Spring Boot & H2 - Quick Guide

Spring Boot & H2 - Overview

What is H2?

H2 database is an open source, embedded and in memory relational database management system. It is written in Java and provides a client/server application. It stores data in system memory instead of disk. Once program is closed, data is also lost. An in memory database is used when we don't want to persist the data and unit test the overall functionality. Some of the other popular in memory databases are HSQLDB or HyperSQL Database and Apache Derby. H2 is the most popular one among other embedded databases.

Advantages of H2 Database

Following is the list of advantages that H2 provides −

- **No configuration** − Spring Boot intrinsically supports H2 and no extra configuration required to configure H2 database.
- **Easy to Use** − H2 Database is very easy to use.
- **Lightweight and Fast** − H2 database is very lightweight and being in memory, it is very fast.
- **Switch configurations** − Using profiles, you can easily switch between production level database and in-memory database.

- **Supports Standard SQL and JDBC** − H2 database supports almost all the features of Standard SQL and operations of JDBC.
- **Web Based Console** − H2 Database can be managed by its web based console application.

Configuring H2 Database

Add H2 Database as maven dependency and that's it.

```
<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

Although, spring boot configures H2 database automatically. We can override the default configurations by specifying them in application.properties as shown below.

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Persisting H2 Data

If persistent storage is needed than add the following configuration in application.properties.

```
spring.datasource.url=jdbc:h2:file:/data/database
spring.datasource.url=jdbc:h2:C:/data/database
```

Spring Boot & H2 - Environment Setup

This chapter will guide you on how to prepare a development environment to start your work with Spring Framework. It will also teach you how to set up JDK, Tomcat and Eclipse on your machine before you set up Spring Framework −

Step 1 - Setup Java Development Kit (JDK)

Java SE is available for download for free. To download click here , please download a version compatible with your operating system.

Follow the instructions to download Java, and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Setting Up the Path for Windows 2000/XP

Assuming you have installed Java in c:\Program Files\java\jdk directory −

- Right-click on 'My Computer' and select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now, edit the 'Path' variable and add the path to the Java executable directory at the end of it. For example, if the path is currently set to **C:\Windows\System32**, then edit it the following way

```
C:\Windows\System32;c:\Program Files\java\jdk\bin
```

Setting Up the Path for Windows 95/98/ME

Assuming you have installed Java in c:\Program Files\java\jdk directory −

- Edit the 'C:\autoexec.bat' file and add the following line at the end −

```
SET PATH=%PATH%;C:\Program Files\java\jdk\bin
```

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

For example, if you use bash as your shell, then you would add the following line at the end of your **.bashrc** −

```
export PATH=/path/to/java:$PATH'
```

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, you will have to compile and run a simple program to confirm that the IDE knows where you have installed Java. Otherwise, you will have to carry out a proper setup as given in the document of the IDE.

Step 2 - Setup Eclipse IDE

All the examples in this tutorial have been written using Eclipse IDE. So we would suggest you should have the latest version of Eclipse installed on your machine.

To install Eclipse IDE, download the latest Eclipse binaries from www.eclipse.org/downloads . Once you download the installation, unpack the binary

distribution into a convenient location. For example, in C:\eclipse on Windows, or /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.

Eclipse can be started by executing the following commands on Windows machine, or you can simply double-click on eclipse.exe

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine −

```
$/usr/local/eclipse/eclipse
```

Step 3 - Setup m2eclipse

M2Eclipse is eclipse plugin which is very useful integration for Apache Maven into the Eclipse IDE. We are using maven in this tutorial to build spring boot project and examples are run within eclipse using m2eclipse.

Install the latest M2Eclipse release by using the Install New Software dialog in Eclipse IDE, and point it to this p2 repository: https://download.eclipse.org/technology/m2e/releases/latest/

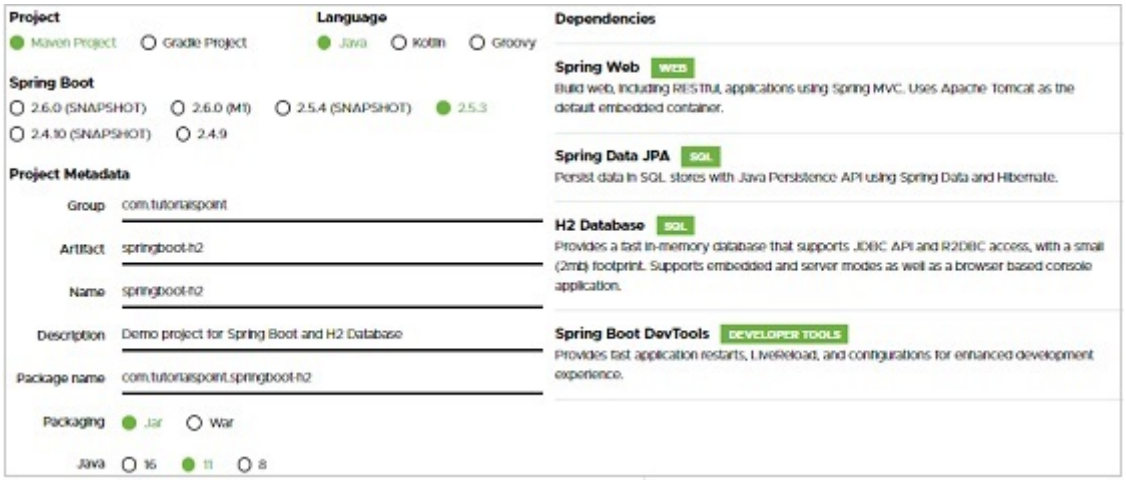
Step 4 - Setup Spring Boot Project

Now if everything is fine, then you can proceed to set up your Spring Boot. Following are the simple steps to download and install the Spring Boot Project on your machine.

- Go to spring Initializr link to create a spring boot project, https://start.spring.io/ .
- Select project as **Maven Project**.

- Select language as **Java**.
- Select Spring Boot version as **2.5.3**.
- Set Project Metadata - Group as **com.tutorialspoint**, Artifact as **springboot-h2**, name as **springboot-h2**, Description as **Demo project for Spring Boot and H2 Database** and package name as **com.tutorialspoint.springboot-h2**.
- Select packaging as **Jar**.
- Select java as **11**.
- Add dependencies as **Spring Web, Spring Data JPA, H2 Database and Spring Boot DevTools**.

Now click on GENERATE Button to generate the project structure.



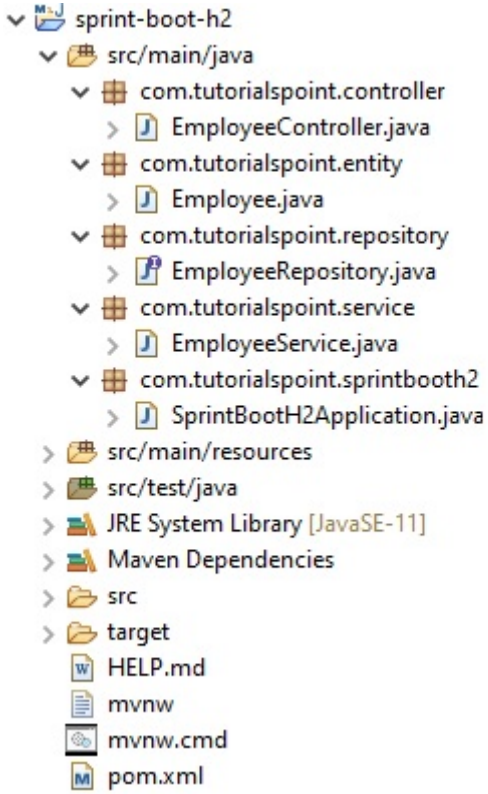
Once the maven based spring boot project is downloaded, then import the maven project into eclipse and rest eclipse will handle. It will download the maven dependencies and build the project to make it ready for further development.

Step 5 - POSTMAN for REST APIs Testing

POSTMAN is a useful tool to test REST Based APIs. To install POSTMAN, download the latest POSTMAN binaries from www.postman.com/downloads/ . Once you download the installable, follow the instructions to install and use it.

Spring Boot & H2 - Project Setup

As in previous chapter Environment Setup , we've imported the generated spring boot project in eclipse. Now let's create the following structure in **src/main/java** folder.



- **com.tutorialspoint.controller.EmployeeController** − A REST Based Controller to

implement REST based APIs.

- **com.tutorialspoint.entity.Employee** − An entity class representing the corresponding table in database.
- **com.tutorialspoint.repository.EmployeeRepository** − A Repository Interface to implement the CRUD operations on the database.
- **com.tutorialspoint.service.EmployeeService** − A Service Class to implement the business opearations over repository functions.
- **com.tutorialspoint.springbooth2.SprintBootH2Application** − A Spring Boot Application class.

SprintBootH2Application class is already present. We need to create the above packages and relevant classes and interface as shown below −

Entity - Entity.java

Following is the default code of Employee. It represents a Employee table with id, name, age and email columns.

```
package com.tutorialspoint.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table
public class Employee {
   @Id
   @Column
```

```
private int id;
@Column
private String name;
@Column
private int age;
@Column
private String email;
public int getId() {
   return id;
}
public void setId(int id) {
   this.id = id;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}
public int getAge() {
   return age;
}
public void setAge(int age) {
   this.age = age;
}
public String getEmail() {
   return email;
}
public void setEmail(String email) {
   this.email = email;
```

```
    }
}
```

Repository - EmployeeRepository.java

Following is the default code of Repository to implement CRUD operations on above entity, Employee.

```
package com.tutorialspoint.repository;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import com.tutorialspoint.entity.Employee;
@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Integer>
}
```

Service - EmployeeService.java

Following is the default code of Service to implement operations over repository functions.

```
package com.tutorialspoint.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.repository.EmployeeRepository;
@Service
public class EmployeeService {
   @Autowired
```

```
   EmployeeRepository repository;
   public Employee getEmployeeById(int id) {
      return null;
   }
   public List<Employee> getAllEmployees(){
      return null;
   }
   public void saveOrUpdate(Employee employee) {
   }
   public void deleteEmployeeById(int id) {
   }
}
```

Controller - EmployeeController.java

Following is the default code of Controller to implement REST APIs.

```
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
```

```
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return null;
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return null;;
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
   }
   @PostMapping("/employee")
   public void addEmployee(@RequestBody Employee employee) {
   }
   @PutMapping("/employee")
   public void updateEmployee(@RequestBody Employee employee) {
   }
}
```

Application - SprintBootH2Application.java

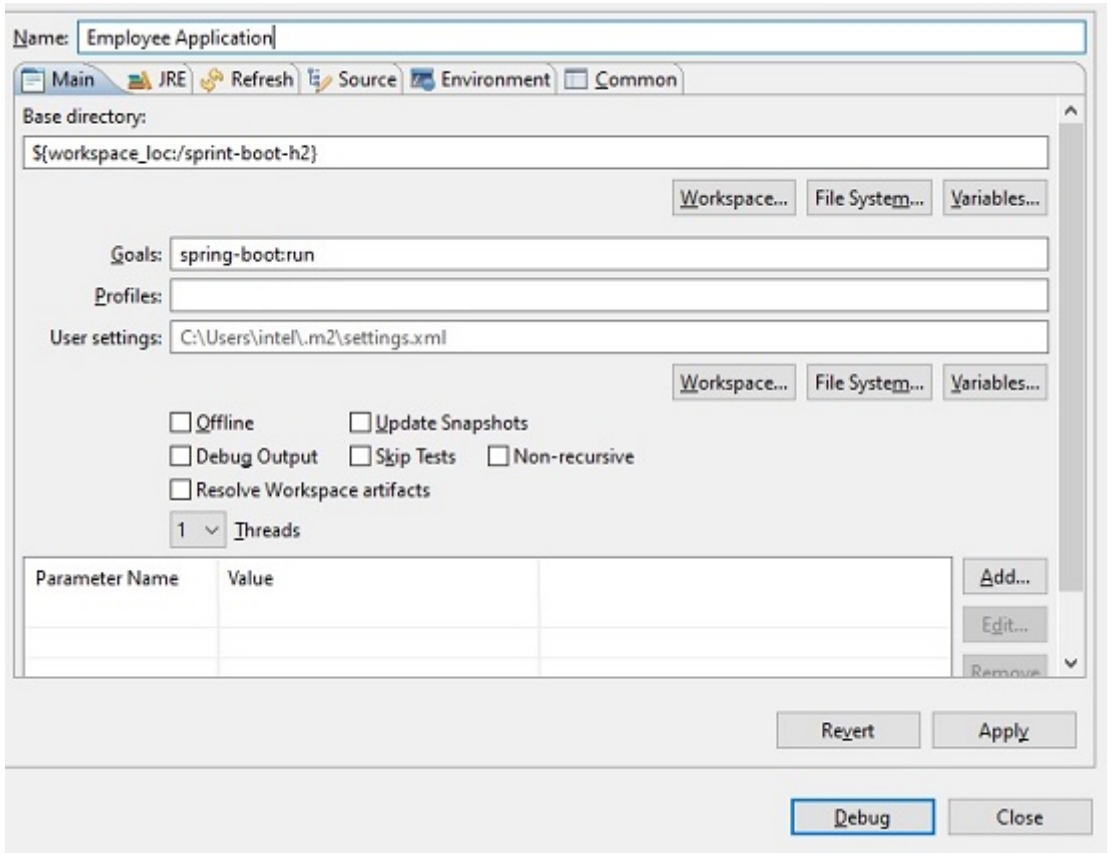
Following is the updated code of Application to use above classes.

```
package com.tutorialspoint.sprintbooth2;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
```

```java
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
@ComponentScan({"com.tutorialspoint.controller","com.tutorialspoint.service"})
@EntityScan("com.tutorialspoint.entity")
@EnableJpaRepositories("com.tutorialspoint.repository")
@SpringBootApplication
public class SprintBootH2Application {
   public static void main(String[] args) {
      SpringApplication.run(SprintBootH2Application.class, args);
   }
}
```

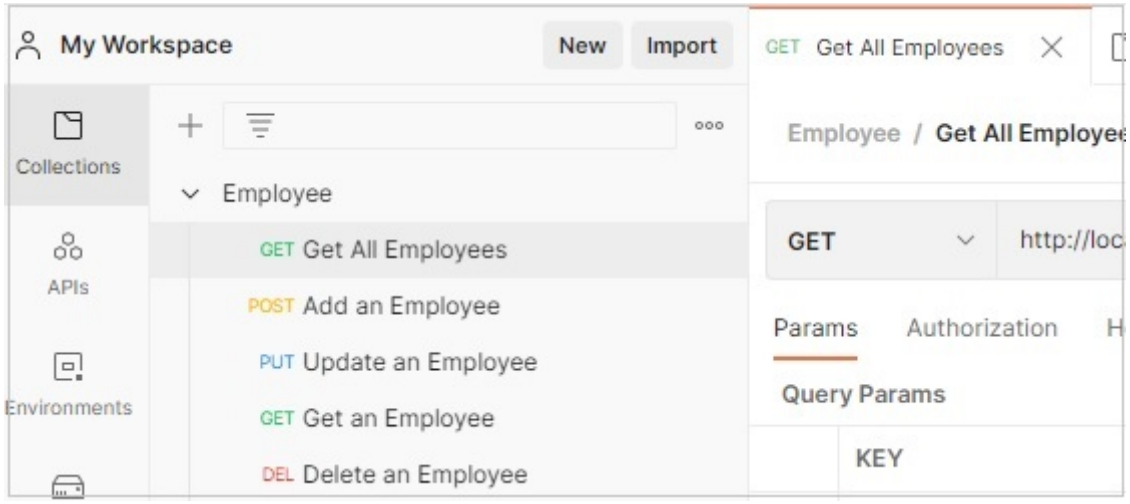
Run/Debug Configuration

Create following **maven configuration** in eclipse to run the springboot application with goal **spring-boot:run**. This configuration will help to run the REST APIs and we can test them using POSTMAN.



Spring Boot & H2 - REST APIs

As in previous chapter Application Setup , we've created the required files in spring boot project. Now create the following collection in POSTMAN to test the REST APIs.



- **GET Get All Employees** − A GET request to return all the employees.
- **POST Add an Employee** − A POST request to create an employee.
- **PUT Update an Employee** − A PUT request to update an existing employee.
- **GET An Employee** − A GET request to get an employee identified by its id.
- **Delete An Employee** − A Delete request to delete an employee identified by its id.

GET All Employees

Set the following parameters in POSTMAN.

- HTTP Method − **GET**
- URL − **http://localhost:8080/emp/employees**

Add an Employee

Set the following parameters in POSTMAN.

- HTTP Method − **POST**

- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}
```

Update an Employee

Set the following parameters in POSTMAN.

- HTTP Method − **PUT**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie.roberts@gmail.com"
}
```

GET An Employees

Set the following parameters in POSTMAN.

- HTTP Method − **GET**
- URL - **http://localhost:8080/emp/employee/1** − Where 1 is the employee id

Delete An Employees

Set the following parameters in POSTMAN.

- HTTP Method − **DELETE**
- URL - **http://localhost:8080/emp/employee/1** − Where 1 is the employee id

Spring Boot & H2 - Console

As in previous chapter Application Setup , we've created the required files in spring boot project. Now let's update the application.properties lying in **src/main/resources** and **pom.xml** to use a different version of **maven-resources-plugin**.

application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
```

pom.xml

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

```
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-resources-plugin</artifactId>
         <version>3.1.0</version>
      </plugin>
   </plugins>
</build>
...
```

Run the application

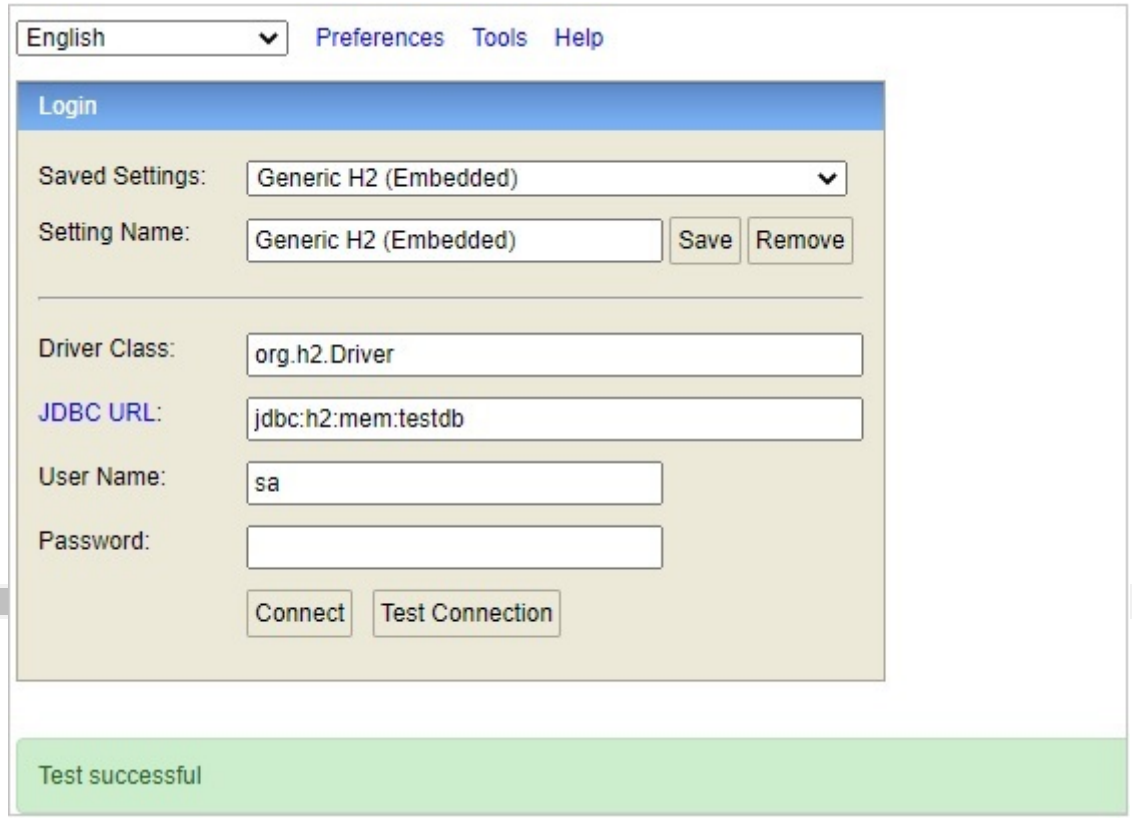
In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

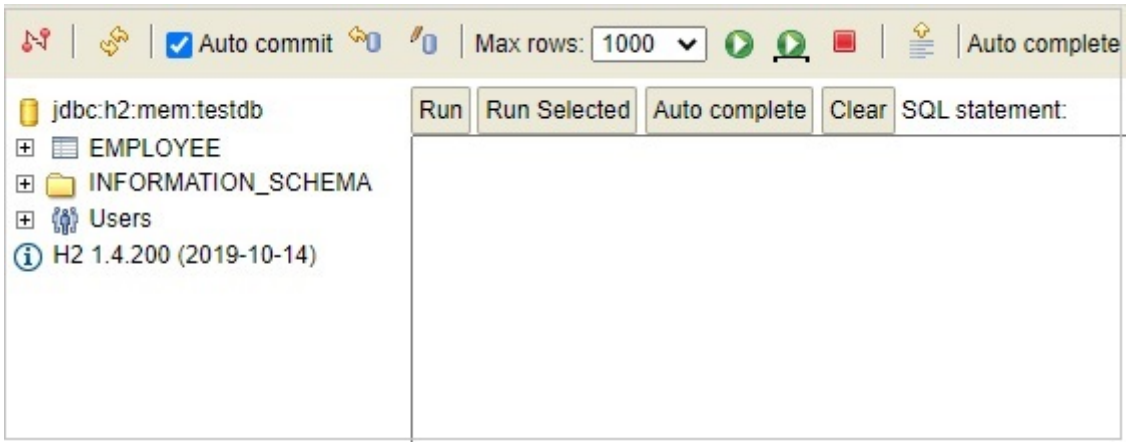
```
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< com.tutorialspoint:sprint-boot-h2 >------------------
[INFO] Building sprint-boot-h2 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.2)

...
2021-07-24 20:51:11.347  INFO 9760 --- [  restartedMain] o.s.b.w.embedded.tomcat.Tom
```

```
: Tomcat initialized with port(s): 8080 (http)
...
2021-07-24 20:51:11.840  INFO 9760 --- [  restartedMain] o.s.b.a.h2.H2ConsoleAutoCon
: H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
...
2021-07-24 20:51:14.805  INFO 9760 --- [  restartedMain] o.s.b.w.embedded.tomcat.Tom
: Tomcat started on port(s): 8080 (http) with context path ''
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, open **localhost:8080/h2-console** in a browser and click on Test Connection to verify the database connection.



Click on Connect button and H2 database window will appear as shown below −



Spring Boot & H2 - Add Record

Let's now update the project created so far to prepare a complete Add Record API and test it.

Update Service

```
// Use repository.save() to persist Employee entity in database
public void saveOrUpdate(Employee employee) {
   repository.save(employee);
}
```

EmployeeService

```
package com.tutorialspoint.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.repository.EmployeeRepository;
@Service
public class EmployeeService {
   @Autowired
   EmployeeRepository repository;
   public Employee getEmployeeById(int id) {
      return null;
   }
   public List<Employee> getAllEmployees(){
      return null;
   }
   public void saveOrUpdate(Employee employee) {
      repository.save(employee);
   }
   public void deleteEmployeeById(int id) {
   }
}
```

Update Controller

```
// Use service.saveOrUpdate() to persist Employee entity in database
@PostMapping("/employee")
public void addEmployee(@RequestBody Employee employee) {
   employeeService.saveOrUpdate(employee);
}
```

EmployeeController

```java
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return null;
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return null;;
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
   }
   @PostMapping("/employee")
   public void addEmployee(@RequestBody Employee employee) {
```

```
        employeeService.saveOrUpdate(employee);
    }
    @PutMapping("/employee")
    public void updateEmployee(@RequestBody Employee employee) {
    }
}
```

Run the application

In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

```
[INFO] Scanning for projects...
...
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, Use Postman to make a POST request −

Set the following parameters in POSTMAN.

- HTTP Method − **POST**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
```

```
   "name": "Julie",
   "email": "julie@gmail.com"
}
```

Click on Send Button and check the response status to be OK. Now open H2-Console and verify the inserted record using following query −

```
Select * from Employee;
```

It should display following result −

```
ID    AGE    EMAIL              NAME
1     35   julie@gmail.com      Julie
```

Spring Boot & H2 - Get Record

Let's now update the project created so far to prepare a complete Get Record API and test it.

Update Service

```
// Use repository.findById() to get Employee entity by Id
public Employee getEmployeeById(int id) {
   return repository.findById(id).get();
}
```

EmployeeService

```
package com.tutorialspoint.service;
import java.util.ArrayList;
```

```java
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.repository.EmployeeRepository;
@Service
public class EmployeeService {
   @Autowired
   EmployeeRepository repository;
   public Employee getEmployeeById(int id) {
      return repository.findById(id).get();
   }
   public List<Employee> getAllEmployees(){
      return null;
   }
   public void saveOrUpdate(Employee employee) {
      repository.save(employee);
   }
   public void deleteEmployeeById(int id) {
   }
}
```

Update Controller

```java
// Use service.getEmployeeById() to get Employee entity from database
@GetMapping("/employee/{id}")
public Employee getEmployee(@PathVariable("id") int id) {
   return employeeService.getEmployeeById(id);
}
```

EmployeeController

```
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return null;
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return employeeService.getEmployeeById(id);
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
   }
```

```
    @PostMapping("/employee")
    public void addEmployee(@RequestBody Employee employee) {
       employeeService.saveOrUpdate(employee);
    }
    @PutMapping("/employee")
    public void updateEmployee(@RequestBody Employee employee) {
    }
}
```

Run the application

In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

```
[INFO] Scanning for projects...
...
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, Use Postman to make a POST request to add a record first.

Set the following parameters in POSTMAN.

- HTTP Method − **POST**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}
```


Click on Send Button and check the response status to be OK. Now make a GET Request to get that record.

Set the following parameters in POSTMAN.

- HTTP Method − **GET**
- URL − **http://localhost:8080/emp/employee/1**

Click the send button and verify the response.


```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}
```


Spring Boot & H2 - Get All Records

Let's now update the project created so far to prepare a complete Get All Records API and test it.

Update Service

```
// Use repository.findAll() to get all Employee records
public List<Employee> getAllEmployees(){
   List<Employee> employees = new ArrayList<Employee>();
   repository.findAll().forEach(employee -> employees.add(employee));
   return employees;
}
```

EmployeeService

```
package com.tutorialspoint.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.repository.EmployeeRepository;
@Service
public class EmployeeService {
   @Autowired
   EmployeeRepository repository;
   public Employee getEmployeeById(int id) {
      return repository.findById(id).get();
   }
   public List<Employee> getAllEmployees(){
      List<Employee> employees = new ArrayList<Employee>();
      repository.findAll().forEach(employee -> employees.add(employee));
      return employees;
   }
   public void saveOrUpdate(Employee employee) {
```

```
      repository.save(employee);
   }
   public void deleteEmployeeById(int id) {
   }
}
```

Update Controller

```
// Use service.getAllEmployees() to get a list of employees from database
@GetMapping("/employees")
public List<Employee> getAllEmployees(){
   return employeeService.getAllEmployees();
}
```

EmployeeController

```
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
```

```
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return employeeService.getAllEmployees();
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return employeeService.getEmployeeById(id);
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
   }
   @PostMapping("/employee")
   public void addEmployee(@RequestBody Employee employee) {
      employeeService.saveOrUpdate(employee);
   }
   @PutMapping("/employee")
   public void updateEmployee(@RequestBody Employee employee) {
   }
}
```

Run the application

In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

```
[INFO] Scanning for projects...
...
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, Use Postman to make a POST request to add a record first.

Set the following parameters in POSTMAN.

- HTTP Method − **POST**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}
```

Click on Send Button and check the response status to be OK. Now make a GET Request to get all records.

Set the following parameters in POSTMAN.

- HTTP Method − **GET**
- URL − **http://localhost:8080/emp/employees**

Click the send button and verify the response.

```
[{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}]
```

Spring Boot & H2 - Update Record

Let's now update the project created so far to prepare a complete Update Record API and test it.

Update Controller

```
// Use service.saveOrUpdate() to update an employee record
@PutMapping("/employee")
public void updateEmployee(@RequestBody Employee employee) {
   employeeService.saveOrUpdate(employee);
}
```

EmployeeController

```
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```java
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return employeeService.getAllEmployees();
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return employeeService.getEmployeeById(id);
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
      employeeService.deleteEmployeeById(id);
   }
   @PostMapping("/employee")
   public void addEmployee(@RequestBody Employee employee) {
      employeeService.saveOrUpdate(employee);
   }
   @PutMapping("/employee")
   public void updateEmployee(@RequestBody Employee employee) {
      employeeService.saveOrUpdate(employee);
```

```
    }
}
```

Run the application

In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

```
[INFO] Scanning for projects...
...
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, Use Postman to make a POST request to add a record first.

Set the following parameters in POSTMAN.

- HTTP Method − **POST**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie@gmail.com"
}
```

Click on Send Button and check the response status to be OK.

Now make a Put Request to update that records.

Set the following parameters in POSTMAN.

- HTTP Method − **PUT**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
   "email": "julie.roberts@gmail.com"
}
```

Click the send button and verify the response status to be OK.

Now make a GET Request to get all records.

Set the following parameters in POSTMAN.

- HTTP Method − **GET**
- URL − **http://localhost:8080/emp/employees**

Click the send button and verify the response.

```
[{
   "id": "1",
   "age": "35",
```

```
   "name": "Julie",
   "email": "julie.roberts@gmail.com"
}]
```

Spring Boot & H2 - Delete Record

Let's now update the project created so far to prepare a complete Delete Record API and test it.

Update Service

```
// Use repository.deleteById() to delete an Employee record
public void deleteEmployeeById(int id) {
   repository.deleteById(id);
}
```

EmployeeService

```
package com.tutorialspoint.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.repository.EmployeeRepository;
@Service
public class EmployeeService {
   @Autowired
   EmployeeRepository repository;
```

```
    public Employee getEmployeeById(int id) {
        return repository.findById(id).get();
    }
    public List<Employee> getAllEmployees(){
        List<Employee> employees = new ArrayList<Employee>();
        repository.findAll().forEach(employee -> employees.add(employee));
        return employees;
    }
    public void saveOrUpdate(Employee employee) {
        repository.save(employee);
    }
    public void deleteEmployeeById(int id) {
        repository.deleteById(id);
    }
}
```

Update Controller

```
// Use service.deleteEmployeeById() to delete an employee by id
@DeleteMapping("/employee/{id}")
public void deleteEmployee(@PathVariable("id") int id) {
    employeeService.deleteEmployeeById(id);
}
```

EmployeeController

```
package com.tutorialspoint.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
```

```java
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.service.EmployeeService;
@RestController
@RequestMapping(path = "/emp")
public class EmployeeController {
   @Autowired
   EmployeeService employeeService;
   @GetMapping("/employees")
   public List<Employee> getAllEmployees(){
      return employeeService.getAllEmployees();
   }
   @GetMapping("/employee/{id}")
   public Employee getEmployee(@PathVariable("id") int id) {
      return employeeService.getEmployeeById(id);
   }
   @DeleteMapping("/employee/{id}")
   public void deleteEmployee(@PathVariable("id") int id) {
      employeeService.deleteEmployeeById(id);
   }
   @PostMapping("/employee")
   public void addEmployee(@RequestBody Employee employee) {
      employeeService.saveOrUpdate(employee);
   }
   @PutMapping("/employee")
```

```
    public void updateEmployee(@RequestBody Employee employee) {
    }
}
```

Run the application

In eclipse, run the **Employee Application** configuration as prepared during Application Setup

Eclipse console will show the similar output.

```
[INFO] Scanning for projects...
...
2021-07-24 20:51:14.823  INFO 9760 --- [  restartedMain] c.t.s.SprintBootH2Applicati
: Started SprintBootH2Application in 7.353 seconds (JVM running for 8.397)
```

Once server is up and running, Use Postman to make a POST request to add a record first.

Set the following parameters in POSTMAN.

- HTTP Method − **POST**
- URL − **http://localhost:8080/emp/employee**
- BODY − **An employee JSON**

```
{
   "id": "1",
   "age": "35",
   "name": "Julie",
```

```
    "email": "julie@gmail.com"
  }
```

Click on Send Button and check the response status to be OK.

Now make a Delete Request to delete that records.

Set the following parameters in POSTMAN.

- HTTP Method − **DELETE**
- URL − **http://localhost:8080/emp/employee/1**

Click the send button and verify the response status to be OK.

Now make a GET Request to get all records.

Set the following parameters in POSTMAN.

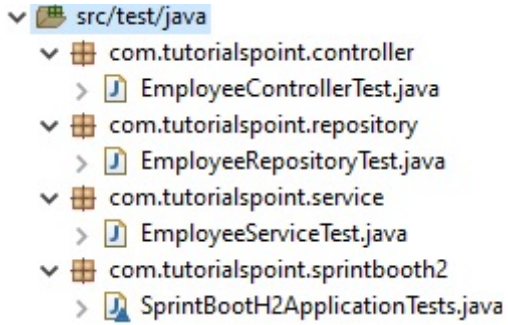
- HTTP Method − **GET**
- URL − **http://localhost:8080/emp/employees**

Click the send button and verify the response.

```
[]
```

Spring Boot & H2 - Unit Test Controller

As in previous chapter we've completed our REST APIs. Now let's create the following structure in **src/main/test** folder.



- **com.tutorialspoint.controller.EmployeeControllerTest** − A Unit Tester Class to unit test all methods of EmployeeController.
- **com.tutorialspoint.repository.EmployeeRepositoryTest** − A Unit Tester Class to unit test all methods of EmployeeRepository.
- **com.tutorialspoint.service.EmployeeServiceTest** − A Unit Tester Class to unit test all methods of EmployeeService.

SprintBootH2ApplicationTests class is already present. We need to create the above packages and relevant classes.

EmployeeControllerTest

To test a REST Controller, we need the following annotation and classes −

- **@ExtendWith(SpringExtension.class)** − Mark the class to run as test case using SpringExtension class.
- **@SpringBootTest(classes = SprintBootH2Application.class)** − Configure the Spring Boot application.
- **@AutoConfigureMockMvc** − To automatically configure the MockMVC to mock HTTP Requests and Response.

- **@Autowired private MockMvc mvc;** − MockMvc object to be used in testing.
- **@MockBean private EmployeeController employeeController** − EmployeeController mock object to be tested.

Following is the complete code of EmployeeControllerTest.

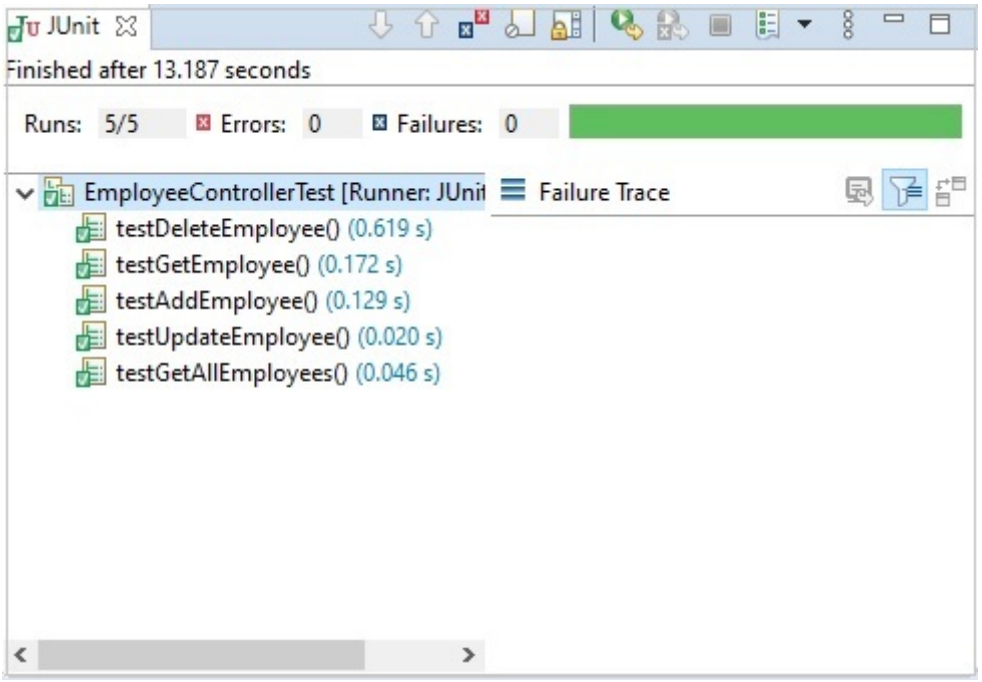
```
package com.tutorialspoint.controller;
import static org.hamcrest.core.Is.is;
import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.doNothing;
import static org.springframework.http.MediaType.APPLICATION_JSON;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilde
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers
import java.util.ArrayList;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMoc
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.sprintbooth2.SprintBootH2Application;
```

```java
@ExtendWith(SpringExtension.class)
@SpringBootTest(classes = SprintBootH2Application.class)
@AutoConfigureMockMvc
public class EmployeeControllerTest {
    @Autowired
    private MockMvc mvc;
    @MockBean
    private EmployeeController employeeController;
    @Test
    public void testGetAllEmployees() throws Exception {
        Employee employee = getEmployee();
        List<Employee> employees = new ArrayList<>();
        employees.add(employee);
        given(employeeController.getAllEmployees()).willReturn(employees);
        mvc.perform(get("/emp/employees/").contentType(APPLICATION_JSON)).andExpe
            .andExpect(jsonPath("$[0].name", is(employee.getName())));
    }
    @Test
    public void testGetEmployee() throws Exception {
        Employee employee = getEmployee();
        given(employeeController.getEmployee(1)).willReturn(employee);
        mvc.perform(get("/emp/employee/" + employee.getId()).contentType(APPLICAT
            .andExpect(jsonPath("name", is(employee.getName())));
    }
    @Test
    public void testDeleteEmployee() throws Exception {
        Employee employee = getEmployee();
        doNothing().when(employeeController).deleteEmployee(1);
        mvc.perform(delete("/emp/employee/" + employee.getId()).contentType(APPLI
            .andExpect(status().isOk()).andReturn();
    }
```

```
    @Test
    public void testAddEmployee() throws Exception {
        Employee employee = getEmployee();
        doNothing().when(employeeController).addEmployee(employee);
        mvc.perform(post("/emp/employee/").content(asJson(employee)).contentType(
            .andExpect(status().isOk()).andReturn();
    }
    @Test
    public void testUpdateEmployee() throws Exception {
        Employee employee = getEmployee();
        doNothing().when(employeeController).updateEmployee(employee);
        mvc.perform(put("/emp/employee/").content(asJson(employee)).contentType(A
            .andExpect(status().isOk()).andReturn();
    }
    private Employee getEmployee() {
        Employee employee = new Employee();
        employee.setId(1);
        employee.setName("Mahesh");
        employee.setAge(30);
        employee.setEmail("mahesh@test.com");
        return employee;
    }
    private static String asJson(final Object obj) {
        try {
            return new ObjectMapper().writeValueAsString(obj);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

Run the test cases.

Right Click on the file in eclipse and select **Run a JUnit Test** and verify the result.



Spring Boot & H2 - Unit Test Service

To test a Service, we need the following annotation and classes −

- **@ExtendWith(SpringExtension.class)** − Mark the class to run as test case using SpringExtension class.
- **@SpringBootTest(classes = SprintBootH2Application.class)** − Configure the Spring Boot application.
- **@MockBean private EmployeeService employeeService** − EmployeeService mock object to be tested.

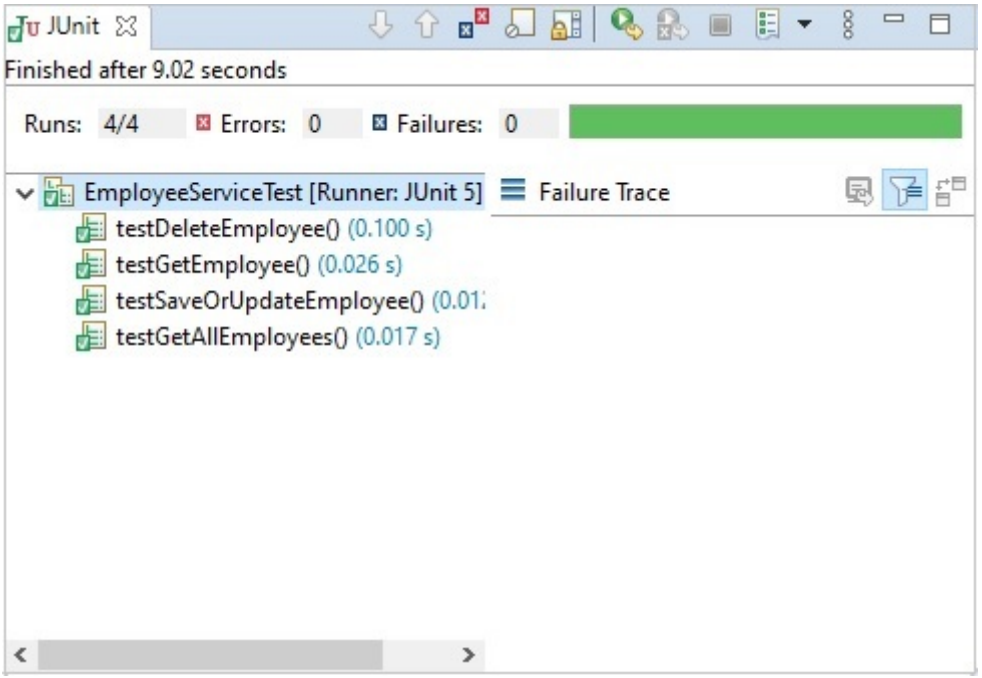
Following is the complete code of EmployeeServiceTest.

```
package com.tutorialspoint.service;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.doNothing;
import java.util.ArrayList;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.sprintbooth2.SprintBootH2Application;
@ExtendWith(SpringExtension.class)
@SpringBootTest(classes = SprintBootH2Application.class)
public class EmployeeServiceTest {
   @MockBean
   private EmployeeService employeeService;
   @Test
   public void testGetAllEmployees() throws Exception {
      Employee employee = getEmployee();
      List<Employee> employees = new ArrayList<>();
      employees.add(employee);
      given(employeeService.getAllEmployees()).willReturn(employees);
      List<Employee> result = employeeService.getAllEmployees();
      assertEquals(result.size(), 1);
   }
   @Test
```

```
    public void testGetEmployee() throws Exception {
        Employee employee = getEmployee();
        given(employeeService.getEmployeeById(1)).willReturn(employee);
        Employee result = employeeService.getEmployeeById(1);
        assertEquals(result.getId(), 1);
    }
    @Test
    public void testDeleteEmployee() throws Exception {
        doNothing().when(employeeService).deleteEmployeeById(1);
        employeeService.deleteEmployeeById(1);
        assertTrue(true);
    }
    @Test
    public void testSaveOrUpdateEmployee() throws Exception {
        Employee employee = getEmployee();
        doNothing().when(employeeService).saveOrUpdate(employee);
        employeeService.saveOrUpdate(employee);
        assertTrue(true);
    }
    private Employee getEmployee() {
        Employee employee = new Employee();
        employee.setId(1);
        employee.setName("Mahesh");
        employee.setAge(30);
        employee.setEmail("mahesh@test.com");
        return employee;
    }
}
```

Run the test cases.

Right Click on the file in eclipse and select **Run a JUnit Test** and verify the result.



Spring Boot & H2 - Unit Test Repository

To test a Repository, we need the following annotation and classes −

- **@ExtendWith(SpringExtension.class)** − Mark the class to run as test case using SpringExtension class.
- **@SpringBootTest(classes = SprintBootH2Application.class)** − Configure the Spring Boot application.
- **@Transactional** − To mark repository to do CRUD Operation capable.
- **@Autowired private EmployeeRepository employeeRepository** −

EmployeeRepository object to be tested.

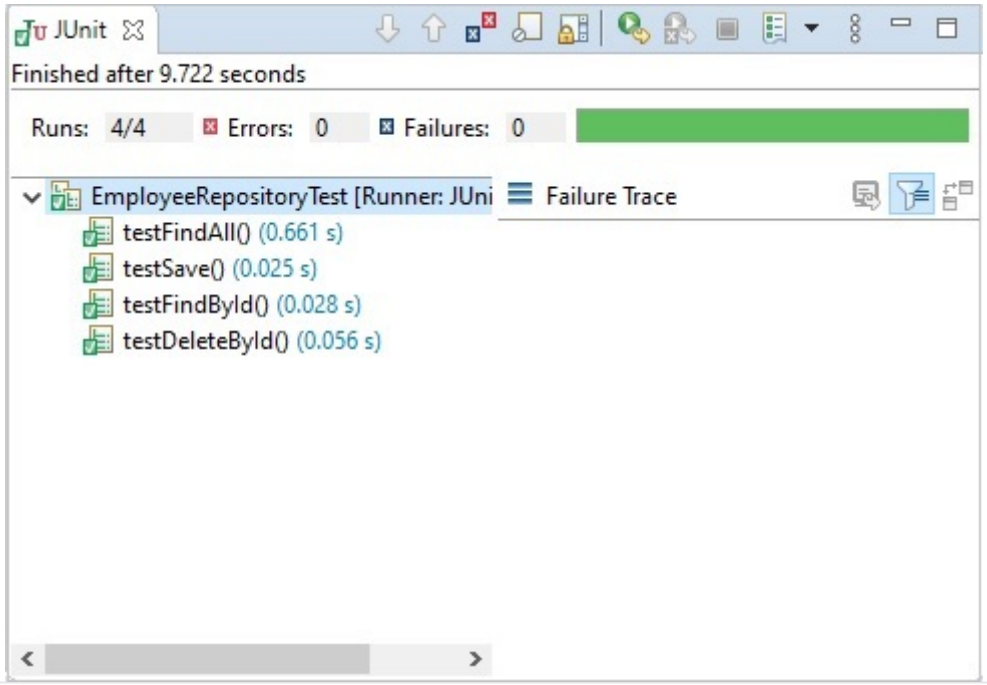
Following is the complete code of EmployeeRepositoryTest.

```
package com.tutorialspoint.repository;
import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.ArrayList;
import java.util.List;
import javax.transaction.Transactional;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.sprintbooth2.SprintBootH2Application;
@ExtendWith(SpringExtension.class)
@Transactional
@SpringBootTest(classes = SprintBootH2Application.class)
public class EmployeeRepositoryTest {
   @Autowired
   private EmployeeRepository employeeRepository;
   @Test
   public void testFindById() {
      Employee employee = getEmployee();
      employeeRepository.save(employee);
      Employee result = employeeRepository.findById(employee.getId()).get();
      assertEquals(employee.getId(), result.getId());
   }
   @Test
   public void testFindAll() {
```

```
        Employee employee = getEmployee();
        employeeRepository.save(employee);
        List<Employee> result = new ArrayList<>();
        employeeRepository.findAll().forEach(e -> result.add(e));
        assertEquals(result.size(), 1);
    }
    @Test
    public void testSave() {
        Employee employee = getEmployee();
        employeeRepository.save(employee);
        Employee found = employeeRepository.findById(employee.getId()).get();
        assertEquals(employee.getId(), found.getId());
    }
    @Test
    public void testDeleteById() {
        Employee employee = getEmployee();
        employeeRepository.save(employee);
        employeeRepository.deleteById(employee.getId());
        List<Employee> result = new ArrayList<>();
        employeeRepository.findAll().forEach(e -> result.add(e));
        assertEquals(result.size(), 0);
    }
    private Employee getEmployee() {
        Employee employee = new Employee();
        employee.setId(1);
        employee.setName("Mahesh");
        employee.setAge(30);
        employee.setEmail("mahesh@test.com");
        return employee;
    }
}
```

Run the test cases.

Right Click on the file in eclipse and select **Run a JUnit Test** and verify the result.



Print Page