API TESTING INTERVIEW QUESTION

What is API Testing?

API Testing checks if APIs (Application Programming Interfaces) work correctly, reliably, and securely. It ensures APIs meet their purpose by testing their performance, security, and functionality.

Example:
A weather app uses an API to fetch live weather data from a server. API testing ensures the app gets the correct data (e.g., temperature, forecast) without errors.

Types of API

1. **Web API**
   Used for communication over the internet (e.g., RESTful API, SOAP, GraphQL).
   Example: A payment gateway like PayPal's API processes payments for e-commerce websites.

2. **Operating System API**
   Helps apps talk to operating systems.
   Example: Windows API lets an app open files on your computer.

3. **Library API**
   Provides pre-made functions for software development.
   Example: A library API like TensorFlow helps developers build machine-learning models.

4. **Hardware API**
   Helps software interact with devices like printers or cameras.

Example: A Camera API in a photo editing app to capture pictures directly.

Tools for API Testing

1. **Postman**
   Test APIs by sending requests like GET or POST and viewing responses.
   Example: Test a login API by sending a username and password and checking if you get a success token.

2. **Swagger**
   Generate and test APIs directly from detailed API documentation.
   Example: Use Swagger to check how to send a request to fetch user details.

3. **SoapUI**
   Specifically for testing SOAP APIs.
   Example: Test a SOAP API for retrieving bank account balance using XML.

1. **HTTP Status Codes**
   Codes that show request outcomes.

   - **200 (Success):** Request worked.
   - **404 (Not Found):** Resource doesn't exist.
   - **500 (Server Error):** Problem on the server side.
     Example: A "500" error is returned if the server crashes when you try to book a ticket.

2. **JSON (JavaScript Object Notation)**
   A lightweight format to send and receive data.
   Example: {"name": "John", "age": 30}

3. **Authentication**

   Confirms user identity to secure APIs.

   - **Token-based:** Temporary keys for access.
   - **OAuth:** Used by apps like Google to let others access user data securely.

     Example: A login API uses a token to let a user stay logged in without entering the password again.

4. **Error Handling**

   - **HTTP Errors:** Server-side problems (e.g., 500 = Server Error).
   - **Validation Errors:** Input issues (e.g., missing fields).

     Example: If an email field is empty during registration, an error like "Email is required" will appear.

5. **Query Parameters**

   Adds extra details to requests.

   Example: api.com/search?query=book passes the search keyword "book" to the API.

1. **Mock Servers**

   Fake servers to test APIs when the real backend isn't ready.

   Example: Use a mock server to test a shopping cart API before the server is live.

2. **Dynamic Data**

   Use tools to handle timestamps or random data during testing.

   Example: Send a unique order ID each time you test an order API.

3. **Load Testing**

   Simulates heavy usage to test API performance.

   Example: Test an API with 1,000 simultaneous requests to see if it can handle the traffic.

How to Keep API Documentation Updated

Use tools like Swagger or OpenAPI to auto-generate and maintain API guides.

Functional API Testing Using Postman

1. Functional API testing ensures the reliability and functionality of an API. We check whether the API behaves as expected and delivers the correct output.

2. When we start testing, the process begins by receiving the API and its accompanying documentation. Swagger is commonly used for documenting APIs.

3. Swagger provides details such as the base URL, paths, query parameters, request methods (e.g., GET, POST), expected output, payloads, and pre-request scripts.

4. For example, if a GET request is included in the documentation, we use the information provided, make the request, and properly document the outcome.

5. **What is payload?**

   Payload refers to the extra required details sent with a request, often used in PATCH or PUT requests. It usually contains metadata (data about data). For instance, a payload may specify additional instructions or information to be processed.

6. **Pre-request scripts**

   Pre-request scripts are present on the request side in Postman. For example:

   - We run a request.
   - Check the response.
   - Apply pre-request scripts if required.

7. To validate the response of an API, we start by checking if the status code is correct. Common status codes include:

   - 200: Query ran successfully.
   - 404: Resource not found.
   - 500: Internal server error.

8. If there are 5 APIs in a collection, and the first two pass while the third fails, we debug by:

   - Using Postman's console to troubleshoot.
   - Changing the sequence of APIs to isolate the issue.
   - Checking the network tab for detailed logs.
   - Examining headers, parameters, and payloads.
   - Verifying if the API works in Swagger.
   - Discussing the issue with the development team if needed.

9. **Postman Variable Scopes**

   Variables in Postman work within specific scopes:

   - **Global**: Accessible throughout all collections.
   - **Local**: Limited to a single request or script.
   - **Collection**: Shared within a collection.

- **Environment**: Specific to the environment set.

Assertions in Postman

Assertions are used to validate the response. For example:

```
pm.expect(pm.response.code).to.equal(200);
```

10. **Path and Query Parameters in Postman**

    - **Path Parameters**: These are part of the base URL, e.g., `amazon.com/products/{id}`.
    - **Query Parameters**: These are appended to the URL, e.g., `amazon.com/products?category=electronics`.

11. **Scenario: API Taking Too Long**

    If an API takes too much time to respond, you can:

    - Introduce a delay.
    - Adjust the timeout (request timeout).
    - Use an if-condition to handle the delay.
    - Write custom code in the test section to manage such scenarios.

For example, in Postman, you can set a timeout as follows:

```
pm.setTimeout(30000); // Timeout in milliseconds
```