FastAPI

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
```

Upload File

Tutorials

- FastAPI File Upload Download Tutorial

Installation

```
pip install fastapi uvicorn python-multipart
```

Quick Test

```
# curl -X POST "http://127.0.0.1:8000/uploadfile/" -H "accept: application/json" -H "Content-
Type:multipart/form-data" -F "file=@your.pdf"
@app.post("/uploadfile/")
async def create_upload_file(file: UploadFile | None = None):
    if not file:
        return {"message": "No upload file sent"}
    else:
        return {"filename": file.filename}

@app.post("/tempfile/")
async def temp_file(file: UploadFile | None = None):
    with tempfile.NamedTemporaryFile(delete=False) as temp_file:
        content = await file.read()
        temp_file.write(content)
    temp_file.close

    return {"filepath": temp_file.name, "filename": file.filename, "content_type":
file.content_type, "size": file.size}
```

Client Test with curl

```
curl -X POST "http://127.0.0.1:8000/uploadfile/" -H "accept: application/json" -H "Content-
Type:multipart/form-data" -F "file=@/path/to/your.pdf"
```

Upload single file

```
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.responses import JSONResponse
from typing import List
import os
import shutil
from pathlib import Path

# Create upload directory
UPLOAD_DIR = Path("uploads")
UPLOAD_DIR.mkdir(exist_ok=True)

app = FastAPI(title="FastAPI File Upload Service")

@app.post("/upload/single")
async def upload_single_file(file: UploadFile = File(...)):
    """Upload a single file with basic validation"""
    if file.filename == "":
        raise HTTPException(status_code=400, detail="No file selected")

    file_path = UPLOAD_DIR / file.filename

    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    return {
        "filename": file.filename,
        "content_type": file.content_type,
        "size": file.size,
        "location": str(file_path)
    }
```

With validations

validators.py:

```
from pathlib import Path
from fastapi import UploadFile
```

```
class DocumentValidator:
    def __init__(self, max_size: int = 10 * 1024 * 1024):  # 10MB default
        self.max_size = max_size
        self.allowed_extensions = {'.pdf', '.txt', '.json'}

    async def validate_file(self, file: UploadFile) -> dict:
        """Check if the document file is valid"""
        result = {"valid": True, "errors": []}

        # Check if user selected a file
        if not file.filename or file.filename.strip() == "":
            result["valid"] = False
            result["errors"].append("No file selected")
            return result

        # Check file extension
        file_ext = Path(file.filename).suffix.lower()
        if file_ext not in self.allowed_extensions:
            result["valid"] = False
            result["errors"].append(
                f"File extension '{file_ext}' not allowed. Use: .pdf, .txt, or .json"
            )

        # Read file to check size
        content = await file.read()
        await file.seek(0)  # Reset file pointer for later use

        # Check file size
        file_size = len(content)
        if file_size > self.max_size:
            result["valid"] = False
            result["errors"].append(
                f"File too large ({file_size:,} bytes). Maximum: {self.max_size:,} bytes"
            )

        return result
```

main.py

```python
from fastapi import FastAPI, File, UploadFile, HTTPException
from fastapi.responses import JSONResponse
from typing import List
import os
import shutil
import uuid
from pathlib import Path
from datetime import datetime
from validators import DocumentValidator

# Create upload directory
UPLOAD_DIR = Path("uploads")
UPLOAD_DIR.mkdir(exist_ok=True)

app = FastAPI(title="FastAPI File Upload Service")

# Create validator instance
doc_validator = DocumentValidator(max_size=25 * 1024 * 1024)  # 25MB limit

@app.post("/upload/single")
async def upload_single_file(file: UploadFile = File(...)):
    """Upload a single file with validation"""

    # Validate the file first
    validation = await doc_validator.validate_file(file)

    if not validation["valid"]:
        raise HTTPException(
            status_code=400,
            detail={
                "message": "File validation failed",
                "errors": validation["errors"]
            }
        )

    # Create unique filename to prevent conflicts
    file_ext = Path(file.filename).suffix
    unique_filename = f"{uuid.uuid4()}{file_ext}"
    file_path = UPLOAD_DIR / unique_filename
```

```
    try:
        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)
    except Exception as e:
        raise HTTPException(
            status_code=500,
            detail=f"Failed to save file: {str(e)}"
        )

    return {
        "success": True,
        "original_filename": file.filename,
        "stored_filename": unique_filename,
        "content_type": file.content_type,
        "size": file.size,
        "upload_time": datetime.utcnow().isoformat(),
        "location": str(file_path)
    }

@app.get("/")
async def root():
    return {"message": "FastAPI File Upload Service is running"}
```

Deployment

Tutorials

- Sample FastAPI application for deployment in the production

Authentication

Tutorials

- [Authentication and Authorization with FastAPI: A Complete Guide | Better Stack Community]

Basic Authentication

auth.py

```
import secrets
from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import HTTPBasic, HTTPBasicCredentials

security = HTTPBasic()

def authenticate_user(credentials: HTTPBasicCredentials = Depends(security)):
    # In a real application, you'd verify against a database
    correct_username = secrets.compare_digest(credentials.username, "admin")
    correct_password = secrets.compare_digest(credentials.password, "secret")

    if not (correct_username and correct_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid credentials",
            headers={"WWW-Authenticate": "Basic"},
        )

    return credentials.username
```

main.py

```
from fastapi import FastAPI, Depends
from auth import authenticate_user

app = FastAPI(title="Authentication Demo", version="1.0.0")
```

```
@app.get("/")
async def root():
    return {"message": "Welcome to FastAPI Authentication Demo"}

@app.get("/protected")
async def protected_route(current_user: str = Depends(authenticate_user)):
    return {"message": f"Hello {current_user}, this is a protected route!"}
```