7.3 API Testing

The meaning of API abbreviation is a well-known, as application programming interface. Actually, it is an interface or communication protocol between two points, usually client and server, which are responding to certain request in predefined format — usually called “contract”. However, API may refer to various things, as it can be software library, web service, remote API, etc. Earlier, it used to refer to software library of framework. However, today first association is web API.

API testing is also being widely used by software testers due to the difficulty of creating and maintaining GUI-based automation testing. It involves directly testing APIs as part of integration testing, to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer, which implies that the execution of the API tests is usually much faster compared to GUI based testing. API testing is considered critical when an API serves as the primary interface to application logic since GUI tests can be difficult to maintain with the short release cycles and frequent changes commonly used with agile software development and DevOps.

API testing is used to determine whether APIs return the expected response (in the expected format) for a broad range of feasible requests, react properly to edge cases such as failures and unexpected/extreme inputs, deliver responses in an acceptable amount of time, and respond securely to potential security attacks. Service virtualization is used in conjunction with API testing to isolate the services under test as well as expand test environment access by simulating API/services that are not accessible for testing.

There are specific challenges in API testing:

- Main challenges in Web API testing are Parameter Combination, Parameter Selection, and Call Sequencing
- There is no GUI available to test the application which makes it difficult to give input values
- Validating and Verifying the output in a different system is little difficult for testers
- Parameter's selection and categorization is required to be known to the testers
- Exception handling function needs to be tested
- Coding knowledge is necessary for testers

Types of API testing:

- Functional Testing
- Load and Performance Testing
- Security Testing
- API documentation testing
- Negative Testing
- Unit testing
- Integration testing
- End-2-End testing

Testing can be both manual and automated. API testing commonly includes testing REST APIs or SOAP web services with JSON or XML message payloads being sent over HTTP(S), JMS and MQ.

The importance of API testing is growing nowadays, as it is not used only for testing standalone APIs, put also testing of web, desktop and mobile apps that have API in background. It Is almost impossible to imagine a modern app today that is not using API at some point, especially as modern apps are thin clients, decoupled and decomposed at layers.

As it was said at the beginning of this section, today API usually refers to Web API. Two main types of Web APIs are SOAP and REST. That is why we will continue with explaining what SOAP and REST are, but the first step is to revisit the HTTP protocol, as it is protocol used by all web APIs.

7.3.1 HTTP

HTTP (Hyper Text Transfer Protocol) is an application protocol by which web content is delivered from a web server to a web browser. It represents the foundation of data communication for the World Wide Web (image):

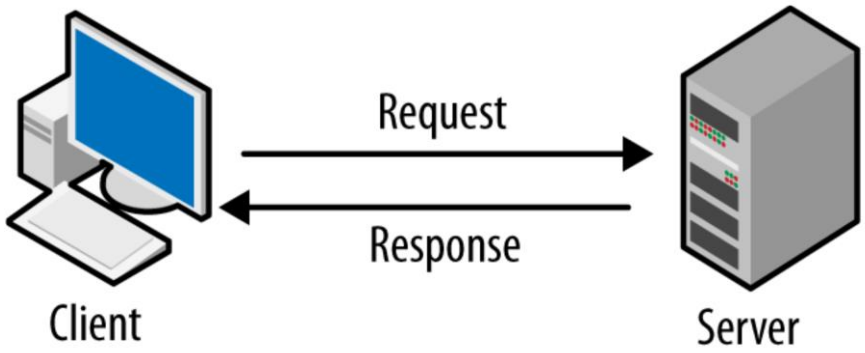


Figure X: Server — client interaction

HTTP methods

HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performs on the identified resource. These methods (HTTP/1.1) are listed in table bellow:

Request method	RFC	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	Optional	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	Optional	Yes	No	Yes	No
CONNECT	RFC 7231	Optional	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

HTTP/1.1 methods

An example of the client request would be:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

The server response to this request could be:

```
HTTP/1.1 200 OK
Date: Thursday, 5 May 2022 …
Content-Type: text/html, charset=UTF-8
…
<html>
<head>
   <title>
      IMI QA
   </title>
</head>
<body>
      Kvalitet i testiranje softvera
</body>
</html>
```

HTTP status codes

When a server responds to a request, it always includes the status code in the response header. The first digit of the status code specifies one of five standard classes of responses. These are:

- 1xx – Informational
- 2xx – Success
- 3xx – Redirection
- 4xx – Client error
- 5xx – Server error

7.3.2 REST

REST (Representational State Transfer) is a software architectural style that defines a set of constraints to be used for creating web services. Web services that conform to the REST architectural style can be called RESTful.

RESTful systems typically (but not always) communicate over HTTP with the same HTTP verbs (**GET, POST, PUT, DELETE, etc.**) that web browsers use to retrieve web pages and to send data to remote servers.

Web service APIs that adhere to the REST architectural constraints are called RESTful APIs. HTTP-based RESTful APIs are defined with the following aspects:

- a base URL, such as http://api.example.com/resources
- a media type that defines state transition data elements. The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as URL or as complex as Java applet
- standard HTTP methods (e.g. OPTIONS, GET, PUT, POST, DELETE).

Nowadays, REST is dominating in popularity for building public APIs. You can find many examples all over the Internet, especially since all big social media sites provide REST APIs so that developers can seamlessly integrate their apps with the platform. These public APIs also come with detailed documentation where you can get all the information you need to pull data through the API.

The REST architecture allows API providers to deliver data in multiple formats such as plain text, HTML, XML, YAML, and JSON, which is one of its most loved features.

For testing REST APIs, there is a large number of tools and frameworks. However, widely adopted is the Postman tool, and also a RESTAssured framework for Java language. An example of the test written using RESTAssured is shown below.

7.3.3 SOAP

SOAP stands for **Simple Object Access Protocol**. It's a messaging protocol for interchanging data in a decentralized and distributed environment. SOAP can work with any application layer protocol, such as HTTP, SMTP, TCP, or UDP. It returns data to the receiver in XML format. Security, authorization, and error-handling are built into the protocol and, unlike REST, it doesn't assume direct point-to-point communication. Therefore, it performs well in a distributed enterprise environment.

SOAP follows a formal and standardized approach that specifies how to encode XML files returned by the API. A SOAP message is, in fact, an ordinary XML file that consists of the following parts:

- Envelope (required) — This is the starting and ending tags of the message.
- Header (optional) - It contains the optional attributes of the message. It allows you to extend SOAP message in a modular and decentralized way.
- Body (required) — It contains the XML data that the server transmits to the receiver.
- Fault (optional) — It carries information about errors occurring during processing the message.

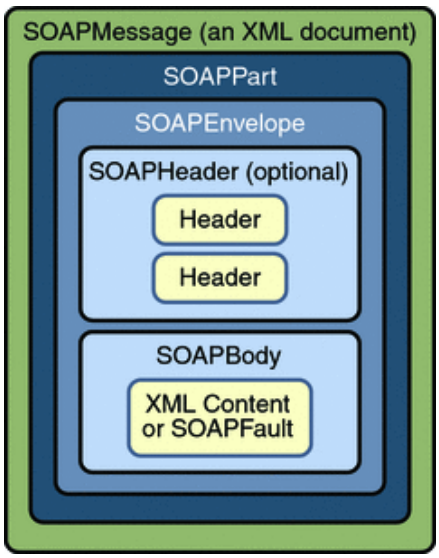


Figure X: SOAP message structure

Here is how an ordinary SOAP message looks like:

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
            env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
     <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
     <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
```

```
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
            env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
     <n:name>Fred Bloggs</n:name>
    </n:passenger>
 </env:Header>
 <env:Body>
    <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
     <p:departure>
       <p:departing>New York</p:departing>
       <p:arriving>Los Angeles</p:arriving>
       <p:departureDate>2007-12-14</p:departureDate>
       <p:departureTime>late afternoon</p:departureTime>
       <p:seatPreference>aisle</p:seatPreference>
     </p:departure>
     <p:return>
       <p:departing>Los Angeles</p:departing>
       <p:arriving>New York</p:arriving>
       <p:departureDate>2007-12-20</p:departureDate>
       <p:departureTime>mid-morning</p:departureTime>
       <p:seatPreference></p:seatPreference>
     </p:return>
    </p:itinerary>
 </env:Body>
</env:Envelope>
```

Here is a comparison table of SOAP and REST that highlights the main differences between the two API styles:

#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

For testing SOAP APIs, SoapUI is the common tool of choice for both functional and load testing. It is open-source and supports testing of REST APIs as well.

7.3.4 How to verify API tests

Verification of API tests is specific and differs from UI testing. Most common approaches are:

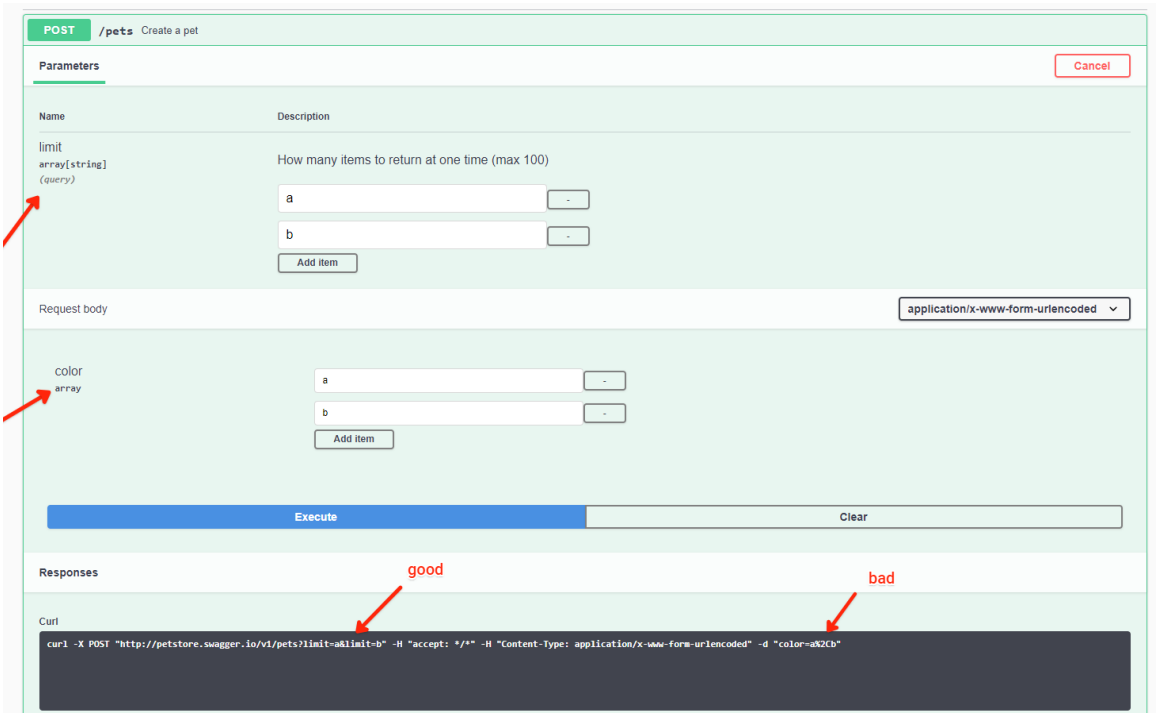
- Using HTTP status codes to assert the test. The client sends a request, and the server sends a response that has some of the following codes in its header:
  1x - Informational, 2xx - Success, 3xx - Redirection, 4xx - Client error, 5xx - Server error
- Verify response payload (JSON body)
- Verify response headers
- Verify correct application state and performance (operation was completed successfully but took an unreasonable amount of time, the test fails)
- Always try both positive and negative scenarios (200 status code is not good when the test is negative)

7.3.5 API testing tools and technologies

There are a lot of tools that can be used for both manual and automation API testing. Those are Postman, JMeter, SoapUI, Katalon studio, Teleric, etc. For automation testing these tools frequently use some scripting language like Java Script scripting and similar. There are a few tools only for manual testing like Wizdler and Swagger (OpenAPI).

Many tools were first developed for SOAP, but later added support for REST, and finally REST testing became the most important feature. That is a case for example with SoapUI, which even contains word "SOAP" in its name, but it is mainly used for REST testing today.

However, the most convenient way for API automation is use of framework in combination with some OOP language, like Java and RestAssured, Kotlin and Rest Assured, Ruby on Rails, Groovy and Grails, etc. Using this kind of API automation, increase possibilities of integrating tests with Continuous integration/Continuous development pipelines and related processes.



Example of manual API testing via Swagger

Code snip of automated test with JAVA/RestAssured framework

https://www.toolsqa.com/rest-assured/rest-api-end-to-end-test/
https://medium.com/chaya-thilakumara/rest-assured-api-testing-part-1-e96a2f284a6e