

# Introduction to Spring Boot

By Ram Lovewanshi

# What Is Spring Boot

- Spring Boot is a module of the Spring Framework.
- It is used to create stand-alone, production-grade Spring Based Applications with minimum efforts.
- It is developed on top of the core Spring Framework.
- Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above it.

# Spring Boot vs Spring

- **Spring:** Spring framework is the most popular application development framework of Java. The main feature of the Spring Framework is dependency Injection or Inversion of Control (IoC). With the help of Spring Framework, we can develop a loosely coupled application. It is better to use if application type or characteristics are purely defined.
- **Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

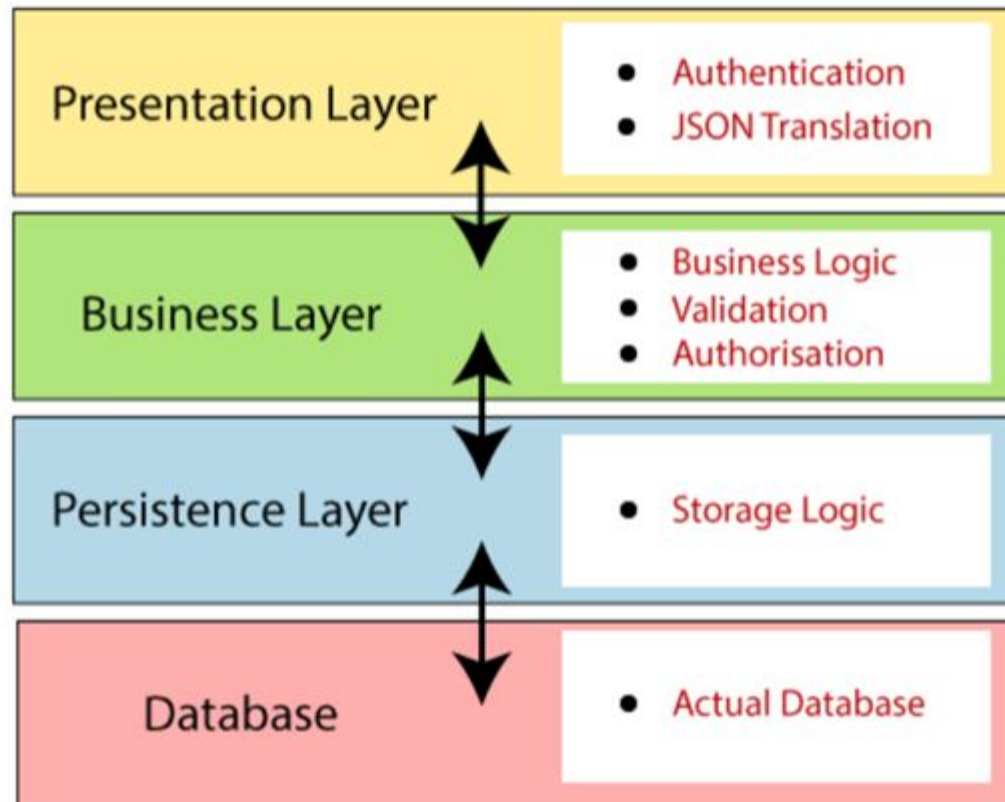
# Spring Boot vs Spring MVC

- **Spring Boot:** Spring Boot makes it easy to quickly bootstrap and start developing a Spring-based application. It avoids a lot of boilerplate code. It hides a lot of complexity behind the scene so that the developer can quickly get started and develop Spring-based applications easily.
- **Spring MVC:** Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities. It is an HTTP oriented web application development framework.


# Spring Boot Layer

- Before understanding the **Spring Boot Architecture**, we must know the different layers and classes present in it. There are four layers in Spring Boot are as follows:
  - Presentation Layer
  - Business Layer
  - Persistence Layer
  - Database Layer

# Spring Boot Layer



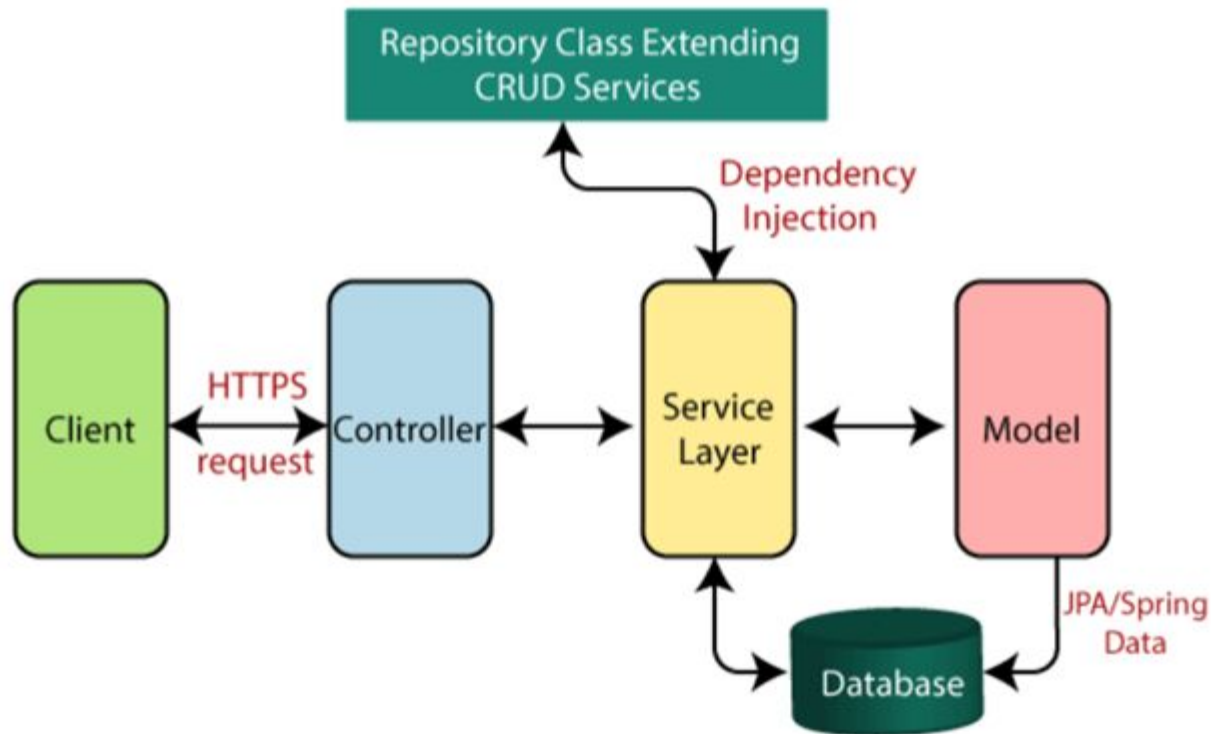
# Spring Boot Layer

- **Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.
- **Business Layer:** The business layer handles all the business logic. It consists of service classes and uses services provided by data access layers. It also performs authorization and validation.
- **Persistence Layer:** The persistence layer contains all the storage logic and translates business objects from and to database rows.
- **Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed. 



# Spring Boot Flow Architecture

Spring Boot flow architecture





# Spring Boot Flow Architecture

- Now we have validator classes, view classes, and utility classes.
- Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for DAO and DAOImpl classes in Spring boot.
- Creates a data access layer and performs CRUD operation.
- The client makes the HTTP requests.
- The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
- A JSP page is returned to the user if no error occurred.



# What is Spring Initializer?

- Spring Initializr is a web-based tool provided by the Pivotal Web Service.
- With the help of Spring Initializr, we can easily generate the structure of the Spring Boot Project.
- It offers extensible API for creating JVM-based projects.

# Maven Tools

Maven is a powerful project management tool that is based on POM (project object model). It is used for project build, dependency, and documentation.

# What maven does do?

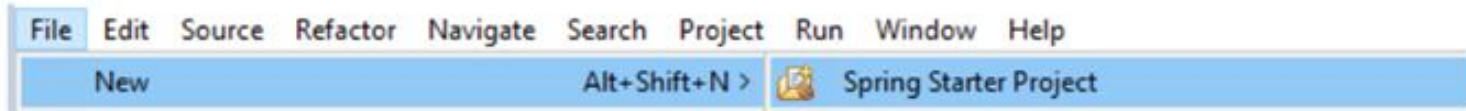
- We can easily build a project using maven.
- We can add jars and other dependencies of the project easily using the help of maven.
- Maven provides project information (log document, dependency list, unit test reports, etc.)
- Maven is very helpful for a project while updating the central repository of JARs and other dependencies.
- With the help of Maven, we can build any number of projects into output types like the JAR, WAR, etc without doing any scripting.

# What maven does do?

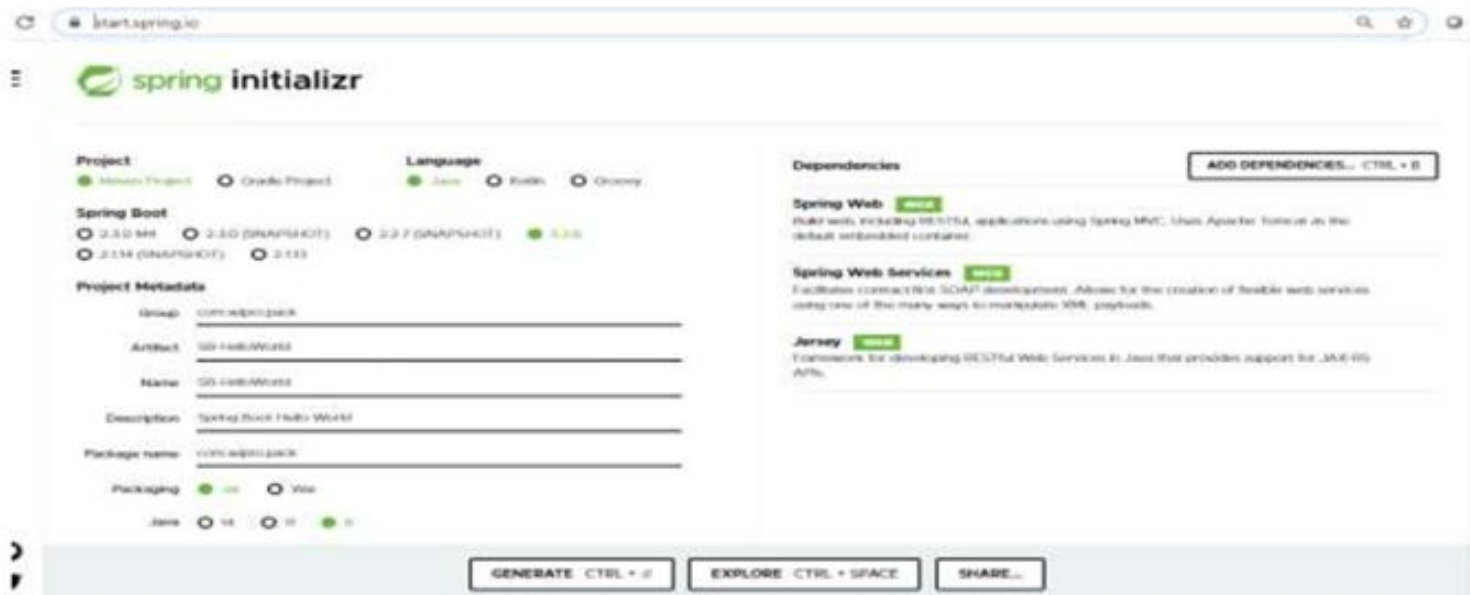
- Maven also helps in managing the project's build lifecycle, including tasks like compiling, testing, packaging, and deploying the code.
- Maven provides a standard project structure, making it easy for developers to understand the layout of the project and locate specific files.
- Maven simplifies the process of managing project dependencies, ensuring that the correct versions of libraries and frameworks are used throughout the project.

# Spring Boot Project Creation

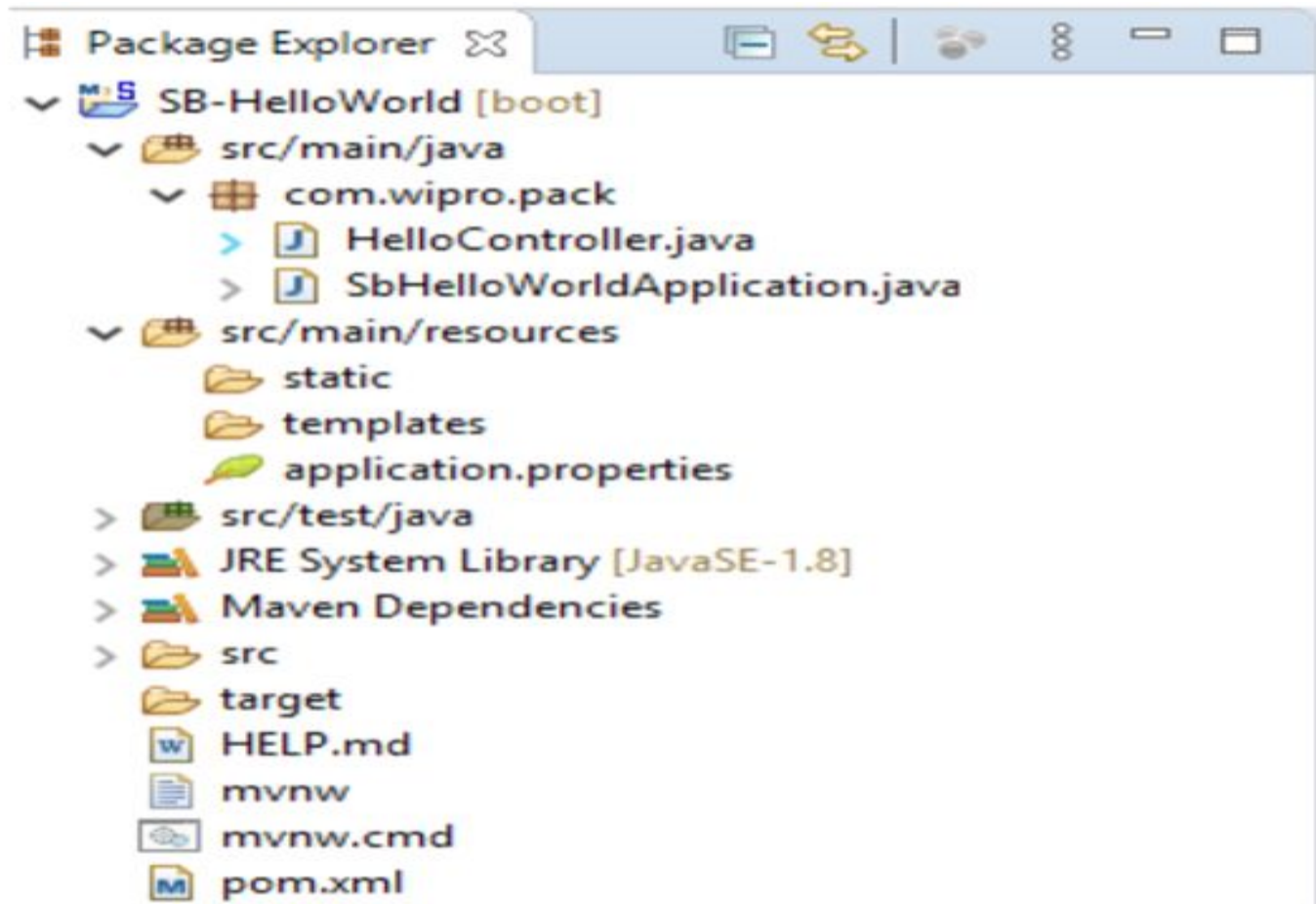
- Using STS :



- Using Spring Initializr : [start.spring.io](https://start.spring.io)



# Spring Boot Project Structure





# Spring Boot Hello World

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SbHelloWorldApplication {

    public static void main(String[] args) {
        SpringApplication.run(SbHelloWorldApplication.class, args);
    }
}
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;

@Controller
public class HelloController {

    @RequestMapping("/hello")
    public String sayHello() {
        return "Hello World";
    }
}
```

← → ↻ ⓘ localhost:8282/hello

Hello World

# Spring Boot – How Thymeleaf Works?

Thymeleaf is a Java library, template engine used to parse and render the data produced by the application to template files – thus providing transformation. It is just like HTML but is provided with more attributes for working with rendered data. It allows caching of the parsed data/file to increase efficiency while at production. Types of templates it can process are – HTML, JAVASCRIPT, CSS, XML, TEXT, RAW.

# Spring Boot – How Thymeleaf Works?

## **Template engines used with Spring-Boot:**

- Thymeleaf
- jsp

# Spring Boot Web App

- Since we are going to use JSP in our example, we need to add the below dependency in our pom.xml as per our embedded tomcat version.
- Because mainly Spring Boot is concentrating on REST and Micro Services.

```
<dependency>  
    <groupId>org.apache.tomcat</groupId>  
    <artifactId>tomcat-jasper</artifactId>  
    <version>9.0.33</version>  
</dependency>
```

- Here, we are going to use two annotations `@Controller` and `@RequestMapping`.

# How to find out embedded tomcat version

While running the previous example, We can find the embedded tomcat version from the console like below.

```
. Starting DemoApplication on E 150130383 with PID 34012
: No active profile set, falling back to default profile
erver : Tomcat initialized with port(s): 9022 (http)
vice  : Starting service [Tomcat]
ngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
r      : At least one JAR was scanned for TLDs yet contained no
       : Initializing Spring embedded WebApplicationContext
ntext  : Root WebApplicationContext: initialization completed i
```

We can get the dependency from the below link

<https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper>

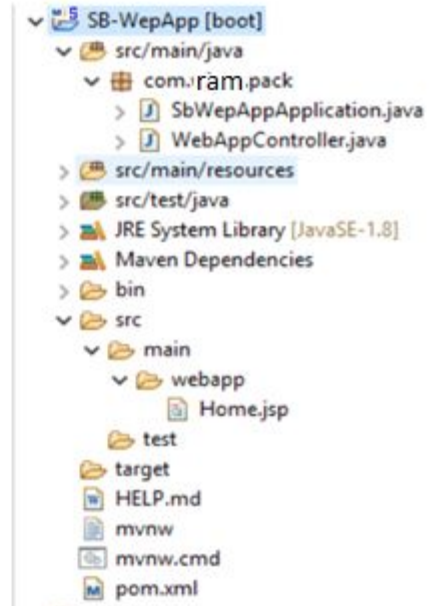
# Spring Boot Web App

```
package com.ram.pack;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class WebAppController {

    @RequestMapping("/home")
    public String home() {
        return "Home.jsp";
    }

}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color=green><b>Welcome to Home Page</b></font>
</body>
</html>
```



← → ↻ ⓘ localhost:8123/home

Welcome to Home Page

# Spring Web App Using Properties File

- In our previous example, We have created Home.jsp file inside “webapp” folder. Because by default the spring boot will check this folder to find the view pages.
- Also in our “WebAppController”, we have hard coded the extensions of view page as “.jsp”.
- In case, If I want to store all my view pages in a separate folder and I want change the view page extension based on customer requirement with out touching my Controller then the properties file will come to the picture.



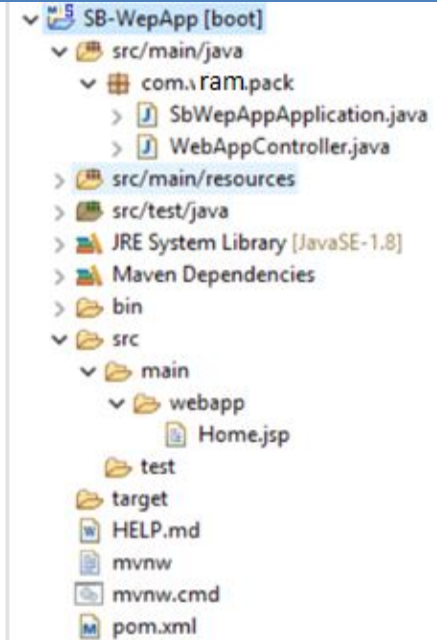
# Spring Web App Using Properties File

```
package com.ram .pack;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class WebAppController {

    @RequestMapping("/home")
    public String home() {
        return "Home.jsp";
    }

}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color=green><b>Welcome to Home Page</b></font>
</body>
</html>
```



← → ↻ ⓘ localhost:8123/home

Welcome to Home Page

# SB Web App Accepting Client Data

## SB Web App Accepting Client Data.

```
package com.ram .pack;
import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class WebAppController {
    @RequestMapping("/home")
    public String home() {
        return "Home";
    }
    @RequestMapping("/welcome")
    public String welcome(HttpServletRequest req) {
        String name = req.getParameter("name");
        req.setAttribute("name", name);
        return "Welcome";
    }
}
```

### Home.jsp

```
<form action="welcome"><font color=green><b>
Enter Your Name : <input type="text"
name="name"/><br>
<input type="submit" value="Submit"/>
</form></b></font>
```

### Welcome.jsp

```
<body>
<font color=green><b>WelCome
${name}</b></font>
</body>
```

← → ↻ ⓘ localhost:8123/home

Enter Your Name :

← → ↻ ⓘ localhost:8123/welcome?name=Valan

Welcome **Ram Lovewanshi**

# Spring Boot MVC

## Spring Boot MVC.

```
package com. ram.pack;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class WebAppController {
    @RequestMapping("/home")
    public String home() {
        return "Home";
    }
    @RequestMapping("/welcome")
    public ModelAndView welcome(@RequestParam("name")
String name) {
        ModelAndView mv = new ModelAndView();
        mv.addObject("name", name);
        mv.setViewName("Welcome");
        return mv;
    }
}
```

### Home.jsp

```
<form action="welcome"><font color=green><b>
Enter Your Name : <input type="text"
name="name"/><br>
<input type="submit" value="Submit"/>
</form></b></font>
```

### Welcome.jsp

```
<body>
<font color=green><b>Welcome
${name}</b></font>
</body>
```

← → ↻ ⓘ localhost:8123/home

Enter Your Name : Ram Lovewanshi

Submit

← → ↻ ⓘ localhost:8123/welcome?name=Valan

Welcome Ram Lovewanshi

# Introduction To Web Services

## What is Web Services?

- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
- Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks.

# Benefits of Web Services?

## Benefits of Web Services?

- **Loosely Coupled**

Each service exists independently of the other services. Individual pieces of the application to be modified without impacting unrelated areas.

- **Ease of Integration**

Data is isolated between applications. Web Services act as glue between these and enable easier communications within and across organizations.

- **Service Reuse**

Takes code reuse a step further. A specific function within the domain is only ever coded once and used over and over again by consuming applications.

## Web Services Architecture.

- The simplest web service system has two participants:
  - A service **producer** (provider)
  - A service **consumer** (requester)
  
- The provider presents the interface and implementation of the service, and the requester uses the Web service.



# Types of web Services

## Types of Web Services.

- There are mainly two types of web services.
  - SOAP web services.
  - RESTful web services



# Introduction to RESTful Web Services

## What is RESTful Web Services?

- REST is used to build Web services that are lightweight, maintainable, and scalable in nature.
- A service which is built on the REST architecture is called a RESTful service.
- The underlying protocol for REST is HTTP, which is the basic web protocol.
- REST stands for **RE**presentational **S**tate **T**ransfer.
- REST uses various representation to represent a resource like text, JSON, XML.

# HTTP METHODS

## HTTP Methods

GET

POST

PUT

DELETE

# REST GET METHOD

## What is GET Method?

- This request is used to get a resource from a server.
- If we perform a GET request, the server looks for the data we requested and sends it back to us.
- In other words, a GET request performs a READ operation.
- This is the default request method.

# REST GET Method Example

```
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Employee {
    @Id
    private int eid;
    private String ename;
    private int esalary;

    //Getters & Setters

    @Override
    public String toString() {
        return "Employee [eid=" + eid + ", ename="
            + ename + ", esalary=" + esalary + "]";
    }
}
```

```
data.sql
1 insert into employee values (101, 'valan', 2000);
2 insert into employee values (102, 'arasu', 3000);
3 insert into employee values (103, 'vijay', 4000);
4 insert into employee values (104, 'raj', 5000);
5 insert into employee values (105, 'kumar', 6000);
```

# REST GET Method Example

```
package com.ram.pack.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import com.wipro.pack.dao.EmployeeDao;
@RestController
public class EmployeeController {

    @Autowired
    EmployeeDao dao;

    @GetMapping("/getEmployees")
    public String getEmployees() {
        return dao.findAll().toString();
    }

    @GetMapping("/getEmployee/{eid}")
    public String getEmployee(@PathVariable("eid")
        Integer eid) {
        return dao.findById(eid).toString();
    }
}
```

## Note:

@RestController = @Controller +  
@ResponseBody

In case of @Controller, The return value will be consider as a View Page name.

But here we are not returning view page name.  
We are returning a data.

So If it is a @Controller then we have to add  
@ResponseBody annotations.

But @RestController will take it by default.

# REST GET Method Example

## REST GET Method – Example.

```
package com.ram.pack.dao;

import org.springframework.data.repository.CrudRepository;

import com.ram.pack.model.Employee;

public interface EmployeeDao extends CrudRepository<Employee, Integer>{

}
```

← → ↻ ⓘ localhost:8123/getEmployee/102 ☆

Optional[Employee [eid=102, ename=arasu, esalary=3000]]

← → ↻ ⓘ localhost:8123/getEmployees ☆ 🌐 🍷 | 📄 🟡

[Employee [eid=101, ename=valan, esalary=2000], Employee [eid=102, ename=arasu, esalary=3000], Employee [eid=103, ename=vijay, esalary=4000], Employee [eid=104, ename=raj, esalary=5000], Employee [eid=105, ename=kumar, esalary=6000]]

# REST GET Method Example

## REST GET Method – Example.

- If we look at our output. We can see it is displaying a `List<Employee>` as String.
  - But We want to produce the output as json format.
  - Spring Boot will perform this by default.
  - To get this, Our dao should extend `JpaRepository` instead of `CrudRepository`.
  - `JpaRepository` is extending `CrudRepository`.
-



# REST GET Method Example

```
@RestController
public class EmployeeController {

    @Autowired
    EmployeeDao dao;

    @GetMapping("/getEmployees")

    public List<Employee> getEmployees() {
        return dao.findAll();
    }

    @GetMapping("/getEmployee/{eid}")
    public Optional<Employee> getEmployee(@PathVariable("eid") Integer eid) {
        return dao.findById(eid);
    }
}
```

# POSTMAN

## What is Post Man?

- Postman is a popular API client that makes it easy for developers to create, share, test and document APIs.
- This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses.
- The result - more efficient and less tedious work.
- <https://www.postman.com/downloads/>

# REST POST METHOD

## What is POST Method?

- This request is used to create a new resource on a server.
- If we perform a POST request, the server creates a new entry in the database and tells us whether the creation is successful.
- In other words, a POST request performs an CREATE operation.

# REST POST METHOD

```
@RestController
public class EmployeeController {

    @Autowired
    EmployeeDao dao;

    @GetMapping("/getEmployees")

    public List<Employee> getEmployees() {
        return dao.findAll();
    }

    @GetMapping("/getEmployee/{eid}")
    public Optional<Employee>
    getEmployee(@PathVariable("eid") Integer eid) {
        return dao.findById(eid);
    }

    @PostMapping("/createEmployee")
    public Employee createEmployee(Employee emp) {
        return dao.save(emp);
    }
}
```

```
package com. ram. pack.dao;

import
org.springframework.data.jpa.repository.JpaRepository;

import com..ram. .pack.model.Employee;

public interface EmployeeDao extends JpaRepository<Employee,
Integer>{

}
```

# REST PUT Method

## What is PUT Method?

- This request is used to update a resource on a server.
- If we perform a PUT request, the server updates an entry in the database and tells us whether the update is successful.
- In other words, a PUT request performs an UPDATE operation.

# REST PUT Method Example

```
@RestController
public class EmployeeController {

    @Autowired
    EmployeeDao dao;

    @GetMapping("/getEmployees")
    public List<Employee> getEmployees() {
        return dao.findAll();
    }

    @GetMapping("/getEmployee/{eid}")
    public Optional<Employee>
    getEmployee(@PathVariable("eid") Integer eid) {
        return dao.findById(eid);
    }

    @PostMapping("/createEmployee")
    public Employee createEmployee(Employee emp) {
        return dao.save(emp);
    }

    @PutMapping("/updateEmployee")
    public Employee updateEmployee(Employee emp) {
        return dao.save(emp);
    }
}
```

# REST DELETE METHOD

## What is DELETE Method?

- This request is used to delete a resource from a server.
- If we perform a DELETE request, the server deletes an entry in the database and tells us whether the deletion is successful.
- In other words, a DELETE request performs a DELETE operation.



# REST DELETE METHOD

```
@RestController
public class EmployeeController {
    @Autowired
    EmployeeDao dao;
    @GetMapping("/getEmployees")
    public List<Employee> getEmployees() {
        return dao.findAll();
    }
    @GetMapping("/getEmployee/{eid}")
    public Optional<Employee>
    getEmployee(@PathVariable("eid") Integer eid) {
        return dao.findById(eid);
    }
    @PostMapping("/createEmployee")
    public Employee createEmployee(Employee emp) {
        return dao.save(emp);
    }
    @PutMapping("/updateEmployee")
    public Employee updateEmployee(Employee emp) {
        return dao.save(emp);
    }
    @DeleteMapping("/deleteEmployee/{eid}")
    public void deleteEmployee(@PathVariable("eid")
    Integer eid) {
        dao.deleteById(eid);
    }
}
```