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

Lesson 2: REST APIs & OpenAI Developer APIs

Priscilla Emasoga

COM3550 Undergraduate Ambassadors Scheme
University of Sheffield

Overview

- **What is an API?**
  - Web APIs
- **REST APIs**
  - REST API Principles
- **OpenAI Developer APIs**
  - Chat Completions API

What is an API?

- An API (Application Programming Interface) is a set of protocols that enable different software applications to communicate with each other.
- Acts as an intermediary between different systems or components of a system.
- Facilitates the exchange of data between them.

- **Example:** A restaurant: The customer (user) orders food through a waiter (API), who communicates with the kitchen (server) and returns the food (response) to the customer.

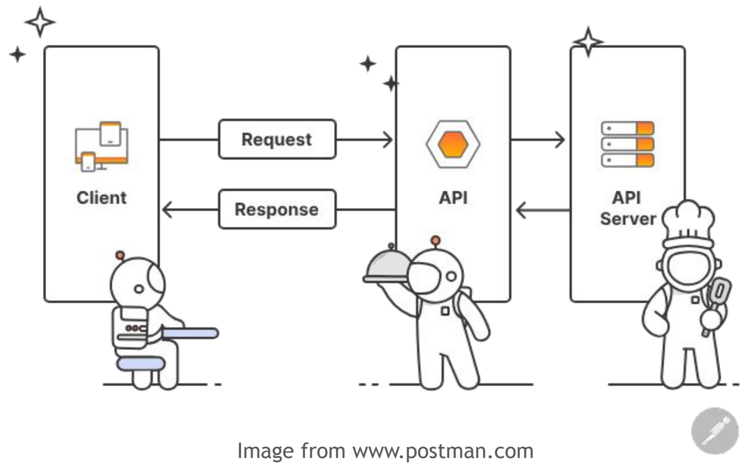


Image from www.postman.com

Web APIs

- A Web API is an API that can be accessed using the HTTP protocol.
    - Not all APIs are web APIs.
    - Some APIs are used only for communication between two applications on the same computer and do not require a web connection.

- Web APIs can be categorized based on their **architecture**
    - **REST APIs** – Use standard HTTP methods, stateless, widely used.
    - **SOAP APIs** – XML-based, strict structure, used in enterprise applications.
    - **GraphQL APIs** – Flexible queries, fetch specific data efficiently.
    - **gRPC APIs** – High-performance, uses Protocol Buffers.

- We will focus on REST APIs

REST APIs

- REST stands for **Representational State Transfer**
  - An architectural style that defines a set of standards for building Web APIs
  - REST APIs (RESTful APIs) are APIs that follow the the REST standards

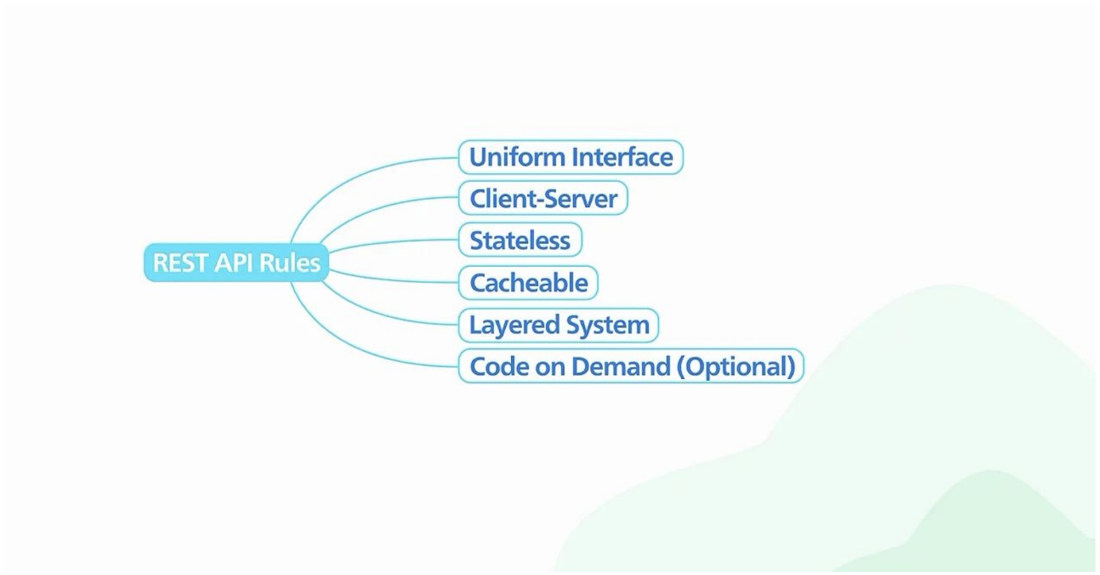


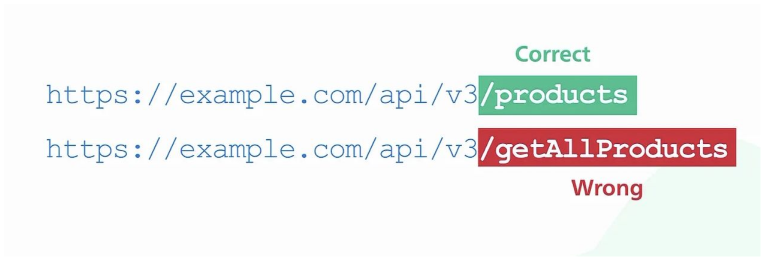
Image from www.bytebytego.com

REST API Principles

(1) Uniform Interface

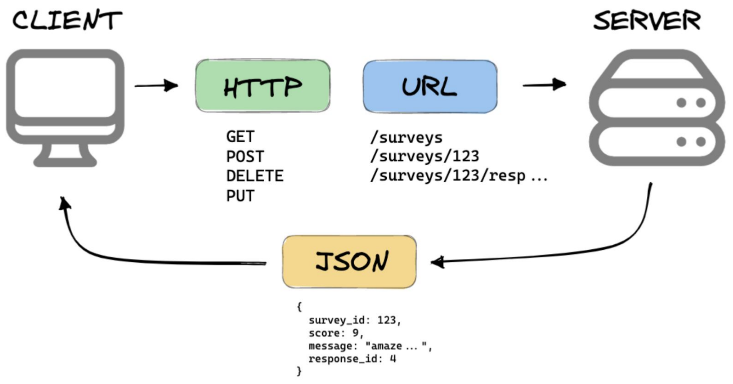
- A RESTful API organises resources into a set of unique URIs (**Uniform Resources Identifiers**)
- URIs identify different resources in a server e.g:
    - `https://example.com/api/v3/products`
    - `https://example.com/api/v3/users`

- Resources should be grouped by noun and not verb



```
https://example.com/api/v3/products
https://example.com/api/v3/getAllProducts
```

REST API Principles

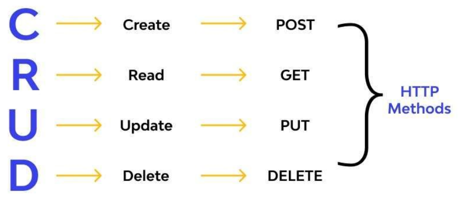


```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```

- Clients (e.g., your web or mobile app) interact with resources by sending a request to the **endpoint** of that resource using HTTP

POST	Create a new resource (e.g., adding a new user).
GET	Read data (e.g., fetching product details).
PUT	Update an existing resource (e.g., editing user details).
DELETE	Delete a resource (e.g., deleting a user account).

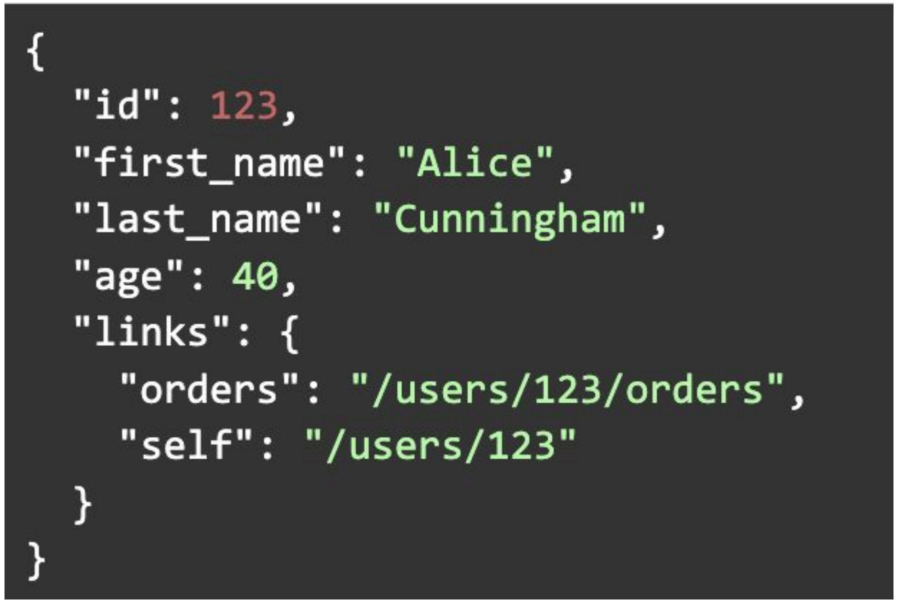
- Server returns a response (HTML/JSON/XML) and HTTP status codes
- Status codes inform clients about the result of their requests
    - **200 OK** = request succeeded
    - **404 Not found** = requested resource doesn't exist
    - **500 Internal Server Error** = a server-side error occurred



REST API Principles

Example Response:

- A GET request to `https://example.com/users/123` might return:



```json
{
  "id": 123,
  "first_name": "Alice",
  "last_name": "Cunningham",
  "age": 40,
  "links": {
    "orders": "/users/123/orders",
    "self": "/users/123"
  }
}
```

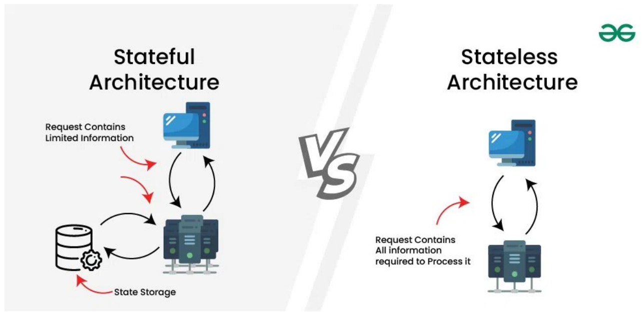
REST API Principles

(2) Client-Server Architecture

- The client and server (the backend API) are separated
  - **Client:** Responsible for the user interface and user experience
  - **Server:** Handles the data storage, processing and business logic

(3) Stateless

- Each request is treated independently i.e the server doesn't maintain any session information about the previous request
- Hence, requests must contain ALL the information the server needs to process it


Stateful Architecture
Request Contains Limited Information
State Storage
VS
Stateless Architecture
Request Contains All information required to Process it

REST API Principles

(4) Cacheability

- Responses define themselves as cacheable or not to improve performance
- If a response is cacheable, clients can reuse them for subsequent requests
    - **Example:** A GET request to `/products` might return a list of products along with cache headers (e.g., Cache-Control: max-age=3600) to instruct clients to cache the result for an hour.

(5) Layered System

- A client cannot tell whether it is connected directly to the end server, or to an intermediary along the way

(6) Code on demand (optional)

- Servers may send executable code to clients, allowing the client to run that code locally
- Most APIs return only data (JSON/XML), so not always needed

OpenAI Developer APIs

OpenAI Developer APIs

- OpenAI provides APIs that enable developers to integrate AI-powered models like GPT, DALL·E, and Whisper into applications.

- Examples:
  - **Chat Completions API** - text generation / completion
  - **Images API** - image generation, edits and variations
  - **Audio API** - text-to-speech (TTS), speech-to-text (STT)
  - **Moderations API** - check for harmful contents in text/images
  - **Embeddings API** - generate vector representation of text

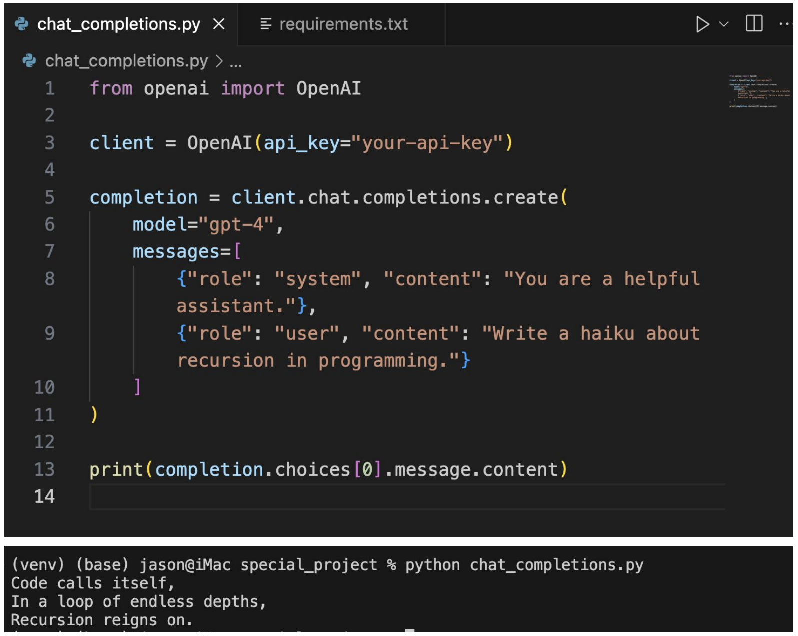
- For this project, we will be using Chat API

Read the docs ⇒ https://platform.openai.com/docs

Chat Completions API

Unlike Web APIs, OpenAI APIs function as library-based APIs, which can be integrated into programs by importing the appropriate library for a given programming language.

Python Code Example:



```python
from openai import OpenAI

client = OpenAI(api_key="your-api-key")

completion = client.chat.completions.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a helpful
        assistant."},
        {"role": "user", "content": "Write a haiku about
        recursion in programming."}
    ]
)

print(completion.choices[0].message.content)
```

```
(venv) (base) jason@iMac special_project % python chat_completions.py
Code calls itself,
In a loop of endless depths,
Recursion reigns on.
```

Chat Completions API

Let's break down the most important parameters you can use when making API calls

- **model**
    - Specifies which GPT model to use (e.g "gpt-4", "gpt-3.5-turbo")
    - Each model has different capabilities and cost tokens
- **messages**
    - An array of message objects that forms basis of the conversation
    - Each message needs a 'role' and 'content'
        - Roles can be:
            - system (sets behavior)
            - user (your input)
            - assistant (AI's responses)
- **temperature**
    - Controls randomness in responses
    - Ranges between 0 to 2
        - 0 = very focused, deterministic
        - 1 = more creative, varied

Chat Completions API

Let's break down the most important parameters you can use when making API calls

- **max_tokens**
  - Limits the length of the response
  - Higher values = longer responses = costing more tokens

🎯 **Best practices**

- Start with default parameters and adjust as needed
- Always include a clear **system** message
- Monitor your token usage!
- Temperature control
  - Lower temperature (0.1-0.3) for factual/coding tasks
  - Higher temperature (0.7-1.0) for creative tasks

For complete list of parameters, see https://platform.openai.com/docs/api-reference/chat

Summary

Summary

- APIs allow different software applications to communicate.
- REST APIs are APIs that follow the REST principles
    - (uniform interface, client-server, stateless, cacheable, layered system)
- OpenAI APIs enable seamless integration of AI models into applications.
- Chat Completions API can be used with different models to generate human-like text and code.
    - Each model cost different amount of tokens!

Questions?