Пояснительная записка к дипломному проекту

на тему: «**Сравнение различных подходов к реализации REST API: Django Rest Framework, FastAPI и Flask¬RESTful: Разработать REST API с использованием Django Rest Framework, FastAPI и Flask¬RESTful, провести их сравнение»**

Автор: Чибисов Сергей Юрьевич

**СОДЕРЖАНИЕ**

стр.

1. Введение.

1.1. Обоснование выбора темы 2

1.2. Определение цели и задач разработки 3

1. Основные понятия и определения. 3
2. Методы и подходы к разработке 6
3. Обзор инструментов и библиотек для разработки REST API на Python с использованием фреймворков 8
   1. Flask 8
   2. Django Rest Framework 9
   3. FastAPI 9
4. Разработка REST API 10
   1. Подготовка среды 10
   2. Установка необходимых библиотек для каждой технологии 11
5. Реализация REST API 12
   1. Пример реализации API с использованием Django Rest Framework 12
   2. Пример реализации API с использованием FastAPI 18
   3. Пример реализации API с использованием Flask¬Restfull 20
6. Сравнительный анализ 21
7. Заключение 24
8. **Введение**
   1. **Обоснование выбора темы.**

Значимость REST в современном веб разработке

1. Простота и гибкость:

­ REST использует HTTP — широко известный протокол, что облегчает понимание и использование. Благодаря этому, разработчики могут быстро создавать и интегрировать API.

2) Масштабируемость:

­ Отсутствие состояния между запросами позволяет легко масштабировать серверные приложения, улучшая их производительность под нагрузкой.

3) Совместимость:

­ RESTful API могут быть легко интегрированы с различными клиентскими приложениями, сколько бы ни было платформ (мобильные, веб-приложения и т. д.).

4) Универсальность:

­ REST можно использовать с различными форматами данных, что позволяет выбирать наиболее подходящий под конкретные нужды проекта.

5) Кэширование:

­ Возможности кэширования встроены в саму архитектуру, что позволяет улучшить производительность приложений и снизить нагрузку на сервер.

6) Экосистема и стандарты:

­ REST API активно поддерживаются с помощью библиотек и инструментов (например, Swagger для документирования, Postman для тестирования), что упрощает их разработку и поддержку.

7) Сообщество и зрелость:

­ REST стал стандартом в разработке веб сервисов, что привело к созданию обширного сообщества и множества ресурсов, обучающих материалам и лучшим практикам.

REST как архитектурный стиль имеет огромную значимость в современной веб разработке. Он обеспечивает множество преимуществ, таких как простота, масштабируемость и гибкость, что делает его подходящим для большинства веб-приложений и API. REST продолжает оставаться основой для разработки веб сервисов, и его популярность только растет с увеличением числа мобильных и облачных приложений.

Технологии для разработки REST API на языке Python выделяется рядом факторов, определяющих эффективность, масштабируемостью и удобство обслуживания приложения.

Использование REST API в Python позволяет разработчикам создавать мощные и эффективные системы, которые могут легко взаимодействовать и обмениваться данными друг с другом, что делает их важными инструментом в современной веб разработки.

* 1. **Определение цели и задач разработки**

Цель исследования: разработать REST API с использованием Django Rest Framework, FastAPI и Flask¬RESTful, провести их сравнение. Улучшение взаимодействия между клиентом и сервером, оптимизация производительности и масштарбируемости, провести их сравнение.

Задачи исследования:

1. Определение требований к API.
2. Исследование различных библиотек и фреймворков (Flask, Django, FastAPI).
3. Сравнение реализации REST API на различных фреймворках (Flask, Django, FastAPI). Плюсы и минусы каждого Фреймворка.
4. Написание дипломной работы: составить дипломную работу, включающую в себя введение, обзор фреймворков, разработки приложений и результаты исследования, анализ результатов, выводы и рекомендации.
5. **Основные понятия и определения.**

**REST API**

1. REST (Representational State Transfer)

­ Архитектурный стиль разработки API, основанный на стандартных HTTP¬методах и принципах. Сосредоточен на использовании ресурсов и их представлениях.

2) API (Application Programming Interface)

¬ Набор правил и протоколов, который позволяет различным программам взаимодействовать друг с другом.

3) Ресурс

¬ Объект или представление данных, доступное через API. Например, пользователи, продукты, заказы и т. д.

1. HTTP¬методы

Основные методы для взаимодействия с ресурсами:

¬ GET: Получить данные.

¬ POST: Создать новый ресурс.

¬ PUT: Обновить существующий ресурс.

¬ DELETE: Удалить ресурс.

1. URI (Uniform Resource Identifier)

¬ Уникальный идентификатор ресурса в API, обычно представляющий собой URL.

1. Форматы данных

¬ Часто используется JSON (JavaScript Object Notation) для обмена данными, но также могут использоваться XML или другие форматы.

7) Состояние (State)

¬ HTTP¬запросы могут быть безсостояниеми, при этом вся необходимая информация передается с каждым запросом (разделение состояния между клиентом и сервером).

**REST API на Flask**

1. Flask

¬ Легковесный веб фреймворк для Python, который позволяет быстро разрабатывать API. Поддерживает расширяемость и простоту.

1. Flask¬RESTful

¬ Расширение для Flask, которое упрощает создание REST API путем предоставления инструментов для определения ресурсов и маршрутов.

1. Маршруты

¬ Определяют, какие функции будут вызваны для обработки входящих запросов для определенных URL.

**REST API на Django**

1. Django ¬ полноценный веб фреймворк, предоставляющий мощные инструменты для разработки приложений, в том числе API.
2. Django REST framework (DRF) ¬ расширение для Django, которое облегчает создание веб API, предоставляя множество функций, таких как аутентификация, сериализация и документация.
3. Сериализация ¬ процесс преобразования сложных типов данных (например, объектов моделей Django) в форматы, понятные для передачи (например, JSON).
4. Views и ViewSets ¬ Views представляют логику обработки запросов маршрутов, касающихся одного ресурса.

**REST API на FastAPI**

1. FastAPI ¬ современный высокопроизводительный веб¬фреймворк для Python, созданный для разработки API с использованием Python type hints. Поддерживает автоматическую документированию API.
2. Async и Await ¬ FastAPI поддерживает асинхронное программирование, что позволяет обрабатывать запросы более эффективно, особенно под высокой нагрузкой.
3. Pydantic ¬ библиотека, используемая в FastAPI для валидации данных и сериализации модельных данных при помощи Python type hints.
4. Автодокументация ¬ FastAPI автоматически генерирует документацию для вашего API (Swagger и ReDoc) на основе аннотаций типов в коде.
5. **Методы и подходы к разработке.**

Перечень методов и подходов к разработке REST API, которые помогут создать эффективное, надежное и безопасное приложение:

1. Выбор архитектуры.

¬ RESTful архитектура: применение принципов REST, таких как безсостояние, использование стандартных HTTP методов и четкая структура URL.

¬ API first подход: Разработка API на начальном этапе проекта с последующим созданием клиентской части.

1. Проектирование ресурсов.

¬ Идентификация ресурсов: Четкое определение, какие данные (ресурсы) будут доступны через API и как они будут представлены (например, в виде JSON).

¬ Структурирование URL: Следование согласованным правилам для построения URL (‘/api/users’, ‘/api/products’).

1. Использование стандартных HTTP методов.

¬ GET: используется для получения данных.

¬ POST: Обработка создания ресурсов.

¬ PUT/PATCH: Обновление существующих ресурсов.

¬ DELETE: Удаление ресурсов.

1. Обработка версий API.

¬ Версионирование: Включение версии API в URL (например, ‘/api/v1/...’) или в заголовки запросов, чтобы управлять изменениями и улучшениями в API без нарушений для текущих пользователей.

1. Использование статус-кодов HTTP.

Применение соответствующих статус-кодов в ответах API для информирования клиента о результате операции:

¬ 200 OK: Запрос выполнен успешно.

¬ 201 Created: Ресурс создан.

¬ 204 No Content: Успешное выполнение запроса без возврата содержимого.

¬ 400 Bad Request: Неверный запрос.

¬ 404 Not Found: Ресурс не найден.

¬ 500 Internal Server Error: Ошибка на сервере.

1. Аутентификация и авторизация

¬ OAuth 2.0: Протокол для авторизации, часто применяемый для защищенных ресурсов.

¬ JWT (JSON Web Tokens): Метод аутентификации, основанный на токенах, который позволяет клиентам получать доступ к API.

1. Документирование API

¬ OpenAPI (Swagger): Использование спецификаций для документирования API, что позволяет пользователям легко понять, как взаимодействовать с API.

¬ Автоматическая генерация документации: В случае использования фреймворков (например, FastAPI), генерация документации может происходить автоматически на основе типов и описаний запросов.

1. Валидация входных данных

¬ Использование библиотек для валидации данных запросов (например, Pydantic для FastAPI или Django REST framework serializers).

¬ Проведение валидации на уровне API, чтобы предотвратить попадание недопустимых данных в систему.

1. Обработка ошибок

¬ Глобальная обработка ошибок: Определение структуры ответа об ошибках, чтобы клиент всегда знал, что произошло (например, содержащий код ошибки и сообщение).

¬ Логирование ошибок для последующего анализа и исправления проблем.

1. Кэширование

¬ Использование кэширования для повышения производительности и уменьшения нагрузки на сервер, особенно для часто запрашиваемых ресурсов. Например, кэширование ответов на уровне сервера или клиентского кэша.

1. Тестирование API

¬ Юнит¬тесты: Проверка отдельных компонентов на правильность.

¬ Интеграционные тесты: Проверка взаимодействия различных частей системы.

¬ API¬тесты: Использование инструментов (например, Postman, pytest) для тестирования функциональности API.

1. Мониторинг и аналитика

¬ Внедрение инструментов для мониторинга производительности API и аналитики использования (например, Google Analytics, Prometheus).

¬ Отслеживание запросов и анализ их производительности и нагрузки.

Использование этих методов и подходов поможет создать REST API, который будет надежным, масштабируемым и удобным для пользователей.

1. **Обзор инструментов и библиотек для разработки REST API на Python с использованием фреймворков Flask, Django и FastAPI**
   1. **Flask**

Основной фреймворк:

¬ Flask: Легковесный и гибкий веб¬фреймворк, идеален для быстрых прототипов и небольших проектов.

Расширения:

¬ Flask¬RESTful: Расширение для Flask, которое упрощает создание REST API, предоставляя классы для определения ресурсов и маршрутов.

Flask¬SQLAlchemy: Инструмент для работы с базами данных через SQLAlchemy, упрощает работу с моделями данных.

¬ Flask¬JWT¬Extended: позволяет легко добавлять аутентификацию на основе JSON Web Token (JWT) к вашему API.

¬ Flask¬Cors: упрощает настройку Cross¬Origin Resource Sharing (CORS) для вашего API, что необходимо для работы с клиентами на разных доменах.

¬ Flask¬Migrate: Расширение для управления миграциями базы данных, используя Alembic.

Инструменты для тестирования:

¬ pytest: Инструмент для тестирования, который можно использовать вместе с Flask для проверки функциональности API.

¬ Postman: Популярный инструмент для тестирования HTTP¬запросов, позволяет удобно отправлять запросы к вашему API и просматривать ответы.

* 1. **Django**

Основной фреймворк:

¬ Django: полноценный веб фреймворк, обеспечивающий множество встроенных функций для создания веб приложений.

Расширения:

¬ Django REST framework (DRF): Основное расширение для удобного создания REST API на Django, предоставляет сериализацию, аутентификацию, документацию и многие другие функции.

¬ Django ORM: Встроенный инструмент для работы с базами данных, позволяет легко взаимодействовать с моделями и выполнять запросы.

¬ Django Channels: Расширение для работы с асинхронными задачами и WebSocket протоколом, что позволяет делать API более интерактивным.

Инструменты для тестирования:

¬ Django Test Client: Встроенный инструмент для тестирования API, позволяющий имитировать HTTP¬запросы к вашему приложению.

¬ Postman: так же, как и в случае с Flask, используется для тестирования API.

* 1. **FastAPI**

Основной фреймворк:

¬ FastAPI: Современный, высокопроизводительный веб¬фреймворк для создания API с использованием Python type hints.

Инструменты и библиотеки:

¬ Pydantic: используется для валидации данных и определения схем. Обеспечивает удобный и понятный способ определения моделей данных.

¬ Starlette: на базе Starlette построен FastAPI. Этот фреймворк обеспечивает асинхронные функции и обработку запросов.

¬ FastAPI Users: поддерживает аутентификацию пользователей, включая различные механизмы (JWT, OAuth2 и др.).

¬ HTTPX: асинхронный HTTP-клиент, который можно использовать для выполнения запросов к API.

Автоматическая документация:

¬ Swagger и ReDoc: FastAPI автоматически генерирует документацию для вашего API, доступную по адресу ‘/docs’ и ‘/redoc’.

Инструменты для тестирования:

¬ pytest: как и для других фреймворков, pytest может быть использован для тестирования функциональности API.

¬ Postman: Инструмент для тестирования запросов и обработки ответов.

**Заключение:**

Каждый из этих фреймворков (Flask, Django и FastAPI) предоставляет уникальные инструменты и возможности, что позволяет разработчикам выбирать лучший подход в зависимости от требований проекта. Flask хорошо подходит для легковесных приложений, Django предлагает полноценное решение для больших систем, а FastAPI отражает современные тенденции разработки с акцентом на производительность и асинхронность. Все три фреймворка имеют активные сообщества и множество ресурсов, что делает их отличным выбором для разработки REST API на Python.

1. **Разработка REST API**
   1. **Подготовка среды**

Среда разработки, которая использовалась для реализации проектов является PyCharm.

Для эффективного управления зависимостями проектов устанавливается и активируется виртуальное окружение для каждого проекта. Оно помогает легко переносить проект с одного ПК на другой, а также делиться через интернет. Установка модуля для работы с виртуальным окружением:

¬ pip install virtualenv

Активация виртуального окружения:

¬ venv\Scripts\activate #Для Windows

**5.2. Установка необходимых библиотек для каждой технологии**

Установка Django и Django Rest Framework:

Установка Django: ¬ pip install Django

Создание проекта Django: ¬ django¬admin startproject myproject

Переход в папку проекта: ¬ cd myproject

Установка Django Rest Framework: ¬ pip install djangorestframework

Чтобы фреймворк работал необходимо его добавить в настройки проекта. Для этого открываем файл «warehouse\_crud/settings.py» и прописываем в настройках:

INSTALLED\_APPS = [

…,

'rest\_framework',

]

Установка FastAPI и Uvicorn

Для установки FastAPI лучше использовать команду: ­ pip install fastapi[all].

Эта команда позволяет установить все зависимости с нужными версиями, что позволяет безболезненно использовать FastAPI.

Установка Flask и Flask¬Restful

Для установки необходимо прописать две команды:

pip install flask

pip install flask¬restfull

1. **Реализация REST API**

**Описание REST API**.

Основная идея приложения — это взаимодействие пользователя со складом товаров (сервером) с помощью HTTP запросов на заказ (отгрузки) товаров. После регистрации с помощью POST-запроса на сервер информация о пользователе попадает в базу данных при использовании Django Rest Framework или во временное «хранилище» - в приложениях на FastAPI и Flask-restfull. Пользователь делает GET-запрос для получения списка товаров, имеющихся на складе, выбирает необходимые и добавляет в корзину (PUT-запрос). При необходимости он может удалить свою корзину выполнив DELETE-запрос. В данном проекте реализованы базовые CRUD приложения, а также методы администрирования склада как добавление, изменение параметров, удаление по id и очистка базы данных для пользователей и товаров.

* 1. **Пример реализации API с использованием Django Rest Framework**

При реализации API на Django использовались следующие библиотеки:

­ asgiref==3.8.1

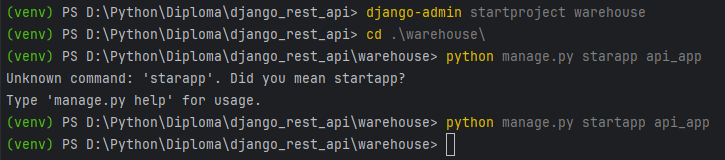
­ Django==5.1.2

­ djangorestframework==3.15.2

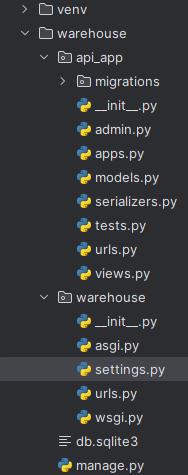
­ sqlparse==0.5.1

­ tzdata==2024.2

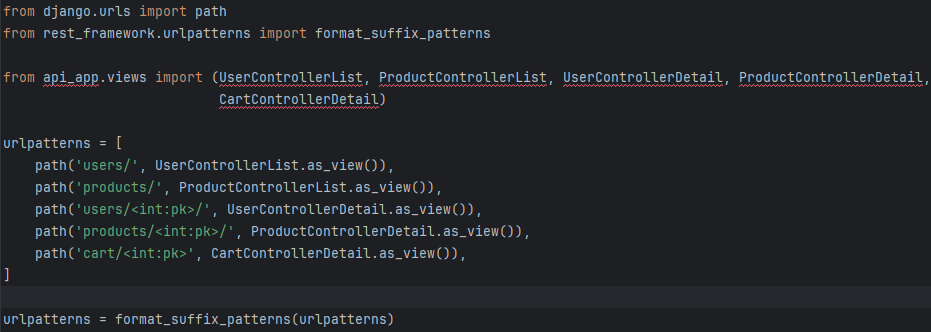
После установки нужных библиотек созданы проект и приложение.



Файловая структура приложения выглядит следующим образом:

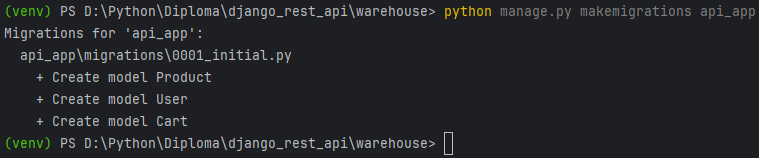


В файле models.py написаны модели миграции User, Product, Cart, после - сериализаторы, чтобы переводить наборы запросов и экземпляры моделей в JSON-данные. Сериализация также определяет, какие данные вернет API в ответ на запрос клиента. Созданы представления в файле views.py. Настроены маршруты:

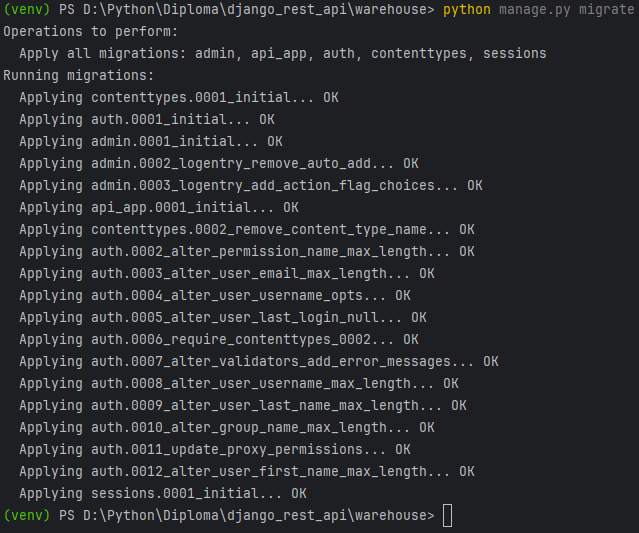


В настройках проекта settings.py подключены приложения 'api\_app' и 'rest\_framework'.

Созданы таблицы для моделей в базе данных для всех приложений из списка INSTALLED\_APPS. Делается инициализирующая миграция для моделей.



Применена миграция для всех приложений. База данных будет соответствовать созданным моделям.



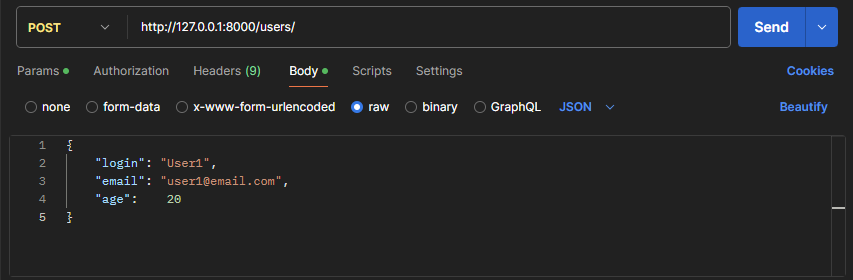
Базовый URL приложения: http//127.0.0.1:8000/

Сервер будет доступен для пользователей после запуска. Для этого необходимо в проекте зайти в Terminal и выполнить команду:

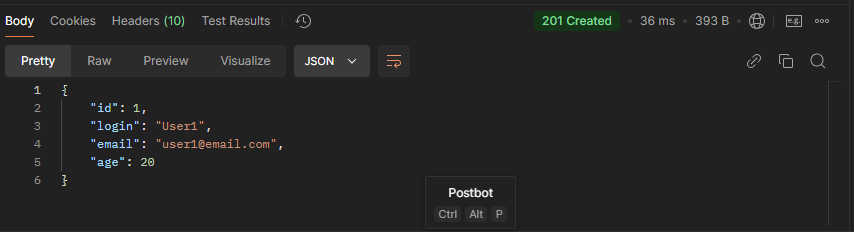
­ python manage.py runserver

Аутентификация пользователя реализована в данном приложении с использованием HTTP метода POST.

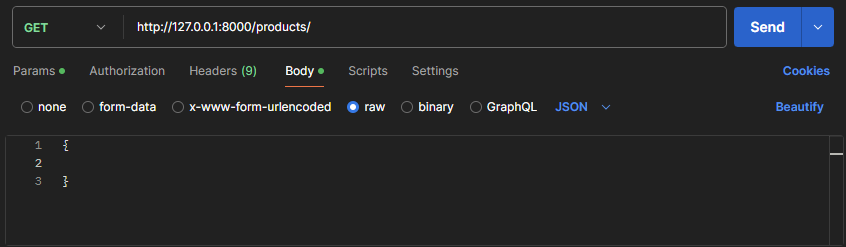
Пример использования POST-запроса в приложении Postman:



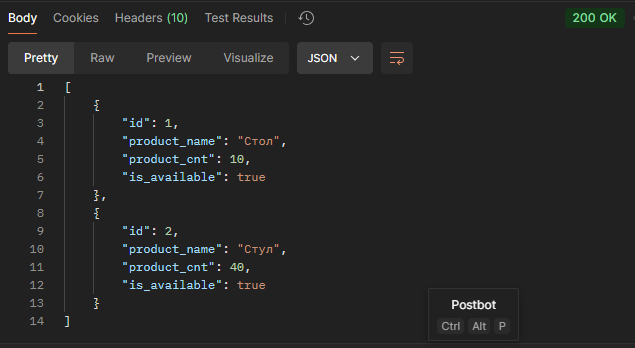
Результат запроса:



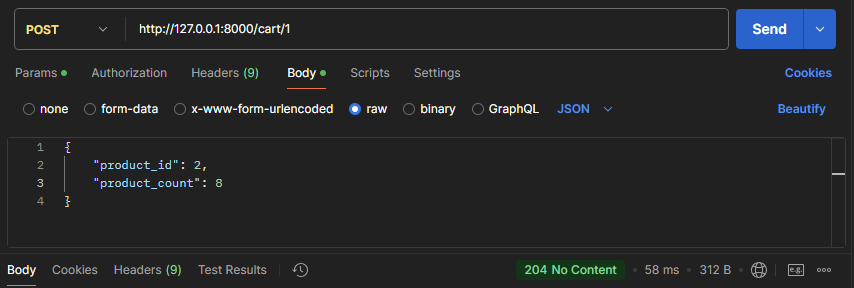
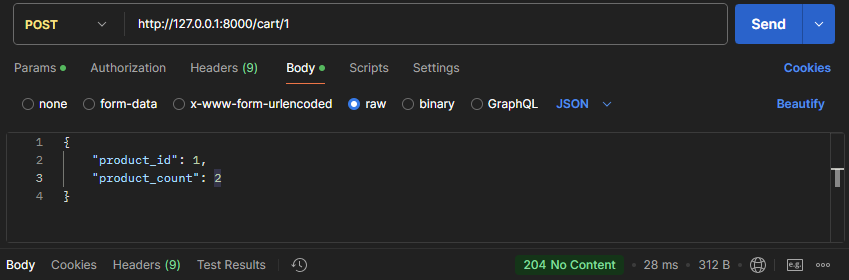
Для получения списка товаров на складе(сервере) пользователю необходимо выполнить следующий HTTP метод GET:



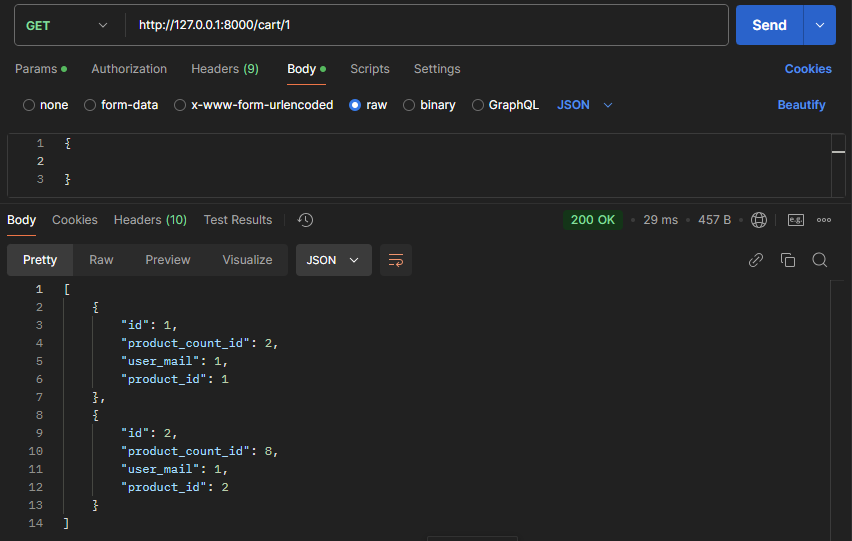
Результат запроса:



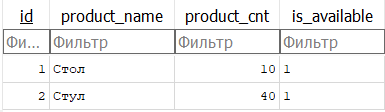
После получения списка товаров пользователь выбирает необходимые и добавляет в корзину, используя HTTP метод POST (в конце URL нужно указать id клиента, а в самом запросе указывается id и количество товара):

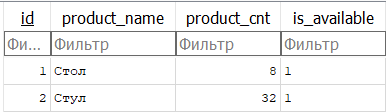


В результате товары добавлены в корзину. Для проверки корзины пользователю необходимо выполнить GET-запрос:



В базе данных видно, как изменилось число товаров в таблице.





* 1. **Пример реализации API с использованием FastAPI**

При реализации API на FastAPI использовались следующие библиотеки:

annotated-types==0.7.0

anyio==4.6.2.post1

certifi==2024.8.30

click==8.1.7

colorama==0.4.6

dnspython==2.7.0

email\_validator==2.2.0

fastapi==0.115.4

fastapi-cli==0.0.5

h11==0.14.0

httpcore==1.0.6

httptools==0.6.4

httpx==0.27.2

idna==3.10

itsdangerous==2.2.0

Jinja2==3.1.4

markdown-it-py==3.0.0

MarkupSafe==3.0.2

mdurl==0.1.2

orjson==3.10.11

pydantic==2.9.2

pydantic-extra-types==2.10.0

pydantic-settings==2.6.1

pydantic\_core==2.23.4

Pygments==2.18.0

python-dotenv==1.0.1

python-multipart==0.0.17

PyYAML==6.0.2

rich==13.9.4

shellingham==1.5.4

sniffio==1.3.1

starlette==0.41.2

typer==0.13.0

typing\_extensions==4.12.2

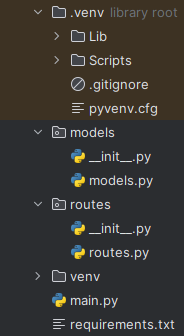
ujson==5.10.0

uvicorn==0.32.0

watchfiles==0.24.0

websockets==14.0

Файловая структура приложения выглядит следующим образом:



В файле models.py написаны pydantic модели и схемы User, Product, Cart. Создана маршрутизация запросов в файле routes.py. Файл main.py предназначен для настройки приложения, в нем определяется экземпляр класса FastAPI и подключаем пути маршрутов из routes.py и создаем пустые user\_list, product\_list, cart\_list («временное хранилище»).

Приложение написанное на FastAPI отличается от Django REST Framework, тем, что не базы данных, эмитируется пустыми списками, а также имеет автоматическая документация API (Swagger UI) и альтернативная документация API(ReDoc) без дополнительных настроек и подключений.

* 1. **Пример реализации API с использованием Flask¬RESTful**

При реализации API на Flask-RESTful использовались следующие библиотеки:

aniso8601==9.0.1

blinker==1.9.0

click==8.1.7

colorama==0.4.6

Flask==3.0.3

Flask-RESTful==0.3.10

itsdangerous==2.2.0

Jinja2==3.1.4

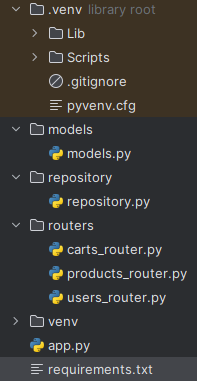
MarkupSafe==3.0.2

pytz==2024.2

six==1.16.0

Werkzeug==3.1.3

Файловая структура приложения выглядит следующим образом:



В файле models.py написаны модели и схемы User, Product, Cart. Создана маршрутизация запросов в файле users\_routes.py, products\_routes.py, carts\_routes.py. Файл app.py предназначен для настройки приложения, в нем определяется экземпляр класса Flask, инициализируется проект и подключаются пути маршрутов из users\_routes.py, products\_routes.py, carts\_routes.py. Написаны методы для выполнения функций соответствующие запросам в файле repository.py.

Приложение написанное на Flask-RESTful отличается так же от Django REST Framework, как и FastAPI, тем, что не базы данных, эмитируется пустыми списками.

**7. Сравнительный анализ**

Сравнительный анализ различных подходов к реализации REST API с использованием Django Rest Framework, FastAPI и Flask-Restful включает в себя несколько важных критериев. Ниже представлено несколько аспектов, по которым можно сравнить эти фреймворки:

1. Производительность

- FastAPI: Этот фреймворк создан с учетом асинхронного программирования, что позволяет ему обрабатывать большое количество параллельных запросов. FastAPI использует ASGI и основан на Starlette, что делает его одним из самых быстрых фреймворков для разработки API.

- Django Rest Framework (DRF): несмотря на то, что DRF предлагает множество функций, он основан на Django, который, как правило, медленнее, чем FastAPI, из-за своей синхронной природы. Однако, в последних версиях Django добавлена поддержка асинхронного программирования, что может улучшить производительность.

- Flask-Restful: Flask имеет более легковесную архитектуру и может быть настроен для оптимизации производительности, но не поддерживает полноценно асинхронные вызовы без сторонних библиотек (например, Gevent).

2. Легкость использования

- Django Rest Framework: из-за того, что это полный фреймворк, DRF имеет богатый набор встроенных возможностей для работы с аутентификацией, сериализацией и другими аспектами. Документация обширная и хорошо структурированная.

- FastAPI: имеет простой и современный интерфейс. Аннотации типов делают код более читаемым и облегчают валидацию данных. Документация FastAPI очень понятная и содержит много примеров.

- Flask-Restful: Flask достаточно прост и гибок. Он предоставляет минималистичный подход, что делает его легким для изучения. Однако, для того чтобы учитывать такие функции, как аутентификация и сериализация, может понадобиться множество дополнительных библиотек.

3. Функциональные возможности

- Django Rest Framework: поддерживает множество функций «из коробки», таких как аутентификация, защита от CSRF, поддержка различных форматов данных, админская панель и интеграция с ORM.

- FastAPI: поддерживает валидацию данных, автоматическую генерацию документации (Swagger), а также встроенные механизмы работы с зависимостями. Но требует дополнительных библиотек для полноценной работы с базами данных и аутентификацией.

- Flask-Restful: предоставляет базовые функции для создания RESTful API, но для расширенной функциональности может потребоваться интеграция дополнительных библиотек.

4. Сообщество и поддержка

- Django Rest Framework: имеет большое сообщество, множество обучающих материалов и плагинов. Подходит для крупных проектов и долгосрочной поддержки.

- FastAPI: Сообщество быстро растет. Несмотря на то, что он относительно новый, пользователи активно делятся примерами и обучающими материалами.

- Flask-Restful: Flask и его расширения, такие как Flask-Restful, имеют широкую базу пользователей и активную документацию, поскольку Flask был на рынке дольше, чем FastAPI.

5. Безопасность

- Django Rest Framework: обеспечивает различные механизмы аутентификации и авторизации, поддержку токенов и CSRF-защиту.

- FastAPI: предоставляет функционал для аутентификации и авторизации, но это требует ручной настройки и может нуждаться в дополнительных библиотеках.

- Flask-Restful: Безопасность будет зависеть от используемых дополнительных библиотек. Основной функционал для аутентификации нужно добавлять вручную.

6. Применение

- Django Rest Framework: рекомендуется для крупных проектов, где нужен полностью функциональный фреймворк с интеграцией ORM, администрирование и другими функциями.

- FastAPI: Идеален для современных приложений, где важна скорость и производительность, а также для проектов с асинхронными требованиями.

- Flask-Restful: подходит для небольших проектов и приложений, где необходимо быстро создать API без лишних затрат времени на настройку.

Заключение

Выбор фреймворка для разработки REST API зависит от конкретных требований проекта. Если цель — максимальная производительность и современный подход, стоит выбрать FastAPI. Если нужен полный функционал с поддержкой множества возможностей «из коробки», то Django Rest Framework будет лучшим выбором. Flask-Restful лучше подойдет для небольших приложений, где важна гибкость и легкость в настройках.

**Результаты тестирования производительности**

Проведены нагрузочные тесты для каждого из фреймворков. Postman использовался для проверки времени отклика и пропускной способности (requests per second).

Пример результатов

|  |  |  |
| --- | --- | --- |
| Фреймворк | Время отклика (мс) | Запросов в секунду |
| Django REST Framework | 200 | 50 |
| FastAPI | 50 | 200 |
| Flask-Restful | 120 | 80 |

Как видно из таблицы, FastAPI показывает наилучшую производительность, что подтверждает его преимущества в асинхронной обработке запросов. Django REST Framework, хотя и мощный, оказался значительно медленнее по сравнению с другими фреймворками.

**Удобство разработки**

- Django REST Framework: разработка с использованием DRF была громоздкой для маленького проекта, но было легко освоить благодаря его обширной документации и встроенным инструментам. Однако, из-за его функциональности, проект может оказаться более сложным в обслуживании.

- FastAPI: В FastAPI концепция аннотаций типов и автоматическая генерация документации значительно упростила процесс разработки, что позволило быстро проверить и задокументировать API.

- Flask-Restful: Flask оказался легким и быстрым в разработке малого проекта.

**Обсуждение функциональных возможностей**

- Django REST Framework: предоставляет множество встроенных функций, таких как аутентификация, администрирование и поддержка сериализации данных. Это делает его идеальным для больших проектов, но перегружает небольшие.

- FastAPI: отличается своей простой конфигурацией аутентификации и валидации данных, однако для сложных приложений может потребоваться больше усилий по интеграции сторонних библиотек.

- Flask-Restful: очень гибок, но часто требует дополнительных библиотек для стандартных функций, таких как аутентификация и работа с базами данных, что может увеличить время разработки.

**Поддержка и сообщество**

- Django REST Framework обладает обширным сообществом и множеством ресурсов, что делает его предпочтительным вариантом для долгосрочных проектов.

- FastAPI: быстро растущее сообщество и активное развитие, что обеспечивает его актуальность и поддержку.

- Flask-Restful также имеет широкую поддержку, однако сообщество менее активно по сравнению с Django и FastAPI.

**9. Заключение**

На основе представленных результатов и обсуждений, можно сформулировать несколько общих выводов о каждом из фреймворков:

- Если цель — создание высокопроизводительного API с минимальными задержками, FastAPI будет отличным выбором.

- Для проектов, требующих многофункциональности и готовых к расширению инструментов, лучше подойдет Django REST Framework.

- Flask-Restful рекомендуется для небольших проектов, где важна простота и возможность быстрой настройки.

Приложение: ссылка на репозиторию дипломного проекта

https://github.com/SChibisov/Diploma