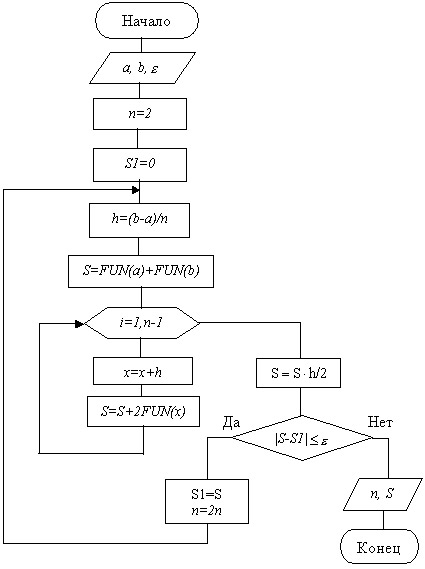
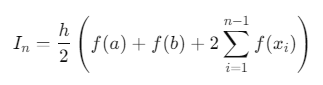


Рисунок 2. Блок-схема метода трапеций

Формула для метода трапеций выглядит следующим образом:



Этот метод является достаточно простым и эффективным для небольших значений n, но также имеет свои ограничения при увеличении размерности, так как для точности вычислений необходимо увеличивать количество разбиений. Математически параллельное интегрирование можно представить как разбиение всего отрезка [a,b] на несколько частей, каждая из которых интегрируется независимо в своем потоке. Для этого отрезок делится на t частей, где t — количество потоков. Каждый поток интегрирует свою часть отрезка, после чего результаты суммируются.

Тесты проводились на функции f = sin(x). При изменении границ интегрирования время практически не менялось, поэтому для тестов был выбран отрезок [0, 100].

|  |  |  |  |
| --- | --- | --- | --- |
|  | Кол-во разбиений | Кол-во потоков | Время, мс |
| Последовательный способ | 1000 | - | 0 |
| Параллельный способ | 1000 | 2 | 33 |
| Последовательный способ | 10000 | - | 0,1 |
| Параллельный способ | 10000 | 2 | 254,2 |
| Последовательный способ | 100000000 | - | 906,7 |
| Параллельный способ | 100000000 | 2 | 795,3 |
| Последовательный способ | 100000000 | - | 940,1 |
| Параллельный способ | 100000000 | 4 | 553,1 |
| Последовательный способ | 100000000 | - | 949,8 |
| Параллельный способ | 100000000 | 6 | 515,6 |

Видно, что, как и в случае с СЛАУ кол-во разделений напрямую влияет на результат, как и количество используемых потоков, однако здесь 6 потоков работают эффективнее, чем четыре.

В рамках реализации проекта были разработаны и использованы теоретические подходы, которые обеспечивают научно обоснованную основу работы системы:

1. Анализ требований к образовательной системе

Проведен анализ курса "Математическое моделирование природных систем", включающий ключевые темы и задачи моделирования. Определены типовые задачи, решение которых можно программно описать с использованием параллельных вычислений (например, задачи численного интегрирования; краевые задачи, сводящиеся к решению трехдиагональной матрицы).

1. Формализация задач моделирования

Обоснован выбор численных методов для моделирования, таких как: Методы прямоугольников, трапеций, Симпсона для численного интегрирования; Методы сведения краевой задачи к задаче Коши для решения краевых задач.

1. Теория параллельных вычислений

Рассмотрены различные языки программирования (например GO, TypeScript, C#, C++ и Python) с поддержкой технологии параллельного вычисления для ускорения расчетов. Рассмотрены различные модели параллелизма и их применимость для задач численного интегрирования и решения трехдиагональных матриц.

# 2.2. Практические решения

В процессе реализации проекта были найдены и применены практические решения для достижения поставленных целей:

1. Оптимизация расчетов через параллельное выполнение

Реализованы численные алгоритмы с поддержкой параллельных вычислений на языке TypeScript с применением Web Workers. Достигнуто ускорение моделирования по сравнению с последовательным выполнением.

1. Интерактивный пользовательский интерфейс

Разработан интерфейс для пользователей, включающий в себя такой функционал, как ввод параметров интегрирования, ввод параметров параллельного решения, отображение результатов расчетов в виде графиков.

1. Модульная архитектура

Разделение системы на модули, например модуль расчета, модуль визуализации, модуль взаимодействия с пользователем.

# 2.3. Процесс разработки

Мы начали разработку с проектирования общей структуры приложения. Решили, что будем использовать фронтенд, бэкенд и базу данных (рис. 3).

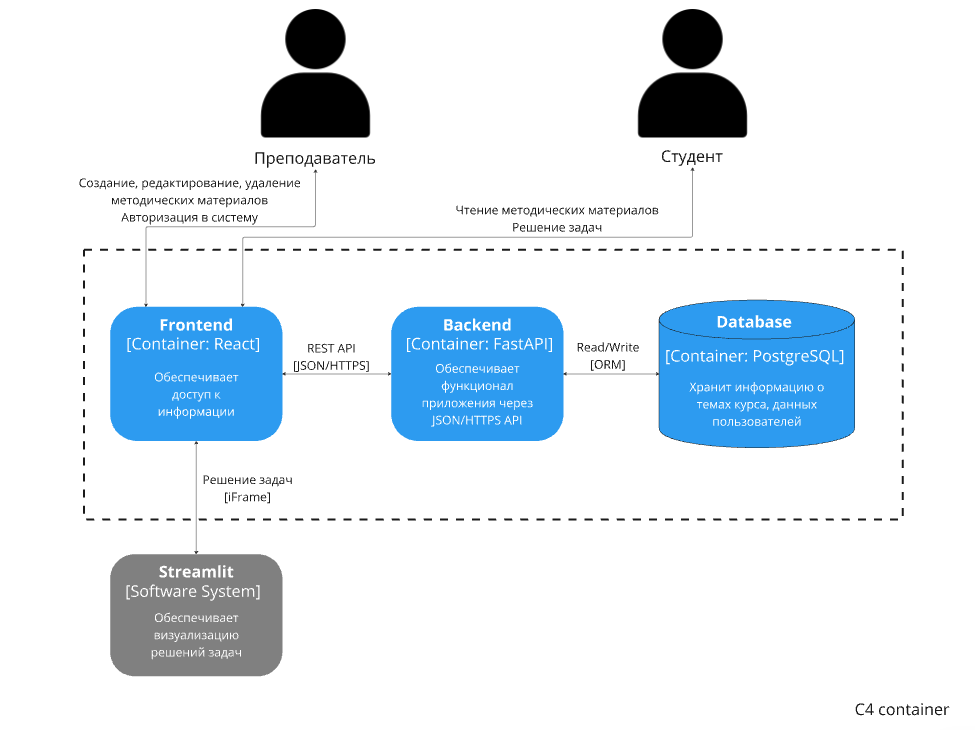


Рисунок 3. Архитектура приложения

Разработали базу данных на PostgreSQL, определив основные таблицы: users, courses, experiments, themes, затем определили, какие поля будут в этих таблицах. Пример таблицы Themes с данными:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | course\_id | title | content | updated\_at | creator\_id | description |
| 2 | 1 | Последовательности и интегралы | json | 2024-11-18 15:57:50 | 1 | desc |

В таблице используются следующие поля:

* + - 1. id – представляет собой целочисленный идентификатор темы.
      2. course\_id – представляет собой целочисленный идентификатор курса, к которой принадлежит тема.
      3. title – представляет собой строку с названием темы.
      4. content – представляет собой json с содержанием темы. В json хранятся типы используемых блоков (text, formula) и их данные.
      5. updated\_at – представляет собой временную метку создания темы.
      6. creator\_id – представляет собой идентификатор пользователя, создавшего тему.
      7. description – представляет собой строковое описание темы.

Общая схема базы данных представлена на рисунке 4.

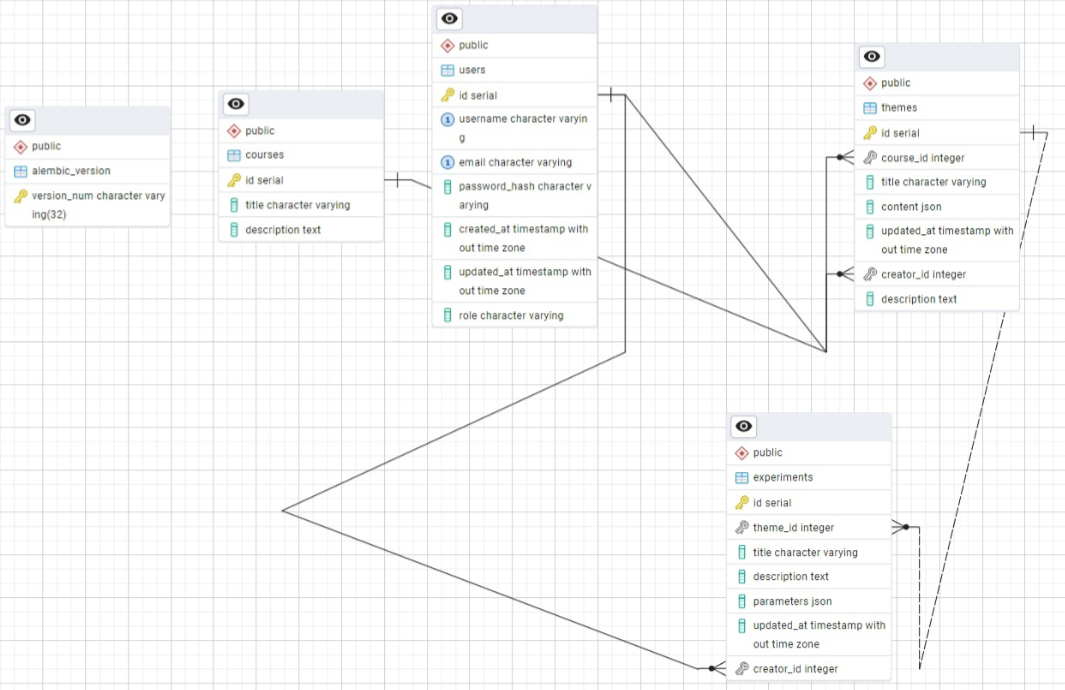


Рисунок . Схема базы данных

Более подробно этот вопрос будет рассмотрен позже, при описании API, так как для базы данных большая часть была создана при помощи ORM.

Далее мы приступили к разработке API для базы данных. Сначала определили, что будем использовать FastAPI для создания REST API и SQLAlchemy в качестве ORM[6] для работы с базой данных. Создали основную директорию проекта, в корне добавили такие файлы, как .env, .gitignore, README.md, requirements.txt, а также конфигурационные файлы для Docker: docker-compose.yml и Dockerfile. Это позволило нам сразу настроить окружение для разработки и последующего деплоя.

Далее мы сосредоточились на настройке подключения к базе данных. Была создана функция для подключения к базе данных и объект сессии для работы с моделями. В конфигурацию добавили параметры подключения из .env. Это позволило хранить все чувствительные данные, не отправляя их в репозиторий.

После настройки подключения мы перешли к созданию моделей. В папке models мы написали несколько файлов, каждый из которых представлял отдельную модель для работы с конкретной сущностью: курсы, эксперименты, темы и пользователи. Мы определили в этих файлах структуры таблиц, их поля, связи между ними. Например, модель пользователя включает идентификатор, юзернейм, email и хэш пароля и т.д, а модель курса содержит поля для названия и описания. Эти модели были описаны с использованием ORM, через SQLAlchemy.

Следующим шагом мы перешли к созданию API-эндпоинтов (рис. 5). Мы создали папку routers и начали с разработки маршрутов для базовых операций. Например, в файле auth.py мы реализовали регистрацию и авторизацию пользователей, настроили JWT-токен для защиты эндпоинтов. Фрагмент кода auth.py представлен листингом в приложении 1. В других маршрутах, таких как course.py или experiments.py, мы разработали эндпоинты для работы с курсами и экспериментами: добавление, удаление, обновление, получение данных. Фрагмент кода course.py представлен листингом в приложении 2. Кроме того, были разработаны схемы чтобы валидировать входные и выходные данные через Pydantic. Например, схема пользователя должна содержать поля, которые будут проверяться при авторизации или регистрации.



Рисунок 5. Документация Swagger, содержащая эндпоинты

После завершения разработки основных эндпоинтов мы приступили к тестированию. В папке tests создали несколько тестов, например, для авторизации (test\_auth.py) и базовой функциональности приложения (test\_main.py). Фрагмент кода для тестирования авторизации представлен листингом в приложении 3. Для тестирования использовали pytest, настроили данные и тестовую базу, чтобы изолировать тесты от реальной базы данных. Мы проверяли, чтобы все эндпоинты работали корректно, возвращали ожидаемые статусы и данные.

На завершающем этапе мы подготовили проект для деплоя. Использовали Docker для создания контейнера. В файле docker-compose.yml настроили взаимодействие между контейнерами приложения и базы данных. Создали deploy.yaml, чтобы настроить CI/CD с помощью GitHub Actions. Это позволило автоматически тестировать и деплоить проект при пуше в основную ветку репозитория.

Далее приступили к созданию фронтенд части. Сначала создали базовую структуру проекта, используя фреймворк Vite, который был выбран для сборки и быстрой разработки фронтенда на TypeScript и React. После инициализации проекта, мы добавили все необходимые конфигурации: файлы tsconfig.json для TypeScript, а также файлы vite.config.ts для настройки сборщика. Далее был подготовлен index.html как основной файл разметки, в который подключается корневой компонент React — App.tsx. В main.tsx мы настроили корневой рендеринг приложения с использованием библиотеки ReactDOM.

В папке src были разделены файлы на отдельные категории: components для компонентов, pages для отдельных страниц приложения, services для взаимодействия с бэкендом, а также types для хранения типов и интерфейсов TypeScript. Далее мы приступили к разработке главного компонента App.tsx, где подключили маршрутизацию с помощью библиотеки React Router. Для этого мы добавили зависимости маршрутизатора и настроили роутинг в App.tsx, указав пути и соответствующие страницы. Фрагмент кода для роутинга представлен листингом в приложении 4.

Следующим этапом стало создание страниц приложения в папке pages. Мы начали с LoginPage.tsx, где реализовали форму авторизации с полями для ввода логина и пароля. Для обработки аутентификации был создан сервис authService.tsx в папке services. В этом файле мы написали функции для отправки данных на сервер с помощью fetch или библиотеки axios и обработки ответов. Мы добавили проверку ошибок, обработку токенов и сохранение их в localStorage для дальнейшего использования.

После этого приступили к разработке страниц ThemesPage.tsx и ThemeContentPage.tsx. В ThemesPage.tsx реализовано отображение списка тем (рис. 6), которые запрашиваются от сервера через сервис themeService.tsx. Для этого в themeService.tsx были созданы функции для получения, добавления и редактирования тем. На этой странице мы добавили возможность перехода к конкретной теме, где уже в ThemeContentPage.tsx отображалось её содержание. На странице реализована логика для отображения информации через получение данных конкретной темы по её id из API.

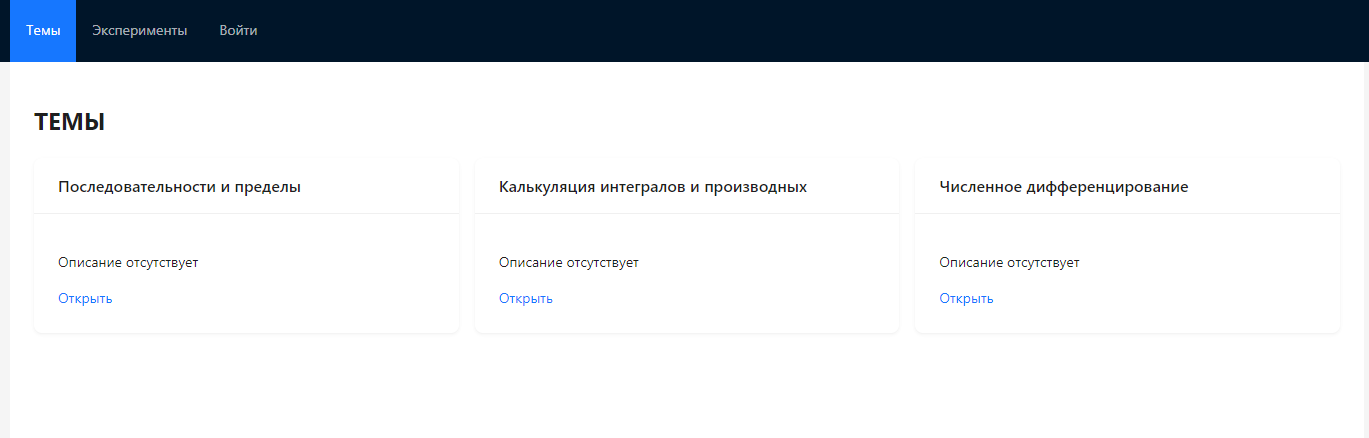


Рисунок 6. Фрагмент страницы «Темы»

Далее были созданы CreateThemePage.tsx и EditThemePage.tsx, которые отвечали за создание и редактирование тем. На этих страницах мы использовали формы, созданные в ThemeForm.tsx. В этом компоненте написана универсальная форма с валидацией полей и отправкой данных на сервер через уже существующие функции из themeService.tsx. При ошибках валидации, они отображаются под соответствующими полями. Фрагмент кода редактирования содержания тем представлен листингом в приложении 5.

После страниц с темами была разработана страница ExperimentsPage.tsx, где отображается список экспериментов. Для работы с ними мы создали аналогичный themeService сервис. Здесь мы написали функции для получения и данных по экспериментам. Сами функции написаны на TypeScript и выполняются на компьютере пользователя. Такое решение было принято, чтобы не нагружать сервер, так как одновременно сайтом могут пользоваться много пользователей. Кроме того, для распараллеливания нужно иметь минимум два потока, аренда сервера с таким процессором была бы дорогостоящей. Распараллеливание реализовано при помощи средства запуска скриптов в фоновом потоке Web Workers. Код реализации функций представлен листингами в приложениях 6-7. Также был добавлен ExperimentContentPage.tsx, где реализовано отображение конкретного содержимого эксперимента (рис. 7).

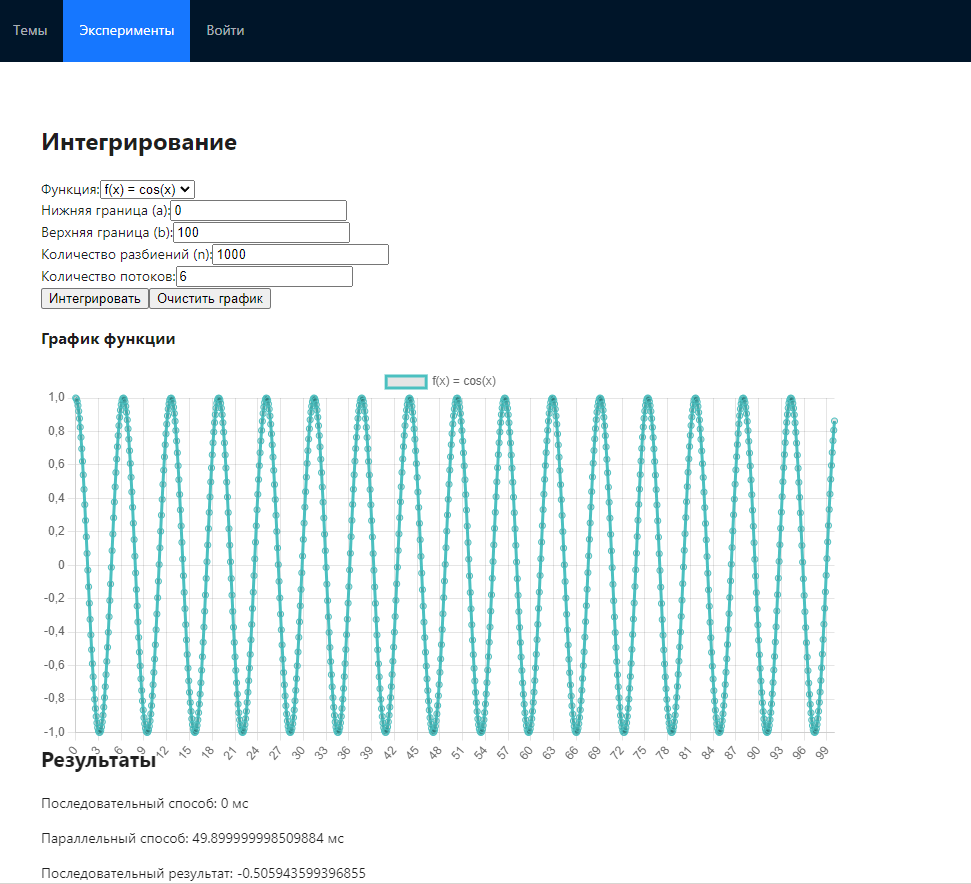


Рисунок 7. Фрагмент страницы эксперимента

Затем мы приступили к созданию форм для ввода математических формул. Был реализован компонент для ввода и редактирования математических формул. Этот компонент взаимодействовал с библиотекой для работы с математической разметкой MathLive. Это позволило в удобной форме вводить формулы через специальный редактор, который затем переводит формулу в LaTeX для отправки на сервер (рис. 8).

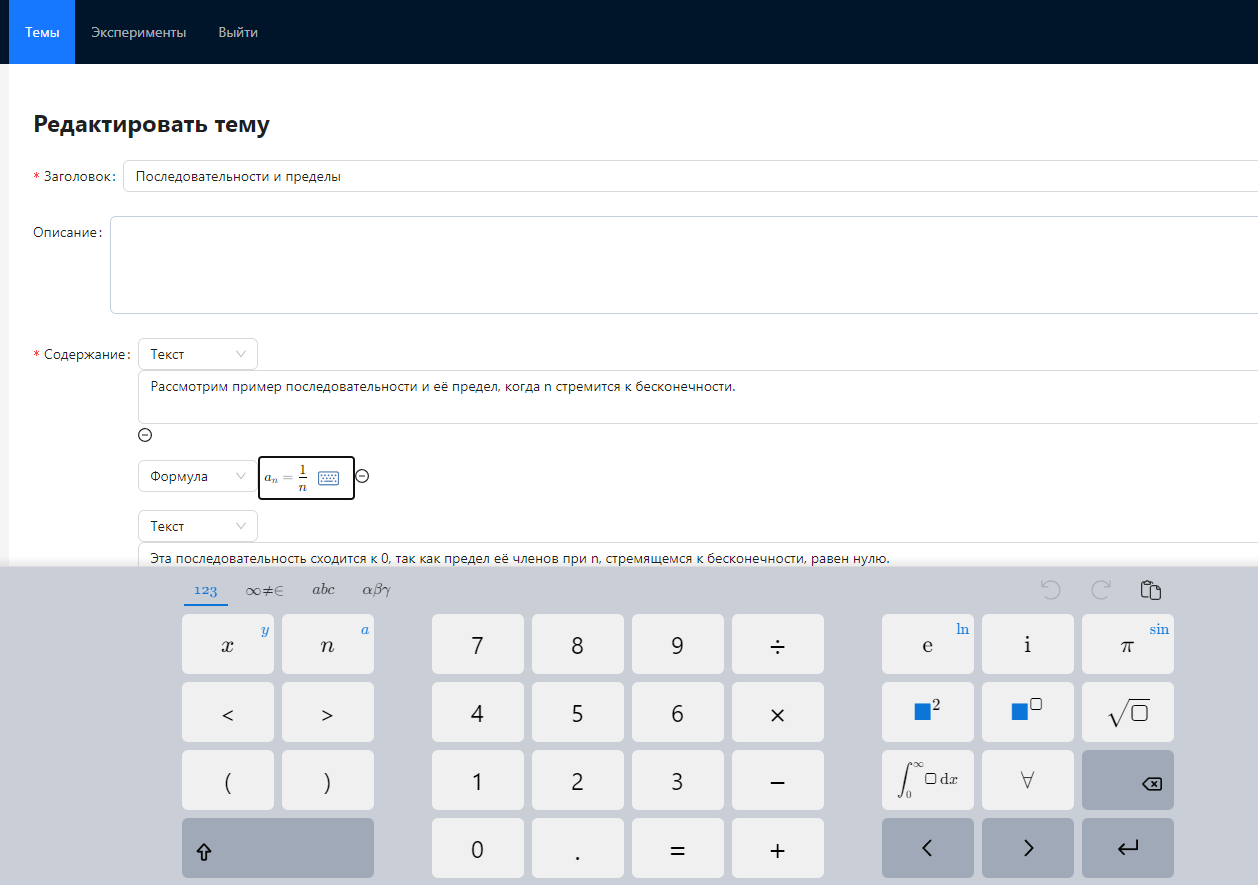


Рисунок 8. Редактор формул LaTeX при редактировании темы

После разработки всех страниц и компонентов мы занялись стилизацией приложения. Для этого использовались глобальные стили в index.css. На этом этапе также была проведена работа с адаптивностью, чтобы приложение корректно отображалось на разных экранах.

Для развертывания приложения в интернете был арендован VPS (виртуальный частный сервер) на базе операционной системы Linux. На этом сервере был установлен Docker для контейнеризации приложения и Nginx для проксирования HTTP-запросов и обеспечения доступа к приложению через интернет. В процессе развертывания было настроено взаимодействие между контейнерами API-приложения и базой данных с использованием Docker Compose.

Для автоматизации деплоя через GitHub Actions был создан SSH-ключ, который позволил безопасно подключаться к серверу и развертывать приложение без необходимости вручную вводить данные для аутентификации. SSH-ключ был добавлен на сервер, а также использован в настройках GitHub Actions для безопасного взаимодействия с сервером.

В GitHub Actions для каждого репозитория были прописаны соответствующие секреты: SSH-ключи, логины, пароли для подключения к серверу, а также другие конфиденциальные данные, такие как переменные окружения для базы данных, пути к приложениям. Это обеспечило безопасный процесс деплоя и автоматизацию на каждом этапе, начиная от тестирования и сборки приложения до его развертывания на сервере.

На сервере был настроен конфигурационный файл для Nginx, который обеспечил доступ к приложению через домен nsmcourse.ru. Nginx выполняет роль обратного прокси-сервера, перенаправляя запросы на соответствующие контейнеры и обеспечивая необходимую производительность и безопасность.

В процессе разработки был использован Git для контроля версий и управления кодом. Все изменения в проекте фиксировались в репозиториях, и через GitHub проводился процесс автоматического деплоя, что позволило минимизировать ошибки и ускорить развертывание обновлений. На финальном этапе было произведено тестирование приложения на всех страницах, проверка работы взаимодействия с API и корректность отображения данных. Мы выявили и устранили ошибки, а также оптимизировали код в компонентах и сервисах.

Все поставленные задачи были выполнены. В процессе работы над проектом был разработан бэкенд, включающий API с использованием фреймворка FastAPI, а также база данных на PostgreSQL. База данных содержит таблицы для хранения необходимых данных, а API позволяет получать к ним доступ с помощью маршрутов, обеспечивающих создание, редактирование и удаление этих данных. Также был разработан фронтенд на React и Typescript, взаимодействующий с бэкендом и поддерживающий необходимые возможности, такие как навигация, ввод математических формул и т. д. В него также был интегрирован модуль проведения математических экспериментов, а проект имеет систему автоматизированного развертывания с помощью Docker и GitHub Actions.

# ГЛАВА 3. ФУНКЦИОНАЛ И ТЕСТИРОВАНИЕ ПРОЕКТА

# 3.1. Пользовательский интерфейс

Интерфейс системы состоит из нескольких основных экранов. После авторизации студент или преподаватель попадает на главную страницу, где видит список разделов курса. Сверху находится панель навигации с доступом к разделам: "Темы", "Эксперименты", "Тесты" и "Отчеты".

Экран "Методические материалы" состоит из двух частей: слева расположено дерево тем с разделами и подразделами, а справа открывается содержимое выбранной темы в виде текста с иллюстрациями и графиками. В верхней части экрана предусмотрена строка поиска для быстрого нахождения тем по ключевым словам.

Экран "Эксперименты" предлагает форму для ввода параметров, таких как размеры данных, количество потоков или шаг расчета. После ввода параметров кнопка "Запустить эксперимент" активирует расчет, а результаты отображаются ниже в виде графика, таблицы или диаграммы. Под графиком отображаются краткие текстовые выводы о результатах эксперимента.

Экран "Тесты" показывает список доступных заданий. При выборе теста открывается страница с вопросами: студенты выбирают ответы, вводят значения или загружают файлы с результатами. После завершения теста система автоматически оценивает работу и показывает результаты.

Для преподавателей предусмотрен экран редактирования материалов. Он содержит текстовый редактор с полем для добавления текста, изображений и таблиц. Слева отображается структура курса, где можно создавать новые темы или изменять существующие.

В разделе "Отчеты" преподаватели видят список студентов с информацией об их активности и результатах тестов. Клик по студенту открывает детализированный отчет с прогрессом по темам и результатами экспериментов.

Дизайн интерфейса предусматривает четкую структуру с минимальным количеством элементов на экране, чтобы пользователю было удобно сосредоточиваться на текущей задаче.

# 3.2. Функциональные возможности

Функциональные возможности разрабатываемой интерактивной образовательной системы включают авторизацию, управление методическими материалами, проведение математических экспериментов, визуализацию данных, навигацию по материалам курса, обновление материалов в реальном времени. Авторизация обеспечивает безопасный вход в систему для преподавателей. После авторизации преподаватели могут управлять методическими материалами, добавляя, редактируя и удаляя их. При проведении математических экспериментов студенты могут изменять входные параметры и анализировать результаты в наглядной форме с использованием графиков и диаграмм. Визуализация данных упрощает восприятие результатов и делает их более доступными для анализа. Навигация организована по темам, что позволяет быстро находить нужную информацию. Обновление материалов возможно в реальном времени, что поддерживает их актуальность. Поддержка многопользовательской работы обеспечивает стабильность системы даже при высокой нагрузке.

# 3.3. Тестирование проекта

Тестирование использовалось для выявления и устранения ошибок, проверки корректности работы компонентов системы. В нашем проекте тестирование проводилось как вручную, так и с использованием автоматических тестов, что позволило убедиться в правильности функционирования всех функциональных блоков системы, в том числе бэкенда, фронтенда и базы данных. Тестирование применялось на всех этапах разработки проекта, оно позволяло выявлять проблемы и сразу же исправлять их.

Тестировались в том числе блоки, к которым у пользователя нет доступа напрямую, например API приложения. Пользователь не будет обращаться напрямую в API, однако его корректная работа критична для правильного функционирования всей системы. В ходе тестирования API проверялась правильность обработки запросов, валидация данных, корректность формирования ответов и обработка ошибок. Активно применялись автоматические тесты, написанные с использованием pytest. Эти тесты позволяли проверять корректность работы API и бизнес-логики системы. Тестировались различные сценарии работы с данными: создание, обновление и удаление информации, обработка ошибок, а также тестировалась работа с базой данных, включая правильность выполнения SQL-запросов. Также отрабатывались различные ситуации, такие как обработка некорректных запросов. Автоматические тесты позволяют проверять правильность работы системы во время разработки, так как могут возникать ситуации, при которых добавление нового функционала может мешать работе старого. После каждого введения нового функционала запускалась серия тестов, и, если они проходили успешно, функционал внедрялся в систему.

Рассмотрим использование автоматического тестирования на примере функционала авторизации. В ходе этого тестирования проверялись различные сценарии: ввод верных данных, ввод неверных данных, ввод пустых данных. Правильной работой сервиса считалось наличие отклика на каждый сценарий. При вводе верных данных пользователь должен авторизоваться в систему, в остальных случаях система должна оставаться работоспособной и сообщать пользователю причину, по которой авторизация не была успешной. Отработка тестов представлена на рисунке 9.

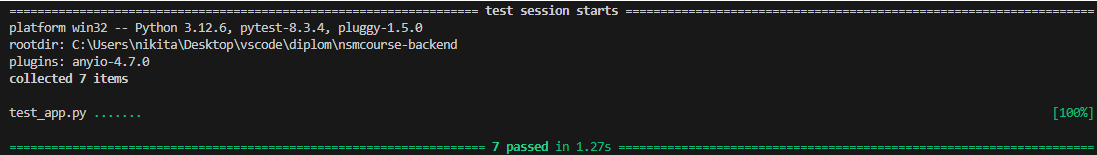


Рисунок 9. Тестирование API

В процессе тестирования фронтенда также важно было убедиться, что все компоненты интерфейса работают как ожидается, и что взаимодействие с бэкендом происходит корректно. Для этого применялись ручные тесты, которые включали проверку функциональности всех пользовательских интерфейсов, таких как формы регистрации, авторизации, взаимодействие с динамическими элементами страницы, а также корректность отображения данных, получаемых от API. Возможные сценарии использования интерфейса были протестированы вручную, чтобы убедиться в правильности работы всех функций. При тестировании также уделялось внимание проверке адаптивности приложения на различных устройствах и браузерах, что гарантировало пользователям доступ к функционалу на любых экранах. Так как мы адаптируем приложение под десктоп, тестировалось минимальное разрешение 800x600 и максимальное 4К, для этого использовались разные инструменты, такие как тестирование в браузере с изменением размеров окна, а также тестирование на реальных устройствах. Эти тесты помогли выявить и устранить проблемы с отображением и взаимодействием на различных устройствах.

# ЗАКЛЮЧЕНИЕ

В рамках работы был разработан проект интерактивной образовательной системы, в котором реализован механизм централизованного хранения, управление и использование методических материалов для дисциплины "Математическое моделирование природных систем". Основными задачами проекта являлись оптимизация доступа к актуальным методическим материалам, организация эффективного проведения математических экспериментов и улучшение визуализации результатов расчетов.

Система построена на современной клиент-серверной архитектуре. Серверная часть отвечает за хранение данных и выполнение расчетных задач, а клиентская часть обеспечивает взаимодействие с пользователем и визуализацию информации. Использование стек-технологий Vite, React и TypeScript позволило создать модульное, производительное и масштабируемое приложение. Организация структуры проекта с четким разделением на страницы, компоненты и сервисы обеспечила упрощение поддержки и добавления нового функционала.

В проекте были реализованы ключевые модули, включая модуль ввода параметров моделирования, модуль расчетов с поддержкой параллельных вычислений и модуль визуализации результатов. Взаимодействие между элементами системы организовано через API, что позволяет эффективно передавать данные и обеспечивать целостность системы. Для обработки математических выражений была интегрирована библиотека MathLive, а для построения графиков и анимаций использованы внешние JavaScript-библиотеки, что позволило наглядно представлять результаты экспериментов.

Для развертывания системы в интернете был использован VPS на базе Linux, на котором установлены Docker и Nginx. Это позволило организовать контейнеризацию приложения, настроить взаимодействие с базой данных и обеспечить безопасный доступ через домен nsmcourse.ru. Автоматизация деплоя была настроена с использованием GitHub Actions, что ускоряет процесс развертывания и позволяет автоматически тестировать и обновлять систему. Это решение обеспечивает гибкость, безопасность и высокую доступность приложения.

Реализация проекта продемонстрировала возможность создания инструмента для образовательного процесса. Система упрощает доступ к методическим материалам для студентов и преподавателей, снижает временные затраты на актуализацию данных и предоставляет мощные инструменты для проведения математических экспериментов. Благодаря модульной архитектуре и использованию современных технологий система обладает высокой гибкостью и потенциалом для дальнейшего масштабирования, что делает её универсальной в образовательной среде.

# СПИСОК ЛИТЕРАТУРЫ

1. Web-платформа для создания интерактивных обучающих курсов по вычислительным методам Т.Г. Салуев, И.В. Оселедец, Р.Ю. Фадеев // ИСА РАН URL: http://www.isa.ru/aidt/2014-01/46\_51.pdf (дата обращения: 16.09.2024)
2. YouGile - о компании // YouGile URL: https://ru.yougile.com/about (дата обращения: 16.09.2024).
3. Основы языка программирования Python // Ru-center URL: https://www.nic.ru/help/osnovy-yazyka-programmirovaniya-python\_11662.html (дата обращения: 10.10.2024). Основы работы с функциями и модулями.
4. Документация // FastAPI URL: https://fastapi.tiangolo.com/ru/ (дата обращения: 12.10.2024). Работа с зависимостями и валидацией данных.
5. PostgreSQL // Википедия URL: https://ru.wikipedia.org/wiki/PostgreSQL (дата обращения: 16.10.2024). Основные функции и возможности СУБД.
6. ORM // SimpleOne URL: https://simpleone.ru/glossary/orm/ (дата обращения: 16.10.2024). Преимущества и недостатки использования ORM.
7. SQLAlchemy // Википедия URL: https://ru.wikipedia.org/wiki/SQLAlchemy (дата обращения: 16.10.2024). Основные компоненты и архитектура библиотеки.
8. Документация // Alembic URL: https://alembic.sqlalchemy.org/en/latest/ (дата обращения: 14.10.2024). Создание и применение миграций базы данных.
9. Как работает Docker: подробный гайд от техлида // Skillbox Media URL: https://skillbox.ru/media/code/kak-rabotaet-docker-podrobnyy-gayd-ot-tekhlida/ (дата обращения: 16.11.2024). Создание и управление Docker-контейнерами.
10. Что такое Postman и как его использовать в тестировании API // Хекслет URL: https://ru.hexlet.io/blog/posts/postman (дата обращения: 16.11.2024). Создание и отправка HTTP-запросов. Тестирование и отладка API с помощью Postman.
11. Архитектура REST // Хабр URL: https://habr.com/ru/articles/38730/ (дата обращения: 12.10.2024). HTTP-методы и статус-коды.
12. React Учебник // schooksw3 URL: https://www.schoolsw3.com/react/index.php (дата обращения 16.11.2024) Создание и управление компонентами.
13. Документация TypeScript // TypeScriptLang URL: https://www.typescriptlang.org/docs/ (дата обращения 16.11.2024) Основные типы и интерфейсы. Работа с декораторами и модулями.
14. Документация Vite // ViteDev URL: https://vite.dev/guide/ (дата обращения 16.11.2024) Создание и настройка проекта с Vite.
15. Документация MathLive // CortexJS URL: https://cortexjs.io/mathlive/ (дата обращения 16.11.2024) Интеграция MathLive в веб-приложения.
16. Основы вычислительной математики Н. М. Гаврилова // URL: http://iside.distance.ru/w/Books/56733.pdf (дата обращения 25.09.2024)
17. Nginx // URL: https://nginx.org/en/ (дата обращения: 15.12.2024). Основные конфигурационные файлы и директивы. Настройка виртуальных хостов и проксирования.
18. Решение систем линейных уравнений методом Гаусса // Томский государственный педагогический университет URL: https://koi.tspu.ru/koi\_books/gazizov/l9p02.htm (дата обращения: 19.12.2024).
19. Метод трапеций // Томский государственный педагогический университет URL: https://koi.tspu.ru/koi\_books/gazizov/l10p03.htm (дата обращения: 19.12.2024).

Приложение 1

Фрагмент кода auth.py

@auth\_router.post("/token")

def login(

form\_data: OAuth2PasswordRequestForm = Depends(),

db: Session = Depends(get\_db)

):

token = authenticate\_user(db, form\_data.username, form\_data.password)

if not token:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Incorrect email or password",

)

return {"access\_token": token, "token\_type": "bearer"}

@auth\_router.post("/register\_admin")

def register\_admin(

username: str,

email: str,

password: str,

db: Session = Depends(get\_db)

):

existing\_user = db.query(User).filter(User.email == email).first()

if existing\_user:

raise HTTPException(status\_code=400, detail="Email already registered")

hashed\_password = hash\_password(password)

new\_user = User(

username=username,

email=email,

password\_hash=hashed\_password,

role="admin",

created\_at=datetime.utcnow(),

updated\_at=datetime.utcnow()

)

db.add(new\_user)

db.commit()

db.refresh(new\_user)

return {"detail": "Admin registered successfully"} quantity = Column(Float, nullable=False)

status = Column(SmallInteger, nullable=False, default=0)

is\_called = Column(Boolean, nullable=False, default=False)

is\_push\_sent = Column(Boolean, nullable=False, default=False)

delay = Column(SmallInteger, nullable=False, default=0, server\_default='0')

course = relationship('Course', backref='intakes')

def \_\_repr\_\_(self):

return f"id: {self.id}"

Листинг SEQ Листинг \\* ARABIC 1

# ПРИЛОЖЕНИЯ 1-7

Приложение 2

Фрагмент кода course.py

course\_router = APIRouter()

@course\_router.get("/courses", response\_model=list[CourseResponse])

def get\_courses(db: Session = Depends(get\_db)):

courses = db.query(Course).all()

if not courses:

raise HTTPException(status\_code=404, detail="courses not found")

return courses

@course\_router.post("/courses", response\_model=CourseResponse)

def create\_course(course: CourseCreateUpdate, db: Session = Depends(get\_db)):

db\_course = Course(

title=course.title,

description=course.description

)

db.add(db\_course)

db.commit()

db.refresh(db\_course)

return db\_course

Листинг SEQ Листинг \\* ARABIC 2

Приложение 3

Фрагмент кода теста авторизации пользователей

def test\_login(db\_session):

db\_session.query(User).filter(User.email == "testuser@example.com").delete()

db\_session.commit()

password = "testpassword"

hashed\_password = hash\_password(password)

test\_user = User(username="testuser", email="testuser@example.com", password\_hash=hashed\_password, role="user", created\_at="2024-01-01", updated\_at="2024-01-01")

db\_session.add(test\_user)

db\_session.commit()

response = client.post("/token", data={"username": "testuser@example.com", "password": password})

assert response.status\_code == 200

assert "access\_token" in response.json()

response = client.post("/token", data={"username": "testuser@example.com", "password": "wrongpassword"})

assert response.status\_code == 401

assert response.json() == {"detail": "Incorrect email or password"}

Листинг SEQ Листинг \\* ARABIC 3

Приложение 4

Фрагмент кода роутинга

<Layout className="layout">

<Header>

<div className="logo" />

<Menu theme="dark" mode="horizontal" defaultSelectedKeys={['1']} items={menuItems} />

</Header>

<Content style={{ padding: '0 50px' }}>

<div className="site-layout-content">

<Routes>

<Route path="/" element={<Navigate to="/themes" replace />} />

<Route path="/themes" element={<ThemesPage isAdmin={isAdmin} />} />

<Route path="/themes/:themeId" element={<ThemeContentPage />} />

<Route path="/themes/new" element={<CreateThemePage />} />

<Route path="/themes/:themeId/edit" element={<EditThemePage />} />

<Route path="/experiments" element={<ExperimentsPage />} />

<Route path="/experiments/:experimentId" element={<ExperimentContentPage />} />

<Route path="/login" element={<LoginPage onLogin={() => setIsAdmin(true)} />} />

</Routes>

</div>

</Content>

</Layout>

Листинг 4

Приложение 5

Фрагмент кода формы для редактирования содержания тем

<Form.Item label="Содержание" required>

{content.map((item, index) => (

<div key={index} style={{ marginBottom: "10px" }}>

<Select

value={item.type}

onChange={(value) => handleContentChange(index, "type", value)}

style={{ width: "120px" }}

>

<Select.Option value="text">Текст</Select.Option>

<Select.Option value="formula">Формула</Select.Option>

</Select>

{item.type === "text" ? (

<Input.TextArea

value={item.value}

onChange={(e) => handleContentChange(index, "value", e.target.value)}

required

/>

) : (

<FormulaInput

value={item.value}

onChange={(value) => handleContentChange(index, "value", value)}

/>

)}

<MinusCircleOutlined onClick={() => removeContentItem(index)} />

</div>

))}

<Button

type="dashed"

icon={<PlusOutlined />}

onClick={addContentItem}

>

Добавить содержимое

</Button>

</Form.Item>

Листинг 5

Приложение 6

Фрагмент кода параллельного вычисления численного интеграла

const parallelIntegration = (funcString: string, a: number, b: number, n: number, threads: number) => {

return new Promise<number>((resolve) => {

threads = Math.min(threads, n);

const chunkSize = Math.ceil(n / threads);

const workers: Worker[] = [];

const results: number[] = new Array(threads);

let completedThreads = 0;

for (let i = 0; i < threads; i++) {

const worker = new Worker(new URL('./integrationWorker.ts', import.meta.url));

workers.push(worker);

worker.onmessage = (e) => {

results[i] = e.data;

console.log(`поток ${i} | результат: ${e.data}`);

completedThreads++;

if (completedThreads === threads) {

const totalResult = results.reduce((acc, res) => acc + res, 0);

console.log('финальный результат:', totalResult);

resolve(totalResult);

}

};

const start = a + i \* (b - a) / threads;

const end = a + (i + 1) \* (b - a) / threads;

console.log(`поток ${i}: a = ${start}, b = ${end}`);

worker.postMessage({ funcString, a: start, b: end, n: chunkSize });

}

});

};

Приложение 7

Фрагмент кода реализации WebWorker для вычисления интеграла

self.onmessage = function (e) {

const { funcString, a, b, n } = e.data;

const f = eval(`(${funcString})`);

console.log(`вычисление интеграла на диапазоне [${a}, ${b}]`);

const localResult = trapezoidalRule(f, a, b, n);

postMessage(localResult);

};

const trapezoidalRule = (f: (x: number) => number, a: number, b: number, n: number): number => {

const h = (b - a) / n;

let sum = 0.5 \* (f(a) + f(b));

for (let i = 1; i < n; i++) {

sum += f(a + i \* h);

}

return sum \* h;

};

Листинг 6

Листинг 7

Листинг 8

Листинг 9

Листинг 10

Листинг 11