|  |  |
| --- | --- |
| **Sr No** | **Topic** |
| **1** | Spring Boot Project Creation 4 Types |
| **2** | Spring Boot Starters |
| **3** | Spring Boot Data JPA |
| **4** | Spring Boot REST API |
| **5** | Spring Boot Exception Handling |
| **6** | Spring Boot WAR Packaging |
| **7** | Spring Boot MVC With JSP |
| **8** | Spring Boot MVC With Freemarker |
| **9** | Spring Boot MVC With Themyleaf |
| **10** | Spring Boot Hibernate integration |
| **11** | Spring Boot AngularJS integration |
| **12** | Spring Boot MyBatis Annotation integration |
| **13** | Spring Boot MyBatis XML integration |
| **14** | Spring Boot JDBC Single DataSource |
| **15** | Spring Boot JDBC Multiple DataSource |
| **16** | Spring Boot XML Request and Response |
| **17** | Spring Boot Gmail Integration |
| **18** | Spring Boot Security Integration |
| **19** | Spring Boot YAML |
| **20** | Spring Boot Actuator |
| **21** | Spring Boot Jetty and Undertow server integration |
| **22** | Spring Boot Gradle Integration |
| **23** | Spring Boot Internationalization |
| **24** | Spring Boot + Swagger 2 |
| **25** | Spring Boot + Swagger2- Understanding various Swagger annotations |
| **26** | Spring Boot Admin |
| **27** | Spring Boot MongoDb REST Integration |
| **28** | Spring Boot Logging |
| **29** | Spring Boot Angular 7 Integration |
| **30** | Spring Boot + ActiveMQ Integration |
| **31** | Spring Boot+ Apache Camel Integration |
| **32** | Spring Boot Batch Integration |
| **33** | Sprint Boot File Upload |
| **34** | Spring Boot Apache Kafka Integration |
| **35** | Spring Boot RabbitMQ – Producer and Consumer |
| **36** | Spring Boot Devtools |
| **37** | Spring Boot Profiles |
| **38** | Spring Boot CommandLineRunner Interface |
| **39** | Spring Boot Testing |
| **40** | Spring Boot Docker |
| **41** | Enable Http/Https in Spring Boot |
| **42** | Spring Boot as a windows Service |
| **43** | Spring Boot HikariCP |
| **44** | Spring boot JPA call MySQL procedure |
| **45** | Spring boot cloud eureka server |
| **46** | Spring boot cloud eureka client |
| **47** | Spring boot custom error pages |
| **48** | Spring boot basic authentication database |
| **49** | Spring boot common dbcp2 connection pool |
| **50** | Spring Boot Custom Banners |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |



It is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework. It is used to create standalone spring based application that you can just run because it needs very little spring configuration.

Spring Boot does not generate code and there is absolutely no requirement for XML configuration. It uses **convention over configuration** software design paradigm that means it decrease the effort of developer.

* Create stand-alone Spring applications that can be started using java -jar.
* Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
* It provides opinionated 'starter' POMs to simplify your Maven configuration.
* It automatically configure Spring whenever possible.
* It provides production-ready features such as metrics, health checks and externalized configuration.
* Absolutely no code generation and no requirement for XML configuration.
* It is very easy to develop Spring Based applications with Java or Groovy.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
* It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle.
* It provides lots of plugins to work with embedded and in-memory Databases very easily.

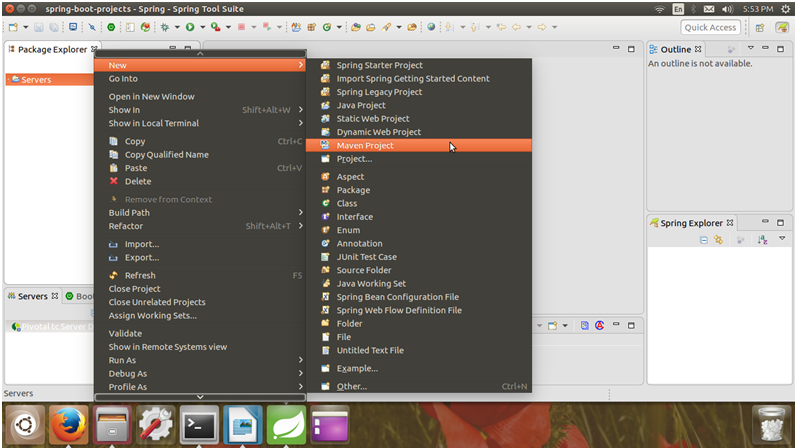
What Is Spring Boot, Spring Boot Tutorial

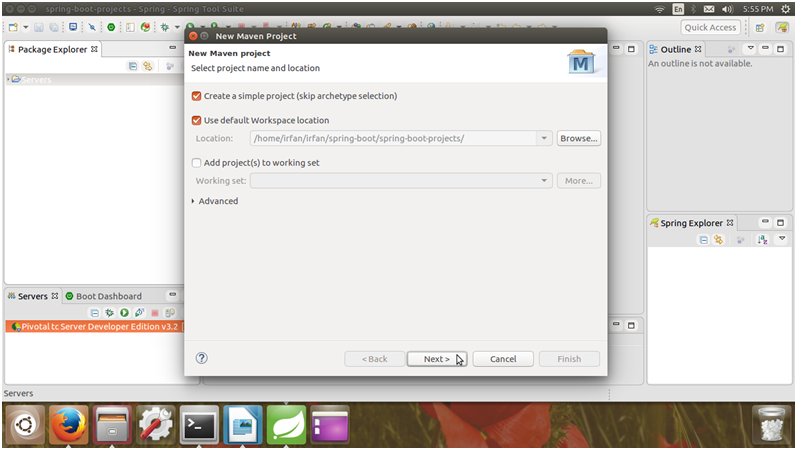
There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.

* Spring Maven Project
* Spring Starter Project Wizard
* Spring Initializer
* Spring Boot CLI
* **Spring Boot Maven Project**

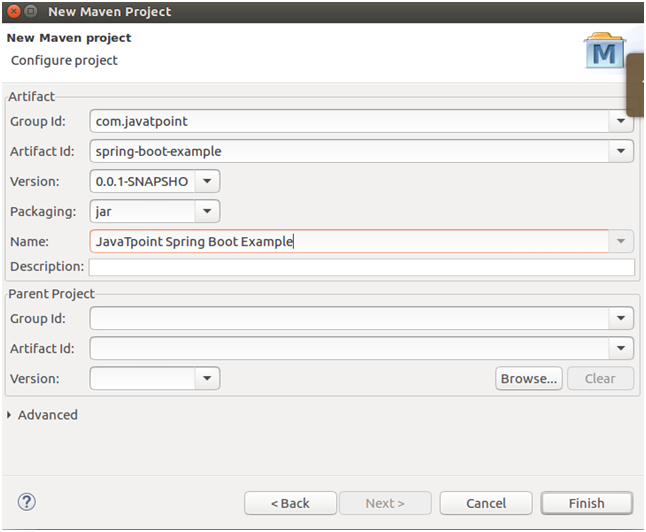
Creating Spring Boot project by creating maven project. It includes the following steps.

1. Select project type.

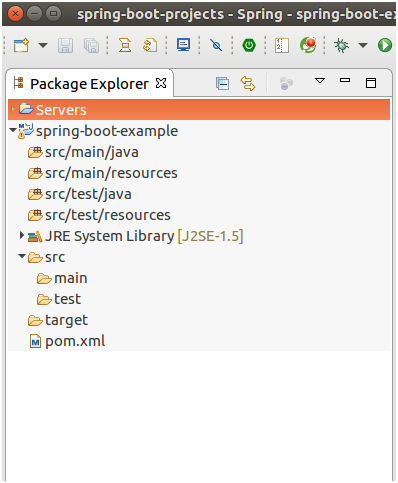




Configure project by providing project name.



After clicking finish, Spring boot project has been created. Our new project looks like the following screen shot.



This Maven project has a pom.xml file which contains the following default configuration.

// 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>**com.javatpoint**</groupId>**

**<artifactId>**spring-boot-example**</artifactId>**

**<version>**0.0.1-SNAPSHOT**</version>**

**<name>**JavaTpoint Spring Boot Example**</name>**

**</project>**

We need to configure it in order to make it a Spring Boot project. Here, we are adding parent to our Maven project. It is used to declare that our project is a child to this parent project.

**<parent>**

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

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

**<version>**1.4.2.RELEASE**</version>**

**</parent>**

After that add the following dependency to the pom.xml file. Here, we are adding web dependency by adding spring-boot-starter-web.

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**</dependencies>**

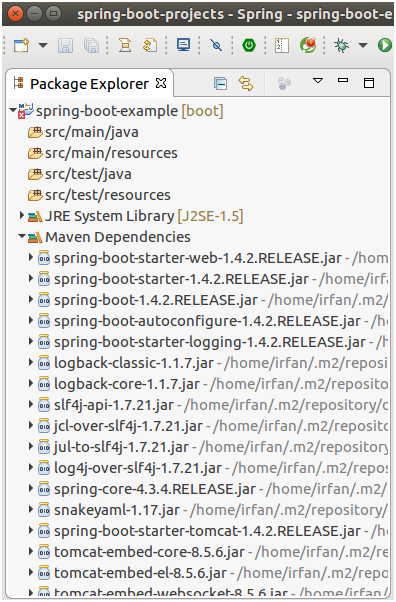
After that add Java version for the project.

**<properties>**

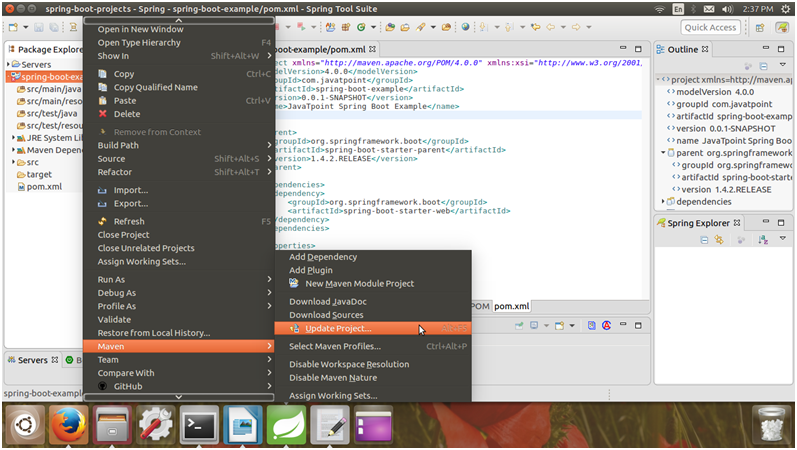
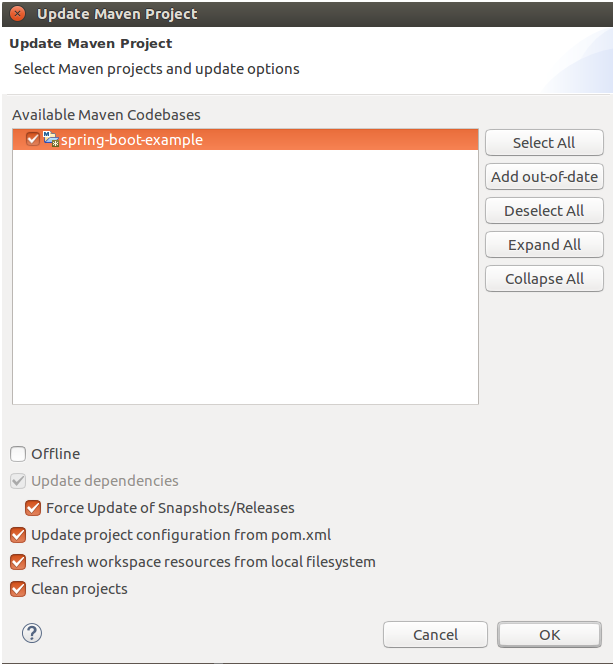
**<java.version>**1.8**</java.version>**

**</properties>**

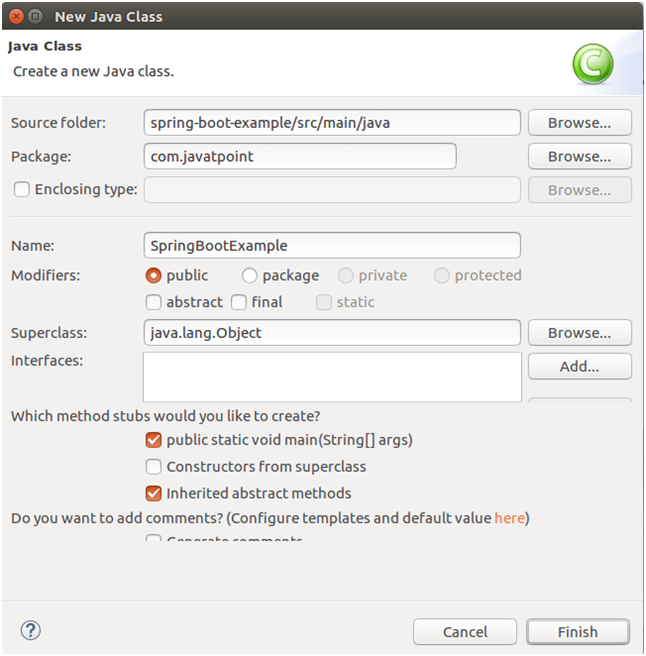
Now, our project should have the following directly structure. You can notice that maven has created a new dependency folder to store the jar files.



Update Maven project each time after including new dependencies.

After updating Maven. Now, let's make it runnable. Create a Java class inside the src/main/java package.



After creating class call the static run method of SpringApplication class. In the following code, we are calling run method and passing class name as argument.

// SpringBootExample.java

**package** com.javatpoint;

**import** org.springframework.boot.SpringApplication;

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

 @SpringBootApplication

publicclass SpringBootExample {

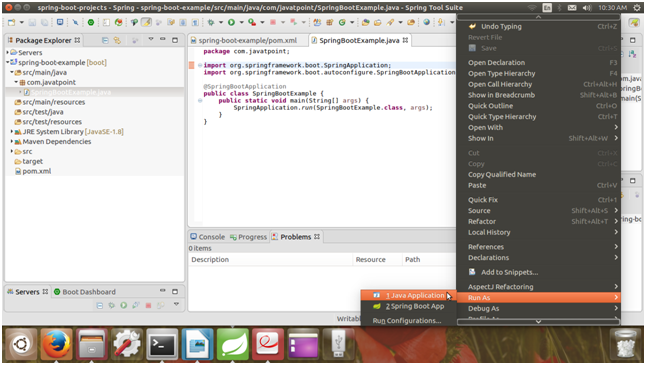
    publicstaticvoid main(String[] args) {

        SpringApplication.run(SpringBootExample.**class**, args);

  }

}

Now, run this class as a Java Application.



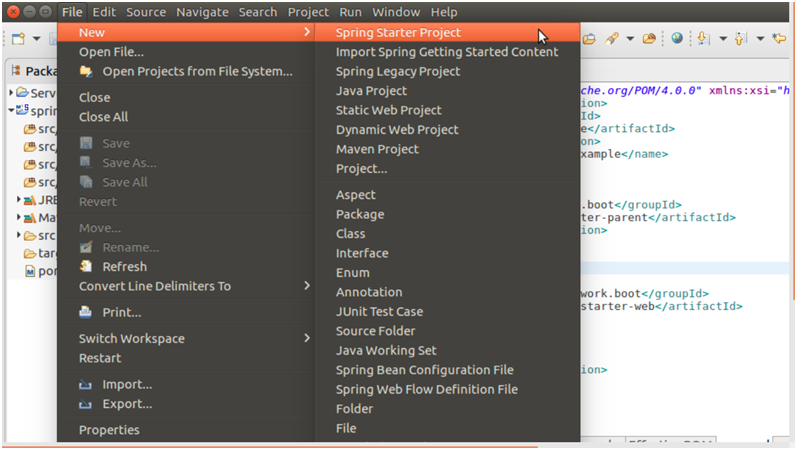
It displays the following output.



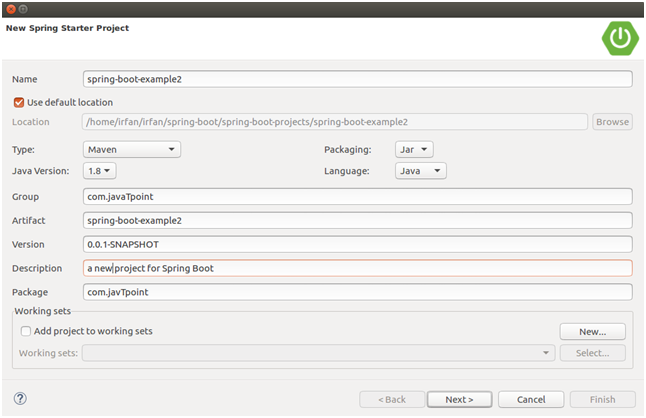
The line **Started SpringBootExample in 14.047 seconds (JVM running for 17.391)** in output window shows that our application is started.

* **Spring Starter Project Wizard**

Select Starter Project

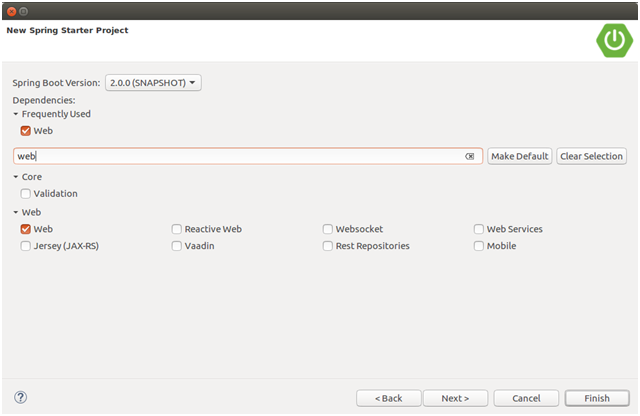


Provide project details.



Select dependency as web to create web project.

After finishing, the default pom.xml file for the project is created as below:



// pom.xml

**<?**xmlversion**xmlversion**="1.0"encoding="UTF-8"**?>**

**<**projectxmlns**projectxmlns**="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.javatpoint**</groupId>**

**<artifactId>**new-spring-boot-example2**</artifactId>**

**<version>**0.0.1-SNAPSHOT**</version>**

**<packaging>**jar**</packaging>**

**<name>**spring-boot-example2**</name>**

**<description>**a new project for Spring Boot**</description>**

**<parent>**

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

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

**<version>**2.0.0.BUILD-SNAPSHOT**</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-test**</artifactId>**

**<scope>**test**</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**<repositories>**

**<repository>**

**<id>**spring-snapshots**</id>**

**<name>**Spring Snapshots**</name>**

**<url>**https://repo.spring.io/snapshot**</url>**

**<snapshots>**

**<enabled>**true**</enabled>**

**</snapshots>**

**</repository>**

**<repository>**

**<id>**spring-milestones**</id>**

**<name>**Spring Milestones**</name>**

**<url>**https://repo.spring.io/milestone**</url>**

**<snapshots>**

**<enabled>**false**</enabled>**

**</snapshots>**

**</repository>**

**</repositories>**

**<pluginRepositories>**

**<pluginRepository>**

**<id>**spring-snapshots**</id>**

**<name>**Spring Snapshots**</name>**

**<url>**https://repo.spring.io/snapshot**</url>**

**<snapshots>**

**<enabled>**true**</enabled>**

**</snapshots>**

**</pluginRepository>**

**<pluginRepository>**

**<id>**spring-milestones**</id>**

**<name>**Spring Milestones**</name>**

**<url>**https://repo.spring.io/milestone**</url>**

**<snapshots>**

**<enabled>**false**</enabled>**

**</snapshots>**

**</pluginRepository>**

**</pluginRepositories>**

**</project>**

This project auto generates a Java file as given below inside the **src/main/java.**

**package** com.javtpoint;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootExample2Application {

**public** **static** **void** main(String[] args) {

        SpringApplication.run(SpringBootExample2Application.**class**, args);

    }

}

Now we can run this as a Java Application and it will produce the following output.

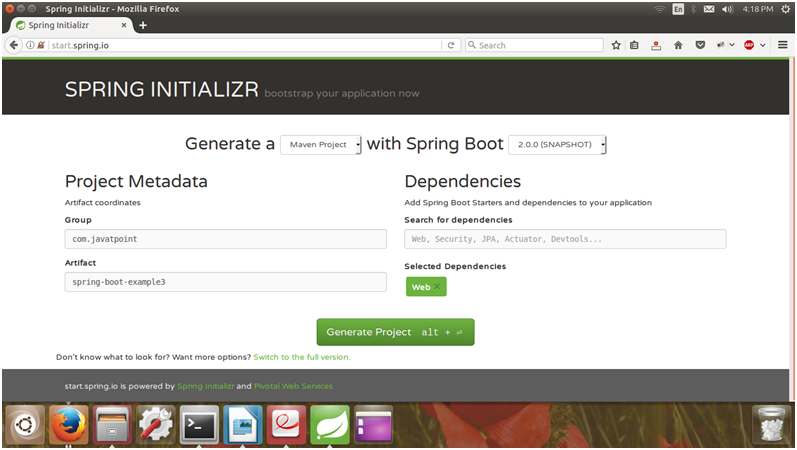


Observe the line **Started SpringBootExample2Application in 23.788 seconds (JVM running for 30.19)** in the output section. It shows that our application is started.

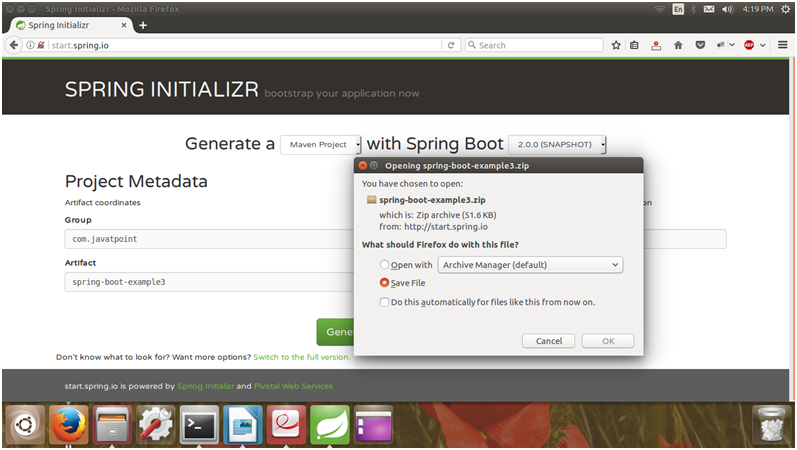
* **Spring Initializr**

It is a web tool which is provided by Spring on official site. You can create Spring Boot project by providing project details.

Select Maven project and dependencies. Fill other details and click on generate project.



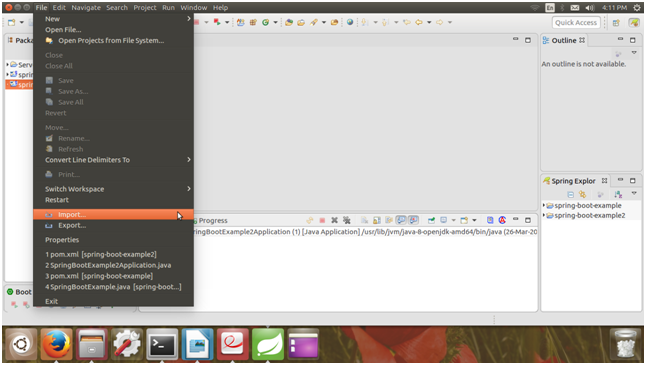
After clicking it asked for download project.



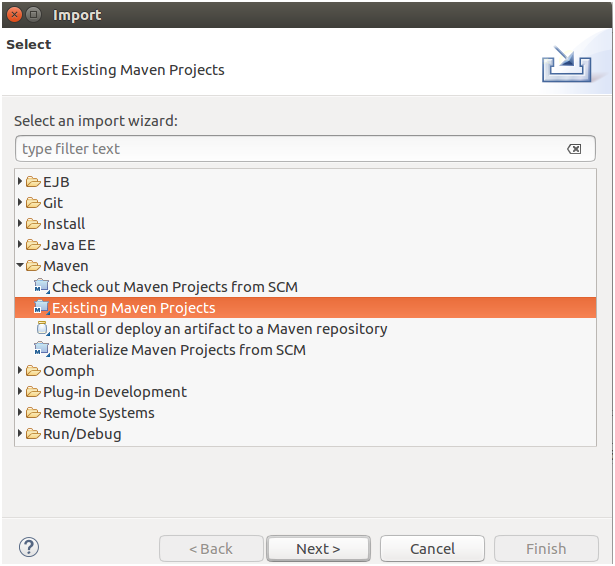
Save project and extract it.

Now import this project by using import option from the STS (Spring Tool Suite) IDE.

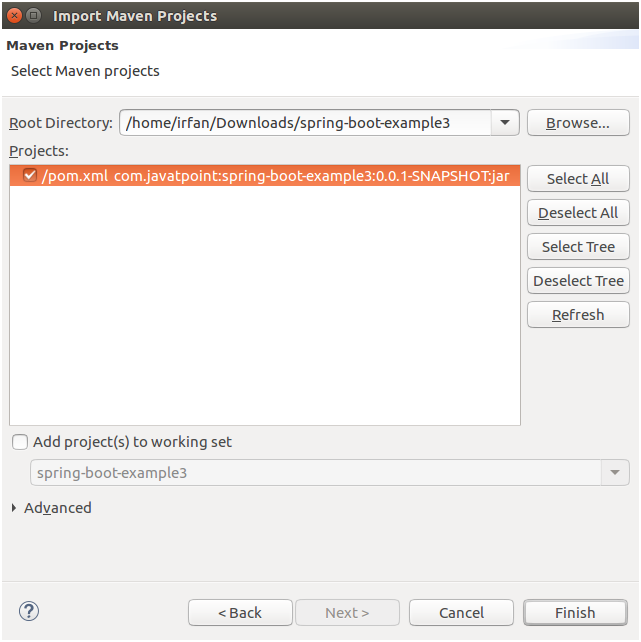
Importing project



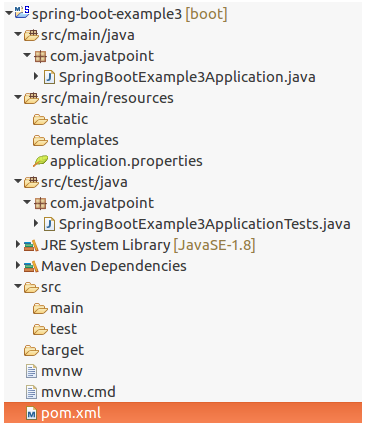
Select project type in Maven.



Select source project



After finishing, our Spring Boot Project should look like this.



It has a pom.xml file which contains all the dependencies and configuration for the project. It should look like this.

// 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.javatpoint</groupId>

<artifactId>spring-boot-example3</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-example3</name>

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

<parent>

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

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

<version>2.1.0.BUILD-SNAPSHOT</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-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

It also generates a Java file in the **src/main/java** directory. This file has same name as you mentioned in artifact. The default created Java file is the below.

// SpringBootExample3Application.java

**package** com.javatpoint;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootExample3Application {

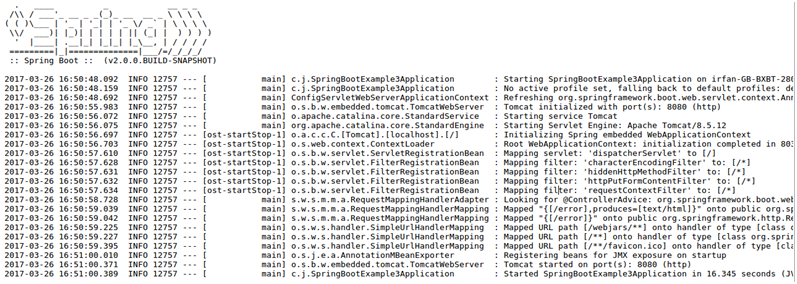
**public** **static** **void** main(String[] args) {

        SpringApplication.run(SpringBootExample3Application.**class**, args);

    }

}

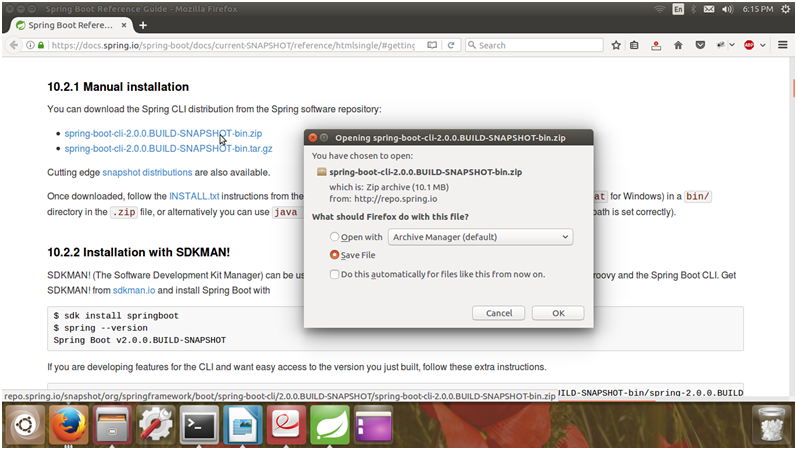
We can run this project by selecting Java Application from run options . After running, it produces the following output.



Observe the line **Started SpringBootExample3Application in 16.345 seconds (JVM running for 19.909)**. It shows that the Spring Boot application has started.

* **Spring Boot CLI**

It is a tool which you can download from the official site of Spring Framework. Here, we are explaining steps. Download the CLI tool from official site as we are doing here.



Create a controller in the groovy file with file name: SpringBootCliExample.groovy

@RestController

class SpringBootCliExample {

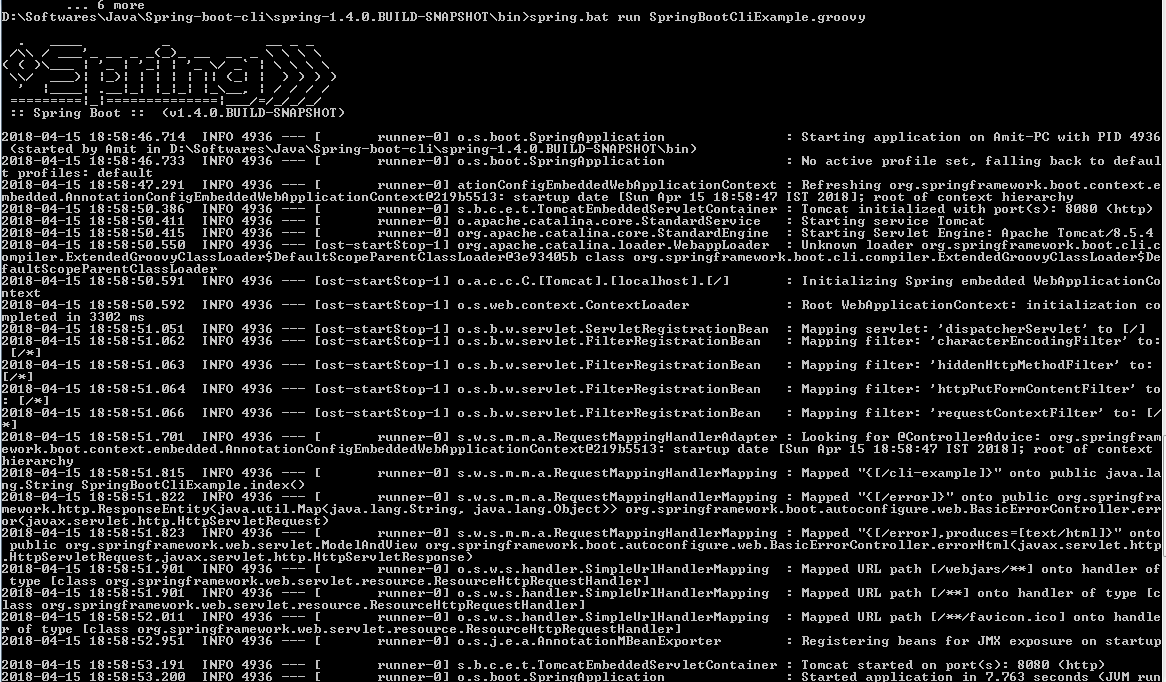
@RequestMapping("/cli-example")

String index() {

"Spring Boot Example using Cli is Running Successfully.";

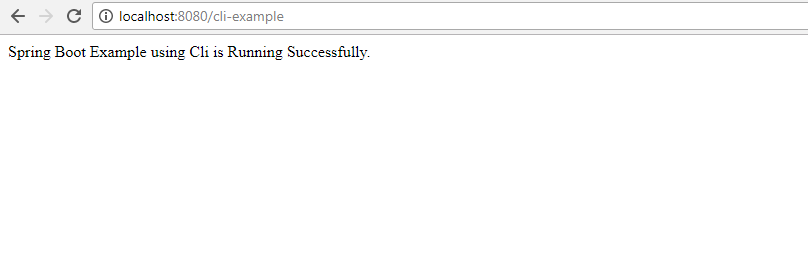
}

}



This project is running on the port 8080. So, we can invoke it on any browser by using the following url.

<http://localhost:8080/cli-example>



* **Spring Boot Starters**

Starters are a set of convenient dependency descriptors which we can include in our application.Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the **spring-boot-starter-data-jpa** dependency in your project.

Starter should follow a naming pattern like: **spring-boot-starter**-\*, where \* is a particular type of application. This naming structure is intended to help when you need to find a starter.

The following application starters are provided by Spring Boot under the org.springframework.boot group:

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-thymeleaf | It is used to build MVC web applications using Thymeleaf views. |
| spring-boot-starter-data-couchbase | This is used for Couchbase document-oriented database and Spring Data Couchbase. |
| spring-boot-starter-artemis | It is used for JMS messaging using Apache Artemis. |
| spring-boot-starter-web-services | It is used for Spring Web Services. |
| spring-boot-starter-mail | It is used to support Java Mail and Spring Framework's email sending. |
| spring-boot-starter-data-redis | It is used for Redis key-value data store with Spring Data Redis and the Jedis client. |
| spring-boot-starter-web | It is used for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container. |
| spring-boot-starter-data-gemfire | It is used to GemFire distributed data store and Spring Data GemFire. |
| spring-boot-starter-activemq | It is used to JMS messaging using Apache ActiveMQ. |
| spring-boot-starter-data-elasticsearch | It is used to Elasticsearch search and analytics engine and Spring Data Elasticsearch. |
| spring-boot-starter-integration | It is used for Spring Integration. |
| spring-boot-starter-test | It is used to test Spring Boot applications with libraries including JUnit, Hamcrest and Mockito. |
| spring-boot-starter-jdbc | It is used for JDBC with the Tomcat JDBC connection pool. |
| spring-boot-starter-mobile | It is used for building web applications using Spring Mobile. |
| spring-boot-starter-validation | It is used for Java Bean Validation with Hibernate Validator. |
| spring-boot-starter-hateoas | It is used to build hypermedia-based RESTful web application with Spring MVC and Spring HATEOAS. |
| spring-boot-starter-jersey | It is used to build RESTful web applications using JAX-RS and Jersey. An alternative to spring-boot-starter-web. |
| spring-boot-starter-data-neo4j | It is used for Neo4j graph database and Spring Data Neo4j. |
| spring-boot-starter-data-ldap | It is used for Spring Data LDAP. |
| spring-boot-starter-websocket | It is used for building WebSocket. applications using Spring Framework?s WebSocket support. |
| spring-boot-starter-aop | It is used for aspect-oriented programming with Spring AOP and AspectJ. |
| spring-boot-starter-amqp | It is used for Spring AMQP and Rabbit MQ. |
| spring-boot-starter-data-cassandra | It is used for Cassandra distributed database and Spring Data Cassandra. |
| spring-boot-starter-social-facebook | It is used for Spring Social Facebook. |
| spring-boot-starter-jta-atomikos | It is used for JTA transactions using Atomikos. |
| spring-boot-starter-security | It is used for Spring Security. |
| spring-boot-starter-mustache | It is used for building MVC web applications using Mustache views. |
| spring-boot-starter-data-jpa | It is used for Spring Data JPA with Hibernate. |
| spring-boot-starter | It is used for core starter, including auto-configuration support, logging and YAML. |
| spring-boot-starter-groovy-templates | It is used for building MVC web applications using Groovy Templates views. |
| spring-boot-starter-freemarker | It is used for building MVC web applications using FreeMarker views. |
| spring-boot-starter-batch | It is used for Spring Batch. |
| spring-boot-starter-social-linkedin | It is used for Spring Social LinkedIn. |
| spring-boot-starter-cache | It is used for Spring Framework?s caching support. |
| spring-boot-starter-data-solr | It is used for the Apache Solr search platform with Spring Data Solr. |
| spring-boot-starter-data-mongodb | It is used for MongoDB document-oriented database and Spring Data MongoDB. |
| spring-boot-starter-jooq | It is used for jOOQ to access SQL databases. An alternative to spring-boot-starter-data-jpa or spring-boot-starter-jdbc. |
| spring-boot-starter-jta-narayana | It is used for Spring Boot Narayana JTA Starter. |
| spring-boot-starter-cloud-connectors | It is used for Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku. |
| spring-boot-starter-jta-bitronix | It is used for JTA transactions using Bitronix. |
| spring-boot-starter-social-twitter | It is used for Spring Social Twitter. |
| spring-boot-starter-data-rest | It is used for exposing Spring Data repositories over REST using Spring Data REST. |

Spring Boot production starters

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-actuator | It is used for Spring Boot?s Actuator which provides production ready features to help you monitor and manage your application. |
| spring-boot-starter-remote-shell | It is used for the CRaSH remote shell to monitor and manage your application over SSH. Deprecated since 1.5. |

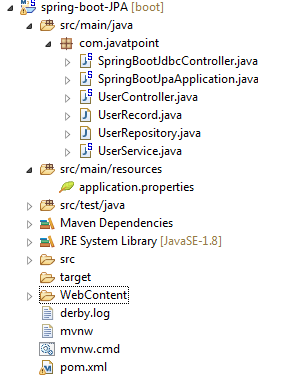
Spring Boot technical starters

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-undertow | It is used for Undertow as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-jetty | It is used for Jetty as the embedded servlet container. An alternative to spring-boot-starter-tomcat. |
| spring-boot-starter-logging | It is used for logging using Logback. Default logging starter. |
| spring-boot-starter-tomcat | It is used for Tomcat as the embedded servlet container. Default servlet container starter used by spring-boot-starter-web. |
| spring-boot-starter-log4j2 | It is used for Log4j2 for logging. An alternative to spring-boot-starter-logging. |

* Spring Boot JPA

Spring Boot provides **spring-boot-starter-data-jpa** starter to connect Spring application with relational database efficiently. You can use it into project POM (Project Object Model) file.

* **Project Structure:**

****

* **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.javatpoint</groupId>

<artifactId>spring-boot-JPA</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-JPA</name>

<description>A new project of Spring Boot JPA</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath />

</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>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.9</version>

</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**

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

spring.datasource.username=root

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create-drop

server.port=8085

* **UserRecord.java:**

**package** com.javatpoint;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

@Entity

**public** **class** UserRecord {

@Id

**private** String id;

**private** String name;

**private** String email;

**public** UserRecord() {

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

} }

* **UserRepository.java:**

**package** com.javatpoint;

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

**public** **interface** UserRepository **extends** JpaRepository<UserRecord, String> {

}

* **UserService.java:**

**package** com.javatpoint;

**import** java.util.List;

**import** javax.annotation.Resource;

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

**import** org.springframework.stereotype.Service;

@Service

**public** **class** UserService {

@Autowired

@Resource

**private** UserRepository userRepository;

**public** List<UserRecord> getAllUsers(){

**return** userRepository.findAll();

}

**public** UserRecord getUser(String id){

**return** userRepository.findOne(id);

}

**public** **void** addUser(UserRecord userRecord){

userRepository.save(userRecord);

}

**public** **void** deleteUser(String id){

userRepository.delete(id);

}

**public** UserRecord updateUser(UserRecord user) {

UserRecord updatedUser = userRepository.findOne(user.getId());

updatedUser.setId(user.getId());

updatedUser.setName(user.getName());

updatedUser.setEmail(user.getEmail());

System.***out***.println("Id="+updatedUser.getId()+"\tName="+updatedUser.getName()+"\tEmail="+updatedUser.getEmail());

**return** updatedUser;

}

}

* **UserController.java**:

**package** com.javatpoint;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** UserController {

@Autowired

**private** UserService userService;

@RequestMapping("/")

**public** List<UserRecord> getAllUser(){

**return** userService.getAllUsers();

}

@RequestMapping(value="/add-user", method=RequestMethod.***POST***)

**public** **void** addUser(@RequestBody UserRecord userRecord){

userService.addUser(userRecord);

}

@RequestMapping(value="/user/{id}", method=RequestMethod.***GET***)

**public** UserRecord getUser(@PathVariable String id){

**return** userService.getUser(id);

}

@RequestMapping(value="/user/delete/{id}", method=RequestMethod.***DELETE***)

**public** **void** deleteUser(@PathVariable String id){

userService.deleteUser(id);

}

@RequestMapping(value="/update-user/{id}", method=RequestMethod.***PUT***)

**public** **void** updateUser(@RequestBody UserRecord userRecord){

userService.updateUser(userRecord);

}

}

* **SpringBootJpaApplication.java:**

**package** com.javatpoint;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootJpaApplication {

**public** **static** **void** main (String [] args) {

SpringApplication.*run* (SpringBootJpaApplication. **Class**, args);

}

}

Run SpringBootJpaApplication as Java Application and using Postman perform CRUD operations

* **Spring Boot WAR Packaging :**

In Spring boot applications, default packaging is jar which is deployed in embedded servers. If you want to generate a war file for deployment in seperate application server instances such as Jboss, Weblogic or tomcat, then follow below instructions.

**Step 1) Declare packaging type ‘war’**

First logical step is to declare the packaging type ‘war’ in pom.xml file.

|  |
| --- |
| <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;>    ...    <packaging>war</packaging>    ...  </project> |

It indicates the project’s artifact type. Please note that when no packaging is declared, Maven assumes the artifact is the default: jar.

**Step 2) Set embedded server dependency scope to ‘provided’**

We may want to have embedded server (e.g. tomcat) in development environment because of its usefulness in fast development lifecycle, but we certainly not want those server jars to be included in finally generated maven artifact or war file. To do so, set scope of embedded server dependency to ‘provided’.

Scope ‘provided’ indicates you expect the JDK or a container to provide the dependency at runtime. This scope is only available on the compilation and test classpath, and is not transitive.

<dependency>

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

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

<scope>provided</scope>

</dependency>

**Step 3) Spring Boot Servlet Initializer**

The traditional way of deployment is making the Spring Boot Application **@SpringBootApplication** class extend the **SpringBootServletInitializer**class. Spring Boot Servlet Initializer class file allows you to configure the application when it is launched by using Servlet Container.

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.boot.builder.SpringApplicationBuilder;

**import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

**public** **class** SpringBootWarPackagingApplication **extends** SpringBootServletInitializer {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootWarPackagingApplication.**class**, args);

}

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

**return** application.sources(SpringBootWarPackagingApplication.**class**);

}

@RequestMapping(value = "/")

**public** String hello() {

**return** "Hello World from Tomcat";

}

}

**Step 4) Setting Main Class**

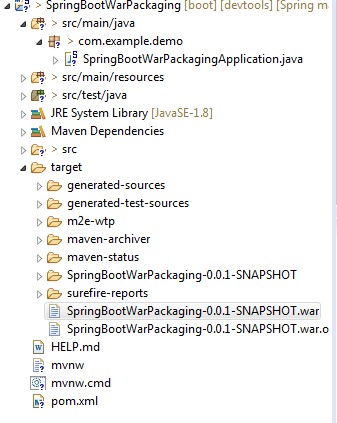
In Spring Boot, we need to mention the main class that should start in the build file. For this purpose, you can use the following pieces of code −

For Maven, add the start class in **pom.xml** properties as shown below –

<start-class>com.tutorialspoint.demo.DemoApplication</start-class>

**Step 5) Packaging your Application**

mvn clean install



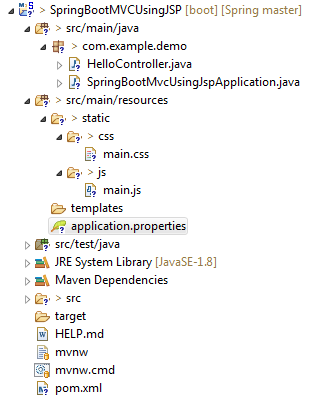
Copy and paste the war file inside tomcat\webapps

Start the tomcat

<http://localhost:8080/SpringBootWarPackaging/>



* **Spring Boot MVC With JSP :**



**pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</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>

</dependencies>

**HelloWorldController.java:**

**package** com.example.demo;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

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

@Controller

**public** **class** HelloController {

@GetMapping({"/", "/hello"})

**public** String hello(Model model, @RequestParam(value="name", required=**false**, defaultValue="World") String name) {

model.addAttribute("name", name);

**return** "hello";

}

}

**Hello.jsp:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<title>Hello ${name}!</title>

<link href=*"/css/main.css"* rel=*"stylesheet"*>

</head>

<body>

<h2 class=*"hello-title"*>Hello ${name}!</h2>

<script src=*"/js/main.js"*></script>

</body>

</html>

**main.css:**

*.hello-title*{

color: *darkgreen*;

}

**main.js:**

(**function**(){

console.log("Hello World!");

})();

**application.properties:**

spring.mvc.view.prefix: /WEB-INF/jsp/

spring.mvc.view.suffix: .jsp

**SpringBootMvcUsingJspApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.boot.builder.SpringApplicationBuilder;

**import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

**public** **class** SpringBootMvcUsingJspApplication **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

**return** application.sources(SpringBootMvcUsingJspApplication.**class**);

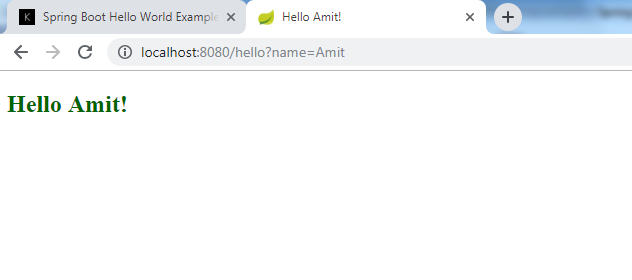
}

**public** **static** **void** main(String[] args) {

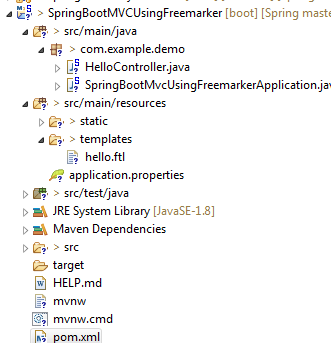
SpringApplication.*run*(SpringBootMvcUsingJspApplication.**class**, args);

}

}



* **Spring Boot MVC With Freemarker :**



**pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</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>

</dependencies>

**HelloWorldController.java:**

**package** com.example.demo;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

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

@Controller

**public** **class** HelloController {

@GetMapping({"/", "/hello"})

**public** String hello(Model model, @RequestParam(value="name", required=**false**, defaultValue="World") String name) {

model.addAttribute("name", name);

**return** "hello";

}

}

**Hello.ftl:**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<title>Hello ${name}!</title>

<link href=*"/css/main.css"* rel=*"stylesheet"*>

</head>

<body>

<h2 class=*"hello-title"*>Hello ${name}!</h2>

<script src=*"/js/main.js"*></script>

</body>

</html>

**main.css:**

*.hello-title*{

color: *darkgreen*;

}

**main.js:**

(**function**(){

console.log("Hello World!");

})();

**application.properties:**

spring.freemarker.template-loader-path: classpath:/templates

spring.freemarker.suffix: .ftl

**SpringBootMvcUsingFreemarkerApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

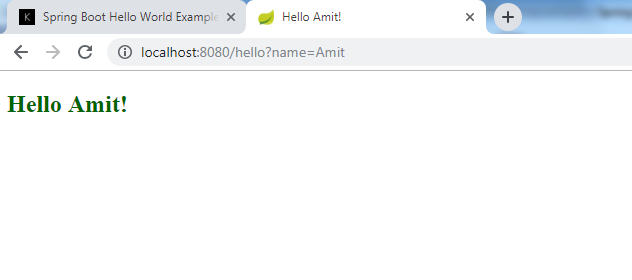
**public** **class** SpringBootMvcUsingFreemarkerApplication{

**public** **static** **void** main(String[] args) {

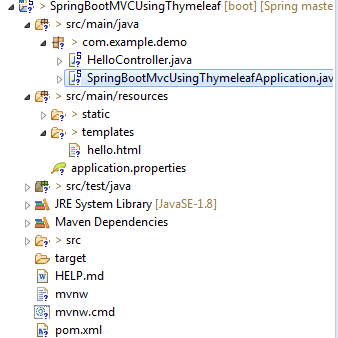
SpringApplication.*run*(SpringBootMvcUsingFreemarkerApplication.**class**, args);

}

}



* **Spring Boot MVC With Thymeleaf :**



**pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</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>

</dependencies>

**HelloWorldController.java:**

**package** com.example.demo;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

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

@Controller

**public** **class** HelloController {

@GetMapping({"/", "/hello"})

**public** String hello(Model model, @RequestParam(value="name", required=**false**, defaultValue="World") String name) {

model.addAttribute("name", name);

**return** "hello";

}

}

**Hello.html:**

<!DOCTYPE html>

<html lang=*"en"* xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<meta charset=*"UTF-8"*>

<title th:text=*"'Hello, ' + ${name} + '!'"*></title>

<link href=*"/css/main.css"* rel=*"stylesheet"*>

</head>

<body>

<h2 class=*"hello-title"* th:text=*"'Hello, ' + ${name} + '!'"*></h2>

<script src=*"/js/main.js"*></script>

</body>

</html>

**main.css:**

*.hello-title*{

color: *darkgreen*;

}

**main.js:**

(**function**(){

console.log("Hello World!");

})();

**application.properties:**

pring.thymeleaf.template-loader-path: classpath:/templates

spring.thymeleaf.suffix: .html

spring.thymeleaf.cache: false

**SpringBootMvcUsingThymeleafApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

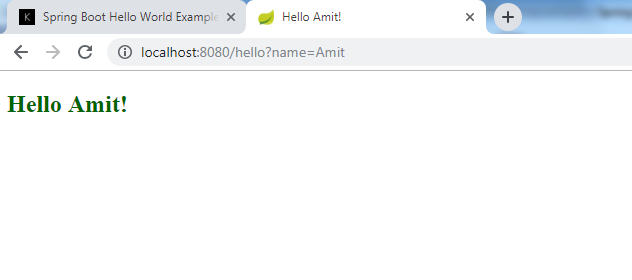
**public** **class** SpringBootMvcUsingThymeleafApplication{

**public** **static** **void** main(String[] args) {

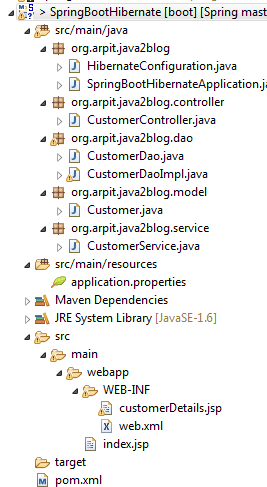
SpringApplication.*run*(SpringBootMvcUsingThymeleafApplication.**class**, args);

}

}



* **Spring Boot Hibernate Integration:**



* **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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.arpit.java2blog</groupId>

<artifactId>SpringBootHibernateExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootHibernateExample Maven Webapp</name>

<url>http://maven.apache.org</url>

<parent>

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

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

<version>1.5.3.RELEASE</version>

</parent>

<dependencies>

<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>mysql</groupId>

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

</dependency>

<!-- JSTL for JSP -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.threeten/threetenbp -->

<dependency>

<groupId>org.threeten</groupId>

<artifactId>threetenbp</artifactId>

<version>0.7.2</version>

</dependency>

</dependencies>

<build>

<finalName>SpringBootHibernateExample</finalName>

</build>

</project>

* **application.properties :**

spring.mvc.view.prefix: /WEB-INF/

spring.mvc.view.suffix: .jsp

# Database Configuration

db.driver: com.mysql.jdbc.Driver

db.url: jdbc:mysql://localhost:3306/CustomerData

db.username: root

db.password:

# Hibernate Configuration

hibernate.dialect: org.hibernate.dialect.MySQL5Dialect

hibernate.show\_sql: true

hibernate.hbm2ddl.auto: create

entitymanager.packagesToScan: org

spring.jpa.properties.hibernate.enable\_lazy\_load\_no\_trans=true

* **HibernateConfiguration.java:**

**package** org.arpit.java2blog;

**import** java.util.Properties;

**import** javax.sql.DataSource;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.jdbc.datasource.DriverManagerDataSource;

**import** org.springframework.orm.hibernate5.HibernateTransactionManager;

**import** org.springframework.orm.hibernate5.LocalSessionFactoryBean;

**import** org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration

@EnableTransactionManagement

**public** **class** HibernateConfiguration {

@Value("${db.driver}")

**private** String DRIVER;

@Value("${db.password}")

**private** String PASSWORD;

@Value("${db.url}")

**private** String URL;

@Value("${db.username}")

**private** String USERNAME;

@Value("${hibernate.dialect}")

**private** String DIALECT;

@Value("${hibernate.show\_sql}")

**private** String SHOW\_SQL;

@Value("${hibernate.hbm2ddl.auto}")

**private** String HBM2DDL\_AUTO;

@Value("${entitymanager.packagesToScan}")

**private** String PACKAGES\_TO\_SCAN;

@Bean

**public** DataSource dataSource() {

DriverManagerDataSource dataSource = **new** DriverManagerDataSource();

dataSource.setDriverClassName(DRIVER);

dataSource.setUrl(URL);

dataSource.setUsername(USERNAME);

dataSource.setPassword(PASSWORD);

**return** dataSource;

}

@Bean

**public** LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = **new** LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource());

sessionFactory.setPackagesToScan(PACKAGES\_TO\_SCAN);

Properties hibernateProperties = **new** Properties();

hibernateProperties.put("hibernate.dialect", DIALECT);

hibernateProperties.put("hibernate.show\_sql", SHOW\_SQL);

hibernateProperties.put("hibernate.hbm2ddl.auto", HBM2DDL\_AUTO);

sessionFactory.setHibernateProperties(hibernateProperties);

**return** sessionFactory;

}

@Bean

**public** HibernateTransactionManager transactionManager() {

HibernateTransactionManager transactionManager = **new** HibernateTransactionManager();

transactionManager.setSessionFactory(sessionFactory().getObject());

**return** transactionManager;

}

}

* **SpringBootHibernateApplication.java:**

**package** org.arpit.java2blog;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootHibernateApplication {

**public** **static** **void** main(String[] args)

{

SpringApplication.*run*(SpringBootHibernateApplication.**class**, args);

}

}

* **Customer.java:**

**package** org.arpit.java2blog.model;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

/\*

\* This is our model class and it corresponds to Customer table in database

\*/

@Entity

@Table(name="CUSTOMER")

**public** **class** Customer{

@Id

@Column(name="id")

@GeneratedValue(strategy=GenerationType.***IDENTITY***)

**int** id;

@Column(name="customerName")

String customerName;

@Column(name="email")

String email;

**public** Customer() {

**super**();

}

**public** Customer(String customerName,String email) {

**super**();

**this**.customerName=customerName;

**this**.email=email;

}

**public** String getCustomerName() {

**return** customerName;

}

**public** **void** setCustomerName(String customerName) {

**this**.customerName = customerName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

}

* **CustomerDao.java:**

**package** org.arpit.java2blog.dao;

**import** java.util.List;

**import** org.arpit.java2blog.model.Customer;

**public** **interface** CustomerDao {

**public** List<Customer> getAllCustomers() ;

**public** Customer getCustomer(**int** id) ;

**public** Customer addCustomer(Customer customer);

**public** **void** updateCustomer(Customer customer) ;

**public** **void** deleteCustomer(**int** id) ;

}

* **CustomerDaoImpl.java:**

**package** org.arpit.java2blog.dao;

**import** java.util.List;

**import** org.arpit.java2blog.model.Customer;

**import** org.hibernate.Hibernate;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

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

**import** org.springframework.stereotype.Repository;

@Repository

**public** **class** CustomerDaoImpl **implements** CustomerDao{

@Autowired

**private** SessionFactory sessionFactory;

**public** **void** setSessionFactory(SessionFactory sf) {

**this**.sessionFactory = sf;

}

**public** List<Customer> getAllCustomers() {

Session session = **this**.sessionFactory.getCurrentSession();

List<Customer> customerList = session.createQuery("from Customer").list();

**return** customerList;

}

**public** Customer getCustomer(**int** id) {

Session session = **this**.sessionFactory.getCurrentSession();

Customer customer = (Customer) session.get(Customer.**class**, id);

**return** customer;

}

**public** Customer addCustomer(Customer customer) {

Session session = **this**.sessionFactory.getCurrentSession();

session.save(customer);

**return** customer;

}

**public** **void** updateCustomer(Customer customer) {

Session session = **this**.sessionFactory.getCurrentSession();

Hibernate.*initialize*(customer);

session.update(customer);

}

**public** **void** deleteCustomer(**int** id) {

Session session = **this**.sessionFactory.getCurrentSession();

Customer p = (Customer) session.load(Customer.**class**, **new** Integer(id));

**if** (**null** != p) {

session.delete(p);

}

}

}

* **CustomerService.java:**

**package** org.arpit.java2blog.service;

**import** java.util.List;

**import** javax.transaction.Transactional;

**import** org.arpit.java2blog.dao.CustomerDao;

**import** org.arpit.java2blog.model.Customer;

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

**import** org.springframework.stereotype.Service;

@Service("customerService")

**public** **class** CustomerService {

@Autowired

CustomerDao customerDao;

@Transactional

**public** List<Customer> getAllCustomers() {

**return** customerDao.getAllCustomers();

}

@Transactional

**public** Customer getCustomer(**int** id) {

**return** customerDao.getCustomer(id);

}

@Transactional

**public** **void** addCustomer(Customer customer) {

customerDao.addCustomer(customer);

}

@Transactional

**public** **void** updateCustomer(Customer customer) {

customerDao.updateCustomer(customer);

}

@Transactional

**public** **void** deleteCustomer(**int** id) {

customerDao.deleteCustomer(id);

}

}

* **CustomerController.java:**

**package** org.arpit.java2blog.controller;

**import** java.util.List;

**import** org.arpit.java2blog.model.Customer;

**import** org.arpit.java2blog.service.CustomerService;

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

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

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

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

@Controller

**public** **class** CustomerController {

@Autowired

CustomerService customerService;

@RequestMapping(value = "/getAllCustomers", method = RequestMethod.***GET***, headers = "Accept=application/json")

**public** String getAllCustomers(Model model) {

List<Customer> listOfCustomers = customerService.getAllCustomers();

model.addAttribute("customer", **new** Customer());

model.addAttribute("listOfCustomers", listOfCustomers);

**return** "customerDetails";

}

@RequestMapping(value = "/", method = RequestMethod.***GET***, headers = "Accept=application/json")

**public** String goToHomePage() {

**return** "redirect:/getAllCustomers";

}

@RequestMapping(value = "/getCustomer/{id}", method = RequestMethod.***GET***, headers = "Accept=application/json")

**public** Customer getCustomerById(@PathVariable **int** id) {

**return** customerService.getCustomer(id);

}

@RequestMapping(value = "/addCustomer", method = RequestMethod.***POST***, headers = "Accept=application/json")

**public** String addCustomer(@ModelAttribute("customer") Customer customer) {

**if**(customer.getId()==0)

{

customerService.addCustomer(customer);

}

**else**

{

customerService.updateCustomer(customer);

}

**return** "redirect:/getAllCustomers";

}

@RequestMapping(value = "/updateCustomer/{id}", method = RequestMethod.***GET***, headers = "Accept=application/json")

**public** String updateCustomer(@PathVariable("id") **int** id,Model model) {

model.addAttribute("customer", **this**.customerService.getCustomer(id));

model.addAttribute("listOfCustomers", **this**.customerService.getAllCustomers());

**return** "customerDetails";

}

@RequestMapping(value = "/deleteCustomer/{id}", method = RequestMethod.***GET***, headers = "Accept=application/json")

**public** String deleteCustomer(@PathVariable("id") **int** id) {

customerService.deleteCustomer(id);

**return** "redirect:/getAllCustomers";

}

}

* **customerDetails.jsp:**

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

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

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

<html>

<head>

<style>

*.blue-button*{

background: *#25A6E1*;

filter: *progid*: *DXImageTransform.Microsoft.gradient*( *startColorstr*=*'#25A6E1',endColorstr*=*'#188BC0',GradientType*=*0)*;

padding:*3px 5px*;

color:*#fff*;

font-family:*'Helvetica Neue',sans-serif*;

font-size:*12px*;

border-radius:*2px*;

-moz-border-radius:*2px*;

-webkit-border-radius:*4px*;

border:*1px solid #1A87B9*

}

**table** {

font-family: *"Helvetica Neue", Helvetica, sans-serif*;

width: *50%*;

}

**th** {

background: *SteelBlue*;

color: *white*;

}

**td,th**{

border: *1px solid gray*;

width: *25%*;

text-align: *left*;

padding: *5px 10px*;

}

</style>

</head>

<body>

<form:form method=*"post"* modelAttribute=*"customer"* action=*"*${pageContext.request.contextPath}*/addCustomer"*>

<table>

<tr>

<th colspan=*"2"*>Add Customer</th>

</tr>

<tr>

<form:hidden path=*"id"* />

<td><form:label path=*"customerName"*>Customer Name:</form:label></td>

<td><form:input path=*"customerName"* size=*"30"* maxlength=*"30"*></form:input></td>

</tr>

<tr>

<td><form:label path=*"email"*>Email:</form:label></td>

<td><form:input path=*"email"* size=*"30"* maxlength=*"30"*></form:input></td>

</tr>

<tr>

<td colspan=*"2"*><input type=*"submit"*

class=*"blue-button"* /></td>

</tr>

</table>

</form:form>

</br>

<h3>Customer List</h3>

<c:if test=*"*${!**empty** listOfCustomers}*"*>

<table class=*"tg"*>

<tr>

<th width=*"80"*>Id</th>

<th width=*"120"*>Customer Name</th>

<th width=*"120"*>Email</th>

<th width=*"60"*>Edit</th>

<th width=*"60"*>Delete</th>

</tr>

<c:forEach items=*"*${listOfCustomers}*"* var=*"customer"*>

<tr>

<td>${customer.id}</td>

<td>${customer.customerName}</td>

<td>${customer.email}</td>

<td><a href=*"*<c:url value='*/updateCustomer/*${customer.id}' />*"* >Edit</a></td>

<td><a href=*"*<c:url value='*/deleteCustomer/*${customer.id}' />*"* >Delete</a></td>

</tr>

</c:forEach>

</table>

</c:if>

</body>

</html>



**Same Example Using AngularJS:**

<html>

<head>

<script

src=*"https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"*></script>

<title>Spring Boot AngularJS example</title>

<script type=*"text/javascript"*>

**var** app = angular.module("CustomerManagement", []);

//Controller Part

app.controller("CustomerController", **function**($scope, $http) {

$scope.customers = [];

$scope.customerForm = {

id : -1,

customerName : "",

email : ""

};

//Now load the data from server

\_refreshCustomerData();

//HTTP POST/PUT methods for add/edit customer

// with the help of id, we are going to find out whether it is put or post operation

$scope.submitCustomer = **function**() {

**var** method = "";

**var** url = "";

**if** ($scope.customerForm.id == -1) {

//Id is absent in form data, it is create new customer operation

method = "POST";

url = '/addCustomer';

} **else** {

//Id is present in form data, it is edit customer operation

method = "PUT";

url = '/addCustomer';

}

$http({

method : method,

url : url,

data : angular.toJson($scope.customerForm),

headers : {

'Content-Type' : 'application/json'

}

}).then( \_success, \_error );

};

//HTTP DELETE- delete customer by Id

$scope.deleteCustomer = **function**(customer) {

$http({

method : 'DELETE',

url : '/deleteCustomer/' + customer.id

}).then(\_success, \_error);

};

// In case of edit, populate form fields and assign form.id with customer id

$scope.editCustomer = **function**(customer) {

$scope.customerForm.customerName = customer.customerName;

$scope.customerForm.email = customer.email;

$scope.customerForm.id = customer.id;

};

/\* Private Methods \*/

//HTTP GET- get all customers collection

**function** \_refreshCustomerData() {

$http({

method : 'GET',

url : 'http://localhost:8080/getAllCustomers'

}).then(**function** successCallback(response) {

$scope.customers = response.data;

}, **function** errorCallback(response) {

console.log(response.statusText);

});

}

**function** \_success(response) {

\_refreshCustomerData();

\_clearFormData()

}

**function** \_error(response) {

console.log(response.statusText);

}

//Clear the form

**function** \_clearFormData() {

$scope.customerForm.id = -1;

$scope.customerForm.customerName = "";

$scope.customerForm.email = "";

};

});

</script>

<style>

*.blue-button* {

background: *#25A6E1*;

filter: *progid*: *DXImageTransform.Microsoft.gradient*( *startColorstr*=*'#25A6E1',*

*endColorstr*=*'#188BC0', GradientType*=*0)*;

padding: *3px 5px*;

color: *#fff*;

font-family: *'Helvetica Neue', sans-serif*;

font-size: *12px*;

border-radius: *2px*;

-moz-border-radius: *2px*;

-webkit-border-radius: *4px*;

border: *1px solid #1A87B9*

}

*.red-button* {

background: *#CD5C5C*;

padding: *3px 5px*;

color: *#fff*;

font-family: *'Helvetica Neue', sans-serif*;

font-size: *12px*;

border-radius: *2px*;

-moz-border-radius: *2px*;

-webkit-border-radius: *4px*;

border: *1px solid #CD5C5C*

}

**table** {

font-family: *"Helvetica Neue", Helvetica, sans-serif*;

width: *50%*;

}

**caption** {

text-align: *left*;

color: *silver*;

font-weight: *bold*;

text-transform: *uppercase*;

padding: *5px*;

}

**th** {

background: *SteelBlue*;

color: *white*;

}

**tbody** **tr***:nth-child(even)* {

background: *WhiteSmoke*;

}

**tbody** **tr** **td***:nth-child(2)* {

text-align: *center*;

}

**tbody** **tr** **td***:nth-child(3)***,** **tbody** **tr** **td***:nth-child(4)* {

text-align: *center*;

font-family: *monospace*;

}

**tfoot** {

background: *SeaGreen*;

color: *white*;

text-align: *right*;

}

**tfoot** **tr** **th***:last-child* {

font-family: *monospace*;

}

**td,** **th** {

border: *1px solid gray*;

width: *25%*;

text-align: *left*;

padding: *5px 10px*;

}

</style>

<head>

<body ng-app=*"CustomerManagement"* ng-controller=*"CustomerController"*>

<h1>Customer Mart</h1>

<form ng-submit=*"submitCustomer()"*>

<table>

<tr>

<th colspan=*"2"*>Add/Edit customer</th>

</tr>

<tr>

<td>Customer Name</td>

<td><input type=*"text"* ng-model=*"customerForm.customerName"* /></td>

</tr>

<tr>

<td>Email</td>

<td><input type=*"text"* ng-model=*"customerForm.email"* /></td>

</tr>

<tr>

<td colspan=*"2"*><input type=*"submit"* value=*"Submit"*

class=*"blue-button"* /></td>

</tr>

</table>

</form>

<table>

<tr>

<th>Customer Name</th>

<th>Email</th>

<th>Operations</th>

</tr>

<tr ng-repeat=*"customer in customers"*>

<td>{{ customer.customerName }}</td>

<td>{{ customer.email }}</td>

<td><a ng-click=*"editCustomer(customer)"* class=*"blue-button"*>Edit</a>

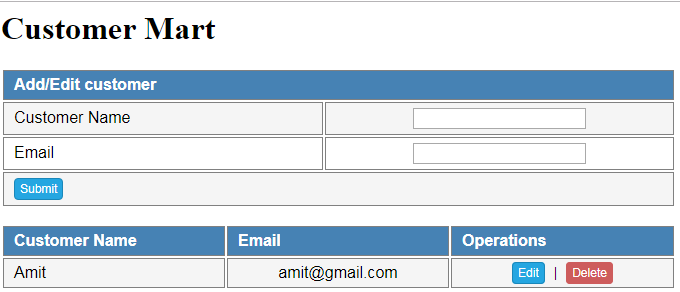
| <a ng-click=*"deleteCustomer(customer)"* class=*"red-button"*>Delete</a></td>

</tr>

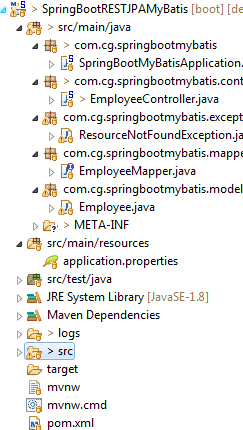
</table>

</body>

</html>



* **Spring Boot MyBatis Annotation Based Integration:**



* **application.properties:**

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url = jdbc:mysql://localhost:3306/spring-orm

spring.datasource.username = root

spring.datasource.password =

server.port=8085

logging.level.org.springframework=WARN

logging.level.com.spring.ibatis.UserMapper=DEBUG

logging.file=logs/spring-boot-logging.log

* **Employee.java:**

**package** com.cg.springbootmybatis.model;

**import** java.io.Serializable;

**import** javax.persistence.Entity;

**import** javax.persistence.EntityListeners;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

**import** javax.validation.constraints.NotBlank;

**import** org.springframework.data.jpa.domain.support.AuditingEntityListener;

@Entity

@Table(name = "employee")

@EntityListeners(AuditingEntityListener.**class**)

**public** **class** Employee **implements** Serializable {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** **int** salary;

**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** getSalary() {

**return** salary;

}

**public** **void** setSalary(**int** salary) {

**this**.salary = salary;

}

}

* **EmployeeMapper.java:**

**package** com.cg.springbootmybatis.mapper;

**import** java.util.List;

**import** org.apache.ibatis.annotations.Delete;

**import** org.apache.ibatis.annotations.Insert;

**import** org.apache.ibatis.annotations.Mapper;

**import** org.apache.ibatis.annotations.Param;

**import** org.apache.ibatis.annotations.Select;

**import** org.apache.ibatis.annotations.Update;

**import** com.cg.springbootmybatis.model.Employee;

@Mapper

**public** **interface** EmployeeMapper {

@Insert("INSERT INTO EMPLOYEE ( name, salary ) VALUES ( #{emp.name}, #{emp.salary})")

Integer insertEmployee(@Param("emp") Employee emp) **throws** Exception;

@Select("select \* from employee")

List<Employee> getAllEmployees();

@Select("select \* from employee where id = #{id}")

Employee findById(@Param("id") Integer id);

@Delete("DELETE from employee where id = #{id}")

Integer deleteById(@Param("id") Integer id);

@Update("UPDATE EMPLOYEE SET name = #{emp.name},salary = #{emp.salary} WHERE id = #{emp.id}")

Integer updateEmployee(@Param("emp") Employee emp) **throws** Exception;

}

* **ResourceNotFoundException.java:**

**package** com.cg.springbootmybatis.exception;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.***NOT\_FOUND***)

**public** **class** ResourceNotFoundException **extends** RuntimeException {

**private** String resourceName;

**private** String fieldName;

**private** Object fieldValue;

**public** ResourceNotFoundException( String resourceName, String fieldName, Object fieldValue) {

**super**(String.*format*("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));

**this**.resourceName = resourceName;

**this**.fieldName = fieldName;

**this**.fieldValue = fieldValue;

}

**public** String getResourceName() {

**return** resourceName;

}

**public** String getFieldName() {

**return** fieldName;

}

**public** Object getFieldValue() {

**return** fieldValue;

}

}

* **EmployeeController.java:**

**package** com.cg.springbootmybatis.controller;

**import** java.util.List;

**import** javax.validation.Valid;

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

**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.PutMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.cg.springbootmybatis.exception.ResourceNotFoundException;

**import** com.cg.springbootmybatis.mapper.EmployeeMapper;

**import** com.cg.springbootmybatis.model.Employee;

@RestController

@RequestMapping("/api/employee/")

**public** **class** EmployeeController {

@Autowired

EmployeeMapper employeeMapper;

// Get All Employees

@GetMapping("/all")

**public** List<Employee> getAllNotes() {

**return** employeeMapper.getAllEmployees();

}

// Create a new Employee

@PostMapping("/create")

**public** Integer createNote(@Valid @RequestBody Employee employee) **throws** Exception {

**return** employeeMapper.insertEmployee(employee);

}

// Delete a Employee

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

**public** ResponseEntity<?> deleteEmployee(@PathVariable(value = "id") Integer id) {

Employee employee = employeeMapper.findById(id);

**if** (employee != **null**)

employeeMapper.deleteById(id);

**else**

**throw** **new** ResourceNotFoundException("Employee", "id", id);

**return** ResponseEntity.*ok*().build();

}

// Get a Single Employee

@GetMapping("/get/{id}")

**public** Employee getNoteById(@PathVariable(value = "id") Integer id) {

Employee employee = employeeMapper.findById(id);

**if** (employee != **null**)

**return** employee;

**else**

**throw** **new** ResourceNotFoundException("Employee", "id", id);

}

// Update a Employee

@PutMapping("/update/{id}")

**public** Integer updateEmployee(@PathVariable(value = "id") Integer id, @Valid @RequestBody Employee employeeDetails)

**throws** Exception {

Integer updatedEmployee = 0;

Employee employee = employeeMapper.findById(id);

**if** (employee != **null**) {

System.***out***.println("Employee Details to update:");

System.***out***.println(

employeeDetails.getId() + "\t" + employeeDetails.getName() + "\t" + employeeDetails.getSalary());

employee.setId(id);

employee.setName(employeeDetails.getName());

employee.setSalary(employeeDetails.getSalary());

updatedEmployee = employeeMapper.updateEmployee(employee);

} **else**

**throw** **new** ResourceNotFoundException("Employee", "id", id);

**return** updatedEmployee;

}

}

* **SpringBootMyBatisApplication.java:**

**package** com.cg.springbootmybatis;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication

@EnableJpaAuditing

**public** **class** SpringBootMyBatisApplication {

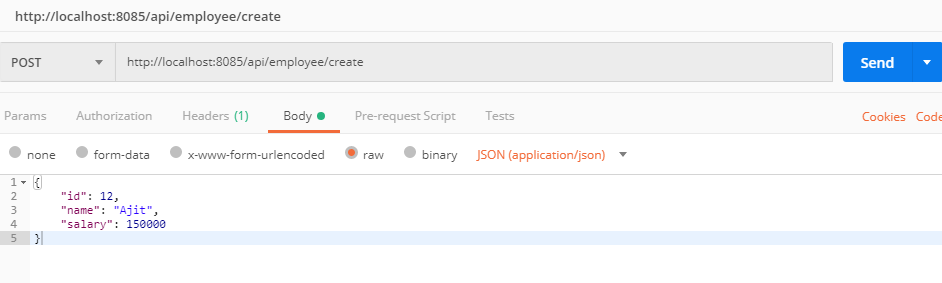
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootMyBatisApplication.**class**, args);

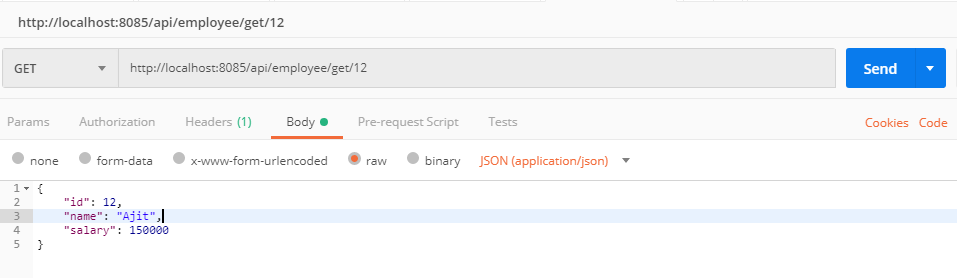
}

}

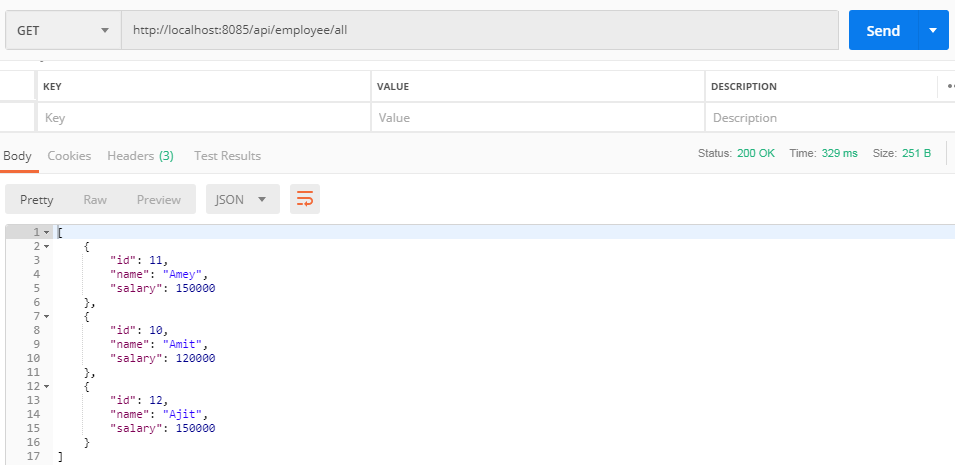
**A] Add Employee**



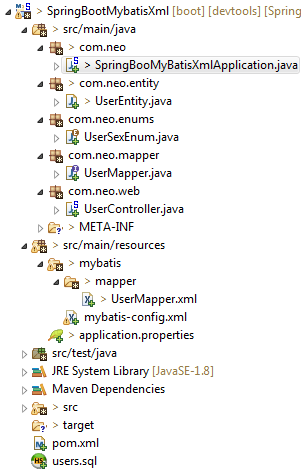
**B] GET Employee Details**



**C] GET All Employees**



* **Spring Boot MyBatis XML Based Integration:**



* **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.neo</groupId>

<artifactId>spring-boot-mybatis-xml</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>spring-boot-mybatis-xml</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.1.1</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

<excludes>

<exclude>bootstrap-test.properties</exclude>

<exclude>bootstrap-dev.properties</exclude>

<exclude>bootstrap-pro.properties</exclude>

<exclude>bootstrap.properties</exclude>

</excludes>

</resource>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

<includes>

<include>bootstrap-${env}.properties</include>

<include>bootstrap.properties</include>

</includes>

</resource>

</resources>

<plugins>

<plugin>

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

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

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

</project>

* **application.properties :**

mybatis.config-locations=classpath:mybatis/mybatis-config.xml

mybatis.mapper-locations=classpath:mybatis/mapper/\*.xml

mybatis.type-aliases-package=com.neo.entity

spring.datasource.driverClassName = com.mysql.jdbc.Driver

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

spring.datasource.username = root

spring.datasource.password =

server.port=8085

* **mybatis-config.xml :**

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

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<typeAliases>

<typeAlias alias=*"Integer"* type=*"java.lang.Integer"* />

<typeAlias alias=*"Long"* type=*"java.lang.Long"* />

<typeAlias alias=*"HashMap"* type=*"java.util.HashMap"* />

<typeAlias alias=*"LinkedHashMap"* type=*"java.util.LinkedHashMap"* /> <typeAlias alias=*"ArrayList"* type=*"java.util.ArrayList"* />

<typeAlias alias=*"LinkedList"* type=*"java.util.LinkedList"* />

</typeAliases>

</configuration>

* **UserMapper.xml**

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace=*"com.neo.mapper.UserMapper"* >

<resultMap id=*"BaseResultMap"* type=*"com.neo.entity.UserEntity"* >

<id column=*"id"* property=*"id"* jdbcType=*"BIGINT"* />

<result column=*"userName"* property=*"userName"* jdbcType=*"VARCHAR"* />

<result column=*"passWord"* property=*"passWord"* jdbcType=*"VARCHAR"* />

<result column=*"user\_sex"* property=*"userSex"* javaType=*"com.neo.enums.UserSexEnum"*/>

<result column=*"nick\_name"* property=*"nickName"* jdbcType=*"VARCHAR"* />

</resultMap>

<sql id=*"Base\_Column\_List"* >

id, userName, passWord, user\_sex, nick\_name

</sql>

<select id=*"getAll"* resultMap=*"BaseResultMap"* >

SELECT

<include refid=*"Base\_Column\_List"* />

FROM users

</select>

<select id=*"getOne"* parameterType=*"java.lang.Long"* resultMap=*"BaseResultMap"* >

SELECT

<include refid=*"Base\_Column\_List"* />

FROM users

WHERE id = #{id}

</select>

<insert id=*"insert"* parameterType=*"com.neo.entity.UserEntity"* >

INSERT INTO

users

(userName,passWord,user\_sex,nick\_name)

VALUES

(#{userName}, #{passWord}, #{userSex}, #{nickName})

</insert>

<update id=*"update"* parameterType=*"com.neo.entity.UserEntity"* >

UPDATE

users

SET

userName = #{userName},

passWord = #{passWord},

nick\_name = #{nickName}

WHERE id = #{id}

</update>

<delete id=*"delete"* parameterType=*"java.lang.Long"* >

DELETE FROM users WHERE id =#{id}

</delete>

</mapper>

* **UserEntity.java:**

**package** com.neo.entity;

**import** java.io.Serializable;

**import** com.neo.enums.UserSexEnum;

**public** **class** UserEntity **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Long id;

**private** String userName;

**private** String passWord;

**private** UserSexEnum userSex;

**private** String nickName;

**public** UserEntity() {

**super**();

}

**public** UserEntity(String userName, String passWord, UserSexEnum userSex) {

**super**();

**this**.passWord = passWord;

**this**.userName = userName;

**this**.userSex = userSex;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getPassWord() {

**return** passWord;

}

**public** **void** setPassWord(String passWord) {

**this**.passWord = passWord;

}

**public** UserSexEnum getUserSex() {

**return** userSex;

}

**public** **void** setUserSex(UserSexEnum userSex) {

**this**.userSex = userSex;

}

**public** String getNickName() {

**return** nickName;

}

**public** **void** setNickName(String nickName) {

**this**.nickName = nickName;

}

@Override

**public** String toString() {

// **TODO** Auto-generated method stub

**return** "userName " + **this**.userName + ", pasword " + **this**.passWord + "sex " + userSex.name();

}

}

* **UserSexEnum.java**

**package** com.neo.enums;

**public** **enum** UserSexEnum {

***MAN***, ***WOMAN***

}

* **UserMapper.java:**

**package** com.neo.mapper;

**import** java.util.List;

**import** com.neo.entity.UserEntity;

**public** **interface** UserMapper {

List<UserEntity> getAll();

UserEntity getOne(Long id);

**void** insert(UserEntity user);

**void** update(UserEntity user);

**void** delete(Long id);

}

* **UserController.java:**

**package** com.neo.web;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.neo.entity.UserEntity;

**import** com.neo.mapper.UserMapper;

@RestController

@RequestMapping("user-portal")

**public** **class** UserController {

@Autowired

**private** UserMapper userMapper;

@RequestMapping(value="/getUsers",method=RequestMethod.***GET***)

**public** List<UserEntity> getUsers() {

List<UserEntity> users=userMapper.getAll();

**return** users;

}

@RequestMapping(value="/getUser/{id}",method=RequestMethod.***GET***)

**public** UserEntity getUser(@PathVariable("id") Long id) {

UserEntity user=userMapper.getOne(id);

**return** user;

}

@RequestMapping(value="/add",method=RequestMethod.***POST***)

**public** **void** save(@RequestBody UserEntity user) {

userMapper.insert(user);

}

@RequestMapping(value="/update/{id}",method=RequestMethod.***PUT***)

**public** **void** update(@PathVariable("id") Long id,UserEntity user) {

UserEntity userExists=userMapper.getOne(id);

**if**(userExists != **null**)

userMapper.update(user);

}

@RequestMapping(value="/delete/{id}",method=RequestMethod.***DELETE***)

**public** **void** delete(@PathVariable("id") Long id) {

userMapper.delete(id);

}

}

* **SpringBooMyBatisXmlApplication.java:**

**package** com.neo;

**import** org.mybatis.spring.annotation.MapperScan;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

@MapperScan("com.neo.mapper")

**public** **class** SpringBooMyBatisXmlApplication {

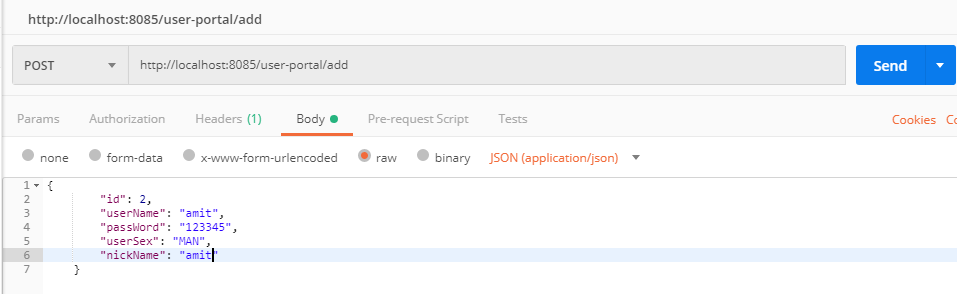
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBooMyBatisXmlApplication.**class**, args);

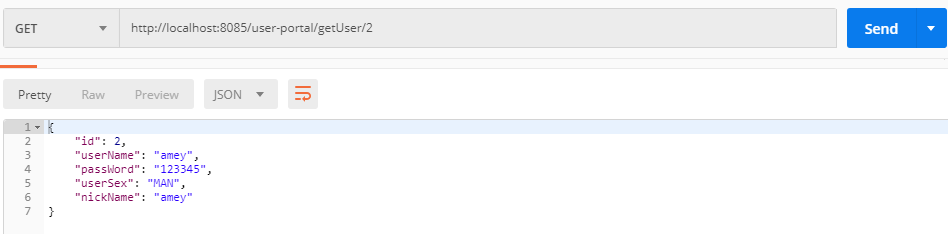
}

}

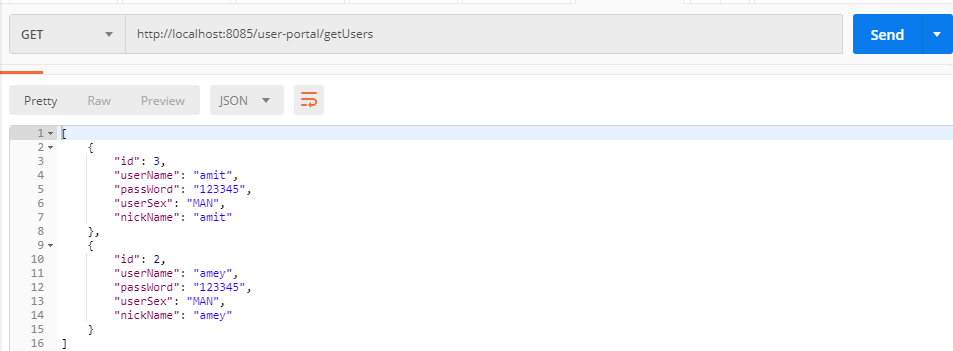
**A] Add User**



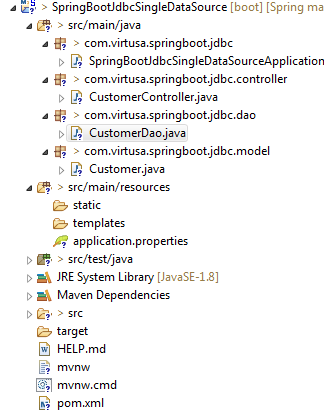
**B] Get User Details**



**C] Get All Users**



* **Spring Boot JDBC Single DataSource :**



* **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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootJdbcSingleDataSource</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootJdbcSingleDataSource</name>

<description>Spring Boot Jdbc Single DataSource</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<!-- MySql dependency -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<dependency>

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

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

</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**

spring.datasource.url=jdbc:mysql://localhost/spring-orm

spring.datasource.username=root

spring.datasource.password=

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

* **Customer.java**

**package** com.virtusa.springboot.jdbc.model;

**public** **class** Customer {

**private** **int** id;

**private** String custName;

**private** String address;

**private** String phone;

**private** String contactPerson;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getCustName() {

**return** custName;

}

**public** **void** setCustName(String custName) {

**this**.custName = custName;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getContactPerson() {

**return** contactPerson;

}

**public** **void** setContactPerson(String contactPerson) {

**this**.contactPerson = contactPerson;

}

}

* **CustomerDao.java**

**package** com.virtusa.springboot.jdbc.dao;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

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

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Repository;

**import** com.virtusa.springboot.jdbc.model.Customer;

@Repository

**public** **class** CustomerDao {

@Autowired

**private** JdbcTemplate jdbcTemplate;

**private** **static** **final** String ***SQL*** = "select \* from customer";

**public** List<Customer> getAllCustomers() {

List<Customer> customers = **new** ArrayList<Customer>();

List<Map<String, Object>> rows = jdbcTemplate.queryForList(***SQL***);

**for** (Map<String, Object> row : rows)

{

Customer customer = **new** Customer();

customer.setId((**int**)row.get("id"));

customer.setCustName((String)row.get("name"));

customer.setAddress((String)row.get("address"));

customer.setPhone((String)row.get("phone"));

customer.setContactPerson((String)row.get("contact"));

customers.add(customer);

}

**return** customers;

}

}

* **CustomerController.java**

**package** com.virtusa.springboot.jdbc.controller;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.virtusa.springboot.jdbc.dao.CustomerDao;

**import** com.virtusa.springboot.jdbc.model.Customer;

@RestController

**public** **class** CustomerController {

@Autowired

**public** CustomerDao dao;

@RequestMapping("/getCustInfo")

**public** List<Customer> customerInformation() {

List<Customer> customers = dao.getAllCustomers();

**return** customers;

}

}

* **SpringBootJdbcSingleDataSourceApplication.java**

**package** com.virtusa.springboot.jdbc;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootJdbcSingleDataSourceApplication {

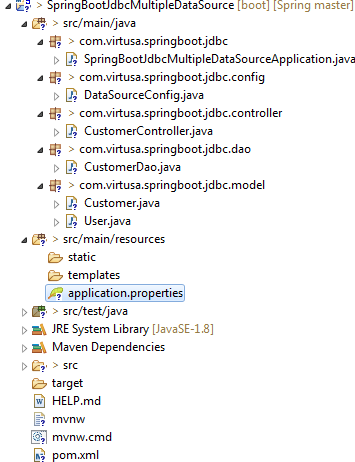
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootJdbcSingleDataSourceApplication.**class**, args);

}

}

* **Spring Boot JDBC Multiple DataSource :**



* **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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootJdbcMultipleDataSource</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootJdbcMultipleDataSource</name>

<description>Spring Boot Jdbc Multiple DataSource</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<!-- MySql dependency -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!-- Configuration -->

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<dependency>

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

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

</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**

spring.datasource.url=jdbc:mysql://localhost/spring-orm

spring.datasource.username=root

spring.datasource.password=

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

spring.secondDatasource.url=jdbc:mysql://localhost/spring-orm

spring.secondDatasource.username=root

spring.secondDatasource.password=

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

* **Customer.java:**

**package** com.virtusa.springboot.jdbc.model;

**public** **class** Customer {

**private** **int** id;

**private** String custName;

**private** String address;

**private** String phone;

**private** String contactPerson;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getCustName() {

**return** custName;

}

**public** **void** setCustName(String custName) {

**this**.custName = custName;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getContactPerson() {

**return** contactPerson;

}

**public** **void** setContactPerson(String contactPerson) {

**this**.contactPerson = contactPerson;

}

}

* **User.java**

**package** com.virtusa.springboot.jdbc.model;

**public** **class** User {

**private** **int** id;

**private** String firstName;

**private** String lastName;

**private** String emailId;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmailId() {

**return** emailId;

}

**public** **void** setEmailId(String emailId) {

**this**.emailId = emailId;

}

}

* **DataSourceConfig.java**

**package** com.virtusa.springboot.jdbc.config;

**import** javax.sql.DataSource;

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.boot.jdbc.DataSourceBuilder;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.Primary;

**import** org.springframework.jdbc.core.JdbcTemplate;

@Configuration

**public** **class** DataSourceConfig {

@Bean(name="datasource")

@Primary

@ConfigurationProperties(prefix = "spring.datasource")

**public** DataSource firstDataSource() {

**return** DataSourceBuilder.*create*().build();

}

@Bean(name="secondDatasource")

@ConfigurationProperties(prefix = "spring.secondDatasource")

**public** DataSource secondDataSource() {

**return** DataSourceBuilder.*create*().build();

}

@Bean

**public** JdbcTemplate jdbcTemplateOne(@Qualifier("firstDataSource") DataSource ds) {

**return** **new** JdbcTemplate(ds);

}

@Bean

**public** JdbcTemplate jdbcTemplateTwo(@Qualifier("secondDataSource") DataSource ds) {

**return** **new** JdbcTemplate(ds);

}

}

* **CustomerDao.java**

**package** com.virtusa.springboot.jdbc.dao;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

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

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Repository;

**import** com.virtusa.springboot.jdbc.model.Customer;

**import** com.virtusa.springboot.jdbc.model.User;

@Repository

**public** **class** CustomerDao {

@Autowired

**private** JdbcTemplate jdbcTemplate;

**private** **static** **final** String ***CUSTSQL*** = "select \* from customer";

**public** List<Customer> getAllCustomers() {

List<Customer> customers = **new** ArrayList<Customer>();

List<Map<String, Object>> rows = jdbcTemplate.queryForList(***CUSTSQL***);

**for** (Map<String, Object> row : rows)

{

Customer customer = **new** Customer();

customer.setId((**int**)row.get("id"));

customer.setCustName((String)row.get("name"));

customer.setAddress((String)row.get("address"));

customer.setPhone((String)row.get("phone"));

customer.setContactPerson((String)row.get("contact"));

customers.add(customer);

}

**return** customers;

}

**private** **static** **final** String ***USERSQL*** = "select \* from user";

**public** List<User> getAllUsers() {

List<User> users = **new** ArrayList<User>();

List<Map<String, Object>> rows = jdbcTemplate.queryForList(***USERSQL***);

**for** (Map<String, Object> row : rows)

{

User user = **new** User();

user.setId((**int**)row.get("id"));

user.setFirstName((String)row.get("first\_name"));

user.setLastName((String)row.get("last\_name"));

user.setEmailId((String)row.get("email"));

users.add(user);

}

**return** users;

}

}

* **CustomerController.java**

**package** com.virtusa.springboot.jdbc.controller;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.virtusa.springboot.jdbc.dao.CustomerDao;

**import** com.virtusa.springboot.jdbc.model.Customer;

**import** com.virtusa.springboot.jdbc.model.User;

@RestController

**public** **class** CustomerController {

@Autowired

**public** CustomerDao dao;

@RequestMapping("/getCustInfo")

**public** List<Customer> customerInformation() {

List<Customer> customers = dao.getAllCustomers();

**return** customers;

}

@RequestMapping("/getUserInfo")

**public** List<User> userInformation() {

List<User> users = dao.getAllUsers();

**return** users;

}

}

* **SpringBootJdbcMultipleDataSourceApplication.java**

**package** com.virtusa.springboot.jdbc;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootJdbcMultipleDataSourceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootJdbcMultipleDataSourceApplication.**class**, args);

}

}

* **Spring Boot XML Request and Response :**

1. Import below dependency in pom.xml:

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

1. **Customer.java**

**package** com.example.demo;

**public** **class** Customer {

**private** **int** custNo;

**private** String name;

**private** String country;

**public** Customer() {

}

**public** Customer(**int** custNumber, String name, String country) {

**this**.custNo = custNumber;

**this**.name = name;

**this**.country = country;

}

**public** **int** getCustNo() {

**return** custNo;

}

**public** **void** setCustNo(**int** custNo) {

**this**.custNo = custNo;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getCountry() {

**return** country;

}

**public** **void** setCountry(String country) {

**this**.country = country;

}

}

1. **CustomerController.java**

**package** com.example.demo;

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

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

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** CustomerController {

@GetMapping(path = "/get-cust-info")

**public** Customer customerInformation() {

Customer cust = **new** Customer();

cust.setCustNo(100);

cust.setName("Bank of America");

cust.setCountry("United States");

**return** cust;

}

@PostMapping(path = "/save-cust-info")

**public** String customerInformation(@RequestBody Customer cust) {

**return** "Customer information saved successfully ::." + cust.getCustNo() + " " + cust.getName() + " " + cust.getCountry();

}

}

1. **SpringBootWithXmlRequestResponseApplication.java**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootWithXmlRequestResponseApplication {

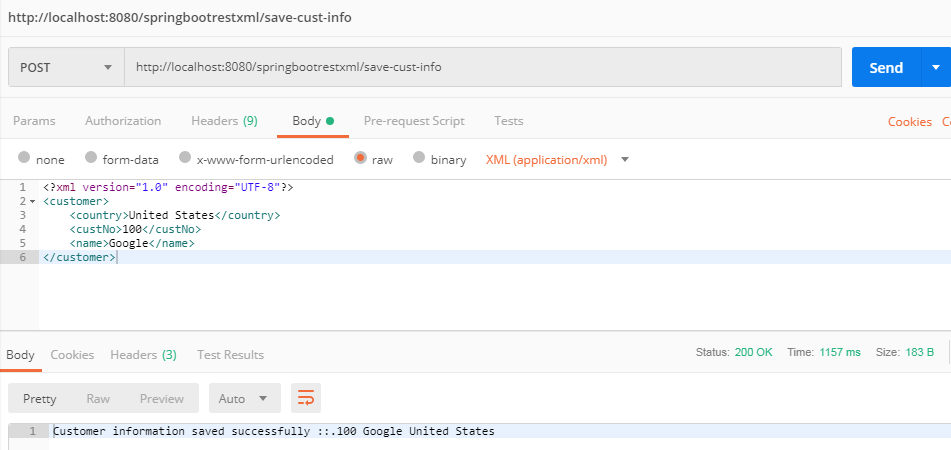
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootWithXmlRequestResponseApplication.**class**, args);

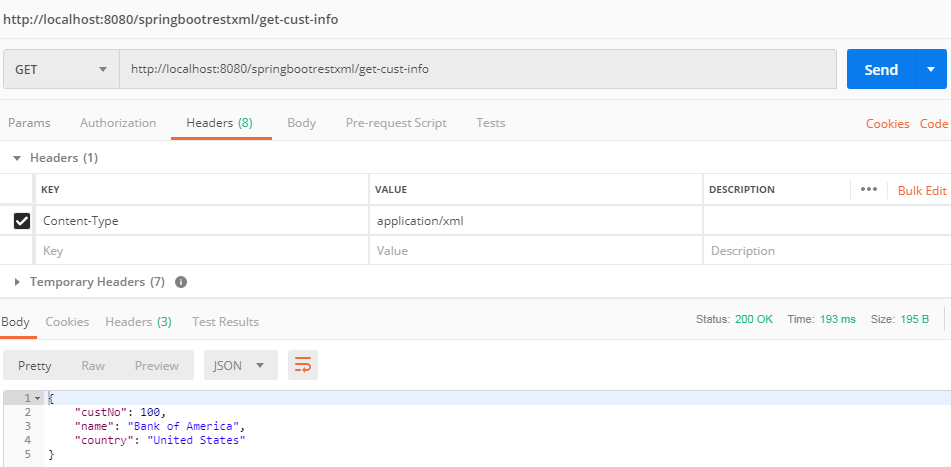
}

}

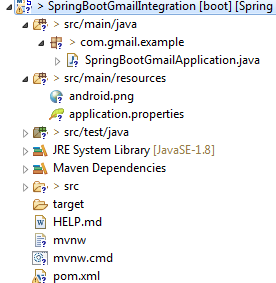
**XML Request:**

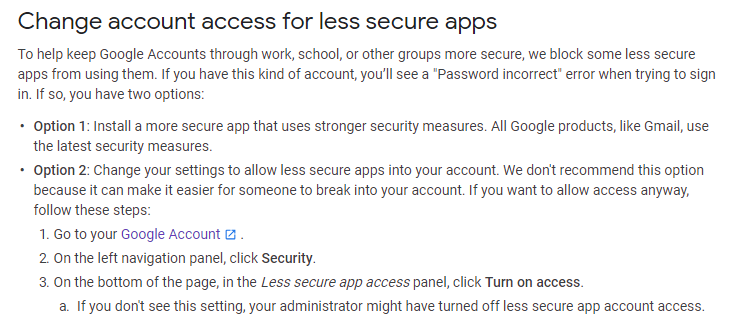


**XML Response:**



* **Spring Boot Gmail Integration :**





* **application.properties:**

logging.level.org.springframework.mail=DEBUG

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=amitbaramatimca

spring.mail.password=Rahman12#$

# Other properties

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

spring.mail.properties.mail.smtp.connectiontimeout=5000

spring.mail.properties.mail.smtp.timeout=5000

spring.mail.properties.mail.smtp.writetimeout=5000

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

* **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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootGmailIntegration</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootGmailIntegration</name>

<description>Spring Boot How to send email via SMTP</description>

<properties>

<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-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.0</version>

</plugin>

</plugins>

</build>

</project>

* **SpringBootGmailApplication.java**

**package** com.gmail.example;

**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.ClassPathResource;

**import** org.springframework.mail.SimpleMailMessage;

**import** org.springframework.mail.javamail.JavaMailSender;

**import** org.springframework.mail.javamail.MimeMessageHelper;

**import** javax.mail.MessagingException;

**import** javax.mail.internet.MimeMessage;

**import** java.io.IOException;

@SpringBootApplication

**public** **class** SpringBootGmailApplication **implements** CommandLineRunner {

@Autowired

**private** JavaMailSender javaMailSender;

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootGmailApplication.**class**, args);

}

@Override

**public** **void** run(String... args) {

System.***out***.println("Sending Email...");

**try** {

sendEmail();

sendEmailWithAttachment();

} **catch** (MessagingException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("Done");

}

**void** sendEmail() {

SimpleMailMessage msg = **new** SimpleMailMessage();

msg.setTo("amitbaramatimca@gmail.com");

msg.setSubject("Testing from Spring Boot");

msg.setText("Hello World \n Spring Boot Email");

javaMailSender.send(msg);

}

**void** sendEmailWithAttachment() **throws** MessagingException, IOException {

MimeMessage msg = javaMailSender.createMimeMessage();

MimeMessageHelper helper = **new** MimeMessageHelper(msg, **true**);

helper.setTo("amitbaramatimca@gmail.com");

helper.setSubject("Testing from Spring Boot");

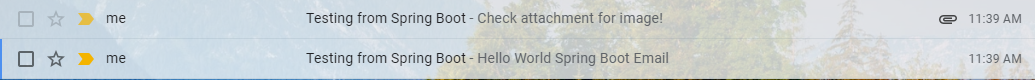
helper.setText("<h1>Check attachment for image!</h1>", **true**);

helper.addAttachment("my\_photo.png", **new** ClassPathResource("android.png"));

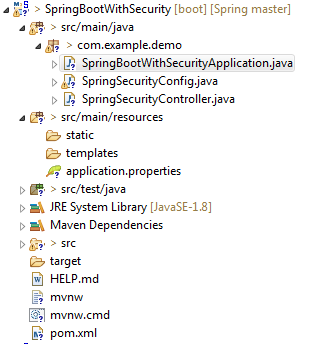
javaMailSender.send(msg);

}

}



* **Spring Boot Security Integration :**



* **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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootWithSecurity</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootWithSecurity</name>

<description>Spring Boot With Security</description>

<properties>

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-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>

* **SpringSecurityConfig.java**

**package** com.example.demo;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

**public** **class** SpringSecurityConfig **extends** WebSecurityConfigurerAdapter {

// Authentication : set user/password details and mention the role

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.inMemoryAuthentication().passwordEncoder(org.springframework.security.crypto.password.~~NoOpPasswordEncoder~~.~~getInstance~~())

.withUser("user").password("pass").roles("USER")

.and()

.withUser("admin").password("pass").roles("USER", "ADMIN");

}

// Authorization : mention which role can access which URL

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic().and().authorizeRequests()

.antMatchers("/userlogin").hasRole("USER")

.antMatchers("/adminlogin").hasRole("ADMIN")

.and()

.csrf().disable().headers().frameOptions().disable();

}

}

* **SpringSecurityController.java**

**package** com.example.demo;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** SpringSecurityController {

@RequestMapping("/userlogin")

**public** String userValidation() {

**return** "User: Successfully logged in!";

}

@RequestMapping("/adminlogin")

**public** String adminValidation() {

**return** "Admin: Successfully logged in!";

}

}

* **SpringBootWithSecurityApplication.java**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootWithSecurityApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootWithSecurityApplication.**class**, args);

}

}

* **Spring Boot How to Change Default Context Path:**
* **Using application.properties**

server.servlet.context-path=/springbootrestxml

* **Using Java Code Changes**

**package** com.example.demo;

**import** org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;

**import** org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Server **implements** EmbeddedServletContainerCustomizer {

@Override

**public** **void** customize(ConfigurableEmbeddedServletContainer container)

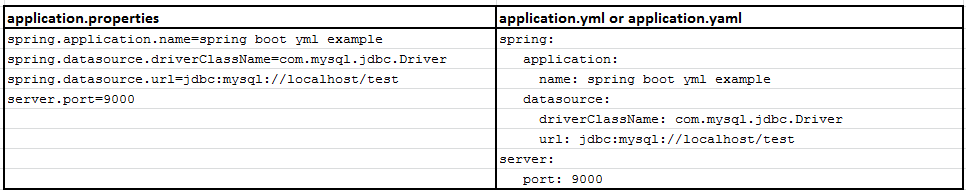
{

container.setContextPath("/springbootrestxml ");

}

}

* **Spring Boot yaml or yml(Yet Another Markup Language):**



* **Spring Boot Actuator:**

The actuator provides production-ready features for Spring Boot application. It will help us to check and manage our application in the production environment. We don’t need any code to get these features since they are available once the actuator dependency is in the class-path. The actuator provides features like auditing, health, metrics, environment information, thread dump etc. using HTTP endpoints.

<dependency>

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

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

</dependency>

**Available Endpoints:**

Here is the list of the popular endpoints provided by Spring Boot Actuator

| **ID** | **Description** | **Sensitive Default** |
| --- | --- | --- |
| actuator | Provides a hypermedia-based “discovery page” for the other endpoints. Requires Spring HATEOAS to be on the classpath. | true |
| auditevents | Exposes audit events information for the current application. | true |
| autoconfig | Displays an auto-configuration report showing all auto-configuration candidates and the reason why they ‘were’ or ‘were not’ applied. | true |
| beans | Displays a complete list of all the Spring beans in your application. | true |
| configprops | Displays a collated list of all @ConfigurationProperties. | true |
| dump | Performs a thread dump. | true |
| env | Exposes properties from Spring’s ConfigurableEnvironment. | true |
| flyway | Shows any Flyway database migrations that have beenapplied. | true |
| health | Shows application health information (when the application is secure, a simple ‘status’ when accessed over an unauthenticated connection or full message details when authenticated). | false |
| info | Displays arbitrary application info. | false |
| loggers | Shows and modifies the configuration of loggers in the application. | true |
| liquibase | Shows any Liquibase database migrations that have been applied. | true |
| metrics | Shows ‘metrics’ information for the current application. | true |
| mappings | Displays a collated list of all @RequestMapping paths. | true |
| shutdown | Allows the application to be gracefully shutdown (not enabled by default). | true |
| trace | Displays trace information (by default the last 100 HTTP requests). | true |

**Customizing endpoints:**

Endpoints can be customized using Spring properties. You can change if an endpoint is enabled, if it is considered sensitive and even its id.

For example, here is an application.properties that changes the sensitivity and id of the beans endpoint and also enables shutdown.

endpoints.beans.id=springbeans

endpoints.beans.sensitive=false

endpoints.shutdown.enabled=true

By default, all endpoints except for shutdown are enabled. If you prefer to specifically “opt-in” endpoint enablement you can use the endpoints.enabled property. For example, the following will disable all endpoints except for info:

endpoints.enabled=false

endpoints.info.enabled=true

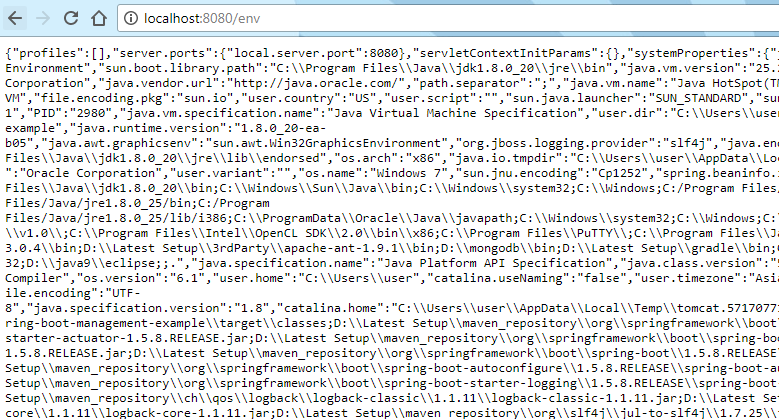
Likewise, you can also choose to globally set the “sensitive” flag of all endpoints. By default, the sensitive flag depends on the type of endpoint (see the table above). For example, to mark all endpoints as sensitive except info:

endpoints.sensitive=true

endpoints.info.sensitive=false

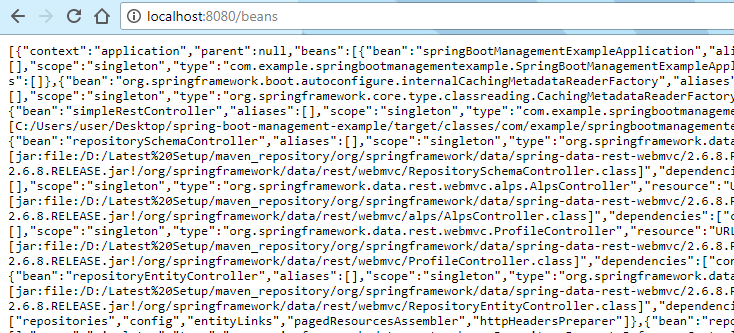
#### http://localhost:8080/env

This will give all the environmental configuration about the server.

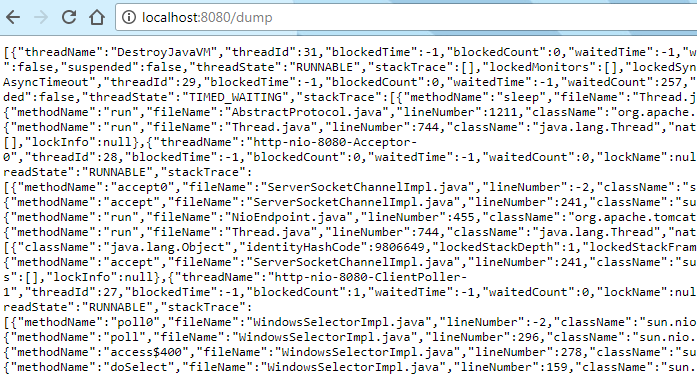


#### http://localhost:8080/beans

This will give all the spring beans loaded in the context.



#### <http://localhost:8080/dump>: This will give the current server thread dump.



#### http://localhost:8080/health

This will give generic health of the application and server.

#### Endpoint health Output

#### http://localhost:8080/metrics

|  |  |
| --- | --- |
| **{ "mem": 316656,**  **"mem.free": 169495,**  **"processors": 4,**  **"instance.uptime": 1449726,**  **"uptime": 1463662,**  **"systemload.average": -1.0,**  **"heap.committed": 263168,**  **"heap.init": 131072,**  **"heap.used": 93672,**  **"heap": 1846272,**  **"nonheap.committed": 54400,**  **}** Customize the management server port **To customize the management endpoint port, we need to add this entry in the applicapplication.properties file.**   |  | | --- | | **management.port=8081**   * **Spring Boot Jetty and Undertow server integration:**   **Jetty :**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **<exclusions>**  **<exclusion>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-tomcat</artifactId>**  **</exclusion>**  **</exclusions>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-jetty</artifactId>**  **</dependency>**  **</dependencies>**  **Undertow :**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **<exclusions>**  **<exclusion>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-tomcat</artifactId>**  **</exclusion>**  **</exclusions>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-undertow</artifactId>**  **</dependency>**  **</dependencies>** |  * **Spring Boot Gradle Integration :**      * **build.gradle**   **buildscript {**  **repositories {**  **mavenCentral()**  **}**  **dependencies {**  **classpath("org.springframework.boot:spring-boot-gradle-plugin:1.5.9.RELEASE")**  **// spring boot gradle plugin will used for generate jar**  **}**  **}**  **repositories {**  **mavenCentral()**  **}**  **description = "Spring boot Gradle Example"**  **apply plugin: 'java' // plugin for java**  **apply plugin: 'org.springframework.boot' // plugin for spring boot framework**  **jar {**  **baseName = 'spring-boot-gradle-example' // jar file name**  **version = '1.0.0' // application version**  **}**  **dependencies {**  **compile("org.springframework.boot:spring-boot-starter-web") // web application dependency**  **}**  **sourceCompatibility = 1.8 // for Java 1.8**  **targetCompatibility = 1.8**  **group 'spring-boot-demo'**  **version '1.0-SNAPSHOT'**   * **application.properties**   **server.port=8181**   * **DemoController.java**   **package com.example.demo;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **@RestController**  **public class DemoController {**  **@RequestMapping(value = "hello")**  **public String hello(){**  **return "Spring boot Gradle Example";**  **}**  **}**   * **SpringBootGradleApplication.java**   **package com.example.demo;**  **import org.springframework.boot.SpringApplication;**  **import org.springframework.boot.autoconfigure.SpringBootApplication;**  **@SpringBootApplication**  **public class SpringBootGradleApplication {**  **public static void main(String[] args) {**  **SpringApplication.*run*(SpringBootGradleApplication.class, args);**  **}**  **}**   * **Spring Boot Internationalization :**   Internationalization or Localization will show you the web page content based on your chosen  language in which y you want to view the page.    **Create language files in different languages :**  **i18n/messages\_bn.properties : Bengali**  **msg=Spring Boot MVC \u098F \u0986\u09A8\u09CD\u09A4\u09B0\u09CD\u099C\u09BE\u09A4\u09BF\u0995\u09BE\u09AF\u09BC\u09A8 \u0989\u09A6\u09BE\u09B9\u09B0\u09A3**  **welcome=\u09AC\u09BE\u0982\u09B2\u09BE \u09AD\u09BE\u09B7\u09BE \u09B8\u09CD\u09AC\u09BE\u0997\u09A4\u09AE**  **chooseLang=\u0986\u09AA\u09A8\u09BE\u09B0 \u09AD\u09BE\u09B7\u09BE \u099A\u09AF\u09BC\u09A8 \u0995\u09B0\u09C1\u09A8**  **copyright=\u0995\u09AA\u09BF\u09B0\u09BE\u0987\u099F**  **year=2018**  **i18n/messages\_hi.properties : Hindi**  **msg=Spring Boot MVC \u092E\u0947\u0902 \u0905\u0902\u0924\u0930\u094D\u0930\u093E\u0937\u094D\u091F\u094D\u0930\u0940\u092F\u0915\u0930\u0923 \u0909\u0926\u093E\u0939\u0930\u0923**  **welcome=\u0939\u093F\u0928\u094D\u0926\u0940 \u092D\u093E\u0937\u093E \u092E\u0947\u0902 \u0906\u092A\u0915\u093E \u0938\u094D\u0935\u093E\u0917\u0924 \u0939\u0948**  **chooseLang=\u0905\u092A\u0928\u0940 \u092D\u093E\u0937\u093E \u091A\u0941\u0928\u0947\u0902**  **copyright=\u0938\u0930\u094D\u0935\u093E\u0927\u093F\u0915\u093E\u0930**  **year=2018**  **i18n/messages\_fr.properties : French**  **msg=Internationalisation exemple dans Spring Boot MVC**  **welcome=Bienvenue à la langue française**  **chooseLang=Choisissez votre langue**  **copyright=Droit d'auteur**  **year=2018**  **i18n/messages\_nl.properties : Dutch**  **msg=Internationalisering Voorbeeld in Spring Boot MVC**  **welcome=Welkom bij Nederlandse Taal**  **chooseLang=Kies uw taal**  **copyright=Auteursrecht**  **year=2018**  **i18n/messages.properties : English**  **msg=Internationalization Example in Spring Boot MVC**  **welcome=Welcome to English Language**  **chooseLang=Choose your language**  **copyright=Copyright**  **year=2018**  **Creating Application config class :**  The below application configuration class defines Spring beans for loading messages in the chosen language and LocalResolver for handling locales. By default we set English as the default page language.  We have overridden the addInterceptor() method to pass a parameter called lang in the URL, which will give us the value of the current locale selected from the locale dropdown.  **package** com.example.demo;  **import** java.util.Locale;  **import** org.springframework.context.MessageSource;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.support.ReloadableResourceBundleMessageSource;  **import** org.springframework.web.servlet.LocaleResolver;  **import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;  **import** org.springframework.web.servlet.config.annotation.~~WebMvcConfigurerAdapter~~;  **import** org.springframework.web.servlet.i18n.CookieLocaleResolver;  **import** org.springframework.web.servlet.i18n.LocaleChangeInterceptor;  @Configuration  **public** **class** Internationalization **extends** ~~WebMvcConfigurerAdapter~~ {  @Bean  **public** MessageSource messageSource() {  ReloadableResourceBundleMessageSource messageSource = **new** ReloadableResourceBundleMessageSource();  messageSource.setBasename("classpath:i18n/messages");  messageSource.setDefaultEncoding("UTF-8");  **return** messageSource;  }  @Bean  **public** LocaleResolver localeResolver() {  CookieLocaleResolver lr = **new** CookieLocaleResolver();  lr.setDefaultLocale(Locale.***US***);  **return** lr;  }  @Override  **public** **void** addInterceptors(InterceptorRegistry registry) {  LocaleChangeInterceptor interceptor = **new** LocaleChangeInterceptor();  interceptor.setParamName("lang");  registry.addInterceptor(interceptor);  } |
| **Creating controller class :** |

The below controller class just maps the URL with the view and the corresponding view file’s output gets displayed on the browser. We have put the @Controller annotation on the class to make it Spring controller.

**package** com.example.demo;

**import** org.springframework.stereotype.Controller;

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

@Controller

**public** **class** HelloController {

@GetMapping("/")

**public** String hello() {

**return** "view";

}

}

**Creating main class :**

Create below main class to startup the Spring Boot application. We have specified the base package so that all classes which are annotated with spring annotation will automatically be detected by the Spring container.

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.boot.builder.SpringApplicationBuilder;

**import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

**public** **class** SpringBootMvcUsingJspApplication **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

**return** application.sources(SpringBootMvcUsingJspApplication.**class**);

}

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootMvcUsingJspApplication.**class**, args);

}

}

**Creating view file :**

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

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

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1">

<title><spring:message code="msg" /></title>

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script type="text/javascript">

$(document).ready(function() {

var selItem = localStorage.getItem("locales");

$('#locales').val(selItem ? selItem : 'en');

$("#locales").change(function() {

var selectedOption = $('#locales').val();

if (selectedOption) {

window.location.replace('?lang=' + selectedOption);

localStorage.setItem("locales", selectedOption);

}

});

});

</script>

<style>

legend { width: auto; margin-left: auto; margin-right: auto;}

</style>

</head>

<body><div style="width: 600px; margin: auto;">

<fieldset>

<legend>

<spring:message code="msg" />

</legend>

<p><label><spring:message code="chooseLang" /></label> <select

id="locales">

<option value="en">English</option>

<option value="bn">?????</option>

<option value="hi">?????</option>

<option value="fr">Français</option>

<option value="nl">Nederlands</option>

</select>

</p>

</fieldset>

<div style="clear: both"></div>

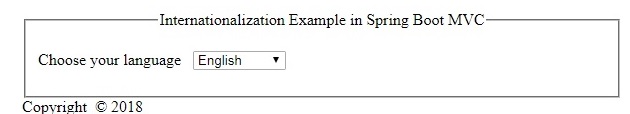
<div><spring:message code="copyright" />©<spring:message code="year" />

</div></div>

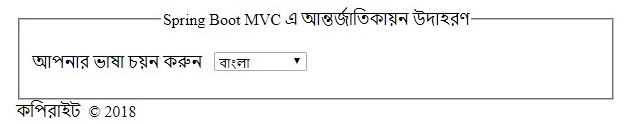
</body>

**Testing the application:**

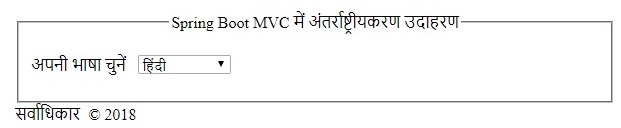
When you see the default language – English

[](https://www.jeejava.com/wp-content/uploads/2018/05/spring-boot-internationalization-english.jpg)

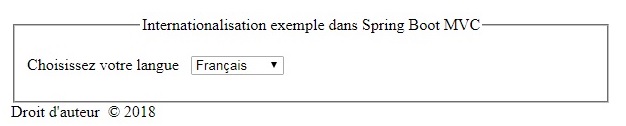
When you select the language – Bengali

[](https://www.jeejava.com/wp-content/uploads/2018/05/spring-boot-internationalization-bengali.jpg)

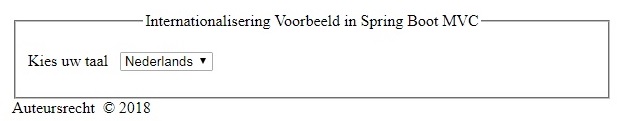
When you select the language – Hindi

[](https://www.jeejava.com/wp-content/uploads/2018/05/spring-boot-internationalization-hindi.jpg)

When you select the language – French



When you select the language – Dutch

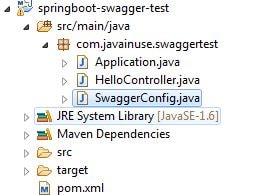


* **Spring Boot + Swagger 2:**

Documentation of REST Services we develop is very important. This documentation should help consumers of the service know which all services are available, the signatures, and the expected input. Also there should be some simple way to test if the service is up. The exposed services are bound to change so simultaneously the documentation would also need to be updated. If this is done manually then it will be a very tedious process, especially as the number of REST services increase. This is where swagger comes into picture. It helps automate this documentation process. Moreover, every change in the API should be simultaneously described in the reference documentation. Accomplishing this manually is a tedious exercise, so automation of the process was inevitable

# What is Swagger

Swagger is widely used for visualizing APIs, and with Swagger UI it provides online sandbox for frontend developers. For the tutorial, we will use the Springfox implementation of the Swagger 2 specification. Swagger is a tool, a specification and a complete framework implementation for producing the visual representation of RESTful Web Services. It enables documentation to be updated at the same pace as the server. When properly defined via Swagger, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Thus Swagger removes the guesswork in calling the service.



**pom.xml:**

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.4.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.4.0</version>

</dependency>

**SpringBootSwaggerApplication.java:**

**package** com.example.swagger.test;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootSwaggerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootSwaggerApplication.**class**, args);

}

}

**HelloController.java:**

**package** com.example.swagger.test;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** HelloController {

@RequestMapping(method = RequestMethod.***GET***, value = "/api/swaggertest")

**public** String sayHello() {

**return** "Swagger Hello World";

}

}

**SwaggerConfig.java:**

**package** com.example.swagger.test;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.Profile;

**import** com.google.common.base.Predicate;

**import** springfox.documentation.builders.ApiInfoBuilder;

**import** springfox.documentation.service.ApiInfo;

**import** springfox.documentation.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

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

**import** **static** com.google.common.base.Predicates.*or*;

@Profile("swagger-enabled-for-qa")

@Configuration

@EnableSwagger2

**public** **class** SwaggerConfig {

@Bean

**public** Docket postsApi() {

**return** **new** Docket(DocumentationType.***SWAGGER\_2***).groupName("public-api")

.apiInfo(apiInfo()).select().paths(postPaths()).build();

}

**private** Predicate<String> postPaths() {

**return** *or*(*regex*("/api/posts.\*"), *regex*("/api/swaggertest.\*"));

}

**private** ApiInfo apiInfo() {

**return** **new** ApiInfoBuilder().title("JavaInUse API")

.description("JavaInUse API reference for developers")

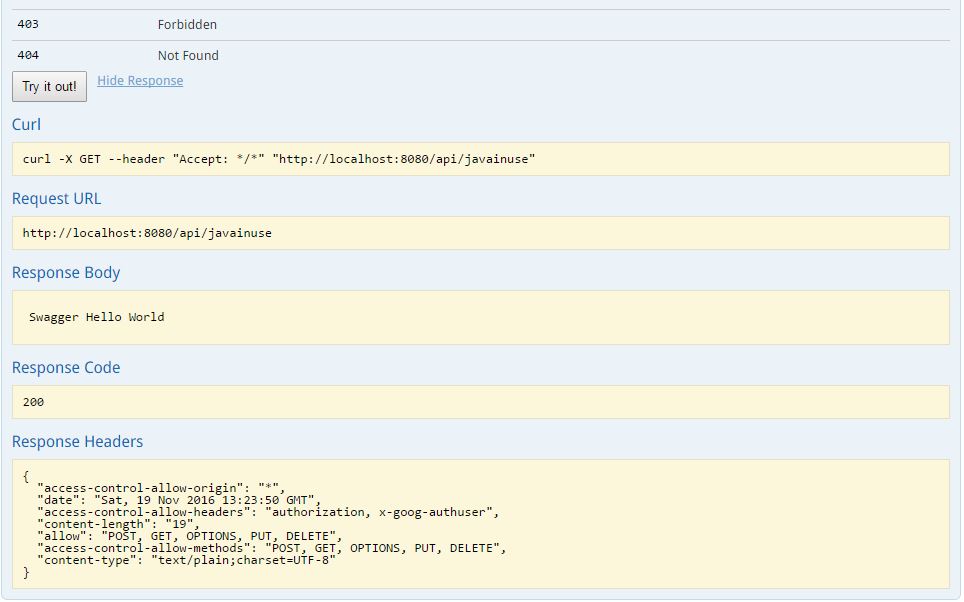
.termsOfServiceUrl("http://javainuse.com")

.~~contact~~("javainuse@gmail.com").license("JavaInUse License")

.licenseUrl("javainuse@gmail.com").version("1.0").build();

}

}

To enable the Swagger 2 we use the annotation @EnableSwagger2.  
A Docket bean is defined and using its select() method we get an instance of ApiSelectorBuilder. ApiSelectorBuilder we configure the endpoints exposed by Swagger.  
After the Docket bean is defined, its select() method returns an instance of ApiSelectorBuilder, which provides a way to control the endpoints exposed by Swagger.  
Using the RequestHandlerSelectors and PathSelectors we configure the predicates for selection of RequestHandlers  
  
Using the Try it button we can also check if the service is up.   
  
  
**Spring Boot + Swagger2- Understanding various Swagger annotations-**

@ApiOperation-

This annotation is used to describe the exposed REST API. It describes an operation or typically a HTTP method against a specific path. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The value of the annotation is a short description on the API. Since this is displayed in the list of operations in Swagger-UI and the location is limited in size, this should be kept short (preferably shorter than 120 characters) |
| notes | The notes allows you to give significantly more details about the operations (e.g. you can include request samples and responses here) |
| nickname | The nickname for this API. |

**@ApiOperation(value = "getGreeting", notes="get greeting",nickname = "getGreeting")**

@RequestMapping(method = RequestMethod.GET, value = "/api/javainuse")

public <Hello> sayHello() {

ArrayList<Hello> arrayList= new ArrayList<>();

arrayList.add(new Hello());

return arrayList;

}

@ApiResponses-

This annotation is used to describe the expected responses for the REST API. The @ApiResponse describes a concrete possible response. It cannot be used directly on the method and needs to be included in the array value of @ApiResponses (whether there's one response or more). It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| ApiResponse | The @ApiResponse describes a concrete possible response |

@ApiOperation(value = "getGreeting", nickname = "getGreeting")

**@ApiResponses(value = {**

**@ApiResponse(code = 500, message = "Server error"),**

**@ApiResponse(code = 404, message = "Service not found"),**

**@ApiResponse(code = 200, message = "Successful retrieval",**

**response = Hello.class, responseContainer = "List") })**

@RequestMapping(method = RequestMethod.GET, value = "/api/javainuse")

public <Hello> sayHello() {

ArrayList<Hello> arrayList= new ArrayList<>();

arrayList.add(new Hello());

return arrayList;

}

If the user has default response messages which are to be applied to all the REST APIs then these can be specified when defining the Docket bean. Hence these will not need to be applied at the method level. For example if the response for code 404 and 500 is going to be same through out all services.

@Bean

public Docket postsApi() {

Docket docket=new Docket(DocumentationType.SWAGGER\_2);

docket.groupName("public-api")

.apiInfo(apiInfo()).select().paths(postPaths()).build();

docket.globalResponseMessage(RequestMethod.GET, ImmutableList.of(new ResponseMessageBuilder()

.code(400)

.message("Bad Request")

.responseModel(new ModelRef("Error")).build(),new ResponseMessageBuilder()

.code(500)

.message("Internal Server Error")

.responseModel(new ModelRef("Error")).build()));

return docket;

@ApiParam-

This annotation is used to describe the exposed REST API. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The value is a short description of the parameter |
| required | If the parameter is optional or required. |
| defaultValue | Specify defaultValue of the parameter. |

@ApiOperation(value = "getGreeting", nickname = "getGreeting")

@ApiResponses(value = {

@ApiResponse(code = 500, message = "Server error"),

@ApiResponse(code = 200, message = "Successful retrieval",

response = Hello.class, responseContainer = "List") })

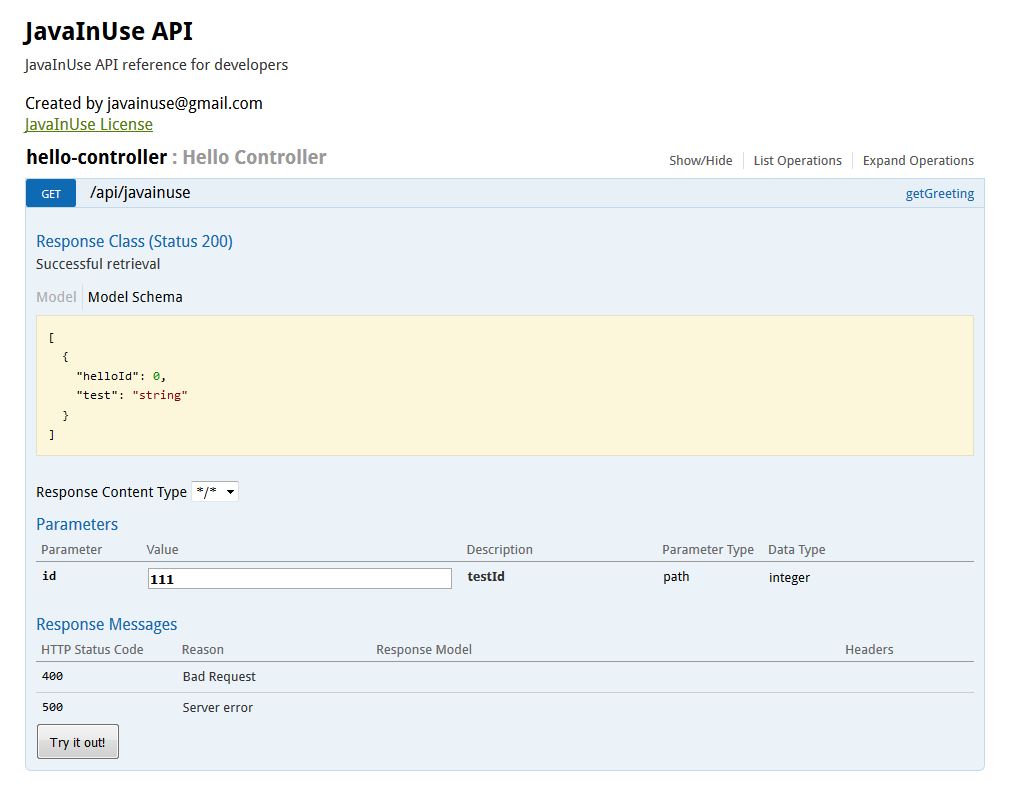
@RequestMapping(method = RequestMethod.GET, value = "/api/javainuse")

public List<Hello> sayHello(**@ApiParam(value = "testId",**

**required = true, defaultValue = "111")** @PathVariable(ID) final int institutuionId) {

return new Hello();

}

We get the Swagger UI as-  
  


@ApiModel-

The @ApiModel allows you to manipulate the meta data of a model from a simple description or name change to a definition of polymorphism. We have used it to create a response class Hello with default values. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| value | The name displayed for the Model class |
| description | The description of the model class |

@ApiModelProperty-

The @ApiModelProperty allows controlling Swagger-specific definitions such as allowed values, and additional notes. It also offers additional filtering properties in case you want to hide the property in certain scenarios. We use this parameter for specifying default values to the Response model class Hello. It takes the following parameters-

|  |  |
| --- | --- |
| **Annotation Parameter** | **Description** |
| position | The position of the field in the reponse class during display using swagger. |
| value | The value of the field when using Swagger. For example the default value of the path varaible will be 111 for class Hello. |
| required | If the field is optional or required. |

package com.javainuse.swaggertest;

import io.swagger.annotations.ApiModel;

import io.swagger.annotations.ApiModelProperty;

**@ApiModel**

public class Hello {

private int helloId;

private String test;

**@ApiModelProperty(position = 1, required = true, value = "1")**

public int getHelloId() {

return helloId;

}

public void setHelloId(int helloId) {

this.helloId = helloId;

}

**@ApiModelProperty(position = 2, required = true, value = "helloTest")**

public String getTest() {

return test;

}

public void setTest(String test) {

this.test = test;

}

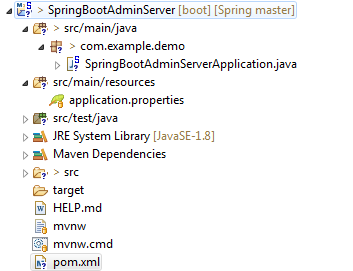
}

* **Spring Boot Admin :**

**Spring Boot admin** is a community project use to manage and monitor your [Spring Boot](https://www.javadevjournal.com/spring-boot/) applications. The client application gets register themselves with the admin server (via Http) or is discovered using Spring Cloud discover server like Eureka, Consul.

Each client application needs to have Spring Actuator jars in it. The endpoints provided by the Actuator jar is polled by the Spring Boot Admin server to get the metrics of that particular application.

**Creating Spring Boot Admin Server:**



**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootAdminServer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootAdminServer</name>

<description>Spring Boot Admin Server</description>

<properties>

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

<spring-boot-admin.version>2.1.4</spring-boot-admin.version>

</properties>

<dependencies>

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-starter-server</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-dependencies</artifactId>

<version>${spring-boot-admin.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Enable the admin server by adding **@EnableAdminServer** at your main class.

**SpringBootAdminServerApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

**import** de.codecentric.boot.admin.server.config.EnableAdminServer;

@SpringBootApplication

@EnableAdminServer

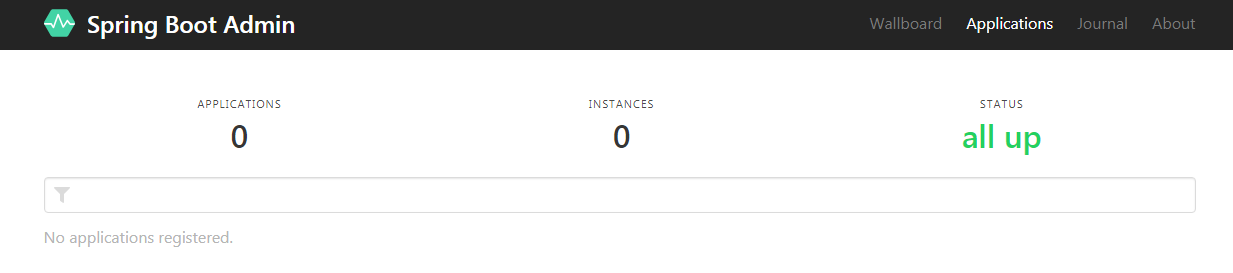
**public** **class** SpringBootAdminServerApplication {

**public** **static** **void** main(String[] args) {

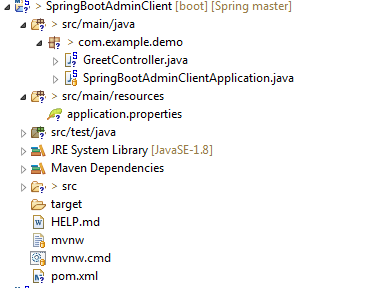
SpringApplication.*run*(SpringBootAdminServerApplication.**class**, args);

}

}



**Setting Up an Admin Client:**



**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootAdminClient</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootAdminClient</name>

<description>Spring Boot Admin Client</description>

<properties>

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

<spring-boot-admin.version>2.1.4</spring-boot-admin.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-starter-client</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-dependencies</artifactId>

<version>${spring-boot-admin.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**GreetController.java:**

**package** com.example.demo;

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

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** GreetController {

@GetMapping("/greet")

**public** String greet() {

**return** "Hi!! there...";

}

}

**SpringBootAdminClientApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootAdminClientApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootAdminClientApplication.**class**, args);

}

}

**application.properties:**

server.port=8060

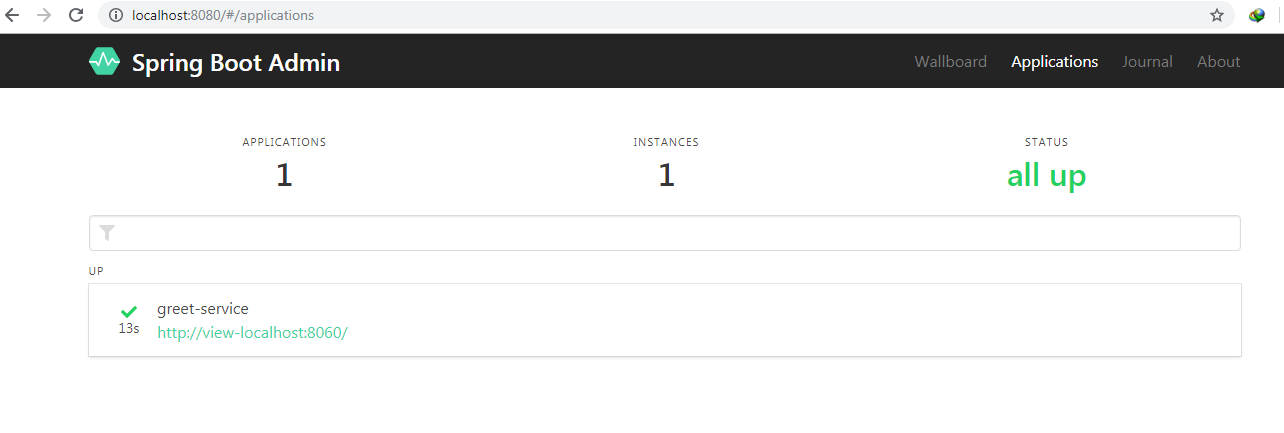
spring.application.name=greet-service

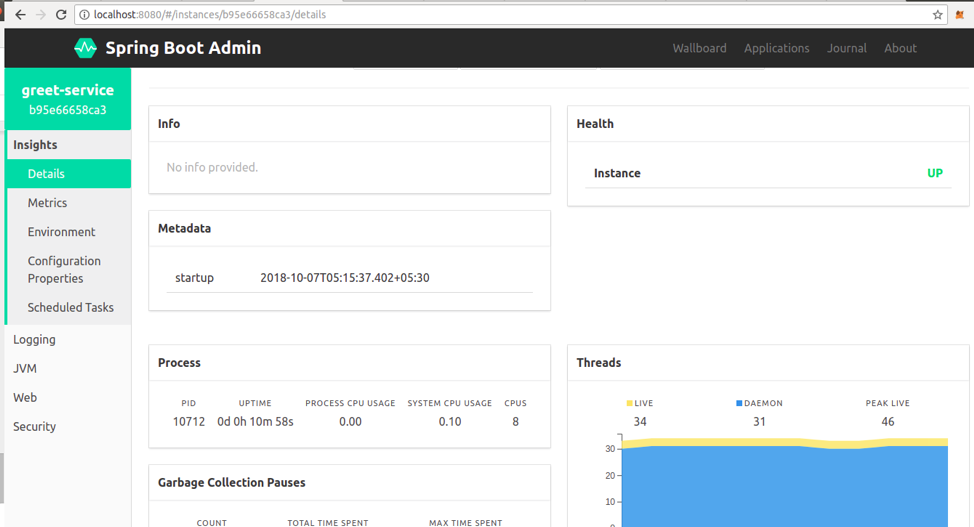
spring.boot.admin.client.url=http://localhost:8080

management.endpoints.web.exposure.include=\*

**spring.boot.admin.client.url** is a mandatory property which is the URL of the Spring Boot Admin Server to register at.**management.endpoints.web.exposure.include** is used to expose all the actuators endpoints.

Now boot the application up and visit the Admin server.





As you can see, you can get much info about your application using the UI.

* **Spring Boot Profiles :**

When we create an application, we need to deploy it in different kinds of environments, such as development, QA, staging and production.The application configuration in each of these environments will be different.One of the approaches to handling application configuration is to create something called a profile. Spring Boot has the concept of a profile built in.

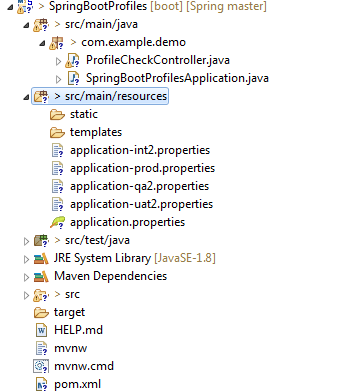
You can define default configuration in application.properties. Environment specific overrides can be configured in specific files:

* application-dev.properties
* application-qa.properties
* application-stage.properties
* application-prod.properties

Here are a couple of ways of setting the active profile:

* At the time of launching the Java application
  + -Dspring.profiles.active=qa - in the VM properties, OR
* Do the following in the application.properties file
  + spring.application.profiles=qa.

Depending on which profile is currently the active, the appropriate configuration is picked up.



**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootProfiles</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootProfiles</name>

<description>Spring Boot Profiles</description>

<properties>

<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-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=8081

**application-dev.properties:** server.port=5050

**application-qa.properties:** server.port=6060

**application-uat.properties:** server.port=7070

**application-prod.properties:** server.port=9090

**ProfileCheckController.java:**

**package** com.example.demo;

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

**import** org.springframework.core.env.Environment;

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

**import** org.springframework.web.bind.annotation.RestController;

**import** java.util.logging.Logger;

@RestController

**public** **class** ProfileCheckController {

**private** **static** **final** Logger ***logger*** = Logger.*getLogger*(ProfileCheckController.**class**.getName());

@Autowired

**private** Environment environment;

@GetMapping ("/checkProfile")

**public** String checkProfile() {

**return** "Spring Boot is running under " + environment.getActiveProfiles()[0] + " Profile";

}

}

**SpringBootProfilesApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootProfilesApplication {

**public** **static** **void** main(String[] args) {

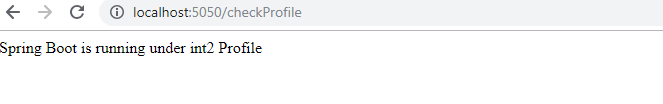
SpringApplication.*run*(SpringBootProfilesApplication.**class**, args);

}

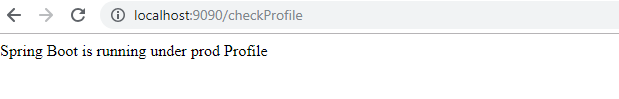
}



**Dev Profile:**



**Prod Profile:**



* **Spring Boot cloud eureka server:**

### Steps to configure Eureka server

1. Add **spring-cloud-starter-eureka-server** dependency in pom.xml
2. Enable eureka server using annotation in main class **:@EnableEurekaServer**
3. Add server port and default url in **application.properties or application.yml**
4. Build application and open dashboard



**application.yml :**

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

register-with-eureka: **false**

fetch-registry: **false**

serviceUrl:

defaultZone : http://${eureka.instance.hostname}:${server.port}/eureka/

server:

eviction-interval-timer-in-ms: 1000

**eureka.client.register-with-eureka :** Do not resister self as client

**eureka.client.serviceUrl.defaultZone :** Self Server URL

**eureka.server.eviction-interval-timer-in-ms :** Heart bit to check register services(client)

**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootAdminEurekaServer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootAdminEurekaServer</name>

<description>Spring Boot Admin Eureka Server</description>

<properties>

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

<spring-cloud.version>Greenwich.SR1</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**EurekaServerApplication.java:**

**package** com.javadeveloperzone;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

**import** org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan

@EnableEurekaServer // Indicate Eureka Server Application

**public** **class** EurekaServerApplication {

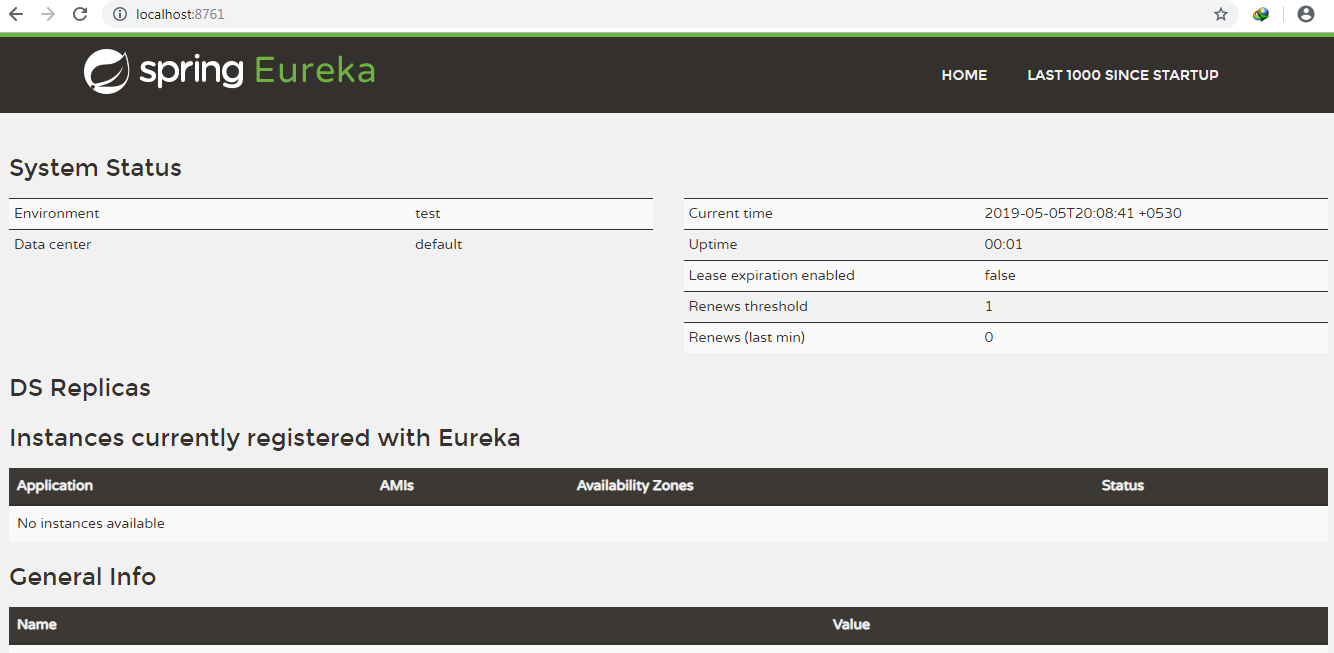
**public** **static** **void** main(String[] args) **throws** Exception {

SpringApplication.*run*(EurekaServerApplication.**class**, args); // it wil start application

}

}

Open : http://localhost:8761/

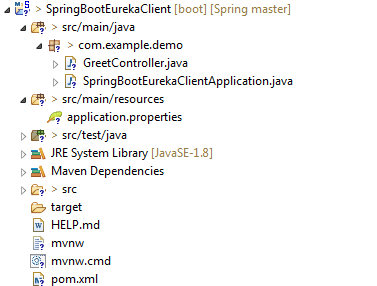


* **Spring Boot cloud eureka Client:**

We are assuming that Spring boot cloud eureka server us running on 8761 port.

Generally, Eureka client application is rest service which exposes REST services. Which will be accessible from direct UI or another Spring boot services.

* We can also register multiple instances of the same service to the server. In case of multiple instances of the same server requires load balancing.
* **@EnableEurekaClient** annotation is used to consider service as a eureka client
* **spring-cloud-starter-eureka** must be available in CLASSPATH
* **eureka.client.serviceUrl.defaultZone** configuration requires in application.properties or application.yml. Which indicate URL of eureka server in which client application will be registered.
* **spring.application.name** in application.properties  will be considered as a name of service which will access services.



**application.properties:**

server.port=8383

spring.application.name=greet-service

eureka.instance.hostname=localhost

eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:8761/eureka/

**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootEurekaClient</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootEurekaClient</name>

<description>Spring Boot Admin Eureka Cllient</description>

<properties>

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

<spring-cloud.version>Greenwich.SR1</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**GreetController.java:**

**package** com.example.demo;

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

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** GreetController {

@GetMapping("/greet")

**public** String greet() {

**return** "Hi!! there...";

}

}

**SpringBootEurekaClientApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

**import** org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@EnableEurekaClient

@ComponentScan

**public** **class** SpringBootEurekaClientApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootEurekaClientApplication.**class**, args);

}

}

Let’s check eureka serverhttp://localhost:8761/



We can see that Greet-service is registered.

* **Spring boot common dbcp2 connection pool**

**DBCP : Data Base Connection Pooling**

To communicate with the database requires the database connection and create the connection with the database is heavy operation. While performing each operation requires opening connection and after performing, database operation closes the connection but this process will take too much time for opening and closing connection every database operations. To solve this issue database connection pooling is used.

The connection pool can maintain multiple connections with the database when the demand of  Connection object at that time return object from the pool instead of creating a new connection every time.



**application.properties:**

spring.datasource.url=jdbc:mysql://localhost/spring-orm

spring.datasource.username=root

spring.datasource.password=

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

**Step 1: Remove tomcat-jdbc connection pool**

While working with spring-boot-starter-jdbc, Spring boot will default used a tomcat-jdbc connection pool. so first of all need to remove this dependency:

<dependency>

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

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

<exclusions>

<exclusion>

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

<artifactId>tomcat-jdbc</artifactId>

</exclusion>

</exclusions>

</dependency>

**Step 2: Add commons-dbcp2 dependency**

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-dbcp2</artifactId>

<version>2.2.0</version>

</dependency>

* DBCP 2 compiles and runs under Java 7 only (JDBC 4.1)
* DBCP 1.4 compiles and runs under Java 6 only (JDBC 4)
* DBCP 1.3 compiles and runs under Java 1.4-5 only (JDBC 3)

**All the classes are same as SpringBootJdbcSingleDataSource**

* **Spring Boot HikariCP**

[HikariCP](https://github.com/brettwooldridge/HikariCP) is very popular and known database connection pooling library, especially for performance and concurrency matters. Spring boot by default use tomcat connection pooling but we can configure HikariCP easily with spring boot.

**Step 1: Remove tomcat-jdbc connection pool**

While working with spring-boot-starter-jdbc, Spring boot will default used a tomcat-jdbc connection pool. so first of all need to remove this dependency:

<dependency>

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

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

<exclusions>

<exclusion>

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

<artifactId>tomcat-jdbc</artifactId>

</exclusion>

</exclusions>

</dependency>

**Step 2: Add HikariCP dependency**

<dependency>

<groupId> com.zaxxer </groupId>

<artifactId> HikariCP </artifactId>

<version>2.7.8</version>

</dependency>

**All the classes are same as SpringBootJdbcSingleDataSource**

* **Spring boot JPA call MySQL procedure**

DELIMITER //

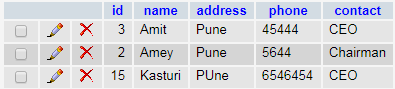
CREATE PROCEDURE getCustomerList()

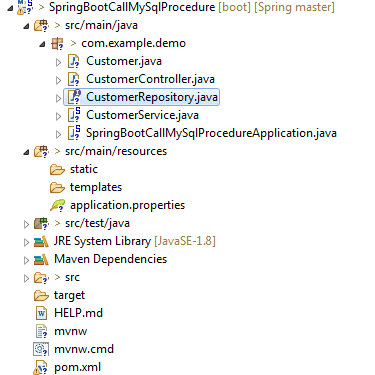
BEGIN

SELECT \* FROM customer;

END //

DELIMITER;





**application.properties:**

spring.datasource.url=jdbc:mysql://localhost/spring-orm

spring.datasource.username=root

spring.datasource.password=

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

**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>

<parent>

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

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

<version>2.1.4.RELEASE</version>

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

</parent>

<groupId>com.example</groupId>

<artifactId>SpringBootCallMySqlProcedure</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootCallMySqlProcedure</name>

<description>Spring Boot Call MySql Procedure</description>

<properties>

<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-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.zaxxer</groupId>

<artifactId>HikariCP</artifactId>

<version>2.7.8</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.6</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**customer.java:**

**package** com.example.demo;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

@Entity

**public** **class** Customer {

@Id

**private** **int** id;

@Column(name = "name")

**private** String custName;

@Column

**private** String address;

@Column

**private** String phone;

@Column(name = "contact")

**private** String contactPerson;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getCustName() {

**return** custName;

}

**public** **void** setCustName(String custName) {

**this**.custName = custName;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getContactPerson() {

**return** contactPerson;

}

**public** **void** setContactPerson(String contactPerson) {

**this**.contactPerson = contactPerson;

}

}

**CustomerRepository.java:**

**package** com.example.demo;

**import** java.util.List;

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

**import** org.springframework.data.jpa.repository.Query;

**import** org.springframework.transaction.annotation.Transactional;

@Transactional

**public** **interface** CustomerRepository **extends** JpaRepository<Customer,Integer> {

@Query(nativeQuery = **true**,value = "call getCustomerList") // call store procedure

List<Customer> getCustomerList();

}

**CustomerService.java:**

**package** com.example.demo;

**import** java.util.List;

**import** javax.annotation.Resource;

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

**import** org.springframework.stereotype.Service;

**import** com.example.demo.Customer;

**import** com.example.demo.CustomerRepository;

@Service

**public** **class** CustomerService {

@Autowired

@Resource

**private** CustomerRepository customerRepository;

**public** List<Customer> getCustomerList() {

**return** customerRepository.getCustomerList();

}

}

**CustomerController.java:**

**package** com.example.demo;

**import** java.util.List;

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

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** CustomerController {

@Autowired

**public** CustomerService customerService;

@RequestMapping("/getCustInfo")

**public** List<Customer> customerInformation() {

**return** customerService.getCustomerList();

}

}

**SpringBootCallMySqlProcedureApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootCallMySqlProcedureApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootCallMySqlProcedureApplication.**class**, args);

}

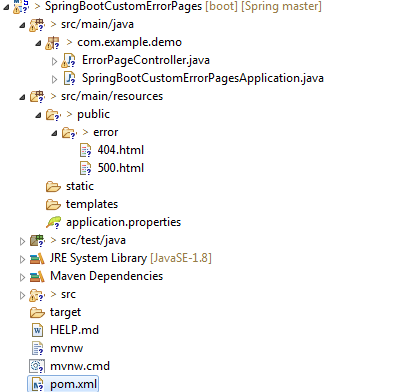
}



* **Spring boot Custom Error Pages:**

During web application development, Resource playing important roles so many times those resources may be moved to somewhere else or not available at the particular location at that time server will return default error page to the browser, handle those types of pages are too much important and display a proper message to the user.

For example, if 404 want to override then inside resources public/error/ create 404.html page, Name of page like STATUS\_CODE.html and remaining things will be handled by spring boot.



**pom.xml:**

<dependency>

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

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

</dependency>

**404.html :**

<html>

<head>

<title>Spring boot custom error page example</title>

</head>

<body>

<h1>This is Spring boot custom error page demo page - 404 status code</h1>

</body>

</html>

**500.html :**

<html>

<head>

<title>Spring boot custom error page example</title>

</head>

<body>

<h1>This is Spring boot custom error page demo page - 500 status code</h1>

</body>

</html>

**ErrorPageController.java:**

**package** com.example.demo;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** ErrorPageController {

@RequestMapping("/demo")

**public** String SpringBootHello() {

**int** i=5/0; // it will throws exception ArithmeticException: / by zero

**return** "demo";

}

}

**SpringBootCustomErrorPagesApplication.java:**

**package** com.example.demo;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** SpringBootCustomErrorPagesApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootCustomErrorPagesApplication.**class**, args);

}

}

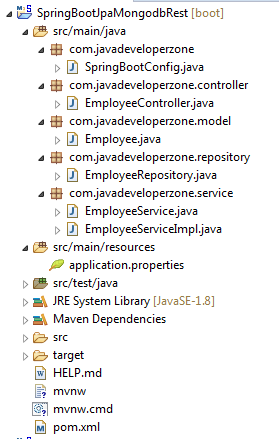
**Case 1) unknown.html:** This page does not exist so application will show custom 404.html



**Case 2) localhost:8080/demo :** There is an ArithmeticException so 500.html will be shown



* **Spring Boot MongoDb REST Integration:**



**NOTE:** Make sure that MongoDB server is running, By default it will run on 27017 port.

* **pom.xml:**

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

* **application.properties:**

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=demodb

* **SpringBootConfig.java:**

**package** com.javadeveloperzone;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@SpringBootApplication

@ComponentScan

@EnableMongoRepositories("com.javadeveloperzone.repository")

// Using a root package also allows the @ComponentScan annotation to be used without needing to specify a basePackage attribute

**public** **class** SpringBootConfig {

**public** **static** **void** main(String[] args) **throws** Exception {

SpringApplication.*run*(SpringBootConfig.**class**, args

}

}

**@EnableMongoRepositories**: It will enable MongoDB repository and we have provided based package here so it will be considered as a repository based package.

* **Employee.java:**

**package** com.javadeveloperzone.model;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "Employee")

**public** **class** Employee {

@Id // primary key

**private** String employeeId;

**private** String employeeName;

**private** String employeeRole;

**public** String getEmployeeId() {

**return** employeeId;

}

**public** **void** setEmployeeId(String employeeId) {

**this**.employeeId = employeeId;

}

**public** String getEmployeeRole() {

**return** employeeRole;

}

**public** **void** setEmployeeRole(String employeeRole) {

**this**.employeeRole = employeeRole;

}

**public** String getEmployeeName() {

**return** employeeName;

}

**public** **void** setEmployeeName(String employeeName) {

**this**.employeeName = employeeName;

}

}

**@Document** indicates the MongoDB document.

**@Id** indicate the primary key, It will not allow duplicate value. When the data type is String at that time it will automatically generate random Id.

* **EmployeeController.java:**

**package** com.javadeveloperzone.controller;

**import** com.javadeveloperzone.model.Employee;

**import** com.javadeveloperzone.service.EmployeeService;

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

**import** org.springframework.web.bind.annotation.\*;

@RestController // for rest response

**public** **class** EmployeeController {

@Autowired

**private** EmployeeService employeeService;

// to add new employee

@RequestMapping(value = "save",method = RequestMethod.***POST***)

**public** Employee save(Employee employee){

**return** employeeService.save(employee);

}

// to update employee

@RequestMapping(value = "update",method = RequestMethod.***POST***

**public** Employee update(Employee employee){

**return** employeeService.update(employee);

}

// list of all employee

@RequestMapping(value = "list",method = RequestMethod.***GET***)

**public** java.util.List<Employee> listEmployee() {

**return** employeeService.findAll();

}

// delete specific employee using employee id

@RequestMapping(value = "delete", method = RequestMethod.***DELETE***

**public** **void** delete(@RequestParam("id")String id){

employeeService.delete(id);

}

}

* **EmployeeRepository.java**

**package** com.javadeveloperzone.repository;

**import** com.javadeveloperzone.model.Employee;

**import** org.springframework.data.mongodb.repository.MongoRepository;

**import** org.springframework.data.mongodb.repository.Query;

**import** org.springframework.stereotype.Repository;

**import** java.util.List;

@Repository

**public** **interface** EmployeeRepository **extends** MongoRepository<Employee,String> {

@Override

List<Employee> findAll(); // find all Employee

@Query(value = "{ 'employeeName' : ?0 }")

List<Employee> findByEmployeeName(String name); // find employee by name

Employee findByEmployeeId(String id); // find

@Override

**void** delete(String s); // delete by ID

}

**@Repository** indicate repository which will communicate with MongoDB database and extands MongoRepository interface which contains basic methods to fetch data from MongoDB

**@Query** we can write a custom query which is supported by MongoDB

* **EmployeeService.java:**

package com.javadeveloperzone.service;

import com.javadeveloperzone.model.Employee;

import java.util.List;

public interface EmployeeService {

List<Employee> findAll();

Employee save(Employee employee);

void delete(String employeeId);

Employee update(Employee employee);

}

* **EmployeeServiceImpl.java:**

package com.javadeveloperzone.service;

import com.javadeveloperzone.repository.EmployeeRepository;

import com.javadeveloperzone.model.Employee;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List<Employee> findAll() {

return employeeRepository.findAll();

}

@Override

public void delete(String employeeId) {

employeeRepository.delete(employeeRepository.findOne(employeeId));

}

@Override

public Employee save(Employee employee) {

return employeeRepository.save(employee);

}

@Override

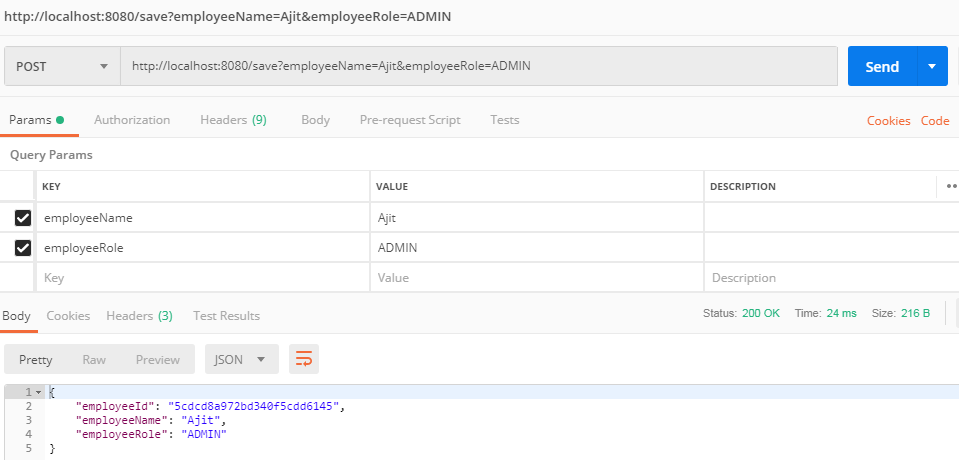
public Employee update(Employee employee) {

return employeeRepository.save(employee);

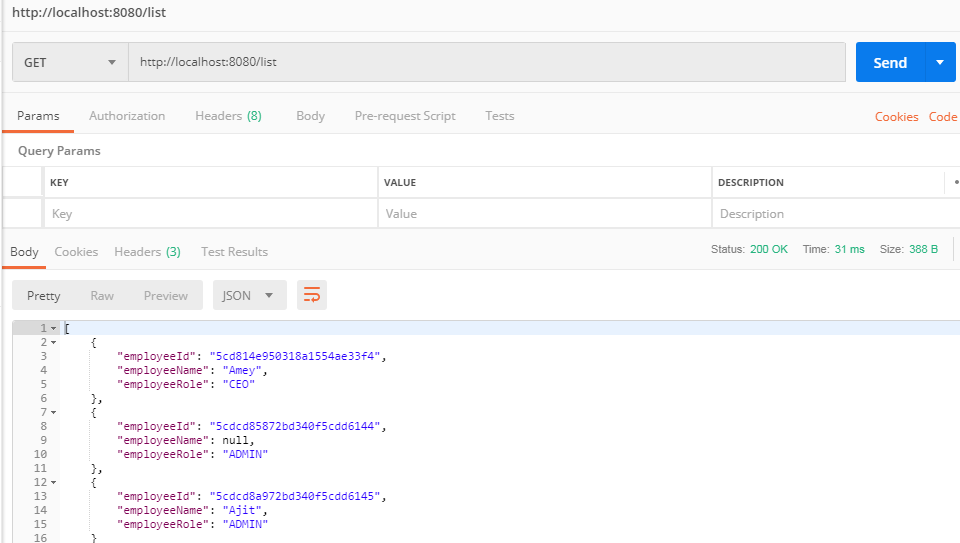
}

}

1. Add Employee :



1. List Of All Employees



* **Spring boot as windows service:**

**Step 1: Download Service Wapper**

Download winsw service wrapper from here. For example, we have download winsw-2.1.2-bin.exe

**Step 2: Rename Service Wapper**

Rename winsw-2.1.2-bin.exe to my-spring-boot-service.exe

**Step 3: Create Configuration XML file**

Create my-spring-boot-service.xml in the same location where my-spring-boot-service.exe is available.

**NOTE:** Make sure that .exe and .xml file name must be same, so our .exe file name is my-spring-boot-service.exe and .xml is my-spring-boot-service.xml

##### **my-spring-boot-service.xml**

* Here are some configuration related to services like service name, service id and description.[Here are more options about configuration XML file.](https://github.com/kohsuke/winsw/blob/master/doc/xmlConfigFile.md)
* Executable indicates the location of java.exe file which will be responsible to execute .jar file.  Write java.exe‘s **full path** otherwise it service will throw an error like: Error 1067: The process terminated unexpectedly.
* Arguments where is our actual command to execute .jar, Our jar name is demo.jar
* We can also change user context in which service will be run, [Here is a document to change user context](https://github.com/kohsuke/winsw/blob/master/doc/xmlConfigFile.md#service-account)

**<service>**

**<id>**my-spring-boot-service**</id>**

**<name>**my-spring-boot-service**</name>**

**<description>**Java Developer Zone - Spring boot as windows service example**</description>**

**<executable>**C:\Program Files\Java\jre1.8.0\_51\bin\java**</executable>**

**<arguments>**-jar "%BASE%\demo.jar"**</arguments>**

**<logmode>**rotate**</logmode>**

**</service>**

**Step 4: Copy Jar File**

* Copy jar file at same directory where my-spring-boot-service.exe and my-spring-boot-service.xml files are available.

**Step 5: Install Service**

* Now open terminal, Go to the directory where our files are available and execute below command:
* my-spring-boot-service.exe install

2018-05-16 20:42:51,066 INFO - Installing the service with id 'my-spring-boot-service'

**Step 6: Start & Stop Service**

* > my-spring-boot-service.exe start

2018-05-16 20:52:50,395 INFO - Starting the service with id 'my-spring-boot-service'

* > my-spring-boot-service.exe stop

2018-05-16 20:54:29,264 INFO - Stopping the service with id 'my-spring-boot-service'

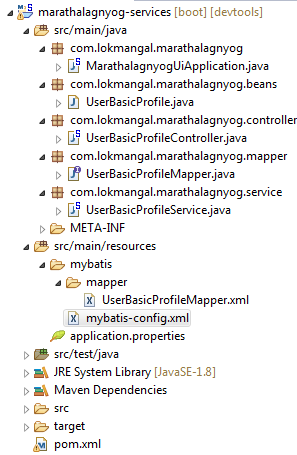
##### **Step 7: Uninstall Service**

* > my-spring-boot-service.exe uninstall

2018-05-16 20:50:37,897 INFO - Uninstalling the service with id 'my-spring-boot- service'

2018-05-16 20:50:37,906 WARN - The service with id 'my-spring-boot-service' is running. It may be impossible to uninstall it

* **Spring Boot Angular 7 Integration**
* **Spring Boot REST API with MyBatis : Backend**



**application.properties:**

mybatis.config-locations=classpath:mybatis/mybatis-config.xml

mybatis.mapper-locations=classpath:mybatis/mapper/\*.xml

mybatis.type-aliases-package=com.lokmangal.marathalagnyog.beans

spring.datasource.driverClassName = com.mysql.jdbc.Driver

spring.datasource.url = jdbc:mysql://localhost:3306/marathalagnyog?autoReconnect=true&useSSL=false

spring.datasource.username = root

spring.datasource.password =

server.port=8085

**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.neo</groupId>

<artifactId>MarathaLagnYog</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>MarathaLagnYog</name>

<description>MarathaLagnYog</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.1.1</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.6</version>

</dependency>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

<excludes>

<exclude>bootstrap-test.properties</exclude>

<exclude>bootstrap-dev.properties</exclude>

<exclude>bootstrap-pro.properties</exclude>

<exclude>bootstrap.properties</exclude>

</excludes>

</resource>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

<includes>

<include>bootstrap-${env}.properties</include>

<include>bootstrap.properties</include>

</includes>

</resource>

</resources>

<plugins>

<plugin>

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

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

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

</project>

**UserBasicProfileMapper.xml:**

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace=*"com.lokmangal.marathalagnyog.mapper.UserBasicProfileMapper"* >

<resultMap id=*"UserBasicProfileResultMap"* type=*"com.lokmangal.marathalagnyog.beans.UserBasicProfile"* >

<id column=*"profile\_id"* property=*"profileId"* jdbcType=*"BIGINT"* />

<result column=*"first\_name"* property=*"firstName"* jdbcType=*"VARCHAR"* />

<result column=*"last\_name"* property=*"lastName"* jdbcType=*"VARCHAR"* />

<result column=*"gender"* property=*"gender"* jdbcType=*"VARCHAR"* />

<result column=*"birth\_date"* property=*"birthDate"* jdbcType=*"VARCHAR"* />

<result column=*"marital\_status"* property=*"maritalStatus"* jdbcType=*"VARCHAR"* />

<result column=*"religion"* property=*"religion"* jdbcType=*"VARCHAR"* />

<result column=*"caste"* property=*"caste"* jdbcType=*"VARCHAR"* />

<result column=*"sub\_caste"* property=*"subCaste"* jdbcType=*"VARCHAR"* />

<result column=*"password"* property=*"password"* jdbcType=*"VARCHAR"* />

<result column=*"status"* property=*"status"* jdbcType=*"VARCHAR"* />

<result column=*"album\_status"* property=*"albumStatus"* jdbcType=*"VARCHAR"* />

</resultMap>

<sql id=*"Base\_Column\_List"* >

profile\_id, first\_name, last\_name, gender, birth\_date, marital\_status, religion, caste, sub\_caste, password, status, album\_status

</sql>

<select id=*"getUserBasicProfiles"* resultMap=*"UserBasicProfileResultMap"* >

SELECT

<include refid=*"Base\_Column\_List"* />

FROM tbl\_basic\_profile

</select>

<select id=*"getUserBasicProfileByProfileId"* parameterType=*"java.lang.Long"* resultMap=*"UserBasicProfileResultMap"* >

SELECT

<include refid=*"Base\_Column\_List"* />

FROM tbl\_basic\_profile

WHERE profile\_id = #{profileId}

</select>

<insert id=*"insertUserBasicProfile"* parameterType=*"com.lokmangal.marathalagnyog.beans.UserBasicProfile"* >

INSERT INTO

tbl\_basic\_profile

(first\_name, last\_name, gender,birth\_date, marital\_status, religion, caste, sub\_caste, password, status, album\_status)

VALUES

(#{firstName}, #{lastName}, #{gender}, #{birthDate}, #{maritalStatus}, #{religion}, #{caste}, #{subCaste}, #{password}, #{status}, #{albumStatus})

</insert>

<update id=*"updateUserBasicProfile"* parameterType=*"com.lokmangal.marathalagnyog.beans.UserBasicProfile"* >

UPDATE

tbl\_basic\_profile

SET

first\_name = #{firstName},

last\_name = #{lastName},

gender = #{gender},

birth\_date = #{birthDate},

marital\_status = #{maritalStatus},

religion= #{religion},

caste = #{caste},

sub\_caste = #{subCaste},

password = #{password},

status = #{status},

album\_status = #{albumStatus}

WHERE

profile\_id = #{profileId}

</update>

<delete parameterType=*"java.lang.Long"* id=*"deleteUserBasicProfile"*>

DELETE FROM

tbl\_basic\_profile

WHERE

profile\_id =#{id}

</delete>

</mapper>

**UserProfileMapper.xml:**

**package** com.lokmangal.marathalagnyog.beans;

**import** java.io.Serializable;

**import** java.sql.Date;

**public** **class** UserBasicProfile **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Long profileId;

**private** String firstName;

**private** String lastName;

**private** String gender;

**private** Date birthDate;

**private** String maritalStatus;

**private** String religion;

**private** String caste;

**private** String subCaste;

**private** String password;

**private** String status;

**private** String albumStatus;

/\*\*-

\* **@return** the profileId

\*/

**public** Long getProfileId() {

**return** profileId;

}

/\*\*

\* **@param** profileId the profileId to set

\*/

**public** **void** setProfileId(Long profileId) {

**this**.profileId = profileId;

}

/\*\*

\* **@return** the firstName

\*/

**public** String getFirstName() {

**return** firstName;

}

/\*\*

\* **@param** firstName the firstName to set

\*/

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

/\*\*

\* **@return** the lastName

\*/

**public** String getLastName() {

**return** lastName;

}

/\*\*

\* **@param** lastName the lastName to set

\*/

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

/\*\*

\* **@return** the gender

\*/

**public** String getGender() {

**return** gender;

}

/\*\*

\* **@param** gender the gender to set

\*/

**public** **void** setGender(String gender) {

**this**.gender = gender;

}

/\*\*

\* **@return** the birthDate

\*/

**public** Date getBirthDate() {

**return** birthDate;

}

/\*\*

\* **@param** birthDate the birthDate to set

\*/

**public** **void** setBirthDate(Date birthDate) {

**this**.birthDate = birthDate;

}

/\*\*

\* **@return** the maritalStatus

\*/

**public** String getMaritalStatus() {

**return** maritalStatus;

}

/\*\*

\* **@param** maritalStatus the maritalStatus to set

\*/

**public** **void** setMaritalStatus(String maritalStatus) {

**this**.maritalStatus = maritalStatus;

}

/\*\*

\* **@return** the religion

\*/

**public** String getReligion() {

**return** religion;

}

/\*\*

\* **@param** religion the religion to set

\*/

**public** **void** setReligion(String religion) {

**this**.religion = religion;

}

/\*\*

\* **@return** the caste

\*/

**public** String getCaste() {

**return** caste;

}

/\*\*

\* **@param** caste the caste to set

\*/

**public** **void** setCaste(String caste) {

**this**.caste = caste;

}

/\*\*

\* **@return** the subCaste

\*/

**public** String getSubCaste() {

**return** subCaste;

}

/\*\*

\* **@param** subCaste the subCaste to set

\*/

**public** **void** setSubCaste(String subCaste) {

**this**.subCaste = subCaste;

}

/\*\*

\* **@return** the password

\*/

**public** String getPassword() {

**return** password;

}

/\*\*

\* **@param** password the password to set

\*/

**public** **void** setPassword(String password) {

**this**.password = password;

}

/\*\*

\* **@return** the status

\*/

**public** String getStatus() {

**return** status;

}

/\*\*

\* **@param** status the status to set

\*/

**public** **void** setStatus(String status) {

**this**.status = status;

}

/\*\*

\* **@return** the albumStatus

\*/

**public** String getAlbumStatus() {

**return** albumStatus;

}

/\*\*

\* **@param** albumStatus the albumStatus to set

\*/

**public** **void** setAlbumStatus(String albumStatus) {

**this**.albumStatus = albumStatus;

}

}

**UserProfileMapper.java**

**package** com.lokmangal.marathalagnyog.mapper;

**import** java.util.List;

**import** com.lokmangal.marathalagnyog.beans.UserBasicProfile;

**public** **interface** UserBasicProfileMapper {

**public** List<UserBasicProfile> getUserBasicProfiles();

**public** UserBasicProfile getUserBasicProfileByProfileId(Long id);

**public** **void** insertUserBasicProfile(UserBasicProfile user);

**public** **void** updateUserBasicProfile(UserBasicProfile user);

**public** **void** deleteUserBasicProfile(Long id);

}

**UserServiceProfile.java:**

**package** com.lokmangal.marathalagnyog.service;

**import** java.util.List;

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

**import** org.springframework.stereotype.Component;

**import** com.lokmangal.marathalagnyog.beans.UserBasicProfile;

**import** com.lokmangal.marathalagnyog.mapper.UserBasicProfileMapper;

@Component

**public** **class** UserBasicProfileService {

@Autowired

**private** UserBasicProfileMapper userBasicProfileMapper;

**public** List<UserBasicProfile> getUserBasicProfiles() {

**return** userBasicProfileMapper.getUserBasicProfiles();

}

**public** UserBasicProfile getUserBasicProfileByProfileId(Long id) {

**return** userBasicProfileMapper.getUserBasicProfileByProfileId(id);

}

**public** **void** insertUserBasicProfile(UserBasicProfile userBasicProfile) {

userBasicProfileMapper.insertUserBasicProfile(userBasicProfile);

}

**public** **void** updateUserBasicProfile(UserBasicProfile userBasicProfile) {

userBasicProfileMapper.updateUserBasicProfile(userBasicProfile);

}

**public** **void** deleteUserBasicProfile(Long id) {

userBasicProfileMapper.deleteUserBasicProfile(id);

}

}

**UserBasicProfileController.java**

**package** com.lokmangal.marathalagnyog.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.lokmangal.marathalagnyog.beans.UserBasicProfile;

**import** com.lokmangal.marathalagnyog.service.UserBasicProfileService;

@RestController

@RequestMapping("marathalagnyog")

**public** **class** UserBasicProfileController {

@Autowired

**private** UserBasicProfileService userBasicProfileService;

@GetMapping(value = "/getUserBasicProfiles")

**public** List<UserBasicProfile> getUserBasicProfiles() {

**return** userBasicProfileService.getUserBasicProfiles();

}

@GetMapping(value = "/getUserBasicProfile/{id}")

**public** UserBasicProfile getUserBasicProfileByProfileId(@PathVariable("id") Long id) {

**return** userBasicProfileService.getUserBasicProfileByProfileId(id);

}

@PostMapping(value = "/addUserBasicProfile")

**public** **void** save(@RequestBody UserBasicProfile userBasicProfile) {

userBasicProfileService.insertUserBasicProfile(userBasicProfile);

}

@PutMapping(value = "/updateUserBasicProfile/{id}")

**public** **void** update(@PathVariable("id") Long id, @RequestBody UserBasicProfile userBasicProfile) {

UserBasicProfile userBasicProfileExists = getOne(id);

**if** (userBasicProfileExists != **null**)

userBasicProfileService.updateUserBasicProfile(userBasicProfile);

}

@DeleteMapping(value = "/deleteUserBasicProfileByProfileId/{id}")

**public** **void** deleteUserBasicProfile(@PathVariable("id") Long id) {

userBasicProfileService.deleteUserBasicProfile(id);

}

**public** UserBasicProfile getOne(Long id) {

**return** userBasicProfileService.getUserBasicProfileByProfileId(id);

}

}

**MarathalagnyogUiApplication.java**

**package** com.lokmangal.marathalagnyog;

**import** org.mybatis.spring.annotation.MapperScan;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

@MapperScan("com.lokmangal.marathalagnyog.mapper")

**public** **class** MarathalagnyogUiApplication {

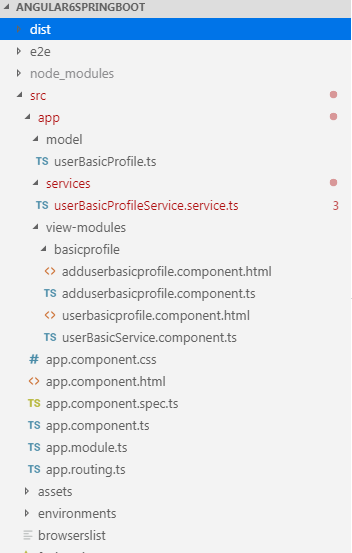
**public** **static** **void** main(String[] args) {

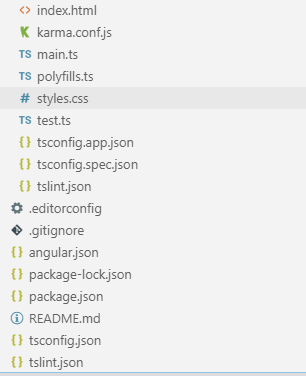
SpringApplication.*run*(MarathalagnyogUiApplication.**class**, args);

}

}

* **Angular 7 :Frontend**





**Package.json:**

{

"name": "MarathaLagnyog",

"version": "0.0.0",

"scripts": {

"ng": "ng",

"start": "ng serve",

"build": "ng build",

"test": "ng test",

"lint": "ng lint",

"e2e": "ng e2e"

},

"private": true,

"dependencies": {

"@angular/animations": "^7.2.8",

"@angular/common": "^7.2.8",

"@angular/compiler": "^7.2.8",

"@angular/core": "^7.2.8",

"@angular/forms": "^7.2.8",

"@angular/http": "^7.2.8",

"@angular/platform-browser": "^7.2.8",

"@angular/platform-browser-dynamic": "^7.2.8",

"@angular/router": "^7.2.8",

"bootstrap": "^4.3.1",

"core-js": "^2.5.4",

"font-awesome": "^4.7.0",

"jquery": "^3.4.1",

"primeng": "^7.0.4",

"rxjs": "^6.0.0",

"zone.js": "^0.8.26"

},

"devDependencies": {

"@angular/compiler-cli": "^6.0.3",

"@angular-devkit/build-angular": "~0.6.8",

"typescript": "~2.7.2",

"@angular/cli": "~7.3.9",

"@angular/language-service": "^6.0.3",

"@types/jasmine": "~2.8.6",

"@types/jasminewd2": "~2.0.3",

"@types/node": "~8.9.4",

"codelyzer": "~4.2.1",

"jasmine-core": "~2.99.1",

"jasmine-spec-reporter": "~4.2.1",

"karma": "~1.7.1",

"karma-chrome-launcher": "~2.2.0",

"karma-coverage-istanbul-reporter": "~2.0.0",

"karma-jasmine": "~1.1.1",

"karma-jasmine-html-reporter": "^0.2.2",

"protractor": "~5.3.0",

"ts-node": "~5.0.1",

"tslint": "~5.9.1"

}

}

**app.module.ts:**

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { FormsModule, ReactiveFormsModule} from '@angular/forms';

import { ButtonModule } from 'primeng/components/button/button';

import { TableModule } from 'primeng/table';

import { AppComponent } from './app.component';

import { HttpClientModule } from '@angular/common/http';

import { UserBasicProfileService } from 'src/app/services/userBasicProfileService.service'

import { UserBasicProfileComponent } from 'src/app/view-modules/basicprofile/userBasicService.component';

import { AddUserBasicProfileComponent } from 'src/app/view-modules/basicprofile/adduserbasicprofile.component';

import { RouterModule, Routes } from '@angular/router';

import { appRoutes } from './app.routing';

@NgModule({

declarations: [

UserBasicProfileComponent,

AddUserBasicProfileComponent,

AppComponent

],

imports: [

BrowserModule,

FormsModule,

ReactiveFormsModule,

HttpClientModule,

ButtonModule,

TableModule,

RouterModule.forRoot(appRoutes),

],

providers: [UserBasicProfileService],

bootstrap: [AppComponent]

})

export class AppModule { }

**app.routing.ts :**

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { UserBasicProfileComponent } from 'src/app/view-modules/basicprofile/userBasicService.component';

import { AddUserBasicProfileComponent } from 'src/app/view-modules/basicprofile/adduserbasicprofile.component';

export const appRoutes: Routes = [

{ path: 'users', component: UserBasicProfileComponent },

{ path: 'add-user', component: AddUserBasicProfileComponent }

];

**app.component.ts:**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title : string = 'Angular Spring Boot REST Application';

count: number = 0;

clickEvent() {

this.count++;

}

}

**app.component.html:**

<div style="text-align:center">

<h1>

{{title}}

</h1>

<a routerLink="/users"> List Users</a>

<a style="margin-left:10px" routerLink="/add-user">Add User</a>

<br/>

<router-outlet></router-outlet>

</div>

**UserBasicProfile.ts:**

export class UserBasicProfile {

profileId: number;

firstName: string;

lastName: string;

gender: string;;

birthDate: string;

maritalStatus: string;

religion: string;

caste: string;

subCaste: string;

password: string;

status: string;

albumStatus: string;

}

**UserBasicProfileService.ts:**

import { Injectable } from '@angular/core';

import { HttpClient, HttpParams } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable()

export class UserBasicProfileService {

private baseUrl = 'http://localhost:8085/marathalagnyog';

constructor(private http: HttpClient) { }

getUserBasicProfileList(): Observable<any> {

return this.http.get(this.baseUrl+ '/getUserBasicProfiles');

}

getUserBasicProfileByProfileId(profileId): Observable<any> {

return this.http.get(this.baseUrl+ '/getUserBasicProfile/'+ profileId);

}

}

**UserBasicProileComponent.ts:**

import { Component, OnInit, ViewEncapsulation } from '@angular/core';

import { Observable } from 'rxjs';

import { UserBasicProfileService } from 'src/app/services/userBasicProfileService.service';

import { UserBasicProfile } from 'src/app/model/userBasicProfile';

@Component({

selector: 'app-userBasicProfile',

templateUrl: './userbasicprofile.component.html',

encapsulation: ViewEncapsulation.None

})

export class UserBasicProfileComponent implements OnInit {

profileId: number = 1109;

firstName : string = '';

lastName : string = '';

cols: any[];

userBasicProfileData : UserBasicProfile[] = [];

constructor(private userBasicProfileService: UserBasicProfileService) {

}

ngOnInit() {

this.getUserBasicProfileList();

this.cols = [

{ field: 'profileId', header: 'Profile Id' },

{ field: 'firstName', header: 'First Name' },

{ field: 'lastName', header: 'Last Name' },

{ field: 'gender', header: 'Gender' },

{ field: 'birthDate', header: 'Date of Birth' },

{ field: 'maritalStatus', header: 'Marital Status' },

{ field: 'religion', header: 'Religion' },

{ field: 'caste', header: 'Caste' },

{ field: 'subCaste', header: 'Sub Caste' },

{ field: 'password', header: 'Password' },

{ field: 'status', header: 'Status' },

{ field: 'albumStatus', header: 'Album Status' },

];

}

getUserBasicProfileList() {

this.userBasicProfileService.getUserBasicProfileList().subscribe(

result => {

result.forEach(record => {

this.profileId = record["profileId"];

this.firstName = record["firstName"];

this.lastName = record["lastName"];

this.userBasicProfileData.push({ 'profileId' : record["profileId"],

'firstName':record["firstName"],

'lastName':record["lastName"],

'gender':record["gender"],

'birthDate':record["birthDate"],

'maritalStatus':record["maritalStatus"],

'religion':record["religion"],

'caste':record["caste"],

'subCaste':record["subCaste"],

'password':record["password"],

'status':record["status"],

'albumStatus':record["albumStatus"],

});

});

},

error => {

console.error('error in userBasicProfileService.getUserBasicProfileList');

}

);

}

getUserBasicProfileByProfileId(profileId) {

this.userBasicProfileService.getUserBasicProfileByProfileId(this.profileId).subscribe(

result => {

console.log(result["profileId"]);

console.log(result["firstName"]);

},

error => {

console.error('error in userBasicProfileService.getUserBasicProfileByProfileId');

}

);

}

}

**UserBasicProfile.component.html:**

<p-table [columns]="cols" [value]="userBasicProfileData" sortMode="multiple" #dt [paginator]="true" [rows]="2">

<ng-template pTemplate="caption">

Users List

</ng-template>

<ng-template pTemplate="header" let-columns>

<tr>

<th \*ngFor="let col of columns" [pSortableColumn]="col.field">

{{col.header}}

<p-sortIcon [field]="col.field"></p-sortIcon>

</th>

</tr>

<tr>

<th \*ngFor="let col of columns" [ngSwitch]="col.field">

<input pInputText type="text" (input)="dt.filter($event.target.value, col.field, col.filterMatchMode)">

</th>

</tr>

</ng-template>

<ng-template pTemplate="body" let-user let-columns="columns">

<tr>

<td \*ngFor="let col of columns">

{{user[col.field]}}

</td>

</tr>

</ng-template>

</p-table>

* **Spring boot Custom Banners:**

By default, Spring Boot application will display the following banner on startup

**. \_\_\_\_ \_ \_\_ \_ \_**

**/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \**

**( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \**

**\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )**

**' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /**

**=========|\_|==============|\_\_\_/=/\_/\_/\_/**

**:: Spring Boot :: (v1.5.7.RELEASE)<**

**Creating Custom Banner:**

**Use** <https://devops.datenkollektiv.de/banner.txt/index.html> and banner text

Copy it and paste inside banner.txt inside src/main/resources/folder

**Custom Location :** banner.location=classpath:/path/to/banner/custom-banner.txt

,--. ,--. ,--. ,--.

| .' / ,--,--. ,---. ,-' '-. ,--.,--. ,--.--. `--'

| . ' ' ,-. | ( .-' '-. .-' | || | | .--' ,--.

| |\ \ \ '-' | .-' `) | | ' '' ' | | | |

`--' '--' `--`--' `----' `--' `----' `--' `--'

* **Spring boot CommandLineRunner Interface:**

*CommandLineRunner interface in Spring Boot* provides an option to run a specific piece of code when the application is fully started. This interface called automatically by the Spring Boot after the initial bootstrapping of application.

1. **CommandLineRunner**

@Component

public class CustomCommandLineRunner implements CommandLineRunner {

private static final Logger LOG = LoggerFactory.getLogger(CustomCommandLineRunner.class);

@Override

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

LOG.info("Custom command line runner is excuted with command line arguments: {}", Arrays.toString(args));

}

}

CommandLineRunner interface offers a single run method, which is called just before SpringApplication.run(… ) completes. If we run our code, following log will be visible on the server console.

2018-07-06 21:54:11.096 INFO 27045 --- [ main] c.j.SpringBootExampleApplication : Started SpringBootExampleApplication in 2.195 seconds (JVM running for 2.998)

2018-07-06 21:54:11.098 INFO 27045 --- [ main] c.j.commandline.CustomCommandLineRunner : Custom command line runner is excuted with command line arguments: []

Copy

The CommandLineRunner interfaces provide access to application arguments as a simple string array.

2. **CommandLineRunner Ordering**

**2.1 Ordering using Ordered interface**

Implement *Ordered interface* and getOrder() method to provide priority for the custom runner.

@Component

public class CustomCommandLineRunner implements CommandLineRunner, Ordered {

private static final Logger LOG = LoggerFactory.getLogger(CustomCommandLineRunner.class);

@Override

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

LOG.info("Custom command line runner is excuted with command line arguments: {}", Arrays.toString(args));

}

@Override

public int getOrder() {

return 2;

}

}

**2.2 Ordering using @Order annotation**

Implement @Order annotation to provide priority for the custom runner.

@Component

@Order(1)

public class CustomCommandLineRunner2 implements CommandLineRunner {

private static final Logger LOG = LoggerFactory.getLogger(CustomCommandLineRunner2.class);

@Override

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

LOG.info("Calling second command line runner with arguments {}", Arrays.toString(args));

}

}

Copy

If we run our application, following output is visible on the server console

2018-07-06 22:03:13.906 INFO 27190 --- [ main] c.j.SpringBootExampleApplication : Started SpringBootExampleApplication in 1.811 seconds (JVM running for 2.555)

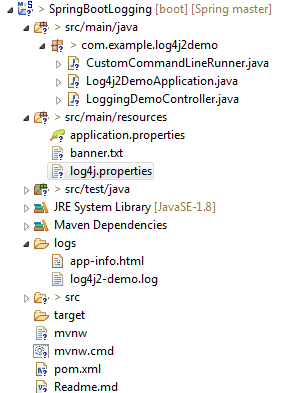
2018-07-06 22:03:13.907 INFO 27190 --- [ main] c.j.c.CustomCommandLineRunner2 : Calling second command line runner with arguments []

2018-07-06 22:03:13.907 INFO 27190 --- [ main] c.j.commandline.CustomCommandLineRunner : Custom command line runner is excuted with command line arguments: []

Copy

The lower the number, the higher the precedence

* **Spring boot Logging:**



We can configure log4j using 3 ways

* log4j.xml
* log4j.properties
* log4j.json

1. **log4j.xml :**

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

<Configuration status="warn">

<Properties>

<Property name="LOG\_PATTERN">

%d{yyyy-MM-dd HH:mm:ss.SSS} %5p ${hostName} --- [%15.15t] %-40.40c{1.} : %m%n%ex

</Property>

</Properties>

<Appenders>

<RollingFile name="fileLogger" fileName="logs/app-info.html"

filePattern="logs/app-info-%d{yyyy-MM-dd}.html">

<HTMLLayout charset="UTF-8" title="Howtodoinjava Info Logs" locationInfo="true" />

<Policies>

<TimeBasedTriggeringPolicy interval="1" modulate="true" />

<SizeBasedTriggeringPolicy size="10 MB" />

</Policies>

</RollingFile>

<RollingFile name="FileAppender" fileName="logs/log4j2-demo.log"

filePattern="logs/log4j2-demo-%d{yyyy-MM-dd}-%i.log">

<PatternLayout>

<Pattern>${LOG\_PATTERN}</Pattern>

</PatternLayout>

<Policies>

<TimeBasedTriggeringPolicy interval="1" />

<SizeBasedTriggeringPolicy size="10MB" />

</Policies>

<DefaultRolloverStrategy max="10"/>

</RollingFile>

<Console name="console" target="SYSTEM\_OUT">

<PatternLayout pattern="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n" />

</Console>

</Appenders>

<Loggers>

<Logger name="com.example.log4j2demo" level="debug" additivity="false">

<appender-ref ref="fileLogger" level="debug" />

<appender-ref ref="FileAppender" level="debug" />

</Logger>

<Root level="debug" additivity="false">

<appender-ref ref="console" />

</Root>

</Loggers>

</Configuration>

1. **log4j.properties :**

log4j.rootLogger=INFO, file

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=logs/log4j2-demo.log

log4j.appender.file.MaxFileSize=10MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=[%t] %-5p %c %x - %m%n

1. **log4j.json :**

{

"configuration": {

"name": "Default",

"appenders": {

"RollingFile": {

"name":"File",

"fileName":"logs/log4j2-demo.log",

"filePattern":"logs/log4j2-demo-%d{MM-dd-yy-HH-mm-ss}-%i.log.gz",

"PatternLayout": {

"pattern":"%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"

},

"Policies": {

"SizeBasedTriggeringPolicy": {

"size":"10 MB"

}

},

"DefaultRolloverStrategy": {

"max":"10"

}

}

},

"loggers": {

"root": {

"level":"debug",

"appender-ref": {

"ref":"File"

}

}

}

}

}

**Pom.xml:**

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

**LoggingDemoController.java :**

**package** com.example.log4j2demo;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

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

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** LoggingDemoController {

**private** **static** **final** Logger ***log*** = LogManager.*getLogger*(LoggingDemoController.**class**);

@GetMapping("/greeting")

**public** String sayHello() {

***log***.info("Info log statement for LoggingDemoController");

***log***.warn("Warn log statement for LoggingDemoController");

***log***.error("Error log statement for LoggingDemoController");

**return** "greeting";

}

}

**Log4j2DemoApplication.java:**

**package** com.example.log4j2demo;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

**import** org.springframework.boot.ApplicationArguments;

**import** org.springframework.boot.ApplicationRunner;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** Log4j2DemoApplication **implements** ApplicationRunner {

**private** **static** **final** Logger ***logger*** = LogManager.*getLogger*(Log4j2DemoApplication.**class**);

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Log4j2DemoApplication.**class**, args);

}

@Override

**public** **void** run(ApplicationArguments applicationArguments) **throws** Exception {

***logger***.debug("Debugging log");

***logger***.info("Info log");

***logger***.warn("Hey, This is a warning!");

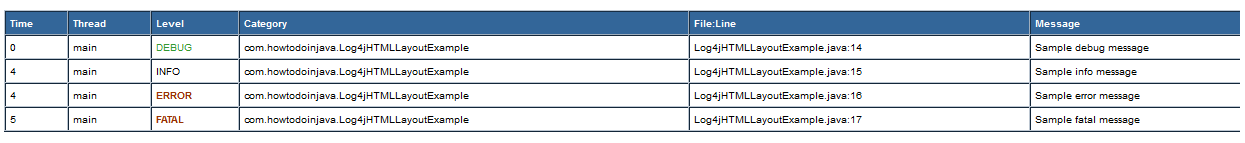
***logger***.error("Oops! We have an Error. OK");

***logger***.fatal("Damn! Fatal error. Please fix me.");

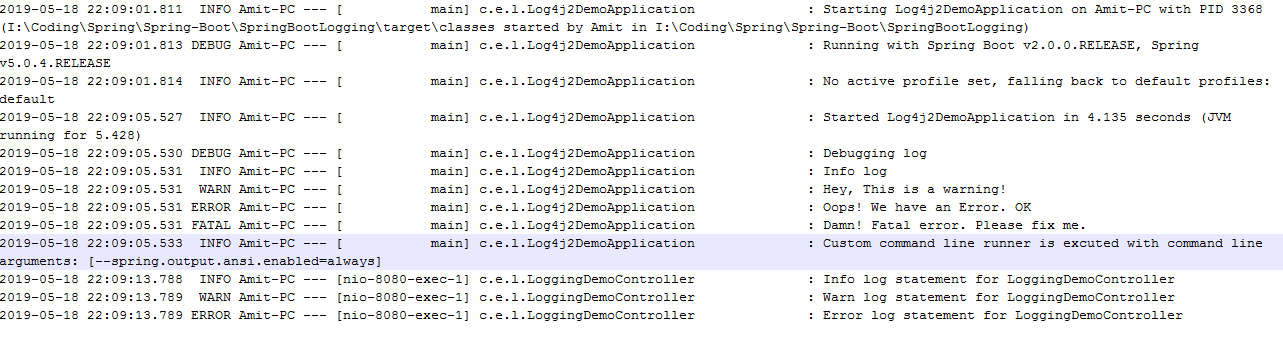
}

}

**HTML O/p:**



**Log File:**



# Spring Boot 2 REST API

## 1. Maven dependencies

At first, [create a simple maven web project](https://howtodoinjava.com/maven/maven-web-project-in-eclipse/) and update following spring boot dependencies in pom.xmlfile.

The important dependencies are spring-boot-starter-parent and spring-boot-starter-web .Starter web dependency transitively includes more dependencies to build a web application such as spring-webmvc, spring-web, hibernate-validator, tomcat-embed-core, tomcat-embed-el, tomcat-embed-websocket, jackson-databind, jackson-datatype-jdk8, jackson-datatype-jsr310 and jackson-module-parameter-names.

|  |
| --- |
| **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.howtodoinjava.demo</groupId>      <artifactId>springbootdemo</artifactId>      <version>0.0.1-SNAPSHOT</version>      <packaging>jar</packaging>        <name>SpringBootDemo</name>      <description>Spring Boot2 REST API </description>        <parent>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-starter-parent</artifactId>          <version>2.0.5.RELEASE</version>          <relativePath />      </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-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> |

## 2. Spring Boot 2 REST API Controller

* In Spring, a controller class, which is capable of serving REST API requests, is called rest controller. It should be annotated with **@RestController** annotation.
* The resource uris are specified in **@RequestMapping** annotations. It can be applied at class level and method level both. Complete URI for an API is resolved after adding class level path and method level path.
* We should always write **produces** and **consumes** attributes to specify the mediatype attributes for the API. Never reply on assumptions.

In given controller, we have two API methods. Feel free to add more methods as needed.

1. **HTTP GET /employees** – Returns list of the employees.
2. **HTTP POST /employees** – Add an employee in the employees collection.

|  |
| --- |
| **EmployeeController.java** |
| package com.howtodoinjava.rest.controller;   import java.net.URI;   import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.GetMapping;  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 org.springframework.web.servlet.support.ServletUriComponentsBuilder;   import com.howtodoinjava.rest.dao.EmployeeDAO;  import com.howtodoinjava.rest.model.Employee;  import com.howtodoinjava.rest.model.Employees;    @RestController  @RequestMapping(path = "/employees")  public class EmployeeController  {      @Autowired      private EmployeeDAO employeeDao;        @GetMapping(path="/", produces = "application/json")      public Employees getEmployees()      {          return employeeDao.getAllEmployees();      }        @PostMapping(path= "/", consumes = "application/json", produces = "application/json")      public ResponseEntity<Object> addEmployee(@RequestBody Employee employee)      {          Integer id = employeeDao.getAllEmployees().getEmployeeList().size() + 1;          employee.setId(id);            employeeDao.addEmployee(employee);            URI location = ServletUriComponentsBuilder.fromCurrentRequest()                                      .path("/{id}")                                      .buildAndExpand(employee.getId())                                      .toUri();            return ResponseEntity.created(location).build();      }  } |

We can control and customize a lots of implementation details using application.properties file. But to keep this demo simple, I am leaving it blank.

## 3. @SpringBootApplication

Our REST APIs skeleton is ready. Now we need to configure Spring to detect our rest controller (using auto scanning) and deploy apis in embedded tomcat server. Thankfully, Spring boot makes all these things very easy by using the concept of [auto configuration](https://howtodoinjava.com/spring-boot2/springbootapplication-auto-configuration/).

**Auto-configuration** attempts to guess and configure beans we you are likely to need. Auto-configuration classes are usually applied based on the jars in application classpath and the beans we have defined additionally in **@Configuration** classes.

In this case, it does following things.

1. It detects **spring-webmvc** so configure default spring mvc application beans. It help in scan and configure @RestController and similar annotations.
2. It detects embed tomcat jars so configure embedded tomcat for us.
3. It detects JSON jars so configure JSON support to APIs.

|  |
| --- |
| **SpringBootDemoApplication.java** |
| package com.howtodoinjava.rest;    import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;    @SpringBootApplication  public class SpringBootDemoApplication {        public static void main(String[] args) {          SpringApplication.run(SpringBootDemoApplication.class, args);      }  } |

## 4. Model classes and DAO

These classes are not directly related to REST. Still lets take a look how they have been written.

|  |
| --- |
| Employee.java |
| package com.howtodoinjava.rest.model;   public class Employee {        public Employee() {        }        public Employee(Integer id, String firstName, String lastName, String email) {          super();          this.id = id;          this.firstName = firstName;          this.lastName = lastName;          this.email = email;      }        private Integer id;      private String firstName;      private String lastName;      private String email;        //Getters and setters        @Override      public String toString() {          return "Employee [id=" + id + ", firstName=" + firstName + ",                  lastName=" + lastName + ", email=" + email + "]";      }  } |

|  |
| --- |
| **Employees.java** |
| package com.howtodoinjava.rest.model;  import java.util.ArrayList;  import java.util.List;    public class Employees  {      private List<Employee> employeeList;        public List<Employee> getEmployeeList() {          if(employeeList == null) {              employeeList = new ArrayList<>();          }          return employeeList;      }        public void setEmployeeList(List<Employee> employeeList) {          this.employeeList = employeeList;      }  } |

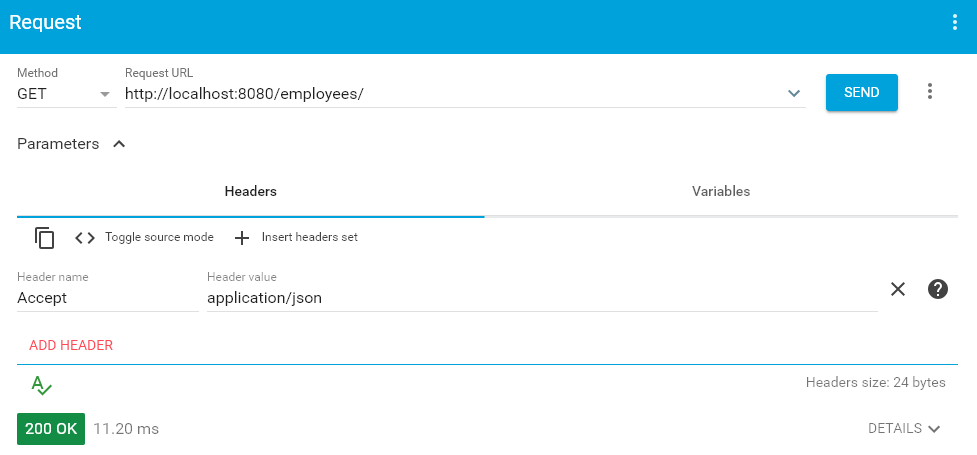
DAO class uses a static list to store data. Here we need to implement actual database interaction.

|  |
| --- |
| **EmployeeDAO.java :** |
| package com.howtodoinjava.rest.dao;  import org.springframework.stereotype.Repository;  import com.howtodoinjava.rest.model.Employee;  import com.howtodoinjava.rest.model.Employees;    @Repository  public class EmployeeDAO  {      private static Employees list = new Employees();        static      {          list.getEmployeeList().add(new Employee(1, "Lokesh", "Gupta","howtodoinjava@gmail.com"));          list.getEmployeeList().add(new Employee(2, "Alex", "Kolenchiskey", "abc@gmail.com"));          list.getEmployeeList().add(new Employee(3, "David", "Kameron", "titanic@gmail.com"));      }        public Employees getAllEmployees()      {          return list;      }        public void addEmployee(Employee employee) {          list.getEmployeeList().add(employee);      }  } |

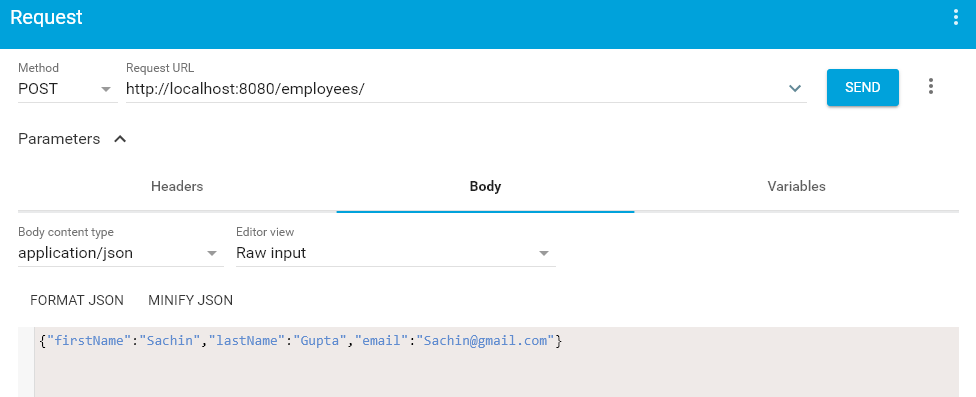
## 5. Spring Boot REST Demo

To start the application, run the main() method in SpringBootDemoApplication class. It will start the embedded tomcat server. In server logs, you will see that API have been registered in spring context. 5.1. HTTP GET /employees

Once server is UP, access the API using some rest client.



|  |
| --- |
| API response |
| {      "employeeList": [      {          "id": 1,          "firstName": "Lokesh",          "lastName": "Gupta",          "email": "howtodoinjava@gmail.com"      },      {          "id": 2,          "firstName": "Alex",          "lastName": "Kolenchiskey",          "email": "abc@gmail.com"      },          {              "id": 3,              "firstName": "David",              "lastName": "Kameron",              "email": "titanic@gmail.com"          }      ],  } 5.2. HTTP POST /employees |





# Spring boot exception handling

In this spring boot example, we will see primarily two major validation cases –

1. HTTP POST /employees and request body does not contain valid values or some fields are missing. It will return HTTP status code 400 with proper message in response body.
2. HTTP GET /employees/{id} and INVALID ID is sent in request. It will return HTTP status code 404 with proper message in response body.

## 1. Create REST APIs and model classes

|  |
| --- |
| **EmployeeRESTController.java** |
| @PostMapping(value = "/employees")  public ResponseEntity<EmployeeVO> addEmployee (@RequestBody EmployeeVO employee)  {      EmployeeDB.addEmployee(employee);      return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);  }    @GetMapping(value = "/employees/{id}")  public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") int id)  {      EmployeeVO employee = EmployeeDB.getEmployeeById(id);        if(employee == null) {           throw new RecordNotFoundException("Invalid employee id : " + id);      }      return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);  } |

|  |
| --- |
| **EmployeeVO.java** |
| @XmlRootElement(name = "employee")  @XmlAccessorType(XmlAccessType.FIELD)  public class EmployeeVO extends ResourceSupport implements Serializable  {      private Integer employeeId;      private String firstName;      private String lastName;      private String email;        public EmployeeVO(Integer id, String firstName, String lastName, String email) {          super();          this.employeeId = id;          this.firstName = firstName;          this.lastName = lastName;          this.email = email;      }        public EmployeeVO() {      }    } |

## 2. Spring boot exception handling – REST request validation

#### 2.1. Default spring validation support

To apply default validation, we only need to add relevant annotations in proper places. i.e.

1. **Annotate model class with required validation specific annotations such as @NotEmpty, @Email etc.**

|  |
| --- |
| @XmlRootElement(name = "employee")  @XmlAccessorType(XmlAccessType.FIELD)  public class EmployeeVO extends ResourceSupport implements Serializable  {      private static final long serialVersionUID = 1L;      public EmployeeVO(Integer id, String firstName, String lastName, String email) {          super();          this.employeeId = id;          this.firstName = firstName;          this.lastName = lastName;          this.email = email;      }        public EmployeeVO() {      }       private Integer employeeId;        @NotEmpty(message = "first name must not be empty")      private String firstName;        @NotEmpty(message = "last name must not be empty")      private String lastName;        @NotEmpty(message = "email must not be empty")      @Email(message = "email should be a valid email")      private String email;  } |

1. **Enable validation of request body by @Valid annotation**

|  |
| --- |
| @PostMapping(value = "/employees")  public ResponseEntity<EmployeeVO> addEmployee (@Valid @RequestBody EmployeeVO employee)  {      EmployeeDB.addEmployee(employee);      return new ResponseEntity<EmployeeVO>(employee, HttpStatus.OK);  } |

#### 2.2. Exception model classes

Default spring validation works and provide information overload about error, and that’s why we should customize it according to our application’s need. We shall provide only required error information with very clear wordings. Extra information is also not suggested.

It is always a good advise to create exceptions that are meaningful and describe the problem well enough. One way is to create seperate classes to denote specific business usecase failure and return them when that usecase fail.

e.g. I have created RecordNotFoundException class for all buch scenarios where a resource is requested by it’s ID, and resource is not found in the system.

|  |
| --- |
| **RecordNotFoundException.java** |
| package com.howtodoinjava.demo.exception;   import org.springframework.http.HttpStatus;  import org.springframework.web.bind.annotation.ResponseStatus;    @ResponseStatus(HttpStatus.NOT\_FOUND)  public class RecordNotFoundException extends RuntimeException  {      public RecordNotFoundException(String exception) {      super(exception);      }  } |

Similarly, I have wrote an special class which will be returned for all failure cases. Having consistent error message structure for all APIs, help the API consumers to write more robust code.

|  |
| --- |
| **ErrorResponse.java** |
| import java.util.List;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name = "error")  public class ErrorResponse  {      public ErrorResponse(String message, List<String> details) {          super();          this.message = message;          this.details = details;      }        //General error message about nature of error      private String message;        //Specific errors in API request processing      private List<String> details;       //Getter and setters  } |

#### 2.3. Custom ExceptionHandler

Now add one class extending ResponseEntityExceptionHandler and annotate it with @ControllerAdvice annotation.

ResponseEntityExceptionHandler is a convenient base class for to provide centralized exception handling across all @RequestMapping methods through @ExceptionHandler methods. @ControllerAdvice is more for enabling auto-scanning and configuration at application startup.

|  |
| --- |
| **CustomExceptionHandler.java** |
| package com.howtodoinjava.demo.exception;   import java.util.ArrayList;  import java.util.List;   import org.springframework.http.HttpHeaders;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.validation.ObjectError;  import org.springframework.web.bind.MethodArgumentNotValidException;  import org.springframework.web.bind.annotation.ControllerAdvice;  import org.springframework.web.bind.annotation.ExceptionHandler;  import org.springframework.web.context.request.WebRequest;  import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;    @SuppressWarnings({"unchecked","rawtypes"})  @ControllerAdvice  public class CustomExceptionHandler extends ResponseEntityExceptionHandler  {      @ExceptionHandler(Exception.class)      public final ResponseEntity<Object> handleAllExceptions(Exception ex, WebRequest request) {          List<String> details = new ArrayList<>();          details.add(ex.getLocalizedMessage());          ErrorResponse error = new ErrorResponse("Server Error", details);          return new ResponseEntity(error, HttpStatus.INTERNAL\_SERVER\_ERROR);      }        @ExceptionHandler(RecordNotFoundException.class)      public final ResponseEntity<Object> handleUserNotFoundException(RecordNotFoundException ex, WebRequest request) {          List<String> details = new ArrayList<>();          details.add(ex.getLocalizedMessage());          ErrorResponse error = new ErrorResponse("Record Not Found", details);          return new ResponseEntity(error, HttpStatus.NOT\_FOUND);      }        @Override      protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {          List<String> details = new ArrayList<>();          for(ObjectError error : ex.getBindingResult().getAllErrors()) {              details.add(error.getDefaultMessage());          }          ErrorResponse error = new ErrorResponse("Validation Failed", details);          return new ResponseEntity(error, HttpStatus.BAD\_REQUEST);      }  } |

Above class handles multiple exceptions including RecordNotFoundException and it also handle request validation errors in @RequestBody annotated object.

## Demo

1) HTTP GET /employees/1 [VALID]

|  |
| --- |
| HTTP Status : 200    {      "employeeId": 1,      "firstName": "John",      "lastName": "Wick",      "email": "howtodoinjava@gmail.com",  } |

2) HTTP GET /employees/23 [INVALID]

|  |
| --- |
| HTTP Status : 404    {      "message": "Record Not Found",      "details": [          "Invalid employee id : 23"      ]  } |

3) HTTP POST /employees [INVALID]

|  |
| --- |
| **Request** |
| {      "lastName": "Bill",      "email": "ibill@gmail.com"  } |

|  |
| --- |
| **Response** |
| HTTP Status : 400    {      "message": "Validation Failed",      "details": [          "first name must not be empty"      ]  } |

4) HTTP POST /employees [INVALID]

|  |
| --- |
| **Request** |
| {      "email": "ibill@gmail.com"  } |

|  |
| --- |
| **Response** |
| HTTP Status : 400    {      "message": "Validation Failed",      "details": [          "last name must not be empty",          "first name must not be empty"      ]  } |

5) HTTP POST /employees [INVALID]

|  |
| --- |
| **Request** |
| {      "firstName":"Lokesh",      "email": "ibill\_gmail.com" //invalid email in request  } |

|  |
| --- |
| **Response** |
| HTTP Status : 400    {      "message": "Validation Failed",      "details": [          "last name must not be empty",          "email should be a valid email"      ]  } |

## 4. REST request validation annotations

In above example, we used only few annotations such as @NotEmpty and @Email. There are more such annotations to validate request data. Check them out when needed.

|  |  |
| --- | --- |
| **ANNOTATION** | **USAGE** |
| @AssertFalse | The annotated element must be false. |
| @AssertTrue | The annotated element must be true. |
| @DecimalMax | The annotated element must be a number whose value must be lower or equal to the specified maximum. |
| @DecimalMin | The annotated element must be a number whose value must be higher or equal to the specified minimum. |
| @Future | The annotated element must be an instant, date or time in the future. |
| @Max | The annotated element must be a number whose value must be lower or equal to the specified maximum. |
| @Min | The annotated element must be a number whose value must be higher or equal to the specified minimum. |
| @Negative | The annotated element must be a strictly negative number. |
| @NotBlank | The annotated element must not be null and must contain at least one non-whitespace character. |
| @NotEmpty | The annotated element must not be null nor empty. |
| @NotNull | The annotated element must not be null. |
| @Null | The annotated element must be null. |
| @Pattern | The annotated CharSequence must match the specified regular expression. |
| @Positive | The annotated element must be a strictly positive number. |
| @Size | The annotated element size must be between the specified boundaries (included). |

# Spring Boot Devtools

If you have worked on latest UI development frameworks e.g. Node, [angular](https://howtodoinjava.com/angularjs/angularjs-tutorial-helloworld-example/), gulp etc. then you must have appreciated the auto-reload of UI in browser whenever there is change in some code. Its pretty useful and saves a lot of time.

Well, same feature can be utilized in spring boot applications using spring-boot-devtools dependency provided features. Let’s learn about enabling these features and using them.

**Enabling Dev Tools Module**

To enable dev tools in spring boot application is very easy. Just add the spring-boot-devtoolsdependency in your build file.

|  |
| --- |
| **pom.xml** |
| <dependencies>      <dependency>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-devtools</artifactId>          <optional>true</optional>      </dependency>  </dependencies> Static Resource Caching To improve the performance, dev tools cache the static content/template files to serve them faster to browser/client. This is very good feature in production where every milli-second [performance improvement](https://howtodoinjava.com/best-practices/improving-web-application-performance/) matters. But in development environment, it can be a problem and cause stale cache problem and you may not see your changes immediatly in browser. Dev tools module provide this capability by setting few properties.  By default, this feature is disabled. You can enable it to use in production environment by setting a property.  There are many such UI template libraries that support this feature. e.g. thymeleaf, freemarker, groovy, mustache etc.   |  | | --- | | **application.properties** | | #spring.freemarker.cache = true //set true in production environment  spring.freemarker.cache = false //set false in development environment; It is false by default.   //Other such properties   spring.thymeleaf.cache = false  spring.mustache.cache = false  spring.groovy.template.cache = false | |

## Automatic UI refresh

The spring-boot-devtools module includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. Precondition is that your browser should have supported extention for it. By default, live reload is enabled. If you wish to disable this feature for some reason, then set spring.devtools.livereload.enabled property to false.

|  |
| --- |
| **application.properties** |
| spring.devtools.livereload.enabled  = false #Set false to disable live reload |

#### Excluding Resources from auto-reload

By default, Auto-reload works on these paths:

1. /META-INF/maven
2. /META-INF/resources
3. /resources
4. /static
5. /public
6. /templates

If you want to disable auto-reload in browser for files in few of these paths, then use spring.devtools.restart.exclude property. e.g.

|  |
| --- |
| spring.devtools.restart.exclude=static/\*\*,public/\*\* |

#### Watching/Excluding Additional Paths

There may be few files not in classpath, but you still may want to watch those addtional files/paths to reload the application. To do so, use the spring.devtools.restart.additional-paths property.

|  |
| --- |
| spring.devtools.restart.additional-paths=script/\*\* |

Similarily, If you want to keep those defaults and **add additional exclusions**, use the spring.devtools.restart.additional-exclude property instead.

|  |
| --- |
| spring.devtools.restart.additional-exclude=styles/\*\* |

## Automatic server restart

Auto-restart means reloading the java classes and consiguration at server side. After the server side changes are re-deployed dynamically, server restart happen and load the modified code and configutation.

#### Enable/disable logging of auto-configuration changes

By default, each time your application restarts, a report showing the condition evaluation delta is logged. The report shows the changes to your application’s auto-configuration as you make changes such as adding or removing beans and setting configuration properties.

To disable the logging of the report, set the following property:

|  |
| --- |
| spring.devtools.restart.log-condition-evaluation-delta = false |

#### Disabling Restart

To disable the restart of server on non-static code changes, use the property spring.devtools.restart.enabled.

|  |
| --- |
| spring.devtools.restart.enabled = false |

#### Using a Trigger File

Automatic restarts may be desirable on every file change and sometimes can slower down development time due to frequent restarts. To solve this problem, you can use a trigger file. Spring boot will keep monitoring that file and once it will detect any modification in that file, it will restart the server and reload all your previous changes.

Use spring.devtools.restart.trigger-file property to mention the trigger file for your application. It can be any external or internal file.

|  |
| --- |
| spring.devtools.restart.trigger-file = c:/workspace/restart-trigger.txt |

## Global settings file

Setting all your favorite configutation options everytime for all your spring boot projects or modules may become a duplicate effort. You can minimize it using a global setting file. Then individual projects/module will inherit all custom settings from global file, and if needed they can override any specific setting per project basis.

To create global file, go to your system’s user’s home directory and create a file named .spring-boot-devtools.properties. (Please note that file name start with a dot). Not use this global property file to configure globally available options.

|  |
| --- |
|  |
| spring.devtools.restart.trigger-file = c:/workspace/restart-trigger.txt Spring boot basic authentication database : Here is spring boot basic authentication database using Spring security. During RESTful web service development, basic authentication is a primary requirement so that it is only accessible from authenticated users. Spring Security provides basic authentication using JDBC database authentication |



**Pom.xml:**

**<dependency>**

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

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId> <!--starter require for spring boot spring security-->**

**<artifactId>spring-boot-starter-security</artifactId>**

**</dependency>**

**<dependency>**

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

**<artifactId>spring-boot-starter-jdbc</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>mysql</groupId> <!-- Its related to mysql-->**

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

**<version>5.1.6</version>**

**</dependency>**

**application.properties :**

It’s data source properties which contain information related to connecting with a database. app.datasource is base properties name. which has been used when autowire @ConfigurationProperties("app.datasource").

1. app.datasource.url=jdbc:mysql://localhost/demo\_database
2. app.datasource.username=root
3. app.datasource.password=
4. app.datasource.driver-class-name=com.mysql.jdbc.Driver

**SpringBootConfig :**

package com.javadeveloperzone;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import javax.sql.DataSource;

@SpringBootApplication

@ComponentScan({"com.javadeveloperzone"})

// Using a root package also allows the @ComponentScan annotation to be used without needing to specify a basePackage attribute

public class SpringBootConfig {

public static void main(String[] args) throws Exception {

SpringApplication.run(SpringBootConfig.class, args); // it wil start application

}

@Bean(value = "datasource")

@ConfigurationProperties("app.datasource")

public DataSource dataSource() {

return DataSourceBuilder.create().build();

}

}

**SecurityConfiguration :**

package com.javadeveloperzone;

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

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

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.stereotype.Component;

import javax.sql.DataSource;

@Configuration

@Component

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

@Qualifier("datasource")

private DataSource dataSource;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.jdbcAuthentication().dataSource(dataSource)

.authoritiesByUsernameQuery("select employeeName, employeeRole FROM employee where employeeName=?")

.usersByUsernameQuery("select employeeName,employeePassword as password,1 FROM employee where employeeName=?");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.httpBasic() // it indicate basic authentication is requires

.and()

.authorizeRequests()

.antMatchers( "/index").permitAll() // /index will be accessible directly, no need of any authentication

.anyRequest().authenticated(); // it's indicate all request will be secure

http.csrf().disable();

}

}

**SpringBootExampleController.java:**

package com.javadeveloperzone.controller;

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

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

/\*\*

\* Created by JavaDeveloperZone on 19-07-2017.

\*/

@RestController

public class SpringBootExampleController {

@RequestMapping("/")

public String SpringBootHello() {

return "spring boot basic authentication database";

}

}