

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**  
**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**  
**Кафедра технологий программирования**

**ОСОБЕННОСТИ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ В**  
**ВЕБ-ПРИЛОЖЕНИИ НА ОСНОВЕ PYTHON И FLASK**

Курсовая работа

Кражевского Алексея Игоревича,  
студента 3 курса 13 группы,  
специальность «прикладная  
информатика»

Научный руководитель:  
Старший преподаватель,  
Зенько Татьяна Алексеевна

Минск, 2023

## РЕФЕРАТ

**Ключевые слова:** PYTHON, FLASK, WEB-APPLICATION, API, SQL, SQLALCHEMY, DOCKER.

**Объект исследования** – python и фреймворки Flask, SQLAlchemy и их возможности при создании веб-приложения и работы с базами данных, особенности объектно-реляционного отображения с использованием вышеперечисленных технологий, а также исследование технологий ORM на примере SQLAlchemy.

**Цели работы** – разработать веб-приложение на основе языка Python, рассмотреть возможности и особенности работы с базами данных фреймворка SQLAlchemy.

**Методы исследования** – анализ, моделирование.

**Результатами являются** – веб-приложение «Блог-энциклопедия еды» на языке Python.

**Область применения** – программная инженерия, проектирование программных систем и разработка корпоративных приложений.

## РЭФЕРАТ

**Ключавыя словы:** PYTHON, FLASK, WEB-APPLICATION, API, SQL, SQLALCHEMY, DOCKER.

**Аб'ект даследавання** – Python і фрэймворкі Flask, SQLAlchemy і іх магчымасці пры стварэнні вэб-прыкладання і працы з базамі даных.

**Мэты працы** – распрацаваць вэб-прыкладанне на аснове мовы Python, разгледзець магчымасці і асаблівасці працы з базамі даных фрэймворка SQLAlchemy, асаблівасці аб'ектна-рэляцыйнага адлюстравання з выкарыстаннем вышэй пералічаных тэхналогій, а таксама даследаванне тэхналогій ORM на прыкладзе SQLAlchemy.

**Метады даследавання** – аналіз, мадэляванне.

**Вынікамі з'яўляюцца** – вэб-прыкладанне «Блог-энцыклапедыя ежы» на мове Python.

**Вобласць ужывання** – праграмная інжынерыя, праектаванне праграмных сістэм і распрацоўка карпаратыўных прыкладанняў.

## ESSAY

**Keywords:** PYTHON, FLASK, WEB-APPLICATION, API, SQL, SQLALCHEMY, DOCKER.

**Object of research** – Python and Flask, SQLAlchemy frameworks and their possibilities while creating web-application and working with databases, features of object-relational mapping using the above technologies, as well as the study of ORM technologies using the example of SQLAlchemy.

**Purpose** – to develop web-application using Python language, inspect peculiarities of working with databases using SQLAlchemy framework.

**Methods of research** – analysis, modeling.

**The results are** – web-application “Blog-encyclopedia of food” in Python.

**Scope** – software engineering, software system design and enterprise application development.

## СОДЕРЖАНИЕ

<b>ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ</b>	5
<b>ВВЕДЕНИЕ</b>	6
<b>Глава 1. МЕТОДЫ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ НА ОСНОВЕ PYTHON И FLASK</b>	8
1.1 Сравнение фреймворков для разработки web-приложений на языке Python	8
1.1.1 Django	8
1.1.2 Flask	9
1.1.3 FastAPI	10
1.1.4 Сравнение трех фреймворков	11
1.2. Объектно-реляционное отображение и хранение данных	13
1.2.1 Сравнение ORM, доступных для веб-разработки на Python	14
1.2.2 Разбор фреймворка SQLAlchemy	15
1.3 Контейнеризация веб-приложений с помощью Docker	15
<b>Глава 2. ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ НА ОСНОВЕ PYTHON, FLASK И POSTGRESQL</b>	19
2.1 Задачи проекта	19
2.2 Функциональные требования	19
2.3 Нефункциональные требования	20
2.4 Проектирование структуры и архитектуры проекта	20
2.4.1 Диаграмма вариантов использования	20
2.4.2 Диаграммы деятельности	21
2.4.3 Проектирование архитектуры базы данных	22
<b>Глава 3. РЕАЛИЗАЦИЯ WEB-ПРИЛОЖЕНИЯ</b>	25
3.1 Основные элементы веб-приложения	25
3.2 Реализация надстройки для работы с базой данных	33
3.3 Хранение данных	33
3.4 Авторизация	34
<b>ЗАКЛЮЧЕНИЕ</b>	36
<b>СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ</b>	38
<b>ПРИЛОЖЕНИЯ</b>	39

## **ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

REST	передача репрезентативного состояния (REpresentational State Transfer),
API	программный интерфейс приложения (Application Programming Interface),
GraphQL API	синтаксис, который описывает как запрашивать данные,
JSON	текстовый формат обмена данными, основанный на JavaScript,
UML	унифицированный язык моделирования (Unified Modeling Language).

## **ВВЕДЕНИЕ**

В современных реалиях обойтись без использования интернета практически невозможно. Все пользуются доступом в интернет каждый день вне зависимости от цели – это может быть работа, отдых, поиск какой-либо информации или же просто общение.

И тем, к чему люди обращаются каждый день, являются веб сайты, которые позволяют пользователям получать тот контент, который они хотят.

Итак, в разработанном веб-приложении объединяются две вещи – простота доступа сайта и вся информация о еде. Ведь все люди так или иначе готовят, не так ли? В создаваемом веб-приложении пользователям предоставляется возможность делиться рецептами своих любимых блюд, изучать традиционные блюда различных стран и культур, оценивать их, оставлять комментарии и узнавать новые рецепты прямиком с сайта.

Данное веб-приложение может быть полезным для гурманов, поваров, домохозяек (и домохозяев) или же для любых людей, так или иначе интересующихся готовкой.

Данное веб-приложение разработано на базе языка Python и веб-фреймворка Flask.

Объектом исследования являются особенности объектно-реляционного отображения с использованием вышеперечисленных технологий, а также исследование технологий ORM на примере SQLAlchemy.

Примененные в данной работе методы исследования включают в себя изучение и анализ литературы и документации используемых фреймворков и технологий, обобщение изученных сведений, моделирование и проектирование функциональных и нефункциональных требований и спецификаций, и разработка прототипа веб-приложения на основе Python и Flask.

В первой главе рассматриваются примененные технологии и объясняется их выбор, а также рассмотрены особенности объектно-реляционного отображения в веб-приложении на Python и Flask и рассмотрена технология докеризации веб-приложений.

Во второй главе фиксируются функциональные и нефункциональные требования к создаваемому веб-приложению, прилагаются разработанные диаграммы, схемы и архитектура проекта и базы данных.

В третьей главе представляется реализация веб-приложения и описывается процесс разработки данного приложения.



# **Глава 1. МЕТОДЫ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ НА ОСНОВЕ PYTHON И FLASK**

## **1.1 Сравнение фреймворков для разработки web-приложений на языке Python**

Фреймворк – коллекция пакетов и модулей, которые используются для написания программного обеспечения и помогают разработчикам не акцентировать внимание на низкоуровневых деталях процесса разработки. При написании веб-приложений на Python, любой разработчик столкнется с выбором необходимого фреймворка для работы с максимальной эффективностью.

Три наиболее популярных фреймворка при разработки веб-приложений это Flask, Django и FastAPI [3]. Каждый из них может пригодиться в различных ситуациях и для различных целей, а значит стоит разобраться когда и какой фреймворк использовать.

### **1.1.1 Django**

Django – это бесплатный фреймворк с открытым исходным кодом, который используется для написания веб-сайтов. Он был создан Саймоном Уиллисом и Адрианом Холовати в 2003 году и использует модель Model-Template-View.

Django нравится разработчикам за «прямолинейность» и за поощрение повторного использования кода. Этот фреймворк является одним из самых популярных во всем мире и используется в таких известных компаниях как Instagram, YouTube и так далее.

К плюсам Django можно отнести [3]:

Структура кода при использовании Django очень эффективна, а значит разработчики легко могут добавить сайту функционал;

С помощью Rest Framework в Django (называется Django Rest Framework)

можно достаточно легко создавать Web APIs;

Django предоставляет защиту от SQL инъекций и некоторых атак на сайт, что повышает защиту веб-приложения.

Из минусов можно выделить [3]:

Django использует ORM, которая была создана до SQLAlchemy, что делает работу с базами данных в Django менее гибкой;

Данный фреймворк является довольно громоздким (из-за множества модулей, которые можно использовать несколько раз), что может ограничить скорость работы программистов;

Сам фреймворк может быть медленнее, чем аналоги, из-за своей громоздкости;

У моделей в Django может не быть никаких примесей, только простое наследование.

Итак, Django можно использовать при создании проектов любого объема. Он подойдет при создании простых сайтов или же при написании высокопроизводительных сайтов. Например, он подойдет для написания сайта-менеджмента школы, онлайн-магазинов, онлайн-бронирования и т.д.

Django поддерживает создания таких вещей, как верификация по фото, системы онлайн почт, пользовательские уведомления и т.д.

### **1.1.2 Flask**

Flask – микро веб-фреймворк, т.е. он содержит минимальные настройки и служит для создания небольших веб-приложений. Он предоставляет такие опции, как шаблоны движков (таких как ORM), кеширование и аутентификацию. Flask создавался как простой, быстрый фреймворк для расширения приложений и микросервисов. Также можно сказать, что данный фреймворк является оберткой над Werkzeug и Jinja.

Преимущества Flask [3]:

Flask гибкий и удобный. Многие части фреймворка изменяемы, что не очень распространено среди веб-фреймворков;

Он подходит для начинающих, т.к. он достаточно простой, что дает разработчикам пространство для обучения и лучшего понимания;

Flask позволяет unit тестирование, имеет встроенный сервер для разработки.

К минусам можно отнести [3]:

Flask использует модули, что является сторонним вмешательством и может привести к дырам в безопасности. Модули находятся между фреймворком и разработчиком;

Flask обрабатывает запросы последовательно, один за другим, так что не зависимо от количества запросов. Они обрабатываются последовательно, что увеличивает время ответа.

Flask можно использовать для коммерческих проектов. Он может помочь быстро начать работу, но он не очень хорош при нагрузках на сайт. Например, он может подойти при создании статических сайтов, ботов для Twitter или Facebook. Flask не рекомендуется использовать для создания высоконагруженного корпоративного программного обеспечения.

### **1.1.3 FastAPI**

FastAPI – это современный, быстрый и высокопроизводительный веб-фреймворк на Python с открытым исходным кодом. Он используется для написания Web APIs и основан на подсказках стандартного типа Python 3.6+.

Плюсы FastAPI [3]:

Существует библиотека `graphql-python`, которая позволяет легко писать GraphQL API с помощью FastAPI;

FastAPI основан на таких стандартах, как JSON Schema (для валидации структуры JSON данных), OAuth 2.0 (протокол авторизации, который является

стандартом), OpenAPI (общедоступный API);

Также FastAPI проверяет тип данных даже в глубоко вложенных JSON запросах.

Минусы FastAPI [3]:

FastAPI – относительно новый фреймворк, и его сообщество довольно небольшое по сравнению с другими фреймворками, и у него нет детальной документации. Существует достаточно небольшое количество образовательных материалов для данного фреймворка.

Использовать FastAPI стоит при условии, что главная конечная цель – скорость. Он используется в Netflix из-за своей скорости работы и возможностях быстрого увеличения и исправления ошибок.

#### **1.1.4 Сравнение трех фреймворков**

Рассмотрим такие критерии сравнения, как пакеты, сообщества, производительность, гибкость и возможность обучения.

Среди всех трех фреймворков, у Django есть наибольшее количество пакетов, которые обеспечивают повторное использование кода. Он сам по себе является полноценным фреймворком для создания готовых решений, в отличие от Flask и FastAPI, минималистичных фреймворков, ориентированных на скорость.

Также у Django самое большое сообщество из-за своей популярности и широкого применения по сравнению с остальными. Далее идет Flask, у которого тоже достаточно большое количество пользователей. И наконец, FastAPI, с наименьшим сообществом из-за своей относительно новой даты появления.

По производительности фреймворки можно расположить в следующем порядке: FastAPI, Flask, Django. Такой порядок обоснован преимуществами и

недостатками каждого фреймворка, рассмотренных ранее.



В плане гибкости можно сказать что самым гибким является Flask, затем FastAPI, и затем Django. Это можно объяснить минималистичностью фреймворков, и предоставляемой возможностью программистам дать волю фантазии.

И наконец, по образовательным ресурсам побеждает Django, ввиду своей популярности и многозадачности, далее идет Flask, так как он также достаточно популярен среди разработчиков, и, наконец, FastAPI, с наименьшим количеством онлайн-курсов или статей. Это совершенно не значит, что Django изучить проще всего, можно даже сказать, что наиболее простым для изучения будет FastAPI, из-за своего прямого подхода к разработки веб-приложений.

#### Рисунок 1.1 – Сравнение трех фреймворков [3]

Обобщим вышеперечисленные критерии сравнения трех фреймворков.

После тщательного сравнения фреймворков Django, Flask и FastAPI, можно сделать вывод, в каких целях и для решения каких задач стоит использовать каждый из них. Django идеально подходит для создания корпоративных веб-сайтов, у него есть много встроенных функций, что полезно для крупных проектов.

	<b>django</b>	 <b>Flask</b>	 <b>FastAPI</b>
<b>Performance speed</b>	Normal	Faster than Django	The fastest out there
<b>Async support</b>	<b>YES</b> with restricted latency	<b>NO</b> needs Asyncio	<b>YES</b> native async support
<b>Packages</b>	<b>Plenty</b> for robust web apps	<b>Less than Django</b> for minimalistic apps	<b>The least of all</b> for building web apps faster
<b>Popularity</b>	The most popular	The second popular	Relatively new
<b>Learning</b>	Hard to learn	Easier to learn	The easiest to learn

Flask хорошо подходит для разработчиков, которые хотят быстро создать прототип веб-приложения и написать Web API. В то же время, FastAPI служит для создания быстрых и расширяющихся сайтов.

Можем сделать вывод, что стоит выбирать фреймворк для решаемых задач. В целом у каждого из этих фреймворков есть все необходимое для создания и поддержания веб-приложений. Они часто используются программистами и являются отличными решениями для написания сайтов. Для решения исследуемой задачи выбран фреймворк Flask из-за скорости разработки сайта, из-за его гибкости и практичности при создании относительно небольших сайтов и разработки Web API.

## 1.2. Объектно-реляционное отображение и хранение данных

ORM (Object Relational Mapper), или объектно-реляционное отображение данных, - технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

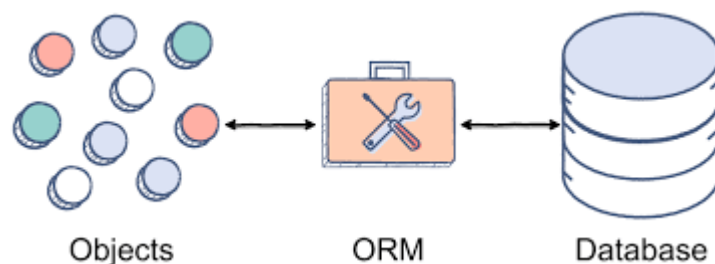


Рисунок 1.2 – Схема работы ORM [6]

В случае языка Python, для работы с базами данных существует библиотека SQLAlchemy [5], которая используется в качестве ORM, и транслирует классы в Python на таблицы в реляционных базах данных и автоматически конвертирует вызовы функций в SQL запросы. Данный фреймворк предоставляет стандартный интерфейс, позволяющий разработчикам писать код, не зависящий от используемой базы данных, и поддерживающий огромный спектр различных баз данных.

### 1.2.1 Сравнение ORM, доступных для веб-разработки на Python

Прежде чем переходить к разбору функциональности фреймворка SQLAlchemy, стоит рассмотреть возможности, положительные стороны и недостатки других доступных ORM-решений.

**1. DjangoORM и ponyORM.** В нашем случае данные решения не подходят. Оба данных фреймворка не поддерживают работу в асинхронном режиме, к тому же DjangoORM, как видно из названия, используется в фреймворке Django, который при разработке веб-приложения не используется [5].

**2. Peewee.** Данный фреймворк поддерживает работу в асинхронном режиме, но только с Core-функциями. Однако этот фреймворк не так популярен, и для него нет большого количества обучающих ресурсов, что затрудняет его понимание и изучение.

**3. Gino, tortoise-orm.** Два новых проекта, которые только начали развиваться и по которым отсутствует документация. По этой причине данные решения не исследовались в рамках проекта на практике. Фреймворк Gino

основан на SQLAlchemy Core, а tortoise-orm больше похож на DjangoORM своим способом доступа к данным [5].

Таким образом, был выбран фреймворк для работы с базами данных, SQLAlchemy, по причине его популярности, большого разнообразия обучающих ресурсов и относительной простоты в использовании.

### 1.2.2 Разбор фреймворка SQLAlchemy

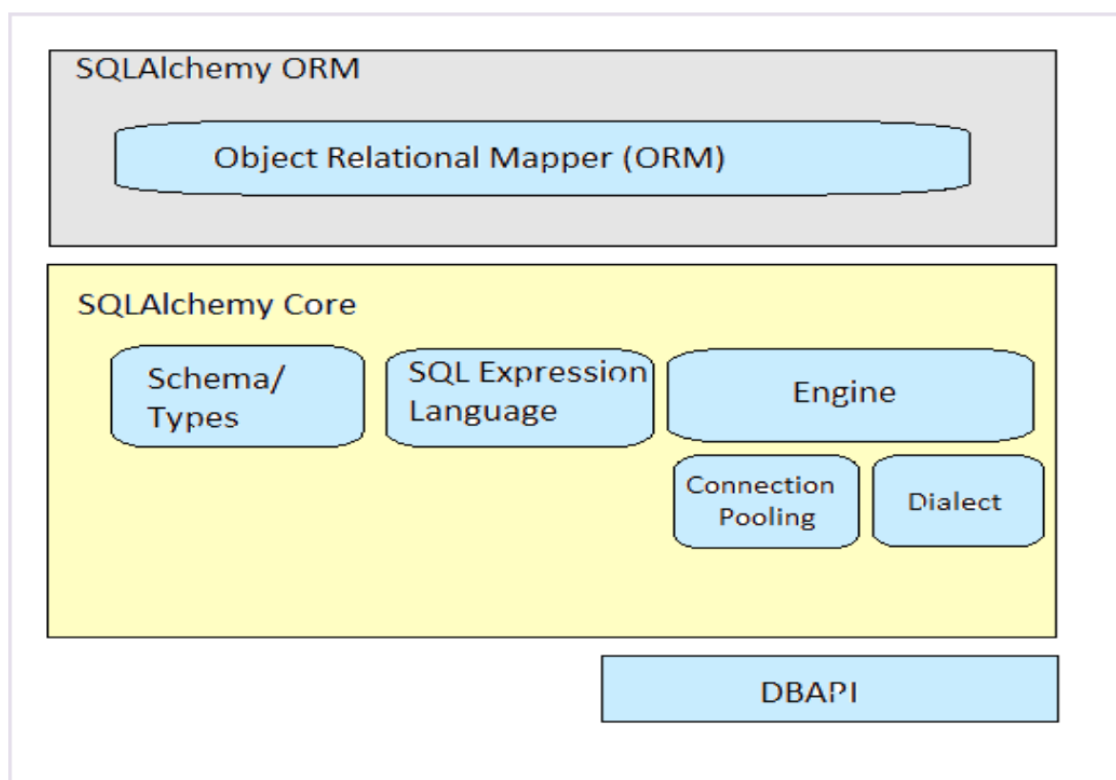


Рисунок 1.3 – Схема работы SQLAlchemy Core [7]

Фреймворк SQLAlchemy состоит из двух основных частей, как видно из рисунка 1.2. Первая часть – SQLAlchemy Core – это абстракция над реляционной базой данных. Вторая часть – SQLAlchemy ORM – это связка между используемой реляционной базой данных и ее объектным представлением, написанным на языке Python. DBAPI на рисунке отвечает за непосредственное общение с базой данных, через SQL-запросы.

## 1.3 Контейнеризация веб-приложений с помощью Docker

На сегодняшний день, разработчикам нужно быстро создавать, поставлять



и запускать приложения. И главным помощником в этом является технология контейнеризации и, в частности, Docker.

С помощью Docker можем публиковать, тестировать и развертывать программное обеспечение, в то же время поддерживая полный контроль над своей инфраструктурой. Данная технология значительно уменьшает время от написания кода нового продукта до выпуска его готовой версии.

Контейнеризацией называют подход к разработке программного обеспечения, при котором все зависимости приложения, его конфигурация, и само приложение упаковывается все вместе в один образ контейнера.

После контейнеризации приложение может быть протестировано в качестве модуля, также может быть развернуто как экземпляр контейнера в хостовой операционной системе или же на виртуальной машине на облаке.

Docker работает с технологией контейнеризации, т.е. создает контейнеры приложений. Docker container (контейнер) – это коллекция зависимостей приложения и кода приложения, организованная как программное обеспечение, что позволяет быстро запускать приложения в ряде возможных вычислительных сред.

Docker image (образ), в свою очередь, является неким планом, который отвечает за то, как приложение запускается. Для создания своих образов, чтобы Docker знал как их собирать, используется так называемый файл Dockerfile с набором инструкций по запуску приложения.

Инструкции из данного файла выполняются пользователем в командном интерпретаторе для того, чтобы в конечном счете создать образ необходимого приложения.

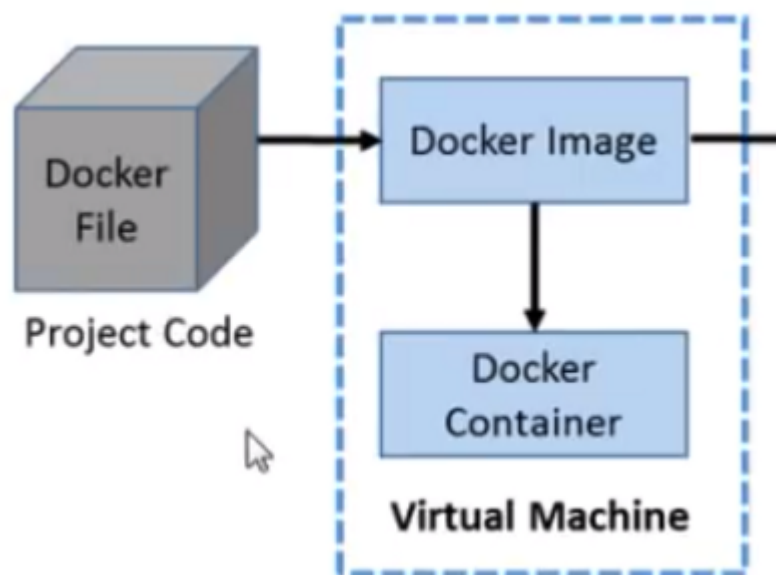


Рисунок 1.4 – Схема работы Dockerfile [8]

Архитектура Docker – это архитектура клиент-сервер. Клиент Docker общается со своим демоном, расположенным либо на хостовой операционной системе, либо удаленно, тогда общение будет происходить через REST API или через сокет [8].

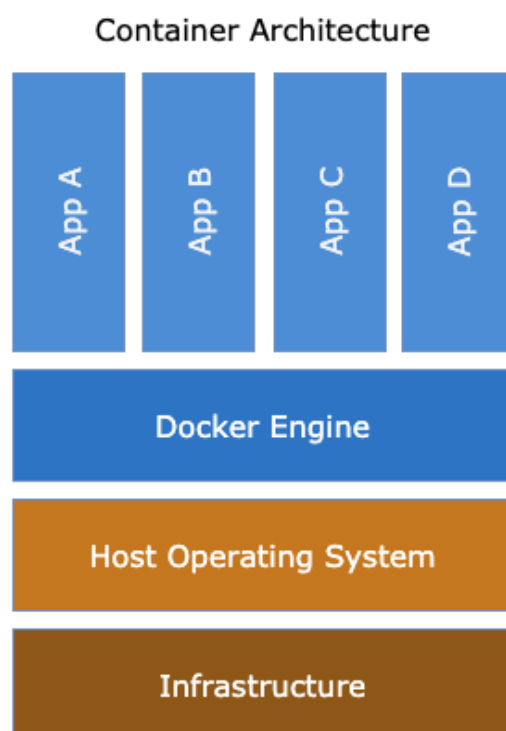


Рисунок 1.5 – Архитектура Docker [2]

Главными компонентами Docker можно считать следующие:

Образ (image)

Реестр (registry)

Контейнер (container)

Про образ и контейнер речь уже заходила, а про реестр еще нет.

Docker реестр хранит образы. Можно использовать публичные или приватные реестры для скачивания образов или же для загрузки своих образов на реестры. Самым известным публичным реестром является Docker Hub, где хранится огромное количество доступных официальных или пользовательских образов.

## **ВЫВОДЫ**

В данной главе рассматривалась технология контейнеризации с помощью Docker, которая будет использоваться для контейнеризации разрабатываемого веб-приложения.

Для контейнеризации веб-приложения будет создаваться Dockerfile для создания собственного Docker image. Из созданного Dockerfile будет создаваться контейнер, подходящий для разрабатываемого приложения, и контейнеризированное приложение будет тестироваться локально по адресу localhost на указанном порту.

## **Глава 2. ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ НА ОСНОВЕ PYTHON, FLASK И POSTGRESQL**

### **2.1 Задачи проекта**

Задачей проекта является разработка веб-приложения на базе языка Python и фреймворков Flask, SQLAlchemy Core с названием «Блог-энциклопедия еды» с следующим функционалом:

1. авторизация;
2. выкладывать записи в общий доступ;
3. комментировать и оценивать существующие записи;
4. оставлять и читать комментарии;
5. простой и нативный доступ к существующему на сайте контенту.

Веб-приложение должно состоять из нескольких страниц-категорий, базы данных MariaDB, WebAPI на базе Flask и ORM на базе SQLAlchemy Core.

Далее находятся подглавы с требованиями, которые делятся на функциональные и нефункциональные. Функциональные требования задают поведение системы, нефункциональные требования определяют характер поведения системы.

### **2.2 Функциональные требования**

Список функциональных требований к разрабатываемому веб-приложению:

Быстрая загрузка сайта.

Авторизация на сайте.

Выход из аккаунта на сайте.

Возможность просматривать страницы на сайте.

Возможность посмотреть информацию о блюде на сайте.

Сохранение оставленных на сайте комментариев, рецептов и оценок.

Редактирование своих записей на сайте.

Удаление оставленных на сайте записей.

Удаление своего профиля.

## **2.3 Нефункциональные требования**

Список нефункциональных требований к разрабатываемому веб-приложению:

Интуитивно понятный интерфейс сайта.

Простота в использовании для любого конечного пользователя.

## **2.4 Проектирование структуры и архитектуры проекта**

UML (Unified Modeling Language) — это унифицированный язык моделирования, который применяют для объектно-ориентированного анализа и проектирования.

### **2.4.1 Диаграмма вариантов использования**

Диаграммы вариантов использования, или, как их еще называют, диаграммы прецедентов, описывают такой набор действий (вариантов использования), который системы должны или могут выполнять в сотрудничестве с одним или несколькими внешними пользователями системы (участниками). Каждый вариант использования должен обеспечивать некоторый конечный результат для участников или других заинтересованных сторон системы. Описывается поведение системы с точки зрения пользователя. Соответственно, действующим лицом в диаграмме является пользователь.

В ходе разработки спецификации, которая будет использоваться при разработке веб-приложения, были составлены следующие диаграммы вариантов использования.

1. Неавторизованный пользователь заходит на сайт (Рисунок 2.1 – Диаграмма вариантов использования «Вход неавторизованного пользователя на сайт»).

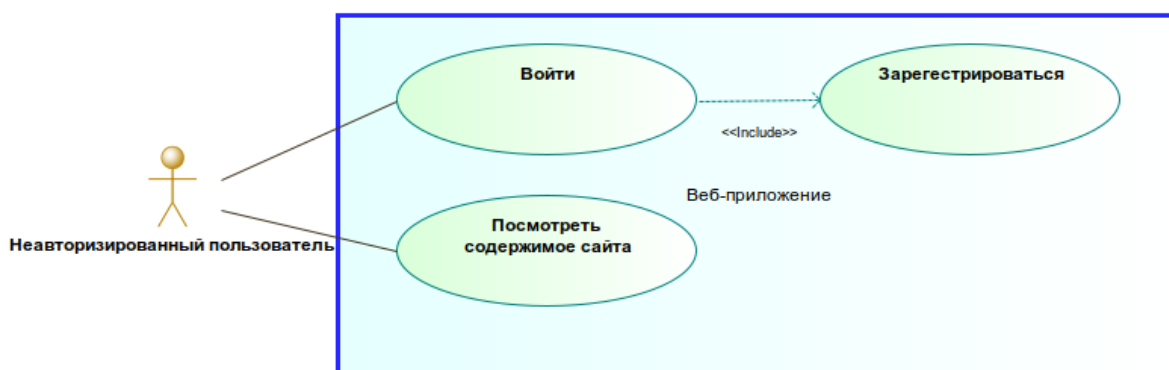


Рисунок 2.1 – Диаграмма вариантов использования «Вход неавторизованного пользователя на сайт».

2. Авторизованный пользователь заходит на сайт (Рисунок 2.2 – Диаграмма вариантов использования «Авторизованный пользователь заходит на сайт»).

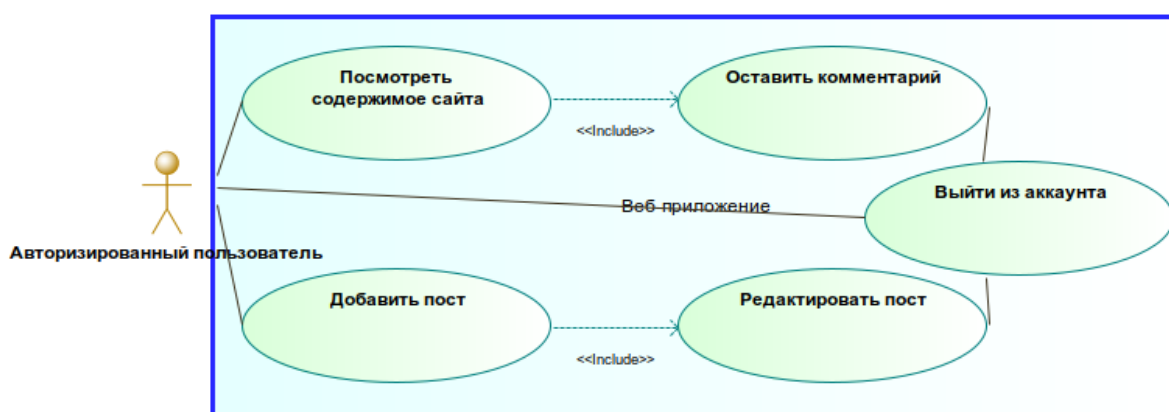


Рисунок 2.2 – Диаграмма вариантов использования «Авторизованный пользователь заходит на сайт».

## 2.4.2 Диаграммы деятельности

Диаграмма деятельности – это поведенческая диаграмма, которая описывает аспекты системы, являющиеся динамическими. Она отображает некоторые алгоритмы (или же последовательности) действий, которые можно сделать в системе. Цель диаграммы – показать возможное поведение пользователя, который хочет достичь определенной цели (произвести определенное действие) с системой.

Далее следуют некоторые диаграммы деятельности для возможных ситуаций.

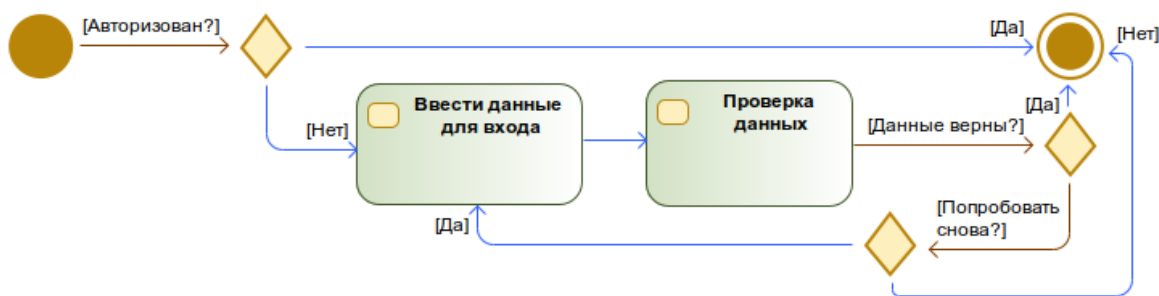


Рисунок 2.3 – Диаграмма деятельности «Вход/Регистрация».

На диаграмме деятельности «Вход/Регистрация» (Рисунок 2.3 – Диаграмма деятельности «Вход/Регистрация») предложено оптимальное количество действий для того, чтобы пользователь смог войти в аккаунт или же зарегистрироваться. Это можно объяснить следующим образом – чем быстрее пользователь сможет войти на сайт или же создать аккаунт, тем быстрее он сможет пользоваться всеми возможностями веб-приложения и создавать собственный контент. Далее представлены диаграммы, которые строились по схожему принципу.

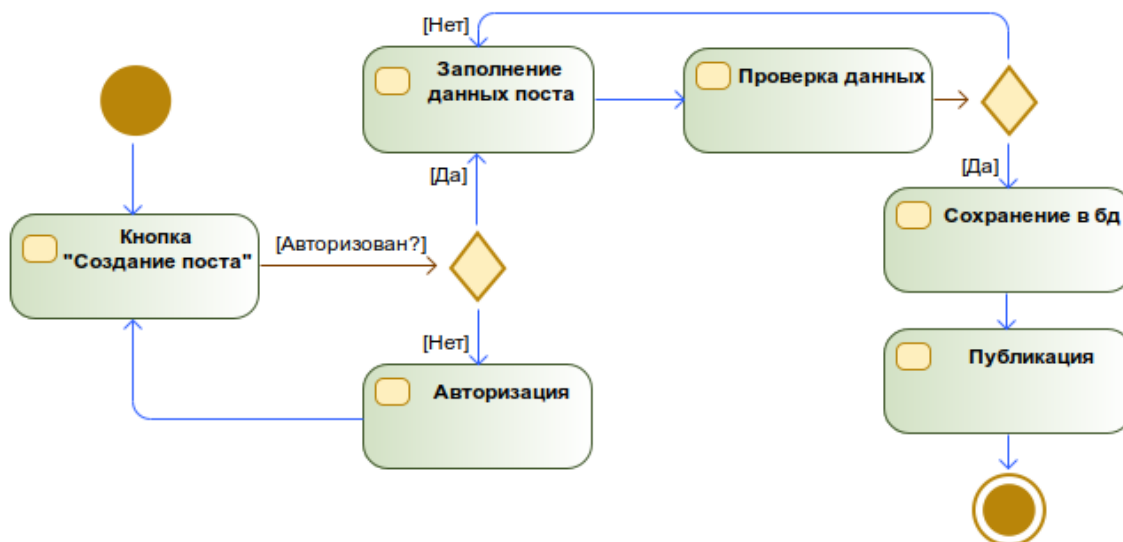


Рисунок 2.4 –

Диаграмма деятельности «Создать и опубликовать пост».

### 2.4.3 Проектирование архитектуры базы данных

База данных – основа работы веб-приложения.

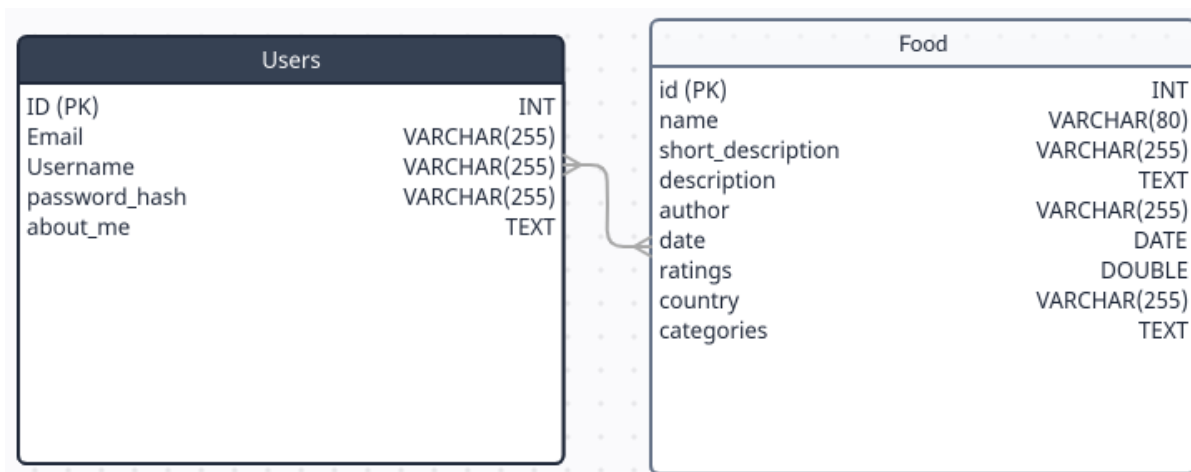


Рисунок 2.5 – Схема базы данных для веб-приложения.

Как видно из представленной на Рисунке 2.5 схемы базы данных, сама база данных состоит из двух таблиц – User и Food.

Таблица User содержит поля email – для хранения адреса электронной почты пользователя, поле username – для хранения юзернейма пользователя, который он указывал при регистрации и который указывается как его имя на сайте. Поле password\_hash используется для безопасного хранения паролей в базе данных, то есть вместо хранения пароля в чистом виде, в базу данных заносится результат вычисления хэш-функции от используемого пароля, и при аутентификации успешный вход происходит при совпадении результата хэш-функции от указываемого при входе пароля и хэша пароля, хранящегося в базе данных.

В таблице Food содержатся поля short\_description, в котором хранится короткое (одно-два предложения) описание блюда. В поле description содержится полное описание блюда, в поле author содержится юзернейм пользователя, создавшего пост о данном блюде. В поле рейтинг хранится среднее значение оценок среди всех пользователей, оставивших свою оценку. И, наконец, в поле date хранится дата создания поста.

## ВЫВОДЫ

В данной главе были составлены функциональные и нефункциональные



требования к разрабатываемому веб-приложению.

Были составлены следующие диаграммы:

Диаграммы вариантов использования

Диаграммы деятельности

Также была разработана и описана схема базы данных для веб-приложения.

## Глава 3. РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ

### 3.1 Основные элементы веб-приложения

При открытии главной страницы веб-приложения пользователь может перейти по любой доступной ссылке на другую страницу сайта.

Вверху любой страницы находится статический хедер (так называемая «шапка» сайта), содержащий ссылки на все доступные страницы сайта (см. Рисунок 3.1).



Рисунок 3.1 – Хедер сайта.



Рисунок 3.2 – Хедер сайта после авторизации.

Все доступные для перехода страницы сайта:

1. Главная (Home);
2. Список всех блюд (All dishes);
3. Блюда, отсортированные по странам (Countries);
4. Блюда по категориям (Categories);
5. Избранные блюда (Favourites);
6. Страница «About» (About);
7. Поиск по веб-приложению;
8. Кнопки «Войти» и «Зарегистрироваться»;
9. Кнопка «Профиль» для авторизованных пользователей;
10. Страница «Создать новый пост»;
11. Страница «FAQ» (Frequently Asked Questions).

Теперь вкратце рассмотрим функционал и содержимое каждой страницы по отдельности.

**Главная страница (Home)** – на главной и основной странице

веб-приложения будет отображаться список, состоящий из 20 последних добавленных блюд по дате.

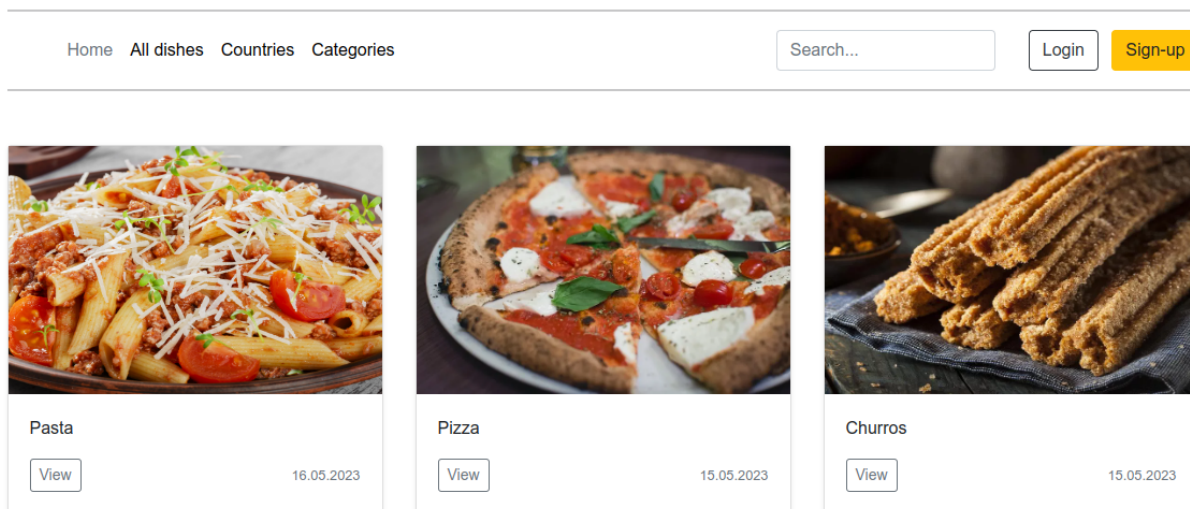


Рисунок 3.3 – Главная страница сайта.

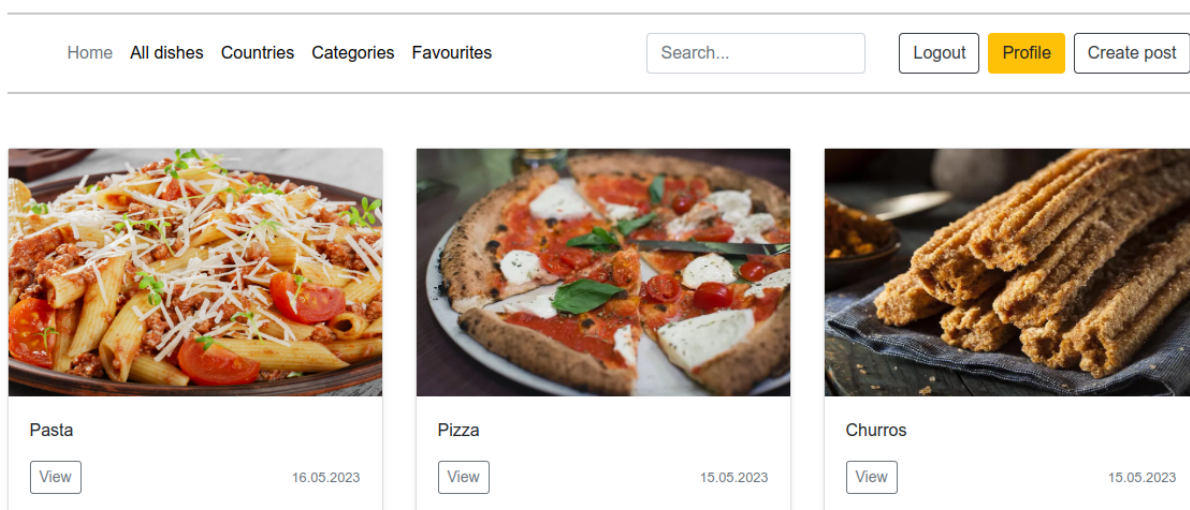


Рисунок 3.4 – Главная страница сайта после авторизации.

**Профиль (Profile)** – своя страница для каждого пользователя, которая отображает основную информацию о зарегистрированном пользователе.

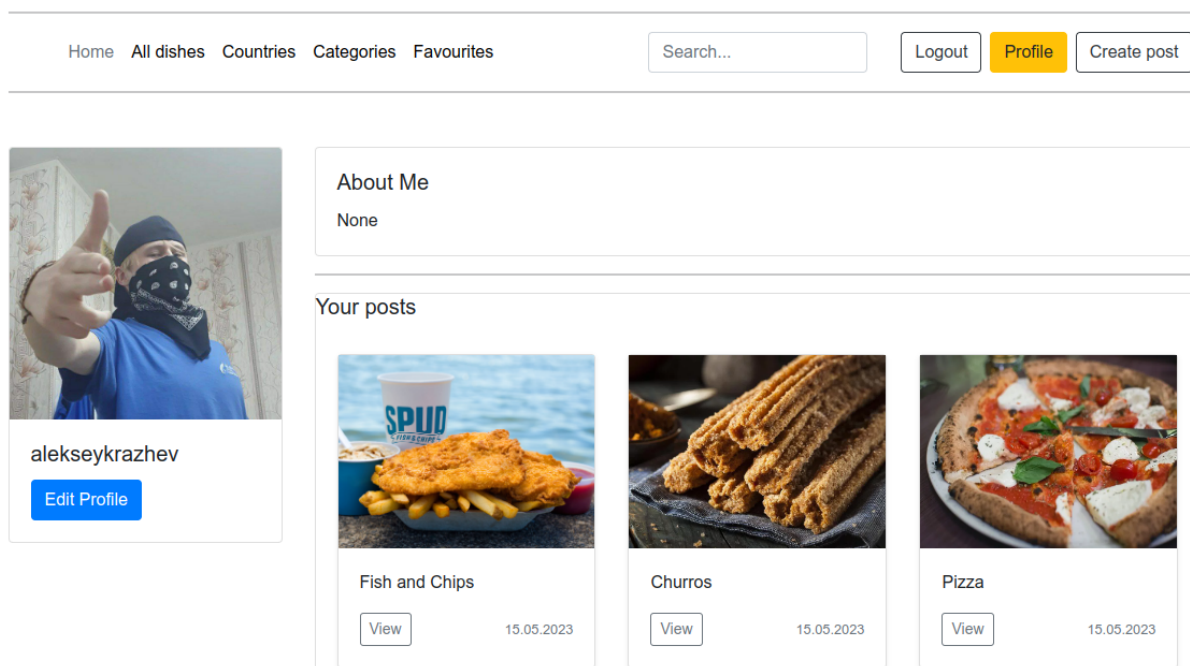


Рисунок 3.5 – Страница профиля пользователя

**Список всех блюд (All dishes)** – страница, содержащая в себе список всех блюд, добавленных пользователями на сайт.

При просмотре блюда можно нажать на интересующее вас блюдо, чтобы посмотреть подробную информацию о блюде, в том числе полное описание, рецепт, автора публикации и многое другое.

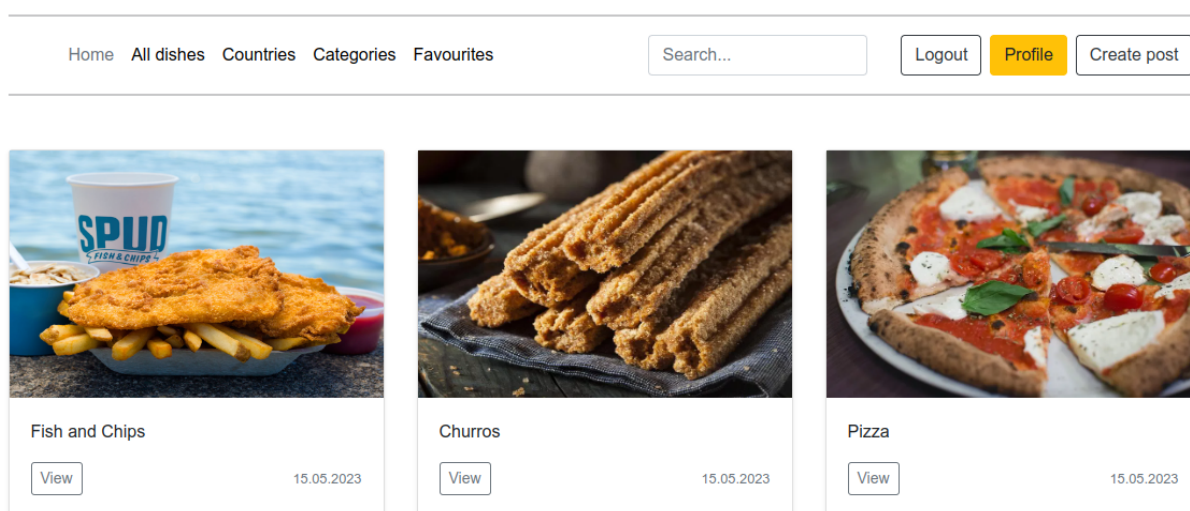


Рисунок 3.6 – Страница «Список всех блюд».

**Блюда, отсортированные по странам (Countries)** – данная страница содержит список стран, при выборе определенной страны будет выводиться

информация о традиционной в этой стране кухне, о популярных блюдах и авторах публикаций из этой страны.

## Select a country

---





	Aruba
	Afghanistan
	Angola
	Anguilla

Рисунок 3.7 – Страница выбора страны блюда.

**Блюда по категориям (Categories)** – предлагает список категорий по которым отсортированы все блюда, выложенные пользователями на сайте. При выборе определенной категории пользователь получает все блюда, которые относятся к выбранной категории.

## Select your category

---

Everlasting Classics
Spicy Food
Deserts
Main Dishes

Рисунок 3.8 – Страница выбора категории блюда

**Избранные блюда (Favorites)** – список всех блюд, которые зарегистрированный пользователь отметил с помощью функции «Добавить в избранное». Данная страница доступна только зарегистрированным

ПОЛЬЗОВАТЕЛЯМ, ВЫПОЛНИВШИМ ВХОД НА САЙТ.

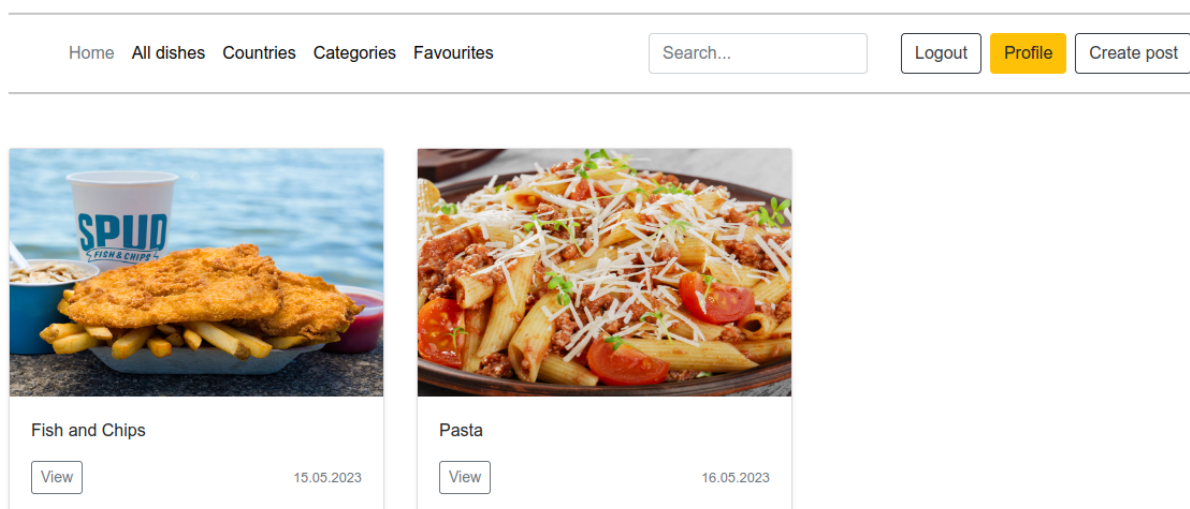


Рисунок 3.9 – Страница избранных блюд

**Страница «About» (About)** – на данной странице доступна информация о разработчиках данного веб-приложения и список часто задаваемых вопросов с ответами. Также присутствует кнопка для обратной связи с командой разработчиков для отправки информации о проблемах сайта и возможных улучшениях.

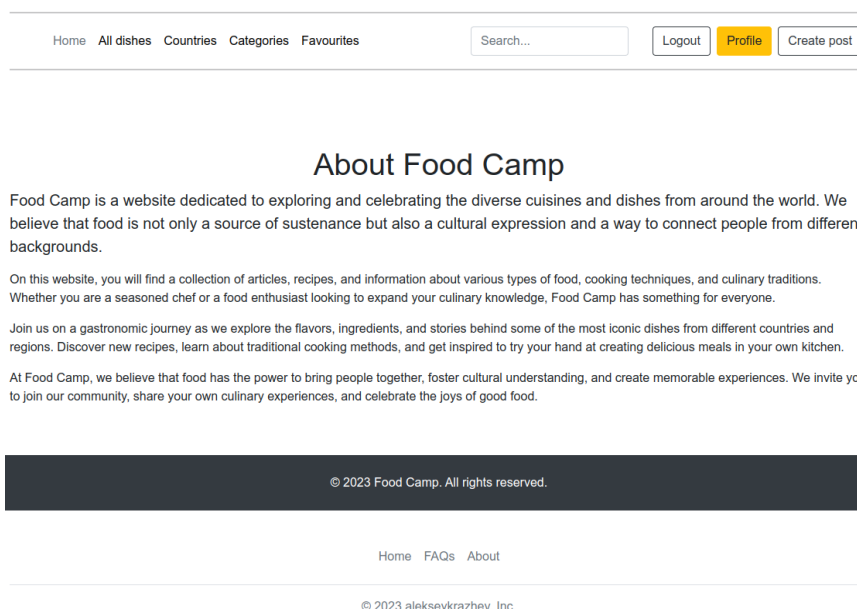


Рисунок 3.10 – Страница «About»

**Поиск по веб-приложению** – строка поиска по созданным на сайте постам для быстрого доступа к искомым материалам и информации об определенном

блюде на сайте.

A rectangular search bar with a light gray border and rounded corners. The text "Search..." is displayed inside in a light gray font.

Рисунок 3.11 – Поле поиска по сайту.

**Кнопки «Войти» и «Зарегистрироваться»** – кнопки, которые выполняют соответственно вход на сайт и регистрацию на сайте для доступа к полному функционалу сайта.

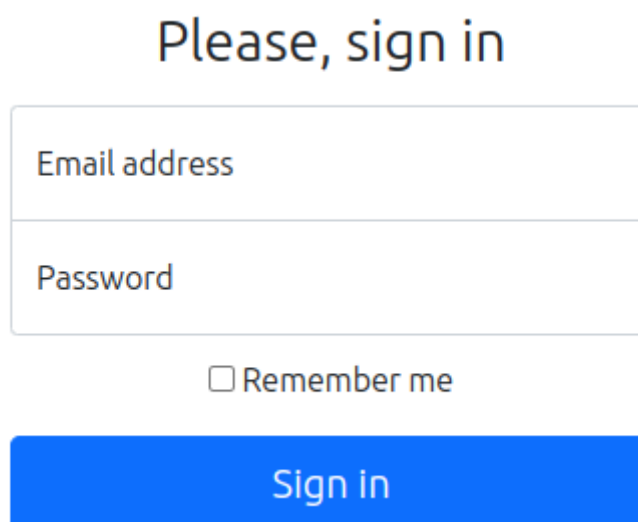
A sign-in form with the heading "Please, sign in" in a large, dark gray font. Below the heading are two stacked input fields: "Email address" and "Password". Below these fields is a checkbox labeled "Remember me". At the bottom is a blue button with the text "Sign in" in white.

Рисунок 3.12 – Окно входа на сайт

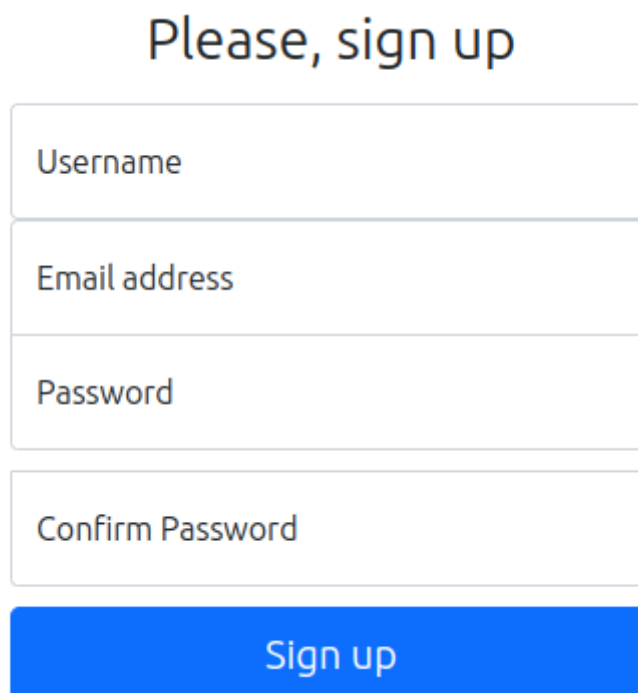
A sign-up form with the heading "Please, sign up" in a large, dark gray font. Below the heading are four stacked input fields: "Username", "Email address", "Password", and "Confirm Password". At the bottom is a blue button with the text "Sign up" in white.

Рисунок 3.13 – Окно регистрации на сайте

Также происходит проверка введенных данных с помощью регулярных выражений, и выводится ошибка в случае, если данные введены некорректно.

The image shows a web form titled "Please, sign in". It contains a text input field labeled "Email address" with the text "invalidemail" entered. A red dashed underline is under the text. Below the input field is a yellow warning message box that says: "Please include an '@' in the email address. 'invalidemail' is missing an '@'." Below the message box is a checkbox labeled "Remember me". At the bottom of the form is a blue button labeled "Sign in".

Рисунок 3.14 – Пример проверки адреса электронной почты

The image shows a web form titled "Please, sign up". Above the form fields is a section titled "Password is not secure:" followed by a list of requirements: "- Minimum length of 8 characters", "- At least one uppercase letter", "- At least one lowercase letter", "- At least one digit", and "- At least one special character (such as !@#\$%^&\*)". Below this list are four text input fields labeled "Username", "Email address", "Password", and "Confirm Password". At the bottom of the form is a blue button labeled "Sign up".

Рисунок 3.15 – Проверка защищенности пароля

**Страница «Создать новый пост»** – страница, доступная авторизованным пользователям, позволяющая создать новую публикацию на веб-сайт.



---

Home All dishes Countries Categories Favourites

Search...

Logout Profile Create post

---

## Create Food Post

Dish name

Dish name

A few words about this dish

Short description

Upload image

Choose File No file chosen

Where is this dish originally from

Select here

Select appropriate category

Select here

Description

description

Create Post Cancel

Рисунок 3.16 – Страница создания публикации

**Страница «FAQ» (Frequently Asked Questions)** – страница, позволяющая увидеть часто задаваемые вопросы, связанные с различными блюдами и не только.

---

Home All dishes Countries Categories Favourites

Search...

Logout Profile Create post

---

## Frequently Asked Questions

Question 1: What is the origin of sushi?

Question 2: What are some popular Italian dishes?

Italy is famous for its delicious cuisine. Some popular Italian dishes include pizza, pasta, lasagna, risotto, bruschetta, tiramisu, and gelato. Italian cuisine varies by region, and each region has its own specialties and traditional recipes.

Question 3: What are some famous street foods from around the world?

Question 4: What are some traditional Chinese dishes?

Question 5: What are some traditional Indian desserts?

Рисунок 3.17 – Страница часто задаваемых вопросов

## 3.2 Реализация надстройки для работы с базой данных

Для удобства работы с базой данных, использования минимального количества кода для написания SQL-запросов и исследования особенностей объектно-реляционного отображения на языке программирования Python, в ходе курсовой работы была разработана “надстройка” над библиотекой SQLAlchemy Core (ORM) в виде небольшой собственной библиотеки Python [6]. Разработанная библиотека поддерживает работу в асинхронном режиме, которая при выполнении запросов реализуется Python-библиотекой `asyncio`.

Разработанная библиотека включает в себя переопределение обычной работы с библиотекой SQLAlchemy, в частности переопределение синтаксиса составления SQL-запросов для DML операций `Select`, `Update`, `Insert` и `Delete`.

Для работы с SQL-запросом `Select` был разработан класс `SelectQuery` (Приложение А), с определением методов самых распространенных SQL операций, таких как `JOIN`, `OUTERJOIN`, `WHERE`, `GROUP BY`, `ORDER BY`, `DISTINCT`, `OFFSET`, `LIMIT`.

Для работы с SQL-запросом `Delete` разработан класс `DeleteQuery`, позволяющий выполнять базовые запросы по удалению данных из таблицы базы данных.

Для работы с SQL-запросами `Select` и `Update` был разработан класс `StatementQuery` с одинаковым набором операций для каждого запроса, позволяющий выполнять базовые запросы.

## 3.3 Хранение данных

Для хранения данных в разработанном веб-приложении используется база данных MariaDB, с таблицами `Users` и `Food`, как описано ранее. Все основные данные, такие как названия, описания и так далее, хранятся в базе данных.

В свою очередь, для хранения изображений используется Dropbox, и в частности Dropbox API и библиотека Python для работы с ним с одноименным

названием dropbox, с помощью чего возможно загружать изображения в облачное хранилище прямо в коде приложения (Приложение В). Для более быстрой загрузки уже скачанных изображений, используется библиотека Flask-cache, которая сохраняет в кэш браузера недавно загруженные картинки и индексирует их для быстрого доступа, на определенный интервал.

Для работы с базой данных и, в частности, использования ORM были созданы два класса-модели, FoodModel и UserModel (Приложение Б), в которых объявляются поля из базы данных (соответствующей таблицы), и методы для выполнения SQL-запросов, позволяющие выполнять такие запросы в асинхронном режиме.

### **3.4 Авторизация**

Для авторизации пользователя в приложении используется встроенный в фреймворк Flask модуль Flask-login, с помощью которого можно легко добавить возможность авторизации в веб-приложение.

Для работы данного модуля необходимо добавить наследование класса UserMixin в класс-модель пользователя, в моем случае UserModel, а также некоторые поля и методы, такие как is\_active и так далее. Большинство таких полей нет необходимости добавлять, так как они наследуются из класса-родителя UserMixin. Также необходимо добавить метод load\_user с соответствующим декоратором, для получения авторизованного в данной сессии пользователя.

## **ВЫВОДЫ**

В данной главе были рассмотрены:

1. Реализация основных функций веб-приложения.
2. Использованы понятия создания REST Web API.
3. Настроена работа с базой данных.
4. Реализованы все сценарии работы с веб-приложением.

5. Реализация надстройки над SQLAlchemy
6. Реализация авторизации пользователя

## **ЗАКЛЮЧЕНИЕ**

В современных реалиях обойтись без использования интернета практически невозможно. Практически все пользуются доступом в интернет каждый день, в независимости от цели – это может быть работа, чтобы как-то скрасить свой досуг, узнать какую-либо информацию или же просто пообщаться.

И тем, к чему люди обращаются каждый день, являются веб сайты, которые позволяют пользователям получать тот контент, который они хотят.

Итак, в разрабатываемом веб-приложении объединяются две вещи – простота доступа сайта и вся информация о еде. В создаваемом веб-приложении пользователям предоставляется возможность делиться рецептами своих любимых блюд, изучать традиционные блюда различных стран и культур, оценивать их, оставлять комментарии и узнавать новые рецепты прямиком с сайта.

Данное веб-приложение может быть полезным для гурманов, поваров, домохозяек (и домохозяев) или же для всех людей, так или иначе интересующихся готовкой.

В процессе выполнения работы были рассмотрены и решены следующие задачи:

1. Проектирование выполнено в курсовом проекте
2. Полный функционал веб-приложения.
3. Все страницы на сайте.
4. Использование Docker для контейнеризации веб-приложения.
5. Использование MariaDB вместо SQLite.
6. Использование своей реализации ORM на основе SQLAlchemy.

Для проектирования разрабатываемого приложения использовалась среда моделирования Modelio 5.1.

Для написания исходного кода и разработки/тестирования приложения использовалась среда разработки PyCharm Professional 2022.2.3.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Учебник «Steven F. Lott. Master Object-Oriented Python» [Электронный ресурс] — Packt Publishing, 2019 – Дата доступа 20.11.2022
2. Статья «Управление данными с помощью Python, SQLite и SQLAlchemy» [Электронный ресурс] — <https://proglib.io/p/upravlenie-dannymi-s-pomoshchyu-python-sqlite-i-sqlalchemy-2020-10-21> – 2020 – Дата доступа 05.12.2022
3. Учебник «Rehan Haider. Web API Development with Python» [Электронный ресурс] — CloudBytes, 2021 – Дата доступа 09.11.2022
4. Учебник «Hironobu Suzuki. The Internals of MariaDB» [Электронный ресурс] — interdb.jp, 2018 – Дата доступа 05.12.2022
5. Статья «Мега-Учебник Flask, часть 4: База данных» [Электронный ресурс] — <https://habr.com/ru/post/196810/> – 2013 – Дата доступа 12.12.2022
6. Статья «Пишем обертку над SQLAlchemy Core» [Электронный ресурс] — <https://habr.com/ru/company/domclick/blog/557638/> – 2021 – Дата доступа 01.12.2022
7. Статья «REST API with Flask and MariaDB» [Электронный ресурс] — <https://blog.teclado.com/first-rest-api-flask-mariadb-python/> – 2022 – Дата доступа 25.11.2022

## ПРИЛОЖЕНИЯ

### *Приложение А*

```
class SelectQuery:
    def __init__(self, model, fields):
        self._model = model
        self._from = model
        _fields = []

        self._relations = {field.class_ if isinstance(field, QueryableAttribute) else field
for field in fields}

        if model in self._relations:
            self._relations.remove(model)

        for field in fields:
            if isinstance(field, QueryableAttribute):
                _fields.append(getattr(field.class_.__table__.c, field.key))
            elif isinstance(field, sa.Column):
                _fields.append(field)
            elif hasattr(field, '__table__'):
                for f in field.__table__.c:
                    _fields.append(f)
            else:
                _fields.append(field)

        self._stmt = sa.select([model, *self._relations]).with_only_columns(_fields)

    def join(self, right, on):
        if self._from is None:
            self._from = sa.join(self._model, right, on)
        else:
            self._from = self._from.join(right, on)
```



```

        return self

    async def get(self, conn):
        if self._from is not None:
            self._stmt = self._stmt.select_from(self._from)
        result = await conn.execute(self._stmt)
        result = result.first()
        return result

    async def all(self, conn):
        if self._from is not None:
            self._stmt = self._stmt.select_from(self._from)
        result = await conn.execute(self._stmt)
        return result.fetchall()

    @property
    def raw_sql(self):
        stmt = self._stmt
        if self._from is not None:
            stmt = stmt.select_from(self._from)
        return stmt.compile(compile_kwargs={'literal_binds': True})

    def _create_model(self, model_class, data):
        model_fields = {
            getattr(field, 'name'): data[str(field)] for field in model_class.__table__.c if
            str(field) in data
        }
        model = model_class(**model_fields)
        for relation in model_class.__mapper__.relationships:
            if relation.mapper.class_ in self._relations:
        return model

```

```
class UserModel(UserMixin, Base, DB):  
    __tablename__ = 'users'  
  
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  
  
    email = db.Column(db.String(80), unique=True)  
  
    username = db.Column(db.String(100), unique=True)  
  
    password_hash = db.Column(db.String())  
  
    about_me = db.Column(db.String())  
  
    favourites = db.Column(db.String())  
  
    def __init__(self, email, username, password=None, about_me=None,  
favourites=None):  
        super().__init__()  
        self.email = email  
        self.username = username  
        self.password_hash = password  
        self.about_me = about_me  
        self.favourites = favourites  
  
    def check_password(self, password):  
        return check_password_hash(self.password_hash, password)  
  
    @staticmethod  
    async def get_by_id(id):  
        async with engine.begin() as conn:  
            await conn.run_sync(meta.create_all)  
            user = await UserModel.select() \
```

```

        .where(UserModel.id == id) \

        .get(conn)

    await engine.dispose()

    return user

@staticmethod
async def get_by_email(email):

    async with engine.begin() as conn:

        await conn.run_sync(meta.create_all)

        user = await UserModel.select() \

            .where(UserModel.email == email) \

            .get(conn)

        await engine.dispose()

    return user

@staticmethod
async def get_by_username(username):

    async with engine.begin() as conn:

        await conn.run_sync(meta.create_all)

        user = await UserModel.select() \

            .where(UserModel.username == username) \

            .get(conn)

        await engine.dispose()

    return user

@staticmethod
async def create_user(user):

```

```

async with engine.begin() as conn:

    await conn.run_sync(meta.create_all)

    _ = await UserModel.insert().values(
        email=user.email,
        username=user.username,
        password_hash=user.password_hash,
    ).execute(conn)

    await engine.dispose()

return True

@staticmethod
async def update_user(user):

    async with engine.begin() as conn:

        await conn.run_sync(meta.create_all)

        _ = await UserModel.update() \
            .values(
                email=user.email,
                username=user.username,
                password_hash=user.password_hash,
                about_me=user.about_me,
                favourites=user.favourites) \
            .where(UserModel.id == user.id) \
            .execute(conn)

        await engine.dispose()

    return True

```

```
@cache.memoize(timeout=720)
def get_image(name):
    image = None

    try:
        dbx = dropbox.Dropbox(dropbox_access_token)
    except (dropbox.exceptions.AuthError,
dropbox.dropbox_client.BadInputException):
        get_access_token()
        dbx = dropbox.Dropbox(dropbox_access_token)
    try:
        dropbox_path = f'/Apps/food-camp/{name}.jpg'
        _, response = dbx.files_download(dropbox_path)
        image_data = response.content
        image = base64.b64encode(image_data).decode('utf-8')
    except (dropbox.exceptions.HttpError, dropbox.exceptions.ApiError):
        pass

    return image
```