ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

СОГЛАСОВАНО
Руководитель ВКР,
доцент департамента
программной инженерии
факультета компьютерных наук,
канд. ф.-м. наук
_____ Г.Н. Жукова
«31» ____05_____ 2023 г.

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия», профессор
департамента программной
инженерии, канд. техн. наук
_____ В. В. Шилов
«31» _____мая_____ 2023 г.

Выпускная квалификационная работа
(проектно-исследовательская)

на тему: **Серверная часть приложения для генерации и управления индивидуальными студенческими заданиями для курса «Компьютерный практикум по алгебре на Python»**

по направлению подготовки 09.03.04 «Программная инженерия»

ВЫПОЛНИЛ
студент группы БПИ196
образовательной программы
09.03.04 «Программная инженерия»
_____Н.Д. Сахаров_____
И.О. Фамилия
_____
Подпись, Дата

Москва 2023

Реферат

Данная выпускная квалификационная работа описывает разработку серверной части приложения на Python, предназначенного для генерации и управления индивидуальными студенческими заданиями для курса «Компьютерный практикум по алгебре на Python». В работе представлен анализ существующих решений, описание используемых технологий и методов разработки. Основное внимание уделено реализации функциональности серверной части, включая создание и хранение заданий, обработку запросов от пользователей и генерацию индивидуальных вариантов для студентов. Выпускная квалификационная работа содержит 41 страницу, 3 главы, 17 иллюстраций, 19 источников.

Ключевые слова: клиент-серверное приложение;

Abstract

This graduation qualification project describes the development of the server-side of an application in Python, designed for generating and managing individual student assignments for the "Computer Practicum in Algebra with Python" course. The work presents an analysis of existing solutions, a description of the technologies and development methods used. The main focus is on implementing the functionality of the server-side, including task creation and storage, handling user requests, and generating individual variations for students.

The paper contains 41 pages, 3 chapters, 17 illustrations, and references to 19 sources.

Keywords: client-server application.

Используемые определения и термины

1. API (Application Programming Interface) – Набор правил и протоколов для взаимодействия между компонентами программного обеспечения.
2. HTTP (Hypertext Transfer Protocol) – Протокол передачи данных в Интернете.
3. Эндпоинт – Конечная точка доступа к API или сервису.
4. JSON (JavaScript Object Notation) - это простой формат обмена данными, основанный на тексте, который широко применяется для передачи структурированной информации между клиентскими и серверными приложениями. JSON представляет собой набор пар "ключ-значение", где ключи представлены в виде строк, а значения могут быть строками, числами, логическими значениями, массивами, объектами или null.
5. JSON Web Token (JWT) — это открытый стандарт для создания токенов доступа, основанный на формате JSON.
6. ASGI(Asynchronous Server Gateway Interface)[16] - это спецификация, предназначенная для асинхронного взаимодействия между веб-серверами и веб-приложениями на Python.
7. URL (Uniform Resource Locator) - это адрес, который идентифицирует конкретный ресурс в Интернете.

Оглавление

Введение

Курс «Компьютерный практикум по алгебре на Python» предлагает студентам широкий спектр учебных материалов и лабораторных работ, однако для максимального усвоения учебного материала необходима практика.

Данная выпускная квалификационная работа посвящена разработке серверной части приложения, которое позволит генерировать индивидуальные студенческие задания для каждой из лабораторных работ курса «Компьютерный практикум по алгебре на Python». Созданное приложение поможет преподавателям упростить процес раздачи вариантов заданий и следить за их выполнением студентами, а также облегчить процессы проверки и обсуждения работ.

Во второй главе работы описан краткий обзор существующих решений и обоснование выбора технологий и инструментов разработки.

Третья глава посвящена описанию архитектуры серверной части приложения. В ней представлены описания всех модулей, их функциональности и реализации, а также использованных алгоритмов и структур данных. Также будут описаны методы взаимодействия между модулями и взамиодействие с базами данных, необходимые для обеспечения эффективной работы приложения.

В четвертой главе будет рассмотрен подход к тестированию разработанного приложения. Будет описана инфраструктура, разработанная для проведения тестирования, включая используемые инструменты и технологии. Также будут подробно рассмотрены методы тестирования, включая функциональное тестирование приложения и тестирование отдельных модулей.

Глава 1. Обзор аналогов и выбор технологий

1.1. Описание проблем, решаемых в работе

Для эффективного выполнения лабораторных работ в образовательных учреждениях необходимо регулярно проверять и оценивать работы студентов. Однако, существующие подходы к проверке и оценке лабораторных работ, такие как ручная проверка или использование шаблонных заданий, имеют свои недостатки.

Основные проблемы существующих подходов включают:

- Трудоемкость: ручная генерация и раздача лабораторных работ занимает много времени, особенно при большом количестве студентов;
- Неэффективность: использование шаблонных заданий не учитывает индивидуальные особенности каждой лабораторной работы;
- Недостаточная защищенность от списывания: при использовании шаблонных заданий или проверке лабораторных работ вручную может возникнуть проблема списывания между студентами.

Для решения этих проблем необходимо разработать приложение для генерации и управления индивидуальными лабораторными заданиями, которое позволит автоматизировать процессы проверки и оценки лабораторных работ.

Будут реализованы следующие функции:

- Генерация индивидуальных вариантов для каждой лабораторной работы;
- Предоставление отчетов преподавателю для контроля оценки.

Таким образом, разработка приложения позволит значительно упростить и ускорить процессы выдачи, проверки и оценки лабораторных работ, а также уменьшить возможность списывания между студентами.

1.2. Обзор существующих аналогов

На данный момент существует ряд программных продуктов, которые могут использоваться для генерации и управления индивидуальными лабораторными заданиями. Рассмотрим несколько из них.

1.2.1. Яндекс.Контест

Яндекс.Контест — это система автоматической проверки программного кода, которая позволяет организовывать и проводить соревнования по программированию, а также тестирование знаний студентов. Система предоставляет возможность создавать индивидуальные задания и проверять их автоматически.

Однако, Яндекс.Контест не предоставляет возможность создавать индивидуальные варианты для каждого студента, а также не учитывает индивидуальные особенности каждой лабораторной работы.

1.2.2. Google Classroom

Google Classroom — это система управления обучением, которая позволяет организовывать и проводить занятия в форме онлайн-курсов. Система предоставляет возможность создавать индивидуальные задания и проверять их автоматически.

Однако, Google Classroom, как и предыдущий аналог, не предоставляет возможность создавать индивидуальные варианты для каждого студента, а также не учитывает индивидуальные особенности каждой лабораторной работы.

1.2.3. Moodle

Moodle — это свободная система управления обучением, которая позволяет организовывать и проводить занятия в форме онлайн-курсов. Система предоставляет возможность создавать индивидуальные задания и проверять их автоматически, а также генерировать индивидуальные варианты для каждого студента.

Однако, Moodle может быть сложной в настройке и не всегда предоставляет достаточную гибкость при настройке лабораторных работ.

1.2.4. Вывод

Таблица 1.1. Сравнение аналогов и разработанного приложения

Особенности / Приложение	Яндекс Контест	Google Classroom	Moodle	ВКР
Возможность создания индивидуальных заданий	-	+	+	+
Возможность генерации индивидуальных вариантов для каждого студента	-	-	+	+
Возможность комментирования работ	-	+	+	+
Удобное представление результатов для студентов	-	+	+	+
Удобное представление результатов для преподавателей	-	+	+	+
Гибкость в настройке лабораторных работ	-	-	-	+
Простота в настройке и использовании	+/-	+	-	+

Таким образом, ни один из рассмотренных аналогов не решает всех задач, стоящих перед нашим приложением. Будет проведен дальнейший анализ существующих решений и выбраны наиболее подходящие технологии и инструменты для реализации нашего приложения.

1.3. Выбор технологий и инструментов

1.3.1. Нефункциональные требования к серверу

Сервер для данного проекта не предполагет больших нагрузок, так как планируется использовать его на 1 курсе. Также разработка велась одним человеком, поэтому нужно выбирать технологии разработки, не требующие большие объемы ресурса разработчиков.

Сфомулирую несколько требований к серверу, которые важно было учесть при выборе стека технологий:

- Сервер должен выдерживать нагрузку в 100-150 rps минимум, при этом число пользователей не предполагет больше 1-2 тысяч.
- Сервер должен обеспечивать безопасность хранения пользовательских данных и данных авторизации.
- Необходима масштабируемость генерации вариантов, а также простота добавления новых генераторов.
- Архитетура должна позволять расширить базу курсов.
- Сервер должен легко поддерживаться и не требовать больших ресурсов для разработки.
- Сервер должен продолжительное время хранить работы студентов и их варианты лабраторных работ.
- Поддержка RESTful API, необходимая для удобста взаимодействия с клиентом.

1.3.2. Язык программирования

Выбор языка программирования зависит от нескольких факторов: простоты написания кода, наличия готовых решений, наличия хорошей документации и широкого комьюнити разработчиков.[1] Для данного проекта подходят языки программирования с достаточным количеством готовых библиотек для работы с Jupyter Notebook.

Было рассмотренно несколько вариантов:

- Python имеет достаточно большое количество готовых библиотек для работы с Jupyter Notebook и лучше прочих интегрируется с ним. Все остальные требования к серверу очень покрываются готовыми решениями с хорошей документацией

и активным сообществом разработчиков. Python имеет не самую высокую производительность (по сравнению с Java или C++), но в рамках данного проекта не ожидается большой нагрузки и этот минус можно компенсировать выбором фреймворка.

- Java подходит для разработки серверной части приложения, имеет широкое общество разработчиков и множество готовых решений. Но в контексте данного проекта интеграция с модулями Python для генерации заданий реализуется на порядок сложнее, чем в случае с Python.

- C#: имеет достаточно большое количество готовых библиотек для работы с Jupyter Notebook и имеет хорошую документацию, однако он сильнее связан с операционной системой Windows, а разрабатывать было легче и предпочтительнее на Linux. Могла возникнуть проблема с дополнительная работой по портированию кода.

Учитывая приведенные преимущества и недостатки языков программирования в контексте разработки данного приложения, из всех расмотренных вариантов для предлагаемого проекта лучше всего подходит Python. Так как проектам на Python критически важен менеджмент зависимостей, я буду использовать pipenv[2] для удобного создания окружения проекта.

1.3.3. Фреймворк для серверной разработки

Многое в процессе разработки зависит от выбора фреймворка, включая простоту написания кода, интеграцию с базами данных, производительность сервера. В данном проекте я рассматривал несколько вариантов фреймворков: Django[3] и FastAPI [4, 5].

Django - это полноценный фреймворк с широким функционалом, но в контексте данного проекта, его функционал не потребуется. Кроме того, для работы с Django нужно большое количество кода, что усложнит и замедлит процесс разработки.

FastAPI является быстрым и легковесным фреймворком. Он имеет удобный синтаксис, хорошую документацию и компенсирует накладные расходы на интерпритацию Python. FastAPI не имеет встроенных средств для работы с базами данных, но это легко дополняется сторонними библотеками, такими как SqlAlchemy[6] и FastAPI-Security[7].

1.3.4. Хранение данных

Данные пользователей

Хранение данных очень важный аспект проекта. В связи с этим было принято решение использовать SQL базу данных, которая позволяет хранить и структурировать данные нашего приложения. В качестве диалекта я выбрал PostgreSQL[8], который имеет отличную производительность, поддерживает многопоточность и имеет хорошую масштабируемость. PostgreSQL - один из самых популярных диалектов в соверменной

серверной разработке, поэтому найти всю необходимую информацию по разработке было совсем не сложно.

Для работы с базой данных я выбрал библиотеку SQLAlchemy, которая позволяет работать с базами данных разных типов. Она имеет широкие возможности по работе с ORM, позволяет гибко настраивать запросы к базе данных, а также имеет мощные инструменты для миграции данных и управления схемой базы данных. Это позволило более безопасно огранизовать тестирование. Использование SQLAlchemy сократитило время разработки и обеспечило безопасность приложения благодаря поддержке ORM и возможности работать с транзакциями.

В связи с тем, что в нашем проекте будут храниться личные данные пользователей, включая авторизационные данные, важно обеспечить безопасность хранения паролей. В статье [9] описывается специфика хранения паролей, которой я придерживался при разработке.

Файлы решений и вариантов лабораторных работ

Когда речь идет о хранении файлов, стоит учитывать не только возможности, но и ограничения различных баз данных. PostgreSQL позволяет хранить файлы в колонках формата LargeBinary или Binary, но это не очень удобно и затруднит запросы с использованием сущностей решений и вариантов.

Хранение файлов в отдельной таблице почти эквивалентно хранению их в другой NoSQL базе данных с точностью замены ключа внутри PostgreSQL на ключ в новой базе данных. Поэтому я принял решение хранить файлы отдельно. Есть несколько решений в таком подходе. Я предложил два варианта:

- RocksDB[10] - это распределенная база данных ключ-значение, которая позволяет хранить данные локально. Она отлично подходит для проектов, которые не требуют большого объема данных и не нуждаются в масштабировании. Также RocksDB имеет отличную производительность при работе с большим количеством небольших файлов.

- MongoDB[11] поддерживает хранение и обработку больших объемов данных, что может быть важно для проектов, где требуется хранить много файлов. MongoDB также позволяет быстро выполнять запросы к данным благодаря индексации, что может быть полезно при поиске конкретного файла или группы файлов. Кроме того, она предлагает широкие возможности по работе с репликацией и шардингом, что позволяет масштабировать базу данных по мере роста проекта.

Однако, MongoDB также имеет некоторые минусы, такие как более сложную настройку и управление базой данных по сравнению с RocksDB. В данном проекте было принято решение использовать RocksDB, так как он не требует дополнительного развертывания. Если в будущем потребуется масштабировать проект, можно будет рассмотреть возможность замены RocksDB на MongoDB.

1.3.5. Архитектура сервера

На сегодняшний день существуют различные подходы к архитектуре сервера, однако выбор подходящего решения требует анализа специфики проекта, его целей и ограничений. Я ознакомился с различными ресурсами с целью выбора архитектуры сервиса. Исходя из статьи об архитектурных стилях [12], мной были рассмотрены значимые преимущества и недостатки двух основных архитектурных стилей, каждый из которых, на мой взгляд, мог бы быть актуален при разработке данного проекта: микросервисная архитектура и монолитный подход.

Микросервисная архитектура

Главными плюсами микросервисной архитектуры в современной разработке являются:

- Гибкость и масштабируемость, каждый сервис может быть разработан и масштабирован независимо от других;
- Легче поддерживать приложение с ростом его функциональности и нагрузки;
- Легко заменять или обновлять сервисы без влияния на другие части приложения;
- Показывает себя лучше при работе с большими объемами данных.

Однако стоит учитывать, что микросервисный подход также имеет некоторые недостатки:

- Сложное развертывание и управление;
- Требует наличия сложной инфраструктуры для взаимодействия между сервисами;
- Может иметь большее количество точек отказа, так как каждый сервис может содержать собственные ошибки или проблемы;
- Может быть медленнее в работе, из-за расходов на взаимодействие между сервисами.

В рамках данного проекта плюсы микросервисной архитектуры не покрывают ее же минусов. Так как разработка ведется одним человеком паралельно разработать сервисы невозможно. Также очень важна простота развертывания и поддержки, которой микросервисы похвататься не могут.

Монолитная архитектура

Плюсы монолитного подхода:

- Простота разработки и управления;

- Эффективнее при обработке небольших объемов данных;
- Проще тестировать, поскольку все элементы находятся в одном месте;
- Меньшие накладные расходы на сетевое взаимодействие.

Минусы:

- Может стать менее эффективным и более сложным в управлении с ростом размера и функциональности приложения;
- Может быть менее гибким при обновлении и развертывании;
- Изменение одного элемента может повлиять на всё приложение в целом;
- Труднее масштабировать в случае роста нагрузки.

В моем проекте основную ценность представляет простота разработки и тестирования. Проблемы масштабируемости не так важны, ведь проект не предполагает значительного роста количества пользователей. Кроме того, было принято несколько решений о хранении данных, которые в случае необходимости облегчат процесс масштабирования.

Итоговая архтиектура

В результате сравнения я выбрал монолитный подход. Его минусы легко скомпенсировать формированием строгого API модулей и выбором рассмотрением альтернатив технологий на случай расширения (например, использование MongoDB вместо RocksDB). Кроме того, FastAPI позволяет разделять API на группы, каждая из которых представляет собой подприложеие. Это поможет перевести проект на микросервисы в случае необходимости.

1.3.6. Тестирование

Тестирование является важной частью разработки любого проекта, поэтому я уделил ему большое внимание. Для тестирования фукциональности сервера используется testsuite и unittest - инструменты для автоматического тестирования.

Testsuite[13] предоставляет ряд инструментов для организации тестов, таких как фикстуры (fixtures), которые позволяют подготовить данные для тестов и обеспечить их воспроизводимость, а также assert-методы, которые позволяют проверить ожидаемый результат работы теста. Инструмент также обладает возможностью группировки тестов в тест-сьютах и запуска их с помощью командной строки. Это позволяет облегчить процесс автоматического тестирования и ускорить процесс выявления и исправления ошибок. В проекте testsuite используется для тестирования общей функциональности сервера и отдельных сценариев его использования.

Unittest[14] позволяет организовывать тесты в виде классов и методов, а также использовать механизмы фикстур для создания начального состояния тестовой среды. Модуль предоставляет возможность использовать множество assert-методов для проверки значений и выполнения условий. Этот иструмент используется для тестирования классов или отдельных групп функций.

Нагрузочное тестирование является важной частью процесса разработки приложения и позволяет уставновить количество запросов и пользователей, которое приложение обрабатывает без проблем. Нагрузочное тестирование помогает выявить узкие места в приложении, которые могут привести к сбоям или недоступности системы, и позволяет принимать меры для устранения этих проблем. В данном проекте для нагрузочного тестирования использовался Locust.

Locust[15] - это инструмент для проведения нагрузочного тестирования веб-приложений. Он позволяет создавать сценарии, в которых множество пользователей симулируют работу с приложением, чтобы проверить его способность обрабатывать большое количество запросов и одновременных пользователей. Его я использую на последнем этапе разработки, чтобы убедиться что приложение может обрабатывать больше пользователей, чем ожидается при его использовании.

Выбор данных инструментов для тестирования показал свою эффективность и помог обеспечить высокое качество кода. Тестирование также помогло убедиться в надежности и функциональности сервера, а также быстро выявлять и исправлять ошибки.

Глава 2. Реализация серверной части приложения

2.1. Описание архитектуры приложения

2.1.1. Архитектура ASGI в FastAPI

ASGI(Asynchronous Server Gateway Interface)[16] - это спецификация, предназначенная для асинхронного взаимодействия между веб-серверами и веб-приложениями на Python. ASGI был создан как альтернатива WSGI (Web Server Gateway Interface).

ASGI работает на основе событийной модели. Каждый запрос обрабатывается как отдельное событие, которое передается от сервера к приложению. В свою очередь, приложение может асинхронно обрабатывать события, что означает, что оно не будет блокироваться во время обработки запроса и сможет обрабатывать другие запросы.

Обработка запроса на сервере соответствующем ASGI будет происхолить следующим образом:

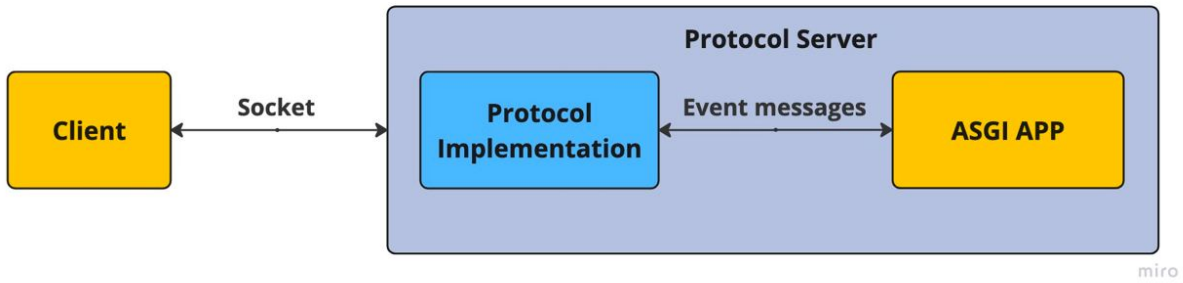

Рис. 2.1. Схема получения и обработки запроса

1. Сервер получает запрос от клиента.
2. Запрос передается в ASGI-приложение как событие. Событие содержит информацию о запросе, включая URL, метод запроса, заголовки и тело запроса.
3. Приложение обрабатывает событие асинхронно. Это может включать в себя выполнение долгосрочных операций, таких как запросы к базе данных или вызовы других API, без блокирования обработки других запросов.
4. После завершения обработки события, приложение возвращает ответ серверу. Ответ также представляет собой событие и содержит информацию, которую необходимо отправить клиенту, включая код состояния HTTP, заголовки и тело ответа.
5. Сервер отправляет ответ клиенту.

Так как вся эта процедура выполняется асинхронно, сервер не будет ожидать завершения обработки запроса перед тем, как начать обработку следующего. Это позволяет ASGI-серверам обрабатывать большее количество запросов за тот же период времени по сравнению с синхронными серверами.

ASGI - приложения представляют собой одну асинхронную функцию, которая принимает три аргумента: scope, receive и send.

- Scope содержит информацию о текущем соединении, включая тип протокола (HTTP, WebSocket и т. д.) и детали соединения, такие как IP-адрес клиента и URL-запрос.

- Receive и send - это функции, которые используются для получения событий от сервера и отправки событий обратно на сервер.

Такой интерфейс позволяет встраивать общие уровни обработки в сервере, таких как проверка запросов на наличие авторизационных данных или выставление таймаута обработки запросов. Также можно разбивать обработчики запросов на независимые группы, которые фактически представляют собой отдельное ASGI приложение. При таком подходе очень просто писать код, который будет хорошо масштабироваться и при необходимости может быть без особых усилий разделен на несколько сервисов.

Благодаря такому архитектурному решению получилось реализовать несколько групп обработчков, которые инкапсулируют работу с сущностями приложения. Более подробно об этом будет сказано в секции API. При незанчительном расширении функциональности обработчиков каждая группа может представлять собой отдельный сервис, но при текущих задачах проекта это не требовалось.

2.1.2. Модули приложения

В этой секции представленно общее описание модулей, их назначения и фукнциональности.

Обработка запросов

В общем представлении бработка запросов построена на взаимодействии модулей API, validation, schemas и token.

- API: Основной модуль, в нем находятся API-маршруты, которые позволяют взаимодействовать с сервером через определенные эндпоинты. Он реализует функциональность по обработке запросов, связанных с авторизацией и регистрацией пользователей, а также управлением группами, лабораторными работами, вариантами заданий и решениями студентов.

- Validation: Модуль отвечает за валидацию данных на сервере. Он содержит функции и правила для проверки наличия доступа пользователей к данным, а также

обеспечивает соблюдение определенных условий и правил при регистрации пользователей.

- Schemas: В этом модуле находятся файлы, определяющие схемы данных, используемые для автоматической валидации и описания структуры входящих и исходящих данных при работе с API сервера.

- Token: Данный модуль отвечает за работу с токенами аутентификации и авторизации. Он обеспечивают генерацию, проверку и управление токенами, которые используются для аутентификации пользователей и обеспечения безопасности запросов к API сервера.

Работа с хранилищами

В проекте есть два хранилища данных: PostgreSQL и RocksDB. Модули database, models и crud обеспечивают взаимодействие с SQL базой данных. Модуль storage относится только к хранилищу файлов.

- Database: Модуль обеспечивает установку соединения, создание сессий для взаимодействия с базой данных, создание схемы базы данных в случае необходимости.

- Models: В этом модуле находятся файлы, определяющие модели данных, которые используются на сервере. Они описывают структуру и связи между различными сущностями системы, такими как пользователи, преподаватели, студенты, группы, лабораторные работы, решения и другие.

- CRUD: В этом модуле реализованы операции CRUD (создание, чтение, обновление, удаление) для всех сущностей системы.

- Storage: Модуль содержит реализацию хранилища данных на основе RocksDB, обеспечивающего эффективное хранение и извлечение данных. Также в нем присутствует базовый класс для работы с файловым хранилищем данных, который поможет перевести проект на другое хранилище файлов в случае необходимости.

Генерация вариантов

Есть два модуля, использующихся для генерации вариантов: Generation и Tasks. Generation описывает инфраструктуру генерации, а tasks специфику генерации заданий для каждой лабораторной работы.

- Generation: Модуль реализует общую функциональность по генерации индивидульных заданий для студентов. Он содержит базовый класс генератора, описывающий строгий интерфейс и генерирующий пример лабораторной работы, и генераторы прочих лабораторных работ. Также в этом модуле представлены примеры генерируемых вспомогательных файлов, использующихся в создании вариантов.

- Tasks: В этом модуле содержатся функции для генерации описания индивидуальных заданий. Все реализации генерации отдельных заданий в лабораторных работах реализуются в отдельных файлах.

Вспомогательные модули

В этой секции описаны модули общего назначения, которые участвуют в конфигурированнии работы сервера.

- Main: Этот модуль содержит основной файл запуска сервера, он инициализирует необходимые компоненты.

- Config: В этом модуле содержатся файлы, отвечающие за конфигурацию сервера. Они определяют настройки, такие как параметры подключения к базе данных, порт сервера, настройки безопасности и другие параметры, необходимые для правильной работы сервера.

Общее описание взаимодействия модулей

Основные модули, описанные выше, реализуют функциональность по управлению лабораторными вариантами. Каждый модуль инкапсулирует работу с определенным видом данных, которые передаются между модулями.

На схеме ниже показано, какого рода информация передается при взаимодействии модулей.

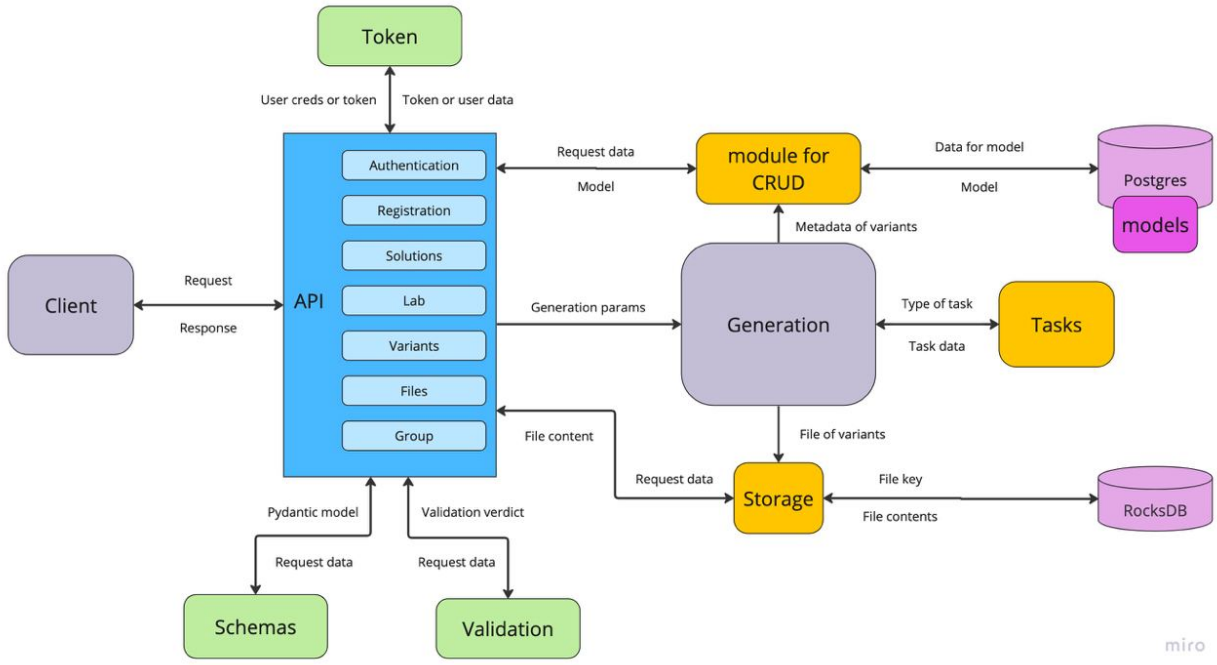

Рис. 2.2. Схема взаимодействия основных модулей

2.2. Описание архитектуры базы данных

2.2.1. Схема базы данных

База данных обеспечивает хранение информации в удобном для обработки формате. Схема проектировалась с учетом задач проекта для удобства объединения пользователей в группы, учета прав доступа пользователей к данным и сбора статистики по результатам работ.

Схема содержит несколько сущеностей, которые представляют информацию о пользователях, их лабораторных работах и решениях. Для каждой сущности приводится неформальное описание связей с другими сущностями и ее общее назначение.

- Пользователь:
  Пользователь в данном контексте представляет собой учетные данные для входа в систему, которые могут принадлежать как студенту, так и преподавателю. Эта сущность используется для управления доступом к системе и обеспечения безопасности данных.

- Студент:
  Студент представляет собой индивидуального ученика или участника курса. Он состоит в группе, и ему назначаются идивидуальные варианты заданий. Заданий может быть несколько по разным лабораторным работам. Сущность хранит личные данные пользователя и служит для проверки доступов к конкретным варинтам заданий и решениям.

- Преподаватель:
  Преподаватель - это пользователь, который ведет группы студентов. Преподаватели создают и назначают лабораторные работы, а также проверяют их. Каждый преподаватель может иметь несколько лабораторных работ, групп и вариантов лабораторных работ для проверки.

- Группа:
  Группа объединяет студентов и помогает организовывать учебный процесс. Группу ведет один преподаватель, и на группу раздаются варианты лабораторной работы.

- Лабораторная работа:
  Лабораторная работа представляет собой определенный набор заданий, которые студентам нужно выполнить в рамках курса. Каждая лабораторная работа имеет варианты, которые раздаются студентам. Сущность хранит информацию о лабораторной работе, включая ее описание и время начала и окончания, а также ответственного преподавателя и группу, которой она назначена.

- Вариант лабораторной работы:
  Вариант лабораторной работы - это конкретное задание, которое было назначено определенному студенту в рамках лабораторной работы. Каждый вариант уникален для студента и именно к нему будет привязанно решение и результат студента.

- Решение лабораторной работы:
  Решение лабораторной работы - это ответ студента на вариант лабораторной работы. Эта сущность предназначена для сбора информации о решении студента: самого решения, комментариев и результата. Студент может отправить несколько решений по одному варианту.

Далее рассматривается более формальное представление схемы в виде ER-диаграммы. На ней отображены отношения между сущностями и показаны поля с идентификаторами, при помощи которых эти связы реализованы.

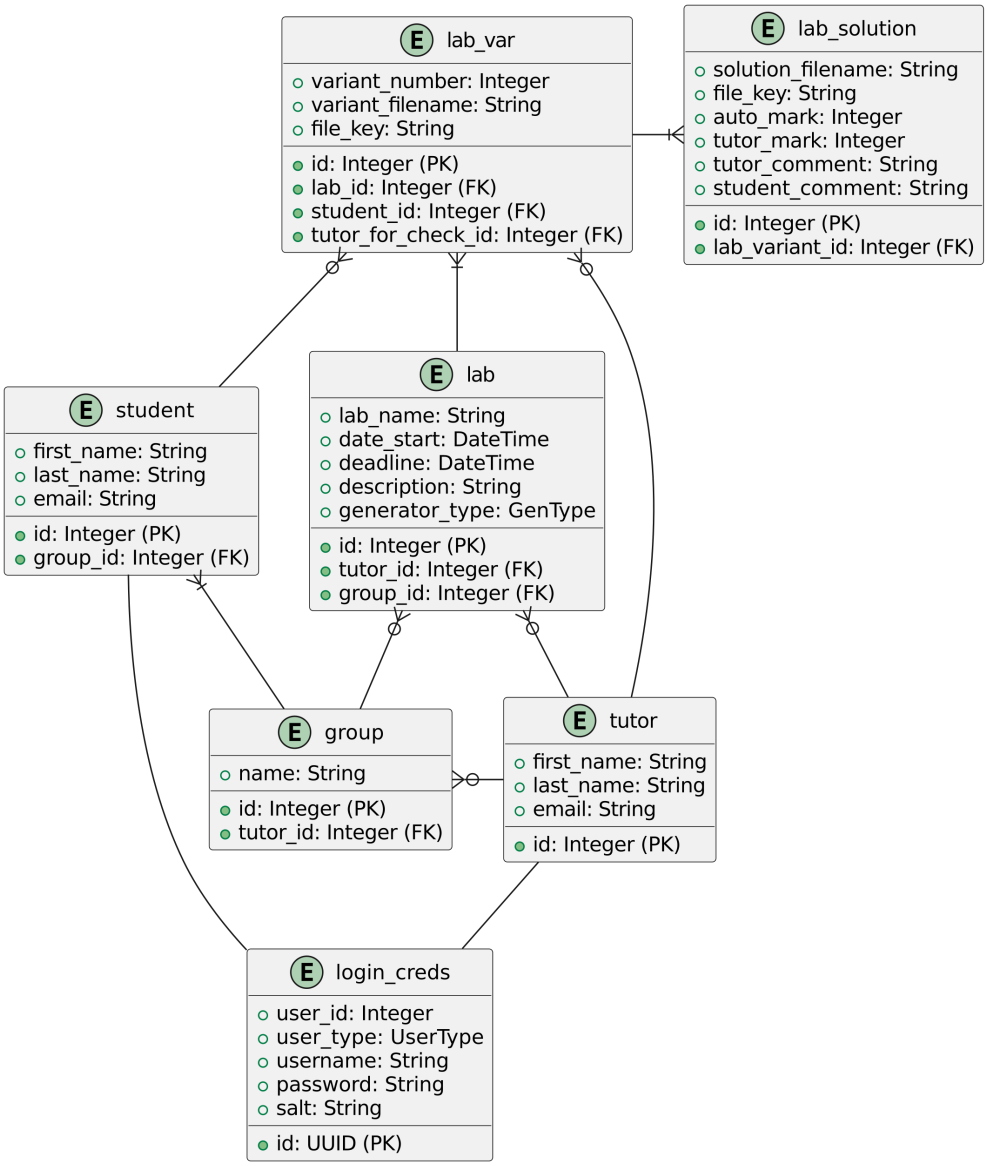

Рис. 2.3. ER диаграмма базы данных

Разработанная база данных следует принципам нормализации данных и соответствует третьей нормальной форме (3NF)[17], так как для проекта среднего размера это сотояние, при котором данные хорошо структурированы и на проектирование схемы не затраченно много ресурсов[18]. Основные критерии выполнены следующим образом:

- Первая нормальная форма (1NF): Каждое поле в таблице содержит только атомарные (неделимые) значения. Все записи уникально идентифицируются посредством первичного ключа.
- Вторая нормальная форма (2NF): Во всех таблицах не ключевые атрибуты полностью зависят от первичного ключа, и нет ситуаций, когда атрибут зависит только от части составного первичного ключа.
- Третья нормальная форма (3NF): Не существует транзитивных функциональных зависимостей.

В результате, схема базы данных обеспечивает надежное хранение данных, избегает избыточности и обеспечивает эффективность операций с данными.

Схема предусматривает расширение и улучшение системы и предполагает возможное добавление таблиц с информацией о решениях лабораторных работ. Например, вынос комментариев в отдельную таблицу и поддержку чата для комментирования работы, но в данном проекте не предполагалось детального комментирования.

2.2.2. SqlAlchemy и Postgres

Подключение и работа с базой данных в этом проекте реализованы с использованием библиотеки SQLAlchemy. Библиотека поддерживает диалект Postgres и обеспечивает удобный и понятный объектно-реляционный интерфейс для работы с данными.

Ключевым элементом при работе с SQLAlchemy является сессия. Сессия представляет собой временное "окно"для работы с базой данных, через которое можно выполнять операции CRUD. Все операции, проводимые в рамках одной сессии, можно рассматривать как единую транзакцию: либо все они успешно выполняются, либо, в случае возникновения ошибки, все изменения откатываются.

В проекте используется асинхронная версия сессии, предоставляемая SQLAlchemy. Это позволяет обрабатывать операции с базой данных без блокировки основного потока выполнения, что повышает количество запросов, которые сервис может обработать одновременно.

Подключение к базе данных происходит при старте сервера. Создаются нужные таблицы, если их не было ранее. Все секреты, необходимые для подключения к базе данных, сервер получает из переменных окружения предварительно выставленных в специальном файле.

После установки подключения мы переходим к обработке данных. Для каждой сущности в системе создан отдельный модуль, который содержит функции для управ-

ления ей, а также релизацию запросов фильтрации разного рода. Это обеспечивает четкую организацию кода и упрощает его поддержку.

2.2.3. Хранение файлов

Одним из важных компонентов системы является механизм хранения файлов. Для его реализации была выбрана RocksDB, база данных NoSQL типа. RocksDB отлично подходит для хранения больших объемов данных в виде пар ключ-значение и обеспечивает высокую производительность при выполнении операций чтения и записи.

Ддя доступа к хранилищу файлов используется шаблон проектирования Singleton. Он позволяет создать единственный экземпляр RocksDB во всем приложении, что обеспечивает централизованный доступ к хранилищу файлов и исключает возможность конфликтов при обращении к базе данных.

Класс RocksDBStorage реализует абстрактный класс FileStorage, который определяет интерфейс для всех операций с файлами. Это делает систему гибкой и расширяемой, позволяя легко добавить поддержку других NoSQL баз данных при необходимости.

Для этого достаточно создать новый класс, наследующий от FileStorage, и реализовать все необходимые методы, такие как save_file, get_file и delete_file, согласно требованиям выбранной базы данных. Таким образом, благодаря использованию этого подхода, система остается открытой для интеграции с другими технологиями хранения данных, такими как MongoDB. Это может потребоваться в случае развития проекта и развертывания нескольких экземпляров сервера.

2.3. Реагистрация и Авторизация

Система авторизации на этом сервере построена с использованием стандарта JSON Web Token (JWT). Основной подход заключается в том, что при успешной регистрации и авторизации пользователю выдается токен, который содержит информацию о его идентификаторе и типе пользователя (студент или преподаватель). Этот токен используется для аутентификации при дальнейших запросах.

2.3.1. Хранение авторизационных данных

При регистрации нового пользователя, пароль, введенный пользователем, не хранится в исходном виде. Вместо этого, применяется процедура, известная как "соление"и "хеширование"пароля. "Соль"в данном контексте - это случайная строка, которая добавляется к паролю пользователя перед хешированием. Это увеличивает безопасность хранения пароля, так как даже если два пользователя выберут одинаковые пароли, их хеши будут различны из-за уникальной "соли". Далее, соленый пароль подвергается хешированию с использованием алгоритма SHA-256, что преобразует его в уникальную строку. Этот хеш-код и сохраняется в базе данных.

Преимущество данного подхода заключается в том, что даже при утечке данных из базы, злоумышленники не смогут восстановить исходные пароли, так как процесс хеширования является односторонним. Кроме того, даже при совпадении паролей у разных пользователей, их хеш будет уникален благодаря использованию соли. Функции generate_salt и generate_salted_password в модуле utils отвечают за создание "соли"и хеширование паролей соответственно. Функция verify_password используется для проверки пароля, введенного пользователем, с хешированным паролем в базе данных.

2.3.2. JWT и Сессии пользователей

Для управления сессиями и аутентификации пользователей используются JWT-токены.

JWT (JSON Web Token) - это открытый стандарт (RFC 7519) для безопасного обмена информацией между двумя сторонами. Структурно JWT состоит из трех частей:

- Заголовок (Header): Заголовок состоит из двух частей: типа токена, который является JWT, и алгоритма подписи (HMAC SHA256 и другие).
- Полезная нагрузка (Payload): Payload - часть токена, которая содержит фактические данные, которые мы хотим передать между сервером и клиентом.

  Эти данные часто включают в себя утверждения о пользователе, которые могут быть использованы для идентификации и авторизации пользователя. Утверждения - это просто пары ключ-значение, которые содержат информацию о пользователе. Например, утверждение может быть 'user_id': 123, что означает, что пользователь, которому принадлежит этот токен, имеет ID пользователя 123.
- Подпись (Signature): Это защищенная цифровая подпись, которая гарантирует, что данные токена не были изменены в процессе передачи.

  Signature создается на основе заголовка (Header) и полезной нагрузки (Payload) JWT, а также секретного ключа, который хранится на сервере. Эти элементы объединяются и обрабатываются алгоритмом, указанным в заголовке.

  Когда сервер получает JWT, он может проверить подпись, используя тот же алгоритм и секретный ключ. Если данные токена были изменены, подпись не будет совпадать, и сервер будет знать, что токен недействителен.

JWT-токены генерируются при успешной авторизации пользователя и содержат идентификатор пользователя и его тип. Этот токен далее используется для идентификации пользователя в дальнейших запросах. Для предачи токена используется поле Authorization с токеном в заголовках запроса.

Токен проверяется при каждом запросе. С помощью функции decode_access_token в модуле token декодируется токен и проверяется его валидность.

Если токен истек или недействителен, то выдается исключение с соответствующим сообщением об ошибке.

Как описано ранее есть две роли - студент и преподаватель. Это различие обрабатывается в функции auth_by_token, где после успешной проверки токена идентифицируется тип пользователя и возвращается соответствующий объект пользователя.

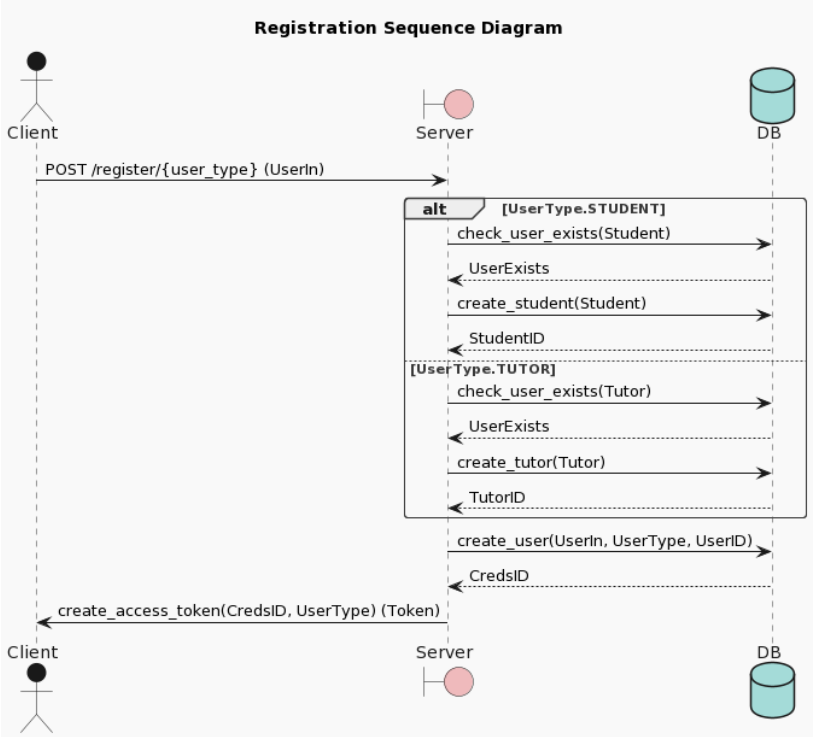

Рис. 2.4. Схема процесса регистрации

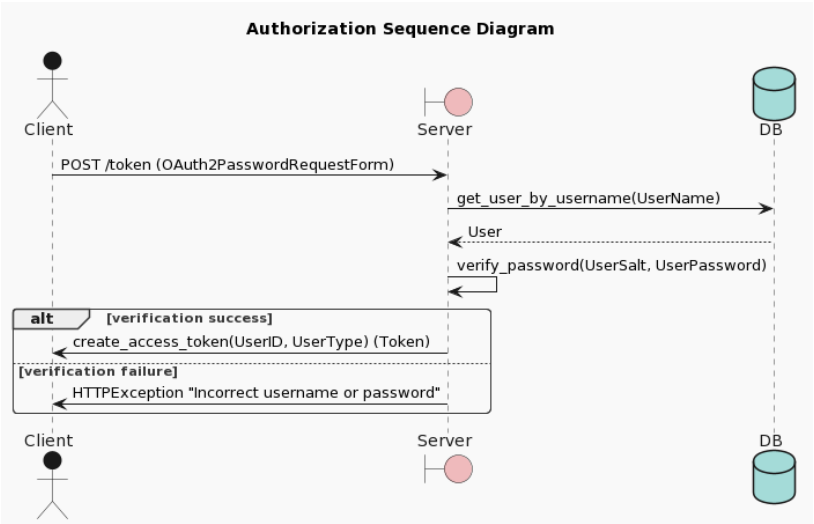

Рис. 2.5. Схема процесса авторизации

2.3.3. Регистрация

Процесс регистрации начинается с того, что клиент отправляет POST-запрос на сервер с указанием типа пользователя и информацией о пользователе. Как видно из

диаграммы 2.4, сервер сначала проверяет существование пользователя, затем создает новую запись в базе данных и, наконец, возвращает JWT-токен для аутентификации этого пользователя.

2.3.4. Авторизация

Авторизация - это процесс подтверждения личности пользователя. Он представлен на диаграмме 2.5. Клиент отправляет POST-запрос с учетными данными на сервер. Затем сервер обращается к базе данных, чтобы получить информацию о пользователе, и верифицирует предоставленные учетные данные. В случае успешной верификации сервер генерирует JWT-токен и отправляет его клиенту.

Далее во все защищенные запросы пользовтель должен передавать этот токен, по нему будет определяться личность и права доступа пользователя. Это позоволяет не присылать логин и пароль на каждое действие.

2.4. API

2.4.1. Система роутеров

Роутинг в контексте веб-приложения - это процесс сопоставления запросов к конкретным endpoint'ам (то есть URL'ам) с соответствующим кодом для их обработки. Роутеры - это модули или компоненты, которые содержат определения путей. Эти пути соответствуют различным URL-адресам, и каждый путь связан с функцией, которая будет выполняться при получении запроса на этот URL.

При поступлении HTTP-запроса FastAPI определяет, какой роутер и какой путь внутри этого роутера соответствует URL-адресу запроса. Затем он выполняет функцию или метод, связанный с этим путем, передавая ему любые параметры из URL, а также информацию из заголовков и тела HTTP-запроса.

Роутеры служат для обработки конкретных категорий запросов, сгруппированных по функциональности. Болле подробно о функциональности каждого из рутеров в следующих пунктах.

2.4.2. Регистрация и авторизация

Процессы регистрации и авторизации вынесены в разные модули для удобства масштабирования при необходимости.

- registration_router: Роутер отвечает за регистрацию новых пользователей в системе. Он принимает и обрабатывает данные нового пользователя для создания нового аккаунта.

- login_router: Роутер обрабатывает все операции, связанные с процессом аутентификации пользователей. Он позволяет пользователям входить в систему, используя свои учетные данные.

Более подробное описание функциональности представлено на диаграммах 2.4 и 2.5 соответственно в разделе авторизации.

2.4.3. Группы

Роутер управляет всеми операциями, связанными с группами пользователей. Он позволяет создавать, получать и обновлять информацию о группах пользователей в системе. Рассмотрим несколько основных запрсов, важных для работы с группами: создание и получение группы, добавление студента и обновление группы.

- Для создания группы (схема на рисунке 2.10) преподаватель задает имя. Только у него будут права на изменение этой группы.
- Для получения группы (схема на рисунке 2.11) пользователь предоставляет свой токен и id группы, которую хочет получить.
- Для обновления группы (схема на рисунке 2.12) также нужны токен и id группы, а также сами изменения. При обновлении группы проверяется, есть ли у пользовтеля права на ее изменение.
- Для обновления группы (схема на рисунке 2.13) также нужны токен и id студента. Каждый студент добавляется в группу сам или может быть добавлен преподавателем. Для этого указывается id студента, который будет добавляться в группу. Перед добавлением будет проверка на то, есть ли права у пользователя добавить этого студента.

2.4.4. Лабораторные работы

Роутер управляет всеми запросами, связанными с лабораторными работами. Он позволяет создавать, получать и обновлять информацию о лабораторных работах. Основные запросы представляют собой создание лабораторной работы, получение одной или списка работ.

- Для создания лабораторной работы (схема на рисунке 2.14) требуется токен и информация о работе: дедлайн, название, группа, на которую выдается работа и тип генерации.
- Получение всех существующих лабораторных, одной или лабораторных по перподавателю имееют похожу схему с получением. В этих случаях обращение к базе данных происходит исключительно в режиме чтения.
- Получение списка лабораторных для студента имеет болле сложную схему (схема на рисунке 2.15), так как сущность лабораторной работы не хранит в себе id пользователей, которым она назначена. Для этого просматриваются все варианты заданий студета, и по ним составляеся список лабораторных, в которых он участвует.

2.4.5. Варианты лабораторных работ

Роутер позволяет сгенерировать и раздать индивидуальные варинаты заданий на группы студентов. Кроме того, он отвечает за получение и изменение всех вариантов.

- Основным запрсом является генерация вариантов и их раздача на группу (схема на рисунке 2.16). Запрос разрешен только для преподавателей и не требует данных, кроме id лабораторной работы. Вся необходимая информация уже была задана при создании лабораторной работы. В обработке запроса запрашиваются все параметры генерации и участники. После проверки всех прав и существования сущностей происходит генерация самих вариантов и распределение их между студентами в группе. Запрос является очень легковесным и разрыв соеднинения может повлять только на запуск генерации. В случае неудачного запуска пользователь получит код ответа с ошибкой.

- Схемы запросов на получение и изменение данных о варианте строятся аналогично ранее описаным для других сущностей.

2.4.6. Ответы с решением студентов

Роутер позволяет сохранять, получать информацию о решениях студентов. Также с его помощью обрабатываются запросы по оценке и комментированию работы.

- Самым важным запросом является загрузка решения, его схема представлена на рисунке 2.17. Прежде чем сохранять файл с помощью токена, проверяется право доступа пользователя и валидируется расширение файла. Запрос осуществляется с помощью стриминга, в случае разрыва соединения файл не будет загружен на сервер и пользователь получит код ответа с ошибкой.

- Запросы на получение информации имеют схему, подобную 2.11, с учетом прав доступа. Важно заметить, что в результате этого запроса мы не получим файл, а только информацию о решении, такую как оценка, имя файла, ключ файла для скачивания и комментарии преподавателя и студента.

- Запрос на оценку работы по сути является обычным обновлением сущности и имеет схему, подобную 2.12.

2.4.7. Файлы

Роутер отвечает за скачивание файлов. Для загрузки был выделен отдельный роутер, так как файлы предпочтительнее скачивать асинхронно, не затрагивая при этом метаинформацию.

- Запрос на получение варинта представлен на схеме 2.18. Перед получением файла из базы данных проверяется наличие варината и прав доступа у пользовтеля к нему.

- Запрос на получение решения представлен на схеме 2.19. В этом случае перед получением файла проверяется существование решения с запрошенным ключом файла, после этого проверяется варинат задания и права доступа к этим данным.

Все запросы возвращают файл с помощью стриминга. Поэтому в случае разрыва соединения пользователь не получит файл полностью и может повтороить запрос.

2.5. Генерация вариантов лабораторных работ

2.5.1. Формат для генерации

Для удобства разработки генераторов использовались промежуточные файлы. Так как варианты должны быть в формате .ipynb, поиск ошибки в генерации может занять много времени, ведь при нарушении структуры файла его будет сложно прочесть человеку. С помощью промежуточных файлов можно избежать такой проблемы.

В таком случае генерация делится на 2 задачи: придумать структуру промежуточных файлов и уметь преобразовывать их в .ipynb и генерировать промежуточные файлы с заданиями.

Вспомогательные файлы

Пример стукруры такого файла хранится в репозитории проекта в модуле генерации. Его ситаксис достаточно прост и удобен для чтения. Это файл .py с особыми комментариями:

```
# BEGIN_CELL MD
"""
# Q1: Write a Python function
that returns the sum of two numbers
"""
# END_CELL

# BEGIN_CELL CODE
def add_numbers(a, b):
    return a + b
# END_CELL
```

Рис. 2.6. Синтаксис вспомогательного файла

Каждая клетка обозначается комментариями, и в начале указывается тип клетки. Такие файлы проще смотреть разработчикам и выявлять аномалии при генерации, а также понимать, в какой функции генерации может быть проблема. Файлы в таком формате обрабатываются и преобразовываются в формат .ipynb с помощью модуля nbformat.

2.5.2. Добавление новых генераторов

Для удобства добавления новых генераторов был разработан базовый класс генератора. В нем определен один метод, который должен возвращать лист готовых вариантов по заданному количеству. Чтобы добавить новый тип генерации, нужно создать новый класс, который будет наследовать базовый генератор и переопределить этот метод. При создании генератора задается префикс, который будет учитываться при генерации имен файлов.

Для нового генератора использование промежуточных файлов опционально, но для их использования определены вспомогательные функции преобразования.

Для более удобного доступа к генератрорам существует единственная функция, в которую передается информация для генерации. Реализация функции позволяет автоматически выбрать нужный генератор по типу и вызвать его метод генерации, а после сохранить файлы в хранилище. При добавлении генератора необходимо добавить новый тип генерации в перечисление типов, а также обновить функцию выбора генератора по типу добавлением нового условия на проверку типа.

2.5.3. Генерация лабораторной работы

Для генерации вариатов существует отдельный модуль Tasks. В нем определяются функции, с помощью которых генерируются отдельные условия заданий в лабораторной работе. В каждом файле этого модуля собранны функции по одной лабораторной работе. Необходимый базис знаний для генерации был взят из книги [19]. Для создания математиеских объектов используются пакеты Python, такие как sympy и numpy.

В каждой лабораторной работе у заданий своя специфика. Более подробно можно увидеть в коде проекта. Несколько примеров функций генерации задания:

- Система линейных уравнений с известным решением: для ее генерации создается вектор решений и случайная матрица, с помощью их умножения мы получаем вектор значений, который вместе с матрицей и составит систему.

- Эта функция генерирует СЛАУ с заданным числом переменных и уравнений. Сначала она создает символ для параметра. Затем создается матрица коэффициентов и вектор свободных членов и выбирается место для параметра.

```
def get_generator_by_type(type: GenType, prefix: str):
    if type == GenType.BASE:
        return NotebookGenerator(prefix=prefix)

    elif type == GenType.PRACTICE_1:
        return Practice_1(prefix=prefix)
                . . .

async def generate_for_group(
    lab: Lab, group: Group, students: List[Student]
) -> List[Variant]:
    prefix = f"L{lab.id}_GR{group.id}"

    generator = get_generator_by_type(
                    type=lab.generator_type,
                    prefix=prefix)

    variants = await generator.generate_notebooks(
                                        n=len(students))

    store_vars = await save_variants(variants=variants)

    return store_vars
```

Рис. 2.7. Функции генерции вариантов и выбора генератора

```
        def generate_known_solution():
            return np.random.randint(-10, 11, (5, 1))

        def generate_system(known_solution):
            A = np.random.randint(-10, 11, (3, 5))
            B = np.dot(A, known_solution)
            return A, B
```

Рис. 2.8. Генерация слау с известным решением

```
def generate_SLAE(n_variables=5,
                  n_equations=3, param_symbol="p"):
    param = sp.symbols(param_symbol)
    coef_matrix = np.random.randint(-10, 11,
        size=(n_equations, n_variables)).tolist()
    free_terms = np.random.randint(-10, 11,
        size=n_equations).tolist()

    rand_i = np.random.randint(n_equations)
    rand_j = np.random.randint(n_variables)
    coef_matrix[rand_i][rand_j] = param

    coef_matrix = sp.Matrix(coef_matrix)
    free_terms = sp.Matrix(free_terms)
    return coef_matrix, free_terms
```

Рис. 2.9. Генерация системы линейных уравнений с параметром

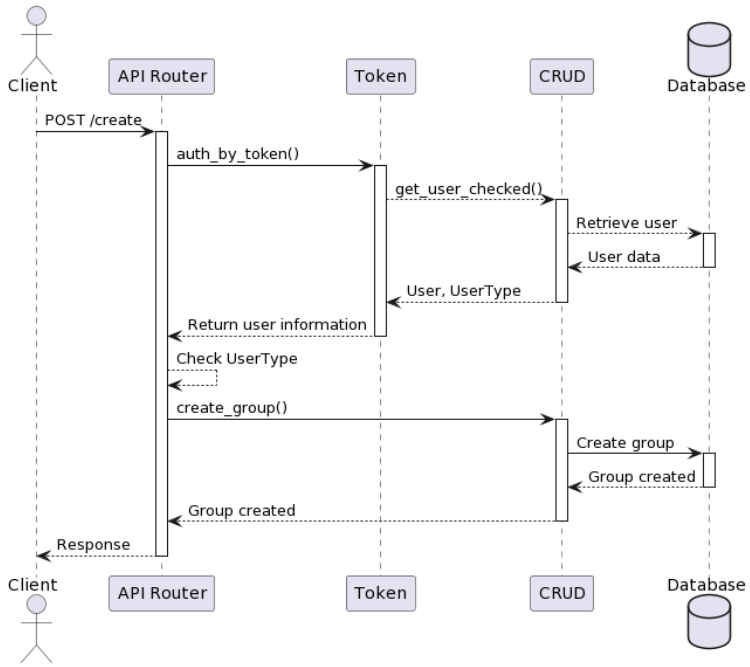

Рис. 2.10. Схема создания группы

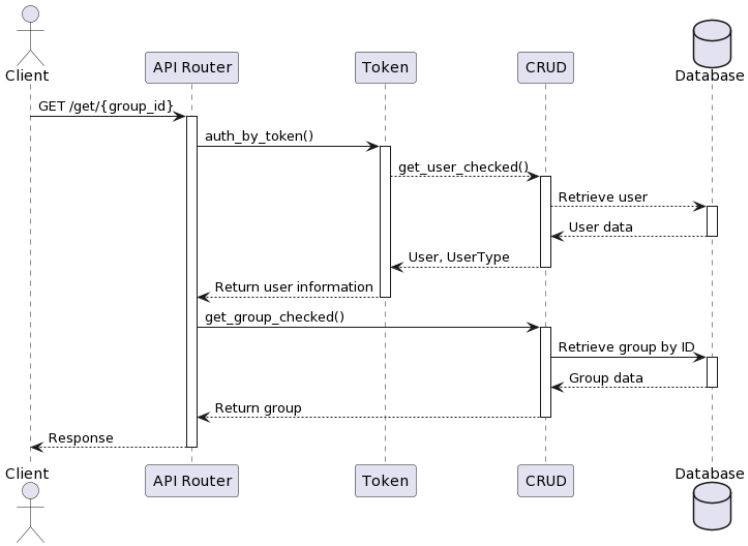

Рис. 2.11. Схема получения группы

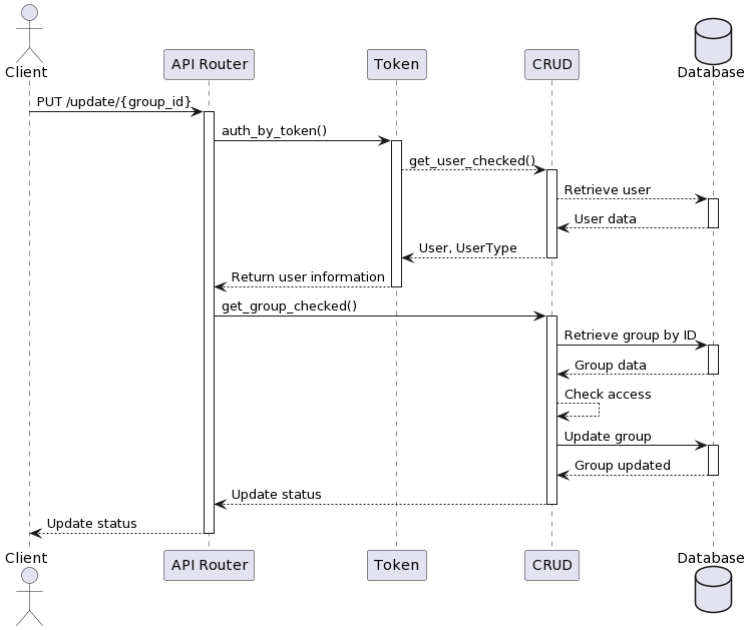

Рис. 2.12. Схема обновления группы

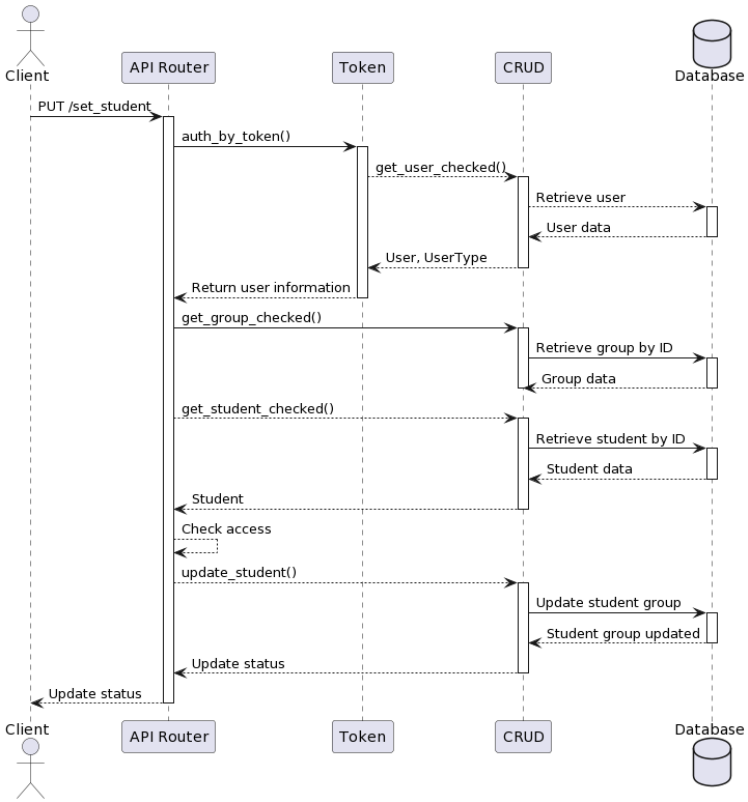

Рис. 2.13. Схема добавления студента в группу

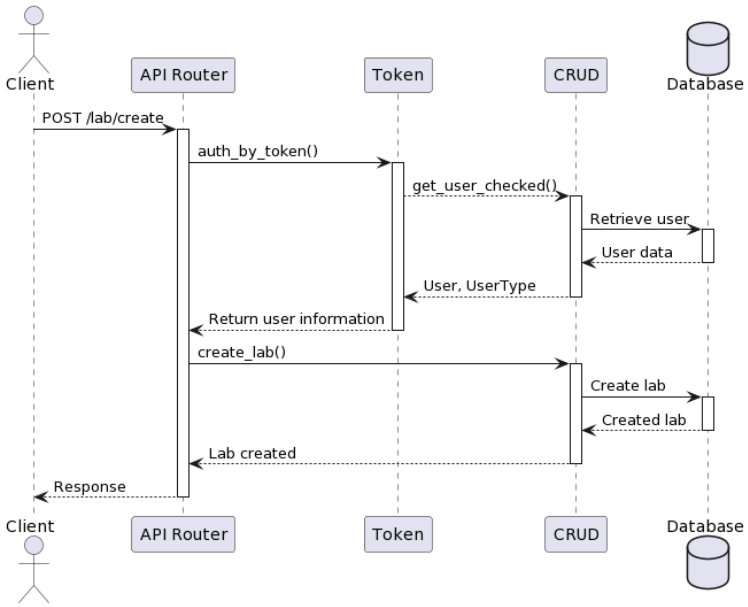

Рис. 2.14. Схема создания лабораторной работы

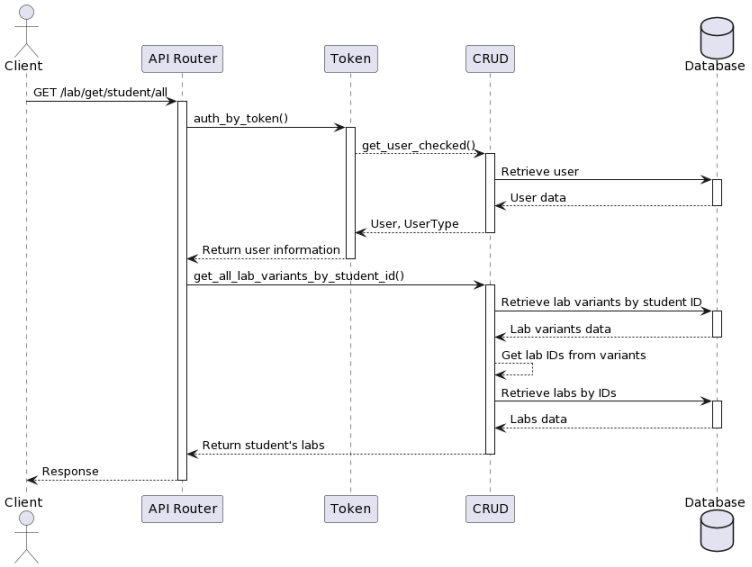

Рис. 2.15. Схема получения лабораторной работы

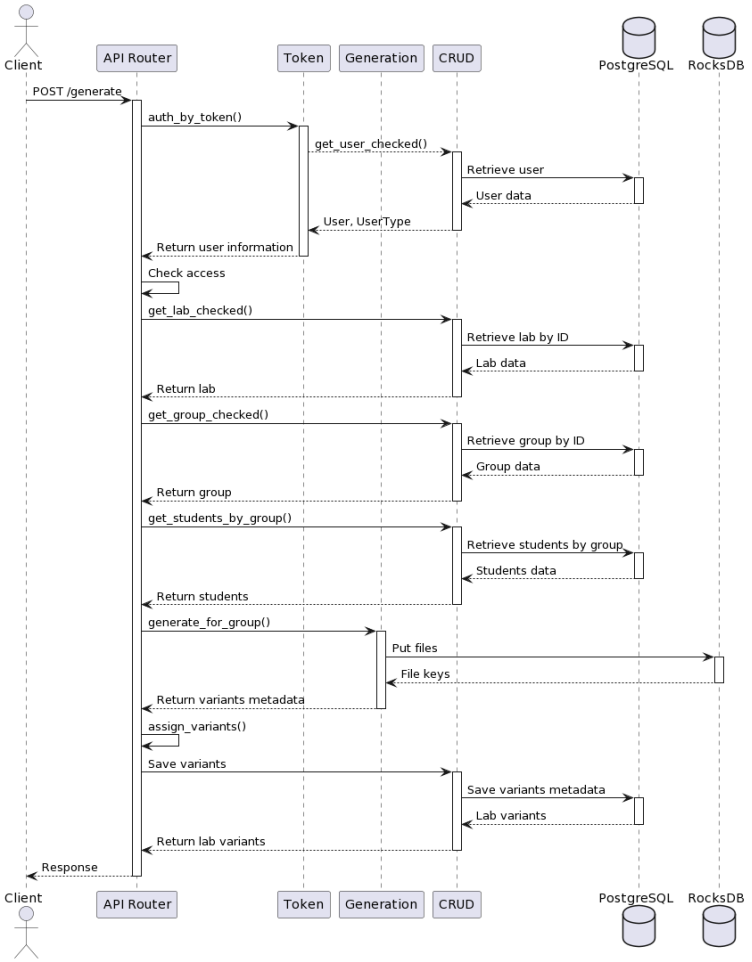

Рис. 2.16. Схема генерации вариантов лабораторной работы

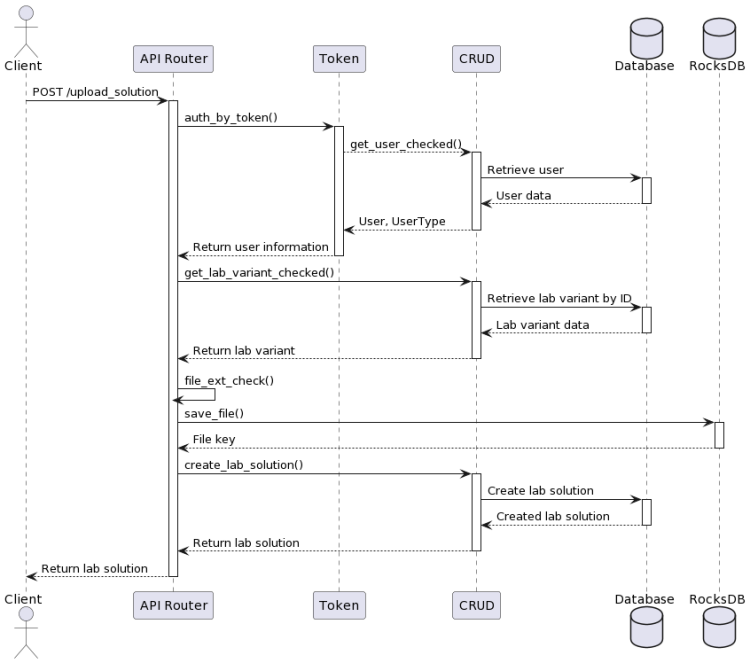

Рис. 2.17. Схема загружки решения студента

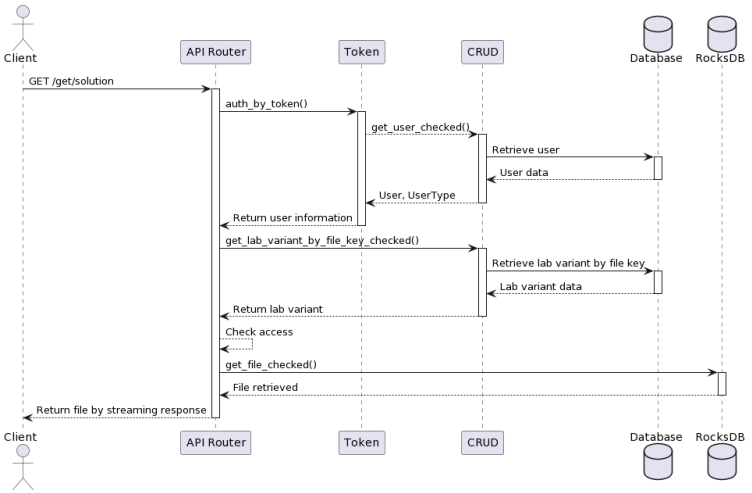

Рис. 2.18. Схема скачивания файла варианта

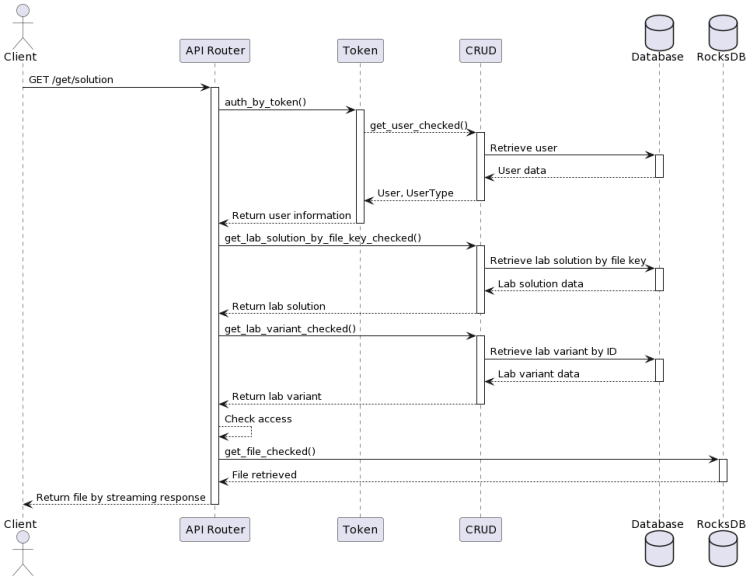

Рис. 2.19. Схема скачивания файла решения

Глава 3. Тестирование

3.1. Описание инфрастурктуры тестирования

В проекте используется pytest и unittest для тестирования функциональности сервера.

Реализовано интеграционное тестирование, которое проверяет функицональность обработки запросов. Для этого у проекта есть отдельная тестовая база данных. Также было реализовано юнит тестирование отдельных классов и функций сервера.

3.1.1. Инструменты для создания состояний сервера

Все тесты обработки запрсов сервера реализованы с помощью фикстур. Фикстура - это инструмент, который облегчает выполнение тестов. Она создает необходимые объекты или ресурсы, настраивает окружение и предоставляет доступ к этим объектам в тестовом коде.

- Для тестового взаимодействия с сервером используется асинхронный клиент AsyncClient, который позволяет отправлять HTTP-запросы к серверу. Фикстура client создает и настраивает этот клиент для каждого теста.
- Некоторые фикстуры также выполняют настройку базы данных, создание пользователей, групп и других сущностей. Например, фикстура test_student создает тестового студента и соответствующего пользовательского аккаунта в базе данных. Фикстура test_group создает тестовую группу студентов и связанных с ней студентов.
- Фикстуры также используются для реализации некоторых сценариев. Например, фикстура test_vars_with_group симулирует раздачу вариантов лабораторных работ для группы студентов без запросов на сервер.

Все фикстуры, реализованные для тестирования лежат в файле conftest.py.

3.1.2. Описание стурктуры тестов

Все тесты разделенны на несколько секций. Каждая секция соответствует роутеру, описаному ранее. Общий подход к реализаци заключен в том, чтобы проверить, что на корректных данных мы получаем правильное поведение. Помимо этого по каждому запросу есть ряд негативных тестов, которые проверяют запросы, в которых не хватает данных или они введены некорректно. Негативные тесты учитывают не только корректность данных, но и права доступа пользовтателей. Разработка тестов помогла выявить проблемы с ограничением доступов, а также корректность сохранения файлов вариантов.

Краткое описание каждой группы тестов по файлам:

- Файл test_login.py Содержит тесты, связанные с процессом входа в систему пользователей. В этих тестах проверяется, что пользователи могут успешно авторизоваться и получить доступ к системе. А также проверяется запрет на вход с некорректными учетными данными.

- Файл test_registration.py В тестах проверяется, что пользователи могут успешно зарегистрироваться в системе.

- Файл test_file.py Файл с тестами для проверки корректного скачивания файлов.

- Файл test_lab_variants.py Содержит тесты для проверки генерации вариантов лаборатоной работы. В них проверяется создание необходимых сущностей и файлов, а также корректность раздачи и выдачи прав доступа к работам.

- Файл test_lab.py Тесты в этом файле проверяют все операции, связанные с лабораторными работами: создание, обновление, передачу другому преподавателю.

- Файл test_solution.py Содержит тесты, связанные с обработкой решений лабораторных работ. В этих тестах проверяется сохранение, получение и оцека решений, предоставленных студентами.

- Файл test_group.py Содержит тесты, связанные с работой с группами студентов. В этих тестах проверяется создание, редактирование и управление группами студентов.

3.2. Нагрузочное тестирование

3.2.1. Подход к оргагнизации тестирования

Для проведения нагрузочного тестирования использовался инструмент locust. Locust это открытый инструмент для нагрузочного тестирования, который позволяет моделировать поведение пользователей и измерять реакцию сервера на множественные запросы.

Тестирование с locust представляет собой многократное повторнеие заданного пользовтельского сценария. Сценариев может быть несколько, и все они задаются кодом на Python. При запуске теста можно контролировать количество запросов на сервер, которые посылаются в одну секунду параллельно. Во время теста этот параметр постепенно увеличивается и позволяет проверить отзывчивость сервера при разных уровнях нагрузки.

Для тестирования сервера был реализован сценарий, в котором пользователь регистирировался и запрашивал токен авторизации с некоторой периодичностью. Этот сценарий помог выяснить, какое количество пользователей может выдерживать сервер в обычное время.

Также был реализован сценарий, при котором пользователь загружал файл на сервер. Организация сценария была не совсем точной, но помогла оценить нагрузку, которую сервер может выдержать в моменты дедлайна лабораторной работы.

Locust предоставляет достаточно подробный отчет о времени исполнения запросов, количестве ошибок и количестве запросов в секунду.

3.2.2. Результаты тестирования

Рисунок 3.1 на графиках видно, что сервер может выдерживать нагрузку больше той, на которую был расчитан при составлении требований, при условии, что тесты проводились на технически слабом оборудовании.

Рисунок 3.2 на графиках можно видеть, что сервер достаточно отзывчив даже при загрузке десятков работ в секунду. В рамках теста была отброшена авторизация, чтобы проверить,+ насколько сильно замедляет этот процесс обращение в базу данных, поэтому время запросов несколько отличается от предыдущего теста.

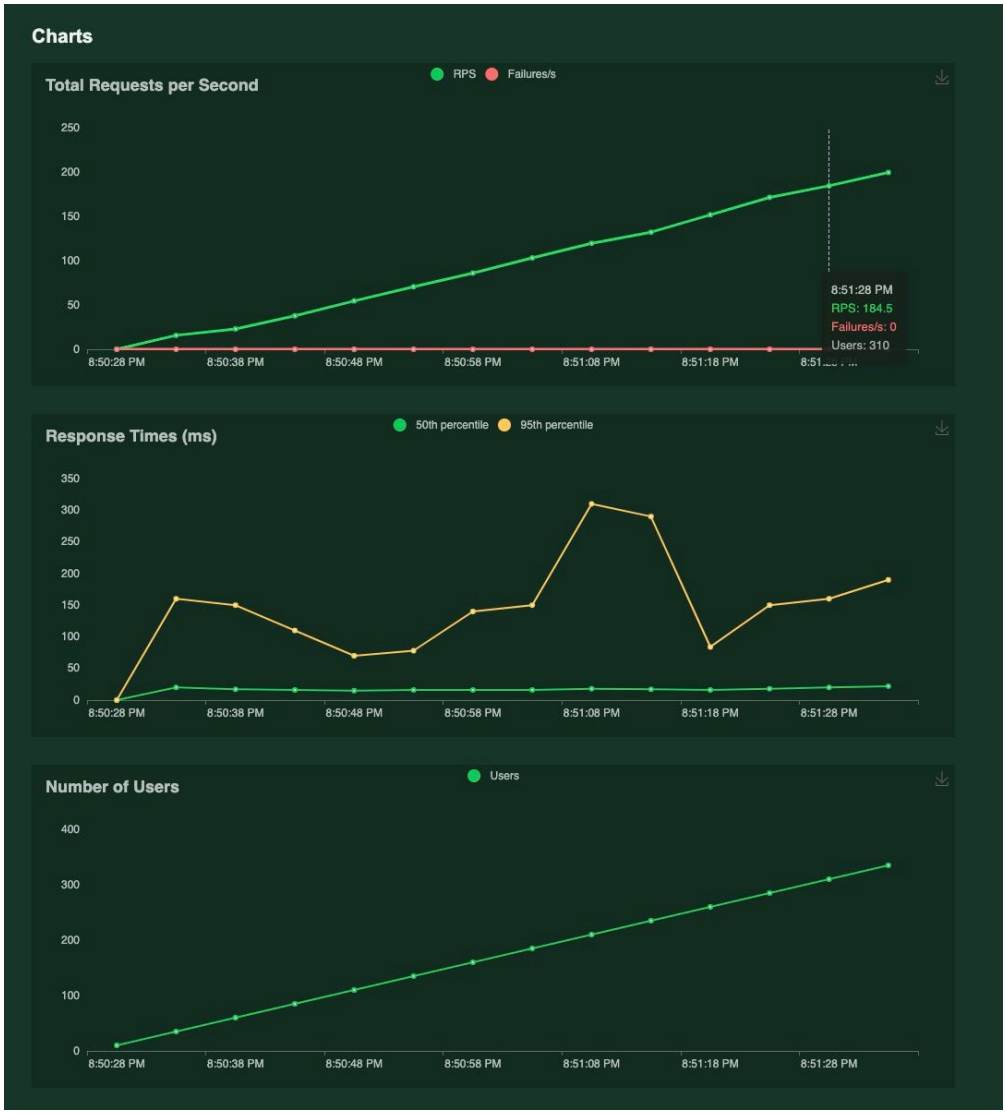

Рис. 3.1. Графики теста регистрации и авторизации

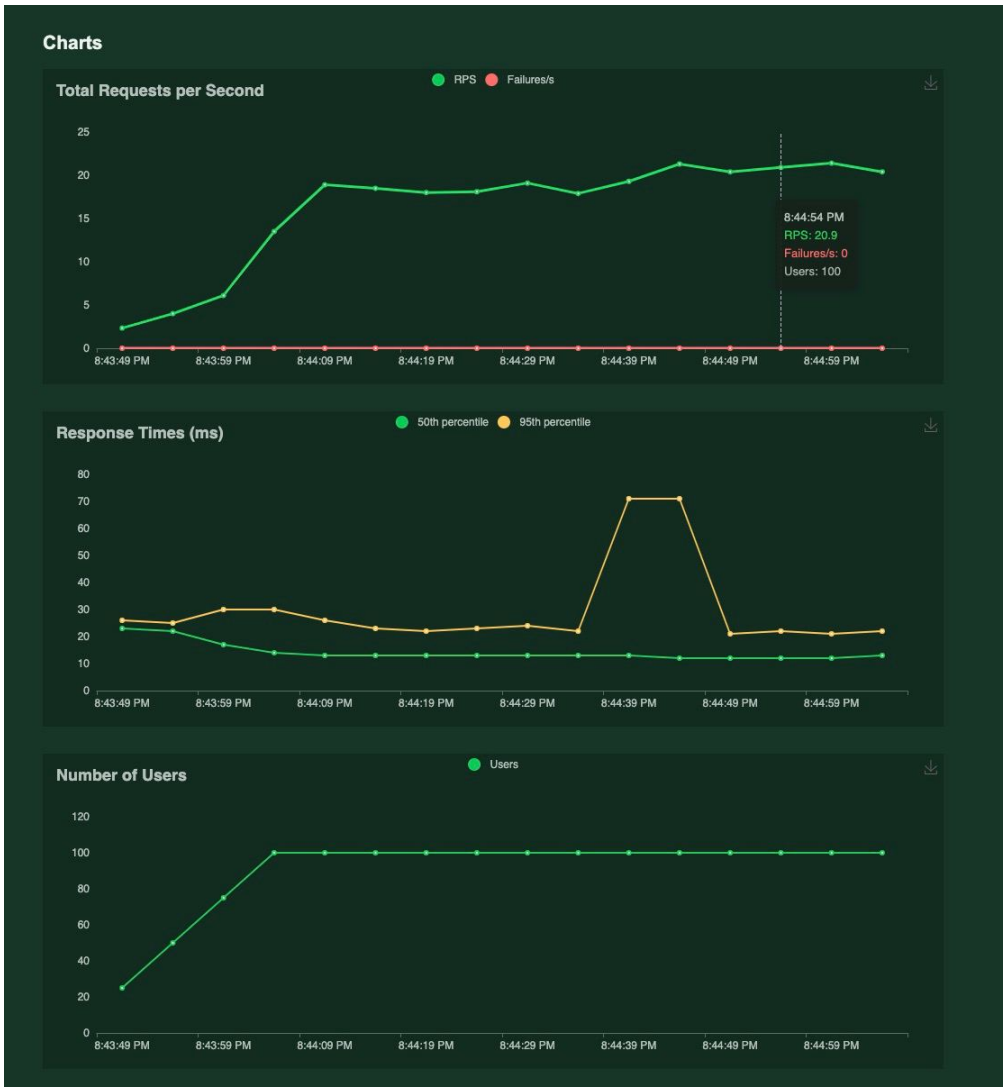

Рис. 3.2. Графики теста загрузки файлов

Библиографический список

1. Olga Shimko. Top 5 Languages to Server-Side Scripting in 2023. — 2023. — URL: https://www.qulix.com/about/blog/the-best-server-side-language/ (дата обр. 08.05.2023).
2. Pipenv: Python Dev Workflow for Humans. — 2023. — URL: https://docs.pipenv.org/ (дата обр. 08.05.2023).
3. Django documentation. — 2023. — URL: https://docs.djangoproject.com/en/4.1/ (дата обр. 08.05.2023).
4. FastAPI. — 2023. — URL: https://fastapi.tiangolo.com/ (дата обр. 08.05.2023).
5. Gaurav Sharma. Django vs Fast API: A Detailed Comparison. — 2022. — URL: https://medium.com/@ShortHills%5C_Tech/django-vs-fast-api-a-detailed-comparison-df8d00f3c3b2 (дата обр. 08.05.2023).
6. SQLAlchemy - The Database Toolkit for Python. — 2023. — URL: https://www.sqlalchemy.org/ (дата обр. 08.05.2023).
7. Security - First Steps - FastAPI. — 2023. — URL: https://fastapi.tiangolo.com/tutorial/security/first-steps/ (дата обр. 08.05.2023).
8. PostgreSQL: The world's most advanced open source database. — 2019. — URL: https://www.postgresql.org/ (дата обр. 08.05.2023).
9. Jon Xavier. The ultimate guide to strong passwords. — 2019. — URL: https://medium.com/fleetsmith/the-ultimate-guide-to-strong-passwords-in-2019-488bb8614c83 (дата обр. 08.05.2023).
10. RocksDB | A persistent key-value store | RocksDB. — 2023. — URL: https://rocksdb.org/ (дата обр. 08.05.2023).
11. MongoDB. — 2023. — URL: https://www.mongodb.com/ (дата обр. 08.05.2023).
12. Maryana Demchenko. Monolithic Architecture vs Microservice Architecture Compared | NCube. — 2020. — URL: https://ncube.com/blog/microservices-vs-monolithic-which-architecture-suits-best-for-your-project (дата обр. 08.05.2023).
13. TestSuite — удобный инструмент автоматического тестирования / Хабр. — 2020. — URL: https://habr.com/ru/companies/uipath/articles/531216/ (дата обр. 08.05.2023).
14. unittest — Unit testing framework. — 2023. — URL: https://docs.python.org/3/library/unittest.html (дата обр. 08.05.2023).
15. Locust Documentation — Locust 2.15.1 documentation. — 2023. — URL: https://docs.locust.io/en/stable/ (дата обр. 08.05.2023).

16. Введение в ASGI: становление асинхронной веб-экосистемы Python / Хабр. — 2023. — URL: https://habr.com/ru/articles/482936/ (дата обр. 08.05.2023).

17. Нормализация отношений. Шесть нормальных форм / Хабр. — 2015. — URL: https://habr.com/ru/articles/254773/ (дата обр. 08.05.2023).

18. Скромное руководство по схемам баз данных / Хабр. — 2020. — URL: https://habr.com/ru/companies/vk/articles/501598/ (дата обр. 08.05.2023).

19. Кострикин А. Введение в алгебру. — Физматлит, 1994.