# Creating a Web Application With Spring Boot

create a simple web application with Spring Boot. We will develop a simple application with login functionality as well as the functionality to show a list of todos. We will follow the MVC pattern using JSP as the view.

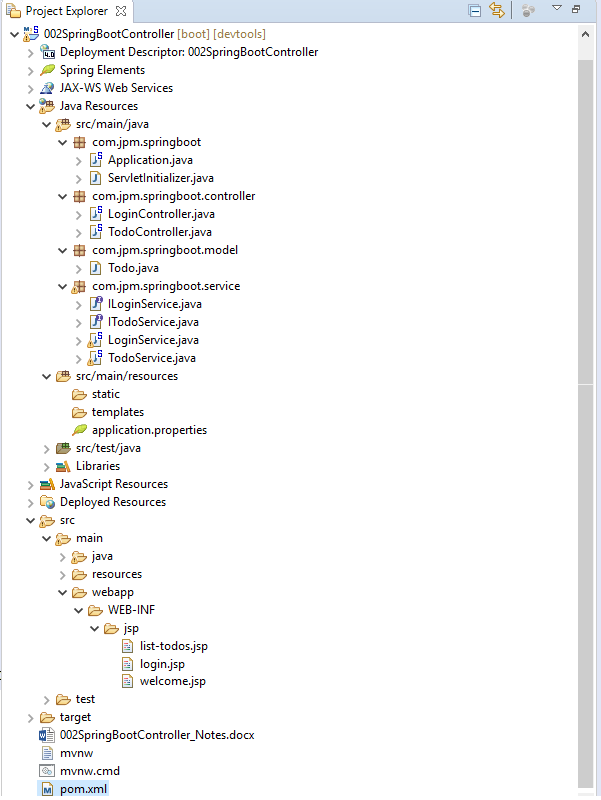
**Overview of the Web Application**

We will build a todo list page (un-formatted) with basic login features.

**Files**

The following screenshot shows an eclipse project with all the files we will create.

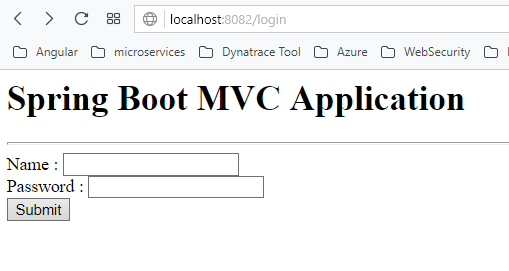
Project Structure



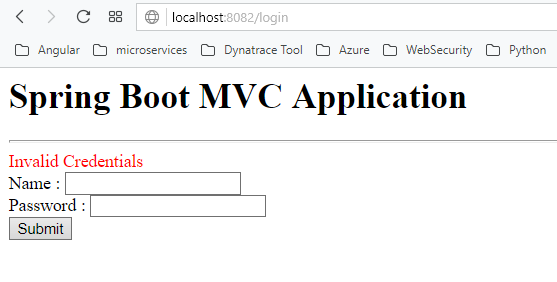
A brief overview of all files:

* LoginService, TodoService - Contains the business logic. LoginService has simple hard coded validation for user ids and passwords. TodoService contains a service method to retrieve a list of todos.
* login.jsp, welcome.jsp, list-todos.jsp - The name clearly explains what each of these views contains.
* LoginController, TodoController - Act as Controllers in the MVC pattern. LoginController has a little bit of flow. If the user enters a valid user id and password combination, they would be redirected to the welcome page. Otherwise, the login page will be shown with the error message.
* pom.xml - Important dependencies are Spring Boot Starter Web and tomcat-embed-jasper. We will talk about these later.
* application.properties - This is typically used to configure frameworks in Spring Boot. In this example, we would configure our view resolver in application.properties.

Screen shots



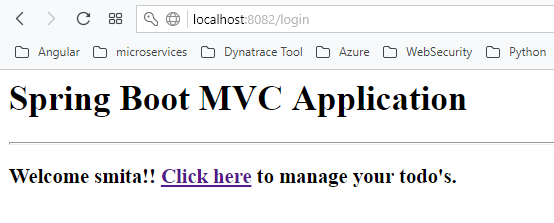
If you enter wrong credentials



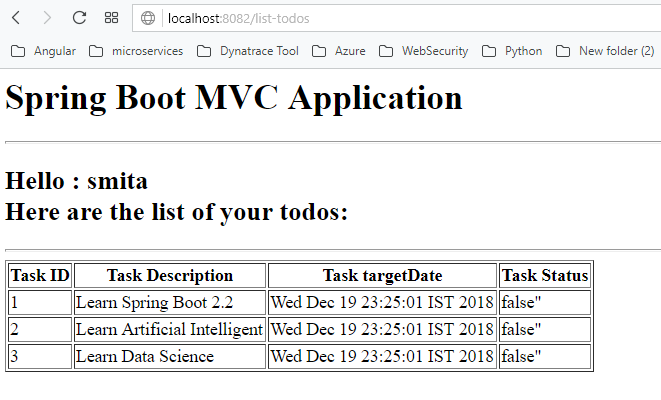
If your enter correct Credentials

Username-> smita

Password -> smita@123



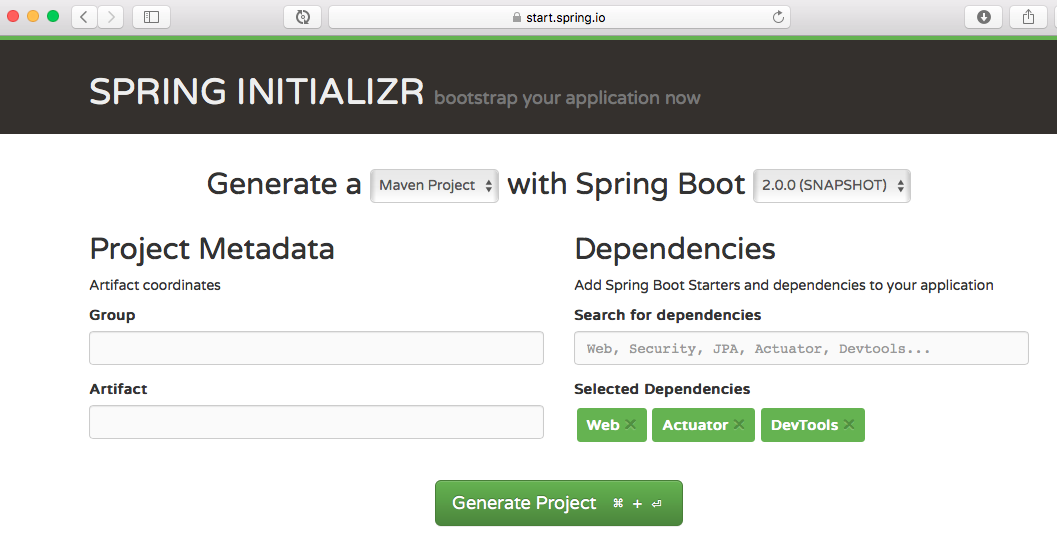
If you click on the link -> you can see the list of todo’s



**Bootstrapping a Web Application With Spring Initializr**

Creating a Web application with Spring Initializr is a cake walk. We will use Spring Web MVC as our web framework.

**Spring Initializr**[**http://start.spring.io/**](http://start.spring.io/)**is great tool to bootstrap your Spring Boot projects.**



As shown in the image above, we need to perform the following steps:

* Launch Spring Initializr and:
  + Choose com.jpm.springboot as the Group.
  + Choose student-services as the Artifact.
  + Choose from the following dependencies:
    - Web
    - Actuator
    - DevTools
* Click Generate Project.
* Import the project into Eclipse.
* Or you can also create Spring starter project in STS.

**Project Dependencies**

Spring Boot Starter Web provides all the dependencies and the auto configuration needed to develop web applications. We should use the first dependency.

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>2.1.1.RELEASE</version>**

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

**</parent>**

**<groupId>com.jpm.springboot</groupId>**

**<artifactId>002SpringBootController</artifactId>**

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

**<packaging>war</packaging>**

**<name>002SpringBootController</name>**

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

**<properties>**

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

**</properties>**

**<!-- Spring Boot Starter Web provides all the dependencies and the auto**

**configuration needed to develop web applications. We should use the first**

**dependency. -->**

**<dependencies>**

**<!-- jstl -->**

**<dependency>**

**<groupId>javax.servlet</groupId>**

**<artifactId>jstl</artifactId>**

**<version>1.2</version>**

**</dependency>**

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

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

**<scope>runtime</scope>**

**</dependency>**

**<dependency>**

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

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

**<scope>provided</scope>**

**</dependency>**

**<dependency>**

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

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

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

**</dependency>**

**<!-- We want to use JSP as the view. Default embedded servlet container**

**for Spring Boot Starter Web is tomcat. To enable support for JSP’s, we would**

**need to add a dependency on tomcat-embed-jasper. -->**

**<dependency>**

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

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

**<scope>provided</scope>**

**</dependency>**

**</dependencies>**

**<!-- Dependencies can be classified into: Spring - core, beans, context,**

**app Web MVC - (Spring MVC) Jackson - for JSON Binding Validation - Hibernate**

**Validator, Validation API Embedded Servlet Container - Tomcat Logging - logback,**

**slf4j Any typical web application would use all these dependencies. Spring**

**Boot Starter Web comes pre-packaged with these. As a developer, I would not**

**need to worry about either these dependencies or their compatible versions. -->**

**<build>**

**<plugins>**

**<plugin>**

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

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

**</plugin>**

**</plugins>**

**</build>**

**</project>**

### Auto Configuration

Spring Boot Starter Web auto configures the basic things we needed to get started. To understand the features Spring Boot Starter Web brings in, let's run StudentServicesApplication.java as a Java Application and review the log.

Mapping servlet: 'dispatcherServlet' to [/]

Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)

Mapped URL path [/webjars/\*\*] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

Spring Boot Starter Web auto-configures:

* Dispatcher Servlet.
* Error Page.
* Web Jars to manage your static dependencies.
* Embedded Servlet Container - Tomcat is the default.

## Configuring a View Resolver in *src/main/resources/application.properties* file

We would have our jsp’s in /WEB-INF/jsp/. We would need to configure the view resolver with the prefix and suffix.

**server.port=8082**

**#Configuring a View Resolver**

**#We would have our jsp in /WEB-INF/jsp/.**

**#We would need to configure the view resolver with the prefix and suffix.**

**spring.mvc.view.prefix=/WEB-INF/jsp/**

**spring.mvc.view.suffix=.jsp**

## Login Service

Has the basic logic for authentication. Hardcoded business logic.

***package com.jpm.springboot.service;***

***public interface ILoginService {***

***public boolean validateUser(String userid, String password);***

***}***

***package com.jpm.springboot.service;***

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

***/\*@author Smita***

***\****

***\*/***

***@Service***

***public class LoginService implements ILoginService{***

***@Override***

***public boolean validateUser(String userid, String password) {***

***return userid.equalsIgnoreCase("smita")***

***&& password.equalsIgnoreCase("smita@123");***

***}***

***}***

## Login Controller

* public String showLoginPage(ModelMap model): Mapped to the \login Get Method, this method shows the login page.
* @Autowired ILoginService service: LoginService implements LoginService and has the validation logic.
* showWelcomePage(ModelMap model, @RequestParam String name, @RequestParam String password Mapped to the \login Post Method, this method validates the user id and password. Redirects to welcome page if login is successful.

***package com.jpm.springboot.controller;***

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

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

***import org.springframework.ui.ModelMap;***

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

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

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

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

***import com.jpm.springboot.service.LoginService;***

***/\*Login Controller***

***public String showLoginPage(ModelMap model): Mapped to the \login Get Method, this method shows the login page.***

***@Autowired LoginService service: LoginService has the validation logic.***

***showWelcomePage(ModelMap model, @RequestParam String name, @RequestParam String passwordMapped to the \login Post Method, this method validates the user id and password. Redirects to welcome page if login is successful.***

***\*/***

***@Controller***

***@SessionAttributes("name")***

***public class LoginController {***

***@Autowired***

***LoginService service;***

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

***public String showLoginPage(ModelMap model){***

***return "login";***

***}***

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

***public String showWelcomePage(ModelMap model, @RequestParam String name, @RequestParam String password){***

***boolean isValidUser = service.validateUser(name, password);***

***if (!isValidUser) {***

***model.put("errorMessage", "Invalid Credentials");***

***return "login";***

***}***

***model.put("name", name);***

***model.put("password", password);***

***return "welcome";***

***}***

***}***

## Login View - JSP

Simple login page with user id and password form fields. If an error message is populated into the model, ${errorMessage} will show the authentication failure error message.

***<%-- Login View - JSP***

***Simple login page with user id and password form fields. If an error message is populated into the model, ${errorMessage} will show the authentication failure error message.***

***--%>***

***<html>***

***<head>***

***<title>Spring Boot MVC Application</title>***

***</head>***

***<body>***

***<h1>Spring Boot MVC Application</h1>***

***<hr>***

***<font color="red">${errorMessage}</font>***

***<form method="post">***

***Name : <input type="text" name="name" /> <br>Password : <input***

***type="password" name="password" /><br> <input type="submit" />***

***</form>***

***</body>***

***</html>***

## Welcome View - JSP

The welcome page is shown on successful authentication. Shows the name of the login user and a link to manage your todos.

***<!-- Welcome View - JSP***

***The welcome page is shown on successful authentication. Shows the name of the login user and a link to manage your todos.***

***-->***

***<html>***

***<head>***

***<title>Spring Boot MVC Application</title>***

***</head>***

***<body>***

***<h1>Spring Boot MVC Application</h1>***

***<hr>***

***<h3>***

***Welcome ${name}!! <a href="/list-todos">Click here</a> to manage your***

***todo's.***

***</h3>***

***</body>***

***</html>***

## Todo Model and Business Service

We are using a basic todo with id, user, description, and a target date.

**package com.jpm.springboot.model;**

**/\*\* @author Smita \* \*/**

**/\*Todo Model and Business Service**

**We are using a basic todo with id, user, description, and a target date.**

**\*/**

**import java.util.Date;**

**public class Todo {**

**private int id;**

**private String user;**

**private String desc;**

**private Date targetDate;**

**private boolean status;**

**//Getters, Setters, Constructors, toString, equals and hash code**

**public Todo() {**

**// TODO Auto-generated constructor stub**

**}**

**public Todo(int id, String user, String desc, Date targetDate, boolean status) {**

**super();**

**this.id = id;**

**this.user = user;**

**this.desc = desc;**

**this.targetDate = targetDate;**

**this.status = status;**

**}**

**@Override**

**public int hashCode() {**

**final int prime = 31;**

**int result = 1;**

**result = prime \* result + ((desc == null) ? 0 : desc.hashCode());**

**result = prime \* result + id;**

**result = prime \* result + (status ? 1231 : 1237);**

**result = prime \* result + ((targetDate == null) ? 0 : targetDate.hashCode());**

**result = prime \* result + ((user == null) ? 0 : user.hashCode());**

**return result;**

**}**

**public int getId() {**

**return id;**

**}**

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

**this.id = id;**

**}**

**public String getUser() {**

**return user;**

**}**

**public void setUser(String user) {**

**this.user = user;**

**}**

**public String getDesc() {**

**return desc;**

**}**

**public void setDesc(String desc) {**

**this.desc = desc;**

**}**

**public Date getTargetDate() {**

**return targetDate;**

**}**

**public void setTargetDate(Date targetDate) {**

**this.targetDate = targetDate;**

**}**

**public boolean isStatus() {**

**return status;**

**}**

**public void setStatus(boolean status) {**

**this.status = status;**

**}**

**@Override**

**public String toString() {**

**return "Todo [id=" + id + ", user=" + user + ", desc=" + desc + ", targetDate=" + targetDate + ", status="**

**+ status + "]";**

**}**

**@Override**

**public boolean equals(Object obj) {**

**if (this == obj)**

**return true;**

**if (obj == null)**

**return false;**

**if (getClass() != obj.getClass())**

**return false;**

**Todo other = (Todo) obj;**

**if (desc == null) {**

**if (other.desc != null)**

**return false;**

**} else if (!desc.equals(other.desc))**

**return false;**

**if (id != other.id)**

**return false;**

**if (status != other.status)**

**return false;**

**if (targetDate == null) {**

**if (other.targetDate != null)**

**return false;**

**} else if (!targetDate.equals(other.targetDate))**

**return false;**

**if (user == null) {**

**if (other.user != null)**

**return false;**

**} else if (!user.equals(other.user))**

**return false;**

**return true;**

**}**

**}**

**/\***

**\* Our todo service uses a simple ArrayList to store a list of todos. It offers**

**\* a method to retrieve the todos.**

## \*/ List Todos - JSP

This is the page that will show our list of todos. This is a completely unformatted page. During the subsequent steps, we will beautify this page and create more functionalities so you can add, delete, and update todos.

<!-- List Todos - JSP

This is the page that will show our list of todos. This is a completely unformatted page. During the subsequent steps, we will beautify this page and create more functionalities so you can add, delete, and update todos.

-->

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

<html>

<head>

<title>Spring Boot MVC Application</title>

</head>

<body>

<h1>Spring Boot MVC Application</h1>

<hr>

<h2>

Hello : ${name}<br>Here are the list of your todos:

</h2>

<hr>

<table border=*"1"*>

<thead>

<tr>

<th>Task ID</th>

<th>Task Description</th>

<th>Task targetDate</th>

<th>Task Status</th>

</tr>

</thead>

<tbody>

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

<tr>

<td colspan=*"8"*>No task to Display</td>

</tr>

</c:if>

<c:if test=*"*${**not empty** todos}*"*>

<c:forEach items=*"*${todos}*"* var=*"task"*>

<tr class=*""*>

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

<td>${task.desc}</td>

<td>${task.targetDate}</td>

<td>${task.status}"</td>

</tr>

</c:forEach>

</c:if>

</tbody>

</table>

</div>

</body>

</html>

## Todo Controller

The Todo Controller has a simple method to retrieve the list of todos and populate it into the model. It redirects to the list-todos view.

**package com.jpm.springboot.controller;**

**/\*\***

**\* @author Smita \*/**

**/\*Todo Controller**

**The Todo Controller has a simple method to retrieve the list of todos and populate it into the model. It redirects to the list-todos view.**

**\*/**

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

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

**import org.springframework.ui.ModelMap;**

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

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

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

**import com.jpm.springboot.service.ITodoService;**

**@Controller**

**@SessionAttributes("name")**

**public class TodoController {**

**@Autowired**

**ITodoService service;**

**@RequestMapping(value="/list-todos", method = RequestMethod.*GET*)**

**public String showTodos(ModelMap model){**

**String name = (String) model.get("name");**

**model.put("todos", service.retrieveTodos(name));**

**return "list-todos";**

**}**

**}**

## Running the Application

We use a Spring Boot Application class to launch our application.

**package com.jpm.springboot;**

**/\*\***

**\* @author Smita \*/**

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

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

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

**/\*Running the Application**

**We use a Spring Boot Application class to launch our application.\*/**

**@SpringBootApplication**

**@ComponentScan("com.jpm.springboot")**

**public class Application {**

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

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

**}**

**}**

The complete code is on my