Пояснительная записка к проекту: “Создание многостраничного сайта для реализации функционала, создания и редактирования рецептов”.

Работали над проектом:

Шафоростова Елизавета

И Дунаев Глеб

Г. Москва 2025г.

Оглавление

1. Введение
2. Основная часть
   1. Теоретическая часть
      1. Планирование работы
      2. Составление базовых требований, целей, задач сайта
      3. Архитектура проекта
      4. Технологический стек и его обоснование
      5. Алгоритм обработки формы
   2. Практическая часть
      1. Дизайн-макет
      2. Создание базы данных и взаимодействие с ней
      3. Разработка серверной части
      4. Разработка клиентской части
      5. Развёртывание проекта через docker
      6. Выкладка сайта на хостинг
      7. Дальнейшая поддержка
3. Заключение
4. Литературные источники
5. Приложение
6. Введение.

Разработка многостраничного веб-сайта для реализации конструктора различных рецептов, в котором мы предоставляем для пользователей возможность создавать свои уникальные рецепты, а также выкладывать их в ленту.

Цель: создание многостраничного сайта для предоставления пользователям возможности создания своего рецепта посредством заполнения формы и ее обработки при помощи различных технологий и алгоритмов.

Задачи:

* Планирование
* Составление базовых требований, целей, задач сайта
* Выбор технологий
* Выбор архитектуры для реализации проекта
* Создание дизайн-макета
* Реализация серверной и клиентской частей и последующее соединение
* Создание архитектуры БД и проработка взаимодействия с ней
* Выкладка продукта на хостинг
* Развертывание проекта

Сроки реализации: конец марта – начало мая 2025 г.

2. Основная часть

2.1. Теоретическая часть

2.1.1. Планирование работы

В первую очередь, был сформирован примерный план работы. Получили:

1. Наладить совместную работу
2. Создание описания, ТЗ к проекту
3. Создать репозиторий git
4. Проанализировать дизайн макеты других сайтов
5. Подобрать палитру для сайта
6. Продумывание наполнения формы – основной точки притяжения сайта
7. Попутно, сбор информации, формирование алгоритма для обработки формы для рецепта
8. Создание дизайн-макета
9. Создание базовой архитектуры проекта
10. Создание общего функционала
11. Создание визуальной наполняющей
12. Соединение серверной и клиентской частей
13. Развертывание через docker
14. Тестирование
15. Запуск сайта на хостинг сервисе

2.2.2. Составление базовых требований, целей, задач сайта

На данном этапе было составлено ТЗ к проекту. Самыми важным было:

- Требование к:

1. Функционалу: аутентификация, анализ, обработка и сбор информации из форм, поддержка клиентской части, маршрутизация, уведомление и рассылка, привлечение.
2. ПО: PyCharm Pro, Pg Admin, Figma, Git, Docker Desktop.
3. Эксплуатации: желательно экран с периферийными устройствами для ввода информации (клавиатура и мышь).
4. Требование к адаптации: не предусмотрена.
5. Требование к масштабируемости: проект должен спокойно, максимально эффективно быть масштабируем.
6. Требование к безопасности: использование алгоритма шифрования HS-256 и секретного ключа, использование токенов и куки. Формирование html на данных после запроса на сервер, перезагрузка страницы.
7. Требование к навигации: понятная простая, интуитивна. Основной принцип – достижимость по уровню доступа, где администратор получает высший доступ.
8. Требования к дизайну: минимализм, монохром.
9. Требование к документации: хотелось бы очень ее иметь.
10. Требование к архитектуре: монолитная архитектура.
11. Требование к сохранности файлов: взаимодействие через. сервер. Локально хранятся самые важные константы в .env.

-Также стоит упомянуть о целях и задачах, что преследует и выполняет, соответственно, наш сайт:

1. Привлечение клиентов.
2. Создание, распространение и сохранение рецептов.
3. Создание внутренней среды для распространения своих рецептов, возможно, идей и взглядов.
4. Коммуникация поддержки и пользователей.
5. Привлечение пользователя сети интернет к регистрации на нашем ресурсе.

-Основная целевая аудитория:

1. Посетители.

2. Пользователи.

3. Другие лица.

- Разграничение доступа: Информация, размещаемая на сайте, является общедоступной. Пользователей сайта можно разделить на 2 части в соответствии с правами доступа:

* 1. Посетители;
  2. Администратор;
  3. Зарегистрированные пользователи;

Посетители имеют доступ только к общедоступной информации;

Зарегистрированные пользователи имеют доступ к сохранению и распространению своих рецептов, также весь функционал посетителей;

Администратор имеет доступ к БД и мет по своему усмотрению редактировать контент, выкладываемый, делать рассылки, анонсы, объявления, делать предупреждение и блокировку пользователей. Также доступен весь функционал зарегистрированного пользователя. Доступ к администрированию предоставляется владельцем или администратором высшего уровня при помощи выдачи временного ключа-токена.

2.1.3. Архитектура проекта

Наша архитектура состоит из двух частей: backend и fronted. Для начала:

I. Общая структура проекта:

project\_name/

├── backend/ # Серверная часть (FastAPI)

├── frontend/ # Клиентская часть (JS, CSS, HTML)

├── docs/ # Документация (опционально)

├── tests/ # Тесты (опционально)

├──. env # Переменные окружения

├──. gitignore # Файлы, которые нужно игнорировать в Git

├──. dockerignore # Файлы, которые нужно игнорировать в Docker file

├── Dockerfile # файл для развертывания при помощи Docker

├── docker-compose # Файл надстройки для развертки сервиса

├── alembic.ini # Установочный файл миграции

├── /migrations # Каталог для миграции

├── README.md # Описание проекта

└── requirements.txt # Зависимости Python

Структура frontend (JS, CSS, HTML):

frontend/

├── base.html # Основной HTML-файл

├── css/

│ ├── style.css # Основной файл стилей

│ └── ...

├── js/

└──base.js # Основной JavaScript-файл

Структура backend:

backend/

│

├── app/ # Основное приложение

│ ├── base/ # Базовые модули

│ │ ├── constant.py # Константы проекта

│ │ ├── database.py # Настройки и подключение к БД

│ │ ├── mail.py # Работа с почтой

│ │ └── service.py # Общие сервисные функции

│ │

│ ├── bots/ # Модули для ботов

│ │ ├── main.py # Основной бот

│ │ ├── main\_2.py # Второстепенный бот

│ │ ├── model.py # Модели данных для ботов

│ │ └── service.py # Сервисные функции для ботов

│ │

│ └── web/ # Веб-часть приложения

│ ├── admin/ # Админ-панель

│ ├── auth/ # Аутентификация

│ ├── notification/ # Уведомления

│ ├── pages/ # Страницы

│ ├── recipes/ # Работа с рецептами

│ ├── support/ # Поддержка

│ ├── tests/ # Тесты

│ ├── users/ # Пользователи

│ ├── view/ # Представления (views)

│ ├── error.py # Обработка ошибок

│ └── main.py # Основной веб-модуль

│

├── migrations/ # Миграции базы данных

│ └── env.py # Настройки Alembic

│

├── tests/ # Глобальные тесты

│

├── .env # Переменные окружения

├── README # Описание проекта

├── script.py.mako # Шаблон для миграций

├── alembic.ini # Конфигурация Alembic

├── docker-compose.yml # Docker Compose

├── Dockerfile # Docker-конфигурация

├── requirements.txt # Зависимости Python

└── run.py # Точка входа в приложение

Вся структура построена именно так, потому что так правильно).

2.1.4. Технологический стек и обоснование

Данный раздел посвящен выбору технологий. Так выбирались они из соображения, что это монолитное приложение, которое в будущем можно считать среднего уровня нагруженности.

1. Язык программирования:

* Python – основной язык backend-разработки.
* Используется для создания API, работы с базой данных, логики ботов и веб-интерфейса.
* JS – работа с DOM и событийностью

2. Веб-фреймворки:

* Flask – микрофреймворк для создания API и веб-приложений.
* Нужен для обработки HTTP-запросов, маршрутизации и интеграции с другими компонентами.
* Flask-RESTful расширение для создания RESTful API.
* Flask-WTF– интеграция форм с Flask.

3. Работа с данными

* SQLAlchemy – ORM для работы с базами данных.
* Alembic– инструмент для миграций БД.

4. Аутентификация и безопасность

* python-jose– работа с JWT-токенами.
* Используется в модуле auth/ для аутентификации пользователей.
* WTForms– валидация форм.
* Нужен для регистрации, входа и других форм в web/auth/.

5. Взаимодействие с внешними сервисами

* requests– HTTP-запросы к внешним API.
* Может использоваться в support/ или notification/ для интеграций.
* selenium– автоматизация браузера.

7. Деплой и инфраструктура

* docker-compose.yml, Dockerfile – контейнеризация приложения.
* Для развёртывания в изолированной среде (БД, бэкенд, боты).

2.1.5. Алгоритм обработки формы

На данном этапе был продуман алгоритм по обработке формы, получили:

-Разделы формы для обязательной обработки:

1. Наименование;
2. Тема;
3. Категория;
4. Описание;
5. Категория;
6. Ингредиенты;
7. Этапы рецепта;
8. Результат;

-Также необязательные поля для заполнения:

1. Хэш теги;
2. Ссылки;
3. Картинки;
4. Использование нейросетей;

-Порядок и форма обработки:

1. После произведение события клика мыши происходит сбор информации, получаем json file;
2. Далее данный файл обрабатывается нейросетями (Опционально);
3. После происходит адаптация к макету, заведомо созданному;
4. Далее пользователю позволяют сохранить рецепт в формате: PNG, HTML, PDF.
5. Сам же рецепт сохраняется на сайте в БД для дальнейшего отображения в виде HTML файла;

2.2 Практическая часть

2.2.1. Дизайн-макет

На данном этапе, основываясь на базовых характеристиках, что были заложены в начале, было произведена работа по созданию дизайн макета.

Во-первых, данный этап помимо главной цели этапа – дизайна включал еще и цель согласования навигации сайта.

В целом, сначала были определены фреймы с названиями и размерами данного, названия соответствуют их содержанию и смыслу. Далее были созданы макеты для оценки сочетаний палитры. После созданы тестовые секции блоков. Оценка и согласование. И так несколько раз, пока не получили требуемый объем дизайн-блоков.

2.2.2. Создание базы данных и взаимодействие с ней

Название этапа провоцирует боль везде. Для начала был план:

1. Создание Postgres сервера и БД c именем Recipe maker
2. На основе логики навигации, полученной во время прошлого этапа, была реализована архитектура БД, что включает в себя 8 таблиц:

* Users – здесь храниться вся информация о пользователях;
* Usertokens – токены пользователей;
* Texts – различные тексты;
* Recipes – вся информация о рецептах пользователей;
* Pages – страницы сайта;
* Images – все изображения;
* Alembic version - …
* Ai data - …

1. Далее следовал процесс настройки миграции БД при помощи библиотеки alembic:

* Была создан каталог /migrations
* Был отредактирован файл конфигурации для осуществления процесса

1. Позже был написан общий сервис на SQL Alchemy для работы с БД, который включал базовый функционал, который мог наследоваться и переопределяется.
2. Написание наследованных от BaseServvice класса своих TableNameService классов.

На этом данный этап заканчивается. В итоге мы получили полную удовлетворяющею нашим запросам структуру для хранения данных.

* + 1. Разработка серверной части

И вот и серверная часть. Этап, что привел наш проект в действие. Основные этапы реализации:

1. Реализация структуры проекта (см. структура backend выше);
2. Установка требуемых зависимостей и библиотек;
3. Прописывание констант и секретов, в constant.py и .env файлах соответственно;
4. Прописывание. gitignore файла;
5. Прописывание app/web приложения, где main.py отвечает за Flask приложение, что включает в себя несколько BluePrint, что представляют собой сервисы приложения. Также есть каталог статической наполненности и каталоги для осуществления работы сервиса или работы с таблицей БД;
6. Создание соединения с нейросетями: YANDEXGPT, GIGA CHAT, YANDEXART;
7. Настройка взаимодействия, создание ботов для работы с данными API.
8. Настройка почты сайта: регистрация аккаунта, получение пароля, установочных данных. Прописывание логики отправки, удаление, оповещения, просмотра почты;
9. Соединение, централизация нашего монолитного приложения в единый файл run.py;
10. Прописывание файла для обработки ошибок;

Как итог, получили серверное приложение, пока что без особого интерфейса. В дальнейшем вносились коррективы в функционал серверной части.

* + 1. Разработка клиенткой части

Самая эстетичная часть проекта, не считая разработки макет-дизайна. Включает в себя:

* Верстку макетов;
* Шаблонизацию кода;
* Работу с событийностью;
* Работа с информации со стороны клиента;
* Работа со стилями и отображением динамического контента;
* Классовую структуру для отображения блоков информации и шаблонизации;
* Вызов ошибок;
* Навигацию;

Основная реализация ложится в большей степени на Python, а именно FLASK-WTF, jinja, что естественно облегчает использование JS. Из важного присутствуют своих классы для отображения определенной информации, также есть функции для обработки определенных событий.

* + 1. Развертка при помощи Docker

Неприятий этап проекта, но очень нужный и полезный. Он позволил разом запускать нам БД и веб сервис. Собственно, он для этого и создан. В идеале процесс для совершения развёртки таков:

* Скачивание DockerDesktop;
* Проверка присутствия виртуализации в настройках BIOS;
* Проверка установленного образа linux для windows
* Создание .dockerignore, DockerFile, docker-compose.yml файлов;
* Добавление команд для сборки и финальной развёртки приложения;(DOCKERFILE)
* Приписывание базовой информации в doccker-compose.yml;
* Начало развертки запуском команды в cmd

Потрепал же нервы этот нам DOCKER

* + 1. Выкладка сайта на хостинг

На данном этапе проекта производилось достаточно массивное тестирование всего функционала, для дальнейшего переброса проекта в общий доступ. Для своих целей мы выбрали хостинг: \*\*\*. Нам требовалось для установки нашего предложения там:

* Запустить там docker;
* Поправить чуток DockerFile для создания пользователя минимального доступа из соображения безопасности;
* Перекинуть все файлы внутри проектной папки
* Настроить зависимости;
* Настроить доменное имя;
* Узнать порты;
* Запустить сборку;
* Проверить работоспособность;

Как итог, сайт выложен на хостинг \*\*\* и работоспособен.

2.2.7. Дальнейшая поддержка

На данный момент наш проект имеет не узкопрофильную направленность тематики. Хотелось бы:

* Провести рефакторинг кода;
* Провести максимальное тестирование функционала и его откладку;
* Настроить логи в коде;
* Доработать интерфейс;
* Создать более структурированную систему доступа к контенту;
* Добавить приватный доступ к определенной доле функционала;
* Оптимизировать работу БД;
* Добавить кэширование;
* Создать функции для резервного копирования и переноса БД;
* Усилить безопасность, проверив код на косяки;

1. Заключение

Цель проекта достигнута. В какой-то степени сайт соответствует заявленному ТЗ и, моментами, превышает запланированный масштаб работ и ожидаемых результатов. Задачи выполнены все.