LLMs in Action: Developing a COM AI Player for an Interactive Tic-Tac-Toe Game using OpenAI Developer API

Lesson 3: Building Modern Web Servers with Flask

Priscilla Emasoga

COM3550 Undergraduate Ambassadors Scheme
University of Sheffield

Overview

- **What is Flask?**
  - Why use Flask?
  - Alternative frameworks
- **The Tic-Tac-Toe Web Server**
  - Project structure & features
  - Key components deep dive
  - DEMO
- **Best Practices: Why an Additional Server was Necessary?**

What is Flask?

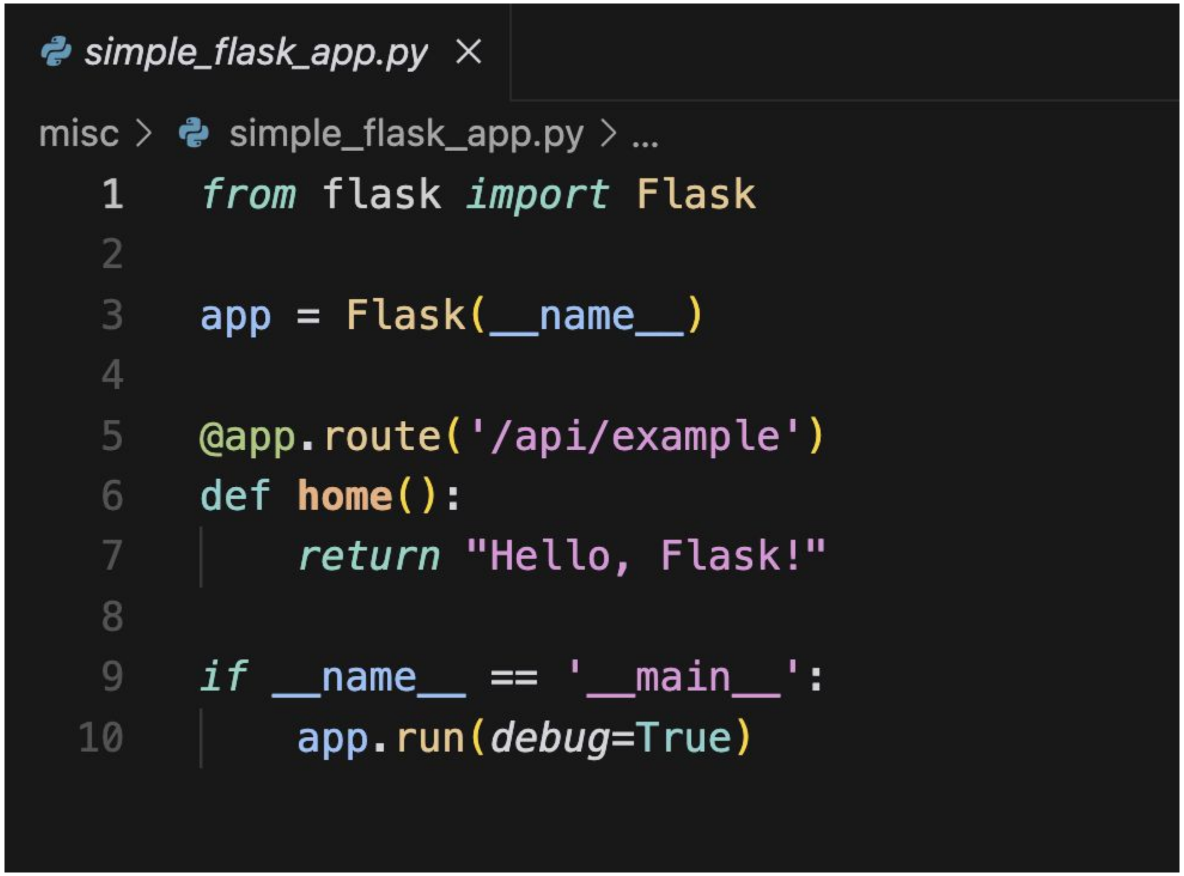
What is Flask?

- A micro web framework for Python
- Used for building APIs, web applications, and microservices.
  - Ideally can be used to build full-stack applications
  - We will focus on APIs
- Examples of companies using Flask: **Netflix**, **LinkedIn**, and **Pinterest**…

Why Flask?

- Simple and lightweight
- Easy to learn and implement – great for small projects.
- Built-in development server and debugger
- Production-ready with extensions
  - **Extensions** are additional libraries & packages
  - Installed to enhance functionality

Example: A Simple Flask Application



```python
from flask import Flask

app = Flask(__name__)

@app.route('/api/example')
def home():
    return "Hello, Flask!"

if __name__ == '__main__':
    app.run(debug=True)
```

Alternative frameworks

Some other alternative frameworks for web development in Python include:

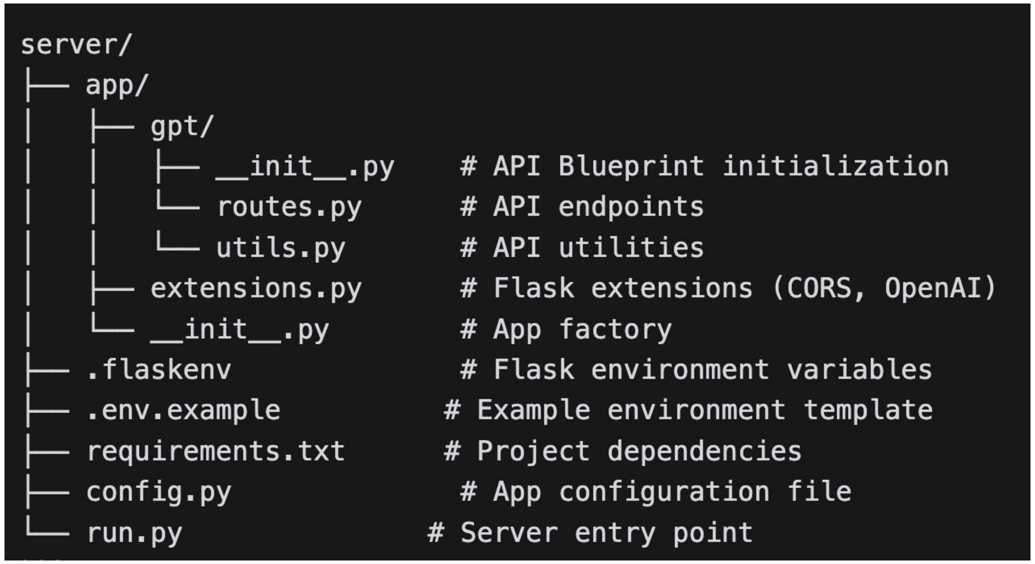
- **Django**
  - Older python full-stack web framework
    - Built-in features e.g Auth, ORM, admin panel,etc
  - Ideal for complex applications

- **Fast API**
  - Relatively new and fast rising
  - Better async support with ASGI
  - Best for high-performance APIs & real-time apps

The Tic-Tac-Toe Web Server

Project structure & features

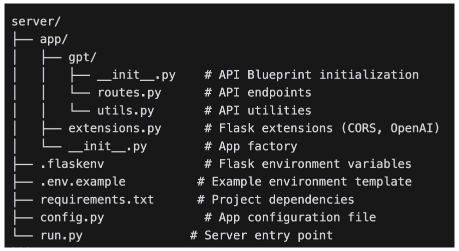
- Flask is lightweight – it doesn't enforce a strict project structure
    - i.e do what works best for you
- However, a good project structure is crucial for system scalability
    - Circular imports are a nightmare in python
- Our **tic-tac-toe server** follows the **Digital Ocean** recommended standard:



```
server/
├── app/
│   ├── gpt/
│   │   ├── __init__.py    # API Blueprint initialization
│   │   └── routes.py      # API endpoints
│   │   └── utils.py       # API utilities
│   ├── extensions.py      # Flask extensions (CORS, OpenAI)
│   └── __init__.py        # App factory
├── .flaskenv              # Flask environment variables
├── .env.example          # Example environment template
├── requirements.txt      # Project dependencies
├── config.py              # App configuration file
└── run.py               # Server entry point
```

Digital Ocean Article ⇒
https://www.digitalocean.com/community/tutorials/how-to-structure-a-large-flask-application-with-flask-blueprints-and-flask-sqlalchemy

Key Components Deep Dive



```
server/
├── app/
│   ├── gpt/
│   │   ├── __init__.py    # API Blueprint initialization
│   │   └── routes.py      # API endpoints
│   │   └── utils.py       # API utilities
│   ├── extensions.py      # Flask extensions (CORS, OpenAI)
│   └── __init__.py        # App factory
├── .flaskenv              # Flask environment variables
├── .env.example          # Example environment template
├── requirements.txt      # Project dependencies
├── config.py              # App configuration file
└── run.py               # Server entry point
```

API Endpoints

The server contains only one resource (gpt) and two endpoints

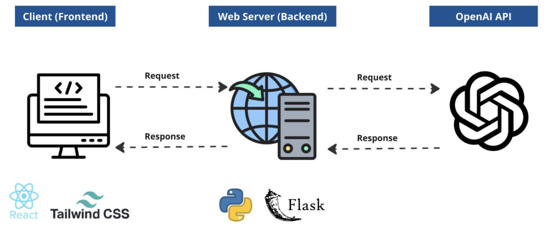
- GET: **/api/gpt/**
  - Check API status
- POST, GET: **/api/gpt/move**
  - Uses GPT engine to generate a Tic-Tac-Toe move
  - **params**
    - board - the game nbo
    - difficulty - used to determine model to use

Extensions

- CORS
- OpenAI

Key Components Deep Dive

AI Integration (How it works)



When a request is sent to **/gpt/move** endpoint:

- API receives request with current board state
- Using prompt engineering, the server creates a **prompt**
  - Explicitly defines the game's **rules**, **goal** (to win)
  - and **expected output** (JSON)
- Sends request to OpenAI's model
- Receives the AI's next move and forwards it to the client

Example Prompt used:

"You are an opponent in a 3x3 Tic-Tac-Toe game. You're playing as 'O' and your goal is to win.
 Suggest the indexes of the next move as 'row,col' in JSON format. Do not suggest cells that are already occupied."
"Given the current Tic-Tac-Toe board: {board} Make the next move for 'O':"

DEMO: DeepDive into the Tic-Tac-Toe Webserver

```
def get_move():
    """
    Use the GPT engine to generate a move for the AI player.
    :return: The move as a JSON object {row: int, col: int}
    """

    # Get the data from the request
    data = request.get_json()
    board = data.get('board')
    difficulty = data.get('difficulty')

    # Data validation: Verify that the board is provided
    if not board:
        return {"error": "Board is required."}, 400

    # Get the model based on the difficulty
    model = get_model(difficulty)

    # Send a request to the GPT engine to get AI move
    response = client.chat.completions.create(
        model=model,
        temperature=0.4,
```

```
* Debugger is active!
* Debugger PIN: [illegible]
```

Why an Additional Server was Necessary?

OpenAI provides a JavaScript package for direct client-side use, but this bad practice

- **API Key Security:**
  - API keys in frontend code is a major security risk
  - Always store them securely on the server using environment variables

- **Performance Optimization:**
  - Offload computationally expensive tasks to the backend
  - Efficiently handle multiple concurrent request on the backend

- **Scalability:**
  - Allows for future expansion with caching, authentication, and monitoring.
  - Can integrate with additional AI models or databases.

Summary

Summary

- Flask is a lightweight web framework for building APIs.
- The Tic-Tac-Toe AI server uses OpenAI model with prompt engineering for move generation.
- Additional server integration ensures security, better performance, and scalability.

Questions?