**1. REST Introduction**

**What is REST?**

The REST stands for **REpresentational State Transfer**.

Let's understand the meaning of each word in the REST acronym.

* **State** means data
* **REpresentational** means formats (such as XML, JSON, YAML, HTML, etc)
* **Transfer** means carrying data between consumer and provider using the HTTP protocol

**REpresentational State Transfer**

* REST was originally coined by Roy Fielding, who was also the inventor of the HTTP protocol.
* A REST API is an intermediary Application Programming Interface that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.
* The REST architectural style has quickly become very popular over the world for designing and architecting applications that can communicate.
* The need for REST APIs increased a lot with the drastic increase of mobile devices. It became logical to build REST APIs and let the web and mobile clients consume the API instead of developing separate applications.

**2. REST Architecture**

The below diagram shows the typical REST architecture:

Let's understand a few web service term's by looking into the above architecture:

**Request and Response:** Request is the input to a web service, and the response is the output from a web service.

**Message Exchange Format:**It is the format of the request and response. There are two popular message exchange formats: XML and JSON.

**Service Provider or Server:** The service provider is one that hosts the web service.

**Service Consumer or Client:**A service consumer is one who is using a web service.

It is the responsibility of the consumer means client application to prepare and send HTTP request message

It is the responsibility of the business component (developed by a service provider) to prepare and send the HTTP response message

**3. REST Architectural Constraints**

An API that has the following constraints is known as RESTful API:

**Client-server architecture:** The client is the front-end and the server is the back-end of the service. It is important to note that both of these entities are independent of each other.

**Stateless:** No data should be stored on the server during the processing of the request transfer. The state of the session should be saved at the client’s end.

**Cacheable:**The client should have the ability to store responses in a cache. This greatly improves the performance of the API.

**Uniform Interface:** This constraint indicates a generic interface to manage all the interactions between the client and server in a unified way, which simplifies and decouples the architecture.

**Layered System:**The server can have multiple layers for implementation. This layered architecture helps to improve scalability by enabling load balancing.

**Code on Demand:**This constraint is optional. This constraint indicates that the functionality of the client applications can be extended at runtime by allowing a code download from the server and executing the code.

Read more about constraints at [**https://www.javaguides.net/2018/06/rest-architectural-constraints.html**](https://www.javaguides.net/2018/06/rest-architectural-constraints.html)

**4. REST Key Concepts**

**Resource**

The fundamental concept of a REST-based system is the resource. A resource is anything you want to expose to the outside world, through your application.

**Example 1:** Resources for Employee Management System:

- Employee

- Department

- Projects

- Task

- Address

**Example 2:**Resources for Student Management System:

- Student

- Teacher

- School

- Class

- Subject

**URI - Uniform Resource Identifier**

The resource can be identified by a **Uniform Resource Identifier (URI).** For web-based systems, HTTP is the most commonly used protocol for communicating with external systems. You can identify a unique resource using a URI.

Consider, we are developing a simple blog application and you can define URIs for a blog Post resource:

GET—**http://localhost:8080/api/posts/:** Returns a list of all posts

GET—**http://localhost:8080/api/posts/2**: Returns a post whose ID is 2

POST—**http://localhost:8080/api/posts/**: Creates a new Post resource

PUT—**http://localhost:8080/api/posts/2:** Updates a POST resource whose ID is 2

DELETE—**http://localhost:8080/api/posts/2:** Deletes a POST resource whose ID is 2

**Sub-resource**

In REST, the relationships are often modeled by a sub-resource. Use the following pattern for sub-resources.

GET  /{resource}/{resource-id}/{sub-resource}

GET  /{resource}/{resource-id}/{sub-resource}/{sub-resource-id}

POST /{resource}/{resource-id}/{sub-resource}

**Example:**

GET  /{post}/{post-id}/{comments}

GET  /{post}/{post-id}/{comments}/{comment-id}

POST /{post}/{post-id}/{comments}

**Use sub-resources child object cannot exist without its parent.**

**HTTP Methods**

Common HTTP verbs:

•  GET—To get a collection or a single resource

•  POST—To create a new resource

•  PUT—To update an existing resource

•  DELETE—To delete a collection or a single resource

Read more about HTTP methods at [**REST API - HTTP Methods**](https://www.javaguides.net/2021/01/rest-api-http-methods.html)

**HTTP Status Code**

Some of the frequently used status codes in this class are as follows:

* **200 OK:**This code indicates that the request is successful and the response content is returned to the client as appropriate.
* **201 Created:** This code indicates that the request is successful and a new resource is created.
* **400 Bad Request:** This code indicates that the server failed to process the request because of the malformed syntax in the request. The client can try again after correcting the request.
* **401 Unauthorized:**This code indicates that authentication is required for the resource. The client can try again with appropriate authentication.
* **403 Forbidden:** This code indicates that the server is refusing to respond to the request even if the request is valid. The reason will be listed in the body content if the request is not a HEAD method.
* **404 Not Found:**This code indicates that the requested resource is not found at the location specified in the request.
* **500 Internal Server Error:**This code indicates a generic error message, and it tells that an unexpected error occurred on the server and that the request cannot be fulfilled.

Spring Boot, a module of the Spring framework, facilitates **Rapid Application Development**(RAD) capabilities.

This **Spring tutorial**includes basic to advanced topics of Spring Boot, like Basics of Spring Boot, Spring Boot core, Spring Boot REST API, Spring Boot with Microservices, Spring Boot with Kafka, Spring Boot with Database and Data JPA, etc.

**What is Spring Boot?**

**Spring Boot**is an open-source Java framework used to create a Micro Service. Spring boot is developed by Pivotal Team, and it provides a faster way to set up and an easier, configure, and run both simple and web-based applications. It is a combination of Spring Framework and Embedded Servers. The main goal of Spring Boot is to reduce development, unit test, and integration test time and in Spring Boot, there is no requirement for XML configuration.

**Prerequisite of Spring Boot**

* Understanding of Java syntax and core concepts such as classes, objects, inheritance, interfaces, and exception handling
* Basic understanding of the Spring framework
* Knowledge of Web Technologies
* Familiarity with build tools such as Maven or Gradle

**What is Spring Boot?**

**Spring Boot**is an open-source Java framework used to create a Micro Service. Spring boot is developed by Pivotal Team, and it provides a faster way to set up and an easier, configure, and run both simple and web-based applications. It is a combination of Spring Framework and Embedded Servers. The main goal of Spring Boot is to reduce development, unit test, and integration test time and in Spring Boot, there is no requirement for XML configuration.

**Prerequisite of Spring Boot**

* Understanding of Java syntax and core concepts such as classes, objects, inheritance, interfaces, and exception handling
* Basic understanding of the Spring framework
* Knowledge of Web Technologies
* Familiarity with build tools such as Maven or Gradle

## What is Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the **RAD (Rapid Application Development)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.