1. What is an API?

API stands for Application Programming Interface. Think of it as a messenger or intermediary that allows different software applications to communicate and exchange data with each other. It defines a set of rules and specifications that govern how these interactions should happen.

Imagine you're at a restaurant. You (one application) want to order food (request data/actions) from the kitchen (another application). You don't go directly into the kitchen; instead, you interact with the waiter (the API). The waiter takes your order (request), conveys it to the kitchen, and brings you the food (response). The waiter follows a specific protocol (the API's rules) to ensure everything works smoothly.

APIs are essential for modern software development. They enable:

- **Integration:** Connecting different systems together.
- **Reusability:** Leveraging existing functionality without rewriting code.
- **Abstraction:** Hiding complex implementation details.
- **Innovation:** Building new applications by combining existing services.

2. What is a REST API?

REST stands for Representational State Transfer. It's an architectural style for building web APIs that uses standard HTTP methods (like GET, POST, PUT, DELETE) to interact with resources (data). REST APIs are designed to be stateless, meaning each request from a client to a server must contain all the information needed to understand and process the request. The server doesn't store any client context between requests.

Key characteristics of REST APIs:

- **Client-Server:** The client (e.g., a web browser or mobile app) requests resources from the server.
- **Stateless:** Each request is independent and contains all the necessary information.
- **Cacheable:** Responses can be cached to improve performance.
- **Uniform Interface:** Consistent use of HTTP methods and resource identifiers (URLs).
- **Layered System:** Intermediate servers (proxies, gateways) can be used.
- **Code-On-Demand (Optional):** Server can send executable code to the client.

REST APIs are very popular because they are simple, scalable, and easy to understand. They are commonly used for web services and mobile apps.

Difference between JSON and XML

JSON (JavaScript Object Notation) and XML (Extensible Markup Language) are both data-interchange formats, meaning they are used to structure and transmit data between systems. However, they have some key differences:

In short:

- JSON is lightweight, human-readable, and easy to parse, making it ideal for web APIs and data exchange.
- XML is more verbose and complex but provides more flexibility for representing complex data structures and metadata.

The choice between JSON and XML depends on the specific needs of the application. JSON has become increasingly popular in recent years due to its simplicity and efficiency.

Sources and related content

Feature	JSON	XML
Syntax	Uses key-value pairs, curly braces {} for objects, square brackets [] for arrays.	Uses tags, angle brackets < >, and attributes.
Data Types	Limited set of data types (string, number, boolean, array, object, null).	Supports more complex data types through attributes and elements.
Readability	Generally considered more readable due to its simpler syntax.	Can be more verbose and less readable, especially for complex data structures.
Parsing	Easier and faster to parse in most programming languages, especially JavaScript.	Parsing can be more complex and resource-intensive.
Size	Typically smaller in size compared to equivalent XML data.	Can be larger due to the verbose markup.
Use Cases	Commonly used for web APIs, mobile apps, and data exchange where simplicity and speed are important.	Used in enterprise applications, configuration files, and situations where complex data structures and metadata are required.

REST key components

- HTTP Methods
- Get – receive data
- Post – create new data
- Put – update
- Patch –
- Dalete-

http status code

- 200 ok – request successful

- 201 create - new resource created

- 401 unauthorized – authentication required

- 400 bad request – invalid input

- 404 not found – resource available

- 500 internal server error - api failure

Restful URL structure

- Base url
- Resource path
- Query parameters

Challenges and limitations

- Rate limiting
- Versioning
- Error handeling

Assignment 3 - Take public apis take json format data in code itself and and do the testing using pytest use oops functions .

Libraries – request , json , pytest, assertions

```
import json
import requests
import pytest
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)

# API Endpoint for Open-Meteo
URL = "https://api.open-meteo.com/v1/forecast?latitude=37.7749&longitude=-
122.4194&current_weather=true"

# Test API response and create JSON file
def test_weather_api_response():
    try:
        response = requests.get(URL)
        response.raise_for_status()  # Raise an error for bad responses
    except requests.exceptions.RequestException as e:
```

```
        logging.error("Request failed: %s", e)
        pytest.fail("API request failed")

    # Check if the response is successful
    assert response.status_code == 200, "API response was not successful"

    data = response.json()

    # Save the data to a JSON file
    with open("weather_data.json", "w") as json_file:
        json.dump(data, json_file, indent=4)

    # Check if the expected fields are present in the response
    assert "current_weather" in data, "Current weather data not found in
response"

    # Debugging output
    logging.info("Weather Data: %s", data)


if __name__ == "__main__":
    pytest.main()
```

PS C:\Users\91902\Desktop\apexaIQ> pytest "assignment 3/test_new_api.py" –v

JSON FILE – WEATHER_DATA.JSON

This Python code performs the following actions:

1. **Imports necessary libraries:**
   - **json:** For working with JSON data.
   - **requests:** For making HTTP requests to the API.
   - **pytest**: For writing and running tests.
   - **logging:** For logging information and errors.
2. **Configures logging:**
   - **logging.basicConfig(level=logging.INFO**) sets up basic logging to display information-level messages and above (warnings, errors, critical).
3. **Defines the API endpoint:**

- **URL** stores the URL of the Open-Meteo weather API endpoint. It's set to retrieve weather data for San Francisco (latitude 37.7749, longitude -122.4194) and requests the current weather.

4. **Defines a test function test_weather_api_response():**

- This function uses **pytest** to create a test case.
- It makes a GET request to the **URL** using **requests.get().**
- It uses a **try...except** block to handle potential errors during the API request. If the request fails (e.g., network issue), it logs the error and uses **pytest.fail()** to mark the test as failed.
- **response.raise_for_status()** checks the HTTP status code. If it's not in the 200-299 range (success), it raises an exception, causing the test to fail.
- a**ssert response.status_code == 200** is another way to assert that the response code is 200 (OK).
- It parses the JSON response using **response.json()** and stores it in the **data** variable.
- It saves the **data** (the weather information) to a file named **weather_data.json** using **json.dump()** with an indent for readability.
- It asserts that the **data** dictionary contains the key "current_weather", ensuring that the expected data is present.
- **logging.info("Weather Data: %s", data)** logs the received weather data. This is helpful for debugging.

5. **Runs the tests:**

- **if __name__ == "__main__":** ensures that **pytest.main()** is only called when the script is run directly (not imported as a module).
- **pytest.main(**) executes the test function defined above. Pytest will automatically discover and run functions that start with **test_.**

  In summary: The code retrieves current weather data from the Open-Meteo API for San Francisco, checks if the API response is successful, saves the data to a JSON file, and verifies that the response contains the expected "current_weather" data. It uses **pytest** for testing and **logging** for informative output. When you run this script, it will execute the test and create the **weather_data.json** file in the same directory.

command –