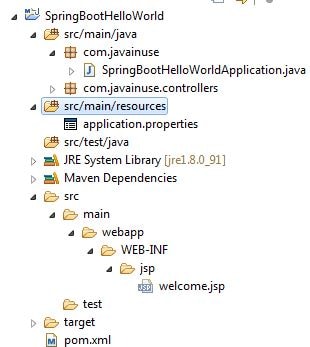
Spring Boot Hello World Application- Create simple controller and jsp view

In this post we create a spring boot application to display a simple welcome page.

Lets Begin-

Maven project will be as follows -   
  
  
Along with the Spring Boot dependencies we have also tomcat-embed-jasper and jstl jar dependencies as these required when jsp view is used. In the pom file add the following dependencies-

<?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.javainuse</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringBootHelloWorld</name>

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

<parent>

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

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

<version>1.4.1.RELEASE</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.apache.tomcat.embed</groupId>

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

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Create the SpringBootHelloWorldApplication.java as below-

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

@RequestMapping maps /welcome request to firstPage() method.

package com.javainuse.controllers;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.servlet.ModelAndView;

@Controller

public class TestController {

@RequestMapping("/welcome")

public ModelAndView firstPage() {

return new ModelAndView("welcome");

}

}

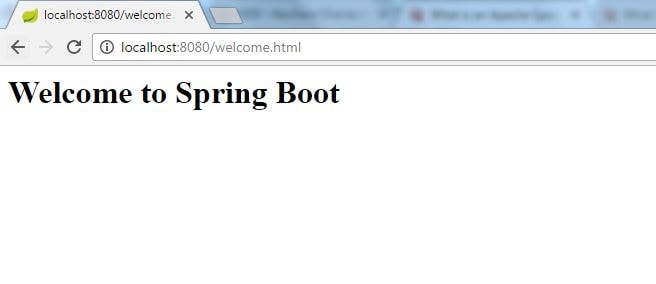
In the application.properties we define the prefix and suffix as follows. So in the above controller the /welcome.html correctly maps to /WEB-INF/jsp/welcome.jsp.

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

spring.mvc.view.suffix:.jsp

Last we define the welcome.jsp as below-

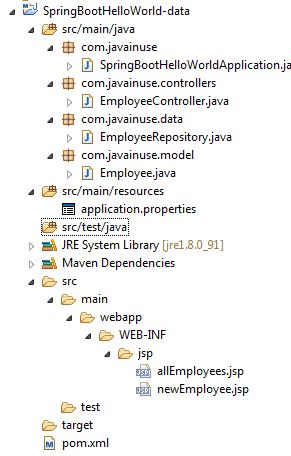
<h1>Welcome to Spring Boot</h1>

Compile and the run the SpringBootHelloWorldApplication.java as a Java application.  
Go to **localhost:8080/welcome**   


Spring Boot Tutorial-Spring Data JPA Simple Example

In this post we create a Employee management system which adds and retrieves employee details.

Lets Begin-

  
We make use of the h2 database. Maven will be as follows-

<?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.javainuse</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringBootHelloWorld</name>

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

<parent>

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

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

<version>1.4.1.RELEASE</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-data-jpa</artifactId>**

**</dependency>**

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Create the SpringBootHelloWorldApplication.java as follows-

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@RestController

@EnableAutoConfiguration

@SpringBootApplication

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

Create the Entity class as follows-

package com.javainuse.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee {

@GeneratedValue(strategy = GenerationType.AUTO)

@Id

private long id;

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDept() {

return dept;

}

public void setDept(String dept) {

this.dept = dept;

}

private String dept;

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", dept=" + dept + "]";

}

}

The Controller we define methods to add Employee record and display employee records as list. Define the controller as follows-

package com.javainuse.controllers;

import java.util.List;

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

import org.springframework.stereotype.Controller;

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

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

import org.springframework.web.servlet.ModelAndView;

import com.javainuse.data.EmployeeRepository;

import com.javainuse.model.Employee;

@Controller

public class EmployeeController {

@Autowired

private EmployeeRepository employeeData;

@RequestMapping(value = "/addNewEmployee.html", method = RequestMethod.POST)

public String newEmployee(Employee employee) {

employeeData.save(employee);

return ("redirect:/list.html");

}

@RequestMapping(value = "/addNewEmployee.html", method = RequestMethod.GET)

public ModelAndView addNewEmployee() {

Employee emp = new Employee();

return new ModelAndView("newEmployee", "form", emp);

}

@RequestMapping(value = "/listEmployees.html", method = RequestMethod.GET)

public ModelAndView employees() {

List<Employee> allEmployees = employeeData.findAll();

return new ModelAndView("allEmployees", "employees", allEmployees);

}

}

Next we define the EmployeeRepository which is an interface that extends the Spring Framework class JpaRepository. JpaRepository class is a generics and takes the following two parameters as arguments-

* What type of Object will this repository be working with- In our case Employee
* Id will be what type of object- Long(since id defined in the Employee class is long)

Thats the only configuration required for the repository class. The required operations like adding and retrieving employee details from DB will now be handled. 

package com.javainuse.data;

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

import com.javainuse.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee

, Long> {

}

Define the newEmployee.jsp

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

<h1>Employees page</h1>

<ul>

<c:forEach items="" var="employee">

<li></li>

</c:forEach>

</ul>

Define the allEmployees.jsp

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

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

<html>

<body>

<h1>Add new employee</h1>

<form:form modelAttribute="form">

<form:errors path="" element="div" />

<div>

<form:label path="name">Name</form:label>

<form:input path="name" />

<form:errors path="name" />

</div>

<div>

<input type="submit" />

</div>

</form:form>

</body>

</html>

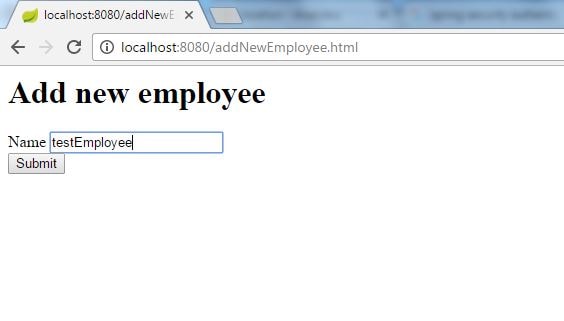
The application.properties will be as follows-

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

spring.mvc.view.suffix:.jsp

spring.datasource.url=jdbc:h2:file:./DB

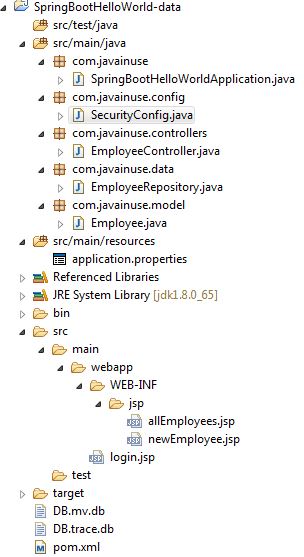
spring.jpa.properties.hibernate.hbm2ddl.auto=update

Compile and the run the SpringBootHelloWorldApplication.java as a Java application.  
Go to **localhost:8080/addNewEmployee.html**   
  
Click add. We will be directed to the allEmployees.jsp  
  


# Spring Boot + Security Hello world Example

In this post we configure a spring boot application to add basic authorization and authentication.[Spring MVC Security](https://www.javainuse.com/spring/sprsec_authprovider) had created a Simple Spring MVC Security example using Basic Authentication . But as can be seen in that post lot of configuration had to be done. This chapter we see how simple it is for configuring security with Spring Boot. We will be adding the spring security configuration for the [Spring Boot web project](https://www.javainuse.com/spring/SpringBoot_DataJPA)developed previously. We have implemented [Spring Boot Form Security Login Hello World Example](https://www.javainuse.com/spring/boot_form_security) here

Lets Begin-

We will modify the project we developed.The maven project will be as follows-   
  
  
In the Maven we only need the **spring-boot-starter-security** dependency.Maven will be as follows-

<?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.javainuse</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringBootHelloWorld</name>

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

<parent>

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

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

<version>1.4.1.RELEASE</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-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<dependency>

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

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

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

**</dependency>**

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

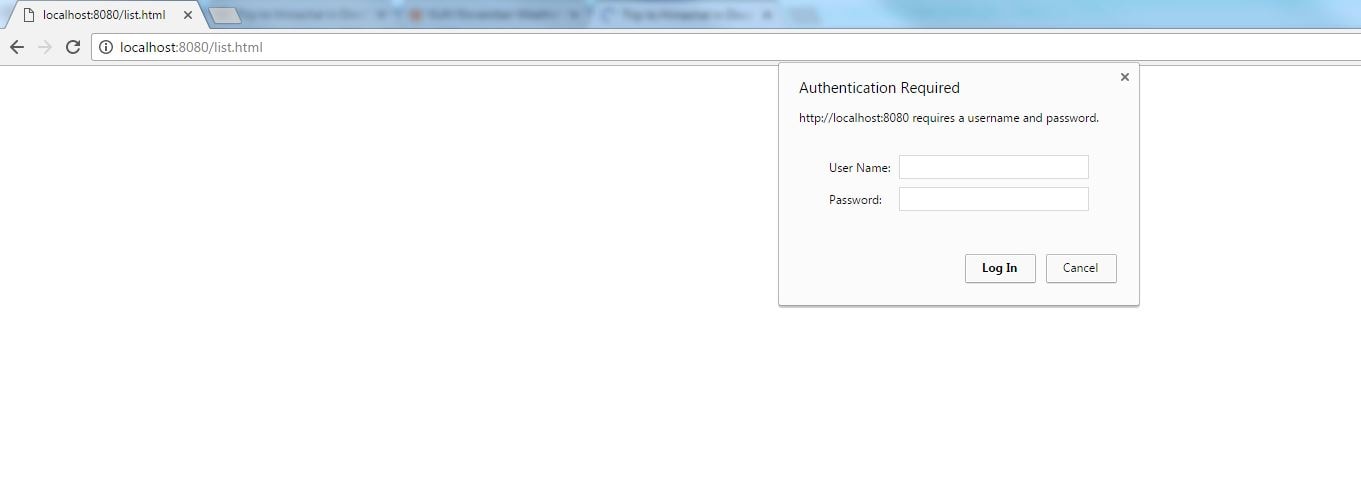
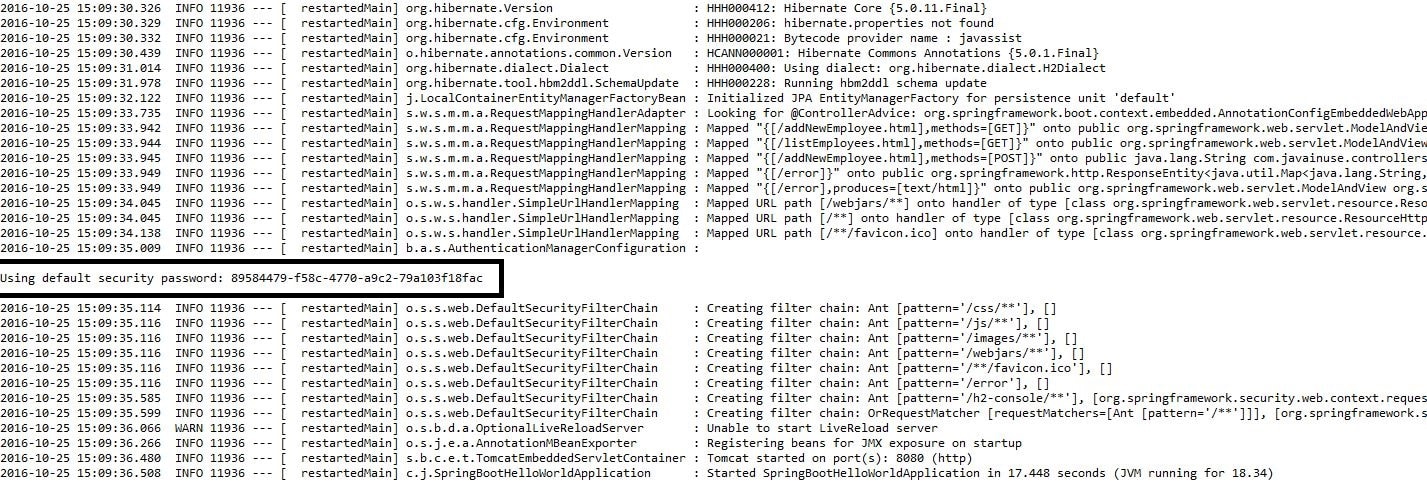
<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Lets not add any security configuration and run this code.  
Goto-**http://localhost:8080/listEmployees.html**   
  
  
We can see that no security configuration is added still it asks for username password. This is expected behaviour. We will see that the password is system generated when we run the boot application.   
  
  
**Thus by just adding the spring boot security starter dependency the basic security has already been configured by default.** Lets customize the security configuration by writing our own authorization and authentication. For this create a new class SecurityConfig that extends the WebSecurityConfigurerAdapter and overrides its methods.

package com.javainuse.config;

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.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().withUser("javainuse")

.password("javainuse").roles("USER");

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.antMatcher("/\*\*").authorizeRequests().anyRequest().hasRole("USER")

.and().formLogin().loginPage("/login.jsp")

.failureUrl("/login.jsp?error=1").loginProcessingUrl("/login")

.permitAll().and().logout()

.logoutSuccessUrl("/listEmployees.html");

}

}

We have added a new page named login.jsp. Users will get redirected to this page for adding credentials.

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

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

<html>

<head>

<title>Employees Login</title>

</head>

<body>

<h1><strong>Employees Login</strong></h1>

<c:url value="/login" var="login"/>

<form:form action="" method="post">

<label>Username:</label> <input type="text" name="username" />

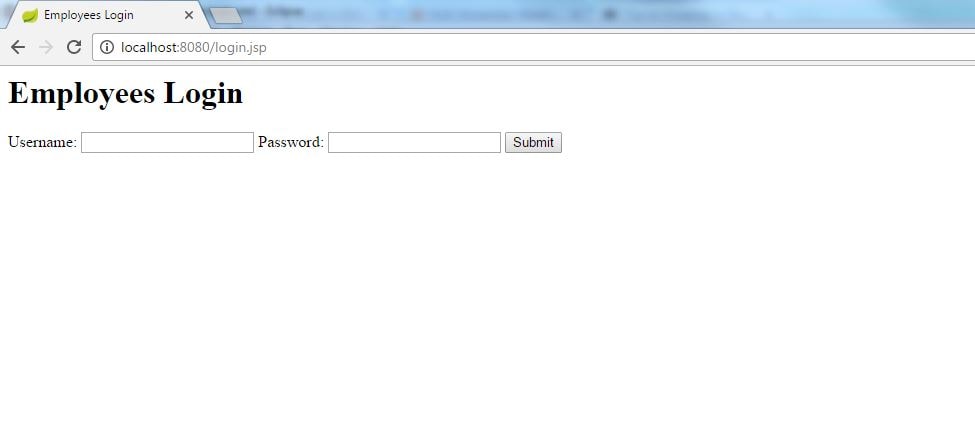
<label>Password:</label> <input type="text" name="password" />

<input type="submit"/>

</form:form>

</body>

</html>

This are the only changes required. Run the application. Goto-**http://localhost:8080/listEmployees.html**  
User will be automatically redirected to localhost:8080/login.jsp   
  
  
On entering the username and password as 'javainuse' the user will be able to visit all the other URLs correctly.