



# Практическое занятие

## FastAPI - Основы веб-разработки

ФИО студента:

Тестовый

Группа:

ИСТ-21

Дата:

27.10.2025



# 1. Установка uv

## Что такое uv?

**uv** — менеджер пакетов Python на основе Rust, созданный для быстрой и надежной работы. Его основная цель — обеспечить максимально быструю установку, разрешение зависимостей и создание виртуальных окружений.

## Преимущества uv




### ▶ Производительность:

- Создание виртуальных окружений в ~80 раз быстрее, чем `python -m venv`
- Установка пакетов в 4-12 раз быстрее без кэширования
- Установка пакетов в ~100 раз быстрее с кэшированием

### ▶ Причины высокой скорости:

- Rust под капотом — компилируется в машинный код
- Параллельная обработка — одновременная загрузка и установка
- Оптимизированное разрешение зависимостей
- Эффективное кэширование

## Почему uv лучше для обучения?

- ▶  **Скорость:** В 10-100 раз быстрее Poetry, мгновенная установка зависимостей
- ▶  **Простота:** Один инструмент для всего: `uv init`, `uv add`, `uv run`
- ▶  **Современность:** Написан на Rust, поддерживает все современные стандарты Python

Установка на Windows:

```
# Способ 1: через winget
winget install astral-sh.uv

# Способ 2: через PowerShell
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

# Способ 3: через pip
pip install uv
```

Проверка установки:

```
uv --version
```

Версия uv на вашей системе:

0.5.11

uv установлен и работает

Основные команды uv

Команда	Описание
uv init <project-name>	Создает и инициализирует проект Python
uv venv	Создает виртуальную среду в текущем проекте

uv add <package-name>	Добавляет пакет к зависимостям проекта
uv pip install -r requirements.txt	Устанавливает зависимости из requirements.txt
uv remove <package-name>	Удаляет пакет из зависимостей
uv run script.py	Запускает скрипт Python внутри проекта
uv sync	Устанавливает все зависимости из uv.lock

Изучил команды uv



## 2. Создание проекта FastAPI

### Создание проекта с uv:

```
# Создание проекта
uv init my-api
cd my-api

# Добавление зависимостей
uv add fastapi uvicorn httpie

# Запуск
uv run uvicorn main:app --reload
```

Проект создан через uv init

### Структура проекта:

```
my-api/
├── .git/           # папка для работы с Git
├── main.py         # автосгенерированный файл приложения
├── pyproject.toml  # файл конфигурации проекта
├── README.md       # описание проекта
└── .python-version # номер применяемой версии Python
```

Изучил структуру проекта

## Установка виртуального окружения через uv:

```
# Создание виртуального окружения
python -m uv venv

# Активация (Windows)
.\.venv\Scripts\activate

# Добавление пакета
python -m uv add matplotlib
```

FastAPI и зависимости установлены

## Пример: График с matplotlib

```
import matplotlib.pyplot as plt

def main():
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
    rates = [13.58, 15.33, 14.26, 13.92, 12.74, 12.2,
             13.44, 15.63, 15.74, 17.08, 17.18, 16.66]

    plt.xlabel("month")
    plt.ylabel("rate")
    plt.plot(months, rates)
    plt.show()

if __name__ == "__main__":
    main()
```

Протестировал пример с matplotlib



## 2.1. Введение в FastAPI

### Обзор

**FastAPI** — это современный веб-фреймворк для создания API на Python, который отличается:

- ▶ Высокой производительностью
- ▶ Простотой использования
- ▶ Современным подходом к разработке

### Основные характеристики

- ▶ **Стандарты и совместимость:**
  - Основан на стандартах OpenAPI и JSON Schema
  - Автоматически генерирует документацию для API
  - Совместим со стандартами OAuth 2.0 и JWT
- ▶ **Производительность:**
  - Поддерживает асинхронное программирование из коробки
  - Позволяет обрабатывать множество запросов одновременно
  - Имеет отличную производительность, сравнимую с Node.js и Go

### Технические особенности

- ▶ Использует аннотации типов Python для валидации данных
- ▶ Код становится более читаемым и надежным



- ▶ Обеспечивает автоматическую сериализацию данных в JSON
- ▶ Построен на основе Starlette и Pydantic
- ▶ Поддерживает интерактивную документацию Swagger UI
- ▶ Доступна альтернативная документация в ReDoc

Изучил особенности FastAPI

### 3. Первое приложение FastAPI

Файл: main.py

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/hi")
def greet():
    return "Hello? World?"
```

#### Ключевые элементы:

- ▶ **app** — объект FastAPI верхнего уровня, представляющий веб-приложение
- ▶ **@app.get("/hi")** — декоратор пути. Он сообщает FastAPI:
  - Запрос к URL /hi на этом сервере должен быть направлен на следующую функцию
  - Применяется только к HTTP-глаголу GET
  - Можно создать отдельные функции для других глаголов (PUT, POST, DELETE)
- ▶ **def greet()** — функция пути, основная точка контакта с HTTP-запросами

#### Запуск сервера:

Способ 1: Через командную строку

```
uvicorn hello:app --reload
```

Запустил через командную строку

## Способ 2: Внутри приложения

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/hi")
def greet():
    return "Hello? World?"

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", reload=True)
```

### Параметры:

- ▶ main:app — файл main.py, переменная app
- ▶ --reload — автоматическая перезагрузка при изменении файла
- ▶ По умолчанию: localhost:8000

Сервер запущен успешно



## 4. Тестирование API

### Способ 1: Браузер

```
http://localhost:8000/hi
```

Протестировал через браузер

### Способ 2: Requests

```
import requests

r = requests.get("http://localhost:8000/hi")
print(r.json()) # 'Hello? World?'
```

### Способ 3: HTTPX

```
import httpx

r = httpx.get("http://localhost:8000/hi")
print(r.json()) # 'Hello? World?'
```

### Способ 4: HTTPie

```
# Полный вывод
http localhost:8000/hi
```

#### Результат HTTPie:

```
HTTP/1.1 200 OK
content-length: 15
content-type: application/json
date: Mon, 27 Oct 2025 14:13:56 GMT
server: uvicorn

'Hello? World?'
```

Протестировал через HTTPie

#### Опции HTTPie:

```
# Только тело ответа
http -b localhost:8000/hi

# Полные данные (запрос + ответ)
http -v localhost:8000/hi
```

#### Опишите результат http -v:

GET /hi HTTP/1.1 - это HTTP-запрос, состоящий из трех частей...  
Accept: \*/\* - это ...  
Accept-Encoding: gzip, deflate - это ...

Изучил опции HTTPie

## Автоматическая документация

FastAPI автоматически генерирует интерактивную документацию API:

- ▶ **Swagger UI:** <http://localhost:8000/docs>
- ▶ **ReDoc:** <http://localhost:8000/redoc>

Открыл Swagger UI и ReDoc



## 5. Параметры в FastAPI

### Компоненты HTTP-запроса:

- ▶ Глагол-оператор (GET) и путь (/hi)
- ▶ Параметры запроса (после ?)
- ▶ HTTP-заголовки
- ▶ Тело запроса

### Определения FastAPI:

- ▶ **Header** — HTTP-заголовки
- ▶ **Path** — URL-адрес
- ▶ **Query** — параметры запроса (после ?)
- ▶ **Body** — тело HTTP-сообщения

### Способы передачи параметров:

- ▶ В пути URL
- ▶ В качестве параметра запроса после символа ? в URL
- ▶ В теле HTTP-сообщения
- ▶ В HTTP-заголовке

### 5.1. Параметры пути (Path Parameters)

**Файл: main.py**

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/hi/{who}")
def greet(who):
    return f"Hello? {who}?"
```

**Тестирование:**

http localhost:8000/hi/Pushkin

**Результат:**

```
HTTP/1.1 200 OK
content-length: 13
content-type: application/json
date: Mon, 27 Oct 2025 14:21:54 GMT
server: uvicorn
```

'Hello? Pushkin?'

Выполнено



## ? 5.2. Параметры запроса (Query Parameters)

Параметры запроса — это строки name=value после символа ? в URL-адресе, разделенные символами &.

**Файл: main.py**

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/hi")
def greet(who):
    return f"Hello? {who}?"
```

**Тестирование:**

```
# Браузер
localhost:8000/hi?who=Pushkin

# HTTPie (способ 1)
http -b localhost:8000/hi?who=Pushkin

# HTTPie (способ 2)
http -b localhost:8000/hi who==Pushkin
```

**Python Requests:**

```
import requests
```

```
# Способ 1
r = requests.get("http://localhost:8000/hi?who=Pushkin")
print(r.json())

# Способ 2
params = {"who": "Pushkin"}
r = requests.get("http://localhost:8000/hi", params=params)
print(r.json())
```

Чем отличается `http localhost:8000/hi?who=Pushkin` от `http localhost:8000/hi/Pushkin`? Когда применяются эти способы?

Напишите отСоздайте эндпоинт, который приветствует пользователя, используя данные из разных источников.

Используйте:вет здесь...

Выполнено

## 5.3. Тело запроса (Request Body)

### Идемпотентность GET-запросов

**Важно:** GET-запросы должны быть идемпотентными (один и тот же вопрос — один и тот же ответ). GET должен только возвращать данные.

Тело запроса используется для:

- ▶ **POST** — создание данных
- ▶ **PUT** — полное обновление
- ▶ **PATCH** — частичное обновление

### Файл: main.py

```
from fastapi import FastAPI, Body

app = FastAPI()

@app.post("/hi")
def greet(who: str = Body(embed=True)):
    return f"Hello? {who}?"
```

### Тестирование:

```
# HTTPie
http -v localhost:8000/hi who=Pushkin
```

### Python Requests:

```
import requests

data = {"who": "Pushkin"}
r = requests.post("http://localhost:8000/hi", json=data)
print(r.json())
```

Выполнено

## 5.4. HTTP-заголовки (Headers)

Файл: main.py

```
from fastapi import FastAPI, Header

app = FastAPI()

@app.post("/hi")
def greet(who: str = Header()):
    return f"Hello? {who}?"

@app.get("/agent")
def get_agent(user_agent: str = Header()):
    return user_agent
```

Тестирование:

```
# HTTPie
http -v localhost:8000/hi who:Pushkin
http -v localhost:8000/agent
```

Результат /agent:

HTTPie/3.2.4

### Преобразование заголовков

FastAPI автоматически:

- ▶ Переводит ключи заголовков в нижний регистр
- ▶ Преобразует дефис (-) в нижнее подчеркивание (\_)

Выполнено



## 6. Комбинирование источников данных

В одной функции пути можно использовать несколько методов одновременно:

- ▶ URL-параметры (Path)
- ▶ Параметры запроса (Query)
- ▶ Тело запроса (Body)
- ▶ HTTP-заголовки (Header)
- ▶ Cookie-файлы

### Рекомендации по выбору метода:

Метод	Когда использовать
Path	Идентификация ресурса (RESTful)
Query	Дополнительные аргументы (пагинация, фильтры)
Body	Большие объемы данных, модели
Header	Метаданные, аутентификация

**Важно:** При использовании подсказок типов, аргументы автоматически проверяются библиотекой Pydantic.



Изучил комбинирование параметров



## 7. Коды состояния HTTP

```
@app.get("/happy", status_code=200)
def happy():
    return ":) "
```

### Тестирование:

http localhost:8000/happy

Выполнено





## 8. Установка заголовков ответа

```
from fastapi import FastAPI, Response

app = FastAPI()

@app.get("/header/{name}/{value}")
def header(name: str, value: str, response: Response):
    response.headers[name] = value
    return "normal body"
```

### Тестирование:

http localhost:8000/header/marco/polo



Выполнено



## 9. Типы ответов

### Основные типы:

- ▶ **JSONResponse** (по умолчанию)
- ▶ **HTMLResponse**
- ▶ **PlainTextResponse**
- ▶ **RedirectResponse**
- ▶ **FileResponse**
- ▶ **StreamingResponse**

### Пользовательский Response:

```
from fastapi import Response

@app.get("/custom")
def custom():
    return Response(
        content="Custom content",
        media_type="text/plain",
        status_code=200,
        headers={"X-Custom": "Header"}
    )
```



Выполнено

## 10. Самостоятельные задания

### Задание 1: Приветствие пользователя

Создайте эндпоинт, который приветствует пользователя, используя данные из разных источников.

#### Используйте:

- ▶ **Path:** имя пользователя
- ▶ **Query:** язык приветствия (ru, en)
- ▶ **Body:** возраст пользователя
- ▶ **Header:** город пользователя

#### Как тестировать:

```
# HTTPie
http POST localhost:8000/greet/Ivan lang==ru age:=25 city:Moscow

# Ожидаемый ответ:
{
  "greeting": "Привет, Ivan!",
  "age": 25,
  "city": "Moscow",
  "language": "ru"
}
```

#### Решение:

```
import matplotlib.pyplot as plt

def main():
    months = [class="string">"Jan", class="string">"Feb", class="string">"Mar", class="string">"Apr", class="string">"May", class="string">"Jun",
              class="string">"Jul", class="string">"Aug", class="string">"Sep", class="string">"Oct", class="string">"Nov", class="string">"Dec"]
    rates = [13.58, 15.33, 14.26, 13.92, 12.74, 12.2,
             13.44, 15.63, 15.74, 17.08, 17.18, 16.66]

    plt.xlabel(class="string">"month")
    plt.ylabel(class="string">"rate")
    plt.plot(months, rates)
    plt.show()
```

Выполнено

## Задание 2: Калькулятор с настройками

Создайте эндпоинт-калькулятор, который выполняет операции с числами.

### Используйте:

- ▶ **Path:** операция (add, subtract, multiply, divide)
- ▶ **Query:** первое число
- ▶ **Body:** второе число
- ▶ **Header:** количество знаков после запятой

**Как тестировать:**

```
# HTTPie - сложение
http POST localhost:8000/calc/add num1==10.5 num2:=5.3 precision:3

# HTTPie - умножение
http POST localhost:8000/calc/multiply num1==7 num2:=8 precision:0

# Ожидаемый ответ:
{
  "operation": "multiply",
  "num1": 7.0,
  "num2": 8.0,
  "result": 56.0,
  "precision": 0
}
```

**Решение:**

```
import matplotlib.pyplot as plt

def main():
    months = [class="string">"Jan", class="string">"Feb", class="string">"Mar", class="string">"Apr", class="string">"May", class="string"
              class="string">"Jul", class="string">"Aug", class="string">"Sep", class="string">"Oct", class="string">"Nov", class="string"
    rates = [13.58, 15.33, 14.26, 13.92, 12.74, 12.2,
             13.44, 15.63, 15.74, 17.08, 17.18, 16.66]

    plt.xlabel(class="string">"month")
    plt.ylabel(class="string">"rate")
    plt.plot(months, rates)
    plt.show()
```

Выполнено

## Выводы по занятию

### На данном занятии мы изучили:

Используйте:Создайте эндпоинт, который приветствует пользователя, используя данные из разных источников.

Используйте:Создайте эндпоинт, который приветствует пользователя, используя данные из разных источников.

### Что нового узнал(а):

источников.

Используйте:Создайте эндпоинт, который приветствует пользователя, используя данные из разных источников.

### Что было трудно для понимания:

источников.

Используйте:Создайте эндпоинт, который приветствует пользователя, используя данные из разных источников.