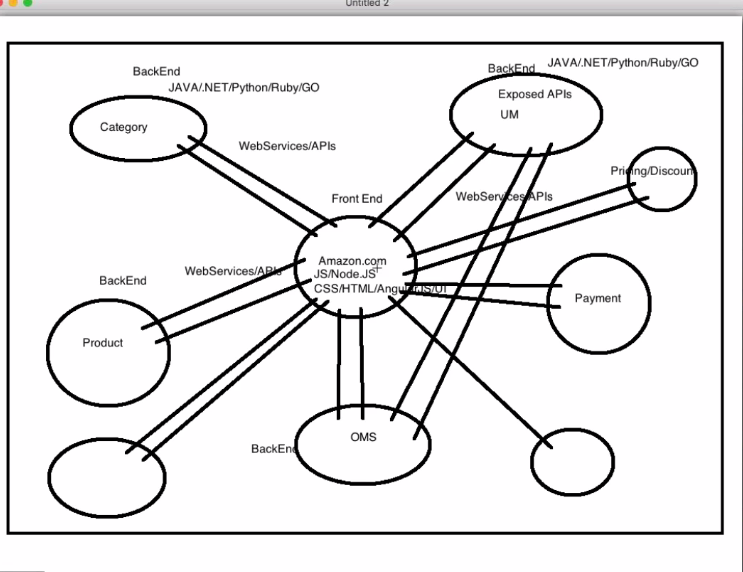
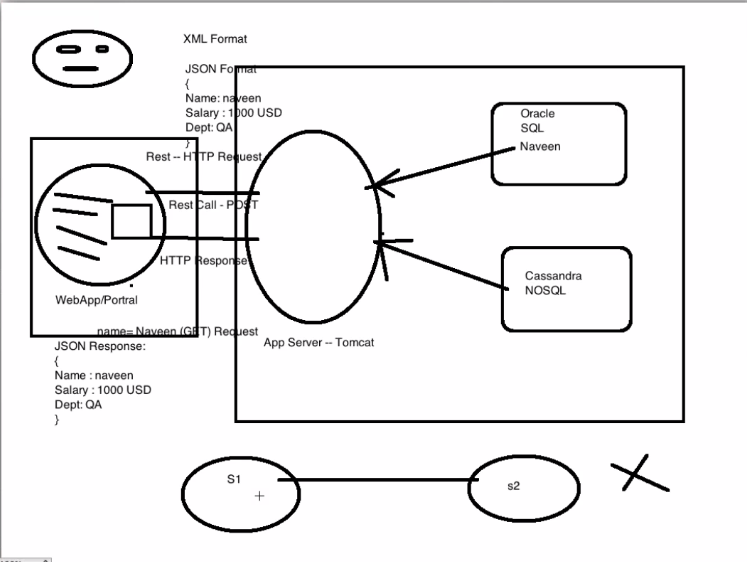
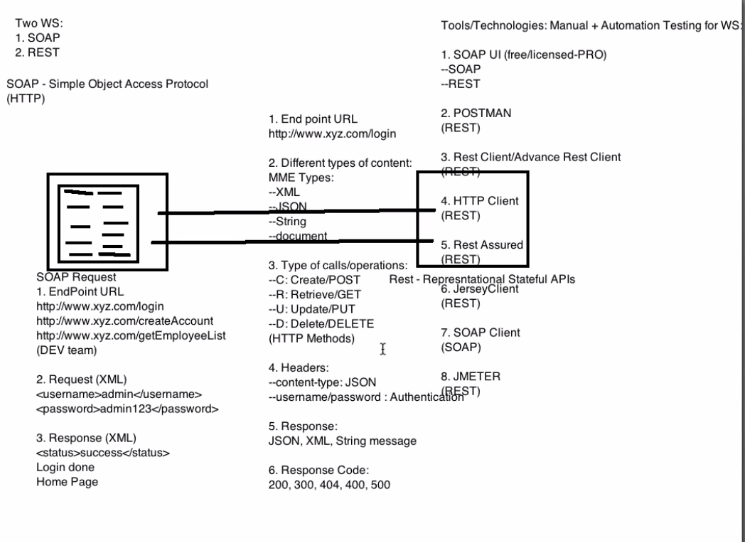
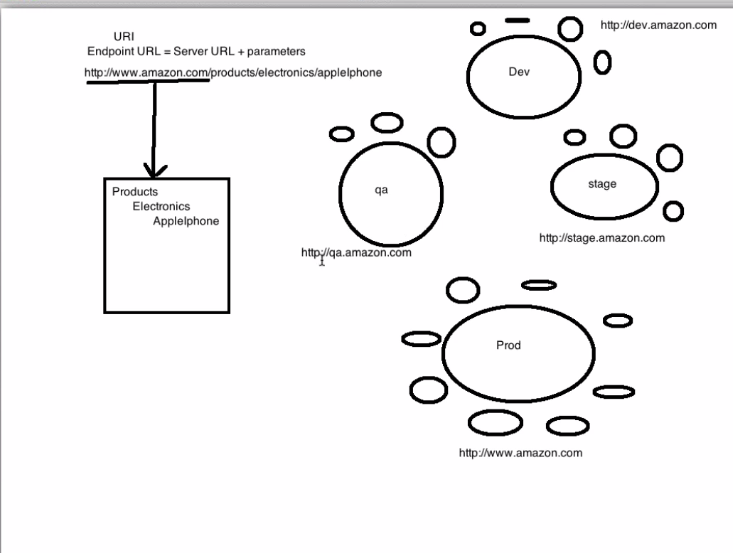
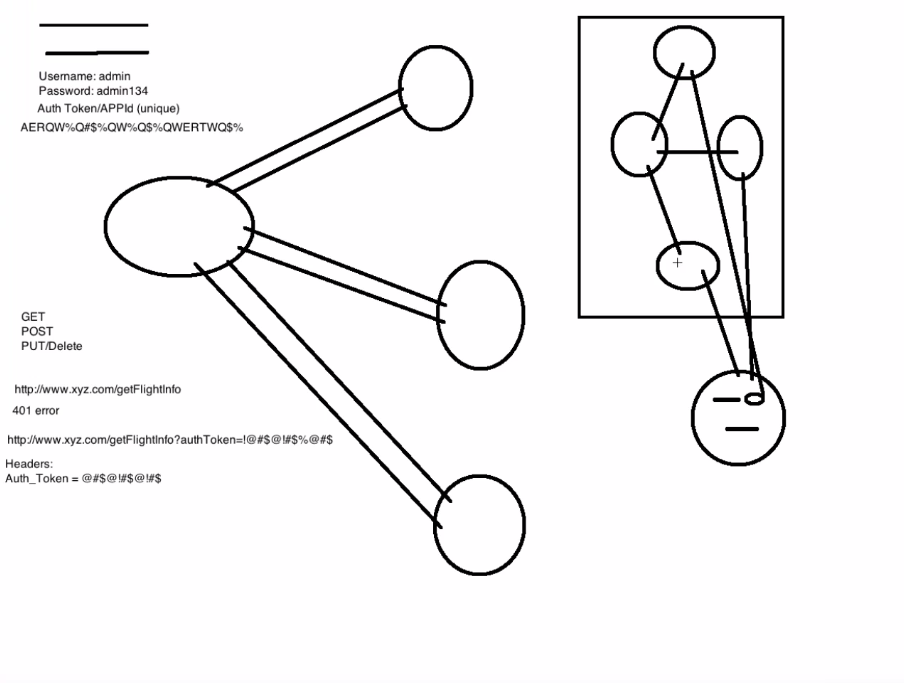
**API Document**

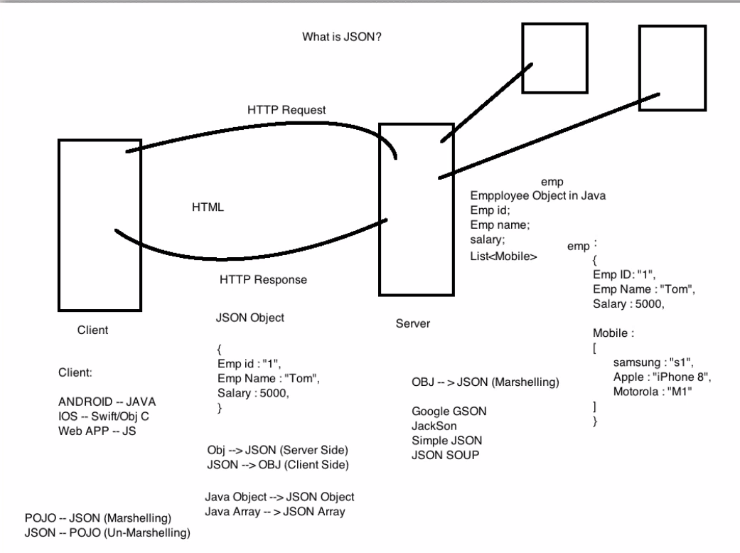




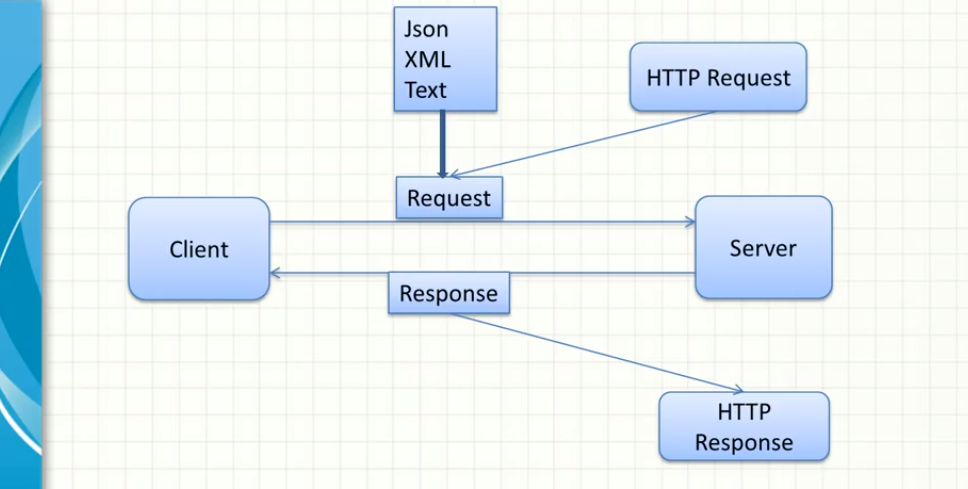








* 1xx (Informational): The request was received, continuing process
* 2xx (Successful): The request was successfully received, understood, and accepted
* 3xx (Redirection): Further action needs to be taken in order to complete the request
* 4xx (Client Error): The request contains bad syntax or cannot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



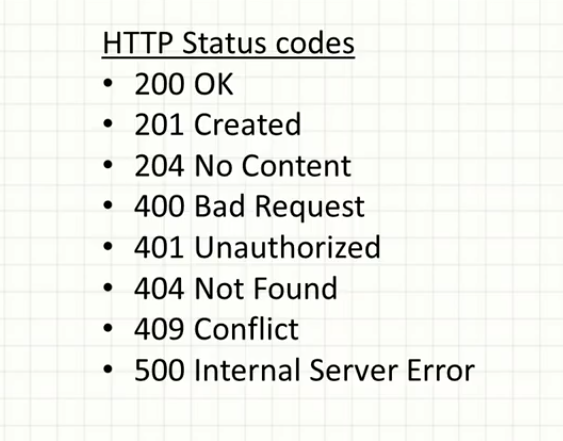
Request can be any format and response also can be any type

Ex request- xml but response can be JSON and vice versa and It depends on the client, what data they consumed

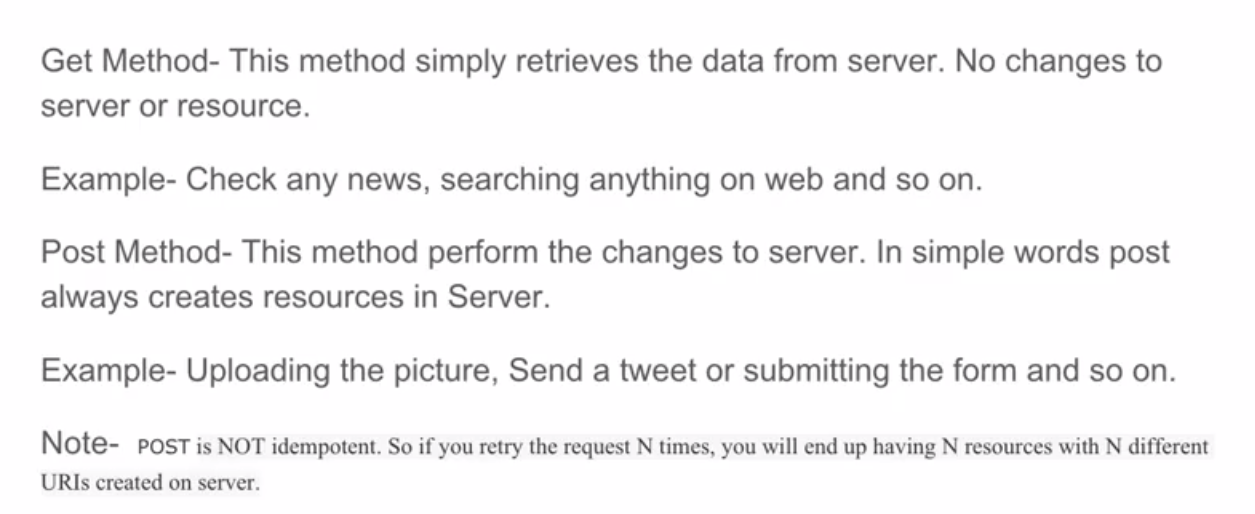
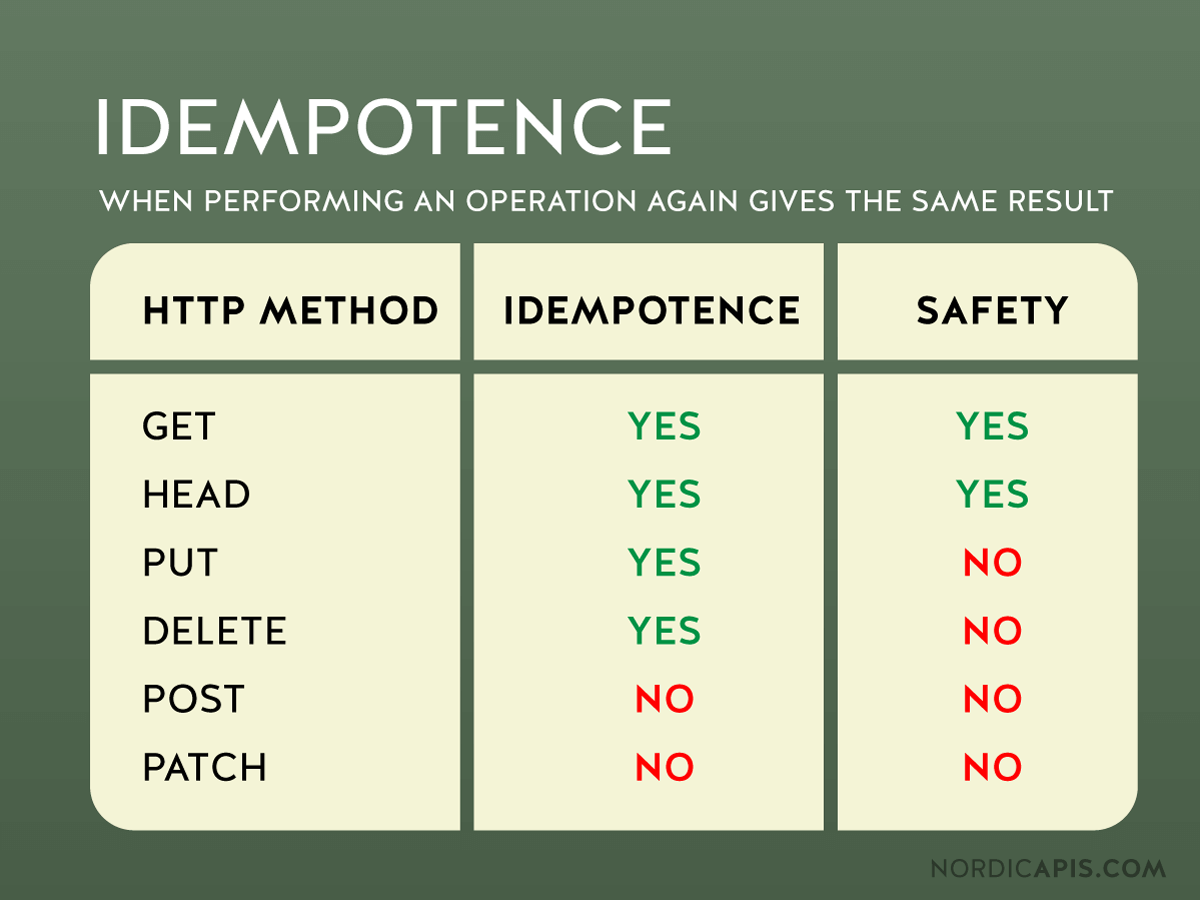
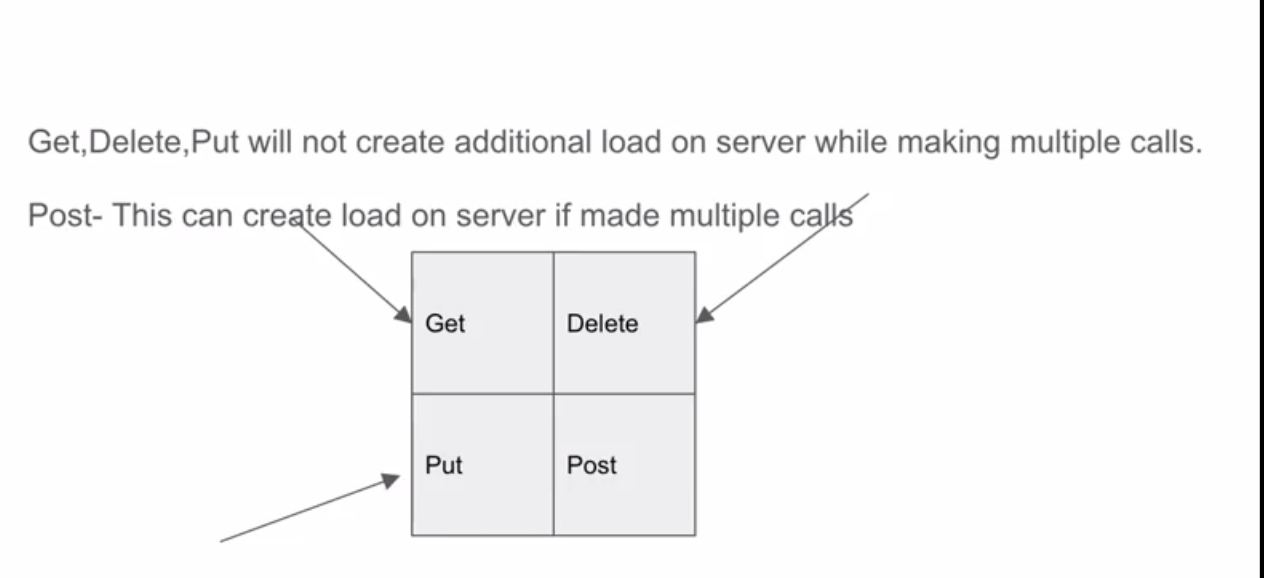
To find API uniquely we need URI

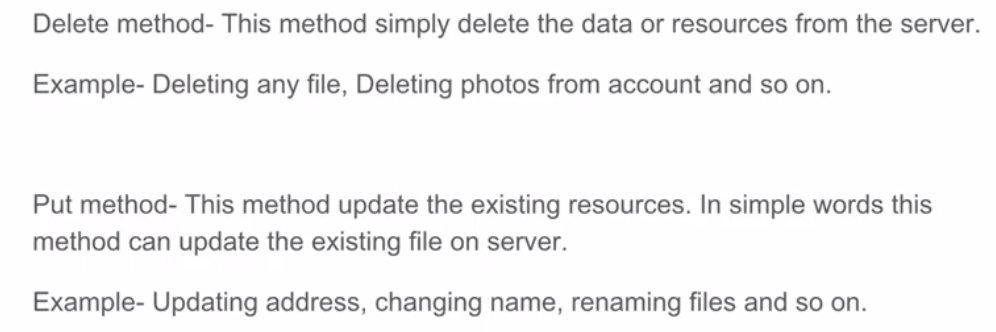
REST-Stateless (never store state of the server), Fast and consume any kind of data (text, xml, json, string), multiple way of authentication, it is an architecture and response depends on the what data client consumed.

SOAP- support only xml (xml is very heavy as compared to JSON) type of data, inbuilt security.



RestAssured : Java library and developed on top of HTTP client, so it support HTTP client method as well as their own and it is very light weight. Code writing will be easy as so many wrapper are created to deal with APIs.





**Notes:**

* Front end development language: JS, CSS, HTML, Angular, React
* Backend development language: Java, Python, C#, PHP, Ruby
* Documentation is very imp for API : It contains all the information related to Classes , Functions, Input/Output, End Point URL, type of HTTP method (CRUD), resource (JSON Payload, Headers, Authentication- Everything is a resource), parameters etc. and that is provided by Developer.
* Documents type: Swagger, Confluence pages etc.
* According to environments (QA, Dev, Stage, UAT, Prod) URL will also change
* 1st backend will be created - server configuration and API - web will come into picture.
* Once backend is developed immediately testing can start
* Backend is fine than why UI testing?
  + At the end customer will work on UI and if GUI is not good, customer will not satisfy, so UI testing is equally important to test.
* Authorization/APP ID/ Authentication Token is very important in API's for the Security reason otherwise anyone can hack the data.
  + 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**

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.

**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(create extra load on the server).

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 keep 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.

Means POST create extra Load on the Server.

Testing an API with POST requests

Here are some tips for testing POST requests:

Create a resource (201 status code)with a POST

Next, make a GET request for that resource, and ensure the data was saved correctly(200 status code).

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.

**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(input/output means all the API’s should work as per business logic), reliability(should provide proper results as per request), performance(respond properly in heavy load as well), security(API’s should be secured with any authentication else anyone can hack).

**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 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 (positive 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.
* 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.
* 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.
* 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.

**Basic tips that you need to know for API testing:**

* 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 (API chaining is also very important to test).

* Specify the API output status: what we need to verify like Status code, Status Line, body, Response time or memory etc.
* 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.
* 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.
* 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.
* 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.
* 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
* 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.
* 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.

**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

**What is API?**

An API (Application Programming Interface) is a software that enables two applications to communicate with each other.

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,JIRA 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
* Time Effective
* Language-Independent
* Easy Integration with GUI

**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 tools could be used for API testing?**

Postman

Katalon Studio

SoapUI

HTTP Client

Jercy client

Rest client

JMeter

Rest-Assured

**What is differences between API Testing and UI Testing?**

API : test the business logic

UI : look and feel, navigations, actions

**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 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
* Confluence pages
* 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.

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 (body/json payload)

**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.

**Differentiate API testing and Unit Testing.**

Unit testing is usually performed by developer 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.

**Explain REST?**

Ans. REST stands for Representational State Transfer. REST is an architectural style of developing web services which take advantage of the ubiquity of HTTP protocol and leverages HTTP method to define actions. It revolves around resource where every component is a resource which can be accessed by a common interface using HTTP standard methods.

In REST architecture, a REST Server provides access to resources and REST client accesses and presents those resources. Here each resource is identified by URIs or global IDs. REST uses different ways to represent a resource like text, JSON, and XML.XML and JSON are the most popular representations of resources these days.

**What Is the Most Popular Way to Represent A Resource In REST?**

Ans. REST uses different representations to define a resource like text, JSON, and XML.

JSON is the most popular representations of resources.

**Which Protocol Is Used B­­­y RESTful Web Services?**

Ans. RESTful web services make use of HTTP protocol as a medium of communication between client and server.

**What Is Messaging in RESTful Web Services?**

Ans. RESTful web services make use of HTTP protocol as a medium of communication between client and server. The client sends a message in the form of an HTTP Request.

In response, the server transmits the HTTP Response. This technique is called Messaging. These messages contain message data and metadata i.e. information about the message itself.

**The Core Components of An HTTP Request?**

Ans. Each HTTP request includes five key elements.

1. The Verb which indicates HTTP methods such as GET, PUT, POST, DELETE.

2. URI stands for Uniform Resource Identifier (URI). It is the identifier for the resource on the server.

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

4. Request Header carries metadata (as key-value pairs) for the HTTP Request message. Metadata could be a client (or browser) type, the format that client supports, message body format, and cache settings.

5. Request Body indicates the message content or resource representation.

**What is Rest Assured?**

In order to test REST APIs, I found REST Assured library so useful. It is developed by JayWay Company and it is a really powerful catalyzer 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.

**How to declare the API details in Rest Assured Test?**

Using Given(), When(), Then()

**Name the Most Commonly Used HTTP Methods Supported By REST?**

Ans. There are a few HTTP methods in REST which are more popular.

* GET -It requests a resource at the request-URL. It should not contain a request body as it will get discarded. Maybe it can be cached locally or on the server.
* POST – It submits information to the service for processing; it should typically return the modified or new resource.
* PUT – At the request URL it updates the resource.
* DELETE – It removes the resource at the request-URL.
* OPTIONS -It indicates the supported techniques.
* HEAD – It returns meta information about the request URL.

**Is There Any Difference Between PUT and POST Operations? Explain It.**

Ans. PUT and POST operation are almost same. The only difference between the two is in the terms of the result generated by them.

* PUT operation is idempotent (can’t change) while POST operation can give a different result.
* POST create extra load on the server but PUT does not.

**What Is URI? Explain Its Purpose in REST Based Web Services. What Is Its Format?**

Ans. URI stands for Uniform Resource Identifier. URI is the identifier for the resource in REST architecture.

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

<protocol>://<service-name>/<ResourceType>/<ResourceID>

**How to compare the response values with Rest Assured Assertion?**

given().

parameters("firstName", "John", "lastName", "Doe").

when().

post("/greetXML").

then().

body("greeting.firstName", equalTo("John")).

body("greeting.lastName", equalTo("Doe"));

**How to Insert cookies in Testing the API using Rest Assured?**

given().cookie("username", "John").when().get("/cookie").then().body(equalTo("username"));

**How to Insert headers in Testing the API using Rest Assured?**

given().header("MyHeader", "Something").

**How to Validate Response Headers with Rest Assured?**

get("/x").then().assertThat().header("headerName", "headerValue").

**How to handle Basic Authentication with Rest Assured?**

given().auth().preemptive().basic("username", "password").when().get("/secured/hello").then().statusCode(200);

**What Do You Understand By Payload In Restful Web Service?**

Ans. Request body of every HTTP message includes request data called as Payload. This part of the message is of interest to the recipient.

We can say that we send the payload in POST method but not in <GET> and <DELTE> methods

**RESTful APIs support two different types of parameters:**

Query parameters: These are appended at the end of a RESTful API endpoint and can be identified by the question mark in front of them. For example, in the endpoint http://md5.jsontest.com/?text=test, "text" is a query parameter (with value "test").

Path parameters: These are part of the RESTful API endpoint. For example, in the endpoint we used earlier: http://ergast.com/api/f1/2017/circuits.json, "2017" is a path parameter value.

**REST Assured also offers a number of other useful features that let you create even more powerful tests, such as:**

The ability to (de-)serialize Plain Old Java Objects (POJOs). This allows you to serialize the properties and values associated with a Java object instance directly into a JSON or an XML document, which can then be sent to a RESTful API using the POST method. This also works the other way around—a JSON or an XML response returned by an API can be deserialized into a POJO instance by REST Assured as well.

Logging requests and responses. This can be especially useful when you need to inspect API responses to create the appropriate checks, or when you want to make sure that the request you're sending to an API is correct. You can choose to log everything (parameters, headers, and body), only headers, only parameters, and much more.

REST Assured comes with a Spring Mock MVC module, allowing you to write tests for Spring controllers using the REST Assured syntax.

**Pros:**

More flexibility with writing tests, as it is based on Java.

Is open source and free to use for everyone.

Can be used with any open source reporting tool.

Easy to integrate with JUnit and TestNG frameworks, for greater added functionality.

Supports Given/When/Then notation, making the code easier to read.

Supports multiple authentication methods, including OAuth1, OAuth2 and Form authentication.

Support for Proxy configuration.

**Cons:**

Requires knowledge of the Java programming language

The user must write code for every test case

Performance testing is not integrated in this tool

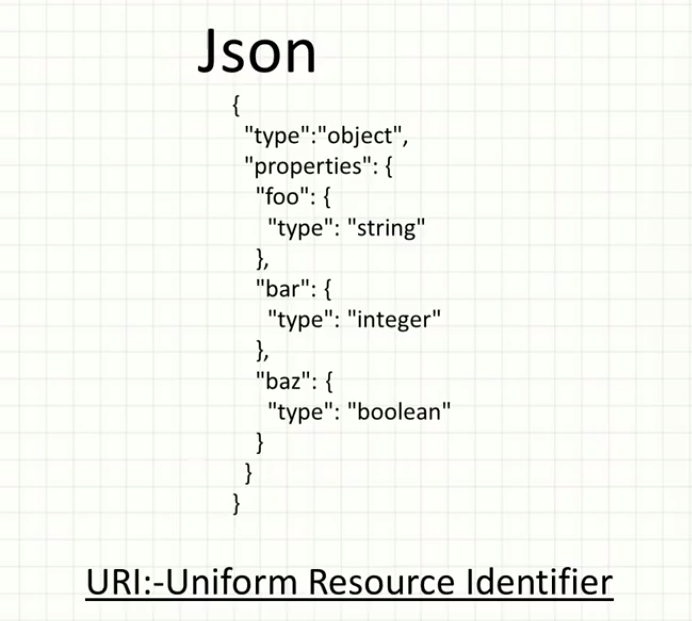
**Json object and Json array**

Arrays in JSON are used to organize a collection of related items (Which could be JSON objects).

For example: [{"name":"item 1"},{"name": "item2} ]

On the other hand, you would use JSONObject when dealing with JSON that begins with curly braces. A JSON object is typically used to contain key/value pairs related to one item. For example: {"name": "item1", "description":"a JSON object"}

Of course, JSON arrays and objects may be nested inside one another. One common example of this is an API which returns a JSON object containing some metadata alongside an array of the items matching your query:



**handshake** - interaction b/w two system

when any company exposing their API'S so that’s their responsibility to maintain the proper API’s documents

Authentication is very important (Auth key/Token, consumer key, username password, security questions)- otherwise anyone can hack and change the data's

when you are entering user username and password in any site, one Auth key/App id, Auth token in generated in backend and it is valid only for sometimes and then session expired(ex.- In financial transaction)

you might pass Authentication with URL or with header

API is all about data (API docs)- and data are provided by the companies.

ex- UBER uses Google maps (google will provides data/docs)