FastAPI

By: Sebastián Ramírez

Python web framework.

To run a FastAPI application, we need ASGI server.

- **Uvicorn**
- **Hypercorn**

Why These Servers?

Handles networking, concurrency, and I/O operations, allowing FastAPI to focus on the application logic.

- Asynchronous Nature: They efficiently handle multiple concurrent requests without creating a new thread for each, which is crucial for performance.
- Performance: These servers are optimized for speed and low resource consumption.
- Integration: They seamlessly integrate with FastAPI's asynchronous capabilities.

Starlette: The Foundation of FastAPI

Starlette is a lightweight ASGI framework/toolkit.
It provides the core components for handling HTTP requests, responses, and websockets.

FastAPI is built on top of Starlette and inherits many of Starlette's core functionalities and adds significant enhancements to create a more developer-friendly and high-performance framework.

Key benefits:
Solid foundation: Starlette provides a stable and efficient base for FastAPI.
Asynchronous capabilities: Both frameworks leverage Python's async/await syntax for optimal performance.
Core functionalities: FastAPI inherits essential features like request/response handling, middleware, and WebSocket support from Starlette.

Threads and Concurrency

**FastAPI uses a single thread to handle multiple requests concurrently.
This is made possible by using asynchronous libraries and frameworks like asyncio.**

Event Loop

The core of asynchronous programming is the event loop, which manages tasks and switches between them based on their readiness.

Async/Await

Python's async/await syntax makes asynchronous code more readable and easier to write.

Event Loop

An event loop is a programming construct that continuously monitors for and responds to events or messages. It's the backbone of asynchronous programming, allowing systems to handle multiple tasks concurrently without blocking the main thread.

01 Event Queue: A list of events waiting to be processed. These events can come from various sources like user interactions, network requests, timers, etc.

02 Call Stack: A data structure that keeps track of the currently executing functions. It operates on a LIFO (Last In, First Out) principle.

03 Callback Queue: A separate queue for storing callbacks that are ready to be executed.

orchestration

Stage 1

- ASGI server receives the incoming HTTP request.
- The request is parsed into an ASGI message, containing details like HTTP method, headers, and body.

Stage 2

- FastAPI receives message.
- Matches request path with defined route handlers.
- Parse and validate data with Pydantic.

Stage 3

- Route handler function is executed.
- Asynchronous operations are handled efficiently using Python's async/await syntax.

Stage 4

- The route handler returns a response object.
- Serialize data into a desired format.
- FastAPI constructs the HTTP response.

Stage 5

- FastAPI passes the response to the ASGI server.
- The server sends the response back to the client.

Strengths

- High Performance
- Asynchronous Support
- Editor support
- Data conversion

- Data Validation: Leveraging Pydantic, FastAPI offers robust data validation and serialization, reducing errors.

- Automatic Documentation: Generates interactive API documentation based on type hints, saving time and improving clarity.

- Dependency Injection: Built-in dependency injection simplifies code organization and testing.

Weaknesses

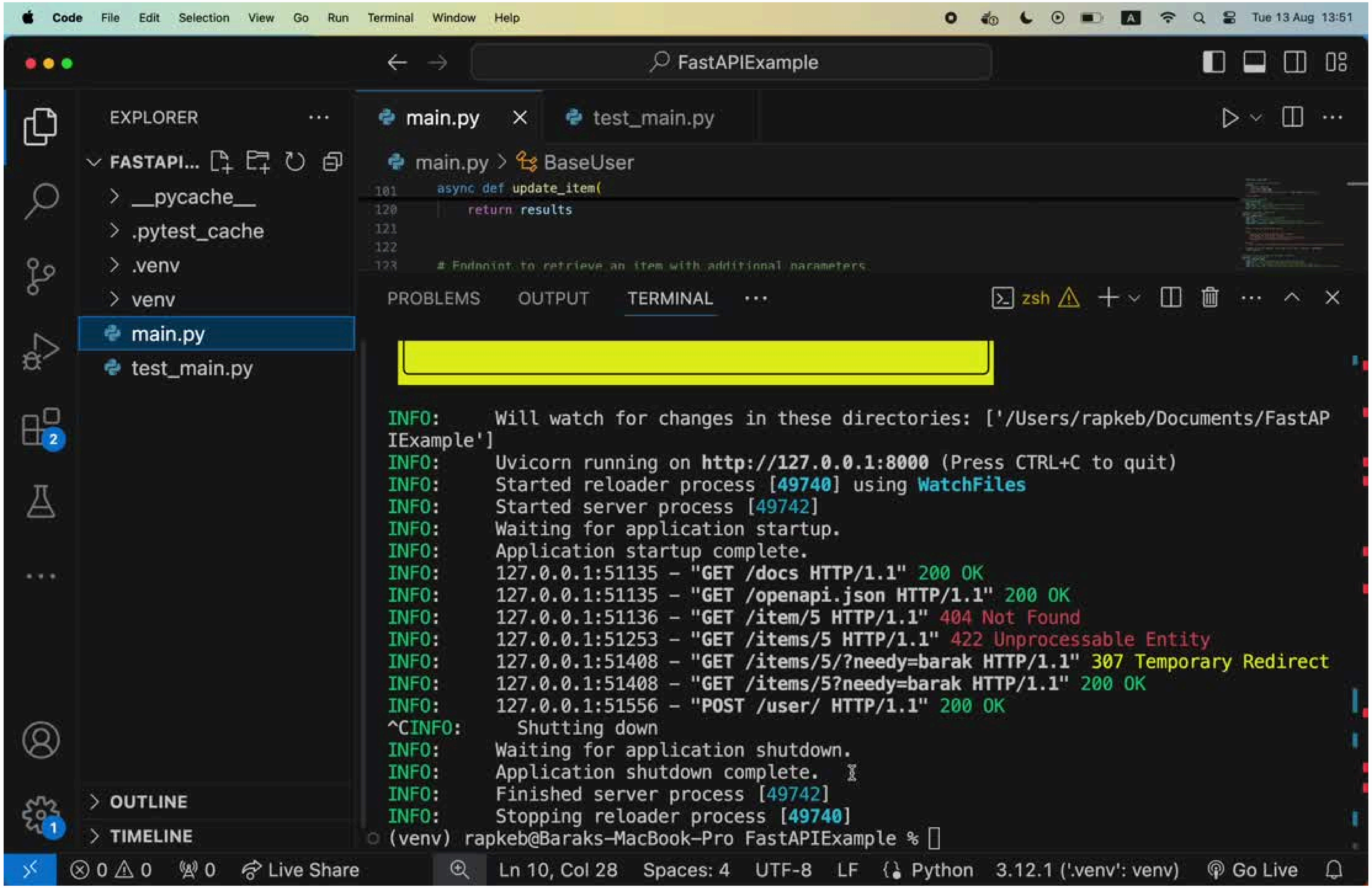
- Relative Immaturity: Compared to more established frameworks like Django, FastAPI has a smaller ecosystem and fewer third-party libraries.
- Less Mature ORM Support: Although SQLAlchemy integration is possible, FastAPI doesn't have a built-in ORM like Django.
- Smaller Community: While growing rapidly, the FastAPI community is still smaller than some other Python frameworks.

FastAPI - A High-Performance Python Framework.

FastAPI is a modern, high-performance Python web framework built on top of Starlette. It excels in building APIs thanks to its asynchronous nature and robust data validation.

At its core, FastAPI leverages an event loop to efficiently handle multiple requests concurrently. While it primarily operates asynchronously, it can also utilize thread pools for specific blocking tasks. This combination ensures optimal performance and responsiveness.

Key advantages of FastAPI include rapid development, automatic interactive documentation, and strong support for data validation through Pydantic. Its ideal use cases span from simple APIs to complex, data-intensive applications.



```python
async def update_item(
    return results


# Endpoint to retrieve an item with additional parameters
```

```
INFO:     Will watch for changes in these directories: ['/Users/rapkeb/Documents/FastAP
IExample']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [49740] using WatchFiles
INFO:     Started server process [49742]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:51135 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:51135 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:51136 - "GET /item/5 HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:51253 - "GET /items/5 HTTP/1.1" 422 Unprocessable Entity
INFO:     127.0.0.1:51408 - "GET /items/5/?needy=barak HTTP/1.1" 307 Temporary Redirect
INFO:     127.0.0.1:51408 - "GET /items/5?needy=barak HTTP/1.1" 200 OK
INFO:     127.0.0.1:51556 - "POST /user/ HTTP/1.1" 200 OK
^CINFO:     Shutting down
INFO:     Waiting for application shutdown.
INFO:     Application shutdown complete.
INFO:     Finished server process [49742]
INFO:     Stopping reloader process [49740]
(venv) rapkeb@Baraks-MacBook-Pro FastAPIExample %
```