FastAPI - Introduction

Last Updated : 11 Jul, 2025

FastAPI is a modern and high-performance Python web framework used to build APIs quickly and efficiently. Designed with simplicity it allows developers to create RESTful APIs using Python's type hints which also enable automatic validation and error handling.

One of FastAPI’s key features is its ability to generate interactive API documentation automatically making it easier to test and understand API endpoints. It is an ideal choice for beginners and professionals who want to build fast, secure and scalable web applications with minimal effort.

Features of FastAPI

Automatic Documentation: FastAPI auto-generates interactive API docs using OpenAPI standard making it easy to test and understand your APIs.
Python Type Hints: uses type hints to validate inputs and generate docs automatically making code more readable and less error-prone.
Data Validation: FastAPI uses Pydantic models to validate and serialize/deserialize request data automatically.
Asynchronous Support: Supports async and await allowing non-blocking code for better performance in I/O-heavy apps.
Dependency Injection: Built-in support for dependency injection helps keep code modular, testable and clean.
Built-in Security: Includes OAuth2, JWT and request validation to protect against common security issues.

Installation and Setup of FastAPI

To get started with FastAPI:

1. Install Python 3: Make sure Python 3.7 or above is installed.

2. Install FastAPI: Use pip to install the FastAPI library

```
pip install fastapi
```

3. Install Uvicorn (ASGI server): Uvicorn is a lightweight server used to run FastAPI apps

```
pip install uvicorn
```

Create a Simple API

Example 1:

Let's create a simple web service that responds with "Hello, FastAPI!" when a specific URL is accessed. Using FastAPI, this can be achieved in just a few lines of code. The code is saved in a Python file named main.py.

```
from fastapi import FastAPI

# Create a FastAPI application
app = FastAPI()

# Define a route at the root web address ("/")
@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI!"}
```

Explanation: @app.get("/") tells FastAPI to handle HTTP GET requests sent to the root URL (/) by executing the function below it (read_root).

Now, run the following command to start the FastAPI app with live reload enabled:

```
uvicorn main:app --reload
```

Once the application is running, open your web browser and navigate to

http://localhost:8000/

You should see a message displayed in your browser or the response if you are using an API testing tool like curl or Postman.

```
{"message": "Hello, FastAPI!"}
```

Example 2:
Here, a simple POST API is created using FastAPI. When a user sends a name to /greet endpoint, the server responds with a personalized greeting message.

```
from fastapi import FastAPI

app = FastAPI()

# POST endpoint at /greet
@app.post("/greet")
def greet_user(name: str):
    return {"message": "Hello, " + name + "!"}
```

Explanation: This code creates a POST endpoint /greet that takes a string name and returns a greeting message like "Hello, name!" as JSON.

Similarly, we can perform different CRUD operations like PUT, PATCH and DELETE using FastAPI.

Advantage of FastAPI
Easy to Learn: Beginner-friendly syntax with automatic docs makes it quick to get started.
High Performance: Built on async features, it handles many requests efficiently.
Auto Data Validation: Uses type hints to validate input/output automatically.
Built-in Auth: Supports JWT, OAuth2 and custom authentication easily.
Middleware Support: Add logging, auth and more through simple middleware integration.
Disadvantage of FastAPI
Async Complexity: Requires understanding async/await, which can be tricky for beginners.
Fewer Built-in Tools: Lacks features like Django’s admin panel, ORM or user auth.
Smaller Ecosystem: Fewer plugins, tutorials and community support compared to older frameworks.
Debugging Difficulty: Async debugging is harder than with traditional synchronous code.