Spring Boot MVC with H2 database

- Create new SpringBoot starter project with below dependencies:
  - Web
  - JPA
  - H2
  - DevTools
- Update application.properties file to work with H2 database:

```
server.port=9000
spring.h2.console.enabled=true
spring.datasource.platform=h2
spring.datasource.url=jdbc:h2:mem:Rashmi
```

- Run the application to check if H2 database is working:

  http://localhost:9000/h2-console

- Create a folder “webapp” in src/main location to place .jsp files
- Create a jsp file home.jsp in webapp folder:

```
<form action="addEmployee">
    Employee ID: <input type="text" name="eid"><br>
    Employee Name: <input type="text" name="ename"><br>
    Employee Salary:<input type="text" name="salary"><br>
    <input type="submit">
</form>
```

- Create domain class Employee as:

```
@Entity
public class Employee {
    @Id
    private Integer eid;
    private String ename;
    private Double salary;

    //Setters and Getters
    //ToString method
}
```

- Create an interface which extends CRUDRepository:

```
public interface EmployeeRepo extends CrudRepository<Employee, Integer>{

}
```

- Create a controller class EmployeeController as:

```
@Controller
public class EmployeeController {

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

- Create a sql file “data.sql” inside src/main/resources folder:

```
insert into employee values(1,'Abhishek',50000);
insert into employee values(2,'Rashmi',40000);
insert into employee values(3,'Kanchu',30000);
insert into employee values(4,'Seema',30000);
```

- Autowire EmployeeRepo interface in Controller class as:

```
@Autowired
EmployeeRepo employeeRepo;
```

- Create another method in Controller class to save employee data in H2 database:

```
@RequestMapping("/addEmployee")
    public String addEmployee(Employee employee) {
        employeeRepo.save(employee);
        return "home.jsp";
}
```

- Create another jsp file “showEmployee.jsp” to display employee records:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false"%>
</head>
    <body>
        ${employee}
    </body>
</html>
```

- Create another method in Controller to retrieve data based on given employee ID:

```
@RequestMapping("/getEmployee")
public ModelAndView getEmployee(@RequestParam Integer eid) {
      ModelAndView mv=new ModelAndView("showEmployee.jsp");
        Employee employee=employeeRepo.findById(eid).orElse(new Employee());
        mv.addObject(employee);
        return mv;
}
```

- Add another form in home.jsp to get a search box to enter employee ID to search:

```
<form action="getEmployee">
    Employee ID:<input type="text" name="eid"><br>
    <input type="submit">
</form>
```

- CRUD Repository allows us to create method to find data by any column as below:

```
findBy<ColumnName>("SearchValue");

Example:
findByDepartment("Research") //In Controller
List<Employee> findByDepartment(String department);//In CRUD Repo
extended interface
```

- We can create custom method also by using HQL as below:

```
//CRUD Repo extended interface
@Query("FROM Employee WHERE eid>?1 ORDER BY ename DESC")
List<Employee>findByEidSorted(Integer eid);
//Controller
findByEidSorted(2);
```

- We can add appropriate methods to Controller class to make it work as REST App.

```
@ResponseBody
@RequestMapping("/employees")
public String getAlians() {
    return employeeRepo.findAll().toString();
}
```

```
@RequestMapping("/employees/{eid}")
@ResponseBody
public String getEmployeeById(@PathVariable("eid") Integer eid ) {
    return employeeRepo.findById(eid).toString();
}
```

- Note that we are getting data till now in the form of Array.
  We want our data in JSON format so that we could process it with more flexibility and with added features.
- To do so, we will first change EmployeeRepo interface and make it get extended from JPARepository instead of CRUDRepository.
- Now we will make change to methods in Controller class as below:
  Remove .toString() and change return type of methods:

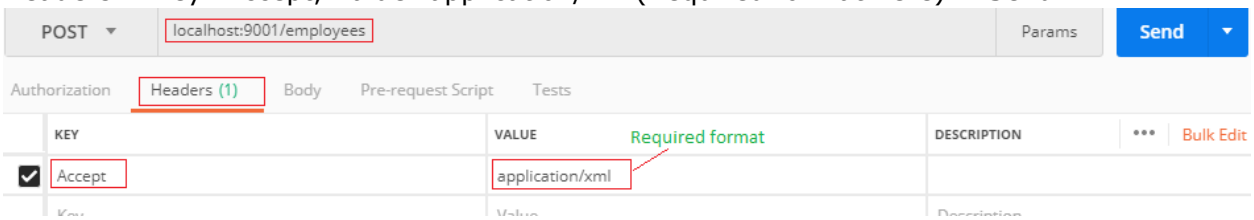
```
@ResponseBody
@RequestMapping("/employees")
public List<Employee> String getAlians() {
    return employeeRepo.findAll().toString();

}
```

```
@RequestMapping("/employees/{eid}")
@ResponseBody
public Optional<Employee> String getEmployeeById(@PathVariable("eid") Integer eid ) {
    return employeeRepo.findById(eid).toString();
}
```

- Now, we will use postman to test out API. (Windows x64 link)
- Postman also helps us with content-negotiation. (Content-negotiation means getting content in our specified format. By default, SpringBoot gives content in json format. But we can specify different formats like pdf, xml etc.)
- We need to add maven dependency as required to get content in a format other than json. (json is supported because of Jackson jar file).

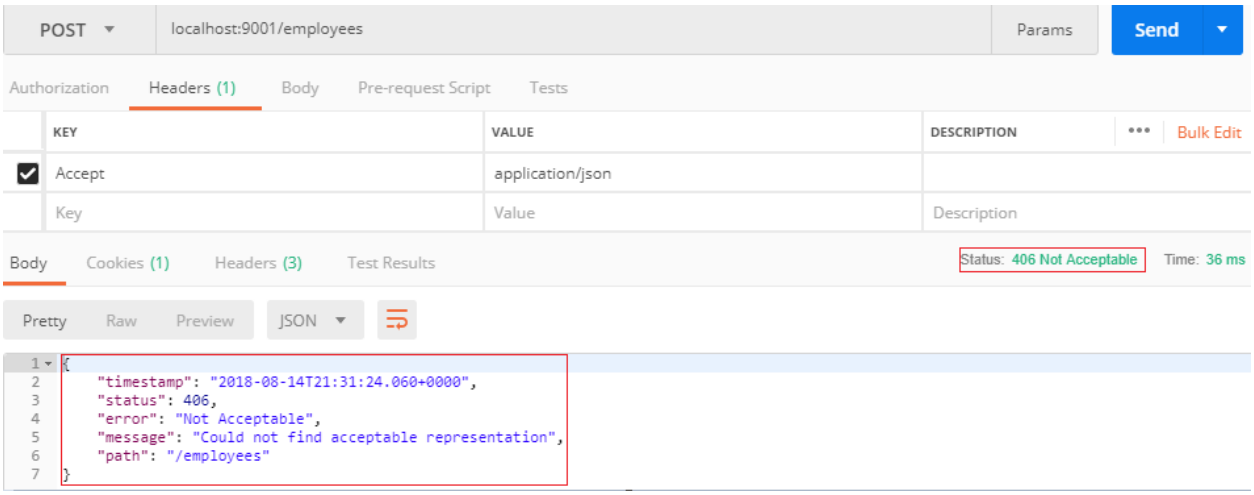
- Example: To get content in XML format, add [jackson-dataformat-xml] maven dependency in pom.xml file.
- Open postman and provide URL (localhost:9001/employees-> this example)
  Headers > Key: Accept, Value: application/xml(Required format here) > Send



- We can control the content format from our application by adding produces property to @RequestMapping annotation in controller class.
  Ex.: To restrict our application to support only xml, we will make below change:

```
@RequestMapping(path="/employees",produces= {"application/xml"})
```

- Now, if we try to access data in any format other than xml, we get error as below :



- We can eliminate mentioning @ResponseBody at all the methods by changing @Controller annotation at controller class to @RestController as below:

```
@RestController ~~@Controller~~
public class EmployeeController {
//
}
```

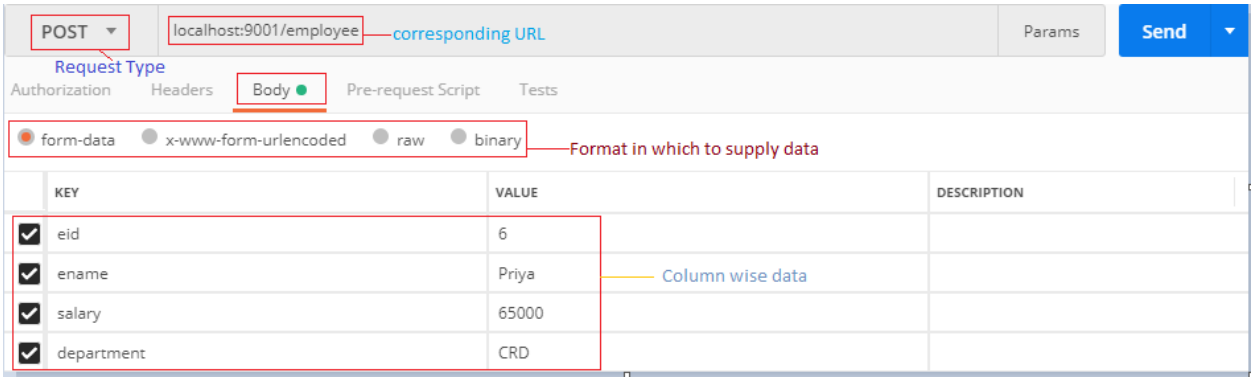
- We should specify request type instead of just RequestMapping:
  GET: To fetch and display the data
  POST: To store/save the data

Example:

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

```
@PostMapping("/employee")
public Employee saveEmployee(Employee employee) {
    employeeRepo.save(employee);
    return employee;
}
```

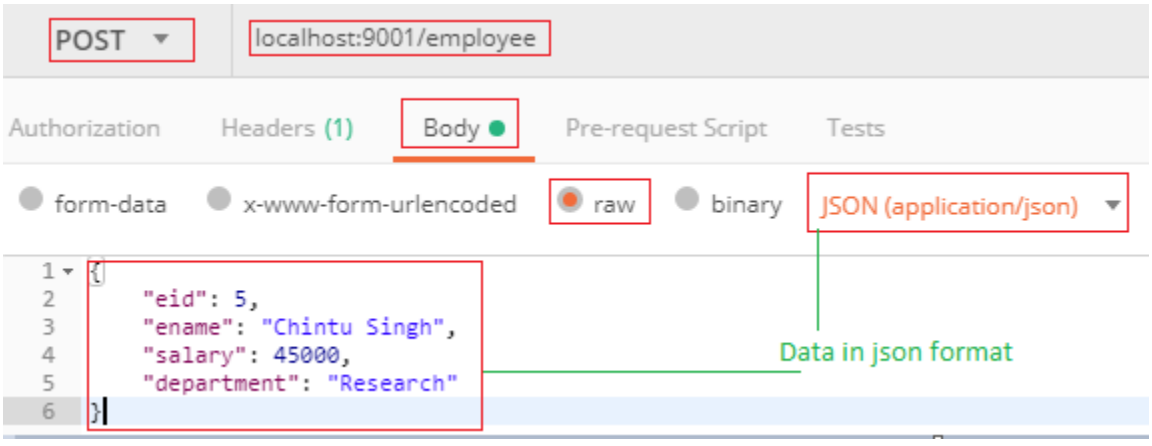
- We can save data also using postman as below:



- Note: We need to add annotation @RequestBody to PostRequest save method if we want to supply row data to save.
  Example:

```
@PostMapping("/employee")
public Employee saveEmployee(@RequestBody Employee employee) {
    employeeRepo.save(employee);
    return employee;
}
```

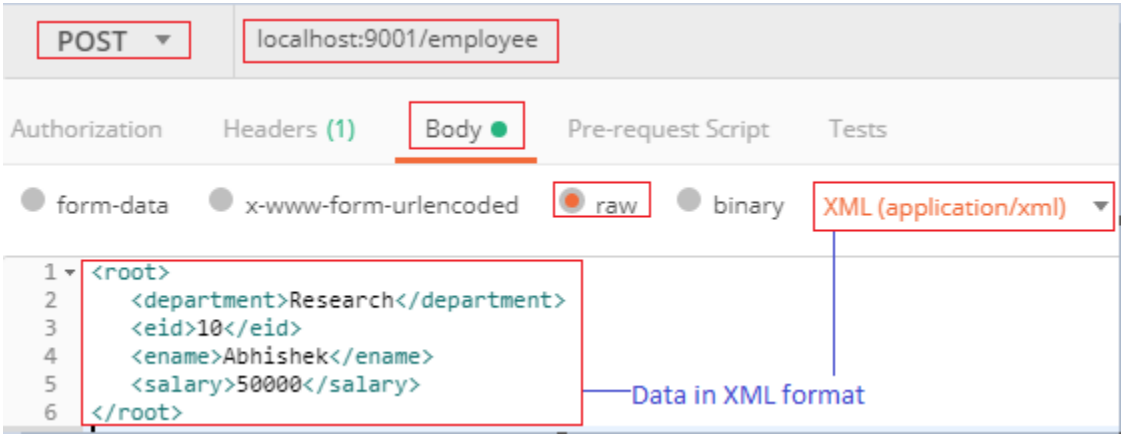
- Now, we need to choose Data format as row and choose type “JSON” as below.



- We can also control which format of data to consume (i.e., save here) by mentioning *consumes* property of @PostMapping annotation of save method as below:

```
@PostMapping(path="/employee",consumes={"application/xml"})
public Employee saveEmployee(@RequestBody Employee employee) {
    employeeRepo.save(employee);
    return employee;
}
```

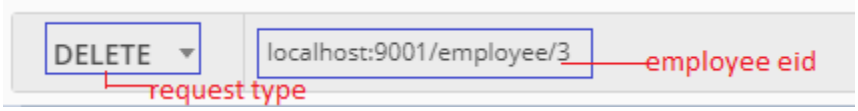
- Now , we need to pass data in XML format to save:



- To delete data via request, we need to write a method in the controller.
  Example:

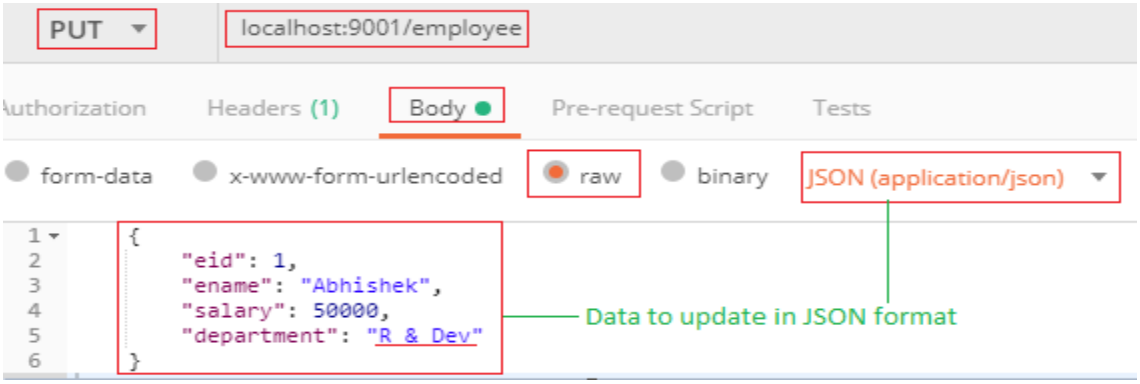
```
@DeleteMapping("/employee/{eid}")
public String deleteEmployee(@PathVariable Integer eid) {
    Employee employee = employeeRepo.getOne(eid);
    employeeRepo.delete(employee);
    return "deleted record for "+eid;
}
```

  Choose request type “DELETE” in postman



- To edit a record via request, we need to write a method in the controller.
  Example:

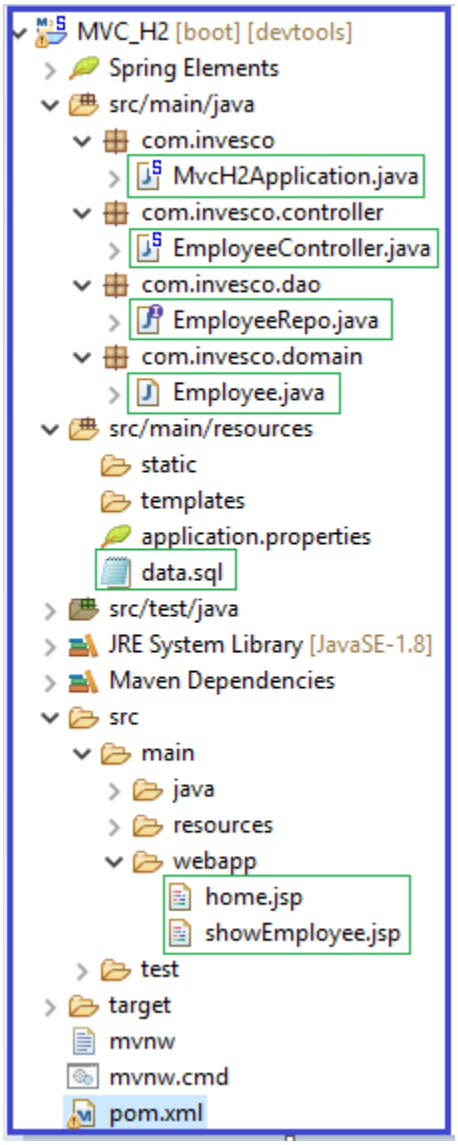
```
@PutMapping(path="/employee")
public Employee saveOrUpdateEmployee(@RequestBody Employee employee) {
       employeeRepo.save(employee);
       return employee;
}
```



*** We don’t need this controller at all if we use Spring Data Rest***

Complete Code

Project Structure:




```java
@SpringBootApplication
public class MvcH2Application {

    public static void main(String[] args) {
        SpringApplication.run(MvcH2Application.class, args);
    }
}
```

MvcH2Application.java

```java
public interface EmployeeRepo extends
JpaRepository<Employee, Integer>{
    List<Employee> findByDepartment(String department);
    List<Employee> findByEidGreaterThan(Integer eid);
    @Query("FROM Employee WHERE eid>?1 ORDER BY ename DESC")
    List<Employee>findByEidSorted(Integer eid);
}
```

EmployeeRepo.java

```java
@Entity
public class Employee {
    @Id
    private Integer eid;
    private String ename;
    private Double salary;
    private String department;

    public Employee() {
    }
    //Constructor using fields

    // Getters and Setters

    //toString()
}
```

Employee.java

```xml
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.6</version>
</dependency>

<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.10</version>
</dependency>
```

pom.xml (Extra dependencies)

```java
@RestController
public class EmployeeController {

    @Autowired
    EmployeeRepo employeeRepo;

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

//  @PostMapping("/addEmployee")
//  public String addEmployee(Employee employee) {
//      employeeRepo.save(employee);
//      return "home.jsp";
//  }

//  @GetMapping("/getEmployee")
//  public ModelAndView getEmployee(@RequestParam Integer eid) {
//      ModelAndView mv=new ModelAndView("showEmployee.jsp");
//      Employee employee=employeeRepo.findById(eid).orElse(new Employee());
//      System.out.println(employeeRepo.findByDepartment("Research"));
//      System.out.println(employeeRepo.findByEidGreaterThan(2));
//      System.out.println(employeeRepo.findByEidSorted(2));
//      mv.addObject(employee);
//      return mv;
//  }

    @PostMapping(path="/employee",consumes={"application/xml"})
    public Employee saveEmployee(@RequestBody Employee employee) {
        employeeRepo.save(employee);
        return employee;
    }

    @GetMapping(path="/employee")//,produces= {"application/xml"})
    public List<Employee> getAlians() {
        return employeeRepo.findAll();
    }

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

    @DeleteMapping("/employee/{eid}")
    public String deleteEmployee(@PathVariable Integer eid) {
        Employee employee = employeeRepo.getOne(eid);
        employeeRepo.delete(employee);
        return "deleted record for "+eid;
    }

    @PutMapping(path="/employee")
    public Employee saveOrUpdateEmployee(@RequestBody Employee employee) {
        employeeRepo.save(employee);
        return employee;
    }
```

EmployeeController.java

```
server.port=9001
spring.h2.console.enabled=true
spring.datasource.platform=h2
spring.datasource.url=jdbc:h2:mem:rashmi
```

application.properties

```
insert into employee values(1,'Research','Abhishek',50000);
insert into employee values(2,'Research','Rashmi',45000);
insert into employee values(3,'Development','Kanchu',55000);
insert into employee values(4,'Marketing','Seema',65000);
```

data.sql

```
<form action="addEmployee">
    Employee ID: <input type="text" name="eid"><br>
    Employee Name: <input type="text" name="ename"><br>
    Employee Salary:<input type="text" name="salary"><br>
    Employee Department:<input type="text"
name="department"><br>
    <input type="submit">
</form>
<form action="getEmployee">
    Employee ID:<input type="text" name="eid"><br>
    <input type="submit">
</form>
```

home.jsp

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false"%>
</head>
    <body>
        ${employee}
    </body>
</html>
```

showEmployee.jsp