CHAPTER 3:

SOAP and REST

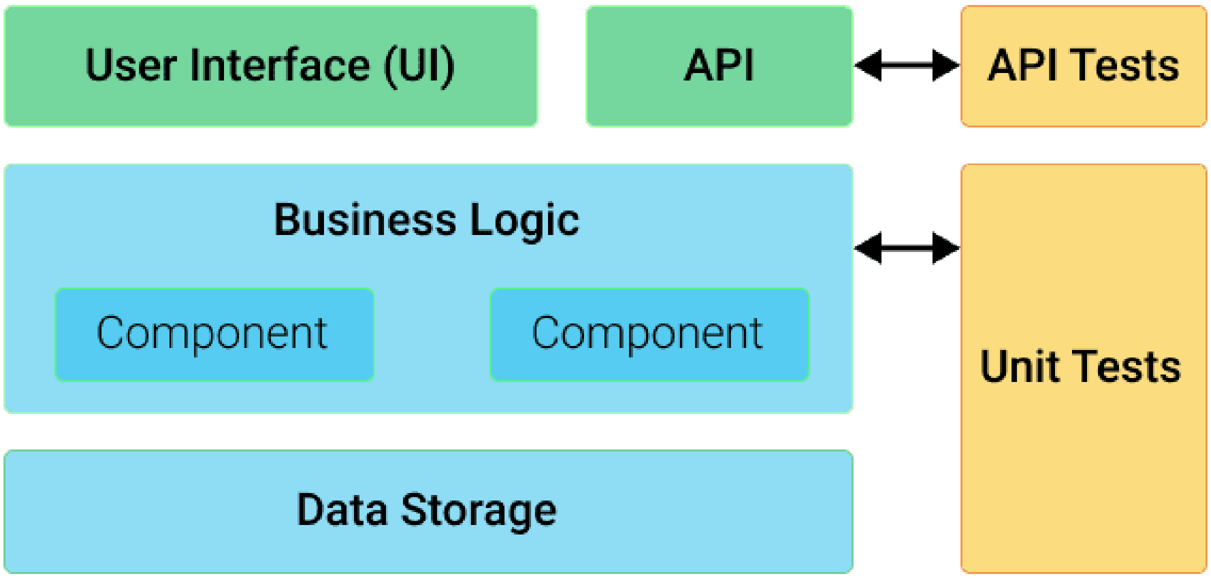
Difference between REST API and SOAP API

There is no direct comparison between SOAP and REST APIs. But there are some points to be listed below which makes you choose better between these two web services. Here are:

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol and REST stands for **RE**presentational **S**tate **T**ransfer.
- Since SOAP is a protocol, it follows a strict standard to allow communication between the client and the server whereas REST is an architectural style that doesn't follow any strict standard but follows six constraints defined by Roy Fielding in 2000. Those constraints are – **Uniform Interface, Client-Server, Stateless, Cacheable, Layered System, Code on Demand**.
- SOAP uses only XML for exchanging information in its message format whereas REST is not restricted to XML and its the choice of implementer which Media-Type to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol but SOAP cannot use REST.
- On behalf of services interfaces to business logic, SOAP uses @WebService whereas REST instead of using interfaces uses URI like @Path.
- SOAP is difficult to implement and it requires more bandwidth whereas REST is easy to implement and requires less bandwidth such as smartphones.
- Benefits of SOAP over REST as SOAP has ACID compliance transaction. Some of the applications require transaction ability which is accepted by SOAP whereas REST lacks in it.
- On the basis of Security, SOAP has SSL(**S**ecure **S**ocket **L**ayer) and WS-security whereas REST has SSL and HTTPS. **In the case of Bank Account Password, Card Number, etc. SOAP is preferred over REST.** The **security issue** is all about your application requirement, you have to build security on your own. It's about what type of protocol you use.

An API (application programming interface) is essentially the "middle man" of the layers and systems within an application or a software.

API testing is performed at the message layer without GUI. It is a part of integration testing that determines whether the **APIs meet the testers' expectations** of functionality, reliability, performance, and security.



There are two broad classes of web service for Web API: SOAP and REST.

SOAP (Simple Object Access Protocol) is a standard protocol defined by the W3C standards for sending and receiving web service requests and responses.

REST (REpresentational State Transfer) is the web standards-based architecture that uses HTTP. Unlike SOAP-based Web services, there is no official standard for RESTful Web APIs.

Here are 5 basic tips that you need to know for API testing:

1. Understand API requirements

Before testing your APIs, you need to answer these questions to thoroughly understand the API's requirements:

- What is the API's purpose?

Knowing the purpose of the API will set a firm foundation for you to well prepare your test data for input and output. This step also helps you define the verification approach. For example, for some APIs, you will verify the responses against the database; and for some others, it is better to verify the responses against other APIs.

- What is the workflow of the application; and where is the API in that flow?

Generally, APIs of an application are used to manipulate its resources in reading (GET), creating (POST), updating (PUT) and deleting (**DELETE**). Knowing the purpose of the API will set a firm foundation for you to well prepare your API testing data for input and output.

In addition, this step also helps you define the verification approach. For example, for some APIs, you will verify the responses against the database; and for some others, it is better to verify the responses against other APIs.

For example, the output of the "Create user" API will be the input of the "Get user" API for verification. The output of the "Get user" API can be used as the input of the "Update user" API, and so on.

2. Specify the API output status

The most common API output you need to verify in API testing is the **response status code**.

Verifying if the response code equals to **200** or not to decide whether an API testing is passed or failed is familiar to new API testers. This is not a wrong verification. However, it does not reflect all test scenarios of the API.

All **API response status codes** are separated into five classes (or categories) in a global standard. The first digit of the status code defines the class of response. The last two digits do not have any class or categorization role.

There are five values for the first digit:

- 1xx (Informational): The request is received and continues to be processed
- 2xx (**Successful**): The request is successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken to complete the request
- 4xx (**Client Error**): The request contains the wrong syntax or cannot be fulfilled
- 5xx (**Server Error**): The server fails to fulfill an apparently valid request. Mostly handled during Development Unit Testing.

However, the actual response status code of an API is specified by the development team that built the API. So as a tester, you need to verify whether:

- The code follows global standard classes
- The code is specified in the requirement.

3. Focus on small functional APIs

In a testing project, there are always some APIs that are simple with only one or two inputs such as login API, get token API, health check API, etc. However, these APIs are necessary and are considered as the "gate" to enter further APIs. Focusing on these APIs before the others will ensure that the API servers, environment, and authentication work properly.

You should also **avoid testing more than one API in a test cas**e. It is painful if errors occur because you will have to debug the data flow generated by API in a sequence. Keep your **testing as simple as possible**. There are some cases in which you need to call a series of API to achieve an end-to-end testing flow. However, these tasks should come after all APIs have been individually tested.

4. Organize API endpoints

A testing project may have a few or even hundreds of APIs for testing. We highly suggest that you organize them into categories for better test management. It takes one extra step but will significantly help you create test scenarios with high coverage and integration.

5. Create positive and negative tests

API testing requires both positive and negative tests to ensure that the API is working correctly. Since API testing is considered a type of black-box testing, both types of **testing are driven by input and output data**. There are a few suggestions for test scenario generation:

- Positive test
  - Verify that the API receives input and returns the expected output as specified in the requirement.

  - Verify that the response status code is returned as specified in the requirement, whether it returns a 2xx or error code.
  - Specify input with minimum required fields and with maximum fields.
- Negative test
  - Verify that the API returns an appropriate response when the expected output does not exist.
  - Perform input validation test.
  - Verify the API's behaviors with different levels of authorization.