**INTRODUCTION**

Hello! I’m Corey Schultz. Welcome to this discussion of my final project. I apologize if I am difficult to understand, because I’m experiencing some allergies today. To be safe, I have attached the script for this video along with the rest of my file submissions.

The majority of this video will be taken up by the PowerPoint presentation I have designed, but I have arranged a section at the end of the video to view my code and student portal directly, since I think it will be easier to explain the coding concepts that way instead of shuffling through screenshots of code. With that said, we have a lot to cover, so let’s get started!

**SRS Document**

We will begin by examining the key aspects of the software requirements specification document created in Week 1 of this course.

I have taken snippets of the sections which are, to me, most important for understanding my design philosophy for this student portal.

First, we have the product scope, which gives a general overview of the features and limitations of the system. This section lists the intended features of the system which I have – hopefully- fully delivered, including student account creation and the ability to enroll in or drop courses.

Next is the user interfaces section. This says that the design conventions of the website must be uniform, which is important for accessibility and readability. You might argue that my visual design is a little too minimalist, but I choose to believe that the simple design makes it easy for users to locate their desired functions. This section also lists the pages which must be available to facilitate the core functionalities.

Following this are two points from the system features section. First is the user registration and login feature, which mentions the importance of preventing the creation of duplicate student IDs. We will revisit this later.

The course browsing and enrollment feature, meanwhile, specifies that courses from all semesters should be available for viewing to the student; that the maximum number of enrolled students should be clearly displayed; that each full course should offer a waitlist for a student to enter in case an enrolled student drops the course; and that a student should be able to cancel their enrollment in any particular course.

**UML Designs**

Building off the basis established by the SRS document, let’s take a look at the various UML diagrams which I created using those specifications.

First is the use case diagram, which is used to show the options available to external actors using the system, as well as the external requirements the system will need to function. Here we can see the student as the external actor interacting with the website, and the web server – which is XAMPP, in this case – and the database which support the website’s functions.

Next is the sequence diagram, which traces the exchanging of different types of messages between the main entities of the program. We see that the student sends login information to the website, and the site in turn sends a login verification request to the database. If the information matches, the database sends back confirmation of verification, and the website displays the user’s profile and course information accordingly. When a student chooses a course for enrollment, the site checks with the database to see if there is an opening in that course. The database returns this status, the student is notified by the website, and their new schedule information is displayed.

On the next slide, we have the activity diagram, which is a way to show the flow of control in the program through the actions taken. We can see that control moves from the student, after submitting their information, to the system, which processes their login credentials. The user can then select a course to enroll in, prompting the system to determine whether it is necessary to place the student on a waiting list, and later giving the student the option to drop a course.

The class diagram displays the system in terms of the interfaces of the software, as well as the features and relationships between those classes. You can see that the student class is linked to the login class, representing the login validation process, and the Course Registration System entity makes use of the Course and Schedule classes for its operations.

Finally, the state diagram is another way to examine the system’s behavior in terms of the state of the system based on the occurrence of different events. The events are similar to the processes we discussed in the previous slides: the system state changes when an account is created, when a student is registered for a course or added to a waitlist, et cetera.

**Database Design**

Given that the student portal is dependent on information stored in the database for almost all its functions, I think it is appropriate to cover the design of the database before examining the code.

The database, called studentPortal, is made up of four tables: tblUser, tblLogin, tblCourse, and tblEnroll.

tblUser is the table which contains student information, and entries to this table are generated via the registration process. As we shall see later, the primary key of this table is the student’s ID, and this acts as a mechanism for preventing two students from having identical IDs. In the interest of keeping the student’s profile realistic, I decided that they should have their email, password, first name, last name, address, and phone number listed in their account information.

Moving clockwise, we see the tblLogin table, which is used for login verification. When a student creates an account, their ID and password are duplicated within this table. Then, when they attempt to login, the ID and password that they enter are cross-referenced with tblLogin, and the user is given access to their account if the values match.

Next is tblEnroll. Each row has only two entries which change: a student ID and a course ID. When a student enrolls in a course, the course ID and their ID are added to this table. Initially, I experimented with creating a table for each student which would act as their schedule before I realized how horrifically inefficient that was. To avoid data processing issues, each entry of tblEnroll has an automatically incrementing primary key, which also means that a student can enroll in multiple courses.

Finally, tblCourse stores the information about each course available to a student, including the course ID, course name, a short description of the course, the name of the professor teaching the course, the semester in which the course is offered, the number of currently enrolled students, and the number of students on the waitlist if the course is full.

**Landing page**

The home page, as you will immediately notice, is very simple. I would definitely like to come back to this and touch it up in the future, especially when I have a firmer understanding of CSS styling methods.

In the meantime, you can see three links: the home link, which loops back to this page; the login link; and the registration link.

**Registration page**

The registration page is where a student can enroll by entering their personal information. As has been mentioned, the ID which the user comes up with will act as the primary key in the User table to keep their account unique. You could certainly try scrolling up to an 8-digit number with the arrow on the side of the entry field, but I would recommend typing the number yourself.

**Login page**

The login page has two entry fields: an ID, which we should have just generated on the previous page, and a password. Once the user has entered these and been validated, they will be given a success message.

**Course Registration**

I changed the course registration page from the way it was laid out in my Week 4 submission. Using a button to enroll for each class with an HTML table complicated the code and processing needlessly, so I decided to use another submission form, where the student can enter the Course ID for the class in which they wish to enroll. If there is space in the class, they will be notified and the number of students listed will be incremented. If not, they will be told that they have been added to the waitlist for the course.

The process of dropping a course is similar. The student enters the Course ID and is notified, and the number of enrolled students is decremented.

**Student Profile Page**

Last is the student’s profile and schedule page. All of the information they submitted during enrollment is available here. Beneath that, an HTML table displays every course in which a student is enrolled along with all of the class details from the previous page.

Each of these pages also features a logout function which returns the user to the portal home page.

**Code Highlights**

Before we look at the portal and code directly, I thought it would be worthwhile to highlight a few snippets of code which represent learning experiences for me. The first is fairly simple, but it was the result of a challenging coding session. I realized that, while I had manually created the tblCourse table for testing purposes, this option would not be available to someone grading my code. So I needed to generate some data, but I also needed to stop the system from trying to reenter this data with every page reload and causes duplication errors. I learned the INSERT IGNORE SQL statement, which accomplishes exactly that.

I also had trouble figuring out the syntax for grabbing all of the course information for a course linked to a student ID in the tblEnroll table. I eventually landed on this nested SELECT statement which also makes use of the IN keyword to grab every instance of a course ID with a student ID that matches the query.

**Using the website**

Now we can finally see the student portal in action! Here is the home page. We’ll create a new student profile.

[Skip to being done typing]

Now we’ll press submit… Great! Okay, now we can see that we have been added to the tblUser and tblLogin tables.

[show database]

Now we’ll try logging in.

[skip typing]

And we’re in! Let’s look at the course and try registering for one.

We’ll try the Data Structures and Algorithms.

[enter it]

Nice. Let’s add the History of the US, too.

[enter it]

As you can see, since the class is full, we’re added to the waitlist instead.

If we check out our profile page, we can see the new class on our schedule, and can see that the number of enrolled students has jumped from 25 to 26. I think that’s enough enrollment for now, so let’s log out.

Thank you very much for listening!