Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Прикладная математика и информатика

КУРСОВАЯ РАБОТА

Программный проект на тему «Кулинарный сайт Cook4You с использованием GPT модели»

Выполнил студент:
Гуков Артём Евгеньевич, БПМИ2210, 3 курс

Руководитель КР:
Доцент, Базовая кафедра Тинькофф,
Иванов Андрей Александрович

Москва 2025

Содержание

1 Основные термины и определения

1 **Cook4You** - название разрабатываемого веб-приложения.

2 **Модель** - генеративная модель искуственного интеллекта для обработки запросов пользователей.

3 **Docker** - программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

4 **API**(Application Programming Interface) — это набор правил и протоколов, позволяющих приложениям взаимодействовать друг с другом. Иначе говоря, это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными.

5 **REST API**(Representational State Transfer Application Programming Interface) — это архитектурный стиль взаимодействия между клиентом и сервером через `HTTP`. Он определяет принципы построения `API`, обеспечивая стандартизированный и эффективный обмен данными между различными системами.

6 **OpenAPI** - это формализованная спецификация и экосистема инструментов, которая предоставляет интерфейс описания `HTTP API`.

7 **Swagger** - инструментальный набора для документирования и визуализации `REST APIs`. Часто используется для создания `OpenAPI`.

8 **CORS**(Cross-Origin Resource Sharing) — это механизм безопасности в веб-разработке, который позволяет или запрещает веб-браузерам делать запросы на серверы, находящиеся на другом домене. Проще говоря, `CORS` определяет, как браузер должен вести себя при отправке

запросов к ресурсам (например, вызовам `API`, изображениям, скриптам) на сервер из другого источника.

9 **MVP**(Minimum Viable Product) — это минимально жизнеспособный продукт. Это начальная версия продукта, которая имеет простейший функционал, но при этом может решать задачу или проблему клиента.

10 **RPS**(Requests Per Second) — это метрика, указывающая количество запросов, которые сервер или система может обрабатывать в секунду.

2 Введение

GitHub репозиторий проекта

В современном мире растёт интерес к персонализированным услугам, включая сферу питания. Люди всё чаще стремятся выбирать продукты и рецепты, которые соответствуют их индивидуальным предпочтениям, диетическим ограничениям и образу жизни. Это обусловлено как возросшей осведомлённостью о важности здорового питания, так и развитием технологий, которые позволяют анализировать данные и предоставлять персонализированные рекомендации. Развитие искусственного интеллекта и его применение в различных областях открывают новые возможности для улучшения качества жизни. Использование технологий машинного обучения в кулинарии позволяет не только автоматизировать процесс подбора рецептов, но и учитывать уникальные потребности каждого пользователя. Это делает тему разработки веб-приложения для генерации персонализированных рецептов актуальной и востребованной.

2.1 Цель

Цель работы: проанализировать целевую аудиторию и конкурентов, изучить новые технологий для создание собственного веб-приложения, способного генерировать рецепты на основе предпочтений пользователей, а также предоставлять рекомендации по ингредиентам и методам приготовления.d

2.2 Задачи

- Исследовательская часть
  - Выявить целевую аудиторию, провести опросы, выявить потребности пользователей

  - Исследовать рынок конкурентов, выявить их преимущества и недостатки

- Программная часть
  - Изучить `API` генеративных моделей
  - Создать схему данных. Выбрать сооветствующую базу данных
  - Реализовать подключение к `API` готовых моделей
  - Настроить работу моделей
  - Интегрировать базу данных
  - Добавить логирование и обработку ошибок
  - Протестировать получившийся функционал
  - Продумать реализацию веб-сайта. Изучить способы его создания
  - Создать `OpenAPI` для запросов, которые веб-сервер будет отправлять на сервер с моделью
  - Реализовать страницу регистрации и авторизации
  - На сервере добавить эндпоинты для регистрации и авторизации
  - Реализовать страницу с чатами пользователя
  - Реализовать переход на страницу с чатами после успешной регистрации или авторизации
  - На сервере добавить эндпоинты для получения информации о чатах
  - Протестировать получившийся функционал

2.3 Используемые технологии

- Языки программирования - `Python` и `JavaScript`

- Основной сервер асинхронный, использование библиотеки `asyncio` в `Python` и `async/await` в `JavaScript`
- `Python` библиотека `bcrypt` для хеширования паролей
- `Python` библиотеки `pytest` для общего тестирования, `pytest_asyncio` для асинхронных тестов, `testcontainers` - для тестирования базы данных
- `Git` в качестве системы контроля версий
- Взаимодействие с `API` генеративных моделей через `Python` билиотеку `g4f`
- Составление `PlantUML` схемы данных
- Запуск `MongoDB` в `docker`. Написание `docker-compose` файла для создания образа и запуска контейнера. Подключение к ней через `Python` библиотеку `motor`.
- Описание `OpenAPI` документации при помощи `Swagger`
- Использование `FastAPI` и `uvicorn` для реализации асинхронного `REST API` сервера
- Интеграция `JWT` токена для идентификации пользователя при помощи `Python` библиотеки `jose`
- Создания сайта и веб-сервера при помощи `React`, `Vite` и `Tailwind CSS`
- Настройка `CORS` для разрешения запросов с веб-сервера на основной сервер.

2.4 Актуальность работы

Практическая значимость этого проекта заключается в способности предоставлять пользователям рецепты, адаптированные под их индивидуальные потребности, что особенно актуально в условиях растущего интереса к персонализированному питанию. Веб-приложение может быть полезным как для индивидуальных пользователей, так и для компаний, занимающихся предоставлением услуг в области питания. Перспективы использования включают дальнейшее развитие функционала, интеграцию с другими сервисами и расширение базы данных для улучшения качества рекомендаций.

В качестве статистики приводятся данные с Яндекс Вордстат, а именно общее число запросов "приготовить" за 23.03.2025 – 23.04.2025:

Формулировка	Число запросов
приготовлять	9 056 664
что приготовить	2 379 880
что можно приготовить	861 665
как приготовить домашний	677 772
как приготовить условиях	598 787
как приготовить в домашних условиях	594 441
приготовляемых на сковороде	549 952
быстро приготавливаемые	477 844
что приготовить на ужин	378 052
что приготовить быстро	301 938
что приготовить вкусного	276 561

Рис. 2.1: Запросы в поисковике

Видно, что количество запросов достаточно велико, а значит у пользователей есть спрос на сервисы, которые быстро и легко смогут предложить блюда, рецепт и даже его финальное изображение!

3 Анализ целевой аудитории

При разработке проекта особое внимание было уделено изучению целевой аудитории. Для определения потребностей и интересов потенциальных пользователей был проведён опрос, результаты которого легли в основу анализа. Исследование позволило выявить основные характеристики аудитории, её мотивацию к использованию сервиса, а также ожидания от функциональности и качества взаимодействия с Cook4You. Респондентам было задано 7 вопросов:

1 Возраст

2 Пол

3 Частота готовки

4 Способы поиска рецептов

5 Трудности при поиске рецептов

6 Какие возможности стали бы для них преимуществом при выборе сервиса

7 Желание пользоваться Cook4You, которое реализует эти преимущества

3.1 Возраст

Формируя портрет целевой аудитории на основе данных, полученных в результате опроса, выявлено преобладание молодого поколения от 18 до 35 лет (70.5%). Это - основные пользователи интернета и различных устройств, а значит - релевантная целевая аудитория.

Рис. 3.1: Возраст респондентов

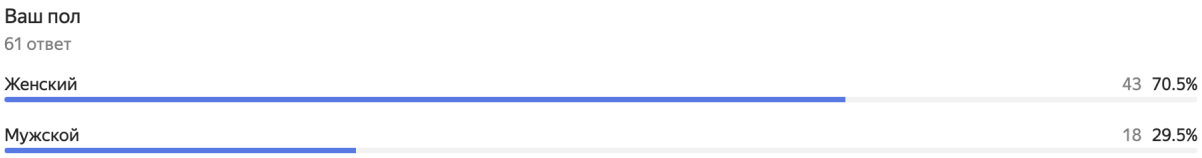

Рис. 3.2: Пол респондентов

3.2 Пол

Преобладает женский пол (70.5%), что соответствует текущим тенденциям.

3.3 Частота готовки

Рис. 3.3: Насколько часто респонденты готовят

Большинство пользователей (80.3%) регулярно готовят дома — хотя бы раз в неделю. Это значит, что сервис будет релевантен для широкой аудитории. Люди действительно нуждаются в помощи с рецептами, подсказками

по готовке и идеями для блюд.

11.5% готовят реже - несколько раз в месяц или реже. Для них сервис тоже может быть полезен, особенно если он будет помогать быстро находить простые рецепты для редких случаев готовки.

8.2% вообще не готовят - это небольшая группа. Скорее всего, их сервис заинтересует меньше. Но если придумать какие-то простые функции, например “что приготовить за 10 минут” или “готовка для тех, кто не любит готовить”, можно привлечь и их.

3.4 Способы поиска рецептов

Рис. 3.4: Как респонденты ищут рецепты

Самый популярный способ — готовить по памяти (63.9%). Большая часть людей часто полагается на уже знакомые рецепты. Это значит, что Cook4You может быть особенно полезен в ситуациях, когда “надоело готовить одно и то же” и хочется чего-то нового.

Браузер (54%) — главный инструмент для поиска новых рецептов, поэтому нужно сделать таргетированную рекламу в браузерах.

Социальные сети и советы других людей (оба по 31.1%). Значит, нужно запустить рекламу в социальных сетях и настроить возможность рекомендации другу.

Кулинарные сайты (24.5%) и YouTube (14.7%) — отдельные источники поиска. Первые - главные конкуренты Cook4You, YouTube же - координально другой подход к готовке. Генерировать видео очень сложно, поэтому заменить YouTube не получится.

Кулинарные приложения (3.2%) и книги (1.6%) почти не используются, значит выбор в сторону веб-приложения был правильным.

3.5 Сложности с поиском рецептов

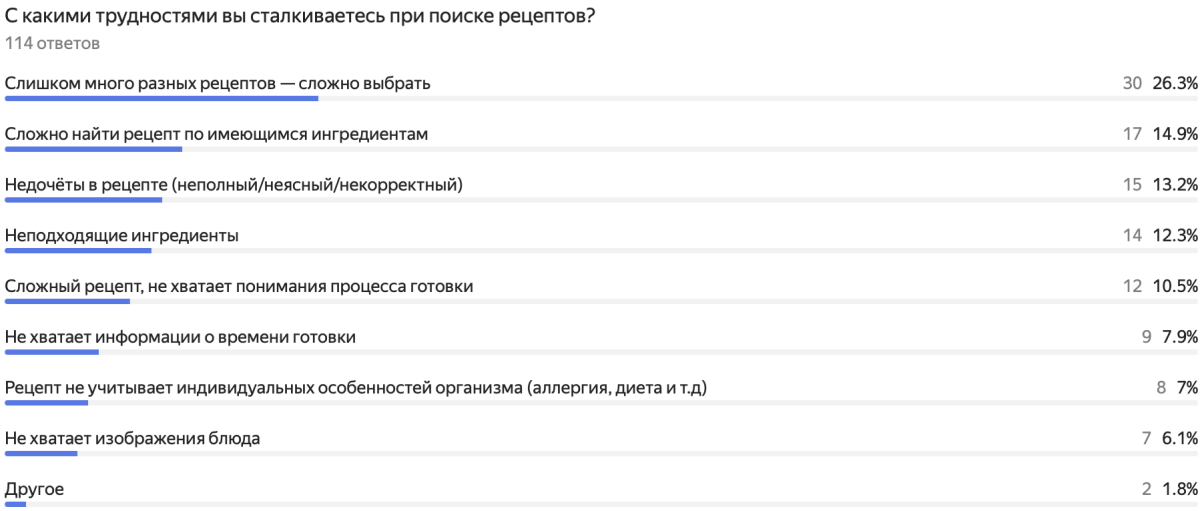

Рис. 3.5: Основные сложности респондентов

Все эти сложности будут решены с помощью генеративной модели, поэтому нужно было понять, насколько они часто встречаются у пользователей.

По материалам опроса, 49.1% отметили сложность выбора из большого количества рецептов, 27.8% трудно находить рецепт по имеющимся ингредиентам, 19.6 – 24.5% имеют трудности с самими рецептами (недочёты, неподходящие ингредиенты, сложность рецепта).

Cook4You сможет генерировать простые и понятные рецепты, исходя из имеющихся ингредиентов, предпочтений и ограничений.

3.6 Возможности Cook4You

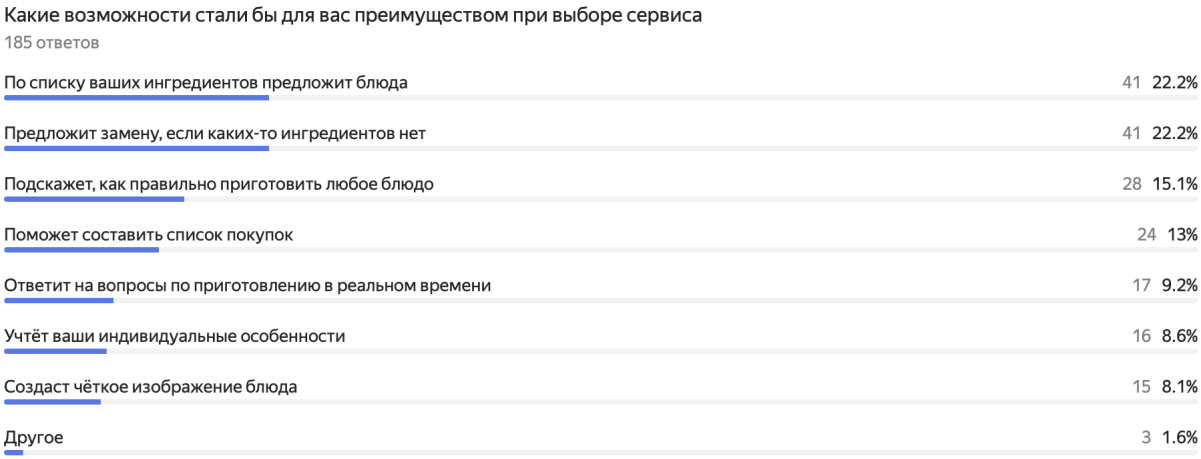

Рис. 3.6: Выбор возможностей Cook4You среди респондентов

Повторяя предыдущий комментарий, здесь ещё более отчётливо видно, что возможности Cook4You очень актуальны для решения поставленных задач потенциальных пользователей.

3.7 Итоговое решение

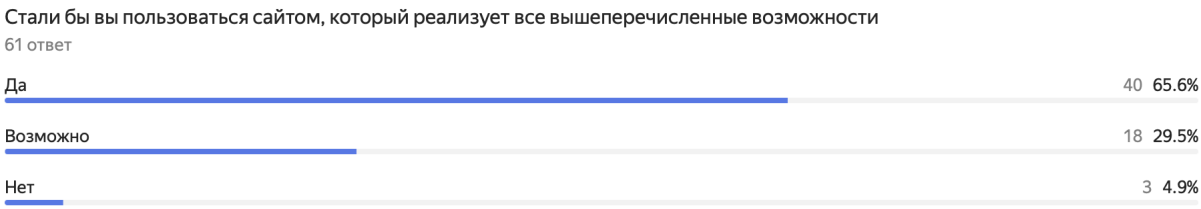

Рис. 3.7: Решение респондентов касательно использования Cook4You

По результатам опроса, 65.6% указали, что готовы попробовать Cook4You и 29.5% возможно это сделают. Это означает, что 95.1% выборки интересна эта тематика, поэтому разработка данного сервиса имеет смысл!

4 Анализ конкурентов

Вторым важным шагом стал анализ уже существующих аналогов. Так как пользователи проявили интерес к возможностям Cook4You (согласно результатам опроса), теперь нужно изучить уже существующие сервисы, которые предлагают такие же услуги. Для этого было задано 2 вопроса в браузере: "помощник для приготовления блюд и составления рецептов"и "кулинарный помощник по генерации рецептов". В итоге в верху выдачи были 2 сайта:

- Лучшие нейросети для генерации рецептов в 2025 году
- 6 нейросетей для тех, кому надоело думать, что приготовить

4.1 AI Recipe Generator

Ссылка на сервис

1 Главный недостаток этого сервиса - англоязычная аудитория. Cook4You изначально будет расчитан на русский сегмент пользователей, а вот AI Recipe Generator плохо поддерживает русский язык

2 Не умеет генерировать изображения

3 В "free" плане доступно 30 рецептов, потом нужно обязательно оплатить подписку

4 Есть 2 подписки - 30 рецептов на месяц за 3$, 100 рецептов на месяц за 6$

5 Оплата недоступна в РФ, поэтому подписку оформить не получится

4.2 Mealpractice

Ссылка на сервис

1 Заблокирован на территории РФ, можно зайти только с ВПН

2 Можно генерировать рецепт только по фиксированным ингредиентам и фильтрам, которых ограниченное количество. Нельзя задать свой вопрос

3 Все блюда на английском, сервис ориентирован на зарубежную аудиторию

4 Дорогостоящая подписка: 8$ = 33 рубля за рецепт или 25$ = 21 рубль за рецепт

4.3 RecipeGPT

Ссылка на сервис

1 Можно генерировать рецепт только по фиксированным ингредиентам и фильтрам, которых ограниченное количество. Нельзя задать свой вопрос

2 Все блюда на английском, сервис ориентирован на зарубежную аудиторию

3 Долгое ожидание на один запрос. Ждал больше 10 минут, ответ не получил

4.4 Mealmind

Ссылка на сервис

1 Сервис не генерирует рецепты, а составляет индивидуальные планы питания, поэтому это другая целевая аудитория

4.5 ChefGPT

Ссылка на сервис

1 Бесплатно на месяц даётся 10 генераций, неограниченный доступ стоит 2.99$

2 Генерирует рецепты по любым ингредиентам, много доступных фильтров (время готовки, сложность, количество человек, индивидуальные особенности)

3 Предоставляет качественное изображение и подробный рецепт

4 Реализована система постов, где можно публиковить свои рецепты. Пользователи могут их открыть, прокомментировать и поставить лайк

5 Сервис поддерживает множество языков, в том числе русский

4.6 Второй сайт

Лучшие нейросети для генерации рецептов в 2025 году

На этом сайте представлены общие нейросети, не заточенные именно под готовку. У них также есть ограниченный бесплатный доступ, нет возможности сгенерировать изображение блюда, и они не обучены именно под готовку.

4.7 Вывод

Единственный хороший сервис - ChefGPT. Однако по опросу можно сделать вывод, что он не проводит активную рекламу на русском сегменте пользователей, потому что мало людей пользуются чем-то подобным. Также очень схожий результат могут выдать обычные ИИ модели, если им

правильно задать промпты. Поэтому задача Cook4You - грамотно настроить модели именно под готовку и проводить рекламу среди русскоязычных пользователей

5 Бэкенд

Весь проект можно разделить на 3 отдельные задачи - исследование аудитории и конкурентов, разработка бэкенда и разработка фронтенда. После успешного анализа, автор начал разрабатывать бэкенд Cook4You.

5.1 API генеративных моделей

Было принято решение сначала научиться взаимодействовать с готовыми ИИ решениями и изучить их API. Автор остановился на библиотеке `g4f`, которая позволяет реализовать MVP и бесплатно слать запросы в провайдеры языковых моделей. Вполне возможно, что при большое RPS, библиотека не выдержит нагрузки или ограничит доступ, но для первоначального варианта она подходит. В случае, если сервис будет набирать популярность, можно будет масштабироваться и подключиться к API OpenAI, Google и т.д. Пока что в проекте реализован только способ обращения к моделям через `g4f`.

5.2 Схема данных и база данных

Изучив принцип работы библиотеки `g4f`, нужно было придумать, как хранить данные пользователей. Так как модели работают с контекстом запросов, нужно для каждого пользователя хранить историю его запросов в различных чатах (по аналогии с ChatGPT). Было принято решение сделать 2 сущности - пользователи и чаты.

- Пользователь будет проходить процедуру регистрации, поэтому надо хранить его email и пароль в хешированном виде. У каждого пользователя будет какое-то количество чатов.
- Для каждого чата надо будет хранить его название, пользователя, которому он принадлежит и контект запросов (как текст, так и изоб-

ражения)

- Дополнительно для статистики надо хранить время создания пользователя и времена создания и изменения чата
- Связь между пользователями и чатами - one to many

Итоговым выбором базы данных стала `MongoDB`. `MongoDB` - одна из самых популярных `NoSQL` баз данных. Так как автор с ней до этого не работал, будет полезно изучить её устройство на примере этого проекта. Также описанная выше модель данных хорошо впишется в модель хранения `MongoDB` Получившаяся схема данных выглядит так:

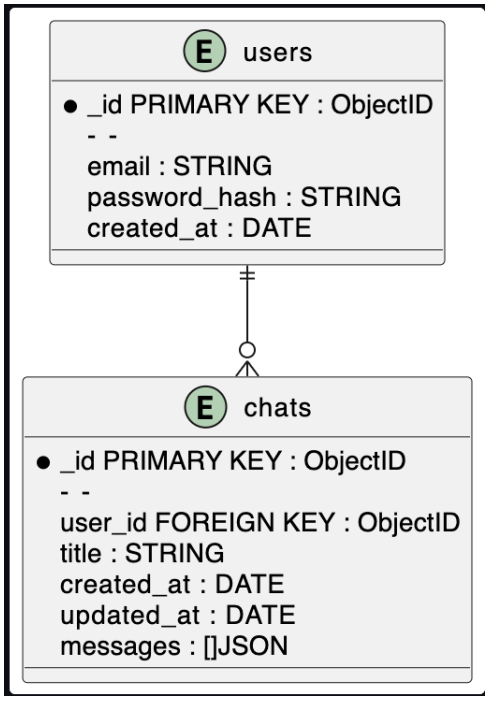

Рис. 5.1: Схема данных

5.3 Подключение к API моделей

Для отправки запросов генеративным моделям, автор создал класс `Model`. Вот так выглядит инициализация объекта этого класса: 5.1

```
def __init__(self, chat_model: str, image_model: str,
image_generation_prompt: str, logger):
```

```
        self.client = AsyncClient()
        self.image_generation_prompt = image_generation_prompt
        self.chat_model = chat_model
        self.image_model = image_model
        self.logger = logger
```

Листинг 5.1: Инициализация объекта класса Model

- chat_model - название модели для обработки текстовых запросов
- image_model - название модели для генерации изображений
- image_generation_prompt - промпт для генерации изображений
- logger - инструмент логгирования

У данного класса есть следующие методы: 5.2

```
async def response(self, history: list[dict]) -> str:
async def create_image(self, prompt: str) -> str:
async def make_prompt_for_image_model(self, history: list[dict]) -> str:
async def image(self, history: list[dict]) -> str:
```

Листинг 5.2: Методы класса Model

- response - послать запрос текстовой модели.
- create_image - послать запрос генератору изображений.
- make_prompt_for_image_model - сгенерировать промпт для генератора изображений по контексту дилога. Запрос отправляется текстовой модели, которая выдаёт итоговый промпт.
- image - функция, которая вызывает предыдущие две и возвращает `url` на итоговое изображение.

5.4 Настройка моделей

Перед запросами важно задать начальный промпт, чтобы модель отвечала только на кулинарные вопросы, делала это как можно точнее и генерировала качественные изображения.

- При создании нового чата, в контекст пользователя добавляется начальный промпт: *«Ты будешь в роли эксперта по готовке. Пользователь будет задавать вопросы касательно приготовления блюд (рецепт блюда, как его готовить, насколько сложно и сколько займёт времени, различные нюансы и уточнения), ты будешь на эти вопросы отвечать. Тебе нельзя разговаривать на другие темы! Если пользователь начнёт задавать вопросы не про готовку, ответь, что не можешь на них отвечать!»*.

- Для генерации изображений применяется следующий промпт: *«Исходя из моих запросов, сформулируй на английском языке полное описание блюда, которое я хочу сейчас приготовить, учитывая тот рецепт, который ты мне написал. Я его отправлю генератору изображений. Очень важно, чтобы описание соответствовало твоему рецепту вплоть до мелочей, иначе изображение не будет описывать ожидаемое блюдо! Не нужно никаких лишний слов и комментариев, только промпт для генератора изображений»*.

Названия моделей, промпты и другие переменные, которые будут встречаться дальше в отчёте, находятся в `.env` файле. Любой, кто захочет изменить настройки проекта, сможет добавить свои значения в `.env` файл.

5.5 Интеграция базы данных

В качестве базы данных была выбрана `MongoDB`. Для ассинхронного взаимодействия с ней в `Python` есть библиотека `motor`. Реализован класс

MongoDB для взаимодействия с базой данных. Вот так выглядит инициализация объекта этого класса: 5.3

```
def __init__(self, username: str, password: str, host: str,
port: str, logger):
        self.username = username
        self.password = password
        self.host = host
        self.port = port
        self.logger = logger

        self.client = AsyncIOMotorClient(
            f"mongodb://{username}:{password}@{host}:{port}")

        self.db = self.client["Cook4You"]
        self.users = self.db["users"]
        self.chats = self.db["chats"]
```

Листинг 5.3: Инициализация объекта класса MongoDB

- username, password - логин и пароль для подключения к базе данных
- host, port - хост и порт базы данных
- client - клиент, через который инициализируется подключение к базе данных
- db - база данных Cook4You, users и chats - две сущности, описанные в схеме данных
- logger - инструмент логгирования

У данного класса есть следующие методы: 5.4

```
async def create_indexes(self):
async def user_id_exists(self, user_id: str) -> bool:
async def chat_exists_for_user(self, chat_id: str, user_id: str) -> bool:
async def create_user(self, email: str, password_hash: str) -> str:
async def create_chat(self, user_id: str, title: str) -> str:
async def add_message(self, chat_id: str, role: str,
```

```
type: str, content: str) -> bool:
async def get_chats_for_user(self, user_id: str):
async def get_chat_messages(self, chat_id: str):
async def delete_chat_for_user(self, chat_id: str, user_id: str) -> bool:
async def delete_all_chats_for_user(self, user_id: str) -> int:
async def get_user_id(self, email: str, password: str) -> str:
async def user_exists(self, email: str) -> bool:
```

Листинг 5.4: Методы класса MongoDB

- create_indexes - создание индексов для ускорения поиска данных.
- user_id_exists - проверка наличия user_id в базе.
- chat_exists_for_user - проверка наличия чата у определённого пользователя.
- create_user - создание нового пользователя.
- create_chat - создание нового чата.
- add_message - добавление сообщения в определённый чат.
- get_chats_for_user - получение всех чатов пользователя.
- get_chat_messages - получение контекста диалога в определённом чате.
- delete_chat_for_user - удаление определённого чата пользователя.
- delete_all_chats_for_user - удаление всех чатов пользователя.
- get_user_id - получение user_id по email и password.
- user_exists - проверка существования email в базе.

Чтобы всё заработало, нужно было поднять сервер с `MongoDB`. Для этого использовался `docker` и `docker-compose` с образом `mongo:5.0`

5.6 Логирование и обработка ошибок

Чтобы собирать логи и отлавливать ошибки, автор добавил общий `logger`, который пишет все логи в файл `logs.log`, а также обернул потенциально опасные места в `try-except`, например:

```
try:
    message = {
        "role": role,
        "type": type,  # "text" или "image"
        "content": content, # "text" или "url"
        "created_at": datetime.now().isoformat()
    }
    result = await self.chats.update_one(
        {"_id": ObjectId(chat_id)},
        {
            "$push": {"messages": message},
            "$set": {"updated_at": datetime.now().isoformat()}
        }
    )
    return result.modified_count > 0
except Exception as e:
    self.logger.error(f"add_message: {e}")
    return False
```

Рис. 5.2: Обработка ошибок и логгирование в методе `MongoDB`

```
try:
    response = await self.client.chat.completions.create(
        model=self.chat_model,
        messages=history
    )

    if response.choices:
        return response.choices[0].message.content

    self.logger.error(f"response: пришёл пустой ответ от провайдера")
    return ""
except Exception as e:
    self.logger.error(f"response: {e}")
    return ""
```

Рис. 5.3: Обработка ошибок и логгирование в методе `Model`

5.7 Пример диалога

- Вначале идёт служебная информация логгера. Затем у пользователя спрашивают email (email) и пароль (password).
- Так как у пользователя ещё нет чатов, то в качестве chat_id пишется -. Затем создаётся новый чат и у пользователя спрашивают его

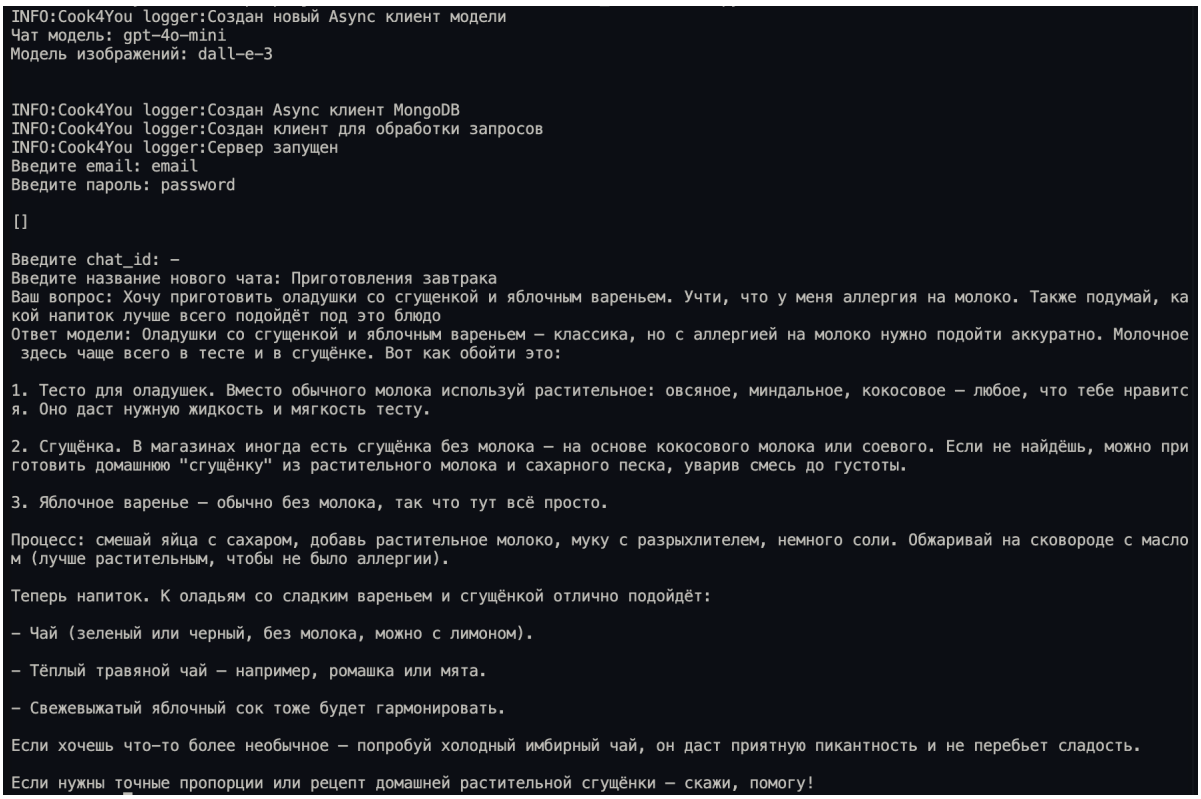

Рис. 5.4: Пример диалога с моделью

название (Приготовление завтрака).

- После этого появляется возможность задать вопрос, например: *«Хочу приготовить оладушки со сгущенкой и яблочным вареньем. Учти, что у меня аллергия на молоко. Также подумай, какой напиток лучше всего подойдёт под это блюдо»*.

- Далее идёт ответ модели

После этого можно попросить модель предоставить более подробный рецепт, указать точное количество ингредиентов и т.д. Также можно увидеть примеры некорректного вопроса и промпта, который модель создаёт для генератора изображений и само изображение.

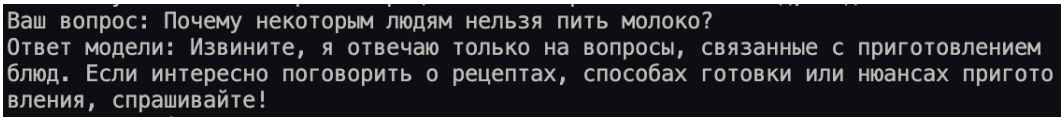

Рис. 5.5: Пример некорректного вопроса

```
Ваш вопрос: image
INFO:Cook4You logger:
prompt для изображения: Fluffy golden pancakes made with plant-based milk, light
ly browned on both sides, served on a white plate with a generous drizzle of hom
emade coconut milk caramel sauce and vibrant apple jam on the side, garnished wi
th a small sprig of mint, set on a rustic wooden table with soft natural lightin
g.
```

Рис. 5.6: Пример промпта, который создала модель для генератора изображений

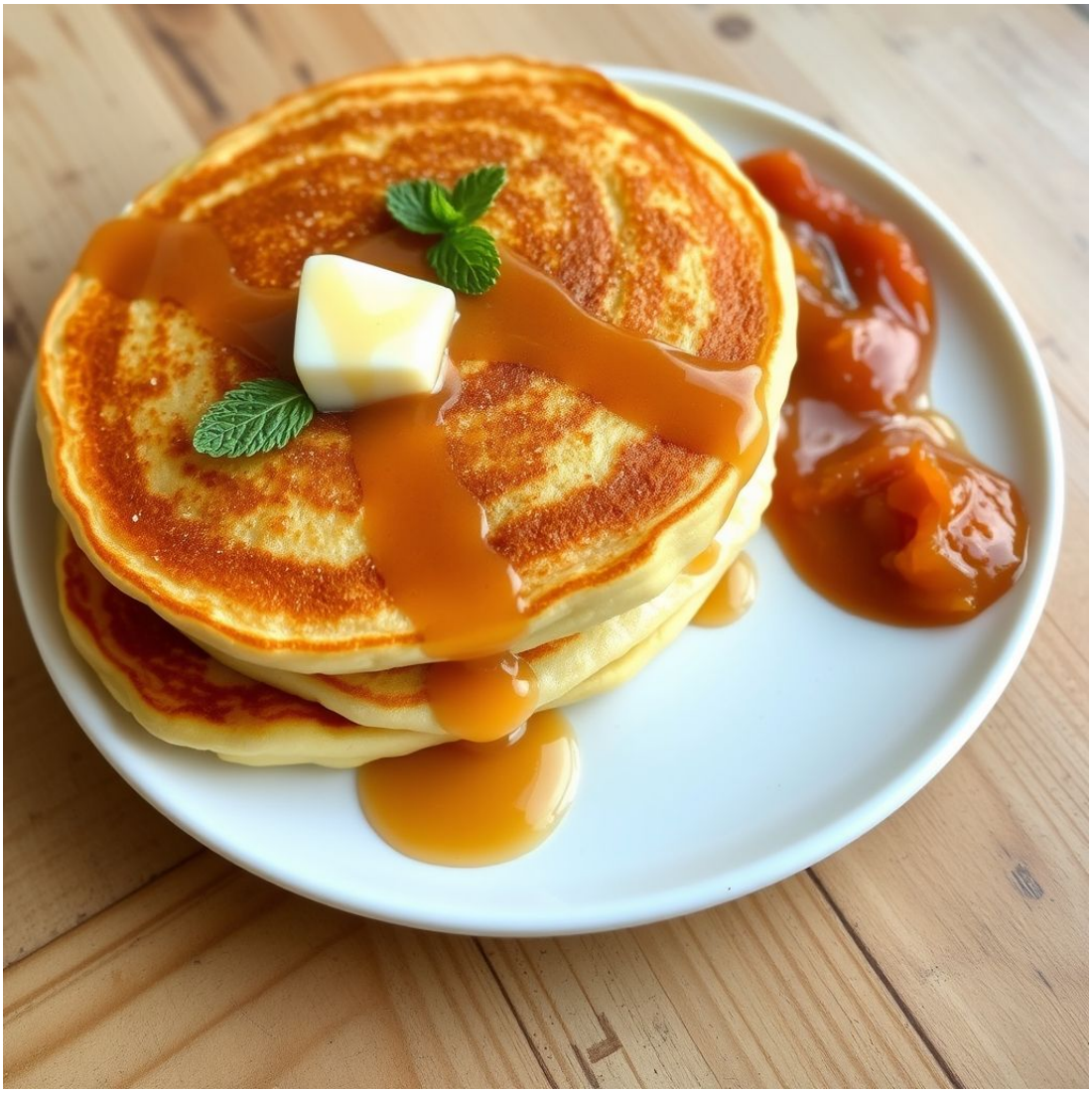

Рис. 5.7: Пример изображения блюда

5.8 Создание сервера

После того, как был реализован фронтенд страницы авторизации и регистрации, нужно было запустить основной сервер, который бы обрабатывал запросы с веб-сервера. Для этого были выбраны `Python` библиотеки `FastAPI` и `uvicorn`.

- FastAPI — это современный веб-фреймворк для создания API на Python. Он позволяет:
  - Быстро создавать REST API с автоматической генерацией OpenAPI документации.
  - Обеспечивает валидацию данных «из коробки» с помощью Pydantic.
  - Поддерживает асинхронность, что позволяет эффективно обрабатывать множество одновременных запросов.
  - Подходит для микросервисной архитектуры и легко масштабируется.
- Uvicorn — это ASGI-сервер (Asynchronous Server Gateway Interface), который:
  - Запускает FastAPI-приложение и принимает входящие HTTP-запросы.
  - Обеспечивает высокую производительность благодаря использованию асинхронного ввода-вывода.
  - Поддерживает WebSocket, HTTP/2 и другие современные веб-технологии.

Итоговый сервер получился очень простым и понятным: 5.5

```
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=[f"http://{SITE_SERVER_HOST}:{SITE_SERVER_PORT}"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.post("/register")
async def register(auth_data: AuthData):
```

```
    email = auth_data.email
    password = auth_data.password

    if not email or not password:
        raise HTTPException(status_code=400, detail="incorrect request")

    exists = await client.check_user_existance(email)
    if exists:
        raise HTTPException(status_code=409, detail="email already exists")
    user_id = await client.new_user(email, password)

    token = create_access_token(data={"user_id": user_id})
    return {"user_id": user_id, "token": token}

@app.post("/login")
async def login(auth_data: AuthData):
    email = auth_data.email
    password = auth_data.password

    if not email or not password:
        raise HTTPException(status_code=400, detail="incorrect request")

    user_id = await client.get_user_id(email, password)
    if len(user_id) == 0:
        raise HTTPException(status_code=401, detail="incorrect auth data")

    token = create_access_token(data={"user_id": user_id})
    return {"user_id": user_id, "token": token}

if __name__ == "__main__":
    logger.info(f"http://{MODEL_SERVER_HOST}:{MODEL_SERVER_PORT}/docs")
    uvicorn.run(
        app,
        host=MODEL_SERVER_HOST,
        port=int(MODEL_SERVER_PORT),
        log_level="info",
    )
```

Листинг 5.5: Основной FastAPI + uvicorn сервер

Чтобы каждый раз не спрашивать email и пароль пользователя, можно

идентифицировать его по JWT токену, который будет описан в разделе 6.2. Вот код, который генерирует этот токен и проверяет его на валидность при помощи библиотеки jose 5.6

```python
def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.now() + timedelta(minutes=JWT_TIME)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, key=JWT_SECRET, algorithm=JWT_ALGO)
    return encoded_jwt

def verify_token(token: str = Depends(oauth2_scheme)) -> str:
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="unavailable token",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, JWT_SECRET, algorithms=[JWT_ALGO])
        user_id: str = payload.get("user_id")
        if user_id is None:
            raise credentials_exception
        return user_id
    except ExpiredSignatureError:
        raise HTTPException(status_code=401, detail="token expired")
    except JWTError:
        raise credentials_exception
```

Листинг 5.6: Генерация и проверка JWT токена

6 Фронтенд

Здесь будет рассмотрен фронтенд Cook4You. Хоть в названии и фигурирует слово «сайт», всё-таки Cook4You - веб-приложение. Обычные сайты представляют собой просто набор статических страниц с информацией. Веб-приложение в свою очередь является программой, которую можно использовать через интернет для выполнения различных задач. Оно отличается от обычного сайта тем, что позволяет пользователям взаимодействовать с ним, вводя данные и получая результаты.

6.1 Технологии для создания веб-приложения

До этого автор ни разу не создавал веб-приложений, поэтому сначала необходимо было понять, как это делать. В качестве стека для фронтенд-разработки автор выбрал связку `React` + `Vite` + `Tailwind CSS`, так как она обеспечивает быструю разработку, высокую производительность и отличную масштабируемость.

- `React` — это библиотека для создания пользовательских интерфейсов. Автор использовал её, чтобы построить приложение на основе компонентов. Каждый экран, кнопка и элемент были реализованы в виде отдельных компонентов, что облегчает поддержку и повторное использование кода.

- `Vite` использовался как инструмент сборки и разработки. Он позволил значительно ускорить процесс запуска приложения и обновления интерфейса при изменении кода благодаря технологии горячей перезагрузки `HMR` (Hot Module Replacement). Конфигурация оказалась простой, а интеграция с `React` заняла всего пару минут.

- Для стилизации интерфейса автор выбрал `Tailwind CSS` — утилитарный фреймворк, позволяющий задавать стили прямо в `JSX`. Это дало

возможность быстро экспериментировать с внешним видом, не отходя от основного кода компонента. Например, чтобы создать кнопку с определённым стилем, не нужно было писать отдельный CSS-файл — достаточно было указать соответствующие классы.

6.2 OpenAPI

Чтобы веб-сервер проекта мог общаться с основным сервером, нужно создать универсальную схему, где будут прописаны все возможные запросы веб-сервера и соответствующие ответы основного сервера. Один из лучших способов это сделать - `OpenAPI` контракт. Удобнее всего составлять `OpenAPI` документация в `Swagger` - мощного инструментального набора для документирования и визуализации `REST APIs`. Пока что в текущем `OpenAPI` контракте прописаны только действия `/login` и `/register` - авторизация и регистрация пользователей. В будущем туда будут добавлены действия и эндпоинты, связанные с чатами.

```
openapi: 3.0.0
info:
  title: Cook4You API
  version: 1.0.0

paths:
  /register:
    post:
      summary: Регистрация нового пользователя
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '/components/schemas/AuthData'
      responses:
        200:
          description: Пользователь успешно зарегистрирован
          content:
```

```
            application/json:
              schema:
                $ref: '/components/schemas/AuthTokenResponse'
        400:
          description: Некорректные параметры запроса
          content:
            application/json:
              schema:
                $ref: '/components/schemas/BadRequestError'
        409:
          description: Пользователь с таким email уже существует
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
                    description: "Email уже существует"
        500:
          description: Ошибка создания пользователя
          content:
            application/json:
              schema:
                $ref: '/components/schemas/InternalServerError'

/login:
  post:
    summary: Аутентификация пользователя
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '/components/schemas/AuthData'
    responses:
      200:
        description: Успешная аутентификация
        content:
          application/json:
```

```
              schema:
                $ref: '/components/schemas/AuthTokenResponse'
        400:
          description: Некорректные параметры запроса
          content:
            application/json:
              schema:
                $ref: '/components/schemas/BadRequestError'
        401:
          description: Неверный email или пароль
        500:
          description: Ошибка аутентификации
          content:
            application/json:
              schema:
                $ref: '/components/schemas/InternalServerError'

components:
  schemas:
    InternalServerError:
      type: object
      properties:
        detail:
          type: string
          description: Внутренняя ошибка

    BadRequestError:
      type: object
      properties:
        detail:
          type: string
          description: Ошибка запроса

    AuthData:
      type: object
      properties:
        email:
          type: string
          format: email
        password:
```

```
        type: string
        format: password

  AuthTokenResponse:
    type: object
    properties:
      token:
        type: string
        description: JWT токен для аутентификации
      user_id:
        type: string
        description: ID пользователя
```

Можно заметить, что в качестве идентификации пользователя используется JWT токен - компактный и безопасный способ передачи данных между клиентом и сервером в виде токена, который чаще всего используется для аутентификации и авторизации пользователей. JWT состоит из трёх частей, разделённых точками:

- Header (заголовок) — указывает тип токена и алгоритм подписи (например, HS256).
- Payload (полезная нагрузка) — содержит данные (например, user_id, login, срок действия токена).
- Signature (подпись) — создаётся на основе заголовка и payload, чтобы проверить подлинность токена.

6.3 Страница регистрации и авторизации

Создав OpenAPI, нужно было реализовать эти запросы на стороне веб-сервера и создать страницу регистрации и авторизации. 6.7

```
const handleSubmit = async (e) => {
    e.preventDefault();

    if (!email || !password) {
```

```
      alert("need email and password");
      return;
    }

    if (!isLogin && password !== repeatPassword) {
      alert("incorrect data");
      return;
    }

    try {
      const url = isLogin
        ? "http://localhost:8080/login"
        : "http://localhost:8080/register";

      const response = await fetch(url, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email: email,
          password: password,
        }),
      });

      const data = await response.json();

      if (!response.ok) {
        alert("response error: " + data.detail);
      } else {
        alert(isLogin ? "success login" : "success registration");
      }
    } catch (err) {
      alert("internal error: " + err.message);
    }
  };
```

Листинг 6.7: Обработчик регистрации и авторизации

Этот код обрабатывает email и пароль, введённые пользователем, и отправ-

ляет их на основной сервер, который проверяет их на подлинность. Затем основной сервер возвращает результат, и он дублируется на этой странице. Итоговая страница выглядит так:

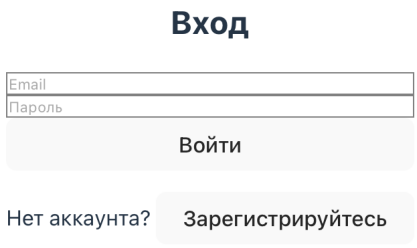

Рис. 6.1: Форма для авторизации

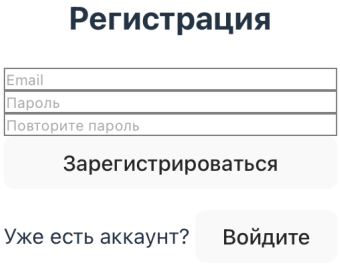

Рис. 6.2: Форма для регистрации

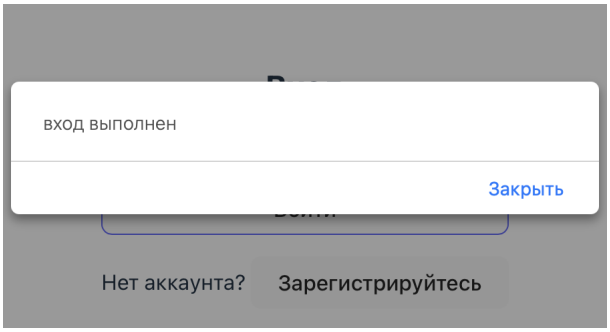

Рис. 6.3: Успешная авторизация

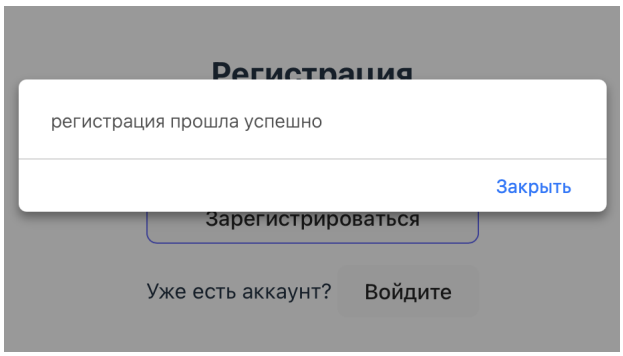

Рис. 6.4: Успешная регистрация

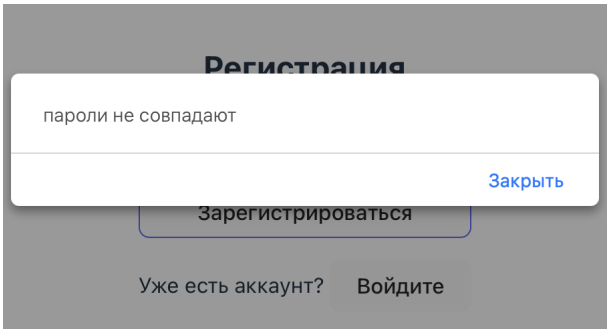

Рис. 6.5: Ввод несовпадающих паролей при регистрации

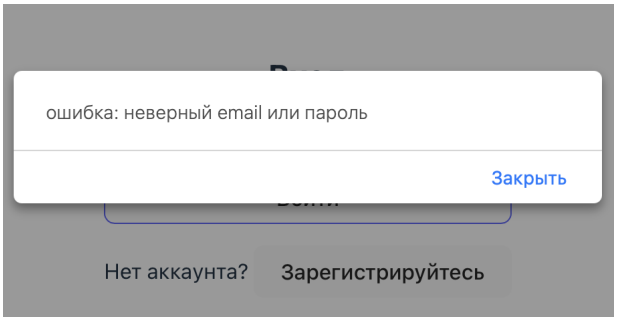

Рис. 6.6: Неверный логин или пароль

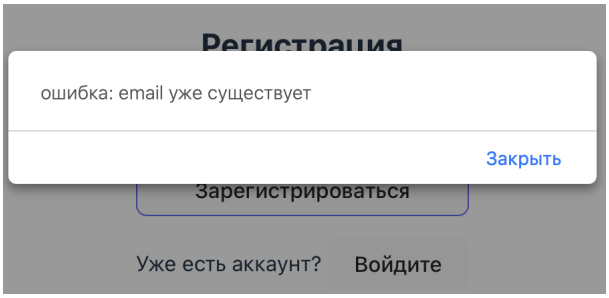

Рис. 6.7: Email уже существует

7 Заключение

7.1 Результаты

По итогам работы было выполнено:

- Изучить `API` генеративных моделей
- Создать схему данных. Выбрать сооветствующую базу данных
- Реализовать подключение к `API` готовых моделей
- Настроить работу моделей
- Интегрировать базу данных
- Добавить логирование и обработку ошибок
- Продумать реализацию веб-сайта. Изучить способы его создания
- Создать `OpenAPI` для запросов, которые веб-сервер будет отправлять на сервер с моделью
- Реализовать страницу регистрации и авторизации
- На сервере добавить эндпоинты для регистрации и авторизации
- Протестировать получившийся функционал

В процессе разработки приобретены и улучшены навыки:

- Взаимодействие с `API` стороннего сервиса через `Python` билиотеку `g4f`
- Продумывание и составление схемы базы данных. Выбор базы данных, подходящей под конкретный случай
- Запуск `MongoDB` в `docker`. Подключение к ней через `Python` библиотеку `motor`. Реализация запросов к базе данных

- Описание OpenAPI документации при помощи Swagger для дальнейшего общения между веб-сервером и основным сервером
- Использование FastAPI и uvicorn для реализации асинхронного REST API сервера
- Интеграция JWT токена для идентификации пользователя при помощи Python библиотеки jose
- Создания сайта и веб-сервера при помощи JS React, Vite и Tailwind CSS
- Настройка CORS для разрешения запросов с веб-сервера на основной сервер. Реализация общения между двумя серверами.

7.2 Развитие работы

- Закончить все основные задачи, описанные в разделе 2.2
- Протестировать MVP на небольшой выборке из целевой аудитории
- Исправить основные недочёты, выявленные после теста
- Провести нагрузочное тестирование, примерно оценить RPS, который выдерживает сервис
- При очень низких значениях, изучить и внедрить платное API известных провайдеров, либо дообучить собственную модель и запустить локально
- Улучшить интерфейс и дизайн веб-приложения с помощью популярных шаблонов
- Запустить рекламу Cook4You

Список литературы

1. Библиотека g4f // pypi.org URL: https://pypi.org/project/g4f/ (дата обращения: 29.04.2025).

2. Swagger // editor.swagger.io URL: https://editor.swagger.io (дата обращения: 29.04.2025).

3. OpenAPI // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/OpenAPI_()(: 29.04.2025).

4. Docker // docker.com URL: https://www.docker.com (дата обращения: 29.04.2025).

5. FastAPI // fastapi.tiangolo.com URL: https://fastapi.tiangolo.com (дата обращения: 29.04.2025).

6. React // react.dev URL: https://react.dev (дата обращения: 29.04.2025).

7. Общее число запросов «приготовить» за 28.03.2025 – 28.04.2025 // wordstat.yandex.ru URL: https://wordstat.yandex.ru/?region=allview=tablewords=приготовить (дата обращения: 29.04.2025).

8. Лучшие нейросети для генерации рецептов в 2025 году // vc.ru URL: https://vc.ru/id2581788/1842793-luchshie-neiroseti-dlya-generacii-receptov-v-2025-godu (дата обращения: 29.04.2025).

9. 6 нейросетей для тех, кому надоело думать, что приготовить // t-j.ru URL: https://t-j.ru/short/ai-recipes/ (дата обращения: 29.04.2025).