**MIND STORM SOFTWARE PVT LTD **  
**Google App Engine Training**

# ExamResults – Step 2 – Build the Web Interface

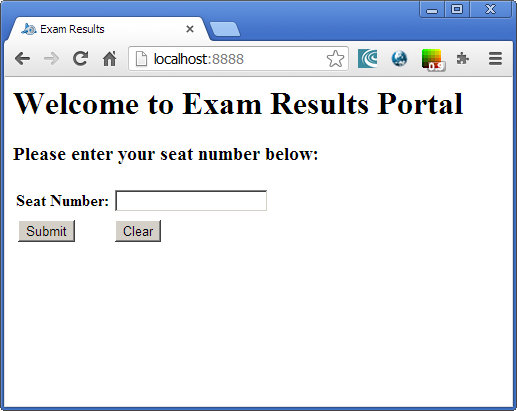
In this session, we are going to build the Web Interface for the ExamResults application. This will mean building out the JSP, Servlets and enabling them in the App Engine application to ensure that the Web Interface flow works out fine.

We will be using dummy data for the results, since we will be building out the database layer in the next session.

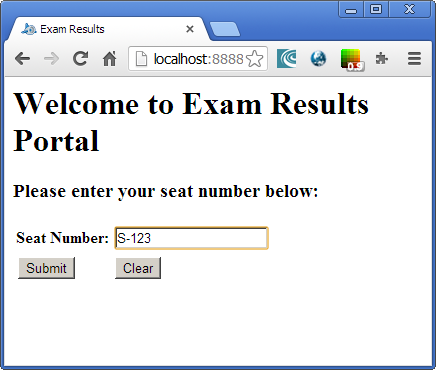
# ExamResults – Step 2 – See it in Action

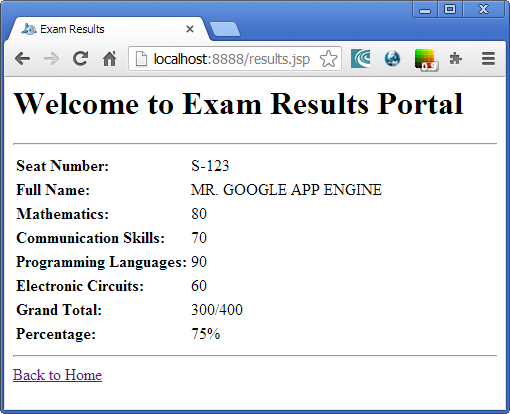
Before we go about the task, let us take a look at the final UI. The focus is going to be on a basic web UI rather than a beautiful UI.

What we want to achieve at the end of this session, is that when we visit the <http://localhost:8888> URL, it shows us the home page as shown below:



We can enter a **seat number** and click on Submit, it will provide us a dummy exam result as shown below:





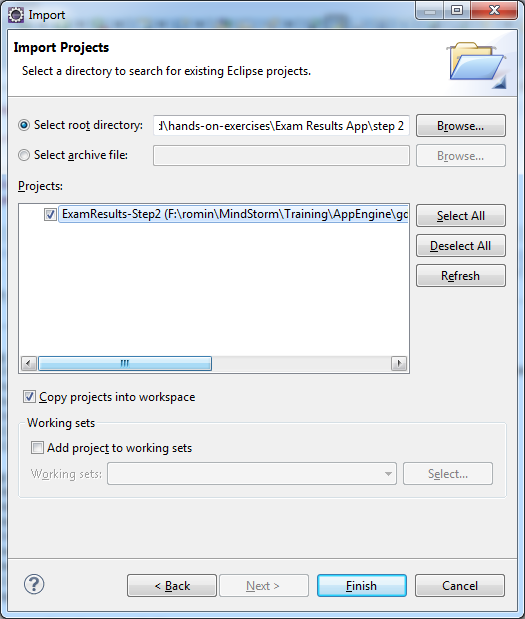
If we do not enter a seat number, it will give us an error page.

# Import the Project

In Eclipse, import the project directly from **hands-on-exercises/Exam Results App/ExamResults-Step 2**

The Steps to import the project are:

1. Click on **File --> Import**
2. Select **General --> Existing Projects into Workspace.** Click on **Next.**
3. In the **Import Projects** dialog, Browse to the folder **hands-on-exercises/Exam Results App/ExamResults-Step 2** and tab out
4. It should show a dialog as given below. Select the Exam Results-Step2 project and ensure that Copy projects into workspace is checked as given below:

****

1. Click on Finish.
2. **Resolve AppEngine SDK:** Select your App Engine SDK. For that, select the Project, Right-click and go to Properties. Then visit Java Build Path and in the Order and Export tab, select the App Engine SDK that you have installed and click on OK.

# Add an ExamResults entity class

The first thing we have done is add a new entity class that can hold the Exam Results. This will be the class that will be eventually populated from the datastore given a Seat Number. For now, we are going to define a simple Entity class named **ExamResult**, which will have the following fields:

The source code for the ExamResult entity is shown below:

**package** com.mindstormsoftware.examresults.entity;

**import** java.io.Serializable;

**public** **class** ExamResult **implements** Serializable {

String seatNumber;

String studentName;

String marks\_Math;

String marks\_CommSkills;

String marks\_Programming;

String marks\_ElectronicCircuits;

String marks\_Total;

String marks\_Percentage;

//Constructors

//Getter & Setter methods

}

We are going with a basic Entity which has the Seat Number, Student Name, marks in 4 subjects , Total marks and Percentage field.

This is a simplistic scenario and we wish to keep it that way. If you are building a generic Exam Results app for different streams, subjects – the entity could get quite complex but that is an exercise for you ☺

This Entity is present in **src/com/mindstormsoftware/examresults/entity** folder.

# (ExamResultsServlet.java)

This servlet will be invoked when the Seat Number is entered and the user clicks on the Submit button (this page is the index.html and we will see that in the next section)

The servlet code will be present in **src/com/mindstormsoftware/examresults** folder and its mappings have been created in the **web.xml** file.

The source code is shown below:

**package** com.mindstormsoftware.examresults;

**import** java.io.IOException;

**import** java.util.logging.Level;

**import** java.util.logging.Logger;

**import** javax.servlet.http.\*;

**import** com.mindstormsoftware.examresults.entity.ExamResult;

@SuppressWarnings("serial")

**public** **class** ExamResultsServlet **extends** HttpServlet {

**private** **static** **final** Logger *\_logger* = Logger.*getLogger*(ExamResultsServlet.**class**.getName());

**public** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** IOException {

//Extract out the input parameters

String seatNumber = req.getParameter("seatnumber");

*\_logger*.info("Received a request for seat number = " + seatNumber);

**try** {

//Check if a Seat Number that is provided is not null or empty

**if** (seatNumber == **null**) **throw** **new** Exception("Seat Number needs to be provided.");

**if** (!seatNumber.isEmpty()) {

//Retrieve the results - currently this will be dummy

//Eventually we will get this from the Datastore

ExamResult dummyResult = getDummyResult(seatNumber);

req.getSession().setAttribute("result", dummyResult);

resp.sendRedirect("results.jsp");

}

**else** {

**throw** **new** Exception("Seat Number needs to be provided.");

}

}

**catch** (Exception ex) {

String logMsg = "Exception in processing request : " + ex.getMessage();

*\_logger*.log(Level.*INFO*,logMsg);

**throw** **new** IOException(logMsg);

}

}

**private** ExamResult getDummyResult(String seatNumber) {

ExamResult ER = **new** ExamResult();

ER.setSeatNumber(seatNumber);

ER.setStudentName("MR. GOOGLE APP ENGINE");

ER.setMarks\_Math("80");

ER.setMarks\_CommSkills("70");

ER.setMarks\_ElectronicCircuits("60");

ER.setMarks\_Programming("90");

ER.setMarks\_Percentage("75");

ER.setMarks\_Total("300");

**return** ER;

}

}

Few things to observe about the source code:

1. We are providing a doPost() method that will be invoked by the index.html page.
2. The doPost() method extracts out the **seatnumber** form field that is provided by the user.
3. If the **seatnumber** request variable is not present or empty, an exception is thrown. All exceptions will get forwarded to the **error.jsp** page that we shall configure in the **web.xml** later on.
4. If the **seatnumber** request is present, it invokes a small utility method named **getDummyResult** that simply returns a hardcoded result for now.
5. On successful flow, the resulting **ExamResult** instance i.e. **dummyResult** in put in the HTTP Session and the flow is forwarded to the **results.jsp** page that will show the data.

# Basic Web pages and flow

We will need 3 pages to be developed:

1. **index.html**: This is the first page that is shown. It is simple form with one input text field named **seatnumber** and a submit button. The submit button results in the form being sent to the **/examresults** endpoint, which is our servlet above.
2. **results.jsp**: This page is shown when the servlet forwards the request to it. All that this form does is extract out the **dummyResult** instance from the HTTP Session and show the different fields for the **ExamResult** entity.
3. **error.jsp**: This is a standard JSP Error page that shows the exception object. For e.g. if the seat number is not provided.

All the source code for the files is present in **hands-on-exercises/Exam Results App/step 2/ExamResults** project and inside the **war** folder.

# App Engine configuration files

We need to configure the web.xml and the appengine-web.xml file.

## web.xml

We shall use the Servlet 2.5 specifications that allow the use of **<error-page>** elements to define which JSP error page should the control go to in case there is an exception thrown.

So we have the following entry to the **web.xml** file after the **<welcome-file-list>** element:

<error-page>

<description>Uncaught exception</description>

<error-code>500</error-code>

<location>/error.jsp</location>

</error-page>

## appengine-web.xml

Since we are using session variables, we will need to enable session management in the App Engine. By default App Engine does not make it enabled. So in the **WEB-INF/appengine-web.xml** file, we have added the following line:

<sessions-enabled>true</sessions-enabled>

# Test locally and Deploy

Once you made the changes, test out the changes locally first. Stop the Web Application if it is already running by clicking on the Stop icon in the Console window. Restart the application i.e. Right-click on Project and select Run As 🡪 Web Application and navigate to <http://localhost:8888>

If you wish to deploy your application to the cloud. This should be straightforward now since we had already set the application ID and deployed it once in the previous session. To deploy to the cloud, right-click on project and then **Google 🡪 Deploy to App Engine**

# Next Session

In the next session, we shall see how to add Datastore support to the application.