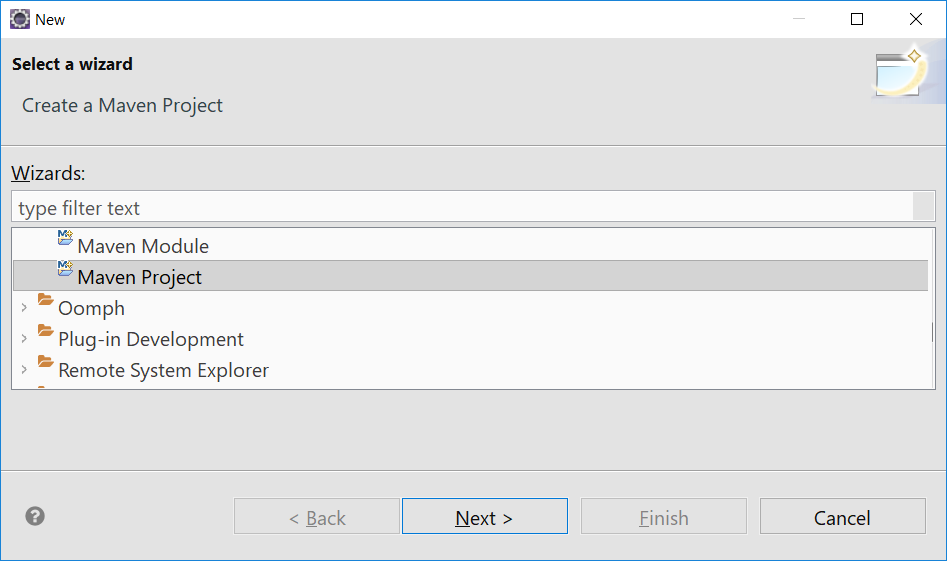
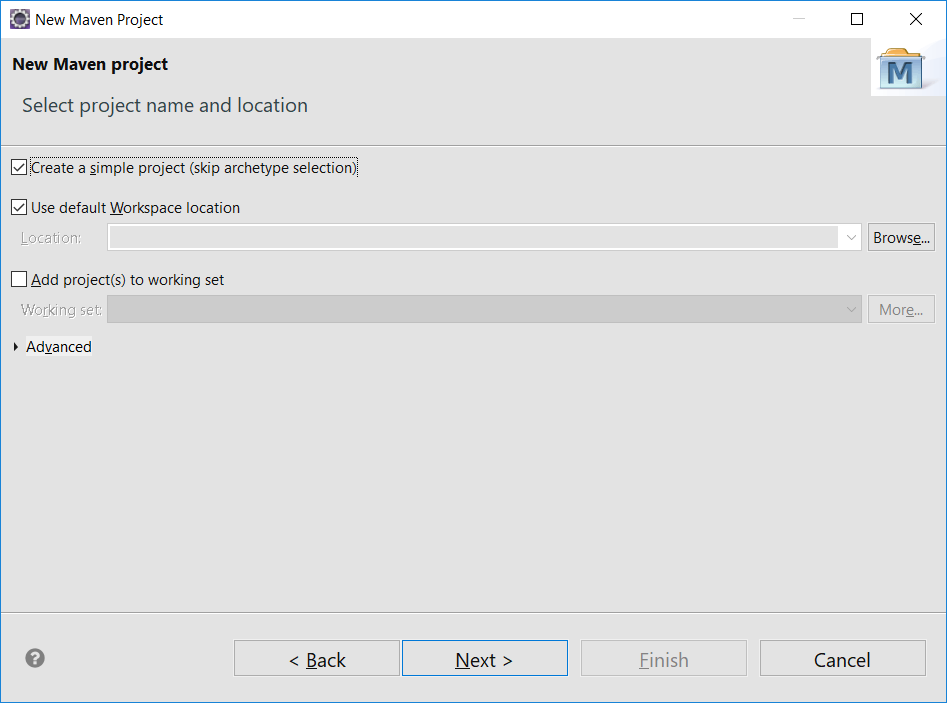
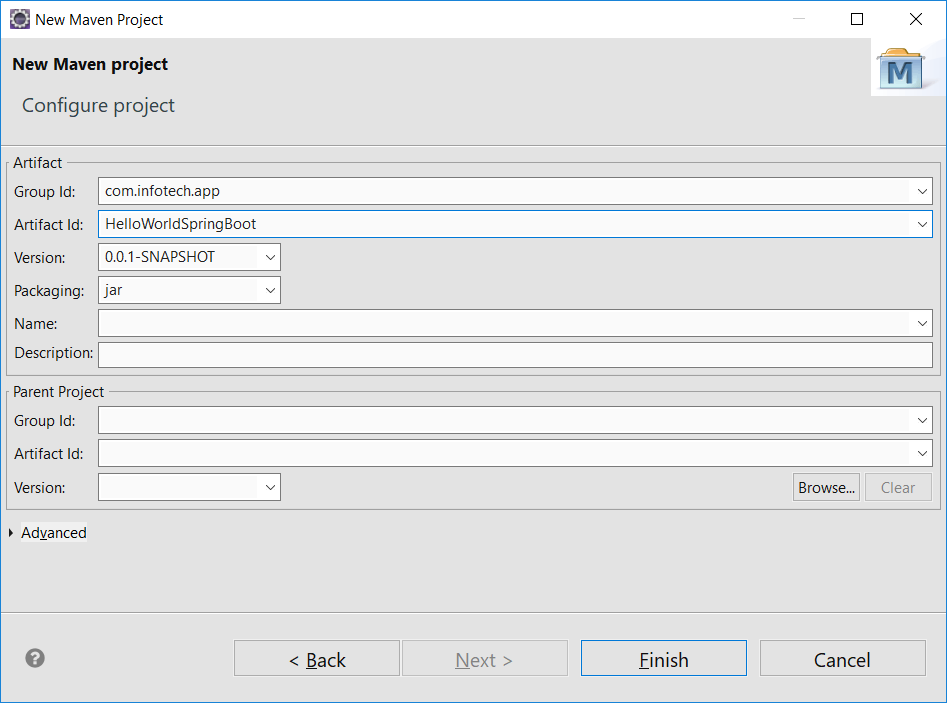
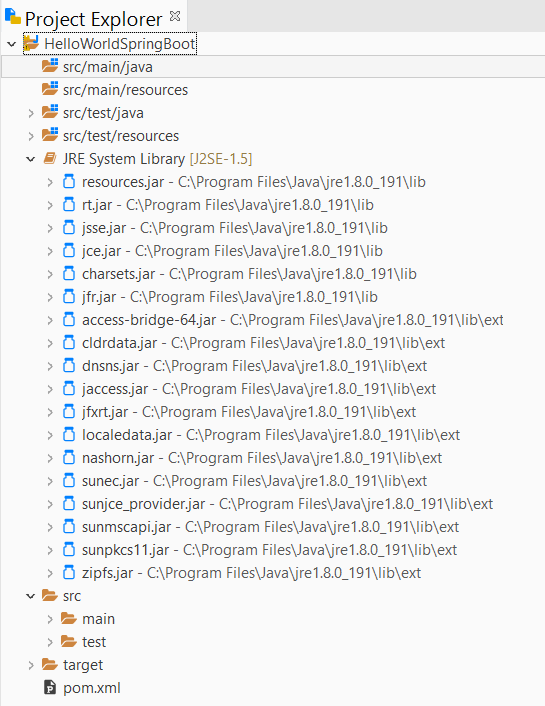
1. First just create a simple maven project.







Click finish



* Now let’s open pom.xml and add parent:

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.2.RELEASE</version>  </parent> |

* Now after adding parent lets add some dependencies:

|  |
| --- |
| <!-- Adding dependencies for the spring project -->  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

Here we don’t need to mention version because spring is smart enough and it will download all the jars according to the version mention in the parent.

* Now specify the java 1.8 version in pom.xml

|  |
| --- |
| <properties>  <java.version>1.8</java.version>  </properties> |

Now right click on the project go to Maven =🡺 Update Project. It will download all the latest jar files mentioned in pom.xml file.

* Now let’s create one class HelloWorldSpringBoot create one main method inside this class. Annotate this class with annotation @SpringBootApplication and import one package for this annotation.

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

|  |
| --- |
| **package** com.infotech.app;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **publicclass** HelloWorldSpringBoot {  **publicstaticvoid** main(String[] args) **throws** Exception {  SpringApplication.*run*(HelloWorldSpringBoot.**class**, args);  }  } |

Here we have passed the HelloWorldSpringBoot.**class** and command line argument **args** so that spring boot application can understand that it is the main class from where the execution has to start.

Now let’s create a RestController class

|  |
| --- |
| **package** com.infotech.app.controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **publicclass** HelloWorldController {  @GetMapping(value="/hello")  //@RequestMapping(value="/hello", method =RequestMethod.***GET***)  **public** String sayHello() {  **return**"Hello World!!”  }  } |

**@RestController:**This controller makes the class as controller class.

**@RequestMapping (value="/hello", method =RequestMethod.*GET*)** Using this annotation we are creating one resource url/hello using this URL we will hit SayHello method of HelloWorldController class.

**@GetMapping**(value="/hello"): This is particularly using for getting resource we don’t need to mention RequestMethod as we have mentioned in case of @RequestMapping

**Difference between (@Controller and @RestController).**

|  |  |
| --- | --- |
| 1. @Controller is an old annotation, exists since Spring 2.5 version. | 1. The **@RestController** is relatively new, added only on Spring 4.0 |
| 2- The @Controller is a common annotation which is used to mark a class as Spring MVC Controller. | 2-@RestController is a special controller used in [RESTFul web services](http://javarevisited.blogspot.sg/2015/08/difference-between-soap-and-restfull-webservice-java.html) and the equivalent of @Controller + @ResponseBody. |
| 3. The @Controller annotation indicates that the class is a "Controller" e.g. a web controller | 3- **@RestController** annotation indicates that the class is a controller where @RequestMapping methods assume @ResponseBody by default i.e. servicing REST API. |
| 4- The @Controller is a specialization of @Component annotation | 4-@RestController is a specialization of @Controller annotation. |
| 5- When the class is annotated with @Controller then every method written inside the class returns the view | 5-if class is marked with  @RestController then every method is written inside the class returns an object instead of a view |
| 6. Another key difference between @RestController and @Controller is that you don't need to use @ResponseBody on every handler method once you annotate the class with @RestController as shown below: | |
| @Controller  public class Book{  @RequestMapping(value={"/book"})  @ResponseBody  public Book getBook(){  //... return book;  }} | @RestController  public class Book{  @RequestMapping(value={"/book"})  public Book getBook(){  //... return book;  }} |

Now run the above application: Go to **HelloWorldSpringBoot** main class. And run as java application

<http://localhost:8080/hello>

Finally the pom.xml file will look like:

|  |
| --- |
| <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.infotech.app</groupId>  <artifactId>HelloWorldSpringBoot</artifactId>  <version>0.0.1-SNAPSHOT</version>  <!-- Adding parent for getting spring boot capabilities -->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.2.RELEASE</version>  </parent>  <!-- Adding dependencies for the spring boot web project -->  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies>  <!-- Adding java 1.8 version for the spring boot project -->  <properties>  <java.version>1.8</java.version>  </properties>  </project> |

**Changing the default port 8080 to 9090**

1. Create one application.properities file inside resources package.
2. Inside the properties file mention [ server.port=9090]
3. Restart the server and try to run the same application. Now instead of 8080 the application will run on 9090 server port.
4. <http://localhost:9090/hello>

**Creating jar file and running through CMD:**

1. For packaging the project in the jar file we have to mention this under root tag <project> in pom.xml

|  |
| --- |
| <packaging>jar</packaging> |
| <build>  <plugins>  <plugin>  <groupId>org.springgramework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

The above plugin will create one executatble jar or war files in the target folder of the application.

1. Now go to the path where pom.xml is present in the application

F:\Eclipse\_Oxygen\_WorkSpace\HelloWorldSpringBoot>



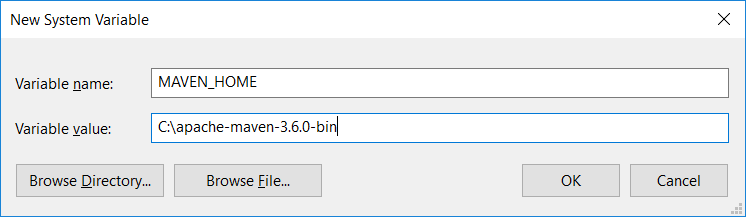
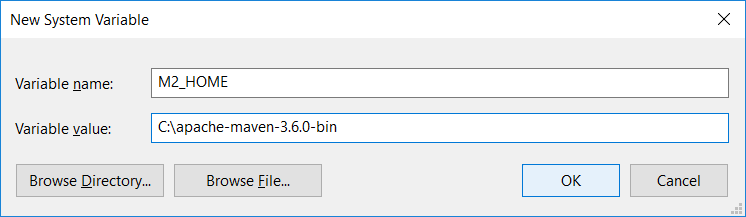
1. Now in this path execute one command mvn install.



The above command will build the application and will generate one jar file in the target fold of the application.

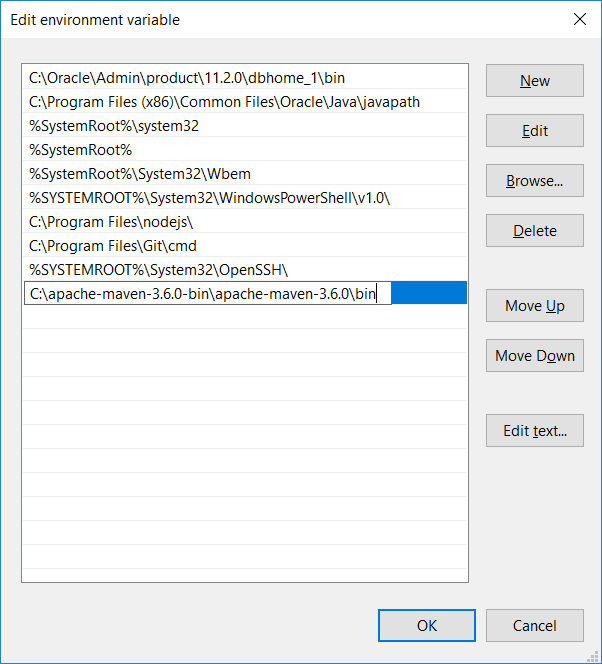
**Note**: If mvn is not recognized as internal or external command then we need to set the maven path in environment variable for M2\_HOME, MAVEN\_HOME

C:\apache-maven-3.6.0-bin

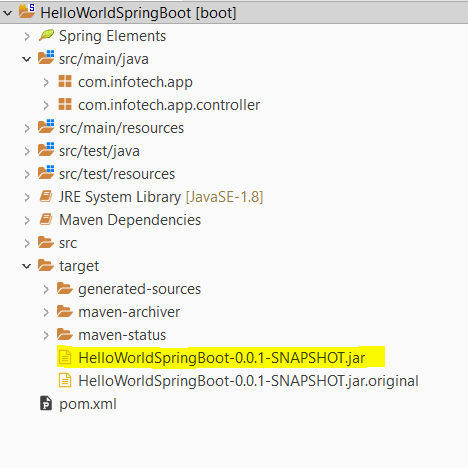


In the Set the maven bin path inside the path variable by editing it.

C:\apache-maven-3.6.0-bin\apache-maven-3.6.0\bin



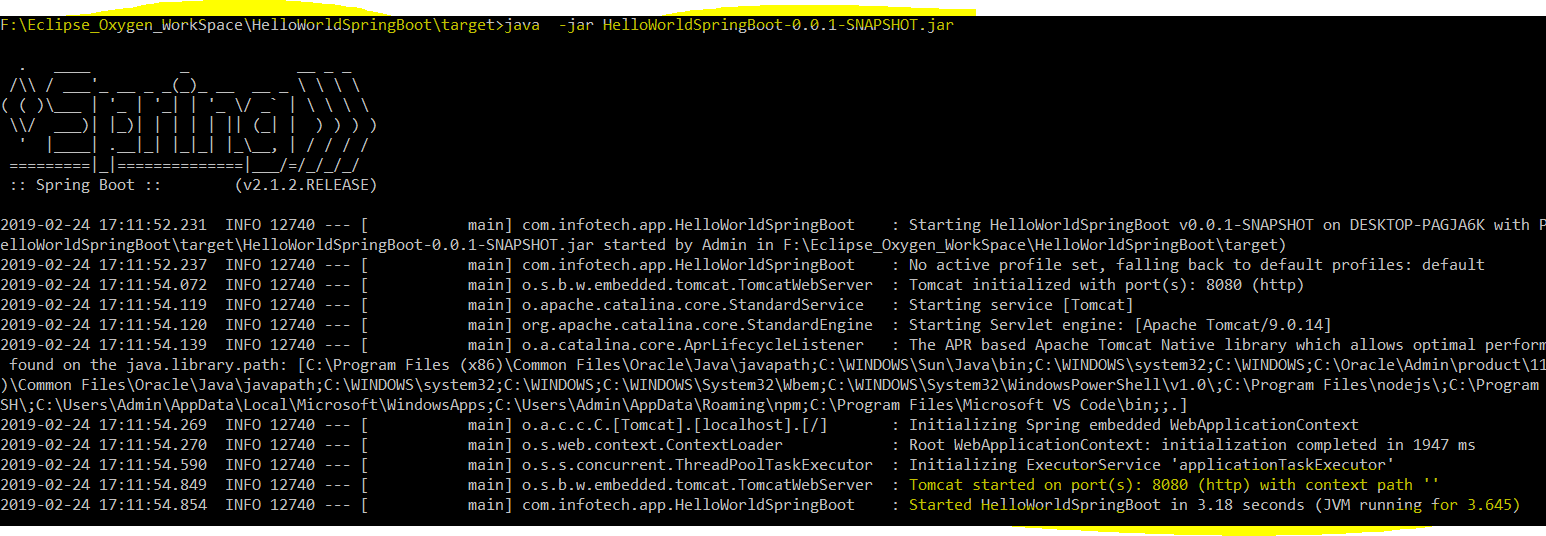
1. To check maven installation run : mvn –version ; it will give you version of installed maven.
2. Now when we run [mvn install] command then it will download the jar file in the target folder.
3. Another way to create jar file is from eclipse only. Go to 🡺 Run 🡪 Run As🡺 Maven Build



Now we can run the application using this jar file from command prompt as given below.

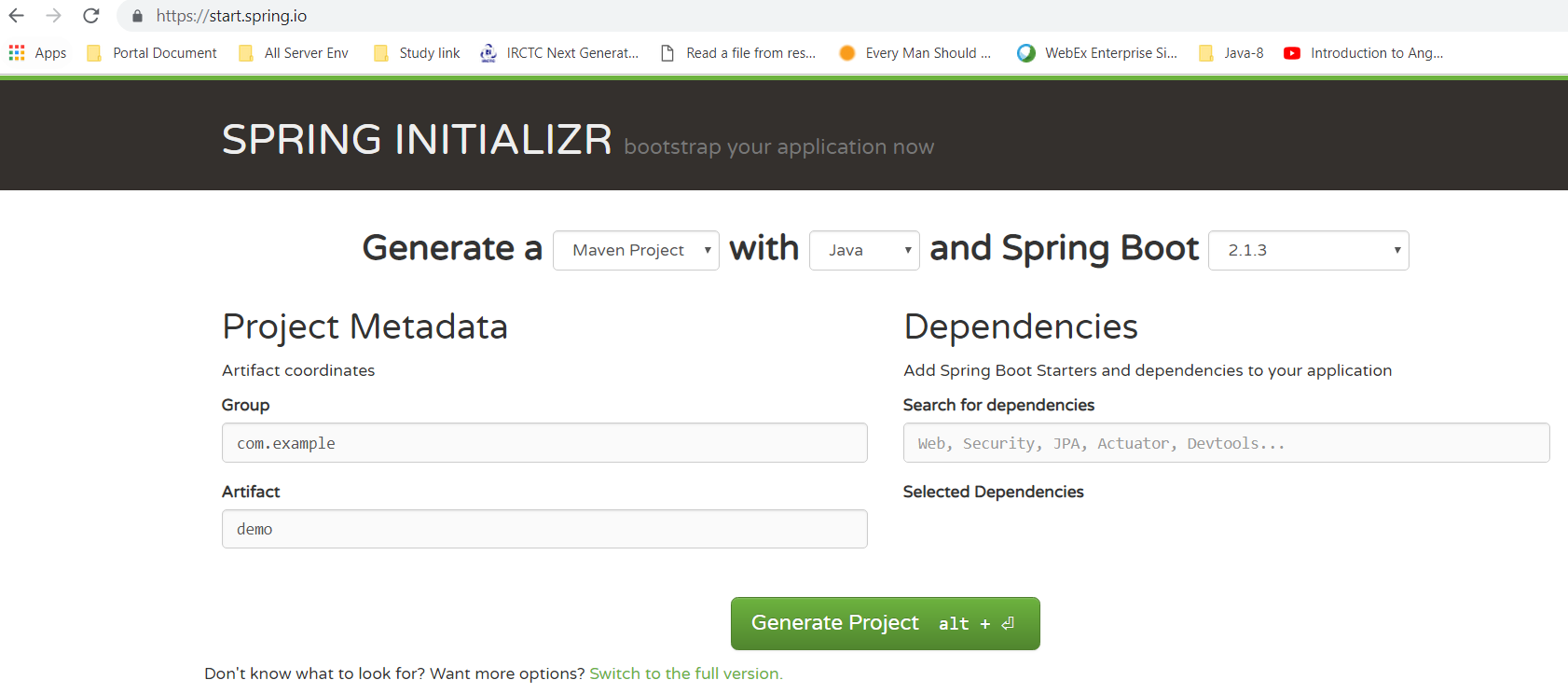
**F:\Eclipse\_Oxygen\_WorkSpace\HelloWorldSpringBoot\target>java -jar HelloWorldSpringBoot-0.0.1-SNAPSHOT.jar**

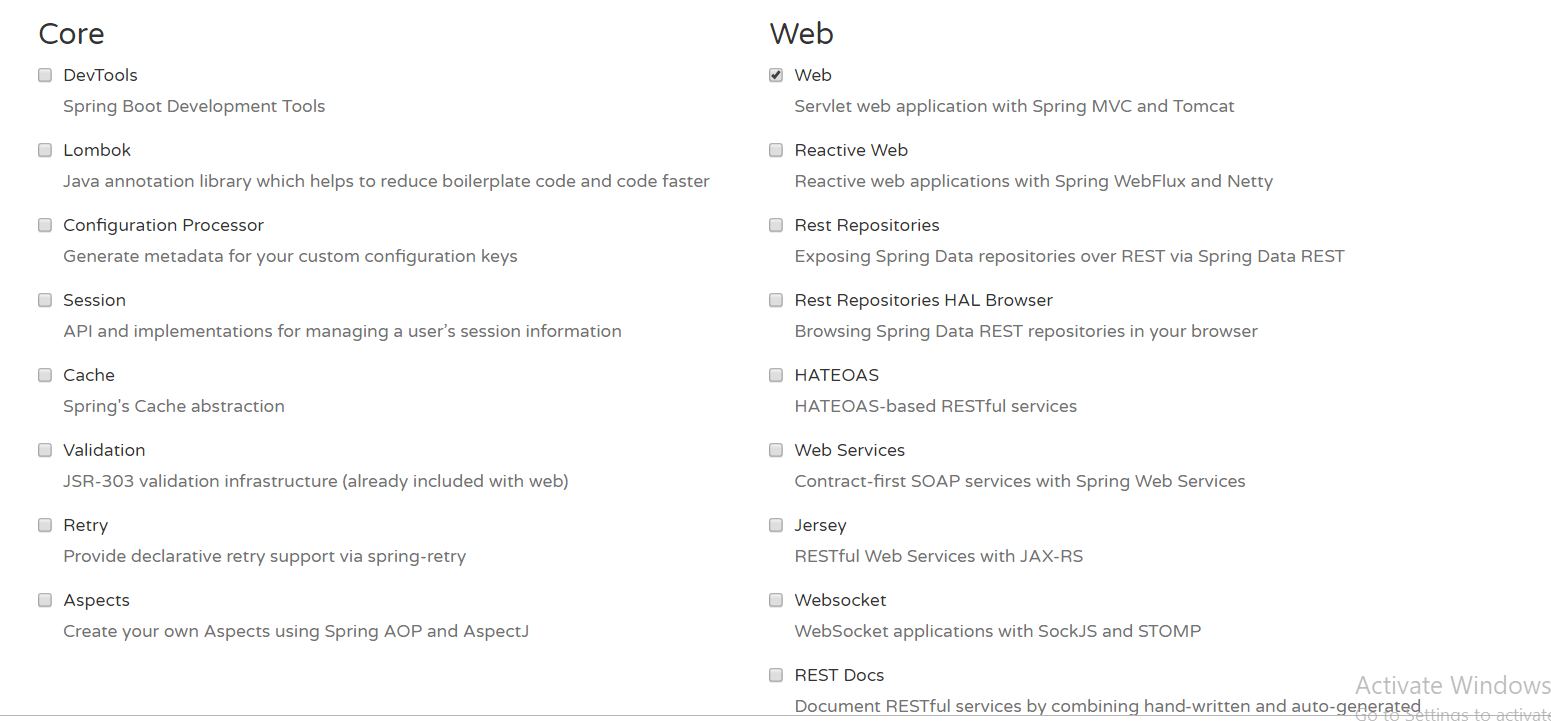
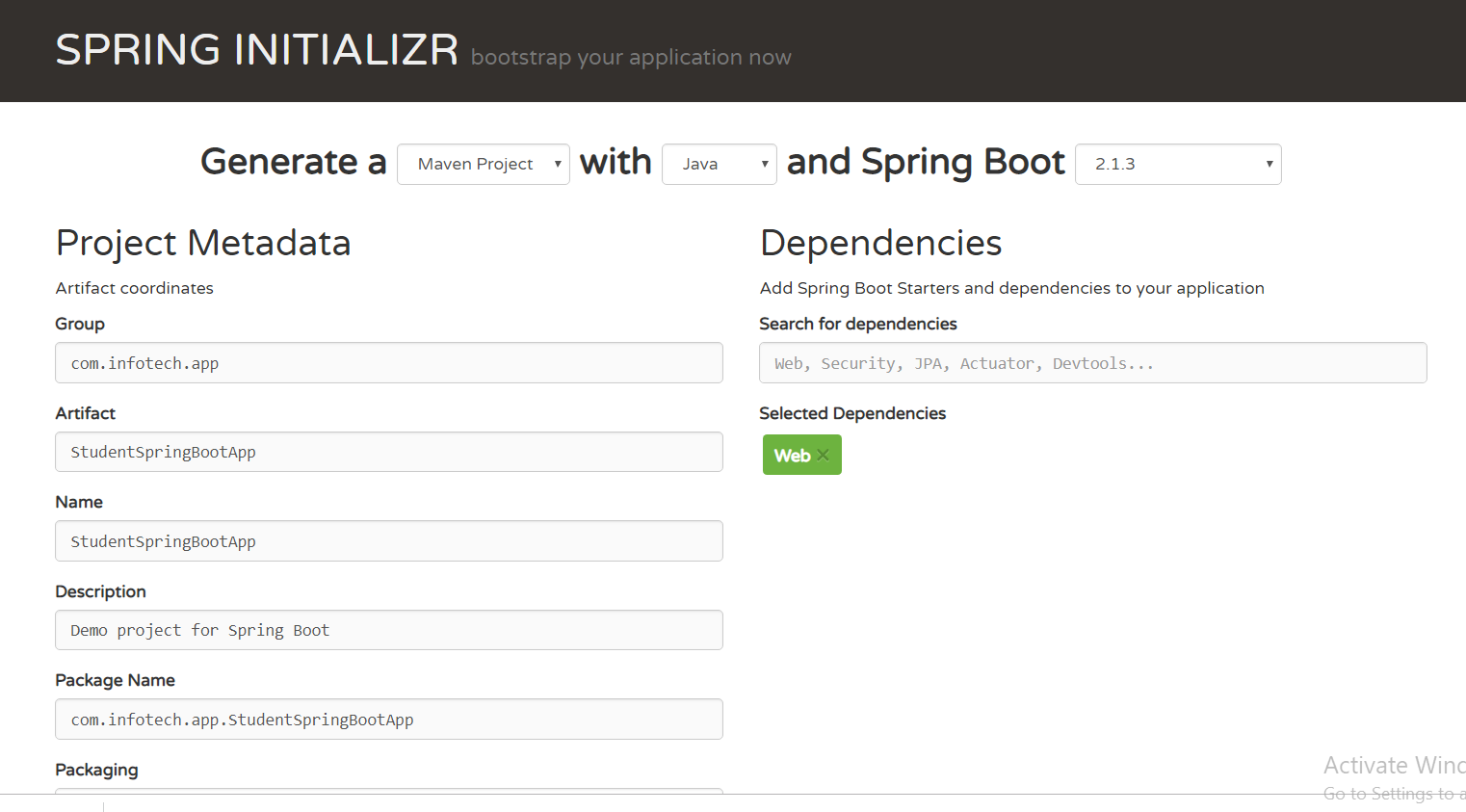




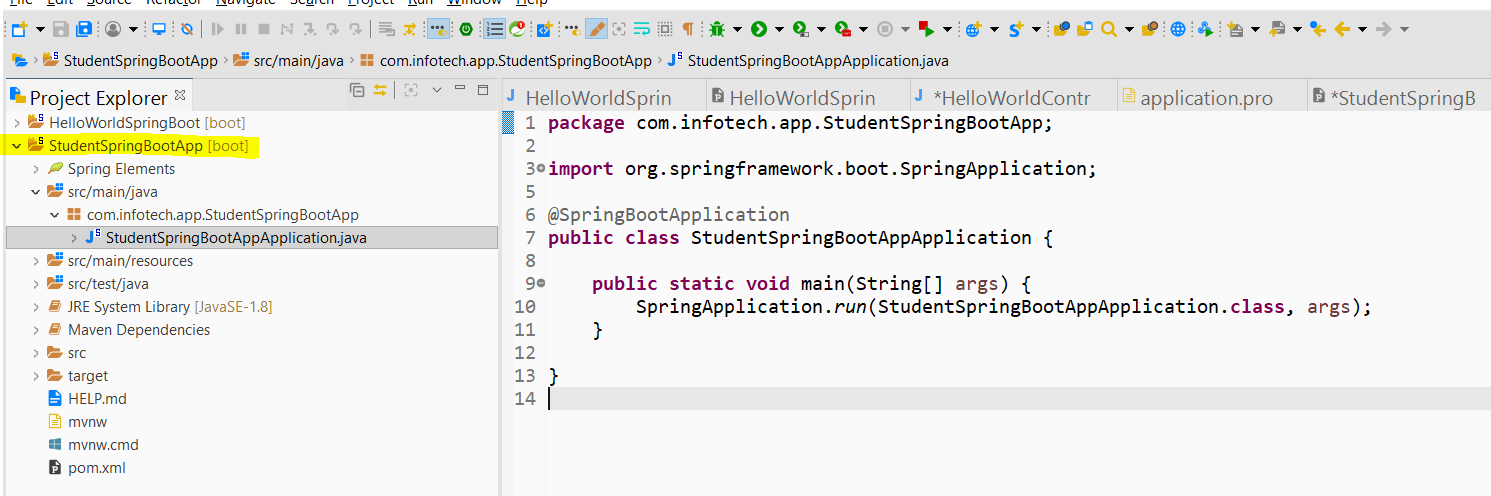
So far we have created the project manually but we can create spring boot project online as well as using STS.

1. **Using online initializer (start.spring.io)**





* Here we can add all the required dependencies online. Like first approach we don’t need to add any dependencies manually.
* As soon as click on Generate Project, it satrt downloading the prject into zip format.
* Now extract the folder and paste the extracted folder into our main workspace.
* Now from the ecliple import ths project [Existing Maven Project].
* In this project we will have one main java file with @SpringBootApplication annotation.
* So now we just need our controller class and need to write our business logic.

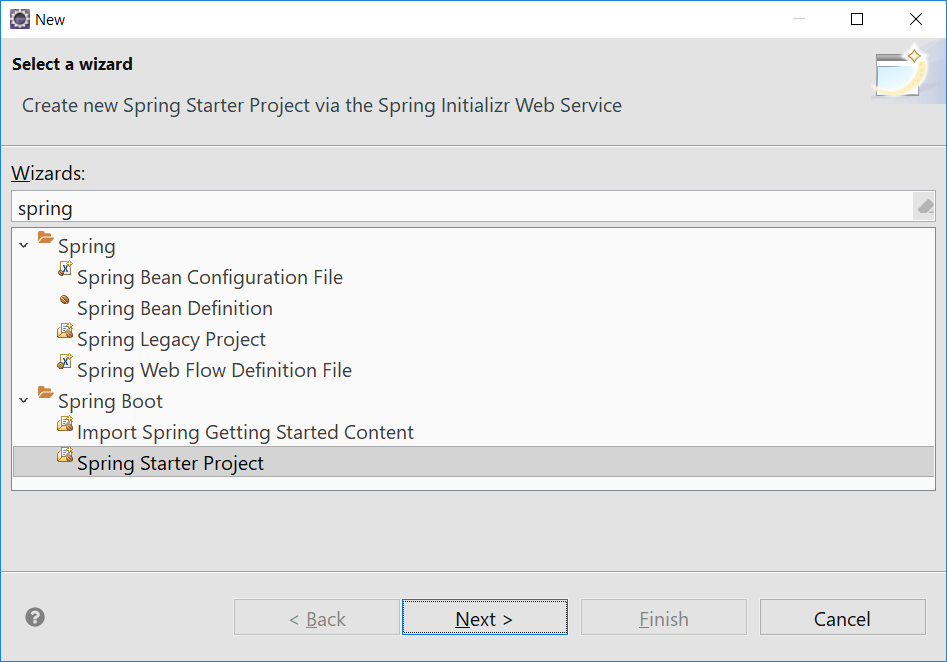


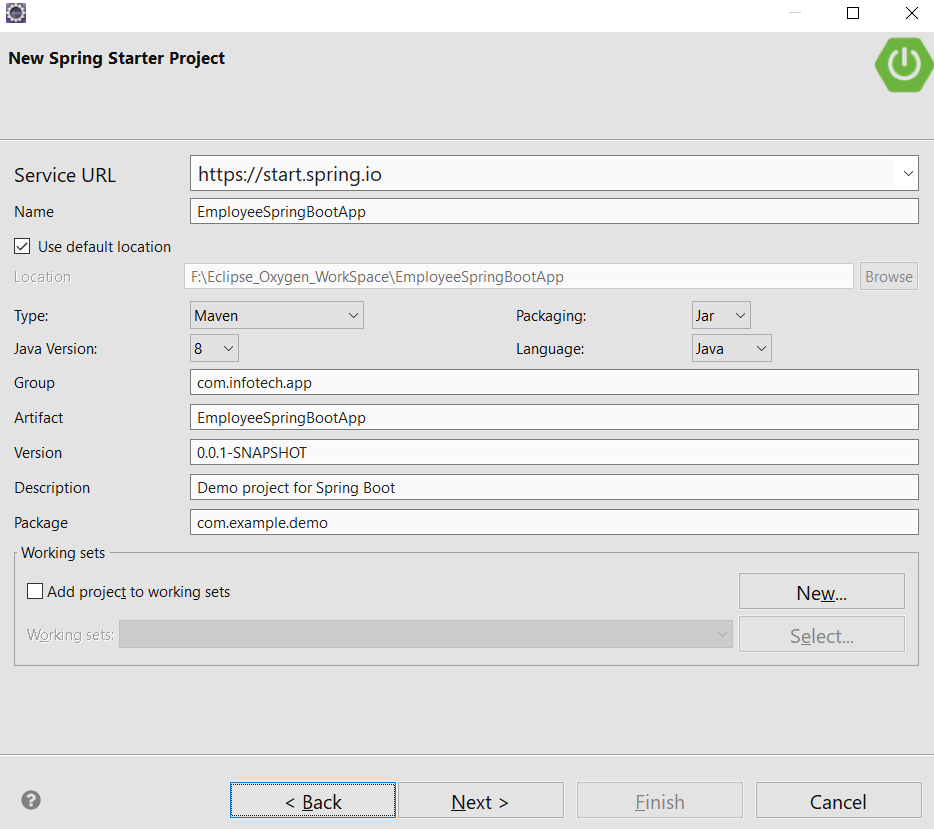
|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <projectxmlns=*"http://maven.apache.org/POM/4.0.0"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.3.RELEASE</version>  <relativePath/><!-- lookup parent from repository -->  </parent>  <groupId>com.infotech.app</groupId>  <artifactId>StudentSpringBootApp</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>StudentSpringBootApp</name>  <description>Demo project for Spring Boot</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> |

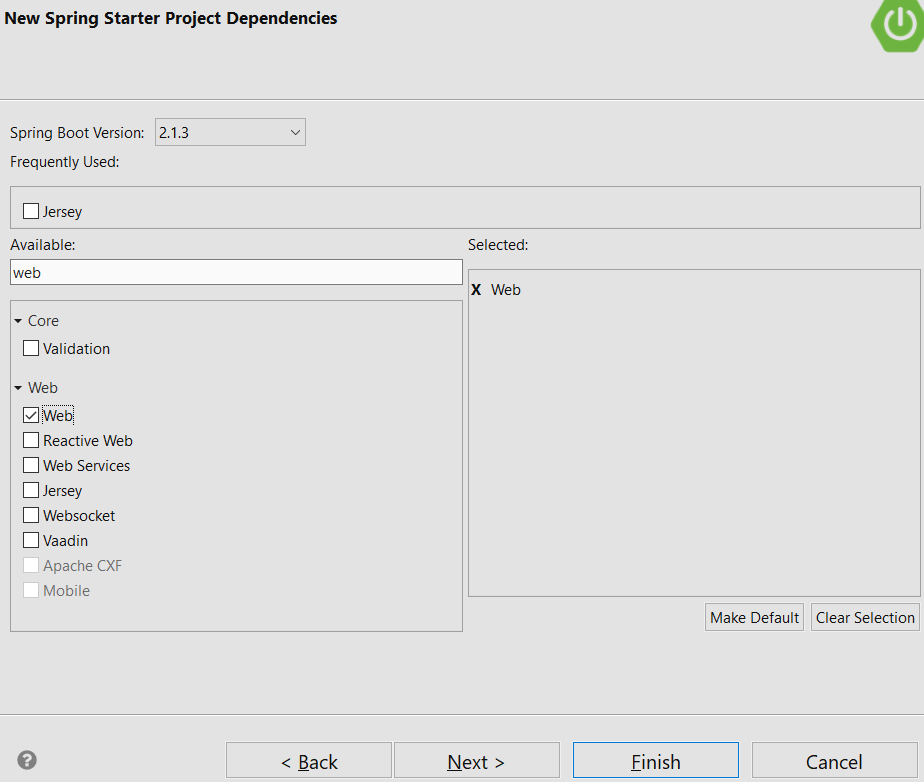
While creting online with one spring web dependecies then we got this pom.xml file. Here we got one dependencies to test the project[spring-boot-starter-test].

**Now we can create the spring boot project with Spring tool suit.**

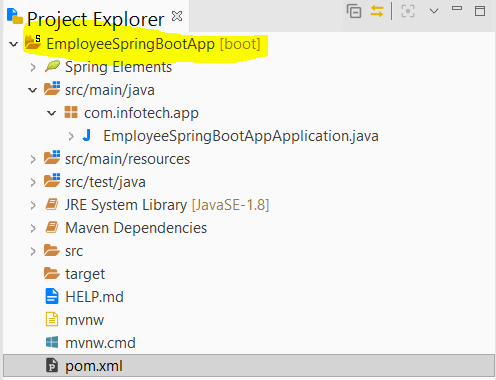
1. We can directly down load STS which is developed on eclipse. And using this STS tool we can directly create spring boot project.
2. Down load any eclipse version and install STS from the market place of the eclipse.
3. Now go to file🡺 new 🡺 Spring Starter Project







**Click Finish:**



|  |
| --- |
| package com.infotech.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class EmployeeSpringBootAppApplication {  public static void main(String[] args) {  SpringApplication.run(EmployeeSpringBootAppApplication.class, args);  }  } |

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <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>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.3.RELEASE</version>  <relativePath/><!-- lookup parent from repository -->  </parent>  <groupId>com.infotech.app</groupId>  <artifactId>EmployeeSpringBootApp</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>EmployeeSpringBootApp</name>  <description>Demo project for Spring Boot</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> |

So as we can see that there is no difference between Projects created through online and using Spring STS tool

**Spring Boot CLI:**

**Spring Boot CLI Setup**

**Step 1:** To create Spring Boot Eclipse project first download **Apache Maven** and **Spring Boot CLI** from their respective official sites.

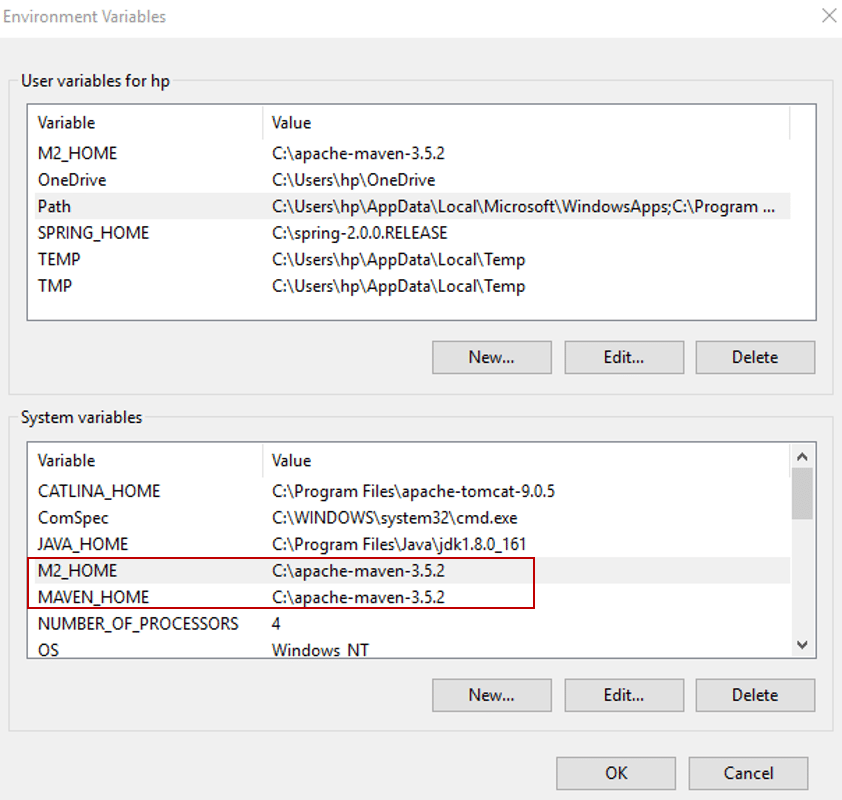
**Step 2:** **Download Apache Maven**

**Step 2.1:** Visit Maven official website and download the [Maven zip](http://www-eu.apache.org/dist/maven/maven-3/3.5.2/binaries/) file, for example, **apache-maven-3.5.2-bin.zip**.

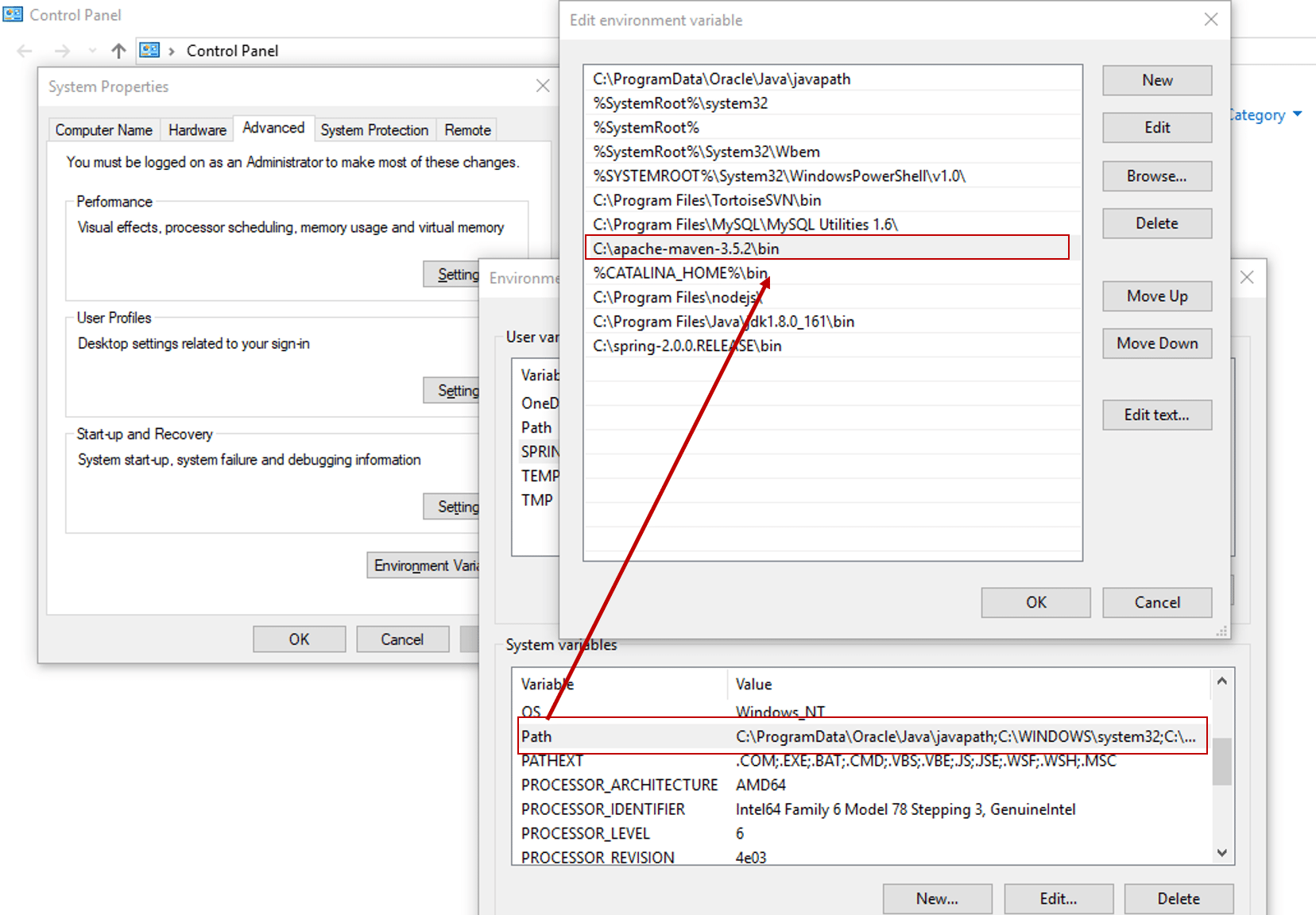
**Step 2.2:** Unzip it to the folder you want to install Maven.

Assume that you unzip to this directory C:\apache-maven-3.5.2.

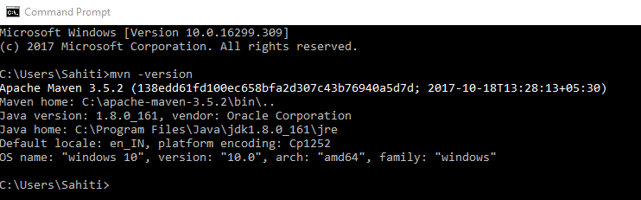
**Step 2.3:** Add**M2\_HOME** and **MAVEN\_HOME** variables in the Windows environment, and point it to your Maven folder.



**Step 2.4:** Update the **PATH** variable, and append the Maven bin folder –C:\apache-maven-3.5.2\bin.Refer to the snapshot below.



**Step2.5:** Verify whether maven is installed or not, with **mvn -version** command in the command prompt.



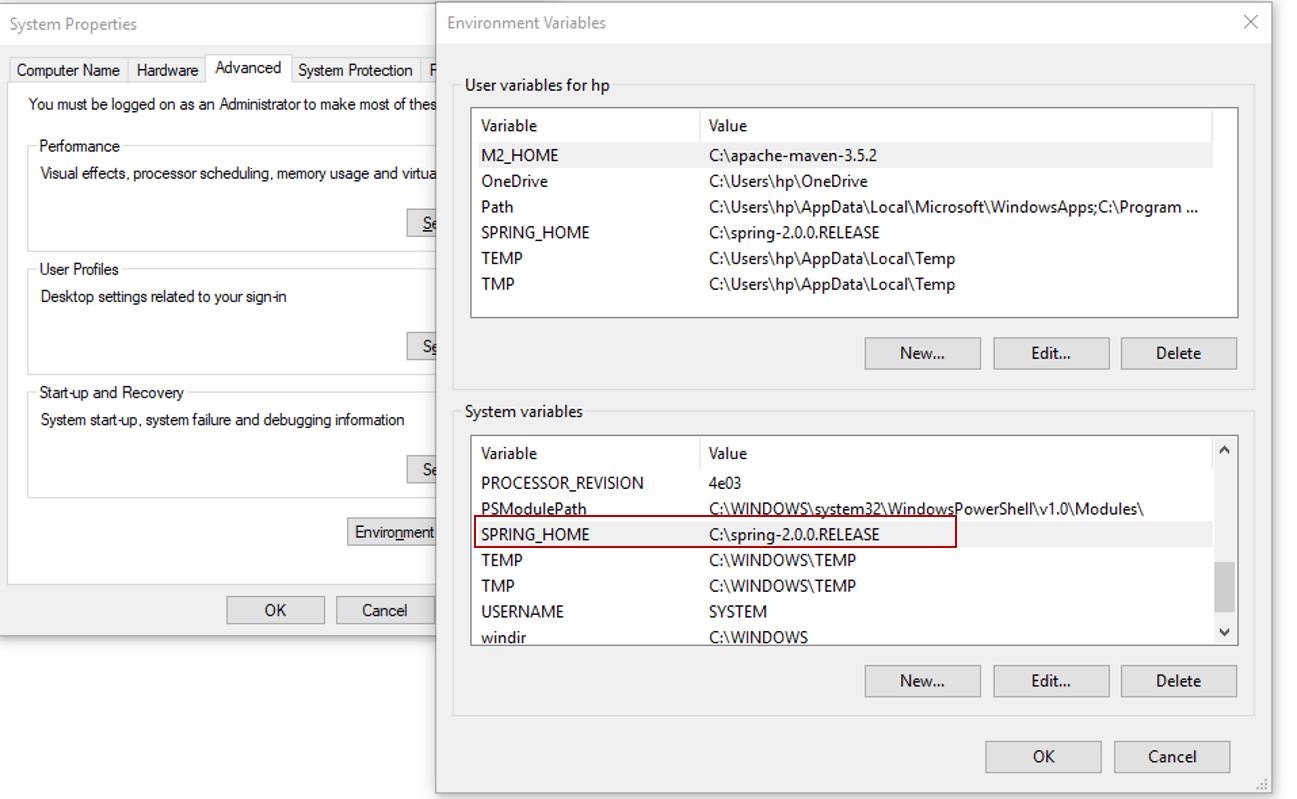
**Step 3:** Download the Spring Boot CLI tool

**Step 3.1:** Download [Spring Boot CLI](https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/) from its official website.

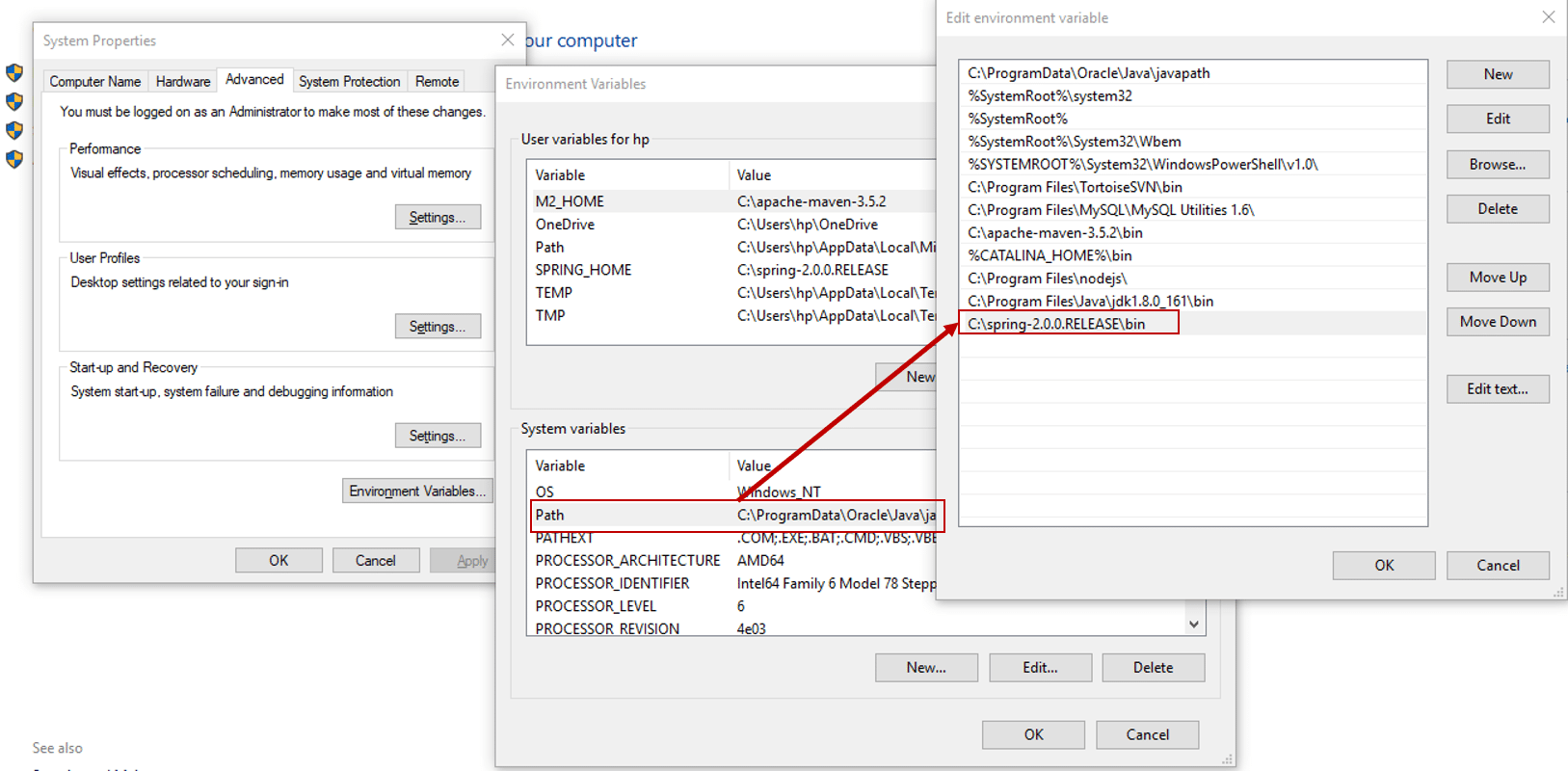
**Step 3.2:** Unzip the downloaded file the folder you want to install Spring CLI.

Assume that you unzip it to this directory C:\spring-2.0.0.RELEASE.

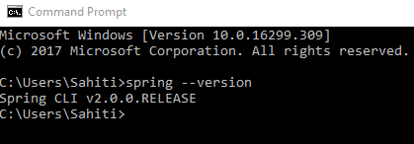
**Step 3.3:** Add **SPRING\_HOME** variable in the Windows environment, and point it to your spring folder. Refer to the snapshot below.



**Step 3.4:** Update the **PATH** variable, and append the Spring bin folder – C:\spring-2.0.0.RELEASE\bin. Refer to the snapshot below.

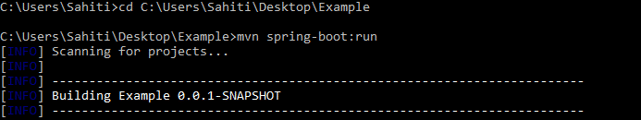


**Step 3.5:** Verify whether Spring is installed or not, with **spring –version** command in the command prompt.

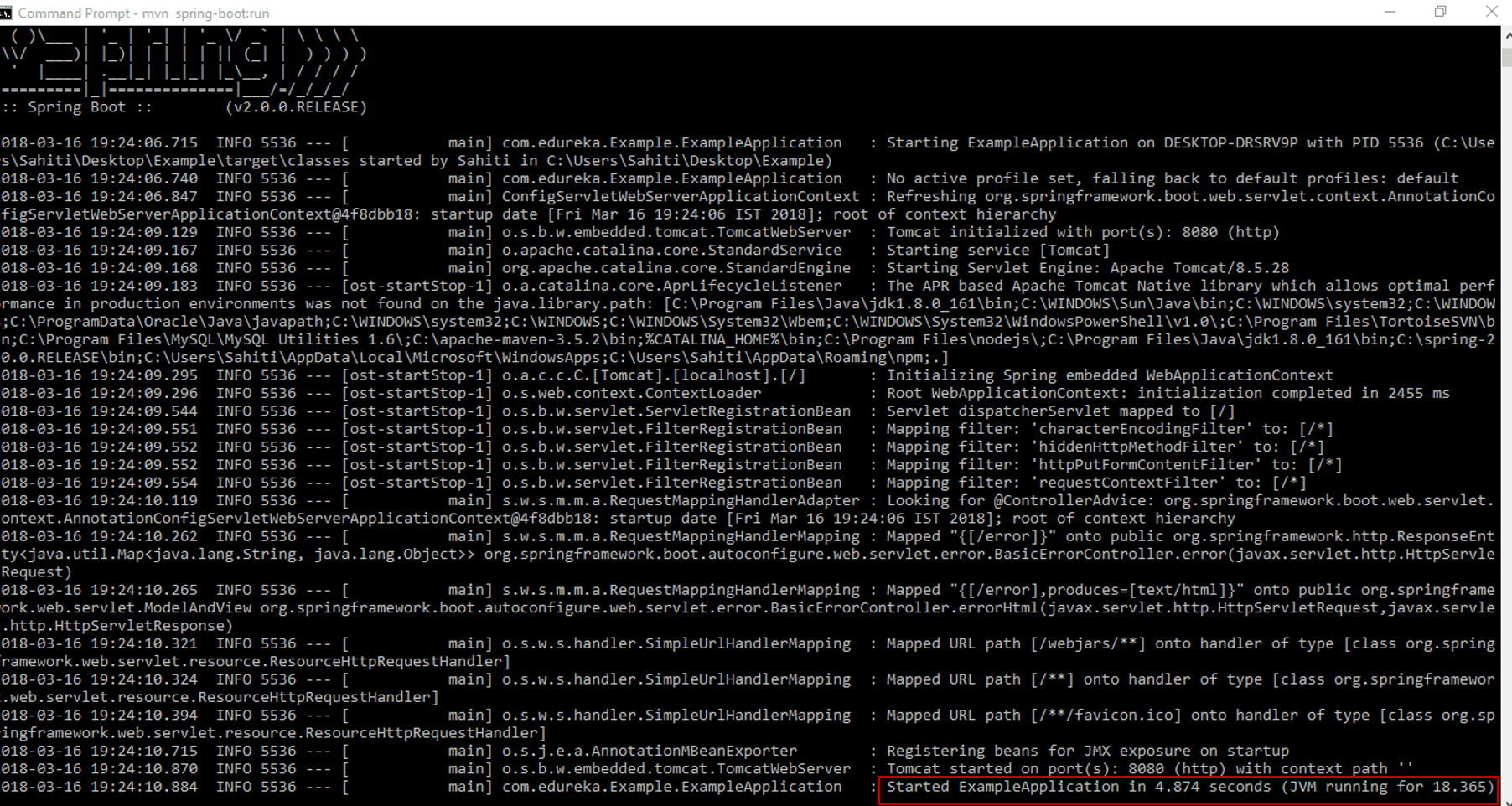


**Step 4:** Now go to your project directory where you have created the spring project and copy the path.

**Step 5:** Change the working directory to the project path on the command prompt using the command**cd**. Assume here the path is C:\Users\Sahiti\Desktop\Example.



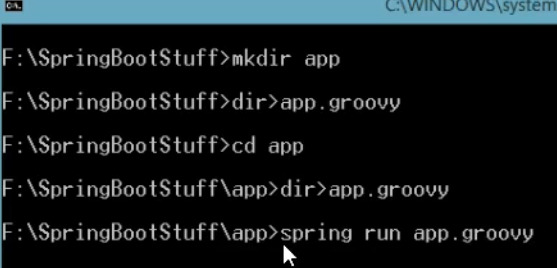
**Step 6:** Run the project using the command **mvn spring-boot:run**



**While creating spring boot application using CLI then:** Groovy command pays a very important role. Here in this case we don’t need to configure anything, we don’t need to import anything and we don’t need to mention return in the class file. Everything will be taken care spring boot only.

In the app.goovy file we just need to mention fallowing code.

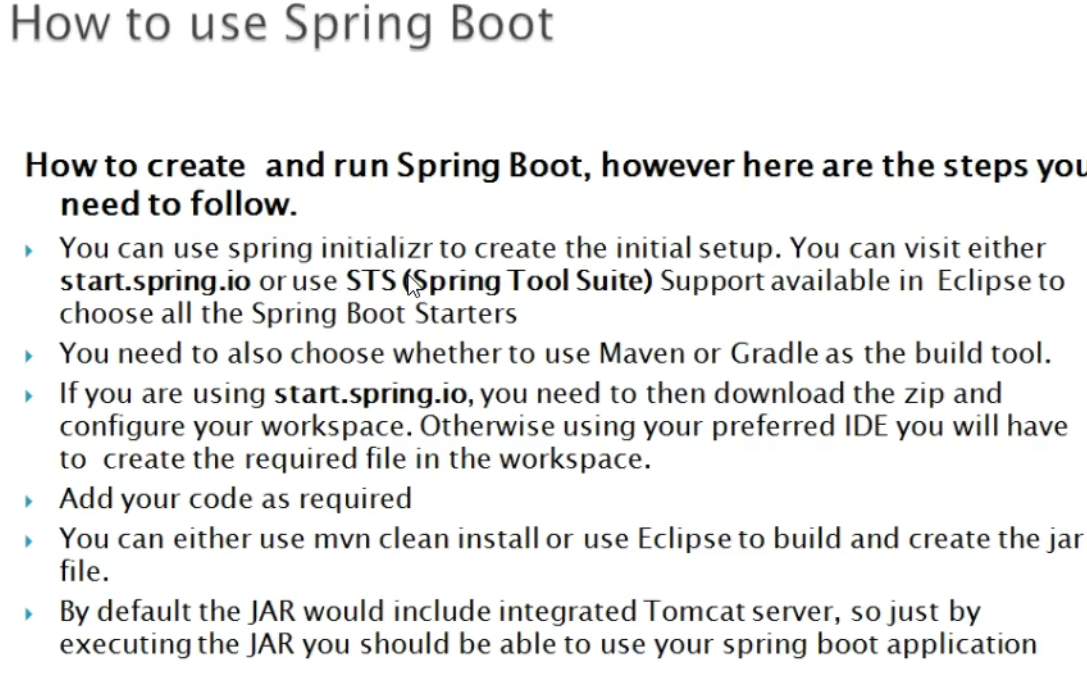
|  |
| --- |
| @RestController  **publicclass** HelloWorldController {    @GetMapping(value="/hello")  @RequestMapping(value="/hello", method =RequestMethod.***GET***)  **public** String sayHello() {  "Hello World!!";  }  } |

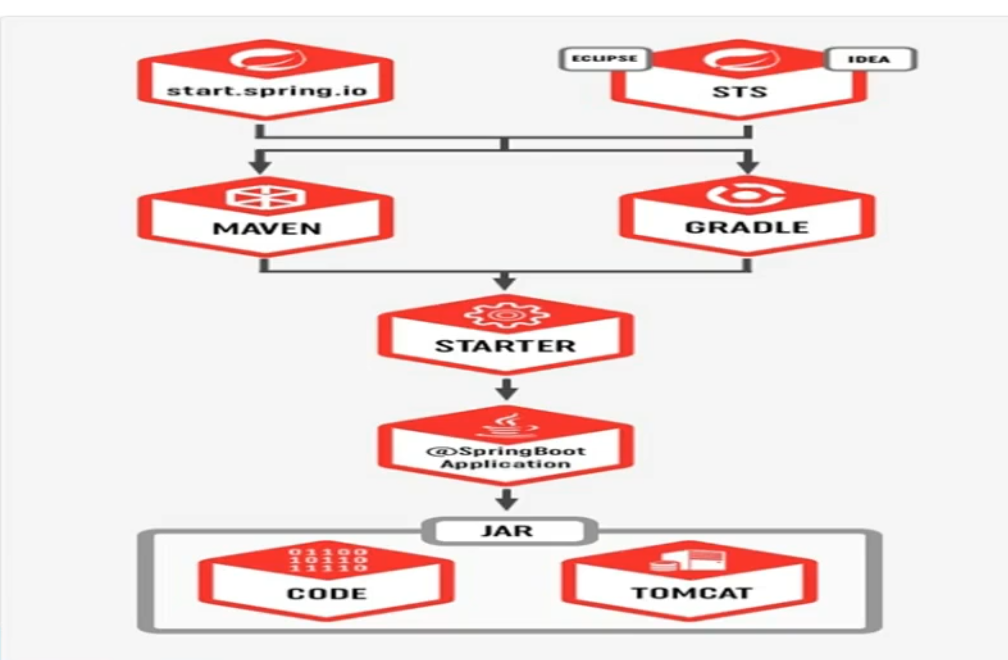


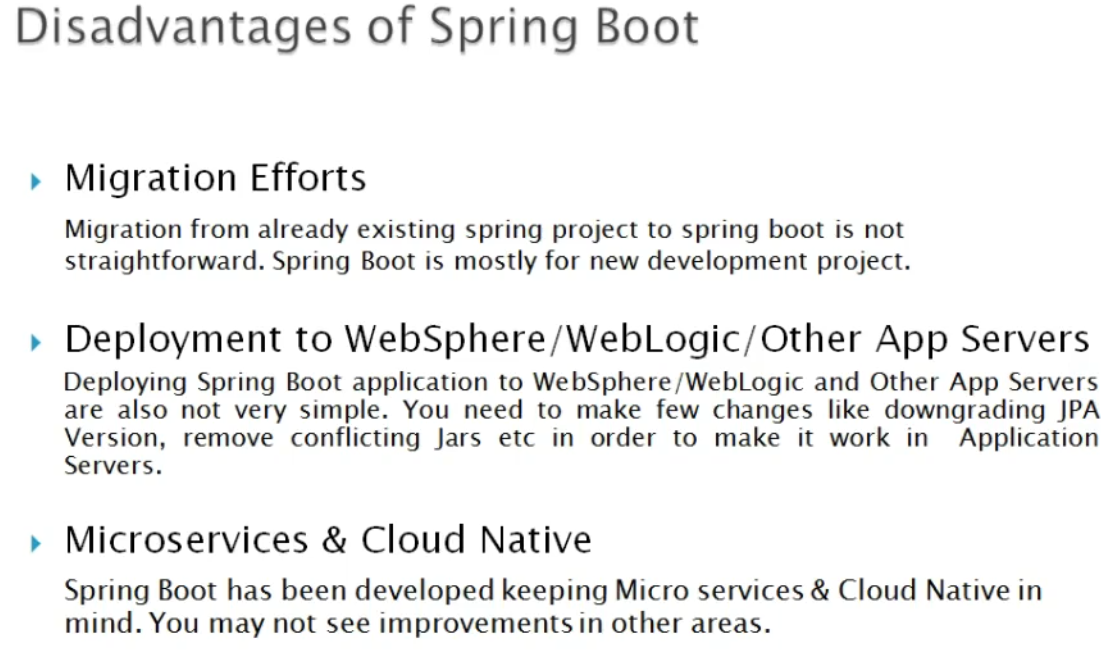
To run the HelloWorldController class we just need to run last commend mentioned in the screen shot.

* Here in this case we are using groovy language to develop the spring application
* We don’t need any main method as starter for spring boot application.

**Advantages and Disadvantages of Spring Boot:**

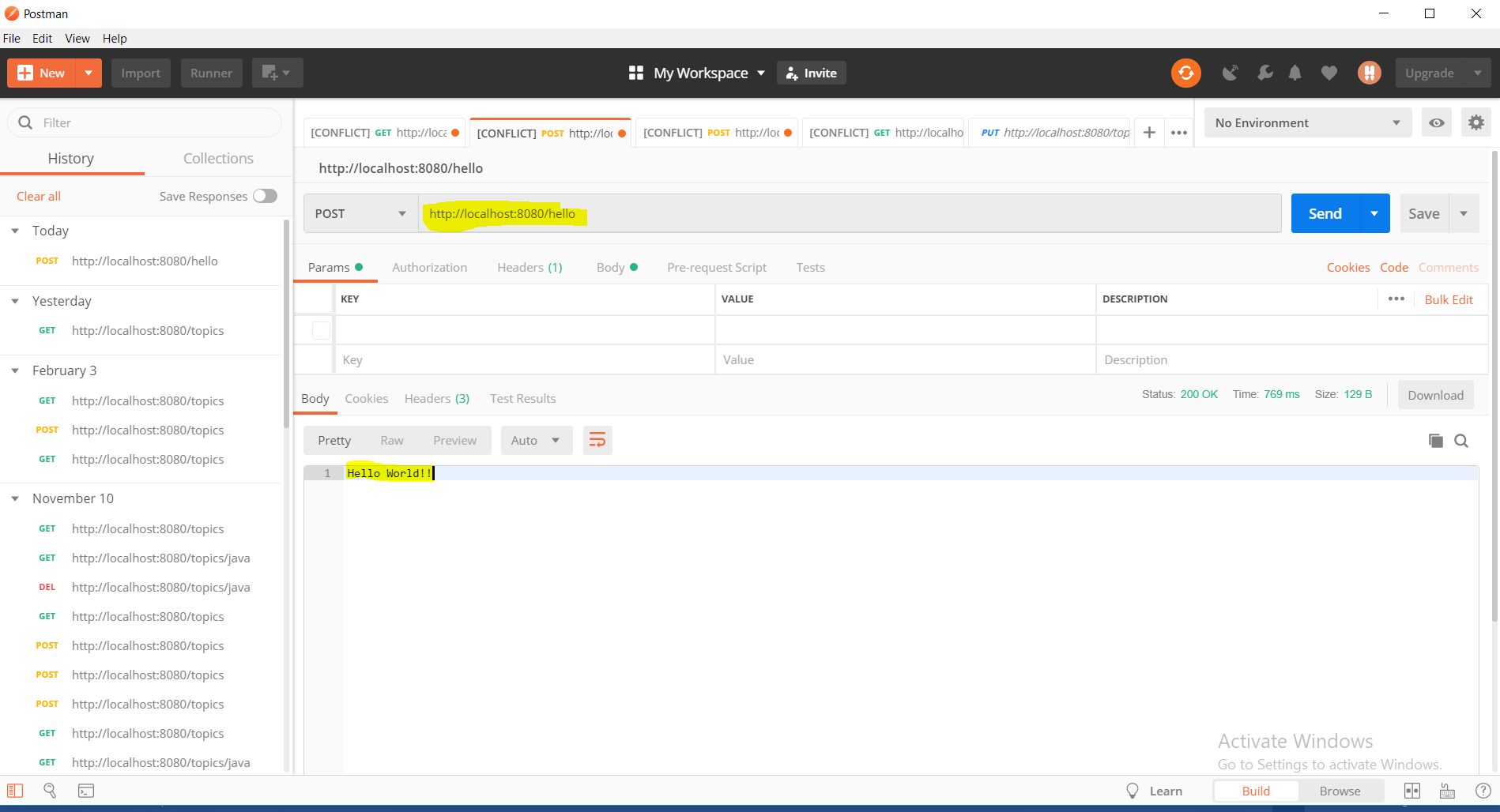






**Installing and setting up Postman in chrome:**

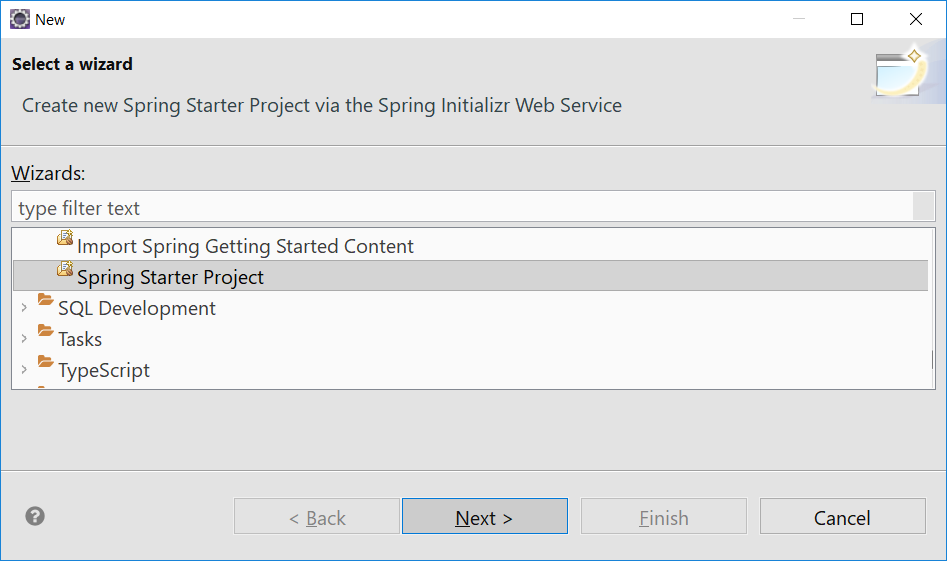
* Postman is a client software through which we can send the request to the Restful service application.
* As we know that when we create any RESTFul application then we used to deploy onto application server and whenever we need to expose the particular server then we use service url to get output in the form of xml or json.
* In RESTFul application generally we used to CRUD operation. And for this CRUD operation we must have some kind of UI.
* But we can do CRUD operation with RESTFul application without UI also. For example in case of SOAP base server to do CRUD operation we use SOAP UI to test the service but SOAP UI is heavy weight.
* Another way is we can use client software POSTMAN which is light weight software. Using POSTMAN we can send the request to RESTFul application and can do all the CRUD operations.

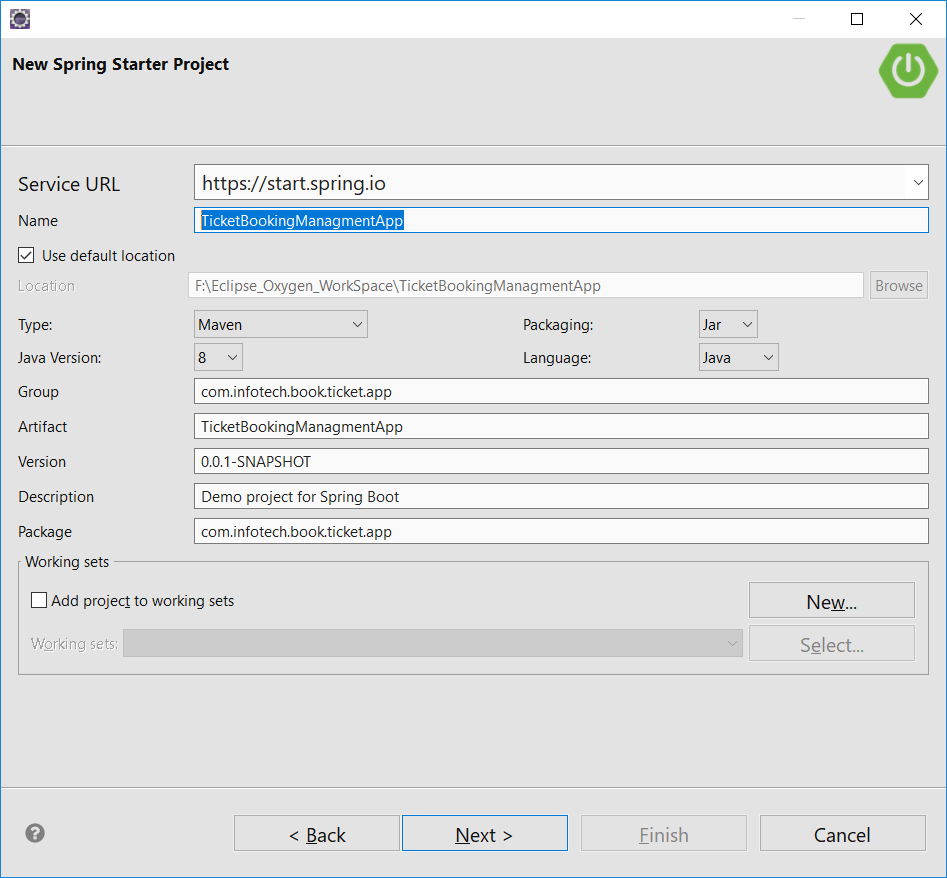


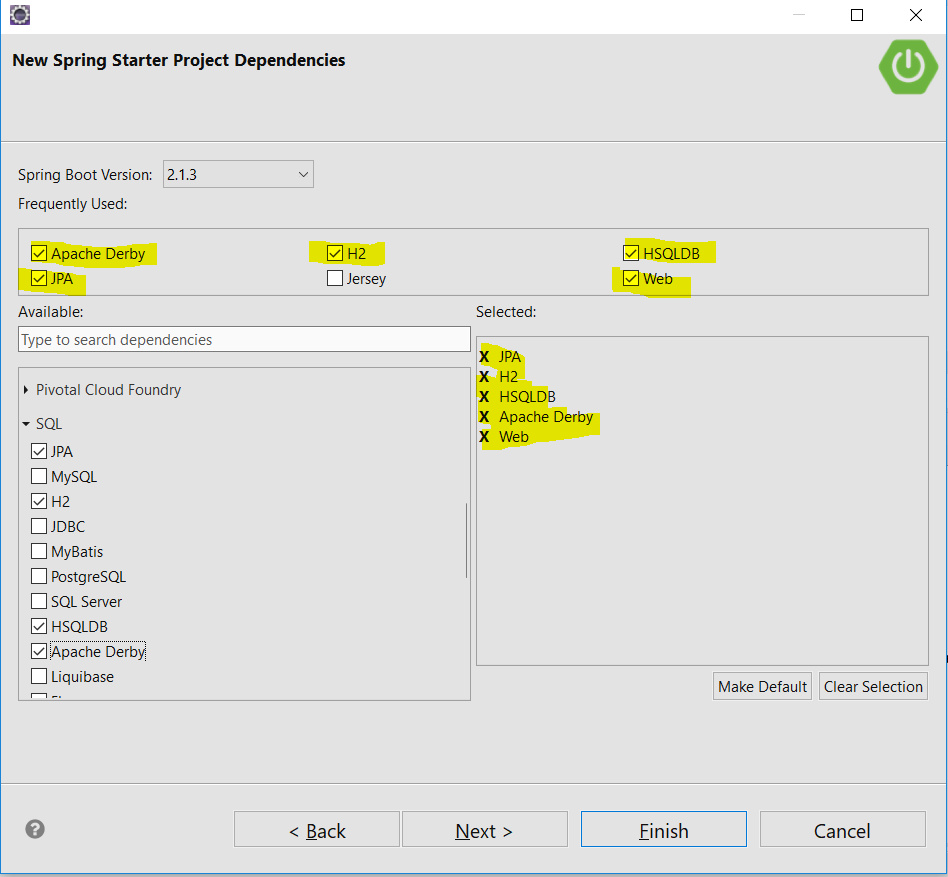
**CRUD Operation in RESTFul application using Spring Boot framework:**

Like when we interacting with DATABASE then we have to do several CRUD operation using queries. Similarly in RESTFul Application we do the CRUD operation using HTTP methods (POST,GET, PUT, and DELETE). Using these method we can CREATE, READ, UPDATE and DELETE resources on the server.

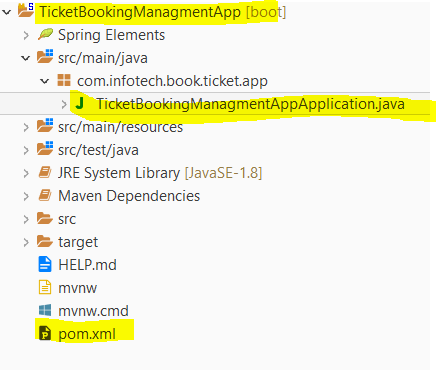
**Let’s create one TicketBookingManagmentApp Project.**







Select the dependencies from the screen above mentioned. Click Finish



|  |
| --- |
| package com.infotech.book.ticket.app;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class TicketBookingManagmentAppApplication {  public static void main(String[] args) {  SpringApplication.run(TicketBookingManagmentAppApplication.class, args);  }} |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <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>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.3.RELEASE</version>  <relativePath/><!-- lookup parent from repository -->  </parent>  <groupId>com.infotech.book.ticket.app</groupId>  <artifactId>TicketBookingManagmentApp</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>TicketBookingManagmentApp</name>  <description>Demo project for Spring Boot</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>com.h2database</groupId>  <artifactId>h2</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.apache.derby</groupId>  <artifactId>derby</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.hsqldb</groupId>  <artifactId>hsqldb</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>    <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

Now let’s create

1. Create one entity class **Ticket** having some properties and getter and setters. And annotate this class @Entity
2. Create one service class **TicketBookingService** and annotate this class with @Service.
3. Create one DAO interface **TicketBookingDAO** and annotate this class with @Repository.
4. Finally let’s create one Controller class**TicketBookingController** and annotate this class with @RestController.
5. Now from the controller class let’s start writing functionality.
6. Create one method CreateTicket inside the controller class. Since this controller class is responsible to call the service method. So let’s create **TicketBookingService**reference and annotate it with @Autowired.
7. Similarly **TicketBookingService**class is responsible to call DAO class method for CRUD operation. So let’s create **TicketBookingDAO**class reference inside **TicketBookingService**and annotate it with @Autowired annotation.
8. Now let’s start code writing for CRUD operation from controller class.
9. For Creating resource:

List of Annotation used so far:

|  |  |
| --- | --- |
| @**RestController** |  |
| @**RequestMapping**(value="/api/tickets") |  |
| @**Autowired** |  |
| @**RequestMapping**(value="/create", method =RequestMethod.POST)or |  |
| @**PostMapping**(value="/create") |  |
| @**RequestMapping**(value="/ticket/{ticketId}", method =RequestMethod.GET) or |  |
| @**GetMapping**(value="/ticket/{ticketId}") |  |
| **@RequestBody**Ticket ticket |  |
| @**PathVariable**("ticketId") Integer ticketId |  |

|  |  |
| --- | --- |
| **package**com.infotech.book.ticket.app.entities;  **import** java.util.Date;  **import**javax.persistence.Column;  **import** javax.persistence.Entity;  **import**javax.persistence.GeneratedValue;  **import**javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import**javax.persistence.Table;  @Entity  @Table(name="ticket\_table")  **publicclassTicket** {  @Id  @GeneratedValue(strategy=GenerationType.***AUTO***)  @Column(name="ticket\_id")  **private** Integer ticket\_id;  @Column(name="passenger\_name", nullable=**false**)  **private** String passenger\_name;  @Column(name="source\_station")  **private** String sourceStation;  @Column(name="dest\_station")  **private** String destination;  @Column(name="email")  **private** String email;  @Column(name="booking\_date")  **private** Date bookingDate;  @Column(name="mobile\_num")  **private** String mobileNum;  //Getters and setters  **public** Integer getTicket\_id() {  **return**ticket\_id;  } | **publicvoid**setTicket\_id(Integer ticket\_id) {  **this**.ticket\_id = ticket\_id;  }  **public** String getPassenger\_name() {  **return**passenger\_name;  }  **publicvoid**setPassenger\_name(String passenger\_name) {  **this**.passenger\_name = passenger\_name;  }  **public** String getSourceStation() {  **return**sourceStation;  }  **publicvoid**setSourceStation(String sourceStation) {  **this**.sourceStation = sourceStation;  }  **public** String getDestination() {  **return**destination;  }  **publicvoid**setDestination(String destination) {  **this**.destination = destination;  }  **public** String getEmail() {  **return**email;  }  **publicvoid**setEmail(String email) {  **this**.email = email;  }  **public** Date getBookingDate() {  **return**bookingDate;  }  **publicvoid**setBookingDate(Date bookingDate) {  **this**.bookingDate = bookingDate;  }  **public**StringgetMobileNum() {  **return**mobileNum;  }  **publicvoid**setMobileNum(String string) {  **this**.mobileNum = string;  }  } |

|  |
| --- |
| **------------------------------TicketBookingControllerClass**--------------------------------  **package**com.infotech.book.ticket.app.controller;  **import**java.util.Optional;  **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.RequestMethod;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.infotech.book.ticket.app.entities.Ticket;  **import** com.infotech.book.ticket.app.service.TicketBookingService;  @RestController  @RequestMapping(value="/api/tickets")  **publicclassTicketBookingController** {  @Autowired  **private** TicketBookingService ticketBookingService;    //@RequestMapping(value="/create", method =RequestMethod.POST) or we can use @PostMapping(value="/create")  @PostMapping(value="/create")  **public** Ticket createTicket(@RequestBody Ticket ticket) {  System.***out***.println("Insert into CreatTicket method");  **return**ticketBookingService.createTicket(ticket);  }  // Creating method to read  //@RequestMapping(value="/ticket/{ticketId}", method =RequestMethod.GET) or we can use @GetMapping(value="/ticket/{ticketId}")  @GetMapping(value="/ticket/{ticketId}")  **public** Ticket getTicketById(@PathVariable("ticketId") Integer ticketId) {    **return**ticketBookingService.getTicketById(ticketId);  }  @GetMapping(value="/ticket/alltickets")  **public** Iterable<Ticket>getAllBookedTickets(){    **return**ticketBookingService.getAllBookedTickets();  }  @DeleteMapping(value="/ticket/{ticketId}")  **publicvoid**deleteTicketById(@PathVariable("ticketId") Integer ticketId) {  ticketBookingService.deleteTicket(ticketId);  }  @PutMapping(value="/ticket/{ticketId}/{newEmail:.+}/{mobileNum}")  **public** Ticket updateTicket(@PathVariable("ticketId") Integer ticketId,  @PathVariable("newEmail") String newEmail,  @PathVariable("mobileNum") String mobileNum) {  **return**ticketBookingService.updateTicket(ticketId,newEmail,mobileNum);  }  } |

|  |
| --- |
| **------------------------------TicketBookingDAO Class**---------------------------------------  **package**com.infotech.book.ticket.app.service;  **import** org.springframework.beans.factory.annotation.Autowired;  **import**org.springframework.stereotype.Service;  **import**com.infotech.book.ticket.app.dao.TicketBookingDAO;  **import** com.infotech.book.ticket.app.entities.Ticket;  @Service  **publicclass** TicketBookingService {  @Autowired  **private** TicketBookingDAO ticketBookingDao;    **public** Ticket createTicket(Ticket ticket) {  System.***out***.println("Insert into service class");  **return**ticketBookingDao.save(ticket);  }  **public** Ticket getTicketById(Integer ticketId) {    **return**ticketBookingDao.findOne(ticketId);  }  **public** Iterable<Ticket>getAllBookedTickets() {  // **TODO** Auto-generated method stub  **return**ticketBookingDao.findAll();  }  **publicvoid**deleteTicket(Integer ticketId) {    ticketBookingDao.delete(ticketId);  }  **public** Ticket updateTicket(Integer ticketId, String newEmail, String mobileNumber) {  Ticket ticketFromDb = ticketBookingDao.findOne(ticketId);  ticketFromDb.setEmail(newEmail);  ticketFromDb.setMobileNum(mobileNumber);  Ticket updatedTicket=ticketBookingDao.save(ticketFromDb);  **return**updatedTicket;    }  } |

|  |
| --- |
| **------------------------------TicketBookingDAOClass**---------------------------------------  package com.infotech.book.ticket.app.dao;  import org.springframework.data.repository.CrudRepository;  import org.springframework.stereotype.Repository;  import com.infotech.book.ticket.app.entities.Ticket;  @Repository  public interface **TicketBookingDAO** extends CrudRepository<Ticket, Integer> {  } |

|  |
| --- |
| **------------------------------TicketBookingDAOClass**---------------------------------------  package com.infotech.book.ticket.app;  import java.util.Date;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.context.ConfigurableApplicationContext;  import com.infotech.book.ticket.app.entities.Ticket;  import com.infotech.book.ticket.app.service.TicketBookingService;  @SpringBootApplication  public class TicketBookingManagmentAppApplication {  public static void main(String[] args) {    ConfigurableApplicationContext applicationCtx= SpringApplication.run(TicketBookingManagmentAppApplication.class, args);  TicketBookingService ticketBookingService =applicationCtx.getBean("ticketBookingService",TicketBookingService.class);  Ticket ticket= new Ticket();    ticket.setPassenger\_name("Arun");  ticket.setBookingDate(new Date());  ticket.setSourceStation("Bangalore");  ticket.setDestination("Allahabad");  ticket.setEmail("arunk.gupta@tcs.com");  ticket.setMobileNum("9900541580");    ticketBookingService.createTicket(ticket);  }  } |

|  |
| --- |
| **------------------------------application.properties**---------------------------------------  #PROFILES  spring.profiles.active=dev  # JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)  spring.jpa.generate-ddl=true  spring.jpa.hibernate.ddl-auto=update  spring.jpa.database=default  spring.jpa.show-sql=true  # DATASOURCE (DataSourceAutoConfiguration&DataSourceProperties)  spring.datasource.continue-on-error=false  spring.datasource.generate-unique-name=false |

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <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>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.19.RELEASE</version>  <relativePath/><!-- lookup parent from repository -->  </parent>  <groupId>com.infotech.book.ticket.app</groupId>  <artifactId>TicketBookingManagmentApp</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>TicketBookingManagmentApp</name>  <description>Demo project for Spring Boot</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.apache.derby</groupId>  <artifactId>derby</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

**Explanation**: Pending:

**Implementation of CommandLinerRunner Interface with spring main method**

|  |
| --- |
| package com.infotech.book.ticket.app;  import java.util.Date;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.context.ConfigurableApplicationContext;  import com.infotech.book.ticket.app.entities.Ticket;  import com.infotech.book.ticket.app.service.TicketBookingService;  @SpringBootApplication  public class TicketBookingManagmentAppApplication {  public static void main(String[] args) {    ConfigurableApplicationContext **applicationCtx**= SpringApplication.run(TicketBookingManagmentAppApplication.class, args);  TicketBookingService **ticketBookingService** =applicationCtx.getBean("ticketBookingService",TicketBookingService.class);  Ticket ticket= new Ticket();  ticket.setPassenger\_name("Arun");  ticket.setBookingDate(new Date());  ticket.setSourceStation("Bangalore");  ticket.setDestination("Allahabad");  ticket.setEmail("arunk.gupta@tcs.com");  ticket.setMobileNum("9900541580");  ticketBookingService.createTicket(ticket);  }  } |

1. In the above class first we are creating an object **ConfigurableApplicationContext**by passing Main class and command line argument.
2. Next using ConfigurableApplicationContext class object (**applicationCtx)** we are creating object of Service class TicketBookingService.
3. Next using Service class object (**ticketBookingService)** we are calling method CreateTicket (ticket).

We can reduce these line of code just by implementing CommandLinerRunner Interface in the main method. As show in example

1. **Implements** CommandLineRunner interface in the main method.
2. @Override **publicvoid** run(String... arg0) **throws** Exception and inside this method we have placed the code which sets the data in entity class and calling service class method createTicket().
3. @Autowired**private** TicketBookingService ticketBookingService; Hence it will create one association in the main class and spring will instantiate TicketBookingService class and put in the container. It is very equal to line mention below.

|  |
| --- |
| ConfigurableApplicationContext **applicationCtx**= SpringApplication.run(TicketBookingManagmentAppApplication.class, args);  TicketBookingService **ticketBookingService** =applicationCtx.getBean("ticketBookingService",TicketBookingService.class); |

**Hence** we can reduce these two line just by implementing Autowiring the Service class and we can do refactoring the earlier main class.

**Note** the run method will execute automatically by the spring boot.

|  |
| --- |
| **package** com.infotech.book.ticket.app;  **import** java.util.Date;  **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.context.ConfigurableApplicationContext;  **import** com.infotech.book.ticket.app.entities.Ticket;  **import** com.infotech.book.ticket.app.service.TicketBookingService;  @SpringBootApplication  **publicclass** TicketBookingManagmentAppApplication **implements** CommandLineRunner {    @Autowired  **private** TicketBookingService ticketBookingService;  **publicstaticvoid** main(String[] args) {    SpringApplication.*run*(TicketBookingManagmentAppApplication.**class**, args);    /\*  ConfigurableApplicationContext applicationCtx= SpringApplication.run(TicketBookingManagmentAppApplication.class, args);  TicketBookingService ticketBookingService=applicationCtx.getBean("ticketBookingService",TicketBookingService.class);  \*/  }  @Override  **publicvoid** run(String... arg0) **throws** Exception {    Ticket ticket= **new** Ticket();  ticket.setPassenger\_name("Arun");  ticket.setBookingDate(**new** Date());  ticket.setSourceStation("Bangalore");  ticket.setDestination("Allahabad");  ticket.setEmail("arunk.gupta@tcs.com");  ticket.setMobileNum("9900541580");  ticketBookingService.createTicket(ticket);  }  } |

So far we have done CRUD operation with embedded (in-built) database derby.

We can identify the datasource name also used in the application.

1. Create one DataSource and annotated it with @Autowired :

@Autowired

Private DataSource datasource;

1. **import** javax.sql.DataSource;
2. And print datasource: you will get datasource information used in app as mentioned below

* Package Name: org.apache.tomcat.jdbc.pool.DataSource@5ca1f591
* DriverClassName=org.apache.derby.jdbc.EmbeddedDriver;
* Password=\*\*\*\*\*\*\*\*;
* URL=jdbc: derby: memory: testdb; create=true;
* Username=SA;
* DataSourceJNDI=null;

|  |
| --- |
| <dependency>  <groupId>org.apache.derby</groupId>  <artifactId>derby</artifactId>  <scope>runtime</scope>  </dependency> |
| <!--<dependency>  <groupId>com.h2database</groupId>  <artifactId>h2</artifactId>  <scope>runtime</scope>  </dependency> |
| <dependency>  <groupId>org.hsqldb</groupId>  <artifactId>hsqldb</artifactId>  <scope>runtime</scope>  </dependency> --> |

In the previous example we have done CRUD with Derby Embedded database. Now we do the same operation using **h2database**then we don’t need to do anything in the code level. The only thing that we need to do that we need to comment Derby and hsqldb and need to uncomment h2database.

Similarly in case of hsqldb also.

**Note** if we change the dependency in pom.xml and if we print the datasource then we will get all the information about used database in the application.

Hence we can used in-memory embedded database for our testing purpose. Because in this database data is available till the server runs. Once the server gets down all the data saved in embedded database is also disappeared.

The advantage of embedded database is that we don’t need to do any configuration. It is lightweight and fast and very helpful in fast development.

Note: Let’s if we are going to used external database then we have to mention all the things (driver name, URL, username, password) in the application.properities file.

**MySQL implementation:**