

SPRINGBOOT

Spring Boot Setup Project:

#1 Create one maven project

>File>New>other>Search Maven > Choose "Maven Project" >Next>Choose Checkbox > Create Simple Project> Next>Enter Details Like>

groupId: org.sathyatech

artifactId: Spring Boot AppOne

Version: 1.0

>Finish

#2 Open pom.xml and add below dependencies in given order only.

- a. Parent Project
- b. Properties
- c. Dependencies
- d. Build-plugin

#3 Create application.properties file

Under src/main/resources with few keys

>right click on src/main/resources > new > other > Search with "File" > Choose File > next> otherName

Ex: application.properties

>Finish

#4 Add Spring Boot Starter class under src/main/java

>Right click on src/main/java >new >Class> Enter name and package>

Ex: com.app, DemoApp > finish

#5 Run DemoApp (Starter Class) and goto browser, Enter URL like,

<http://localhost:2018/>

pom.xml Code:

<project>

```
<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>

</parent>

<properties>

    <java.version>1.8</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>

</project>
```

application.properties

server.port=2018

Starter Class

```
package com.app;
```

```
@SpringBootApplication
```

```
Public class DemoApp{
```

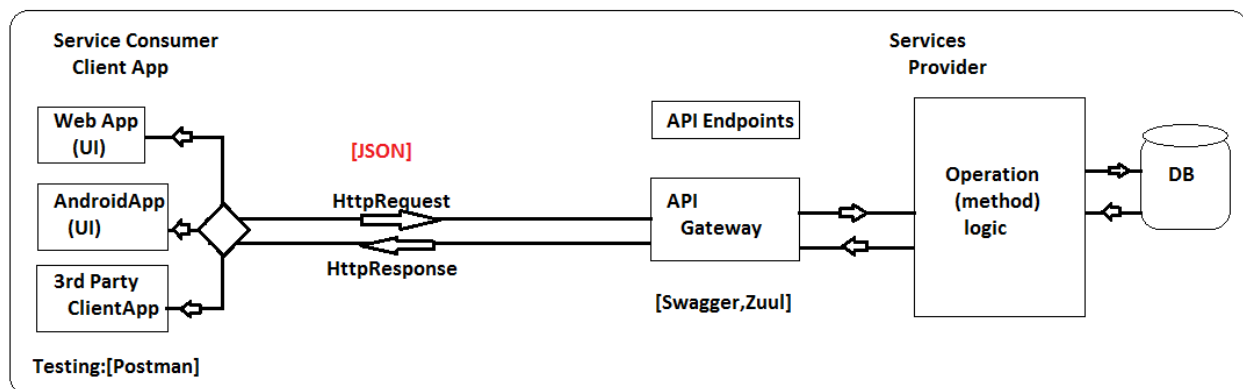
```
    main(String[] args){
```

```
        SpringApplication.run(DemoApp.class,arg);
```

```
    }
```

```
}
```

Generic Application Design using SpringBoot-ReSTWebservices:



SpringBoot ReST webservies are used to define application logic one time which can be reusable to any client/UI applications.

Here UI/Client can be WebApplication (developed using Angular, Spring UI etc...) or Mobile application (developed using Android UI) else it can be 3rd Party Integration (link with Payment Gateway, link with Online Shops, link with other application).

To implement this ReST webservies using SpringBoot, Annotations are:

[1.@RestController](#) [2.@GetMapping](#) [3.@PostMapping](#) [4.@PutMapping](#)

[5.@DeleteMapping](#) [6.@RequestBody](#) [7.@ResponseBody](#)

Predefined classes are: `ResponseEntity<T>`, `RestTemplate`

Note:

#1 default object conversion in ReST is JSON format.

#2 Supports also XML, but not used in real-time b'coz of performance issue.

#3 Uses Http Request and Http Response for consumer-provider communication

#4 API Gateway must be provided for Endpoint Accessing.

What is API Endpoint?

An Endpoint is used to access one Service Provider (method) which contains details like URL (Path), Type (GET, POST, PUT, DELETE...), Input, Output details.

Ex: Endpoint is:

Path = /getBal/{acclId}

Type= GET

Input = acts as Path Param

Output = Balance details as String.

In simple, Endpoint is an information of method to be accessed by client.

Service Provider App using SpringBoot and RestController:

To define Provider application, follow steps:

#1 write one public class with package and apply annotation at class level

@RestController

#2 Define one method in class with some logic (return type can be String, class, collection, etc.)

#3 Every method must have MethodType (GET,POST,PUT,DELETE) and path (URL)

**Path is case-sensitive

#4 To readData from application.properties define one variable in class level and use @Value("\${key}")

Code:

Under src/main/java

```
package com.app;
```

```
@RestController
```

```
public class EmpProvider{
```

```
    @GetMapping("/show")
```

```
    return "Hello";
```

```
    @Value("${server.port}")
```

```
    private String myPort;
```

```
    @GetMapping("/Show")
```

```
    public String showMsg(){
```

```
        return "Hello::Port is"+myPort;
```

```
    }
```

```
}
```

Steps:

#1 Go to start.spring.io

#2 Enter Details like

```
groupId: com.app;
```

```
artifactId: AppOne
```

```
Dependencies: web
```

Click one [Generate Project]

#3 Extract download zip into folder

#4 Import to eclipse (File>Import>Maven)

#5 Modify application.properties and define above code

```
server.port=2018
```

#6 Run Starter class and enter URL

<http://localhost:2018/show>

Note:

#1 Method Type and Uses

@GetMapping → Selecting Resource

@PostMapping → Creating New Resource

@PutMapping → Updating/Modifying existing Resource

@DeleteMapping → Remove existing Resource

Resource: A File (audio/video/text/image) or DB tables

#2 We cannot run same or different application on one port number

One service = one port

If we try to run 2nd process on existed port number, exception is BindException: Address already in use: bind

#3 If no port number is provided then default port number is 8080.

#4 If application is not under basePackage (Spring Boot Starter class package or its sub packages) then it will not be detected. So Object of your class cannot be created.

Starter Class Package	com.app
Ex:	
Class	
A	com.app ✓
A	com ✗
A	com.one ✗
A	com.app.one ✓

#5 To provide multiple base-packages to select all classes use annotation

```
@ComponentScan(basePackages={"pack1","pack2",""})
```

***It must be written at Boot starter class level only.

Ex:

```
package com.app;

@SpringBootApplication

@ComponentScan(basePackages={"com.one","app","ab"})

Public class AppProvider{

    main(String[] args){

        SpringApplication.run(AppProvider.class,args);

    }

}
```

**Now write your rest controller in com.one package also, it works.

JSON: (Java Script Object Notation)

Every SpringBoot Rest Application works by default with JSON Global Data Format.

JSON Format is:

```
{"key":Value}
```

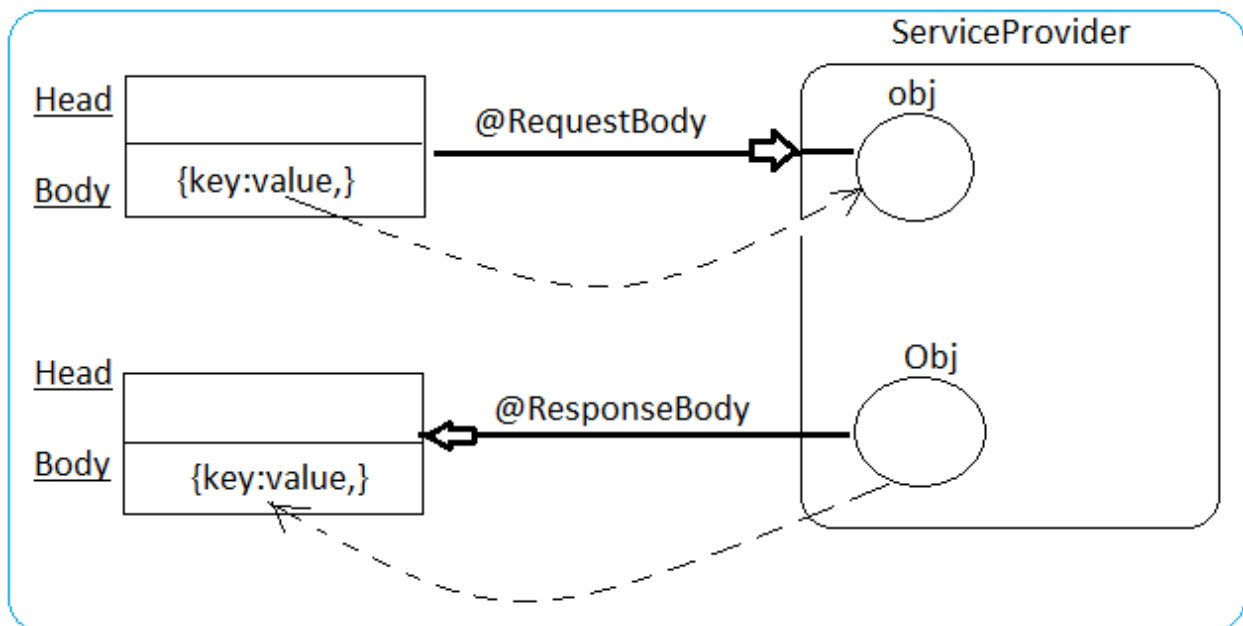
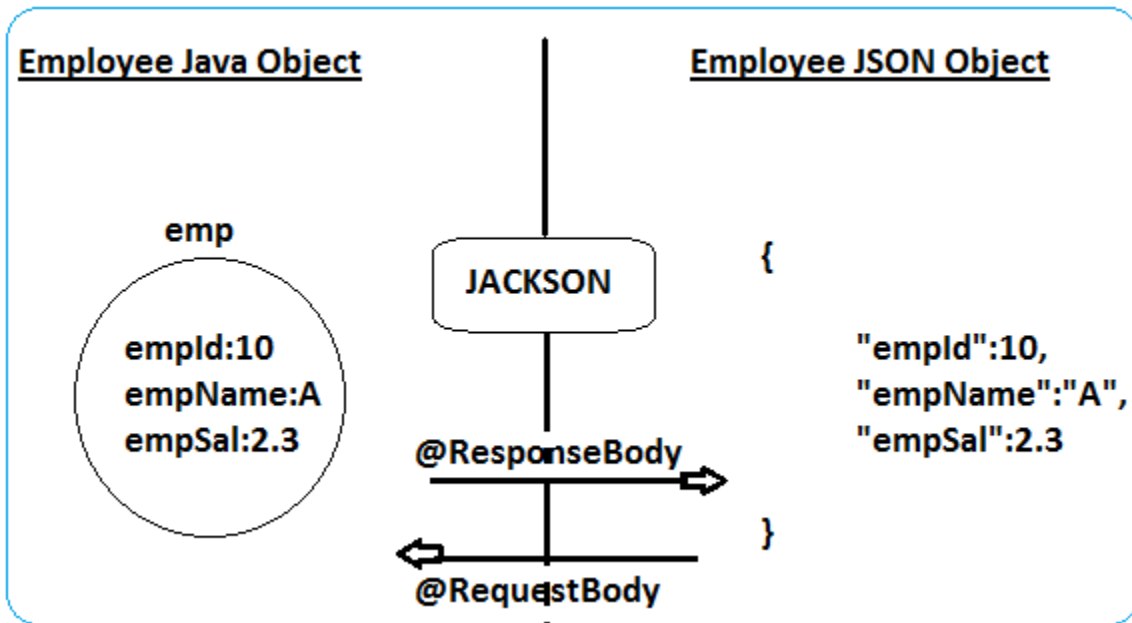
Every Java object can be converted to JSON format even reverse also possible.

JACKSON is an API used by SpringBoot with auto-conversion feature to convert Java Object to JSON format.

@RestController internally follows

@ResponseBody which converts Java object to JSON format.

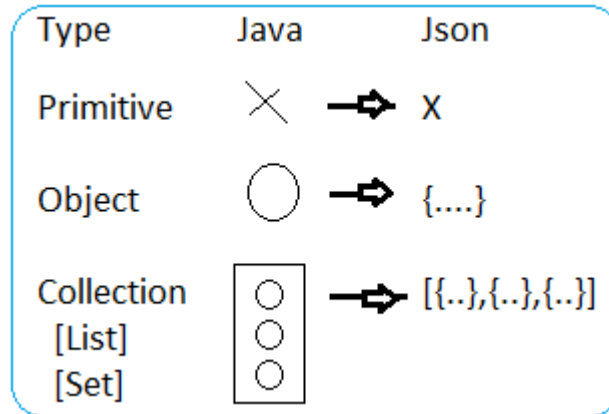
@RequestBody is used to convert JSON to Java object from HttpRequest Body.



JACKSON API supports both Java Object and Java Collection to JSON Format.

For Primitive data no conversion is provided (excepted as String type only).

Example:



RestController method return type can be primitive (String), Class type, Collection Type.

If class or collection type those are auto converted to JSON format.

Example:

Step#1 Download STS (Spring Tool Suite)

<http://spring.io/tools/sts/all>

Step#2 Extract to the folder and goto

".../sts-bundle/sts-3.9.4.RELEASE"

Click here "STS.exe" to start STS

Step#3 Create STS Starter Project

>File>new>spring starter project>Enter details>next>choose dependencies (Search and select web)>next>finish

Step#4 add below class under base package

package com.example.app.controller;

@RestController

Public class EmployeeProvider{

 @GetMapping("/show")

 Public Employee showData(){

 Employee emp = new Employee();

 emp.setEmpId(100);

```

        emp.setEmpName("AA");

        emp.setEmpSal(12.36);

        return emp;

    }

    @GetMapping("/showAll")

    Public List<Employee> emps = arrays.asList(new Employee(10,"A",1.1),new
Employee(11,"B",21.1),new Employee(12,"C",12.5),new Employee(13,"C",14.1));

    return emps;

}

```

****Create Employee.java class also**

```

package com.example.app.model;

public class Employee{

    private int empId;

    private String empName;

    private double empSal;

    //default const, 3-param constr,

    //set-get, toString

}

```

****Run starter class and enter URL**

<http://localhost:2018/show>

<http://localhost:2018/showAll>

Example: SpringWebMvcApplication

SpringWebMvcApplication.java

```
package com.app;

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

@SpringBootApplication

public class SpringWebMvcAppApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringWebMvcAppApplication.class, args);

    }

}
```

ServletInitializer.java

```
package com.app;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override

    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

        return application.sources(SpringWebMvcAppApplication.class);

    }

}
```

Employee.java

```
package com.app.model;

import java.util.List;

import javax.persistence.CollectionTable;
```

```
import javax.persistence.Column;

import javax.persistence.ElementCollection;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OrderColumn;

import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="emptab")

public class Employee {

    @Id

    @Column(name="eid")

    @GeneratedValue(generator="empgen")

    @GenericGenerator(name="empgen",strategy="increment")

    private Integer empId;

    @Column(name="ename")

    private String empName;

    @Column(name="email")

    private String empMail;

    @Column(name="egen")

    private String empGen;

    @Column(name="addr")

    private String empAddr;
```

```

@ElementCollection
@CollectionTable(
    name="emplangstab", //child table
    joinColumns=@JoinColumn(name="eidFk") //key column
)

@Column(name="langs") //element column
@OrderColumn(name="pos") //index column
private List<String> empLangs;

@Column(name="idType")
private String empldType;

@Column(name="idnum")
private String empldNum;

//----const & methods-----

//alt+shift+S,O(De-select-all>OK)

public Employee() {
    super();
}

public Employee(Integer empld) {
    super();
    this.empld = empld;
}

public Employee(Integer empld, String empName, String empMail, String empGen, String
empAddr, List<String> empLangs,
    String empldType, String empldNum) {
    super();

```

```
        this.empld = empld;

        this.empName = empName;

        this.empMail = empMail;

        this.empGen = empGen;

        this.empAddr = empAddr;

        this.empLangs = empLangs;

        this.empldType = empldType;

        this.empldNum = empldNum;
    }

    //alt+shift+S,R (selectAll>OK)
    public Integer getEmpld() {

        return empld;

    }

    public void setEmpld(Integer empld) {

        this.empld = empld;

    }

    public String getEmpName() {

        return empName;

    }

    public void setEmpName(String empName) {

        this.empName = empName;

    }

    public String getEmpMail() {

        return empMail;

    }
```

```
public void setEmpMail(String empMail) {  
    this.empMail = empMail;  
}  
  
public String getEmpGen() {  
    return empGen;  
}  
  
public void setEmpGen(String empGen) {  
    this.empGen = empGen;  
}  
  
public String getEmpAddr() {  
    return empAddr;  
}  
  
public void setEmpAddr(String empAddr) {  
    this.empAddr = empAddr;  
}  
  
public List<String> getEmpLangs() {  
    return empLangs;  
}  
  
public void setEmpLangs(List<String> empLangs) {  
    this.empLangs = empLangs;  
}  
  
public String getEmpIdType() {  
    return empIdType;  
}
```

```

        public void setEmpIdType(String empIdType) {

            this.empIdType = empIdType;

        }

        public String getEmpIdNum() {

            return empIdNum;

        }

        public void setEmpIdNum(String empIdNum) {

            this.empIdNum = empIdNum;

        }

        //alt+shift+S,S(OK)

        @Override

        public String toString() {

            return "Employee [empId=" + empId + ", empName=" + empName + ", empMail=" +
empMail + ", empGen=" + empGen

                + ", empAddr=" + empAddr + ", empLangs=" + empLangs + ",
empIdType=" + empIdType + ", empIdNum="

                + empIdNum + "]";

        }

    }

```

EmployeeRepository.java

```

package com.app.repo;

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

import com.app.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer>{

}

```


IEmployeeService.java

```
package com.app.service;

import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

import com.app.model.Employee;

public interface IEmployeeService {

    public Integer saveEmployee(Employee emp);

    public void deleteEmployee(Integer empId);

    public boolean isEmployeeExistById(Integer empId);

    public Employee getEmployeeById(Integer empId);

    public List<Employee> getAllEmployees();

    //special methods

    public Page<Employee> getAllEmployees(Pageable pageable);

}
```

EmployeeServiceImpl.java

```
package com.app.service.impl;

import java.util.Collections;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;
```

```
import com.app.model.Employee;

import com.app.repo.EmployeeRepository;

import com.app.service.IEmployeeService;

@Service

public class EmployeeServiceImpl implements IEmployeeService{

    @Autowired

    private EmployeeRepository repo;

    @Override

    public Integer saveEmployee(Employee emp) {

        return repo.save(emp).getEmpId();

    }

    @Override

    public void deleteEmployee(Integer empId) {

        repo.deleteByld(empId);

    }

    @Override

    public boolean isEmployeeExistByld(Integer empId) {

        return repo.existsByld(empId);

    }

    @Override

    public Employee getEmployeeByld(Integer empId) {

        Optional<Employee> emp=repo.findByld(empId);

        if(emp.isPresent())

            return emp.get();

        else
```

```

        return null;
    }

    @Override
    public List<Employee> getAllEmployees() {
        List<Employee> emps=repo.findAll();
        Collections.sort(emps, (e1,e2)->e1.getEmpId()-e2.getEmpId());
        return emps;
    }

    //special methods
    @Override
    public Page<Employee> getAllEmployees(Pageable pageable) {
        return repo.findAll(pageable);
    }
}

```

EmployeeValidator.java

```

package com.app.validator;

import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.model.Employee;

@Component
public class EmployeeValidator implements Validator{

```

```

@Override

public boolean supports(Class<?> clazz) {

    return Employee.class.equals(clazz);

}

@Override

public void validate(Object target, Errors errors) {

    Employee e=(Employee)target;

    if(e.getEmpName()==null || "".equals(e.getEmpName().trim())){

        errors.rejectValue("empName",null, "Please enter Employee Name!!");

    }

    /*if(e.getEmpMail()==null || "".equals(e.getEmpMail().trim())) {*/

    if(StringUtils.isEmpty(e.getEmpMail()) ||
StringUtils.containsWhitespace(e.getEmpMail())) {

        errors.rejectValue("empMail",null, "Please enter Employee Mail Id!!");

    }

    if(StringUtils.isEmpty(e.getEmpGen()) ||
StringUtils.containsWhitespace(e.getEmpGen())) {

        errors.rejectValue("empGen",null, "Please choose one Gender !!");

    }

    if(StringUtils.isEmpty(e.getEmpAddr()) ||
StringUtils.containsWhitespace(e.getEmpAddr())) {

        errors.rejectValue("empAddr",null, "Please enter Employee Address !!");

    }

    if(e.getEmpLangs()==null || e.getEmpLangs().isEmpty()) {

        errors.rejectValue("empLangs",null, "Please choose at least one Language !!");

    }

    if(StringUtils.isEmpty(e.getEmpIdType())) {

```

```

        errors.rejectValue("empIdType",null, "Please Choose one Type !!");
    }

    if(StringUtils.isEmpty(e.getEmpIdNum()) ||
    StringUtils.containsWhitespace(e.getEmpIdNum())) {

        errors.rejectValue("empIdNum",null, "Please Enter ID Number !!");
    }

}

}
}

```

EmployeeController.java

```

package com.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.data.web.PageableDefault;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import com.app.model.Employee;
import com.app.service.IEmployeeService;
import com.app.validator.EmployeeValidator;

```

@Controller

```
public class EmployeeController {

    @Autowired

    private EmployeeValidator validator;

    @Autowired

    private IEmployeeService service;

    //1. to Show Register Page

    @GetMapping("/reg")

    public String showReg(ModelMap map) {

        map.addAttribute("employee", new Employee());

        return "EmployeeRegister";

    }

    //2. save data on click submit

    @PostMapping("/insert")

    public String saveEmp(@ModelAttribute Employee employee, Errors errors, ModelMap map) {

        validator.validate(employee, errors);

        if(errors.hasErrors()) {

            map.addAttribute("employee", employee);

        }else {

            Integer empId=service.saveEmployee(employee);

            map.addAttribute("message", "Employee '"+empId+"' created");

            //clear form

            map.addAttribute("employee", new Employee());

        }

    }

}
```

```

        return "EmployeeRegister";
    }

    //3. get Data form DB all rows
    @GetMapping("/viewAll")
    public String showData(ModelMap map) {

        List<Employee> emps=service.getAllEmployees();

        map.addAttribute("emps", emps);

        return "EmployeeData";
    }

    //4. Get Pagination Data
    @GetMapping("/viewPage")

    public String
showPageData(@PageableDefault(page=0,size=3,sort="empId",direction=Direction.ASC) Pageable
pageable,ModelMap map) {

        Page<Employee> emps=service.getAllEmployees(pageable);

        map.addAttribute("page", emps);

        return "EmployeeData";
    }
}

application.properties

#Server PORT Details

server.port=2626

#ViewResolver Details

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

```

#Connection Details

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=root

#Hibernate Details

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

#Pool Details

spring.datasource.hikari.connection-timeout=10000

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=15

spring.datasource.hikari.pool-name=HikariCP

EmployeeRegister.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Employee Register Page!!</title>

</head>


```
<body>

<h1>Welcome to Employee</h1>

<form:form action="insert" method="post" modelAttribute="employee">

<pre>

Name : <form:input path="empName"/>

<form:errors path="empName"/>

Email : <form:input path="empMail"/>

<form:errors path="empMail"/>

Gender: <form:radiobutton path="empGen" value="Male"/>Male <form:radiobutton path="empGen"
value="Female"/>Female

<form:errors path="empGen"/>

Address: <form:textarea path="empAddr"/>

<form:errors path="empAddr"/>

Langs : <form:checkbox path="empLangs" value="ENG"/>ENG

        <form:checkbox path="empLangs" value="HIN"/>HIN

        <form:checkbox path="empLangs" value="TEL"/>TEL

<form:errors path="empLangs"/>

ID   : <form:select path="empIdType">

        <form:option value="">--select--</form:option>

        <form:option value="AADHAR">AADHAR</form:option>

        <form:option value="VOTER ID">VOTER ID</form:option>

        <form:option value="PAN CARD">PAN CARD</form:option>

        <form:option value="OTHER">OTHER</form:option>

    </form:select>

<form:errors path="empIdType"/>

ID NUM: <form:input path="empIdNum"/>
```

```
<form:errors path="empIdNum"/>

<input type="submit" value="Register"/>

</pre>

</form:form>

${message}

</body>

</html>
```

EmployeeData.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Employee Data</title>

</head>

<body>

    <h1>Welcome to Employee Data Page!!</h1>

    <table border="1">

        <tr>

            <th>ID</th>

            <th>NAME</th>

            <th>EMAIL</th>

            <th>GENDER</th>
```

```

        <th>ADDRESS</th>

        <th>LANGUAGES</th>

        <th>ID</th>

        <th>NUMBER</th>

    </tr>

    <c:forEach items="${page.getContent()}" var="e">

        <tr>

            <td><c:out value="${e.empId}" /></td>

            <td><c:out value="${e.empName}" /></td>

            <td><c:out value="${e.empMail}" /></td>

            <td><c:out value="${e.empGen}" /></td>

            <td><c:out value="${e.empAddr}" /></td>

            <td><c:out value="${e.empLangs}" /></td>

            <td><c:out value="${e.empIdType}" /></td>

            <td><c:out value="${e.empIdNum}" /></td>

        </tr>

    </c:forEach>

</table>

<c:if test="${!page.isFirst()}">

    <a href="?page=0">First</a>&nbsp;&nbsp; 

</c:if>

<c:if test="${page.hasPrevious()}">

    <a href="?page=${page.getNumber()-1}">Previous</a>&nbsp;&nbsp; 

</c:if>

```

```

<c:forEach begin="0" end="{page.getTotalPages()-1}" var="i">

    <c:choose>

        <c:when test="{page.getNumber() == i}">

            {i+1}&nbsp;&nbsp; 

        </c:when>

        <c:otherwise>

            <a href="?page={i}">{i+1}</a>&nbsp;&nbsp; 

        </c:otherwise>

    </c:choose>

</c:forEach>

<c:if test="{page.hasNext()}">

    <a href="?page={page.getNumber()+1}">Next</a>&nbsp;&nbsp; 

</c:if>

<c:if test="{!page.isLast()}">

    <a href="?page={page.getTotalPages()-1}">Last</a>&nbsp;&nbsp; 

</c:if>

</body>

</html>

```

pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

```

```
<modelVersion>4.0.0</modelVersion>

<groupId>org.sathyatech</groupId>

<artifactId>SpringWebMvcApp</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<name>SpringWebMvcApp</name>

<description>MVC project for Spring Boot</description>

<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <java.version>1.8</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-test</artifactId>

        <scope>test</scope>

    </dependency>
```

```
<!-- JSTL -->
```

```
<dependency>
```

```
    <groupId>javax.servlet</groupId>
```

```
    <artifactId>jstl</artifactId>
```

```
</dependency>
```

```
<!-- To compile JSP files -->
```

```
<dependency>
```

```
    <groupId>org.apache.tomcat.embed</groupId>
```

```
    <artifactId>tomcat-embed-jasper</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
    <scope>provided</scope>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-devtools</artifactId>

        <scope>runtime</scope>
    </dependency>

    <dependency>

        <groupId>mysql</groupId>

        <artifactId>mysql-connector-java</artifactId>

        <scope>runtime</scope>
    </dependency>

    <dependency>

        <groupId>com.zaxxer</groupId>

        <artifactId>HikariCP</artifactId>
    </dependency>
</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>
    </plugins>
</build>
</project>
```

ResponseEntity as return Type of RestController method:

In application, development we should never use String, classType, Collection as return Type, which may give incomplete output.

Provide these details as ResponseEntity return Type which contains body and status.

Status can be represented using enum HttpStatus given by Spring Framework.

Possible HttpStatus numbers are:

1xx	Information
2xx	Success
3xx	redirect
4xx	Client Side error
5xx	Server Side error

****ResponseEntity<T> must have body of T type (T = DataType specified while using or coding in method)**

We can use ? symbol at method return level or declaration level only. (not at object creation (new creation level)).

? indicate type decided at runtime based on execution flow.

Code Template:

```
package com.app.controller;

@RestController

public class EmployeeProvider{

    @GetMapping("/showget")

    public ResponseEntity<?> showGetData(){

        ResponseEntity<?> resp = null;

        HttpStatus status = null;

        String message = null;

        try{

            //logic
```



```

        status = HttpStatus.ok;//200- success

        message = "Work done successfully";

        resp = new ResponseEntity<String>(message,status);

    }catch(Exception e){

        status = HttpStatus.BAD_REQUEST;

        resp= new ResponseEntity<Exception>(e,status) ;

    }

    return resp;

}
}

```

RestController Method Types:

Spring Boot Rest Webservice supports:

GetMapping – for selecting data

PostMapping – for creating new Resource

PutMapping – for modifying old resource

DeleteMapping – for removing existing resource

Code: -

```

package com.app.controller;

@RestController

public class EmployeeProvider{

    @GetMapping("/showGet")

    public ResponseEntity<String> showget(){

        return new ResponseEntity<String> ("From Get",HttpStatus.ok);

    }

    @PostMapping("/showPost")

```

```
public ResponseEntity<String> showPost(){  
    return new ResponseEntity<String> ("From Post",HttpStatus.ok);  
}  
  
@PutMapping("/showPut")  
public ResponseEntity<String> showPut(){  
    return new ResponseEntity<String> ("From Put",HttpStatus.ok);  
}  
  
@DeleteMapping("/showDelete")  
public ResponseEntity<String> showDelete(){  
    return new ResponseEntity<String> ("From Delete",HttpStatus.ok);  
}  
}
```

POSTMAN Setup and Request:

#1 Goto Chrome browser and enter Postman chrome

#2 click on Chrome extension line


#3 click on "Add to chrome" > "Add app"

#4 Once done re-start Chrome

#5 Enter address: "Chrome://apps/"

#6 Click on Postman and skip registration

#7 Enter details type URL and send

GET  http://localhost:2018/showGet		SEND
Response		Status: 200 OK
Hello from GET		

Path

Path it indicates URL to method to be accessed by client App

Paths are two types:

- a. Static path (/path)
- b. Dynamic path ({path})

Dynamic path always indicates a value in URL as an input to method.

Ex: Path: ../show/{eid}/name

1. ../show/20/ab (404 not found) – b'z name is static and we are giving dynamic.
2. ../show/20.36/name (BAD_REQUEST 400)
3. ../show/9/name (Ok 200)
4. ../show/ab/name (BAD_REQUEST 400)
5. ../show/10/10 (404 NOT FOUND)

To read value entered in path use annotation @PathVariable

Syntax:#1

@PathVariable("key")DT localVar

Syntax:#2

@PathVariable DT Key

Here,

localVar = local VariableName

Key = Dynamic Path Name

Example:

```
package com.app.provider;
```

```
@RestController
```

```
Public class EmployeeProvider{
```

```
    @GetMapping("/Show/{eid}")
```

```
    public String showOne(@PathVariable int eid){
```

```
        return "Hello::Id is"+eid;
```

```
    }
```

```
    @GetMapping("/Show/{eid}/{ename}")
```

```
    public String showTwo(@PathVariable int eid, @PathVariable String ename){
```

```
        return "Hello::Id is"+eid+ "Name is"+ename;
```

```
    }
```

```
    @GetMapping("/Show/{eid}/{ename}/{esal}")
```

```
    public String showAll(@PathVariable int eid, @PathVariable String ename, @PathVariable double esal){
```

```
        return "Hello::Id is"+eid+ "Name is"+ename+"esal"+esal;
```

```
    }
```

```
    @GetMapping("/Show/eid/ename/esal")
```

```
    public String showNone(@PathVariable int eid, @PathVariable String ename){
```


```
        return "Hello::None";
```

```
    }
```

```
}
```

URL: <http://localhost:2018/show/eid> (In postman GET method)

****Key and variable name both should be same or different**

GET  http://localhost:2018/show/10		SEND
Response		Status: 200 OK
Hello:10		

Http-Status-404: URL case/spell is wrong or not matched with any path.

Ex: actual one = show

Entered One = /SHOW, /Show, /Showo, /Shows

Http-Status-405: method not allowed

If method is GET type and request in non-GET (ex: POST) then this error will occur.

Http-Status-400: BAD_REQUEST

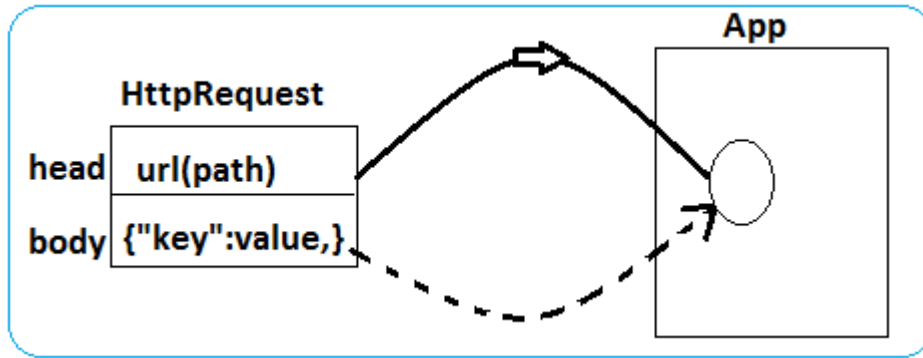
Datatype mismatch for input like PathVariable JSON input, etc.

@RequestBody:

It is used to convert input data (JSON) provided in HttpRequest Body and will be converted to object format for request method.

****GET will not support this operation only PUT, DELETE and POST also supports.**

Flow:



Example:

```
package com.app.model;
```

```
public class Employee{  
    private int empId;  
    private String empName;  
    private double empSal;  
    //def const, set-get, toString  
}
```

```
package com.app.provider;
```

```
@RestController
```

```
public class EmployeeProvider{  
    @PostMapping("/show")  
    public String showData(@RequestBody Employee emp){  
        return "Hello:"+emp.getEmpId()+","+emp.getEmpName+","+emp.getEmpSal();  
    }  
}
```

(1) POST	(2) http://localhost:2018/show	(6) <input type="button" value="SEND"/>
(3) <input checked="" type="radio"/> Body		
(4) <input checked="" type="radio"/> raw		(5) JSON (app/json)
O/p {"empId":10,"empName":"AB", "empSal":34.33}		

Example: SpringReSTMySQL

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example.app</groupId>

    <artifactId>Demo</artifactId>

    <version>1.0</version>

    <packaging>jar</packaging>

    <name>SpringRestMySQL</name>

    <description>Demo project for Spring Boot</description>

    <parent>

        <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.2.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <java.version>1.8</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-data-jpa</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-devtools</artifactId>

        <scope>runtime</scope>

    </dependency>

    <dependency>

        <groupId>mysql</groupId>
```



```
        <artifactId>mysql-connector-java</artifactId>

        <scope>runtime</scope>
    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-test</artifactId>

        <scope>test</scope>
    </dependency>

    <dependency>

        <groupId>com.zaxxer</groupId>

        <artifactId>HikariCP</artifactId>
    </dependency>

    <dependency>

        <groupId>io.springfox</groupId>

        <artifactId>springfox-swagger2</artifactId>

        <version>2.7.0</version>
    </dependency>

    <dependency>

        <groupId>io.springfox</groupId>

        <artifactId>springfox-swagger-ui</artifactId>

        <version>2.7.0</version>
    </dependency>
</dependencies>

<build>

    <plugins>
```

```
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
</project>
```

application.properties

#Server Details

server.port=2525

#Connection Details(DataSource)

#spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=root

Data Source - CP

spring.datasource.hikari.connection-timeout=10000

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=15

spring.datasource.hikari.pool-name=HikariCP

#Hibernate(JPA) Details

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

SpringRestMySQLApplication.java

```
package com.example.demo;

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

@SpringBootApplication

public class SpringRestMySQLApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringRestMySQLApplication.class, args);

    }

}
```

Employee.java

```
package com.example.demo.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity

@Table(name="emptab")

public class Employee {

    @Id

    @Column(name="eid")

    private Integer empld;

    @Column(name="ename")

    private String empName;

    @Column(name="esal")
```

```
private Double empSal;
```

```
public Employee() {
```

```
    super();
```

```
}
```

```
public Employee(Integer empId, String empName, Double empSal) {
```

```
    super();
```

```
    this.empId = empId;
```

```
    this.empName = empName;
```

```
    this.empSal = empSal;
```

```
}
```

```
public Integer getEmpId() {
```

```
    return empId;
```

```
}
```

```
public void setEmpId(Integer empId) {
```

```
    this.empId = empId;
```

```
}
```

```
public String getEmpName() {
```

```
    return empName;
```

```
}
```

```
public void setEmpName(String empName) {
```

```
    this.empName = empName;
```

```
}
```

```
public Double getEmpSal() {
```

```
    return empSal;
```

```

    }

    public void setEmpSal(Double empSal) {

        this.empSal = empSal;

    }

    @Override

    public String toString() {

        return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + "]\n";

    }

}

```

EmployeeRepository.java

```

package com.example.demo.repo;

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

import com.example.demo.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

```

IEmployeeService.java

```

package com.example.demo.service;

import java.util.List;

import com.example.demo.model.Employee;

public interface IEmployeeService {

    public Integer saveEmployee(Employee emp);

    public Boolean isEmployeeExistById(Integer empId);

    public void deleteEmployee(Integer empId);

    public Employee getEmployeeById(Integer empId);

    public List<Employee> getAllEmployees();}

```

EmployeeServiceImpl.java

```
package com.example.demo.service;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import com.example.demo.model.Employee;

import com.example.demo.repo.EmployeeRepository;

@Service

public class EmployeeServiceImpl implements IEmployeeService{

    @Autowired

    private EmployeeRepository repo;

    @Override

    public Integer saveEmployee(Employee emp) {

        return repo.save(emp).getEmpId();

    }

    @Override

    public void deleteEmployee(Integer empId) {

        repo.deleteById(empId);

    }

    @Override

    public Employee getEmployeeById(Integer empId) {

        Optional<Employee> empOp=repo.findById(empId);

        if(empOp.isPresent())
```

```

        return empOp.get();
    }
    else
    {
        return null;
    }
}

@Override

public List<Employee> getAllEmployees() {

    List<Employee> emps=repo.findAll(Sort.by("empId"));

    /*Collections.sort(emps,new Comparator<Employee>() {

        @Override

        public int compare(Employee o1, Employee o2) {

            return o1.getEmpId()-o2.getEmpId();

        }

    });*/

    //Collections.sort(emps,(o1,o2)->o1.getEmpId()-o2.getEmpId());

    return emps;

}

@Override

public Boolean isEmployeeExistById(Integer empId) {

    return repo.existsById(empId);

}

}

```

SwaggerConfig.java

```

package com.example.demo.config;

import static springfox.documentation.builders.PathSelectors.regex;

import static springfox.documentation.builders.RequestHandlerSelectors.basePackage;

```

```
import java.util.ArrayList;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.service.VendorExtension;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2

public class SwaggerConfig {

    @Bean

    public Docket createDocket() {

        return new Docket(DocumentationType.SWAGGER_2)

            .select()

            .apis(basePackage("com.example.demo.controller"))

            .paths(regex("/rest.*"))

            .build()

            .apiInfo(apiInfo());

    }

    public ApiInfo apiInfo() {

        return new ApiInfo(

            "Spring Boot Employee App",
```



```
        "welcome to my Application",  
        "Snap-1.0", "https://sathyatech.com/",  
        new Contact(  
            "RSS",  
            "http://sathyatech.com",  
            "javabyraghu@gmail.com"  
        ),  
        "Apache XYZ", "https://sathyatech.com/", new  
        ArrayList<VendorExtension>());  
    }  
}
```

EmployeeRestController.java

```
package com.example.demo.controller;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import org.springframework.http.HttpStatus;  
  
import org.springframework.http.ResponseEntity;  
  
import org.springframework.web.bind.annotation.DeleteMapping;  
  
import org.springframework.web.bind.annotation.GetMapping;  
  
import org.springframework.web.bind.annotation.PathVariable;  
  
import org.springframework.web.bind.annotation.PostMapping;  
  
import org.springframework.web.bind.annotation.RequestBody;  
  
import org.springframework.web.bind.annotation.RequestMapping;  
  
import org.springframework.web.bind.annotation.RestController;  
  
import com.example.demo.model.Employee;  
  
import com.example.demo.service.IEmployeeService;
```

```
@RestController

@RequestMapping("/rest/emp")

public class EmployeeRestController {

    @Autowired

    private IEmployeeService service;

    //1. save data

    @PostMapping("/save")

    public ResponseEntity<?> saveEmployee(@RequestBody Employee emp){

        String message=null;

        ResponseEntity<String> response=null;

        try {

            Integer empId=service.saveEmployee(emp);

            message="Employee '"+empId+"' Saved";

            response=new ResponseEntity<String>(message,HttpStatus.OK);

        } catch (Exception e) {

            message="Unable to save Employee:Reason"+e.getMessage();

            response=new

ResponseEntity<String>(message,HttpStatus.INTERNAL_SERVER_ERROR);

            e.printStackTrace();

        }

        return response;

    }

    @DeleteMapping("/delete/{empId}")

    public ResponseEntity<String> deleteEmployee(@PathVariable Integer empId){

        String message=null;

        ResponseEntity<String> response=null;
```

```

        try {

            if (empId==null) {

                message="Invalid Input Provided";

                response=new
ResponseEntity<String>(message,HttpStatus.BAD_REQUEST);

            }else if(service.isEmployeeExistById(empId)) {

                service.deleteEmployee(empId);

                message="Employee '"+empId+"' deleted";

                response=new ResponseEntity<String>(message,HttpStatus.OK);

            }else {

                message="Employee '"+empId+"' Not exist";

                response=new
ResponseEntity<String>(message,HttpStatus.NOT_FOUND);

            }

        } catch (Exception e) {

            message="Unable to delete Employee:Reason"+e.getMessage();

            response=new
ResponseEntity<String>(message,HttpStatus.INTERNAL_SERVER_ERROR);

            e.printStackTrace();

        }

        return response;

    }

    @GetMapping("/all")

    public ResponseEntity<?> getAllEmployees(){

        String message=null;

        ResponseEntity<?> response=null;

        try {

```

```
        List<Employee> emps=service.getAllEmployees();

        if(emps==null || emps.size()==0) {

            message="No Data Found";

            response=new
ResponseEntity<String>(message,HttpStatus.NO_CONTENT);

        }else {

            response=new ResponseEntity<List<Employee>>(emps,HttpStatus.OK);

        }

    } catch (Exception e) {

        message="Problem in Fetching data : reason is :"+e.getMessage();

        response=new
ResponseEntity<String>(message,HttpStatus.INTERNAL_SERVER_ERROR);

        e.printStackTrace();

    }

    return response;

}

}
```

Build and Deployment Process:

Build: It is a process of converting our java code to final executable format after compiling all java files.

Here final format means .jar/.war

Deployment: It is a process of placing a jar/war, build in server and starting it.

Note:

#1 In application final build will be placed in target folder.

#2 To build application, we must add <build><plugins>...</plugins></build> in pom.xml file at bottom.

#3 After development, execute maven goals “**maven clean**”, “**maven build**”, “**maven test**” (for build testing). OR all in one “**maven install**”

#4 Set JDK before maven install:

Steps:

>window>preferences>installed JRE>add>brows>choose JDK location

Ex: “c:/Program files/java/jdk 1.8”

>make it as default ok

#5 Update JDK to Project:

>Right click on project > Build Path>configure build path > Choose JRE System library> Edit > Choose workspace default> Apply and close.

#6 Now execute **maven install**:

>Right click on project> Run as

> maven install (choose this goal)

#7 Refresh Project to see generated file under target folder

>Right click on project > Refresh > (F5)

Search for - > __.jar file

#8 goto target location to start server in local machine:

>Right click on target > properties

>Copy location and open in MyComputer

>Shift Right click

>Choose “open cmd” window here

>Type “java – jar Demo-0.0.1-SNAP.jar”

>Enter> goto Browser and enter URL

application.properties:-

server.port=2020

spring.application.name=employee // (this is only temporary name)-optional-not case-sensitive

server.servlet.context-path=/myapp // (this is our main app path that we write in browser)-**case-sensitive (default context-path is /)**

Provider class:

```
package com.app.provider;
```

```
@RestController
```

```
@RequestMapping("/rest/emp") // (this is common path for all method)
```

```
public class EmployeeProvider{
```

```
    @GetMapping("/show")
```

```
    public String showMsg(){
```

```
        return "Hello";
```

```
    }
```

```
}
```

*Here @RequestMapping is used to provide common path for all method if we write at class level.

*default context path provided by Spring Boot is “[NAN]”.So it is mapped to “/”

* Every context path must start with “/”

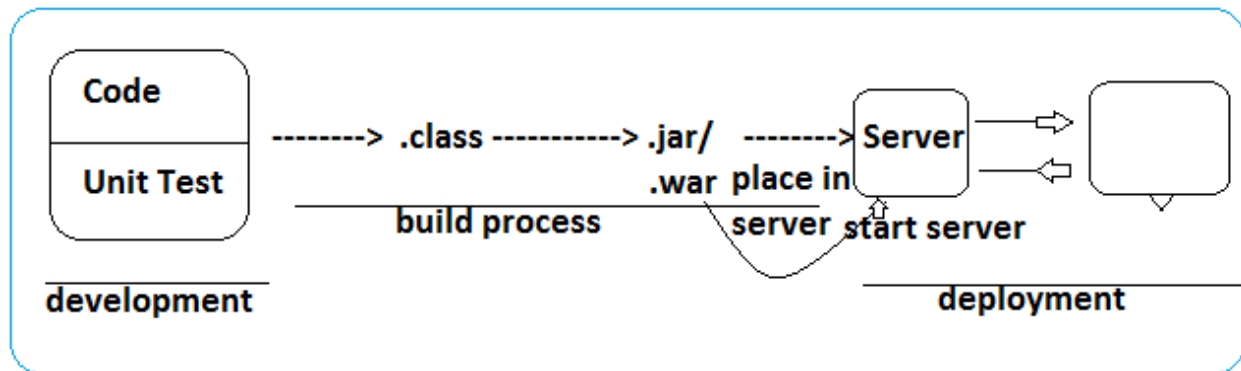
* Application name is used for **Cloud Environment** for **inter communication**.

URL:

<http://localhost:2020/myapp/rest/emp/show>

****To stop server** running using cmd prompt use “**ctrl+c**” [cancel process]

We can run our application in browser or postman



Spring Boot Embedded Component:

Embedded Server: (2)

1. Apache Tomcat (default)
2. Jetty

Embedded Database: (3)

1. H2
 2. HSQL
 3. Apache Derby
- Also support different database like mysql, oracle, postgres, etc. with default **connection pooling** "tomcat server connection pooling" also **supports 3rd party** connection pooling apache DBCP, Hikary.
 - Also supports "**caching**" to reduce round network calls using "**EnCache**" or 3rd party like "**jCache**", "**redis**", etc.

Spring Boot Data JPA:

To perform DB operations, use **Spring Boot Data JPA API** which will write all operations code for programmer.

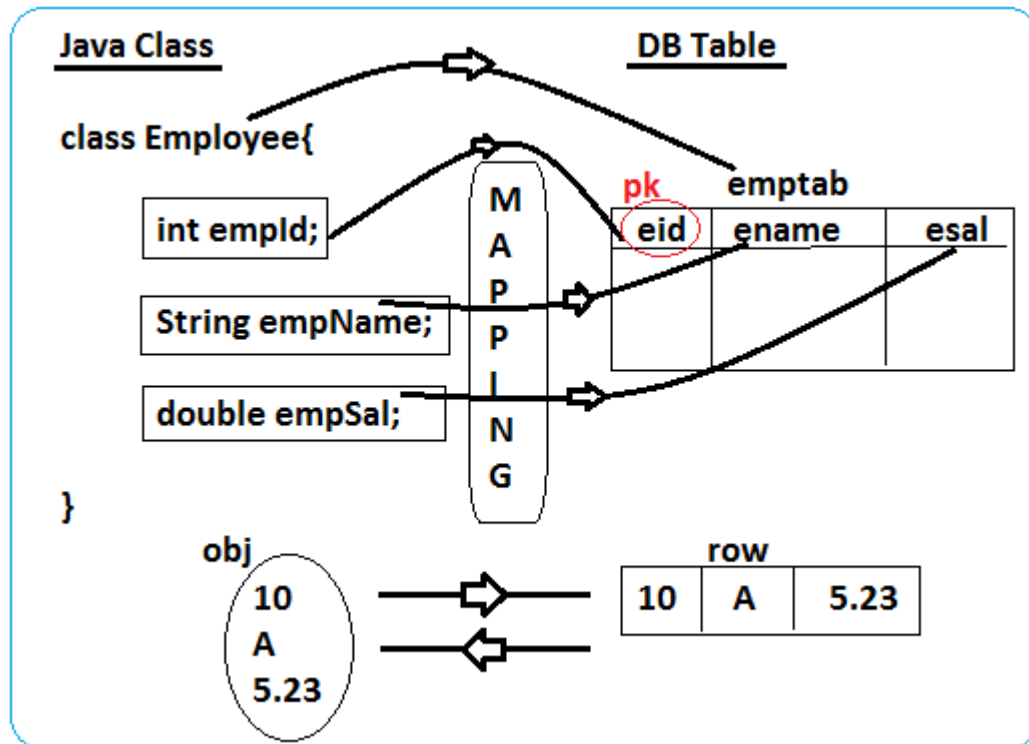
We should define only:

1. Model class/ Entity Class
2. Repository Interface and Add Spring-boot-starter-data-jpa dependency in pom.xml

****Spring Boot auto generates code for Hibernate configuration, Connection Creation, with pooling, basic operations code, pagination, specification, sorting HQL's, Multiplicity, Connection Mapping, etc.**

#1 Model Class:

A class mapped with DB table using JPA Annotations [`@Entity` `@Table` `@Column` `@Id`] is called as "Model Class or Entity Class". It follows ORM (Object Relational Mapping)



#1 Create one Spring Boot Project with dependencies **web, H2, JPA**

>file > new >spring Boot starter project

>Enter project Name> Next

>Choose dependencies

Web, Jpa, H2 > finish

#2 Define Model class in base package

Ex:

```
package com.app.model
```

```
@Entity
```

```
@Table(name="emptab")
```

```
public class Employee{
```



```

    @Id

    @Column(name="eid")

    private Integer empld;

    @Column(name="ename")

    private String empName;

    @Column(name="esal")

    private Double empSal;

    //const, p.const

    //set get

    //toString
}

```

#3 Define Repository Interface which provides all basic operations.

```

package com.app.repo;

public interface EmployeeRepository extends JpaRepository<Employee,Integer>{

}

```

#4 Enable JpaRepository in Spring Boot starter class also implements CommandLineRunner run() method, execute one by one operations.

```

package com.app;

@SpringBootApplication

@EnableJpaRepositories

public class SpringDBOperation implements CommandLineRunner{

    public static void main(String[] args){

        SpringApplication.run(SpringDBOperation.class,args);

    }

    @Autowired

    private EmployeeRepository repo;

```

```

public void run(String...args) throws Exception{

    //1. Save operation

    Employee e1 = new Employee(10, "A", 3.2);

    repo.save(e1);

    //2. Update Operation

    e1.setEmpName("JJ");

    repo.save(e1); // update (if we want to update only this then we write Annotation in
model class at class level)

    //3. Save multiple

    List<Employee> emps = Arrays.asList(new Employee(20,"B",2.2), new
Employee(30,"C",4.2), new Employee(40,"DD",5.2));

    repo.saveAll(emps);

    //4. Display All
    List<Employee> allEmps = repo.findAll();
    // jdk 1.5 (for each loop)
    for(Employee e: allEmps){
        Sysout(e);
    }
    // jdk 1.8 (lamda expression)
    allEmps.forEach((e)->Sysout(e));

    //5. Check exist or not
    boolean exist = repo.existByld(40);
    Sysout(exist);

    //6. Select one
    Employee e2 = repo.getOne(40);
    Sysout(e2);

    //7. Select multiple (not all rows)
    List<Employee> fewEmps = repo.findAllByld(Arrays.asList(20,30,40));
    fewEmps.forEach((e)->Sysout(e));

    //8. Delete one
    repo.deleteByld(50);

```

```

        //9. Delete All
        repo.deleteAll();
    }
}

```

application.properties

server.port=2525

spring.jpa.show-sql=true

Spring Data JPA using MySQL Database:

Step#1 Create Spring Boot Starter project using web and Jpa dependencies

Step#2 Add MySQL dependency tag under <dependency> without version

```

<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

</dependency>

```

Step#3 Provide Datasource (connection) and JPA (Hibernate) details in application.properties file

****Driver class name is optional, it will be auto-detected by dependency and URL**

--application.properties

#server details

server.port=2525

#Connection Details (DataSource)

spring.datasource.driver-class-name=com.mysql.jdbc.driver

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=root

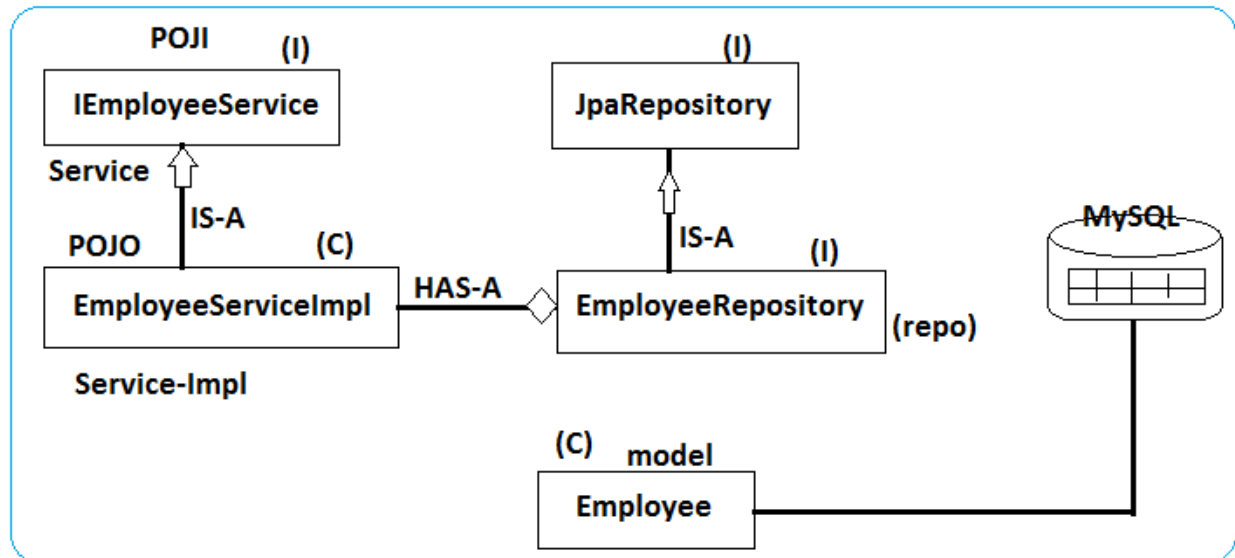
#Hibernate (JPA) details

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**Dialect also auto detected based on URL and driver class

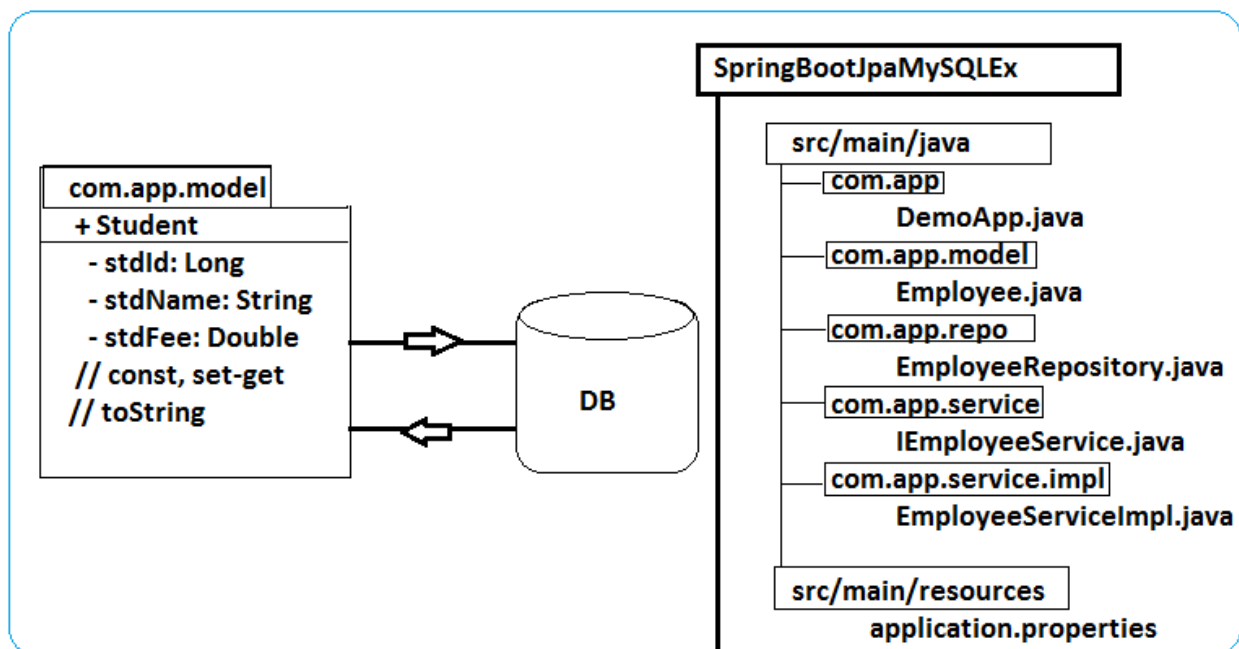
****Dialect** is a class it will generate SQL queries.



Service Layer is used to define Application logic, Transaction Management, Data Sorting, etc.

It follows POJO-POJI design pattern for loosely coupling.

Ex: Home work



Spring Boot Data JPA Query Methods:

1. findBy()
2. @Query

#1 findBy():

Spring Boot Data Jpa has provided default methods findById and findAll methods which are used to fetch data based on Primary key columns only.

To get Data based on non-primary key column define our own method using findBy() syntax, which also supports conditions and Joins like “and, or, in, isNull”.

findBySpringBoot.pdf on thejavatemple fb group

These methods must be added in Repository interface as abstract methods, body(logic) will be provided by Spring Boot Data Jpa only.

Example Methods with repo Code:

```
package com.app.repo;
```

```
public interface EmployeeRepository extends JpaRepository<Employee,Integer>{
```

```
    //select * from emptab where ename=?
```

```
    Employee findByEmpName(String empName);
```

```
    //select * from emptab where esal=?
```

```
    Employee findByEmpSal(Double empSal)
```

```
    //select * from emptab where ename=? and esal=?
```

```
    Employee findByEmpNameAndEmpSal(String empName,Double empSal);
```

```
    //select * from emptab where eid between ? and ?
```

```
    List<Employee> findByEmpIdBetween(Integer empId,Integer empId2);
```

```
    //select * from emptab where ename like ?
```

```
    List<Employee> findByEmpNameLike(String empName);
```

}

#2 @Query Methods in Jpa Repository

This is used to execute on HQL written manually by programmer. We can specify customized query here which stores final result to method return Type.

HQL: Hibernate Query Language

To convert SQL to HQL Format:

Table name -> replace with -> class Name

Column name -> replace with -> variable Name

SQL: select eid from emptab where ename=?

HQL: select empId from com.app.Employee where empName=?

Write Code in Repository Interface:

#1 **@Query**("select count(empId) from com.example.demo.model.Employee")

Long countTotalRowsInEmp();

#2 **@Query**("select max(empSal) from com.example.demo.model.Employee")

Double getMaxSalfromEmployees();

#3 **@Query**("select empName, empSal from com.example.demo.model.Employee where empSal=?1 and empSal=?2")

List<Object[]> getEmpsBySalName(Double empSal, String empName)

? -> Positional Parameter

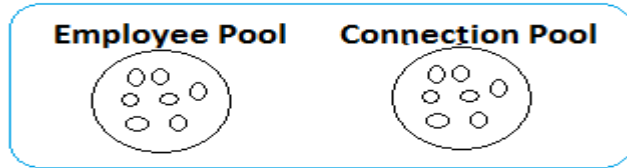
Temporary Memory Management in Spring Boot: - Database Operation

Spring Boot supports two types of temporary memory.

1. Pool
2. Cache

#1 Pool:

It is a group of similar objects that is all objects related to one class type.



Connection Pool means group of readymade connection objects which provides faster execution for database operations. Every Spring Boot application must be configured with one connection pool concept for database programming.

Spring Boot supports multiple connection pooling those are:

1. Hikary CP (for realtime)
2. Tomcat CP
3. Apache DBCP 2.x

#1 **Hikary CP** is a default connection pool used by Spring Boot Data which works faster than other connection pool.

#2 Tomcat connection pool works only with tomcat server, if server changed then it will not work.

#3 Apache DBCP 2.x is a 3rd party service provided by apache for basic connection pool concept. It is recommended to use in development environment not in production environment.

#2 Cache:

It is a group of dissimilar objects.

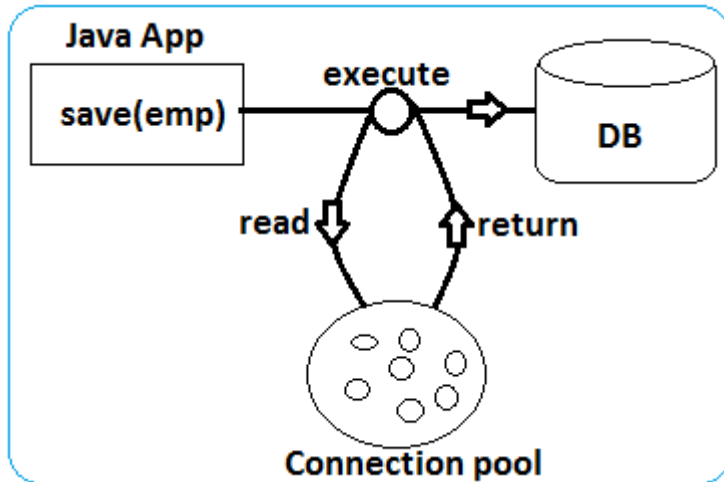
Steps in connection pooling:

#1 Provide Connection Pooling Dependency in pom.xml

#2 Provide pool properties like pool size, timeout, maxsize, ideal time, etc.

#3 On server startup connection pool will be created with initial size.

#4 On performing operations every connection pool object will be called (based on load) and return back to pool.



- Read connection from pool
- Execute operation
- Return connection back to CP

#5 On server shutdown it will close all connection objects that is pool is closed or shutdown.

Hikary CP setup:

#1 In pom.xml add dependency

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikaryCP</artifactId>
</dependency>
```

Every Property Looks like:

spring.datasource.hikari.*=

For Tomcat CP:

spring.datasource.tomcat.*=

For Apache DBCP 2.x:

spring.datasource.dbcp2.*=

application.properties:

spring.datasource.hikari.connection-timeout=10000

spring.datasource.hikary.minimum-idle=15

spring.datasource.hikari.pool-name=HikariCP

#default timeout=6000 sec

#init-size=10, 1 conn=100 operations

Q. How to know which CP is used in Application?

-> Use Autowired for datasource (javax.sql) in any class and print it is reference.

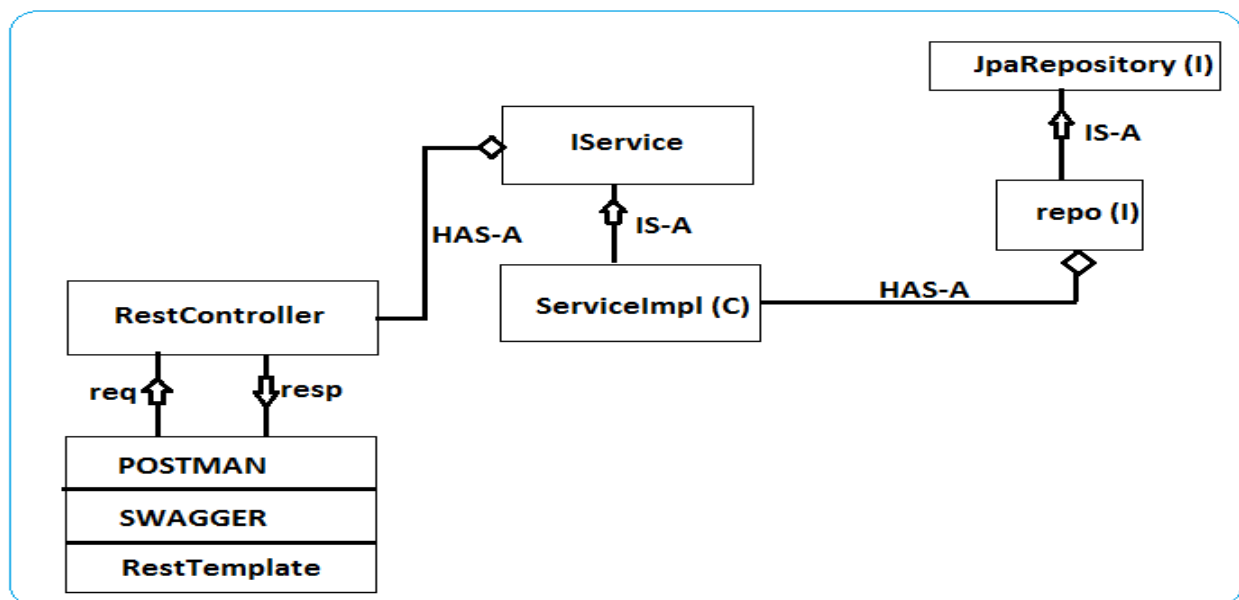
Ex: @Autowired

private Datasource datasource in run() method

 Sysout(datasource)

Spring Boot Provider Application End-to-End:

1. Model class – Employee
2. Repository (I) – EmployeeRepository
3. IService (I) – IEmployeeService
4. ServiceImpl – EmployeeServiceImpl
5. Controller – EmployeeController
6. SpringBootTestClass
7. application.properties
8. pom.xml



Swagger UI Docket:

Swagger is a 3rd party UI tool. It provides easy UI for testing of provider application. To create this UI we need to follow below steps:

Step#1 create Docket with Documentation Type.

Step#2 Select API's having basePackages.

Step#3 Provide path in regular expression.

Step#4 Optional

Provide Meta data also called as API info.

Step#5 Build docket and returns as it is.

All above steps must be provided as “**SwaggerUI**” config class in Spring Boot.

It should be enabled using “**@EnableSwagger2**” annotation.

*Also add **devTool** dependency to perform dev operation like (add **swagger** dependency)

.java→(compile)→.class →build →.jar/.war→(deploy)→place in server and start (without stopping appln) (If we want to change something in appln that we do not close the appln)

To enable this add below dependency in pom.xml

```
<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-devtools</artifactId>

    <scope>runtime</scope>

</dependency>
```

**After Starting application: enter URL like: <http://localhost:2525/swagger-ui.html>

Code:

```
package com.app.config;
```

```
@Configuration
```

```
@EnableSwagger2
```

```
public class SwaggerConfig() {
```

@Bean

```
public Docket createDocket() {  
  
    return new Docket( DocumentationType.SWAGGER_2)  
  
        .select()  
  
        .apis(basePackage("com.app.controller"))  
  
        .path(regex("/rest.*"))  
  
        .build();  
  
}
```

basePackage in -> Request

End-to-End Web application Design using Spring Boot:

By using Spring WEB MVC and Spring Boot services and Data Jpa with other utility classes like spec, validator, util, etc.

We can implement one End-to-End Web application.

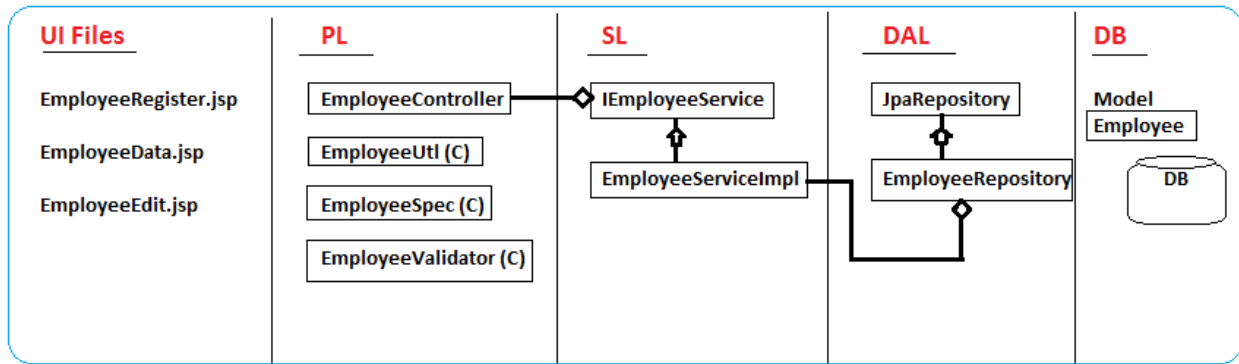
UI Technologies:

SpringUI form tags + Jsp + JSTL + EL + BootStrap

To implement this, we are using layers, loosely coupled model using HAS-A with POJI-POJO.

Consider Example module as Employee then all files are:

1. Model class – Employee
2. Repo – EmployeeRepository
3. IService – IEmployeeService
4. ServiceImpl – EmployeeServiceImpl
5. Controller – EmployeeController
6. Util – EmployeeUtil
7. Specification – EmployeeSpec
8. Validator – EmployeeValidator
9. UI Files – EmployeeRegister, EmployeeData, EmployeeEdit



Name:

Email:

Gender:

☐ Male
☐ Female

Address:

Lang:

☐ EN
☐ HI
☐ MA

Id Type:

--select--

▼

Id Number:

CREATE EMP

Example: SpringBootWebMvcCURDSpecPagination Application

pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.sathyatech</groupId>

```

```
<artifactId>SpringWebMvcApp</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<name>SpringWebMvcApp</name>

<description>MVC project for Spring Boot</description>

<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <java.version>1.8</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-test</artifactId>

        <scope>test</scope>
```

```
</dependency>

<!-- JSTL -->

<dependency>

    <groupId>javax.servlet</groupId>

    <artifactId>jstl</artifactId>

</dependency>

<!-- To compile JSP files -->

<dependency>

    <groupId>org.apache.tomcat.embed</groupId>

    <artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-tomcat</artifactId>

    <scope>provided</scope>

</dependency>

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>
```

```
<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-devtools</artifactId>

    <scope>runtime</scope>

</dependency>

<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <scope>runtime</scope>

</dependency>


<dependency>

    <groupId>com.zaxxer</groupId>

    <artifactId>HikariCP</artifactId>

</dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>

</project>
```

application.properties

#Server PORT Details

server.port=2626

#ViewResolver Details

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

#Connection Details

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/boot

spring.datasource.username=root

spring.datasource.password=root

#Hibernate Details

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

#Pool Details

spring.datasource.hikari.connection-timeout=10000

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=15

spring.datasource.hikari.pool-name=HikariCP

banner.txt

```

  ____  _  _  _
 /  _ \| | | |
\`--. _ \| | | | _ _ _ _
  \-. \/_ \| | | _ \| | | / _ \|
 ^/_/ \| | | | | | | | | | | | |
 \_/_ \| | | | | | | | | | | | |
      _/ |
      |_/
```

ServletInitializer.java

```
package com.app;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(SpringWebMvcAppApplication.class);
    }

}
```

SpringWebMvcAppApplication.java

```
package com.app;

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

```
@SpringBootApplication

public class SpringWebMvcAppApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringWebMvcAppApplication.class, args);

    }

}
```

Employee.java

```
package com.app.model;

import java.util.List;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OrderColumn;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;

@Entity

@Table(name="emptab")

public class Employee {

    @Id

    @Column(name="eid")

    @GeneratedValue(generator="empgen")
```

```

@GenericGenerator(name="empgen",strategy="increment")

private Integer empld;

@Column(name="ename")

private String empName;

@Column(name="email")

private String empMail;

@Column(name="egen")

private String empGen;

@Column(name="addr")

private String empAddr;

@ElementCollection

@CollectionTable(

        name="emplangstab", //child table

        joinColumns=@JoinColumn(name="eidFk") //key column

)

@Column(name="langs") //element column

@OrderColumn(name="pos") //index column

private List<String> empLangs;

@Column(name="idType")

private String empldType;

@Column(name="idnum")

private String empldNum;

//----const & methods-----

//alt+shift+S,O(De-select-all>OK)

public Employee() {

```

```
        super();
    }

    public Employee(Integer empId) {

        super();

        this.empId = empId;

    }

    public Employee(Integer empId, String empName, String empMail, String empGen, String
empAddr, List<String> empLangs,

        String empIdType, String empIdNum) {

        super();

        this.empId = empId;

        this.empName = empName;

        this.empMail = empMail;

        this.empGen = empGen;

        this.empAddr = empAddr;

        this.empLangs = empLangs;

        this.empIdType = empIdType;

        this.empIdNum = empIdNum;

    }

    //alt+shift+S,R (selectAll>OK)

    public Integer getEmpId() {

        return empId;

    }

    public void setEmpId(Integer empId) {

        this.empId = empId;

    }
}
```

```
public String getEmpName() {  
    return empName;  
}  
  
public void setEmpName(String empName) {  
    this.empName = empName;  
}  
  
public String getEmpMail() {  
    return empMail;  
}  
  
public void setEmpMail(String empMail) {  
    this.empMail = empMail;  
}  
  
public String getEmpGen() {  
    return empGen;  
}  
  
public void setEmpGen(String empGen) {  
    this.empGen = empGen;  
}  
  
public String getEmpAddr() {  
    return empAddr;  
}  
  
public void setEmpAddr(String empAddr) {  
    this.empAddr = empAddr;  
}  
  
public List<String> getEmpLangs() {
```

```
        return empLangs;
    }

    public void setEmpLangs(List<String> empLangs) {

        this.empLangs = empLangs;
    }

    public String getEmpIdType() {

        return empIdType;
    }

    public void setEmpIdType(String empIdType) {

        this.empIdType = empIdType;
    }

    public String getEmpIdNum() {

        return empIdNum;
    }

    public void setEmpIdNum(String empIdNum) {

        this.empIdNum = empIdNum;
    }

    //alt+shift+S,S(OK)

    @Override

    public String toString() {

        return "Employee [empId=" + empId + ", empName=" + empName + ", empMail=" +
empMail + ", empGen=" + empGen

        + ", empAddr=" + empAddr + ", empLangs=" + empLangs + ",
empIdType=" + empIdType + ", empIdNum="

        + empIdNum + "]";
    }
}
```

```
}
```

EmployeeRepository.java

```
package com.app.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import com.app.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee,
Integer>,JpaSpecificationExecutor<Employee>{

}
```

EmployeeSpec.java

```
package com.app.spec;

import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import org.springframework.data.jpa.domain.Specification;
import com.app.model.Employee;

public class EmployeeSpec implements Specification<Employee>{

    //filter -> search screen data

    private Employee filter;

    public EmployeeSpec(Employee filter) {

        this.filter = filter;

    }

}
```

@Override

```
public Predicate toPredicate(Root<Employee> root, CriteriaQuery<?> cq, CriteriaBuilder cb) {

    Predicate p=cb.conjunction(); //and

    //cb.disjunction() ; //or

    if(filter.getEmpId()!=null && filter.getEmpId()>0) {

        p.getExpressions()

        .add(

            cb.equal(root.get("empId"), filter.getEmpId())

        );

    }

    if(filter.getEmpName()!=null && !"".equals(filter.getEmpName().trim())) {

        p.getExpressions()

        .add(

            cb.like(root.get("empName").as(String.class),

"%"+filter.getEmpName()+"%")

        );

    }

    if(filter.getEmpMail()!=null && !"".equals(filter.getEmpMail().trim())) {

        p.getExpressions()

        .add(

            cb.like(root.get("empMail").as(String.class),

"%"+filter.getEmpMail()+"%")

        );

    }

    return p;

}
```



```
}
```

EmployeeValidator.java

```
package com.app.validator;

import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.model.Employee;

@Component
public class EmployeeValidator implements Validator{

    @Override
    public boolean supports(Class<?> clazz) {

        return Employee.class.equals(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {

        Employee e=(Employee)target;

        if(e.getEmpName()==null || "".equals(e.getEmpName().trim())){

            errors.rejectValue("empName",null, "Please enter Employee Name!!");
        }

        /*if(e.getEmpMail()==null || "".equals(e.getEmpMail().trim())) {*/

        if(StringUtils.isEmpty(e.getEmpMail()) ||
        StringUtils.containsWhitespace(e.getEmpMail())) {

            errors.rejectValue("empMail",null, "Please enter Employee Mail Id!!");
        }
    }
}
```

```

        if(StringUtils.isEmpty(e.getEmpGen()) ||
        StringUtils.containsWhitespace(e.getEmpGen())) {

            errors.rejectValue("empGen",null, "Please choose one Gender !!");

        }

        if(StringUtils.isEmpty(e.getEmpAddr()) ||
        StringUtils.containsWhitespace(e.getEmpAddr())) {

            errors.rejectValue("empAddr",null, "Please enter Employee Address !!");

        }

        if(e.getEmpLangs()==null || e.getEmpLangs().isEmpty()) {

            errors.rejectValue("empLangs",null, "Please choose at least one Language !!");

        }

        if(StringUtils.isEmpty(e.getEmpIdType())) {

            errors.rejectValue("empIdType",null, "Please Choose one Type !!");

        }

        if(StringUtils.isEmpty(e.getEmpIdNum()) ||
        StringUtils.containsWhitespace(e.getEmpIdNum())) {

            errors.rejectValue("empIdNum",null, "Please Enter ID Number !!");

        }

    }

}

```

IEmployeeService.java

```

package com.app.service;

import java.util.List;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.domain.Specification;

import com.app.model.Employee;

```

```
public interface IEmployeeService {

    public Integer saveEmployee(Employee emp);

    public void deleteEmployee(Integer empId);

    public boolean isEmployeeExistById(Integer empId);

    public Employee getEmployeeById(Integer empId);

    public List<Employee> getAllEmployees();

    //special methods

    public Page<Employee> getAllEmployees(Pageable pageable);

    public Page<Employee> getAllEmployees(Specification<Employee> spec,Pageable pageable);

}
```

EmployeeServiceImpl.java

```
package com.app.service.impl;

import java.util.Collections;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.jpa.domain.Specification;

import org.springframework.stereotype.Service;

import com.app.model.Employee;

import com.app.repo.EmployeeRepository;

import com.app.service.IEmployeeService;

@Service
```

```
public class EmployeeServiceImpl implements IEmployeeService{
```

```
    @Autowired
```

```
    private EmployeeRepository repo;
```

```
    @Override
```

```
    public Integer saveEmployee(Employee emp) {
```

```
        return repo.save(emp).getEmpId();
```

```
    }
```

```
    @Override
```

```
    public void deleteEmployee(Integer empId) {
```

```
        repo.deleteByld(empId);
```

```
    }
```

```
    @Override
```

```
    public boolean isEmployeeExistByld(Integer empId) {
```

```
        return repo.existsByld(empId);
```

```
    }
```

```
    @Override
```

```
    public Employee getEmployeeByld(Integer empId) {
```

```
        Optional<Employee> emp=repo.findByld(empId);
```

```
        if(emp.isPresent())
```

```
            return emp.get();
```

```
        else
```

```
            return null;
```

```
    }
```

```
    @Override
```

```
    public List<Employee> getAllEmployees() {
```

```

        List<Employee> emps=repo.findAll();

        Collections.sort(emps, (e1,e2)->e1.getEmpId()-e2.getEmpId());

        return emps;
    }

    //special methods

    @Override

    public Page<Employee> getAllEmployees(Pageable pageable) {

        return repo.findAll(pageable);

    }

    @Override

    public Page<Employee> getAllEmployees(Specification<Employee> spec, Pageable pageable) {

        return repo.findAll(spec, pageable);

    }

}

```

EmployeeController.java

```

package com.app.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort.Direction;

import org.springframework.data.jpa.domain.Specification;

import org.springframework.data.web.PageableDefault;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

```

```
import org.springframework.validation.Errors;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import com.app.model.Employee;

import com.app.service.IEmployeeService;

import com.app.spec.EmployeeSpec;

import com.app.validator.EmployeeValidator;

@Controller

public class EmployeeController {

    @Autowired

    private EmployeeValidator validator;

    @Autowired

    private IEmployeeService service;

    //1. to Show Register Page

    @GetMapping("/reg")

    public String showReg(ModelMap map) {

        map.addAttribute("employee", new Employee());

        return "EmployeeRegister";

    }

    //2. save data on click submit

    @PostMapping("/insert")

    public String saveEmp(@ModelAttribute Employee employee, Errors errors, ModelMap map) {

        validator.validate(employee, errors);

    }

}
```

```

        if(errors.hasErrors()) {

            map.addAttribute("employee", employee);

        }else {

            Integer empld=service.saveEmployee(employee);

            map.addAttribute("message", "Employee '"+empld+"' created");

            //clear form

            map.addAttribute("employee", new Employee());

        }

        return "EmployeeRegister";

    }

    //3. get Data form DB all rows

    @GetMapping("/viewAll")

    public String showData(ModelMap map) {

        List<Employee> emps=service.getAllEmployees();

        map.addAttribute("emps", emps);

        return "EmployeeData";

    }

    //4. Get Pagination Data

    @GetMapping("/viewPage")

    public String
showPageData(@PageableDefault(page=0,size=3,sort="empld",direction=Direction.ASC) Pageable
pageable,@ModelAttribute Employee filter ,ModelMap map) {

        Specification<Employee> spec=new EmployeeSpec(filter);

        Page<Employee> emps=service.getAllEmployees(spec,pageable);

        map.addAttribute("page", emps);

        map.addAttribute("filter", filter);

```

```
        return "EmployeeData";
    }

    //5. delete employee byId

    @GetMapping("/delete")

    public String doDelete(@RequestParam("empId")Integer eid) {

        service.deleteEmployee(eid);

        //map.addAttribute("message", "Employee deleted successfully");

        return "redirect:viewPage";

    }

    //6. view One employee

    @GetMapping("/viewOne")

    public String viewOne(@RequestParam("empId")Integer empId,ModelMap map) {

        Employee emp=service.getEmployeeById(empId);

        map.addAttribute("emp", emp);

        return "EmployeeView";

    }

    //7. show Edit page

    @GetMapping("/edit")

    public String showEdit(@RequestParam("empId")Integer empId,ModelMap map) {

        Employee e=service.getEmployeeById(empId);

        map.addAttribute("employee", e);

        return "EmployeeEdit";

    }

    //8. update employee

    @PostMapping("/update")
```



```
        public String updateEmp(@ModelAttribute Employee employee) {  
            service.saveEmployee(employee);  
            return "redirect:viewPage";  
        }  
    }  
}
```

EmployeeRegister.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
  
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
  
<html>  
  
<head>  
  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
  
<title>Employee Register Page!!</title>  
  
</head>  
  
<body>  
  
<h1>Welcome to Employee</h1>  
  
<form:form action="insert" method="post" modelAttribute="employee">  
  
<pre>  
  
Name : <form:input path="empName"/>  
  
<form:errors path="empName"/>  
  
Email : <form:input path="empMail"/>  
  
<form:errors path="empMail"/>  
  
Gender: <form:radiobutton path="empGen" value="Male"/>Male <form:radiobutton path="empGen"  
value="Female"/>Female
```

```
<form:errors path="empGen"/>

Address: <form:textarea path="empAddr"/>

<form:errors path="empAddr"/>

Langs : <form:checkbox path="empLangs" value="ENG"/>ENG

        <form:checkbox path="empLangs" value="HIN"/>HIN

        <form:checkbox path="empLangs" value="TEL"/>TEL

<form:errors path="empLangs"/>

ID   : <form:select path="empIdType">

        <form:option value="">--select--</form:option>

        <form:option value="AADHAR">AADHAR</form:option>

        <form:option value="VOTER ID">VOTER ID</form:option>

        <form:option value="PAN CARD">PAN CARD</form:option>

        <form:option value="OTHER">OTHER</form:option>

    </form:select>

<form:errors path="empIdType"/>

ID NUM: <form:input path="empIdNum"/>

<form:errors path="empIdNum"/>

<input type="submit" value="Register"/>

</pre>

</form:form>

${message}

</body>

</html>

EmployeeData.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
        pageEncoding="ISO-8859-1"%>

<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Employee Data</title>

</head>

<body>

    <h1>Welcome to Employee Data Page!!</h1>

    <form:form action="viewPage" method="get" modelAttribute="filter">

    <pre>

        ID : <form:input path="empId"/>

        NAME : <form:input path="empName"/>

        Email:<form:input path="empMail"/>

        <input type="submit" value="Search"/>

    </pre>

    </form:form>

    <table border="1">

        <tr>

            <th>ID</th>

            <th>NAME</th>

            <th>EMAIL</th>

            <th colspan="3">OPERATIONS</th>

        </tr>

    </table>

</body>

</html>
```

```

        <!-- <th>GENDER</th>

        <th>ADDRESS</th>

        <th>LANGUAGES</th>

        <th>ID</th>

        <th>NUMBER</th> -->

    </tr>

    <c:forEach items="{page.getContent()}" var="e">

        <tr>

            <td>

                <a href="viewOne?empld=${e.empld}">

                    <c:out value="${e.empld}" />

                </a>

            </td>

            <td><c:out value="${e.empName}" /></td>

            <td><c:out value="${e.empMail}" /></td>

            <%-- <td><c:out value="${e.empGen}" /></td>

            <td><c:out value="${e.empAddr}" /></td>

            <td><c:out value="${e.empLangs}" /></td>

            <td><c:out value="${e.empldType}" /></td>

            <td><c:out value="${e.empldNum}" /></td> --%>

            <td>

                <a href="delete?empld=${e.empld}">

                </a>

            </td>

```

[illegible]


```
<body>

<h1>Welcome to Employee View Page!!</h1>

<table border="1">

<tr>

    <td>ID</td>

    <td>${emp.empId}</td>

</tr>

<tr>

    <td>NAME</td>

    <td>${emp.empName}</td>

</tr>

<tr>

    <td>EMAIL</td>

    <td>${emp.empMail}</td>

</tr>

<tr>

    <td>GENDER</td>

    <td>${emp.empGen}</td>

</tr>

<tr>

    <td>ADDRESS</td>

    <td>${emp.empAddr}</td>

</tr>

<tr>

    <td>LANGUAGE</td>
```

```
        <td>${emp.empLangs}</td>
    </tr>
    <tr>
        <td>ID TYPE</td>
        <td>${emp.empIdType}</td>
    </tr>
    <tr>
        <td>ID NUM</td>
        <td>${emp.empIdNum}</td>
    </tr>
</table>
</body>
</html>
```

EmployeeEdit.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Edit Page!!</title>
</head>
<body>
<h1>Welcome to Employee Edit Page!!</h1>
```



```
<form:form action="update" method="post" modelAttribute="employee">

<pre>

ID : <form:input path="empId" readonly="true"/>

Name : <form:input path="empName"/>

<form:errors path="empName"/>

Email : <form:input path="empMail" readonly="true"/>

<form:errors path="empMail"/>

Gender: <form:radiobutton path="empGen" value="Male"/>Male <form:radiobutton path="empGen"
value="Female"/>Female

<form:errors path="empGen"/>

Address: <form:textarea path="empAddr" readonly="true"/>

<form:errors path="empAddr"/>

Langs : <form:checkbox path="empLangs" value="ENG"/>ENG

        <form:checkbox path="empLangs" value="HIN"/>HIN

        <form:checkbox path="empLangs" value="TEL"/>TEL

<form:errors path="empLangs"/>

ID : <form:select path="empIdType">

        <form:option value="">--select--</form:option>

        <form:option value="AADHAR">AADHAR</form:option>

        <form:option value="VOTER ID">VOTER ID</form:option>

        <form:option value="PAN CARD">PAN CARD</form:option>

        <form:option value="OTHER">OTHER</form:option>

</form:select>

<form:errors path="empIdType"/>

ID NUM: <form:input path="empIdNum"/>

<form:errors path="empIdNum"/>
```

```

<input type="submit" value="Update"/>

</pre>

</form:form>

</body>

</html>

```

Hibernate List Mapping:

If checkbox is used at UI, at a time multiple values can be selected. So primitive variable like String, Long is not good to use to store all values.

Choose List<String> type, every list type will be stored in a child based on Hibernate Design with 3 columns (key, element, index)

emptab		
eid	lang	pos
100	Eng	0
100	Hin	1
100	Mar	2
[Key]	[element]	[index]

Code:

```

@ElementCollection                                keyColumn

@CollectionTable(name="emptab", joinColumns=@JoinColumns(name="eid"))

@Column(name="lang") //element column

@OrderColumn (name="pos") // index column

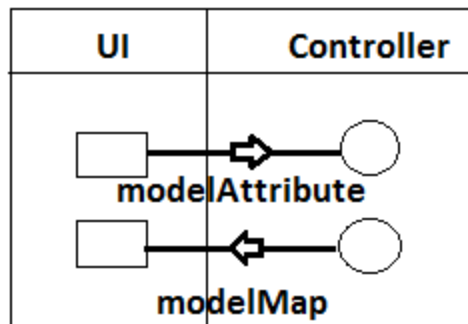
private List<String> emplangs;

```

Spring Boot Controller Code Design:

1. **ModelAttribute:** It is used to fetch form data in the object format. On click submit button, complete Spring Form completed to model Class object. Object be given by programmer at form tag level using <form:form... modelAttribute="employee">
2. To read this data at controller **@ModelAttribute** Employee employee

3. **ModelMap**: It is use to send data from controller to UI. Use method “**addAttribute(key,value)**” to add data to ModelMap, use same key at UI level to read data.



```
package com.app.controller;
```

```
@Controller
```

```
public class EmployeeController {
```

```
    @Autowired
```

```
    private IEmployeeService service;
```

```
    //1. To show register page
```

```
    @GetMapping ("/reg")
```

```
    public String showReg (ModelMap map) {
```

```
        map.addAttribute("employee", new Employee()) {
```

```
            return "EmployeeRegister";
```

```
        }
```

```
    //2. To insert data in DB and clear form
```

```
    public String saveEmp (@ModelAttribute Employee employee, ModelMap map) {
```

```
        Integer empld = service.saveEmployee (employee);
```

```
        map.addAttribute ("employee", new Employee());
```

```
        return "EmployeeRegister";
```

```
    }
```

```

    }
}

```

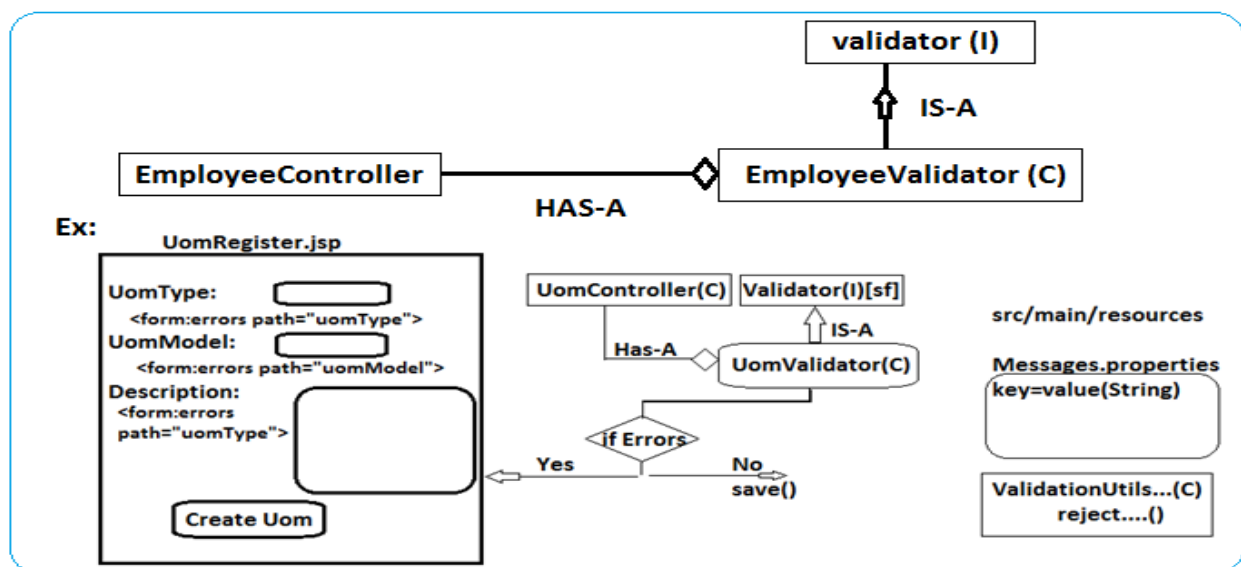
Validator API in Spring Boot Application:

Define one class that implements validator (I) (org.springframework.validation.validator).

In that class override two methods **supports()** and **validate()**

All errors checking must be done in validate() method. Here target means model class object.

Validator class must be used in controller with HAS-A relation show below:



Note:

#1 In Controller add dependency of validator (HAS-A)

#2 Add "Errors errors" parameter to save method as next of ModelAttribute

#3 Call validate() method if no error, save it and send back to UI.

#4 To display all errors use form tags

<form:errors path="*" /> OR

<form:errors path="variableName">

Controller New Code for insert:

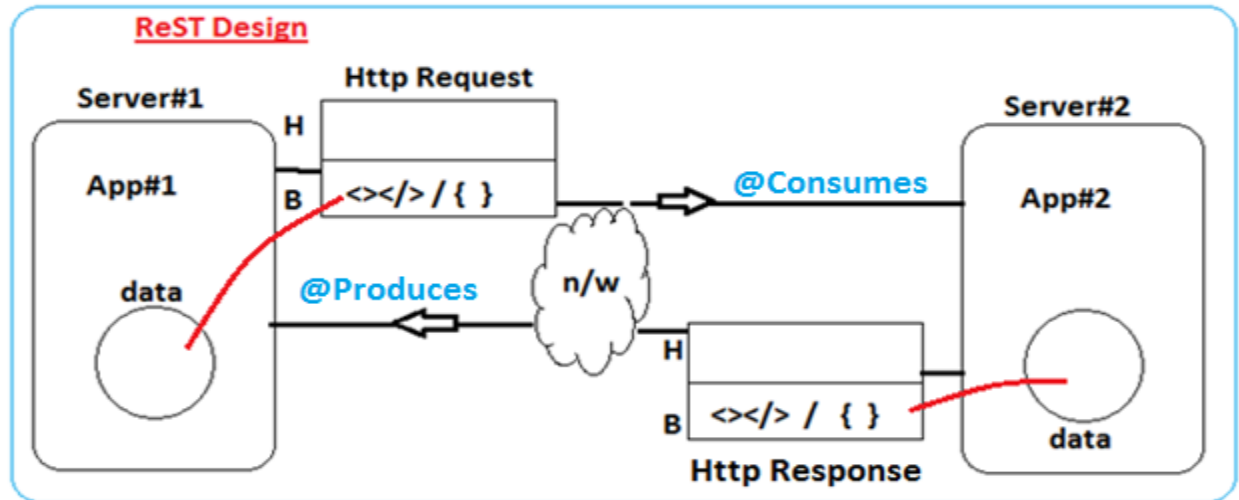
@PostMapping ("/insert")

```
public String saveEmp (@ModelAttribute Employee employee, Errors errors, ModelMap map) {  
  
    //1. Do validate  
  
    validator.validate(employee,errors);  
  
    //2. Check errors if yes- goto UI  
  
    if(errors.hasErrors()){  
  
        map.addAttribute("employee",employee);  
  
    } else {  
  
        //3. If no errors  
  
        Integer empld = service.saveEmployee (employee);  
  
        map.addAttribute ("message","Employee ""+empld+"" created");  
  
        //clear form  
  
        map.addAttribute("employee",new Employee());  
  
    }  
  
    return EmployeeRegister;  
  
}
```

Webservice

MediaType Annotations:

@Consumes and **@Produces** are known as MediaType annotation those are used to convert **JSON/XML** (Global Format Data) to **Object** and **Object** to Global Format (**JSON/XML**).



@Consumes: It will convert Global Data taken from and convert into Object format, This object is given as input to method (as parameter)

@Produces: It converts object which is method return value (output) and placed into **HttpResponseBody**.

Ex: @Consumes ("application/json")

@Produces ("application/json")

```
public Model show (Product p) {  
  
}
```

Here **@Consumes** reads JSON data from **HttpRequest** Body.

This JSON Data will be converted to Product class object and given as input to show ()

show() executes method body by taking product object as input and returns finally model class object.

Model class object will be converted to JSON and placed into **HttpResponseBody**

Here, @Consumes ("application/json") can be written as @Consumes(MediaType.APPLICATION_JSON)

MediaType is a class given from javax.ws.rs.core package and APPLICATION_JSON is a static property which has internal value="application/json"

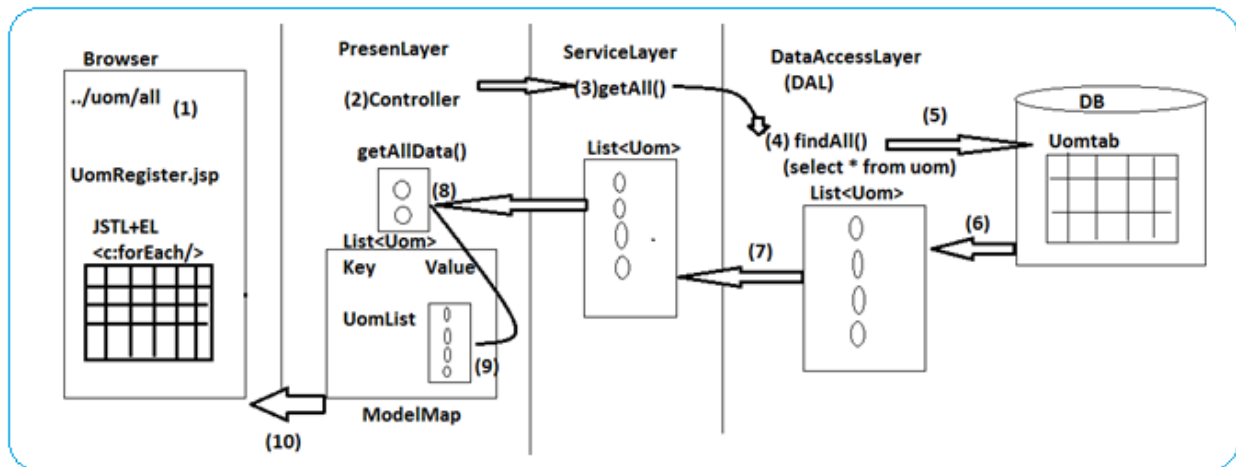
@Consumes ("application/xml") is equal to @Consumes(MediaType.APPLICATION_XML)

Every class object can be converted to JSON, even JSON can be converted to any class object.

Every class object cannot be converted to xml format. Only JAXB (Java Architecture for XML Binding) class object can be converted to xml.

To convert normal class to JAXB class, apply @XmlRootElement on top of class.

Fetching Data from DB to UI:



#1 Define one method in EmployeeController which will be executed for URL request `"/viewAll"`, It should be get data from DB using service layer and send to UI.

@GetMapping ("/viewAll")

```
public String showData (ModelMap map) {  
    List<Employee> emps = service.getAllEmployees();  
    map.addAttribute("emps", emps);  
    return "EmployeeData";  
}
```

#2 Define EmployeeData.jsp; use JSTL forEach loop and display data

```
<html><body><h1> Welcome to Data </h1>  
<c:forEach items="${emps}" var="e">  
<c:out value="${e.empId}"/>  
<c:out value="${e.empName}"/>
```

```
<c:out value="{e.empSal}"/>
</c:forEach>
</html>
```

Pagination Using Spring Boot:

Spring Data JPA has provided interface pagination "**PagingAndSortingRepository**" for pagination with Data sorting process.

It has provided method:

Page <T> findAll (Pageable p)

Where T = Model className

Pageable = Input for pagination like page Number, size of page, sort type.

Page = Output of pagination like page data, next, previous, last, first.

Pageable details must be passed using request URL like

<http://....viewPage?page=3&size=2&sort=empName>

Note:

#1 If number of pages are 10 (ex) then page number starts from zero (0) and ends by size-1 i.e. 9

#2 **isFirst()** and **isLast()** boolean methods are used to identify current page is firstPage (zero) and lastPage (size-1)

#3 It support **hasPrevious()** and **hasNext()** for navigation to next and previous pages.

#4 To provide default pagination details use annotation: **@PageableDefault** with details like: page=0, size=3, sort="empId", direction=DirectionAs.

#5 To get current page number, Method is: **getMethod(): int**

#6 To get the List data from page object:

Use method: **getContent()**.

****boolean hasContent();**

Provider Code for **@Produces**:

#1 web.xml (same as before)

#2 Model class


```
package com.app;

public class Employee {

    private int empId;

    private String empName;

    private double empSal;

    //set-get, const

    //toString()

}
```

#3 Provider class

```
package com.app;

@Path ("/msg")

public class Message {

    @GET

    @Path ("/msg")

    @Produces ("application/JSON")

    public Employee show () {

        Employee e = new Employee ();

        e.setEmpId (10);

        e.setEmpName ("AA");

        e.setEmpSal ("2.2");


        return e;

    }

}
```

#4 Consumer Code

```
package com.app;
```


GET  https://.../show		Send
Body	Head	Status: 200 OK
{ "empId":10,"empName":"A","empSal":2.2}		

@Produces for List: Output look like:

```
[
  {},
  {},
  {}
]
```

EX: Provider Class Code:

```
package com.app;
```

```
@Path ("/msg")
```

```
public class Message {
```

```
    @GET
```

```
    @Path
```

```
    @Produces (MediaType.APPLICATION_JSON)
```

```
    public List <Employee> show () {
```

```
        List <Employee> emp = Arrays.asList ( new Employee (10,"A",2.2);
```

```
                                                new Employee (10,"A",2.2);
```

```
                                                new Employee (10,"A",2.2));
```

```
        return emp;
```

```
    }
```

```
}
```

```

[
  {
    "empId":10,
    "empName":"A",
    "empSal":2.1
  },
  {
    "empId":11,
    "empName":"B",
    "empSal":2.1
  },
  {
    "empId":12,
    "empName":"B",
    "empSal":2.1
  }
]

```

**)Override equals() and hascode() methods in case of collection type set<Employee > or it's equal

//alt+shift+s,h (ok)

@Produced generally Xml:-

#1 On top of Model add annotation **@XMLRootElement** to make it as JAXB class,

package com.app;

// ctrl+shift+O(imports)

@XMLRootElement

```

public class Employee{

    private int empId;

    private String empName;

    private Double empSal;

    //set,get.....to String

#2 Provider class

package com.app;

@Path ("/msg")

public class Message {

    @GET

    @Path

    @Produces (MediaType.APPLICATION_XML)

    public Employee show() {

        emp.setEmpId (10);

        emp.setEmpName ("AA");

        emp.setEmpSal ("2.3");

        return emp;

    }

}

```

O/p: <Employee>

<empId>10</empId>

<empName>AA</empName>

<empSal>2.3</empSal>

</Employee>

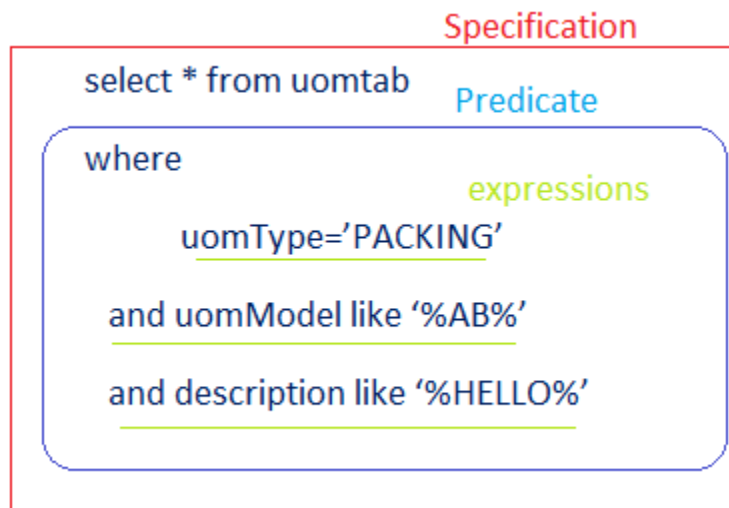
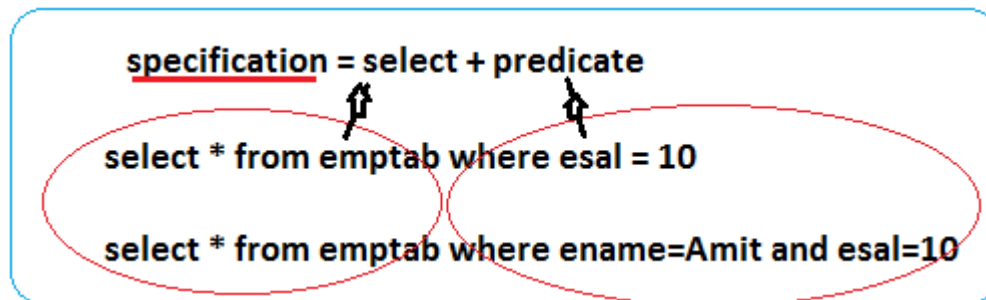
Specification API for Spring Boot Data JPA:

It is used to design one predicate with select for data fetching based on dynamic input given by end user. (Generating sql queries at runtime)

Here, Predicate indicates “where clause creation with conditional gives like and/or”.

In case of search/filter screen concepts by default, we should use **JpaSpecification Executor <I>**.

It provides findAll () method with specification and pagination process.



Method:

findAll(Specification<T> spec, Pageable pageable): Page<T>

****Define one class that implements Specification (I)**

****Override toPredicate(...):Predicate** method which generates CriteriaQuery (In Hibernate) at runtime, which using CriteriaBuilder

CriteriaBuilder (cb): (link all)

select -----> CriteriaQuery (CQ)

from -----> Root (root)

where -----> Predicate (P)

Coding Steps:

#1 Enable process for module using repository interface that extends JpaSpecificationExecutor<T>

#2 Define Specification class that generates predicate based on input CriteriaBuilder, CriteriaQuery, Root.

#3 override toPredicate(): method

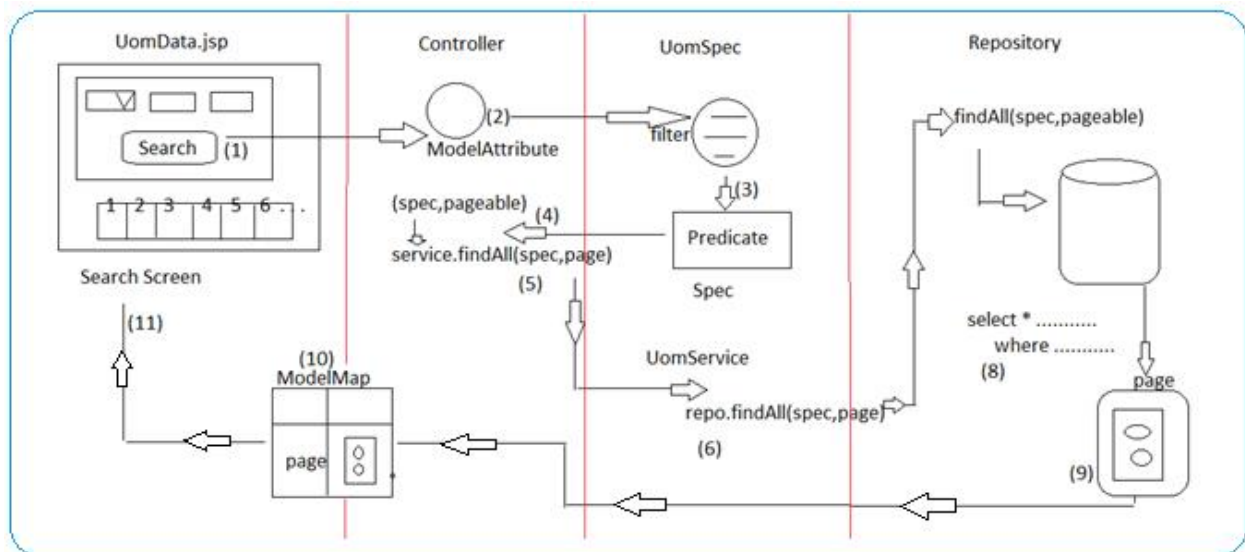
#4 Design Filter Screen/Search Screen using HTML form/Spring form Tag library with type GET (not POST type here).

#5 on click "Search" button form converted to model class object (filter) object

#6 In controller read back spec object and call findAll (spec,pageable):Page<T>

#7 Send Page<T> to UI back to display data also and filter object to ModelMap.

Example:



(and condition) **Conjunction** = select * from employee where empId=? **and** empName=?

(or condition) **Disjunction** = select * from employee where empId=? **or** empName=?

Specification Implementation Steps:

#1 EmployeeRepository (I) must extends One Predefined interface JpaSpecificationExecutor<T> (I) to use method findAll(Specification,Pageable):Page

#2 Define one class for Specification process which must implements Specification(I) given by Spring Boot Data Jpa.

#3 Override method toPredicate():Predicate in above class.

#4 All logic to expression if filter variable is not empty && not null

#5 Add one method in IEmployeeService which gets Pagination Data 'Page<T>' by taking inputs like Spec, Pageable.

#6 Implement method in EmployeeServiceImpl by using findAll(Specification,Pageable):Page<T> which is defined in JpaSpecificationExecutor.

#7 Read Search screen data into one ModelAttribute named as filter.

#8 Create object to Spec class and Pass filter object.

#9 Call Service method getAllEmployees with spec, pageable

#10 Add filter to ModelMap

#11 Define search form with 'GET' type with ModelAttribute ='filter' at EmployeeData.jsp

like operator for String – EmpEmail,(radioButton,email,password)

****** root.get ("employee").in(filter.getEmployees()) [For List Type checkbox]

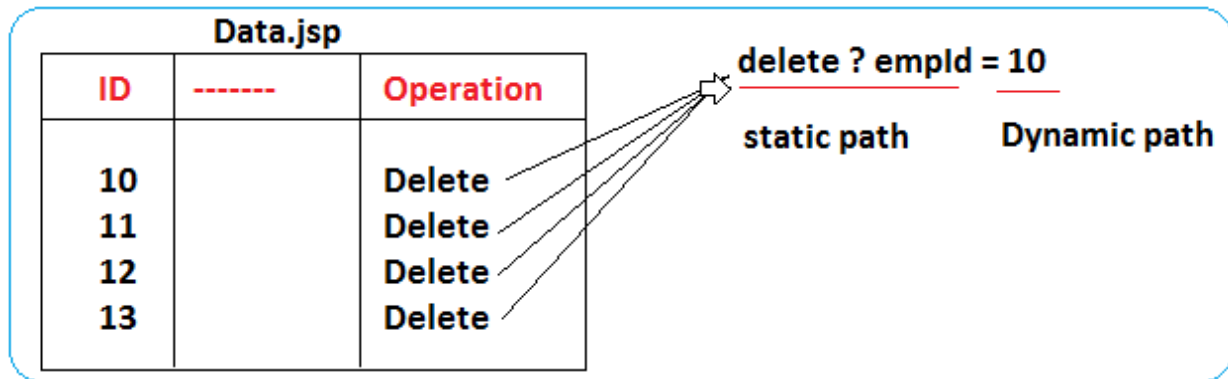
Custom Banner Generator: -

<https://devops.datenkollektiv.de/banner.txt/index.html>

- Create one file named as "banner.txt" under src/main/resources folder.
- Place copied content to this banner.txt file
- Run startup class.

URL Rewriting:

It is a process of constructing url using static path & Dynamic path, by using anchor tag <a> and ExpressionLanguage (EL).



In Data.jsp add below code under <forEach> before closing <tr> tag.

```
<td>
```

```
    <a href="delete?empld=${e.empld}">Delete</a>
```

```
</td>
```

****Read this parameter at controller method using annotation**

```
@RequestParam ("key")DataType localVariable
```

It will support even type conversion also.

****If method return type is used as "redirect:url"**

behave as `response.sendRedirect(url)`

view Employee By Id:

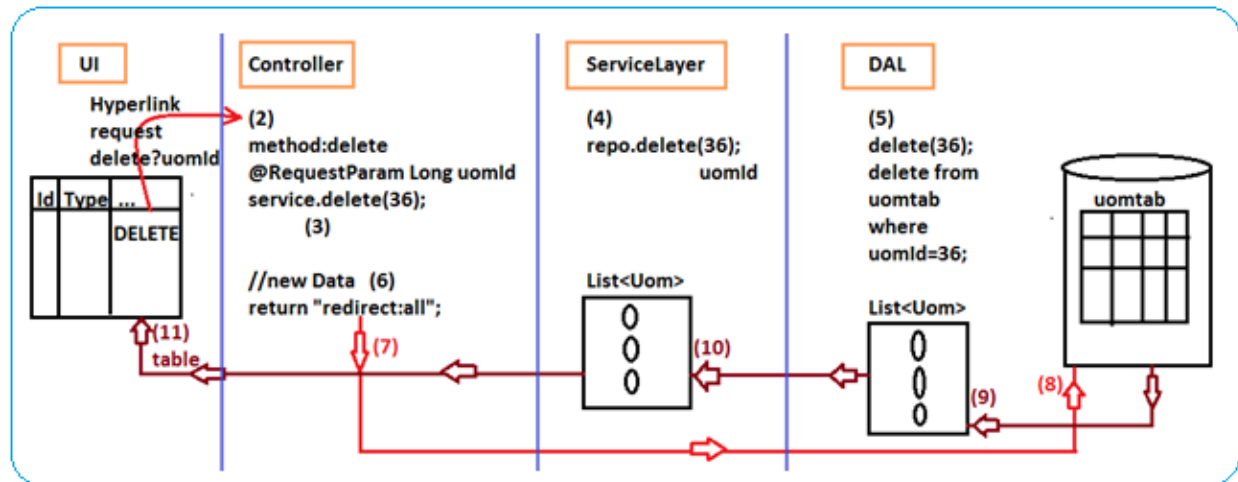
****If module has multiple fields to display data, we must create viewPage to show all values in search page show only few important fields.**

****Also create Hyperlink "view" at every row using URL-Writing like below:**

```
<a href="viewOne?empld=${e.empld}">View</a>
```

****In Controller class, define one method to fetch data based on empld and send same data to EmployeeView page to display it.**

Example:



Data Edit Operation in Spring Boot:

#1 create one Hyperlink ie <a> tag in Data Jsp Page Using URL-ReWriting.

Code:-

```
<a href="edit?empld=${e.empld}">EDIT</a>
```

#2. On click hyperlink request made to EmployeeController, ex URL like:-

http:-----/edit?empld=20

Here

(a)Read @RequestParam empld

(b) Call service.getEmployeeById

(c)read back employee object from DB

(d)send employee to UI Page(modelMap)

#3.Define one JSP named as "EmployeeEdit" with Spring from tag library .we can

Also make few fields as readonly=true

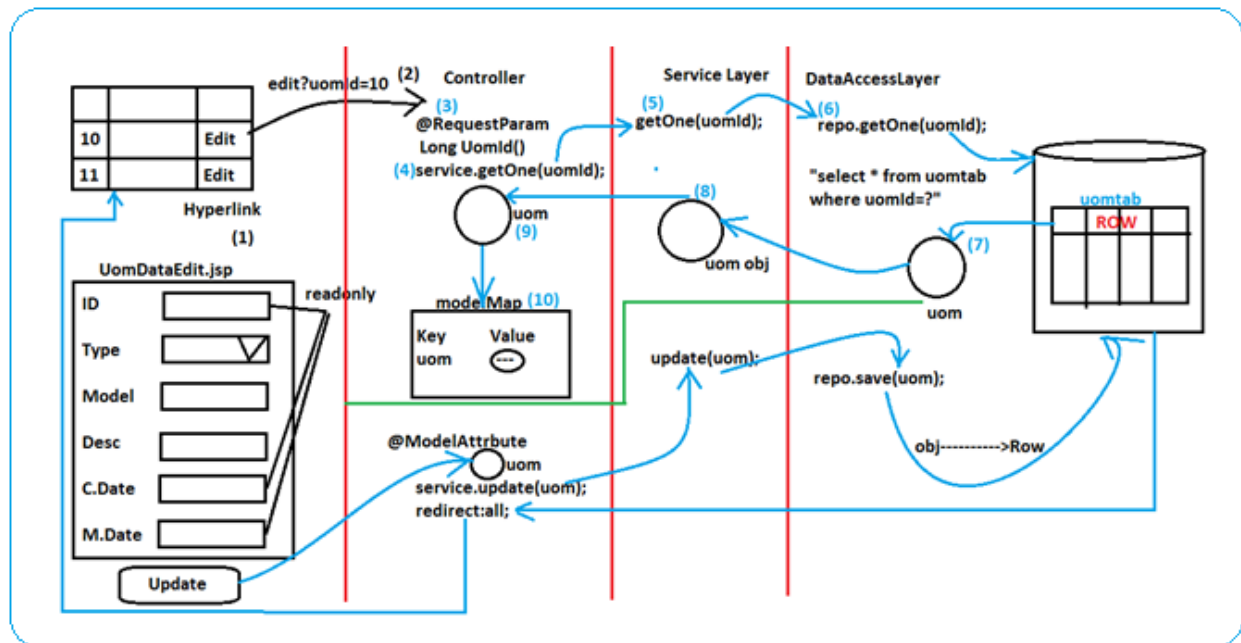
Ex:-<Form:input path="" readonly="true"/>

On click submit(update) button form data will be converted to Object is called as ModelAttribute.

#4 On click submit (update) button form Data will be converted to Object i.e. called as ModelAttribute.

#5 Read ModelAttribute at controller method and call service, saveEmployee (emp) redirect to viewPage to come back "EmployeeData" Jsp with new Data.

Execution Flow Design:



Spring Boot Email Configuration:

- #1 In application.properties file configure host, port, username and password with extra secure properties like `auth, starttls.enable` [tls -> Transport Layer Security]
- #2 Use `JavaMailSender` and `MimeMessageHelper` classes for writing emailCode
- #3 `JavaMailSender (I)` is an interface, for this `JavaMailSenderImpl (C)` class object will be auto created by Spring Boot.
- #4 Use `FileSystemResource` given by boot f/w to send attachments
- #5 In case of exception return false, else return true

Example: `SpringBootEmail`

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>

<groupId>org.sathyatech</groupId>

<artifactId>SpringBootEmail</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<name>SpringBootEmail</name>

<description>MVC project for Spring Boot</description>

<parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <java.version>1.8</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-mail</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
        </build>
</project>

application.properties

#Server Config

server.port=2626

# Email config

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=javaraghu2018@gmail.com

spring.mail.password=2019javaraghu2019

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true
```

ServletInitializer.java

```
package com.app;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override

    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

        return application.sources(SpringBootEmailApplication.class);

    }

}
```

SpringBootEmailApplication.java

```
package com.app;

import java.io.File;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.core.io.FileSystemResource;

import com.app.email.EmailUtil;

@SpringBootApplication

public class SpringBootEmailApplication implements CommandLineRunner{

    public static void main(String[] args) {

        SpringApplication.run(SpringBootEmailApplication.class, args);

    }

    @Autowired

    private EmailUtil email;

    @Override

    public void run(String... args) throws Exception {

        File f=new File("C:\\Users\\Public\\Pictures\\Sample Pictures\\Jellyfish.jpg");

        FileSystemResource fis=new FileSystemResource(f);

        boolean issent=email.sendEmail("javabyraghu@gmail.com", "Test Image...", "Test Image...",fis);

        System.out.println(issent);

        System.exit(0);

    }

}
```

EmailUtil.java

```
package com.app.email;

import javax.mail.internet.MimeMessage;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.io.FileSystemResource;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.stereotype.Component;

@Component

public class EmailUtil {

    @Autowired

    private JavaMailSender mailsender;

    public boolean sendEmail(String to,String subject,String text,FileSystemResource file) {

        boolean sent=false;

        try {

            MimeMessage message=mailsender.createMimeMessage();

            MimeMessageHelper helper=new MimeMessageHelper(message,

file!=null?true:false);

            helper.setTo(to);

            helper.setSubject(subject);

            helper.setText(text);

            helper.setFrom("javaraghu2018@gmail.com");

            if(file!=null)

                helper.addAttachment(file.getFilename(), file);

            mailsender.send(message);

            sent=true;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return sent;

    }

}
```



```

        } catch (Exception e) {

            sent=false;

            e.printStackTrace();

        }

        return sent;

    }

}

```

YML:

It is another way of writing properties configuration in Spring Boot application.

It provides chain method for writing of key=value, also called as Human Readable Format.

It will not accept duplicate levels of keys.

Use symbol “:” with tab space for next level of key.

It is supported by Spring Boot by default. It must be placed in src/main/resources folder.

spring-boot-starter-web auto enable yml concept using “**sakeyaml.x.y.jar**”

[will change application.prop file to yml]

Example:

server:

port:2200

spring:

mvc:

view:

prefix:/WEB-INF/views/

suffix:.jsp

datasource:

driver-class-name:com.mysql.jdbc.driver

<url:jdbc:mysql://localhost:3306/boot>

username:root

password:root

hikari:

connection-timeout:1000

maximum-pool-size:20

minimum-idle:15

pool-name:HikariCP

jpa:

show-sql:true

hibernate:

ddl-auto:update

Security in SpringBoot:

PermitAll: This URL indicated No-Security is implemented.

Authenticated: This category says User must be logged in to access URL.

Authorized: Here both Login and Role required to access URL. Role also called as Authority.

Types of Authentication:

1. InMemoryAuthentication (RAM)
2. JdbcAuthentication (JDBC)
3. UserDetails (ORM)

1. **InMemoryAuthentication:** It is used to store details in RAM. Details like Username, Password and Role.

To implement this concept we must define one class that extends **WebSecurityConfigurerAdapter** and must apply annotation **@EnableWebSecurity**.

*Provide 2 methods for Authentication details and Url details.

Code:

```

package com.app.config;

@Configuration
@EnableWebSecurity

Public class SecurityConfig extends WebSecurityConfigurerAdapter{

    //un,pwd,role

    @Autowired

    public void configureGlobal(AuthenticationManagerBuilder auth){...}

    //url levels

    @override

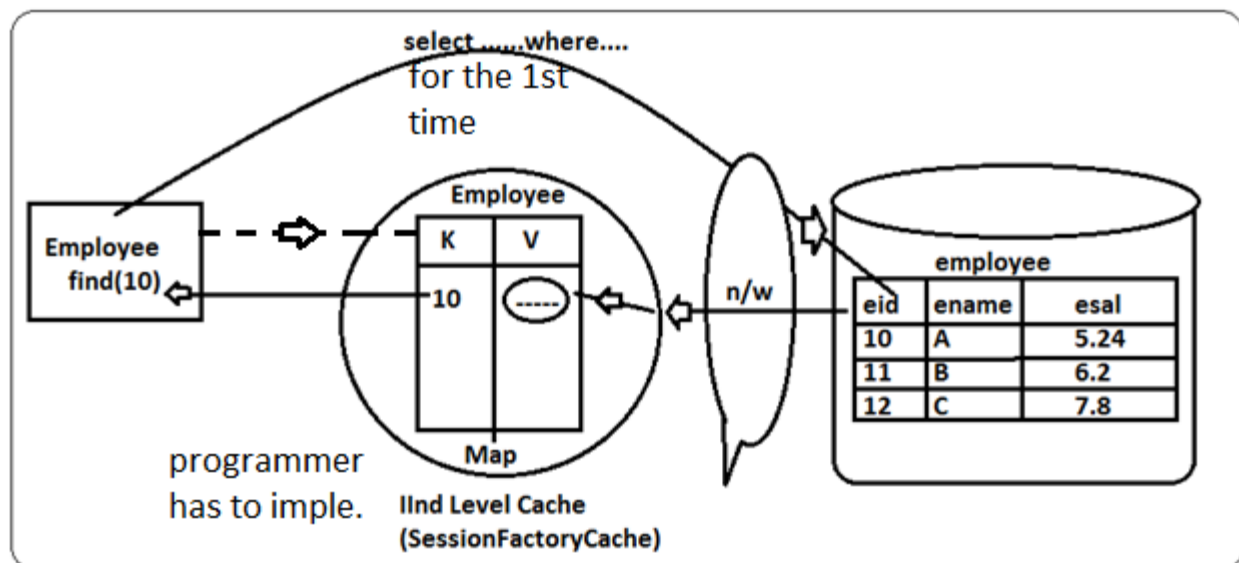
    protected void configure(HttpSecurity http){...}

}

```

Caching Process in SpringBoot:

CacheManager(I) is used to manage cache concept for 2nd level SessionFactory cache in boot applications, which reduces network calls between application and Database. So this improves the performance.



Steps to implement cache Management in Spring Boot Application:

#1 Enable Cache Concept in Application by adding below annotation over starter class level

@EnableCaching

#2 Define Cache manager impl class which manages Second Level Cache (Session Factory Cache)

API: CacheManager(I) Implemented by

ConcurrentMapCacheManager(C)

#3 Apply Annotations over ServiceImpl method

@Cacheable: It must be applied over method which will select one row (ex: findById), Also works for all rows select but not recommended.

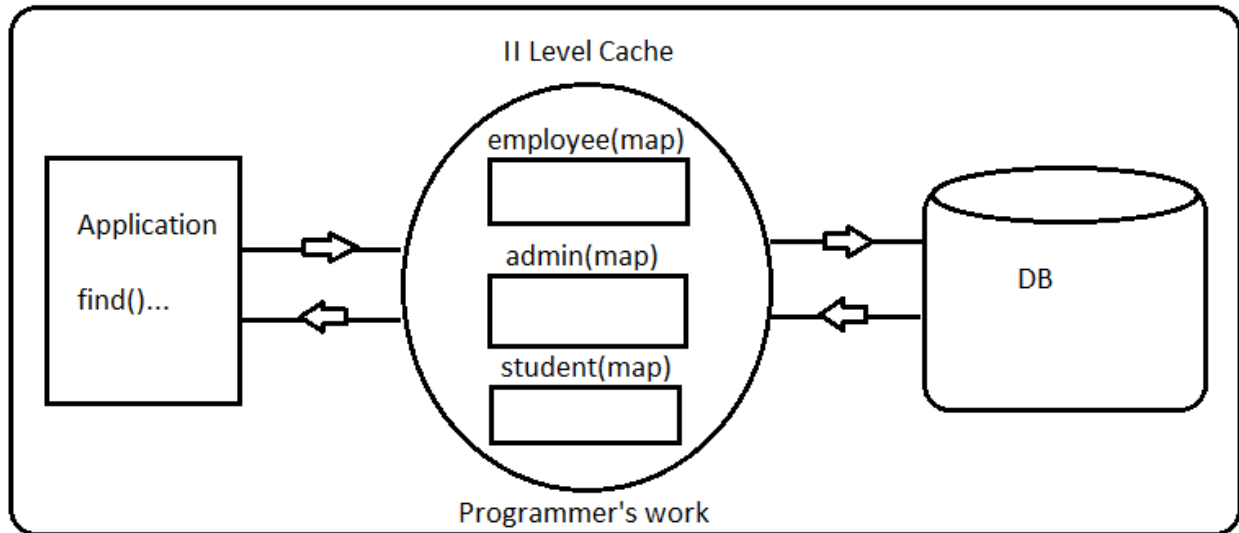
@CacheEvict: It must be applied over delete operations, it will remove data from II-Level cache Map.

@CachePut: It must be applied over update operation, which updates data in cache (II-level) when update() operation is done.

*) Below code creates one Map in II-Level cache with Map name as “employee”

@Bean

```
public CacheManager cm(){  
  
    ConcurrentMapCacheManager cm = new ConcurrentMapCacheManager  
("emplopyee","admin","student");  
  
    return cm;  
  
}
```



*) To place data in employee map code looks like

```
@CachePut(value="employee")
```

```
@CacheEvict(value="employee")
```

```
@Cacheable (value="employee")
```

Profiles Management in SpringBoot:

Based on Environments changes or end client changes, few modifications comes in application.properties and at impl class levels.

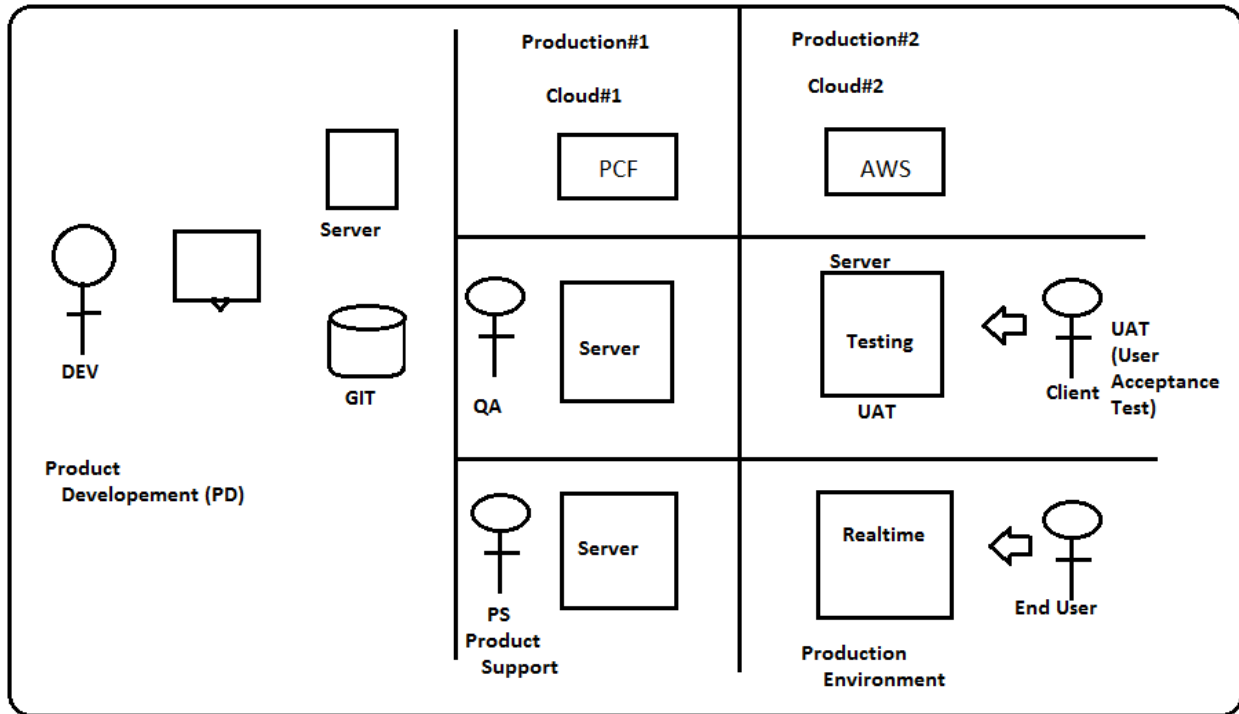
*) Profiles handles

```
>application.properties
```

```
>Choosing Code Type for Env. (Environment Specific Imple.)
```

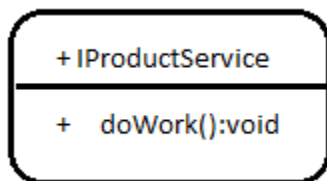
*) We will be having different profiles like Production, QA, DEV, Cloud, UAT, PS, etc. Database Details, Pooling, Logging, Security may get changes from one profile to another.

*) In this case we should not modify existed application.properties, we should define new properties file with profile name.



*) Profiles are used to execute logic selected for environment.

Ex: consider one interface with method



*) Here, default implementation is given by ProductServiceImpl, it will do save work. It is also called as "default Profile".

*) for Customer DHL, it should work as email & save, then define another Impl as DhIProdcutionServiceImpl as profile "dhl".

*) for Customer E-Trans, it should work as print and save, then define another impl as EtransProductionImpl as profile "etrans".

---Profile creations steps----

#1 Properties file syntax:

application-[profileName].properties

#2 Code level syntax:

```
@Profile("profileName")
```

#3 Selection and Execution of Profile System Arguments (VM Arguments) cmd line arguments

spring.profile.active property

Arguments Passing In Java Application:

Java supports mainly 2 types of arguments:

1. Command Line Arguments
2. VM/System Arguments

#1 Command Line Arguments:

These are inputs to main method only.

#2 VM/System Arguments:

It creates variable with data at JVM level, so that it can be accessed in any application running in same JVM, using code

```
String val = System.getProperty("key")
```

It is also called as System Level (JRE) Variable.

Example:

```
Public class Test{  
  
    Public static void main(String[] args){  
  
        Sysout("ID:" + args[0]);  
  
        Sysout("Name:" + args[1]);  
  
        String s = System.getProperty("data");  
  
    }  
  
}
```

>Right click in code area (editor)

>Run As > Run Configuration

>Click on Arguments > Enter Data like

Through CMD --> java Employee -Ddata=ABC CDF 125

(1)

Arguments

(2) Program Args

10 AB

(3) VM Args

-Ddata=Hello

(4)

Apply & Run

>Apply & Run

Consider below example:

#1 For Product Service given with 3 profiles

Default

DHL & Etrans

#2 for every customer and any environment **default profile** will be **activated** by **springBoot**, which reads Properties from application.properties file.

#3 For DHL customers, logic should work in different way not in default way then we are writing another implementation class which is annotated with “**@Profile(“dhl”)**”, in a same way for Etrans customer we should write another implementation annotated with “**@Profile(“ettrans”)**”.

#4 In case of DHL Profile it will search for **application-dhl.properties** file, if **not found** it will read data from **default** file application.properties.

#5 To activate different profiles:

a)Using cmd line Arguments:

--spring.profiles.active=dhl

b)Using System (VM) Arguments

-Dspring.profiles.active=dhl

c)using .properties file

(add key)

spring.profiles.active=dhl

Example: SpringBootProfilesEx

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.sathyatech</groupId>

  <artifactId>SpringBootProfilesEx</artifactId>

  <version>1.0</version>

  <packaging>war</packaging>

  <name>SpringBootProfilesEx</name>

  <description>Demo project for Spring Boot</description>

  <parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <java.version>1.8</java.version>
```

```
</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-tomcat</artifactId>

        <scope>provided</scope>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-test</artifactId>

        <scope>test</scope>

    </dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>
```

</project>

application.properties

server.port=2626

#spring.profiles.active=etrans-prof

myapp.message=Hello from Default

application-dml-prof.properties

myapp.message=Hello from DML

application-etrans-prof.properties

myapp.message=Hello from ETRANS

ServletInitializer.java

package com.sathyatech.app;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

 @Override

 protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

 return application.sources(SpringBootProfilesExApplication.class);

 }

}

SpringBootProfilesExApplication.java

```
package com.sathyatech.app;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import com.sathyatech.app.service.IEmployeeService;

@SpringBootApplication

public class SpringBootProfilesExApplication implements CommandLineRunner {

    public static void main(String[] args) {

        SpringApplication.run(SpringBootProfilesExApplication.class, args);

    }

    @Autowired

    private IEmployeeService service;

    @Override

    public void run(String... args) throws Exception {

        System.out.println(service.getMessage());

        System.exit(0);

    }

}
```

IEmployeeService.java

```
package com.sathyatech.app.service;

public interface IEmployeeService {
```

```
        public String getMessage();  
    }  
}
```

EmployeeServiceImpl.java

```
package com.sathyatech.app.service;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Profile;  
import org.springframework.stereotype.Service;  
  
@Service  
@Profile("default")  
public class EmployeeServiceImpl implements IEmployeeService {  
  
    @Value("${myapp.message}")  
  
    private String myTitle;  
  
    @Override  
  
    public String getMessage() {  
  
        return "From DEFAULT:"+myTitle;  
  
    }  
  
}
```

DHLEmployeeServiceImpl.java

```
package com.sathyatech.app.service;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Profile;  
import org.springframework.stereotype.Service;
```

```
@Service

@Profile("dml-prof")

public class DmlEmployeeServiceImpl implements IEmployeeService {

    @Value("${myapp.message}")

    private String myTitle;

    @Override

    public String getMessage() {

        return "From DML:"+myTitle;

    }

}
```

EtransEmployeeServiceImpl.java

```
package com.sathyatech.app.service;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Service;

@Service

@Profile("etrans-prof")

public class EtransEmployeeServiceImpl implements IEmployeeService {

    @Value("${myapp.message}")

    private String myTitle;

    @Override

    public String getMessage() {

        return "From E-Trans:"+myTitle;

    }

}
```

```
}  
  
}
```

Spring Boot Schedulers:

To execute one task in application automatically (without any manual request or input provided by end user) based on **point of time (exact Date or Time or Both) , Period of Time (days/weeks/months/hours/minutes/sec) gap (not exact date and time).**

Point of Time example:

- 1) 25th Dec – 2018
- 2) 4:00 AM every day
- 3) Jan 2019
- 4) 8:35:42 745 AM 06/15/2018

Period of Time example:

- 1) After 45 days
- 2) For every 2 minutes
- 3) With 15 minute gap
- 4) After 4 hours 10 minutes 3 seconds

Note:

#1 These are used to execute regular work in application like Generate Report on end of day/month/year.

#2 Point of time works based on System/Server time.

#3 Period of time work based on Server/ App start up time.

#4 Expressions value can be provided directly or using EL from properties file.

Syntax: **`${cronExpression}`**

#5 Every Scheduler will be executed automatically, no request should be made by end user.

#6 Only void type methods are eligible for Scheduling.

#7 Point of Time can be provided using **cron expression**.

#8 Period of Time can be provided using **fixedDelay/fixedRate** and even cron expression.

Example Application:

#Step1 Enable Scheduling concept in application using annotation **@EnableScheduling** at starter class level.

#Step2 Define one **Service/Component** class with void method (write any logic)

#Step3 Method must be annotated with **@Scheduled**

#Step4 Start Application

EX#1 fixedDelay Scheduling

@Component

Class Process{

 @Scheduled(fixedDelay=4000)

 Public void doWork(){

 Sysout("Hi..." + new Date());

 }

}

fixedDelay will execute doWork(m1) will be executed with every 4 sec gap on last finish of method.

EX#2 fixedRate Scheduling

@Scheduled(fixedRate=2000)

Public void doWork(){

 Sysout("Hii..." + new Date());

}

In this case, Scheduler wait for given gap or until last method finish work.

Case#1

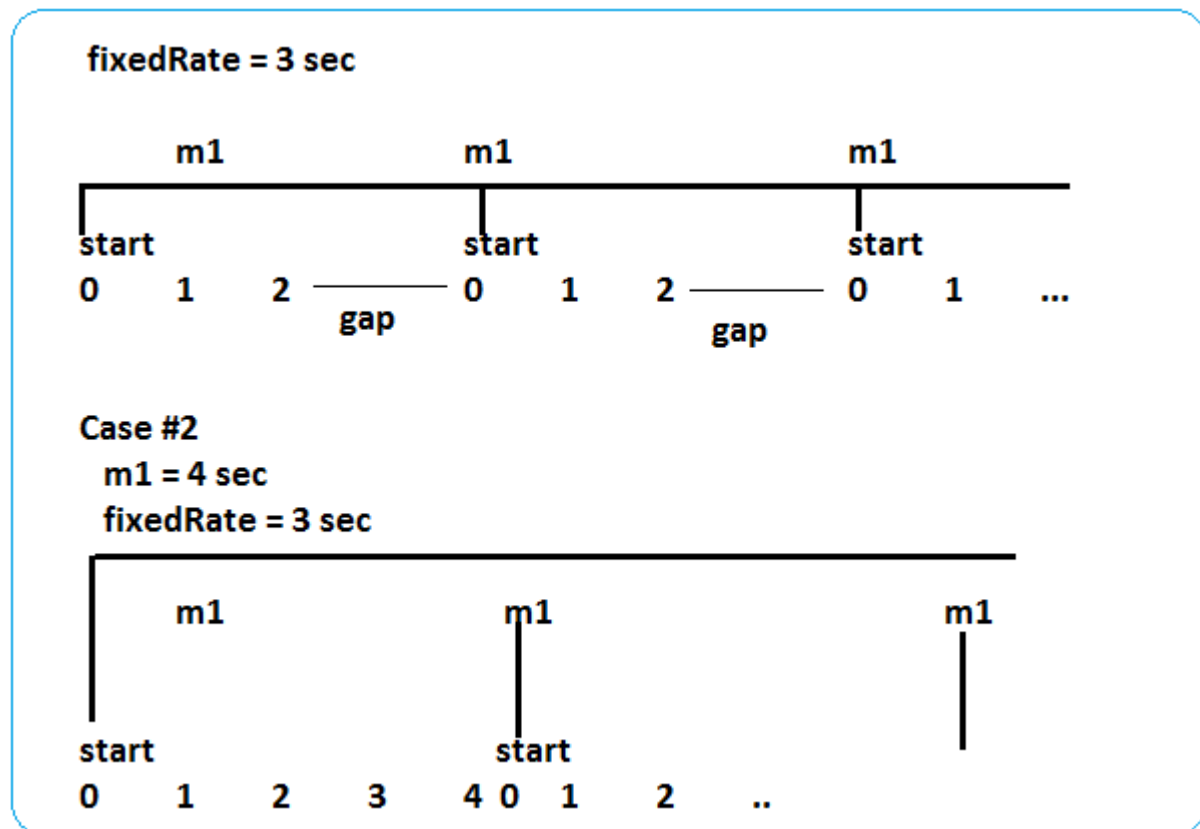
Rate time > method Execution time then

Gap = (rate-method time)

Case#2

Rate time <= method Execution Time then

Gap zero, it wait for until last method finish the work.



fixedRate and fixedDelay both works based on Period of time not based on point of time. To specify task based on **Point of Time** use **Cron expression**.

Cron expression is used to provide both point and period of time.

Ex:

```
@Scheduled (cron="*/10 * * * * *")
```

```
Public void doWork(){
```

```
    Sysout("Hi:"+new Date());
```

```
}
```

Cron syntax:

SEC	MIN	HRS	DAY	MONTH	WEEKDAY
0-59	0-59	0-23	1-31	1-12	SUN-SAT

Possible Symbols: , - * */ ?

, = possible values

- = range

* = not provided

? = any value is ok (applicable for day/week)

*/ = point of time/day

--Few example cron expression--

Ex#1 8 AM = 8:00:00 AM, 6:00:00 AM

Cron = "0 0 8,18 * * *"

Ex#2 1:35 PM = 1:35:00 PM

Cron = "0 35 13 * * *"

Ex#3 8AM-10AM with gap of 30 mins

8:00:00 AM, 8:30:00 AM, 9:00:00AM, 9:30:00 AM, 10:00:00 AM

Cron = "0 0/30 8-10 * * *"

Ex#4 12th Jan 4:35 PM

4:35:00 PM 12th 1 month

Cron = "0 35 16 12 1"

Ex#5

Cron = "10 2 8 * * *"

8:02:10 AM

Ex#6

Cron = "*/10 2 8 * * *"

with gap of 10 sec

8:02:00 > 8:02:10 > 8:02:20 > 8:02:30 > 8:02:40 > 8:02:50

Ex#7

Cron = "25/10 2 8 * * * "

Possible executions: 8:02:25 > 8:02:35 > 8:02:45 > 8:02:55

Ex#8

Jan 8:00:00 AM

Cron = "0 0 8 ? 1 ?"

? = any day ? any week

Example: SpringBootSchedulers

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.sathyatech</groupId>

  <artifactId>SpringBootSchedulers</artifactId>

  <version>1.0</version>

  <packaging>war</packaging>

  <name>SpringBootSchedulers</name>

  <description>Demo project for Spring Boot</description>

  <parent>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-parent</artifactId>

    <version>2.0.2.RELEASE</version>
```

```
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-tomcat</artifactId>

            <scope>provided</scope>

        </dependency>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-test</artifactId>

            <scope>test</scope>

        </dependency>
    </dependencies>

    <build>

        <plugins>
```

```
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

application.properties

server.port=2626

ServletInitializer.java

```
package com.sathyatech.app;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(SpringBootSchedulersApplication.class);
    }
}
```

SpringBootSchedulersApplication.java

```
package com.sathyatech.app;
```

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

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling

public class SpringBootSchedulersApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringBootSchedulersApplication.class, args);

    }

}
```

MyProcess.java

```
package com.sathyatech.app;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class MyProcess {

    private int count=1;

    //@Scheduled(initialDelay=5000,fixedDelay=2000)

    //@Scheduled(initialDelay=5000,fixedDelay=2000)

    @Scheduled(cron="12,35,55 * * * * *")

    public void doWork() {

        System.out.println("Start:"+count);

        System.out.println(new Date()+":Hello"+count);

    }

}
```

```

        try {

            Thread.sleep(3000);

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println(new Date()+"Hello"+count);

        System.out.println("end:"+count);

        count++;

    }

}

```

JMS (Java Messaging Service):

It is used to send and receive messages in text, Object and Collection format between **two Java Applications** by using middleware **MOM (Message Oriented Middleware)** ex: Apache Active MQ.

Messages will be given and taken from destinations

Destinations are 2 types:

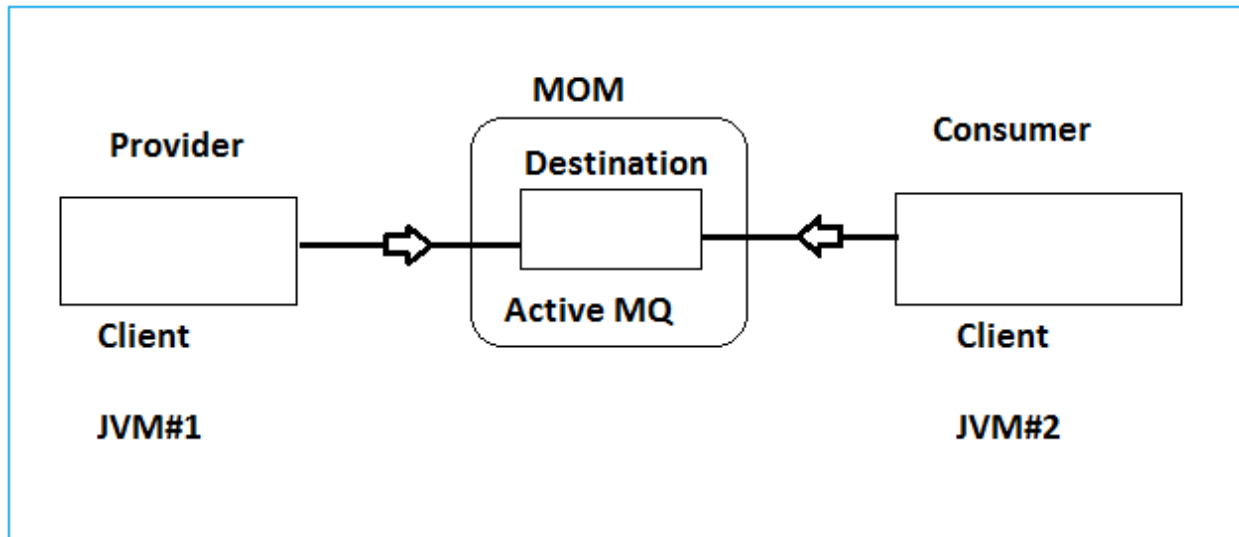
1. Queue
2. Topic

#1 In **Peer-To-Peer (P2P)** communication **Queue** Concept is used, which sends message to **one Consumer** at a time.

#2 Topic is used to send Same message to **multiple Consumers**.

#3 **Legacy JMS** steps:

- a) ConnectionFactory Creation
- b) Connection Creation
- c) Session Creation
- d) Client Creation
- e) Destination
- f) Message Creation and Send/Receive



Spring Boot Actuator:

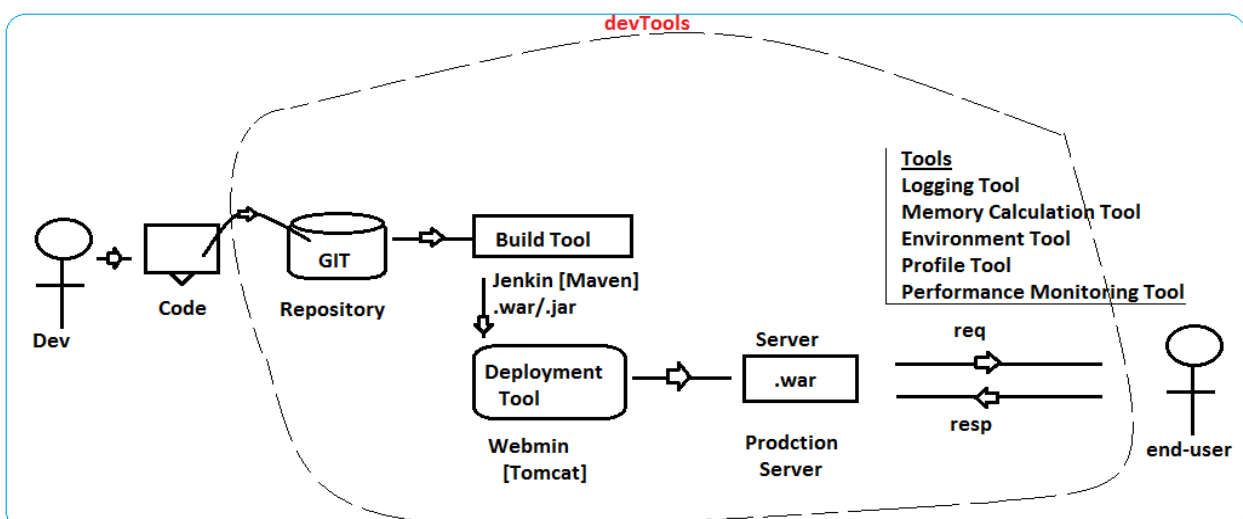
*) **Production Environment:** - Once application is developed then it is given to end customer using one server deployment using one server deployment (Deployment=place war in server and start it) with supportive tools like Logging, Tracing, Health check, Server status, DB Configuration, Current Profile, Memory Details, etc.

This kind of setup is called as production Environment.

*) **Actuator** is a service developed using Spring Boot ReST and Spring Boot Security which provides default setup for Production.

*) This setup can be accessed using URL's (Endpoints) which is also called as “**Production Ready Environment**”.

Flow of Production Environment:



*) To know end point details given by Spring boot Actuator, URL is

<https://docs.springframework.io/spring-boot/docs/current/reference/html/production-ready-endpoint.html>

Steps To create Actuator Project

#1 Define Spring boot Starter Project with Actuator dependency

#2 In pom.xml it should look like

<dependency>

 <groupId>org.springframework.boot</groupId>

 <artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

#3 It should also have Security Dependency added in pom.xml because Spring Boot Actuator is built on top of Spring Boot Security and Spring Boot ReST web services.

#4 Write app logic for DB operations /Email/ ReST Application/ Security (anything).

#5 Start application (Run starter class) and enter the URL, looks like

<https://localhost:2828/actuator/health>

#6 In case of older version use direct end points (SpringBoot 1.5.x). In case of new version use endpoint starts with /actuator (spring Boot 2.x).

Ex: End point request

1.5.x

<http://localhost:2828/beans>

2.x

<http://localhost:2828/actuator/beans>

#7 In Spring Boot 1.5.x, all are enable by default, in 2.x those are in disabled to activate them add one property in application.properties file

management.endpoint.web.exposure.include=*

#8 Endpoints:

- a. **/actuator/auditevents**: It will audit (notedown) all loggIn operations from different networks (success/failure logins).
- b. **/actuator/beans**: It will show objects and their dependencies created in container.
- c. **/actuator/configprops**: It will display all properties and system properties loaded by spring boot.
- d. **/actuator/env**: It will show details of JRE, JDK, Version, Server, Jar files etc. details
- e. **/actuator/health**: It will show current status of server and App possible values [UP/DOWN].
- f. **/actuator/httptrace**: It will show what http request made so far are: ex: /emp/reg, /emp/delete.
- g. **/actuator/loggers**: It will display default logging file provided by SpringBoot
- h. **/actuator/metrics**: Calculations of memory/cache values as map properties shown here.
- i. **/actuator/mappings**: It provides "what url for what method" is provided
- j. **/actuator/scheduledtask**: It will display Scheduler details in App.

** Apply these data provided in JSON format.

Example: Spring5JMSProvider

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.sathyatech</groupId>

    <artifactId>Spring5JMSProvider</artifactId>

    <version>1.0</version>

    <dependencies>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-jms</artifactId>

            <version>5.0.6.RELEASE</version>
```

```
</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-context</artifactId>

    <version>5.0.6.RELEASE</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-core</artifactId>

    <version>5.0.6.RELEASE</version>

</dependency>

<dependency>

    <groupId>org.apache.activemq</groupId>

    <artifactId>activemq-spring</artifactId>

    <version>5.15.4</version>

</dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.apache.maven.plugins</groupId>

            <artifactId>maven-compiler-plugin</artifactId>

            <version>3.7.0</version>

            <configuration>

                <source>1.8</source>
```

```
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

AppConfig.java

```
package org.sathyatech.app.config;

import javax.jms.ConnectionFactory;

import org.apache.activemq.ActiveMQConnectionFactory;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.jms.core.JmsTemplate;

@Configuration

public class AppConfig {

    @Bean

    public ConnectionFactory connectionFactory() {

        ActiveMQConnectionFactory cf=new ActiveMQConnectionFactory();

        cf.setBrokerURL("tcp://localhost:61616");

        return cf;

    }

    @Bean

    public JmsTemplate jmsTemplate() {

        JmsTemplate jt=new JmsTemplate();
```

```
        jt.setConnectionFactory(connectionFactory());  
        return jt;  
    }  
}
```

SendMessageTest.java

```
package org.sathyatech.app.test;  
  
import javax.jms.JMSEException;  
import javax.jms.Message;  
import javax.jms.Session;  
  
import org.sathyatech.app.config.AppConfig;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
import org.springframework.jms.core.JmsTemplate;  
import org.springframework.jms.core.MessageCreator;  
  
public class SendMessageTest {  
  
    public static void main(String[] args) {  
  
        AnnotationConfigApplicationContext c=new  
AnnotationConfigApplicationContext(AppConfig.class);  
  
        JmsTemplate jt=c.getBean(JmsTemplate.class);  
  
        jt.send("my-test-spring", new MessageCreator() {  
  
            @Override  
  
            public Message createMessage(Session ses) throws JMSEException {  
  
                return ses.createTextMessage("SAMPLE ONE");  
  
            }  
  
        });c.close();}}
```

Example: Spring5JMSConsumer

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.sathyatech</groupId>

  <artifactId>Spring5JMSConsumer</artifactId>

  <version>1.0</version>

  <dependencies>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-jms</artifactId>

      <version>5.0.6.RELEASE</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-context</artifactId>

      <version>5.0.6.RELEASE</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-core</artifactId>

      <version>5.0.6.RELEASE</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.apache.activemq</groupId>
            <artifactId>activemq-all</artifactId>
            <version>5.15.4</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

AppConfig.java

```
package org.sathyatech.app.config;
```

```
import javax.jms.ConnectionFactory;

import javax.jms.MessageListener;

import org.apache.activemq.ActiveMQConnectionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jms.annotation.EnableJms;

import org.springframework.jms.listener.DefaultMessageListenerContainer;

import org.springframework.jms.listener.MessageListenerContainer;

@Configuration

@ComponentScan(basePackages= {"org.sathyatech.app"})

@EnableJms

public class AppConfig {

    @Autowired

    private MessageListener messageListener;

    @Bean

    public ConnectionFactory connectionFactory() {

        ActiveMQConnectionFactory c=new ActiveMQConnectionFactory();

        c.setBrokerURL("tcp://localhost:61616");

        return c;

    }

    @Bean

    public MessageListenerContainer listenerContainer() {

        DefaultMessageListenerContainer m=new DefaultMessageListenerContainer();
```



```
        m.setConnectionFactory(connectionFactory());

        m.setDestinationName("my-test-spring");

        m.setMessageListener(messageListener);

        return m;
    }
}
```

MyMessegeListner.java

```
package org.sathyatech.app.listener;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import org.springframework.stereotype.Component;

@Component

public class MyMessageListener implements MessageListener {

    @Override

    public void onMessage(Message message) {

        TextMessage tm=(TextMessage) message;

        try {

            System.out.println(tm.getText());

        } catch (JMSException e) {

            e.printStackTrace();

        }

    }

}
```

```
    }  
}
```

Test.java

```
package org.sathyatech.app.test;  
  
import org.sathyatech.app.config.AppConfig;  
  
import org.sathyatech.app.listener.MyMessageListener;  
  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
public class Test {  
  
    public static void main(String[] args) {  
  
        AnnotationConfigApplicationContext ac=new  
AnnotationConfigApplicationContext(AppConfig.class);  
  
        MyMessageListener ms=ac.getBean(MyMessageListener.class);  
  
    }  
}
```

Spring Boot Log4j Example using Slf4j API:

#1 Spring Boot by default enable with Logging API (Log4j Programing).

#2 It provides default Layout and Appender for both Console and File System.

#3 Programmer has to add Logger object in class level and specify LOG LEVEL in application.properties.

#4 Do not write log4.properties file.

#5 Enable/ Disable framework level loggings.

--Code--

application.properties:

logging.level.com.app=DEBUG

logging.level.org.springframework=OFF

logging.level.org.hibernate=OFF

logging.file=myapp.log

--IN Controller Class Ex:

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
..
```

```
public class EmployeeController{
```

```
    Logger log = LoggerFactory.getLogger(EmpController.class);
```

```
    public String show(..){
```

```
        log.INFO("Welcome to show method");
```

```
    }
```

```
}
```

****Add Actuator dependecy and enter URL**

<http://localhost:2828/actuator/logfile>

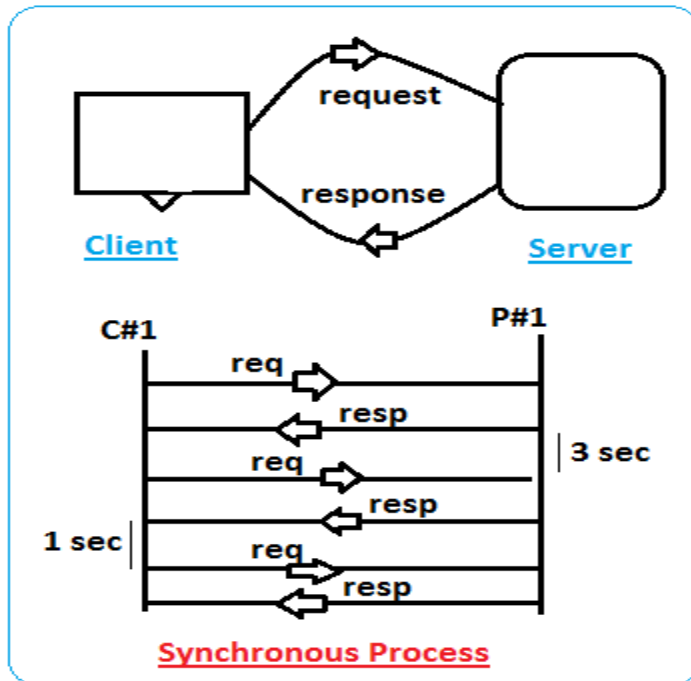
to see logfile (myapp.log).

AJAX Programing in Spring Boot:

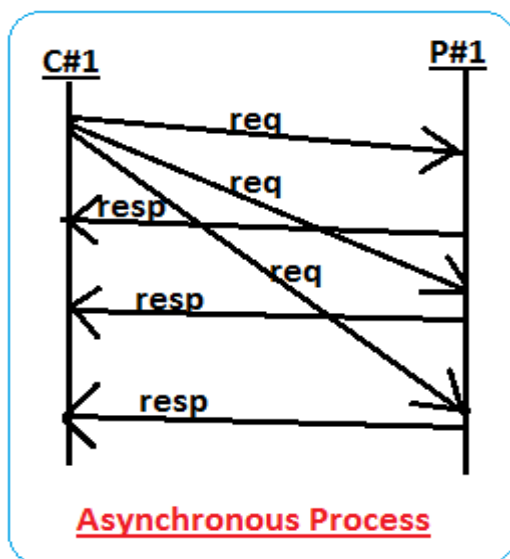
By default every application follows “**Synchronous Process**” in request/response execution.

It means **next request** can be made after receiving **previous response**.

It means request and response are executed in circular chain manner i.e. (one by one).



AJAX stands for “Asynchronous Javascript And XML”, it means application into Asynchronous mode, that means to make new request, client not required to wait for previous response.



AJAX calls are controlled using Events made by inputs (Keyboard, mouse, form inputs,).

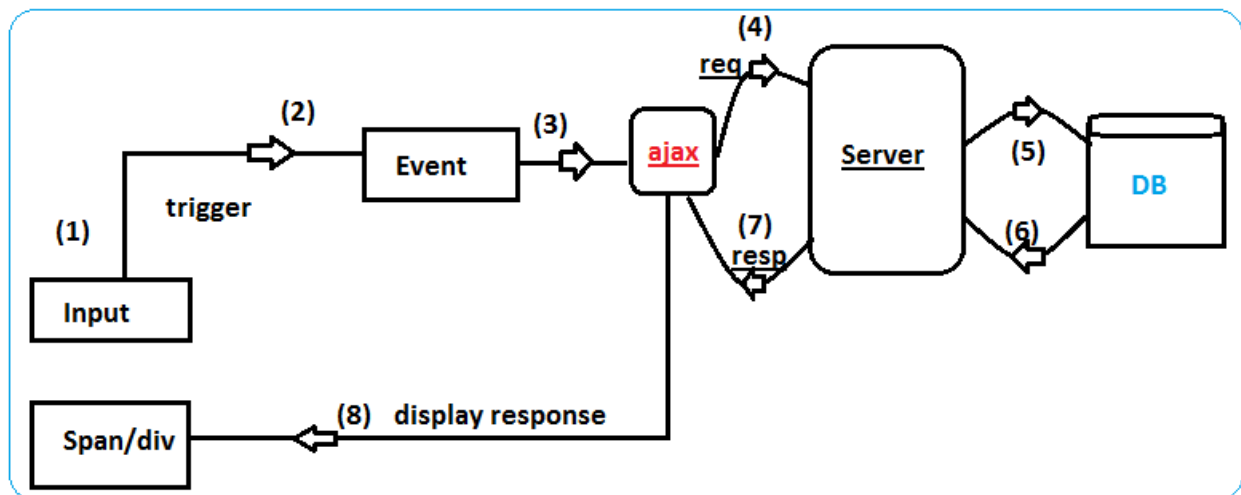
Ex: onKeyPress, onChange, onMouseOver, onclick, onDoubleClick, etc.

All these events are pre-defined in Javascript programming.

Every Input can be connected to AJAX call using events only. This even will be automatically triggered by input component, programmer not required to make even call externally.

Ajax makes request based on even and gets response back from server.

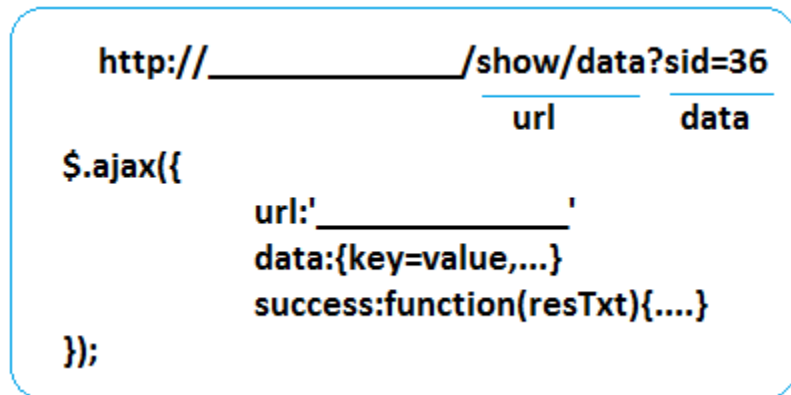
Execution looks like:



*) To make ajax call provide inputs like

- a. URL of application (Controller URL's)
- b. Data as Input (Request Parameters)
- c. success:function (to do work after response received)

Syntax:



AJAX Coding Steps:

#1 Define JSP/HTML page with simple text input with name and id properties.

HTML Form:

```
<input type="text" name="empName" id="empName"/>
```

--OR--

Spring Form:

```
<form:input path="empName">
```

#2 Add Script Line to enable JQuery with its library in JSP/HTML, Head Area

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"> </script>
```

#3 **Define JQuery code for Ajax call based on empName-> change event**

```
<script type="text/javascript">
```

```

$(document).ready(function(){
    $("#empName").change(function(){
        $.ajax({
            url:'getData',
            data:{"empName":$("#empName").val()},
            success:function(resTxt){
                $("#resultArea").text(resTxt);
            }
        })
    })
}

```

```
});
```

```
});
```

```
});
```

#4 To display result define span or div as ``

#5 Define Controller class with method annotated with `@ResponseBody`

`@Controller`

```
Public class HomeController{
```

```
    @GetMapping("/getData")
```

```
    Public @ResponseBody String getName(@RequestParam("empName") String name){
```

```
        return "Hello Mr/Mrs:"+name;
```

```
    }
```

******) In case of `@RestController` not even `@ResponseBody` annotation is required.

UI

```
<script....>
<script type="text/javascript">
$(document).ready(function(){
    $("#empId").change(function(){
        $.ajax({
            url:"getData",
            data:{"eid":$("#empId").val()},
            success:function(resTxt){
                $("#msgTxt").text(resTxt);
            }
        });
    });
});
});
```

Controller

```
@Controller
Public class HomeController{

    @GetMapping("/getData")

    Public @ResponseBody String getName(@RequestParam("empName") String name){

        return "Hello Mr/Mrs:"+name;
    }
}
```

Example: SpringBootJQueryAjaxExample

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>SpringBootjQueryAjaxExample</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <packaging>war</packaging>

    <name>SpringBootjQueryAjaxExample</name>

    <description>Demo project for Spring Boot</description>

    <parent>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.0.2.RELEASE</version>

        <relativePath/> <!-- lookup parent from repository -->

    </parent>

    <properties>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

        <java.version>1.8</java.version>

    </properties>

    <dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-web</artifactId>
```



```
</dependency>

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-tomcat</artifactId>

    <scope>provided</scope>

</dependency>

<dependency>

    <groupId>org.apache.tomcat.embed</groupId>

    <artifactId>tomcat-embed-jasper</artifactId>

    <scope>provided</scope>

</dependency>

<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-test</artifactId>

    <scope>test</scope>

</dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-maven-plugin</artifactId>

        </plugin>

    </plugins>

</build>
```

</project>

application.properties

server.port=2626

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

ServletInitializer.java

```
package com.example.app;
```

```
import org.springframework.boot.builder.SpringApplicationBuilder;
```

```
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
```

```
public class ServletInitializer extends SpringBootServletInitializer {
```

```
    @Override
```

```
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
```

```
        return application.sources(SpringBootJQueryAjaxExampleApplication.class);
```

```
    }
```

```
}
```

HomeController.java

```
package com.example.app;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class HomeController {

    @GetMapping("/home")

    public String show() {

        return "Home";

    }

    @GetMapping("/getData")

    public @ResponseBody String getName(@RequestParam("empName")String name) {

        System.out.println(name);

        return "Hello Mr/Mrs:"+name;

    }

}
```

SpringBootjQueryAjaxExampleApplication.java

```
package com.example.app;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootjQueryAjaxExampleApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringBootjQueryAjaxExampleApplication.class, args);

    }

}
```

Home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script>

$(document).ready(function(){

    $("#empName").change(function(){

        $.ajax({

            url:'getData',

            data:{"empName":$("#empName").val()},

            success:function(resTxt){

                $("#resultBox").text(resTxt);

            }

        });

    });

});

</script>

</head>

<body>

<h1>welcome to JSP Page!!</h1>

<input type="text" id="empName"/>
```

```
<span id="resultBox"></span>
```

```
</body>
```

```
</html>
```

Annotations:

@SpringBootApplication:

#1 @ComponentScan: It must be provided by programmer in case of Spring applications but in Spring Boot it will be auto selected based on Starter class.

Starter class package is taken as base package for application.

So we should create all classes under sub or current package of base package.

#2 @EnableAutoConfiguration: It will create config file based on <dependency> added in pom.xml

And it takes inputs given by application.properties or .yml

EX: Add starter-data-jpa in pom.xml then Spring Boot will create <bean> s for

DriverManagerDataSource, HibernateTransactionManager, HibernateProperties, Etc.

#3 @SpringBootConfiguration: To enable programmer config files, like @Configuration, @Controller, @Service this is written.