Дипломная работа

**Анализ и сравнение написания web-приложений с использованием разных фреймворков:** Разработать простые веб-приложения с использованием Django, Flask и FastAPI, провести их сравнение.

Автор: Панькевич М.В.

2025

**Содержание**

1. Введение-----------------------------------------------------------------------------3

- Актуальность темы

- Потребность рынка

- Практическая значимость

- Личный интерес и перспективы

2. О фреймворках Django, Flask и FastAPI:---------------------------------------4

# 3. Изучение архитектуры каждого фреймворка---------------------------------4

4. Сравнение Django, FastAPI и Flask----------------------------------------------8

# 5. Разработка простых web-приложений с Django, Flask и FastAPI---------9

## 6. [Шаблоны страниц](https://urban-university.ru/members/courses/course999421818026/null)------------------------------------------------------------------11

## 7. Проект c использованием Django.----------------------------------------------12

## 8. Проект c использованием Flask.-------------------------------------------------13

## 9. Проект c использованием FastAPI.----------------------------------------------14

## 10. Анализ--------------------------------------------------------------------------------16

## 11. Заключение--------------------------------------------------------------------------17

***Ведение***

В данной работе будет проведен анализ и сравнение трех популярных фреймворков для разработки web-приложений на языке Python: Django, Flask и FastAPI. Будет рассмотрена архитектура и принципы разработки каждого фреймворка, а также будет проведено сравнение их производительности, безопасности и масштабируемости.

1. **Актуальность темы:** В современном мире веб-разработка становится важной частью цифровой экономики. Существует множество фреймворков, которые позволяют разработчикам создавать веб-приложения, каждый из которых имеет свои особенности, преимущества и недостатки. Анализ и сравнение таких фреймворков, как Django, Flask и FastAPI, позволяет лучше понять их возможности и выбрать наиболее подходящий инструмент для конкретной задачи.
2. **Потребности рынка:** С ростом популярности веб-приложений и сервисов существует высокая потребность в разработчиках, которые умеют эффективно использовать различные фреймворки для создания качественных и производительных приложений. Компании ищут специалистов, способных выбрать оптимальные технологии для реализации своих проектов, что делает изучение различных фреймворков особенно актуальным.
3. **Практическая значимость:** Разработка простых веб-приложений с использованием Django, Flask и FastAPI позволит не только освоить каждый из этих фреймворков, но и получить практический опыт в их сравнении. Это даст возможность выявить сильные и слабые стороны каждого из них, а также понять, в каких случаях лучше использовать тот или иной инструмент.
4. **Личный интерес и потенциальные перспективы**: Изучение и сравнение различных фреймворков для веб-разработки соответствует моему личному интересу к программированию и созданию веб-приложений. Освоение этих технологий не только расширит мои профессиональные навыки, но и откроет новые возможности для карьерного роста в области веб-разработки.

Таким образом, выбор темы "Анализ и сравнение написания веб-приложений с использованием разных фреймворков: Разработка простых веб-приложений с использованием Django, Flask и FastAPI, а также их сравнение" обусловлен актуальностью, потребностями рынка, практической значимостью и личным интересом. Это делает данную тему подходящей для проведения дипломной работы.

**О фреймворках Django, Flask и FastAPI**

**Что такое Django?**

Django - это веб-фреймворк с полным стеком с открытым исходным кодом, написанный на Python, который следует архитектурному шаблону MTV (модель-шаблон-представление) и принципу DRY (Не повторяйтесь). Оно было сделано Адрианом Головати и Саймоном Уиллисоном и было публично опубликовано в 2005 году. Данное название основано на имени певца Джанго Рейнхардта.

**Что такое Flask?**

Flask - это удобный для начинающих веб-фреймворк с открытым исходным кодом, написанный на Python, который следует архитектурному шаблону MVC (модель-представление-контроллер) и поддерживает шаблоны Jinja, которые можно использовать для создания представлений. Оно было сделано Армином Ронахером из Pocoo, международной группы энтузиастов Python, созданной в 2004 году. Название "Flask" является отсылкой к более раннему фреймворку Bottle.

**Что такое FastAPI?**

FastAPI также является микро-веб-фреймворком Starlette-ASGI (Asynchronous Server Gateway Interface) с открытым исходным кодом, написанным на Python и используемым для создания RESTful API. Оно было сделано Себастьяном Рамиресом в декабре 2018 года. Название основано на скорости разработки API, отсюда и "FastAPI".

# **Изучение архитектуры каждого фреймворка.**

# **Архитектура web-приложений на Django.**

Django - это фреймворк, использующий шаблон проектирования Model-View-Controller (MVC). Во многих статьях приводятся объяснения что такое MVC, схемы, а так же говорится о их неточности (несколько примеров: <https://habr.com/ru/articles/321050/>, <https://habr.com/ru/articles/175465/>).

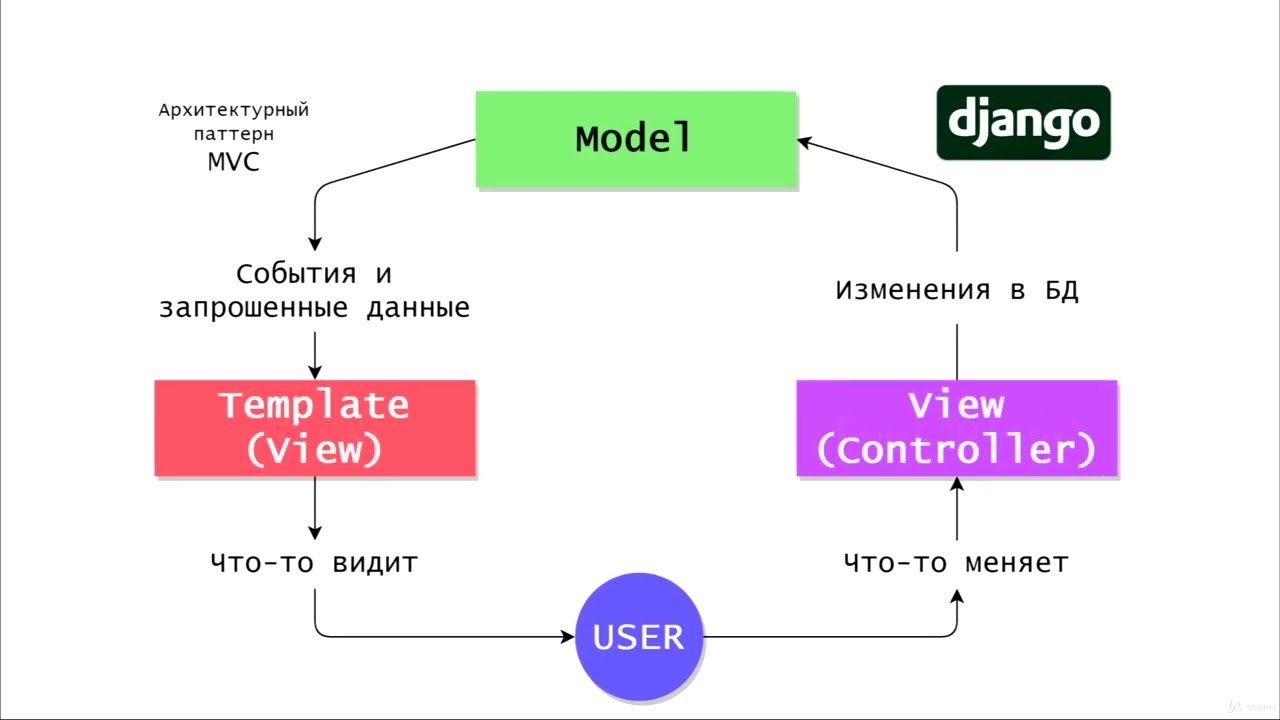
Обычно в таких схемах MVC описывают подобным образом:

• Model — доступ к хранилищу данных • View — это интерфейс, с которым взаимодействует пользователь • Controller — некий связывающий объект между model и view.

Данные распространенные схемы только запутывают и мешают, когда вы хотите написать приложение, в котором есть бизнес-логика.

Стоит обратить внимание на две вещи. Первое, часто под M в MVC подразумевают — модель данных, и говорят, что это некий класс, который отвечает за предоставление доступа к базе данных. Что неверно, и не соответствует классическому MVC и его потомкам MV\*. В классическом MVC под M подразумевается domain model — объектная модель домена, объединяющая данные и поведение. Если говорить точнее, то M в MVC это интерфейс к доменной модели, так как domain model это некий слой объектов, описывающий различные стороны определенной области бизнеса. Где одни объекты призваны имитировать элементы данных, которыми оперируют в этой области, а другие должны формализовать те или иные бизнес-правила. Второе, в Django нет выделенного слоя controller, и когда говорят, что в Django слой views — это контроллер, не стоит этому верить. Нужно обратиться к официальной документации, а точнее к FAQ, тогда можно увидеть, что этот слой вписывается в принципы слоя View в MVC, особенно, если рассматривать DRF, а как такового слоя Controller в Django нет. Как говорится в FAQ, если вам очень хочется аббревиатур, то можно использовать в контексте Django аббревиатуру MTV (Model, Template, and View). Если очень хочется рассматривать Web MVC и сравнивать Django с другими фреймворками, то для простоты можно считать view контроллером. Несмотря на то, что Django не соответствует MVC аббревиатуре, в ней реализуется главный смысл MVC – отделение бизнес-логики от логики представления данных. Но на практике это не всегда так.

Схематично можно представить архитектуру MVT в Django следующим образом:



Основные элементы паттерна: URL dispatcher: при получение запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос. View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления). Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных. Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления. Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

# **Архитектура web-приложений на FastAPI.**

Архитектура веб-приложений на FastAPI основана на принципах современного веб-разработки, которые обеспечивают производительность, простоту в использовании и поддержку асинхронного программирования. FastAPI — это асинхронный веб-фреймворк для Python, который отлично подходит для создания API на основе стандартов OpenAPI и JSON Schema. Вот основные аспекты архитектуры веб-приложений на FastAPI:

1. Структура проекта Стандартная структура проекта может выглядеть следующим образом:

my\_fastapi\_app/

│

├── app/ # Основная папка приложения

│ ├── \_\_init\_\_.py # Инициализация приложения

│ ├── main.py # Точка входа для запуска приложения

│ ├── models.py # Модели данных

│ ├── routes.py # Определение маршрутов (эндпоинтов)

│ ├── schemas.py # Схемы (например, для валидации входных данных)

│ ├── services/ # Логика бизнес-уровня

│ ├── database.py # Настройки базы данных

│ └── utilities.py # Вспомогательные функции

│

├── tests/ # Тесты

│ └── test\_routes.py # Тесты для маршрутов

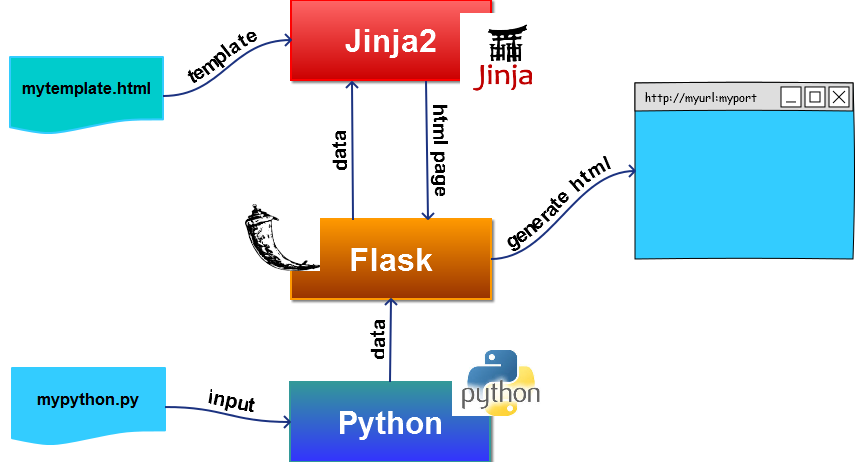
│

├── requirements.txt # Зависимости

└── .venv # Переменные окружения

1. Асинхронное программирование. FastAPI поддерживает асинхронные функции, что позволяет обрабатывать большее количество запросов одновременно.
2. Защита и аутентификация. FastAPI поддерживает OAuth2, JWT и другие методы аутентификации. Можно использовать встроенные зависимости для защиты маршрутов.
3. Автоматическая документация. FastAPI автоматически генерирует документацию для вашего API на основе аннотаций типов. Она доступна по адресам /docs (Swagger UI) и /redoc (ReDoc).
4. Тестирование. FastAPI предоставляет простые методы для тестирования приложения с помощью библиотеки pytest. Можно использовать встроенный тестовый клиент для имитации запросов к API.
5. Разделение логики приложения. Рабочая логика (сервисы) должна быть отделена от маршрутов, что улучшает поддержку приложения и тестируемость. Например, можно создать отдельный класс или набор функций для обработки бизнес-логики.

# **Архитектура web-приложений на Flask.**

Архитектура веб-приложений на Flask основана на концепциях, которые обеспечивают модульность, расширяемость и поддержку. Основные аспекты: 

1. Модульная структура проекта. Flask позволяет организовать код в модули, что делает проект более управляемым.
2. Маршруты и представления (Views), В Flask маршруты устанавливаются с помощью декораторов. Каждый маршрут связан с функцией (представлением), которая обрабатывает запрос и возвращает ответ.
3. Шаблоны и рендеринг. Flask использует Jinja2 для работы с шаблонами, что позволяет динамически вставлять данные в HTML. Шаблоны хранятся в папке templates, и их можно использовать для рендеринга HTML-страниц.
4. Форма и валидация. Flask поддерживает обработку форм через библиотеку Flask-WTF, которая интегрирует настройки для защиты от CSRF и валидации форм.
5. Модели и базы данных. Flask часто используется с SQLAlchemy для работы с базами данных. Можно определять модели, которые представляют таблицы базы данных.
6. Конфигурация. Конфигурация приложения может быть вынесена в отдельный файл (например, config.py) и загружена во время инициализации приложения.
7. Статические файлы. Статические файлы (например, CSS и JavaScript) размещаются в папке static, и к ним можно получить доступ через встроенные функции Flask.
8. Расширяемость и пакетирование. Flask имеет множество расширений, которые можно добавить для улучшения функциональности, таких как: • Flask-Login — для управления аутентификацией пользователей. • Flask-Mail — для отправки электронной почты • Flask-Migrate — для управления миграциями базы данных.
9. Тестирование. Flask поддерживает написание тестов с использованием библиотеки unittest. Можно тестировать маршруты, формы, модели и другое.

# **Сравнение фреймворков Django, Flask и FastAPI.**

| **Параметры** | **Django** | **FastAPI** | **Flask** |
| --- | --- | --- | --- |
| **Тип** | Веб-фреймворк с полным стеком | Микро-веб-фреймворк | Микро-веб-фреймворк |
| **Пример использования** | Создание сложных веб-приложений и API | Создание API и микросервисов | Идеально подходит для создания небольших веб-приложений и простых API |
| **Производительность** | Быстрый способ создания большого веб-приложения | Очень быстрый для создания API и микросервисов | Медленнее из-за ручной проверки и синхронизированного программирования. |
| **Масштабируемость** | Масштабируемый, но его ORM и шаблонизатор могут замедлить масштабирование | Высокая масштабируемость, поскольку в нем используется асинхронизация кода и изменение типов. | Сложно масштабировать, поскольку отсутствует встроенная поддержка ORM или кэширования. |
| **Кривая обучения** | Сложный для начинающих | Просто для начинающих | Умеренный для начинающих |
| **Инструменты для работы с базами данных** | Полный набор | Ограничено, встроенной поддержки нет | Ограничено, встроенной поддержки нет |
| **Асинхронное программирование** | Да, это можно сделать с помощью Asyncio, но немного медленнее. | Да, но быстрее благодаря Pydantic. | Нет, но можем ли мы использовать другие библиотеки. |
| **ORM (Объектно-реляционное отображение)** | ДА | НЕТ | НЕТ |
| **Сообщество** | Большой и активный | Небольшой, но растущий | Большой и активный |
| **Документация** | Огромный | Небольшой, но все еще растущий | Большой |
| **Преимущества** | Это отличный выбор для тех, кто хочет создавать безопасные, масштабируемые, гибкие и простые в обслуживании веб-приложения | Это отличный выбор для создания высокопроизводительных API и микросервисов | Это отличный выбор для создания веб-приложений малого и среднего уровня, где производительность не является большой проблемой, а разработчикам требуется гибкость. |
| **Недостатки** | Это может быть сложным для начинающих, сложным в отладке и не подходящим для небольших проектов | Основные файлы могут быть перегружены и отсутствует встроенная защита. | Отсутствует встроенная поддержка кэширования, ORM, асинхронизации и т. Д |

# **Разработка простых web-приложений с использованием Django, Flask и FastAPI, их сравнение и анализ.**

## В своём проекте я рассмотрю создание одного приложения с использованием Django, Flask и FastAPI.

## Цель сайта –  помочь пользователям, делится изображениями и их описанием, для получения комментариев – обратной связи от других пользователей сайта .

### [**Структура проекта**](https://urban-university.ru/members/courses/course999421818026/null)

## Проект включает следующие ключевые компоненты:

## Домашняя страница:

## Тут есть страница приветствия и навигационная панель.

Навигационная панель состоит из: [Главная](http://127.0.0.1:5000/), [Регистрация](http://127.0.0.1:5000/register), [Вход](http://127.0.0.1:5000/login), [Список пользователей](http://127.0.0.1:5000/users_list), [Изображения](http://127.0.0.1:5000/images)

## К списку пользователей и изображениям, есть доступ только у авторизованных пользователей, так что при попытке зайти на них до захода в аккаунт, должно совершаться перенаправление на страницу авторизации. С неё можно зайти в аккаунт или создать новый.

## Если пользователь авторизировался у него появляется кнопка «выйти» - для выхода из учетной записи

## 

# **Регистрация нового пользователя:**

## 

## [Вход](http://127.0.0.1:5000/login):

## 

## [Список пользователей](http://127.0.0.1:5000/users_list):

## 

## [Изображения](http://127.0.0.1:5000/images):

## 

## В Галерее изображений, есть возможность добавлять новые изображения с описанием, также их можно комментировать. По нажатию на картинку, выведет в отдельное окно, с конкретным изображением, описанием, комментариями и окно добавить комментарий:

## 

## [Шаблоны страниц](https://urban-university.ru/members/courses/course999421818026/null):

## base.html – базовый шаблон, подключающий Bootstrap, на его основании пишутся остальные. Отдельно стоит отметить что в нём прописана навигационная панель home.html – домашняя страница images.html – галерея изображений, есть возможность добавлять новые изображения с описанием, также их можно комментировать, по нажатию на картинку, мы попадём в image\_detail.html image\_detail.html – это отдельное окно, с конкретным изображением – на которое нажали в images, так же там описание, комментарии и окно добавить комментарий.

## login.html – вход в учётную запись register.html – создание учётной записи users\_list.html – список пользователей

## Проект c использованием Django.

## Схематичное изображение структуры проекта.

## Список необходимых библиотек:

asgiref==3.8.1  
Django==5.1.4  
pillow==11.1.0  
sqlparse==0.5.3  
tzdata==2024.2

## Проект c использованием Flask.

## Схематичное изображение структуры проекта.

## 

## Список необходимых библиотек:

alembic==1.14.0  
blinker==1.9.0  
click==8.1.8  
colorama==0.4.6  
dnspython==2.7.0  
email\_validator==2.2.0  
Flask==3.1.0  
Flask-Login==0.6.3  
Flask-Migrate==4.1.0  
Flask-SQLAlchemy==3.1.1  
Flask-WTF==1.2.2  
greenlet==3.1.1  
idna==3.10  
is-disposable-email==1.0.0  
itsdangerous==2.2.0  
Jinja2==3.1.5  
Mako==1.3.8  
MarkupSafe==3.0.2  
SQLAlchemy==2.0.37  
typing\_extensions==4.12.2  
Werkzeug==3.1.3  
WTForms==3.2.1  
wtforms-validators==1.0.0

## Проект c использованием FastAPI.

## Схематичное изображение структуры проекта

## 

## Список необходимых библиотек:

alembic==1.14.0  
annotated-types==0.7.0  
anyio==4.8.0  
bcrypt==4.2.1  
certifi==2024.12.14  
charset-normalizer==3.4.1  
click==8.1.8  
colorama==0.4.6  
databases==0.9.0  
fastapi==0.115.6  
fastapi-login==1.10.3  
greenlet==3.1.1  
h11==0.14.0  
idna==3.10  
itsdangerous==2.2.0  
Jinja2==3.1.5  
Mako==1.3.8  
MarkupSafe==3.0.2  
passlib==1.7.4  
pydantic==2.10.5  
pydantic\_core==2.27.2  
PyJWT==2.10.1  
python-multipart==0.0.20  
requests==2.32.3  
security==1.3.1  
sniffio==1.3.1  
SQLAlchemy==2.0.37  
starlette==0.41.3  
typing\_extensions==4.12.2  
urllib3==2.3.0  
uvicorn==0.34.0

## Анализ.

| **Критерии** | **Django** | **Flask** | **FastAPI** |
| --- | --- | --- | --- |
| Встроенная функциональность | Множество встроенных функций | Для реализации функционала может потребовать интеграцию дополнительных пакетов | Для реализации функционала требует интеграцию дополнительных пакетов |
| Структура проекта | Мене гибкая, требует большего времени для изучения | Минимальная и гибкая | Минимальная и гибкая |
| Синтаксис | Использует свои шаблоны для разметки HTML Управление маршрутами осуществляется через URLconf | Простой, легко читается. Для обработки запросов и маршрутизации использует Python функции | Основывается на стандартных аннотациях Python для обработки данных. Удобные декораторы для маршрутизации и валидации данных |
| Интегрируемость с различными модулями | Имеет большое число сторонних библиотек и пакетов | Поддерживает множество сторонних расширений, однако может потребовать больше настроек. Благодаря гибкости может легко интегрировать сторонние расширения, однако некоторые расширения могут быть несовместимы друг с другом | Быстро интегрируется с современными библиотеками, широкая поддержка асинхронных библиотек |
| Объём занимаемой памяти с библиотеками | 60,2 МБ | 46,5 МБ | 62,5 МБ |
| Объём занимаемой памяти без библиотек | 1,34 МБ | 560 КБ | 3,69 МБ |

## Заключение

## В ходе исследования веб-фреймворков Django, Flask и FastAPI было выявлено, что каждый из них обладает своими уникальными преимуществами и недостатками.

Выбор фреймворка зависит от конкретных потребностей проекта: Django подходит для создания крупных и сложных веб-приложений с богатым функционалом и высокой безопасностью, в особенности, когда важна быстрая разработка с использованием готовых решений. Примеры: интернет-магазины, социальные сети и платформы для взаимодействия с пользователями. Flask идеально подходит для проектов, где требуется легкость и гибкость. Это хороший выбор для небольших приложений и стартапов, где важно быстрое прототипирование и возможность "собирать" приложение по частям. Примеры: сайты-визитки. FastAPI подходит для высокопроизводительных приложений, особенно API, работающих с большим количеством одновременных подключений. Он идеально подходит для проектов, которым необходима скорость, масштабируемость и поддержка асинхронности. Примеры: системы реального времени (чаты).