FastAPI Project

What is FastAPI?

FastAPI is a modern, high-performance Python web framework designed for building APIs quickly and efficiently. It is lightweight, scalable, and leverages Python's type hints to provide a developer-friendly experience.

- **Purpose**: FastAPI simplifies the creation of RESTful APIs for web and mobile applications, microservices, and more.
- **Example**: It can power an API that delivers user data or product information to a mobile app.

Key Features

1. **High Performance**:
    - Built on **Starlette** and **Pydantic**, FastAPI offers performance comparable to Node.js and Go, thanks to its asynchronous capabilities (`async`/`await`).
2. **Automatic Documentation**:
    - Generates interactive API documentation (Swagger UI/ReDoc) accessible at `http://localhost:8000/docs`.
3. **Data Validation**:
    - Uses **Pydantic** for automatic data validation, ensuring correct data types and reducing errors.
4. **Developer-Friendly**:
    - Easy-to-write code with Python type hints, clear error messages, and minimal boilerplate.

Use Cases

- **Web Application Backends**: APIs for mobile apps, web apps, or IoT devices.
- **Microservices**: Independent services for payment systems, authentication, etc.
- **Machine Learning**: Deploy ML models via APIs (e.g., image recognition).
- **Prototyping**: Rapidly test and build proof-of-concept APIs.
- **Real-Time Apps**: Support for chat apps or live dashboards with async features.

Why FastAPI?

FastAPI was created by **Sebastián Ramírez** in **December 2018** to address the limitations of older frameworks like Flask and Django REST. Its goals were to:

- Deliver **high performance** using modern Python features (Python 3.6+).
- Be **easy to use** with minimal code for maximum output.
- Provide **automatic features** like documentation, validation, and error handling.

Today, FastAPI is widely used by companies like Netflix, Uber, and Microsoft due to its speed, simplicity, and robust ecosystem.

Why It's Useful

- **Speed**: Asynchronous support ensures fast request handling.
- **Ease of Learning**: Intuitive for Python developers.
- **Auto-Generated Docs**: No need to write separate documentation.
- **Type Safety**: Reduces bugs with Python type hints.
- **Strong Community**: Integrates with tools like Uvicorn and Pydantic.

Project Setup

To set up a FastAPI project using **UV** (a Python package and project manager), follow these steps:

1. **Install UV**:

For Windows:
pip install uv

OR
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

○

For Mac/Linux:
curl -LsSf https://astral.sh/uv/install.sh | sh

○

Create a Project Directory:

uv init fastapi-project

cd fastapi-project

2.
3. **Create and Activate a Virtual Environment**:

On macOS/Linux:
uv venv

source .venv/bin/activate

-

On Windows:
uv venv

.venv\Scripts\activate

-

Install Dependencies:
Install FastAPI, Uvicorn, and testing tools:

uv add "fastapi[standard]"

uv add --dev pytest pytest-asyncio

This updates `pyproject.toml` with:

```
[project]
name = "fastapi-project"
version = "0.1.0"
description = "A simple FastAPI project"
readme = "README.md"
requires-python = ">=3.8"
dependencies = [
    "fastapi[standard]>=0.115.0"
]
[dependency-groups]
dev = [
    "pytest>=8.3.2",
    "pytest-asyncio>=0.23.8",
]
```

4.

Create a Basic API:
Edit `main.py` to include a simple FastAPI application with GET and POST endpoints:

```
from fastapi import FastAPI

from pydantic import BaseModel


app = FastAPI()


class User(BaseModel):

    name: str

    age: int


@app.get("/")

async def read_root():

    return {"message": "Hello from FastAPI + UV!"}


@app.post("/users/")

async def create_user(user: User):

    return {"message": f"User {user.name} created with age {user.age}"}
```

5.

Running the Application

Start the Server:
Run the FastAPI development server:

```
fastapi dev main.py
```

Alternatively, use Uvicorn directly:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

1.
2. **Test the APIs**:
   Open your browser and visit:
   - `http://localhost:8000` (returns `{"message": "Hello from FastAPI + UV!"}`)
   - `http://localhost:8000/users/` (test with a POST request via Swagger UI)
3. **Explore Interactive Docs**:
   Visit `http://localhost:8000/docs` to interact with the API using Swagger UI.

Testing APIs with Swagger UI

FastAPI automatically generates an interactive testing interface at `http://localhost:8000/docs`. With Swagger UI, you can:

- View all available API routes.
- Test endpoints directly in the browser without writing code.
- See request parameters and response formats.

Example:

- Go to `http://localhost:8000/docs`.
- Find the `/users/` POST endpoint.

Click "Try it out", enter a JSON payload like:
{

"name": "Alice",

"age": 25

}

-
- Click "Execute" to see the response: `{"message": "User Alice created with age 25"}`.

Troubleshooting Common Issues

1. **Port Already in Use**:
   - Error: `Address already in use`.

Solution: Change the port with `--port 8001` or stop the process using the port:
lsof -i :8000

kill -9 <PID>

2. **Module Not Found**:
    - Error: `ModuleNotFoundError: No module named 'fastapi'`.

Solution: Ensure the virtual environment is activated and FastAPI is installed:
uv add "fastapi[standard]"

3. **Invalid JSON Payload**:
    - Error: `422 Unprocessable Entity` in Swagger UI.
    - Solution: Check the JSON format and ensure required fields (e.g., `name`, `age`) are included.

ur code follows PEP 8 style guidelines and includes tests where applicable.

Conclusion

FastAPI is a powerful, fast, and user-friendly framework that simplifies API development. Since its release in 2018, it has become a go-to choice for building modern, high-performance APIs. This project provides a starting point for building your own APIs with FastAPI, UV, and Uvicorn.

For more details, visit the FastAPI documentation.

License

This project is licensed under the MIT License. See the `LICENSE` file for details.