Table of Contents

Overview

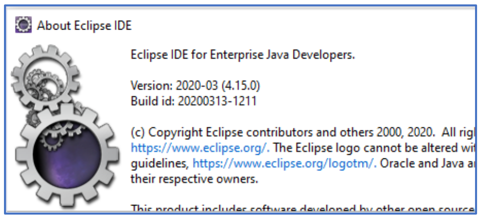
In this course, you'll use several features of this framework, combining their use to build a simple web application. You'll learn how to connect this web application to a MySQL server database, which will be used to store your data. Next, you'll use JDBC templates to query this database and then set up login and register pages on your app. Finally, you'll perform create, read, update, and delete operations using Spring MVC.

Spring MVC is an integral part of the Spring Framework, and uses the Spring Core functionality of inversion of control using dependency injection. A key design principle in Spring MVCs architecture is open for extension, but closed for modification, which makes it easy for developers to follow best practices while developing their app. In this course, we'll bring together a number of features to build a simple web application. We'll connect this web application to a MySQL server database which will be used to store our data.

We'll use JDBC templates to query this database and see how we can set up login and register pages on our app. Once you're done with this course, you will be able to build a web application connected to a MySQL database using the classic three-tier architecture.

Setup Maven Project on Eclipse

1. Open Eclipse IDE



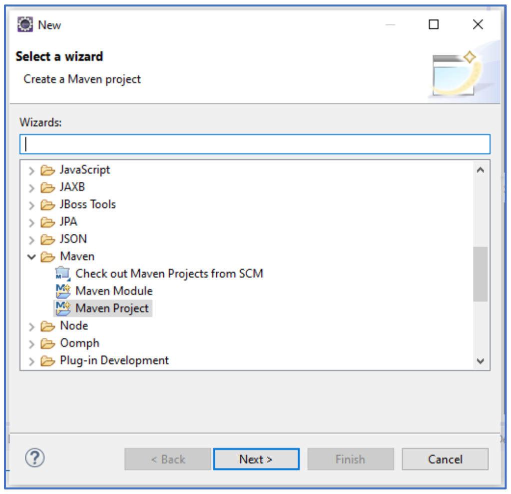
About Eclipse IDE

Eclipse IDE for Enterprise Java Developers.

Version: 2020-03 (4.15.0)
Build id: 20200313-1211

(c) Copyright Eclipse contributors and others 2000, 2020. All rig
https://www.eclipse.org/. The Eclipse logo cannot be altered wi
guidelines, https://www.eclipse.org/logotm/. Oracle and Java a
their respective owners.

2. Create New Maven Project
   File > New > Maven Project



New

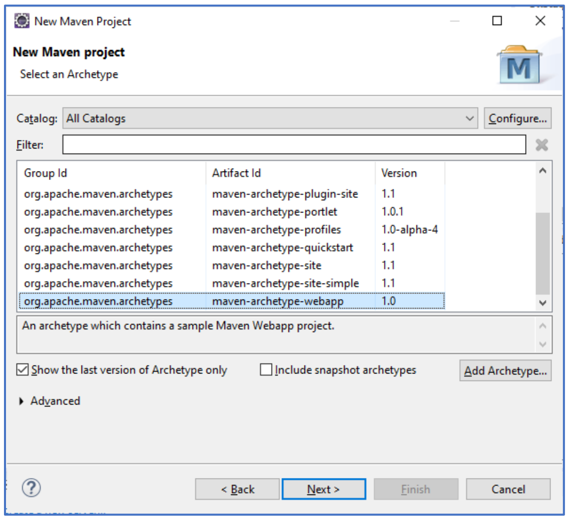
Select a wizard

Create a Maven project

Wizards:

- JavaScript
- JAXB
- JBoss Tools
- JPA
- JSON
- Maven
  - Check out Maven Projects from SCM
  - Maven Module
  - Maven Project
- Node
- Oomph
- Plug-in Development

3. select or choose ***Workspace*** location
4. Choose ***maven-archetype-webapp (ver 1.0)*** as the archetype, click next



New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

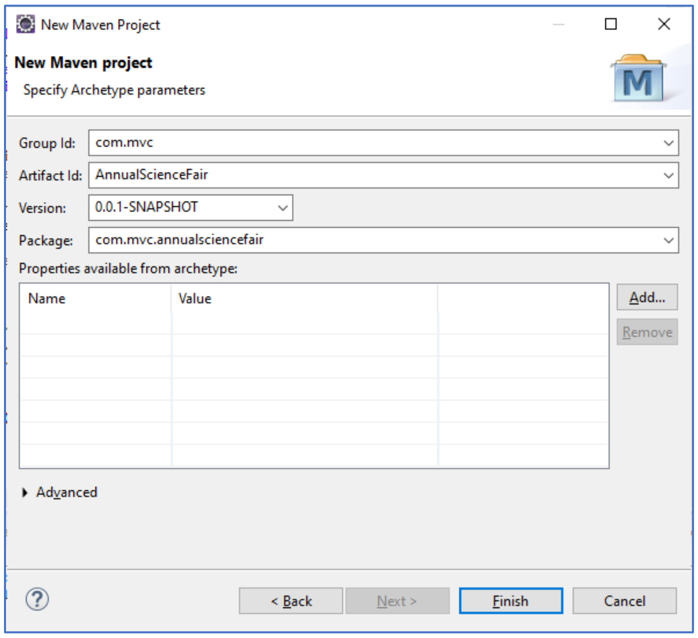
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven Webapp project.

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

Now, in order to generate a project template for a Maven application, we need to choose an archetype. An archetype in Maven is a project templating toolkit. This will set up all of the boilerplate configuration for the kind of project that you want to build using Maven. Now, the kind of project that we want to set up is a web application. So choose the Maven archetype webapp version 1.0

5. Defined the artifact Id, and click finish



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.mvc

Artifact Id: AnnualScienceFair

Version: 0.0.1-SNAPSHOT

Package: com.mvc.annualsciencefair

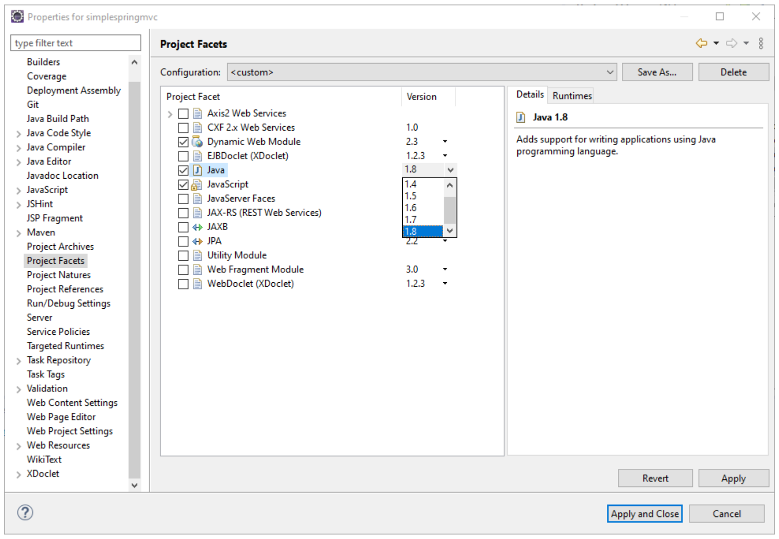
Properties available from archetype:

Name	Value

The Group id makes up part of the identifier that is used to uniquely identify this particular project. The Group id is typically the name of the company or the organization that's creating this project.

The Artifact id is the unique name of the project

6. Setup Runtime Java Version
   on “**Project Facet**” tab, select “**Java**” and set to appropriate JVM version and click Apply



Properties for simplespringmvc

type filter text

Builders, Coverage, Deployment Assembly, Git, Java Build Path, Java Code Style, Java Compiler, Java Editor, Javadoc Location, JavaScript, JSHint, JSP Fragment, Maven, Project Archives, Project Facets, Project Natures, Project References, Run/Debug Settings, Server, Service Policies, Targeted Runtimes, Task Repository, Task Tags, Validation, Web Content Settings, Web Page Editor, Web Project Settings, Web Resources, WikiText, XDoclet

Project Facets

Configuration: <custom>

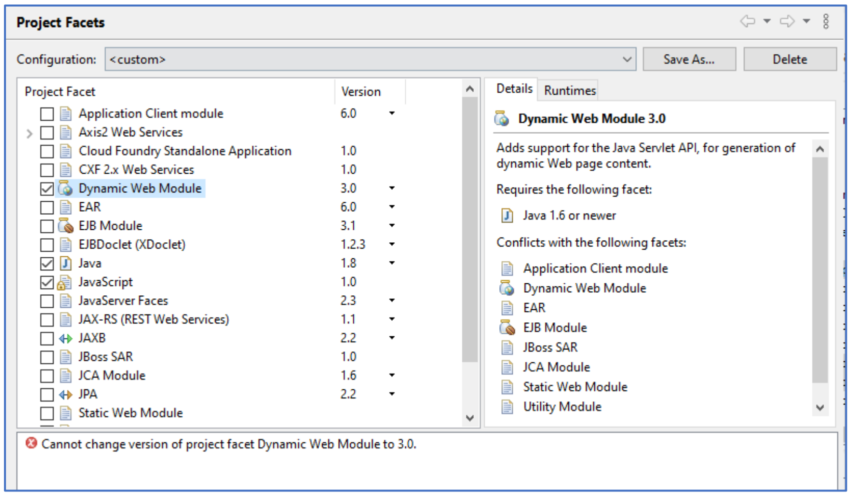
	Project Facet	Version
☐	Axis2 Web Services	
☐	CXF 2.x Web Services	1.0
✓	Dynamic Web Module	2.3
☐	EJBDoclet (XDoclet)	1.2.3
✓	Java	1.8
✓	JavaScript	
☐	JavaServer Faces	
☐	JAX-RS (REST Web Services)	
☐	JAXB	
☐	JPA	2.2
☐	Utility Module	
☐	Web Fragment Module	3.0
☐	WebDoclet (XDoclet)	1.2.3

Version list: 1.4, 1.5, 1.6, 1.7, 1.8

Details: Java 1.8

Adds support for writing applications using Java programming language.

If for any reason you found that you are not able to switch the Project Facet of Dynamic Web Module to version 3.0,



Project Facets

Configuration: <custom>

	Project Facet	Version
☐	Application Client module	6.0
☐	Axis2 Web Services	
☐	Cloud Foundry Standalone Application	1.0
☐	CXF 2.x Web Services	1.0
✓	Dynamic Web Module	3.0
☐	EAR	6.0
☐	EJB Module	3.1
☐	EJBDoclet (XDoclet)	1.2.3
✓	Java	1.8
✓	JavaScript	1.0
☐	JavaServer Faces	2.3
☐	JAX-RS (REST Web Services)	1.1
☐	JAXB	2.2
☐	JBoss SAR	1.0
☐	JCA Module	1.6
☐	JPA	2.2
☐	Static Web Module	

Details: Dynamic Web Module 3.0

Adds support for the Java Servlet API, for generation of dynamic Web page content.

Requires the following facet:

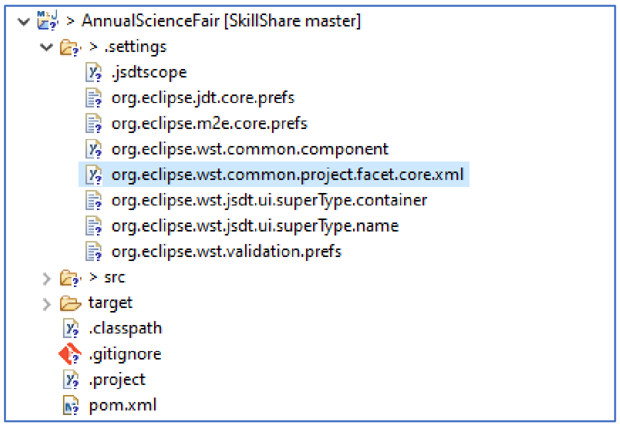
Java 1.6 or newer

Conflicts with the following facets:

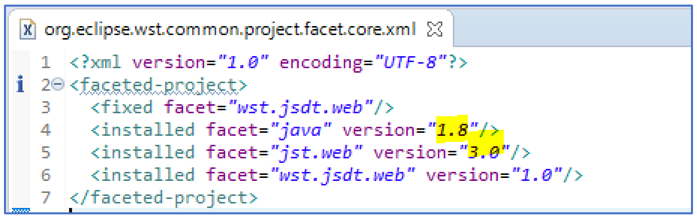
Application Client module
Dynamic Web Module
EAR
EJB Module
JBoss SAR
JCA Module
Static Web Module
Utility Module

Cannot change version of project facet Dynamic Web Module to 3.0.

you should modify org.eclipse.wst.common.project.facet.core.xml files under .setting folder inside your project resource path



```
> AnnualScienceFair [SkillShare master]
    > .settings
        .jsdtscope
        org.eclipse.jdt.core.prefs
        org.eclipse.m2e.core.prefs
        org.eclipse.wst.common.component
        org.eclipse.wst.common.project.facet.core.xml
        org.eclipse.wst.jsdt.ui.superType.container
        org.eclipse.wst.jsdt.ui.superType.name
        org.eclipse.wst.validation.prefs
    > src
    target
    .classpath
    .gitignore
    .project
    pom.xml
```

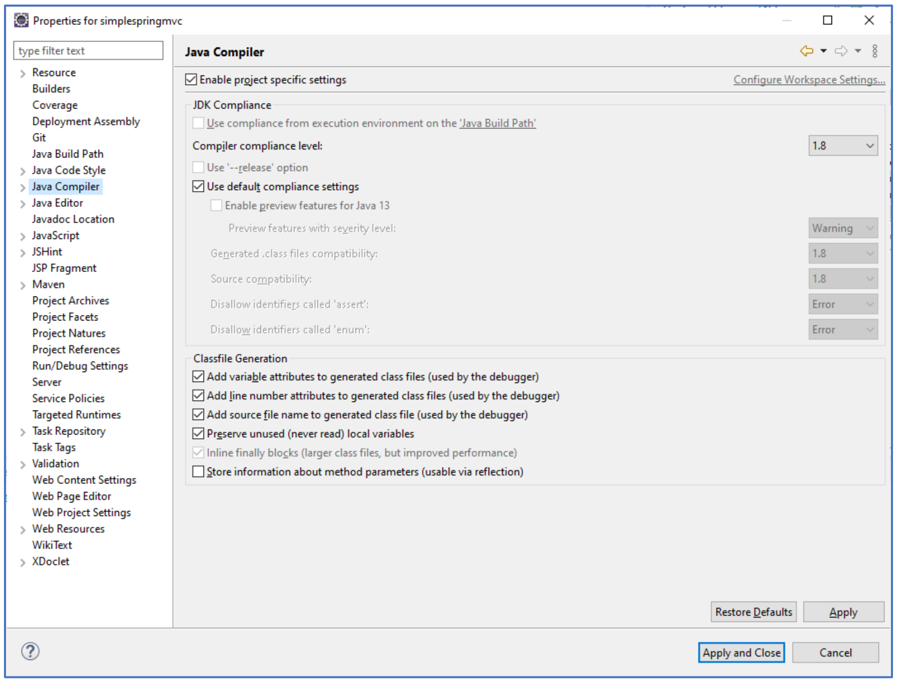


org.eclipse.wst.common.project.facet.core.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>
  <installed facet="java" version="1.8"/>
  <installed facet="jst.web" version="3.0"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
</faceted-project>
```

You should see the project facet changes reflected once you refresh the project.

on "**Java Compiler**" tab, make sure it set to same JVM Version



Properties for simplespringmvc

Java Compiler

☑ Enable project specific settings — Configure Workspace Settings...

JDK Compliance

☐ Use compliance from execution environment on the 'Java Build Path'

Compiler compliance level: 1.8

☐ Use '--release' option

☑ Use default compliance settings

☐ Enable preview features for Java 13

Preview features with severity level: Warning

Generated .class files compatibility: 1.8

Source compatibility: 1.8

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

☑ Add variable attributes to generated class files (used by the debugger)

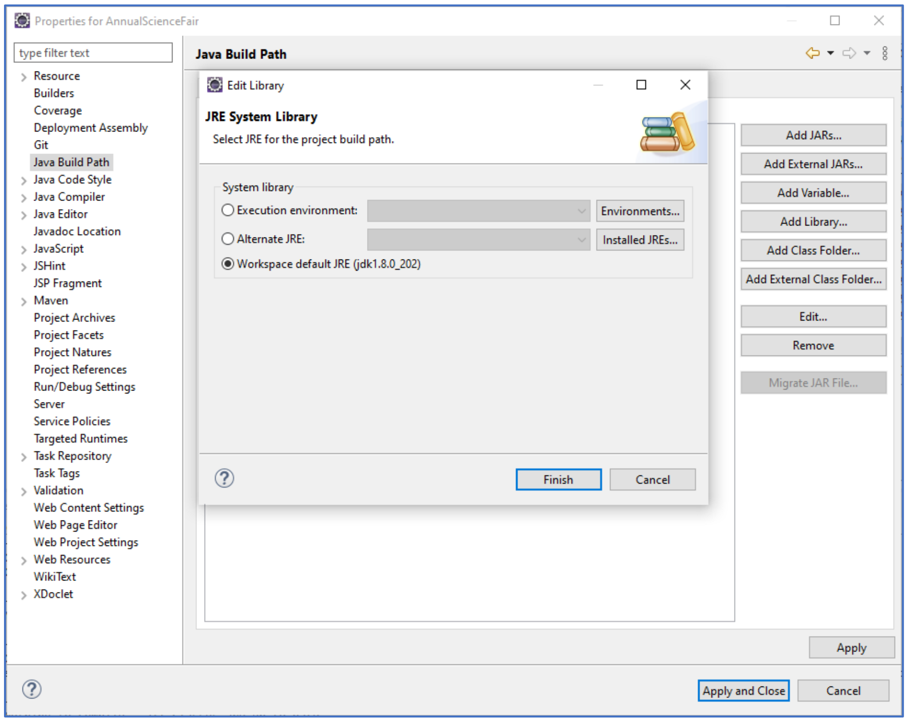
☑ Add line number attributes to generated class files (used by the debugger)

☑ Add source file name to generated class file (used by the debugger)

☑ Preserve unused (never read) local variables

☑ Inline finally blocks (larger class files, but improved performance)

☐ Store information about method parameters (usable via reflection) on “**Java Build Path**” tab, make sure it set to same JVM Version as well



JRE System Library

Select JRE for the project build path.

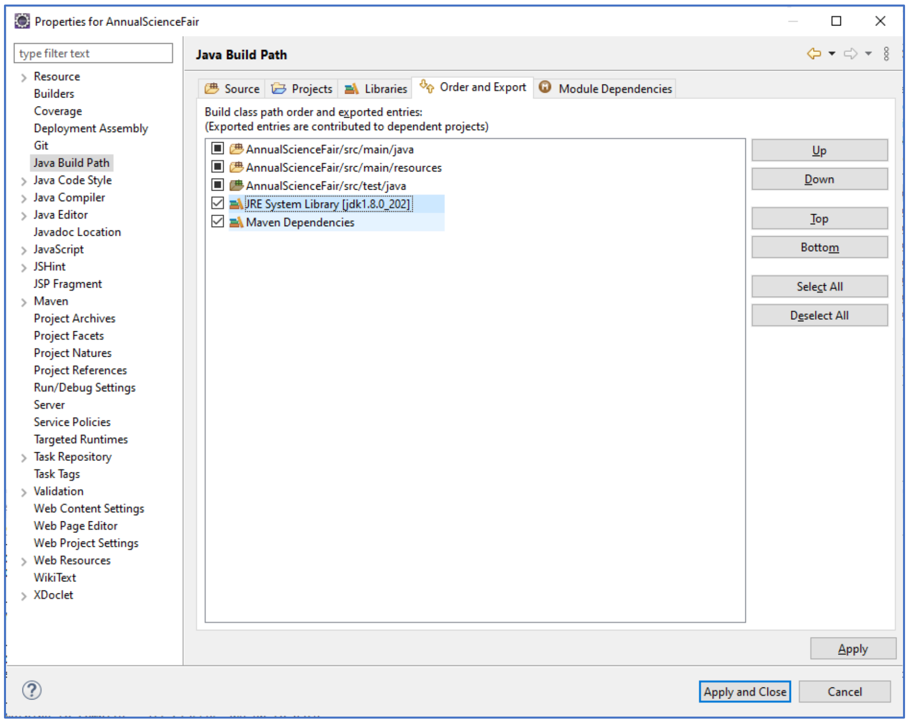
System library

( ) Execution environment:

( ) Alternate JRE:

(●) Workspace default JRE (jdk1.8.0_202)

also make sure that “**Maven Dependencies**” and “**JRE System Library”** is checked, click **Apply and Close**

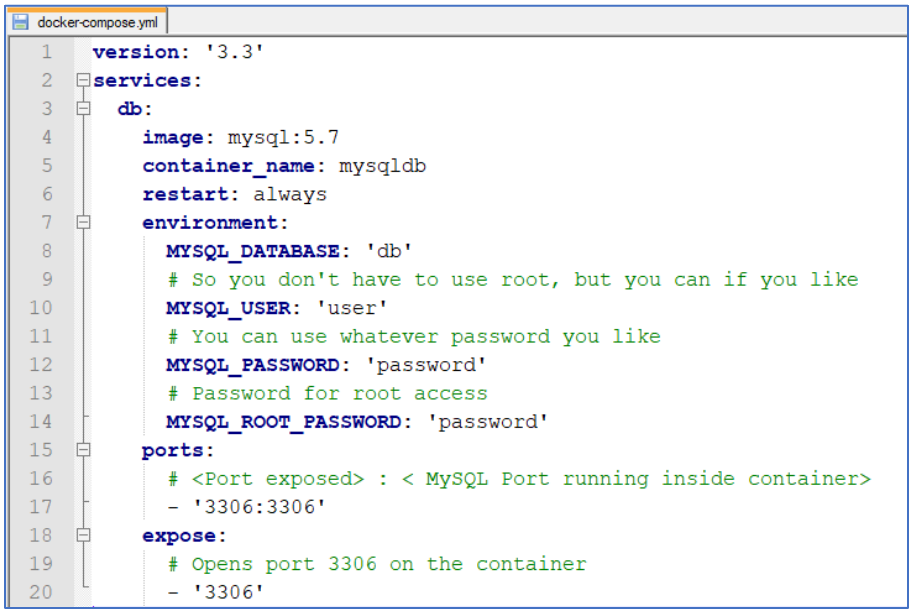


Build class path order and exported entries:
(Exported entries are contributed to dependent projects)

- ■ AnnualScienceFair/src/main/java
- ■ AnnualScienceFair/src/main/resources
- ■ AnnualScienceFair/src/test/java
- ✓ JRE System Library [jdk1.8.0_202]
- ✓ Maven Dependencies

Setup MySQL Database on Docker

Prepare docker-compose.yml with following content



```yaml
version: '3.3'
services:
  db:
    image: mysql:5.7
    container_name: mysqldb
    restart: always
    environment:
      MYSQL_DATABASE: 'db'
      # So you don't have to use root, but you can if you like
      MYSQL_USER: 'user'
      # You can use whatever password you like
      MYSQL_PASSWORD: 'password'
      # Password for root access
      MYSQL_ROOT_PASSWORD: 'password'
    ports:
      # <Port exposed> : < MySQL Port running inside container>
      - '3306:3306'
    expose:
      # Opens port 3306 on the container
      - '3306'
```

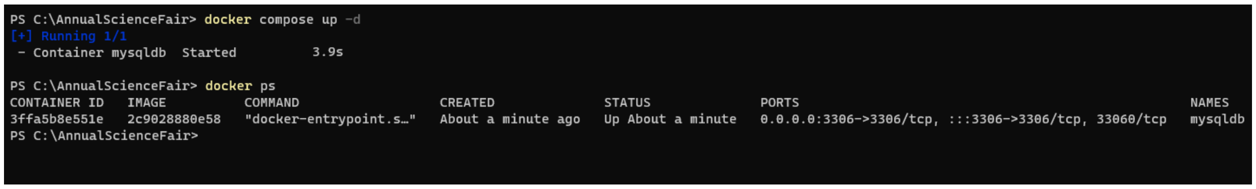
And run following command on terminal

```
$> docker compose up -d
```

To Check if docker container is running with following command

```
$> docker ps
```

Make sure the docker container is up and running



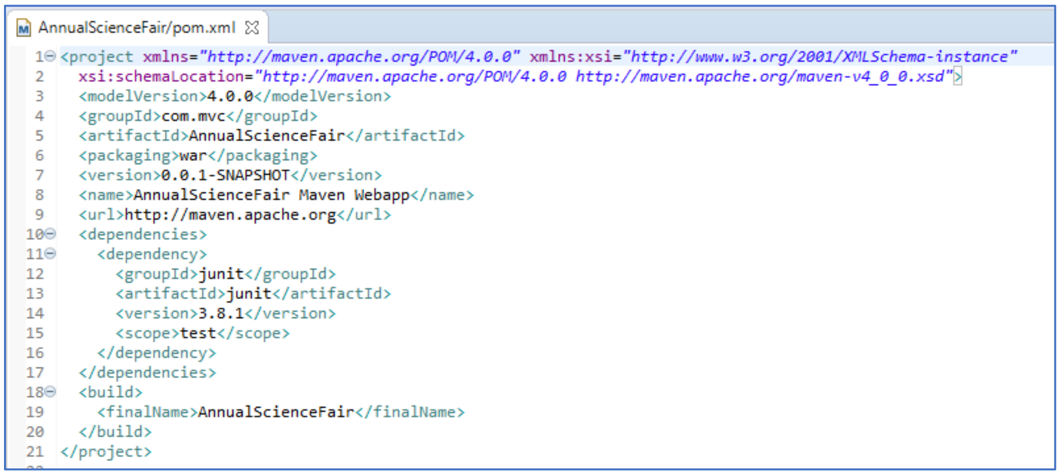
```
PS C:\AnnualScienceFair> docker compose up -d
[+] Running 1/1
 - Container mysqldb  Started          3.9s

PS C:\AnnualScienceFair> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS                                                  NAMES
3ffa5b8e551e   2c9028880e58   "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysqldb
PS C:\AnnualScienceFair>
```

Working With Starter Templates for an App

Starting with this video over the next few videos, we'll build up a complete Spring MVC application, complete with a very nice looking user interface and a connection to the database at the back end. All this app will do is allow the user to create an account which the user can then use to log in and register for an Annual Science Fair. But once you set up a few web pages for this application, you will find that adding additional pages just involves incremental additions to the basic Spring MVC app that we have set up.

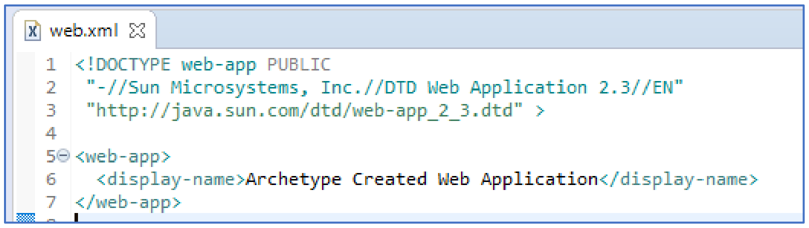
And when you make additions, you can build on the foundational concepts that we've covered so far in this learning path. Here we are on a brand new project for the AnnualScienceFair. That is going to be our web application.

Let's take a look at the pom.xml file. And we'll add dependencies to this file bit by bit. For now, all we have set up is a dependency on junit.


AnnualScienceFair/pom.xml

```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>com.mvc</groupId>
  <artifactId>AnnualScienceFair</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>AnnualScienceFair Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>AnnualScienceFair</finalName>
  </build>
</project>
```

There's nothing in the web.xml file as well. It contains the default values, just the display-name.

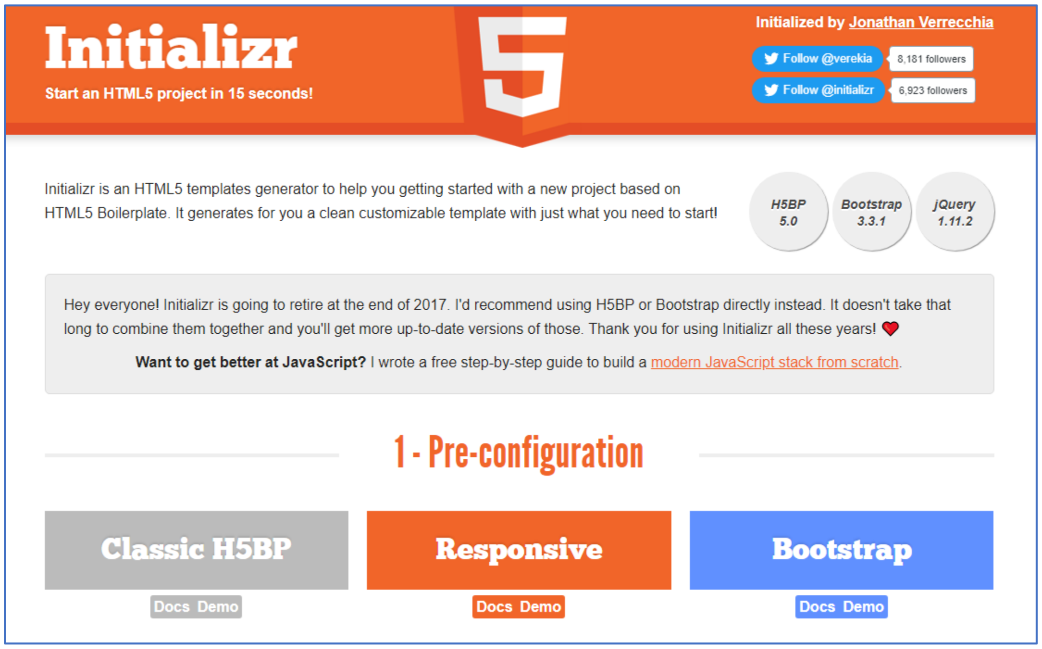

web.xml

```xml
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

At this point all I've done is set up a new Maven web application project. We've done this before. You're familiar with this. Now in order to build my web application, I want some nice starter code. This starter code will be in the form of a template. A template comprising of HTML and CSS files.

That takes care of the basic structure of our front end. Now there are many sites that support this, I've chosen to go with Initializr.com. You can see that it's an HTML5 templates generator to help you get started with a new project, giving you some boilerplate code so though you don't have to code from scratch. Based on what you want your app to use, there are different kinds of preconfigurations available. I'll choose the simple Bootstrap option and there are multiple additional optional parameters that you can have as well.

http://www.initializr.com/


Initializr

Start an HTML5 project in 15 seconds!

Initialized by Jonathan Verrecchia

Follow @verekia 8,181 followers

Follow @initializr 6,923 followers

Initializr is an HTML5 templates generator to help you getting started with a new project based on HTML5 Boilerplate. It generates for you a clean customizable template with just what you need to start!

H5BP 5.0 | Bootstrap 3.3.1 | jQuery 1.11.2

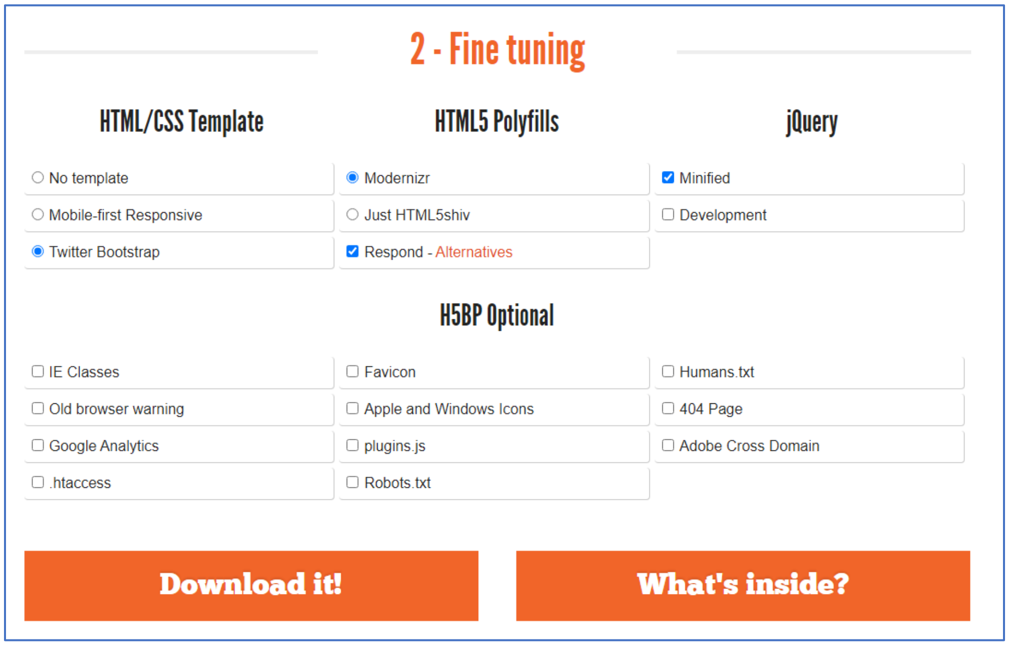
Hey everyone! Initializr is going to retire at the end of 2017. I'd recommend using H5BP or Bootstrap directly instead. It doesn't take that long to combine them together and you'll get more up-to-date versions of those. Thank you for using Initializr all these years!

Want to get better at JavaScript? I wrote a free step-by-step guide to build a modern JavaScript stack from scratch.

1 - Pre-configuration

Classic H5BP (Docs Demo) | Responsive (Docs Demo) | Bootstrap (Docs Demo)

I'm going to go ahead and unselect all of these I'm not interested in IE classes, Old browser warnings, and so on. And once I've made my selection, I'll hit Download it. And this will download a zip file with this boilerplate code onto my local machine.


2 - Fine tuning

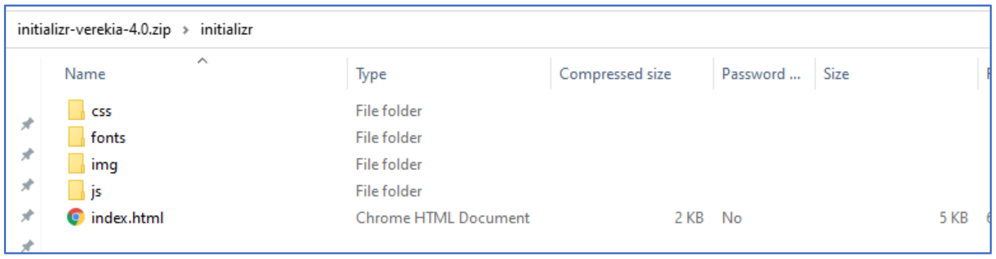
HTML/CSS Template	HTML5 Polyfills	jQuery
○ No template	● Modernizr	☑ Minified
○ Mobile-first Responsive	○ Just HTML5shiv	☐ Development
● Twitter Bootstrap	☑ Respond - Alternatives	

H5BP Optional

☐ IE Classes	☐ Favicon	☐ Humans.txt
☐ Old browser warning	☐ Apple and Windows Icons	☐ 404 Page
☐ Google Analytics	☐ plugins.js	☐ Adobe Cross Domain
☐ .htaccess	☐ Robots.txt	

Download it! | What's inside?

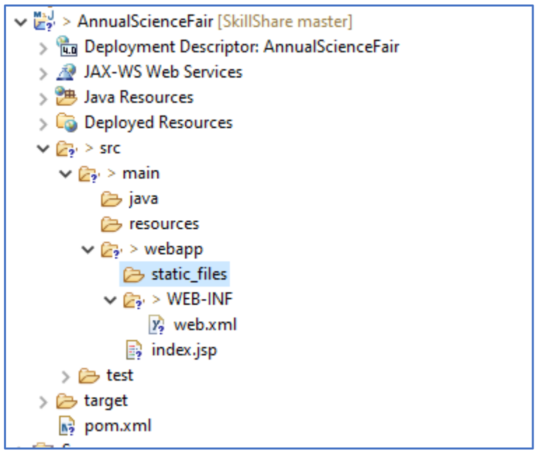
Now within our web application, I'm now going to create a folder that will hold the static_files that I'll use within the webapp. The static_files include CSS files, images, JavaScript files, and other assets that are not dynamically changed by our web application. Now within the webapp sub folder, I'm going to create a new folder and name it static_files. Remember, everything under your webapp sub folder is actually deployed as a part of your application.



initializr-verekia-4.0.zip › initializr

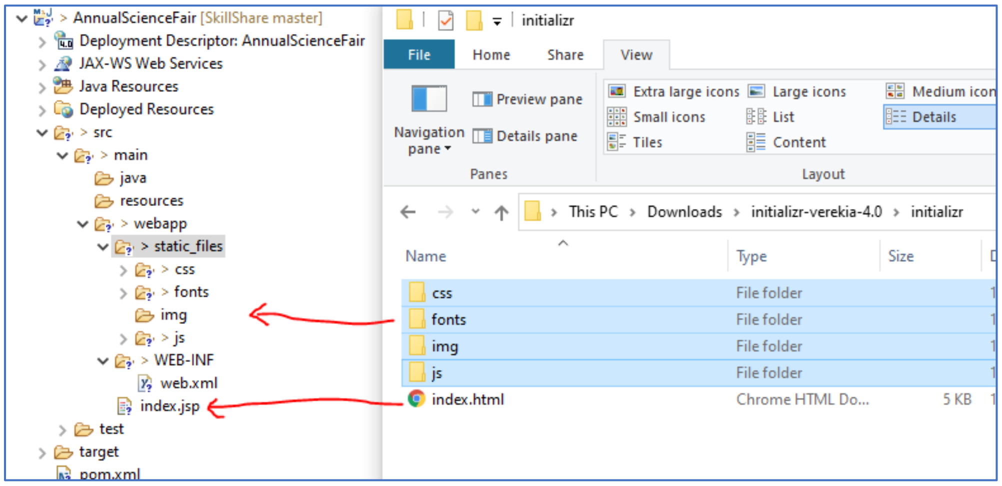
Name	Type	Compressed size	Password ...	Size
css	File folder			
fonts	File folder			
img	File folder			
js	File folder			
index.html	Chrome HTML Document	2 KB	No	5 KB

I'm now going to place my static assets In this static underscore files sub folder under the webapp folder. These static assets are available in the template files that we have downloaded. Here you can see the initializr-verekia-4.0.zip file, I've unzip that file.



```
AnnualScienceFair [SkillShare master]
  Deployment Descriptor: AnnualScienceFair
  JAX-WS Web Services
  Java Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        static_files
        WEB-INF
          web.xml
        index.jsp
    test
  target
  pom.xml
```

And those contents have been unzipped into the initializr directory that you see selected, click through to this initializr directory. And under there you will find four subdirectories CSS, fonts, image, and JavaScript. Simply copy over these four sub directories to your static_files directory, which is part of your web application. The static_files directory here is a part of your Eclipse project.



```
AnnualScienceFair [SkillShare master]
  Deployment Descriptor: AnnualScienceFair
  JAX-WS Web Services
  Java Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        static_files
          css
          fonts
          img
          js
        WEB-INF
          web.xml
        index.jsp
    test
  target
  pom.xml
```

This PC › Downloads › initializr-verekia-4.0 › initializr

Name	Type	Size
css	File folder	
fonts	File folder	
img	File folder	
js	File folder	
index.html	Chrome HTML Do...	5 KB

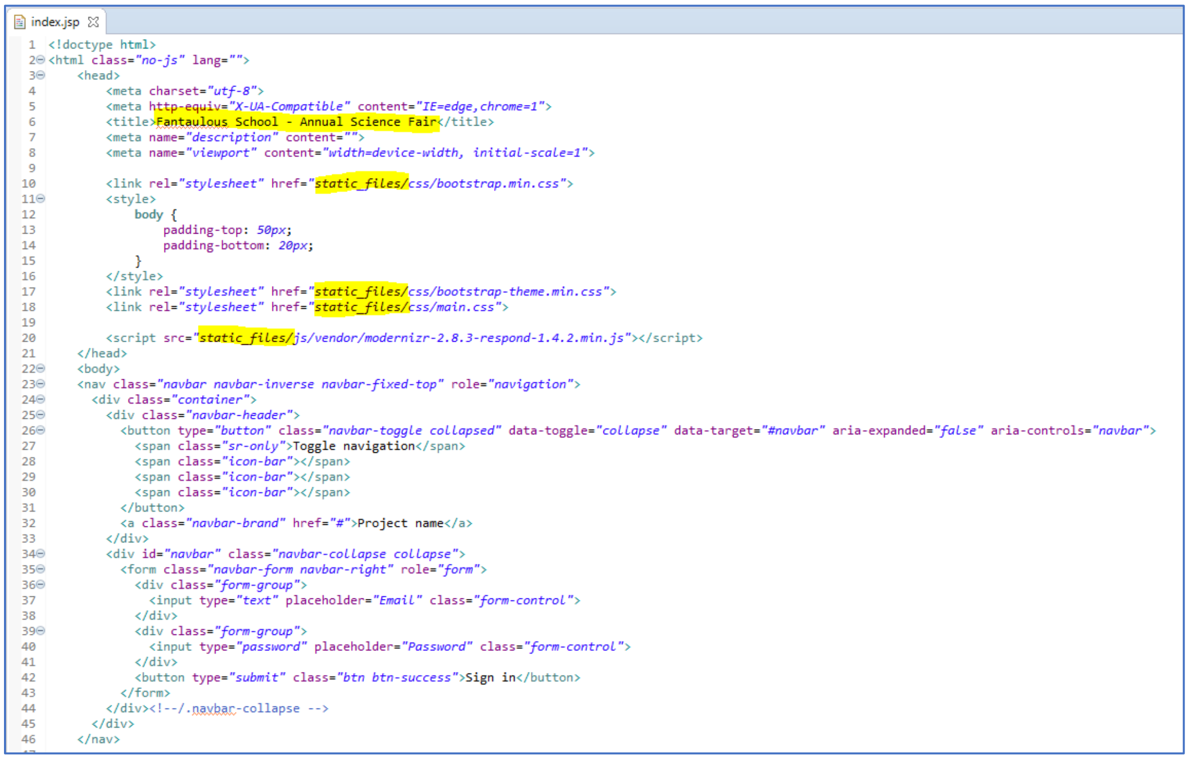
So under your Eclipse project for the AnnualScienceFair, go to the web app folder, go to static_files and copy those four subfolders over here. You don't have to worry about what's in those sub folders, simply copy them.

Remember, this is all part of the boilerplate code that we'll use for our app. Now to set up the main page of our application back to the folder where we had downloaded the boilerplate code. You'll notice the index.html file there, which gives us the main page of our application. We'll use this same template and tweak it a bit. Here's the index.html file and it contains a bunch of HTML code, and references to the CSS files and other JavaScript files for a responsive application.

I'm going to copy this code and paste it into my index.jsp file. This will give me some templated structure for the first page of my web application which I can then tweak based on my preferences.

Now this code will not work as is and the reason for that is all of the references will not be pointing to the right location for our specific application. We'll need to update these references so that they point to where our static_files live. And I'm going to do exactly that. And I'm also going to tweak this application so it represents our AnnualScienceFair.

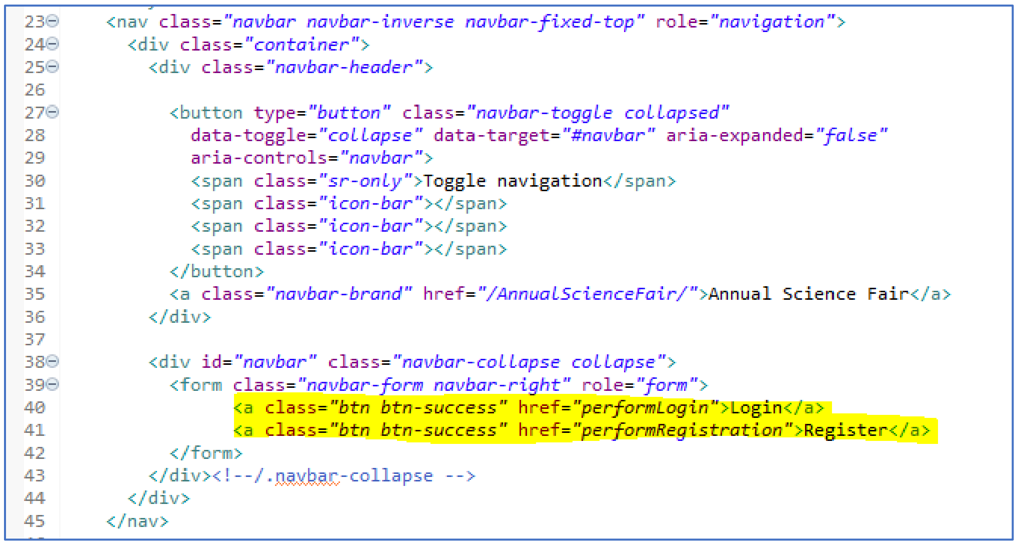
Let me highlight some of the changes I've made. Here on line six. I have the title Fantabulous Schools - Annual Science Fair, on lines 10, 17,18 and 20.


index.jsp

```
1  <!doctype html>
2  <html class="no-js" lang="">
3      <head>
4          <meta charset="utf-8">
5          <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
6          <title>Fantaulous School - Annual Science Fair</title>
7          <meta name="description" content="">
8          <meta name="viewport" content="width=device-width, initial-scale=1">
9
10         <link rel="stylesheet" href="static_files/css/bootstrap.min.css">
11         <style>
12             body {
13                 padding-top: 50px;
14                 padding-bottom: 20px;
15             }
16         </style>
17         <link rel="stylesheet" href="static_files/css/bootstrap-theme.min.css">
18         <link rel="stylesheet" href="static_files/css/main.css">
19
20         <script src="static_files/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>
21     </head>
22     <body>
23     <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
24       <div class="container">
25         <div class="navbar-header">
26           <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
27             <span class="sr-only">Toggle navigation</span>
28             <span class="icon-bar"></span>
29             <span class="icon-bar"></span>
30             <span class="icon-bar"></span>
31           </button>
32           <a class="navbar-brand" href="#">Project name</a>
33         </div>
34         <div id="navbar" class="navbar-collapse collapse">
35           <form class="navbar-form navbar-right" role="form">
36             <div class="form-group">
37               <input type="text" placeholder="Email" class="form-control">
38             </div>
39             <div class="form-group">
40               <input type="password" placeholder="Password" class="form-control">
41             </div>
42             <button type="submit" class="btn btn-success">Sign in</button>
43           </form>
44         </div><!--/.navbar-collapse -->
45       </div>
46     </nav>
```

Notice that I have updated all of the source and href links so that they point to the static_files under our webapp root directory.

This is where the template is stylesheets live, and the JavaScript for our responsive web application. If you scroll down below, you'll find the HTML for the navigation bar, which will be at the top of our application.



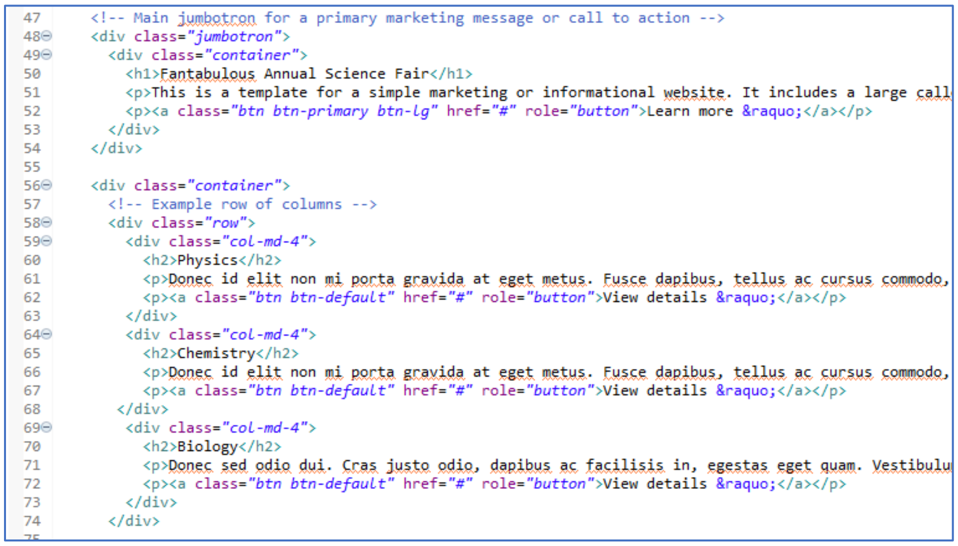
```
23    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
24      <div class="container">
25        <div class="navbar-header">
26
27          <button type="button" class="navbar-toggle collapsed"
28            data-toggle="collapse" data-target="#navbar" aria-expanded="false"
29            aria-controls="navbar">
30            <span class="sr-only">Toggle navigation</span>
31            <span class="icon-bar"></span>
32            <span class="icon-bar"></span>
33            <span class="icon-bar"></span>
34          </button>
35          <a class="navbar-brand" href="/AnnualScienceFair/">Annual Science Fair</a>
36        </div>
37
38        <div id="navbar" class="navbar-collapse collapse">
39          <form class="navbar-form navbar-right" role="form">
40                <a class="btn btn-success" href="performLogin">Login</a>
41                <a class="btn btn-success" href="performRegistration">Register</a>
42          </form>
43        </div><!--/.navbar-collapse -->
44      </div>
45    </nav>
```

On line 35, you can see an href link which points to the Annual Science Fair. This is the homepage link that'll always take us back to our homepage from the other pages in our app.

On lines 40 and 41 you can see two buttons on the navigation bar that we'll wire up in our demos Login and Register.

```
<a class="btn btn-success" href="performLogin">Login</a>
<a class="btn btn-success" href="performRegistration">Register</a>
```

If you scroll down further, you'll see the HTML and contents of the main body of the page. There is a header which says Fantabulous Annual Science Fair. Some details about the science fair and if you scroll further down, you will see the three categories in which projects are accepted Physics, Chemistry, and Biology.



```
    <!-- Main jumbotron for a primary marketing message or call to action -->
    <div class="jumbotron">
      <div class="container">
        <h1>Fantabulous Annual Science Fair</h1>
        <p>This is a template for a simple marketing or informational website. It includes a large call
        <p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more &raquo;</a></p>
      </div>
    </div>

    <div class="container">
      <!-- Example row of columns -->
      <div class="row">
        <div class="col-md-4">
          <h2>Physics</h2>
          <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo,
          <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
        </div>
        <div class="col-md-4">
          <h2>Chemistry</h2>
          <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo,
          <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
       </div>
        <div class="col-md-4">
          <h2>Biology</h2>
          <p>Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulu
          <p><a class="btn btn-default" href="#" role="button">View details &raquo;</a></p>
        </div>
      </div>
```

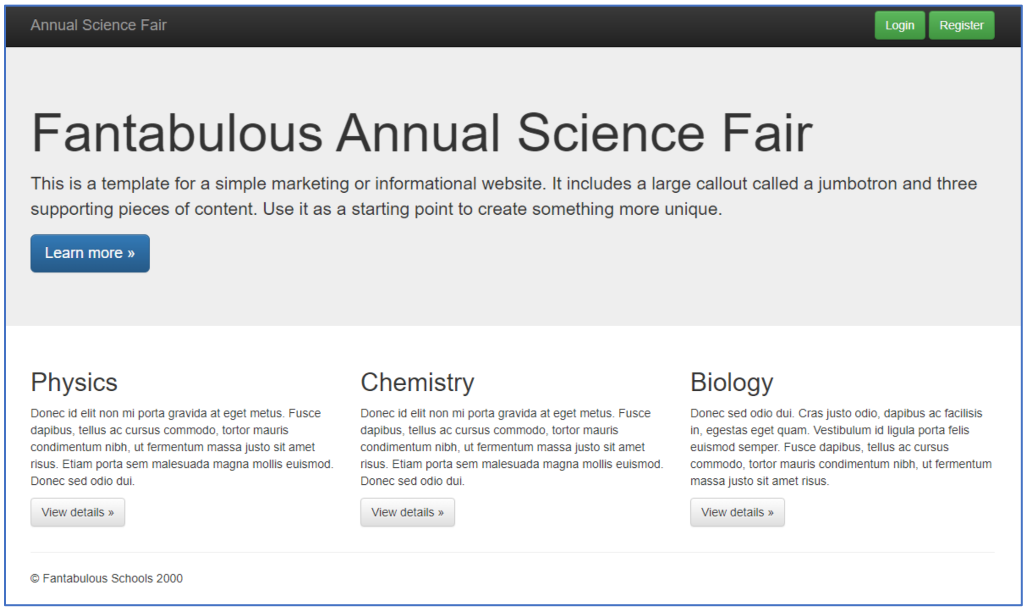
And finally, we have the footer at the very bottom.

```
<footer>
  <p>&copy; Fantabulous Schools 2000</p>.
</footer>
```

You can see on lines 93 and 95 that we've updated the source references to point to the static_files folder.

```
<script src="static_files/js/vendor/bootstrap.min.js"></script>
<script src="static_files/js/main.js"></script>
```

Let's quickly see what this basic application looks like. Note that we haven't added any Spring MVC components yet. This is what our first page will look like. Here is our Annual Science Fair.



Fantabulous Annual Science Fair

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

Learn more »

Physics

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

View details »

Chemistry

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

View details »

Biology

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

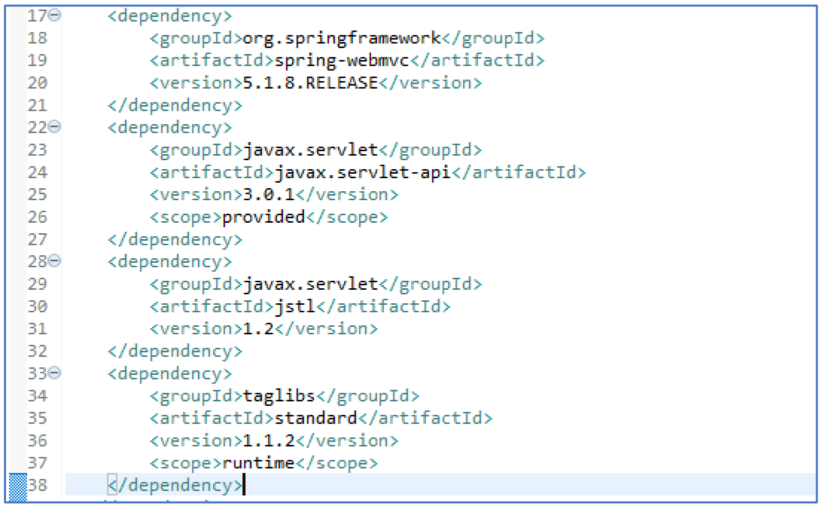
View details »

© Fantabulous Schools 2000

On the top left is the link that will always bring us back to this home page. On the top right, we have the Login and Register buttons. Then we have some info about the science fair. And at the bottom, we have info about the specific categories in which projects are accepted. You can try out the Login and Register buttons. They do nothing. They just bring you back to the current page. They haven't been wired up yet. We'll do the wiring over the course of the next section.

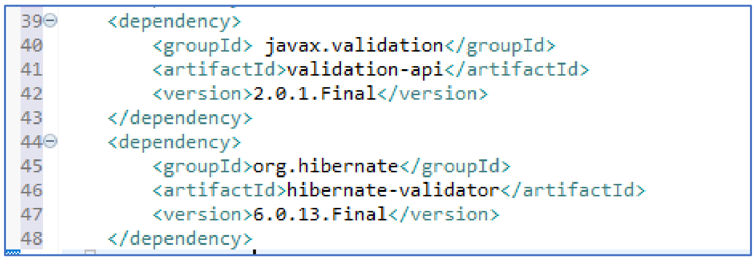
Setting up Login and Register Forms

So far, we've set up the starter templates for our application. In this demo, we'll continue building this application. The first step will be to make it a Spring MVC application. We'll set up the Login and Register forms as well. The first step to make this a Spring MVC application is to set up the right dependencies in the **pom.xml** file. Make sure you depend on the spring-webmvc library.



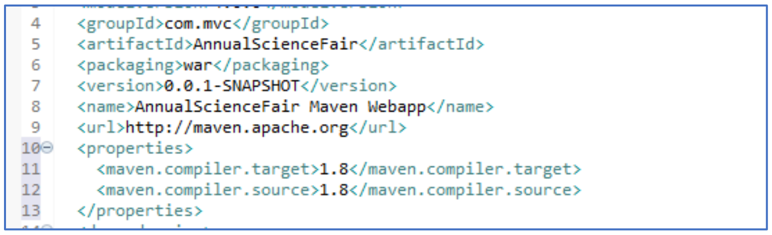
```xml
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.1.8.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.0.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
        <scope>runtime</scope>
    </dependency>
```

We've also added dependencies for the javax.servlet-api, jstl, and the taglibs library.



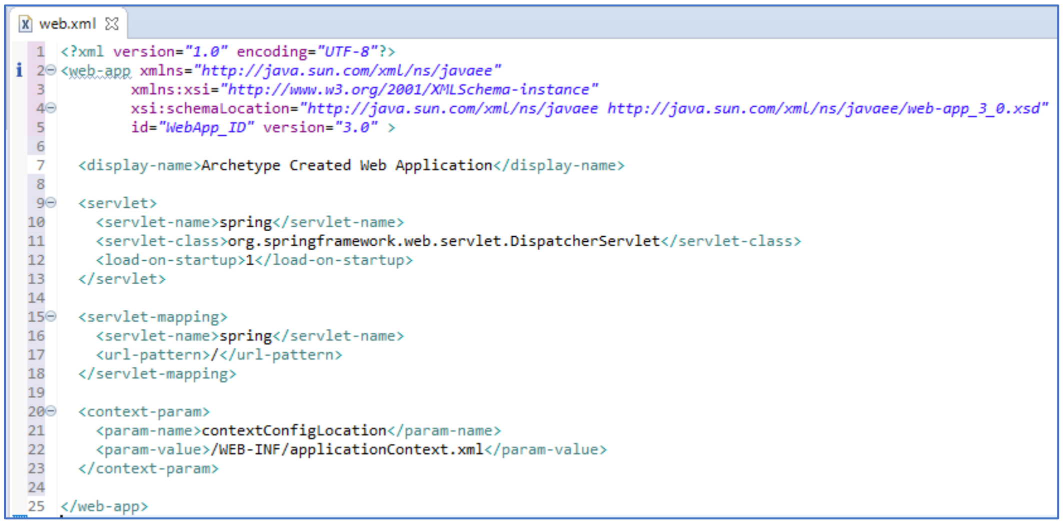
```xml
    <dependency>
        <groupId> javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>2.0.1.Final</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.13.Final</version>
    </dependency>
```

We'll also be performing some form validation in our application. So I've added dependencies for the hibernate.validator and the validation-api. Plus, when you're working with the Maven compiler, it's always good practice to explicitly specify the Java version you want be able to compile with.
Here I've specified the source and target version as 1.8.



```xml
  <groupId>com.mvc</groupId>
  <artifactId>AnnualScienceFair</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>AnnualScienceFair Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
  </properties>
```

Here is the web.xml file in the WEB-INF folder.



```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
 3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
 5         id="WebApp_ID" version="3.0" >
 6
 7   <display-name>Archetype Created Web Application</display-name>
 8
 9   <servlet>
10     <servlet-name>spring</servlet-name>
11     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
12     <load-on-startup>1</load-on-startup>
13   </servlet>
14
15   <servlet-mapping>
16     <servlet-name>spring</servlet-name>
17     <url-pattern>/</url-pattern>
18   </servlet-mapping>
19
20   <context-param>
21     <param-name>contextConfigLocation</param-name>
22     <param-value>/WEB-INF/applicationContext.xml</param-value>
23   </context-param>
24
25 </web-app>
```

This is where I specify the DispatcherServlet, that will be the front controller for the application. This is on line 11.

```
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

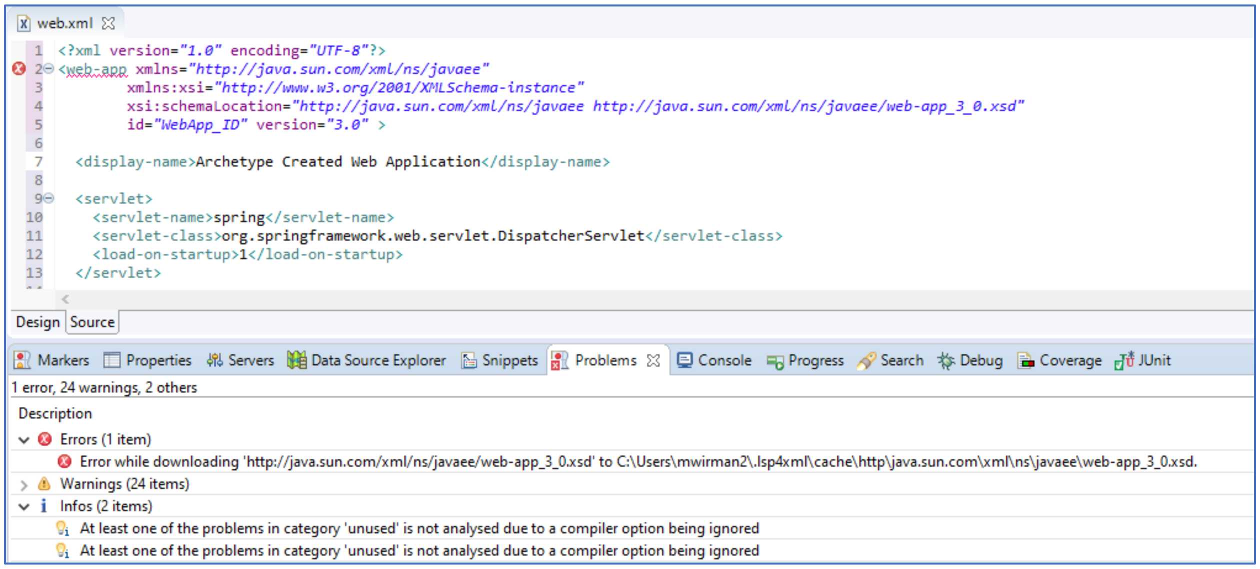
My servlet-mapping has the name spring and matches the url-pattern forward slash.

```
<servlet-mapping>
	<servlet-name>spring</servlet-name>
	<url-pattern>/</url-pattern>
</servlet-mapping>
```

I've also explicitly specified a contextConfigLocation. This tells Spring where to look for additional configuration files, though I've placed my files in the default location in the WEB-INF folder. That's where spring-servlet.xml lives.

```
<param-value>/WEB-INF/applicationContext.xml</param-value>
```

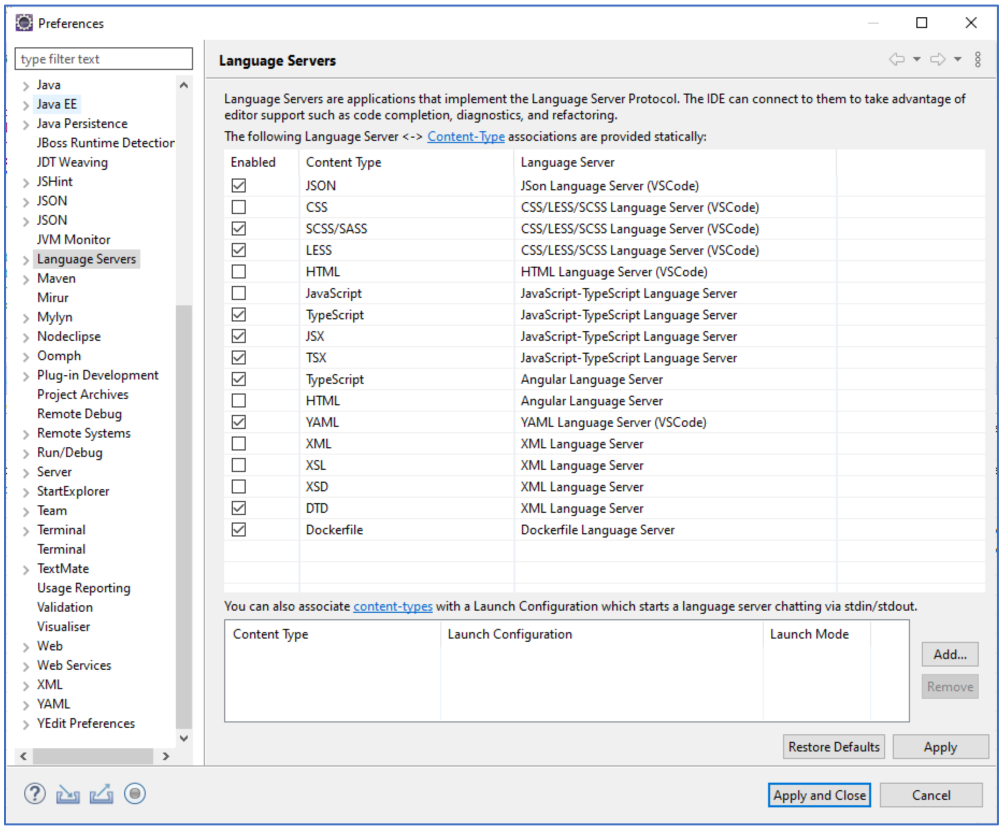
If for some reason your eclipse showing you following error:



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0" >

  <display-name>Archetype Created Web Application</display-name>

  <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

```
1 error, 24 warnings, 2 others
Description
Errors (1 item)
    Error while downloading 'http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd' to C:\Users\mwirman2\.lsp4xml\cache\http\java.sun.com\xml\ns\javaee\web-app_3_0.xsd.
Warnings (24 items)
Infos (2 items)
    At least one of the problems in category 'unused' is not analysed due to a compiler option being ignored
    At least one of the problems in category 'unused' is not analysed due to a compiler option being ignored
```

Try to disable Language Server errors : into Preferences -> Language Servers and turn them all off.



Language Servers

Language Servers are applications that implement the Language Server Protocol. The IDE can connect to them to take advantage of editor support such as code completion, diagnostics, and refactoring.

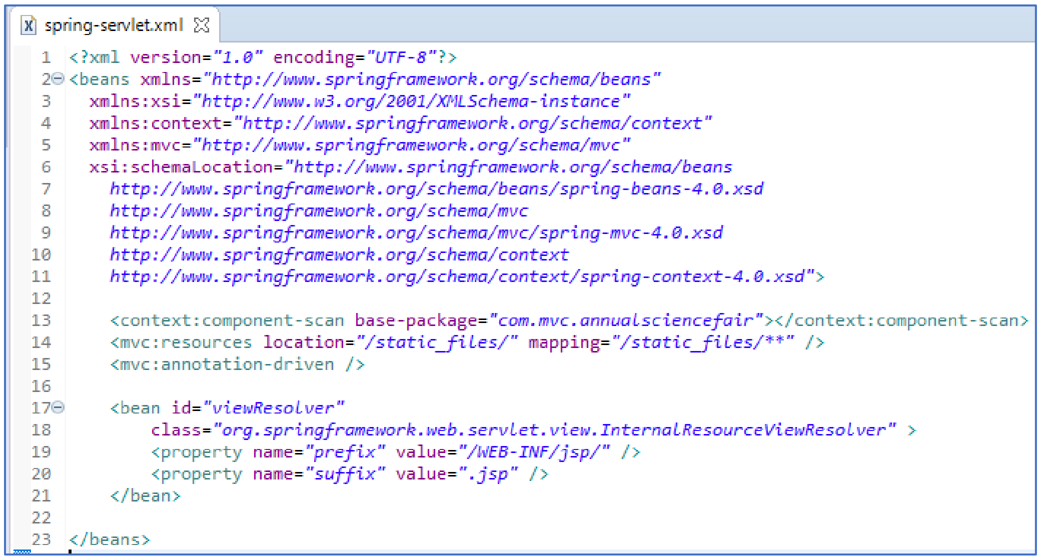
The following Language Server <-> Content-Type associations are provided statically:

Enabled	Content Type	Language Server
✓	JSON	JSon Language Server (VSCode)
	CSS	CSS/LESS/SCSS Language Server (VSCode)
✓	SCSS/SASS	CSS/LESS/SCSS Language Server (VSCode)
✓	LESS	CSS/LESS/SCSS Language Server (VSCode)
	HTML	HTML Language Server (VSCode)
	JavaScript	JavaScript-TypeScript Language Server
✓	TypeScript	JavaScript-TypeScript Language Server
✓	JSX	JavaScript-TypeScript Language Server
✓	TSX	JavaScript-TypeScript Language Server
✓	TypeScript	Angular Language Server
	HTML	Angular Language Server
✓	YAML	YAML Language Server (VSCode)
	XML	XML Language Server
	XSL	XML Language Server
	XSD	XML Language Server
✓	DTD	XML Language Server
✓	Dockerfile	Dockerfile Language Server

You can also associate content-types with a Launch Configuration which starts a language server chatting via stdin/stdout.

Content Type	Launch Configuration	Launch Mode

Let's take a look at spring-servlet-xml.



```xml
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <beans xmlns="http://www.springframework.org/schema/beans"
 3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4   xmlns:context="http://www.springframework.org/schema/context"
 5   xmlns:mvc="http://www.springframework.org/schema/mvc"
 6   xsi:schemaLocation="http://www.springframework.org/schema/beans
 7     http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
 8     http://www.springframework.org/schema/mvc
 9     http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
10     http://www.springframework.org/schema/context
11     http://www.springframework.org/schema/context/spring-context-4.0.xsd">
12
13     <context:component-scan base-package="com.mvc.annualsciencefair"></context:component-scan>
14     <mvc:resources location="/static_files/" mapping="/static_files/**" />
15     <mvc:annotation-driven />
16
17     <bean id="viewResolver"
18         class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
19         <property name="prefix" value="/WEB-INF/jsp/" />
20         <property name="suffix" value=".jsp" />
21     </bean>
22
23 </beans>
```

All of the configuration here should be familiar to us except for mvc : resources.

```xml
<mvc:resources location="/static_files/" mapping="/static_files/**"/>
```

Let's look at the other ones first. We have the component-scan. Spring will scan the base-package="`com.mvc.annualsciencefair`" and all sub-packages of this base package to pick up injectable components, controllers, repositories, services, beans, and so on.

```xml
<context:component-scan
base-package="com.mvc.annualsciencefair"></context:component-scan>
```

The annotation-driven tag will ensure that components which are marked using Java annotations, specifically Spring MVC annotations will be identified and injected.

```xml
<mvc:annotation-driven />
```

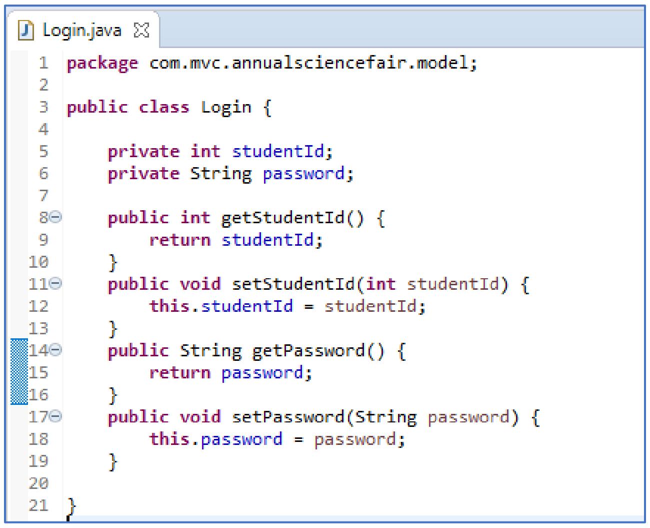
Then I have a bean specification for the InternalResourceViewResolver. All of my views are jsp files that will be in the WEB-INF/jsp folder.

```xml
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
       <property name="prefix" value="/WEB-INF/jsp/" />
       <property name="suffix" value=".jsp" />
</bean>
```

The mvc: resources tag that we have specified on line 15 tells our Spring web application where to look for static resources such as CSS files, JavaScript images, and other assets.

Our mapping attributes says /static_files/**. This means any reference to URLs that start with static_files and refer any sub-folder within it should be mapped to the location /static_files/. The location attribute refers to the physical location of the files on the web server, which is in the static_files folder under the root directory of our application.

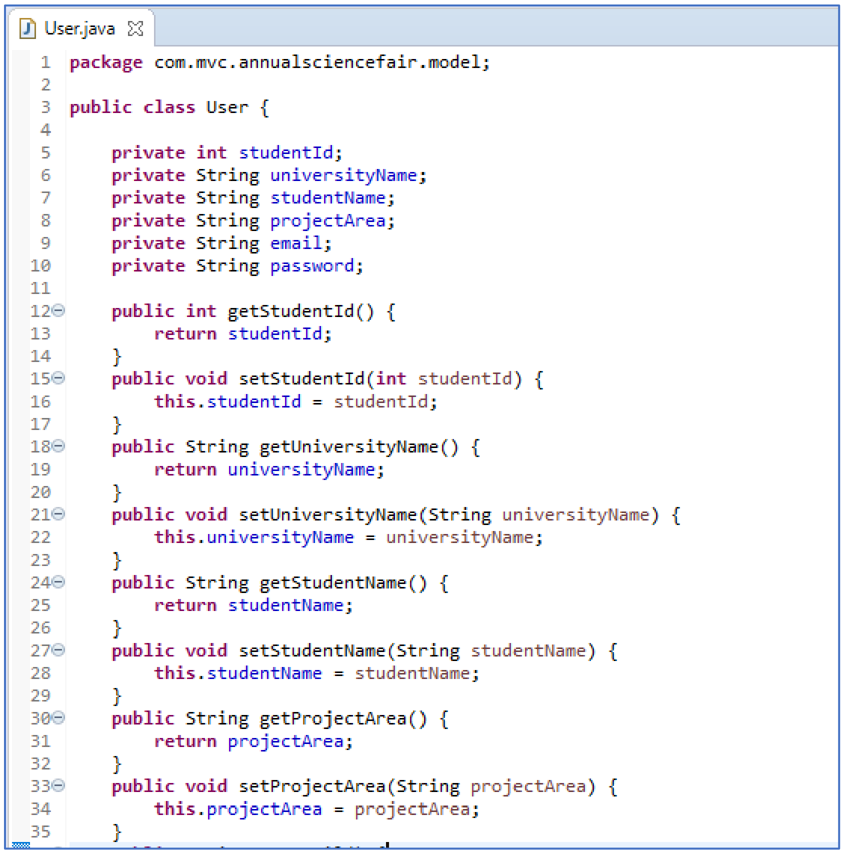
Moving on from configuration files, let's take a look at the model that our application will use. We have two model classes. The first of these is Login.java.



```java
package com.mvc.annualsciencefair.model;

public class Login {

    private int studentId;
    private String password;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

}
```

Login captures all of the information that a user uses to log into the app, the studentId and a password. The studentId is an integer. The password is a string. And this login model object has getters and setters for both of these member variables.

We have another model object that is User.java. The class User represents the model object that a user uses to register for the annual science fair.

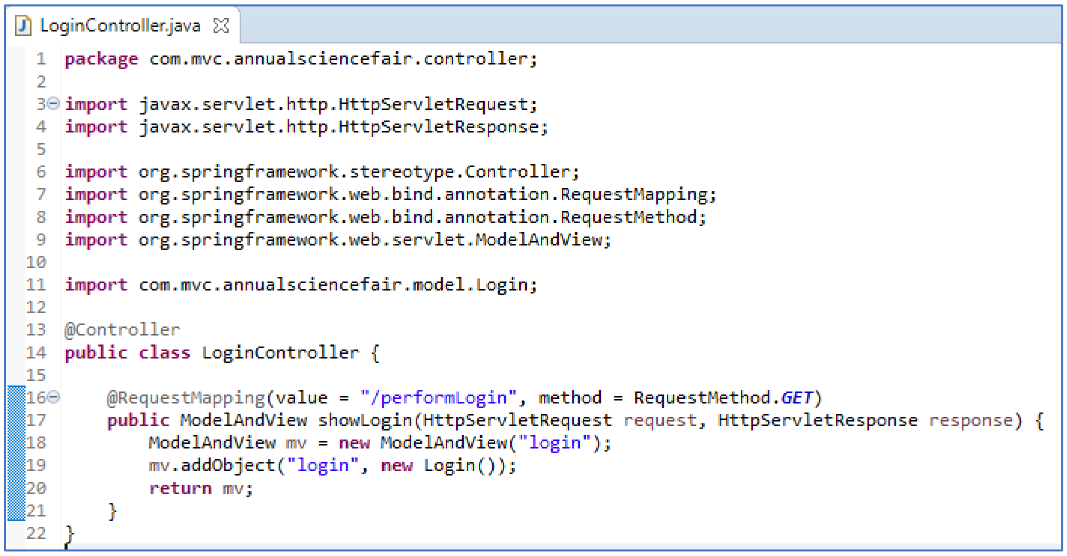


```java
package com.mvc.annualsciencefair.model;

public class User {

    private int studentId;
    private String universityName;
    private String studentName;
    private String projectArea;
    private String email;
    private String password;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getUniversityName() {
        return universityName;
    }
    public void setUniversityName(String universityName) {
        this.universityName = universityName;
    }
    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    public String getProjectArea() {
        return projectArea;
    }
    public void setProjectArea(String projectArea) {
        this.projectArea = projectArea;
    }
```

So there are a number of details that we capture during the registration process, the studentId, the universityName, the studentName, the projectArea, email, and the password for the login. Once the student has registered, the student will be able to log in with the student ID and password. The remaining code for this user object are getters and setters for all of the member variables that are present in the User.java file.

Just a note here that both of the model objects, login as well as user should be serializable. A way to make this explicit is by implementing the serializable interface for both of them. I haven't done that. But you can see that both of these objects contain either primitive types or strings and are serializable.

Next, let's take a look at the **LoginController** which for now contains the code that displays the login form to the user.



```
 1 package com.mvc.annualsciencefair.controller;
 2
 3 import javax.servlet.http.HttpServletRequest;
 4 import javax.servlet.http.HttpServletResponse;
 5
 6 import org.springframework.stereotype.Controller;
 7 import org.springframework.web.bind.annotation.RequestMapping;
 8 import org.springframework.web.bind.annotation.RequestMethod;
 9 import org.springframework.web.servlet.ModelAndView;
10
11 import com.mvc.annualsciencefair.model.Login;
12
13 @Controller
14 public class LoginController {
15
16     @RequestMapping(value = "/performLogin", method = RequestMethod.GET)
17     public ModelAndView showLogin(HttpServletRequest request, HttpServletResponse response) {
18         ModelAndView mv = new ModelAndView("login");
19         mv.addObject("login", new Login());
20         return mv;
21     }
22 }
```

We haven't written the code yet actually performing the login. LoginController is tagged using the **@Controller** annotation. There is one method here with the RequestMapping path of **/performLogin**.

```
@RequestMapping(value = "/performLogin", method = RequestMethod.GET)
```

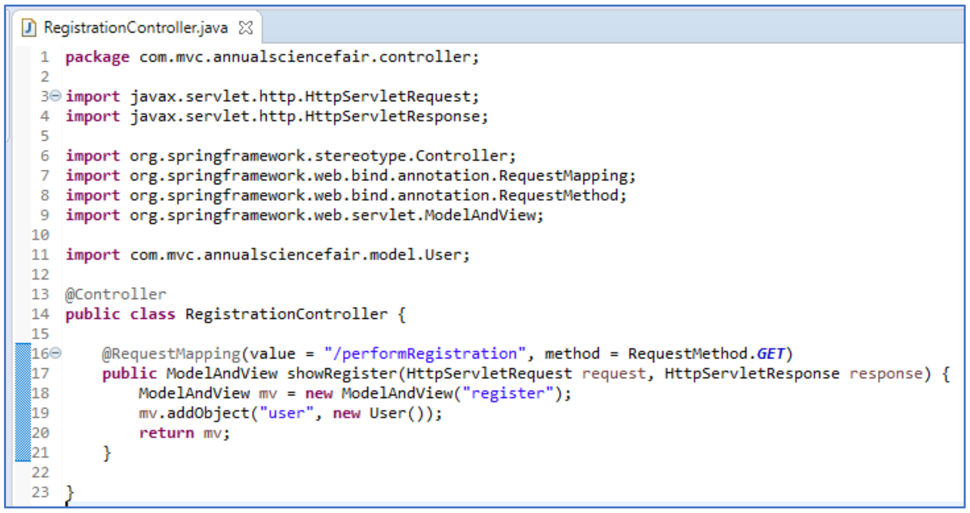
This handler responds to HTTP GET Request. We have explicitly specified that RequestMethod.GET. Within this method, we instantiate a new ModelAndView object. The logical name of our model is login, and this maps to the login.jsp file.

```
ModelAndView mv = new ModelAndView("login");
```

We also instantiate a new login object on line 18. And we add that login object to the ModelAndView.

```
mv.addObject("login", new Login());
```

This object will be the model backing the login form. I've defined a second controller here within the controller package, the **RegistrationController**.



RegistrationController.java

```java
package com.mvc.annualsciencefair.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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.mvc.annualsciencefair.model.User;

@Controller
public class RegistrationController {

    @RequestMapping(value = "/performRegistration", method = RequestMethod.GET)
    public ModelAndView showRegister(HttpServletRequest request, HttpServletResponse response) {
        ModelAndView mv = new ModelAndView("register");
        mv.addObject("user", new User());
        return mv;
    }

}
```

There is one method that I've defined here. We'll define more later. The @RequestMapping is to the path /performRegistration. And this responds to HTTP GET methods.

```java
@RequestMapping(value = "/performRegistration", method = RequestMethod.GET)
```

We instantiate a ModelAndView object. The logical name of our view is register and it corresponds to our register.jsp file.

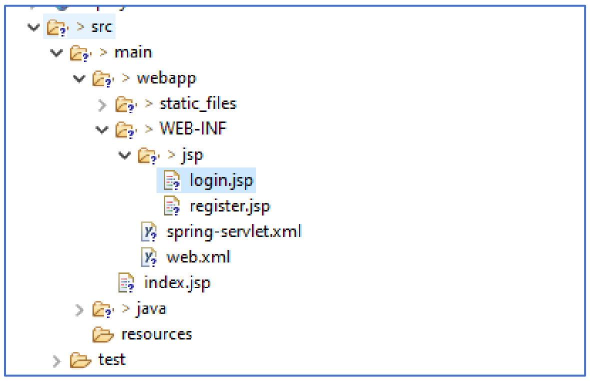
```java
ModelAndView mv = new ModelAndView("register");
```

We add a new User object that's going to be the model backing this view.

```java
mv.addObject("user", new User());
```

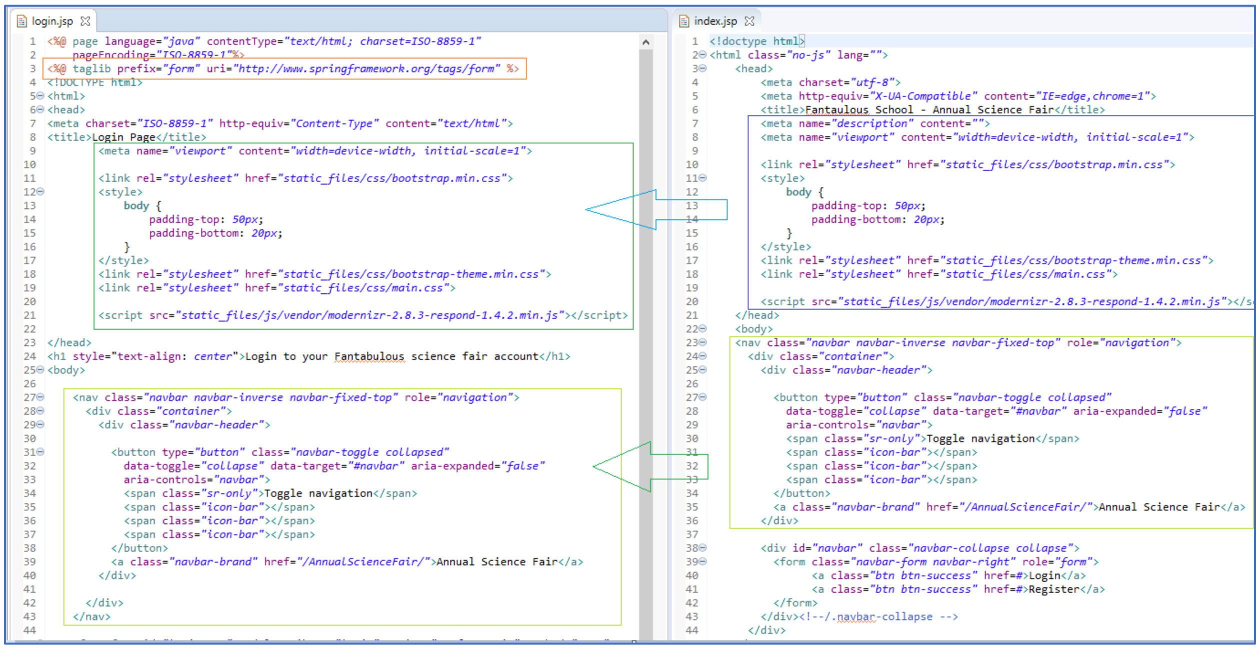
And we return this ModelAndView.

Time to take a look at the view pages. Let's take a look at the **login.jsp file**, the page that users will use to log in.



```
> src
  > main
    > webapp
      > static_files
      > WEB-INF
        > jsp
            login.jsp
            register.jsp
          spring-servlet.xml
          web.xml
        index.jsp
    > java
      resources
  test
```

Now I have a bunch of stuff here in the head tag of this file. This basically is all of the theme-specific HTML and CSS. I've simply copied over the references from the main **index.jsp** page so that all pages have the same theme.



login.jsp

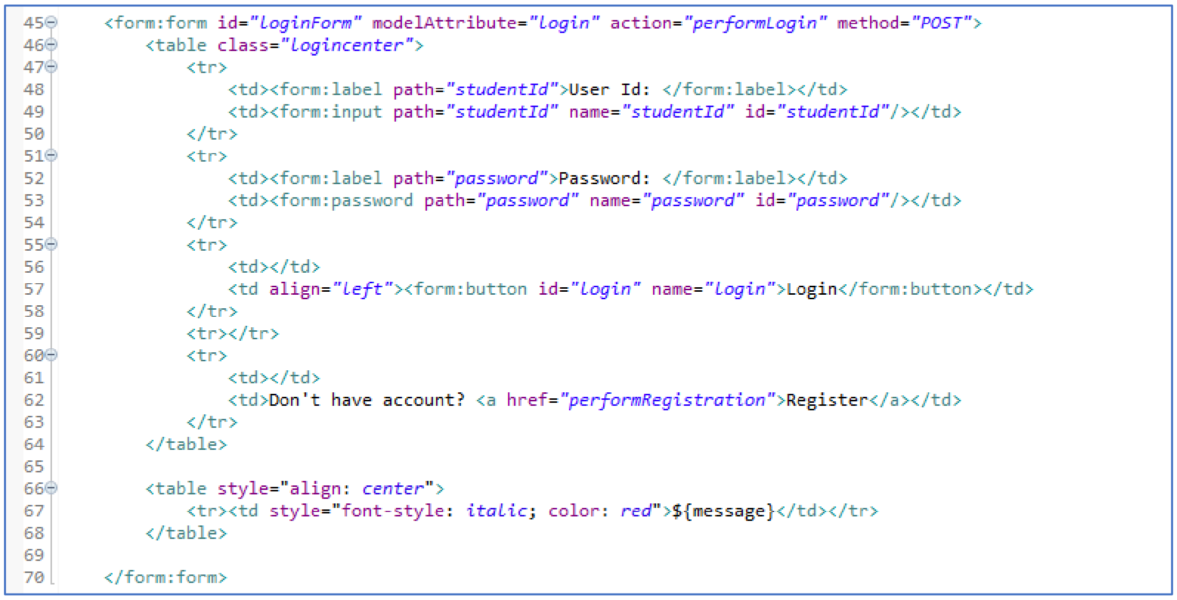
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1" http-equiv="Content-Type" content="text/html">
<title>Login Page</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="static_files/css/bootstrap.min.css">
        <style>
            body {
                padding-top: 50px;
                padding-bottom: 20px;
            }
        </style>
        <link rel="stylesheet" href="static_files/css/bootstrap-theme.min.css">
        <link rel="stylesheet" href="static_files/css/main.css">

        <script src="static_files/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>
<h1 style="text-align: center">Login to your Fantabulous science fair account</h1>
<body>

    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">

          <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse" data-target="#navbar" aria-expanded="false"
            aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="/AnnualScienceFair/">Annual Science Fair</a>
        </div>

      </div>
    </nav>

```

index.jsp

```
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <title>Fantaulous School - Annual Science Fair</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="static_files/css/bootstrap.min.css">
        <style>
            body {
                padding-top: 50px;
                padding-bottom: 20px;
            }
        </style>
        <link rel="stylesheet" href="static_files/css/bootstrap-theme.min.css">
        <link rel="stylesheet" href="static_files/css/main.css">

        <script src="static_files/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></s
    </head>
    <body>
    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">

          <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse" data-target="#navbar" aria-expanded="false"
            aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="/AnnualScienceFair/">Annual Science Fair</a>
        </div>

        <div id="navbar" class="navbar-collapse collapse">
          <form class="navbar-form navbar-right" role="form">
                <a class="btn btn-success" href=#>Login</a>
                <a class="btn btn-success" href=#>Register</a>
          </form>
        </div><!--/.navbar-collapse -->
      </div>
```

If you scroll down below, you'll be able to see the loginForm.

login.jsp



```
45      <form:form id="loginForm" modelAttribute="login" action="performLogin" method="POST">
46          <table class="logincenter">
47              <tr>
48                  <td><form:label path="studentId">User Id: </form:label></td>
49                  <td><form:input path="studentId" name="studentId" id="studentId"/></td>
50              </tr>
51              <tr>
52                  <td><form:label path="password">Password: </form:label></td>
53                  <td><form:password path="password" name="password" id="password"/></td>
54              </tr>
55              <tr>
56                  <td></td>
57                  <td align="left"><form:button id="login" name="login">Login</form:button></td>
58              </tr>
59              <tr></tr>
60              <tr>
61                  <td></td>
62                  <td>Don't have account? <a href="performRegistration">Register</a></td>
63              </tr>
64          </table>
65
66          <table style="align: center">
67              <tr><td style="font-style: italic; color: red">${message}</td></tr>
68          </table>
69
70      </form:form>
```

The loginForm is the jsp form created using the taglibs form control. Its id is loginForm. The modelAttribute is login. This references the model object that backs this form. The action performed on form submission will be performLogin method is post.

On line 49, we have an input element with the path studentId, which maps to the studentId variable of the login object.

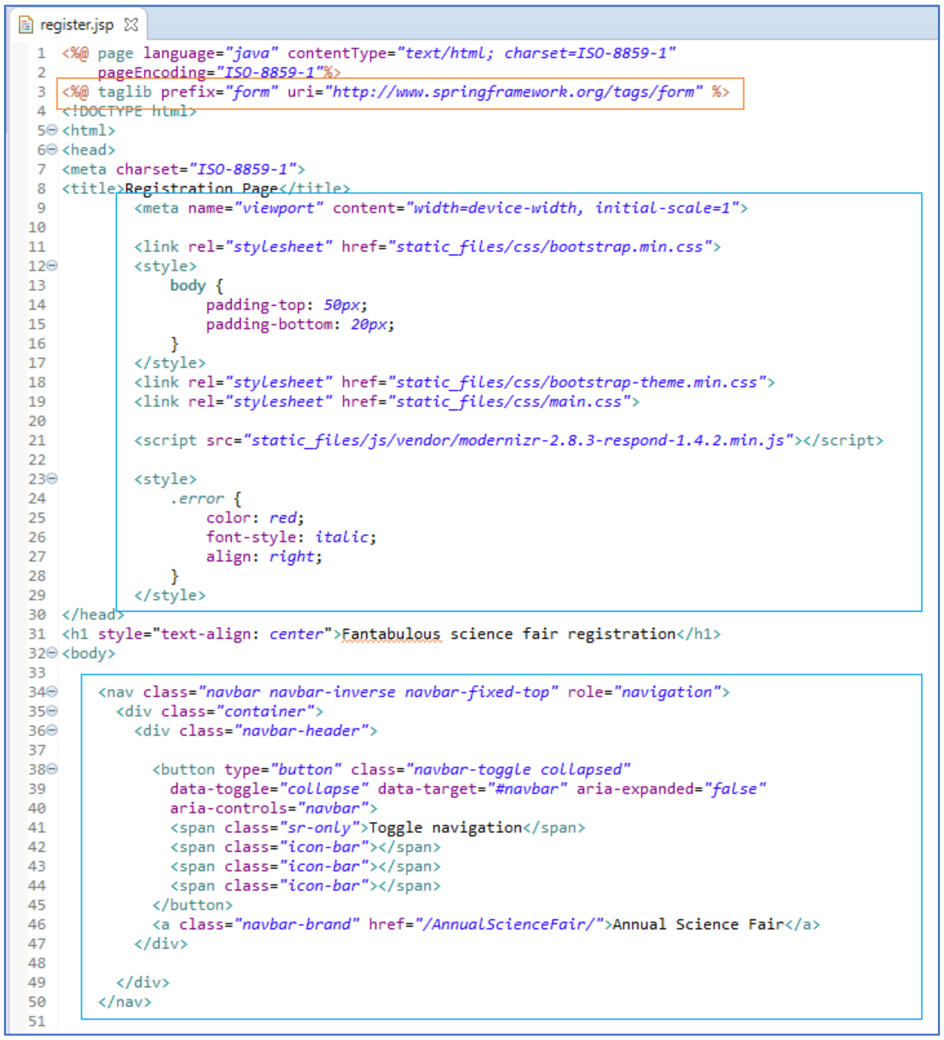
On line 53, we have a form : password element. path="password", which maps to the password member variable of the login object. This is where we'll type in a password. The password will be masked from the user.

On line 57, we have a button which allows us to submit this form. This is the login button.

On line 62, I have a link which takes us to the registration page. If you don't have an account, then go ahead and register. The href attribute for this anchor tag leads to the performRegistration path.

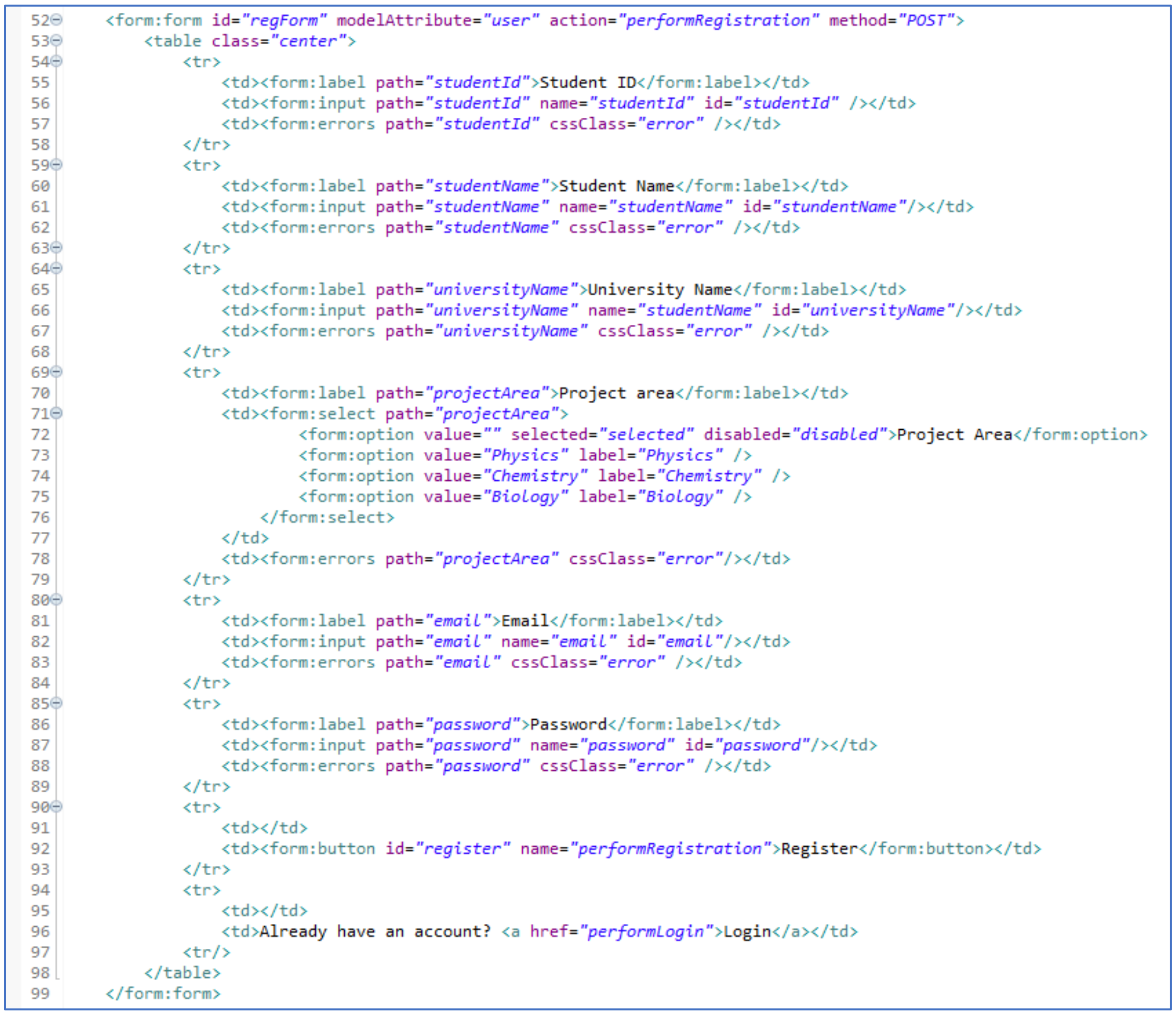
On line 67, I have a message displayed to screen. This is the message that'll be displayed in the case of a login error.

We haven't wired up our controller to handle this yet. We'll now take a look at the registration page **register.jsp** which allows users to register for the annual science fair.


register.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Registration Page</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="static_files/css/bootstrap.min.css">
        <style>
            body {
                padding-top: 50px;
                padding-bottom: 20px;
            }
        </style>
        <link rel="stylesheet" href="static_files/css/bootstrap-theme.min.css">
        <link rel="stylesheet" href="static_files/css/main.css">

        <script src="static_files/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

        <style>
            .error {
                color: red;
                font-style: italic;
                align: right;
            }
        </style>
</head>
<h1 style="text-align: center">Fantabulous science fair registration</h1>
<body>

    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">

          <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse" data-target="#navbar" aria-expanded="false"
            aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="/AnnualScienceFair/">Annual Science Fair</a>
        </div>

      </div>
    </nav>

```

At the top of the page, we have all of the theme-specific HTML and CSS which I've copied over from the **index.jsp** file. Once you scroll past all of the boilerplate stuff, you will come to the actual registration form.



```
52      <form:form id="regForm" modelAttribute="user" action="performRegistration" method="POST">
53          <table class="center">
54              <tr>
55                  <td><form:label path="studentId">Student ID</form:label></td>
56                  <td><form:input path="studentId" name="studentId" id="studentId" /></td>
57                  <td><form:errors path="studentId" cssClass="error" /></td>
58              </tr>
59              <tr>
60                  <td><form:label path="studentName">Student Name</form:label></td>
61                  <td><form:input path="studentName" name="studentName" id="stundentName"/></td>
62                  <td><form:errors path="studentName" cssClass="error" /></td>
63              </tr>
64              <tr>
65                  <td><form:label path="universityName">University Name</form:label></td>
66                  <td><form:input path="universityName" name="studentName" id="universityName"/></td>
67                  <td><form:errors path="universityName" cssClass="error" /></td>
68              </tr>
69              <tr>
70                  <td><form:label path="projectArea">Project area</form:label></td>
71                  <td><form:select path="projectArea">
72                          <form:option value="" selected="selected" disabled="disabled">Project Area</form:option>
73                          <form:option value="Physics" label="Physics" />
74                          <form:option value="Chemistry" label="Chemistry" />
75                          <form:option value="Biology" label="Biology" />
76                      </form:select>
77                  </td>
78                  <td><form:errors path="projectArea" cssClass="error"/></td>
79              </tr>
80              <tr>
81                  <td><form:label path="email">Email</form:label></td>
82                  <td><form:input path="email" name="email" id="email"/></td>
83                  <td><form:errors path="email" cssClass="error" /></td>
84              </tr>
85              <tr>
86                  <td><form:label path="password">Password</form:label></td>
87                  <td><form:input path="password" name="password" id="password"/></td>
88                  <td><form:errors path="password" cssClass="error" /></td>
89              </tr>
90              <tr>
91                  <td></td>
92                  <td><form:button id="register" name="performRegistration">Register</form:button></td>
93              </tr>
94              <tr>
95                  <td></td>
96                  <td>Already have an account? <a href="performLogin">Login</a></td>
97              <tr/>
98          </table>
99      </form:form>
```

You can see the form element on line 52, id="regForm" modelAttribute="user". This is the model object backing our registration form. Action is performRegistration. That is the handler that'll be invoked when we submit this form, method is post.

On line 56, I have a form : input with the path studentId, which maps to the studentId member variable of our user object.

On line 57, you can see a form : errors element that allows me to display errors in the studentId specification. We haven't added any validation to our user object yet.

On line 61, we have a form : input for the studentName, which corresponds to the studentName member variable in the user object. We have a form : errors element for the name as well. Again, no validation has been set up yet.

On line 65, we have a form : input for the universityName. And we have a form : errors element for the university as well.

On line 70, we have a form : select element that allows us to setup a drop-down menu for selection. This is for the projectArea. The three options in the projectArea are Physics, Chemistry, and Biology. The projectArea corresponds to the projectArea member variable in our user object. We have a form : errors element for the projectArea specification as well. We'll add validation for this member variable later on.

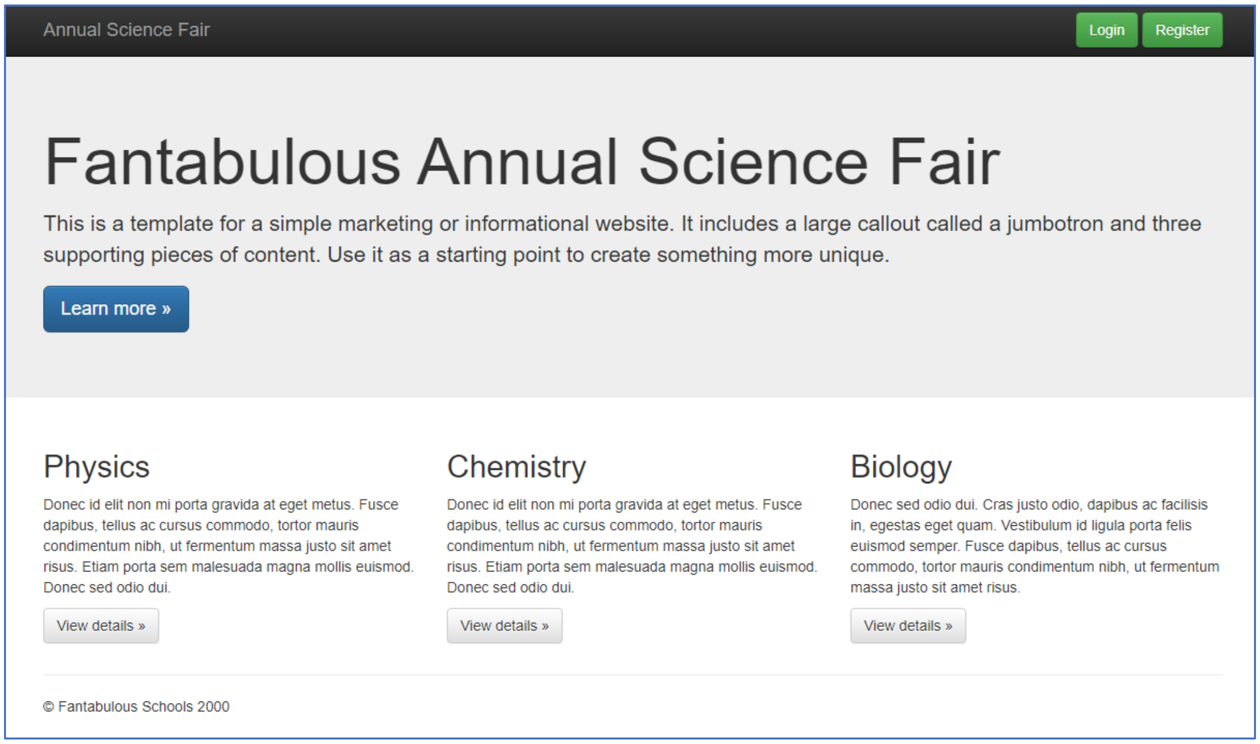
On line 82, we have a form : input for the email address of the user, path="email" corresponding to the email member variable in our user object. We have a form : errors tag as well for validation messages for the email specification.

And finally, the very last input in this registration form is on line 87. We have a form : password element with path="password", where the user specifies the password that he or she wants to use with our registration site. There is a form : errors element associated with the password as well. We have certain requirements of the passwords that users choose.

The button specification on line 92 allows us to submit this form, the register button.

We also have a link at the bottom. If you already have an account, you can choose to log in instead of re-registering. The anchor tag will direct us to the performLogin page.

Time to build this WAR file and then deploy this application to our Tomcat server. This is what the application looks like when it runs.



Fantabulous Annual Science Fair

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

Physics

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Chemistry

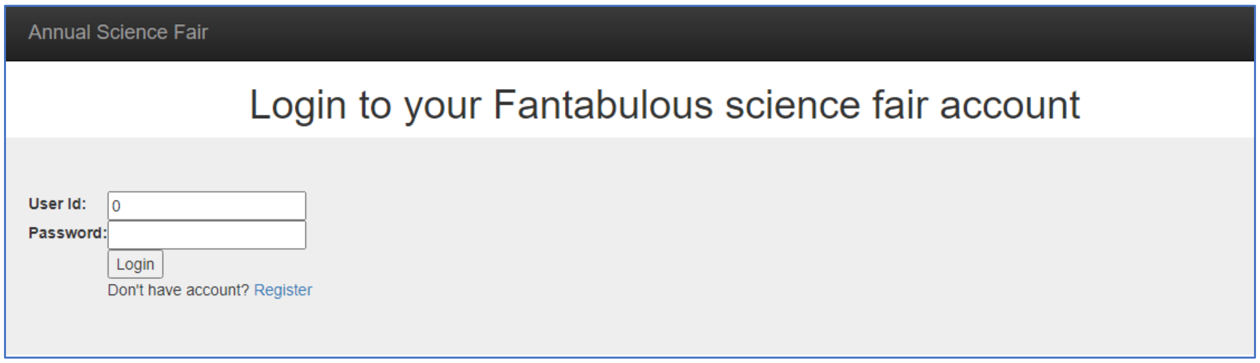
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Biology

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

© Fantabulous Schools 2000

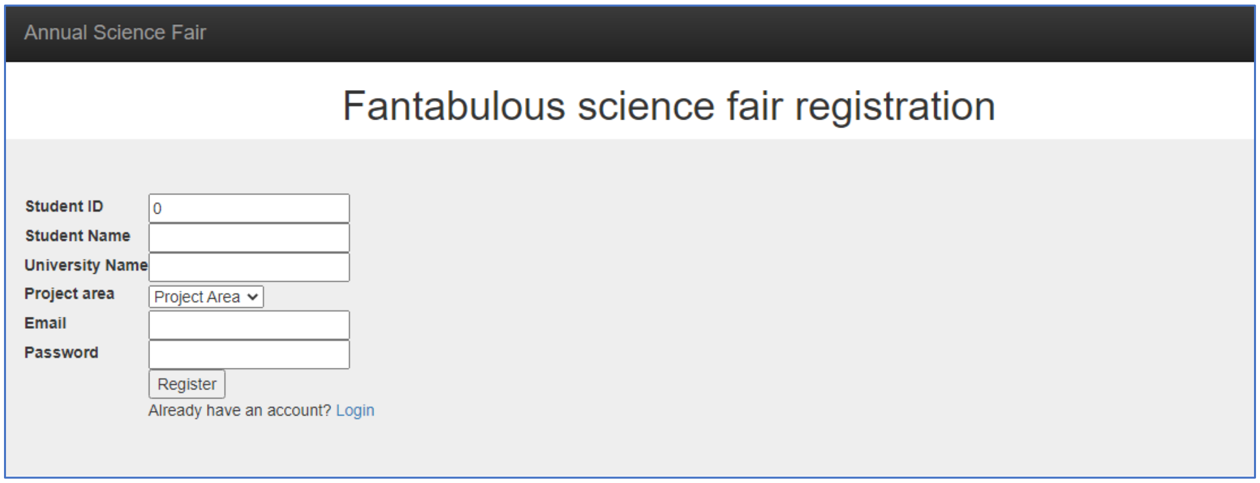
Now we have wired up our Login and Register pages. Click on the Login button and you'll be taken to the Login page, where you need to specify your User Id and Password in order to log in.


Annual Science Fair

Login to your Fantabulous science fair account

User Id:	0
Password:	

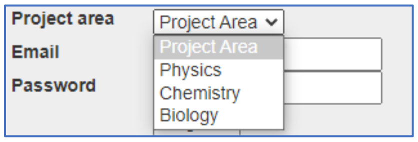
There's a Login button there. If you don't have an account, you can choose to register. If you click on Register here, you'll be directed to the registration form for the Annual Science Fair, where you have to specify a number of different details in order to register.


Annual Science Fair

Fantabulous science fair registration

Student ID	0
Student Name	
University Name	
Project area	Project Area
Email	
Password	

You can see that Project area is a drop-down. Your project can be in Physics, Chemistry, or Biology.



You can choose to register or if you click on the link on the top left, you'll go back to the home page.

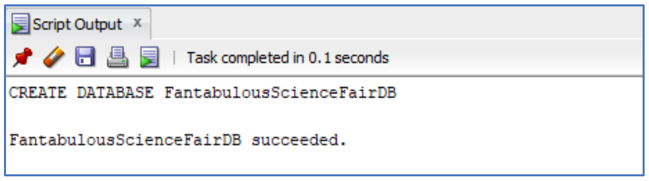
Configuring MySQL Connection

We're currently in the process of building up our Spring MVC application for the Annual Science Fair. We are set up the basic structure of the application, a login and the register forms. And we have also installed the MySQL database. We will now connect our application to the MySQL database and set up the three-tier architecture for our MVC app.

Here we are on the MySQL Workbench. I'm going to create a new database here on my MySQL server called FantabulousScienceFairDB.

```
CREATE DATABASE FantabulousScienceFairDB;

USE FantabulousScienceFairDB;
```

Click on the lightning icon in order to run this command, the database has been successfully created. The next command that I'm going to run is USE FantabulousScienceFairDB. I want to use this database in order to create a table within this.



```
CREATE DATABASE FantabulousScienceFairDB

FantabulousScienceFairDB succeeded.
```

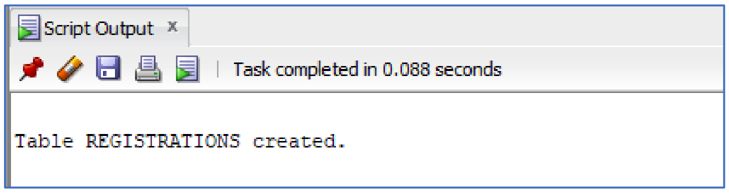
Run this command and this will run through successfully as well.

Now let's go ahead and create a table hold the Registrations for the science fair.

```
CREATE TABLE registrations(
    student_id        INT NOT NULL,
    student_name      VARCHAR(255) NOT NULL,
    university_name   VARCHAR(255) NOT NULL,
    project_area      VARCHAR(25) NOT NULL,
    email             VARCHAR(255) NOT NULL,
    password VARCHAR(25) NOT NULL,
    PRIMARY KEY ( student_id )
);
```

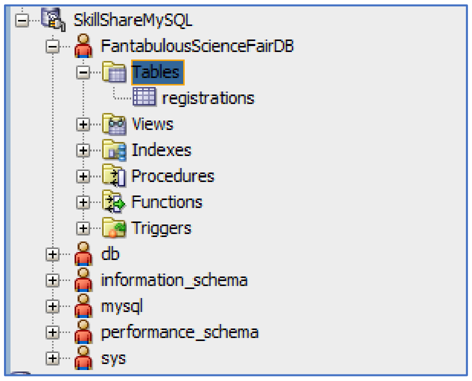
You can see that the columns of this Registrations table map exactly to the member variables of the user object. We have a column for the studentId, studentName, universityName, projectArea, email, and password. All of these values are required and cannot be null. The studentId is an integer and also is the primary key of this Registrations table. Go ahead and execute this CREATE TABLE command so that the

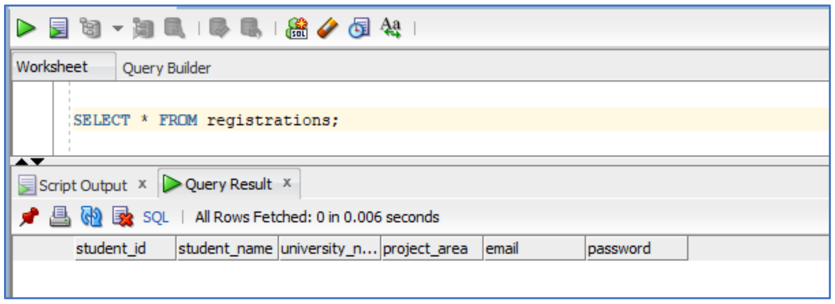
Registrations table is created in our FantabulousScienceFairDB.



```
Table REGISTRATIONS created.
```

On the left navigation pane, you can expand the Tables icon. Go to the Registrations table, and you can see the table has been set up successfully. Let's run a SELECT * operation from the Registrations table, and you'll find that it's currently empty.





```
SELECT * FROM registrations;
```

student_id	student_name	university_n...	project_area	email	password

There are no records. With our database set up successfully, let's head over to Eclipse and take a look at **pom.xml**.

Now, in order to be able to have our Spring MVC application connect to a database, we need to add in a few additional dependencies. Scroll down to the bottom, and notice that I have a dependency on the mysql-connector-java object.

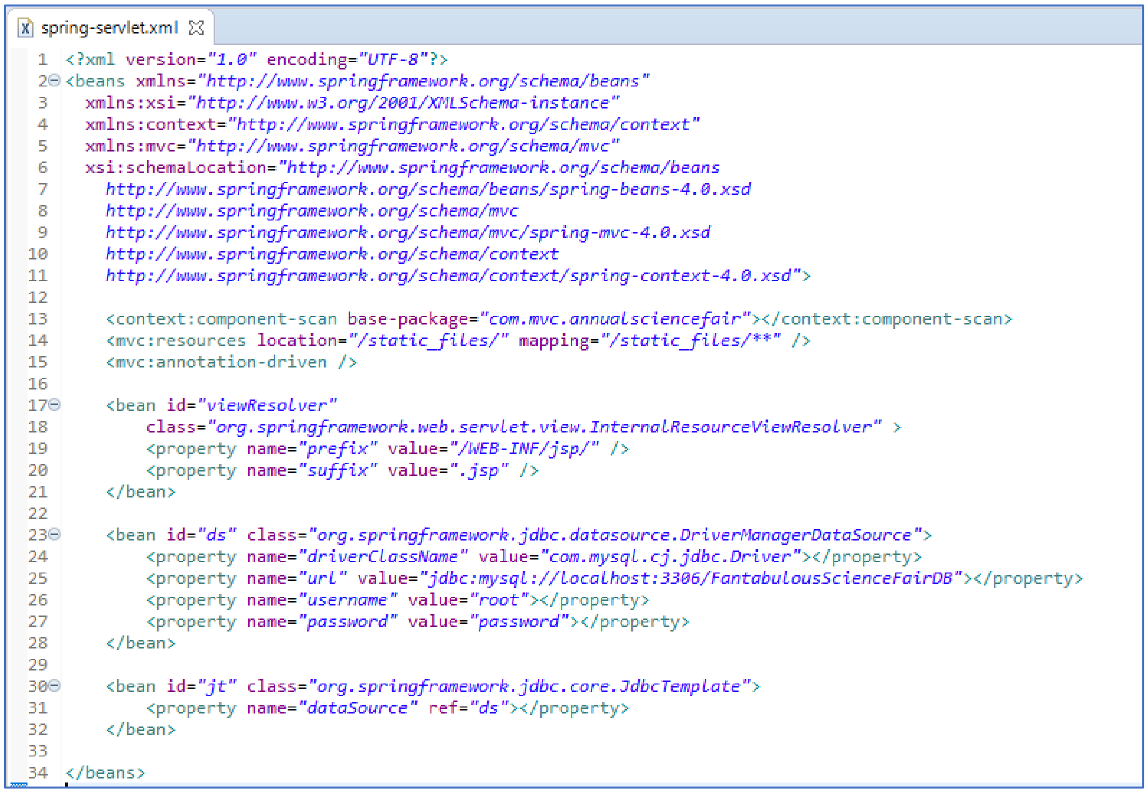
pom.xml

```xml
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.1.2.RELEASE</version>
    </dependency>
```

This is the dependency specified on line 59. I also have a dependency on the spring-jdbc artifact on line 68.

We'll be using the MySQL connector along with JDBC template in order to connect to our underlying MySQL database and run queries, perform insertions, and so on.

We have the connection specifications for the MySQL database in our spring-servlet.xml file. Scroll down to the bottom, and you'll see that we have a bunch of new specifications here in this file.



```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <beans xmlns="http://www.springframework.org/schema/beans"
 3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4   xmlns:context="http://www.springframework.org/schema/context"
 5   xmlns:mvc="http://www.springframework.org/schema/mvc"
 6   xsi:schemaLocation="http://www.springframework.org/schema/beans
 7     http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
 8     http://www.springframework.org/schema/mvc
 9     http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
10     http://www.springframework.org/schema/context
11     http://www.springframework.org/schema/context/spring-context-4.0.xsd">
12
13     <context:component-scan base-package="com.mvc.annualsciencefair"></context:component-scan>
14     <mvc:resources location="/static_files/" mapping="/static_files/**" />
15     <mvc:annotation-driven />
16
17     <bean id="viewResolver"
18         class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
19         <property name="prefix" value="/WEB-INF/jsp/" />
20         <property name="suffix" value=".jsp" />
21     </bean>
22
23     <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
24         <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"></property>
25         <property name="url" value="jdbc:mysql://localhost:3306/FantabulousScienceFairDB"></property>
26         <property name="username" value="root"></property>
27         <property name="password" value="password"></property>
28     </bean>
29
30     <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
31         <property name="dataSource" ref="ds"></property>
32     </bean>
33
34 </beans>
```

The class that this bean maps to is the DriverManagerDataSource, which is a part of the JDBC framework. In order for Spring to be able to connect to this database, you need to specify a URL for the connection, which we do on line 25.

```
<property name="url"
value="jdbc:mysql://localhost:3306/FantabulousScienceFairDB"></property>
```

You can see the value of the url is jdbc:mysql, our SQL database is running on localhost port 3306. If your SQL database is running on a different port, make sure you specify that here. And then we specify the name of the database that we want to connect to in our database server, the FantabulousSciencefairDB.

In order to make the connection, we need to specify a username and password. The username is root and our password happens to be password.

The last bean that we have specified here refers to the JdbcTemplate.

The JdbcTemplate refers to the dataSource bean, as you can see on line 43.

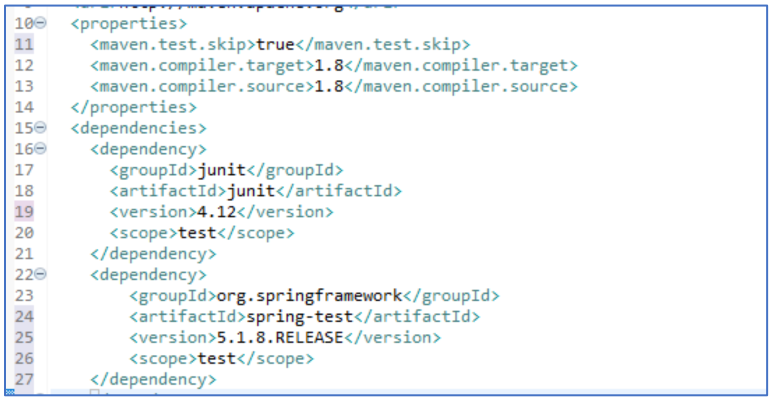
```
<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
      <property name="datasource" ref="ds"></property>
</bean>
```

The JdbcTemplates that we'll use in our application will use the connection specification in our dataSource beans in order to connect to our MySQL database.

Create Unit Test

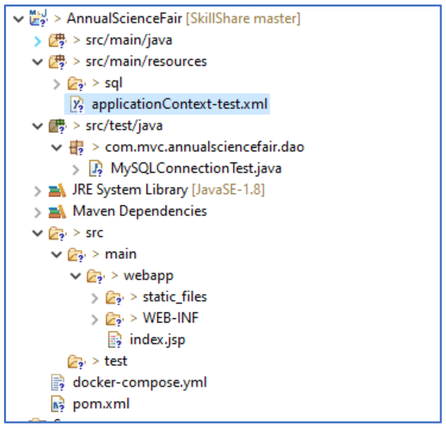
Update pom.xml with **skip test** when doing build and **additional dependency**

1) junit:4.12
2) org.springframework.spring-test:5.1.8-RELEASE

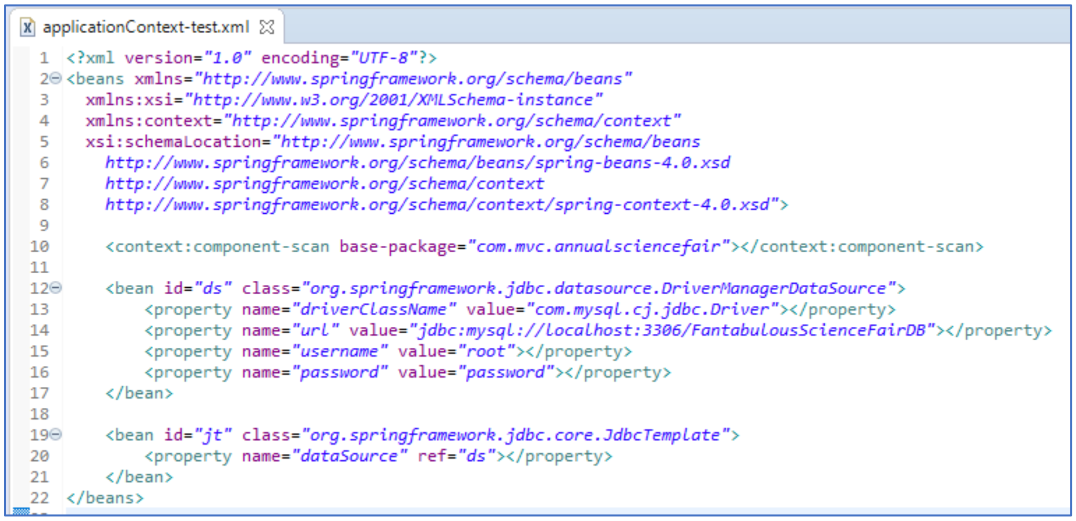


```
<properties>
  <maven.test.skip>true</maven.test.skip>
  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>5.1.8.RELEASE</version>
      <scope>test</scope>
  </dependency>
```

Create new **Application Context for Test** class as that similar with spring-servlet.xml files under **src/main/resource** folder as follows



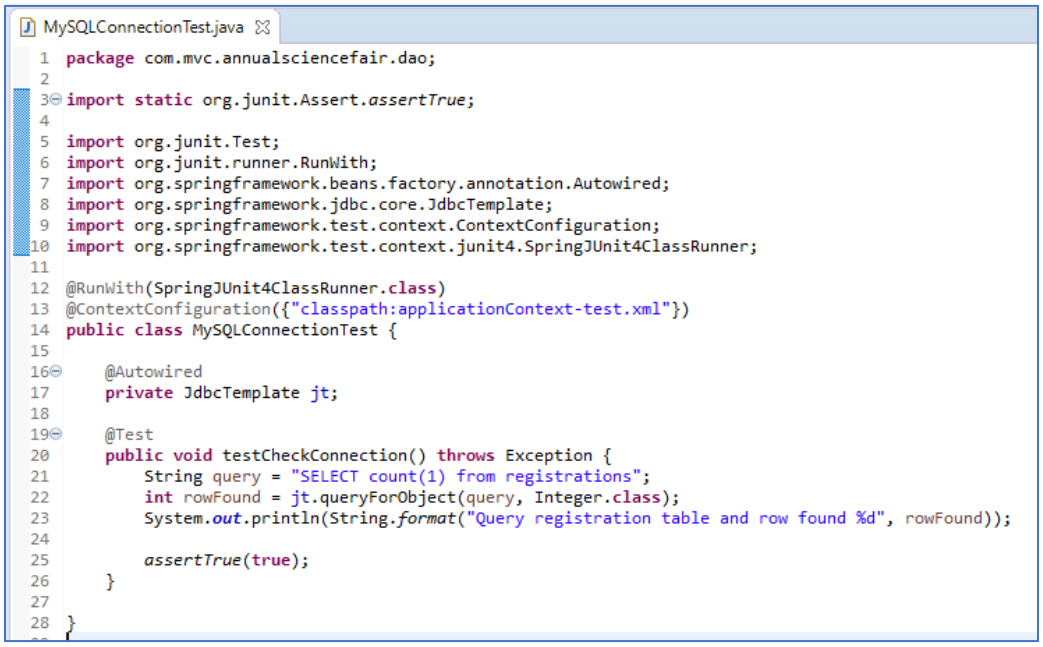
```
> AnnualScienceFair [SkillShare master]
  > src/main/java
  > src/main/resources
    > sql
    applicationContext-test.xml
  > src/test/java
    > com.mvc.annualsciencefair.dao
      MySQLConnectionTest.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  > src
    > main
      > webapp
        > static_files
        > WEB-INF
        index.jsp
    > test
  docker-compose.yml
  pom.xml
```



applicationContext-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="com.mvc.annualsciencefair"></context:component-scan>

    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql://localhost:3306/FantabulousScienceFairDB"></property>
        <property name="username" value="root"></property>
        <property name="password" value="password"></property>
    </bean>

    <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>
</beans>
```

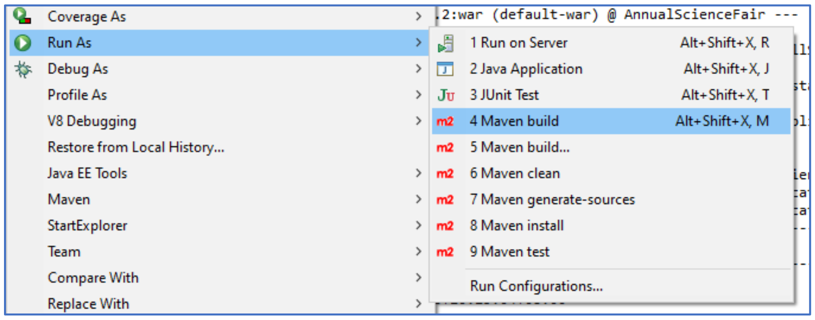
Create Unit Test Class



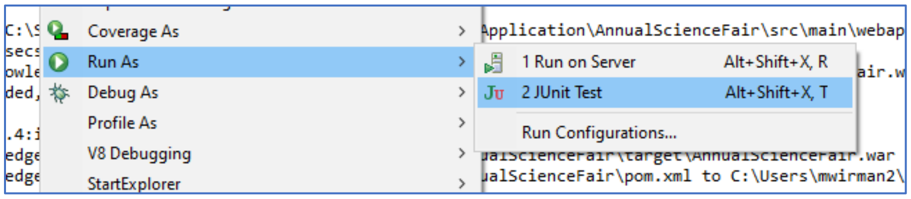
MySQLConnectionTest.java

```java
package com.mvc.annualsciencefair.dao;

import static org.junit.Assert.assertTrue;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"classpath:applicationContext-test.xml"})
public class MySQLConnectionTest {

    @Autowired
    private JdbcTemplate jt;

    @Test
    public void testCheckConnection() throws Exception {
        String query = "SELECT count(1) from registrations";
        int rowFound = jt.queryForObject(query, Integer.class);
        System.out.println(String.format("Query registration table and row found %d", rowFound));

        assertTrue(true);
    }

}
```

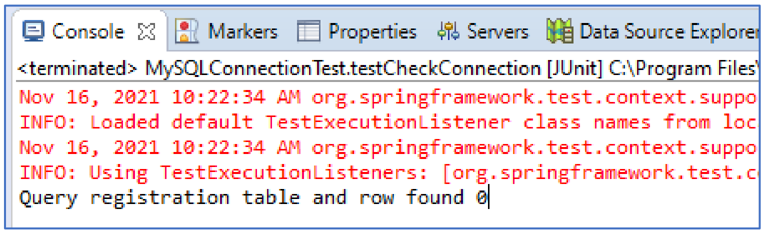
Build the project, by right click on project and select **Run As** > **Maven Build**



Run Unit Test Class by right click on **testCheckConnection** method and select **Run As** > **Junit Test**



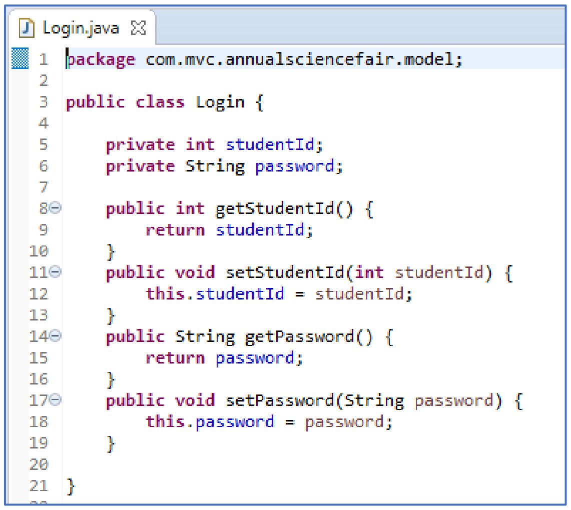
follows is the output



```
<terminated> MySQLConnectionTest.testCheckConnection [JUnit] C:\Program Files
Nov 16, 2021 10:22:34 AM org.springframework.test.context.suppo
INFO: Loaded default TestExecutionListener class names from loc
Nov 16, 2021 10:22:34 AM org.springframework.test.context.suppo
INFO: Using TestExecutionListeners: [org.springframework.test.c
Query registration table and row found 0
```

Implementing 3 Tiers Application

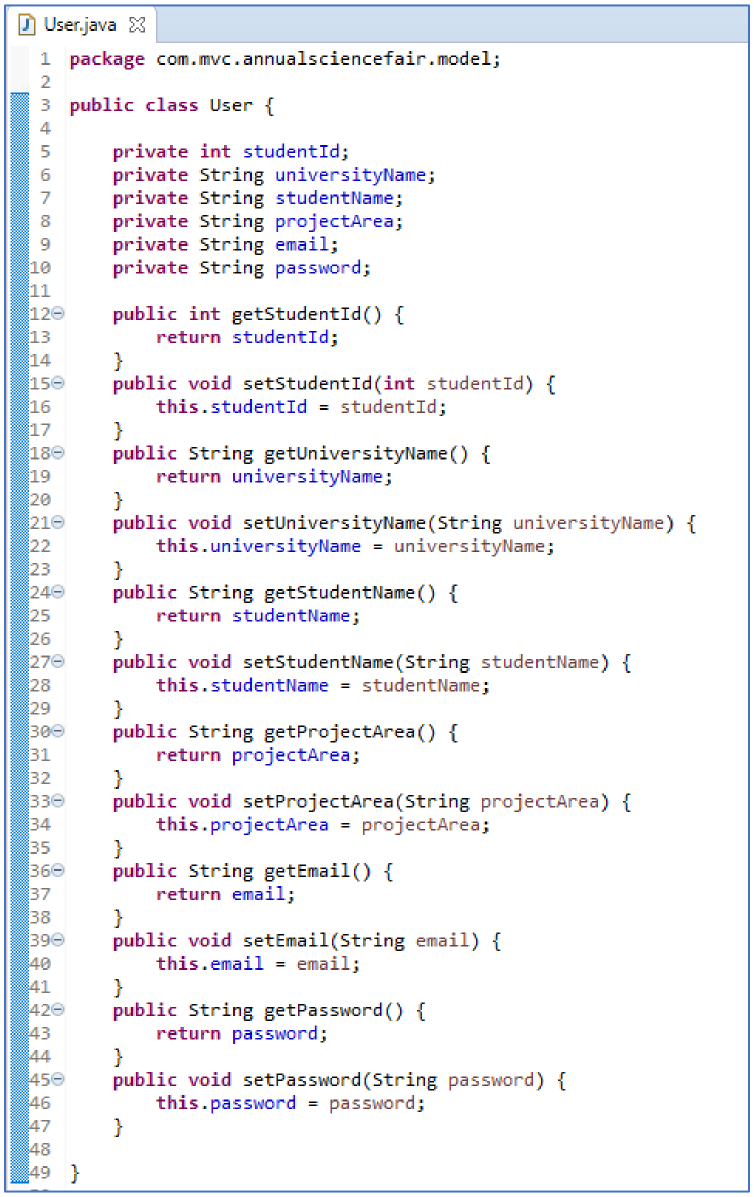
1. Model Layer
   Login.java



```java
package com.mvc.annualsciencefair.model;

public class Login {

    private int studentId;
    private String password;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

}
```

   There is no change to the Login object. This holds the student ID and the password and it has getters and setters.

User.java

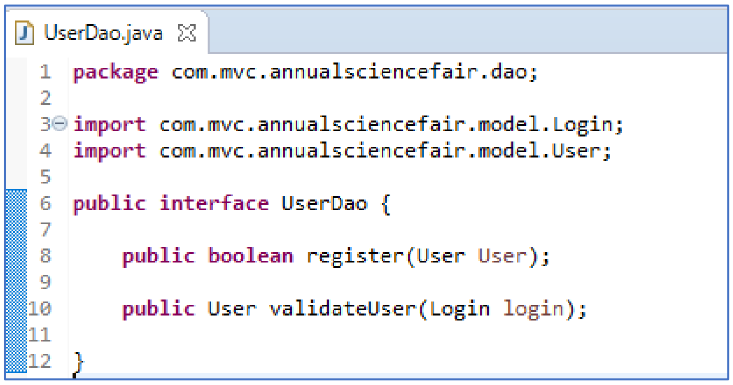

User.java

```java
package com.mvc.annualsciencefair.model;

public class User {

    private int studentId;
    private String universityName;
    private String studentName;
    private String projectArea;
    private String email;
    private String password;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getUniversityName() {
        return universityName;
    }
    public void setUniversityName(String universityName) {
        this.universityName = universityName;
    }
    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    public String getProjectArea() {
        return projectArea;
    }
    public void setProjectArea(String projectArea) {
        this.projectArea = projectArea;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

}
```

There is no change to the user object as well. It contains the same fields as it did in the previous demo, and it contains the getters and setters for all of these fields.

Both of our model objects remain unchanged in this demo.

2. Data Access Layer (The Third Tier of Spring MVC architecture)
   Interface and Implementation Class of Data Access Layer Object:

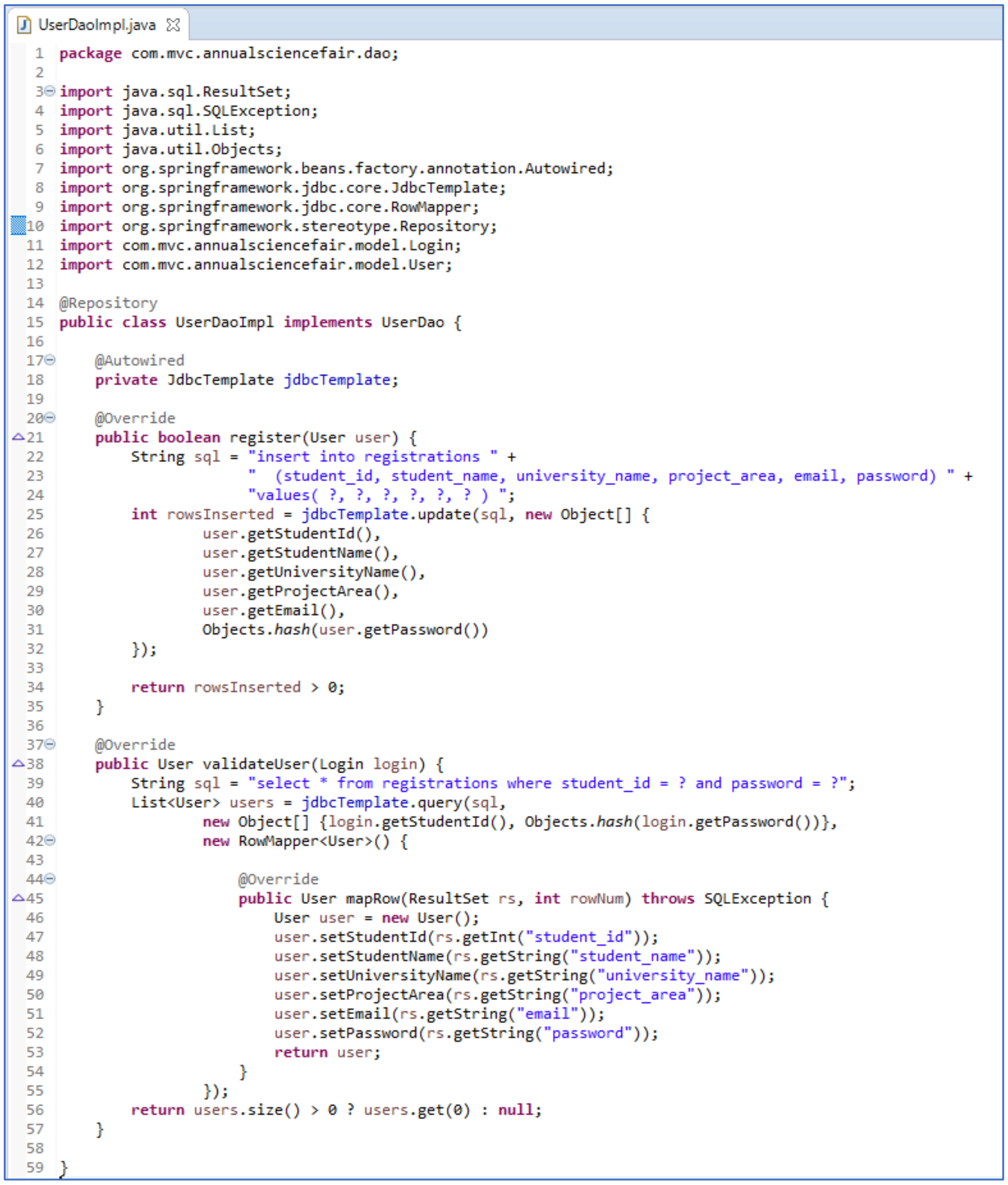


```java
package com.mvc.annualsciencefair.dao;

import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;

public interface UserDao {

    public boolean register(User User);

    public User validateUser(Login login);

}
```

UserDao specifies the interface for methods implemented in this data access layer. The first method here is a register method, which takes in a user model object and registers that user for the science fair, and it returns true if the registration was successful.

The next method is the validateUser method. This takes in a login object which contains the student ID and password. And if it's a match with an already registered user we return the valid user. Otherwise we return null.

The implementation of this UserDao interface is in the **UserDaoImpl** class.



```
 1 package com.mvc.annualsciencefair.dao;
 2 
 3 import java.sql.ResultSet;
 4 import java.sql.SQLException;
 5 import java.util.List;
 6 import java.util.Objects;
 7 import org.springframework.beans.factory.annotation.Autowired;
 8 import org.springframework.jdbc.core.JdbcTemplate;
 9 import org.springframework.jdbc.core.RowMapper;
10 import org.springframework.stereotype.Repository;
11 import com.mvc.annualsciencefair.model.Login;
12 import com.mvc.annualsciencefair.model.User;
13 
14 @Repository
15 public class UserDaoImpl implements UserDao {
16 
17     @Autowired
18     private JdbcTemplate jdbcTemplate;
19 
20     @Override
21     public boolean register(User user) {
22         String sql = "insert into registrations " +
23                      "  (student_id, student_name, university_name, project_area, email, password) " +
24                      "values( ?, ?, ?, ?, ?, ? ) ";
25         int rowsInserted = jdbcTemplate.update(sql, new Object[] {
26                 user.getStudentId(),
27                 user.getStudentName(),
28                 user.getUniversityName(),
29                 user.getProjectArea(),
30                 user.getEmail(),
31                 Objects.hash(user.getPassword())
32         });
33 
34         return rowsInserted > 0;
35     }
36 
37     @Override
38     public User validateUser(Login login) {
39         String sql = "select * from registrations where student_id = ? and password = ?";
40         List<User> users = jdbcTemplate.query(sql,
41                 new Object[] {login.getStudentId(), Objects.hash(login.getPassword())},
42                 new RowMapper<User>() {
43 
44                     @Override
45                     public User mapRow(ResultSet rs, int rowNum) throws SQLException {
46                         User user = new User();
47                         user.setStudentId(rs.getInt("student_id"));
48                         user.setStudentName(rs.getString("student_name"));
49                         user.setUniversityName(rs.getString("university_name"));
50                         user.setProjectArea(rs.getString("project_area"));
51                         user.setEmail(rs.getString("email"));
52                         user.setPassword(rs.getString("password"));
53                         return user;
54                     }
55                 });
56         return users.size() > 0 ? users.get(0) : null;
57     }
58 
59 }
```

On line 15, you can see UserDaoImpl implements UserDao. Now this implementation accesses the underlying MySQL database using JdbcTemplates.

```
@Autowired
private JdbcTemplate jdbcTemplate;
```

Notice that I have Autowired a JdbcTemplate to be injected into this implementation class. If you scroll down below, you'll see the implementation of the register method to register a new user for the science fair.

```
public boolean register(User user) {
```

Takes as an input argument the User model object. We then set up the SQL query to perform an insert into the registrations table.

```
String sql = "insert into registrations
(student_id, student_name, university_name, project_area, email, password)
values( ?, ?, ?, ?, ?, ? ) ";
```

The query on line 22 contains a number of placeholder values as specified using the question marks. We invoke the update method on the JdbcTemplate. Specify the SQL query for the insertion, and specify the values that need to be filled-in, in our parameterized query. A parameterized query is any query with the question marks.

```
int rowsInserted = jdbcTemplate.update(sql, new Object[] {
```

So we specify the student ID, student name, the university name, project area, email, and password. These are all of the values needed to insert a single record in the registrations table which corresponds to registering a user for the annual science fair.

```
        user.getStudentId(),
        user.getStudentName(),
        user.getUniversityName(),
        user.getProjectArea(),
        user.getEmail(),
        Objects.hash(user.getPassword())
```

Notice how I create a hash of the password before I insert the password. You should never store passwords directly in a database because anyone querying the database will be able to see the password. Always use a one way hash. This one way hash ensures that we won't be able to recreate the original password by looking at the hash, but we will be able to compare two passwords by hashing both passwords to see whether they hash to the same value. One thing to note here that it's not really recommended that you use Objects.hash in a real world production grade application. You should use a much stronger hashing algorithm, one that has been approved by security experts. But for the sake of our demo, I'm using Objects.hash.

If a new record was successfully inserted, that is a user was registered, rowsUpdated will be greater than 0, and we return true then.

```
        return rowsInserted > 0;
```

Let's scroll further down, and let's take a look at the method to validate a user that does try to log in.

```
public User validateUser(Login login) {
```

The login object contains the student ID and password for a previously registered user. We do a **select *** operation from the registrations table and see if there is a registration record with this student ID and the same password hash.

```
String sql = "select * from registrations where student_id = ? and password =
?";
```

Once again, we compare the passwords using the hash of the password because passwords haven't been stored directly in our underlying table. We invoke the query method on the JdbcTemplate and map the retrieved result using a UserMapper.

```
List<User> users = jdbcTemplate.query(sql,
                new Object[] {
                        login.getStudentId(),
                        Objects.hash(login.getPassword())
                },
                new RowMapper<User>() {
```

The UserMapper class, as we shall see in just a bit, maps every row retrieved from the underlying table to a user object. If the list of users returned has at least one user, we return the user at index 0, otherwise we return null.

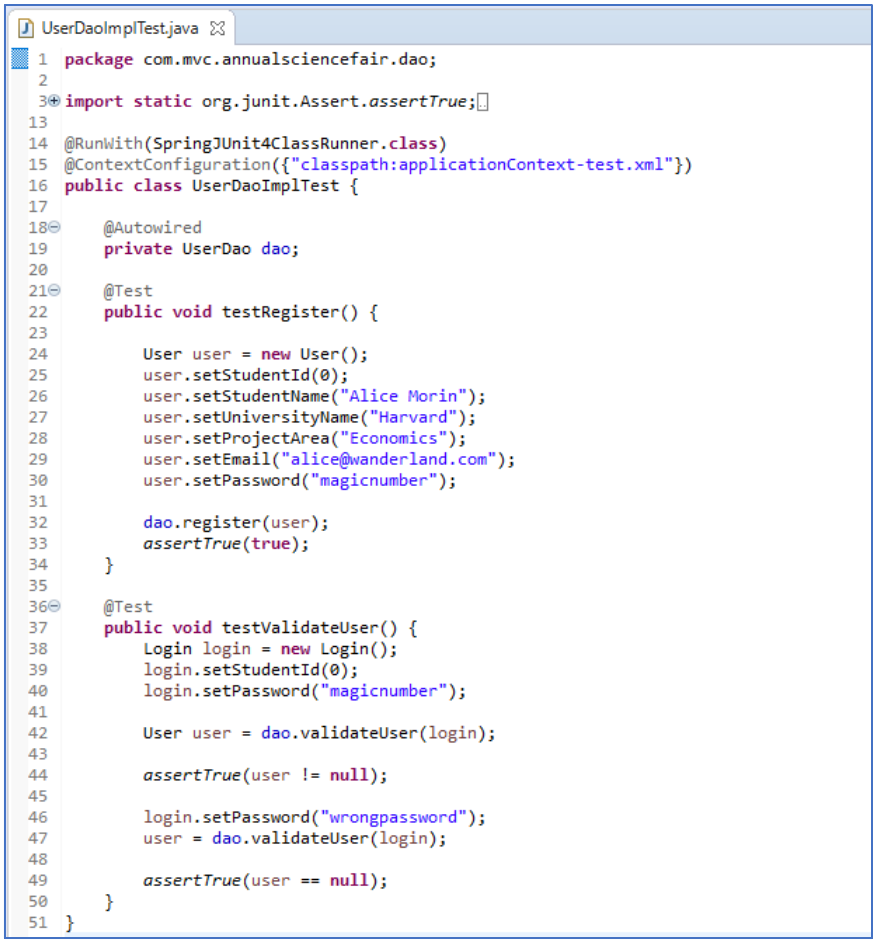
```
                return users.size() > 0 ? users.get(0) : null;
```

Let's quickly take a look at the UserMapper class that implements the RowMapper interface in Jdbc. This overrides just the mapRow method. It takes as an input argument a ResultSet, which contains a single row retrieved from the underlying database table. We instantiate a new user object, and we populate this user object using the values in the individual fields of this result set.

```
@Override
public User mapRow(ResultSet rs, int rowNum) throws SQLException {
                User user = new User();
                user.setStudentId(rs.getInt("student_id"));
                user.setStudentName(rs.getString("student_name"));
                user.setUniversityName(rs.getString("university_name"));
                user.setProjectArea(rs.getString("project_area"));
                user.setEmail(rs.getString("email"));
                user.setPassword(rs.getString("password"));
                return user;
}
```

We use rs.getInt to get the student ID, and we use rs.getString for all of the other column values. Make sure that the names of the columns that you've specified here, while retrieving from the result set, match the column names in your underlying database table. The return value from mapRow is a fully populated user object.

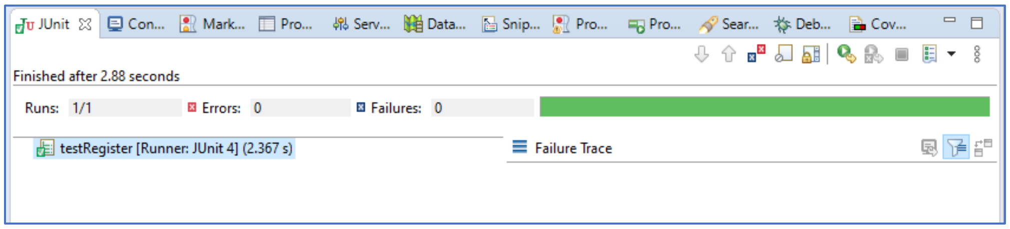
Create Unit Test for UserDaoImpl Class



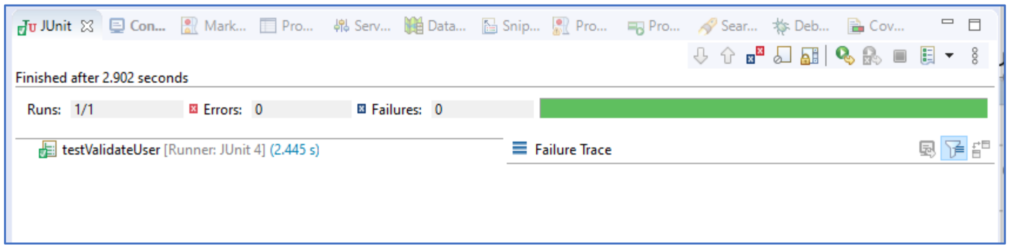
UserDaoImplTest.java

```java
package com.mvc.annualsciencefair.dao;

import static org.junit.Assert.assertTrue;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"classpath:applicationContext-test.xml"})
public class UserDaoImplTest {

    @Autowired
    private UserDao dao;

    @Test
    public void testRegister() {

        User user = new User();
        user.setStudentId(0);
        user.setStudentName("Alice Morin");
        user.setUniversityName("Harvard");
        user.setProjectArea("Economics");
        user.setEmail("alice@wanderland.com");
        user.setPassword("magicnumber");

        dao.register(user);
        assertTrue(true);
    }

    @Test
    public void testValidateUser() {
        Login login = new Login();
        login.setStudentId(0);
        login.setPassword("magicnumber");

        User user = dao.validateUser(login);

        assertTrue(user != null);

        login.setPassword("wrongpassword");
        user = dao.validateUser(login);

        assertTrue(user == null);
    }
}
```

Run unit test Register by right click on test method and see the result

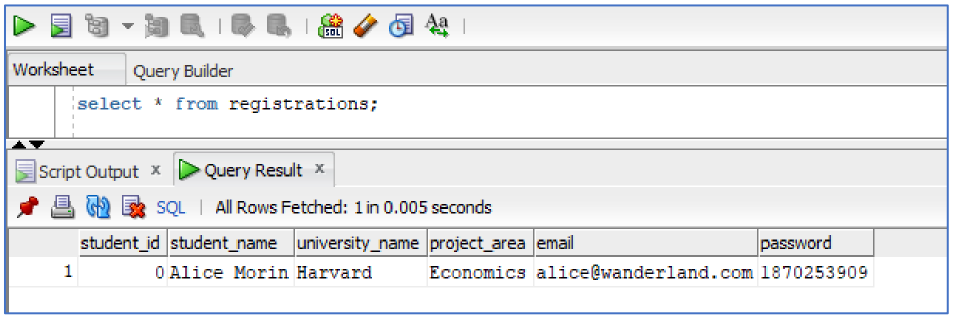


```
Finished after 2.88 seconds
Runs: 1/1    Errors: 0    Failures: 0
testRegister [Runner: JUnit 4] (2.367 s)
```



```
Finished after 2.902 seconds
Runs: 1/1    Errors: 0    Failures: 0
testValidateUser [Runner: JUnit 4] (2.445 s)
```

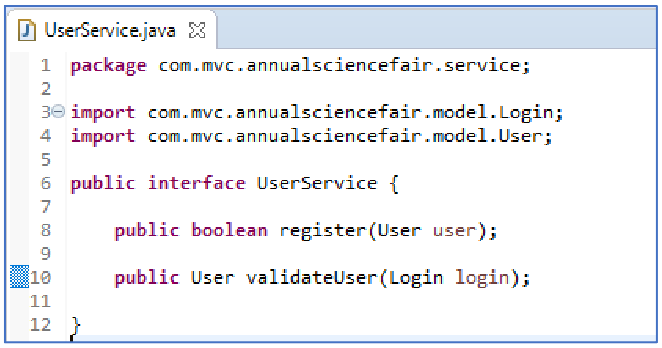
check result in mysql



```
select * from registrations;
```

	student_id	student_name	university_name	project_area	email	password
1	0	Alice Morin	Harvard	Economics	alice@wanderland.com	1870253909

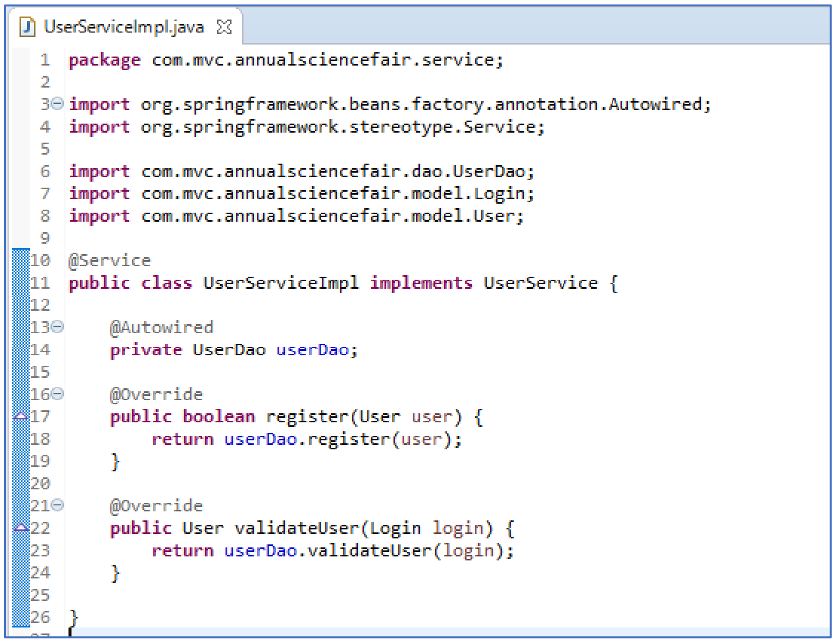
3. Service Layer (The Middle Tier of springMVC architecture)



```java
package com.mvc.annualsciencefair.service;

import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;

public interface UserService {

    public boolean register(User user);

    public User validateUser(Login login);

}
```

We have the interface here, **UserService**, which has two methods, register and validateUser. This is a mirror image of the UserDao interface.

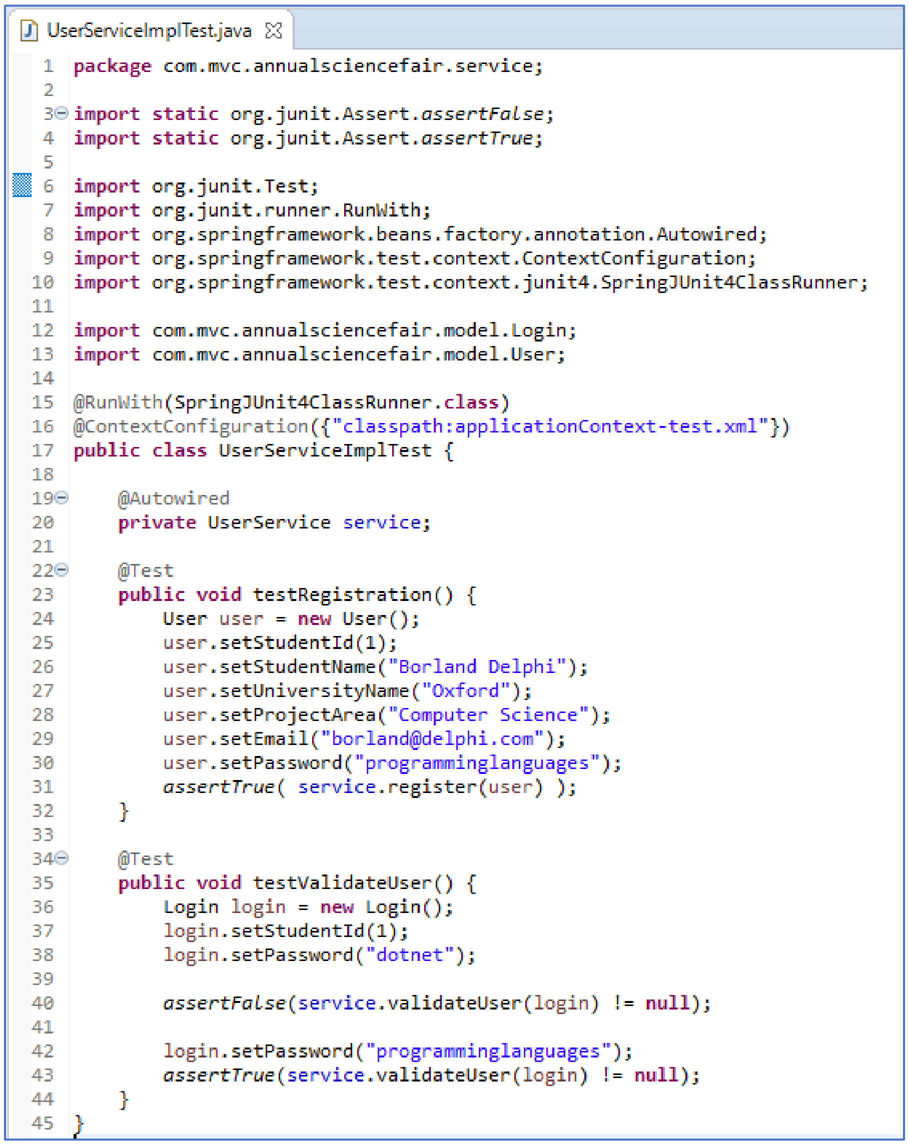
Here is **UserServiceImpl**. Notice that it simply delegates all actions to the UserDao implementation.



```java
package com.mvc.annualsciencefair.service;

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

import com.mvc.annualsciencefair.dao.UserDao;
import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    @Override
    public boolean register(User user) {
        return userDao.register(user);
    }

    @Override
    public User validateUser(Login login) {
        return userDao.validateUser(login);
    }

}
```

You can include additional business logic here if you want to, but we've kept things simple.You can see that we inject the UserDao into this implementation using **@Autowired** and then call userDao.register and userDao.validateUser.

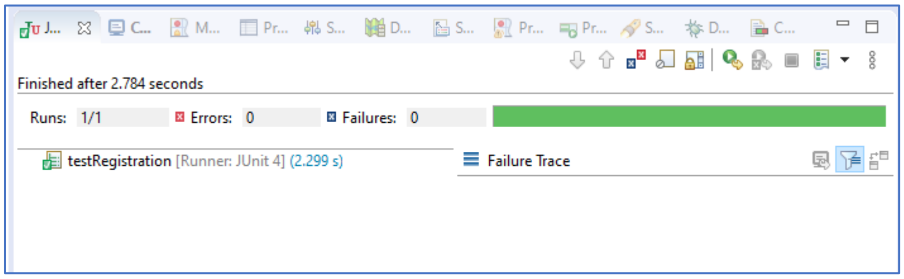
Test Class



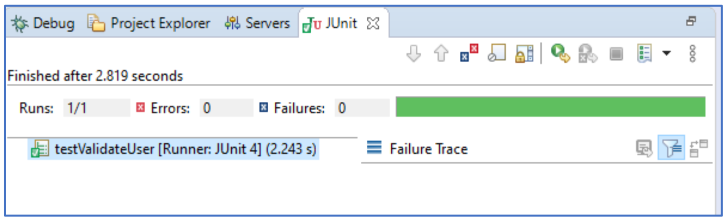
UserServiceImplTest.java

```java
package com.mvc.annualsciencefair.service;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"classpath:applicationContext-test.xml"})
public class UserServiceImplTest {

    @Autowired
    private UserService service;

    @Test
    public void testRegistration() {
        User user = new User();
        user.setStudentId(1);
        user.setStudentName("Borland Delphi");
        user.setUniversityName("Oxford");
        user.setProjectArea("Computer Science");
        user.setEmail("borland@delphi.com");
        user.setPassword("programminglanguages");
        assertTrue( service.register(user) );
    }

    @Test
    public void testValidateUser() {
        Login login = new Login();
        login.setStudentId(1);
        login.setPassword("dotnet");

        assertFalse(service.validateUser(login) != null);

        login.setPassword("programminglanguages");
        assertTrue(service.validateUser(login) != null);
    }
}
```

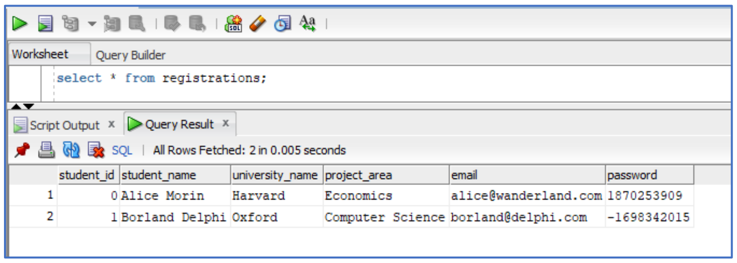
Test Result



```
Finished after 2.784 seconds

Runs: 1/1    Errors: 0    Failures: 0

testRegistration [Runner: JUnit 4] (2.299 s)
```



```
Finished after 2.819 seconds
Runs: 1/1    Errors: 0    Failures: 0
testValidateUser [Runner: JUnit 4] (2.243 s)
```



```
select * from registrations;
```

All Rows Fetched: 2 in 0.005 seconds

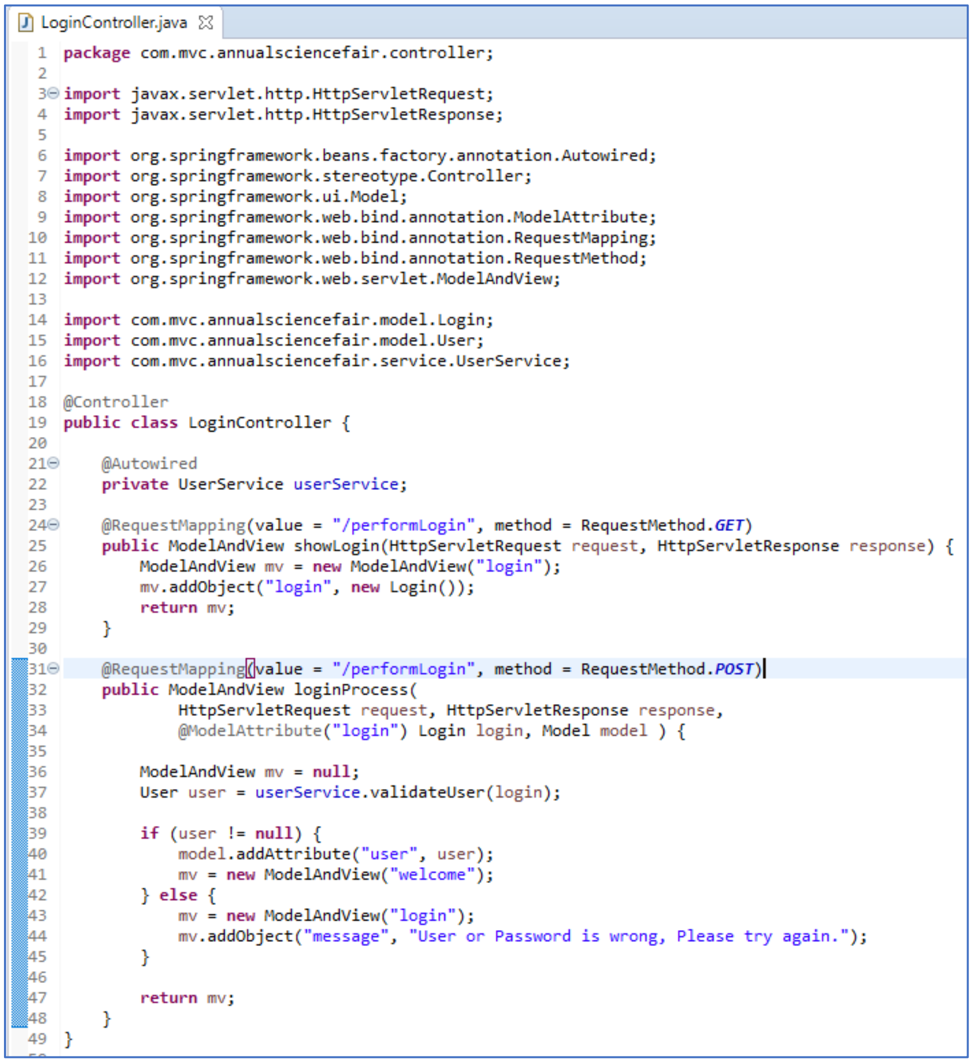
	student_id	student_name	university_name	project_area	email	password
1	0	Alice Morin	Harvard	Economics	alice@wanderland.com	1870253909
2	1	Borland Delphi	Oxford	Computer Science	borland@delphi.com	-1698342015

4. Presentation Layer, Controller

Finally let's take a look at the presentation layer in our springMVC architecture, our controller.

- **LoginController.java**

Here is the login controller, which we've expanded a little bit.


```java
package com.mvc.annualsciencefair.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;
import com.mvc.annualsciencefair.service.UserService;

@Controller
public class LoginController {

    @Autowired
    private UserService userService;

    @RequestMapping(value = "/performLogin", method = RequestMethod.GET)
    public ModelAndView showLogin(HttpServletRequest request, HttpServletResponse response) {
        ModelAndView mv = new ModelAndView("login");
        mv.addObject("login", new Login());
        return mv;
    }

    @RequestMapping(value = "/performLogin", method = RequestMethod.POST)
    public ModelAndView loginProcess(
            HttpServletRequest request, HttpServletResponse response,
            @ModelAttribute("login") Login login, Model model ) {

        ModelAndView mv = null;
        User user = userService.validateUser(login);

        if (user != null) {
            model.addAttribute("user", user);
            mv = new ModelAndView("welcome");
        } else {
            mv = new ModelAndView("login");
            mv.addObject("message", "User or Password is wrong, Please try again.");
        }

        return mv;
    }
}
```

We've autowired and injected the UserService into this login controller.

```java
@Autowired
private UserService userService;
```

The first method that responds to http GET request and is mapped to /performLogin. We are familiar this is what renders the login form.

The second method performs the actual login. The mapping is once again /performLogin, but this method responds to http POST requests.

Notice that we inject into it the login object using **@ModelAttribute** login. Then use the userService injected into this controller to check whether the logged in user is a valid user. This is on line 36, u**serService.validateUser**.

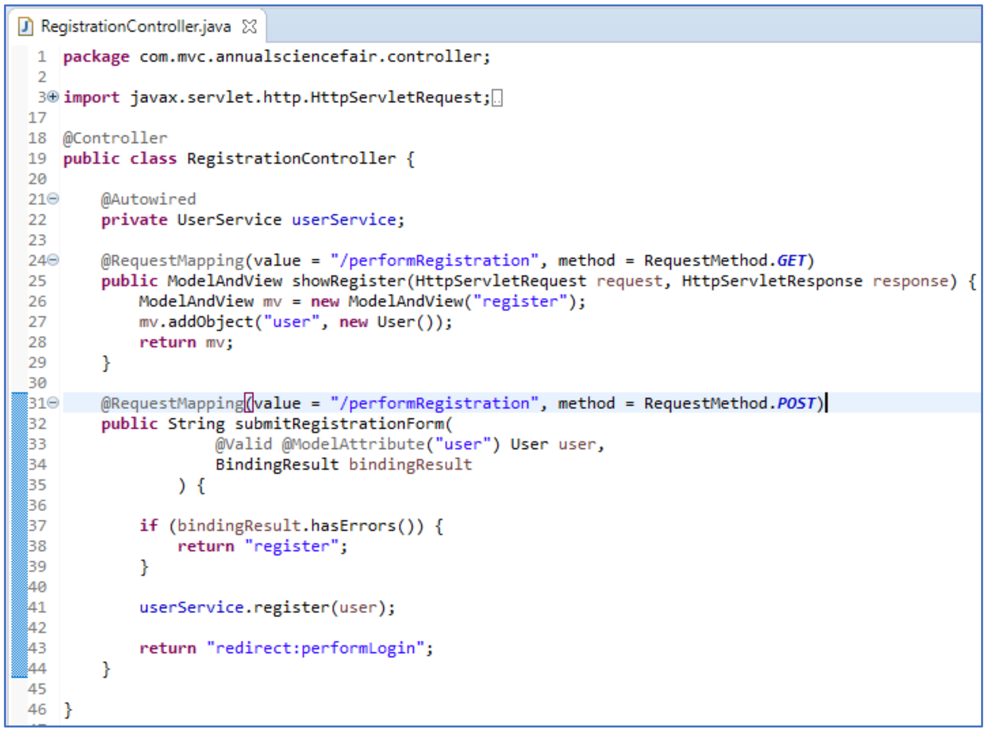
If the object that is returned is not null, that is it is a valid user, then we add this user as an attribute to the model, and then we render the welcome page for a previously registered user.

If the user was not validated, there was a problem either with the username or password.

In this situation, the else block we send the user back to the login page, along with a message *which says username or password is wrong, Please try again.*

- **RegistrationController.java**

Let's take a look at the second controller that we have here in this application and that is the RegistrationController



```java
package com.mvc.annualsciencefair.controller;

import javax.servlet.http.HttpServletRequest;

@Controller
public class RegistrationController {

    @Autowired
    private UserService userService;

    @RequestMapping(value = "/performRegistration", method = RequestMethod.GET)
    public ModelAndView showRegister(HttpServletRequest request, HttpServletResponse response) {
        ModelAndView mv = new ModelAndView("register");
        mv.addObject("user", new User());
        return mv;
    }

    @RequestMapping(value = "/performRegistration", method = RequestMethod.POST)
    public String submitRegistrationForm(
                @Valid @ModelAttribute("user") User user,
                BindingResult bindingResult
            ) {

        if (bindingResult.hasErrors()) {
            return "register";
        }

        userService.register(user);

        return "redirect:performLogin";
    }

}
```

The RegistrationController also uses the UserService, which we inject using the @Autowired annotation.
The first method here map to /performRegistration which responds to http GET request is the same method as before.
What we're interested in looking as is the second method. Again map to **/performRegistration**, but one which responds to POST methods.

This is the method that is invoked when we submit our registration form. Let's take a look at the input arguments of this method, the user object is injected into this method.

```
public String submitRegistrationForm(
            @Valid @ModelAttribute("user") User user,
            BindingResult bindingResult
            ) {
```

This user object has been populated using the registration form. We've added the **@Valid** and **@ModelAttribute** annotations to this user object. The @ModelAttribute indicates that this has been bound to our registration form, and that's the same user object that's passed into the controller. The **@Valid** annotation will apply the validators that we have specified for the user object.

So far we haven't specified any validators, but we shall do so in the next demo. We've also injected in the **bindingResult**. If **bindingResult** has errors, then we'll return the register view. We will rerender the register view to the user.

```
if (bindingResult.hasErrors()) {
      return "register";
}
```

If there are no errors, we'll call userService.register and pass in the user object, and we will then redirect the user to perform login page, so that the newly registered user can log in with his new credentials.

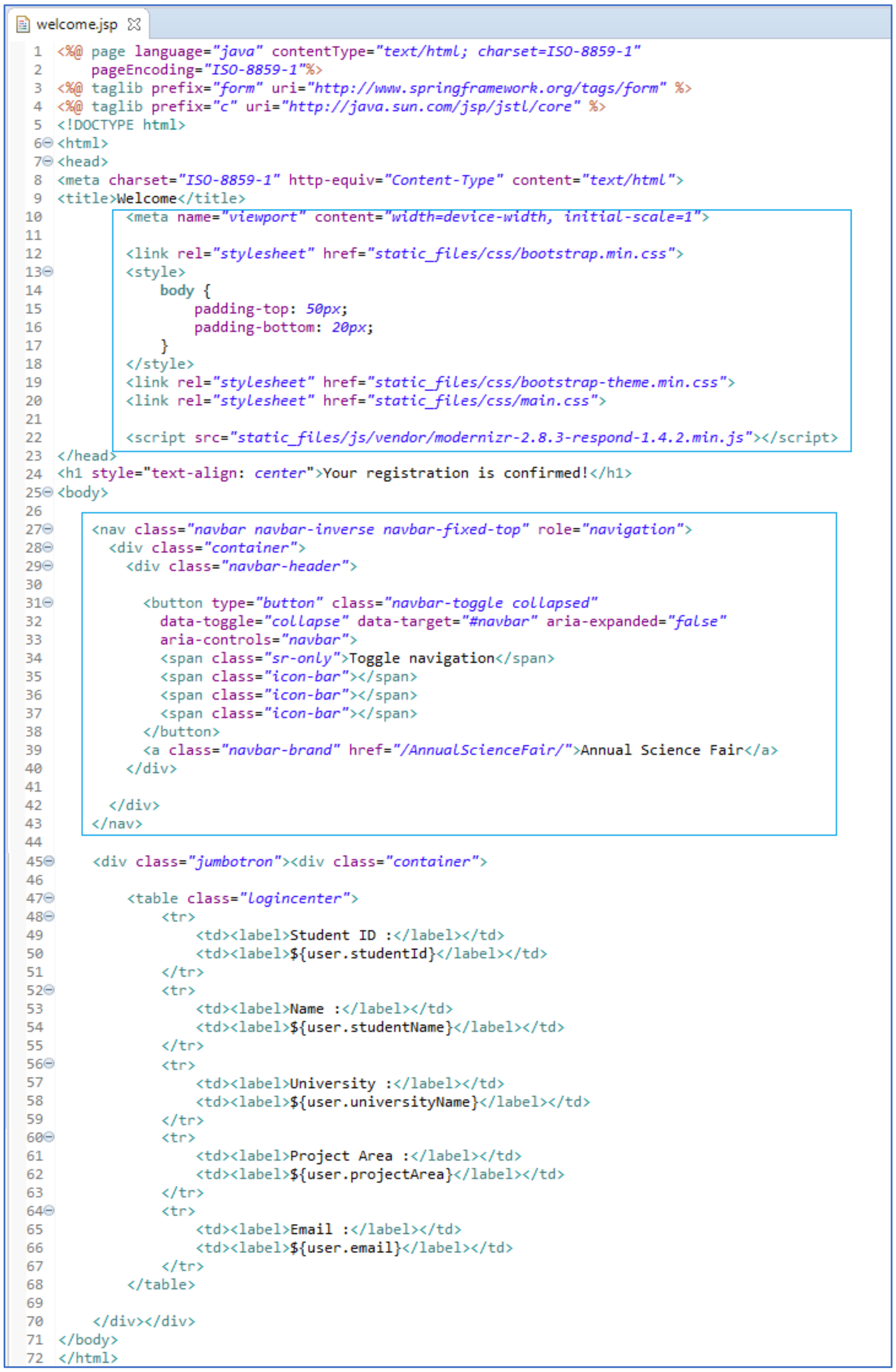
```
userService.register(user);
return "redirect:performLogin";
```

Notice how we specify a redirect using redirect:. A redirect constructs a new request to that page. It's not a continuation of the current request, which means the user object will not be available in this new request.

5. Presentation Layer, View Tier

   Most of our JSP files are the same as the previous demo. There is no change to the index.jsp file. No change to the login.jsp file. You can scroll down and see that everything is exactly the same. There is no change to the register.jsp file as well. You can scroll down and see that everything looks the same. But we have added a new file called welcome.jsp. This is the page that will be rendered when the user successfully logs in after registering for the science fair.

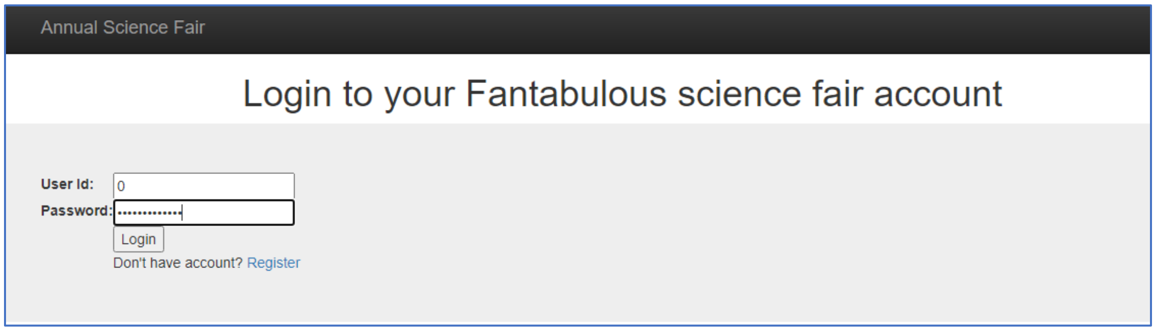
   welcome.jsp



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1" http-equiv="Content-Type" content="text/html">
<title>Welcome</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="static_files/css/bootstrap.min.css">
        <style>
            body {
                padding-top: 50px;
                padding-bottom: 20px;
            }
        </style>
        <link rel="stylesheet" href="static_files/css/bootstrap-theme.min.css">
        <link rel="stylesheet" href="static_files/css/main.css">

        <script src="static_files/js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>
</head>
<h1 style="text-align: center">Your registration is confirmed!</h1>
<body>

    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">

          <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse" data-target="#navbar" aria-expanded="false"
            aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="/AnnualScienceFair/">Annual Science Fair</a>
        </div>

      </div>
    </nav>

    <div class="jumbotron"><div class="container">

        <table class="logincenter">
            <tr>
                <td><label>Student ID :</label></td>
                <td><label>${user.studentId}</label></td>
            </tr>
            <tr>
                <td><label>Name :</label></td>
                <td><label>${user.studentName}</label></td>
            </tr>
            <tr>
                <td><label>University :</label></td>
                <td><label>${user.universityName}</label></td>
            </tr>
            <tr>
                <td><label>Project Area :</label></td>
                <td><label>${user.projectArea}</label></td>
            </tr>
            <tr>
                <td><label>Email :</label></td>
                <td><label>${user.email}</label></td>
            </tr>
        </table>

    </div></div>
</body>
</html>
```

Once again, I have a bunch of boilerplate HTML code here so that the theme across my pages looks the same. And here at the bottom, I have the details of the registration printed out to screen. The controller sends the user object down to the welcome page. We print out the studentId, studentName, universityName, the projectArea, and the email address of the student. That completes our exploration of the updates that we've made to this application.

Let's go ahead and generate a WAR file and deploy that WAR file to our Tomcat server. Here's what our app looks like now.

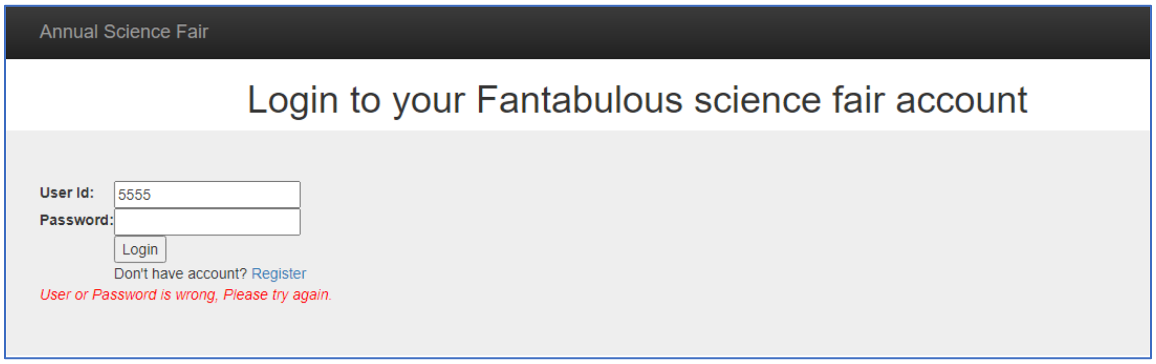


Annual Science Fair

Login to your Fantabulous science fair account

User Id: 0
Password: ••••••••••••

But this time around, we can use the Login and Register pages. Let's head over to the login page and try and log in with the student Id 5555. Specify any password and hit the Login button. And you'll find that you'll immediately get a message which says, *username or password is wrong. Please try again.*



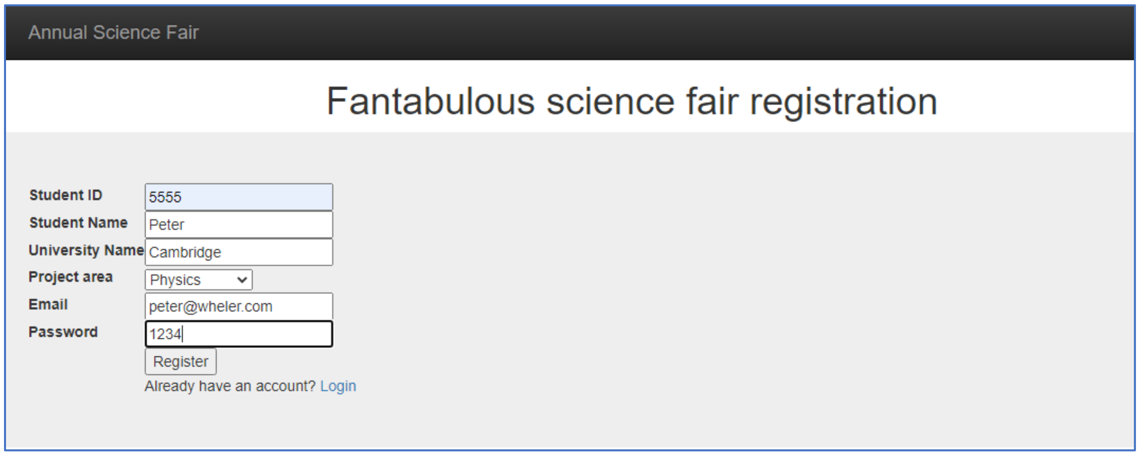
Annual Science Fair

Login to your Fantabulous science fair account

User Id: 5555
Password:

User or Password is wrong, Please try again.

Well, it stands to reason because we haven't registered yet. Let's click on this Register link here and head over to the register page and fill in some values for your registration. I'm going to fill in the registration for the student, Peter, who is at Fantabulous University. Make sure you remember the student ID and the password that you have specified, that's what we'll use to log in.



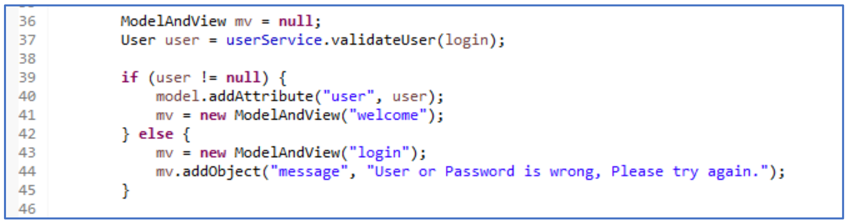
Annual Science Fair

Fantabulous science fair registration

Student ID	5555
Student Name	Peter
University Name	Cambridge
Project area	Physics
Email	peter@wheler.com
Password	1234

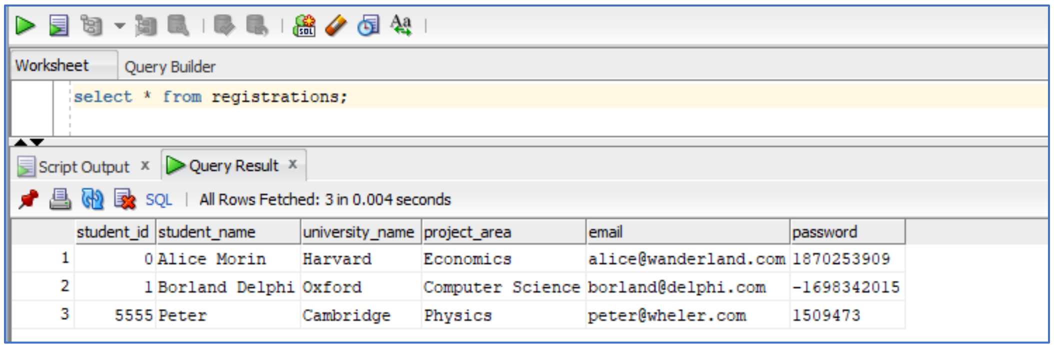
Click on the Register button, and that takes you to the login page indicating that your registration was successful. If you recall our controller code, if the form did not have any errors, and ours didn't, we register the user and redirect to the perform login page.

LoginController.java



```
        ModelAndView mv = null;
        User user = userService.validateUser(login);

        if (user != null) {
            model.addAttribute("user", user);
            mv = new ModelAndView("welcome");
        } else {
            mv = new ModelAndView("login");
            mv.addObject("message", "User or Password is wrong, Please try again.");
        }
```

Before we log in, let's head over to our MySQL Workbench and run a **SELECT * FROM registrations**.



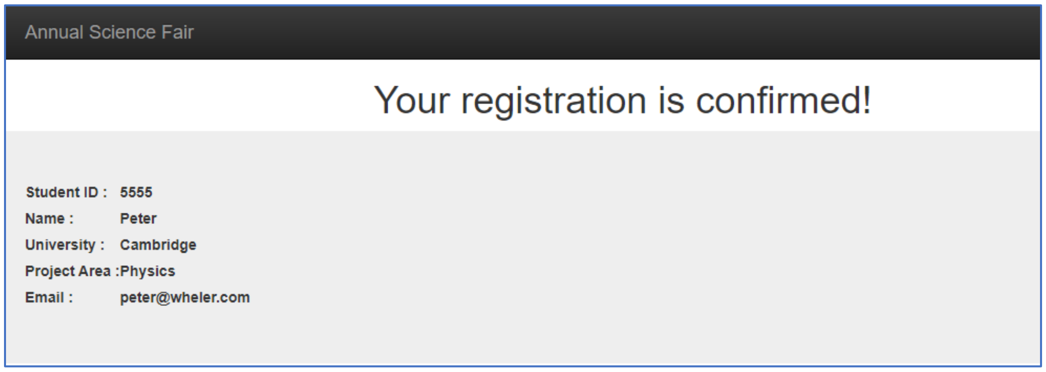
```
select * from registrations;
```

All Rows Fetched: 3 in 0.004 seconds

	student_id	student_name	university_name	project_area	email	password
1	0	Alice Morin	Harvard	Economics	alice@wanderland.com	1870253909
2	1	Borland Delphi	Oxford	Computer Science	borland@delphi.com	-1698342015
3	5555	Peter	Cambridge	Physics	peter@wheler.com	1509473

Previously, we had no registrations, but now you can see that, Peter with student ID 5555 has been successfully registered. The thing to note here is that the password column contains a hash of the password. We don't store passwords directly in our database.

Let's now login using the student Id 5555 and the password that we had specified for Peter. Click on the login button. And if your password is correct, you should be logged in successfully. And you should be able to see the details that you registered Peter with.



Annual Science Fair

Your registration is confirmed!

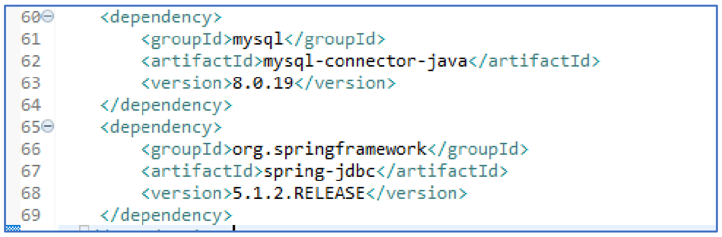
Student ID :	5555
Name :	Peter
University :	Cambridge
Project Area :	Physics
Email :	peter@wheler.com

Let's go back to our home page. Now I'm going to register yet another student. This is Paul. Go ahead and fill in all of the details for Paul and click Register. Once you hit Register, you should be redirected to the login page if registration was successful. Yes it was, let's run a SELECT * FROM Registrations on the MySQL Workbench. And you can see that our registrations table now has two records. Peter, as well as Paul, have successfully registered for the science fair.

Implementing Custom Form Validation

In this demo we'll add validators to our form. In fact, we'll go a step further, we'll also create our own custom validation annotations that we'll add to our form. Let's take a look at the pom.xml. Remember we are connecting to a MySQL database where we store the students who have registered for the annual science fair. In order to connect to the database, I've added the dependency on the mysql-connector-java and spring-jdbc templates.

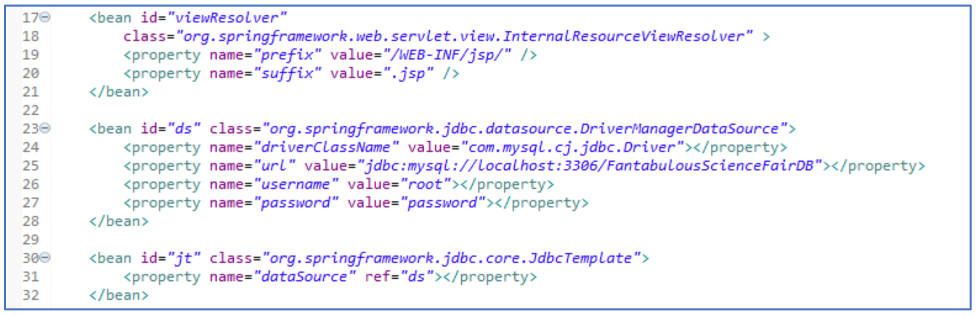
pom.xml



```xml
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.1.2.RELEASE</version>
    </dependency>
```

The configuration parameters for connecting to the database have been specified in spring-servlet.xml in the form of data source beans.

spring-servlet.xml



```xml
    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"></property>
        <property name="url" value="jdbc:mysql://localhost:3306/FantabulousScienceFairDB"></property>
        <property name="username" value="root"></property>
        <property name="password" value="password"></property>
    </bean>

    <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="ds"></property>
    </bean>
```

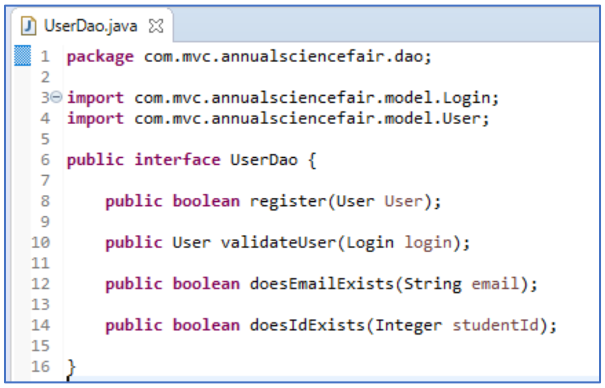
We've specified a bean for the DriverManagerDataSource that will connect to our MySQL database using JDBC. Specifically the FantabulousScienceFairDB database in our MySQL server. We'll use the root username and password.

The JDBC template that we inject into our data access layer objects will use this data source to perform the connection to the underlying database.

Now, in order to be able to use validators, I've made a few updates across the three tiers of our Spring MVC application.

1) ## Prepare Validation Method on Data Access Layer

Let's take a look at the data access object here. **UserDao.java**



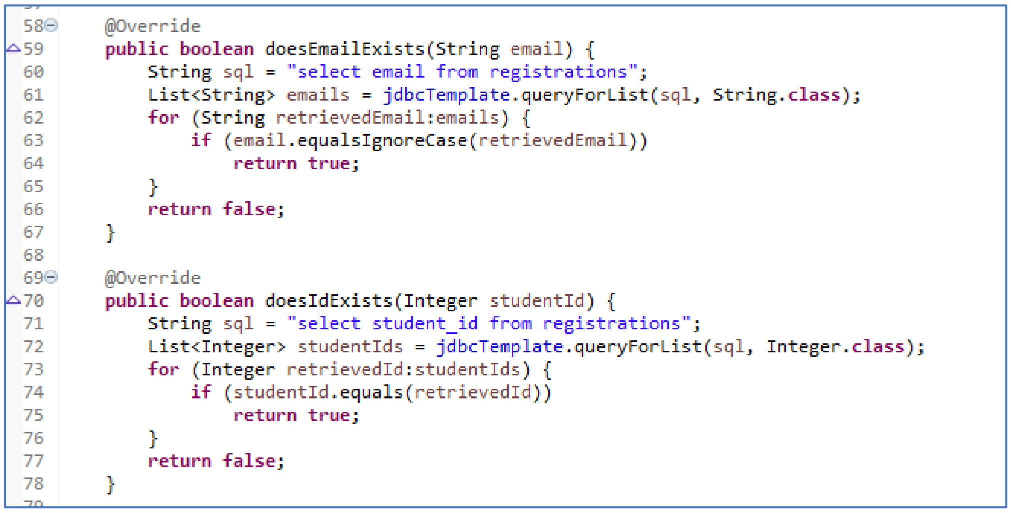
UserDao.java

```java
 1 package com.mvc.annualsciencefair.dao;
 2 
 3 import com.mvc.annualsciencefair.model.Login;
 4 import com.mvc.annualsciencefair.model.User;
 5 
 6 public interface UserDao {
 7 
 8     public boolean register(User User);
 9 
10     public User validateUser(Login login);
11 
12     public boolean doesEmailExists(String email);
13 
14     public boolean doesIdExists(Integer studentId);
15 
16 }
```

I've added two new methods here in the UserDao interface, doesEmailExist and doesIdExist. Both of these methods perform validation checks to see whether there is another registered user with the same email, or the same ID.

Since these validation checks require us to access the database, the methods for these have to be set up across the three tiers of our application. Let's take a look at the implementation of these methods in the UserDaoImpl file. This is the file that uses a jdbcTemplate in order to query our MySQL database.

UserDaoImpl.java



```java
58     @Override
59     public boolean doesEmailExists(String email) {
60         String sql = "select email from registrations";
61         List<String> emails = jdbcTemplate.queryForList(sql, String.class);
62         for (String retrievedEmail:emails) {
63             if (email.equalsIgnoreCase(retrievedEmail))
64                 return true;
65         }
66         return false;
67     }
68 
69     @Override
70     public boolean doesIdExists(Integer studentId) {
71         String sql = "select student_id from registrations";
72         List<Integer> studentIds = jdbcTemplate.queryForList(sql, Integer.class);
73         for (Integer retrievedId:studentIds) {
74             if (studentId.equals(retrievedId))
75                 return true;
76         }
77         return false;
78     }
```

We are already familiar with the register and validate user method implementations. Let's scroll down below and take a look at the implementation for doesEmailExist. The SQL query that we'll run will be to select all email addresses from the Registrations table. The SQL query is on line 50.

We retrieve the list of emails using the query for list method. After retrieving the emails, we run a for each loop through every email. We convert the retrieved email as well as the email passed in as an input argument to the lowercase representation and perform an equals check.

If any email matches, we'll return true. Otherwise, we'll get out of the for each loop and return false, there was no email that matched the input argument.

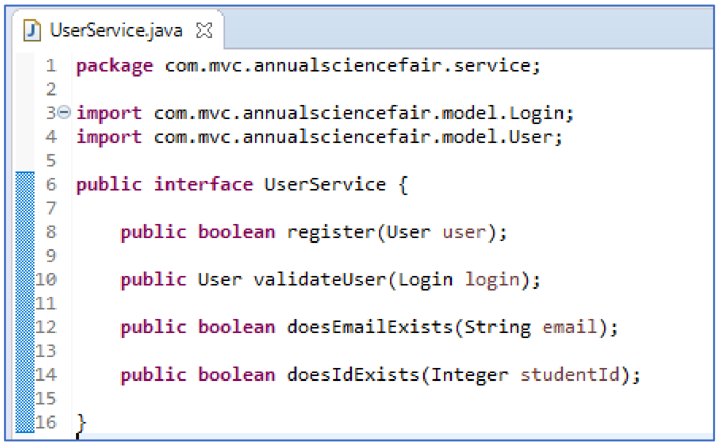
In exactly the same way, we'll perform a check for doesIdExist. We want to see if there is an already registered user with the same student ID. The input argument here is the studentId to be compared with. We'll run a SQL query select student_id from Registrations, and we'll retrieve a list of all student IDs using query for list.

We will run a for each loop through every retrieved Id to see whether it matches the studentId that we passed in for comparison. If there is a match at any point we'll return true, otherwise when we get out of the loop, we'll return false.

2) Prepare Logic Validation Method on Service Layer

In order to access these methods in the controller, we need to update our service layer classes as well here is the interface UserService.

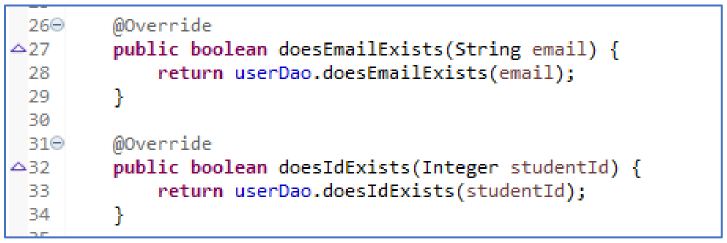
UserService.java



```java
package com.mvc.annualsciencefair.service;

import com.mvc.annualsciencefair.model.Login;
import com.mvc.annualsciencefair.model.User;

public interface UserService {

    public boolean register(User user);

    public User validateUser(Login login);

    public boolean doesEmailExists(String email);

    public boolean doesIdExists(Integer studentId);

}
```

We've added the two methods doesEmailExist and doesIdExist to our service interface as well. The implementation of our service, UserServiceImpl simply calls the corresponding Dao for the implementation. All implementation is delegated to the Dao object.

UserDaoImpl.java



```java
    @Override
    public boolean doesEmailExists(String email) {
        return userDao.doesEmailExists(email);
    }

    @Override
    public boolean doesIdExists(Integer studentId) {
        return userDao.doesIdExists(studentId);
    }
```

Before we get to the interesting stuff that is the custom validators that we have set up, let's quickly look at our controller code. There's absolutely no change here, we have the LoginController that displays the login page, that is the showLogin method.

And we have the login process method where we check the student Id and password to see whether this is a valid user. There is no change in the LoginController code and there is no change in the ReistrationController code as well. We have the first method which shows the registration form and the second method which actually registers a user.

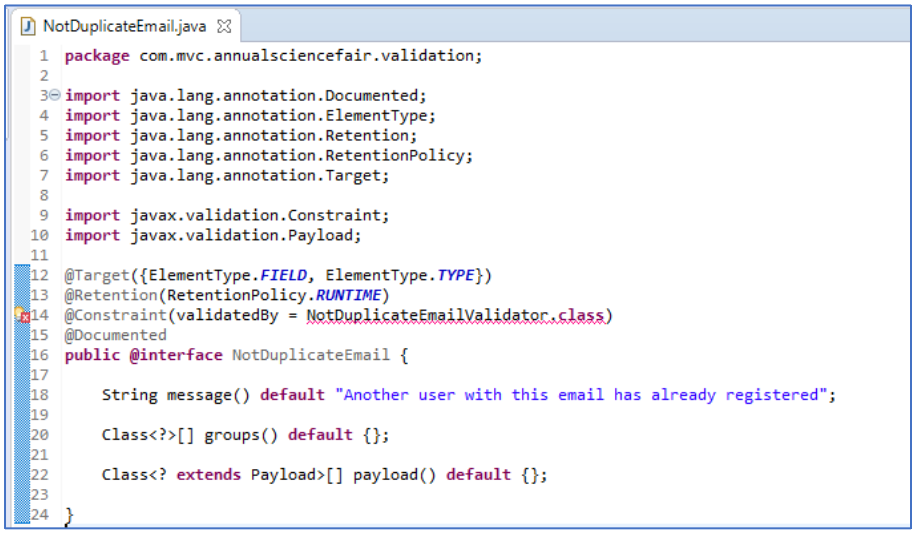
3) Create Custom Validator Component

Now let's get to the interesting stuff, our custom validations. All the custom validators have been placed in the **com.mvc.annualsciencefair.validation** package.

- **Not Duplicate Email** Validator

  Let's take a look at the annotation for our first custom validator, **NotDuplicateEmail**. You can see that I have defined it as an annotation using public **@interface**.

  NotDuplicateEmail.java



```
NotDuplicateEmail.java
 1 package com.mvc.annualsciencefair.validation;
 2 
 3 import java.lang.annotation.Documented;
 4 import java.lang.annotation.ElementType;
 5 import java.lang.annotation.Retention;
 6 import java.lang.annotation.RetentionPolicy;
 7 import java.lang.annotation.Target;
 8 
 9 import javax.validation.Constraint;
10 import javax.validation.Payload;
11 
12 @Target({ElementType.FIELD, ElementType.TYPE})
13 @Retention(RetentionPolicy.RUNTIME)
14 @Constraint(validatedBy = NotDuplicateEmailValidator.class)
15 @Documented
16 public @interface NotDuplicateEmail {
17 
18     String message() default "Another user with this email has already registered";
19 
20     Class<?>[] groups() default {};
21 
22     Class<? extends Payload>[] payload() default {};
23 
24 }
```

  The target for this validation is **ElementType FIELD** and **ElementType TYPE**. The retention policy is **RUNTIME** retention. And the **@Constraint** annotation on line 14, specifies the class that provides an implementation for this annotation.

```
@Target({ElementType.FIELD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = NotDuplicateEmailValidator.class)
@Documented
```

  The NotDuplicateEmail Validator class will actually perform the validation to check that an email address doesn't already exist in the Registrations table. This is just the annotation specification, the API.

```
public @interface NotDuplicateEmail
```

  We've set up this annotation to accept three properties all of this code is quite boilerplate. We have a message, that is the message that will be displayed when the validation fails. We've have a default value for this message, another user with this email has already registered.

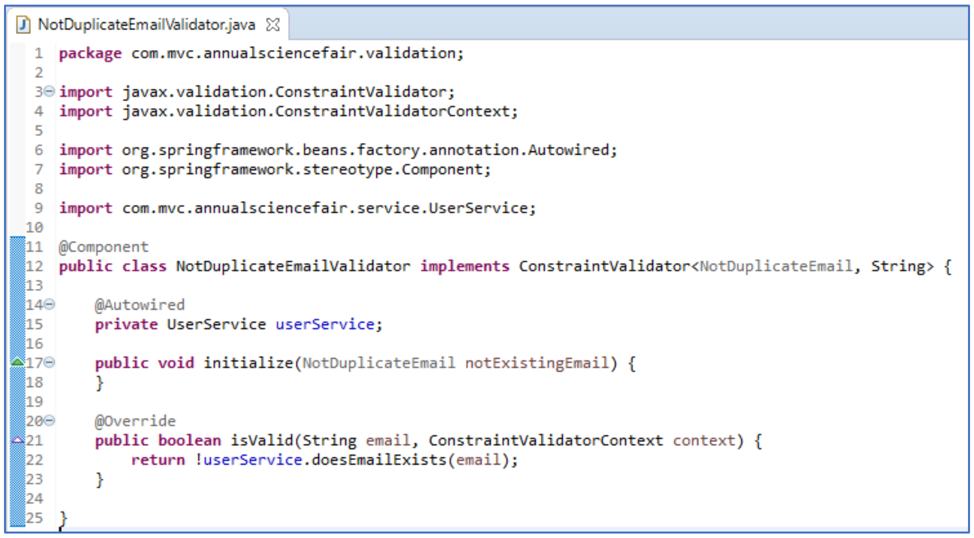
```
String message() default "Another user with this email has already
registered";
```

All validation annotations have to specify the group's property as well as the payload property with default set to empty array.

```
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {};
```

Now let's switch over and take a look at the implementation of this validator which is the NotDuplicateEmail Validator class.

NotDuplicateEmailValidator.java



NotDuplicateEmailValidator.java

```
package com.mvc.annualsciencefair.validation;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

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

import com.mvc.annualsciencefair.service.UserService;

@Component
public class NotDuplicateEmailValidator implements ConstraintValidator<NotDuplicateEmail, String> {

    @Autowired
    private UserService userService;

    public void initialize(NotDuplicateEmail notExistingEmail) {
    }

    @Override
    public boolean isValid(String email, ConstraintValidatorContext context) {
        return !userService.doesEmailExists(email);
    }

}
```

Notice that it implements the interface ConstraintValidator, which is a generic interface. The first generic parameter NotDuplicateEmail corresponds to the annotations API for this validator. The second generic parameter defines the type of data to which this validator annotation is applied, String data in our case.

```
@Component
public class NotDuplicateEmailValidator implements
ConstraintValidator<NotDuplicateEmail, String> {
```

In order to perform validation this class needs the UserService, which we inject into this class using **@Autowired**.

```
@Autowired
private UserService userService;
```

We then set up the initialize method, which takes as an input argument the NotDuplicateEmail annotation.

```
public void initialize(NotDuplicateEmail notExistingEmail) {
}
```

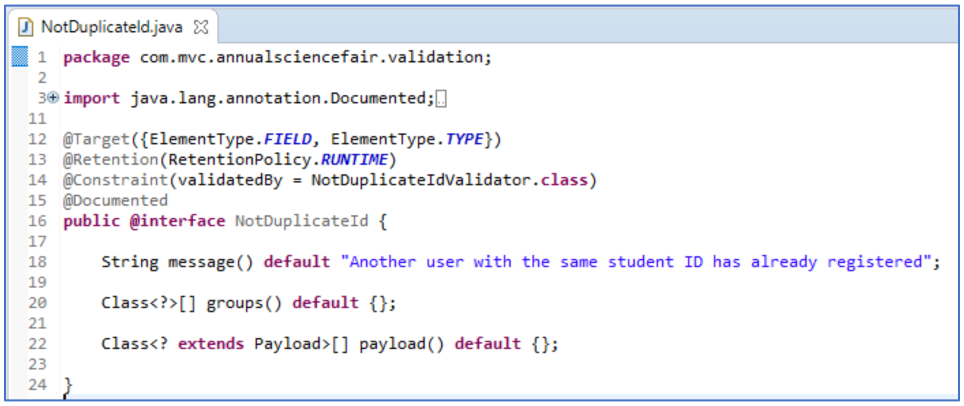
We do nothing in the initialize method, but you can add some code here if you require to initialize something before performing the validation.

Our actual validation code is in the isValid method.

```
public boolean isValid(String email, ConstraintValidatorContext context)
{
        return !userService.doesEmailExists(email);
}
```

It takes as an input argument, the email address that is a **String** and a **ConstraintValidatorContext**. It then uses the **UserService** to check whether the email exists. And we've already seen how that is implemented. If the email doesn't already exist in our database, we'll return true and things will be fine, otherwise this validation will fail.

- **Not Duplicate ID Validator**
  We set up our second custom validator in exactly the same way. The NotDuplicateIdValidator, here is the annotation specification.
  NotDuplicateId.java



```
1  package com.mvc.annualsciencefair.validation;
2
3⊕ import java.lang.annotation.Documented;
11
12 @Target({ElementType.FIELD, ElementType.TYPE})
13 @Retention(RetentionPolicy.RUNTIME)
14 @Constraint(validatedBy = NotDuplicateIdValidator.class)
15 @Documented
16 public @interface NotDuplicateId {
17
18     String message() default "Another user with the same student ID has already registered";
19
20     Class<?>[] groups() default {};
21
22     Class<? extends Payload>[] payload() default {};
23
24 }
```

Notice the **@Constraint** annotation on line 14. The actual implementation of this validator is in the **NotDuplicateIdValidator** class.

```
@Constraint(validatedBy = NotDuplicateIdValidator.class)
```

This validator also accepts three properties. The default message says "Another user with the same student ID has already registered."

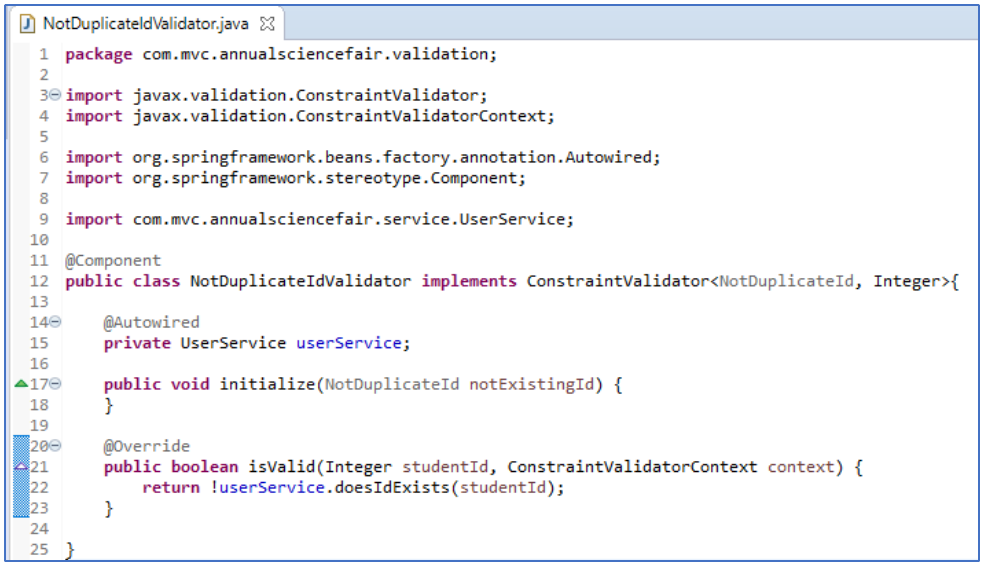
```
String message() default "Another user with the same student ID has
already registered";
```

We also have the groups and payload property, all part of our boilerplate code.

```
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {};
```

Let's take a look at the actual validator, the NotDuplicateIdValidator which implements the ConstraintValidator interface.

NotDuplicateIdValidator.java



NotDuplicateIdValidator.java

```
package com.mvc.annualsciencefair.validation;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

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

import com.mvc.annualsciencefair.service.UserService;

@Component
public class NotDuplicateIdValidator implements ConstraintValidator<NotDuplicateId, Integer>{

    @Autowired
    private UserService userService;

    public void initialize(NotDuplicateId notExistingId) {
    }

    @Override
    public boolean isValid(Integer studentId, ConstraintValidatorContext context) {
        return !userService.doesIdExists(studentId);
    }

}
```

The first generic type is NotDuplicateId that is the annotation corresponding to this validation implementation. And the second generic type is Integer indicating that this validation is applied to integer fields.

```
@Component
public class NotDuplicateIdValidator implements
ConstraintValidator<NotDuplicateId, Integer>{
```

This validator also uses the UserService to check for duplicate IDs. Take a look at the isValid code.

```
@Autowired
private UserService userService;
```

It takes as an input argument, the integer id and the ConstraintValidator context, we use the UserService to see whether the same Id already exists.

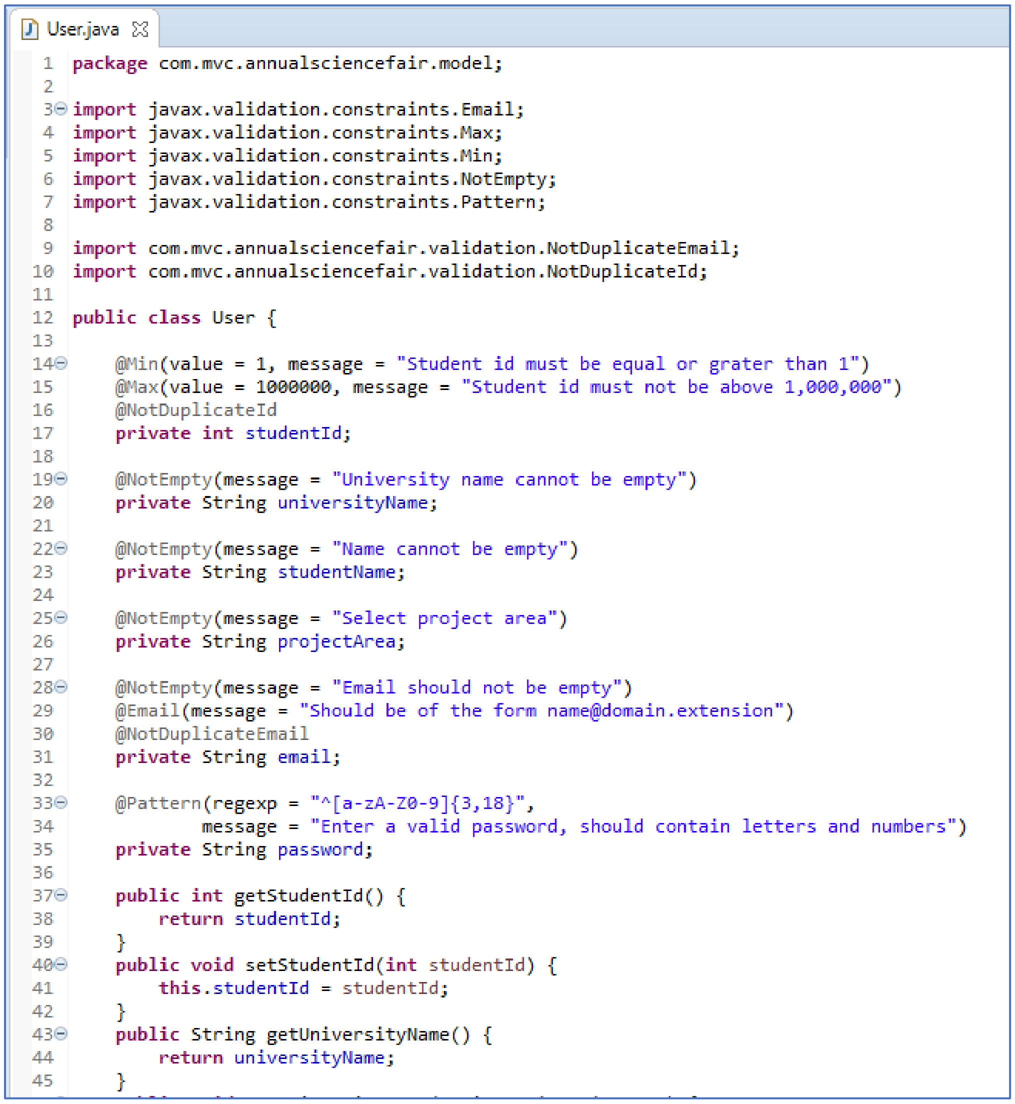
```
public boolean isValid(Integer studentId, ConstraintValidatorContext
context){
        return !userService.doesIdExists(studentId);
}
```

If the Id doesn't exist, we'll return true, indicating that the validation was successful, otherwise we'll return false indicating validation failed.

4) Implementing Validation on Model Object

Now that we've set up these custom validators, let's apply these to the fields of our model object.

Model **User.java**



User.java

```java
package com.mvc.annualsciencefair.model;

import javax.validation.constraints.Email;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Pattern;

import com.mvc.annualsciencefair.validation.NotDuplicateEmail;
import com.mvc.annualsciencefair.validation.NotDuplicateId;

public class User {

    @Min(value = 1, message = "Student id must be equal or grater than 1")
    @Max(value = 1000000, message = "Student id must not be above 1,000,000")
    @NotDuplicateId
    private int studentId;

    @NotEmpty(message = "University name cannot be empty")
    private String universityName;

    @NotEmpty(message = "Name cannot be empty")
    private String studentName;

    @NotEmpty(message = "Select project area")
    private String projectArea;

    @NotEmpty(message = "Email should not be empty")
    @Email(message = "Should be of the form name@domain.extension")
    @NotDuplicateEmail
    private String email;

    @Pattern(regexp = "^[a-zA-Z0-9]{3,18}",
            message = "Enter a valid password, should contain letters and numbers")
    private String password;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getUniversityName() {
        return universityName;
    }
```

We'll use a mix of built-in validators that we've encountered before, as well as the custom validators that we've created. Take a look at the student Id field.

```java
@Min(value = 1, message = "Student id must be equal or grater than 1")
@Max(value = 1000000, message = "Student id must not be above 1,000,000")
@NotDuplicateId
```

There are three validators for this field, **@Min** and **@Max**, indicating that the student Id should be between 1 and 1 million.

We also have our custom **NotDuplicateId** annotation. Remember, this will be validated using the underlying database.

I have the universityName member variable to which I have applied the **@NotEmpty** validation, the university name cannot be empty.

```
@NotEmpty(message = "University name cannot be empty")
private String universityName;
```

To the studentName field, I have the same validation applied, **@NotEmpty**, name cannot be empty as well.

```
@NotEmpty(message = "Name cannot be empty")
private String studentName;
```

You can scroll down below and see other fields have validations as well. The @NotEmpty is applied to the projectArea.

```
@NotEmpty(message = "Select project area")
private String projectArea;
```

The email field has multiple validators, @NotEmpty, @Email, as well as @NotDuplicateEmail.

```
@NotEmpty(message = "Email should not be empty")
@Email(message = "Should be of the form name@domain.extension")
@NotDuplicateEmail
private String email;
```

No two email addresses can be the same in our registrations table.

We also have an additional constraint on the passwords that user specify. Passwords can only be made up of letters and numbers. I use an @Pattern validator, which allows me to specify a regular expression for valid passwords.

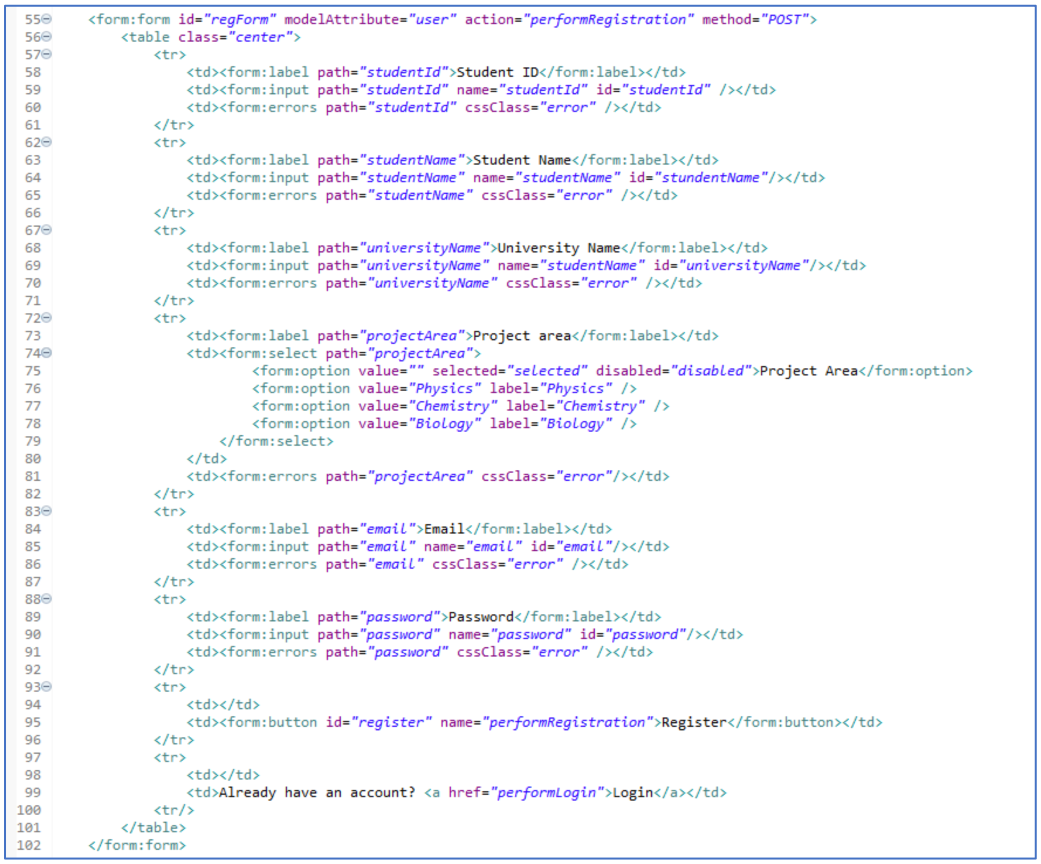
```
@Pattern(regexp = "^[a-zA-Z0-9]{3,18}",
message = "Enter a valid password, should contain letters and numbers")
private String password;
```

This was the new stuff in our app. All the remaining code is the same getters and setters for the user object.

Let's take a look at the Login object, there is absolutely no change here.

5) Placing an Errors Tags on JSP pages

Let's take a look at some of the UI that we have. Here is login.jsp, absolutely no change here as well. Here is register.jsp, no change in this file as well.

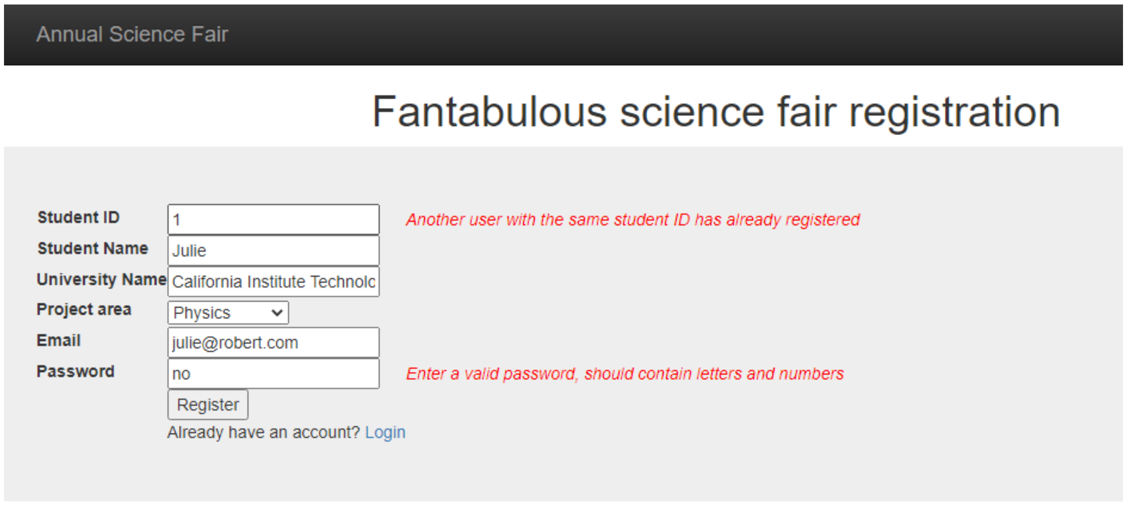


```
<form:form id="regForm" modelAttribute="user" action="performRegistration" method="POST">
    <table class="center">
        <tr>
            <td><form:label path="studentId">Student ID</form:label></td>
            <td><form:input path="studentId" name="studentId" id="studentId" /></td>
            <td><form:errors path="studentId" cssClass="error" /></td>
        </tr>
        <tr>
            <td><form:label path="studentName">Student Name</form:label></td>
            <td><form:input path="studentName" name="studentName" id="stundentName"/></td>
            <td><form:errors path="studentName" cssClass="error" /></td>
        </tr>
        <tr>
            <td><form:label path="universityName">University Name</form:label></td>
            <td><form:input path="universityName" name="studentName" id="universityName"/></td>
            <td><form:errors path="universityName" cssClass="error" /></td>
        </tr>
        <tr>
            <td><form:label path="projectArea">Project area</form:label></td>
            <td><form:select path="projectArea">
                    <form:option value="" selected="selected" disabled="disabled">Project Area</form:option>
                    <form:option value="Physics" label="Physics" />
                    <form:option value="Chemistry" label="Chemistry" />
                    <form:option value="Biology" label="Biology" />
                </form:select>
            </td>
            <td><form:errors path="projectArea" cssClass="error"/></td>
        </tr>
        <tr>
            <td><form:label path="email">Email</form:label></td>
            <td><form:input path="email" name="email" id="email"/></td>
            <td><form:errors path="email" cssClass="error" /></td>
        </tr>
        <tr>
            <td><form:label path="password">Password</form:label></td>
            <td><form:input path="password" name="password" id="password"/></td>
            <td><form:errors path="password" cssClass="error" /></td>
        </tr>
        <tr>
            <td></td>
            <td><form:button id="register" name="performRegistration">Register</form:button></td>
        </tr>
        <tr>
            <td></td>
            <td>Already have an account? <a href="performLogin">Login</a></td>
        <tr/>
    </table>
</form:form>
```

welcome.jsp also remains the same.

We are now ready to build our WAR file and deploy to our Tomcat server, and see how our form works with all of the new validations applied.

Let's test out the validation that we've added to our form. Click on the Register button and head over to the register form and fill in some details for a student.

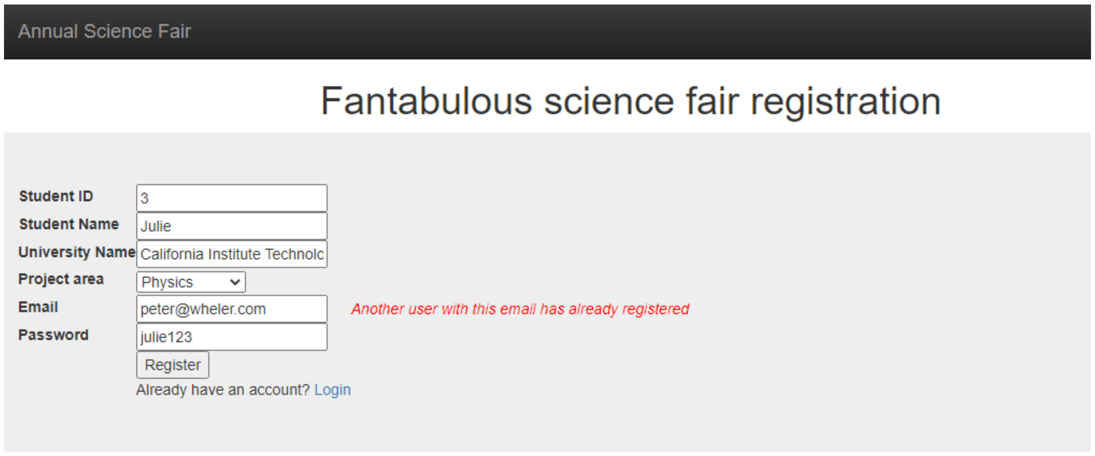

Annual Science Fair

Fantabulous science fair registration

Student ID	1	Another user with the same student ID has already registered
Student Name	Julie	
University Name	California Institute Technolc	
Project area	Physics	
Email	julie@robert.com	
Password	no	Enter a valid password, should contain letters and numbers

Some of these details will be invalid. Here the student ID is invalid, and I've also specified a password with special characters. Click on Register, and there you see it, the error messages. The Min validator on the student ID field basically says student Id must be equal to or greater than 1. And the Pattern validator on the password field says that the pattern should contain only letters or numbers. Our built-in validators for these two fields seem to work just fine.

Let's try out one of our custom validators. I'm going to fix the Student ID field so that it is a numeric value that is valid. Instead, I'm going to have Julie register with an email that already exists in our database, *peter@wheler.com*. Hit Register,


Annual Science Fair

Fantabulous science fair registration

Student ID	3	
Student Name	Julie	
University Name	California Institute Technolc	
Project area	Physics	
Email	peter@wheler.com	Another user with this email has already registered
Password	julie123	

and there is the error. Another user with email has already registered. This is thanks to our custom validator that we wrote code for, @NotDuplicateEmail, that we've applied to the email field in our user object.

we'll try this one last time. This time, we'll have Julie register with an ID that already exist in our database, the ID 5555. Hit Register, and our custom validator works here as well. You can see the error message, another user with the same student ID has already registered. This is thanks to the @NotDuplicateId validator that we have applied to the student ID field.

Quiz

1. If you are using starter templates for your application, what do you need to watch out for in the boilerplate HTML code?
   ✓ Update references to the CSS and other assets to point to your location
   Update the pom.xml file to include CSS dependencies
   Update the boilerplate code to use your code
   Update the contents of the HTML to represent what you want in your site

2. Which of the following JSP elements specifies a password input?
   <form:protected>
   ✓ <form:password>
   <form:masked>
   <form:input type=password>

3. What is the name of the graphical user interface that we can use to connect to a MySQL database server?
   ✓ MySQL Workbench
   MySQL UI
   MySQL View
   MySQL Apache

4. What is the role of MySQL Workbench when working with a MySQL database server?
   ✓ Presents a user interface to create tables, run queries
   A debugging tool to debug issues with your MySQL tables
   Used to configure table metadata in your relational tables
   A performance management tool for MySQL

5. Where do we specify the details of the MySQL connection when working with Spring?
   ✓ Servlet configuration file
   Data access object class
   Service class
   Controller class

6. What does a RowMapper in JDBC do?
   Applies update transformations to rows in a database table
   Inserts rows into a database table by extracting details from an object
   Retrieves rows in the form of Row objects from a relational database table
   ✓ Converts a row retrieved from a relational database to a specific object representation

7. How do you typically store passwords in a database table?
   ✔ Use a one-way hash of the password
   Reverse the password
   You store passwords in sessions and not database tables
   You never store passwords

8. Which annotation on the validation annotation definition gives us information about which class implements the validation?
   @ModelAttribute
   @Validator
   @Valid
   ✔ @Constraint

9. Which of the following is NOT a validator annotation that Spring has built-in support for?
   @NotNull
   @Email
   ✔ @NotDuplicateId
   @Size

10. Which annotation would you apply to an object in the data access layer to mark it as an injectable bean?
    @Service
    @Component
    @Dao
    ✔ @Repository

11. Which two request mappings are equivalent?
    @GetMapping
    @RequestMapping(method=RequestMethod.PUT)
    ✔ @RequestMapping(method=RequestMethod.POST)
    ✔ @PostMapping

12. What does the <form:hidden> tag do?
    Hides the entire form from the user
    Disables the field within a form
    Used for irrelevant fields in a form
    ✔ Hides a field from the user, who cannot see it or modify it

13. The deletion operation is what kind of HTTP request?
    GET
    OPTIONS
    ✔ POST
    HEAD