

## Table of Contents

1.	Spring Boot Maven Project .....	2
2.	Understand web environment and H2 database .....	2
3.	Rest Controller .....	3
4.	HTTP METHODS Mapping .....	4
5.	Prepare model .....	6
6.	Prepare repository .....	7
7.	Prepare service layer .....	8
8.	Prepare Seed Data .....	9
9.	Expose REST API GET for Employees .....	10
10.	Handle error message : ResponseEntity .....	11
11.	REST API – POST .....	11
12.	REST API – DELETE .....	11
13.	REST API – PUT .....	12

## 1. Spring Boot Maven Project

### 1.1. Create a Spring boot maven project with the following dependencies from start.spring.io

The screenshot shows the Spring Initializr web form. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.2.3' is selected. The 'Project Metadata' section shows: Group: com.mvc, Artifact: SpringMVCDemo, Name: SpringMVCDemo, Description: Demo project for Spring Boot, Package name: com.mvc.SpringMVCDemo, Packaging: Jar, and Java: 17. On the right, the 'Dependencies' section shows selected dependencies: H2 Database (SQL), Spring Data JPA (SQL), Spring Boot DevTools (DEVELOPER TOOLS), and Spring Web (WEB). Each dependency has a brief description.

- 1.2. Unzip the project and open in the editor
- 1.3. The embedded tomcat with spring boot web includes a light weight server which is the tomcat core and is capable of processing HTTP requests and send JSON as a response
- 1.4. Applications that use devtools will automatically restart whenever files on the classpath change

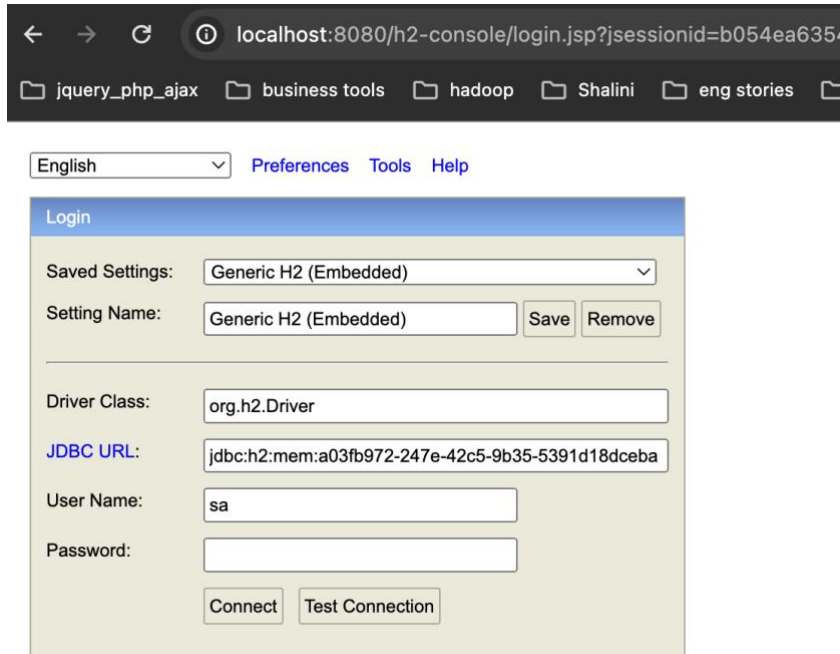
## 2. Understand web environment and H2 database

### 2.1. Run the main method and observe the console.

```
2024-03-06 15:26:50.743 INFO 75368 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 102 ms. Found 0 JDBC repository interfaces.
2024-03-06 15:26:50.765 INFO 75368 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository configuration mode
2024-03-06 15:26:50.766 INFO 75368 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-03-06 15:26:50.807 INFO 75368 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 36 ms. Found 6 JPA repository interfaces.
2024-03-06 15:26:51.914 INFO 75368 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2024-03-06 15:26:51.930 INFO 75368 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-03-06 15:26:51.930 INFO 75368 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2024-03-06 15:26:52.310 INFO 75368 --- [ restartedMain] org.apache.jasper.servlet.TldScanner : At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation time.
2024-03-06 15:26:52.330 INFO 75368 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-03-06 15:26:52.330 INFO 75368 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2855 ms
2024-03-06 15:26:52.418 INFO 75368 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-03-06 15:26:52.615 INFO 75368 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-03-06 15:26:52.615 INFO 75368 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:a031b972-247e-42c5-9b35-5391d18dceba'
2024-03-06 15:26:53.000 INFO 75368 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-03-06 15:26:53.002 INFO 75368 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.11.Final
2024-03-06 15:26:53.193 INFO 75368 --- [ restartedMain] o.hibernate.annotations.common.Version : HCN000000: Hibernate Commons Annotations {5.1.2.Final}
2024-03-06 15:26:53.293 INFO 75368 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
```

### 2.2. Notice tomcat is running on port 8080

- 2.3. H2 console is available at /h2-console with the url starting with jdbc:h2:mem:<some id>. Default username is sa and no password.  
This id is random and keeps changing whenever the server is restarted
- 2.4. Open browser and type in the url <http://localhost:8080/h2-console> and copy paste the h2 url from the console as shown in below screen:



- 2.5. Since url keeps on changing lets override the default url and update the application.properties as follows:
 

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.jpa.show-sql=true
#spring.jpa.generate-ddl=true
```
- 2.6. Now stop the server and restart to see the output in console for the changed url for h2-console.  
Login to h2 database on the browser using **jdbc:h2:mem:testdb**
- 2.7. Tomcat server bby default runs on port 8080. To change the port add below in application.properties file
 

```
server.port=8081
```
- 2.8. Stop the server and restart, you will notice now tomcat is available at port 8081. Verify by going on the browser and typing <http://localhost:8081/h2-console>

### 3. Rest Controller

- 3.1. Create a class HelloRestController as follows:

```
@RestController // @Controller + @ResponseBody
public class HelloRestController{

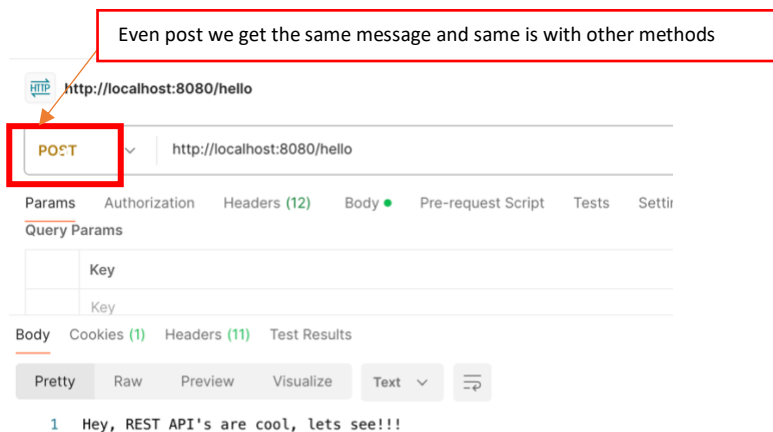
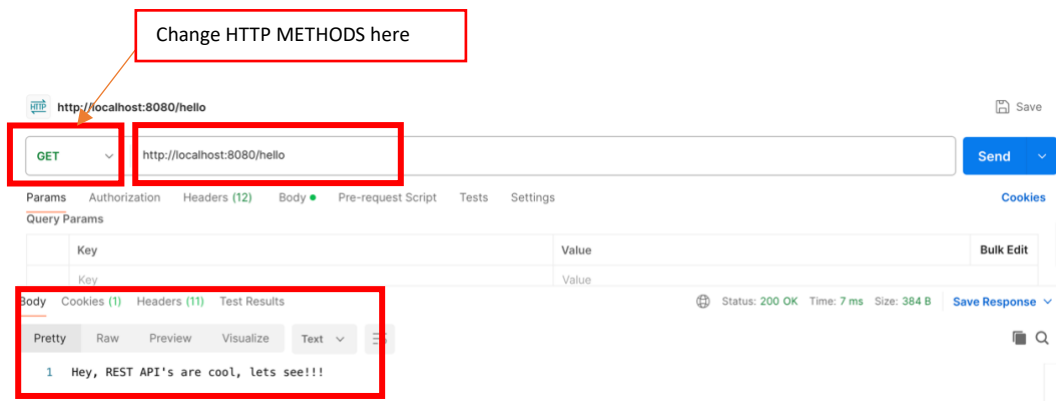
    @RequestMapping("/hello")
```

```

    public String greet()
    {
        return "Hey, REST API's are cool, lets see!!! ";
    }
}

```

- 3.2. Run the class with main method and open the browser at url:  
<http://localhost:8080/hello>
- 3.3. The output should be the message as returned by the greet()
- 3.4. Open postman and type url in address bar, select various HTTP methods on the left as shown below and you should get the same message as output for all HTTP methods:



## 4. HTTP METHODS Mapping

- 4.1. Let's add specific methods to map for specific HTTP requests since methods are used as follows:
  - 4.1.1. GET: fetch data
  - 4.1.2. PUT : update data
  - 4.1.3. POST : insert data
  - 4.1.4. DELETE : delete data
- 4.2. Update HelloRestController as follows:

```

@GetMapping("/get")
public String send()
{
    return "FROM GET ONLY";
}

```

```

}
@PostMapping("/add")
public String post()
{
    return "FROM POST ONLY";
}
@PutMapping("/update")
public String put()
{
    return "FROM PUT ONLY";
}
@DeleteMapping("/delete")
public String delete()
{
    return "FROM DELETE ONLY";
}

```

4.3. Now restart the server and test above methods via postman. The urls for request will be as follows:

- 4.3.1. Change the HTTP method in postman to GET and type in below url to see message from GetMapping  
<http://localhost:8080/get>
- 4.3.2. Change the HTTP method in postman to POST and type in below url to see message from PostMapping  
<http://localhost:8080/add>
- 4.3.3. Change the HTTP method in postman to PUT and type in below url to see message from PutMapping  
<http://localhost:8080/update>
- 4.3.4. Change the HTTP method in postman to DELETE and type in below url to see message from DeleteMapping  
<http://localhost:8080/delete>

4.4. Conventionally, REST URI should only contain noun and verbs are passed as part of HTTP method. Modify the above class to have a single URI for the respective HTTP method:

```

@RestController()
@RequestMapping("/api")
public class HelloRestController{

    ....
    @GetMapping()
    public String send()
    {
        return "FROM GET ONLY";
    }
    @PostMapping()
    public String post()
    {
        return "FROM POST ONLY";
    }
    @PutMapping()
    public String put()
    {
        return "FROM PUT ONLY";
    }
    @DeleteMapping()
    public String delete()

```

```

        {
            return "FROM DELETE ONLY";
        }
    }
}

```

- 4.5. Now restart the server, type below url in postman and just by changing the HTTP methods, respective messages will be displayed.

<http://localhost:8080/api>

## 5. Prepare model

- 5.1. Create Employee class as follows:

```

import jakarta.persistence.*;
import org.antlr.v4.runtime.misc.NotNull;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @NotNull
    private int eid;
    @Column(nullable = true)
    private String ename;
    private String email;
    private String phone;

    private String password;

    public Employee() {
    }

    public Employee(int eid, String ename, String email, String phone, String password) {
        this.eid = eid;
        this.ename = ename;
        this.email = email;
        this.phone = phone;
        this.password = password;
    }
    // getters and setters
    // constructor

    public int getEid() {
        return eid;
    }

    public void setEid(int eid) {
        this.eid = eid;
    }

    public String getEname() {
        return ename;
    }
}

```

```

    }

    public void setName(String ename) {
        this.ename = ename;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Employee{" +
            "eid=" + eid +
            ", ename='" + ename + '\'' +
            ", email='" + email + '\'' +
            ", phone='" + phone + '\'' +
            ", password='" + password + '\'' +
            '}';
    }
}

```

## 6. Prepare repository

### 6.1. Create EmployeeRepository as follows:

```

import com.mvc.SpringBootMVCDemo.entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

    public Employee findByEmail(String email);
}

```

```

// select password from employee where email=?
@Query("select password from Employee where email=:email")
public String findPasswordByEmployeeEmail(String email);
}

```

## 7. Prepare service layer

7.1. Create EmployeeService as follows:

```

import com.mvc.SpringBootMVCDemo.entity.Employee;
import com.mvc.SpringBootMVCDemo.exception.InvalidCredentialsException;
import com.mvc.SpringBootMVCDemo.repo.EmployeeRepository;
import jakarta.persistence.EntityExistsException;
import jakarta.persistence.EntityNotFoundException;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    public EmployeeService() {
        System.out.println("emp service");
    }
    public Employee insertEmployee(Employee employee) {
        if (!employeeRepository.existsById(employee.getId())) {
            Employee savedEmployee = this.employeeRepository.save(employee);
            return savedEmployee;
        }
        throw new EntityExistsException("Employee with id "+employee.getId()+" already exists, hence cannot save");
    }

    public Employee findEmployeeByEmail(String email){
        return this.employeeRepository.findByEmail(email);
    }

    public Employee getEmployeeById(int id) {
        System.out.println("Emp service "+id);
        return this.employeeRepository
            .findById(id).orElseThrow(()-> new EntityNotFoundException("Employee "+id+" not found"));
    }
}

```



```

public List<Employee> getEmployees(){
    List<Employee> employees = new ArrayList<Employee>();
    this.employeeRepository.findAll().forEach(employees::add);
    return employees;
}

public Page<Employee> getFilteredEmployees(Integer pageno, Integer size){

    Pageable pageable = PageRequest.of(pageno, size, Sort.by(Sort.Direction.DESC, "email"));
    List<Employee> employees = new ArrayList<Employee>();
    return this.employeeRepository.findAll(pageable);

}

public Employee updateEmployee(Employee employee) {
    if(!this.employeeRepository.existsById(employee.getId()))
        throw new EntityNotFoundException("Employee "+employee.getId()+" not found and cannot
be updated");
    return this.employeeRepository.save(employee);
}

@Transactional
public boolean deleteEmployee(int eid) {
    if(!this.employeeRepository.existsById(eid))
        throw new EntityNotFoundException("Employee "+eid+" not found and cannot be deleted");
    this.employeeRepository.deleteById(eid);
    return true;
}

public boolean loginEmployee(String email, String password) throws InvalidCredentialsException
{

    String pwd = this.employeeRepository.findPasswordByEmployeeEmail(email);
    if(pwd!=null)
    {
        if(pwd.equals(password))
            return true;
    }
    throw new InvalidCredentialsException("Invalid credentials, Please try again");
}
}

```

## 8. Prepare Seed Data

### 8.1. Add below code in class with main method to create seed data

```

@Autowired
private EmployeeService employeeService;

@Bean
public void initialize()
{
    Employee emp = new Employee();
    emp.setName("Sia");
    emp.setEmail("sia@test.com");
    emp.setPassword("sia123");
}

```

```

        emp.setPhone("9898989898");

        Employee e = employeeService.insertEmployee(emp);

        Employee em = new Employee();
        em.setName("John");
        em.setEmail("john@test.com");
        em.setPassword("john1235");
        em.setPhone("7654323456");

        Employee e = employeeService.insertEmployee(em);

    }

```

## 9. Expose REST API GET for Employees

9.1. Create EmployeeRestController as follows:

```

@RestController
@RequestMapping("/employees")
public class EmployeeRestController {

    @Autowired
    private EmployeeService employeeService;

    @GetMapping()
    public List<Employee> getAllEmployees()
    {
        return this.employeeService.getEmployees();
    }

    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable("id") int id)
    {
        return this.employeeService.getEmployeeById(id);
    }
}

```

9.2. Restart the server and get employees data by sending request to :

// Will return all employees  
<http://localhost:8080/employees>

// Will return employee by id  
<http://localhost:8080/employees/1>

9.3. Trying passing the id of an employee that does not exist and you will see an exception which is not the right way of sending an error response:

<http://localhost:8080/employees/10>.

## 10. Handle error message : ResponseEntity

10.1. Instead of returning Employee object, better to wrap Object instance within the ResponseEntity.

10.2. ResponseEntity : represents the whole HTTP response: status code, headers, and body. As a result, we can use it to fully configure the HTTP response.

If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

ResponseEntity is a generic type. Consequently, we can use any type as the response body:

10.3. Modify the getEmployeeById message to change the response type

```
@GetMapping(path = "/{eid}")
public ResponseEntity<Object> getEmployeeById(@PathVariable int eid) {
    System.out.println("Emp id "+eid);
    Employee emp = this.employeeService.getEmployeeById(eid);
    return ResponseEntity.ok(emp);

    try {
        Employee emp = this.employeeService.getEmployeeById(eid);
        return ResponseEntity.ok(emp);
    }
    catch(Exception e)
    {
        System.out.println(e.getMessage());
        //return ResponseEntity.noContent().build();
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body(new AppResponse(Messages.FAILURE,e.getMessage()));
    }
}
```

10.4. Restart the server and see for the response in postman

## 11. REST API – POST

11.1. Add below method in EmployeeRestController

```
@PostMapping
public ResponseEntity<Object> insertEmployee(@RequestBody Employee employee)
{
    System.out.println(employee);
    if (employeeService.insertEmployee(employee))
        return ResponseEntity.status(HttpStatus.CREATED).build()

    return ResponseEntity.badRequest();
}
```

11.2. Restart the server and test for post

## 12. REST API – DELETE

12.1. Add below method in EmployeeRestController

```
@DeleteMapping("/{eid}")
public Map<String, String> deleteEmployeeById(@PathVariable int eid)
{
}
```

```

        Map<String, String> map = new HashMap<String, String>();
        if(this.employeeService.deleteEmployee(eid))
            map.put("SUCESS", eid+ " deleted");
        else
            map.put("ERROR", eid+ " count not be deleted");
        return map;
    }

```

12.2. Restart the server and test for delete

## 13. REST API – PUT

13.1. Add below method in EmployeeRestController

```

    @PutMapping
    public ResponseEntity<Object> insertEmployee(@RequestBody Employee employee)
    {
        System.out.println(employee);
        if (employeeService.updateEmployee(employee))
            return ResponseEntity.ok(employee)
        return ResponseEntity.badRequest();
    }

```

13.2. Restart the server and test for post