



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.

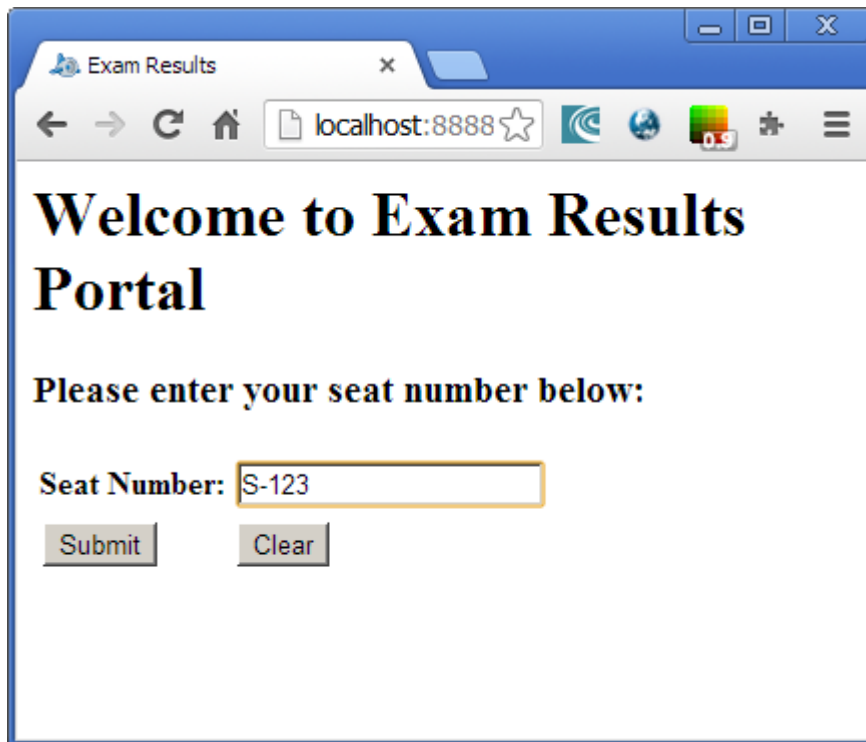
ExamResults – Step 2 – See it in Action

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.

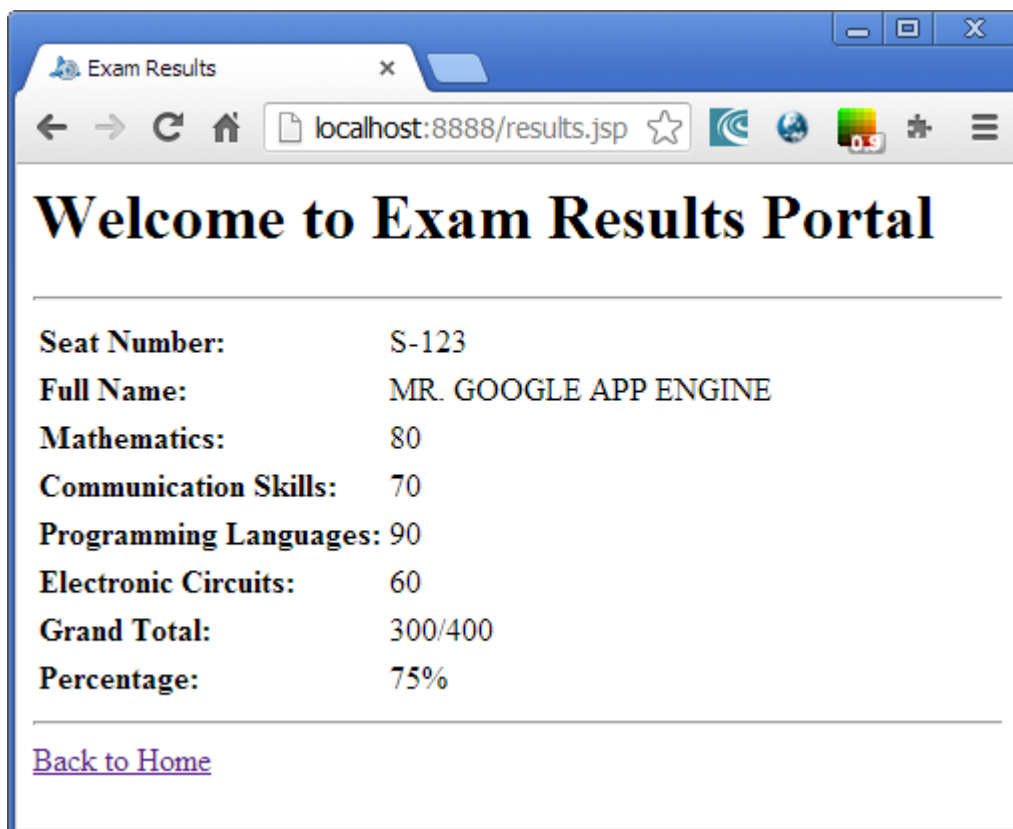
What we want to achieve at the end of this session, is that when we visit the <http://localhost:8888> URL, it shows us the home page as shown below:

The screenshot shows a web browser window with the title 'Exam Results'. The address bar displays 'localhost:8888'. The main content area features the heading 'Welcome to Exam Results Portal' in a large, bold, black serif font. Below this, the text 'Please enter your seat number below:' is displayed in a smaller, bold, black sans-serif font. Underneath, there is a label 'Seat Number:' followed by a text input field. At the bottom of the form, there are two buttons: 'Submit' and 'Clear', both with a light gray background and black text.

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



A screenshot of a web browser window titled "Exam Results". The address bar shows "localhost:8888". The page has a large heading "Welcome to Exam Results Portal". Below the heading, it says "Please enter your seat number below:". There is a text input field labeled "Seat Number:" containing the text "S-123". Below the input field are two buttons: "Submit" and "Clear".

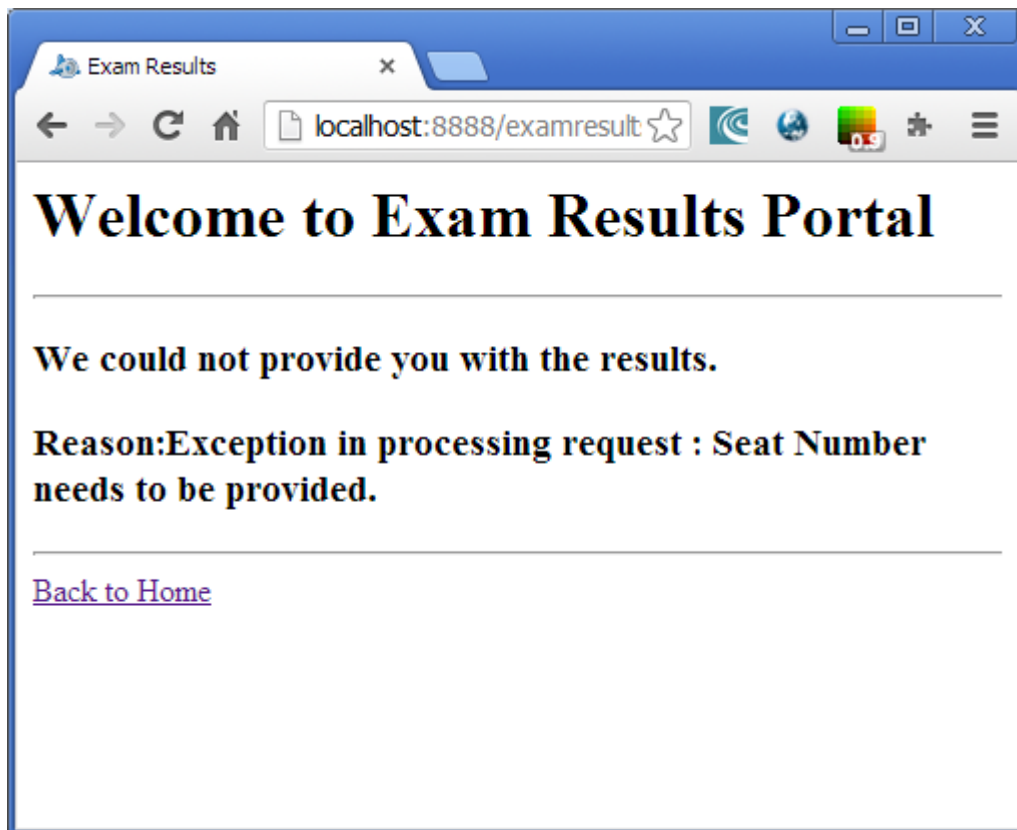


A screenshot of a web browser window titled "Exam Results". The address bar shows "localhost:8888/results.jsp". The page has a large heading "Welcome to Exam Results Portal". Below the heading, there is a table of results:

Seat Number:	S-123
Full Name:	MR. GOOGLE APP ENGINE
Mathematics:	80
Communication Skills:	70
Programming Languages:	90
Electronic Circuits:	60
Grand Total:	300/400
Percentage:	75%

Below the table, there is a link [Back to Home](#).

If we do not enter a seat number, it will give us an error page as shown below:



Import the Project

We will import the Eclipse project at this stage so that you have all the working source code for this step.

1. In Eclipse, go to **File → Import**. This will bring up the Import dialog.
2. Select **General → Existing Projects into Workspace** and click on **Next**.
3. For Select root directory, click on browse and go to **c:\appengine-training\hands-on-exercises\Exam Results App\step 2**
4. Ensure that **Copy projects into workspace** is selected.
5. Click on **Finish**.

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:

You will find the source for this in ExamResults-Step2-WebInterface/src folder

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 the project **src/com/mindstormsoftware/examresults/entity** folder.

Modify the Servlet (ExamResultsServlet.java)

We have modified the earlier ExamResultsServlet.java, that 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.

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** is 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. Please go through the files and add them to your project.

App Engine configuration files

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

web.xml

If you have been developing the application from the previous session, then you will already have an **ExamResultsServlet** mapping created for you. Ensure that it is there, in case you have changed the Servlet name or written a new one in this session.

The other entry that we need to add is for the Error page definition. 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 the entry is added 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 go to the **WEB-INF/appengine-web.xml** file and you will find the following line anywhere inside the root element.

```
<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 created 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.