МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ТУРИСТИЧЕСКОЙ ПЛАТФОРМЫ-АГРЕГАТОРА «ОТКРОЙ САРАТОВ»

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Локута Ольги Олеговны

Научный руководитель
зав.каф.техн.пр.,
к.ф.-м.н., доцент _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

ВВЕДЕНИЕ

В современном мире веб-приложения стали неотъемлемой частью повседневной жизни, охватывая большинство сфер деятельности. Их популярность обусловлена удобством использования, доступностью и высокой скоростью работы. Однако разработка качественного продукта требует не только знаний в области программирования, но и понимания принципов построения архитектуры. Перед разработчиками встаёт задача выбора оптимальных инструментов и технологий, поскольку грамотный и обоснованный выбор стека позволяет существенно упростить и ускорить процесс разработки, а также повысить качество. Таким образом, актуальность темы обусловлена быстрым развитием технологий и ростом требований к качеству и скорости поставки продукта.

В рамках программы «Стартап как диплом» было принято решение о командном создании проекта «Открой Саратов». Это платформа-агрегатор, которая предоставит пользователям — туристам и жителям города — возможность быстро и эффективно находить интересные места, маршруты и мероприятия. Для таких категорий, как владельцы организаций и экскурсоводы города, решение также будет полезно благодаря функционалу размещения информации, что поможет привлечь новых клиентов.

Целью данной дипломной работы является разработка серверной части веб-приложения, которое будет выполнять функции туристической платформы, отвечая как пользовательским требованиям, так и современным стандартам надежности и безопасности.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- анализ предметной области и разработка требований;
- изучение принципов построения архитектуры современных приложений;
- выбор подходящих для данного проекта технологий и инструментов;
- контейнеризация серверной части для обеспечения удобства развертывания;
- создание базы данных и обеспечение связи с ней;
- разработка API для взаимодействия клиентского слоя с логикой сервера;
- реализация загрузки и обработки файлов изображений;
- обеспечение безопасности данных за счет аутентификации.

Структура и объем работы. Для решения поставленных задач выполнена

выпускная квалификационная работа, включающая в себя введение, 2 основные главы, заключение, список использованных источников из 22 наименований и 4 приложения. Работа изложена на 59 страницах.

Первая глава имеет название «Анализ предметной области и выбор технологического стека» и содержит основную информацию о доступных на текущий момент технологиях разработки веб-приложений и обеспечения их непрерывной работы.

Вторая глава имеет название «Реализация работы приложения», данная глава содержит подробное описание процесса выполнения работы.

Выпускная квалификационная работа заканчивается заключением, списком использованных источников, а также приложения с кодом А-Г.

Краткое содержание работы

Постановка задачи. В результате анализа рынка было принято решение о начале разработки проекта для поиска туристических мест и интересных локаций Саратова. Согласно разделению обязанностей внутри проектной команды цель данной работы — реализация серверной части веб-приложения с функционалом туристической платформы. Разработкой клиентского интерфейса платформы занималась Блохина Анастасия Дмитриевна. Экономическую модель и планируемые показатели рассчитывала Волкова Анастасия Сергеевна.

Описание требований к продукту. Функциональные требования к системе включают поддержку трех ролей: частные пользователи (поиск мест и маршрутов), организации и гиды (размещение информации о услугах). Платформа выступает посредником между этими группами. Нефункциональные требования к системе охватывают производительность, масштабируемость, высокую доступность и защиту данных в соответствии с современными стандартами.

Архитектурные подходы к разработке веб-приложений.

Клиент-серверная архитектура. В современных приложениях обычно используется клиент-серверная модель, которая предполагает разделение на две основные части:

- клиент, который посылает запросы к серверу и получает ответы;
- сервер, обрабатывающий запросы и посылающий клиенту ответы.

Модель MVC. Одним из основных подходов к построению веб-приложений является модель MVC (Model — View — Controller). Принцип модели — разделение ответственности между компонентами:

1. Модель (model) отвечает за структурированное хранение данных.
2. Представление (view) — компонент интерфейса.
3. Контроллер (controller) играет роль посредника между моделью и представлением.

Архитектурный стиль REST. Для взаимодействия между частями системы используют API (Application Programming Interface, программный интерфейс приложения). При проектировании API используется архитектурный стиль REST (Representational State Transfer, передача состояния представления), который базируется на принципах клиент-серверного взаимодействия, отсутствия состояния и единообразия интерфейсов.

Сетевой протокол HTTP. В основе REST лежит использование сетево-

го протокола HTTP (HyperText Transfer Protocol, протокол передачи гипертекста), применяющегося для обмена данными по сети. Передача информации осуществляется с использованием запросов (request) и ответов (response).

Обзор технологического стека.

Реляционные базы данных PostgreSQL. Для промышленной разработки обычно применяются реляционные базы данных, которым свойственна структурированность, минимизация избыточности и высокая целостность. PostgreSQL — реляционная СУБД с открытым исходным кодом, которая широко используется из-за своей надежности, масштабируемости и богатого функционала.

Взаимодействие с базой данных на языке Python. Для разработки приложения был выбран Python — язык программирования высокого уровня, набравший популярность за счёт простоты и обширного набора библиотек. В экосистеме Python для взаимодействия с PostgreSQL есть множество библиотек. С учётом требований к системе была выбрана библиотека asyncpg — высокопроизводительный асинхронный драйвер PostgreSQL для Python.

Фреймворк FastAPI. Одним из наиболее популярных фреймворков для создания REST API на языке Python является FastAPI. Он основан на асинхронном программировании, совместим с современными стандартами Python, предоставляет различные варианты маршрутизации и управления запросами. Передаваемые данные валидируются с помощью моделей Pydantic. FastAPI позволяет быстро разрабатывать API с соблюдением современных стандартов.

Обеспечение безопасности данных. Для безопасного хранения паролей в базе данных используется хэширование паролей. На уровне интерфейса для защиты информации необходим процесс идентификации, аутентификации и авторизации. JWT (JSON Web Token) — токен аутентификации, содержащий данные пользователя в зашифрованном виде. Токен выдается при авторизации и затем используется при каждом запросе до истечения времени жизни.

Контейнеризация с помощью Docker. Для автоматизации сборки, установки, тестирования и поставки применяется контейнеризация. Docker — платформа для контейнеризации, которая позволяет упаковывать приложения и их зависимости, реализуя все преимущества использования контейнеров: собственное окружение для частей системы, простота настройки и использования, стабильность, масштабируемость.

Пакетный менеджер UV. При использовании библиотек, не входящих в

стандартный набор, необходимы инструменты управления зависимостями. Для Python такую функциональность предоставляет пакетный менеджер UV.

Утилита Make. Make — утилита для автоматизации повторяющихся задач. Её основная функция заключается в выполнении инструкций на основе зависимостей между файлами и командами. К ключевым преимуществам Make можно отнести кроссплатформенность, универсальность и синтаксис.

Postman для тестирования API. Отладка и тестирование обеспечивают качество приложения. Postman поддерживает все основные методы HTTP, что позволяет провести полноценное тестирование RESTful-сервисов.

Реализация работы приложения.

Создание базы данных. Первый этап работы над проектом — выделение сущностей предметной области. Основные сущности-акторы включают пользователей, которые делятся на 3 категории. Акторы работают с местами, маршрутами, услугами, событиями и отзывами. Для обеспечения связей между таблицами использовались внешние ключи или отдельные связующие таблицы. Хранение изображений было реализовано в файловой системе, а в базу данных записывались пути для обращения. После описания сущностей и связей были проработаны дополнительные детали логики. Выделены ограничения атрибутов, такие как уникальность и допустимые значения. Была настроена реакция на изменения для внешних ключей и описан триггер для подсчёта рейтинга по отзывам. В результате был разработан скрипт инициализации базы данных, создающий все необходимые таблицы и автоматизирующий некоторые операции.

Описание контейнера. Чтобы организовать контейнеризацию проекта был создан docker-compose. Серверная часть содержит 2 сервиса — базу данных и сервер, между которыми необходимо организовать взаимодействие. Для хранения данных в конфигурации настроены тома (volumes). База данных использует образ, представляющий из себя базу данных PostgreSQL на Alpine Linux. Для хранения информации смонтирован том `database`, сохраняющий данные PostgreSQL, а также каталог со скриптами инициализации. Сервис бэкенда строится на основе облегченного образа на базе Alpine Linux с установленными Python 3.13 и UV.

Автоматизация сборки с помощью Make. Для упрощения работы создан ряд задач по управлению контейнерами в Makefile. Задача `start` собирает, создает и запускает контейнеры. Задача `console` запускает интерактивную обо-

лочку Alpine Linux для отладки и ручных операций. Задача `clean` обеспечивает полное очищение среды, освобождая ресурсы и восстанавливая состояние для нового развертывания. Разработанный Makefile позволяет полностью автоматизировать работу с сервисами в контейнере.

Организация программного взаимодействия с базой данных. На основе asyncpg создан класс `Database`, который инкапсулирует основные операции для управления соединениями, выполнения запросов и получения результатов. Класс сохраняет пул соединений (connection pool) для оптимизации. Выполнение запросов реализовано через методы `execute_query()`, который возвращает список словарей Python, и execute_to_model(), преобразующий элементы списка в нужную Pydantic-модель. Получившийся класс сочетает гибкость и производительность асинхронных операций со структурированным управлением соединениями и контролем типов.

Структура проекта и выделение компонентов. Файловая структура проекта выстроена следующим образом: само веб-приложение располагается в файле `app.py`. Маршрутизация описана посредством маршрутизаторов (router) для каждого компонента. Каждый набор маршрутов с одинаковым префиксом отвечает за набор операций над сущностями некоторого типа. Компонентами будут являться сущности предметной области: пользователи, персоны, организации, гиды, места, маршруты, услуги, события и отзывы. Для каждого из компонентов организован собственный экземпляр шаблона MVC, так что директории содержат 3 файла: `model.py`, `query.py` и `router.py`. Они отвечают за модель данных, запросы к базе и маршрутизацию запросов соответственно.

Основные операции над данными. Для непосредственной работы приложения была организована обработка данных для каждого компонента.

Определение моделей данных. Чтобы работать с объектами необходимых типов, были созданы соответствующие Pydantic-модели, в том числе дополнительно описаны наборы данных для создания и обновления. В теле HTTP-запроса поля собственных типов представлены как вложенные JSON-объекты.

Запросы к базе данных. В описанной структуре компонента файл `query.py` содержит текст SQL-запросов, которые понадобятся для основных операций. К ним будут относиться: считывание списка, создание новой записи, обновление или удаление данных, поиск организации по различным параметрам и прочее. С точки зрения работы с базой данных это базовые операции `SELECT`, `INSERT`,

UPDATE, DELETE. При описании запросов для asyncpg используются позиционные параметры ($1, $2 и так далее) для обеспечения безопасности, а чтобы получить созданную записи для дальнейшей работы, используется конструкция RETURNING, возвращающая затронутую операцией строку.

Разработка конечных точек для работы с ресурсами. Чтобы обрабатывать запросы к API, для каждого из компонентов определён FastAPI-маршрутизатор на основе APIRouter. Затем для маршрутизатора описана обработка конкретные запросы по конкретным путям. Например, для компонента организаций создано 6 конечных точек:

— GET / возвращает полный список организаций;
— POST / создает новую организацию по данным из тела запроса;
— GET /{id} находит информацию об организации по идентификатору;
— PUT /{id} обновляет информацию о записи;
— DELETE /{id} удаляет организацию из базы данных;
— POST /{id}/upd обновляет информацию об адресах организации.

Организация маршрутизации для всех компонентов. Были разработаны маршрутизаторы для каждого из компонентов приложения. Чтобы приложение полноценно работало, все маршрутизаторы необходимо подключить к общей точке входа. Там же инициализируется подключение к базе данных с помощью класса Database. Основная функция работы приложения main реализована в файле app.py и при запуске контейнера приложение предоставляет API, обрабатывая запросы по соответствующим адресам и возвращая ответы.

Настройка сортировки и фильтрации для удобства поиска информации. При поиске информации о маршрутах, местах, гидах и организациях пользователю целесообразно фильтровать списки по наиболее актуальным для него параметрам, а также отбирать категории мест и организаций. Для этого использованы опциональные query-параметры sort для сортировки и category для категории. Так как иногда необходимо совмещать сортировку и фильтрацию, то сортировка реализовна на уровне базы данных, а фильтрация по категориям на уровне сервера.

Работа с бинарными файлами изображений. В компоненте uploads реализована работа с изображениями. Сами изображения хранятся в файловой системе, что минимизирует архитектурную сложность. Чтобы FastAPI-приложение также видело директорию, в файле app.py создаётся директория и устанавлива-

ется связь с ней. Приём файлов организуется как обработка multipart/form-data. После проверки типа данных библиотека aiofiles обеспечивает неблокирующую запись на диск, а в базе данных сохраняется название файла. Для получения изображений реализована обработка GET-запроса по id изображения. Преимущества такой реализации — оптимизация базовых запросов, снижение нагрузки на базу данных и возможность дальшейшего подключения файлового хранилища.

Авторизация и защита ресурсов. Для защиты данных необходимо реализовать механизм авторизации. Работать с данными должен только зарегистрированный и авторизованный пользователь, а также не должно быть возможности изменять «чужие» данные. Был создан отдельный компонент для работы с токенами авторизации и разработан собственный класс `JWTBearer` на основе `HTTPBearer`, который содержит данные пользователя и срок действия. Реализованы методы шифрования и дешифрования токена, выделения идентификатора и роли пользователя.

Для получения токена описан запрос POST по маршруту `/auth`, которая принимает данные для входа и выдает токен для дальнейшего взаимодействия с API. После реализации работы с токенами организована защита всех маршрутов. Для некоторых конечных точек важны конкретные данные пользователя, а важен лишь факт аутентификации. А для маршрутов, изменяющих данные, должна быть добавлена проверка ролей, а также проверки `user_id`.

Тестирование разработанного API. Чтобы удостовериться в корректном функционировании серверной части приложения, было проведено тестирование реализованных маршрутов и доступных методов с использованием Postman. Для этого был локально запущен контейнер и отправлялись запросы на `localhost:7777`.

— **Создание пользователя и авторизация в системе.** Сначала нужно создать учетную запись (POST-запрос на создание организации и соответствующего пользователя), затем осуществить вход (POST-запрос с учётными данными) и получить токен аутентификации. При передаче корректного токена пользователь считается авторизованным и получает доступ к ресурсам приложения.

— **Основные операции над организациями.** На примере компонента организаций доступно получение информации (GET) как о всех организаци-

ях, так и о конкретной по `id`. Список можно получить как в порядке по умолчанию (без параметров), так и с заданным принципом сортировки в соответствии с параметром `sort` . Для обновления (PUT) или удаления (DELETE) проверяется, что пользователь представляет эту же организацию (проверка `user_id`) в токене).

Аналогично были протестированы и другие компоненты. После реализации бэкенда стала возможна разработка клиентского интерфейса приложения, и в результате был создан полноценный продукт.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были решены следующие задачи:

- проведен анализ предметной области и выдвинуты требования к продукту;
- изучены принципы и практики построения архитектуры современных приложений;
- подобраны технологии и инструменты, удовлетворяющие требованиям к проекту;
- создана база данных для хранения необходимой информации;
- обеспечена контейнеризация базы данных и серверного слоя для оптимизации управления зависимостями и повышения удобства развертывания;
- организована возможность взаимодействия с ресурсами приложения посредством REST API;
- реализовано хранение, загрузка и получение изображений;
- обеспечена безопасность данных за счёт аутентификации.

Таким образом, все поставленные в рамках бакалаврской работы задачи были выполнены.

В современных условиях быстрого развития как информационных технологий, так и интереса к региональному туризму очевидны актуальность, востребованность и перспективность предлагаемого решения. Возможными дальнейшими шагами развития проекта можно назвать проведение исследования для выявления возможностей повышения удобства и улучшения функциональности, запуск и выход на рынок, проведение переговоров о партнёрстве с другими культурными и туристическими программами по развитию региона.

Проект «Открой Саратов» был представлен экспертной комиссии по программе «Стартап как диплом». После слушания и защиты эксперты дали команде советы по возможным вариантам усовершенствования различных аспектов реализации идеи, а также вынесли положительное заключение о данном проекте с точки зрения его задумки и перспективности в существующих условиях.

В результате разработки был создан продукт, выполняющий заявленные функции туристического агрегатора.

Платформа имеет потенциальные направления развития. К ним можно отнести расширение функционала, такое как интеграции с сервисами карт и платёжными системами, а также внедрение искусственного интеллекта для предоставления пользователям возможностей более эффективного достижения своих

целей.

Тем не менее, существующее решение на этапе MVP уже предоставляет пользователям, в числе которых могут быть как частные лица, так и организации или экскурсоводы, такие полезные возможности, как поиск и сохранение мест и маршрутов, а также размещение информации о себе и своих услугах для представителей бизнеса и культуры.