Getting Started

Introduction

FastAPI is a modern, high-performance web framework for building APIs using Python. Designed with efficiency and developer productivity in mind, it takes full advantage of Python's type hints to provide automatic validation, serialization, and robust error handling. FastAPI is particularly well-suited for building RESTful APIs, microservices, and backend services for real-time applications.

Core Features

- Asynchronous Programming
- Automatic Interactive API Documentation
- Type-Driven Development
- Data Validation with Pydantic
- Built-in Dependency Injection
- Security and Authentication

Tutorials

Official: FastAPI

PyImageSearch

- Getting Started with Python and FastAPI: A Complete Beginner's Guide - PyImageSearch
- Deploying a Vision Transformer Deep Learning Model with FastAPI in Python - PyImageSearch
- Preparing FastAPI for Production: A Comprehensive Guide | by Raman Bazhanau | Medium

Getting Started

Installation

- Installs **FastAPI**, the framework we'll use to build our APIs.

- Installs **Uvicorn**, the Asynchronous Server Gateway Interface (ASGI) that will run the FastAPI app and serve it to clients.
- Installs **PyTest**, a testing framework that allows us to write and run test cases for our FastAPI endpoints efficiently.

```
pip install fastapi uvicorn pytest
```

Verify the installation

```
python -m uvicorn --help
```

Running a Basic Server

main.py: (NOTE: ????????????? `fastapi.py`)

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
async def read_root():
    return {"message": "Hello, World!"}
```

Command to start the API Server

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

Access the Application : `http://127.0.0.1:8000/`

Example Code

Quick Test

```
from fastapi import FastAPI, File, UploadFile, HTTPException


app = FastAPI()


# http://127.0.0.1:8000/
@app.get("/")
async def read_root():
    return {"message": "Hello, World!"}


# http://127.0.0.1:8000/square?num=3
```

```
@app.get("/square")
async def calculate_square(num: int):
    return {"number": num, "square": num ** 2}

# http://127.0.0.1:8000/users/123
@app.get("/users/{user_id}")
async def read_user(user_id: int):
    return {"user_id": user_id}
```

Constraint/Validation

Pyantic: Base Constraint

```
from pydantic import BaseModel, Field

class User(BaseModel):
    name: str = Field(..., max_length=50)
    age: int = Field(..., gt=0, le=100)
    email: str = Field(..., regex="^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$")
```

Pyantic: Custom Error Messages

```
from pydantic import BaseModel, Field

class User(BaseModel):
    name: str = Field(..., max_length=50, description="Name must be under 50 characters")
    age: int = Field(..., gt=0, le=100, description="Age must be between 1 and 100")
    email: str = Field(..., regex="^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$",
description="Invalid email format")
```

Pyantic: Combining Multiple Constraints

```
from pydantic import BaseModel, Field

class User(BaseModel):
    username: str = Field(..., min_length=3, max_length=20, regex="^[a-zA-Z0-9_.]+$")
```

Pyantic: Integration with FastAPI Endpoints

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field

app = FastAPI()

class User(BaseModel):
    name: str = Field(..., max_length=50)
    age: int = Field(..., gt=0, le=100)
    email: str = Field(..., regex="^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$")

@app.post("/user/")
async def create_user(user: User):
    return user
```

Revision #28
Created 2025-12-17 13:56:56 CST by A-Lang (Admin)
Updated 2025-12-22 20:29:18 CST by A-Lang (Admin)