**Application programming interface (API):-**

**What Is API Testing?**

At its most basic level, API testing is intended to reveal bugs: Inconsistencies or deviations from the expected behavior. Continuous testing is also very important to make sure it continues to work when the public has access to it. The risk of putting a bad, and potentially insecure, product on the market is greater than the cost to test it.

API testing is one of the most challenging parts of the chain of software and QA testing because it works to assure that our digital lives run in an increasingly seamless and efficient manner.

While developers tend to test only the functionalities they are working on, testers are in charge of testing both individual functionalities and a series or chain of functionalities, discovering how they work together from end to end.

Here is some of the most common reasons people test their APIs:

1. Make sure it does what it’s supposed to do
2. Make sure it can handle the load
3. Find all the way users can mess things up
4. Make sure your APIs work across devices, browsers, and operating systems
5. It can be costly not to

**API Testing, what, why**

1. Interaction with two system e.g. database connection , library , image processing
2. Two application talk to each other & get the response in form of JSN or SOAP XML format
3. Set of routine , protocol in order to building the any software applications
4. Also known as integration testing
5. Test for Core Functionality
6. Time Effective
7. Language-Independent
8. Easy Integration with GUI

**What You Need To Start API Testing**

The first part of API testing involves setting up a testing environment, with the required set of parameters around the API. This involves configuring the database and server for the application’s requirements.

Once you’ve set up your API testing environment, make an API call right away to make sure nothing is broken before you go forward to start your more thorough testing.

You can start combining your application data with your API tests to ensure that the [API performs as expected](https://www.soapui.org/testing-dojo/best-practices/structuring-your-tests.html)against possible known input configurations.

1. Who is your target audience? Who is your API consumer?
2. What environment/s should the API typically be used?
3. What aspects are you testing?
4. What problems are we testing for?
5. What are your priorities to test?
6. What is supposed to happen in normal circumstances?
7. What could potentially happen in abnormal circumstances?
8. What is defined as a Pass or a Fail? What data is the desired output? What is the chain of events?
9. What other APIs could this API interact with?
10. Who on your team is in charge of testing what?

**What Types of API Testing Can I Do?**

1. [Functionality testing](https://www.soapui.org/professional/soapui-pro.html) — the API works and does exactly what it’s supposed to do.
2. [Reliability testing](https://www.soapui.org/testing-dojo/best-practices/scenario-based-testing.html) — the API can be consistently connected to and lead to consistent results
3. [Load testing](https://www.soapui.org/professional/loadui-pro.html) — the API can handle a large amount of calls
4. Creativity testing — the API can handle being used in different ways.
5. Security testing — the API has defined security requirements including authentication, permissions and access controls. See some API security tips for protecting vital data
6. Proficiency testing — the API increases what developers are able to do.
7. API documentation testing — also called discovery testing, the API documentation easily guides the user.
8. [Negative Testing](https://www.soapui.org/testing-dojo/best-practices/negative-testing.html) — checking for every kind of wrong input the user can possibly supply

You should use automated testing for the following:

* [API functional testing](https://www.soapui.org/docs/functional-testing/getting-started.html)
* Analyzing your functional test coverage to know what you're missing
* Performance testing
* [Data driven testing](https://www.soapui.org/docs/data-driven-tests/functional-tests.html)
* [Load testing](https://www.soapui.org/load-testing/concept.html)
* Error testing
* Testing in multiple languages
* Regression testing

**API Testing Best Practices**

Before you head off on your own and get started with API testing of your very own, here are the top 10 tips we want you to remember when API testing!

1. Test for the typical or expected results first
2. Add stress to the system through a series of API load tests
3. Test for failure. Make sure you understand how your API will fail. Just make sure the API fails consistently and gracefully
4. Group test cases by test category
5. Prioritize API function calls so that it will be easy for testers to test quickly and easily
6. Limit the tests from as many variables as possible by keeping it as isolated as possible
7. See how it handles unforeseen problems and loads by throwing as much as you can at it
8. Perform well-planned call sequencing
9. For complete test coverage, create test cases for all possible API input combinations
10. Automate wherever you can

**How to do API Testing?**

1. **SOAP UI - free**
2. SOAP UI pro - paid
3. Postman - free
4. Postman pro - Paid
5. Rest assured - free

### Katalon Studio

### Swagger.io - paid

### JMeter - free

### Karate DSL – free

1. Airborne - Free
2. Pyresttest- paid
3. Apigee - Paid

**Elements of Web services and explained in detail**

1. SOAP (Simple Object Access Protocol) :-
   * Give the SOAP endpoint as the URL. If you are using a WSDL, then give the path to the WSDL as the URL.
   * Set the request method to POST.
   * Open the raw editor, and set the body type as “text/xml”.
   * In the request body, define the SOAP Envelope, Header and Body tags as required.
2. **WSDL (Web Services Description Language)** – WSDL Usage. WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special **datatypes** used are embedded in the WSDL file in the form of XML Schema
3. **REST (Representational State Transfer)** – Representational State Transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating web services. ... In a Restful web service, **requests** made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format.

**SOAP v/s REST**

**SOAP:-**

1. SOAP stands for Simple Object Access Protocol
2. SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.
3. SOAP is heavier than REST
4. SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.

<?xml version="1.0"?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV

="http://www.w3.org/2001/12/soap-envelope"

SOAP-ENV:encodingStyle

=" http://www.w3.org/2001/12/soap-encoding">

<soap:Body>

<Demo.guru99WebService

xmlns="http://tempuri.org/">

<EmployeeID>int</EmployeeID>

</Demo.guru99WebService>

</soap:Body>

</SOAP-ENV:Envelope>

**REST:-**

1. REST stands for Representational State Transfer
2. REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.
3. REST is light weight
4. REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.

s{"city":"Mumbai","state":"Maharastra"}