**Анализ и сравнение написания web-приложений с использованием разных фреймворков**

**Выполнил студент 72 потока:**

**Фуников Георгий Дмитриевич**

**2024**

**СОДЕРЖАНИЕ**

1. Введение……………………………………………………………………..3  
 1.1 Обоснование выбора темы……………………………………….……..3  
 1.2 Определение цели и задач исследования...…………………………….3  
2. Обзор популярных инструментов для разработки веб-приложений на Python…………………………………………………………………………...4  
 2.1 Django…………………………………………………………………….4  
 2.2 Flask………………………………………………………………………6

2.3FastAPI…………………………………………………………………….8  
3. Проектирование приложения……………………………………………….10  
4. Разработка в соответствии с созданной документацией…………………..20  
5. Анализ и интерпретация результатов……………………………………….22  
6.Заключение…………………………………………………………………….27

**1.ВВЕДЕНИЕ**

**1.1Обоснование выбора темы:**

Разработка веб-приложений становится все более сложной по мере того, как растут требования к функциональности и производительности. Фреймворки веб-приложений предоставляют структурированные подходы к разработке веб-приложений, что упрощает и ускоряет процесс разработки.  
Этот проект посвящен анализу и сравнению различных фреймворков веб-приложений, таких как Django, FastAPI, Flask.

**1.2Определение цели и задач исследования:**

**Цель:**

Анализировать и сравнивать написание веб-приложений с использованием Django, Flask и FastAPI.

**Задачи:**

1. Разработать простые веб-приложения с использованием Django, Flask и FastAPI.
2. Провести сравнительный анализ этих приложений
3. Сделать выводы о преимуществах и недостатках каждого фреймворка и определить ситуации, в которых они наиболее подходят.

**2. Обзор популярных инструментов для разработки разработки веб-приложений на Python**

**2.1 Django**

Django — это высокоуровневый фреймворк с открытым исходным кодом, написанный на языке программирования Python, который используется для создания веб-приложений и веб-сайтов.

Основные особенности Django:

* **Быстрая разработка**. Фреймворк предоставляет встроенные инструменты и библиотеки, которые позволяют быстро создавать функциональные веб-приложения.
* **Поддержка концепции повторного использования кода**. Уже написанные модули и компоненты можно использовать в разных проектах, что значительно ускоряет процесс разработки.
* **Безопасность**. Фреймворк включает в себя множество встроенных механизмов безопасности, таких как защита от SQL-инъекций, межсайтового скриптинга и подделки межсайтовых запросов.
* **Масштабируемость**. Фреймворк разработан с учётом масштабируемости, что позволяет легко расширять приложение по мере роста его популярности. Он поддерживает различные базы данных и может быть развёрнут на различных платформах.
* **Полный стек инструментов**. Фреймворк предоставляет полный стек инструментов для разработки веб-приложений, включая ORM, систему шаблонов, маршрутизацию URL и админ-панель.
* **Портативность**. Фреймворк позволяет легко переносить приложения между различными платформами — например, с локального сервера разработки на облачный хостинг.
* **Универсальность**. Фреймворк подходит для разработки самых разнообразных проектов, от небольших блогов до крупных корпоративных систем.

Области применения фреймворка Django:

* **Создание новостных сайтов и платформ для дистрибуции контента**.  Фреймворк подходит для создания информационных порталов, онлайн-версий печатных журналов или других цифровых СМИ.
* **Разработка CMS**. На Django создают системы управления контентом, на которых работают админки сайтов.
* **Создание CRM**. С помощью фреймворка разрабатывают системы управления процессами в организации и взаимодействия с клиентами.
* **Разработка сервисов для бронирования**.
* **Создание корпоративных порталов и платформ для ведения документооборота**.
* **Разработка алгоритмических генераторов и платформ для анализа данных**, систем фильтрации и машинного обучения.

**2.2 Flask**

Flask - это микрофреймворк. У него нет инструментов и библиотек, характерных для других фреймворков. Вместо них данное программное обеспечение оснащено рядом базовых возможностей. Расширение функциональности «приложения» the Flask осуществляется за счет установки дополнений. Это минималистичное программное обеспечение для быстрой разработки.

Основные особенности Flask:

* **Минимализм**. Flask предлагает компактный набор инструментов для разработки сайтов, что делает его идеальным фреймворком для начинающих разработчиков.

* **Гибкость.** Модульность Flask позволяет разработчикам настраивать веб-приложения в соответствии со своими потребностями
* **Расширяемость**. С помощью плагинов в Flask можно интегрировать любые необходимые функции.

* **Связь с базами данных**. Flask-приложения можно связывать с различными базами для работы с датасетами.

* **Микроподход**. Flask позволяет максимально быстро делать веб-приложения на базе одного файла с минимальной функциональностью.

* **Тестирование**. Проверку и отладку решений Flask осуществляет с помощью фреймворка pytest.

* **Профессиональное комьюнити и служба поддержки**. Flask популярен среди программистов, вокруг него большое сообщество Flask-разработчиков, готовых помочь начинающим.
* **Эффективность**. Flask оптимизирован для производительности и может обрабатывать множество запросов быстро и эффективно.

Области применения фреймворка Flask:

Flask – конструктор, используемый разработчиками в самых разных сферах. Официальное сообщество разрабатывает спектры расширений. О том, что они относятся к Фласк, указывает приставка the flask: flask-sqlalchemy, flask-login и так далее.

Данный инструмент лучше всего использовать для небольших проектов и website, работающих со статическим контентом. Соответствующий микрофреймворк отлично подходит для формирования микросервисов.

**2.3 FastAPI**

FastAPI — это фреймворк для создания лаконичных и довольно быстрых HTTP API-серверов со встроенными валидацией, сериализацией и асинхронностью,  
что называется, из коробки. Стоит он на плечах двух других фреймворков: работой с web в FastAPI занимается Starlette, а за валидацию отвечает Pydantic.

Основные особенности FastAPI:

* **Безопасность типов и проверка**. FastAPI использует подсказки по типам Python и модели Pydantic для автоматической проверки данных и сериализации. В результате получаются типобезопасные API, в которых ошибки обнаруживаются во время компиляции, а не во время выполнения.

* **Высокая производительность**. FastAPI работает со скоростью, сопоставимой с фреймворками Node.js и Go, благодаря эффективному использованию современных функций Python.

* **Автоматическая интерактивная документация API**. FastAPI автоматически генерирует документацию API (Swagger UI и ReDoc) на основе подсказок по типу кода и моделей Pydantic.

* **Система внедрения зависимостей**. FastAPI имеет надёжную систему внедрения зависимостей, которая упрощает управление сложными зависимостями, такими как подключения к базе данных, аутентификация и внешние службы.

* **Асинхронная поддержка**. FastAPI изначально поддерживает асинхронное программирование, позволяя писать асинхронные обработчики маршрутов и в полной мере использовать синтаксис async/await в Python для операций ввода-вывода.

* **Аутентификация и авторизация**. FastAPI предоставляет встроенную поддержку различных методов аутентификации, включая токены JWT, OAuth2 и проверку ключа API.

* **Сериализация и синтаксический анализ данных**. FastAPI обрабатывает автоматическую сериализацию объектов Python в ответы JSON. Он также может анализировать данные входящего запроса, включая параметры запроса, тело запроса (JSON, данные формы) и заголовки, на основе подсказок по типу.

**Области применения фреймворка FastAPI**:

* создание высокопроизводительных API и микросервисов;

* разработка высоконагруженных систем, требующих обработки большого количества запросов одновременно;

* создание защищенных API благодаря встроенной поддержке аутентификации и авторизации;

* интеграция с базами данных, такими как SQLAlchemy и ORM, что упрощает взаимодействие с ними и управление данными;

* реализация популярных функций API, таких как загрузка файлов, валидация входных данных, маршрутизация.

**3. Проектирование приложения**

**Пример работы приложения на Django**

Для начала нам нужно установить Django, его можно установить в командной строке с помощью команды pip install Django (рис.1.1)



(рис.1.1)

Чтобы начать работу с проектом нужно прописать в командной строке следующую команду django-admin startproject Django (рис.1.2)



(рис.1.2)

Дальше нам надо создать приложение в директории Django. Воспользовавшись командой cd (имя вашей директории) переходим в нее и прописываем в командной строке следующую команду py manage.py startapp (имя вашей директории) (рис.1.3)

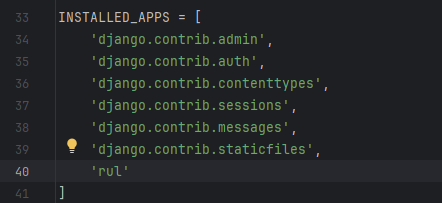


(рис.1.3)

После этого заходим в директорию Django в файл под названием

settings.py и добавим наше приложение в лист INSTALLED\_APPS

(рис.1.4)

(рис.1.4)

Также в этом же файле в листе TEMPLATES внутри листа DIRS

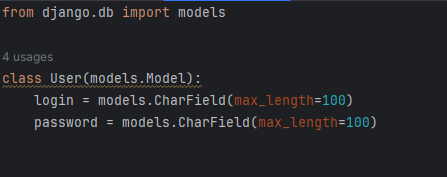
пропишем папку с нашим шаблоном с помощью BASE\_DIR (рис.1.5)



(рис.1.5)

Теперь заходим в директорию, которую ранее создали, у меня это директория url. Дальше переходим в файл под названием models.py,

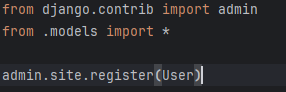
здесь мы создаем модель (рис.1.6)



(рис.1.6)

Дальше в файле admin.py регистрируем модель в админ-панели

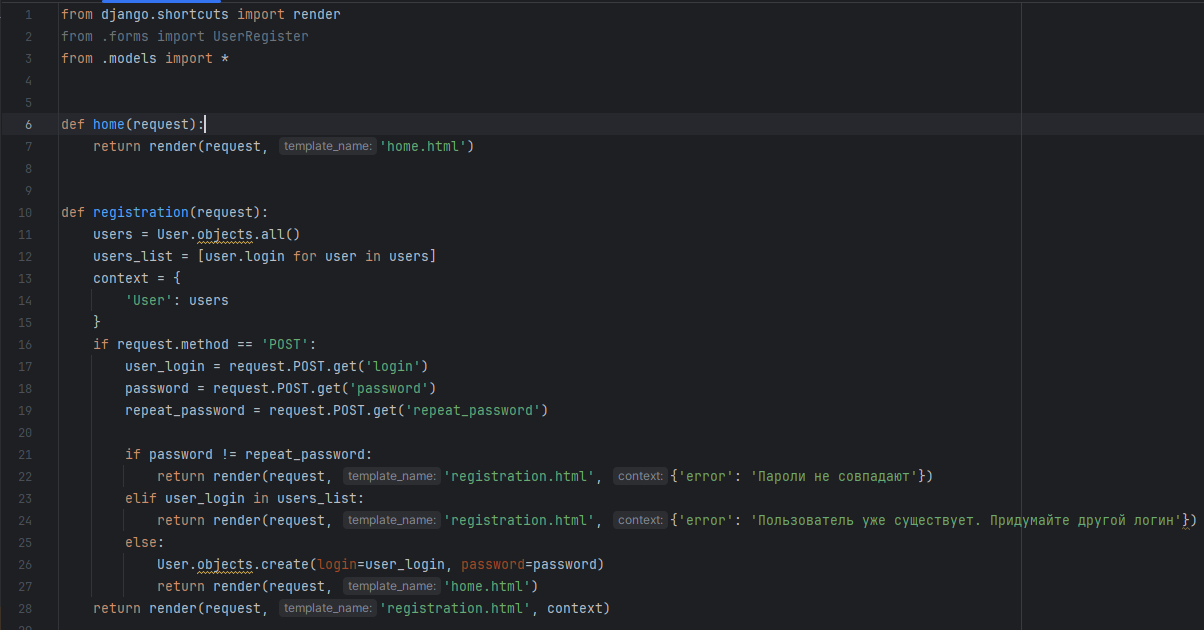
Django (рис.1.7)



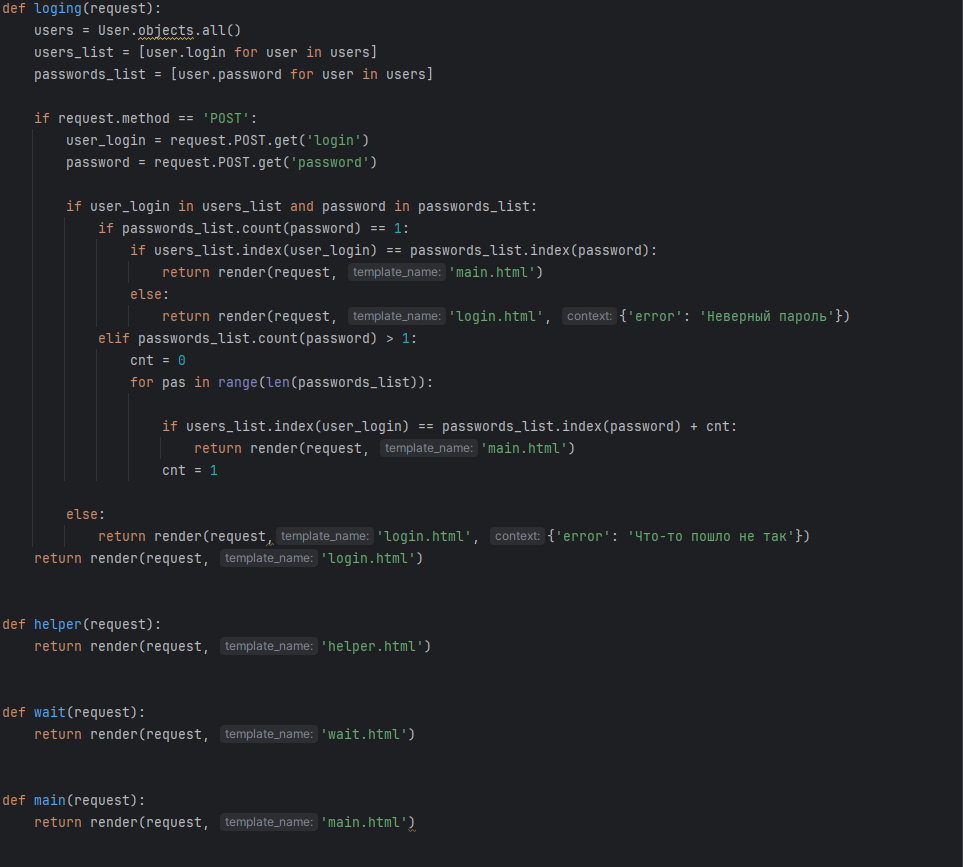
(рис.1.7)

Теперь переходим в файл views.py. В этом файле мы будем создавать функции. В начале мы должны импортировать render (from django.shortcuts import render) также нужно импортировать нашу модель из models (from .models import \*). Дальше создаем функции

home, helper, wait, main для этих функций передаем переменную request и возвращаем render с заготовленным шаблоном. Также создаем функцию registration и вносим туда логику регистрации, в которую будут поступать запросы GET и POST. И создаем функцию loging, в котором будем проверять верность логина и пароля. (рис.1.8) (рис.1.9)

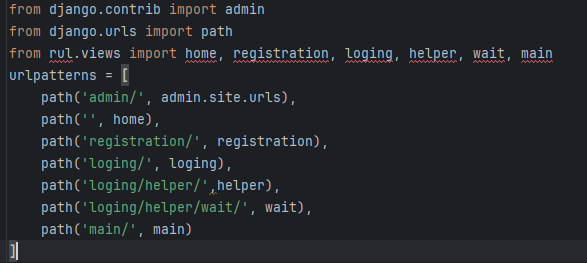


(рис.1.8)



(рис.1.9)

Далее мы переходим в папку urls.py и там импортируем функции и прописываем к ним путь с помощью path (рис.1.10)



(рис.1.10)

После написания всех функций и шаблонов мы можем запустить сервер, введя команду py manage.py runserver (рис.1.11)



(рис.1.11)

**Пример работы приложения на Flask**

Для начала нам нужно установить Flask, его можно установить в командной строке с помощью команды pip install Flask(рис.2.1)



(рис.2.1)

Теперь создаем питоновский файл (название можно вести любое), дальше из библиотеки flask импортируем следующие модули (Flask, render\_template, request) и импортируем sqlite3 для работы с базой данных (рис.2.2)



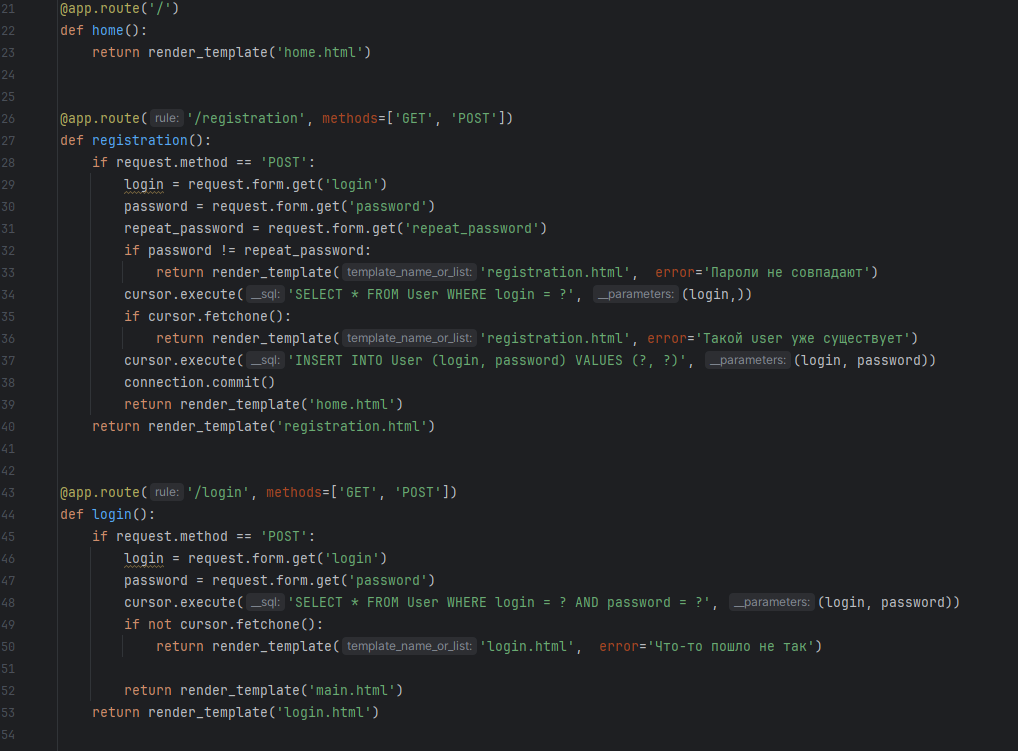
(рис.2.2)

Создадим объект app = Flask() и внутри укажем где Flask должен искать шаблоны. После создаем базу данных. Чтобы её создать нам потребуется прописать соединение с базой данных питона (connection =sqlite3.connect(“название базы данных”)), потом нам нужно создать объект курсор для выполнения операций с базой данных, сюда мы подключаем наше соединение (cursor = connection.cursor()).Потом прописываем создание таблицы User и параметры (id, login, password). (рис2.3)

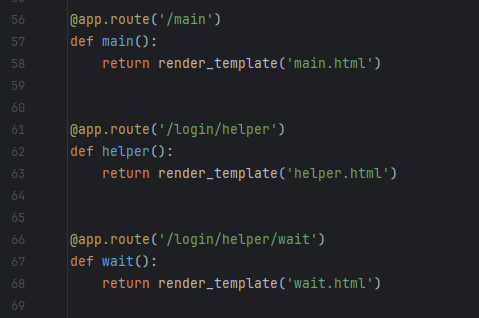


(рис.2.3)

Теперь создаем функции. С помощью .route() подключаем функцию к странице (ссылке) которой прописали в скобках. Теперь во всех функция прописываем логику и с помощью render\_template() отображаем определенный шаблон. (рис.2.4) (рис.2.5)



(рис.2.4)



(рис.2.5)

Также пропишем конструкцию, которая запускает приложение в режиме откладки (в этом режиме все возникающие ошибки будут отображаться на странице браузера). (рис.2.6)



(рис.2.6)

После просто запускаем сайт.

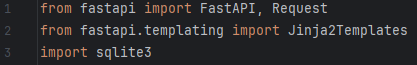
**Пример работы приложения на FastAPI**

Для начала нам нужно установить FastAPI, его можно установить в командной строке с помощью команды pip install fastapi (рис.3.1)



(рис.3.1)

Теперь создаем питоновский файл (название можно вести любое), дальше из библиотеки fastapi импортируем следующие модули (FastAPI, Request), из библиотеки fastapi.templating импортируем следующие модуль (Jinja2Templates) и импортируем sqlite3 для работы с базой данных (рис.3.2)



(рис.3.2)

Создадим объект приложения (app = FastAPI()).Прописываем объект, который создаёт объект шаблонов в FastAPI. После создаем базу данных. Чтобы её создать нам потребуется прописать соединение с базой данных питона (connection =sqlite3.connect(“название базы данных”)), потом нам нужно создать объект курсор для выполнения операций с базой данных, сюда мы подключаем наше соединение (cursor = connection.cursor()). Потом прописываем создание таблицы User и параметры (id, login, password). (рис3.3)



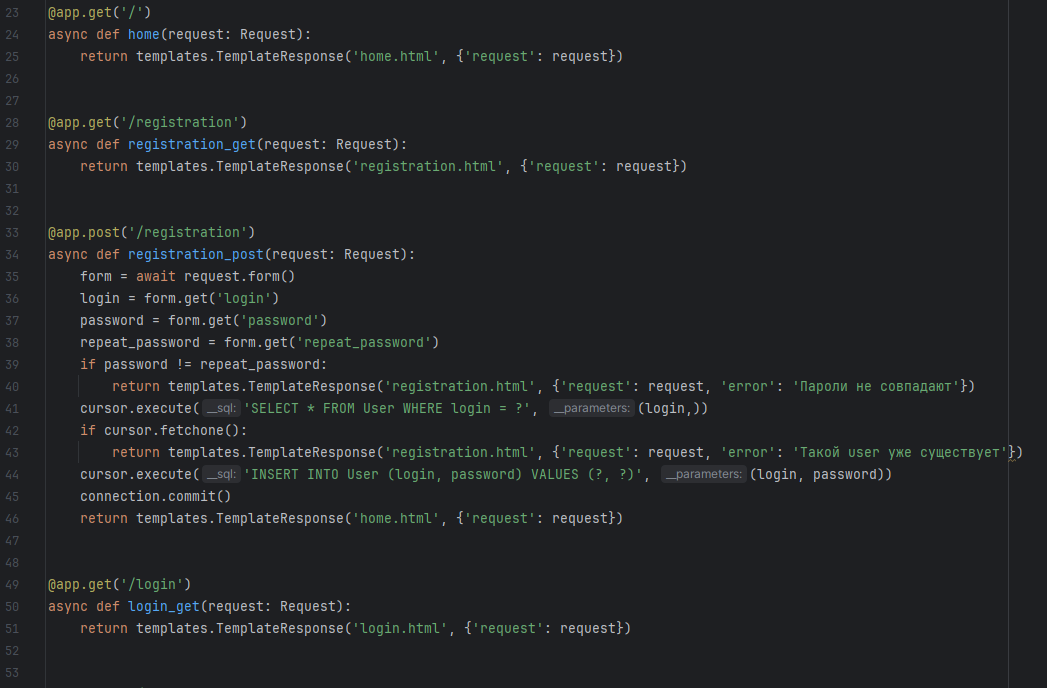
рис(3.3)

Теперь создаем функции. Сверху функции прописываем запрос

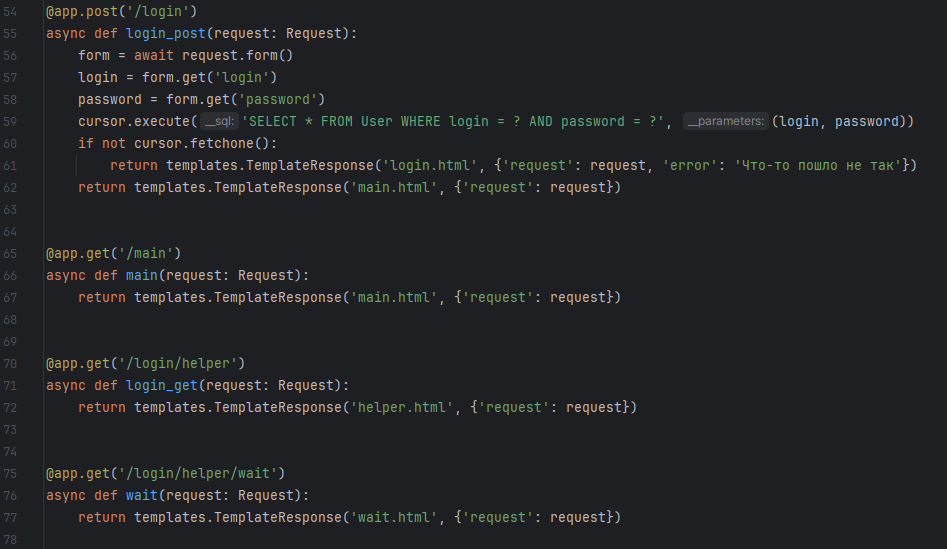
(@app.get() или @app.post()), по этим запросам декоратор будет указывать какой метод использовать для маршрута.

@app.get создаёт маршрут метода GET в корне сайта с результатами, возвращаемыми обёрнутой функцией.

@app.post используется для обработки запросов типа POST, когда клиент отправляет данные на сервер. (рис.3.4) (рис.3.5)



(рис.3.4)



(рис.3.5)

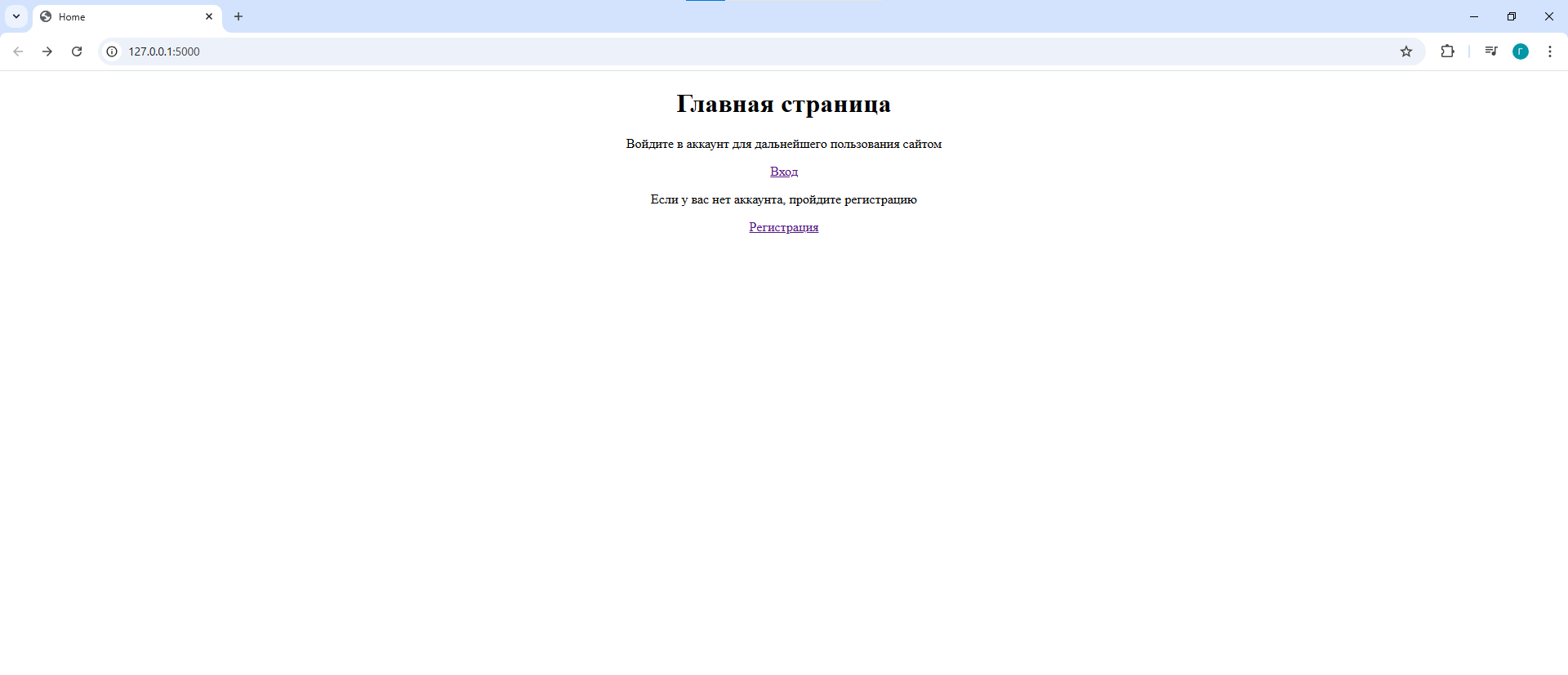
Для запуска программы в командную строку введем py -m uvicorn api:app (api – это название моего файла) (рис.3.6)



(рис.3.6)

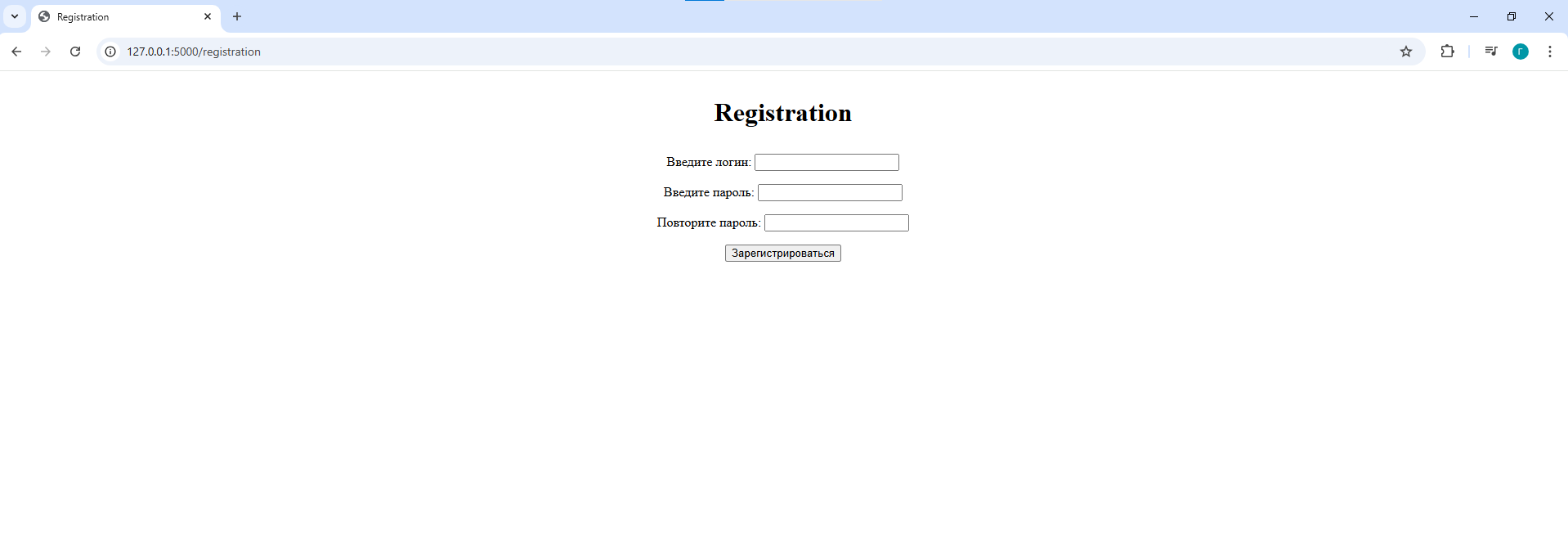
**4.Разработка в соответствии с созданной документацией**

Шаблон home.html (рис.4.1)



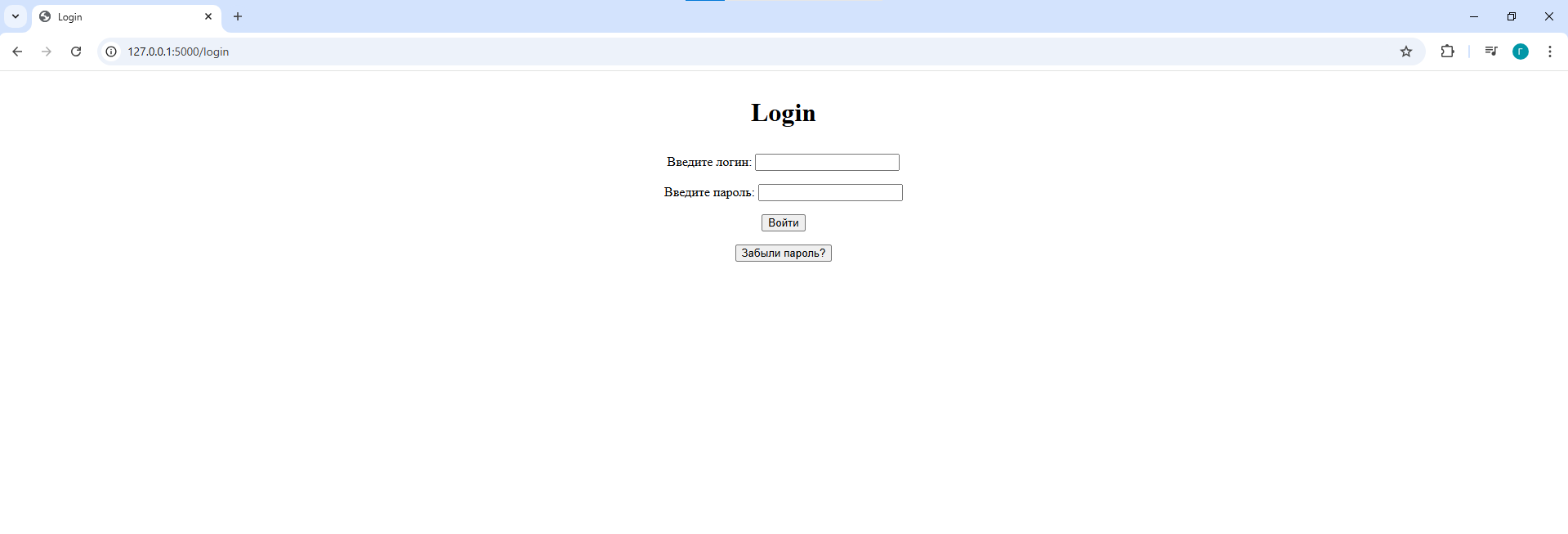
(рис.4.1)

Шаблон registration.html (рис.4.2)



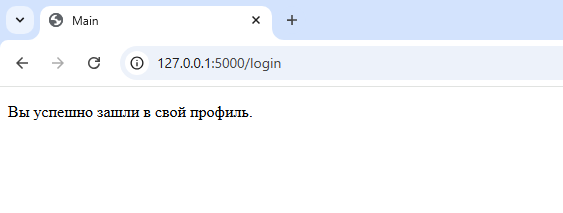
(рис4.2)

Шаблон login.html (рис.4.3)



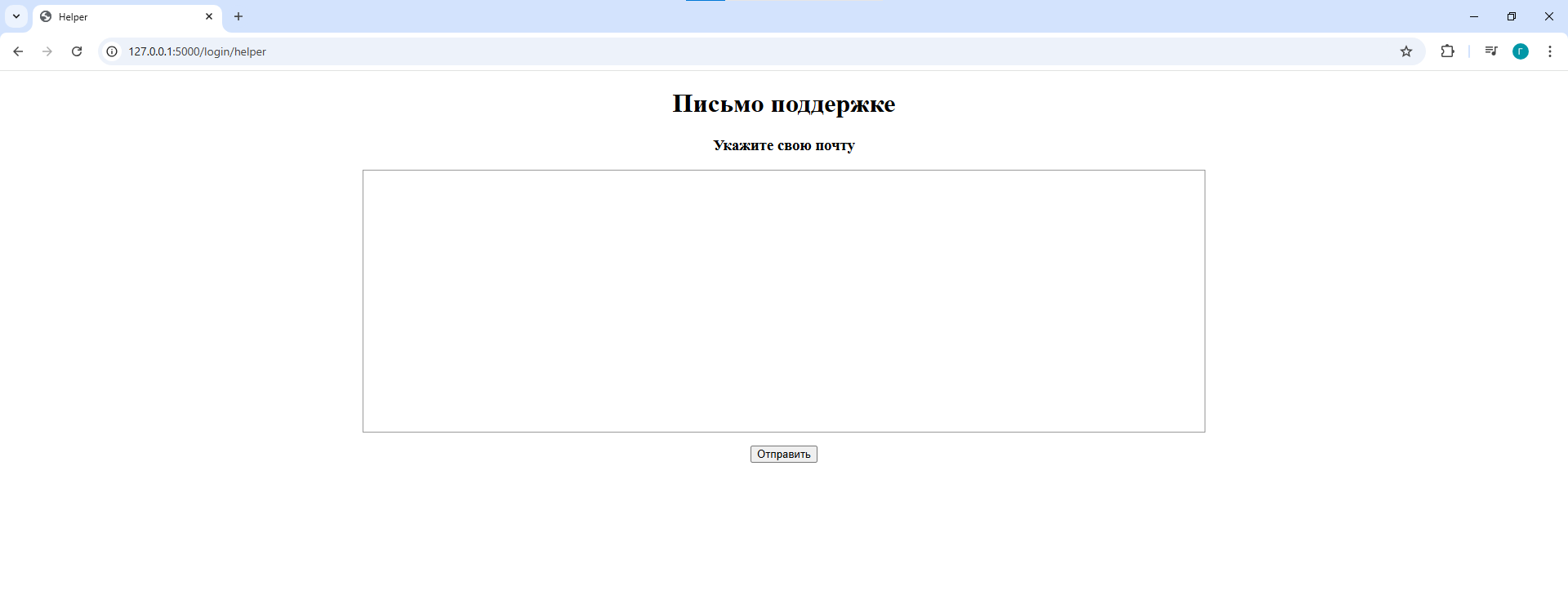
(рис.4.3)

Шаблон main.html (рис.4.4)



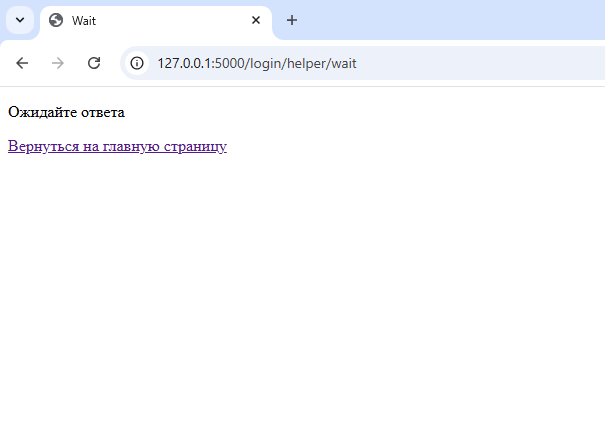
(рис.4.4)

Шаблон helper.html (рис.4.5)



(рис.4.5)

Шаблон wait.html (рис.4.6)



(рис.4.6)

**5. Анализ и интерпретация результатов**

**Сравнение Django и Flask**

**Django**

Плюсы:

* Собственная ORM (Объектно-реляционное отображение) обеспечивает мощную и удобную интеграцию с базами данных.
* Огромная экосистема с большим количеством приложений и библиотек.
* Улучшенная безопасность с помощью встроенных функций, таких как CSRF-защита и управление сеансами.
* Автогенерация административного интерфейса.  
   Масштабируемый и поддерживает большие веб-приложения.

Минусы:

* Более тяжелая и более сложная структура.
* Более длительная кривая обучения по сравнению с Flask.
* Ограничения на выбор API-интерфейсов сторонних разработчиков из-за привязки к собственной ORM.

**Flask**

Плюсы:

* Легкий и простой в использовании фреймворк.
* Низкая кривая обучения, идеально подходит для новичков.
* Расширяемый с помощью расширений для добавления дополнительных функций.
* Поддерживает RESTful API с помощью Flask-RESTful и Flask-API.  
   Подходит для небольших и средних веб-приложений.

Минусы:

* Отсутствие встроенной ORM. Требуется сторонняя библиотека, такая как SQLAlchemy или peewee.
* Меньшая экосистема по сравнению с Django.
* Меньше встроенных функций безопасности.
* Не масштабируется для очень больших веб-приложений.

Выбор между Django и Flask зависит от требований проекта:  
Для больших, сложных и масштабируемых веб-приложений, требующих мощных функций, Django является лучшим выбором.  
Для небольших, простых и быстрых веб-приложений Flask является более подходящим

**Сравнение Django и FastAPI**

**Django**

Плюсы:

* Django пользуется большой популярностью и имеет активное сообщество, которое предоставляет многочисленные ресурсы и документацию.
* Django включает в себя множество сторонних приложений и пакетов, которые расширяют его функциональность.
* Django предоставляет встроенную поддержку таких функций, как администрирование сайта, аутентификация пользователей и управление разрешениями.
* Django поставляется со встроенными инструментами разработки, такими как Django Shell и Django Admin, что упрощает отладку и взаимодействие с приложением.
* Django хорошо масштабируется для крупных приложений и поддерживает различные базы данных и серверы приложений.

Минусы:

* Django имеет относительно высокую кривую обучения и может быть сложен в освоении для новичков.
* Django менее производителен, чем некоторые другие фреймворки, особенно в случае высоконагруженных приложений.
* Приложения Django обычно требуют большего количества кода, что может привести к увеличению времени разработки и обслуживания.
* Django предназначен для монолитных приложений и не подходит для архитектур микросервисов.
* Django устанавливает четкую структуру, что может ограничить гибкость и настройки.

**FastAPI**

Плюсы:

* FastAPI оптимизирован для производительности и способен обрабатывать большое количество запросов в секунду.
* FastAPI имеет дружественный к разработчику API и прост в изучении и использовании.
* FastAPI предоставляет большую гибкость и позволяет легко создавать настраиваемые приложения.
* FastAPI хорошо подходит для архитектур микросервисов и позволяет легко интегрировать его с другими сервисами.
* FastAPI использует современные технологии, такие как asyncio и type hinting, что делает приложения более эффективными и поддерживаемыми.

Минусы:

* Сообщество FastAPI меньше, чем у Django, что может ограничить доступность ресурсов и поддержки.
* FastAPI не предоставляет встроенную поддержку таких функций, как аутентификация пользователей или администрирование сайта.
* Экосистема FastAPI меньше, чем у Django, что может ограничить доступность сторонних приложений.
* FastAPI требует более практического подхода к разработке по сравнению с Django, что может быть сложнее для новичков.
* FastAPI может быть менее масштабируемым для очень крупных приложений по сравнению с Django.

Выбор между Django и FastAPI зависит от конкретных требований проекта:  
  
Для высокопроизводительных, гибких и настраиваемых приложений:

FastAPI является хорошим выбором.

Для полнофункциональных, масштабируемых приложений с большой экосистемой:

Django является предпочтительным вариантом.

Для приложений с низкой кривой обучения, которые легко создавать и поддерживать:

FastAPI может быть лучшим вариантом.

Для архитектур микросервисов:

FastAPI является более подходящим выбором.

Для проектов, где скорость разработки является первоочередной задачей:

FastAPI может предложить более быстрый путь к разработке.

**Сравнение Flask и FastAPI**

**Flask**

Плюсы:

* Простота в использовании: Фласк имеет более простой процесс разработки и более понятный синтаксис, что делает его идеальным для начинающих.
* Гибкость: Фласк предоставляет больше гибкости для разработчиков, позволяя им использовать любые ORM, шаблоны и расширения по своему выбору.
* Сообщество: У Фласка большое и активное сообщество, что обеспечивает отличную поддержку и ресурсы.

Минусы:

* Сложность: По мере роста веб-приложения управление несколькими маршрутами и зависимостями в Фласке может стать сложным.
* Производительность: Фласк не так быстр, как некоторые другие фреймворки, такие как FastAPI.
* Отсутствие встроенной обработки данных: В Фласке отсутствует встроенная поддержка современных инструментов обработки данных, таких как Pydantic.

**FastAPI**

Плюсы:

* Производительность: FastAPI оптимизирован для высокой производительности и может обрабатывать большие объемы запросов с низкой задержкой.
* Встроенная обработка данных: FastAPI имеет встроенную поддержку для Pydantic, что позволяет легко создавать модели данных и схемы проверки запросов.
* Автоматическое создание документации: FastAPI автоматически генерирует интерактивную документацию для вашей API, что упрощает ее использование и интегрирование.

Минусы:

* Сложность обучения: FastAPI имеет более сложный синтаксис и процесс разработки по сравнению с Flask, что может быть сложным для начинающих.
* Меньшее сообщество: Сообщество FastAPI меньше, чем сообщество Flask, что может ограничить поддержку.
* Ограниченная гибкость: FastAPI имеет более фиксированную структуру, что ограничивает гибкость разработчиков при выборе ORM и шаблонов.

Для простых веб-приложений или начинающих разработчиков Flask может быть лучшим вариантом.

Для высокопроизводительных API или приложений, где важны производительность и обработка данных, FastAPI является лучшим выбором.

**6. Заключение**

В процессе данного анализа и сравнения библиотек Django, Flask и FastAPI для веб-разработки стало очевидно, что каждая из них имеет свои преимущества и недостатки, что делает их подходящими для различных типов проектов.

Django:

* Наиболее подходящая платформа для крупных и сложных веб-приложений, требующих масштабируемости, безопасности и широкого набора встроенных функций.
* Предоставляет обширный фреймворк с богатым набором функций, что может сделать разработку проще и быстрее.
* Ограничен в гибкости и требует большего обучения и опыта для использования.
* Подходит для проектов, где требуется масштабируемость, безопасность и структурированный подход к разработке.

Flask:

* Идеальный выбор для небольших и средних веб-приложений, где требуется гибкость, скорость разработки и легкость обучения.
* Предоставляет базовый фреймворк, который позволяет разработчикам создавать собственные функции и расширения.
* Более гибкий и простой в освоении, что делает его подходящим для начинающих и быстрых проектов.
* Может не подходить для крупных и сложных проектов, требующих масштабируемости и надежности.

FastAPI:

* Новейшая и наиболее ориентированная на производительность библиотека, превосходящая Django и Flask в тестах скорости.
* Предоставляет высококлассный дизайн, основанный на \*\*Asynchronous Python (ASGI)\*\*, что делает приложения более быстрыми и отзывчивыми.
* Имеет ограниченный набор функций по сравнению с Django, но позволяет легко интегрироваться со сторонними библиотеками.
* Лучше всего подходит для проектов, где скорость и производительность являются первоочередными задачами.

В конечном счете, выбор подходящей библиотеки зависит от конкретных требований проекта. Для крупных и сложных приложений, требующих масштабируемости и безопасности, Django является лучшим выбором. Flask подходит для небольших и средних приложений, где важны гибкость и простота использования. FastAPI является лучшим выбором для приложений, где производительность является критическим фактором.