**Microservices**

**What is microservice:**

* Microservices is a method of developing software system that tries to focus on building single-function modules with well-defined interfaces and operations i.e. you can build the application in the form of small autonomous services.
* Unlike microservices, a monolith application is built as a single, autonomous unit i.e. an application and all it’s feature/functionalities are stored at one place. This make changes to the application slow as it affects the entire system.  A modification made to a small section of code might require building and deploying an entirely new version of software and perform regression testing of complete application.  Scaling specific functions of an application, also means you have to scale the entire application.
* Microservices solve these challenges of monolithic systems.
* In the simplest form, they help build an application as a suite of small services, each running in its own process and are independently deployable.
* These services may be written in different programming languages and may use different data storage techniques.
* Microservices are often connects via APIs, and can leverage many of the same tools and solutions that have grown in the RESTful and web service ecosystem.
* In short, we can say, **Microservices are always integrated using REST over HTTP (RESTful APIs)**
* In integrations, sometime lightweight messaging protocols are used such as Protobuf or Thrift
* [Testing these APIs](https://smartbear.com/solutions/api-testing/) can help validate the flow of data and information throughout your microservice deployment.
* Just as there is no formal definition of the term microservices, there’s no standard model that you’ll see represented in every system based on this architectural style
* Microservices may or may not provide some form of user interface
* <https://www.youtube.com/watch?v=j1gU2oGFayY>
* Mini applications or services are technically called Microservices.
* Microservices are way of breaking of your applications or services down to into Standalone and independent mini applications or services which can be deployed and execute/run on different hardware or servers. All these mini applications/services talk to each other over REST APIs altogether to provide the functionality of your Product/Application.

**Benefits of Microservices:**

* Deployment Flexibility
* Technology Flexibility
* Can scaled separately

**Disadvantages:**

* Deployment/architecture complexity
* Service discovery

**Microservices Testing Methods/Strategies:**

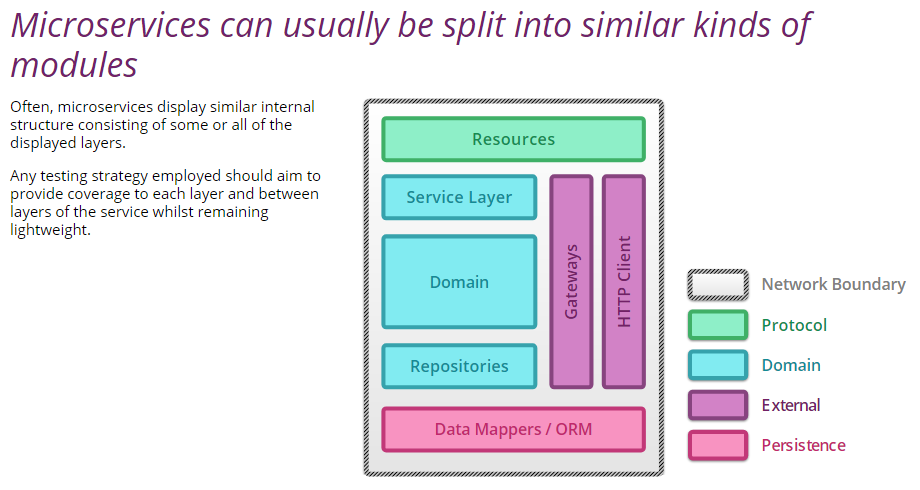
<https://martinfowler.com/articles/microservice-testing/>

<https://www.simform.com/microservice-testing-strategies/>

<https://www.infosys.com/de/documents/microservices-testing-strategies.pdf?soc=tw10205>

<https://www.freecodecamp.org/news/these-are-the-most-effective-microservice-testing-strategies-according-to-the-experts-6fb584f2edde/>

<https://docs.pact.io/>



**Resources:** Resources act as mappers between the application protocol (exposed by service) and messages to objects representing the domain. Typically, they are thin, with responsibility for sanity checking the request and providing a protocol specific response according to the outcome of the business transaction.

**Domain**: Almost all of the service logic resides in a Domain model; represents business domain

**Service layer**: Services coordinates across multiple domain activities

**Repositories**: Repositories act on collection of domain entities and are often persistence backed

* **Unit Testing**:
  + Unit testing focus on smallest part of testable component to make sure if it’s working as expected.
  + Two categories:
    - **Sociable Unit Testing**: This method tests the behavior of modules by observing changes in their state
    - **Solitary Unit Testing**: This method focus on the interactions and collaborations between an Object and its dependencies, which are replaced by test doubles
  + Drawback:
    - For user-facing features they are difficult to mock and may skip important part of system
    - Does not cover interaction between different components or services
* **Integration Testing:**
  + It tests communication paths and interactions between components i.e. tests the interaction between microservice and external services like another microservice or datastore
  + Drawback:
    - Error prone and costly, in terms of man-hours
    - Testing of each individual interaction between components become verify complicated, complex
    - Because of complications, Testing coverage can be less
* **End-To-End Testing**:
  + It involves testing of every moving part of the microservice, ensuring that it can achieve the goal you built it for.
  + Drawback:
    - Can be difficult task
    - Time consuming

**Challenges of Testing Microservices:**

* **Availability**: Since different teams manage their own microservices, it’s difficult to find the time when all microservices are available to test
* **Fragmented and Holistic testing**: Microservices are built to work alone and together with other loosely coupled services. We need to test every component separately, as well as everything together.
* **Knowledge gap**: particularly with Integration testing, whoever conducts the tests, will need to have strong understanding of each service in order to write test cases effectively
* Have to manage multiple repositories and branches, each with their own database schemas
* While working with microservices, we have to keep in mind about API stability and versioning
* We have to create solid set of integration tests for microservice APIs.

**Microservice Testing Strategies for startups:**

* **The Documentation-First Strategy**:
  + Before anybody writes any API changes or either a new API or changes to an API, update the documentation first,
  + have that change reviewed to make sure that it conforms with API conventions and standards which are all documented, and
  + make sure that there’s no breaking change introduced here.
  + Further, you can perform API Contract testing; which involves writing and running tests that ensure the explicit and implicit contract of a microservice works as it should.
* **The full stack in-a-box strategy**:
  + In The full stack in-a-box strategy, we replicate a cloud environment locally and testing everything all in one vagrant instance(“$ vagrant up”).
  + But it’s extremely tricky and
  + difficult to scale up
* The AWS testing strategy:
  + It involves spinning up an Amazon Web Services (AWS) infrastructure for each engineer to deploy and run tests on
  + More scalable approach
  + Everyone has their own AWS account
  + You can push the code that’s on your laptop up into AWS in about ten minutes and just run it in like a real system

**API:**

* Application Programming Interface (API) is the connecting tissue between different Systems or application layers. (in another words, an **Interface** where you tell the **Program** to run in an **Application**)
* It’s being used to connect and transfer data and logic across systems and applications
* It is a set of procedures and functions which allow the consumer to use the underlying microservices/services of an application.
* Application often have three layers:
  + A data layer
  + A service (API) layer
  + A presentation (UI) layer
* The API layer contains the business logic of an application - the rules of how users can interact with services, data, or functions of the app
* API is just a program; we just have to use it on out program e.g. you can use Google search API or Amazon shopping or travel booking APIs in your program
* APIs are platform independent; that can be run on any type of devices or in any program (language barrier is not there)
* API testing generally consists of make requests to a single or multiple API endpoints and validate the responses - whether for performance, security, functional correctness, or just a status check
* API testing puts much more emphasis on the testing of business logic, data responses and security, and performance bottlenecks.
* Major component of API: Request, Response, a Program
* E.g. 1) Search a term ‘testing’ in Google Search page on your machine. Google search page sends the request with search term to Google server, Program gets executed on Google server, and Google server sends the response back to your computer. Here Program location is mentioned into the URL. E.g. [www.google.com/search?q=testing](http://www.google.com/search?q=testing)

Meaning of the url is that, Goto Google server over the internet, goto ‘search’ folder and execute the program / query(q=) with the parameter(?) ‘testing’.

Here, we have used Google Search page to send the request.

Sometimes, API is used directly in our program behind the scene and send the request to server for the information.

**Benefits of API Testing:**

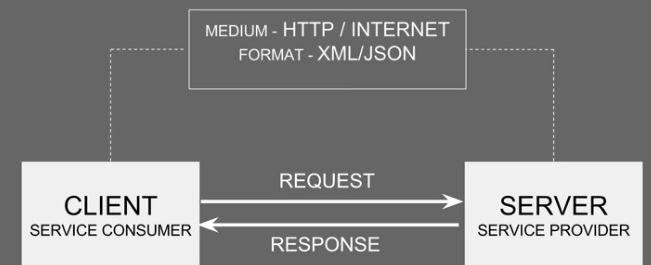
* Earlier Testing: With API testing, once the logic is designed, tests can be built to validate the correctness in responses and data. We don't have to wait for various teams to finish their work or for full applications to be built - test cases are isolated and ready to build immediately
* Easier Test Maintenance: UIs are constantly changing and moving around. API changes are much more controlled and infrequent - often times API definitions files like OpenAPI Spec can help make refactoring tests only a seconds of work.
* Faster time to resolution: When API tests fail, we know exactly where our system broke and where the defect can be found. This helps reduce time triaging bugs
* Speed and coverage of testing: it’s faster than UI/functional testing

**Types of API Testing:**

* Unit Testing: Testing a single endpoint, with a single request; looking for a single response or set of responses. Many times, this type of testing can be done manually via the command line and something like a cURL command or with lightweight tools like SoapUI.
* Integration Testing: It is most often used form of API testing; as APIs are at the center of most integrations between internal and third party services.
* End-To-End Testing: End-to-End testing can help us validate the flow of data and information between a few different API connections
* Performance testing: Previously, load testing was kept in the hands of the few and was difficult to execute in a CI/CD environment. LoadUI Pro is a performance testing tool for RESTful, SOAP, and other web services that enables nearly any team member to embed performance tests into their CI/CD pipeline.

**Web Services:**

* Web (Internet) Services (API): API that uses the internet
* Not all APIs use Web i.e. all Web Services are APIs; But Not all APIs are web services i.e. not all APIs uses Web (internet)
* Web services enables communication (exchange data/info) between two applications over internet (web)
* It provides standard protocol/format for communication (it’s platform independent communication)
* Web Services Use:
  + **XML** or **JSON** **format** of the data over internet
  + **REST, SOAP, or XML/RPC Protocols** to transfer the data over internet
  + **HTTP** used to send a request and receive a response
* In another words, Web services uses **HTTP as medium** and **XML/JSON format** to communicate data between Client and server
* And Web services are implemented in major two types i.e.
  + **SOAP** (Simple Object Access Protocol)
    - It uses medium: HTTP (POST method)
    - and Format: XML
  + **REST** (REpresentational State Transfer)
    - It uses medium: HTTP (GET, POST, PUT, DELETE,… methods)
    - and Format: XML/JSON/TEXT…



* To make use of Web services at Client side (Consumer), Consumer needs to know:
  + What are the services available
  + What are the request and response parameters
  + How to call the web service
* For this, every service provider publishes description (an interface, *not UI*) of it’s web services where all the attributes, features and functionalities of web services are described.
* And This is an XML based interface and is called “**WSDL**” (Web Services Description Language)
* In another words, **“WSDL” is an XML based interface that is used to describe the attributes, features and functionalities of the web services**
* WSDL is the interface of the web service; service provider creates WSDL for it’s web service and consumer can get that WSDL and use it to get web service request and response
* Consumer can get this WSDL from service provider by two ways:
  + If consumer and service provider knows each other then, service provider can directly handover this **WSDL** to consumer for it’s use
  + Another way is that, service provider publishes his web service (thru WSDL) on an online directory from where consumers can query and search the web services. This Online registry/directory is call **UDDI** (Universal Description, Discovery & Integration)
  + UDDI is an XML based standards for publishing and finding web services

**HTTP (Hyper Text Transfer Protocol):**

* ‘www.google.com’ is a simple set of words; HTTP makes its special
* Hyper Text, meaning sends this url to somewhere else over the internet;
* Components of HTTP:
  + **Start Line** (it’s mandatory part)
  + **Headers**
  + **Blank Line**
  + **Body**
* See below table what can contains in to HTTP parts in Requests and Response, as an example

|  |  |  |
| --- | --- | --- |
| **HTTP** | **Requests** | **Response** |
| **Start Line** | HTTP Version#, Method Info (like GET, POST, PUT, DELETE), folder info and parameter in url e.g. /search?q=testing | HTTP Vesion#, Status Code (e.g. 200 (successful), 400 (failure in sending/receiving path), 500 (Failure at server side) |
| **Headers** | Host (url domain i.e. www.abc.com),  Token | Cookies,  File size, what is sending back i.e. HTML type |
| **Blank Line** | It’s nothing, it’s just separator |  |
| **Body** | Information which you are sending | HTML web page with requested info |

**Details of Start Line:**

|  |  |  |
| --- | --- | --- |
|  | **Request** | **Response** |
| What’s Name? | Start Line, Request Line | Start Line, Response Line, Status Line |
| Contains HTTP version? | Yes, HTTP/1.1 | Yes, HTTP/1.1 |
| Contains Methods? | Yes (GET, POST, PUT, DELETE, etc.) | No |
| Contains API program folder location? | Yes (e.g. /Search) | No |
| Contains Parameters? | Yes (e.g. ?q=testing | No |
| Contains Status code? | No | Yes (e.g. 200 = success, 400 = client error, 500 = server error) |
| Format | Method(space)API Program folder Location+Parameters(space)HTTP version  Note: API Program folder location and Parameters are optional | HTTP version(space)Status code |
| Example | GET /search?q=testing HTTP/1.1 | HTTP/1.1 200 OK |

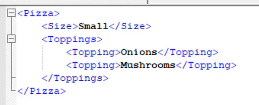
What is **idempotence**?

Can do as many times as you want and result stays same; in other words, ‘safe’ to repeat.

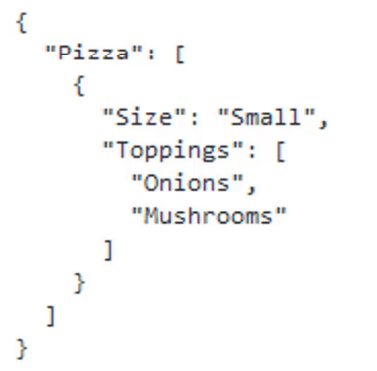
This is related to Start line methods.

|  |  |
| --- | --- |
| **Methods** | **Idempotent? (safe to repeat?)** |
| GET (read) | Yes |
| POST (create) | No |
| PUT (modify) | Yes |
| DELETE (delete) | Yes |

**XML (eXtensible Markup Launguage)**

* One of the field in HTTP Header line is ‘Content-Type’ which tells you what type of data is there in Request or Response.
* There can be many; one of the majorly used as “Application/XML”
* XML holds the data in request/response
* Actual XML content resides into HTTP Body
* XML uses tags in <>; similar to HTML
* E.g. <button>Click Me!</button>
* Here, <button> tag has meaning in HTML. It describes that it’s a button having name “Click Me”
* But, in XML, it just a information type. It can be anything in XML. In HTML, it cannot be anything.
* HTML and XML standards are set by W3C (World Wide Web Consortium)
* XML example
* 

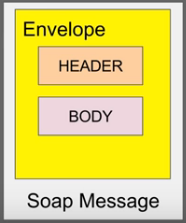
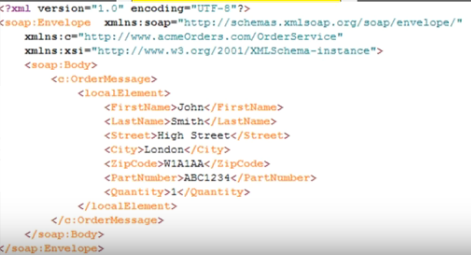
**JSON (JavaScript Object Notion):**

* One of the field in HTTP Header line is ‘Content-Type’ which tells you what type of data is there in Request or Response.
* There can be many; one of the majorly used as “Application/json”
* JSON holds the data in request/response
* Actual JSON content resides into HTTP Body
* JSON uses “Key” : “Value” pair
* JSON example
* 
* XML is more powerful than JSON; able to transfer xml data to another xml, more security, more capabilities
* JSON is more simple and lightweight than XML

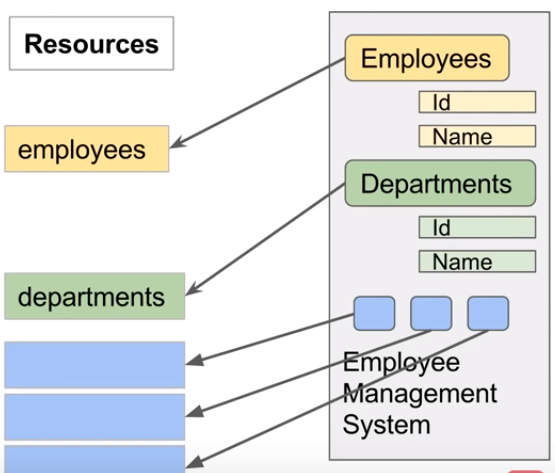
**SOAP and REST:**

* These are the ways to form HTTP request and response

**SOAP (Simple Object Access Protocol):**

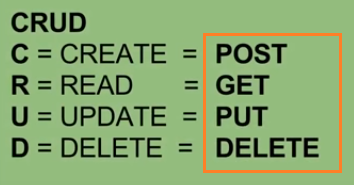
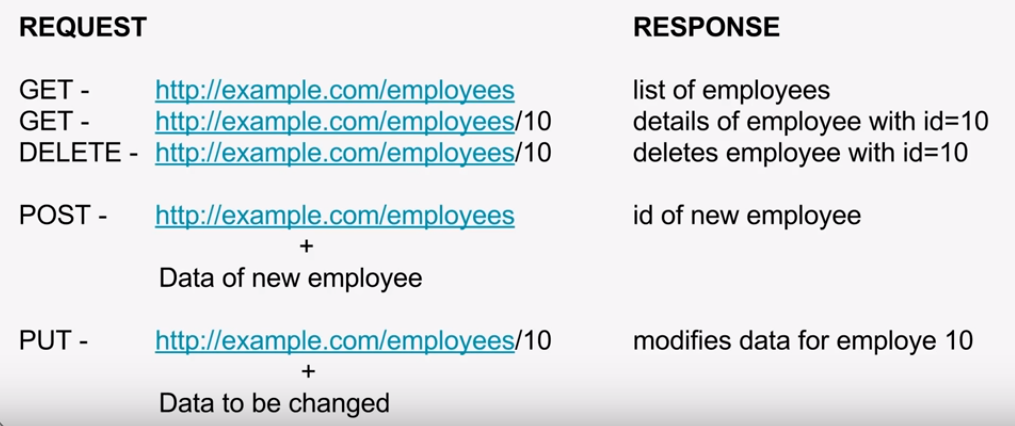
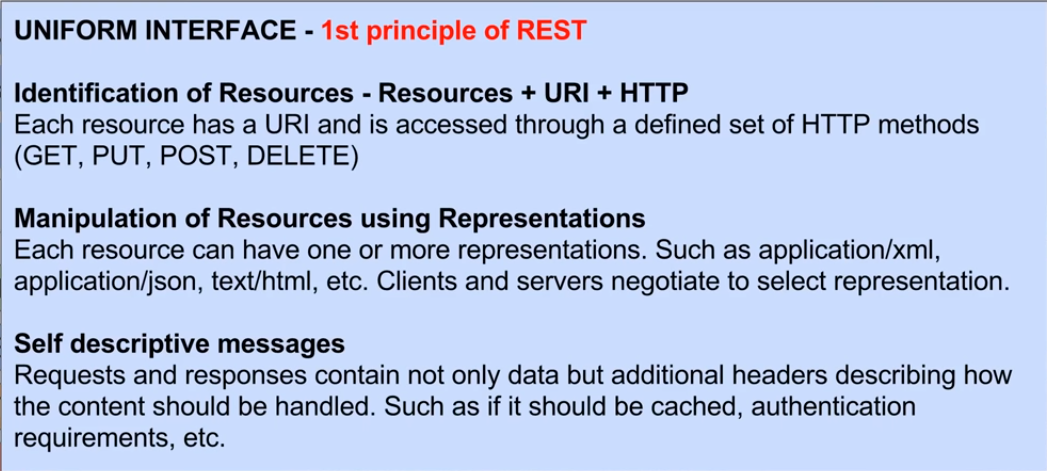
* It’s a way to access Web Service (API over internet) by following some rules (protocols) which are used to form HTTP request/response
* Web services uses SOAP
* Every SOAP API is a web service
* WSDL describes the web services as a SOAP Web Service
* XML content type is always used in HTTP body of SOAP; In Header Line, Content Type is always ‘text/xml’
* SOAP Web Services are the web services which comply the SOAP standards/rules/specifications
* These standards/rules/specifications are written by a internationally central body W3C
* Basic Set of specifications include:
  + SOAP
  + WSDL
  + UDDI
* SOAP is a protocol which defines how the applications talk to each other.
* To talk to each other these application have to follow common format
* And that Format is “XML”
* And XML has to follow “SOAP Message” Standards in SOAP web services
* And SOAP Message Structure consists of:
  + Envelope
    - It’s root element of a soap message. This is the basic unit of the XML document which contains other units like Header & Body
  + Header
    - Header element provides information about the message itself.
    - Header might include authentication, complex types, routing info etc
  + Body
    - Body contains actual data of request which is being sent to server
  + 
  + Example of SOAP message:
  + 

**REST (REpresentational State Transfer)**

* A Web service that communicates information between two applications using REST architecture/principles is called **RESTful Web Services**
* **What is REST?**
  + It is an architectural style
  + Like SOAP, it’s NOT a protocol; it hasn’t any strict specifications; there is no central body to control this
  + It’s just a design principle
  + We can use these principles to design any service for communication of two applications
  + When we apply these principles while designing web service (for client-server communication), we get RESTful Web Service.
* What are the **REST architecture/principles/rules** which make the web service as RESTful web service?
  + **Uniform Interface**
    - **Resource**
      * Everything is a Resource; like every module, information, data, image, entity, document or etc… can be defined as resource
      * There is no specific rule as such
      * Like below screenshot, Employees, departments module can be defined as resource
      * 
    - **URI (Uniform Resource Identifier)**
      * You can access any Resource/data using URI (by URI)
      * If you want to access the employees from the above Employee Management System, you can use URI-

<http://example.com/employees/10> (get employee details which has ID 10)

<https://example.com/department/123/employess> (get all employees from department ID 123)

* + - **HTTP:**
      * How we can use HTTP in REST; we can use HTTP methods in REST
      * 
      * Using HTTP methods along with URI, we can access/modify any resource or resource info
      * 
    - 
    - In another words, Resource represents API/Collection which can be accessed from the server
    - Let’s there are few URLs like Google.com/maps, Google.com/docs, google.com/search, google.com/images

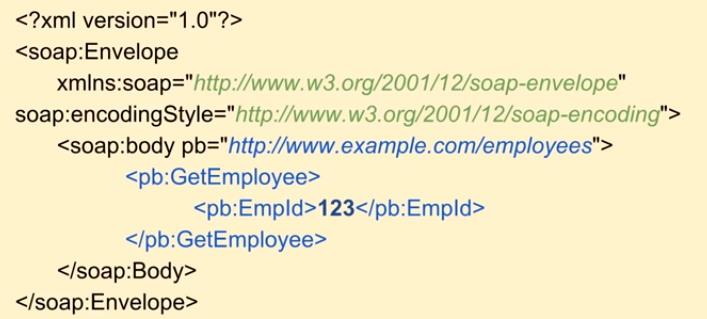
Here, google.com is a base URL/URI and ‘maps’, ‘docs’ are the APIs or resources.

So, request goes to google.com server first and then based on resource, decide which API has to use.

* + - When we send a request to server Base URL+Resource is mandatory
    - Along with that sometime we have to send different Parameters based on type of requests. What are these Parameters?
      * **Path Parameters**
        + Path Parameters are variable parts of a URL path. They are typically used to point to a specific resource within a collection that specific resource is identified by ID
        + E.g. <https://www.google.com/images/12341>
        + Here 12341 is ID of the image which you are requesting for
        + Here, we drill down to sub-resource to find out specific item
      * **Query Parameters**
        + Query Parameter is used to sort/filter the resource
        + Query parameter is identified with “?”
        + E.g. <https://www.amazon.com/orders?sort_by=2/20/2020>
        + Here, unlike path parameter, we don’t goto sub-resource; rather we apply filter on resource or collection itself
    - So, End Point Request URL can be constructed as:

Base URL/resource/Parameters (Path/Query)

* + - E.g. if you want to get employee id 123 details from above system –
    - SOAP request will be



* + - For same, REST request will be simple URL



* + **Stateless**:
    - All client-server communications are stateless; meaning server does not maintain any state of the system
    - i.e. each request from Client to Server much be complete, independent and contains all the data that is required for a request. Server should not maintain any state/status of previous or current request or session
    - If required, client has to maintain the session or store info
    - Because of this, it improves web services performance
  + **Cacheable**
    - Happens at Client side
    - When server sends response to Client, that response may contained some entities/info which are labeled as Cacheable or non-Cacheable; meaning server tells client which data needs to be cached at client side
    - Into the response header, ‘Cache-control’ field is available which tells how long the response need to be stored in Client’s cache
  + **Layered System**
    - Multiple layers can exists between Client and Server
    - These layers are HTTP intermediaries
    - Can be used for message translation / improving performance with caching etc.
    - It includes:
      * **Proxies**
        + A client may send a request to Proxy server instead of main server
        + Proxy server evaluates the request to simplify or control its complexity, etc
      * **Gateways**
        + Gateways may be used for managing traffic on the network, protocol translation etc.
  + **Code On Demand** (optional)
    - Ability to download and execute code at client side
    - For an example, client request a resource and server returns a resource with some java-script which runs on client side
* Every Resource can be represented in multiple ways; so a resource may have a representation in XML or JSON format and whenever there is request from client, representation of a current state of the resource is being transferred between server and client
* Representation = description/presentation of current state of resource
* And that can be in many formats like XML, JSON, Image, HTML, etc
* In another words, REST is nothing but Set Of Rules of communicating data between client and server; i.e. the rules/principles, how the data/information (STATE) of a resource is transferred using Representations.

**End Point / Base URI:**

* Address where API/Web service is hosted on server
* In testing, we hit to the end point to check response from API/Web service

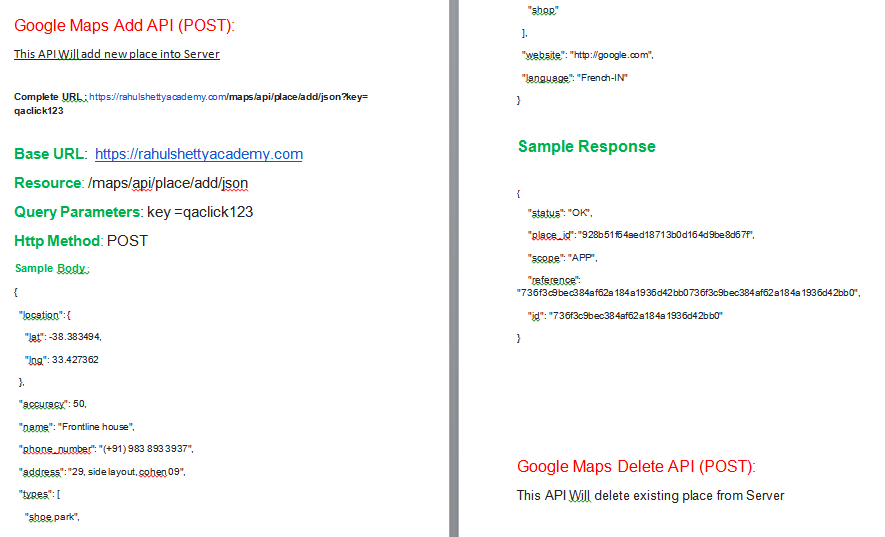
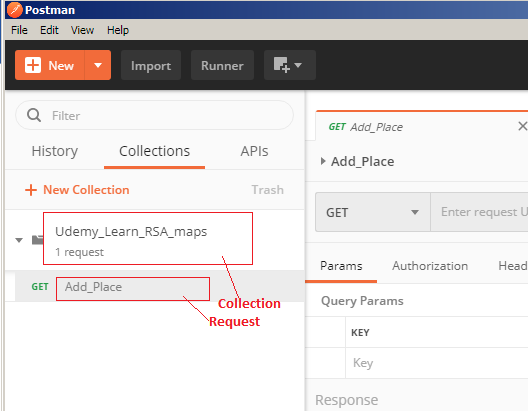
**Contracts:**

* API contracts are nothing but the documentation which includes details of the API e.g. API behavior, Base URL/end points, Resources, Parameters, HTTP methods, expected response etc.
* To Start with API testing, we need to know about Contract first.
* Generally developer provides API contracts to tester so that he can start testing

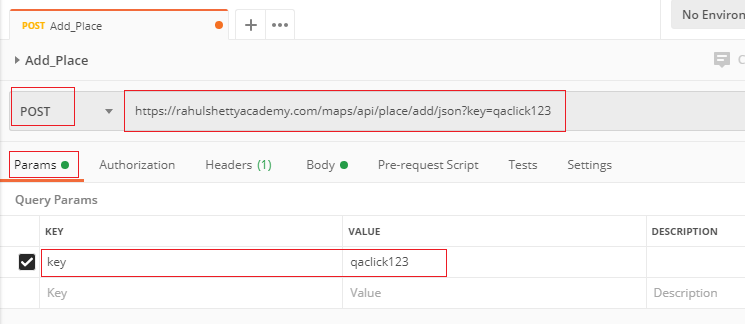
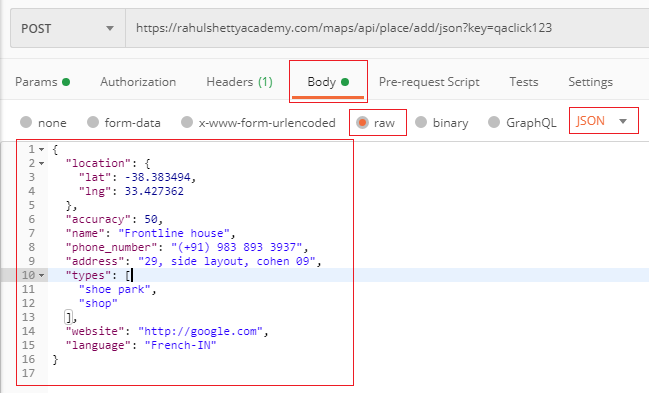
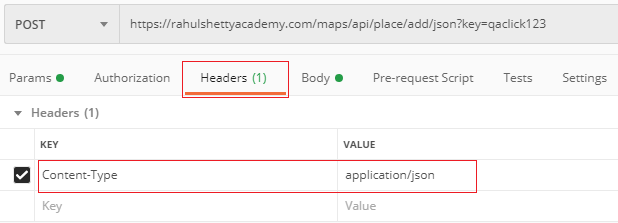
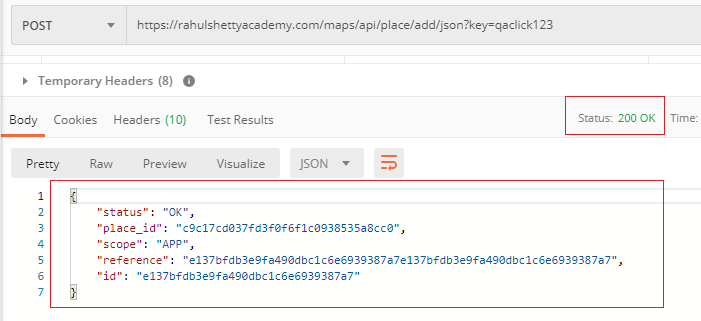
**REST Assured:**

* REST Assured is a JAVA DSL (library) which helps to simplify testing of REST based web services.
* It supports POST, GET, PUT, DELETE, PATCH and HEAD requests
* It can be used to validate and verify response of a request

**REST API Manual Testing (POSTMAN):**

* You should get a contract document from dev team before start testing as below:
* 
* Create new Collection (it’s like a project) in POSTMAN application
* And under it, create new Request
* 
* Into the ‘Enter request section’, Enter your Base URL and the Resource
* Into the PARAMs, enter Query Parameters

POST Method (ADD):

* Select HTTP Method (here, POST)
* Enter sample Body from contract into ‘Body’ section and Select ‘JSON’ format (Raw)
* It’s Sample body, you can change the value inside whatever you want
* Into Header, enter ‘Content-Type’ and it’s value as ‘application/Json’. It’s Format of the body
* Once, click on Send button, request will be sent to the server and you should get a response
* 
* 
* 
* You will get the response as below with status 200 (OK):
* 

GET Method (Retrieve):

* Let’s consider this is a contract for GET request:

*Google Maps get Place API (GET):*

***This API Will get existing place details from Server***

***Complete URL :*** *http://rahulshettyacademy.com/maps/api/place/get/json?place\_id=xxxx&key=qaclick123*

***Base URL****:*  [*https://rahulshettyacademy.com*](https://rahulshettyacademy.com/)

***Resource****: /maps/api/place/get/json*

***Query Parameters****: key, place\_id //( place\_id value comes from Add place response)*

***Http request****: GET*

*Note: Key value is hardcoded and it is always qaclick123*

***Sample Response for the Provided Place\_Id***

*{*

*"location":{*

*"lat" : -38.383494,*

*"lng" : 33.427362*

*},*

*"accuracy":50,*

*"name":"Frontline house",*

*"phone\_number":"(+91) 983 893 3937",*

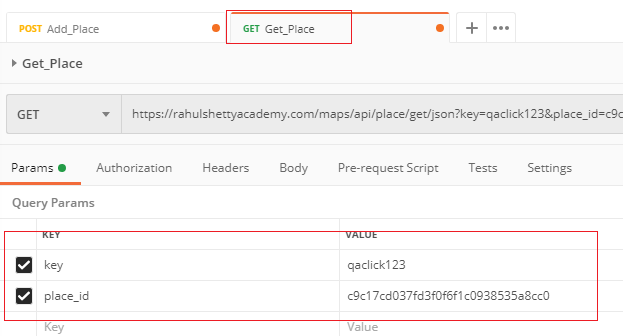
*"address" : "29, side layout, cohen 09",*

*"types": ["shoe park","shop"],*

*"website" : "http://google.com",*

*"language" : "French-IN"*

*}*

* Note that, when we send GET request, we don’t need to send the body; we send everything as part of URL.
* After you send this request, you should get response with expected info and status code as 200.
* 

**DELETE method:**

* It’s similar to POST. As per the contract, Enter baseURL/Resource, Add Body Format, Enter Content-Type in Header, add Query Parameter
* Select Method type as DELETE or POST (here, POST method also works for delete since into the resource, it (should) redirects to Delete API)

**PUT method (Update):**

* It’s similar to POST. As per the contract, Enter baseURL/Resource, Add Body Format, Enter Content-Type in Header, add Query Parameter
* Select Method type as PUT or POST (here, POST method also works for modify since into the resource, it (should) redirects to PUT API)

**REST API Automation:**

Setup:

* Install JAVA
* Setup Environment variables for JAVA
* Install Eclipse (Java editor)
* Create new Java Project in Eclipse
* Download all JARs from Rest-Assured.io > Docs > Downloads
  + Rest-assured…
  + Json-path…
  + Xml-path…
  + Json-schema-validator…
  + Spring-moc-mvc…
  + Scala-support…
  + Kotlin-extensions…
  + Spring-web-test-client…
* Import all these JARs into Java Project (using Add External JAR libraries)
* Goto eclipse menu >Windows >Preferences > Java > Editor > Typing > “Escape Text when pasting into string literal”.. select this option.

This setting will be useful when you paste a JSON file body into string double quote( “”).

Start with Automation:

* Let’s consider below is API contract for POST method:

This API Will add new place into Server

**Complete URL :** [https://rahulshettyacademy.com](https://rahulshettyacademy.com/)**/maps/api/place/add/json?key=** **qaclick123**

**Base URL**:  [https://rahulshettyacademy.com](https://rahulshettyacademy.com/)

**Resource**: /maps/api/place/add/json

**Query Parameters**: key =qaclick123

**Http Method**: POST

**Sample Body** :

{

"location": {

"lat": -38.383494,

"lng": 33.427362

},

"accuracy": 50,

"name": "Frontline house",

"phone\_number": "(+91) 983 893 3937",

"address": "29, side layout, cohen 09",

"types": [

"shoe park",

"shop"

],

"website": "http://google.com",

"language": "French-IN"

}

**Sample Response**

{

"status": "OK",

"place\_id": "928b51f64aed18713b0d164d9be8d67f",

"scope": "APP",

"reference": "736f3c9bec384af62a184a1936d42bb0736f3c9bec384af62a184a1936d42bb0",

"id": "736f3c9bec384af62a184a1936d42bb0"

}

* There are three main sections into Java program that we have to include for testing APIs
  + **given()** :mention all given inputs here (e.g. parameters, headers, body)
  + **when()** :submit the api (e.g. here goes – resource, http method)
  + **then()** :validate the response
* all these three methods goes in concatenation (to each other by dot (.)
* Assign BaseURI

RestAssured.*baseURI* = "https://rahulshettyacademy.com";

* Add static Package manually (**import** **static** io.restassured.RestAssured.\*;)
* 
* given().log().all() 🡺 prints all the logs for given() section in output pane
* .then().log().all() 🡺 prints all the logs for then() section in output pane
* .assertThat() 🡺 assert (verify) expected values with actual response after running this program
* In above program, we are validating:
  + .statusCode(200)
  + . body(“scope”, equalTo(“App”))
  + .header(“server”, “Apache/2.4.18 (Ubuntu”)
* .statusCode(200) 🡺 verify expected status code
* .body(“scope”, equalTo(“App”)) 🡺 verify if ‘scope’ value is ‘App’ in Response Body
  + Note that, here to use ‘equalTo()’ method, we have to import another static package manually .. **import static org.hamcrest.Matchers.\*;**
* .header(“server”, “Apache….”) 🡺 verify if ‘server’ value is expected in Response Header