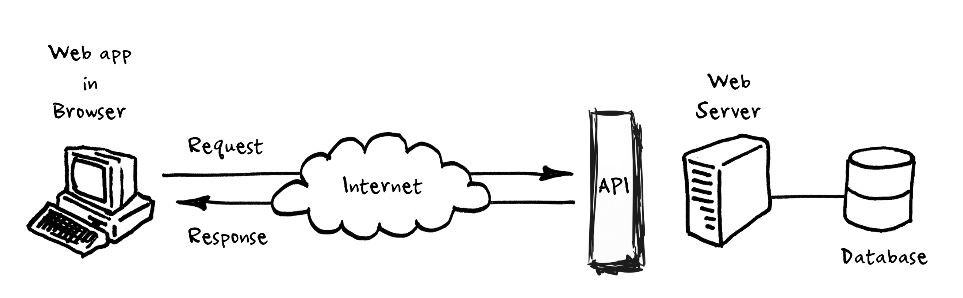
API-Testing

* **API (Application Programming Interface)** is a set of rules and protocols that allow different software applications to communicate and interact with each other.
* It defines how different components of software systems should interact, allowing them to exchange data and functionality without needing to understand the intricate details of each other's internal workings.



**How an API works:**

* **Client:** The client is the application or system that requests data or services from another application.
* **Server:** The server is the application or system that provides data or services to the client.
* **Request:** The client sends a request to the server, specifying the action it wants to perform (like retrieving data or performing an operation).
* **API:** The API acts as an intermediary between the client and server. It defines the available commands, data formats, and protocols that the client and server need to follow to communicate effectively.
* **Processing Request:** The server processes the client's request and performs the necessary actions.
* **Response:** After processing the request, the server sends back a response to the client. This response usually includes the requested data or a confirmation of the performed action.
* **Client Processing:** The client receives the response and processes the data or action accordingly.

**SOAP and REST API’s:**

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are two different approaches to building web services that allow communication between different software applications

|  |  |
| --- | --- |
| SOAP (Simple Object Access Protocol) | REST (Representational State Transfer) |
| 1. **Protocol and Standards:**   **SOAP:** SOAP is a protocol that defines a set of rules for structuring messages and uses XML (eXtensible Markup Language) as its message format. It relies on standards like WSDL (Web Services Description Language) for service description and UDDI (Universal Description, Discovery, and Integration) for service discovery. | 1. **Protocol and Standards:**   REST: REST is an architectural style rather than a protocol, and it operates over standard HTTP methods (GET, POST, PUT, DELETE). It uses a variety of data formats such as JSON, XML, HTML, or plain text for messages. |
| 1. **Message Format:**   SOAP: SOAP messages are usually XML-based, making them more verbose. They include a defined structure for the message body, headers, and often require XML namespaces. | 1. **Message Format:**   REST : REST messages can use various formats, but JSON is commonly used due to its lightweight and easy-to-parse nature. This makes REST messages more compact compared to SOAP messages. |
| **3.State:**  SOAP: SOAP is considered more stateful. It maintains the current state of communication between the client and server throughout the interaction. | **3.State:**  REST: REST is stateless. Each request from the client to the server must contain all the information necessary for the server to understand and fulfill the request. The server does not keep track of the client's previous interactions. |
| **4.HTTP Methods:**  SOAP: SOAP primarily uses the POST method for all its operations. This can limit some of the optimizations and caching mechanisms provided by the underlying HTTP protocol | **4.HTTP Methods:**  REST: REST uses the full range of HTTP methods, including GET (retrieve data), POST (create data), PUT (update data), and DELETE (remove data). This allows REST to leverage the inherent features of the HTTP protocol. |
| **5.Flexibility:**  SOAP: SOAP is more rigid in its structure due to its strict standards. This can lead to more complexity in certain scenarios but also provides a standardized and well-defined approach. | **5.Flexibility:**  REST: REST is more flexible and relies on the principles of using URLs and HTTP methods to interact with resources. This flexibility can make it easier to develop and consume APIs. |
| **5.Ease of Use:**  SOAP: SOAP APIs often require specialized libraries and tools to generate and parse SOAP messages, making it more complex to work with. | **5.Ease of Use:**  REST: REST APIs can be accessed directly from web browsers, and many programming languages have built-in support for making HTTP requests, making them easier to work with. |

In summary, SOAP and REST are different approaches to building APIs with their own strengths and weaknesses. SOAP is more structured and standardized but can be complex, while REST is more flexible and lightweight, making it more commonly used for web services today.

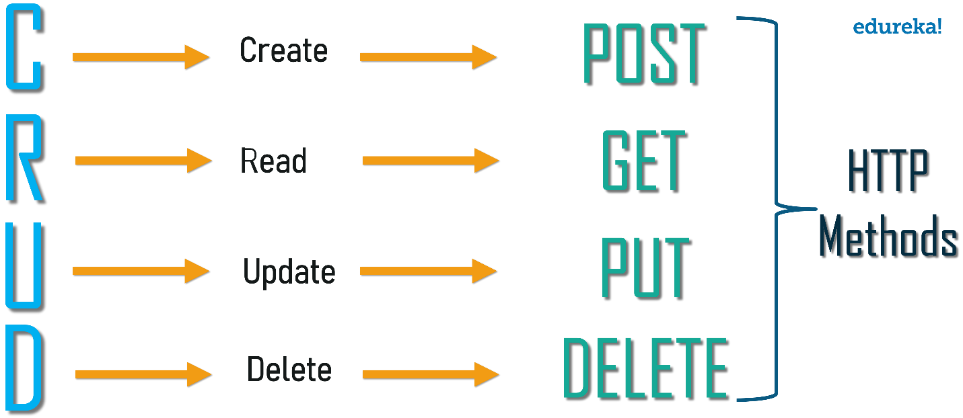
**Difference between API and Web Service?**

* All Web services are APIs but not all APIs are Web services.
* A Web service always needs a network to operate while APIs don’t need a network for operation

**Advantages of API Testing?**

* ***Test for Core Functionality:***API testing provides access to the application without a user interface. The core and code-level of functionalities of the application will be tested and evaluated early before the GUI tests. This will help detect minor issues which can become bigger during the GUI testing.
* ***Time Effective:***API testing usually is less time-consuming than functional GUI testing. The web elements in GUI testing must be polled, which makes the testing process slower. Particularly, API test automation requires less code so it can provide better and faster test coverage compared to GUI test These will result in cost saving for the testing project.
* ***Language-Independent:*** In API testing, data is exchanged using XML or JSON. These transfer modes are completely language-independent, allowing users to select any coding language when adopting automation testing services for the project.
* ***Easy Integration with GUI:*** API tests enable highly integrable tests, which is particularly useful if you want to perform functional GUI tests after API testing.

**HTTP Methods:**



* **CRUD** stands for Create, Read, Update, and Delete – four basic operations that can be performed on data.
* When working with APIs, CRUD operations refer to the standard set of actions that can be performed on resources.

1. **Create (POST):**
   * Purpose: To add a new resource to the server.
   * HTTP Method: POST
2. **Read (GET):**
   * Purpose: To retrieve information about a resource or a list of resources.
   * HTTP Method: GET
3. **Update (PUT or PATCH):**
   * Purpose: To modify an existing resource.
   * HTTP Method: PUT (if you send the entire resource) or PATCH (if you send only the changes)
4. **Delete (DELETE):**
   * Purpose: To remove a resource from the server.
   * HTTP Method: DELETE

**Status Codes:**

A screen shot of a chart

Description automatically generated

**Path and Query Parameters:**

A diagram of a path

Description automatically generated

**PostMan :**API Testing Tool- manual Testing

API Testing 🡪 Request(input)🡪 API 🡪 Response(output) 🡪 Validate Response (eg – status code, response time , data size , response body , cookies , headers).

**Workspace** 🡪 area where we maintain all the files.

1. Create new workspace

**PostMan Collections:**

What is Collection:

* Group of API Requests.

Workspace🡪 Collections 🡪 HTTP Requets

How to Run Collection:

**Response Validations:**

**1.Status Code:**

pm.test("Status code is 201", **function** () {

    pm.response.to.have.status(201);

});

pm.test("Successful POST request", **function** () {

    pm.expect(pm.response.code).to.be.oneOf([201, 202]);

});

**2.Headers:**

pm.test("Content-Type is present", **function** () {

    pm.response.to.have.header("Content-Type");

});

pm.test("Content-Type header value is application/json", **function** () {

    pm.expect(pm.response.headers.**get**('Content-Type')).to.eql('application/json; charset=utf-8')

});

**3.Cookies:**

pm.test("Cookie 'language' is present",**function**(){

pm.expect(pm.cookies.has('language')).to.be.**true**;

});

pm.test("Cookie 'language' has value lang-1234",**function**(){

pm.expect(pm.cookies.**get**('language')).to.eql('lang-1234');

});

**4.Response Time**

pm.test("Response time is less than 200ms", **function** () {   pm.expect(pm.response.responseTime).to.be.below(200);

});

**5.Response Body**

 "data": {

        "id": 2,

        "email": "janet.weaver@reqres.in",

        "first\_name": "Janet",

        "last\_name": "Weaver",

        "avatar": "https://reqres.in/img/faces/2-image.jpg"

    }

**const** jsonData = pm.response.json();

pm.test("Test Response Body values",**function**(){

pm.expect(jsonData.data.id).to.eql(2);

pm.expect(jsonData.data.first\_name).to.eql("Janet");

pm.expect(jsonData.data.last\_name).to.eql("Weaver");

});

Variables:

* Elements that can take different values.
* Used to reuse values at multiple places.
* Helps in avoid repetition.
* Helps us to avoid re-work when value changes.

Scope of Access🡪 Global , Collection , Environment , Local

Local 🡪

(in pre req script)

Environment:

* Create an API Requests
* Create environments
* Add key- value pairs🡪 Variables
* Refer the Variables in request.
* Select the Environment from the dropdown
* Run the Requests.
* Create more environment and execute requests.

API Chaining: Get Data from Response on one API and refer in another API.

* Create an API Request
* Use variables to parameterize the Value to be reffered.

jsonData = JSON.parse(responseBody);(or)

**const** jsonData = pm.response.json();

value=jsonData.name;

pm.collectionVariables.**set**("name",value);

* Add Scripts in Test Script to fetch value from reponse of 1st API.

{

    "name": "*{{name}}*",

    "job": "zion resident"

}

* Run and validate.
* For nodes🡪 <http://jsonpathfinder.com/>

Data Driven Testing:

{

"name":"*{{name}}*",

"job":"*{{job}}*"

}

1.csv file

2.json Data <https://www.convertcsv.com/csv-to-json.htm>