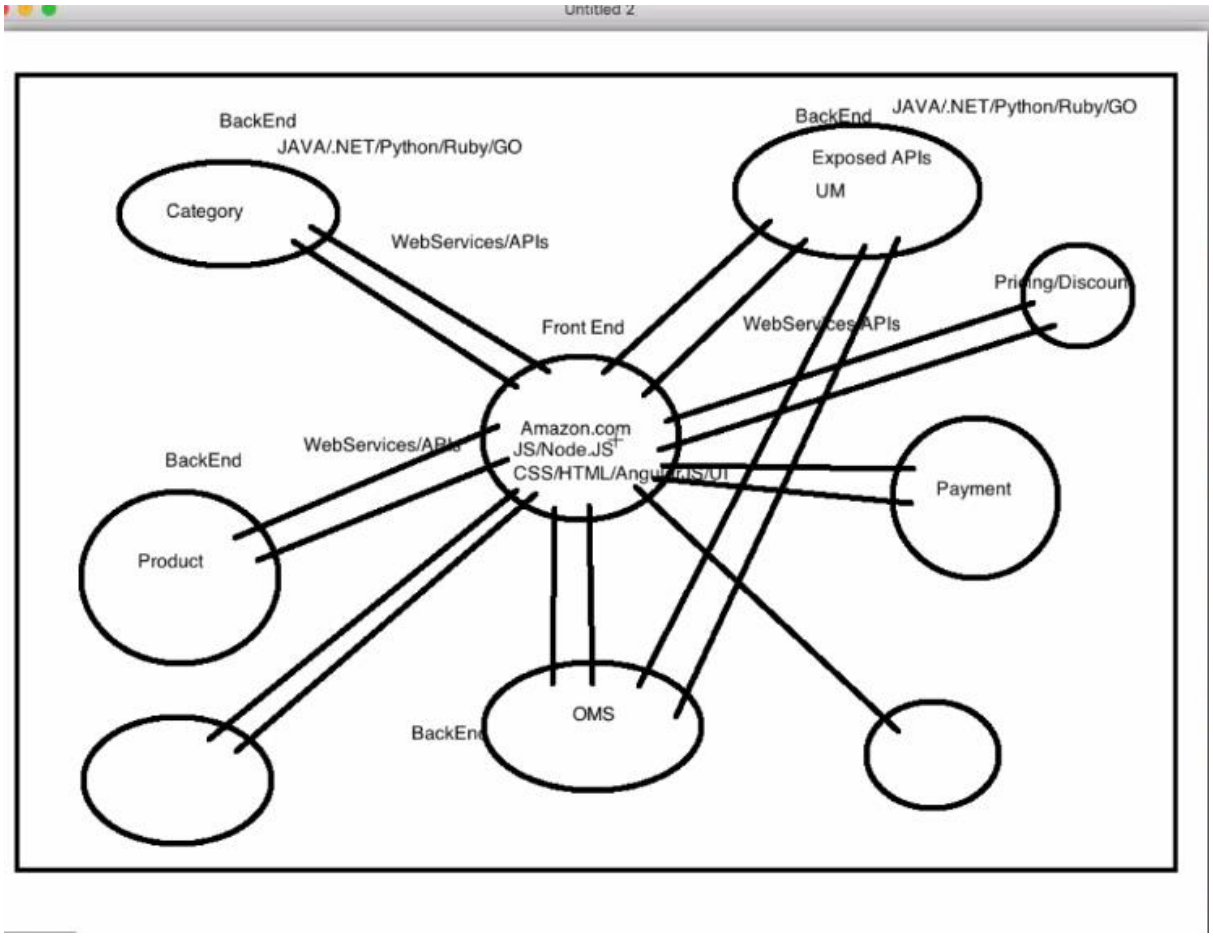
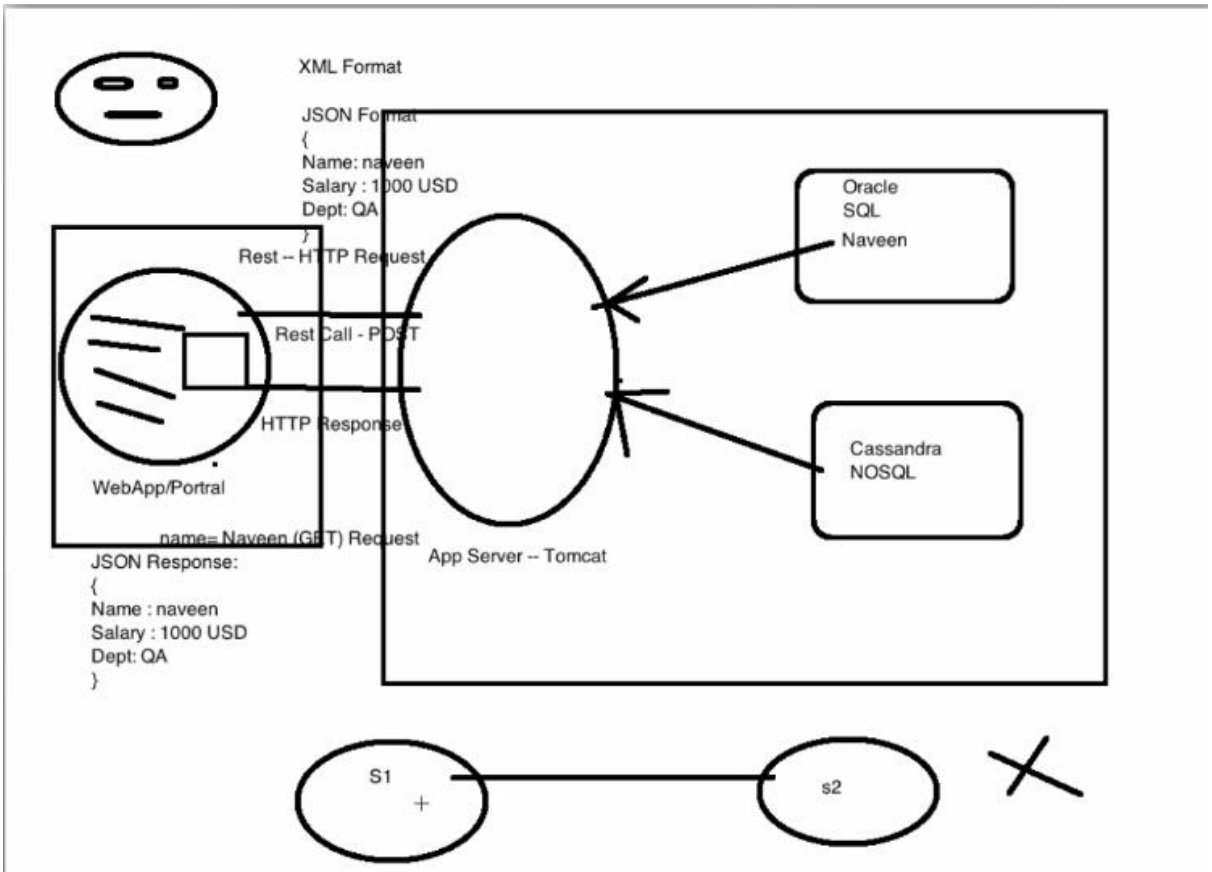
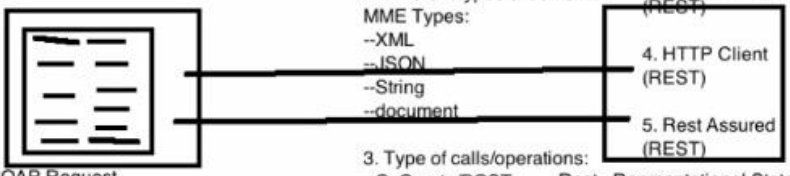
API Document





Two WS:
1. SOAP
2. REST

SOAP - Simple Object Access Protocol
(HTTP)



SOAP Request
1. EndPoint URL
http://www.xyz.com/login
http://www.xyz.com/createAccount
http://www.xyz.com/getEmployeeList
(DEV team)

2. Request (XML)
<username>admin</username>
<password>admin123</password>

3. Response (XML)
<status>success</status>
Login done
Home Page

1. End point URL
http://www.xyz.com/login

2. Different types of content:
MME Types:
--XML
--JSON
--String
--document

3. Type of calls/operations:
--C: Create/POST
--R: Retrieve/GET
--U: Update/PUT
--D: Delete/DELETE
(HTTP Methods)

4. Headers:
--content-type: JSON
--username/password : Authentication

5. Response:
JSON, XML, String message

6. Response Code:
200, 300, 404, 400, 500

Tools/Technologies: Manual + Automation Testing for WS:

1. SOAP UI (free/licensed-PRO)
--SOAP
--REST

2. POSTMAN
(REST)

3. Rest Client/Advance Rest Client
(REST)

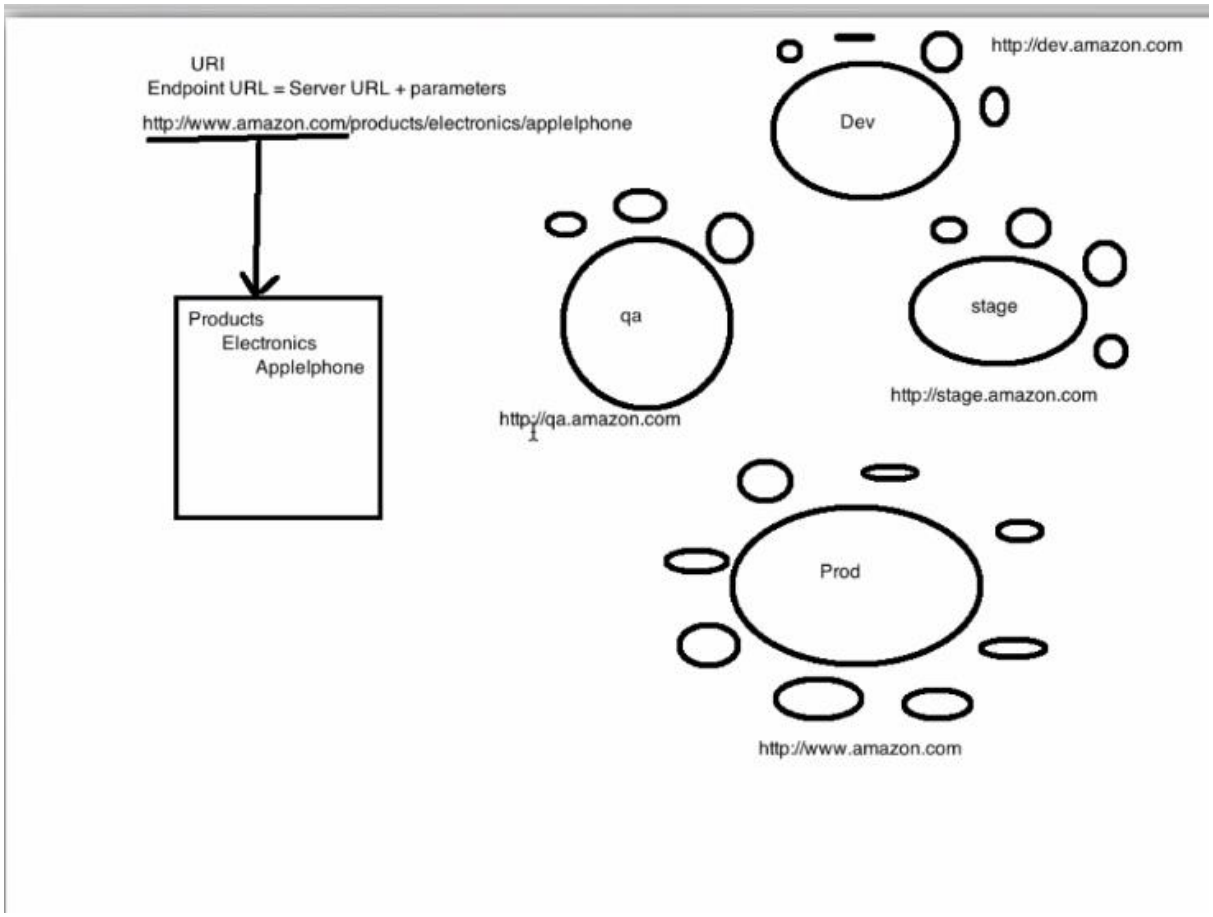
4. HTTP Client
(REST)

5. Rest Assured
(REST)

Rest - Represntational Stateful APIs
6. JerseyClient
(REST)

7. SOAP Client
(SOAP)

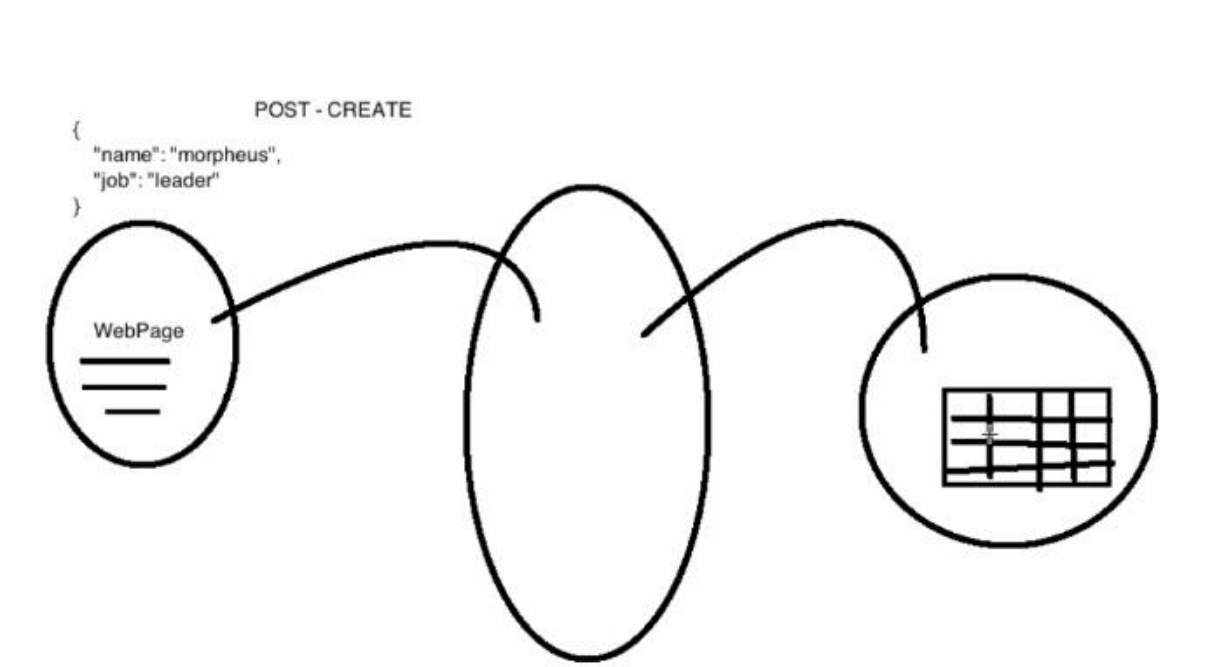
8. JMETER
(REST)

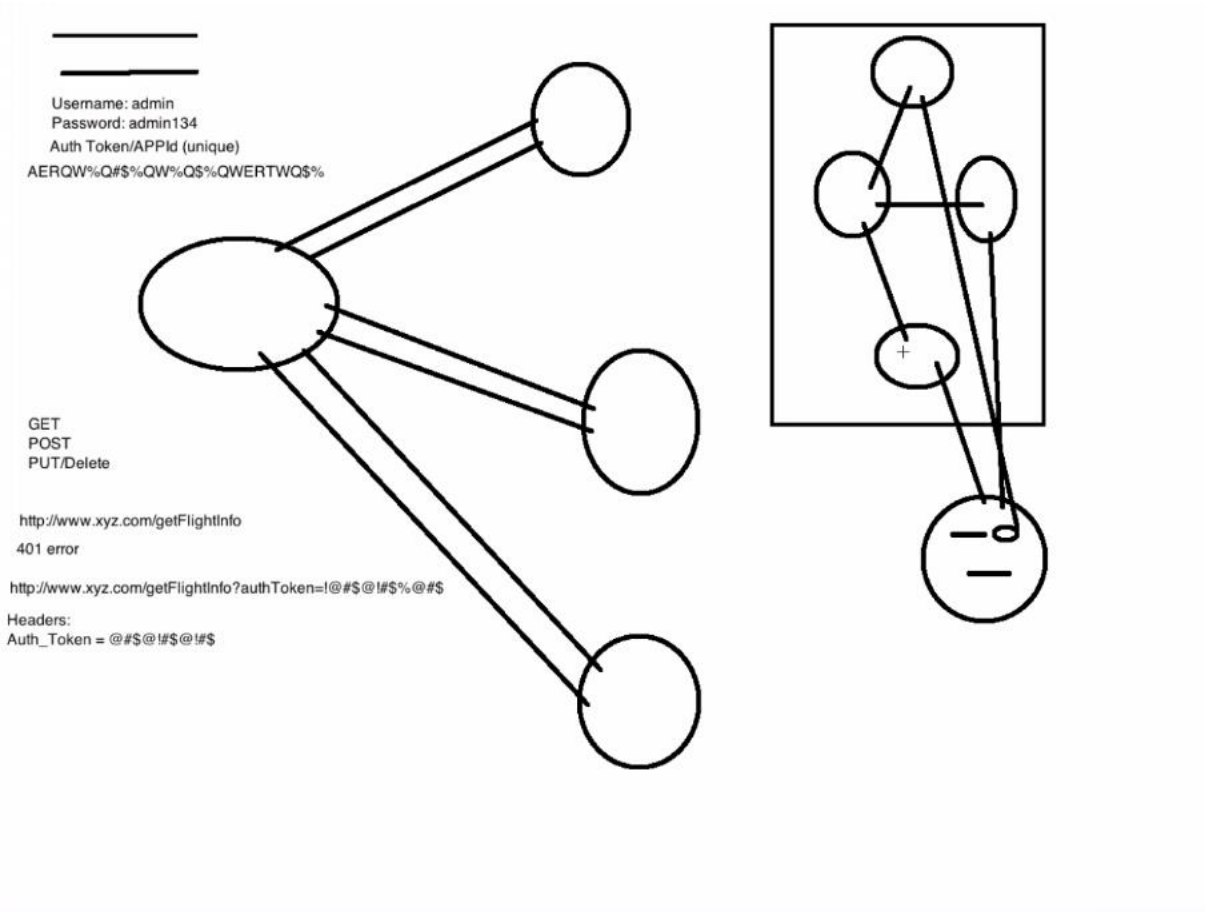


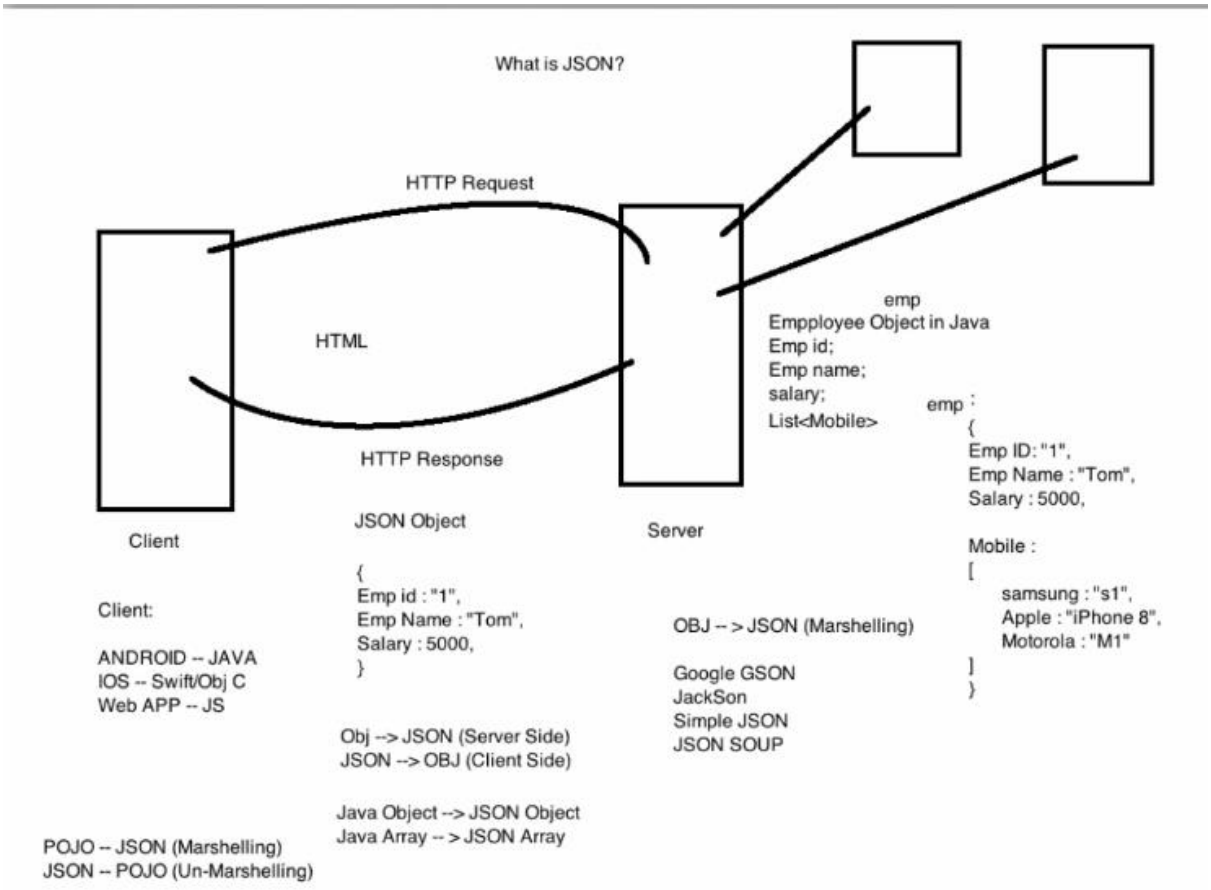
End Point URL will be given by developer

According to environment URL will change

Resources: JSON Payload, Headers, Authentication (Everything is a resource)


POST - CREATE
{
"name": "morpheus",
"job": "leader"
}
WebPage


Username: admin
Password: admin134
Auth Token/APPId (unique)
AERQW%Q#$%QW%Q$%QWERTWQ$%
GET
POST
PUT/Delete
http://www.xyz.com/getFlightInfo
401 error
http://www.xyz.com/getFlightInfo?authToken=!@#$@!#$%@#$
Headers:
Auth_Token = @#$@!#$@!#$



- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

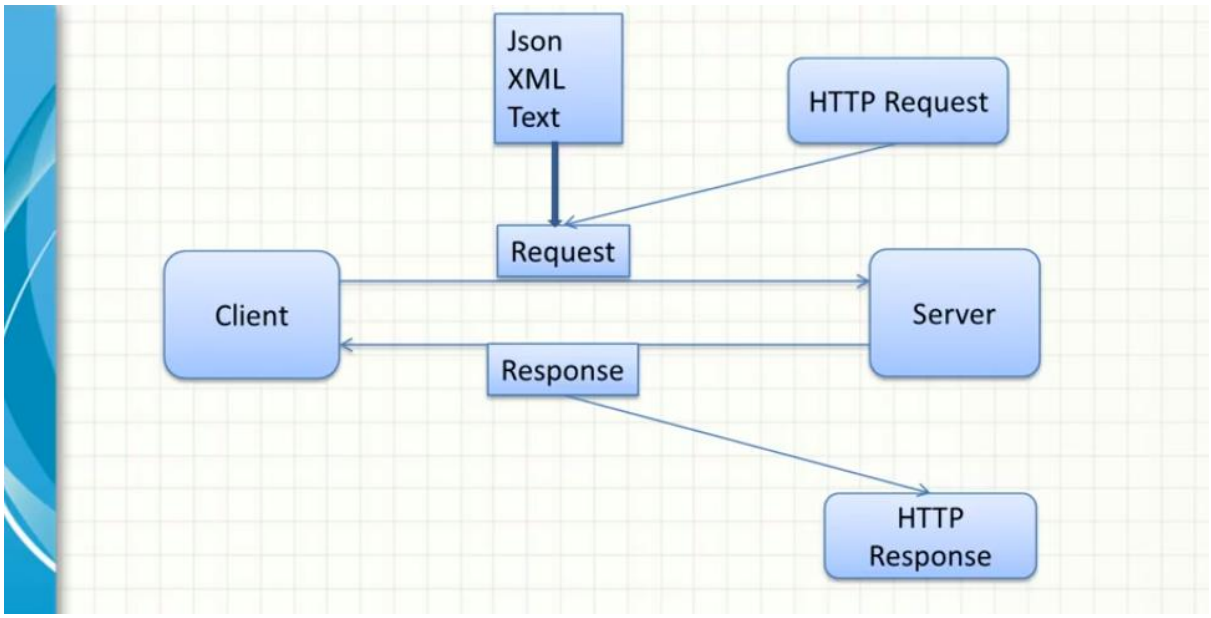
Private API Authentication

Private API requests require a staff members email address (or username if using Black Box) and password. There are 2 ways to pass this information to the API.

1. HTTP Basic Authentication
you may pass in the username and password via HTTP basic authentication. Libraries such as curl generally support this.

2. URL Parameters
The information may be passed directly in the URL as parameters. Example:

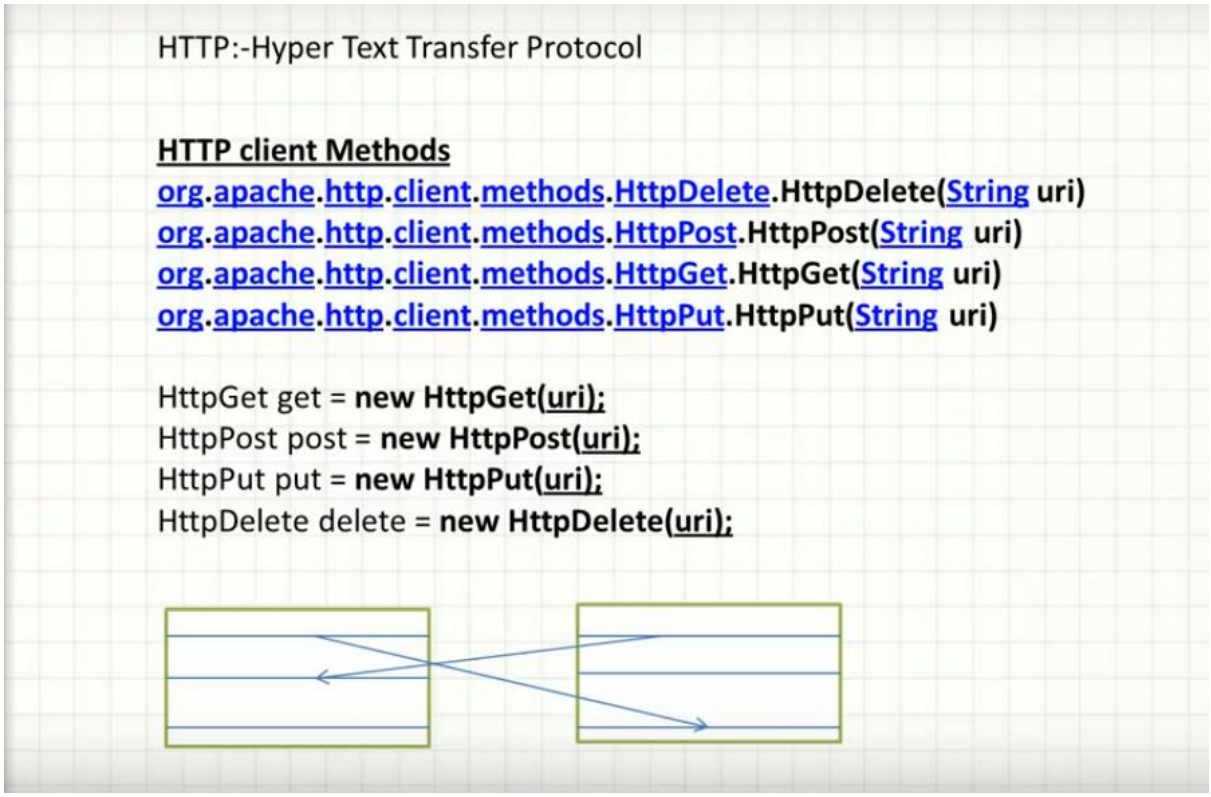
*/api/index.php?method=request.get**&username=tim@domain.com&password=abc12***



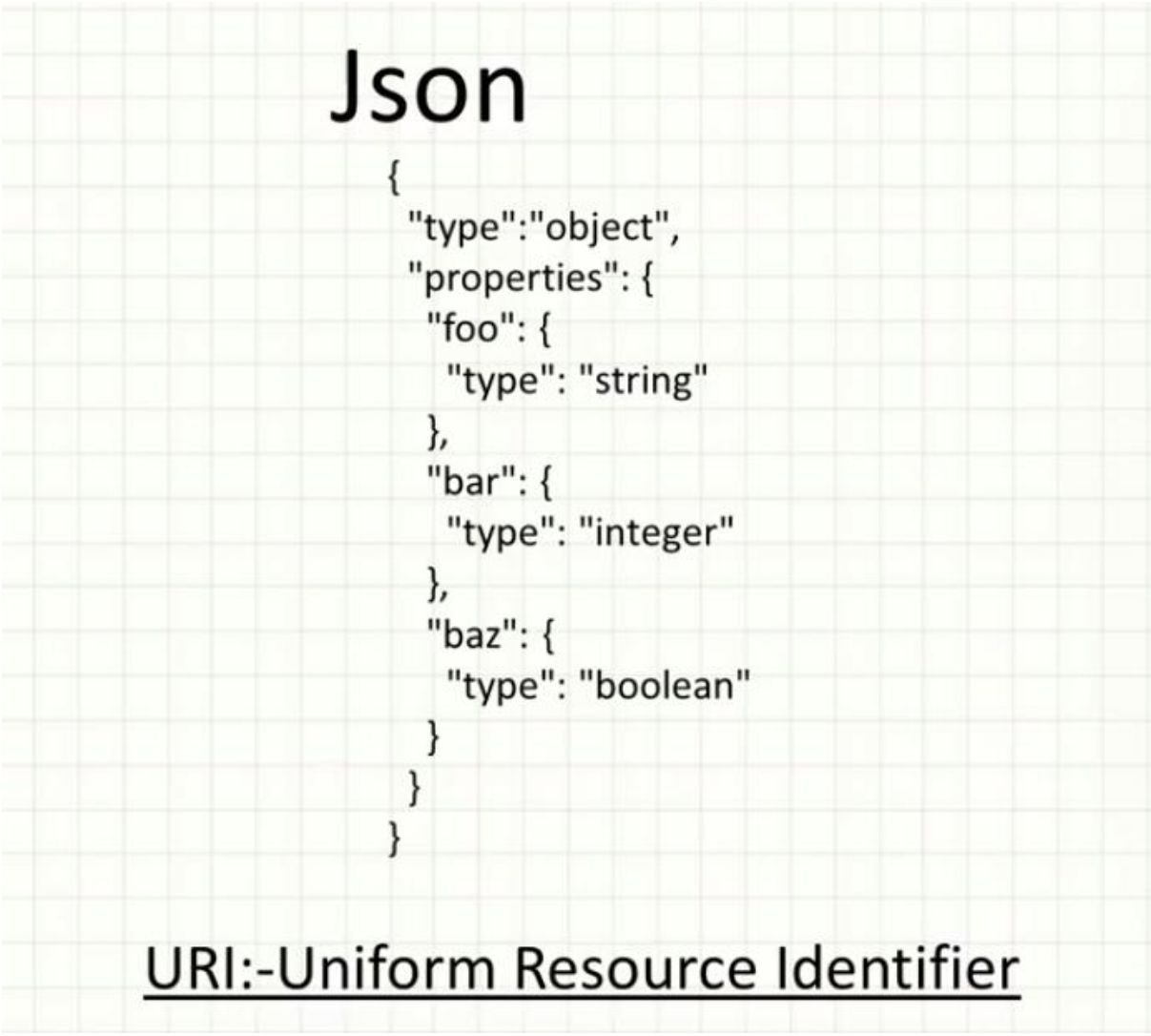
Request can be any format and response also can be any type

Ex request- xml but response can be JSON and vice versa.

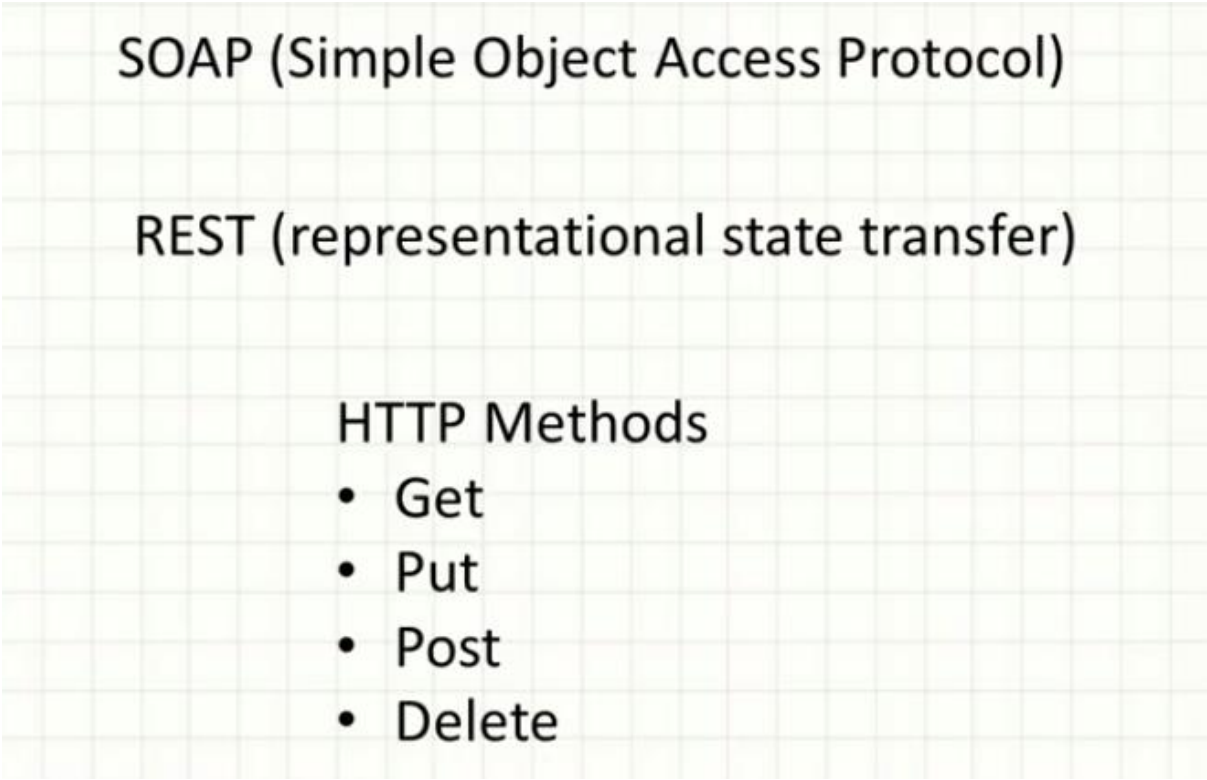
It depends on the client what data they consumed



HTTP- one text links to other text

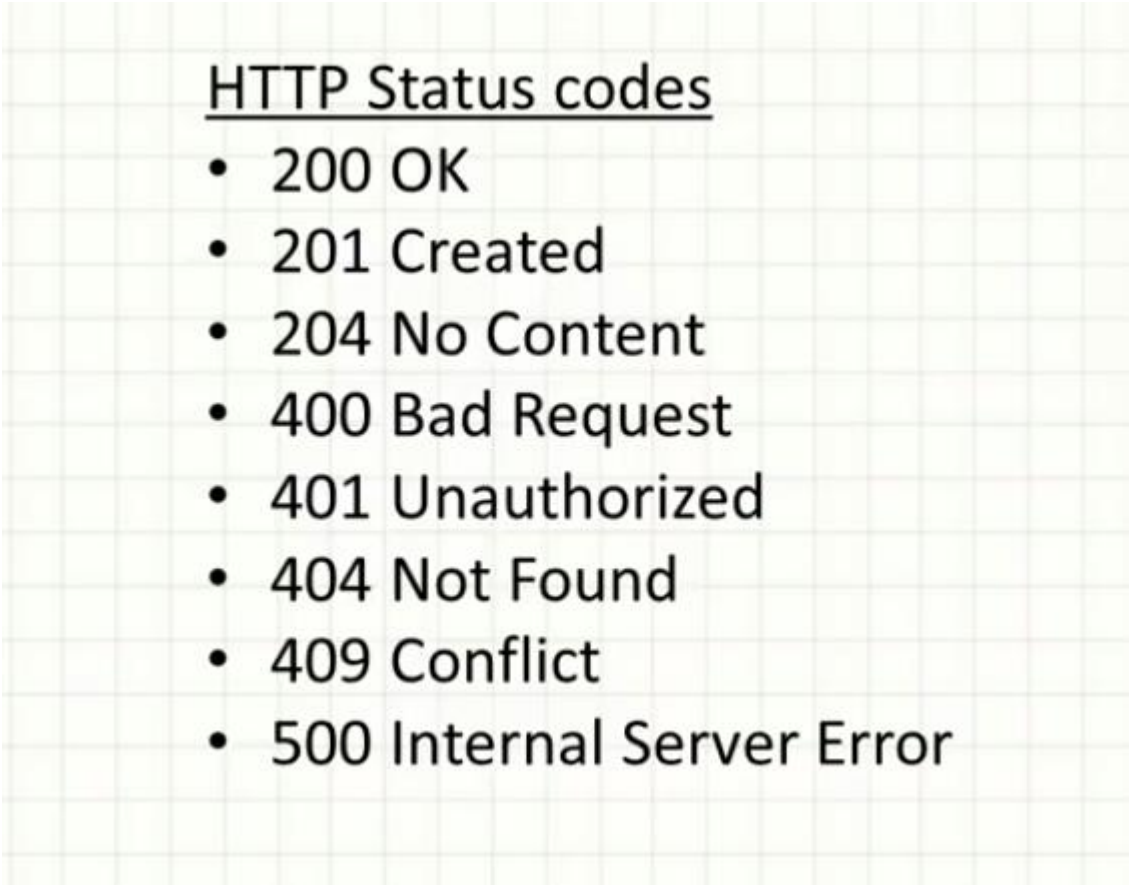


To find API uniqly we need URI or for every call we need URI

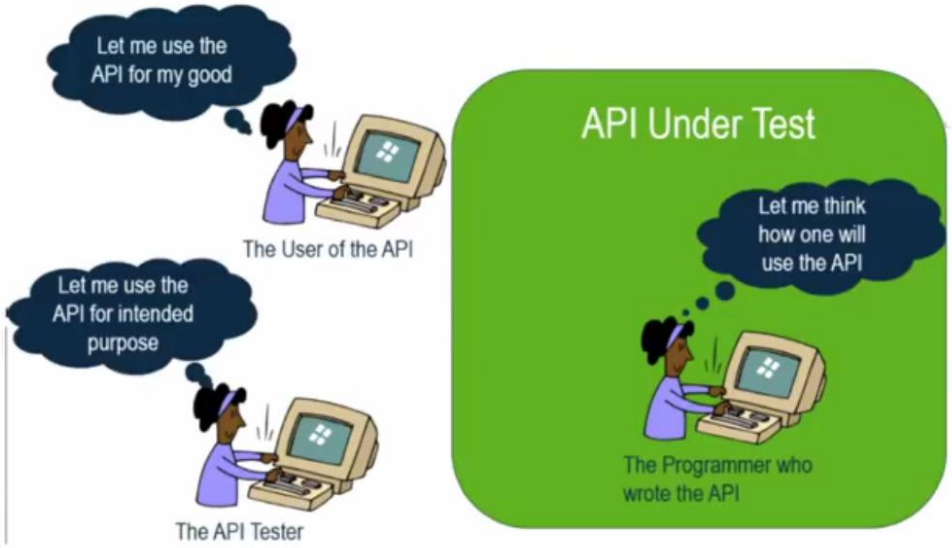


REST-Stateless, Fast and consume any kind of data (text, xml, json, string) and it is an architecture. Response depends on the what data client consumed.

SOAP- only xml



Post- use for create and update both

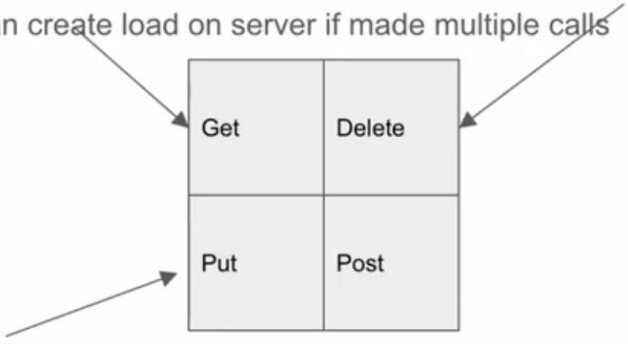


What is Rest-Assured

- Rest-Assured is an open-source Java Domain-Specific Language (DS
- Using Rest Assured we can eliminates large amount of code to test complex API response and output.
- It support both XML and JSON format.
- “Rest Assured is so easy even your grand mother can learn it.”

Get,Delete,Put will not create additional load on server while making multiple calls.

Post- This can create load on server if made multiple calls



Get Method- This method simply retrieves the data from server. No changes to server or resource.

Example- Check any news, searching anything on web and so on.

Post Method- This method perform the changes to server. In simple words post always creates resources in Server.

Example- Uploading the picture, Send a tweet or submitting the form and so on.

Note- POST is NOT idempotent. So if you retry the request N times, you will end up having N resources with N different URIs created on server.

Delete method- This method simply delete the data or resources from the server.

Example- Deleting any file, Deleting photos from account and so on.

Put method- This method update the existing resources. In simple words this method can update the existing file on server.

Example- Updating address, changing name, renaming files and so on.

Rest Assured

```
RequestSpecification request = RestAssured.given();

request.header("Content-Type", "application/json");

JSONObject requestParams = new JSONObject();

requestParams.put("id", "10");

request.body(requestParams.toJSONString());

Response response = request.post("/posts");

System.out.println(response.body().asString());
```

Test URL :

https://en.wikipedia.org/wiki/API_tes...

http://openweathermap.org/

http://restcountries.eu/rest/v1/name/...

http://parabank.parasoft.com/parabank...

http://freegeoip.net/json/yahoo.com

https://github.com/rest-assured/rest-...

http://www.hascode.com/2011/10/testin...

https://knowledgetester.org/2014/10/2...

Front end development language: JS, CSS, HTML

Backend development language: Java

Documentation is very imp for API : It contains all the information related to Classes , Functions, Payload, Input/Output, Authentication ,Headers, URL and that will be provided by Developer.

1st backend will created server configuration and API than web will come into picture.

Once backend is developed immediately testing can start

Backend is fin than why UI testing? At the end customer will work on UI and if GUI is not good customer won't satisfy, so UI testing is equally important to test.

Authorization/APP ID/ Authentication Token is very important in API's for the Security reason.

Authentication

Username and password

Secret key/token

Security question

Authentication key/token

JSON- is the universal notation to transfer the information-very light weight

Marshling: POJO-JSON -Serialization

Unmarshling: JSON-POJO -De Serialization

GET

GET requests are the most common and widely used methods in APIs and websites. Simply put, the GET method is used to retrieve data from a server at the specified resource. For example, say you have an API with a user's endpoint. Making a GET request to that endpoint should return a list of all available users.

Since a GET request is only requesting data and not modifying any resources, it's considered a safe and idempotent method.

Testing an API with GET requests

When you're creating tests for an API, the GET method will likely be the most frequent type of request made by consumers of the service, so it's important to check every known endpoint with a GET request.

At a basic level, these things should be validated:

Check that a valid GET request returns a 200 status code.

Ensure that a GET request to a specific resource returns the correct data. For example, GET /users returns a list of users.

GET is often the default method in HTTP clients, so creating tests for these resources should be simple with any tool you choose.

POST

In web services, POST requests are used to send data to the API sever to create or update a resource. The data sent to the server is stored in the request body of the HTTP request.

The simplest example is a contact form on a website. When you fill out the inputs in a form and hit Send, that data is put in the response body of the request and sent to the server. This may be JSON, XML, or query parameters (there's plenty of other formats, but these are the most common).

It's worth noting that a POST request is non-idempotent. It mutates data on the backend server (by creating or updating a resource), as opposed to a GET request which does not change any data. Here is a great explanation of idempotent.

Means POST create extra Load on the Server.

Testing an API with POST requests

The second most common HTTP method you'll encounter in your API tests is POST. As mentioned above, POST requests are used to send data to the API server and create or update a resource. Since POST requests modify data, it's important to have API tests for all of your POST methods.

Here are some tips for testing POST requests:

Create a resource with a POST request and ensure a 200 status code is returned.

Next, make a GET request for that resource, and ensure the data was saved correctly.

Add tests that ensure POST requests fail with incorrect or ill-formatted data.

POST Call-

define baseURI- given() - create JSONObject - put data- header-body-response

PUT

Similar to POST, PUT requests are used to send data to the API to create or update a resource. The difference is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly make have side effects of creating the same resource multiple times.

Testing an API with PUT requests

Testing an APIs PUT methods is very similar to testing POST requests. But now that we know the difference between the two (idempotency), we can create API tests to confirm this behaviour.

Check for these things when testing PUT requests:

Repeatedly call a PUT request always returns the same result (idempotent).

After updating a resource with a PUT request, a GET request for that resource should return the new data.

PUT requests should fail if invalid data is supplied in the request -- nothing should be

DELETE

The DELETE method is exactly as it sounds: delete the resource at the specified URL. This method is one of the more common in RESTful APIs so it's good to know how it works.

If a new user is created with a POST request to /users, and it can be retrieved with a GET request to /users/{{userid}}, then making a DELETE request to /users/{{userid}} will completely remove that user.

Testing an API with DELETE requests

DELETE requests should be heavily tested since they generally remove data from a database. Be careful when testing DELETE methods, make sure you're using the correct credentials and not testing with real user data.

A typical test case for a DELETE request would look like this:

Create a new user with a POST request to /users

With the user id returned from the POST, make a DELETE request to /users/{{userid}}

A subsequent GET request to /users/{{userid}} should return a 404 not found status code.

In addition, sending a DELETE request to an unknown resource should return a non-200 status code.

Top Five Challenges in Test Automation:

1. Effective Communicating and Collaborating in Team - That is how will get idea about our scope of testing like what to automate, what is the approach, what standard we need to follow, in how many browsers/platform/environments we need to test

2. Selecting a Right Tool - Which fulfil our business requirements, if we are going with Open source tool/Libraries like Selenium than we should be aware of the Pros and Cons.

3. Demanding Skilled Resources- Able to design and understand Framework from the scratch, should have understating of programming language

4. Selecting a Proper Testing Approach - What to test / what to automate, Reusability, How to reduce effort in both implementation and maintenance of test script and test suite? Will automation test suites be having a long lifetime? How to generate useful test reports and metrics?

5. High Upfront Investment Cost - Initial cost is high but always focus on ROI, Reusability, Maintenance, Regression, Smoke, Sanity suites.

3 Tier Application:

1. Presentation Tier: UI

2. Business Tier: API – Actual business logic will be written here (Message Layer)

3. DB Tier: Database

What to check in API: functionality, reliability, performance, security.

When to test in API: Before UI testing, so we can find defects at early stages and it is very fast and quick, less maintenance effort because there is no navigations, no UI and all.

API Testing is performed at the most critical layer, the Business Layer, where business logic processing is carried out, and all transactions between User Interface and Database happen.

API testing yields the highest ROI compared to all other testing type performed by testers.

Advantages of API testing:

Language independent

Data is exchanged via XML and JSON, so any language can be used for automation, independent from the languages used to develop the application. XML and JSON are typically structured data so the verification is fast and stable.

GUI independent

We can perform API testing within the application prior to GUI testing. Early testing will get feedback sooner and improve the team's productivity.

Improved test coverage

Most API/services have specifications, allowing us to create automated tests with high coverage, including functional testing (happy cases, negative cases) and non-functional testing.

Faster releases

It is common that executing a UI regression test suite takes 8-10 hrs while the same scenario with API testing takes only 1-2 hours.

Type of API Testing:

Validation Testing: as an assurance of the correct development against the stated user needs and requirements.

Functional Testing: Core and Code level testing.

Load Testing: Load testing is generally done after a specific unit, or the whole codebase has been completed .Load testing is performed to ensure the performance under both normal and at peak conditions. Will check with normal Load, Peak Load and Overload.

Runtime/Error Detection:

Monitoring: The runtime of the compiled code is tested for different implementation errors, handler failures, and other intrinsic issues to help ensure the codebase does not contain any insecurity.

Execution Errors: The code needs to show the response to valid requests and marks the failure for invalid requests predictably and in a known way, just the same with valid requests.

Resource Leaks: The requests which are invalid or commonly illegal will be submitted to the API to test the memory, resource, data, or operation leaks/insecurities

Error Detection: The known failure scenarios will be used to test the code to make sure the errors are appropriately figured and resolved.

Security Testing: They are used to ensure the API implementation is secure from external threats. Security testing also includes additional steps like the validation of encryption methodologies, and of the design of the access control for the API. It also contains the user rights management and validating authorization checks to access the resources.

Penetration Testing

Penetration testing is considered the second test in the process if auditing. In this testing type, the users with limited API knowledge will try to attack to assess the threat vector from an outside perspective, which is about functions, resources, processes, or aim to the entire API and its components.

Fuzz Testing

Fuzz testing is another step in the whole security audit process. In this testing type, a vast amount of random data (referred to as "noise" or "fuzz"), will be input into the system with the aim to try a forced crash or any negative behaviour. This fuzz test will help test the API in term of the limits to prepare for the "worst case scenarios."

API testing best practices:

Priorities: Positive tests-Negative tests (to verify invalid information, validation on special characters, uncommon input, etc.)

Functional and Non-functional testing priorities: Functional and non-functional testing should be performed at the same time with equal priority.

End-points management: organize the tests and the endpoints will influence the productivity, effectiveness, and maintenance of your tests.

Data-driven: The most important feature of any API test tools is the capability of the data-driven approach. Lacking data-driven ability will lead to test data hardcoded, duplicated test scripts, test verification, therefore causing massive effort at maintenance phase. Together with data-driven, there are some important notes for test data that we should pay attention seriously: data types, blank, empty, null string. With RESTful web services, the input data is typically in JSON format, and missing value of a specific key is considered differently (null, empty) in some cases. To avoid these ambiguous test scenarios, using a subset of input data models are highly suggested.

Challenges in API testing:

Initial Setup of API testing

Update the Schema of API testing: The schema (data formatting which handles requests and responses for the API) should be maintained during the testing process. Any changes in the program that create additional parameters for the API calls must be reflected in the schema configuration.

Testing Parameter Combinations: APIs handle communication between systems by assigning data values to parameters and passing these parameters through data requests. Testing all possible parameter request combinations in the API is necessary to detect the problems regarding the specific configurations.

Sequencing the API Calls: It is a challenge for testing teams when API needs to be in a specific order to work correctly. For instance, a call to return a user's profile information goes through before the profile is created, the request will return an error. The process can become even more significantly difficult when involving more applications.

Validating Parameters: Software testing team may find it challenging invalidating the parameters sent through API requests because of the sheer number of parameters and use cases for those parameters. Testers need to ensure all parameter data uses the correct string or numerical data type, fits within length restrictions, fits within the designated value range and pass other validation criteria.

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.

Basic tips that you need to know for API testing:

1. Understand API requirements -

What is the API's purpose?: What we need to test, what are the inputs/outputs, what are verification and validation points or 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?

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: what we need to verify like Status code, Status Line, Response time or memory etc.

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 case. 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.

5. Leverage automation capability for API testing

Leverage automation capability for your API testing as much and as early as possible. Here are some significant benefits of automating API tests:

Test data and execution history can be saved along with API endpoints. This makes it easier to rerun tests later.

API tests are stable and changed with care. An API reflects a business rule of the system. Any change in the API needs an explicit requirement; so testers can always stay alert of any changes and adjust them on time.

Test execution is much faster compared to Web UI test

API testing is considered as black-box testing in which the users send input and get output for verification. Automation with a data-driven approach — i.e. applying different datasets in the same test scenario — can help increase API test coverage

Data input and output follows some specific templates or models so that you can create test scripts only once. These test scripts can also be reused throughout the entire testing project.

API tests can be performed at the early stage of the software development lifecycle. An automation approach with mocking techniques can help verify API and its integration before the actual API is developed. Hence, the level of dependency within the team is reduced.

6. Choose a suitable automation tool

Does the tool support the authorization methods that your AUT services require? Here are some authorization methods that your API can use:

No Auth

Bearer Token

Basic auth

Digest Auth

NTLM Authentication

OAuth 1.0

OAuth 2.0

Hawk Authentication

AWS Signature

This is an essential task since you cannot start testing an API without authorization.

Does the tool support data-driven methods?

7. Choose suitable verification methods

While the response status code tells the status of the request, the response body content is what an API returns with the given input. An API response content varies from data types to sizes. The responses can be in plain text, a JSON data structure, an XML document, and more. They can be a simple few-word string (even empty), or a hundred-page JSON/XML file. Hence, it is essential to choose a suitable verification method for a given API.

8. 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's 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 behaviours with different levels of authorization.

9. Live testing process

Test scheduling with built-in test commands

Integration with test management tools and defect tracking tools

Continuous Integration with various leading CI tools

Visual log reports generation

Once the testing process is completed, you can get the result of those tests every day. If failed tests occur, you can check the outputs and validate issues to have proper solutions.

10. Do not underestimate API automation testing

API testing flow is quite simple with three main steps:

Send the request with necessary input data

Get the response having output data? Verify that the response returned as expected in the requirement

API Interview Question

What is API?

An API (Application Programming Interface) is a software intermediary that enables two applications to communicate with each other. It comprises a number of subroutine definitions, logs, and tools for creating application software.

In an API testing interview, you could be asked to give some API examples, here are the well-known ones: Google Maps API, Amazon Advertising API, Twitter API, YouTube API, etc.

What is API Testing?

In the modern development world, many web applications are designed based on three-tier architecture model. These are:

Presentation Tier – User Interface (UI)

Logic Tier – Business logic is written in this tier. It is also called Business Tier. (API)

Data Tier – Here information and data is stored and retrieved from a Database. (DB)

Ideally, these three layers (tiers) should not know anything about the platform, technology, and structure of each other. We can test UI with GUI testing tools and we can test logic tier (API) with API testing tools. Logic tier comprises of all of the business logic and it has more complexity than the other tiers and the test executed on this tier is called as API Testing.

API testing tests logic tier directly and checks expected functionality, reliability, performance, and security. In the agile development world, requirements are changing during short release cycles frequently and GUI tests are more difficult to maintain according to those changes. Thus, API testing becomes critical to test application logic.

In GUI testing we send inputs via keyboard texts, button clicks, drop-down boxes, etc., on the other hand in API testing we send requests (method calls) to the API and get output (responses). These APIs are generally REST APIs or SOAP web services with JSON or XML message payloads being sent over HTTP, HTTPS, JMS, and MQ.

What are main differences between API and Web Service?

All Web services are APIs but not all APIs are Web services.

All web services need to be exposed over web (HTTP) but All APIs need not be exposed over web (i.e. HTTP)

Web services might not contain all the specifications and cannot perform all the tasks that APIs would perform.

A Web service uses only three styles of use: SOAP, REST and XML-RPC for communication whereas API may be exposed to in multiple ways e.g. DLL files in C/C++, Jar files/ RMI in java, Interrupts in Linux kernel API etc.

A Web service always needs a network to operate while APIs don't need a network for operation.

What are some architectural styles for creating a Web API?

Bellows are four common Web API architectural styles:

HTTP for client-server communication

XML/JSON as formatting language

Simple URI as the address for the services

Stateless communication

Who can use a Web API?

Web API can be consumed by any clients which support HTTP verbs such as GET, PUT, DELETE, and POST.

Since Web API services do not require configuration, they can be easily used by any client.

In fact, even portable devices such as mobile devices can easily use Web API, which is undoubtedly the biggest advantage of this technology.

What are the 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 the 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 automation. These will result in the 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 code language when adopting automation testing services for the project.

Easy Integration with GUI: API tests enable highly integral tests, which is particularly useful if you want to perform functional GUI tests after API testing. For instance, simple integration would allow new user accounts to be created within the application before a GUI test started.

Some common protocols used in API testing?

Many protocols are now available to be used in API testing, such as JMS, REST, HTTP, UDDI and SOAP

What are the common API testing types?

Most tests fit broadly into these following nine categories:

Validation Testing

Functional Testing

Load testing

Runtime/ Error Detection

Security testing

Penetration (pen) testing

Fuzz testing

What are tools could be used for API testing?

Postman

Katalon Studio

SoapUI

Tricentis Tosca

Apigee

JMeter

Rest-Assured

What are differences between API Testing and UI Testing?

API enables communication between two separate software systems. A software system implementing an API contains functions or subroutines that can be executed by another software system.

On the other hand, UI (User Interface) testing refers to testing graphical interface such as how users interact with the applications, testing application elements like fonts, images, layouts etc. UI testing basically focuses on look and feel of an application.

What are major challenges faced in API testing?

Parameter Selection

Parameter Combination

Call sequencing

Output verification and validation

Another important challenge is providing input values, which is very difficult as GUI is not available in this case.

What are the testing methods that come under API testing?

Unit testing

End to End Integration testing

Functional testing

Load testing to test the performance under load

Usability and Reliability testing to get consistent results

Security and Penetration testing to validate all types of authentication

Automation testing to create and run scripts that require regular API calls

What is API documentation?

The API documentation is a complete, accurate technical writing giving instructions on how to effectively use and integrate with an API. It is a compact reference manual that has all the information needed to work with the API, and helps you answer all the API testing questions with details on functions, classes, return types, arguments, and also examples and tutorials.

What are API documentation templates that are commonly used?

There are several available API documentation templates help to make the entire process simple and straightforward, such as:

Swagger

Slate

FlatDoc

API blueprint

RestDoc

Web service API specification

What is a Restful Web Services?

Mostly, there are two kinds of Web Services which should be remembered:

SOAP (Simple Object Access Protocol) – an XML-based method to expose web services.

REST (Representational State Transfer) is an architectural style for developing web services over HTTP protocol and uses HTTP method to define actions. It revolves around resource where every component being a resource that can be accessed through a shared interface using standard HTTP methods.

Web services developed in the REST style are referred to as RESTful web services. These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation like JSON and a set of HTTP methods.

What is a "Resource" in REST?

REST architecture treats any content as a resource, which can be either text files, HTML pages, images, videos or dynamic business information.

REST Server gives access to resources and modifies them, where each resource is identified by URIs/ global IDs.

What are the core components of an HTTP request?

An HTTP request contains five key elements:

An action showing HTTP methods like GET, PUT, POST, DELETE.HEAD

Uniform Resource Identifier (URI), which is the identifier for the resource on the server.

HTTP Version, which indicates HTTP version, for example-HTTP v1.1.

Request Header, which carries metadata (as key-value pairs) for the HTTP Request message. Metadata could be a client (or browser) type, format supported by the client, format of a message body format, cache settings, and so on.

Request Body, which indicates the message content or resource representation.

What is URI? What is the main purpose of REST-based web services and what is its format?

URI stands for Uniform Resource Identifier. It is a string of characters designed for unambiguous identification of resources and extensibility via the URI scheme.

The purpose of a URI is to locate a resource(s) on the server hosting of the web service.

A URI's format is <protocol>://<service-name>/<ResourceType>/<ResourceID>

What is payload in Restful Web services?

The "payload" is the data you are interested in transporting. This is differentiated from the things that wrap the data for transport like the HTTP/S Request/Response headers, authentication, etc.

What is the upper limit for a payload to pass in the POST method?

<GET> appends data to the service URL. But, its size shouldn't exceed the maximum URL length. However, <POST> doesn't have any such limit.

So, theoretically, a user can pass unlimited data as the payload to POST method. But, if we consider a real use case, then sending POST with large payload will consume more bandwidth. It'll take more time and present performance challenges to your server. Hence, a user should take action accordingly.

Enlist some of the API examples which are very well known and popular.

There are several such examples, enlisted below are some most popular ones:

Google Maps API: These are designed mainly for mobile and desktop use with the help of flash interface and JavaScript.

Amazon Advertising API: Amazon is known for their products and thus their advertising API accesses their product to discover their functionality and thus advertise accordingly.

Twitter: The API for twitter is usually in two categories, one for accessing data and the other for interacting with twitter search.

YouTube: This API used for YouTube includes various functionalities including videos, live streaming, player, etc.

Differentiate API testing and Unit Testing.

Unit testing is usually performed by testers where every functionality is tested separately.

API testing is performed by the testers for end to end testing of the functionality.

Unit testing have the limited scope of testing, thus basic functionalities are only considered for testing.

API testing have the broader scope of testing, all issues that are functional are considered for testing.

Unit testing is a form of white box testing.

API testing is a form of black box testing.

Usually, unit testing is done before the code is included in the build.

API testing is performed after the build is ready for testing.

In Unit testing the Source code is involved in this form of testing.

In API testing Source code is not involved in this form of testing.

What is Rest Assured?

In order to test REST APIs, We have REST Assured library. It is developed by Jay Way Company and it is a really powerful catalyser for automated testing of REST-services. REST-assured provides a lot of nice features, such as DSL-like syntax, XPath-Validation, Specification Reuse, easy file uploads and with those features we will handle automated API testing much easier.

Rest Assured has a gherkin type syntax which is as BDD (Behaviour Driven Development):

Also, you can get JSON response as a string and send it to the JsonPath class and use its methods to write more structured tests.

How to Make a POST Request with Rest Assured?

Rest API URL – URL of the Rest API

API Body – Body of the Rest API. Example: {“key1”:”value1”,”key2”:”value2”}

setContentType() – Pass the “application/json”, “application/xml” or “text/html” etc. headers to setContenType() method.

Authentication credentials – Pass the username and password to the basic() method or if there is no authentication leave them blank basic(“”,””)

SOAP or Rest APIs, which method to use?

SOAP is the heavyweight choice for Web service access. It provides the following advantages/disadvantage when compared to REST:

SOAP is not very easy to implement and requires more bandwidth and resources.

SOAP message request is processed slower as compared to REST and it does not use web caching mechanism.

WS-Security: While SOAP supports SSL (just like REST) it also supports WS-Security which adds some enterprise security features.

WS-Atomic Transaction: Need ACID Transactions over a service, you’re going to need SOAP.

WS-ReliableMessaging: If your application needs Asynchronous processing and a guaranteed level of reliability and security. Rest doesn’t have a standard messaging system and expects clients to deal with communication failures by retrying.

If the security is a major concern and the resources are not limited then we should use SOAP web services. Like if we are creating a web service for payment gateways, financial and telecommunication related work, then we should go with SOAP as here high security is needed.

REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

Since REST uses standard HTTP, it is much simpler.

REST is easier to implement, requires less bandwidth and resources.

REST permits many different data formats whereas SOAP only permits XML.

REST allows better support for browser clients due to its support for JSON.

REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.

If security is not a major concern and we have limited resources. Or we want to create an API that will be easily used by other developers publicly then we should go with REST.

If we need Stateless CRUD operations then go with REST.

REST is commonly used in social media, web chat, mobile services and Public APIs like Google Maps.

RESTful service returns various MediaTypes for the same resource, depending on the request header parameter “Accept” as application/xml or application/json for POST and /user/1234.json or GET /user/1234.xml for GET.

REST services are meant to be called by the client-side application and not the end user directly.