**CST-341 Activity Guide**

**Special Note:** Each activity will build on the previous set of activities ending up with a single application (built incrementally).

Contents

[Activity 1: Installation and Validation 2](#_Toc529770302)

[Assignment 1A: Tools Installation & Validation 2](#_Toc529770303)

[Activity 2: Spring MVC 3](#_Toc529770304)

[Assignment 2A (Part 1): Configure a Spring MVC Application 3](#_Toc529770305)

[Assignment 2A (Part 2): Creating Models, Views, and Controllers in Spring MVC 4](#_Toc529770306)

[Assignment 2B: Creating Forms in Spring MVC 6](#_Toc529770307)

[Assignment 2C: Creating Layouts in Spring MVC 10](#_Toc529770308)

[Activity 3: SpringBeans & DI 12](#_Toc529770309)

[Assignment 3A: Design and Implement SpringBeans 12](#_Toc529770310)

[Assignment 3B: SpringBeans Lifecycle and Scope 14](#_Toc529770311)

[Assignment 3C: SpringBeans Configuration Annotations 15](#_Toc529770312)

[Assignment 3D: REST Based Services 17](#_Toc529770313)

[Assignment 3E: Open Source Technology Research 18](#_Toc529770314)

[Activity 4: SpringJDBC & JavaDB 20](#_Toc529770315)

[Assignment 4A: Setting Up JavaDB (Derby) Database 20](#_Toc529770316)

[Assignment 4B: Using JDBC to Query Data from the JavaDB (Derby) Database 22](#_Toc529770317)

[Assignment 4C: Using Spring JDBC to Query Data from the JavaDB (Derby) Database 24](#_Toc529770318)

[Activity 5: Maven 25](#_Toc529770319)

[Assignment 5A: Convert a Project to Maven 25](#_Toc529770320)

[Activity 6: jQuery 26](#_Toc529770321)

[Assignment 6A: Working with jQuery 26](#_Toc529770322)

[Assignment 6B: Working with jQuery Data Table 31](#_Toc529770323)

[Activity 7: Bootstrap 32](#_Toc529770324)

[Assignment 7A: Learning Bootstrap 32](#_Toc529770325)

[Assignment 7B: Making ICA Application Responsive 33](#_Toc529770326)

# Activity 1: Installation and Validation

## Assignment 1A: Tools Installation & Validation

**Overview**

In this activity, students will demonstrate the ability to install and validate Enterprise Java development environment.

**Execution**

Execute this assignment according to the following guidelines:

1. Follow the instructions in the ‘Tools Install Guide’ located in the course materials.
2. Create a new Dynamic Web Application project by selecting the File->New->Dynamic Web Application menu options. Name your project *assignment1a*. Make sure Apache Tomcat v8.5 is your target runtime. Click the finish button.
3. Create a new Java Servlet by selecting the File->Servlet menu options. Name your Servlet Test (using a package is optional). Click the finish button.
4. Start your Tomcat Server and ensure there are no startup errors.
5. Build and run the *Test* Servlet on the Tomcat Server by selecting the Test class then click the Run->Run As->Run on Server menu options. Select the Tomcat Server and click the finish button.
   1. Validate that the Servlet application can be deployed to the Tomcat server.
   2. Validate that the Servlet application is functional.
   3. Take a screen shot.
6. Change the String from “Served at: “ to “Hello World” that is returned from doGet() method from the *Test* Servlet. Validate that the Hot Deployment functions properly, the application is automatically deployed to the Tomcat server when code changes are made, and the server does not require a restart when code changes are made.
7. Take a screen shot.

**Submission**

Submit the following to the learning management system:

1. Screen shot of the *Test* Servlet application running in the IDE internal browser with the screen shot including the browser address bar.
2. Screen shot of the Console tab output of the *Test* Servlet application and after code changes have been made to the Test.java file.

# Activity 2: Spring MVC

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission. **Note:** Various resources are available within the required readings to provide assistance when completing each assignment.

## Assignment 2A (Part 1): Configure a Spring MVC Application

**Overview**

In this activity, students will demonstrate the ability to configure a Dynamic Web Application for Spring MVC.

**Execution**

Execute this activity according to the following guidelines:

1. Install Eclipse and the JDK per the “Tools Install Guide,” located within the course materials.
2. Download Apache Commons Logging at <https://commons.apache.org/logging>?
3. Download the latest release Spring Distribution at <https://repo.spring.io/release/org/springframework/spring>
4. Start Eclipse.
5. Create a new Eclipse Project:
   1. Create a new Dynamic Web Project by selecting the File->New->Dynamic Web Project menu items.
   2. Name your Project as *assignment2a*, make sure the Target runtime is set to Apache Tomcat 8.5, and click the finish button.
   3. Copy and paste (or drag and drop) the following files into the WebContent/WEB-INF/lib directory of your project:

From the Tomcatcat8.5/lib folder:

servlet-api.jar

From the Apache Logging folder (where x.y.x is your download version):

commons-logging-x.y.z.jar

From the Spring/libs folder (where x.y.x is your download version):

spring-aop-x.y.z.jar, spring-beans-x.y.z.jar, spring-context-x.y.z.jar

spring-core-x.y.z.jar, spring-expression-x.y.z.jar

spring-webmvc-x.y.z.jar, spring-web-x.y.z.jar

1. Configure a Spring Application (web.xml):
   1. Use the resource provided by the instructor as a template for the web.xml file which is located in the WebContent/WEB-INF folder
2. Configure a Spring Application (applicationConfiguration.xml):
   1. Create a new file named applicationConfiguration.xml in the WebContent/WEB-INF folder. A new file can be created by right clicking on the WEB-INF folder and selecting the New->File menu options.
   2. Use the template provided by the instructor for the applicationConfiguration.xml file.
3. Test your application:
   1. Select your Servers tab in Eclipse.
   2. Right click on the Tomcat Server and select the Add and Remove….context menu.
   3. Add the assignment2a application to the configured section.
   4. Click finish.
   5. Start your Tomcat Server.
   6. Ensure there are not startup errors on the Console.
   7. Take a screen shot of the Console.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screen shot of the browser display from Part 1 Step 8g.

## Assignment 2A (Part 2): Creating Models, Views, and Controllers in Spring MVC

**Overview**

In this activity, students will demonstrate the ability to create controllers and bind data to views via models.

**Execution**

Execute this assignment according to the following guidelines:

1. Build a simple Hello World Controller (simply controller method):
   1. Create a new Java class named *HelloWorldController* in the com.gcu package. Click the finish button.
   2. Add the @Controller and @RequestMapping("/hello") annotations at the class level.
   3. Add a public method *sayHello()* that returns a String. Implement the method by simply returning "Hello World!"
   4. Add the @RequestMapping(path = "/test1", method = RequestMethod.GET) and @ResponseBody annotations to the *sayHello()* method.
   5. Test the Controller by going to the localhost:8080/assignment2a/hello/test1 URL. Take a screenshot of browser output.
2. Enhance the simple Hello World Controller (simply controller method with Model Map passed in):
   1. Add a public method *sayHello(ModelMap model)* that returns a String. Implement the method by simply returning a view as a String named *hello*. Also, call the addAttribute(“message”, “Hello Spring MVC”) method on the model parameter.
   2. Add the @RequestMapping(path = "/test2", method = RequestMethod.GET) annotations to the *sayHello()* method.
   3. Create a folder under the WEB-INF folder called *pages*.
   4. Create a new JSP file in the WEB-INF/pages folder named *hello.jsp*. This can be done by right clicking on the WEB-INF/pages folder and selecting the New->JSP File menu options.
   5. Add a code snippet <h2>${message}</h2> to the body of the JSP.
   6. Test the Controller by going to the localhost:8080/assignment2a/hello/test2 URL. Take a screenshot of browser output.
3. Enhance the simple Hello World Controller (simply controller method with created model and view):
   1. Add a public method *sayHello1()* that returns a ModelAndView object. Implement the method by creating an instance of a ModelAndView object. The ModelAndView can be created as follows:
      1. Create an instance of ModelAndView object.
      2. Call the addObject(“message”, “Hello World from ModelAndView”) on the ModelAndView object created.
      3. Call the setViewName(“hello”) on the ModelAndView object created
   2. Add the @RequestMapping(path = "/test3", method = RequestMethod.GET) annotations to the *sayHello1()* method.
   3. Test the Controller by going to the localhost:8080/assignment2a/hello/test3 URL. Take a screenshot of browser output.
4. Enhance the simple Hello World Controller (simply controller method with a HTTP Request Parameter and Model Map passed in as parameters):
   1. Add a public method *sayHello(@RequestParam(“message) String message, ModelMap model)* that returns a String. Implement the method by simply returning a view as a String named *hello*. Also, call the addAttribute(“message”, message) method passing the passed in model back to the view.
   2. Add the @RequestMapping(path = "/test4", method = RequestMethod.GET) annotations to the *sayHello()* method.
   3. Add an anchor tag to the *hello.jsp* with an href of test4?Your message here.
   4. Test the Controller by going to the localhost:8080/assignment2a/hello/test4 URL. Take a screenshot of browser output.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screen shot of the browser display from Part 2 Step 1e.
2. Screen shot of the browser display from Part 2 Step 2f.
3. Screen shot of the browser display from Part 2 Step 3c.
4. Screen shot of the browser display from Part 2 Step 4d.

## Assignment 2B: Creating Forms in Spring MVC

**Overview**

In this activity, students will demonstrate the ability to create Forms and perform Data Validation on posted Form data.

**Execution**

Execute this assignment according to the following guidelines:

1. Create new project named *assignment2b*. Copy and paste all of the jar files from the WEB-INF/lib directory and the web.xml as well as applicationConfiguration.xml files from your *assignment2a* project into the new project. Create a folder called *pages* that resides under the WEB-INF directory. You will also need to remove the old project from the Tomcat Server and add your new project to the Server.
2. Create the User Object Model:
   1. Create a new Java Class named *User* in the com.gcu.model package.
   2. Add private firstName and lastName properties of type String with getters and setters (use the Eclipse IDE to generate the getters/setters).
   3. Add a default constructor that initializes all private properties to their default values.
   4. Add a non-default constructor that initializes all private properties (use Eclipse IDE to generate the constructor).
3. Create the Controller and Views to add and display a User:
   1. Create a new Java Class named *UserController* in the com.gcu.controller package.
   2. Add a public *displayForm()* method that returns a ModelAndView. Implement the method by returning an instance of a ModelAndView that uses its constructor that provided a View and Model.
      1. Return new ModelAndView(“addUser”, “user”, new User())
   3. Add the @RequestMapping(path = "/add", method = RequestMethod.GET) annotations to the *displayForm()* method.
   4. Create a new JSP file *addUser.jsp* within the WEB-INF/pages directory. Add test message “I am here” to body of the JSP.
   5. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL and validate that the correct response is displayed in your browser.
   6. Update the *addUser.jsp* to add a User Entry Form:
      1. Add the Spring tag library declaration to the top of the *addUser.jsp*.

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

* + 1. Create a Spring MVC compliant form using the form:form tag that has a method attribute of “POST”, an action attribute of “adduser”, and a modelAttribute set to “user”.
    2. Add 2 labels, First Name and Last Name, to the form using the form:label tag.
    3. Add 2 text controls to the form using the form:input tag and set the path attribute for each control to match the corresponding firstName and lastName properties from the *User* model.
    4. Add 1 submit button to the form using an HTML input tag.
    5. Use a HTML table to control the layout of the form.
  1. Update the Controller:
     1. Add a public *addUser(@ModelAttribute(“user” User user)* method that returns a ModelAndView.
     2. Add the @RequestMapping(path = "/adduser", method = RequestMethod.POST) annotations to the addUser() method.
     3. Implement the method by returning an instance of a ModelAndView that uses its constructor that provided a View and Model.
        1. Return new ModelAndView(“addUser”, “user”, user)
        2. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL, submit the User Form, and validate that the correct response is displayed in your browser.
  2. Create a JSP to display the User:
     1. Add a new JSP file named *displayUser.jsp* within the WEB-INF/pages directory.
     2. Create an HTML table to display the First and Last Name of the User Model passed to this View. All Model properties can be displayed by enclosing the Model property in ${}.
     3. Update the *adduser()* method in the Controller to navigate to the *displayUser.jsp*.
     4. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL, submit the User Form, and validate that the correct response is displayed in your browser.
     5. Test and take a screenshot.
  3. Download the JSTL library from <http://central.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar> and copy/paste the JSTL jar to the WEB-INF/lib directory.
  4. Create a JSP to display a list of Users:
     1. Add a new JSP file named *displayUsers.jsp* within the WEB-INF/pages directory.
     2. Update the *adduser()* method in the Controller to navigate to the *displayUsers.jsp* and return a list of User Models to the View instead of a single User Model object.
     3. Create an HTML table to display a List of Users First and Last Name of the List<User> Model passed to this View. All Model properties can be displayed by enclosing the Model property in ${}.
     4. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL, submit the User Form, and validate that the correct response is displayed in your browser.
     5. Test and take a screenshot.

1. Add Form Data Validation:
   1. Go to <http://central.maven.org/maven2/> and drill into org/hibernate/hibernate-validator to download version 5.4.1 of the Hibernate Validator library then copy/paste this into the WEB-INF/lib directory.
   2. Go to <http://central.maven.org/maven2/> and drill into org/jboss/logging/jboss-logging to download version 3.3.1 of the JBoss Logging library then copy/paste this into the WEB-INF/lib directory.
   3. Go to <http://central.maven.org/maven2/> and drill into com/fasterxml/classmate to download version 1.3.3 of the XML library then copy/paste this into the WEB-INF/lib directory.
   4. Go to <http://central.maven.org/maven2/> and drill into javax/validation/validation-api to download the latest version of the JSR 349/380 Bean Validation API library then copy/paste this into the WEB-INF/lib directory.
   5. Add data validation annotations to the User Model:
      1. Add @NotNull and @Size data annotations (set message attribute to appropriate error message) for the firstName and lastName model properties. Set the min and max attributes accordingly.
      2. Add @NotNull, @Min, and @Max data annotations (set message attribute to appropriate error message) for the gender model property. Set the min and max attributes accordingly.
   6. Add data validation check to the Controller:
      1. Update the *adduser()* method in the Controller so the method signature contains the proper data validation parameters:

*addUser(@Valid @ModelAttribute(“user” User user), BindingResult result)*

* + 1. Call the hasErrors() method on the result method parameter and if true then return the user method parameter as the Model, the addUser View, with a ModelAndView object.
  1. Display data validation errors in the Add User Form View:
     1. Within the form:form tags add the form:errors tag with the path attribute set to “\*”.
     2. Within the form:input tags add the form:errors tag with the path attribute set to “firstName” or “lastName” to correspond to the User Model property names.
  2. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL, submit the User Form with invalid data, and validate that the correct response is displayed in your browser
  3. Test and take a screenshot.

1. Add a new property to the User Model:
   1. User Model: add a new *gender* property of type int with getters/setters and update both the default and non default constructors (use the Eclipse IDE to generate the getters/setters).
   2. Controller: fix up compilation errors after the User Model update.
   3. Add User View: in the add form add a radio button control using the form:radiobutton tag to set the gender of the User and in the display views update to display the gender property from the User.
   4. Display List of Users View: Use the JSTL *if* or *choose* tag the display proper gender labels depending on the value of the gender properly.
   5. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL, submit the User Form, and validate that the correct response is displayed in your browser
   6. Test and take a screenshot.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screen shot of the browser display from 2B Step 3h.
2. Screen shot of the browser display from 2B Step 3j.
3. Screen shot of the browser display from 2B Step 4i.
4. Screen shot of the browser display from 2B Step 5f.

## Assignment 2C: Creating Layouts in Spring MVC

**Overview**

In this activity, students will demonstrate the ability to create layouts using Apache Tiles in a Spring MVC application.

**Execution**

Execute this assignment according to the following guidelines:

1. Open up the *assignment2b* project.
2. Add the Apache Tiles Dependent Libraries:
   1. Go to <http://central.maven.org/maven2/> and drill into the following projects and download the newest jar files then copy/paste this into the WEB-INF/lib directory.
      1. org/apache/tiles/tiles-request-servlet
      2. org/apache/tiles/tiles-request-api
      3. org/apache/tiles/tiles-request-jsp
      4. org/apache/tiles/tiles-jsp
      5. org/apache/tiles/tiles-servlet
      6. org/apache/tiles/tiles-core
      7. org/apache/tiles/tiles-api
      8. org/apache/tiles/tiles-autotag-core
      9. org/apache/tiles/tiles-autotag-core-runtime
      10. org/apache/tiles/tiles-autotag-jsp
      11. org/apache/tiles/template
      12. org/slf4j/slf4j-api
      13. commons-digester/commons-digester
      14. commons-beanutils/commons-beanutils
3. Create the Template directories:
   1. Create a *layouts* directory within the WEB-INF directory.
   2. Create a *template* directory within the WEB-INF/layouts directory.
4. Update the Spring Configuration file (provided by the instructor) to configure Tiles, enable Tiles View Resolver, and remove the existing Spring View Resolver.
5. Create the Tiles Definition files (provided by the instructor) in the WEB-INF/layouts directory.
6. Create the Tiles Default Template (full HTML compliant page) along with the Header and Footer (page fragments) provided by the instructor in the WEB-INF/layouts/template directory.
7. Update ALL existing JSP’s:
   1. Remove all content from the JSP’s except what is inside the body tag for each JSP (remember we are now working with page fragments).
8. Test the new Layouts:
   1. Test the Controller and JSP by going to the localhost:8080/assignment2b/user/add URL and validate that the correct response is displayed in your browser.
   2. Test and take a screenshot for the Add User and Display List of Users pages.

**Assignment 2D: Research Activity**

Execute this assignment according to the following guidelines:

1. Identify 3 open source libraries that were used during the completion of Activity 2.
2. In a 200-300 word written report, summarize the following:
   1. What open source license(s) is used?
   2. How and where could a developer download the source code?
   3. How could a developer get involved in contributing?
   4. Who has ownership for any code modifications made?

**Submission**

Submit the following to the learning management system:

**Assignment 2A**

1. Screen shot of the browser display from Part 1 Step 8g.
2. Screen shot of the browser display from Part 2 Step 1e.
3. Screen shot of the browser display from Part 2 Step 2f.
4. Screen shot of the browser display from Part 2 Step 3c.
5. Screen shot of the browser display from Part 2 Step 4d.

**Assignment 2B**

1. Screen shot of the browser display from 2B Step 3h.
2. Screen shot of the browser display from 2B Step 3j.
3. Screen shot of the browser display from 2B Step 4i.
4. Screen shot of the browser display from 2B Step 5f.

**Assignment 2C**

1. Screen shot of the Add User page browser display from 2C Step 8b.
2. Screen shot of the Display Users List page browser display from 2C Step 8b.
3. Open Source Library Research paper.

**Assignment 2D**

1. A 200-300 word written report.

# Activity 3: SpringBeans & DI

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission. **Note:** Various resources are available within the required readings to provide assistance when completing each assignment.

## Assignment 3A: Design and Implement SpringBeans

**Overview**

In this assignment, students will create business services using Spring Beans and inject concrete implementation of business service using Spring IoC.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

**SpringBean Business Services:**

1. Create a Java Interface named *OrdersBusinessInterface* in a new *com.gcu.business* package. Add a public void test() method to the interface.
2. Create a SpringBean by right clicking on the project and selecting the New-> Class menu options. Create the SpringBean in the *com.gcu.business* package with a class name of *OrdersBusinessService* that implements the *OrdersBusinessInterface* interface. Implement the test() method that simply prints “Hello from the OrdersBusinessService” using the System.out.println.
3. Create a SpringBean by right clicking on the project and selecting the New->Class menu options. Create the SpringBean in the *com.gcu.business* package with a class name of *AnotherOrdersBusinessService* that implements the OrdersBusinessInterface interface. Implement the test() method that simply prints “Hello from the AnotherOrdersBusinessService” using the System.out.println.

**SpringBean Configuration:**

1. Update the applicationConfiguration.xml (Spring Configuration) file and add the SpringBean to the application:
   1. Order Business Service Bean:
      1. id=”ordersService”
      2. class=”com.gcu.business.OrdersBusinessService”
   2. User Controller Bean:
      1. id=”userController”
      2. class=”com.gcu.controller.UserController”
      3. property of name ordersService with ref of ordersService
   3. User Controller:
      1. Add a class scoped variable *service* of type *OrdersBusinessInterface*.
      2. Add a public setter method setOrdersService() that takes a single argument of type *OrdersBusinessInterface*.
2. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data. The test string from the *OrdersBusinessService* should be visible on the Console. Take a screenshot.
3. Change the Business Service implementation in the Spring Configuration file:
   1. Update the applicationConfiguration.xml (Spring Configuration) file and change the implementation file for the Orders Service SpringBean:
      1. Order Business Service Bean:
         1. id=”ordersService”
         2. class=”com.gcu.business.AnotherOrdersBusinessService”.
4. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data. The test string from the *AnotherOrdersBusinessService* should be visible on the Console. Take a screenshot.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot of the OrdersBusinessService executing from Controller.
2. Screenshot of the User Controller code with injection code.
3. Screenshot of the SpringBean Configuration file.
4. Screenshot of the AnotherOrdersBusinessService executing from Controller.

## Assignment 3B: SpringBeans Lifecycle and Scope

**Overview**

In this assignment, students will add diagnostic code to a SpringBean to validate its lifecycle and use various SpringBean Scopes.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

**Update the Service SpringBeans:**

1. Add a public void init() method and destroy() method to the *OrdersBusinessInterface*.
2. Implement the init() and destroy() methods in the *OrdersBusinessService* and the *AnotherOrdersBusinessService* classes. Simply put some println statements to print the name of the method and class that is being called.

**Update SpringBean Configuration:**

1. Update the applicationConfiguration.xml (Spring Configuration) file:
   1. Order Business Service Bean:
      1. init-method=”init”
      2. destroy-method=”destroy”
      3. class=”com.gcu.business.OrdersBusinessService”
2. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data multiple times. Note when the Bean’s init() method gets called and how many times init() gets called by looking through the console output. Take a screenshot. In a readme.txt file explain in 2-3 sentences how the code worked (when init() got called) and why the number of calls to init() where made.

**Update the Spring Configuration:**

1. Update the applicationConfiguration.xml (Spring Configuration) file:
   1. Add XSD’s:

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

* 1. Order Business Service Bean:
     1. Scope=”singleton” | “prototype” | “request” | “session”
     2. Within bean tag: <aop:scoped-proxy proxy-target-class="false" />

1. Change the scope attribute in the Orders Service Bean to singleton, then prototype, then request, and then session. For each scope value perform the following:
   1. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data multiple times. Note when the Bean’s init() method gets called and how many times init() gets called by looking through the console output. Take a screenshot. In a readme.txt file explain in 2-3 sentences how the code worked (when init() got called) and why the number of calls to init() where made.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot of lifecycle with README.txt.
2. Screenshots of scopes with README.txt.

## Assignment 3C: SpringBeans Configuration Annotations

**Overview**

In this assignment, students will enable and use auto-wiring as well as use Java Annotations as opposed to XML to configure SpringBeans.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

**Use Auto-Wiring:**

1. Enable auto-wiring by adding the following tag to the Spring Configuration file:
   1. <context:annotation-config/>
2. Remove the current property injector configuration for the User Controller so your Bean configuration is as follows:
   1. <bean id="userController" class="com.gcu.controller.UserController" >
   2. </bean>
3. Add an @Autowired annotation to the User Controller’s setOrdersService() method.
4. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data. Verify that the application functions properly. Take a screenshot.

**Use Java Annotation Configuration (versus XML):**

1. Create a new Java Class named *ApplicationConfiguration* in the *com.gcu* package.
2. Update the *ApplicationConfiguration* class as follows:
   1. Add the *@Configuration* annotation to the class.
   2. Add a new public method *getUserController()* that returns a *UserController*, whose implementation creates and returns an instance of a *UserController*, where the method is marked with a *@Bean(name=”userController”)* annotation.
   3. Add a new public method *getOrdersService()* that returns a *OrdersBusinessInterface*, whose implementation creates and returns an instance of a *OrdersBusinessService*, where the method is marked with a *@Bean(name=”ordersService”)* annotation.
   4. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data. Verify that the application functions properly. Take a screenshot.
   5. Add initMethod=”init” and destroyMethod=”destroy” attributes to the @Bean annotation for the *getOrdersService()* method.
   6. Add @Scope(value=”singleton”, proxyMode=ScopedProxuMode.TARGET\_CLASS) to the getOrdersService() method.
   7. Change the scope value in the @Scope annotation in the *getOrdersService()* method to singleton, then prototype, then request, and then session. For each scope value perform the following:
      1. Deploy the code and execute the application specifying the TestForm.jsp using the internal IDE browser. Submit the form with valid data multiple times. Note when the Bean’s init() method gets called and how many times init() gets called by looking through the console output. Take a screenshot.

**Project Cleanup:**

1. Pick whether you want to use XML or Java Annotations for your future projects. Spring actually allows you to use both XML and Java Annotations if you wish. Refer to the Spring Framework documentation for how to set the proper values in the @Configuration annotation and other annotations you may need to add the Java *ApplicationConfiguration* class.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot of auto-wire.
2. Screenshots of scopes using Java Annotations.

## Assignment 3D: REST Based Services

**Overview**

In this assignment, students will create a REST based service using Spring.and create a REST based service using JAX-RS.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

**REST Service using Spring:**

1. Download the following Jackson library from <http://central.maven.org/maven2> and copy/paste the JSTL jar to the WEB-INF/lib directory:

/com/fasterxml/jackson/core/jackson-databind/2.9.1/jackson-databind-2.9.1.jar

/com/fasterxml/jackson/core/jackson-core/2.9.1/jackson-core-2.9.1.jar

/com/fasterxml/jackson/core/jackson-annotations/2.9.1/jackson-annotations-2.9.1.jar

1. Create a Java class named UserService1 in the com.gcu.services package.
2. Mark the class with @RestController() annotation and @RequestMapping("/service1") annotation.
3. Add a public method getUsers() that returns List<User> and mark the method with the @GetMapping("/users") annotation.
4. In the implementation of getUsers() create a List of Users using the com.gcu.model.User class. Return the list of Users.
5. Test your Service at http://localhost:8080/assignment3d/service1/users. Take a screen shot.

**REST Service using JAX-RS:**

1. Download the following Jersey Container library from <http://central.maven.org/maven2> and copy/paste the Jersey Container jars to the WEB-INF/lib directory:

/org/glassfish/jersey/bundles/jaxrs-ri/2.9.1/jaxrs-ri-2.9.1.jar

/org/glassfish/jersey/bundles/repackaged/jersey-guava/2.9.1/jersey-guava-2.9.1.jar

/org/glassfish/jersey/containers/jersey-container-grizzly2-servlet/2.9.1/jersey-container-grizzly2-servlet-2.9.1.jar

1. Download latest Jersey Bundle from <https://jersey.github.io/download.html> then copy all Jars from api, ext, and lib directories to WEB-INF/lib directory.
2. Configure the Jersey Servlet in the web.xml:

<!-- Jersey Servlet -->

<servlet>

<servlet-name>Jersey Web Services</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>com.gcu.services</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>Jersey Web Services</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

1. Create a Java class named UserService2 in the com.gcu.services package.
2. Mark the class with @Path("service2") annotation.
3. Add a public method getUsers() that returns List<User> and mark the method with the @GET, @Path("/users"), and @Produces("application/json") annotations.
4. In the implementation of getUsers() create a List of Users using the com.gcu.model.User class. Return the list of Users.
5. Test your Service at http://localhost:8080/assignment3d/rest/service2/users using your desktop browser. Take a screen shot.
6. Test your Service at http://localhost:8080/assignment3d/rest/service2/users using Postman. Refer to “Using Postman,” within the Course Materials for additional information. Take a screen shot.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot of Spring based REST Service.
2. Screenshots of JAX-RS based REST Service from your desktop browser.
3. Screenshots of JAX-RS based REST Service from Postman.

## Assignment 3E: Open Source Technology Research

**Overview**

In this assignment, students will research open source technologies.

**Execution**

Execute this assignment according to the following guidelines:

Write a 300-500 word paper that includes the following discoveries from your research:

1. An outline of the history of open source computing and technologies.
2. Identification of two predominant companies in the past or present who base their business model on open source technologies. What are the companies? What is the history of each of these companies? What products are they basing their business model on?
3. Identification of two open source libraries that you did not research in Activity 2, that have been used to complete activities in this course. Summarize the following:
   1. What open source license(s) is used?
   2. How and where could a developer download the source code?
   3. How could a developer get involved in contributing?
   4. Who has ownership for any code modifications made?

**Submission**

Submit the following to the learning management system:

**Assignment 3A**

1. Screenshot of the OrdersBusinessService executing from Controller.
2. Screenshot of the User Controller code with injection code.
3. Screenshot of the SpringBean Configuration file.
4. Screenshot of the AnotherOrdersBusinessService executing from Controller.

**Assignment 3B**

1. Screenshot of lifecycle with README.txt.
2. Screenshots of scopes with README.txt.

**Assignment 3C**

1. Screenshot of auto-wire.
2. Screenshots of scopes using Java Annotations.

**Assignment 3D**

1. Screenshot of Spring based REST Service.
2. Screenshots of JAX-RS based REST Service from your desktop browser.
3. Screenshots of JAX-RS based REST Service from Postman.

**Assignment 3E:**

1. Open Source Research paper.

# Activity 4: SpringJDBC & JavaDB

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission.

## Assignment 4A: Setting Up JavaDB (Derby) Database

**Overview**

In this assignment, students will Setup the IDE to access the JavaDB (Derby) database included with the JDK as well as setup example tables and columns in the JavaDB database.

**Execution**

The following steps can be used to configure the Eclipse IDE to access the JavaDB database:

1. Download Apache Derby for Java 8 from the following link: <http://www.codejava.net/java-se/jdbc/connect-to-apache-derby-java-db-via-jdbc>, follow the download link from Step 1 on this page.
2. Open the Database Development Perspective in the IDE by selecting the Window->Perspective->Open Perspective->Other->Database Development menu options.
3. Add a Database Connection to the perspective by right clicking on the Database Connections tree and selecting the New->Derby menu option, name the connection JavaDB, and click the Next button. From the Derby Connection Profile dialog click the + next to the Drivers dropdown, select the newest Derby Embedded JDBC Driver option, click the JAR List tab, remove the default entry, click the Add JAR/Zip button, and navigate to the *derby.jar* which is located in the Derby install directory under the lib folder. Test the driver by entering a Username of *user* and a password of *derby*, check the Save password checkbox, check the Create database checkbox, and click the Test Connection button. Ensure that you are able to connect successfully to the database. Click the finish button.

NOTE: You have the option to also use MySQL Database. If you chose to use MySQL Database versus the Derby Database you will need to add the MySQL JDBC Driver to your project and select MySQL as the database from step 3 in the above directions. The MySQL JDBC Driver can be downloaded from the MySQL Home Page.

The following steps can be used to create the tables and columns in the database for use in upcoming activities.

1. Open the SQL Editor for the default MyDB database by right clicking on the JavaDB database and selecting the Open SQL Scrapbook menu options. Select the MyDB database from the database dropdown list.
2. In the center pane of the SQL Scrapbook enter the SQL statements to perform the following functions (or use the embedded DDL script in the document provided by the instructor, DO NOT cut and paste the SQL statements from this Word document without cleansing):

-- Create Schema

DROP TABLE testapp.Orders;

DROP SCHEMA testapp RESTRICT;

CREATE SCHEMA testapp;

-- Create Orders Table

CREATE TABLE testapp.Orders

(

ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY(START WITH 1, INCREMENT BY 1),

ORDER\_NO VARCHAR(25) NOT NULL,

PRODUCT\_NAME VARCHAR(128) NOT NULL,

PRICE FLOAT NOT NULL,

QUANTITY INTEGER NOT NULL,

CONSTRAINT primary\_key PRIMARY KEY (ID)

);

-- Insert some Test Orders

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000000’, ‘This is Product 1’, 1.00, 1);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000001’, ‘This is Product 2’, 2.00, 2);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000002’, ‘This is Product 3’, 3.00, 3);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000003’, ‘This is Product 4’, 4.00, 4);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000004’, ‘This is Product 5’, 5.00, 5);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000005’, ‘This is Product 6’, 6.00, 6);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000006’, ‘This is Product 7’, 7.00, 7);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000007’, ‘This is Product 8’, 8.00, 8);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000008’, ‘This is Product 9’, 9.00, 9);

INSERT INTO testapp.Orders(ORDER\_NO, PRODUCT\_NAME, PRICE, QUANTITY) VALUES(‘0000000009’, ‘This is Product 10’, 10.00, 10);

1. Save the above SQL script in the project in a folder called *scripts*.
2. Right click on the Scrapbook and select the Execute All menu options. Make sure the schema, table, and data are all created without error.
3. Expand the Schemas. Expand the TESTAPP schema. Expand the Tables. Right click on the ORDERS table and select the Data->Sample Contents menu options. Verify test data exists.

Execute a SQL statement in the Scrapbook to select all rows from the Orders table. Take a screenshot of SQL results. The Orders table should have 10 rows of test data.

**Submission**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot from the output of the select statement.

## Assignment 4B: Using JDBC to Query Data from the JavaDB (Derby) Database

**Overview**

In this assignment, students will use JDBC to query data from a relational database and insert data to a relational database.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

1. Add derby.jar in the Derby home directory which is located in the Derby install directory under the lib folder to WEB-INF/lib in your project.
2. Create an Orders DAO:
   1. Create a new Model class named *Order* in the *model* package. The model should have 3 properties: orderNo of type String, productName of type String, price of type float, and quantity of type int. Add getter/setter for all properties
   2. Create a Java Interface named *DataAccessInterface* in the *data* package. Add the following methods to the interface class and use generics in the class design:

public interface DataAccessInterface <T>

{

public List<T> findAll();

public T findById(int id);

public boolean create(T t);

public boolean update(T t);

public boolean delete(T t);

}

* 1. Create a Java class named *OrdersDataService* in the *data* package and that implements the *DataAccessInterface*.
  2. Add the *OrdersDataService* SpringBean configuration to the Spring Configuration File.

1. Integrate the Orders DAO into the Orders Business Service:
   1. Add a new public method *getAllOrders()* that returns a List<Order> to the *OrdersBusinessService* and the *OrdersBusinessInterface*.
   2. Inject the *OrdersDataService* into the *OrdersBusinessService* by using an Autowire annotation using either setter injector or property injector. Call the *OrdersDataService findAll()* method from the *OrdersBusinessService getAllOrders()*.
   3. In the User Controller change the *test*() call to call the *getAllOrders*().
   4. Implement the *getAllOrders()* in the *OrdersDataService*:
      1. Get a connection to the database using the JDBC URL for Mac or Windows (see examples below) using a username of *user* and password of *derby*.
      2. NOTE: the URL in the above connection string needs to be set to:
         1. Mac: /Users/[your login name]/MyDB
         2. Windows: C:\\Users\\[your login name]\\MyDB
      3. All imports for the following code should be from the java.sql package!
      4. Load the JDBC Driver by calling:

Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();

* + 1. Get a database connection by calling:

Connection conn = DriverManager.getConnection(url, username, password).

* + 1. Print a message to the console "Success!!" after the getConnection() to get a connection to the Database.
    2. Surround the getConnection() call with a try catch exception block.
    3. In the catch block of the exception print a message to the console "Failure!!".
    4. Close the connection to the database by calling the close() on the Connection returned.
  1. Run the application and validate that a connection can be made to the database.
  2. Take a screen shot of the console output.

1. Run a query from the database:
   1. Refacttor the *getAllOrders()* in the *OrdersDataService*:
      1. Make Connection a local variable and initialize to null. Move the Connection close to a finally block (and check for null connection).
      2. Create a Statement from the Connection using createStatemen().
      3. Create a local String variable and initialize this to the following SQL Statement: SELECT \* FROM tsetapp.ORDERS.
      4. Execute the SQL by calling the executeQuery() method on the Statement and save the ResultSet into a variable rs.
      5. Loop thru the ResultSet using the next() and read the ID column using getInt(), read the PRODUCT\_NAME column using getString(), and read the PRICE column using the getFloat(). Print all values to the console for each Order.
      6. Call the close() on the ResultSet when done with loop.
   2. Create a new View named *displayOrders.jsp*, pass the Orders List as the Model to the View, and implement the View using JSTL and a HTML table.
   3. Run the application.
   4. Take a screen shot of the Orders List screen.

Add and update the following components in the project:

1. Run an insert and update statement:
   1. Update the Orders DAO *create()*.
      1. Load the database driver.
      2. Get a connection to the database.
      3. Create a Statement from the Connection using createStatemen().
      4. Create a local String variable and initialize this to an INSERT SQL Statement to insert an Order into the testapp.ORDERS table.
      5. Execute the SQL Insert by calling the executeUpdate() method on the Statement.
2. Run the application.
3. Take a screen shot of the Orders List screen.

**Submission**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Screenshot from Step 3f.
2. Screenshot from Step 4d.

## Assignment 4C: Using Spring JDBC to Query Data from the JavaDB (Derby) Database

**Overview**

In this assignment, students will use Spring JDBC to query data from a relational database and insert data to a relational database.

**Execution**

Execute this assignment according to the following guidelines. Add and update the following components:

1. Add Data Source Spring Bean, Transaction manager Spring Bean, and Data Source injection into DAO into the Spring Configuration File.
2. Add the Spring JDBC libraries spring-jdbc-4.3.9.RELEASE.jar, spring-tx-4.3.9.RELEASE.jar, and spring-aspects-4.3.9.RELEASE.jar to WEB-INF/lib
3. Go to <http://central.maven.org/maven2/> and download:
   1. spring-dao-2.0.8.jar from org/springframework/spring-dao
   2. commons-dbcp2-2.1.1.jar from org/apache/commons/commons-dbcp2
   3. commons-pool2-2.4.2.jar from org/apache/commons/commons-pool2
   4. aspectjweaver-1.8.9.jar from org/aspectj/aspectjweaver
4. Refactor Orders DAO:
   1. Add Setter Injector for DataSource and create a Spring JDBC template.
   2. Replace JDBC code with Spring JDBC Template code for the findAll() and create().
   3. Run the application.
   4. Take a screen shot of the Orders List screen.

**Submission**

Submit the following to the learning management system:

**Assignment 4A:**

1. Screenshot from the output of the select statement.

**Assignment 4B:**

1. Screenshot from Step 3F.
2. Screenshot from Step 4D.

**Assignment 4C:**

1. Screenshot from Step 4D.

# Activity 5: Maven

**Note:** Various resources are available within the required readings to provide assistance when completing this assignment.

## Assignment 5A: Convert a Project to Maven

**Overview**

In this assignment, students will convert a project to Maven.

**Execution**

Execute this assignment according to the following guidelines:

1. Make a copy of your project.
2. Remove all JAR files from WEB-INF/lib directory.
3. Convert to Maven Project by right clicking on Project and select the Configure->Convert to Maven Project menu options and accepting all default options.
4. In the web.xml change the version from 2.4 to 3.1 (and also change the XSD reference from j2ee to javaee). See example below. Ensure there are no errors in your project.

<web-app xmlns="<http://xmlns.jcp.org/xml/ns/javaee>"

         xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

         xsi:schemaLocation="<http://xmlns.jcp.org/xml/ns/javaee>

<http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd>"

         version="3.1">

1. Update Maven pom.xml file (located in root folder of the project) with all proper dependencies and build configuration. Right click on project and select the Maven->Update Project menu options. See example below. Ensure there are not build errors.
2. Deploy and test all of the functionality that has been built from assignments 2A through 4B.
3. Take a screen of your application that displays the list of Orders and from the JSON output from Service 1.
4. Build the application in Eclipse using your Maven POM file.
5. Take a screen shot of your expanded *target* directory in your project workspace.

**Submission**

Submit the following to the learning management system:

**Assignment 4B:**

1. Screenshot from Step 7.
2. Screenshot from Step 9.

**Assignment 5A:**

1. A screen shot of your expanded target directory in your project workspace

# Activity 6: jQuery

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission. **Note:** Various resources are available within the required readings to provide assistance when completing each assignment.

Assignment 6A: Working with jQuery

**Overview**

In this assignment, students will Code pages using jQuery selectors, CSS. DOM, and AJAX.

**Execution**

Execute this assignment according to the following guidelines:

1. jQuery Activity 1 - Load jQuery library and execute JavaScript functions using jQuery:
   1. Create a standard HTML document with a head and body.
   2. In the header set the title and script to import jquery library:

<title>CST-341 jQuery In-Class Activity 1</title>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>

* 1. Add a function in JavaScript named sayHello() that uses document.write() to write a welcome string to the DOM.
  2. Add a function in JavaScript named sayName() that uses document.write() to write your name to the DOM. Save the function into a variable named sayName.
  3. Use jQuery to execute the 2 JavaScript functions using $(document).ready():

// Use jQuery document.ready() and pass it the sayHello() method

$(document).ready(sayHello);

// Use jQuery document.ready() and pass it an anonymous method

$(document).ready(function(){document.writeln("My first jQuery code.<br/>");});

// Use jQuery document.ready() and pass it the sayName() method

$(document).ready(sayName);

* 1. Open the HTML page and take a screenshot of the rendered page.

1. jQuery Activity 2 - Create an Object in JavaScript and use jQuery to write into the DOM:
   1. Create a standard HTML document with a head and body.
   2. In the header set the title and script to import jquery library:

<title>CST-341 jQuery In-Class Activity 2</title>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>

* 1. Declare a JavaScript Object named User that contains a firstName, lastName, and age property.
  2. Add a function in JavaScript named displayUser() and use jQuery to access the DOM and write HTML into the DOM (write the User firstName, lastName, and age properties):

$("#fn").html(User.firstName);

$("#users").find("#ln").html(User.lastName);

$("#users").find("#age").html(User.age);

* 1. Use jQuery to execute the JavaScript functions using $(document).ready():

$(document).ready(displayUser);

* 1. Create a standard HTML Table with the following specifications:

Table: Width of 50% and Border of 1 with an id of users.

Header Row: Labels of First Name, Last Name, and Age each with a width of 33%. Enclose the Header Row with <thead> tags.

User Row: column 1 with an id="fn", column 1 with an id="ln", column 1 with an id="age" all with content of ?, and row aligned with center. Enclose the Table Row with an opening and closing <tbody> tags.

* 1. Open the HTML page and take a screenshot of the rendered page.

1. jQuery Activity 3 - Use jQuery to dynamically build a table into the DOM:
   1. Copy the HTML file from jQuery Activity 2.
   2. Remove the User Row from the Users Table but leave the <tbody> tags.
   3. Update the User Object to be a function with a non-default constructor that initializes the firstName, lastName, and age property.
   4. Create an empty array declaration called users.
   5. Initialize the users array by instantiating the Users Object with desired data.
   6. Change the name of displayUser() to displayUsers() (and also update the any uses of this method).
   7. In displayUsers() dynamically build a table in JavaScript, iterate over the user array, and for each user do the following:

var x = 0;

var y = 0;

var table = document.getElementById("users");

for (var user of users)

{

var row = table.insertRow(-1);

row.setAttribute("align", "center");

var cell1 = row.insertCell();

var cell2 = row.insertCell();

var cell3 = row.insertCell();

row.id = x;

cell1.id = x + "\_" + y++;

cell2.id = x + "\_" + y++;

cell3.id = x + "\_" + y++;

++x;

y = 0;

}

* 1. In displayUsers() after building the a table in JavaScript, iterate over the user array, and for each user do the following:

x = 0;

y = 0;

for (var user of users)

{

$("#" + x + "\_" + y++).html(user.firstName);

$("#" + x + "\_" + y++).html(user.lastName);

$("#" + x + "\_" + y++).html(user.age);

++x;

y = 0;

}

* 1. Open the HTML page and take a screenshot of the rendered page.

1. jQuery Activity 4 - Use jQuery to apple CSS to table in the DOM:
   1. Copy the HTML file from jQuery Activity 3.
   2. Style the header row of the table using CSS: style="background-color:#A0A0A0".
   3. Style the odd and even rows of the user data rows using CSS: row.style.backgroundColor = x & 1 ? "#FFFFFF" : "#F0F0F0";
   4. Comment out of step c.
   5. Style the odd and even rows of the user data rows using jQuery: $("#" + x).css("backgroundColor", x & 1 ? "#FFFFFF" : "#F0F0F0");
   6. Create a new function in JavaScript named clicked() that simply displays a message using alert().
   7. Setup a mouse click event on the first User using JQuery by calling $("#0\_0").bind('click', clicked);
   8. Open the HTML page and take a screenshot of the rendered page.
2. jQuery Activity 5 - Use jQuery to make an AJAX call to a REST Service and render its JSON in a table:
   1. Add a new REST API to the UserService1 class.
   2. Define a SpringBean for the UserService1 class in the Spring Configuration file.
   3. Inject and Autowire the OrdersBusinessService into the UserService1 class
   4. Define new REST API: GET request at /orders that returns a List of Orders using the OrdersBusinessService.getAllOrders().
   5. Creat a new JSP displayOrders2.jsp in the pages folder. Add this View to the views.xml.
   6. In the UserController change the return view in /adduser to displayOrders2.
   7. Add a reference to the jQuery script to the defaultTemplate.jsp.
   8. Create a standard HTML Table with the following specifications:

Table: Width of 50% and Border of 1 with an id of orders.

Header Row: Labels of Order Number, Product Name, Price, and quantity each with a width of 25%. Enclose the Header Row with <thead> tags.

For the Table Row simply put an opening and closing <tbody> tag.

* 1. Create a script tag: <script></script>.
  2. Create a JavaScript function displayUsers():

var x = 0;

var y = 0;

var table = document.getElementById("orders");

for (var order of orders)

{

var row = table.insertRow(-1);

row.setAttribute("align", "center");

var cell1 = row.insertCell();

var cell2 = row.insertCell();

var cell3 = row.insertCell();

var cell4 = row.insertCell();

row.id = x;

cell1.id = x + "\_" + y++;

cell2.id = x + "\_" + y++;

cell3.id = x + "\_" + y++;

cell4.id = x + "\_" + y++;

cell1.innerHTML = order.orderNo;

cell2.innerHTML = order.productName;

cell3.innerHTML = order.price;

cell4.innerHTML = order.quantity;

row.style.backgroundColor = x & 1 ? "#FFFFFF" : "#F0F0F0";

++x;

y = 0;

}

* 1. Create a JavaScript function getOrders():

$.ajax(

{

type: "GET",

url: "/assignment6/service1/orders",

dataType: "json",

success: function(data)

{

displayOrders(data);

},

error: function (xhr, ajaxOptions, thrownError)

{

alert(xhr.status);

alert(thrownError);

}

}

* 1. Call the getOrders function when the page is ready using $(document).ready(getOrders).
  2. Run the application and take a screenshot of the rendered Orders page.

**Submission**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

**Assignment 6A:**

1. Screenshot from jQuery Activity 1.
2. Screenshot from jQuery Activity 2.
3. Screenshot from jQuery Activity 3.
4. Screenshot from jQuery Activity 4.
5. Screenshot from jQuery Activity 5.

Assignment 6B: Working with jQuery Data Table

**Overview**

In this assignment, students will Code page using jQuery Data Table and AJAX.

**Execution**

Execute this assignment according to the following guidelines:

1. Open the Eclipse project from the Activity 5D (or copy to a new project named assignment6).
2. Prepare a View Page:
   1. Copy displayUsers2.jsp to displayUsers3.jsp. Refresh your project. Add this View to the views.xml.
   2. In the UserController change the return view in /adduser to displayOrders3.
3. Add support for jQuery Table Data:
   1. Add the following DataTable CSS and JS support files to the defaultTemplate.jsp.

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css">

<script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>

1. Run project to ensure the /add URI and application still functions properly. Also validate all CSS and JS files were loaded by using the Network feature of your browsers Developer Tools.
2. Create a DataTable in displayUsers3.jsp:
   1. Add a class="display" to the table tag.
   2. Remove the JavaScript function displayOrders() from the code.
   3. In the success callback handler in .ajax replace the call to displayOrders() with the following code:

// Display Orders in the jQuery Data Table

$('#orders').dataTable({

"data": data,

"columns": [{ "data": "orderNo" },{ "data": "productName" },{ "data": "price" },{ "data": "quantity" }]

} );

1. Run the application and take a screenshot of the rendered Orders page.

**Submission**

Submit the following to the learning management system:

**Assignment 6A:**

1. Screenshot from jQuery Activity 1.
2. Screenshot from jQuery Activity 2.
3. Screenshot from jQuery Activity 3.
4. Screenshot from jQuery Activity 4.
5. Screenshot from jQuery Activity 5.

**Assignment 6B:**

1. Screenshot of the rendered Orders page.

# Activity 7: Bootstrap

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission. **Note:** Various resources are available within the required readings to provide assistance when completing each assignment.

Assignment 7A: Learning Bootstrap

**Overview**

In this assignment, students will learn the basics of the Bootstrap library by completing a number of online tutorials.

**Execution**

Execute this assignment according to the following guidelines:

1. Go to the online tutorial at <https://www.tutorialrepublic.com/>
2. Click on Bootstrap Tutorial link.
3. Complete the following tutorials (make elaborate use of the Try This Code feature):
   1. Bootstrap Introduction
   2. Bootstrap Get Started
   3. Bootstrap Grid System
   4. Bootstrap Typography
   5. Bootstrap Tables
   6. Bootstrap Forms
   7. Bootstrap Input Groups
   8. Bootstrap Buttons
   9. Bootstrap Button Groups
   10. Bootstrap Images
   11. Bootstrap Navs
   12. Bootstrap Navbar
4. Build a responsive HTML Demo Page that demonstrates the use of the Grid System, Table, Form, Images, and NavBar.
5. Run the Demo Page with the Responsive Browser Developer Tools enabled in Chrome or Safari. Take screenshots of the rendered Demo Page for an iPhone SE/5, iPhone 6/7, iPad Pro, and Desktop ensuring that for all devices there is NO horizontal scrolling.

**Submission**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Bootstrap demonstration HTML Page.
2. Screenshots of HTML Page in mobile, tablet, and desktop sizes.

Assignment 7B: Making ICA Application Responsive

**Overview**

In this assignment, students will add Bootstrap to an existing ICA application, make a Form responsive, make a Data Table responsive, and add a responsive NavBar to the application.

**Execution**

Execute this assignment according to the following guidelines:

1. Open the Eclipse project from Activity 6B (or copy the project to a project named assignment7).
2. Make the application responsive by adding support for Bootstrap:
   1. Add the following Bootstrap and Responsive Data Table support files to the defaultTemplate.jsp.

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdn.datatables.net/responsive/2.2.0/css/dataTables.responsive.css”>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css">

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">

<script src="//cdn.datatables.net/responsive/2.2.0/js/dataTables.responsive.js"></script>

1. Run project to ensure the /add URI and application still functions properly. Also validate all CSS and JS files were loaded by using the Network feature of your browsers Developer Tools.
2. Update the application (referencing the Bootstrap Tutorials and Demo Page from ICA Activity 7A) and make the following updates to the application:
   1. Add a NavBar to the header. Reference the NavBar tutorial at Tutorial Republic.
   2. Make the Add User Form responsive (remove table for layout). Reference the Forms tutorial at Tutorial Republic.
   3. Add the responsive attribute to the jQuery Data Table configuration:

$('#orders').dataTable({

"responsive" : true,

* 1. Create a new HTML table below the jQuery Orders Table and make it responsive by adding the class=”table table-responsive” as a table tag attribute. Create the table with the following 4 rows. Reference the Table tutorial at Tutorial Republic.
     1. Column 1: header of Row
     2. Column 2: header of First Name
     3. Column 3: header of Last Name
     4. Column 4: header of Information
     5. Populate 4 rows in the table and make sure the Information column is between 100 and 150 characters.

1. Run the application with the Responsive Browser Developer Tools enabled in Chrome or Safari. Take screenshots of the rendered Add Form page and the Orders page for an iPhone SE/5, iPhone 6/7, iPad Pro and Desktop ensuring that for all devices there is NO horizontal scrolling.
2. Set the jQuery Data Table Responsive configuration to false and remove the class=”table table-responsive” attribute from the Users table.
3. Run the application with the Responsive Browser Developer Tools enabled in Chrome or Safari. Take screenshots of the rendered Add Form page and the Orders page for an iPhone SE/5, iPhone 6/7, iPad Pro and Desktop and note which devices where there IS horizontal scrolling.
4. Undo the changes made in step 6.

**Submission**

Submit the following to the learning management system:

**Assignment 7A:**

1. Bootstrap demonstration HTML Page.
2. Screenshots of HTML Page in mobile, tablet, and desktop sizes.

**Assignment 7B:**

1. Screenshots for responsive Add and Orders pages for the iPhone SE/5, iPhone 6/7, iPad Pro and Desktop.
2. Screenshots for non-responsive Add and Orders pages for the iPhone SE/5, iPhone 6/7, iPad Pro and Desktop.