**Capstone - Exercise 8: Case Study Front End**

In this exercise, you build the Case Study Web front end and connect it through to the database.

**Step-1:**

Open the Cloud Platform Console at [https://console.cloud.google.com](https://console.cloud.google.com/).

Click on the three horizontal bars at the left most side of the blue bar near the top of the browser window. *Select Compute Engine*.

Select *VM Instances*. You should see the virtual machine you created earlier.

Click on the checkbox to the left of the VM name and then select *START*. It will take a few moments to start.

Click on *SSH* to start a terminal window.

**Change the host name to student:** Find the icon that looks like a gear in the upper right-hand corner of this terminal browser window and select *Change Linux User Name*. Enter *student* and *click Change*. Now, notice the prompt that says "student@lab:~$"



Start the MySQL server.

*cd*

*cd devops-lesson-4/lab-4.1*

**Step 2**

Start the vncserver.

*vncserver*

Connect the VNC client on your local machine to the server on port 5901.  
Start Eclipse and open up the DevOps project.

**Step 3**

We are now going to walk through the code.  
For each file described, open it up in Eclipse and read through the code. The code we have seen before won't be described again here.

The file *devops.jetty.WebServer* implements a Web server using the Jetty API. Most Web servers are stand alone. Jetty is designed to be embedded in applications. The code listens on port 8080, enables the processing of JSPs, and tells it the Web content is in the webapps directory.

The file *webapp/WEB-INF/web.xml* is the entry point of the application. It loads the Spring framework which is responsible for creating objects and wiring them together. The most important thing is it loads *web-context.xml*. It also loads *application-context.xml* which we will look at later.

The file *web-context.xml* tells Spring that there are annotated POJOs in the *devops.mvc* package. It also tells Spring that the views are JSP files in the WEB-INF/jsp directory.

The file *devops.mvc.MonitoringController* is found by Spring because of the *@Controller* annotation. Spring will automatically instantiate this class. The *@Autowired* annotation on the *setDataManager* method will cause Spring to inject a data manager automatically. The home method responds to the URL *index.html* and returns the view defined in index.jsp. The monitoring method responds to the URL *monitoring.html*. It uses the data manager to get the title and a list of Docker objects. These are passed via the model map into the JSP monitoring.jsp.

The file monitoring.jsp takes the model data and converts it into HTML. Anything in ${} extracts data from the model map. The ${} item in the table call get methods on a Docker POJO.

The file *devops.busines.DataManager* is an interface connecting the presentation and business tiers. It defines the methods which the controller can use.

The file *devops.business.DataManagerImpl* implements the interface. The DAO is injected into the setter method by Spring.

Finally, the file *application-context.xml* tells Spring to instantiate a data manager and a DAO. It tells Spring to inject the DAO into the data manager. The *@Autowired* annotation you saw in the controller causes implicit injection of the data manager created here into the controller.

**Step 3**

Run the application.  
Run *devops.jetty.WebServer.java* as a Java application.

On your local machine, connect a Web browser to port 8080 on the external IP address of your virtual machine.

You should see the opening Web page.  
Click on the link and you should see the Docker events from the database.

If you have time, run the back end again and add more events.  
Refresh the page by clicking on the link and see the new data.

You will need to stop the lab computer at the end of each day to prevent it from accumulating costs during the evening and night.

From the Web UI, you can navigate to the Compute Engine section and select your lab computer. When it is selected, click on the icon representing the "Stop" operation as shown below:

