# **Hands On Lab 1**

**Spring MVC**

# **Creating a Web Application With Spring Boot**

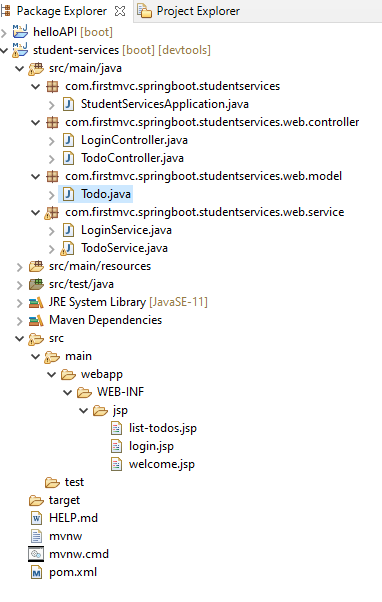
This lab guide will help you create a simple web application with Spring Boot. We will develop a simple application with login functionality as well as the functionality to show a list of todos. We will follow the MVC pattern using JSP as the view.

## **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.

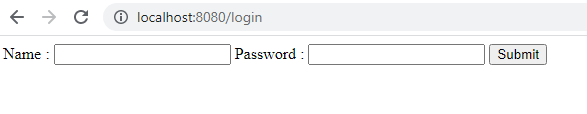


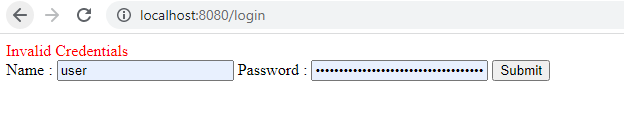
A brief overview of all files:

* 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.

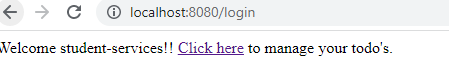
### **Screenshots of the Application**

Login Page

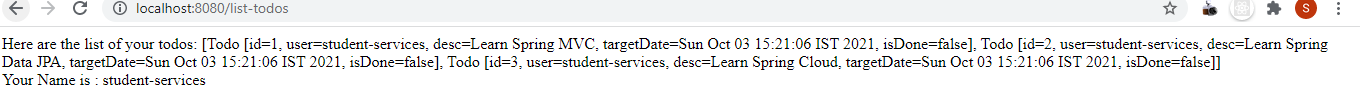
Login Page if user enters invalid user id and password



Welcome Page

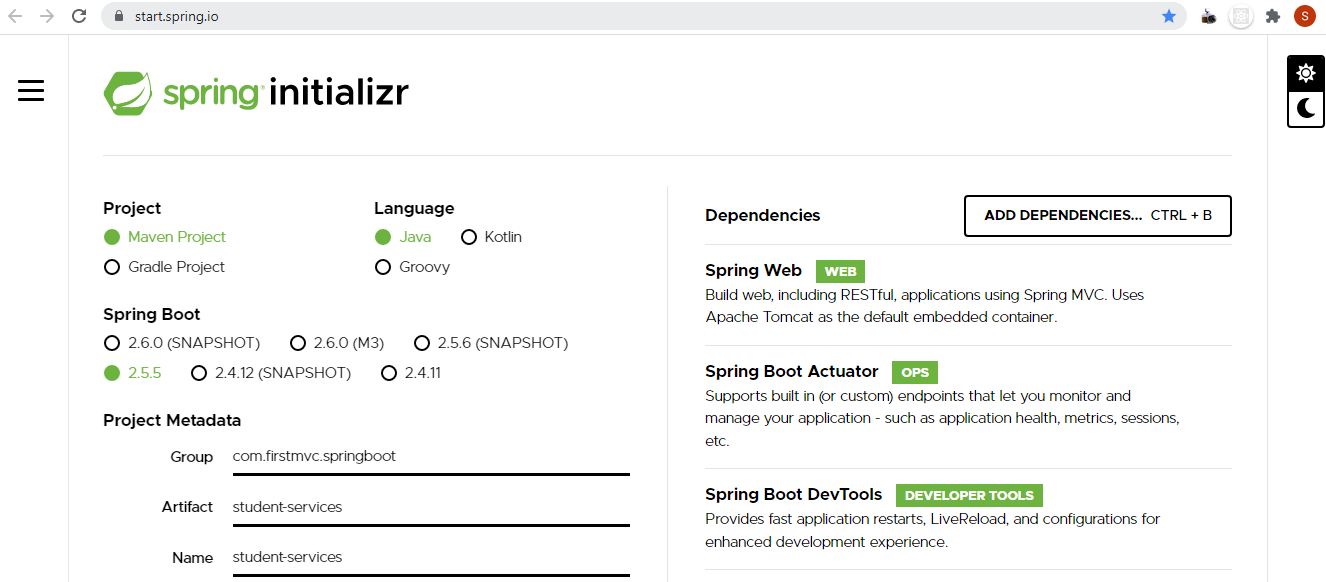


List Todos Page



## **Bootstrapping a Web Application With Spring Initializr**

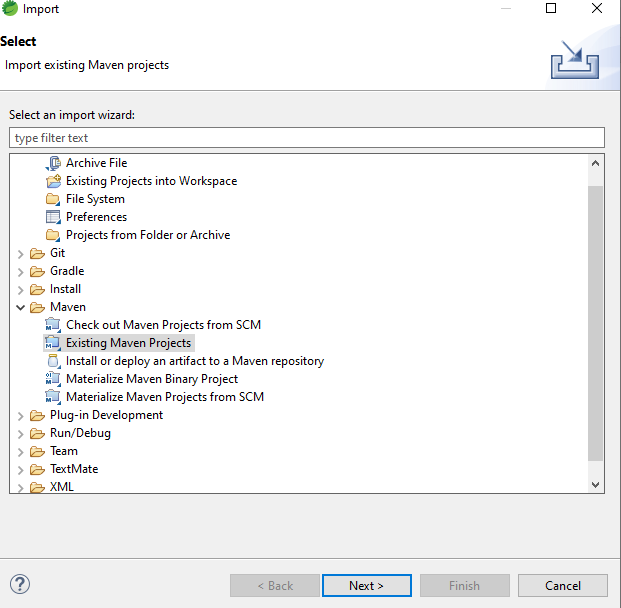
Creating a Web application with Spring Initializr is a cake walk. We will use Spring Web MVC as our web framework.

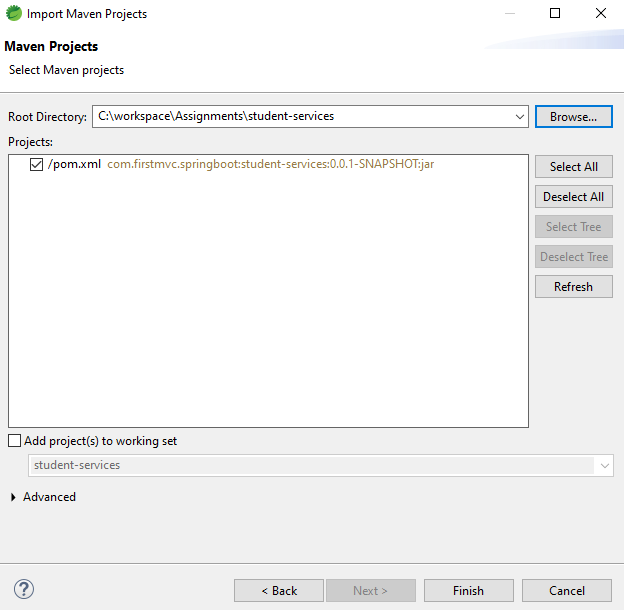




As shown in the image above, we need to perform the following steps:

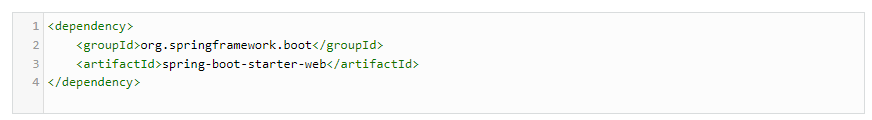
* Launch Spring Initializr and:
  + Choose com.firstmvc.springboot as the Group.
  + Choose student-services as the Artifact.
  + Choose from the following dependencies:
    - Web
    - Actuator
    - DevTools
* Click Generate Project.
  + Unzip and make sure the top-level directory is student-services and doesn't have another student-services folder nested within it.
  + Copy the extracted files to your workspace.
  + Import the project into Eclipse.



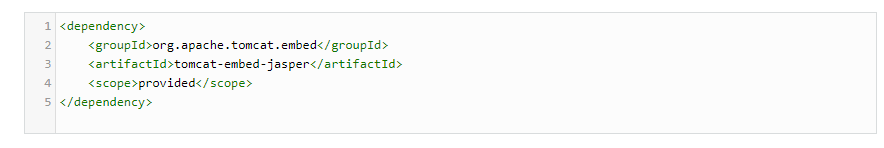


## **Project Dependencies**

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

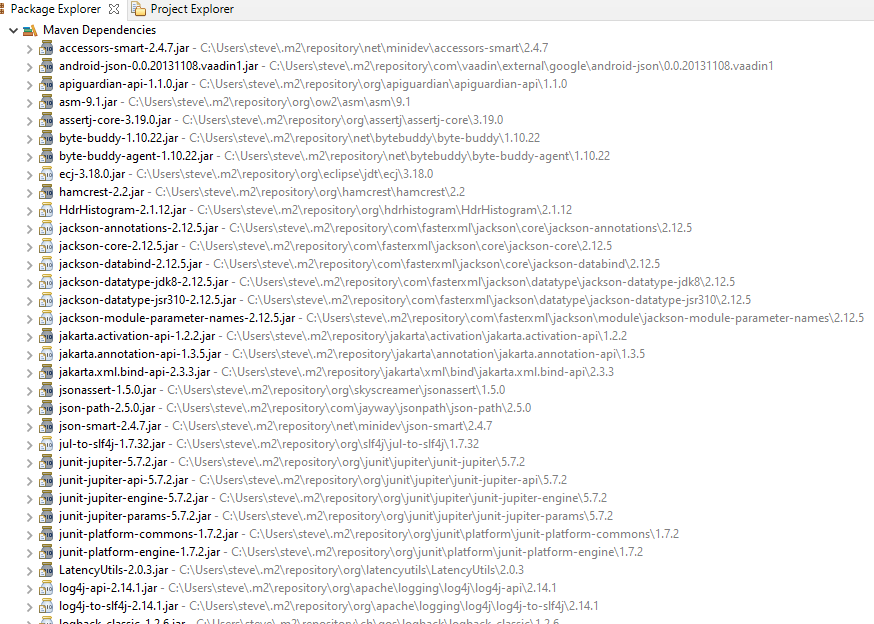


We want to use JSP as the view. Default embedded servlet container for Spring Boot Starter Web is tomcat. To enable support for JSP’s, we would need to add a dependency on tomcat-embed-jasper.



### **Spring Boot Starter Web Dependencies**

Following screenshot shows some of the different dependencies that are added into our application because of Spring Boot Starter Web.



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

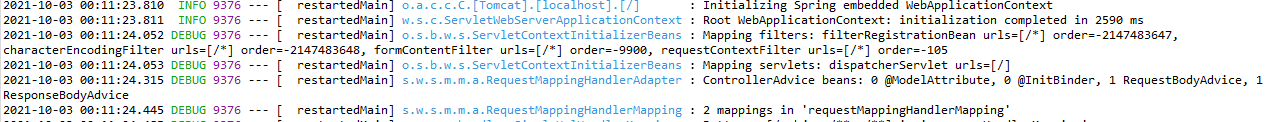
Any typical web application would use all these dependencies. Spring Boot Starter Web comes pre-packaged with these. As a developer, I would not need to worry about either these dependencies or their compatible versions.

### **Auto Configuration**

Spring Boot Starter Web auto configures the basic things we needed to get started. Add this line to your application.properties.



To understand the features Spring Boot Starter Web brings in, let's run StudentServicesApplication.java as a Java Application and review the log.



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**

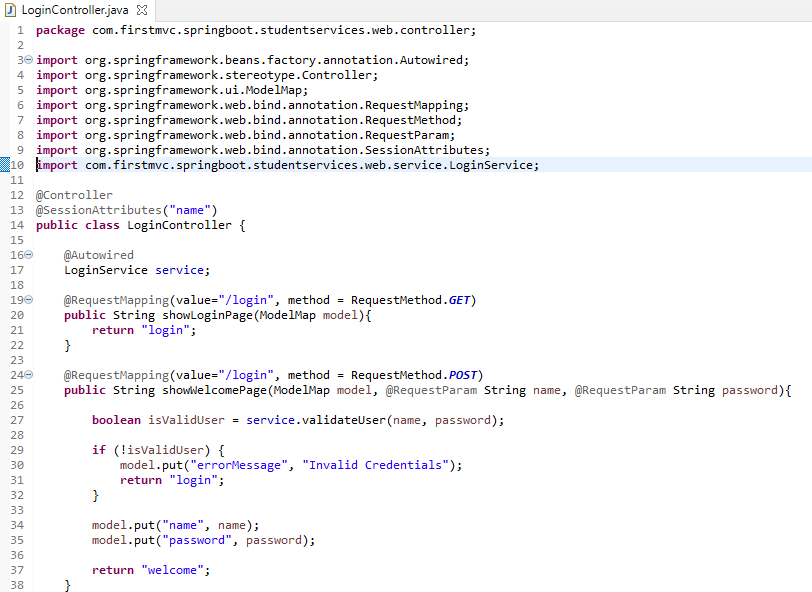
We would have our jsp’s in /WEB-INF/jsp/. We would need to configure the view resolver with the prefix and suffix.

Add the following lines at the top of application.properties.



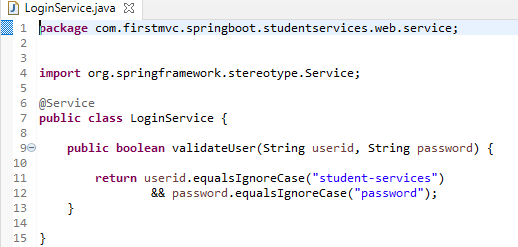
## **Login Controller**

* 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 password Mapped to the \login Post Method, this method validates the user id and password. Redirects to welcome page if login is successful.



## **Login Service**

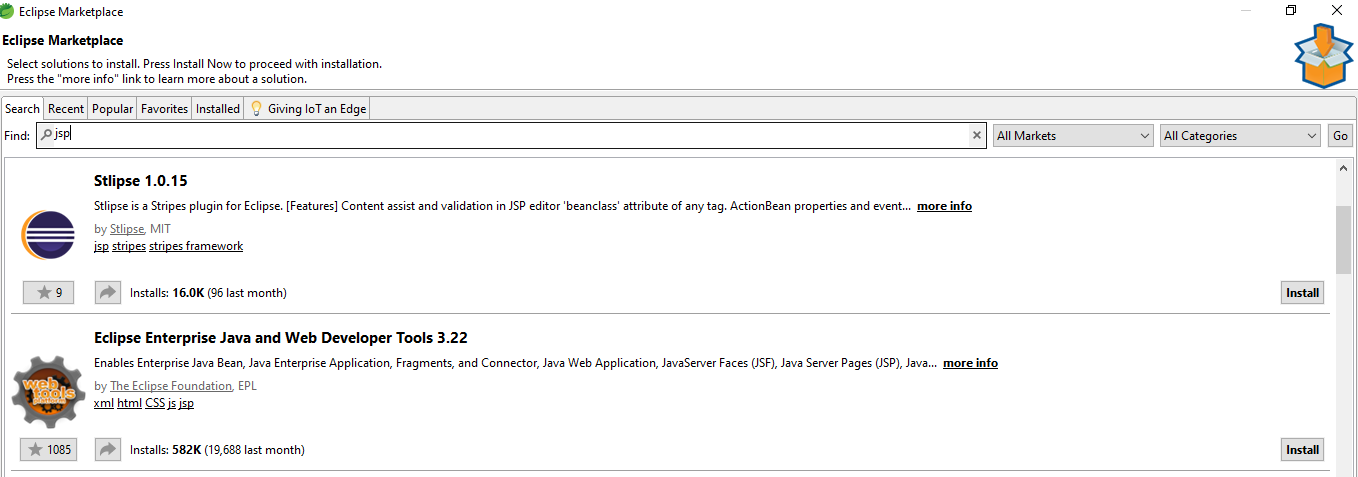
Has the basic logic for authentication. Hardcoded business logic.



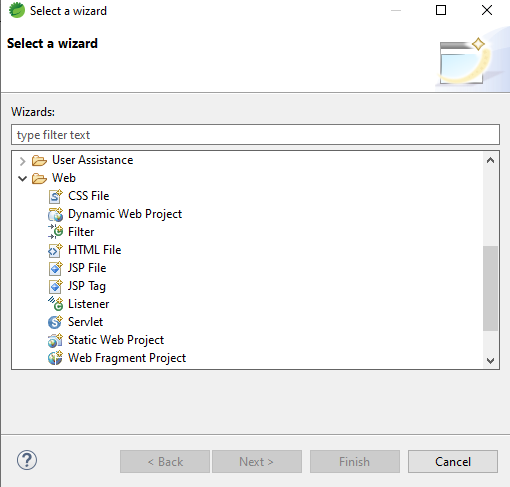
## **Install Enterprise Java and Web Developer Tools in STS**

For the Spring tool suite 4, by default, JSP file support is not included in the suite.

In your suite, just go to Help -> Eclipse Marketplace -> type Eclipse Enterprise Java and Web Developer Tools (version = latest one is 3.22) and install it.



Then restart your suite. Your jsps should now look as in the screenshots below. Additionally, New -> Other -> will have a few more wizards available for web development.



## **Login View - JSP**

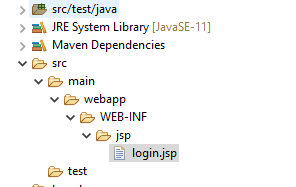
Simple login page with user id and password form fields. If an error message is populated into the model, ${errorMessage} will show the authentication failure error message.

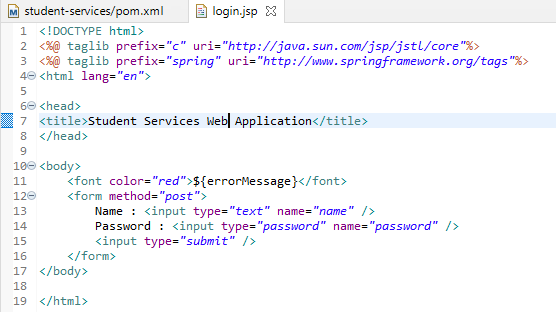
Add the following dependency to pom.xml.



Create the directory structure as shown below and then create a new jsp file named login.jsp.

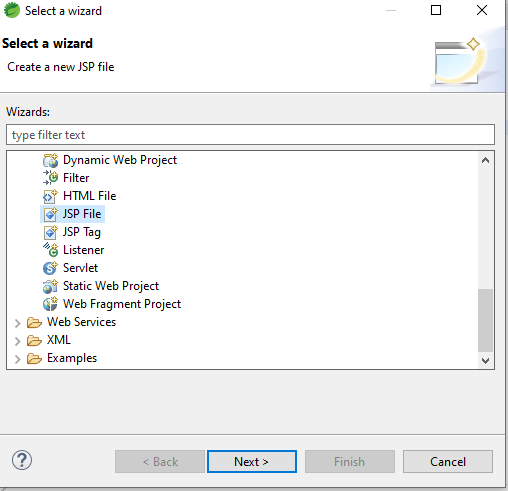
Right click jsp folder ,then New ->Other ->File.

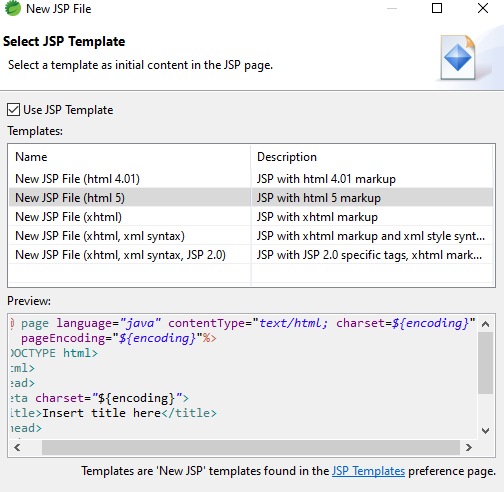
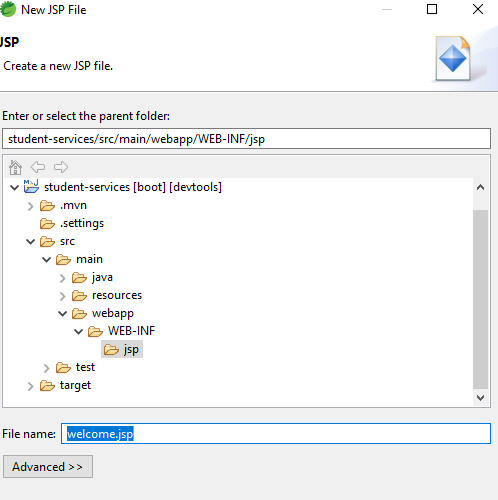




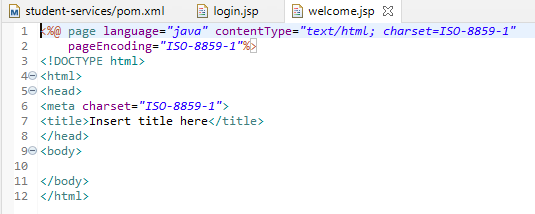
## **Welcome View - JSP**

The welcome page is shown on successful authentication. Shows the name of the login user and a link to manage your todos.Place it in the jsp folder you create.

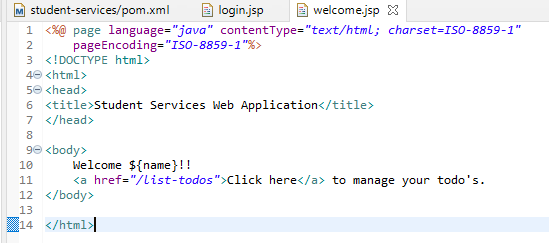




Your welcome.jsp should initially look like this.

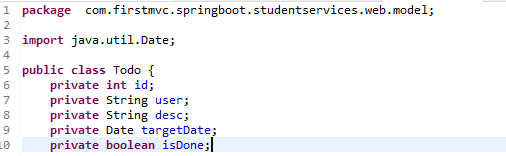


Edit welcome.jsp to include the below content.

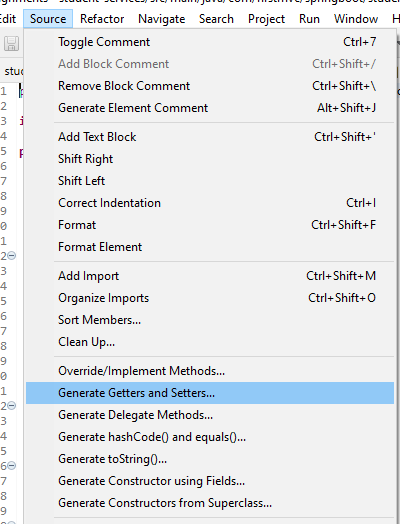


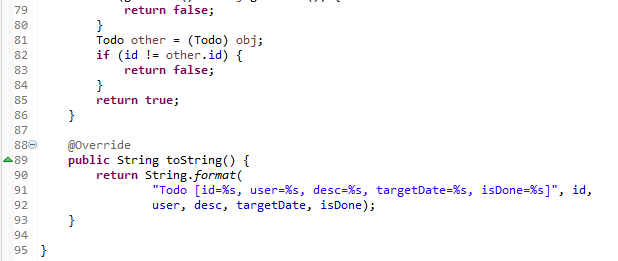
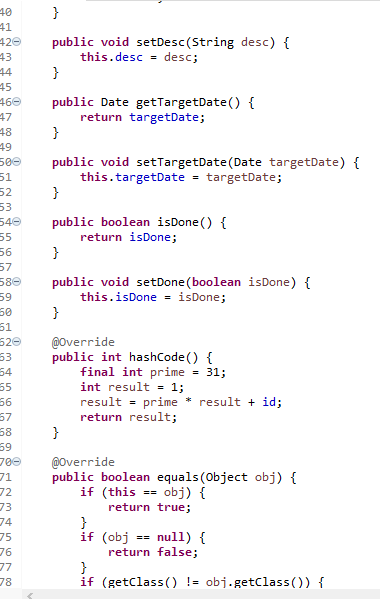
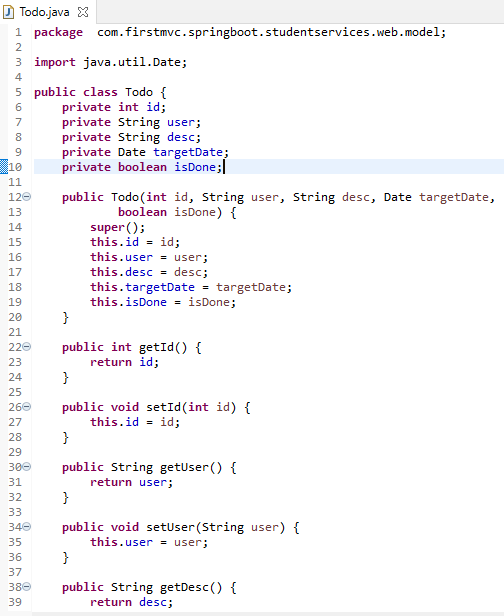
## **Todo Model and Business Service**

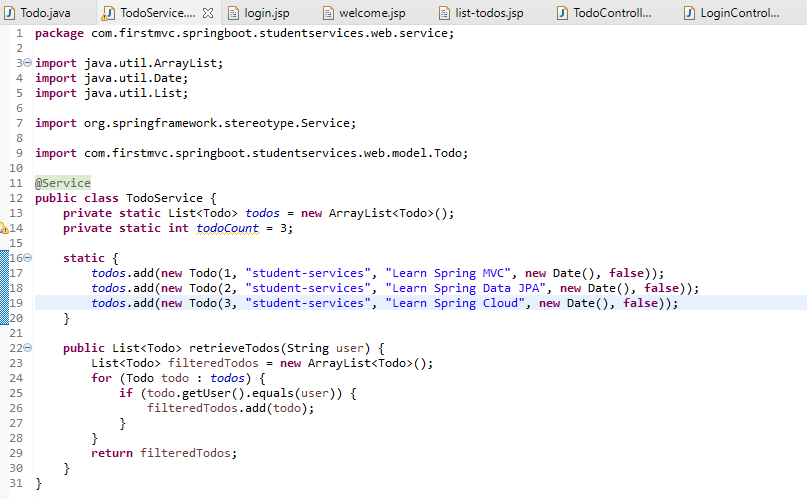
We are using a basic todo with id, user, description, and a target date.



You can save yourself some typing by using the IDE.

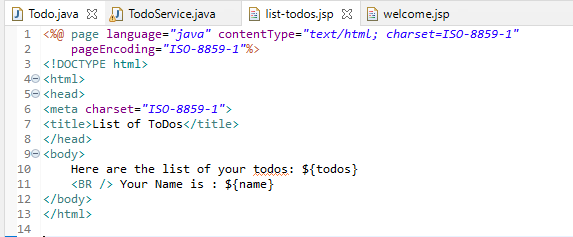


The complete class file is as below. 

Our todo service uses a simple ArrayList to store a list of todos. It offers a method to retrieve the todos.  
  


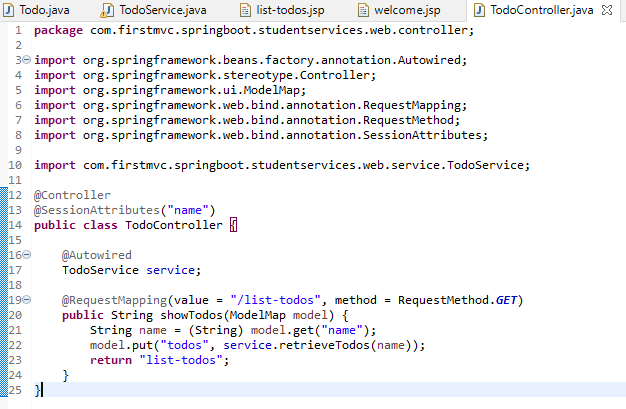
## **List Todos - JSP**

This is the page that will show our list of todos. This is a completely unformatted page. Create the list-todos.jsp in the jsp folder.



## **Todo Controller**

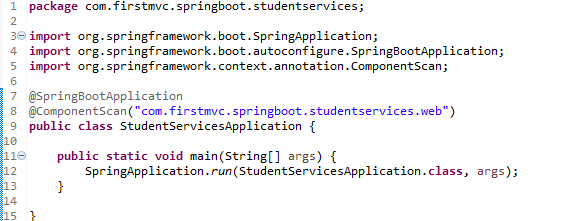
The Todo Controller has a simple method to retrieve the list of todos and populate it into the model. It redirects to the list-todos view.



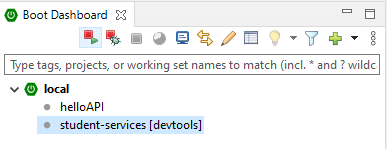
## **Running the Application**

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

Add the ComponentScan annotation to the Spring Boot Application class.

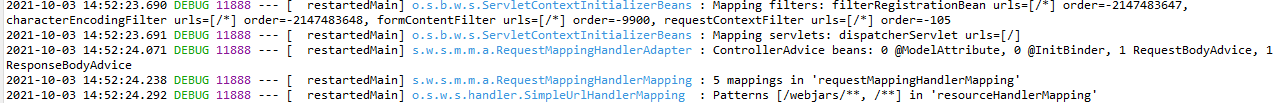


You can run this as a simple Java application from the Boot Dashboard.



 When you run this you should see the application starting up. Below are some of the extracts from the log. You should verify that all the request mappings are properly mapped.

The application can be launched at <http://localhost:8080/login> and enter user id/password combination of student-services/password.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*