SPRING REST

- Process to create and run the Spring REST project (no view, no tsp or servlet)
  - Create Maven project
    - Packaging war
  - Dependencies
    - Spring-context
    - Spring webmvc
    - Servlet-api
    - Jackson
    - Jakarta jaxb
    - Mysql-connector
    - Validator
    - Slf4j
    - Log4j
  - Configure web.xml
  - Configure servlet-context.xml, root-context.xml
  - Create controller, service, model, ado, exceptional handling
  - Build and package as war
  - Deploy war into webserver(Tomcat)

  ——

- Spring REST will not have view resolver
  The response willl be in the form of data

FRONT END ——> REST API ———> BACK END.
(Angular js, react js)
(API, ServiceLayer, DataLayer)

- Web Services
  - Reusable ,Contract based, Interoperability, Heterogeneous, Loosely coupled
  - SOAP (Simple Object Access Protocol)
    - Resides on top of HTTP layer
    - The request from client converted to SOAP format (SOAP client) and

sent to the service having another SOAP client to produce the service.
    - SOAP clients sits on both front and back ends (disadvantage of SOAP web service)
- REST
    - REST - REST + HTTP
    - Representational State Transfer
- ————————————————————————

REST:
Rest is not a protocol like HTTP.
It is a set of guidelines to build a scalable, performant,fault-tolerant and extendable system
REST stands for REpresentational State Transfer.
It means when a RESTful API is called, the server will transfer to the client a representation(html,json,xml data) of the state of the requested resource.

RESTful WebService:
If a web service implements REST guidelines, then it is called RESTful web service.
A RESTful web application exposes information about itself in the form of information about its resources. It also enables the client to take actions on those resources, such as create new resources (i.e. create a new user) or change existing resources (i.e. edit a post).
In order for your APIs to be RESTful, you have to follow a set of constraints when you write them. The REST set of constraints will make your APIs easier to use and also easier to discover, meaning a developer who is just starting to use your APIs will have an easier time learning how to do so.

REST stands for REpresentational State Transfer.
It means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource.
For example, when a developer calls Instagram API to fetch a specific user (the resource), the API will return the state of that user, including their name, the number of posts that user posted on Instagram so far, how many followers they have, and more.
The representation of the state can be in a JSON format, and probably for most APIs this is indeed the case. It can also be in XML or HTML format.
What the server does when you, the client, call one of its APIs depends on 2 things that you need to provide to the server:

- An identifier for the resource you are interested in. This is the URL for the resource, also known as the endpoint. In fact, URL stands for Uniform Resource Locator.

- The operation you want the server to perform on that resource, in the form of an HTTP method, or verb. The common HTTP methods are GET,

POST, PUT, and DELETE.

For example, fetching a specific Twitter user, using Twitter's RESTful API, will require a URL that identify that user and the HTTP method GET.
Another example, this URL: www.twitter.com/jk_rowling has the unique identifier for J. K. Rowling's Twitter user, which is her username, jk_rowling. Twitter uses the username as the identifier, and indeed Twitter usernames are unique — there are no 2 Twitter users with the same username.
The HTTP method GET indicates that we want to get the state of that user.

RESTful Service:
Communication of data occurs in JSON format.
CRUD, Get all students, Patch(updating the required field)

REST Standards:
POST /students
GET /students
GET /students/{student_id}
PUT /student/{student_id}
DELETE /student/{student_id}
PATCH /student/{student_id}

In order for an API to be RESTful, it has to adhere to 6 constraints:

1. Uniform interface :
2. Client — server separation
3. Stateless
4. Layered system

5. Cacheable

6. Code-on-demand

1. Uniform interface:

- The request to the server has to include a resource identifier
- The response the server returns include enough information so the client can modify the resource
- Each request to the API contains all the information the server needs to perform the request, and each response the server returns contain all the information the client needs in order to understand the response.
- The result of the uniform interface is that requests from different clients look the same, whether the client is a chrome browser, a linux server, a python script, an android app or anything else.

2. Client — server separation

The client and the server act independently each on its own (if server gets changed, client should not be affected), and the interaction between them is only in the form of requests, initiated by the client only, and responses, which the server send to the client only as a reaction to a request. The server just sits there waiting for requests from the client to come. The server doesn't start sending away information about the state of some resources on its own.

3. Stateless

Stateless means the server does not remember anything about the user who uses the API. It doesn't remember if the user of the API already sent a GET request for the same resource in the past, it doesn't remember which resources the user of the API requested before, and so on. Instead for the initial request, it sends a token for the client so the client can access the session with that token.

Note: No client data is stored on the server. If the client need to maintain stateful app, where the user logs in once and does other authorized operations after that, then each request from the client should contain all the information(ex: token) necessary to service the request – including authentication and authorization details.

4. Layered system
Between the client who requests a representation of a resource's state, and the server who sends the response back, there might be a number of servers in the middle. These servers might provide a security layer, a caching layer, a load-balancing layer, or other functionality. Those layers should not affect the request or the response. The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request.
5. Cacheable
This means that the data the server sends contain information about whether or not the data is cacheable. If the data is cacheable, it might contain some sort of a version number. The version number is what makes caching possible: since the client knows which version of the data it already has (from a previous response), the client can avoid requesting the same data again and again. The client should also know if the current version of the data is expired, in which case the client will know it should send another request to the server to get the most updated data about the state of a resource.
Note: Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.

6. Code-on-demand
This constraint is optional — an API can be RESTful even without providing code on demand.
The client can request code from the server, and then the response from the server will contain some code, usually in the form of a script, when the response is in HTML format. The client then can execute that code.

Difference between HTTP and REST
HTTP is a contract, a communication protocol and REST is a concept.
REST is an architectural style which may use HTTP, FTP or other communication protocols but is widely used with HTTP.

Difference between SOAP and REST
Representational state transfer (REST) is a set of architectural principles. Simple object access protocol (SOAP) is an official protocol maintained by the World Wide Web Consortium (W3C). The main difference is that SOAP is a protocol while REST

is not.

Terminology in REST WEB SERVICES

- Client — the client is the person or software who uses the API. It can be a developer, for example you, as a developer, can use Twitter API to read and write data from Twitter, create a new tweet and do more actions in a program that you write. Your program will call Twitter's API. The client can also be a web browser. When you go to Twitter website, your browser is the client who calls Twitter API and uses the returned data to render information on the screen.

- Resource — a resource can be any object the API can provide information about. In Instagram's API, for example, a resource can be a user, a photo, a hashtag. Each resource has a unique identifier. The identifier can be a name or a number.

- AP1 - Application Programming Interface

A medium between the client and the server.

It is a contract (or specification) promised by the software which it will honor if other software wants to interact with it for performing business operations. API allows two or more software applications to talk to each other through a well-defined computing interface.

API Example in Real Life

Mobile apps can be one of the very familiar examples of APIs. Mobile applications generally contain all the UI related parts of the application. All the real-time data displayed in the mobile apps is fetched through the APIs. All the weather information, emails, game scores, live telecasts, and many more such real-time data is fetched from APIs deployed on the server.