REST ASSURED

What is API Testing?

API testing checks if APIs (Application Programming Interfaces) work correctly. It focuses on:
1. Functionality: Does the API do what it's supposed to do?
2. Performance: Is it fast and efficient?
3. Security: Is the data safe?

APIs are like messengers between applications. Testing ensures these messages are sent, received, and processed correctly.

Example (with WhatsApp):
- When you send a message on WhatsApp, the app uses an API to communicate with its server.
1. Your phone sends the message to the server through the API.
2. The server processes it and forwards it to the recipient.
- API Testing checks:
  - Is the message sent correctly?
  - Does the recipient get the exact message?
  - How fast does the message reach?

In API testing, tools like Postman or Rest Assured are used to simulate these processes and verify their accuracy without needing a UI (like the WhatsApp app).

Difference between REST and SOAP APIs

1. **REST (Representational State Transfer)**
   - Protocol: Uses HTTP/HTTPS.
   - Data format: Primarily JSON (also XML, HTML, etc.).
   - Speed: Faster and lightweight.
   - Usage: Mostly used in modern web services and mobile apps.
   - Example: WhatsApp uses REST for communication between the app and the server.

2. **SOAP (Simple Object Access Protocol)**
   - Protocol: Uses HTTP, SMTP, TCP, etc.
   - Data format: XML only.
   - Speed: Slower and heavier because of XML formatting.
   - Usage: Typically used in enterprise-level applications (banking, payment systems).
   - Example: SOAP is used in older services like banking transactions for higher security and reliability.

Key Differences:
- Data format: REST supports JSON, making it faster, while SOAP only supports XML.
- Complexity: REST is simpler and more flexible; SOAP is more complex and rigid.
- Performance: REST is faster and lightweight compared to SOAP's heavy XML-based messaging.

Overview of Rest Assured
- Rest Assured is a Java-based library used for testing REST APIs.
- It allows easy testing of API requests and responses.
- It is commonly used in automating API tests in Java-based environments.
- With Rest Assured, you can send requests (GET, POST, PUT, DELETE) to APIs and verify their responses.

Advantages of using Rest Assured for API testing
- Easy to use: Simplifies API testing by providing simple syntax and methods.
- Supports all HTTP methods: Can handle GET, POST, PUT, DELETE, etc., for testing APIs.
- Good integration: Easily integrates with Java testing frameworks like JUnit and TestNG.
- Built-in support for JSON and XML: Supports both formats to parse and validate API responses.
- Automation-friendly: Great for automating API tests in CI/CD pipelines.

Rest Assured makes it easy to automate and validate APIs without requiring complex setup or external tools.

What is REST?
- REST (Representational State Transfer) is an architectural style for building web services. It relies on stateless communication and uses standard HTTP methods for interactions. RESTful APIs allow clients to communicate with servers and access resources in a simple and scalable way.

REST principles:
- Statelessness: Each request from a client to a server must contain all the information the server needs to understand and respond to the request. The server does not store any session information.
- Uniform Interface: The API should have a consistent and standardized way of interacting, making it easier for developers to understand and use.
- Client-Server architecture: The client and server are independent of each other. The client sends requests, and the server processes them and sends back responses.

Example:
In WhatsApp, when you send a message (client), the server processes it and sends it to the recipient. The client doesn't need to know how the server works, it just interacts with the API.

HTTP Methods in REST:
- GET: Retrieves data from the server (eg: get the list of all messages).
- POST: Sends data to the server (eg: send a new message).
- PUT: Updates existing data on the server (eg: update a message).
- DELETE: Removes data from the server (eg: delete a message).
- PATCH: Partially updates existing data (eg: update just the text of a message).

HTTP Status Codes:
- 200 OK: The request was successful and the server responded with the requested data.
- 400 Bad Request: The request is invalid or malformed (eg: missing required fields).
- 404 Not Found: The requested resource could not be found
(eg: trying to retrieve a message that doesn't exist).
- 500 Internal Server Error: The server encountered an unexpected error and could not process the request.

Example with WhatsApp:
- When you try to fetch your messages (GET), if everything is fine, you get a 200 OK response.
- If you try to send a message with missing data (POST), the server might return a 400 Bad Request error.
- If you try to open a chat that doesn't exist (GET), the server will return 404 Not Found.
- If the server crashes while sending a message (POST), you might get a 500 Internal Server Error.

Making Simple GET Requests

- **Sending a GET request and receiving a response**:
A GET request is used to fetch data from a server. In the case of WhatsApp, a GET request might be used to fetch the list of messages from a server.

Example in code (using Java and Rest Assured):

```java
import io.restassured.RestAssured;
  import io.restassured.response.Response;

  public class GetRequestExample {
      public static void main(String[] args) {
          // Send a GET request to the WhatsApp API to fetch messages
          Response response = RestAssured.get("https://api.whatsapp
              .com/messages");

          // Print the response status code, body, and headers
          System.out.println("Status Code: " + response.getStatusCode
              ());
          System.out.println("Response Body: " + response.getBody
              ().asString());
          System.out.println("Response Headers: " + response
              .getHeaders());
      }
  }
```

- **Extracting response data (status code, body, headers)**:
 After sending a GET request, we can extract useful information such as the status code (e.g., 200 OK), the body (which contains the data sent by the server), and the headers (which contain metadata about the response).

Example Explanation:
1. Status Code: Shows if the request was successful.
2. Response Body: Contains the data returned from the server (e.g., list of messages).
3. Response Headers: Provides information about the response, like content type or server details.

Example with WhatsApp:
- When you send a GET request to fetch your messages, the server might respond with:
  - Status Code: 200 OK (if everything is fine).
  - Response Body: A JSON object containing your list of messages.
  - Response Headers: Information like content type (`application/json`) or server details.

Understanding Response Structure

- **JSON and XML response formats**:
  APIs commonly return data in JSON or XML formats.
  - JSON (JavaScript Object Notation): A lightweight, easy-to-read data format often used in web services.
  - XML (Extensible Markup Language): A markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

Example of a JSON response (from WhatsApp API):

```
{
  "messages": [
    {
      "sender": "John",
      "message": "Hello!",
      "timestamp": "2024-12-10T10:00:00"
    },
    {
      "sender": "Alice",
      "message": "Hi, how are you?",
      "timestamp": "2024-12-10T10:05:00"
    }
  ]
}
```

Example of an XML response:

```
<messages>
  <message>
    <sender>John</sender>
    <message>Hello!</message>
    <timestamp>2024-12-10T10:00:00</timestamp>
  </message>
  <message>
    <sender>Alice</sender>
    <message>Hi, how are you?</message>
    <timestamp>2024-12-10T10:05:00</timestamp>
  </message>
</messages>
```

- **Extracting data from JSON/XML responses using Rest Assured**:
 Rest Assured provides easy ways to extract data from both JSON and XML responses. You can use methods like **".getBody().jsonPath()"** for JSON responses and **".getBody().xmlPath()"** for XML responses.

Example in code (using Rest Assured for extracting data from a JSON response):

```java
import io.restassured.RestAssured;
import io.restassured.response.Response;

public class ExtractJsonExample {
    public static void main(String[] args) {
        // Send a GET request and get the response
        Response response = RestAssured.get("https://api.whatsapp.com
            /messages");

        // Extract a message sender's name from the JSON response
        String sender = response.getBody().jsonPath().getString
            ("messages[0].sender");
        System.out.println("Sender of the first message: " + sender);

        // Extract the first message content
        String message = response.getBody().jsonPath().getString
            ("messages[0].message");
        System.out.println("First message: " + message);
    }
}
```

Example for XML response extraction:

```java
import io.restassured.RestAssured;
import io.restassured.response.Response;

public class ExtractXmlExample {
    public static void main(String[] args) {
        // Send a GET request and get the response
        Response response = RestAssured.get("https://api.whatsapp.com
            /messages");

        // Extract a message sender's name from the XML response
        String sender = response.getBody().xmlPath().getString
            ("messages.message[0].sender");
        System.out.println("Sender of the first message: " + sender);

        // Extract the first message content
        String message = response.getBody().xmlPath().getString
            ("messages.message[0].message");
        System.out.println("First message: " + message);
    }
}
```

Explanation:

1. In the JSON example, we use `jsonPath()` to access elements in the JSON response (like the sender and message).
2. In the XML example, we use `xmlPath()` to access elements in the XML response.

This approach helps you extract specific data easily from the response, which is useful when validating or processing API responses in your tests.