CS 5244 Internet Software
Semester Project: Enrollment System
Design Document

Scott Chappell
8/11/2013

Introduction

The goal of this project was to create an enrollment system web application that consists of several jsp and servlet pages. The size is big enough to demonstrate many of the features from a real world e-commerce application.

Design

The design for my web application consists of nine jsp's and six servlets; in addition to two helper classes, the basic enrollment manager, and an external CSS style sheet. One jsp displays and is responsible for one page that the user views on their screen. I tried to make it so that one button on a jsp corresponds to one servlet where the parameters are sent and processed. The CourseCart servlet is a slight exception to this rule as it processes three buttons from the cart JSP.

My enrollment system starts on the index page where you need to go to the administration page to initialize the system. This step is important to create a new basic enrollment manager object and to create a new user data map object. The map object is used to store a collection of student id numbers and user data registration objects. From the admin page you should add courses to the system otherwise the application doesn't accomplish much. The admin page is really separate from the rest of the application, but it's important for starting the system.

The user actually starts on a login page. But they will receive an error message if they try to type anything in the fields because their information is not yet in the system. They need to click the register link which sends them to the student registration page. All the student registration data is stored in a separate UserData class. When the user first encounters the registration page, a new UserData object is created with blank fields. The user is viewing the actual data in the UserData object, even when the fields appear to be blank. When they enter data in the fields and hit the submit button, the parameters are sent to the StudentRegistration servlet which sends the values to the UserData object regardless of data validation. This happens so the fields can be pre-populated in the event that the user enters the wrong data.

In my design, the servlet handles all the validation. Once validation has succeeded, the UserData object is placed in a map with a newly created student ID as the key and stored in the

application object. If the validation fails, error messages are displayed on the user's screen, under each field if necessary.

If everything is successful, a registration success page pops up and displays the user's student ID number and a link to the login page. The student login page asks for the user's student ID number (that was just displayed), and a password that they entered during registration. When the user hits the login button, the parameters are sent to the StudentLogin servlet for validation. Incorrect values will return an "Invalid student id or password" message. If validation succeeds, the user is sent to the main enrollment page.

On the main enrollment page, the user will see a list of all the available courses. Each course has a link to a course detail page where the user can view the course name, capacity, available seats, and description. One jsp is able to generate all the details necessary for the course detail page with the use of query strings. Query strings enable the system to generate a course code when a course link is clicked. You can use that code (in this case, just the course name) to find all the information you need for that course. If the course has available seats, a button will be available that will allow the user to add the course to a cart. When the user clicks on the add-to-cart button, it sends that course name to the CourseCart servlet. The CourseCart servlet places courses in a collection to represent all the courses in the cart. The only error occurs if the same course already exists in the cart. An error message will appear informing the user of this problem.

Next the cart contents page pops up with the recently added course in the cart. You have three buttons to choose from, clear cart, remove, and checkout. Clear cart removes all the courses from the cart. The remove button removes a single or multiple courses from the cart depending on if the corresponding checkbox has been selected next to each course.

Finally, the checkout button sends the contents of the cart to the CartCheckout servlet. This servlet is the exception where it handles all three buttons on the cart contents page. Depending on the button selected, a different action will occur. Three errors can occur when you hit the checkout button; the cart may be empty, the course may be full, or the student might already be enrolled in the same course. If any of these actions occur, the transaction does not complete, an error message pops up telling the user why, and no course is enrolled even if only one course has an error.

After the checkout button is clicked, the last page is the student detail page. This page displays all the courses a student is enrolled in and the course history for that student. I have logic displaying

only current courses in the currently enrolled section and only completed courses in the course history section.

The last servlet is the StudentLogout servlet that will invalidate the session when the user clicks on the logout link. Logout links exist on the main enrollment page, the course detail page, the cart contents page, and the student detail page. Basically, all the pages that have the username displayed. Each of these pages require a logged in user, otherwise the user is redirected back to the login page. The system stores an authentication token in the session object. Each of these pages requires that authentication token to access them.

Synchronization is very important for this system to ensure that it is thread safe when used by multiple users. I created a new class to serve as a thread safe wrapper for the basic enrollment manager. Each of the BEM methods is called in a synchronized manner to prevent different threads using the same resource at a bad time. Also, the cart checkout process is synchronized to make sure the BEM remains valid during a critical function of the system. The thread safe wrapper is very useful but it is not full proof. For this project, we are allowed to use the thread safe wrapper as the main thread safe solution. But it should be noted that all shared mutable objects are in danger of thread problems. This application is not completely thread safe because of this! A fix would be to synchronize all the shared mutable objects.

I found that validation of the fields in the registration JSP to be one of the more challenging aspects of this project. I used the internet as a resource to lookup how to use regex for matches() commands. When searching for Java syntax questions on Google, Stackoverflow.com always seems to pop up. A user named Adam Liss had an example of text.matches("[0-9]+"), and I realized I could check the input values against a certain range of values. I used another source from sourceforge.net that described all the regex operators. From this I learned that "+" means match one or more of the given characters in the set.

I arrived at the conclusion that regex was fantastic for the password, state, zip code, and phone number fields. For example, you can use state.matches("[A-Z]+") to ensure that the input is an uppercase letter that is A through Z. I could not figure out a good way to use regex for the name and city fields. I'm not sure how to write a regex statement that allows one blank space but not 50 blank spaces (if the user does a first and last name). So I settled on using name.contains(" "), and had the

validation fail if the name contains two black spaces in a row. It does pass the basic requirements doing it that way.

I was having a really hard time of writing code that would check if each character was a digit or a letter for the password piece of validation. I could not getting Character.isLetter() or Character.isDigit() to work for me regardless of what I tried. I suspect I was just missing a simple syntax thing. But I managed to solve the problem by using Integer.parseInt() on each of the characters. If the system throws a FormatNumberException, you know that you have a letter, and you can catch the exception and increment the letter counter. This was my crazy idea, but I wonder if it's completely inefficient to call the parseInt() method, and to throw an exception just for the sake of counting letters?

I was searching Google again to figure out the best way to make a good external CSS style sheet (I haven't made one before). One post from Stackoverflow.com struck me as particularly brilliant. BalusC mentions that you can use CSS to change the appearance of buttons on your JSP. So I used that method to make the button for my logout form to appear as a logout link. Under the username on four of the pages there will be a logout link that should look like a button, but looks like a link instead. It was a great way to meet the CSS requirement for this project. Also, I used my external CSS style sheet to make my error messages orange, and the links maroon. Go Hookies!

Design Structures

Like assignment 3, a single BEM implementation is stored in the application object. New to this project is a map of student ids and UserData objects that is also stored in the application object. It's very important that this map is stored in the application object as the system can retrieve the user's data independently of the session. Initially the UserData object resides in the session object, but it is placed in the map and stored in the application object after registration is completed.

Several important objects are stored in the session object when it makes sense; including the contents of the cart, the login authentications, and the error messages. All of which are dependent on the individual sessions where they take place. If a new user logs in, they should have a completely different session.

For the error messages of the registration page, I used a hash map to store all the various messages. This allowed me the ability to easily display multiple error messages on the same page. Depending on the error message code, I could give the user the reason that the password or city name

was incorrect. Also, I only had to store and send one object to the session object for all the messages on the registration page.

For the history list of the student detail page, I decided to use a linked list collection. I figured a linked list would work well as I needed the ability to add multiple courses of the same name, and sets do not allow duplicate entries. I found a set to be the ideal choice for the cart as the system does not allow multiple courses of the same name in the cart.

Testing

For testing, I tried to do as many unexpected actions as possible to see if I could break my system. I did succeed in uncovering some interesting bugs. One in particular occurred when I registered and logged in to the system, then I went to the admin page and re-initialized the system. The user is removed from the system, but their session authentication remains. So you will get StudentEnrollmentExceptions on all the pages that require the authentication token and the only way to fix it is to wait until the session times out or to re-deploy the system again. I'm not sure if you would have a problem with this since the user would not be able to re-initialize the system in a normal situation. But I decided to add "session.setAttribute("loggedInUser", null);" to the initialization section of the EnrollmentAdmin servlet just to prevent this problem.

I also found a bug where you could checkout a course and everything would work correctly. But if you complete the course, then log out and back in; it would cause a null point error to add the course a second time with the checkout (but checkout would still complete and the student would be enrolled). I found out that I had called the courseHistory list from the UserData object incorrectly. It was working for the session object, but not for the application object. I almost missed that one! Trying to do unexpected or rare events caught lots of bugs.

I think I caught most of the unexpected errors. But I'll actually be surprised if I got all of them. I was struck by how many if statements were required to prevent certain conditions. Specifically it felt like I was hunting for every possible way the system could throw a null pointer exception, and blocking it with an if statement.

Weaknesses

The biggest weakness of my enrollment application is that I do not have client side validation. I realize that client side validation is an extra, but it is a powerful tool that reduces server load and makes the overall application better. I was hoping to have it added by the end of the project, but I just ran out of time at the end. Finding as many bugs as possible took a little longer than I anticipated, and it seemed like a bad idea to tack on shoddy javascript with the possibility of making mistakes.

The other weakness of my application is a little subjective, but I don't have much design invested in it. On the one hand, you can say that it is very simple and clean looking, but maybe also on the boring side of things. If I had more time, I would probably try to spice it up a little and do more with the external CSS style sheet.

For the student registration page, all the fields are pre-populated, but the two survey questions are not pre-populated. I wasn't sure if it was necessary for the survey questions as I've seen it done both ways for real applications.

Conclusions

Overall, I think my enrollment system does a good job of meeting all the stated requirements. It is really daunting to account for every contingency that can occur with the system. I can only imagine how much of a challenge it would be for a large application. I thought this was a really fun project which demonstrated lots of the features from e-commerce applications found across the web.

References

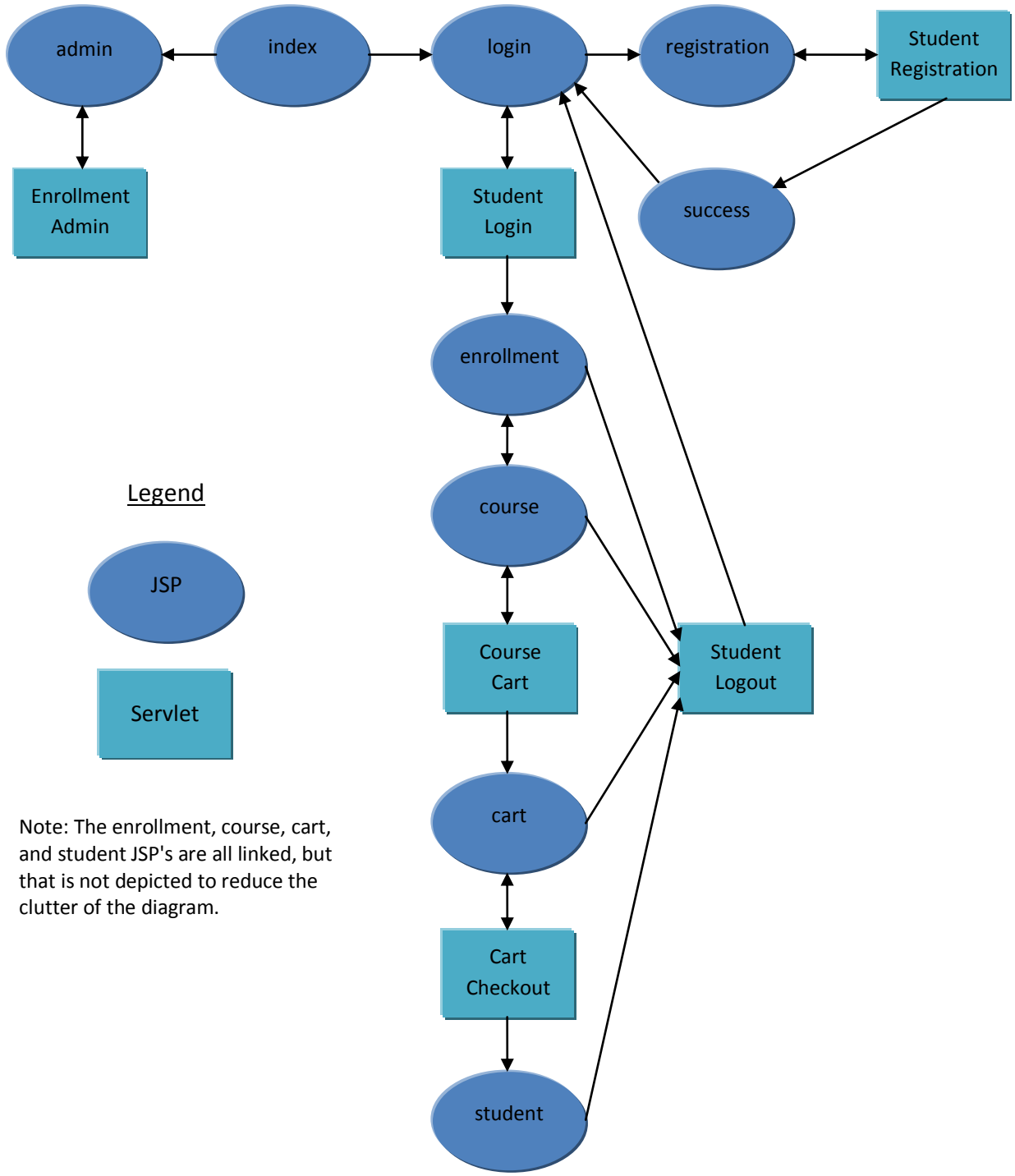
BalusC. "Submit form using a link on JSP". Stackoverflow.com.
http://stackoverflow.com/questions/4114554/submit-form-using-a-link-on-jsp

"Common Operators". Sourceforge.com
http://metahtml.sourceforge.net/documentation/regex/regex_3.mhtml

Liss, Adam. "Java String - See if a string contains only numbers and not letters". Stackoverflow.com.
http://stackoverflow.com/questions/10575624/java-string-see-if-a-string-contains-only-numbers-and-not-letters

"Styling Links". w3Schools.com. http://www.w3schools.com/css/css_link.asp

Semester Project Enrollment System Servlet/JSP Relationship Diagram



Note: The enrollment, course, cart, and student JSP's are all linked, but that is not depicted to reduce the clutter of the diagram.