EAS Assignment Brief

Aston University
Engineering & Applied Science

DC3160: Enterprise Application Technology	**Coursework** Aston SnowDome
Dr. Hai Wang h.wang10@aston.ac.uk	

Assignment Brief/ Coursework Content:

This coursework has ONE (1) compulsory section, worth 70% of the marks, and FOUR (4) optional sections which are worth 30% between them. When you submit your zipped Eclipse/NetBeans project, you will need to specify whether you have tackled any of the optional sections.

The lab instruction is based the use of Java Servlet to implement the system. You are allowed to use Spring Framework to build your system as long as all the required functionalities are realized.

You have been provided with a MySQL database on the teaching server which you should use in your application. Alternatively, you can also use your own MySQL database server. Do not make any structural changes to the schema or tables which affect your code. You do NOT need to submit the database with your work to Blackboard: simply the zipped Eclipse/NetBeans project with all necessary libraries to run the application. You will also need to upload your project to either teaching sever or your AWS environment.

The code that you submit must be your own work. If you need to re-use any code from another individual or from the Web you must CLEARLY mark the beginning and end of that section of code, and give a name or Web reference in the comments. All code will be checked against other code from this year AND previous years using Stanford University's 'Measure Of Software Similarity' application. Please specify clearly the URL which can run your application in the readme file.

Your code should be clearly commented so that a reader can tell how each component is intended to function, and why you have implemented the application as you have.
Your application should run on Tomcat, using Firefox as the default browser. All files and libraries necessary to build and run your application must be presented.

If you want to depart substantially from the style of the lab exercises (e.g., more than one Controller) then please also submit a short explanation of your design decisions (you can type or paste this in on the Blackboard submission) letting me know what architecture you chose, and why. This will help us to mark your work. ***P.S. DON'T USE SCRIPTLETS!!!***

Descriptive details of Assignment:

The Aston SnowDome is offering free snowboarding/skiing lessons for one week, to attract new customers. Your Web application should allow a registered user to select up to three free ski/snowboarding lessons from a list.

A user must log in to view the lesson timetable and book lessons. When they log in, their currently-selected lessons should be retrieved from the database. The list of selected lessons must be stored in the user's session, so that they can view their current selection on demand. It may be stored in the form of a `LessonSelection` class similar to the `ShoppingCart` example from lab, or in another data structure of your choice. An example `LessonSelection` class has been partly written for you. When a user has selected three free lessons, they should be unable to select further options until they cancel a lesson. In this scenario, there's no limit to the number of people who can sign up for each lesson.

The user's selection should be written to the database when they hit a 'Finalise Booking' button in the page footer. When this happens, any existing bookings in the database should be overwritten.

Each lesson is modelled by a `Lesson` class whose properties represent lesson characteristics such as start and end time, level and description. This class has been written for you and should not need extending.

`Lessons` are created from information stored in the MySQL database. This database can be accessed using the `LessonTimetable` and the `LessonSelection`. The timetable presented to all users will be the same, so it is up to you to decide on the appropriate scope for the `LessonTimetable`.
For optional points, you can add functionality as follows. Each optional task is described in more detail on the following pages.

2. Restrict the number of lessons per user. 10%

3. Allow a new user to sign up, choosing a username and password 4%

4. AJAX implementation to check if the selected username exists: 11% (depends on step 3)

5. Client-side validation of username (checking string length) 5% (depends on step 3)

Only do this section if you have completed all the other optional sections.

Client-side operations will require javascript functions, which can be written inline in the head of a JSP as in Lab.

The screenshot examples here are extremely basic (!). They just show the information which a view should display. You are free to make your layout and styling more attractive, but you DON'T have to: there are no points specifically for presentation.

Section 1: worth 70%

You have been given a zipped NetBeans project which contains a homepage (`login.jsp`) and four Java files in a '`model`' package: `Lesson.java`, `LessonSelection.java`, `LessonTimetable.java` and `Users.java`.

NB: some of these Java classes read from and write to the database using pooled connections. If you are familiar with other ways of implementing database persistence (e.g., Hibernate) you are free to use them, and we will be happy to see a variety of approaches, but this is NOT required and will not attract extra marks. If you do use something different, then please let us know when you submit the work.

You also have a `Controller` servlet in the 'controller' package, and this servlet has been mapped in the 'web.xml' file of the project to the URL pattern '**/do/***'. This servlet will control access to a set of Views, which you will create.
The `Controller` servlet instantiates an instance of `Users` to check the validity of login details.

NB: the authentication required for this coursework is an extremely simple form-based approach. ***You should aim to protect against SQL injection****, but you do not need to implement encryption or secure transfer of a user's details. Again, you can do this if you wish, and I will probably enjoy marking it, but there won't be extra marks in this context.*

Your application will allow a user to make various requests. A suggested action name for each is supplied.

Every View except the initial login page must include a navigation bar with links to other pages, a 'logOut' link, and a greeting with the user's name (see below for a basic example).

a) Log in `/login`

When the user enters details on the login form and clicks the 'submit' button, their details should be checked to ensure that they have entered one of the valid username/password combinations. If the combination is valid:

- the user should have a session created for them.
- Their username should be stored in this session, under the key '`user`'.
- The Controller should instantiate a `LessonSelectionBean` (and assign it to the appropriate scope) which will hold any lessons currently selected by that user.
- They should then be directed to a `View` called '`LessonTimetableView.jspx`'. (see section b).

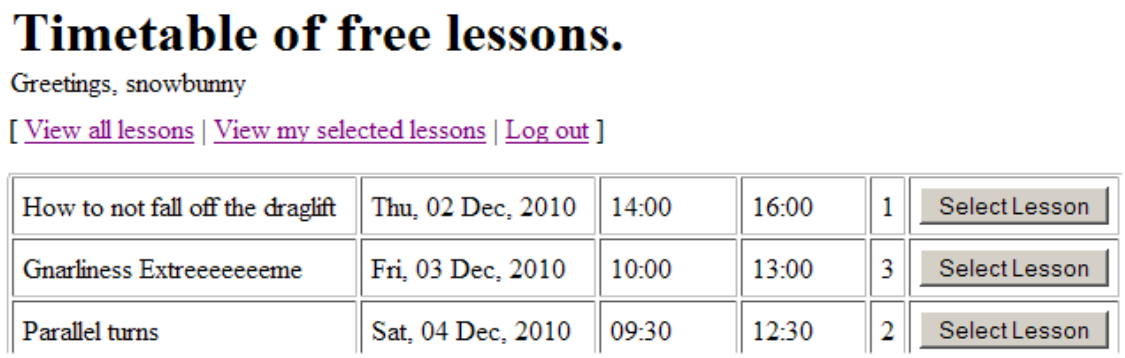
If the combination is not valid, or if an unknown action request is encountered, the user should be taken back to the login page.

b) View a list of ALL free lessons `/viewTimetable`

This View should be a JSPX document which displays the date, start and end time, level and description for each lesson, and a button allowing the user to select that lesson.

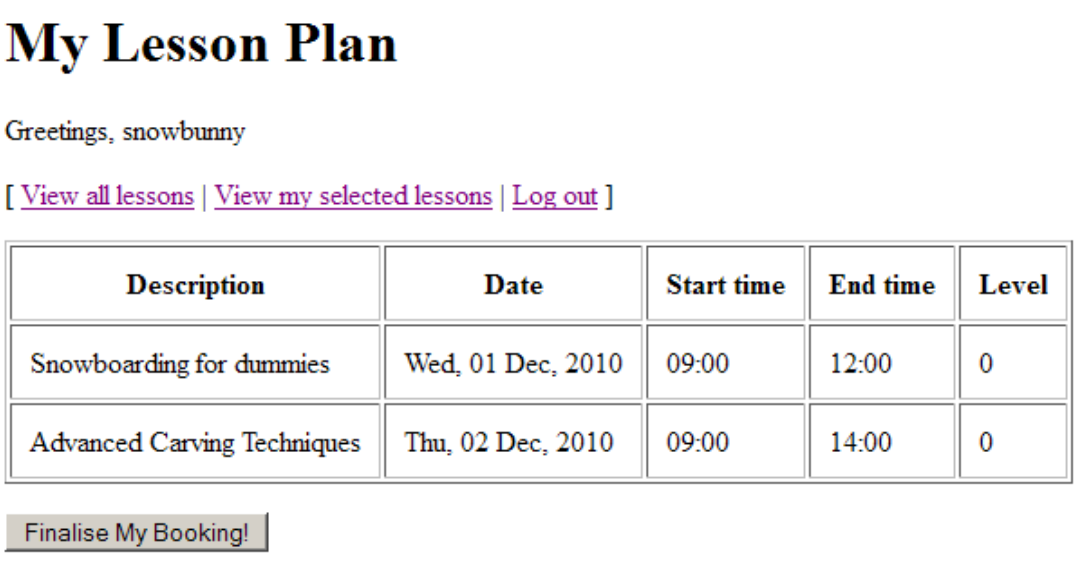
You may lay this information out in a table, or in a simple list as below. Each row should be contained within a separate form. The button at the start of each row should generate a POST

request with pathInfo **`/chooseLesson`**, and the lesson ID as a parameter. For an example of how to do this with a hidden text field, look at `ItemView` in Lab 8. The `chooseLesson` action is described in section c) below.



c) Choose a lesson `/chooseLesson`

When the user selects a lesson on the form above, its ID and information should be stored in their session. Your code should check whether they have already selected that lesson, and if they have, no new entry should be added. They should then be directed to a `View` called '`LessonSelectionView.jspx`'. This should display the list of lessons which the user has selected, with their details. The lesson selection view should include the provided file '`footer.jspx`' which allows the user to finalise the booking.



d) View their chosen lessons `/viewSelection`

This request (made by clicking on the link in the navigation bar) should direct the user to the same JSPX document ('`LessonSelectionView.jspx`') shown in section c) above.

e) Finalise a booking `/finaliseBooking`

This request (made by clicking on the button in the screenshot above) should prompt the `LessonSelection` (or whatever data structure you use) to delete any existing records for this user in the `lessons_booked` database table, and insert their current selection.

e) Log out `/logOut`

This request (made by clicking on the button in the navigation bar) should log the user out and make sure that the session is invalidated (use `HttpSession.invalidate()`).

The files you will need to create and edit for this compulsory section are as follows:

Views (all are JSPX documents):

1. Navigation bar
2. View showing all lessons, and allowing selection
3. View showing the user's selected lessons.

Controller:

You will need to complete the `Controller` servlet in the '`controller`' package.

Model:

1. You will need to complete the code for the `LessonTimetable`. It may be very similar to the `stock` class from Lab.

2. You will need to finish the '`isValid`' method in '`Users.java`' class so that it returns the user's ID if they exist, and a value of -1 if they do not.

3. You will need to create a structure to store the user's selected modules in the session. The easiest option is to finish the class called '`LessonSelectionBean`' in the `model` package, and base it partly on the '`ShoppingCart`' class from lab 8. However, if you have an alternative solution which works reliably, that is fine.

   This class has an '`updateBooking`' method which is responsible for updating the database when the user finalises their booking. You will need to implement this method.

   You will also need to finish the code in the constructor which reads the user's current lesson selection from the database and stores the relevant `Lesson` objects in a HashMap.

Many locations where you need to add or edit code are commented with a //TODO, and some hints.

Section 2 – advanced functionalities: Restrict the number of lessons per user. 10%

For this section, you should make sure that a user can only choose a maximum of three free lessons. When they have chosen three, the 'choose' buttons on their timetable view should become invisible or inactive. In addition, any buttons for lessons they have already chosen should be made inactive.

You should also add 'Cancel' buttons to the user's view of their selected lessons. Clicking the 'Cancel' button will remove it from the current selection held in their session.

To do this task, you might want to add a property to the user's lesson selection object which gives the current size of the selection. This is much easier than trying to directly access a collection's size using Expression Language.

Section 3: Implement user sign-up functionality 4%

For this section, you will need to add a 'newUser' action in the `Controller`, and implement the corresponding '`addUser`' method in `Users.java` which allows a new user to sign up and choose a username/password. If successful, the username and password should be added to the '`clients`' table of the database, and the user should be directed to the login page. Password security is not assessed for this coursework.

Section 4: AJAX implementation to check if selected username exists: 11% (depends on section 3)

In the '`login.jsp`' form, write AJAX code which allows you to check whether a user with a certain username already exists. You should make the '`Sign up as a new user`' button inactive by default.

Your AJAX code may be written using the jQuery library, or using the Firefox-safe XMLHTTPRequest object.

When a username is typed into the '`newUsername`' textbox on the form, and the user moves to another form element, your code should respond to the `onChange` event of the text box as follows:

- Your javascript should construct an AJAX request which sends the name to the `Controller` with the action '`checkName`'.
- The `Controller` should access the `Users` object to see whether a user with that name exists (you will have to write a function for checking this).
- The `Controller` should construct a response in JSON or XML which contains the following information:
    - If user exists: a 'true' value, or a string with a warning.
    - If user does not exist: a 'false' value, or an empty string.

FOR THIS ACTION YOU WILL NEED TO USE THE RESPONSE PRINTWRITER, AND MAKE SURE THAT YOU DON'T FORWARD THE REQUEST AS FOR THE OTHER ACTIONS. BE CAREFUL TO SET THE RESPONSE CONTENT TYPE CORRECTLY. LOOK AT THE 'COSTUMEDETAILS' CODE FROM LAB 4 FOR MORE IDEAS.

You should write a javascript callback function which:

- displays a warning to the user in a `div` or `paragraph` if necessary

- Activates the '`Sign up as a new user`' button if the username does not already exist.

Section 5: Client-side validation of username 5% (depends on section 3)

If you haven't already done so, adapt your '`login.jsp` ' page so that by default, its 'Submit' button is invisible or inactivated. When the username is entered you should catch an event on the textbox, and check to see whether the contents have more than 7 letters. Only if the username fulfils this requirement should the 'Submit' button be activated.

Key Dates:

Any key dates regarding the coursework.

05/11/2020	Coursework set
TBC	Discussion session for Coursework
20/01/2021	Submission deadline
17/02/2021	Feedback return

Submission Details:

To be submitted on Blackboard (see instructions online). Will be available closer to the submission time under "Assessment" tab on Blackboard page.

To be submitted:

1. Zip all your source code with a readme file, which indicates which questions you have attempted.
2. Any files or images as you deem necessary.

Marking Rubric

0-29	Little evidence to show for the coursework
30-40	The implementation is not functional. Very less objectives were achieved
40-49	A functional implementation, yet with very limited functionalities; With some major objectives achieved; No details about implementation itself; Lack of README file;
50-59	Objectives broadly met possibly with a few potential flaws – usability flaws or gaps in functionalities;

60-69	Deliverable substantially meet objectives, normally with few minor flaws; Detailed accounts of implementation; Detailed README;
70-79	Little flaws in the deliverable; Insight and innovation witnessed when describing the key points of implementation;
80+	Deliverable characterized by a very high standard of functionality