Table of Contents

List of Figures

List of Tables

1 – Introduction

This report attempts to present a thorough picture of the Integrated University IT College System's planning and development. The method decided upon as IMAT2718 Integrated Project instruction. The project aimed to build for an Information and Technology College inside a university an efficient and fun course module administration and user registration system. The concept, tools, and approaches used, the development process, and how we applied Agile methodology in the execution of the project are discussed in this paper.

The intended system is a combined solution whereby users could have access to data regarding courses and events hosted at the IT College. It helps course enrolment, module choices, module information checking including course specifics. The system is easy to use and accessible with basic browsing options allowing the student to investigate, form opinions and complete registration.

Technically, ASP.NET Web Application (.NET Framework) allowed us to use Microsoft Visual Studio 2022 to build the frontend and backend of the programme. The website is developed with HTML, CSS, and JavaScript; the server runs using ASP.NET. To let the system run fast and properly, we stored and managed data using the MySQL database. Whereas GitHub became our go-to platform for collocation and version control, Visual Studio 2022 became our ideal environment for programming.

Under an Agile approach, the team worked in quick, continuous sprints towards consistent improvement. Regular team meetings using Microsoft Teams allowed for system development, issue resolution, and communication of progress. Based on Agile principles, the project was developed to guarantee that the final output maintained to meet all the needs and standards by means of progressive releases and partner and end user input.

2 – Vision

Targeting management and choice of IT-related courses and modules in an academic university setting, the Integrated University IT College System is an application-specific, web-enabled solution.

What this project has produced the UniTech College System, a creative online tool meant just for college students enrolled in IT-related degrees. The system can be utilised by students who are looking for searching, selecting and applying for courses and modules in the specific areas such as Artificial Intelligence (AI), Computer Science, Software Engineering, Data Science and Cyber-Security. Handling student data, training choices, and personal accounts, the application links with a management server MySQL database using an easy and quick interface. The goal is to provide an academic virtual environment to help the student all through the learning process.

Who the target group for this system are the students alone, more precisely those joining or presently enrolled at the IT college at the university. The main program users and beneficiaries are this group. From securely logging in to browsing course options, selecting courses, and managing personal data, every element of the system has been developed around them.

Why such an infrastructure was absolutely necessary since manual course and program decisions usually take time, are difficult, and full of mistakes. Through creating a website whereby decisions are made more openly and honestly, we are making learning more accessible to people which decreases uncertainty and it also offers students more control over their own learning journey. It also improves accuracy since the data kept in the system is accurate and easily restorable when the development of writing form or irregular spread sheet data is gone.

The system caters to the needs of the stakeholders and emphasises students' values. Upon entering the website, a student will be able to learn more about the courses and available modules. With a user-friendly interface and appearance, it meets modern online learning requirements. Navigating is simple and elegant, fit for modern online learning standards. Module view, personal information updating, and continuous login sessions enable their daily academic preparation. They would measure success by a 40% reduction in average

time to search for and view module information (as observed at user-acceptance testing), a 90% rate of registration completion within UAT without any intervention by staff, and at least 80% positive usability feedback from the key journeys like browsing courses, viewing modules and registering, having been collected from testing by a group of ten students.

The business purpose or rationale for this project was to address the delays and lack of responsiveness of regular student registration techniques. Most educational organisations still employ and utilise outdated processes that annoy students and slow them down drastically. The UniTech College System overcomes this challenge with a dynamic, student-oriented platform with the frontend and server features constructed using ASP.NET (Web Forms) and MySQL are used to manage large amounts of data. This is therefore an excellent website for students who are looking to join UniTech College, it is future proof for digital permitting students who are enrolling.

3 – Incremental Deliverables (Sprints)

3.1 – Sprint Planning and Timeline

From the project's inception, we used an Agile strategy in which we steadily constructed and enhanced the UniTech College System, with each sprint taking around one week in terms of time, dependent upon the complexity. Every sprint stated precise targets, output, and a break for evaluation. This presented us with the ability to adjust the strategy according to what we needed to achieve, what we need to do differently, and the time we had available as a college student.

The plan for the first sprint was established in the first week of the project. We highlighted the five key sprints required to move the project from concept to delivery. These were designed in a sequence that naturally followed each other, with initial server work first and concluding in final testing and merging.

3.2 – Incremental Breakdown

In the following sub sections, we will be explaining the incremental breakdown we did to make this project happen.

3.2.1 – Sprint 1: Database Design

Database Design In the first sprint we constructed a database layout by applying (MySQl workbench). We incorporated crucial tables in (MySQL) such as Modules, courses, Users and enrolments. For example, in the database we constructed a table named computer science, Software engineering, Cyber security and data science. We also added a statement in the database so that students would understand what the courses are all about if they were interested in any of them. For building an enrolments table this was for persons enrolling into the course. we added the Enrolment ID, User ID, course ID and the foreign keys for user ID and course ID. This aimed to aid students to be able to assist sign on to the degree they desired. We also constructed a user's table so that students or workers would be able to login using their email and password. So, if they were to enrol again, they would be able to examine specifics about what courses for example they choose. We obtained a peer review indicating us where module descriptions were lacking; we addressed this by adding a "description" column to the Modules database and re-executing test queries to ensure complete information showed as anticipated.

3.2.2 – Sprint 2: Basic UI wireframes and courses/module listings

In our second sprint we generated wireframes for our major pages like for example home, login classes about and log out. To structure these pages, we employed asp.net along with aspx, CSS and html in order for us to lay up each page effectively. We concentrated on adding the front end and the rear. The front end seeks to allow students at Unitech college and staff to be able to interact and comprehend with what they are visually being

offered. This encompasses for the front-end planning and structure, the appearance and style and also students' input (interactivity). Backend involves the database processing the requests that the user will immediately see. For example, if the user logs in, that data is then given to the database server to confirm whether they are the valid passwords. For our courses we have specialised wireframes such as Courses list and modules list to offer students further information about their selection. Our original designs were not mobile-responsive, so via a series of three user tests we created responsive CSS media queries and altered breakpoints; follow-up simulator testing revealed correct behaviour on desktop and mobile.

3.2.3 – Sprint 3: User registration/login functionality

For user registration and login features we made sure that users were able to login with the appropriate email and password and create and account on Unitech College. For registration, we included 3 variables. The first input was adding a complete name field. The second option was for the user to enter in their entire email address and the password must consist of 8 characters. With 1 major letter, 1 special character and 1 number. If the users performed these 3 items properly, then they would be able to successfully have their Unitech college account established. Assuming the user has established their account, for them to login they would type in their email that they used to register and password they used to register which are the two entries. We deployed asp.NET to add login, registration and password security to verify these features are established. The passwords were retained in the database and scrambled to prevent users' data from being leaked. Furthermore on the login page if the users typed either the incorrect password or email for more than 3 times, it would lock the user for 15 minutes and it maintains how many unsuccessful attempts the users has made and what was the last failed login attempt in the database. We done this so that the identities had matched what was inside the database. Testers experienced difficulty with ambiguous error messages in a user-acceptance session, therefore we supplied inline suggestions and a real-time strength gauge record mistakes reduced by 30 % while all testers performed the assignment first time.

3.2.4 – Sprint 4: Module selection and module page

For our topic choose, initially customers had 5 possibilities to decide from the uni tech college course website. Users may decide between computer science, software engineering, cyber security, data science and artificial intelligence. Underneath the courses displays 4 parts. Students may only choose 3 of their options. For example, if a person has selected computer science as their study, they may choose 3 out of the 4 following courses for computer science which are java programming, database management and object-oriented programming. On the module page itself, users may choose if they wish to modify registration. This means they may alter course in the modify registration field and go to for example cyber security. Once they save it , will keep the latest registered course that the user has moved to. As indicated in sprint 1, we constructed users tables, module tables classes table and enrolments table in the SQL workbench database to support and deliver the display on the web application. In addition to module selection, if the user was to choose all 4 modules it would give them a mistake stating, 'you only need to choose 3 modules in red'. We placed this in the database to make sure our web application correctly corresponded with what we contributed. Early functional tests provided a misleading error message ("You need to select only 3 modules"), which we had amended to "Please pick a maximum of three modules" with a pre-selection warning added; additional testing indicated no more ambiguity.

3.2.5 – Sprint 5: Final testing and integration

In our recent run all the key components were integrated in our web application. We reviewed all the functional and non-functional tests such as movement control, module selection, user registration and signing in. We made sure that the web application was not sluggish and it was very agile and speedy for users to be able to utilise. We made sure that there was no difficulties with writing and the presentation was clean and straightforward for our users. In addition to this we made sure that the front end and back end had the necessary email forms and valid passwords. Finally, we evaluated the database in MySQL to make sure that there were no copies and we also tested to see if the models were operating. In summary, this is all about taking what we have developed from the previous sprints and making sure that everything on the website works smoothly and operates at a solid working state. We noticed a sluggish JOIN in a load test; we rectified this by

indexing the course_name column and by optimising the query, lowering page-load times down from 450ms to 180ms for under 20 concurrent users.

3.3 – Feedback and Iterative Improvement

In our process of establishing our web application, we made sure that included certain improvements in making our web application stand out and look more appealing to our visitors. We posted a welcome message on our home page to expose visitors to our website to show them who we are. We also chose decent hues for our web application. So, for example when you visit the website, instead of having an all-white backdrop we employed the colour black so that pupils might be able to view the different regions on the upper right-hand side. We also detected various security issues. We reduced the number of times users may attempt to connect using their username and password to prevent brute force attacks. This helped prevent login misuse and advised that students input the proper passwords accurately.

3.4 – Sprint Retrospectives (Summary of Lessons)

For all 5 sprints we made sure that they all were done to a high standards. What these retrospectives help us to know is how we can enhance our sprints to make them appear even better and explain and recognise the different type of difficulties. This helped us to adapt to making rapid and early adjustments, find out what type of issues we may have and overall have a quality produced web application. For example, in our programs, we inserted comments to improve the structure to help us comprehend what each line of code represented and what they are collectively expected to do.

4 – Personas, User Stories, Scenarios & Feature List

In the following sub sections, we will be outlining the system's users (personas), we will also be writing stories illustrating how users might interact with the system, and we will also be developing scenarios and a feature list using the input, action, output model.

4.1 – Personas

Ali – First Year Computer Science Student
Ali is 18 and began attended UniTech College. Finding what the course and courses he can access contain thrills him. He is tech-savvy but inexperienced with academic procedures and wants everything to be neat and transparent.

Sarah – The IT facility Administrator
Charged for course registration, module release, and student data, Sarah works in an executive office. She is in charge of confirming proper course information, student record maintaining, and student program selection

assistance. She requires speedy system access devoid of needless procedures.

James – Final-Year Cyber Security Student
James must complete his studies for his final term; he is 21 years old and has expertise in university systems. He wants to examine course standards and guarantee he can complete. If his personal details change, he also loves to update them.

Dr Patel – Lecturer in Computer Science
Dr. Patel is a senior lecturer who teaches units of Computer Science. He has to sign in to see and access the list of students who are registered in his unit, post reading materials, and make announcements. The most crucial requirement that he has is a secure and intuitive interface to interact with students registered in his unit and manage content with ease.

4.2 – User Stories

Story 1 – Ali:
As a new student, I would want to see a list of several courses via which I may choose one that is intriguing for me.

- **Given** I am on the course selection page,
- **When** I click on a course,
- **Then** I should see a list of related modules.

Story 2 – Sarah:
As an admin, I would want to maintain student information current and accurate by altering data.

- **Given** I have logged in as an admin,
- **When** I search for a student and update their information,
- **Then** the system should save the changes inside the website and database and confirm a message to me and the student that their information has been changed.

Story 3 – James:
As a returning student, I would like to see what courses I'm enrolled for in order to monitor my academic progress.

- **Given** I am logged in,
- **When** I go to my profile,
- **Then** I should see all modules I am currently enrolled in.

Story 4 – Ali:
As a new student, I would like to establish an account using my email address so that I may safeguard my log-in in the future.

- **Given** I am on the registration page,
- **When** I enter a valid name, email, and password,
- **Then** my account should be created, and I should be redirected to the courses page.

Story 5 – James:
As a student, I would value reviewing module material previous to taking them in so that I may make an appropriate option.

- **Given** I am viewing a course,
- **When** I click on a module,
- **Then** I should see its title, code and description.

Story 6 – Dr. Patel:
As a senior lecturer, I wish to view my assigned modules and see enrolled students so that I can manage teaching materials and monitor attendance.

- **Given** I am logged in as a lecturer,

- **When** I select a module I teach,
- **Then** I should be able to see the list of enrolled students and upload resources.

4.3 – Scenarios

Scenario 1: Registration and Course Selection by Ali:
Ali arrives on the main page of UniTech College. He clicks on “Register”, offers his name, email address, and password and is registered. After signing in, he is greeted and offered a selection of available IT lessons. The list contains “Computer Science”. The courses in “Computer Science” are shown. The courses are selected by him and are retained in his biography.

Scenario 2: Sarah Update Student Records:
Sarah logs in using her work account information. She gets into the admin panel and searches for Ali's name. She sees his chosen route and units. She detects a missing piece in his record and adjusts his register to remedy a module code. The system validates the adjustment by displaying a favourable message.

Scenario 3: James Verifies and Edits Details:
James checks in and is taken to his screen. He selects "My Profile" and looks over his selected course and continuing courses. He detects his old phone number, clicks "Edit", and inserts his new one. The system refreshes the new information and sends him to his screen.

Scenario 4: Failed Login Attempt by Ail:
Ali attempts to login but cannot remember his password. He submits wrong information thrice. The account gets locked for 15 minutes with a message stating the reason for the lockout. This helps to guard against a brute-force attack while at the same time informing the user why they are blocked.

Scenario 5: Dr. Patel Sees Enrolled Students and Posts Content:
Dr Patel logs into UniTech via his staff credentials. On his dashboard, he is presented with a list of modules he has been assigned to work on for the current term. He selects “Computer Science” and is shown a list of students enrolled on it. He uploads a PDF of lecture slides and publishes a notice for next week’s seminar. The system also alerts that materials were uploaded successfully and students are notified.

4.4 – Feature List

Feature	Input	Action	Output
User Registration	Users name Users email User password	Once the system is able to be validate, then the information of the users’ inputs should be stored in the database. All the inputs especially the password must realistic.	Once the user has created the account for themselves then they will then be sent back to the courses page.
Course Selection	Select the course box	The system will display the modules that are linked with course.	The system should list the modules that are based on the course
Module Enrolment	Clicking the module boxes	Once the user has selected the modules then it should link it to the user’s profile.	Confirmation of enrolment shown on the user’s profile.
Profile Editing	Updating the users name , email address	The system should update the database itself to reflect on the users’ changes	A message saying ‘success’ will be shown to confirm that the changes have been saved on the user profile.

	and phoner number		
Module and Courses editing	Updating the user's module and courses	The system will show the edited courses and module based on the user.	A message saying success will be shown and that edited modules and courses will be saved on the system.
Login	User logging in with both their email and password.	System matches current user input to the database credentials.	If the password and email is correct, then the system will take them to dashboard. If the password and email is incorrect it will ask them to re-enter the correct email and password. If the user has entered the password and email for 3 times. Then it will lock them out of the system for 15 minutes.

Table 1 - Feature List

5 – Product Backlog

Although our system still requires much work, the backlog points up priorities. These can be new features, fixes for present issues, or enhancements. The list helps us to schedule our activities between sprints so that all development criteria even the most basic ones are finished and finally worked on. Regular review of the list helps us to support our fast development.

5.1 – Features

Many necessary components have been developed or refined to offer a whole functional system:

- **User Account Registration:**
  Users should be able to establish a new account using only a name, email address, and strong password. This information has to be entered into the database and looked for repeated entries.
- **Safe Login System of Approach:**
  The login process has to check credentials against the database and offer suitable remarks should any data be missing or false.
- **Student Dashboards:**
  Along with their exact name after logging in, students get tailored information including their selected course and classes.
- **Course and Module Choice:**

For the students, it is absolutely necessary to be able to search among several courses and choose only one. Following a decision, a list of relevant courses needs to be on show so kids may apply for the ones they wish to study.

- **Profile Management:**
  Users should be able to view and alter their personal information, including phone number, email address, and password.
- **About Us Section Page:**
  Visit the "About Us" page to learn more about the UniTech platform, its goals, and its creators.

5.2 – Bugs

The listed above issues have been resolved to enhance dependability and end-user experience.

- **Login failures silently occasionally:**
  Not always does an invalid password produce an error message, which would perplex consumers. One needs more exact confirmation input to manage this.
- **Inaccurate Module Presentation for Specific Classes:**
  Either due of a database connection issue or incorrect filtering logic, the relevant modules may not always be supplied in the correct way when particular courses are selected.
- **Logout Function Not Redirecting Always:**
  Occasionally, especially on slower networks, the logout button does not return the user to the home page.

5.3 – Enhancements

We have embraced all of the recommended changes, which are meant to enhance user experience generally, boost system security, and give students more control, so strengthening our system.

- **Requirements for Password Complexity:**
  Users joining now have to satisfy a set of password criteria in order to increase account security. The fundamental requirement is that a password must contain a number, a lowercase letter, an initial letter, and a unique character. The creation of an account will be denied otherwise.
- **Lockout of Login After Attempts Failed:**
  For added security, a user's account will be suspended for 15 minutes if they repeatedly enter their login credentials incorrectly. This serves as defence against attacks using brute force. They are permitted to log in again after the allotted time has passed.
- **Guidance for Users Based on Course Selection:**
  A user will be immediately taken to their profile page to review their choices after successfully choosing their course and classes. The system will prompt them to finish this step before continuing if they haven't chosen a course or courses yet.
- **Cancel, Edit, and Drop Selection of Modules and Courses:**
  Users will be in total control of their academic destiny. They might change their preferences, drop an existing program or course, or cancel all of their options. This gives people more flexibility if they change their minds or make a mistake.
- **View the details of every UniTech course and program:**
  Readers can peruse all of UniTech's courses from the Courses page. A student will receive the required lessons after choosing a subject. A lesson's description appears when a student clicks on it, allowing them to see it clearly before choosing wisely.
- **Greetings on the dashboard:**

Users receive a personalised greeting and name when they check in. This enhances the system's aesthetics and personalises the layout.

- **Highlight of the Selected Program Visually:**
  It will be obvious to the user which course they are currently reading or editing once they have selected one (for example, by changing the colour or adding a border).
- **Description fields that are already filled in:**
  A person's name and email address will already be displayed in the form if they are asked to edit their biography. This is a time-saving method to avoid typing the same thing over and over.
- **Verification of Email Format:**
  A warning message will appear if the user enters an incorrect email address (for example, one that lacks the "@" symbol).
- **Simple calendar display:**
  A basic timetable that shows the current month and days is shown on the screen, though it is not entirely dynamic. Future iterations might add dates or events to this.

5.4 – Version Control Process (GitHub)

We adopted GitHub version control, which facilitates collaborative code management, change observation, pull request viewing, and rollbacks when necessary. For clarity and auditability, each member must create branches, commit frequently, and provide commit statements outlining the reasons behind their changes.

6 – System Architecture

6.1 – System Overview

The UniTech College System is an ASP.NET Web Forms, MySQL, and HTML/CSS-based web platform used for both functionality and user interface. The system has a number of core elements: a student and staff frontend interface to enter pages like login, registration, courses, modules, and profile; a server component used to manage logic, authentication, and communication with the database; and a MySQL database used to safely store user data, course data, enrolments, and module selections. All in all, the above elements provide users with the ability to register, login, view courses, list modules to choose from and view own details and have an interaction with a dynamic and responsive educational setting.

6.2 – Database Design

The following sections describe the frontend user interface high-level diagram, the backend logic high-level diagram and database schema with tables and relations. They were generated with draw.io to give a concise graphical representation of how system parts are interrelated. Each diagram contains a short description to indicate how it is meant to represent and fit into the overall UniTech College System architecture.

6.2.1 – Frontend UI

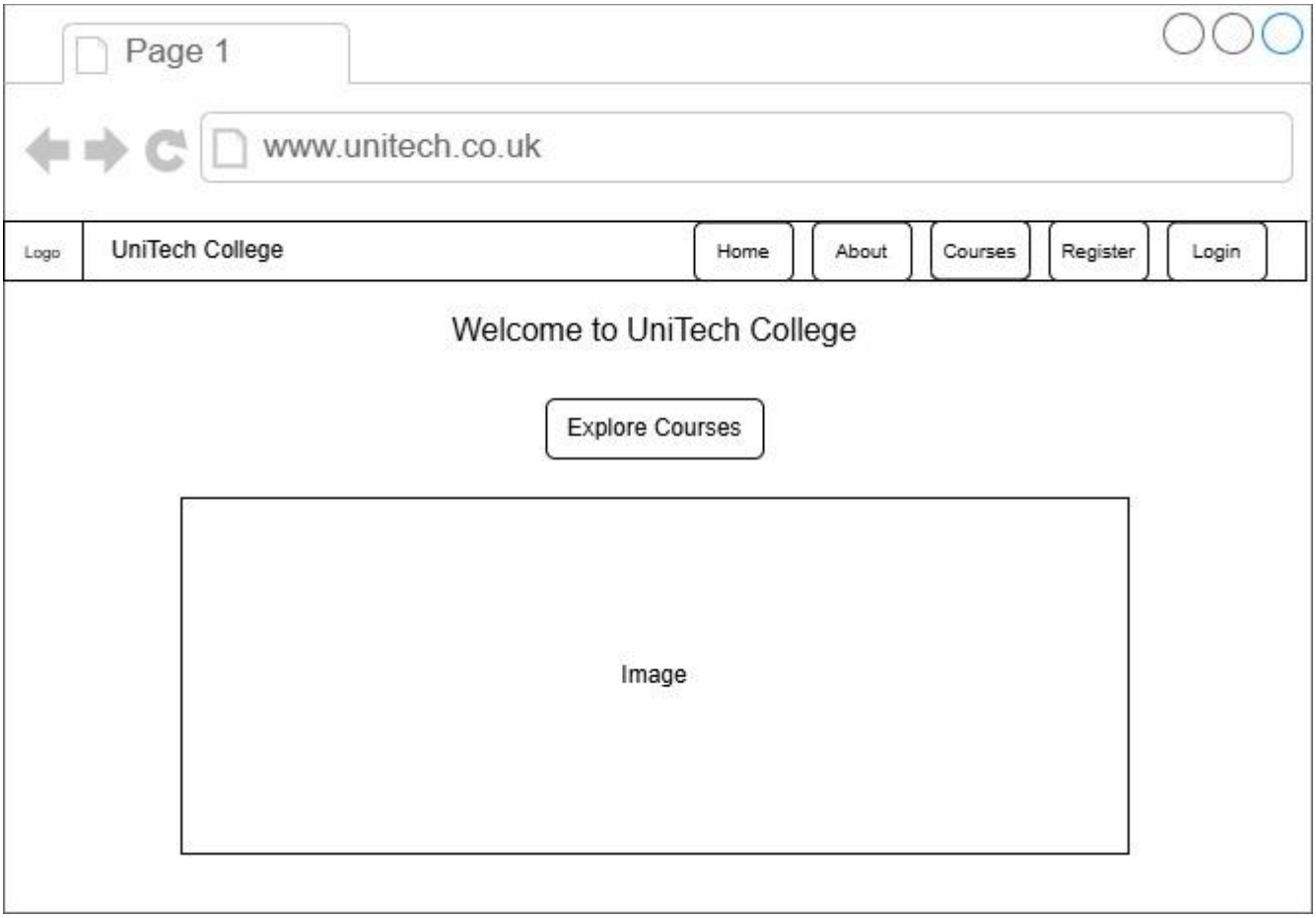


Figure 1 - Home Page

The homepage of UniTech College has a rather generic layout. It shows the fundamental design principles and specifications guiding the visual elements of the site. The top navigation bar has "Home," "About," "Courses," "Register," and "Login" to the left of "UniTech College." This gives a clean, straightforward framework for easy page navigation.
A kind greeting is displayed. The "Explore Courses" call-to-action button is located beneath the header that reads "Welcome to UniTech College." There is a space near the bottom where you can add an image to highlight a location where you can add visual content to increase participation.

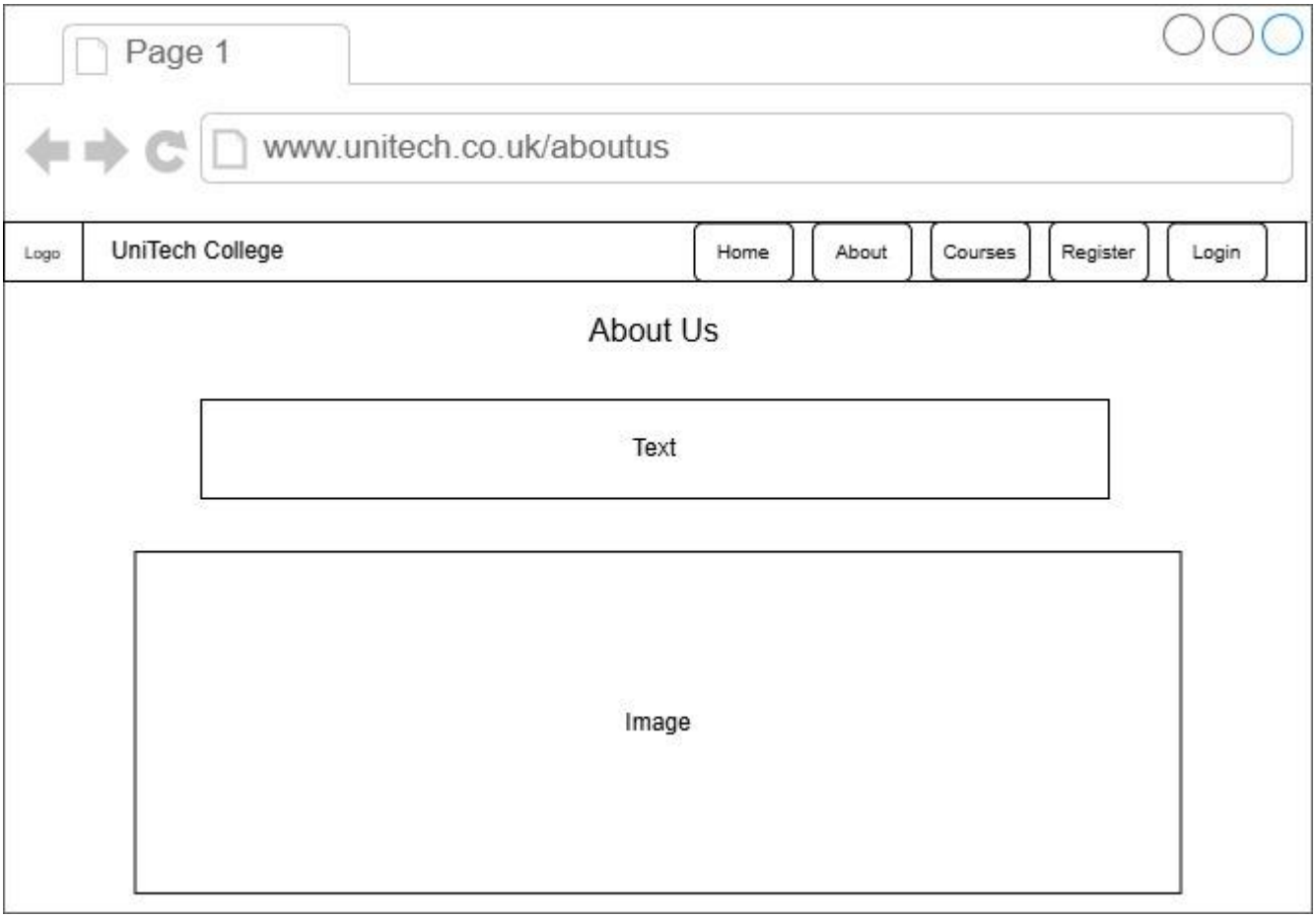


Figure 2 - About Us Page

The "About Us" structure is shown in Figure 2. The navigation bar is the same on all the other pages to maintain a consistent user experience. The headline "About Us," a block with the institution's descriptive text, and a larger image block that visually highlights its mission, goal, or culture are all included in the main section.

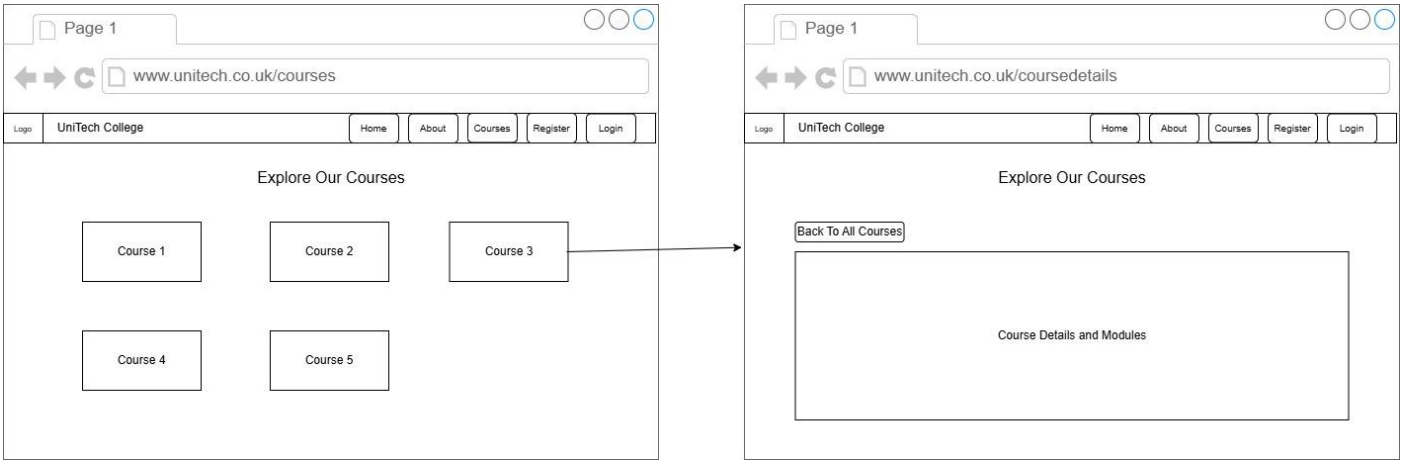


Figure 3 - Courses Page

The "Courses" page will appear as follows when you click the "Courses" button. Users are greeted by the title "Explore Our Courses" and a grid layout with five placeholders showing accessible courses. There will be a name, a synopsis, and a link to additional information for each course. The classic navigation bar remains exactly the same to maintain uniformity.

Figure 4 - Register Page

The purpose of the Register page (Figure 4) is to enable a new user to create an account. A form with "Full Name," "Email," "Password," and "Register" buttons is located in the centre. A user-friendly interface is enhanced by simple design and clear labelling. A seamless transition between pages is made possible by the consistent navigation bar.

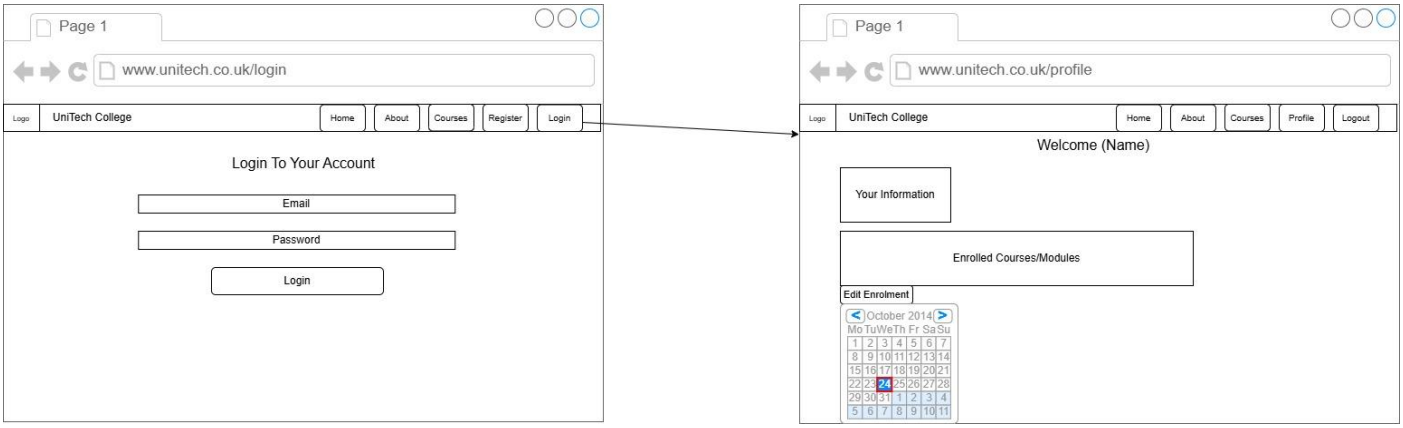


Figure 5 - Login Page

Figure 5 shows the login page where users can access their accounts. The entry area has a "Login" button in addition to "Email" and "Password" fields. The layout is similar to that of the Register page to maintain consistency in look and feel. The interface offers a simple entry point to the user's personalised dashboard. When a user logs in, the interface greets him by name and provides a quick summary of his personal information and the module or modules he has chosen to enrol in. A calendar widget, a "Edit Enrolment" button, a panel containing the user's personal data, and a "Welcome [Name]" message are the most noticeable components. "Profile" and "Logout" have also been added to the navigation bar.

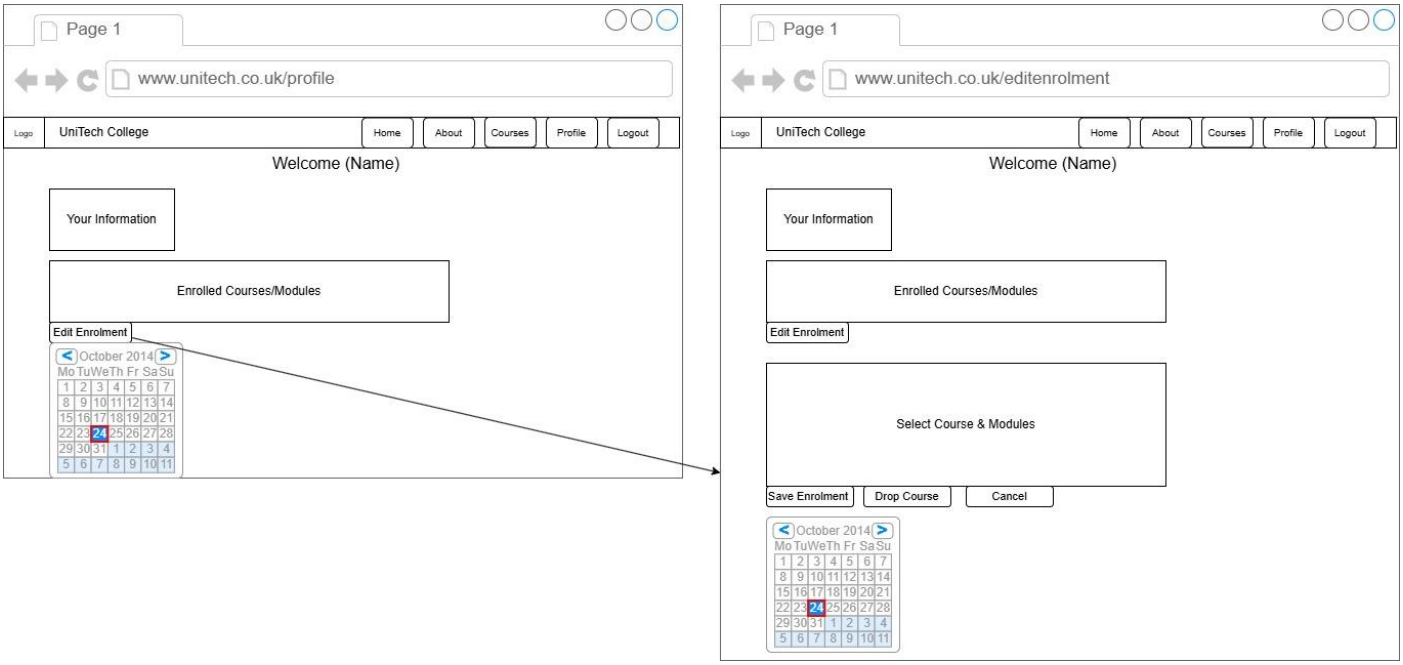


Figure 6 - User Dashboard Interface Post-Login

Figure 6 displays the UniTech College portal's "Edit Enrolment" and "Profile" pages side by side. The user profile interface is located on the left-hand side and comprises an overview of the courses and modules the user has enrolled in, a personalised welcome message, and a personal information section. This configuration gives users a summary of their participation in the study in real time. The "Edit Enrolment" page, which is on the right, allows the user to change the courses and modules they have selected. A form with buttons to "Save Enrolment," "Drop Course," and "Cancel" changes, a calendar, and the option to select a course and modules is used by users to edit their enrolments.

6.2.2 – Backend Logic

The UniTech College System backend is programmed to handle all the basic processing and decision-making operations that feed the frontend presentation. Written with ASP.NET Web Forms and deployed on a MySQL database, the backend does the essential tasks such as user registration, login authentication, administration of the user profile, and course/module registration. Server-side validation is used to ensure that email addresses provided in the process of registration conform to a valid format and that passwords satisfy the specified requirements for security. In addition, a login protection system is in place such that multiple failed login attempts consecutively will get the user's account locked for 15 minutes as a reinforcement of security.

Upon verification, the backend checks if the user has previously selected a course and its constituent modules. If not, they are prompted to finalise their registration before they have access to their dashboard. Each user's individual course and module selection is monitored by the system and stored securely using the appropriate database tables. Backend support is provided for course dropouts and personal info updates as well as editing registrations. Changes synchronisation with the database is sustained on a real-time basis for coherent and responsive experience. Finally, the backend is the operational core of the system that automatically integrates the database layer with the user interface in a seamless and safe platform.

6.2.3 – Database (with tables and relations)

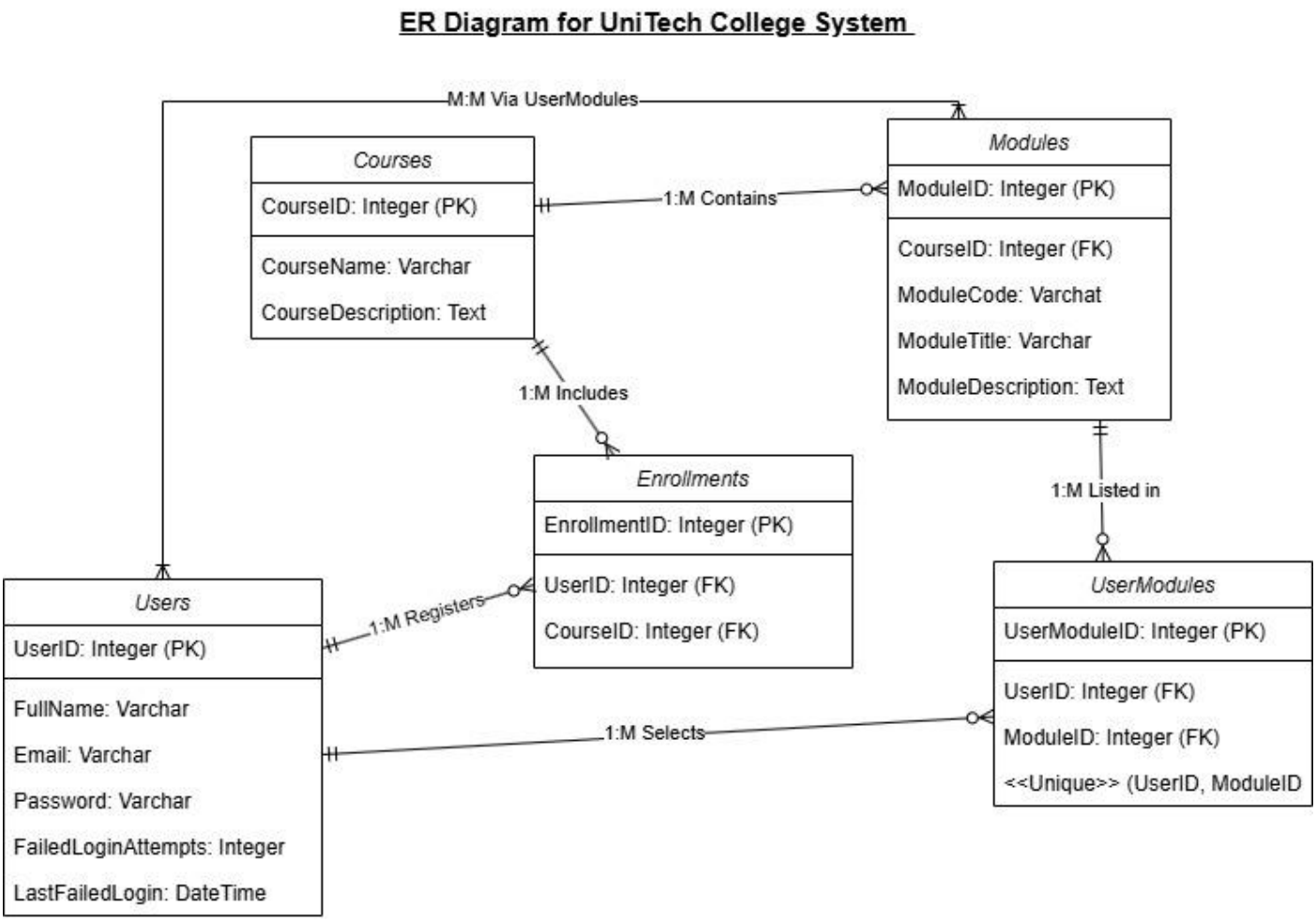


Figure 7 - ERD Diagram

The core part of the UniTech College System database is best shown by the following entity relationship diagram (ERD). It outlines the relations and relationship between the main organisations which are Users, Modules, Courses, Enrolments and UserModules. The diagram shows that a course has many modules (1–to–many), and users can enrol to courses and select different modules. Similarly, each module corresponds with a certain course, all of the enrolments and module selections have something to do with the users. A bridge entity, the UserModules entity serves to establish a many-to- many connectivity between the users and the modules. Every connection incorporates cardinality notation that is, "1 to Many" to enable straightforward and logical database structure and reasoning. Following project objectives, the design assures data security and enables crucial functionalities like course registration, module selection, login and profile management.

6.3 – Interaction Between Components

The UniTech College System's frontend, backend, and database relationships are built on a logical information exchange that makes communication and functionality simple. Server-side scripts written with ASP.NET are used to transfer user input from the frontend to the backend, whether the user is interacting with the frontend while completing the registration form or logging in. In addition to applying logic (such as verifying login credentials or course registrations), the backend verifies the input and communicates directly with the MySQL database.

Backend queries are used to safely execute all data requests, including adding new users, retrieving available courses, saving the enrolment status, and updating personal data. The backend performs the necessary processing after retrieving or updating the data, then properly responds to the frontend so that the user can see the most recent feedback or results on the screen.

The architecture's highly integrated structure allows the frontend to maintain its responsiveness and user-friendliness while the backend handles the database interactions and logic in the background. Both rely on one

another to function properly and create a dependable and effective system that serves as the foundation for the entire UniTech College website.

6.4 – Design Principles

The UniTech College System was developed with long-term efficacy and flexibility in mind, giving priority to fundamental elements like maintainability, modularity, and scalability. Since the system was designed to be extensible (scalable), future additions of modules, users, courses, and functionalities could be made without necessitating a total redesign of the existing framework. It is now possible by structuring the codebase and database design in a way that is both organised and flexible.

We have achieved modularity by dividing the problems into discrete parts, such as self-contained ASP.NET pages for course selection, login, and registration, and dashboards that make it simple for us to test each of these features separately. Because we were able to work on different modules without interfering with one another, this also allowed for collaborative development among the team.

Maintainability was woven into the very structure of the system through the use of reusable code, consistent coding standards, and naming conventions. This makes it possible for any later developer, or even a member of our own team, to read, alter, and debug the system efficiently when they return to the project.

7 – Security and Privacy Concerns

UniTech College System combines several fundamental security and privacy elements into its architecture and design in order to safeguard user data and guarantee the integrity of its website.

7.1 – Authentication and Authorisation

A login system was designed to stop unwelcome users from gaining access to personal panels and course sign-ups. Password hashes with different salts are stored securely in the database using SHA-256, therefore they are useless even in the case of a breach. Sessions are managed using conventional ASP.NET session controls and automatically logged off following a designated inactivity level to prevent threats from shared or public PCs. Future iterations will also incorporate role-based access restriction, so restricting admin-only tasks (such as updating student data) to authorised personnel.

7.2 – Data Protection and Encryption at Rest

The system handles sensitive information including emails, names, and hashed passwords, therefore GDPR compliance was incorporated in from the start. Giving permission when a user registers up makes ensuring that no information is saved that isn't essential for operations. A normalised model is utilised by the system to stop exposure and repetition. All additional sensitive data is encrypted at the database level using AES-256. By default, passwords are hashed, which makes them extra secure. Encryption keys are saved securely in the configuration files of the app, which are secured by web.config encryption in ASP.NET and server level file security to block unauthorised access.

7.3 – Input Validation and SQL Injection Prevention

The server-side sanitisation process makes advantage of native ASP.NET validation rules. Parameterised searches also serve registration, login, course and module selection, and other uses. This helps to greatly lower SQL injection vulnerabilities. Unsuccessful validation efforts are noted for reporting; the system rejects aggressive or malformed user input with specific issue information.

7.4 – Audit Logging and Intrusion Detection

The UniTech College System's basic audit logging capability notes important events including login attempts, profile changes, and registration updates. Every log entry notes the date, activity, and user ID, so enabling managers to track back data as required. Multiple consecutive failed login attempts cause account lockout, so preventing brute force invasion. We note failed login attempts. Examining the logs on a regular basis helps one find odd usage patterns, which offers a strong basis for more advanced threat detection tools in next versions.

7.5 – Ethical Considerations in Handling User Data

Following ethical criteria for software engineering, the program design supports openness and user control. Personalised websites give users total control over their data, so preventing needless data collecting wherever possible. Clear privacy policies made available at registration help to build trust and informed consent by themselves. Only operational employees with a justified need to know have access to company data, personal information is not shared with outside entities.

8 – Code Reliability

Code reliability is an integral part of the UniTech College System development that ensures consistent functionality and users' faith. During the entire course of the project development process, coding technologies have been used towards predicting, localising, and error handling as well as maintaining system performance.

8.1 – Fault Avoidance

In a move towards avoiding run-time faults, defensive coding methods have been employed. Pre-checking before processing that variables contained expected values and that user input conformed to specified specs has already taken place. Null checks, conditional tests, and validation rules have all been employed methodically towards the prevention of any errant behaviour of the application. Try-catch blocks in ASP.NET were used for structured error handling in order to handle run-time exceptions gracefully and not crash the application. Users receive meaningful feedback whenever there is an error and the system remains stable instead of crashing.

8.2 – Input Validation

The area of input validation was one of primary importance because it has a direct bearing on both reliability and security. All of the user input forms, i.e., login and registration forms, were both client and server side validated. Validating the type of the data, the length of the data and the format of the data makes sure that meaningful and valid information is passed. It reduces the incidence of problems with the data as well as protects against injection attacks and silent failures due to malformed inputs. Use of the validation controls in Web Forms also played a role in enforcing required fields as well as constraints on the data prior to backend communications.

8.3 – Fail Safe Defaults and Logging

The system adheres to a fail-safe default approach such that if there is unforeseen user behaviour or system malfunction, the system goes into a safe and stable configuration. For example, if authentication fails for a user or the session times out, access is blocked and the user is sent back to the login page. While basic logging was employed for debugging purposes throughout the development process, the system is capable of being extended in the future with the incorporation of structured logging tools specifically for the monitoring of faults and tracing of user interactions that will improve long-term reliability and fault diagnosis.

9 – Testing

Comprehensive testing was performed across the entire development cycle of the UniTech College System for the assurance of its functionalities and a seamless and safe experience for its users. A range of testing techniques were used, with the focus mainly on functional testing with the aim of testing the key features.

9.1 – Types of Testing Performed

9.1.1 – Unit Testing

The unit testing components such as user login logic, registration forms, and how the data is retrieved were test-driven in isolation separately ensuring that they all worked as desired. The unit tests helped in catching the early logic bugs and ensuring that individual components behaved as desired before the components were merged.

9.1.2 – Integration Testing

Then we performed tests of the way various components communicated with one another. For example, after creating a user, we tested the login module's ability to successfully authenticate the recently entered credentials and set the user up on the dashboard, ensuring that there was proper flow of data between database and modules.

9.1.3 – User Acceptance Testing (UAT)

The end-users consisting of team members in the role of student users used the application to ensure that all the primary functionalities like course registration, updating personal details, and looking up the calendar acted as they should.

While the current implementation focused solely on students, we also discussed and noted for future development plans to support staff and admin which they can login and register a account with UniTech College. These features are currently not developed as of yet or even tested but we acknowledge that during the user acceptance process as important features to provide full administrative and academic access within the system. These different types of roles would require different levels of permissions, such as managing courses, modules, overseeing enrolments, changing student details, updating students recording, changing courses and modules and seeing if the courses and modules are still available.

The feedback from this testing was used for refining the UI and streamlining the navigation. Although staff and admin privileges were not present in this version, our plan still remains a key part of the system's future roadmap to ensure full coverage of all users regardless of if they are a student, admin or staff are all equipped to within a university setting.

9.1.4 – Functional Testing

The most comprehensive form used was functional testing. The purpose of this was to ensure that the system behaves as per the predicted result of user input. The following is a summarised table of the most significant functional tests carried out.

Test Case ID	Function Tested	Description	Expected Result	Actual Result	Pass/Fail
FT001	User registers on the system	The user will input their own details to create their own account	Once the user has completed their registration it should take	Expected result	Passed

			them back to the login page		
FT002	Logging in with authentication	Logging in the correct credentials	The system will take the user to the dashboard page	Expected result	Passed
FT003	Incorrect login details	Logging in with the incorrect details	An error should pop up on the login page until you input the correct logging	Expected result	Passed
FT004	Viewing the courses	See courses on the courses page	There are 5 courses and you have the option to view all 5 of them	Expected result	Passed
FT005	Selecting the courses	You'll be able to click on the courses and select them.	The modules that are linked to the courses can be shown and can be selected under the module.	Expected result	Passed
FT006	Enrolling for modules and courses	Select the course and the modules underneath the course then click save enrolment	The modules and course should be saved on that user's profile page.	Expected result	Passed
FT007	Updating users personal details	Once the users changes details, it should save	The details when updated should be saved and stored inside the database	Expected result	Passed
FT008	Logging out	Click on the logout button that's visible in top right hand corner	The user is taken straight back to the homepage	Expected result	Passed
FT009	The dashboard should display the data	Log in and access the dashboard	Users should be able to see there dashboard with things such as enrolment's and Calander	Expected result	Passed
FT010	Attempting SQL injection home page	Inputting the SQL code in the login field	When the input is rejected then it means that the authorised access is denied.	Expected result	Passed

Table 2 - Functional Testing Table

9.2 – Test Cases with Examples

Below are a few examples of specific test cases and the results obtained

Test Case: FT002 – Login Authentication
Input: Email: shyamdattani18@gmail.com, password: Password1%

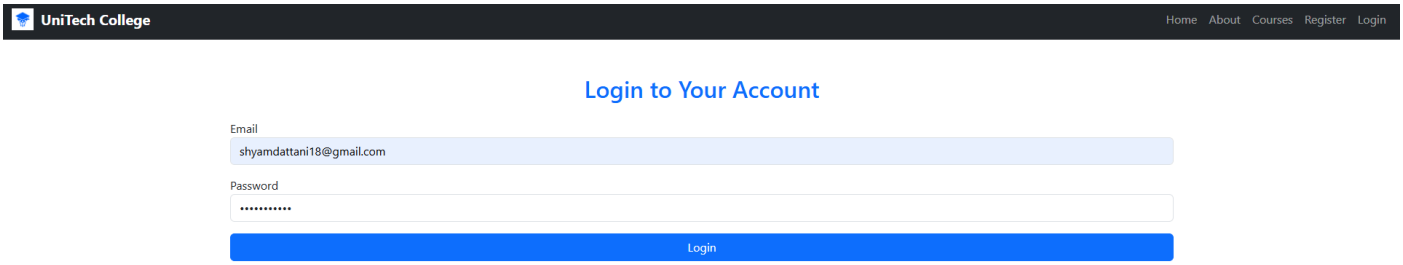


Figure 8- Login Page

Expected Outcome: Redirects to the dashboard page

Actual Outcome: Successfully redirected to the dashboard page

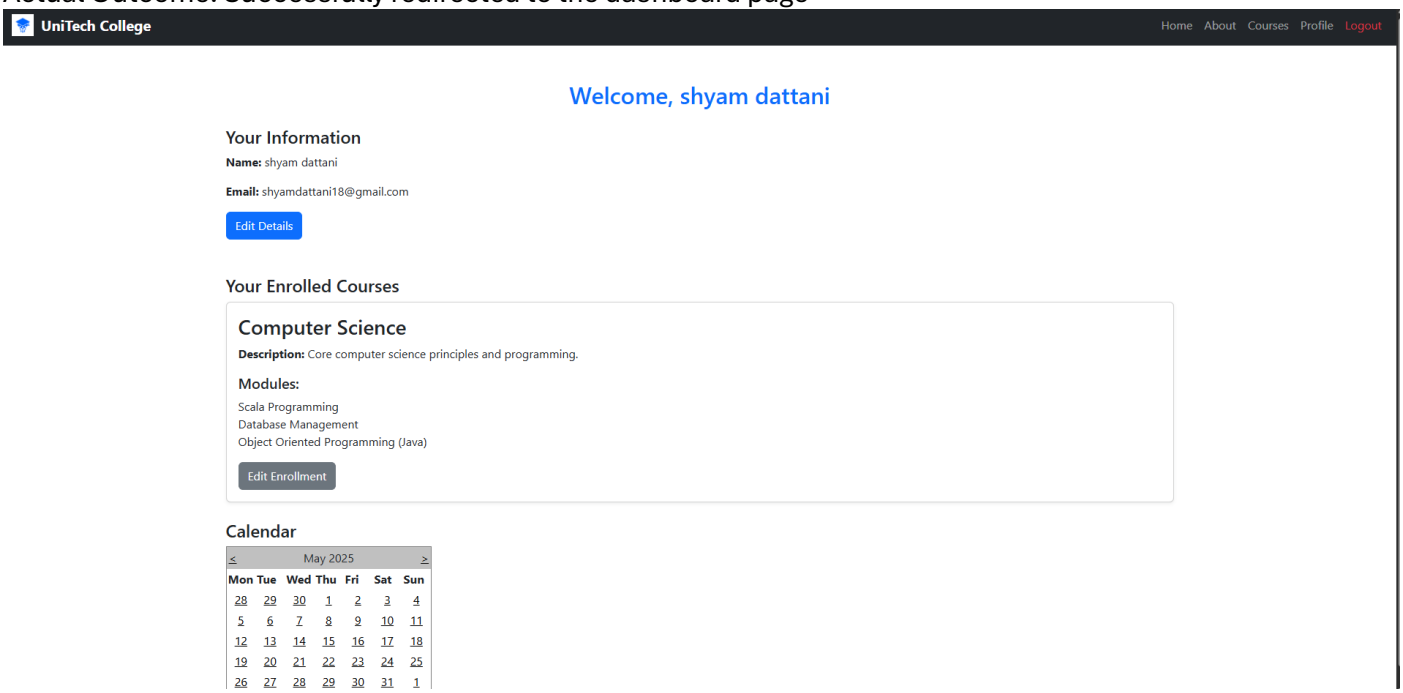


Figure 9 - User Dashboard

Result: Pass

Test Case: FT006 – Module and Course Enrolment

Input: Select Computer Science, Select exactly 3 modules, press the “Save Selection” button.

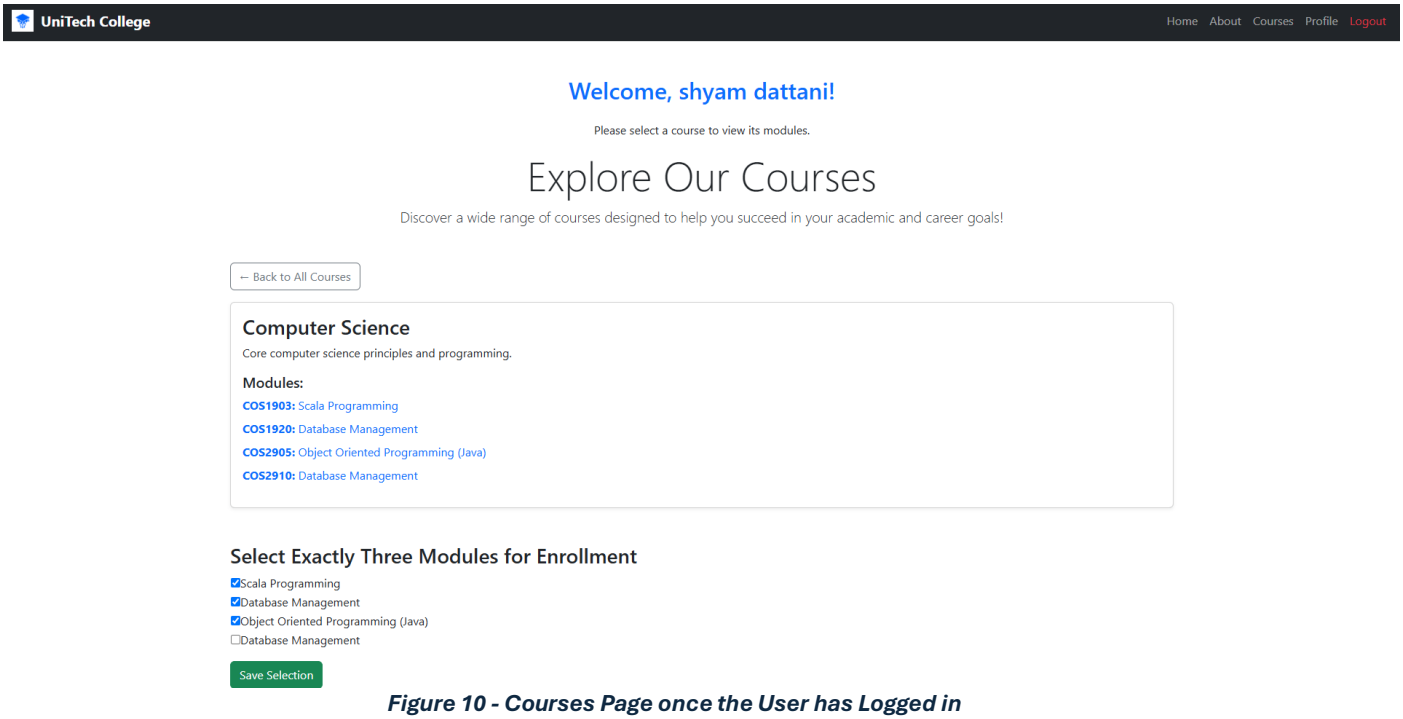


Figure 10 - Courses Page once the User has Logged in

Expected Outcome: Computer Science and the three modules that the user has selected should appear inside the dashboard under the “Enrolled Modules” section.

Actual Outcome: Modules and course successfully listed

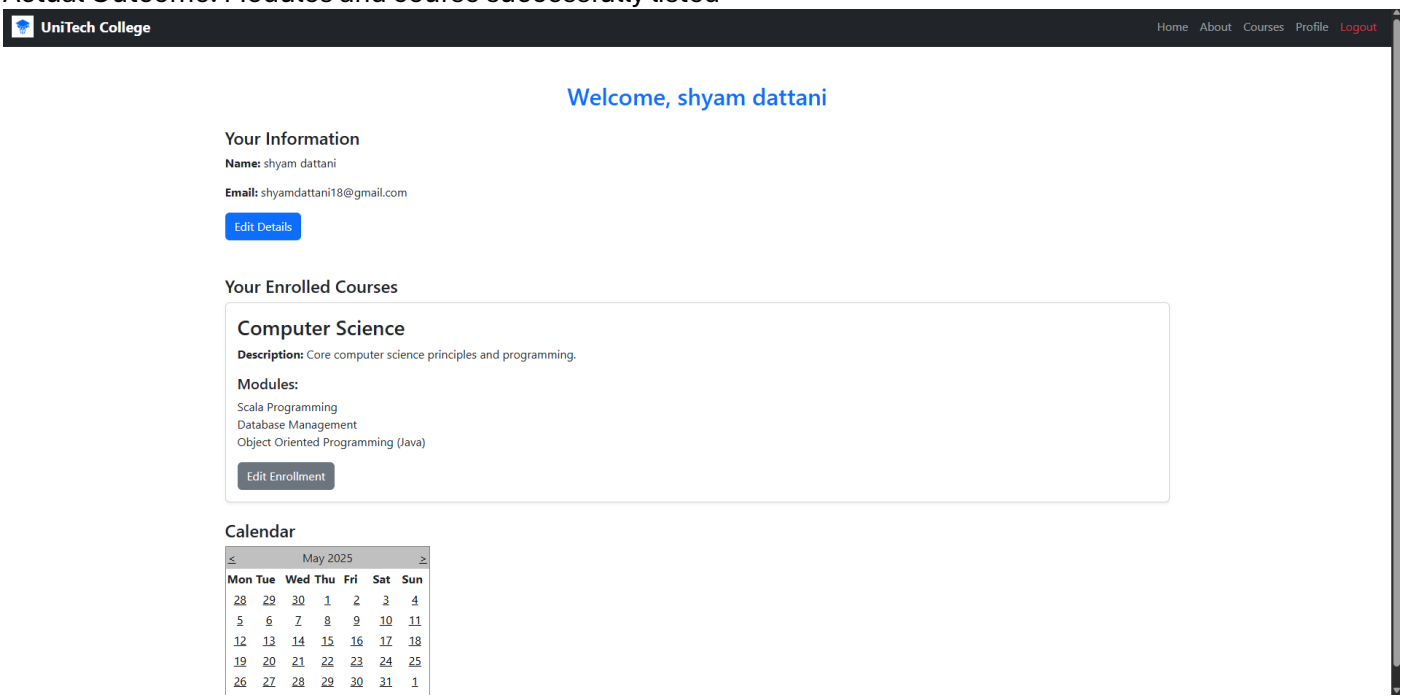


Figure 11 - Dashboard Page of the User once they have selected the course and modules

Result: Pass

Test Case: FT010 – SQL Injection Attempt
Input: Username: 1, password: Password1

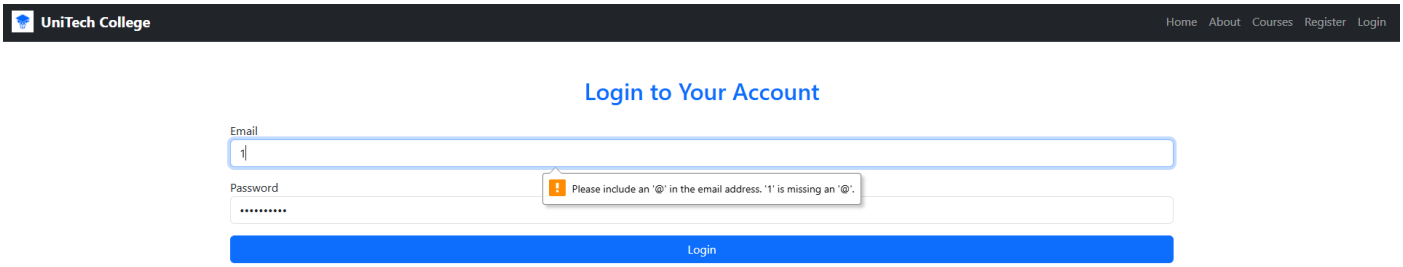


Figure 12 - Login Page

Expected Outcome: Login attempt is blocked.
Actual Outcome: Error message shown, access denied

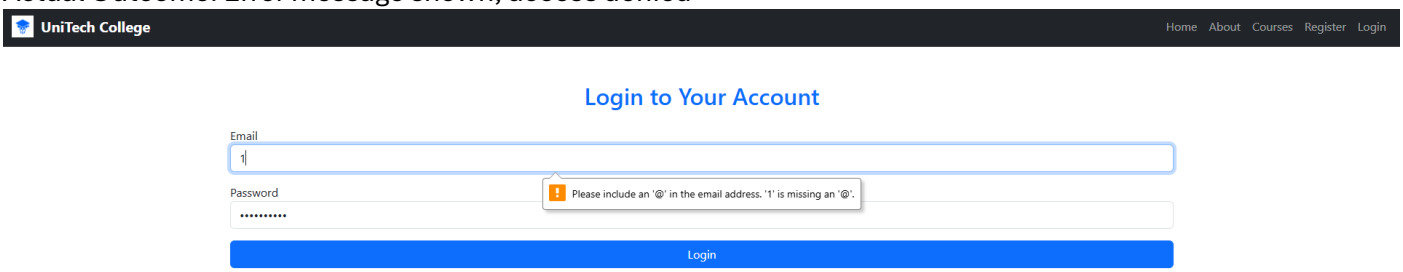


Figure 13 - Login Page with the error message

Result: Pass

10 – Product Prototype

In this section, we will be showing part of the product prototype where a user can register, login, course registration, and they can view their own account.

10.1 – Register

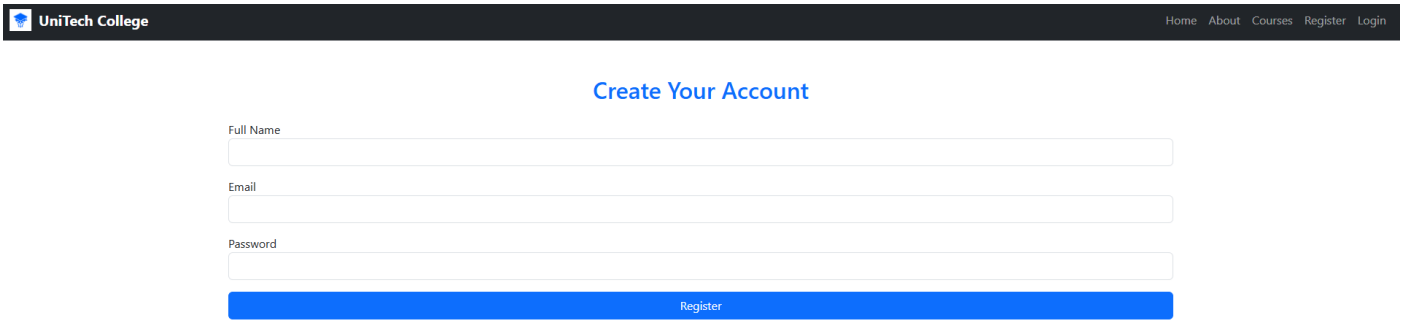


Figure 14 - Register Page

This is what the user will be greeted once they press the “Register” button.

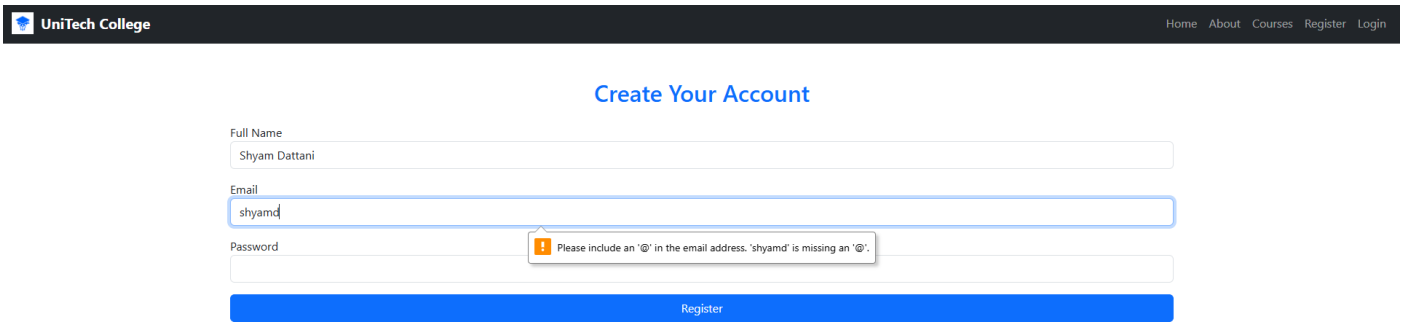


Figure 15 - Register Error

This is what happens when a user doesn’t enter a correct email address. A error message will appear.

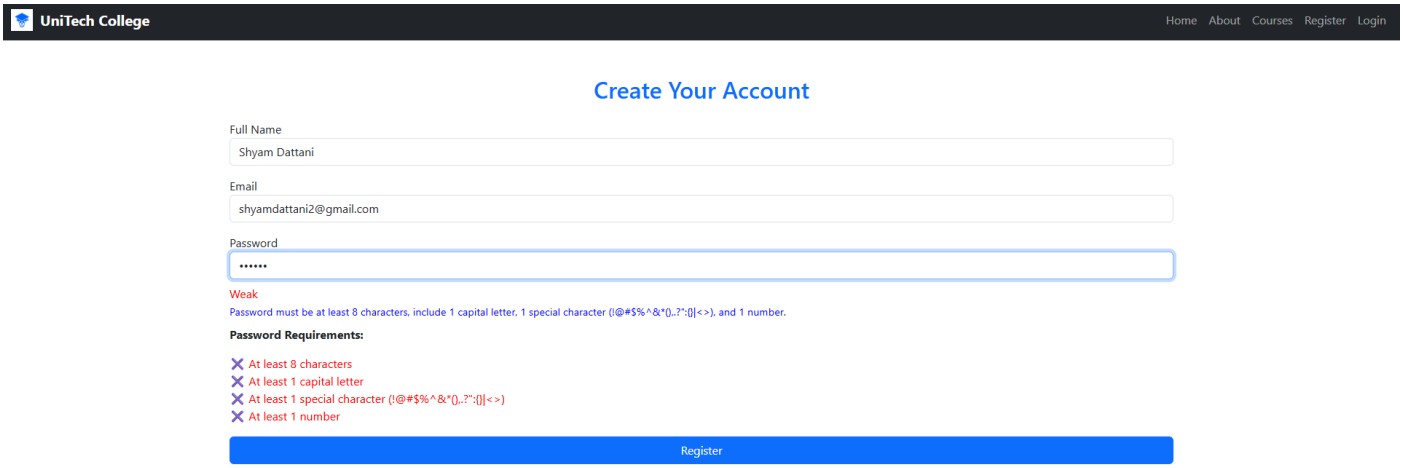


Figure 16 - Register with password error

This is what happens if the user doesn't follow the password requirements they need to follow the password requirements in order to go to the next stage of the application.

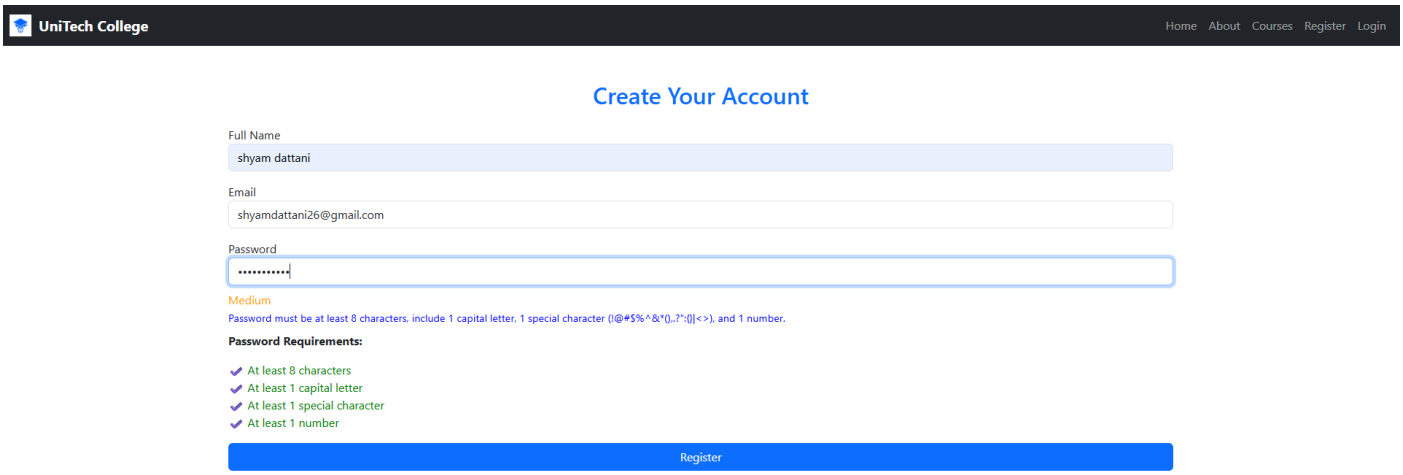


Figure 17 - Registration Page Completed

This is how it should be in terms of the registration process. The user must enter there full name, email address and password, but the password must meet the criteria outlined and then only then the user can create a account.

10.2 – Course Registration

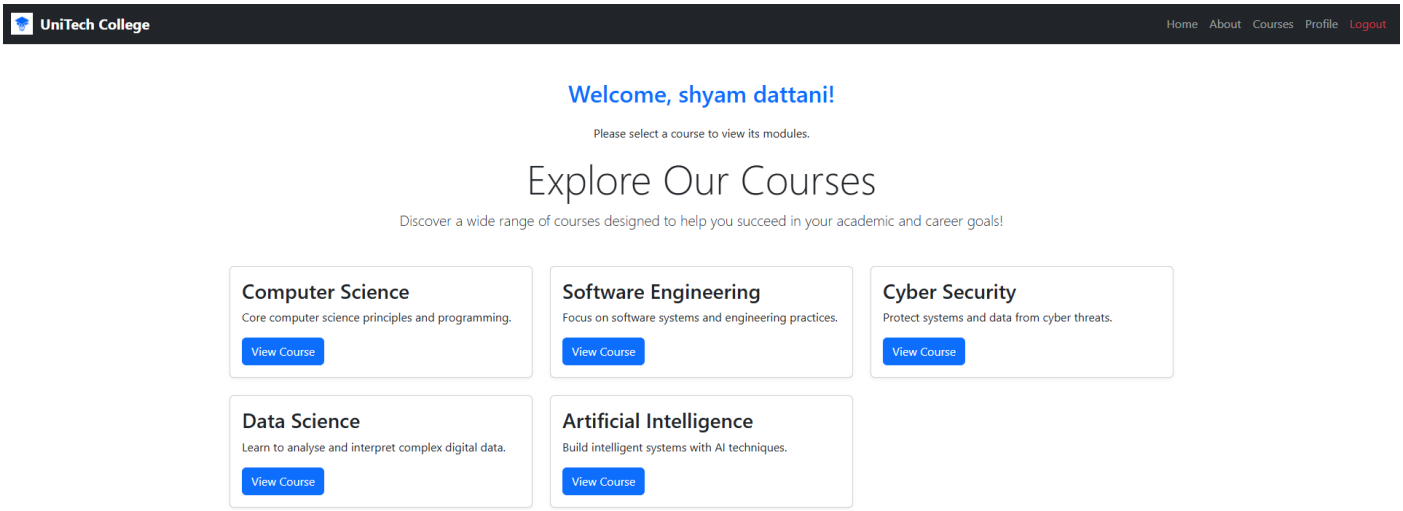


Figure 18 - Course Registration Page

Once the user has successfully created there account they will be taken to this page (Explorer Our Courses) They can then click on a course they wish to view.

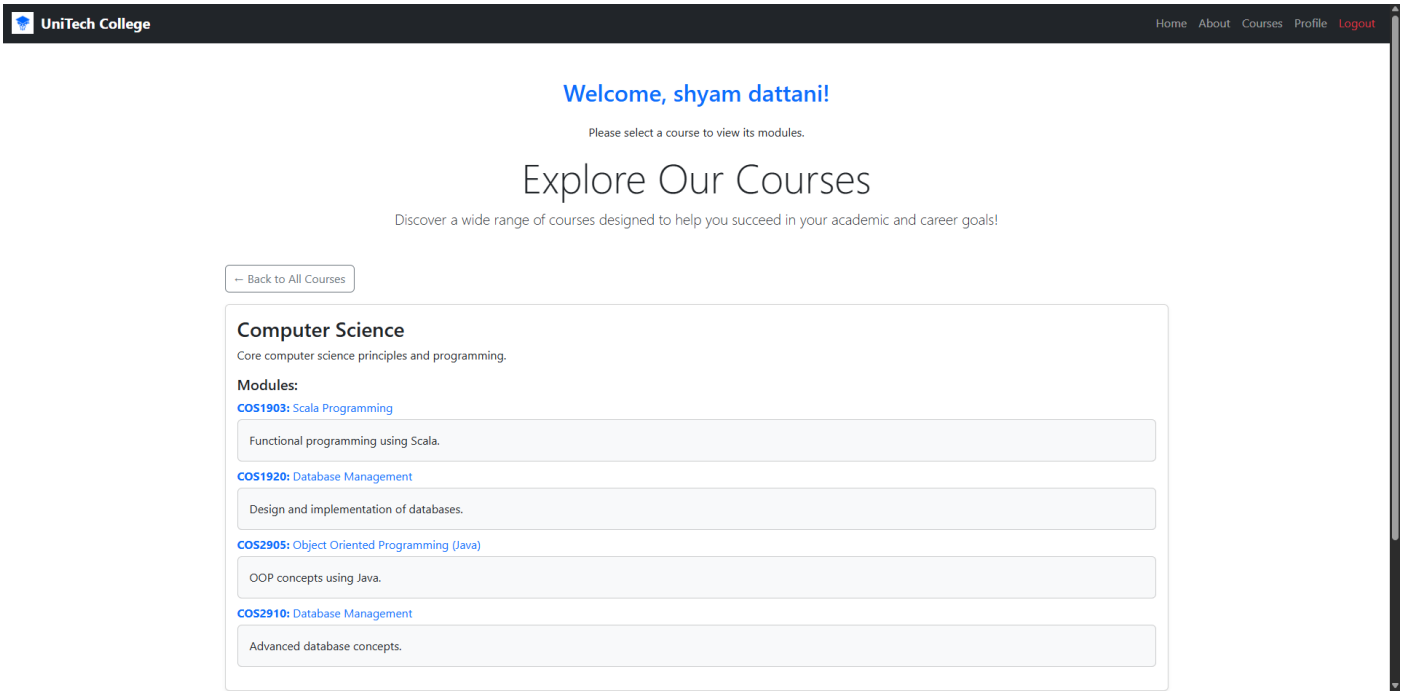


Figure 19 - Explore Our Courses Page

As you can see above, the user has clicked on Computer Science, they can then click on the modules they wish to view.

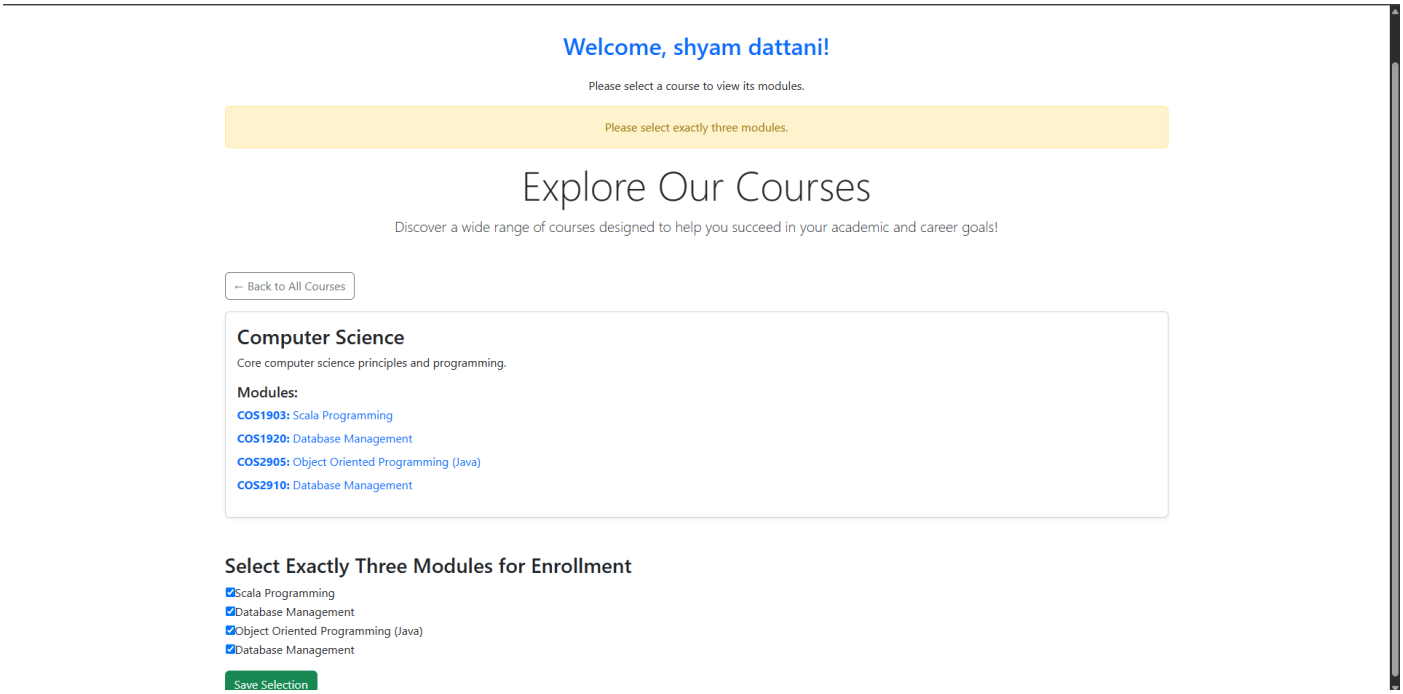


Figure 20 - Error for selecting 3 or more modules

If the user selects more than 3 modules than an error will display saying “Please select exactly three modules”

Figure 21 - Selecting exactly 3 modules for enrolment

After they have viewed the course and modules they need to select exactly 3 modules for enrolment and then press “Save Selection” which will then redirect them to their own account.

10.3 – Profile

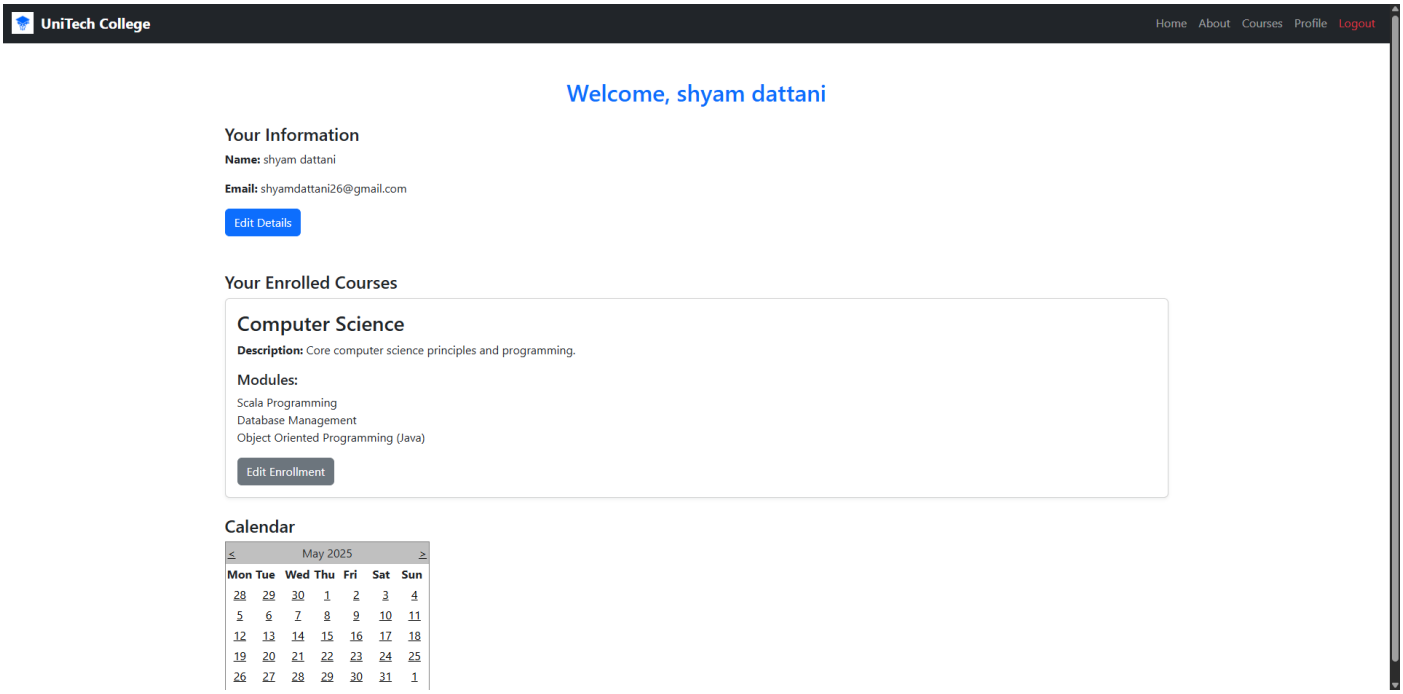


Figure 22 - User being redirected to a new interface

This is the interface the user is taken to after they have selected there course and modules they would wish to do.

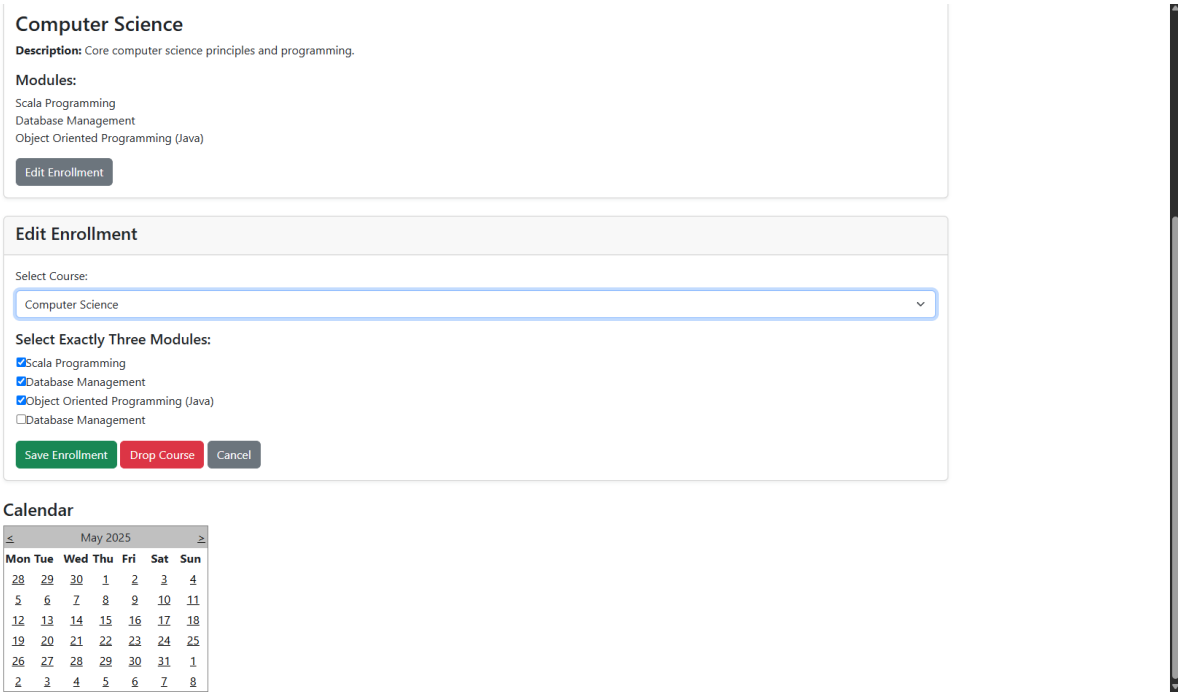


Figure 23 - Editing personal details and saving enrolments

If they click on edit details, they can edit there personal details and the changes will be saved back to there account, if they click on "Edit Enrolment" they will be presented to select the course they wish to change to and the modules, they can either drop the course completely, change the course, edit the modules, save enrolments which saves the new changes inside the account and cancel which cancels the current selection.

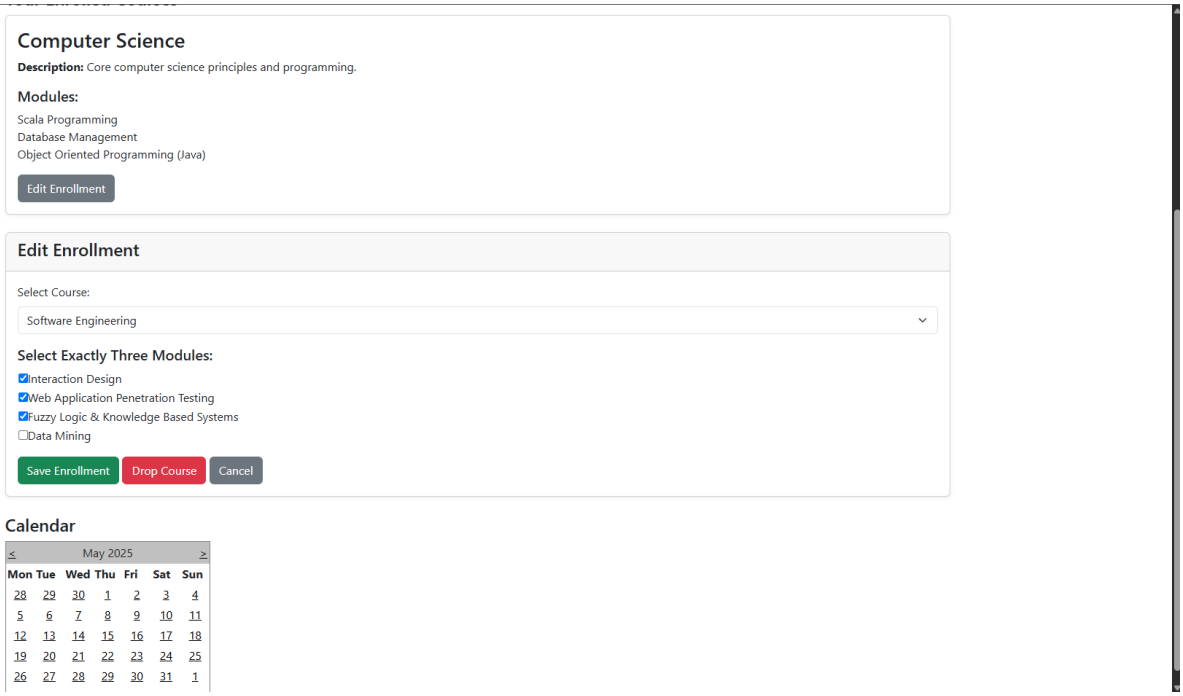


Figure 24 - User changing courses and modules

We will now show you if the user wishes to change the course and modules from Computer Science to Software Engineering, the user clicks on the select a course, they will be presented with a dropdown list of the courses available, they then select the course which they want to and then the 3 modules they wish to do for that specific course and press the save enrolment so that it saves directly onto their account.

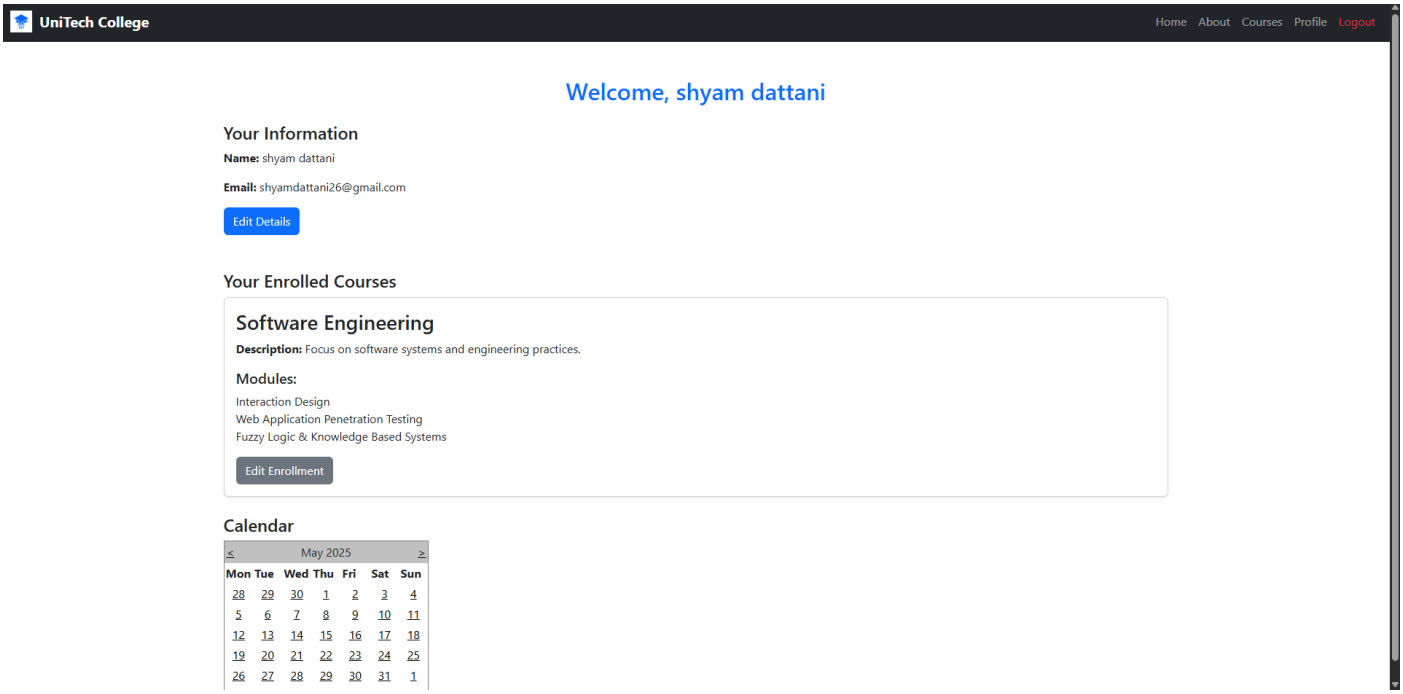


Figure 25 - Evidence of showing the changed modules and courses

As you can see from the above screenshot, the course and modules have changed and they are displaying inside the user's account.

10.4 – Login

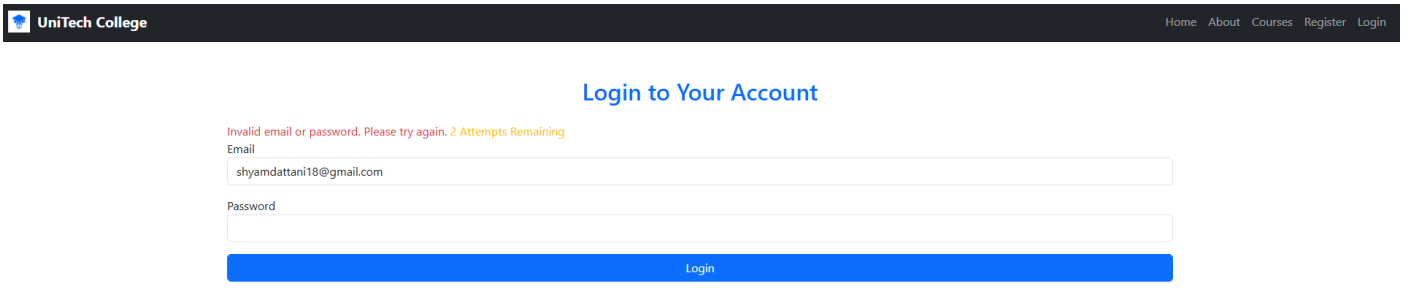


Figure 26 - Incorrect and correct credentials for users

If the user enters their login credentials, incorrectly they will be presented with "Invalid email or password. Please try again. 2 Attempts Remaining." The system in total has 3 attempts after the 3rd attempt it locks out the user for 15 minutes and they are locked out of there account. If the user however enters the correct login details, then they are presented to their profile if they have selected the course and modules, if they haven't then they are redirected to courses page where they will need to select their course and modules they wish to enrol too.

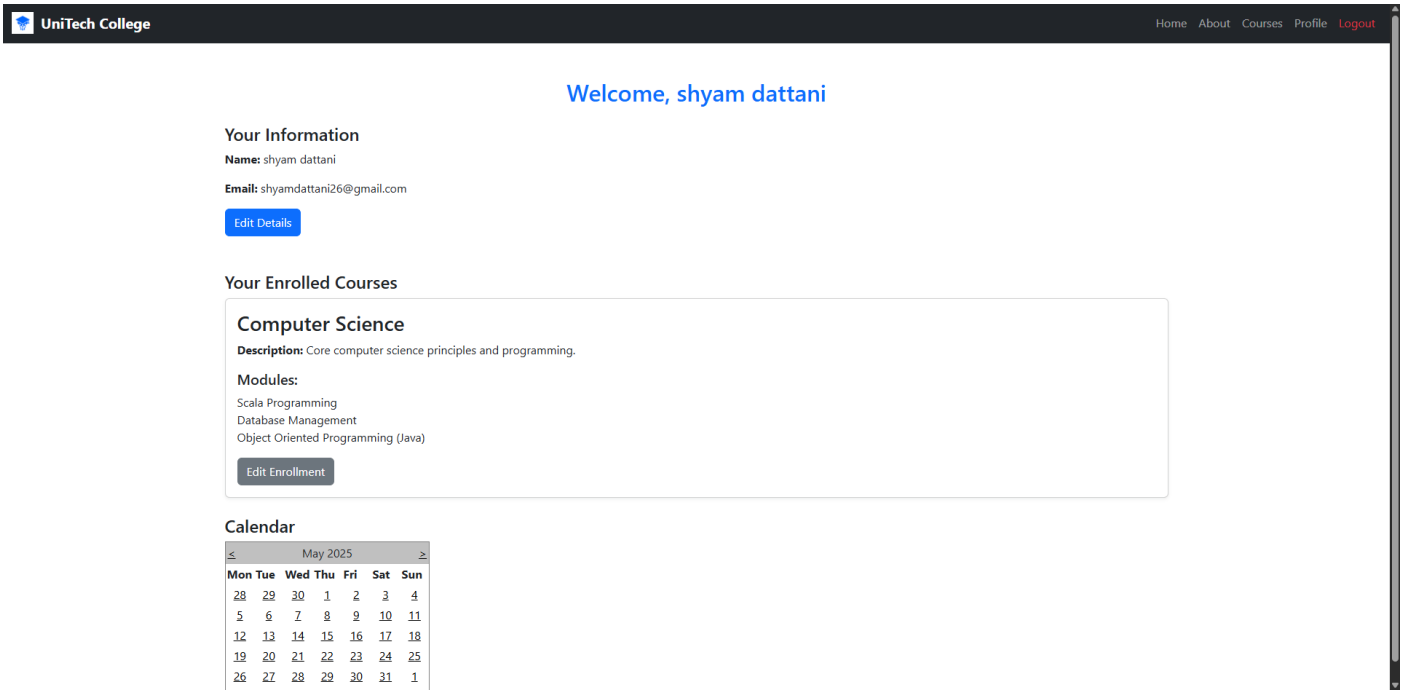


Figure 27 - User successfully logged in

This is what happens once the user has successfully logged into the system.

11 – Reflection

Being part of a five-person team that has built the UniTech College System in this course provided an academic as well as career-enhancing experience. My contribution to the team, challenges we faced, skills I developed, as well as the ways in which agility practices impacted our collaborative process, all feature in this reflective account.

11.1 – My Role in the Group

I was both lead developer and project coordinator for the team. My role mainly involved taking care of the technical aspects of our solution and having our system architecture fulfil the requirements of the task along with the real needs of a college registration system. In addition, I created a linkage between technical implementation and coordination of the group in a manner that would make sure tasks were assigned correctly and people understood what they needed to accomplish.

11.2 – Tasks I Completed

My work focused mainly on front-end and back-end using the ASP.NET Web Forms framework along with a MySQL database hosted and managed with MySQL Workbench. I developed core pages like courses.aspx, dashboard.aspx, login.aspx, and register.aspx and implemented features like course selection, enrolment in a module, as well as profile updating.

Backend-wise, I handled user registration, password management, SQL query implementation, as well as storage. I oversee making contacts in the database secure as well as efficient, for instance, defining main and foreign key constraints as well as fulfilling input validation processes.

In addition, I played an integral part in our project report specifically the system architecture, backend functionality, and test strategy ensuring that our final report was sound technologically, well-written, and aligned with the brief.

11.3 – Challenges Faced and How I Solved Them

My biggest challenge was integrating front-end interactions with MySQL and ASP.NET. There were issues in maintaining the data after postbacks as well as in processing user records with proper queries. I rectified these issues by refactoring the data-access layer cautiously through parameterised queries along with refactoring some portions of code towards proper handling of sessions as well as retrieving the data.

Another problem we faced was getting our group members' programs to function with one another, particularly when merging code of multiple sorts or distinct logic. To tackle this, I recommended a common coding style from the very beginning and that we embrace the adoption of Git version control, as well as periodic brief virtual meetings, to maintain uniformity and avoid any dispute during checks in.

11.4 – Skills and Knowledge Gained

From a perspective of session tracking, server controls, as well as talking to a MySQL workbench, I now have a better grasp technically of ASP.NET. I learned to design a database more critically as well as to ensure referential integrity as well as constructing efficient queries for user and enrolment related data.

On the soft skills side of things, I built up my skill of communicating technical concepts with peers of diverse strengths some more UI-oriented in their expertise, others more oriented toward documentation. This reconfirmed the value of patience, lucidity, and collaborative problem-solving in a group.

Furthermore, I have acquired conceptual as well as practical experience in security practices of input sanitisation, SQL injection protection, as well as session handling. I am sure that these skills can assist me in any profession I pursue which involves software development.

11.5 – Working in an Agile Environment

Throughout our project, we adhered to core agile principles. Even in an educational setting, we worked towards stand-ups at the beginning of each work session to discuss our successes and challenges. We created a Kanban-style task board and worked in sprints to display the list, work in progress, and completed tasks.

Agile mindsets changed our perspective from merely using tools to how we perceived changes and failures. When a module feature failed during Sprint 4, we reassessed our goals, reset our sprint tasks, and made some minor adjustments. The focus on modest performance and team ownership allowed us to continue growing even when things did not go as planned.

I also learnt from agile that we need to value early feedback over perfection. We kept looking at things as they were completed, not after, to minimise changes and get bugs sorted out early. These methods left a lasting impression on improved attitudes towards collaboration, flexibility and continuous performance which I believe are all crucial components of real development projects and they will really help us in the real world.

11.6 – What I Would Do Differently Next Time

If I were to repeat a similar project, I would incorporate regular code review sessions following every sprint. Although we exchanged updates and merged branches using Git, we never took time out to review each other's work thoroughly. Regular use of a peer review cycle would have detected logic problems sooner and brought us uniform quality and consistency across the code base.

I would also recommend more extensive user testing with a larger participant group. Our own was limited to a few users, so we might have missed edge-case behaviour, not to mention access for disabled people. Having tested a large user group would have yielded more informed feedback and reinforced design decisions more comprehensively.

In the future, I would also like to expand the simple calendar view, where students can view important academic dates in easy-to-read form. While our current prototype simply displays a placeholder for doing so,

future development could make it so that students could view impending events or assignment due dates or scheduled timetables directly from within the system dashboard.

Lastly, I would focus on implementing role-based access to various types of users aside from students. As our system now is limited to supporting students alone, integrating admin and staff roles earlier on will enable proper testing and fine-tuning of system permission. These roles will have capabilities of managing course information, managing enrolments, and keeping authorised records. These are the features that are needed to make our system a more complete and effective college system.

11.7 – Conclusion

In summary, this project was a turning point in my professional growth as a developer. Aside from the technological success, it taught me better teamwork, problem-solving in pressure situations, and most importantly, respect for designing systems that not only perform but also ensure they are both safe, supportable, and user-friendly.

12 – Appendix

In the following appendices you will see the code and the database schema that we have created to make our website along with detailed annotations of the code and database schema.

Appendix 1 – Solution Explorer

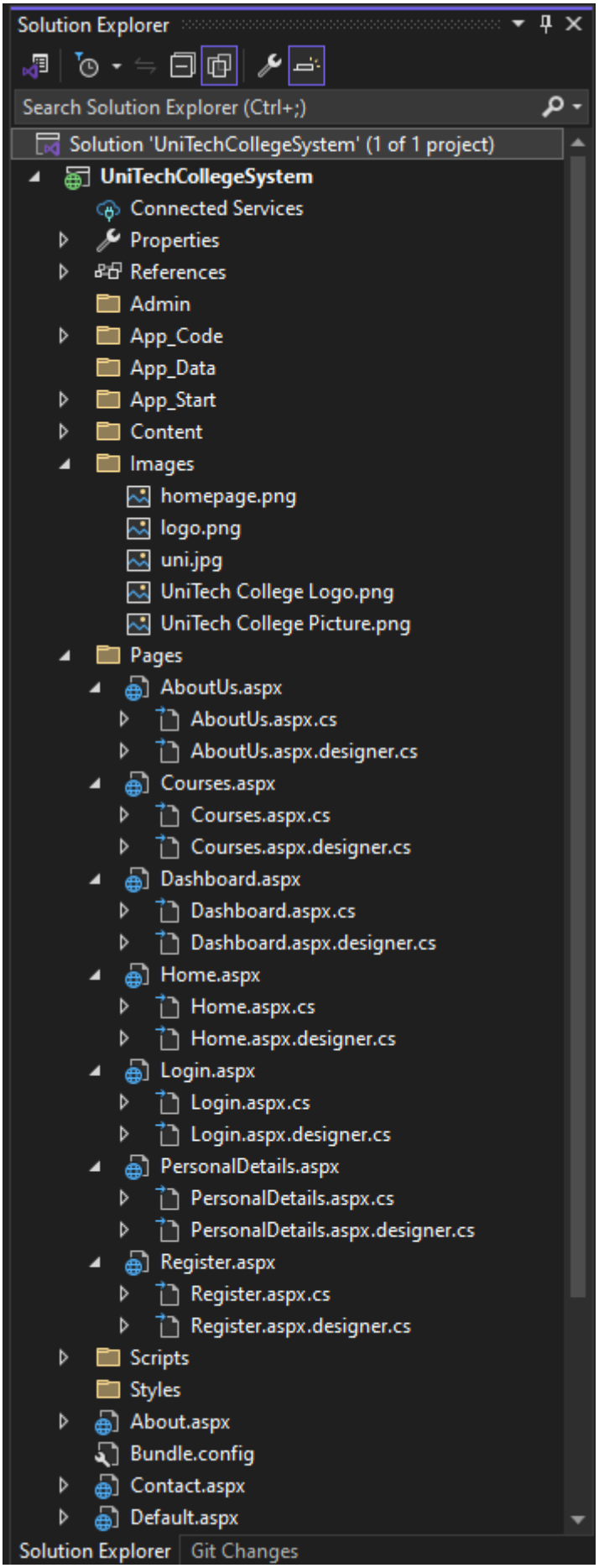


Figure 28 - Files for solution explorer

The diagram above shows the Solution Explorer window of a Visual Studio project under UniTech College System. For a web project most likely created using ASP.NET, this is a directory structure. "Content" includes static files; "Images" includes logo.png and homepage.png; "Pages" includes Home.aspx, AboutUs.aspx, Courses.aspx, Login.aspx, and so forth. Along with a Bundle.config file to manage the resources, there are "Scripts" and "Styles" directories for scripts and styleheets.

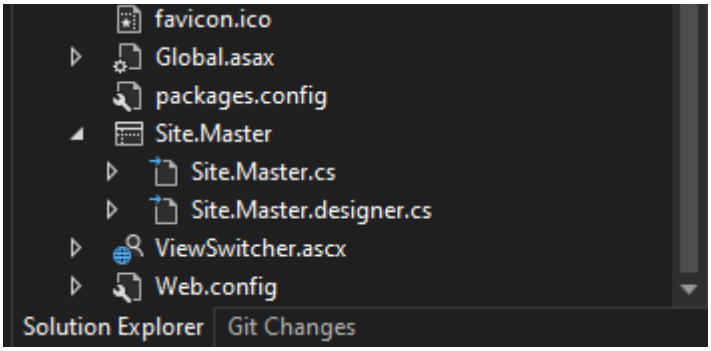


Figure 29 - More files for solution explorer

This screenshot shows another selection of the Solution Explorer inside the UniTechCollegeSystem. It includes key files such as favicon.ico for the website icon. Global.asax for application-level events, and Site.Master for it's design and code files for shared page layout. The ViewSwitcher.aspx file is used for managing different types of views and the Web.Config file stores configuration settings that are required for this application.

Appendix 2 – Site.Master Code

Site.Master

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs" Inherits="UniTechCollegeSystem.SiteMaster" %>

<!DOCTYPE html>
<html lang="en">
<head runat="server">
    <meta charset="utf-8" />
    <title>UniTech College System</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <form runat="server">
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
            <div class="container-fluid">

                <!-- Logo and brand name -->
                <a class="navbar-brand d-flex align-items-center" href='<%= ResolveUrl("~/Pages/Home.aspx") %>'>
                    <img src='<%= ResolveUrl("~/Images/UniTech College Logo.png") %>' alt="UniTech Logo" style="height:30px; width:auto; margin-right:10px;" />
                    <span class="text-white fw-bold">UniTech College</span>
                </a>

                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
                    <span class="navbar-toggler-icon"></span>
                </button>

                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav ms-auto">

                        <!-- Home -->
                        <li class="nav-item">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/Home.aspx") %>'>Home</a>
                        </li>

                        <!-- About just after Home -->
                        <li class="nav-item">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/AboutUs.aspx") %>'>About</a>
                        </li>

                        <!-- Courses -->
                        <li class="nav-item">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/Courses.aspx") %>'>Courses</a>
                        </li>

                        <!-- Not logged in -->
                        <li class="nav-item" runat="server" id="navRegister">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/Register.aspx") %>'>Register</a>
                        </li>
                        <li class="nav-item" runat="server" id="navLogin">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/Login.aspx") %>'>Login</a>
                        </li>

```

Figure 30 - Coding for site.master (master page)

The code above creates the master page structure for the UniTech College site. It has a navigation bar containing links to the Home, About, and Courses pages. There are also links to the Register and Login pages, but only for people who are not logged in. The navbar is styled and responsive using Bootstrap for both desktop and mobile devices. The brand name of the college is displayed at the top left along with the college brand logo. The navigation items are also right-aligned for a tidy layout. Server-side controls are utilised for dynamic URL resolution on the page.

```
                        <!-- Logged in -->
                        <li class="nav-item" runat="server" id="navProfile" visible="false">
                            <a class="nav-link" href='<%= ResolveUrl("~/Pages/Dashboard.aspx") %>'>Profile</a>
                        </li>
                        <li class="nav-item" runat="server" id="navLogout" visible="false">
                            <asp:LinkButton ID="btnLogout" runat="server" CssClass="nav-link text-danger" OnClick="btnLogout_Click">Logout</asp:LinkButton>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>

        <asp:ContentPlaceHolder ID="MainContent" runat="server" />
    </form>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Figure 31 - Navigation links for logged in user

This section of the code provides the logged-in user with navigation links. It has a link to the user dashboard titled "Profile" and a logout button. These are initially made invisible (visible="false") and are shown only when the user is logged in. Logout LinkButton causes the server-side click event to log the user out. All the pages that have this master layout have the same structure for the page, giving the user a consistent experience. There is also included Bootstrap JavaScript for responsive events.

Appendix 3 – Site.Master.cs Code

```
using System;

namespace UniTechCollegeSystem
{
    public partial class SiteMaster : System.Web.UI.MasterPage
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Only run this logic once per page load
            if (!IsPostBack)
            {
                bool isLoggedIn = Session["UserEmail"] != null;

                navLogin.Visible = !isLoggedIn;
                navRegister.Visible = !isLoggedIn;
                navProfile.Visible = isLoggedIn;
                navLogout.Visible = isLoggedIn;
            }
        }

        // Logout handler
        protected void btnLogout_Click(object sender, EventArgs e)
        {
            Session.Clear();
            Response.Redirect("~/Pages/Home.aspx");
        }
    }
}
```

Figure 32 - Hiding or displaying navigation items if the user is logged in or out

This code hides or displays navigation items depending on the user's log-in or log-out state. During page loading, the code tests whether the user is logged in or not based on the check for the existence of the UserEmail session variable. If the user is logged in, the "Profile" and "Logout" options are rendered; otherwise, the "Login" and "Register" options are rendered. The btnLogout_Click procedure logs the user out through the removal of the session and redirecting the user to the home page.

Appendix 4 – AboutUs.aspx Code

```
<%@ Page Title="About Us" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="AboutUs.aspx.cs" Inherits="UniTechCollegeSystem.Pages.AboutUs" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <!-- Container with padding -->
    <div class="container py-5">
        <!-- Centered main heading -->
        <h1 class="display-4 text-center mb-4">About Us</h1>

        <!-- Centered image section -->
        <p class="lead text-center">
            Based in the city of Leicester, UniTech College has a mission to equip the next generation with the skills, knowledge, and mindset to thrive in a tech-driven world.
        </p>

        <!-- University image -->
        <div class="text-center my-4">
            <img src='<%= ResolveUrl("~/Images/uni.jpg") %>' alt="UniTech College Campus" class="img-fluid rounded shadow-sm" style="max-height: 400px;" />
        </div>

        <!-- Academic focus -->
        <p class="text-center">
            At UniTech, we specialise in technology, engineering, digital innovation, and applied sciences, offering industry-relevant courses that blend academic excellence with real-world experience.
        </p>

        <!-- Facilities and partnerships -->
        <p class="text-center">
            Our state-of-the-art facilities, supportive learning environment, and strong ties with local and global industry partners make us a launchpad for ambitious students from all backgrounds.
        </p>
    </div>
</asp:Content>
```

Figure 33 - Code for about us page

The below is the code for the "About Us" portion of the UniTechCollegeSystem site developed using ASP.NET, based on a master page (Site.master), in the language of C#. The page has the heading "About Us" with a short description about UniTech College in Leicester, pointing to the reason why the college is preparing students for the technology-driven world. It is emphasising the college's technology, engineering, and applied science orientation along with industry-defined courses. A photograph of the campus (uni.jpg) along with the description of the modernised facilities of the college, along with the college's affiliations, is centered in the middle of the page for a tidy format.

Appendix 5 – AboutUs.aspx.cs Code

AboutUs.aspx.cs

UniTechCollegeSystem | UniTechCollegeSystem.Pages.AboutUs | Page_Load(object sender, EventArgs e)

```
// Imports System namespace for basic functionality
using System;

// Defines namespace for page classes
namespace UniTechCollegeSystem.Pages
{
    // AboutUs page class, inherits from Page
    1 reference
    public partial class AboutUs : System.Web.UI.Page
    {
        // Page_Load event handler
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}
```

Figure 34 -C# code for about us.aspx.cs

The following is the corresponding C# code (AboutUs.aspx.cs) for the "About Us" page. It is a class called AboutUs inside the namespace UniTechCollegeSystem.Pages that inherits from System.Web.UI.Page. It has a Page_Load event procedure that is blank but can have any desired code whenever the page is loaded.

Appendix 6 – Courses.aspx Code

Courses.aspx

```
<%@ Page Title="Courses" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Courses.aspx.cs" Inherits="UniTechCollegeSystem.Pages.Courses" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div class="container mt-5">
        <!-- Welcome message for logged-in users -->
        <asp:Panel ID="pnlWelcome" runat="server" Visible="false">
            <h2 class="mb-4 text-center text-primary">Welcome, <asp:Label ID="lblUserName" runat="server"></asp:Label>!</h2>
            <p class="text-center">Please select a course to view its modules.</p>
        </asp:Panel>

        <!-- Login status or validation message -->
        <asp:Label ID="lblLoginStatus" runat="server" CssClass="alert alert-warning d-block text-center" Visible="false" Text="Please log in to select a course and its modules." />

        <!-- Page Title and Description -->
        <div class="text-center mb-5">
            <h1 class="display-4">Explore Our Courses</h1>
            <p class="lead">Discover a wide range of courses designed to help you succeed in your academic and career goals!</p>
        </div>

        <!-- Display all courses as clickable boxes -->
        <asp:Panel ID="pnlCourses" runat="server">
            <div class="row">
                <asp:Repeater ID="rptCourses" runat="server" OnItemCommand="rptCourses_ItemCommand">
                    <ItemTemplate>
                        <div class="col-md-4 mb-4">
                            <div class="card h-100 shadow-sm">
                                <div class="card-body">
                                    <h3 class="card-title"><%# Eval("CourseName") %></h3>
                                    <p class="card-text"><%# Eval("CourseDescription") %></p>
                                    <asp:Button ID="btnSelectCourse" runat="server" Text="View Course" CommandName="SelectCourse" CommandArgument='<%# Eval("CourseID") %>' CssClass="btn btn-primary" />
                                </div>
                            </div>
                        </div>
                    </ItemTemplate>
                </asp:Repeater>
            </div>
        </asp:Panel>
```

Figure 35 - Code for courses page

The code is for the "Courses" page (Courses.aspx) of the UniTechCollegeSystem site. It has a master page (Site.master) that is based on C#. The site welcomes the user personally, in case he is logged in, or prompts him to log in otherwise. A title and description are given encouraging the users to browse the available courses. A repeater control is utilised to represent each course in the form of a clickable card containing the course name, description, and "View Course" button for further information.

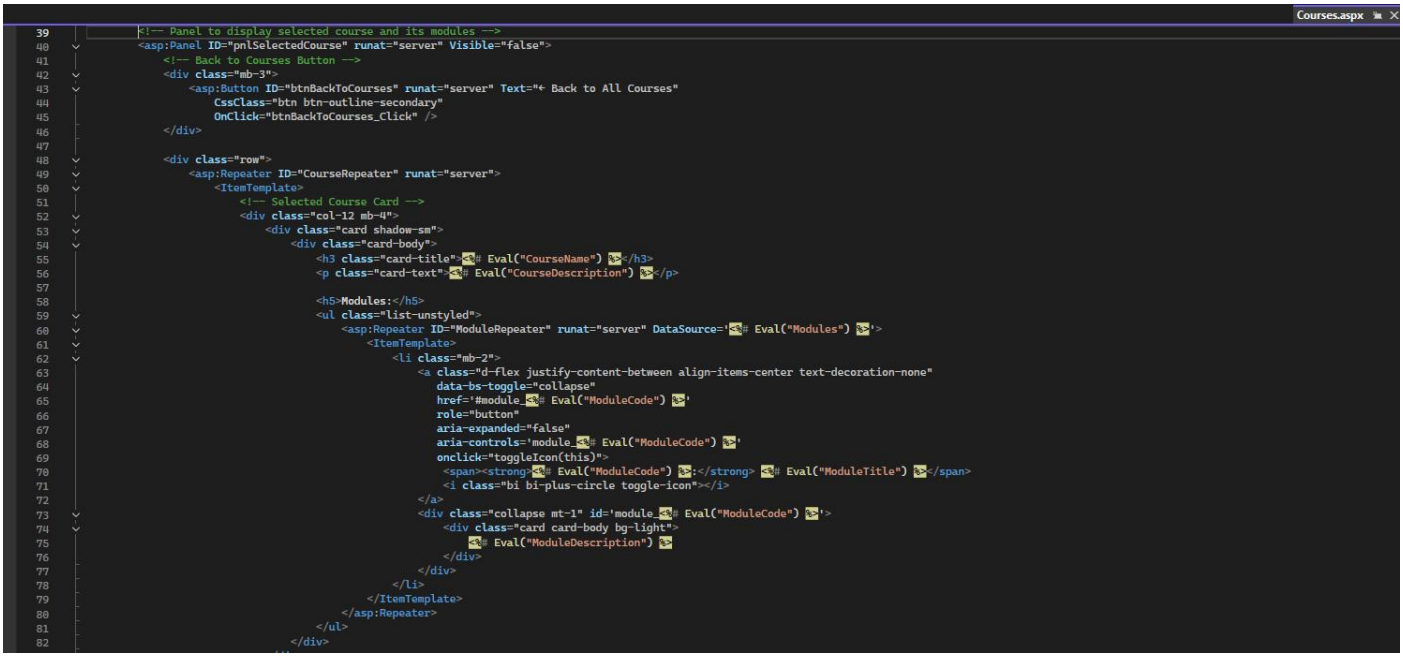


Figure 36 - Continuous code for courses page

Following is the extension of the Courses.aspx page, containing a panel that appears once a course is clicked. It incorporates a "Back to All Courses" link to go back to the list of courses. The name and description of the selected course are shown, along with the list of its modules, displaying a code, a title, and an expandable description section through a repeater control.

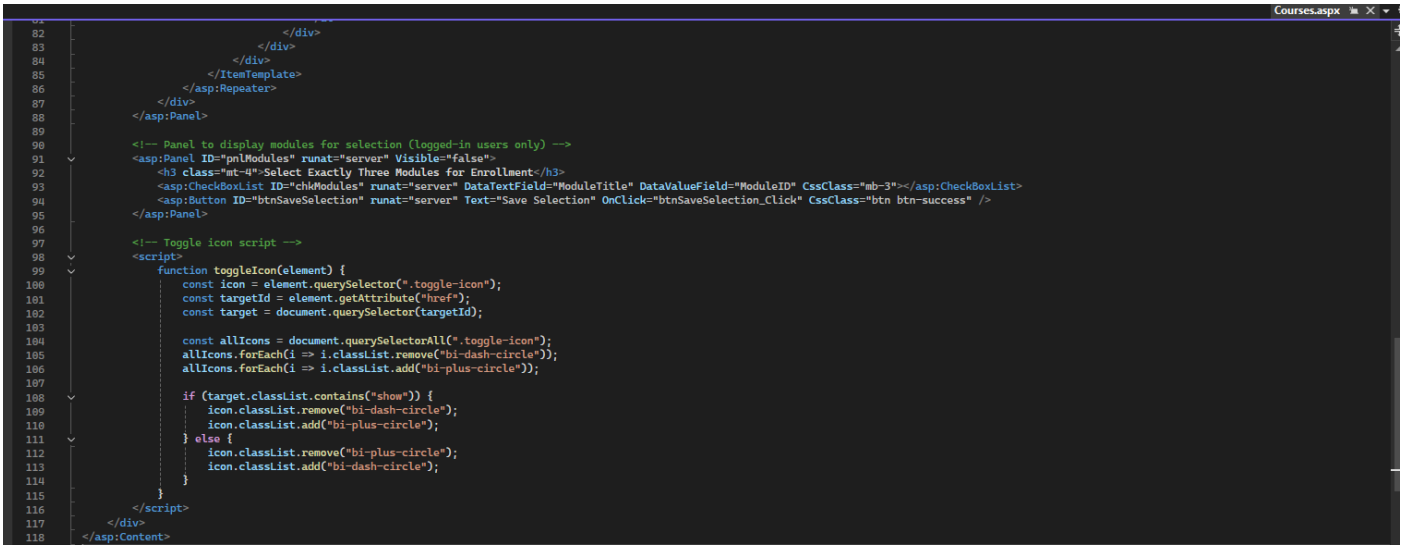
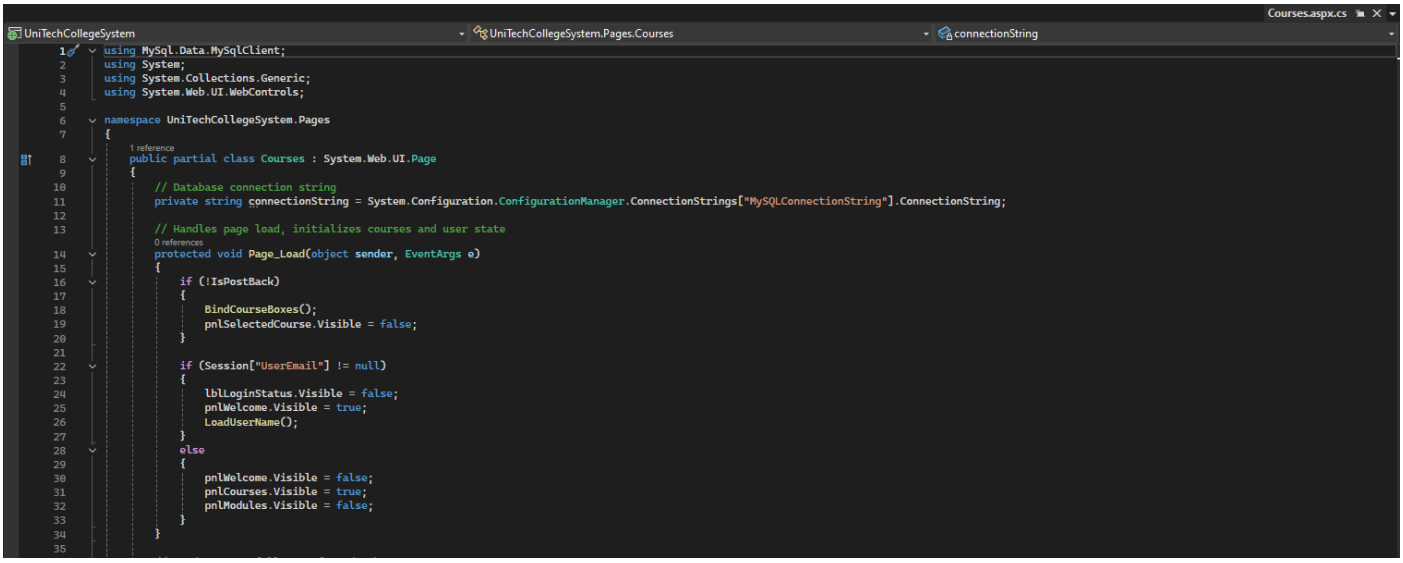


Figure 37 - Last part for courses page(enrollment0

The final part of Courses.aspx has a panel for logged-in users to choose just three modules for enrollment. It employs a checkbox list for the selection of module names and a "Save Selection" button to save the choices. There is a script to toggle the plus/minus icon when opening or closing module information.

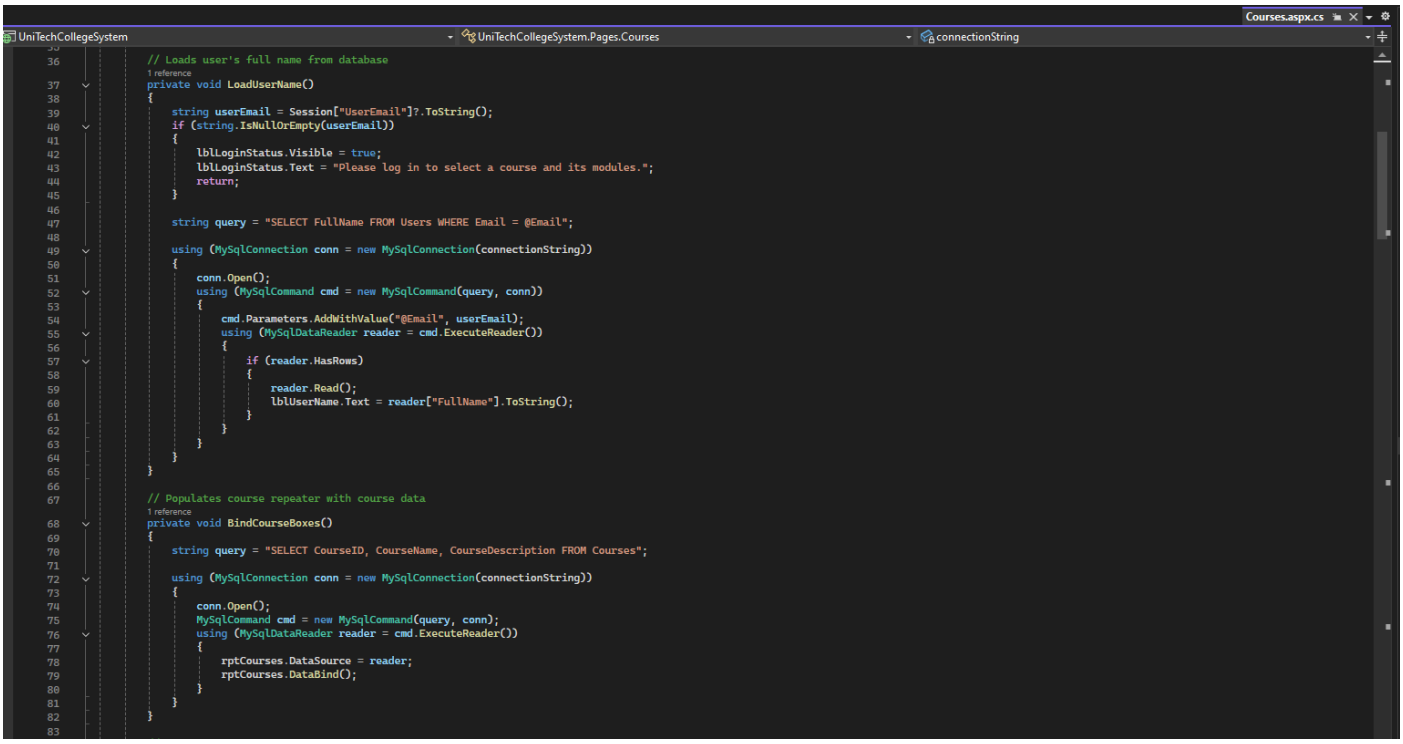
Appendix 7 – Courses.aspx.cs Code



```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Web.UI.WebControls;

namespace UniTechCollegeSystem.Pages
{
    1 reference
    public partial class Courses : System.Web.UI.Page
    {
        // Database connection string
        private string connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["MySQLConnectionString"].ConnectionString;

        // Handles page load, initializes courses and user state
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                BindCourseBoxes();
                pnlSelectedCourse.Visible = false;
            }

            if (Session["UserEmail"] != null)
            {
                lblLoginStatus.Visible = false;
                pnlWelcome.Visible = true;
                LoadUserName();
            }
            else
            {
                pnlWelcome.Visible = false;
                pnlCourses.Visible = true;
                pnlModules.Visible = false;
            }
        }
```

Figure 38 - Code for courses in C# communicating with MySQL database

This is the code-behind class for the Courses.aspx page in C# (Courses.aspx.cs). It handles the page logic and communicates with a MySQL database. The Page_Load function first checks whether this is the first load of the page, binds the course data, and hides the panel for the selected course. It also verifies whether the user is logged in or not and provides the user with a welcome message along with their name or else a prompt.



```
// Loads user's full name from database
1 reference
private void LoadUserName()
{
    string userEmail = Session["UserEmail"]?.ToString();
    if (string.IsNullOrEmpty(userEmail))
    {
        lblLoginStatus.Visible = true;
        lblLoginStatus.Text = "Please log in to select a course and its modules.";
        return;
    }

    string query = "SELECT FullName FROM Users WHERE Email = @Email";

    using (MySqlConnection conn = new MySqlConnection(connectionString))
    {
        conn.Open();
        using (MySqlCommand cmd = new MySqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@Email", userEmail);
            using (MySqlDataReader reader = cmd.ExecuteReader())
            {
                if (reader.HasRows)
                {
                    reader.Read();
                    lblUserName.Text = reader["FullName"].ToString();
                }
            }
        }
    }
}

// Populates course repeater with course data
1 reference
private void BindCourseBoxes()
{
    string query = "SELECT CourseID, CourseName, CourseDescription FROM Courses";

    using (MySqlConnection conn = new MySqlConnection(connectionString))
    {
        conn.Open();
        MySqlCommand cmd = new MySqlCommand(query, conn);
        using (MySqlDataReader reader = cmd.ExecuteReader())
        {
            rptCourses.DataSource = reader;
            rptCourses.DataBind();
        }
    }
}
```

Figure 39 - Courses(load UserName and blindcourseboxes)

The two procedures within the section are: Load UserName loads the user's full name from the database using their session email, and BindCourseBoxes loads course data (ID, name, description) from the database and binds the data to a repeater to be displayed.

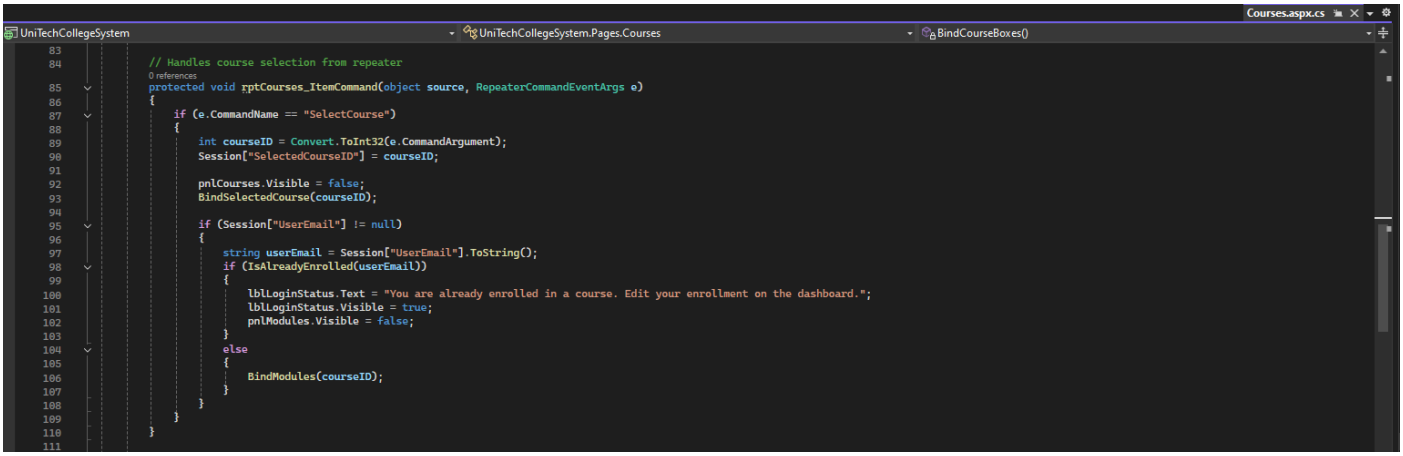


Figure 40 - Functionalities of viewing course

This function handles the selection of the course on clicking "View Course." It saves the selected course ID to the session, removes the core course listing, and populates course data. Should the user also be logged in, this also validates whether or not they are already enrolled.If so, they are presented with a message to edit their enrolments on the dashboard; otherwise, it loads available modules.

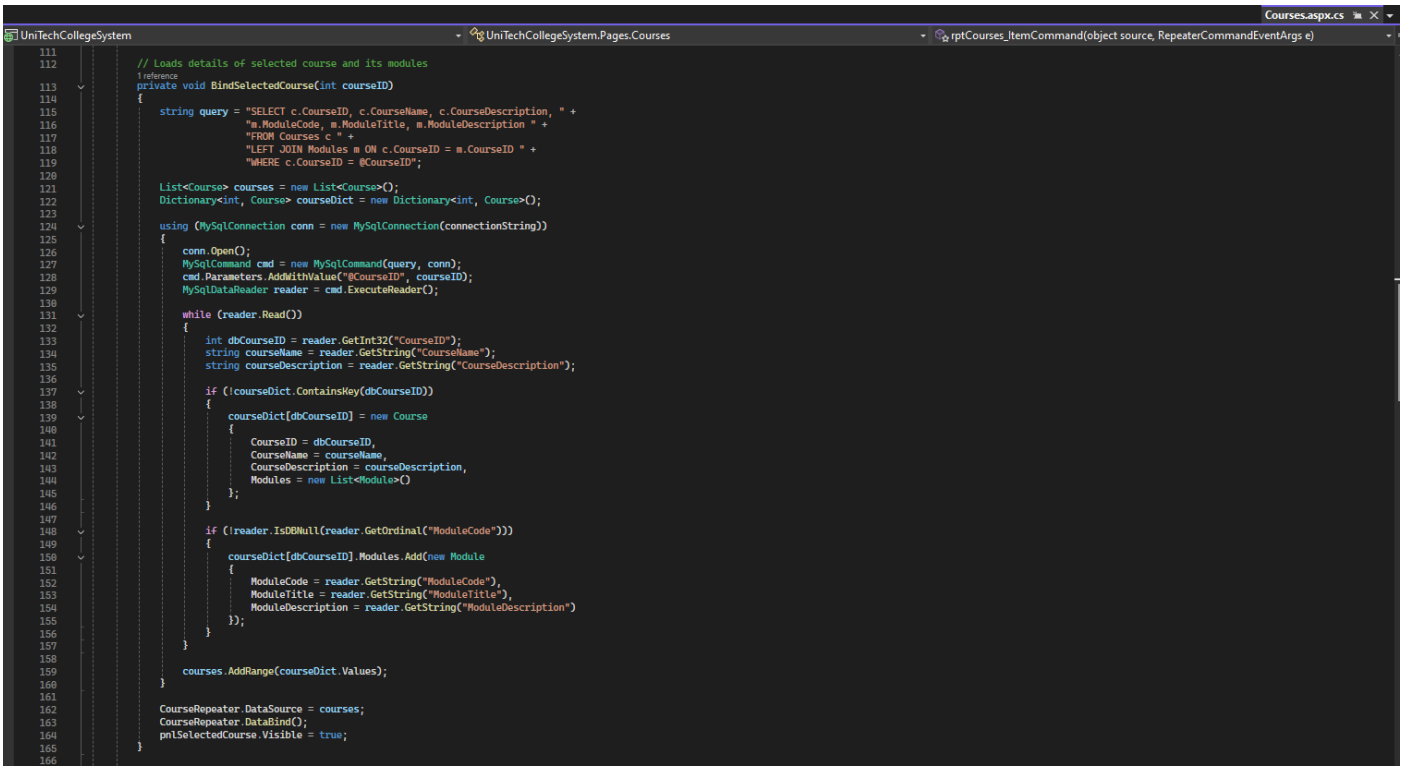


Figure 41 - BlindSelectedCourse function

This section has the BindSelectedCourse function that pulls the selected course and module information from a LEFT JOIN SQL statement. The information is held in a list and presented using a repeater. Then the selected course panel is displayed.

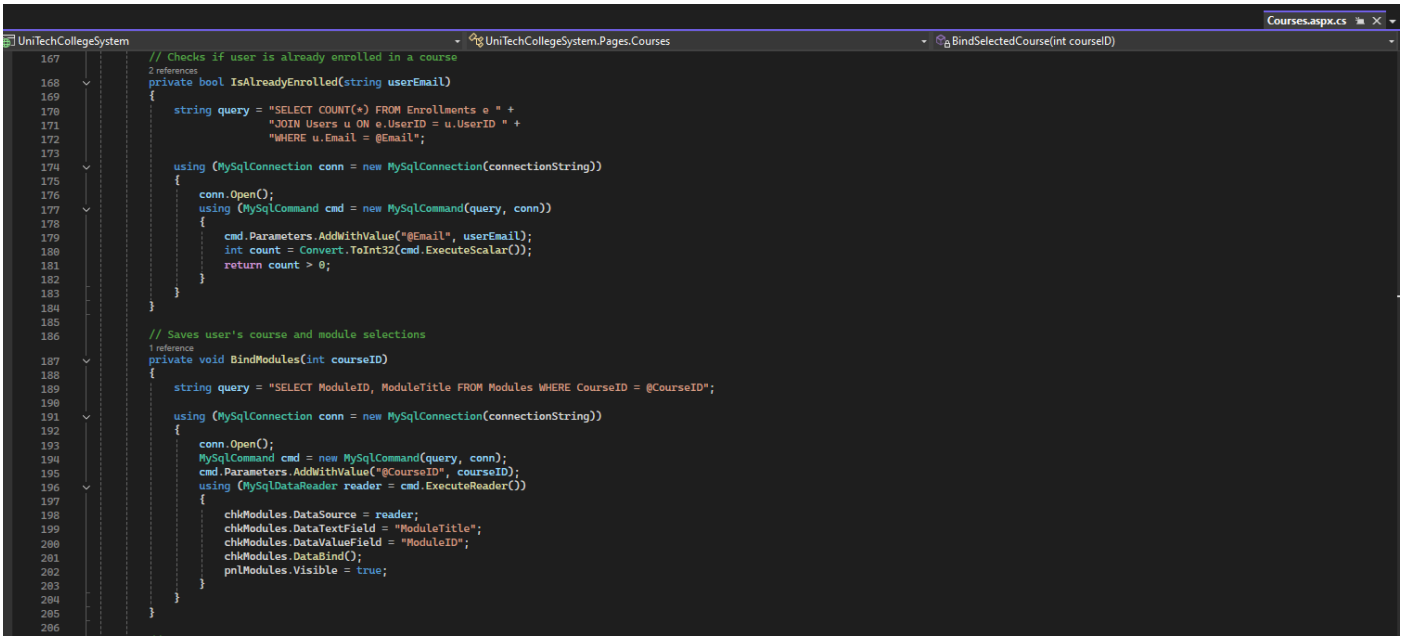


Figure 42 - Functions for IsAlreadyEnrolled and BlindModules

This has two functions: IsAlreadyEnrolled verifies whether a user is already enrolled in a course based on their email address. If so, it returns true. BindModules loads the course title and IDs for the selected course, binds them to a check box list, and opens the modules panel.

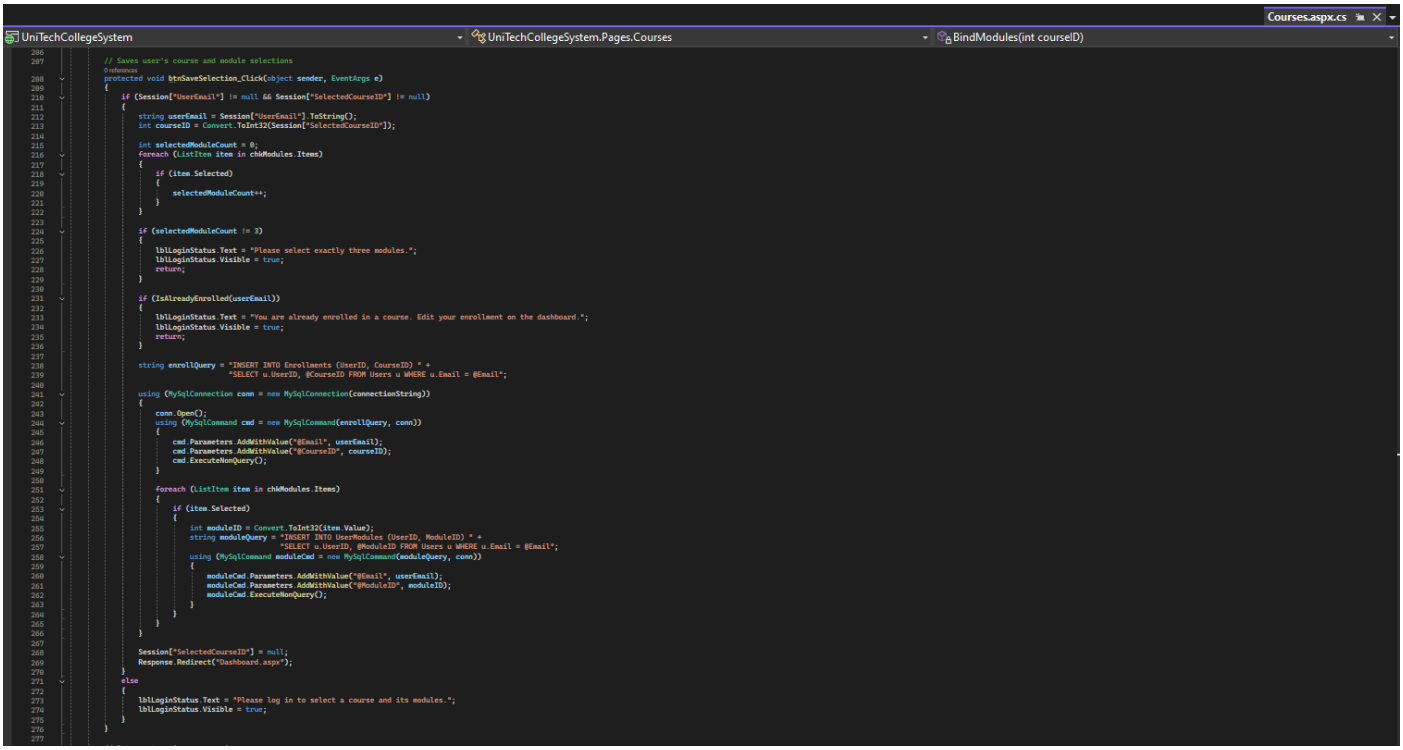


Figure 43 - Save selection feature

This handles the click of the "Save Selection" button. It validates that a logged-in user has selected just three modules. If the user is logged in and is not already enrolled, their selection is saved into the UserModules and Enrollments tables in the MySQL database. Upon saving, the user is redirected to the Dashboard page.

```
Courses.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Courses | btnSaveSelection_Click(object sender, EventArgs e)
278         // Returns to main courses view
            0 references
279         protected void btnBackToCourses_Click(object sender, EventArgs e)
280         {
281             pnlSelectedCourse.Visible = false;
282             pnlCourses.Visible = true;
283             pnlModules.Visible = false;
284             lblLoginStatus.Visible = false;
285         }
286
287         // Course data model
            5 references
288         public class Course
289         {
                1 reference
290             public int CourseID { get; set; }
                1 reference
291             public string CourseName { get; set; }
                1 reference
292             public string CourseDescription { get; set; }
                2 references
293             public List<Module> Modules { get; set; }
294         }
295
296         // Module data model
            3 references
297         public class Module
298         {
                1 reference
299             public string ModuleCode { get; set; }
                1 reference
300             public string ModuleTitle { get; set; }
                1 reference
301             public string ModuleDescription { get; set; }
302         }
303     }
304 }
305
```

Figure 44 - Returning user back to main courses once they've chosen course

This piece of code manages the return of the user to the main courses view upon choosing a course, by manipulating the visibilities of various panels on the site. It also introduces two simple data models for courses and for modules which organise the course and module information that appears on the site. The data models store plain information like the course name and module name, making the system display the right content upon the user selecting a course.

Appendix 8 – Dashboard.aspx Code

```
Dashboard.aspx
1   <%@ Page Title="Dashboard" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Dashboard.aspx.cs" Inherits="UniTechCollegeSystem.Pages.Dashboard" %>
2
3   <asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
4       <div class="container mt-5">
5           <!-- Welcome Header -->
6           <h2 class="mb-4 text-center text-primary">
7               Welcome, <asp:Label ID="lblUserName" runat="server" CssClass="text-primary"></asp:Label>
8           </h2>
9
10          <!-- User Information Section -->
11          <h4>Your Information</h4>
12          <p><strong>Name:</strong> <asp:Label ID="lblFullName" runat="server"></asp:Label></p>
13          <p><strong>Email:</strong> <asp:Label ID="lblEmail" runat="server"></asp:Label></p>
14          <!-- Edit Personal Information Button -->
15          <asp:Button ID="btnEdit" runat="server" Text="Edit Details" CssClass="btn btn-primary mb-4" OnClick="btnEdit_Click" />
16
17          <!-- Enrolled Courses Section -->
18          <h4 class="mt-4">Your Enrolled Courses</h4>
19          <!-- Repeater for course cards with edit enrollment option -->
20          <asp:Repeater ID="rptCourses" runat="server" OnItemDataBound="rptCourses_ItemDataBound">
21              <ItemTemplate>
22                  <div class="card mb-4 shadow-sm">
23                      <div class="card-body">
24                          <h3 class="card-title"><%# Eval("CourseName") %></h3>
25                          <p class="card-text"><strong>Description:</strong> <%# Eval("CourseDescription") %></p>
26                          <h5>Modules:</h5>
27                          <ul class="list-unstyled">
28                              <asp:Repeater ID="rptModules" runat="server">
29                                  <ItemTemplate>
30                                      <li><%# Eval("ModuleTitle") %></li>
31                                  </ItemTemplate>
32                              </asp:Repeater>
33                          </ul>
34                          <!-- New feature: Button to edit enrollment -->
35                          <asp:Button ID="btnEditEnrollment" runat="server" Text="Edit Enrollment" CommandArgument='<%# Eval("CourseID") %>' OnClick="btnEditEnrollment_Click" CssClass="btn btn-secondary" />
36                          <!-- End of new feature: Button to edit enrollment -->
37                      </div>
38                  </div>
39              </ItemTemplate>
40          </asp:Repeater>
41          <!-- Label for no enrolled courses -->
42          <asp:Label ID="lblNoCourses" runat="server" Text="You are not enrolled in any courses." Visible="false" CssClass="text-muted" />
43
44          <!-- Edit Enrollment Panel -->
45          <asp:Panel ID="pnlEditEnrollment" runat="server" Visible="false">
46              <div class="card mb-4 shadow-sm">
47                  <div class="card-header">
48                      <h4 class="card-title">Edit Enrollment</h4>
49                  </div>
50                  <div class="card-body">
51                      <div class="mb-3">
52                          <label for="ddlCourses" class="form-label">Select Course:</label>
53                          <asp:DropDownList ID="ddlCourses" runat="server" CssClass="form-select" AutoPostBack="true" OnSelectedIndexChanged="ddlCourses_SelectedIndexChanged">
54                          </asp:DropDownList>
55                      </div>
```

Figure 45 - ASP.NET code foe dashboard.aspx page

The code below is the ASP.NET code for the Dashboard.aspx page in the UniTechCollegeSystem project. It renders a personalised dashboard for logged-in users indicating their name, email, and enrolled courses along with associated modules. Personal information for users may also be updated or their course registrations updated directly from this page. It is intended to provide students with a clear picture of their academic standing and facilitate ease of interaction with their data. The design is minimalistic and user-friendly to maintain the aim of the assignment brief to create a functional and aesthetically appealing college system.

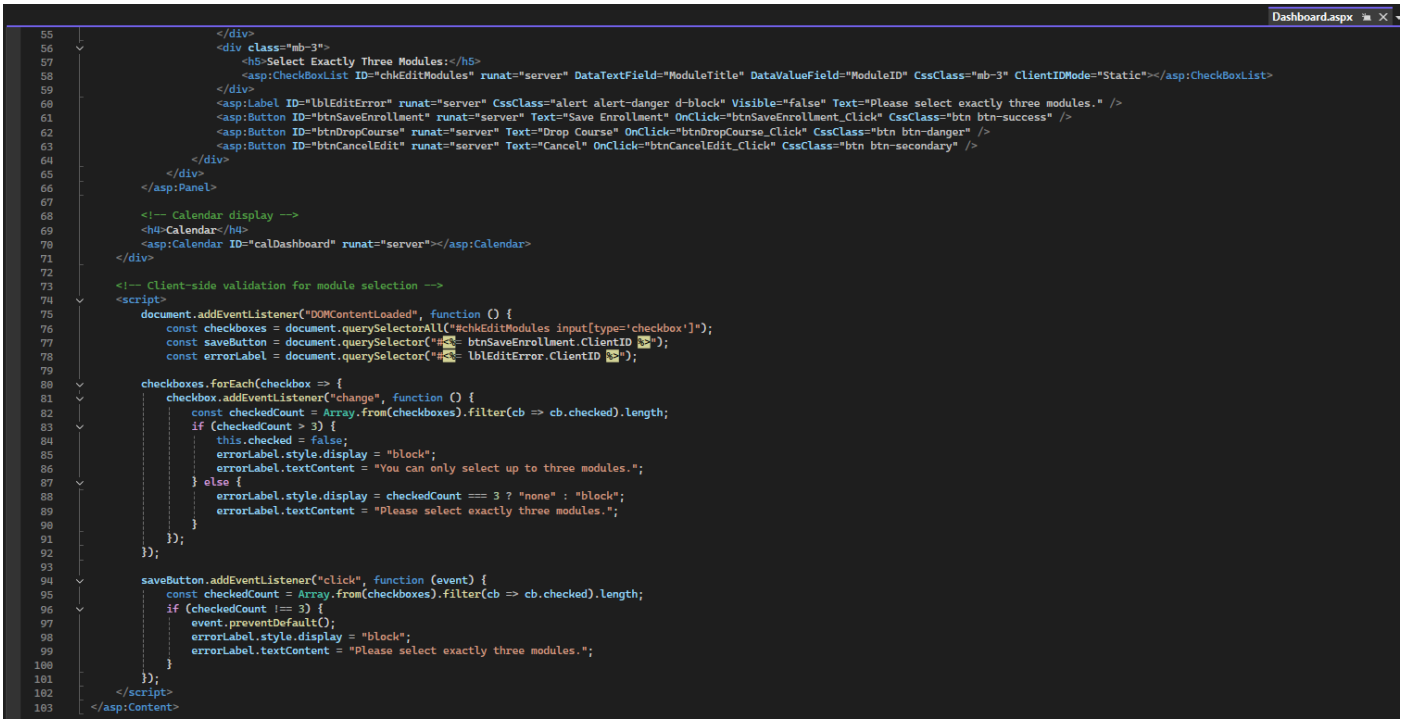


Figure 46 - Editing courses on dashboard

This part of the Dashboard.aspx page allows students to edit their course modules. It provides students with the facility to select only three modules from the available selection while editing the enrolment. Save changes, drop course, or cancel edit buttons have been provided. There is also an inbuilt calendar for scheduling. Client-side validation to enhance user experience is also included to check that students are not capable of choosing more or less than three modules, in line with the aim of the assignment brief to design a user-friendly, easy-to-use, and interactive academic system.

Appendix 9 – Dashboard.aspx.cs Code

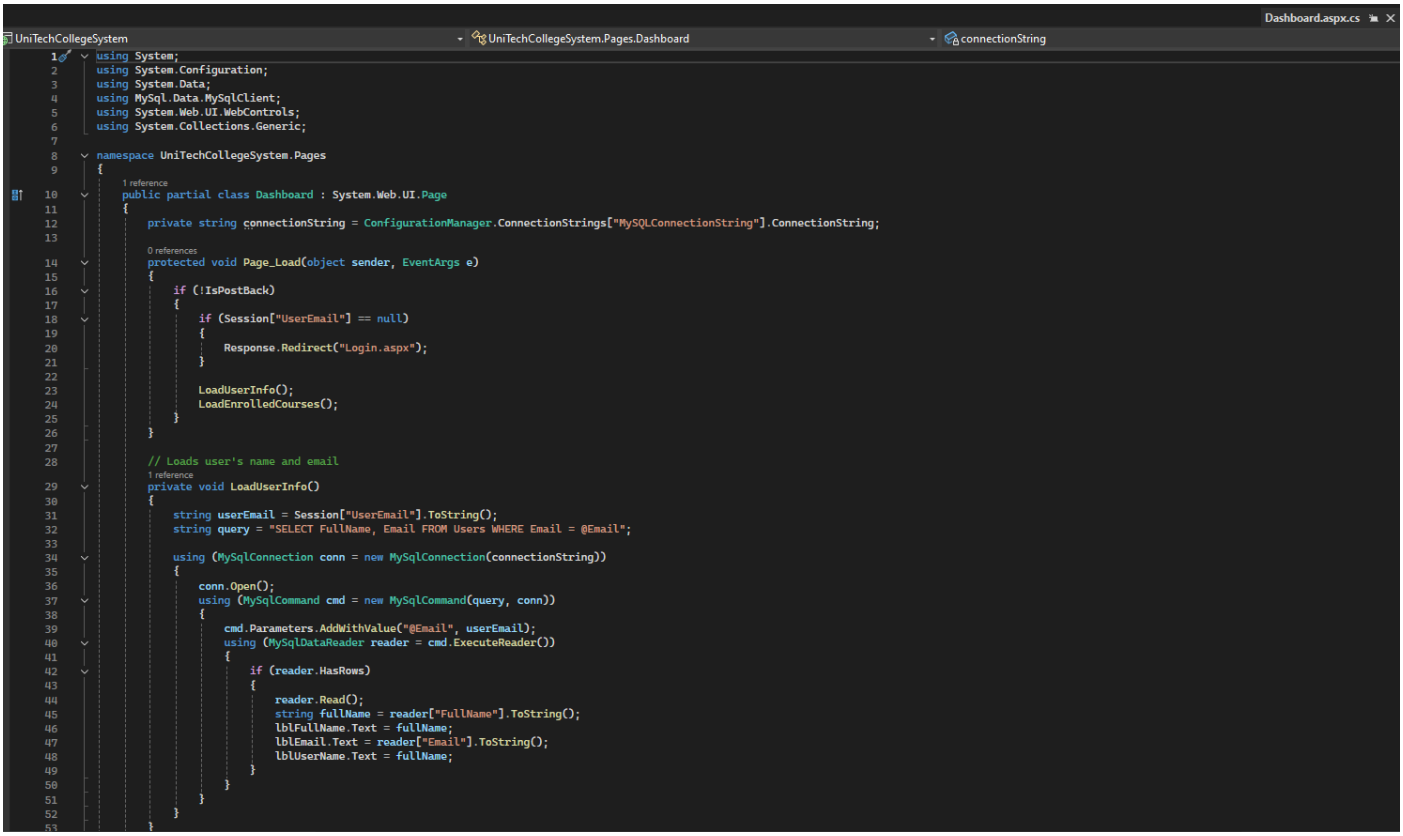


Figure 47 - Back-end logic for dashbaord.aspx

The following is the backend logic for the Dashboard.aspx page in this C# code. It verifies whether a user is logged in or not, redirecting them to the login page in case they are not. After logging in, the code queries the MySQL data to fetch the user's full name and their email. This is in line with the objective of the assignment of enabling a personalised dashboard experience, where students are able to safely check their information upon logging in.

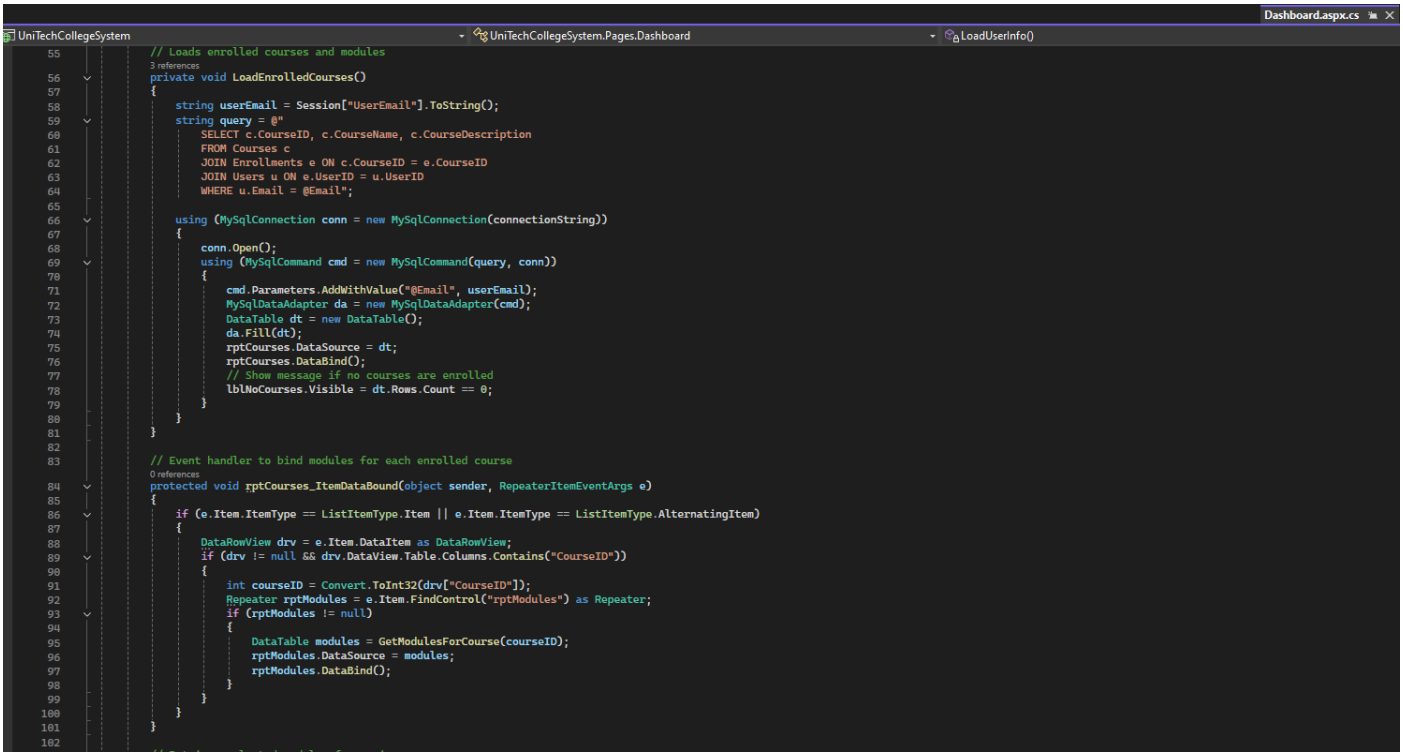


Figure 48 - Indication of the courses the user is enrolled in

This is utilised to indicate to the user the courses for which he is enrolled, including the respective modules. It queries the database with the mail address of the user, retrieves the courses, and displays them on the dashboard. If there are no courses, a message is shown. It also makes sure that for every course displayed, the appropriate modules are listed. It facilitates the objective of the project to provide students with an easy and dynamic overview of their progress.

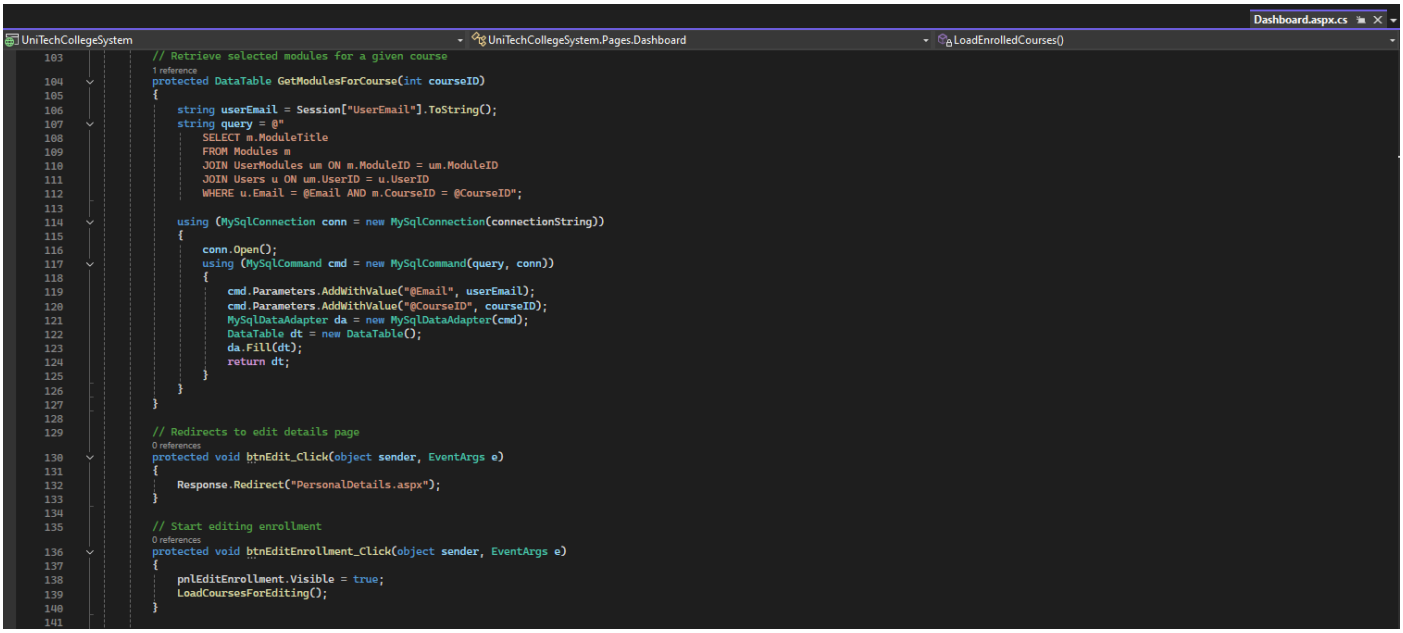


Figure 49 - Loading specific modules for that user and editing enrolment and personal details

This section does two primary things: first, it loads the specific modules that a user has enrolled for a specific course, personalising their dashboard view. Second, it offers edit enrolment and personal details buttons that

redirect the user to the personal details page or display the edit module panel. This makes the goal to allocate the objective to enable students to have choices regarding their module selection and courses.

```
Dashboard.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Dashboard | GetModulesForCourse(int courseID)

    // Load courses for editing
    1 reference
    private void LoadCoursesForEditing()
    {
        string query = "SELECT CourseID, CourseName FROM Courses";

        using (MySqlConnection conn = new MySqlConnection(connectionString))
        {
            conn.Open();
            using (MySqlCommand cmd = new MySqlCommand(query, conn))
            {
                MySqlDataAdapter da = new MySqlDataAdapter(cmd);
                DataTable dt = new DataTable();
                da.Fill(dt);
                ddlCourses.DataSource = dt;
                ddlCourses.DataTextField = "CourseName";
                ddlCourses.DataValueField = "CourseID";
                ddlCourses.DataBind();
            }
        }

        string userEmail = Session["UserEmail"].ToString();
        if (!string.IsNullOrEmpty(userEmail))
        {
            string enrollmentQuery = @"
                SELECT CourseID
                FROM Enrollments e
                JOIN Users u ON e.UserID = u.UserID
                WHERE u.Email = @Email";
            using (MySqlConnection conn = new MySqlConnection(connectionString))
            {
                conn.Open();
                using (MySqlCommand cmd = new MySqlCommand(enrollmentQuery, conn))
                {
                    cmd.Parameters.AddWithValue("@Email", userEmail);
                    object result = cmd.ExecuteScalar();
                    if (result != null)
                    {
                        ddlCourses.SelectedValue = result.ToString();
                        LoadModulesForEditing(Convert.ToInt32(result));
                    }
                    else if (ddlCourses.Items.Count > 0)
                    {
                        LoadModulesForEditing(Convert.ToInt32(ddlCourses.SelectedValue));
                    }
                }
            }
        }
    }
```

Figure 50 - Filling available courses in dropdown list

This code fills the available courses in the dropdown list so that users are capable of editing the course that they have selected. The course is picked for the user, along with the respective modules for the user to edit, should the user already be enrolled in the course. This serves to advance the function of the assignment to give the user an interactive experience to go along with a user-friendly way to manage their options of learning.

```
Dashboard.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Dashboard | LoadCoursesForEditing()

    // Load modules for the selected course during editing
    3 references
    private void LoadModulesForEditing(int courseID)
    {
        string userEmail = Session["UserEmail"].ToString();
        string query = "SELECT ModuleID, ModuleTitle FROM Modules WHERE CourseID = @CourseID";

        using (MySqlConnection conn = new MySqlConnection(connectionString))
        {
            conn.Open();
            using (MySqlCommand cmd = new MySqlCommand(query, conn))
            {
                cmd.Parameters.AddWithValue("@CourseID", courseID);
                MySqlDataAdapter da = new MySqlDataAdapter(cmd);
                DataTable dt = new DataTable();
                da.Fill(dt);
                chkEditModules.DataSource = dt;
                chkEditModules.DataTextField = "ModuleTitle";
                chkEditModules.DataValueField = "ModuleID";
                chkEditModules.DataBind();
            }

            if (!string.IsNullOrEmpty(userEmail))
            {
                string moduleQuery = @"
                    SELECT m.ModuleID
                    FROM UserModules um
                    JOIN Users u ON um.UserID = u.UserID
                    JOIN Modules m ON um.ModuleID = m.ModuleID
                    WHERE u.Email = @Email AND m.CourseID = @CourseID";
                using (MySqlCommand cmd = new MySqlCommand(moduleQuery, conn))
                {
                    cmd.Parameters.AddWithValue("@Email", userEmail);
                    cmd.Parameters.AddWithValue("@CourseID", courseID);
                    using (MySqlDataReader reader = cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            string moduleID = reader["ModuleID"].ToString();
                            ListItem item = chkEditModules.Items.FindByValue(moduleID);
                            if (item != null)
                            {
                                item.Selected = true;
                            }
                        }
                    }
                }
            }
        }
    }
```

Figure 51 - Loading modules for course that has been selected while editing

This section loads the modules for the course that is selected while editing. It loads the available modules for the selected course to populate a checkbox list and also checks the boxes for the modules that have been

selected previously. This makes students' course module selection accessible for viewing and editing in a simple way.

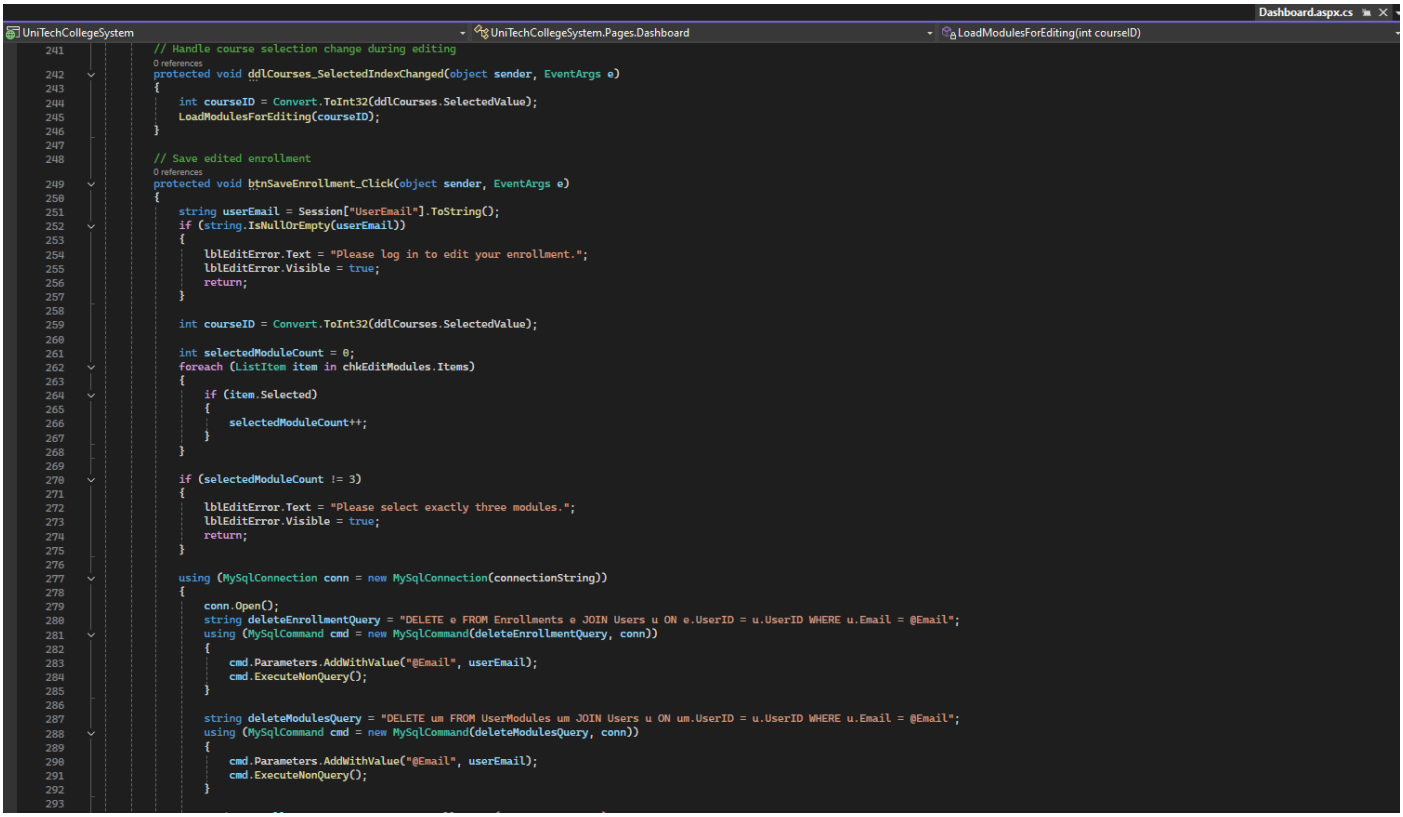


Figure 52 - Managing the user's selection of course on module while editing

This script manages the user selections for the course and module while the user is in the process of editing. If the user chooses a different course, the appropriate modules for the course are loaded. Upon the user clicking the "Save Enrollment" button, this script checks whether the user is logged in and has picked just three modules. If this is the case, the user's course selection, along with the module selection, is updated in the database. If not, an error message is presented.

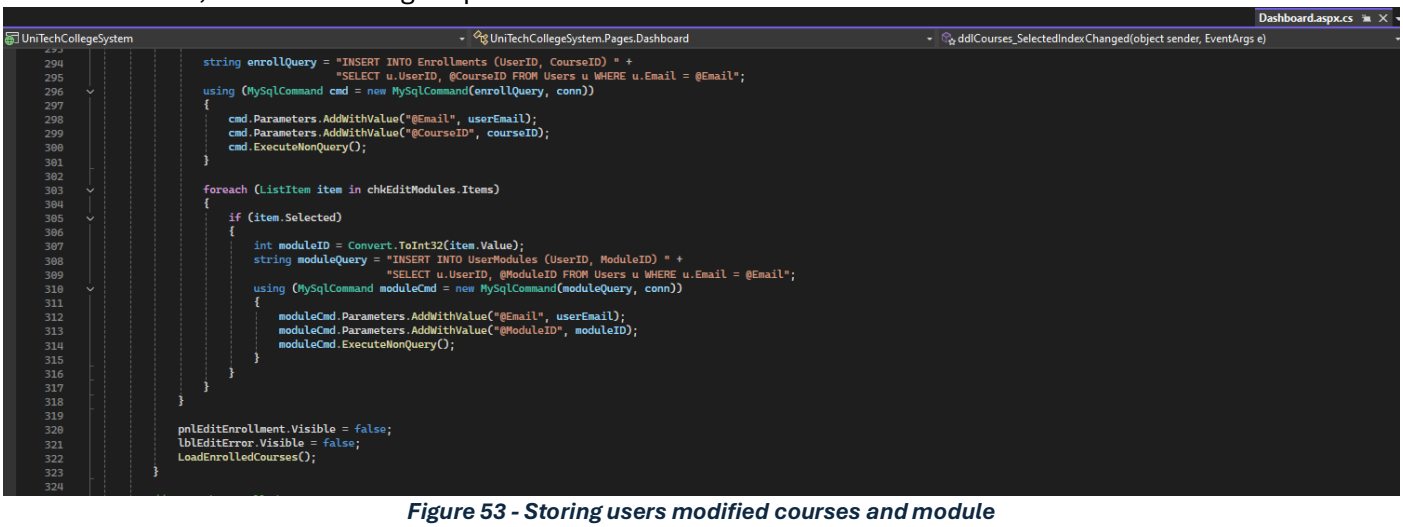


Figure 53 - Storing users modified courses and module

That section of code stores the user's modified course and module choices to the db. Following a check that the user is logged in and has opted for the appropriate number of modules, the new course selection and module choices are added to the appropriate tables. The course is added to the "Enrollments" table, and the module choices are added to the "UserModules" table. After the changes are saved, the edit panel for enrolments is hidden, error messages are cleared, and the enrolled courses list is reloaded.

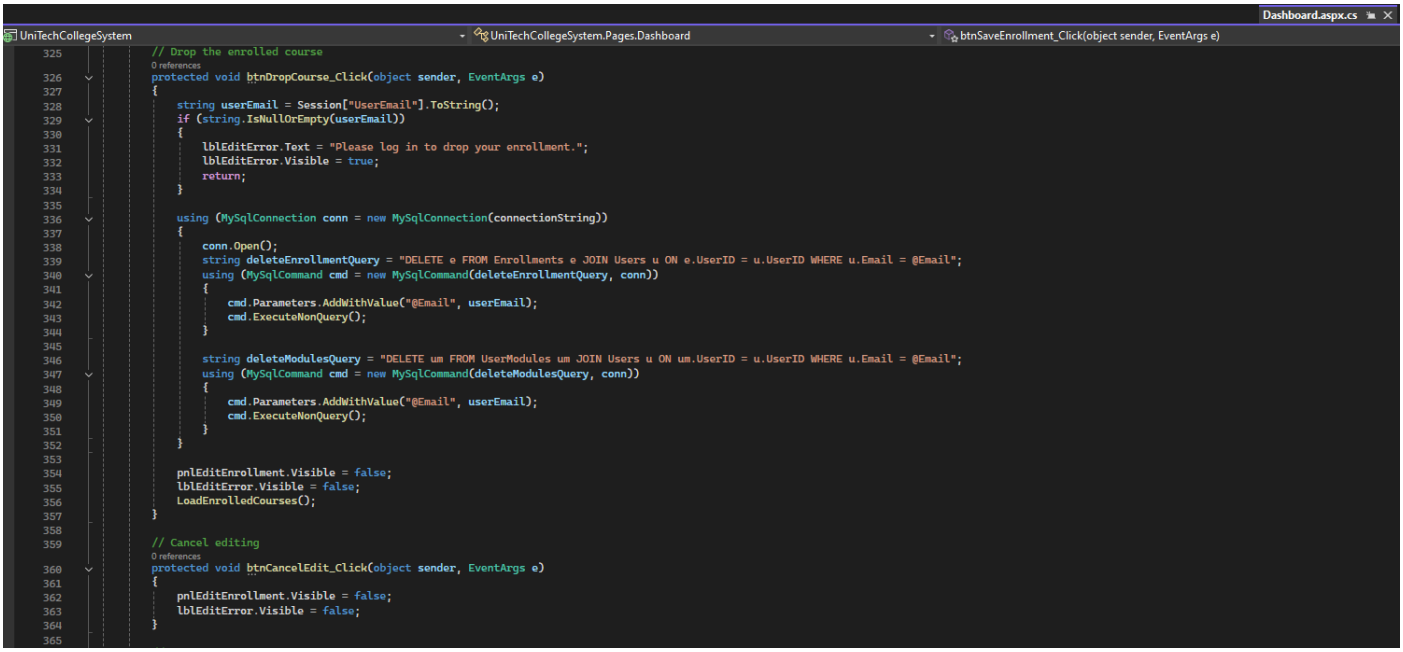


Figure 54 - Handling dropping the course and also cancelling the edit in the enrolment

This is the code that handles dropping the course and cancellation of the edit in the enrollment. Upon clicking on the "Drop Course" button, it first checks whether or not the user is logged in. If the user is logged in, it removes the user's enrollment along with their module choices from the database. The data about the course, along with the modules, is removed from the respective tables. Then, it hides the edit panel for the enrollment and refreshes the list of the courses for which the user is enrolled. If the user wishes to cancel editing, upon clicking the "Cancel Edit" button, the edit panel along with the error message is simply hidden.

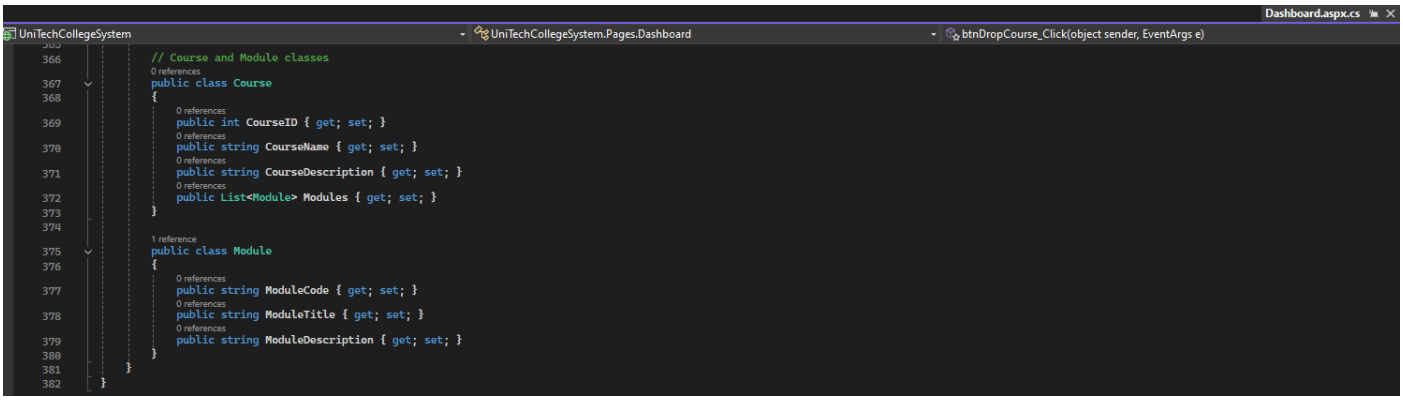


Figure 55 - Creating course and module class

The following code creates two classes, Module and Course, that hold the structure for courses and modules within the system. The properties on the Course class are for the course ID, name, description, and list of modules for that course. The properties for the Module class are the module code, the module's title, and the description. These classes are utilised to represent the data for the modules and courses in the program.

Appendix 10 – Home.aspx Code

Home.aspx

```
<%@ Page Title="Home" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Home.aspx.cs" Inherits="UniTechCollegeSystem.Pages.Home" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <!-- Hero section with a background gradient and padding -->
    <div class="bg-light py-5">
        <div class="container text-center">
            <!-- Headline -->
            <h1 class="display-3 fw-bold text-primary mb-4">Welcome to <span class="text-dark">UniTech College</span></h1>

            <!-- Subheading -->
            <p class="lead text-secondary mb-4">Explore our wide variety of courses and start your journey to success!</p>

            <!-- CTA Button -->
            <a href='<%= ResolveUrl("~/Pages/Courses.aspx") %>' class="btn btn-outline-primary btn-lg px-4 py-2 shadow-sm rounded-pill">
                🚀 Explore Courses
            </a>

            <!-- Homepage Image: Students on Laptops -->
            <div class="mt-5">
                <img src='<%= ResolveUrl("~/Images/UniTech College Picture.png") %>' alt="Students on Laptops" class="img-fluid rounded shadow" style="max-height: 450px;" />
            </div>
        </div>
    </div>

    <!-- Feature highlights section -->
    <div class="container py-5">
        <div class="row g-4">
            <div class="col-md-4">
                <div class="card h-100 shadow-sm border-0">
                    <div class="card-body">
                        <h5 class="card-title text-primary">🛍 Diverse Courses</h5>
                        <p class="card-text">From Computer Science to AI, choose from 5 specialisations tailored to today's tech-driven world.</p>
                    </div>
                </div>
            </div>

            <div class="col-md-4">
                <div class="card h-100 shadow-sm border-0">
                    <div class="card-body">
                        <h5 class="card-title text-success">💡 Expert Instructors</h5>
                        <p class="card-text">Learn from industry professionals and academic experts committed to your success.</p>
                    </div>
                </div>
            </div>

            <div class="col-md-4">
                <div class="card h-100 shadow-sm border-0">
                    <div class="card-body">
                        <h5 class="card-title text-danger">⏳ Career-Focused</h5>
                        <p class="card-text">Our curriculum is designed with employability in mind, equipping you with real-world skills.</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</asp:Content>
```

Figure 56 - Defining homepage structure for Unitech College

The following code is defining the structure for the UniTech College system homepage. It begins by defining the page title, master page, and code-behind file. The page has a hero containing a welcome heading, a short subheading, and a call-to-action button directing to the courses page. There is also an image of students working on laptops. Below that, the page features the top three aspects of the college: diverse courses, expert faculty, and career-oriented curriculum, which are presented in a card format. The structure is accomplished with the help of Bootstrap classes for a neat and responsive user interface.

Appendix 11 – Home.aspx.cs Code

Home.aspx.cs

UniTechCollegeSystem | UniTechCollegeSystem.Pages.Home | Page_Load(object sender, EventArgs e)

```
using System;

namespace UniTechCollegeSystem.Pages
{
    1 reference
    public partial class Home : System.Web.UI.Page
    {
        // Page_Load method is triggered when the page is loaded
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
            // Here you can add any custom logic if needed for the Home page
        }
    }
}
```

Figure 57 - Code for home,aspx page

The following code is the code-behind for the Home.aspx page of the UniTech College system. It has a Page_Load method, which is executed whenever the page loads. The method does not include customised logic but can be added to in the future whenever there is a particular function to include on the home page. The class is located in the UniTechCollegeSystem.Pages namespace.

Appendix 12 – Login.aspx Code

Login.aspx

```
<%@ Page Title="Login" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="UniTechCollegeSystem.Pages.Login" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div class="container mt-5">
        <!-- Page Heading for Login -->
        <h2 class="mb-4 text-center text-primary">Login to Your Account</h2>

        <!-- Label for displaying error/success messages -->
        <asp:Label ID="lblMessage" runat="server" CssClass="text-danger mb-3 d-block" />

        <!-- New feature: Label for displaying remaining attempts -->
        <asp:Label ID="lblAttempts" runat="server" CssClass="text-warning mb-3 d-block" />
        <!-- End of new feature: Label for displaying remaining attempts -->

        <!-- Input field for email -->
        <div class="mb-3">
            <asp:Label ID="lblEmail" runat="server" Text="Email" />
            <asp:TextBox ID="txtEmail" runat="server" TextMode="Email" CssClass="form-control" />
        </div>

        <!-- Input field for password -->
        <div class="mb-3">
            <asp:Label ID="lblPassword" runat="server" Text="Password" />
            <asp:TextBox ID="txtPassword" runat="server" TextMode="Password" CssClass="form-control" />
        </div>

        <!-- Login button which triggers the authentication logic -->
        <asp:Button ID="btnLogin" runat="server" Text="Login" CssClass="btn btn-primary w-100" OnClick="btnLogin_Click" />
    </div>
</asp:Content>
```

Figure 58 - Code for design for the logging in form

This code creates the design for the login form in the UniTech College system. It has a heading for the form, fields for the user to input their email and password, and a button to initiate the process of logging in. There are labels to show messages, for example, error or success messages, and a new addition to indicate the number of attempts available for logging in. The form is made to process user login upon clicking the login button using a procedure indicated in the code-behind.

Appendix 13 – Login.aspx.cs Code

Login.aspx.cs

UniTechCollegeSystem | UniTechCollegeSystem.Pages.Login | connectionString

```
using System;
using System.Configuration;
using System.Security.Cryptography;
using System.Text;
using MySql.Data.MySqlClient;

namespace UniTechCollegeSystem.Pages
{
    1 reference
    public partial class Login : System.Web.UI.Page
    {
        private string connectionString = ConfigurationManager.ConnectionStrings["MySQLConnectionString"].ConnectionString;
        // Constants for login attempt tracking and lockout
        private const int MaxAttempts = 3;
        private const int LockoutMinutes = 15;

        // Page load event
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        // Handle login button click
        0 references
        protected void btnLogin_Click(object sender, EventArgs e)
        {
            string email = txtEmail.Text.Trim();
            string password = txtPassword.Text;

            if (string.IsNullOrWhiteSpace(email) || string.IsNullOrWhiteSpace(password))
            {
                lblMessage.CssClass = "text-danger";
                lblMessage.Text = "Please enter both email and password.";
                lblAttempts.Text = "";
                return;
            }
```

Figure 59 - Login function for UniTech College System

The following is the code for the UniTech College system login function. It validates the user input for the email address and the password to check that both are not blanks. The page is designed to communicate with a MySQL database, and log attempts are monitored to restrict misuse, with a lockout for 15 minutes for three attempts. It displays an error message if the email or the password is blank. The database connection string is retrieved from the configuration settings to maintain secure handling of the connection.

```
// Check lockout status and track failed attempts
int failedAttempts = 0;
bool isLockedOut = false;
DateTime? lastFailedLogin = null;

using (MySqlConnection conn = new MySqlConnection(connectionString))
{
    conn.Open();
    string query = "SELECT FailedLoginAttempts, LastFailedLogin FROM Users WHERE Email = @Email";
    using (MySqlCommand cmd = new MySqlCommand(query, conn))
    {
        cmd.Parameters.AddWithValue("@Email", email);
        using (MySqlDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                reader.Read();
                failedAttempts = reader["FailedLoginAttempts"] != DBNull.Value ? Convert.ToInt32(reader["FailedLoginAttempts"]) : 0;
                lastFailedLogin = reader["LastFailedLogin"] != DBNull.Value ? (DateTime?)reader["LastFailedLogin"] : null;
            }
        }
    }

    if (failedAttempts >= MaxAttempts && lastFailedLogin.HasValue)
    {
        TimeSpan timeSinceLastFailure = DateTime.Now - lastFailedLogin.Value;
        if (timeSinceLastFailure.TotalMinutes < LockoutMinutes)
        {
            isLockedOut = true;
        }
        else
        {
            ResetFailedAttempts(email, conn);
            failedAttempts = 0;
        }
    }

    if (isLockedOut)
    {
        lblMessage.CssClass = "text-danger";
        lblMessage.Text = "Too many failed attempts. Please try again later.";
        lblAttempts.Text = "";
        return;
    }
}
```

Figure 60 - Validating the users login attempt record

This section validates the user's login attempt record. It checks to see whether the user has been locked out for attempting too many times and discovers that there have been multiple unsuccessful attempts to log in. Users who haven't been able to log in three times in the past fifteen minutes are momentarily shut off. If the lockout interval is over, their failed attempts are cleared. If the user is locked out, he is shown an error message, telling him to try later. This secures the login procedure against brute-force attacks.

```
// Attempt to login and retrieve user info
if (TryLoginUser(email, password, out string userName))
{
    // Reset failed attempts on successful login
    using (MySqlConnection conn = new MySqlConnection(connectionString))
    {
        conn.Open();
        ResetFailedAttempts(email, conn);
    }

    // Store in session
    Session["UserEmail"] = email;
    Session["UserName"] = userName;

    // Dynamic redirection based on enrollment status
    try
    {
        string checkEnrollmentsQuery = "SELECT COUNT(*) FROM Enrollments e " +
                                       "JOIN Users u ON e.UserID = u.UserID " +
                                       "WHERE u.Email = @Email";

        using (MySqlConnection conn = new MySqlConnection(connectionString))
        {
            conn.Open();
            using (MySqlCommand cmd = new MySqlCommand(checkEnrollmentsQuery, conn))
            {
                cmd.Parameters.AddWithValue("@Email", email);
                int enrollmentCount = Convert.ToInt32(cmd.ExecuteScalar());
                if (enrollmentCount == 0)
                {
                    Response.Redirect("~/Pages/Courses.aspx", false);
                }
                else
                {
                    Response.Redirect("~/Pages/Dashboard.aspx", false);
                }
            }
        }
    }
    catch (Exception ex)
    {
        lblMessage.CssClass = "text-danger";
        lblMessage.Text = "Error checking enrollments: " + ex.Message;
        return;
    }
}
else
{
```

Figure 61 - Verifying users credentials

The code takes care of the login procedure by verifying the user credentials. If the login is successful, the previous failed attempts are cleared along with the user's email address and name being saved to the session. It next verifies whether the user is enrolled or not. Depending upon their enrolment, the user is either sent to the courses or to their dashboard. Upon failure in this procedure, an error message is also shown to the user. This procedure ensures that the user is routed to the right page once he has logged in based upon the user's existing enrolment.

Login.aspx.cs

UniTechCollegeSystem | UniTechCollegeSystem.Pages.Login | btnLogin_Click(object sender, EventArgs e)

```
129             // Update failed attempts and display remaining attempts
130             failedAttempts++;
131             int attemptsRemaining = MaxAttempts - failedAttempts;
132
133             using (MySqlConnection conn = new MySqlConnection(connectionString))
134             {
135                 conn.Open();
136                 string updateQuery = "UPDATE Users SET FailedLoginAttempts = @Attempts, LastFailedLogin = @LastFailedLogin WHERE Email = @Email";
137                 using (MySqlCommand cmd = new MySqlCommand(updateQuery, conn))
138                 {
139                     cmd.Parameters.AddWithValue("@Attempts", failedAttempts);
140                     cmd.Parameters.AddWithValue("@LastFailedLogin", DateTime.Now);
141                     cmd.Parameters.AddWithValue("@Email", email);
142                     cmd.ExecuteNonQuery();
143                 }
144             }
145
146             lblMessage.CssClass = "text-danger";
147             lblMessage.Text = "Invalid email or password. Please try again.";
148             lblAttempts.CssClass = "text-warning";
149             if (attemptsRemaining > 0)
150             {
151                 lblAttempts.Text = $"{attemptsRemaining} Attempt{(attemptsRemaining == 1 ? "" : "s")} Remaining";
152             }
153             else
154             {
155                 lblMessage.Text = "Too many failed attempts. Please try again later.";
156                 lblAttempts.Text = "";
157             }
158         }
159     }
160
```

Figure 62 - Code for failed login attempts

This part of the code increments the failed login attempts number whenever a user inputs invalid credentials. It stores the most recent attempt in the database and informs the user about the remaining attempts. If the maximum number of attempts is exceeded, a lockout notice is displayed. This prevents other users' unauthorised attempts to log in by restricting repeated failed attempts.

Login.aspx.cs

UniTechCollegeSystem | UniTechCollegeSystem.Pages.Login | btnLogin_Click(object sender, EventArgs e)

```
161     // Attempt to login user and return user name if successful
        1 reference
162     private bool TryLoginUser(string email, string password, out string userName)
163     {
164         bool isValid = false;
165         userName = string.Empty;
166
167         string query = "SELECT FullName, Password FROM Users WHERE Email = @Email";
168
169         try
170         {
171             using (MySqlConnection conn = new MySqlConnection(connectionString))
172             {
173                 conn.Open();
174                 using (MySqlCommand cmd = new MySqlCommand(query, conn))
175                 {
176                     cmd.Parameters.AddWithValue("@Email", email);
177
178                     using (MySqlDataReader reader = cmd.ExecuteReader())
179                     {
180                         if (reader.HasRows)
181                         {
182                             reader.Read();
183                             string storedHashedPassword = reader["Password"].ToString();
184                             string fullName = reader["FullName"].ToString();
185
186                             if (VerifyPasswordHash(password, storedHashedPassword))
187                             {
188                                 userName = fullName;
189                                 isValid = true;
190                             }
191                         }
192                     }
193                 }
194             }
195         }
196         catch (Exception ex)
197         {
198             lblMessage.CssClass = "text-danger";
199             lblMessage.Text = "Login error: " + ex.Message;
200             lblAttempts.Text = "";
201         }
202
203         return isValid;
204     }
```

Figure 63 - Validating the users email and password to what is stored in the database

The code validates whether the user's mail address and password are consistent with the information stored in the database. It hashes the password for secure verification, then, upon successful verification, extracts the user's full name. This is a critical process for logging in and authenticating the user before providing them with access to the system.

```
Login.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Login | TryLoginUser(string email, string password, out string userName)
205
206         // Reset failed login attempts
            2 references
207         private void ResetFailedAttempts(string email, MySqlConnection conn)
208         {
209             string resetQuery = "UPDATE Users SET FailedLoginAttempts = 0, LastFailedLogin = NULL WHERE Email = @Email";
210             using (MySqlCommand cmd = new MySqlCommand(resetQuery, conn))
211             {
212                 cmd.Parameters.AddWithValue("@Email", email);
213                 cmd.ExecuteNonQuery();
214             }
215         }
216
217         // Verify password hash
            1 reference
218         private bool VerifyPasswordHash(string enteredPassword, string storedHashedPassword)
219         {
220             string enteredHashedPassword = HashPassword(enteredPassword);
221             return enteredHashedPassword == storedHashedPassword;
222         }
223
224         // Hash password using SHA256
            1 reference
225         private string HashPassword(string password)
226         {
227             using (SHA256 sha256 = SHA256.Create())
228             {
229                 byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
230                 return Convert.ToBase64String(hashBytes);
231             }
232         }
233     }
234 }
```

Figure 64 - Protecting the login process

This section of the code protects the login process. It reinitialises the failed logins count upon successful login, verifies whether the given password is equal to the saved (hashed) password, and ensures passwords are encrypted with SHA256 before verification or storage.

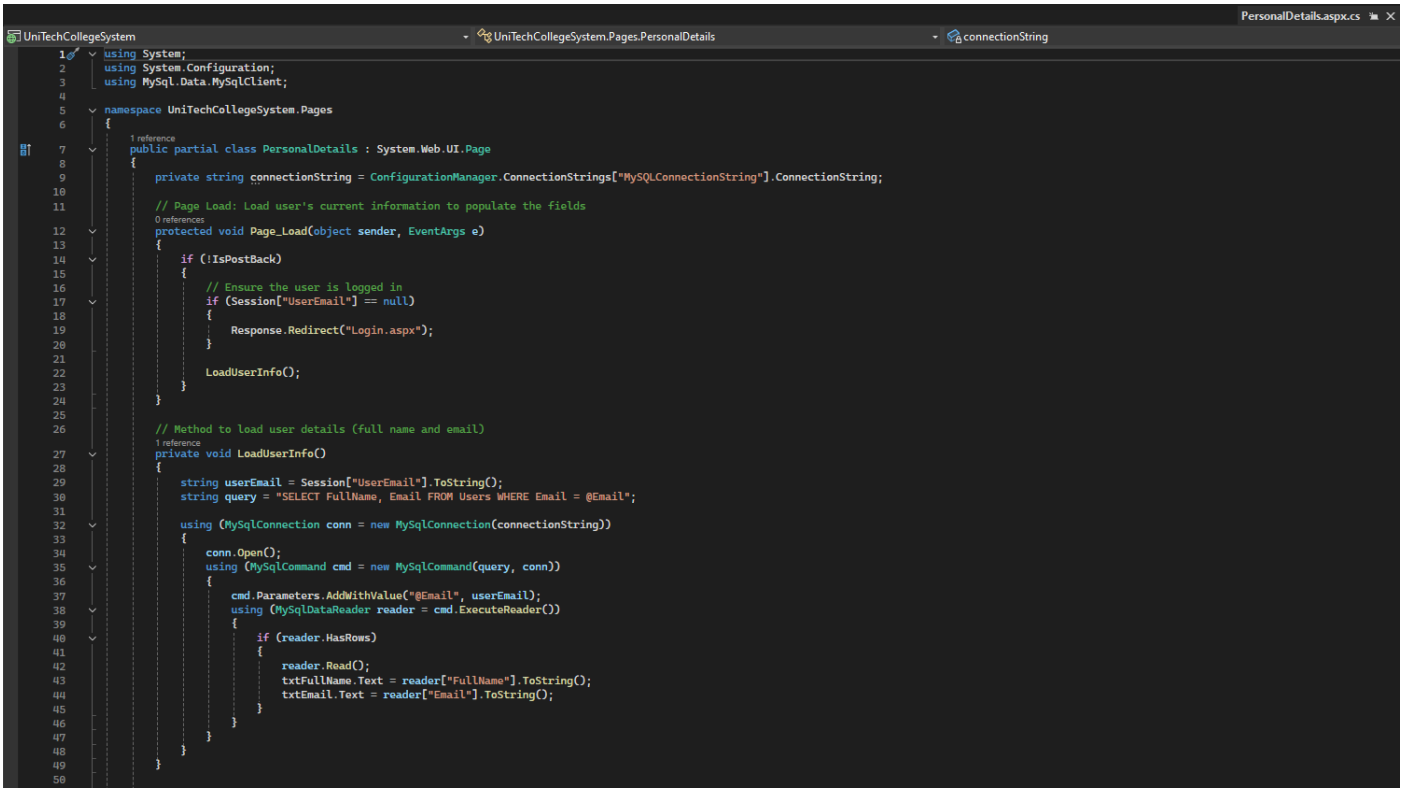
Appendix 14 – PersonalDetails.aspx Code

```
PersonalDetails.aspx
1   <%@ Page Title="Edit Personal Details" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="PersonalDetails.aspx.cs" Inherits="UniTechCollegeSystem.Pages.PersonalDetails" %>
2
3   <asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
4       <div class="container mt-5">
5           <h2 class="mb-4 text-center text-primary">Edit Your Personal Details</h2>
6
7           <!-- Form for editing personal information -->
8           <div class="mb-3">
9               <asp:Label ID="lblFullName" runat="server" Text="Full Name" />
10              <asp:TextBox ID="txtFullName" runat="server" CssClass="form-control" />
11          </div>
12          <div class="mb-3">
13              <asp:Label ID="lblEmail" runat="server" Text="Email" />
14              <asp:TextBox ID="txtEmail" runat="server" TextMode="Email" CssClass="form-control" />
15          </div>
16
17          <!-- Save button to update the details -->
18          <asp:Button ID="btnSave" runat="server" Text="Save Changes" CssClass="btn btn-primary" OnClick="btnSave_Click" />
19      </div>
20  </asp:Content>
```

Figure 65 - Layout for the personal information of users(viewing and editing)

The below code is for the dynamic generation of the users' personal information layout for viewing and editing their full name and email. It is done with a styled form that has input fields along with a save button, linking to the backend for where the updates are processed. The style is designed based on the site's primary layout utilising the master page.

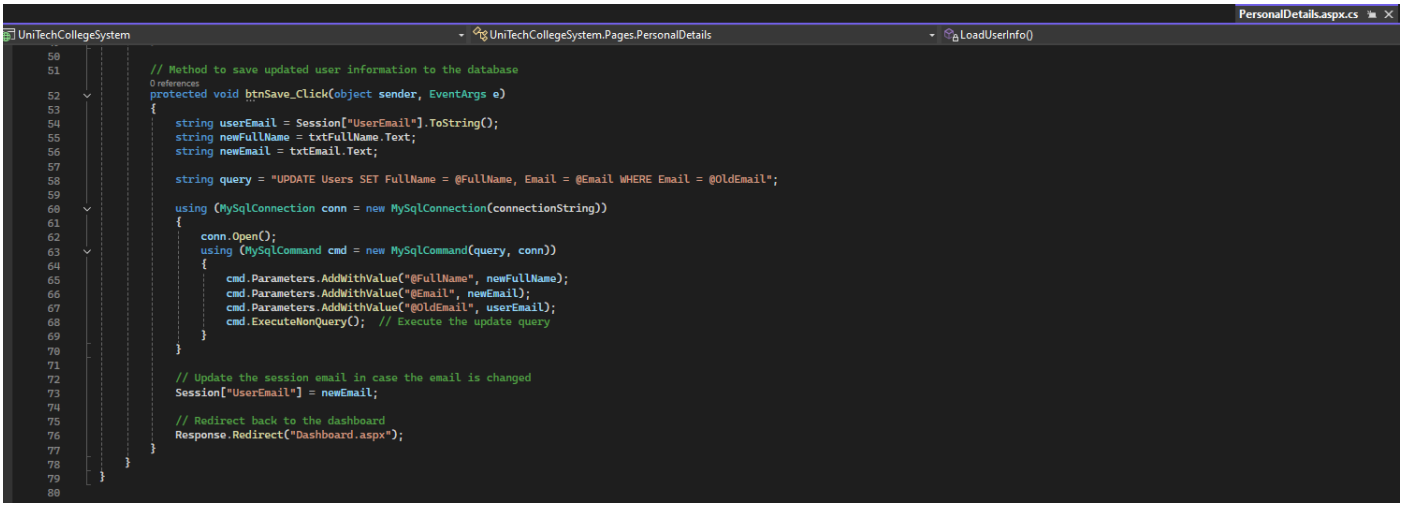
Appendix 15 – PersonalDetails.aspx.cs Code




```
using System;
using System.Configuration;
using MySql.Data.MySqlClient;

namespace UniTechCollegeSystem.Pages
{
    1 reference
    public partial class PersonalDetails : System.Web.UI.Page
    {
        private string connectionString = ConfigurationManager.ConnectionStrings["MySQLConnectionString"].ConnectionString;

        // Page Load: Load user's current information to populate the fields
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // Ensure the user is logged in
                if (Session["UserEmail"] == null)
                {
                    Response.Redirect("Login.aspx");
                }

                LoadUserInfo();
            }
        }

        // Method to load user details (full name and email)
        1 reference
        private void LoadUserInfo()
        {
            string userEmail = Session["UserEmail"].ToString();
            string query = "SELECT FullName, Email FROM Users WHERE Email = @Email";

            using (MySqlConnection conn = new MySqlConnection(connectionString))
            {
                conn.Open();
                using (MySqlCommand cmd = new MySqlCommand(query, conn))
                {
                    cmd.Parameters.AddWithValue("@Email", userEmail);
                    using (MySqlDataReader reader = cmd.ExecuteReader())
                    {
                        if (reader.HasRows)
                        {
                            reader.Read();
                            txtFullName.Text = reader["FullName"].ToString();
                            txtEmail.Text = reader["Email"].ToString();
                        }
                    }
                }
            }
        }
```

Figure 66 - Backend for personal details page

The above code manages the backend of the Personal Details page. Upon loading the page, the code verifies whether the user is logged in or not and then loads the user's full name and email address from the database to populate the form. It makes it easy for users to see their personal data and also update their information.




```
        // Method to save updated user information to the database
        0 references
        protected void btnSave_Click(object sender, EventArgs e)
        {
            string userEmail = Session["UserEmail"].ToString();
            string newFullName = txtFullName.Text;
            string newEmail = txtEmail.Text;

            string query = "UPDATE Users SET FullName = @FullName, Email = @Email WHERE Email = @OldEmail";

            using (MySqlConnection conn = new MySqlConnection(connectionString))
            {
                conn.Open();
                using (MySqlCommand cmd = new MySqlCommand(query, conn))
                {
                    cmd.Parameters.AddWithValue("@FullName", newFullName);
                    cmd.Parameters.AddWithValue("@Email", newEmail);
                    cmd.Parameters.AddWithValue("@OldEmail", userEmail);
                    cmd.ExecuteNonQuery();  // Execute the update query
                }
            }

            // Update the session email in case the email is changed
            Session["UserEmail"] = newEmail;

            // Redirect back to the dashboard
            Response.Redirect("Dashboard.aspx");
        }
    }
}
```

Figure 67 - Saving the changes that was made by the user for their personal information

The code is utilised to save the changes made by a user to their personal information. Upon clicking the "Save Changes" button, the user's full name and the email are updated within the database. If the email is updated, the session is likewise updated to include the changes, and the user is forwarded to the dashboard.

Appendix 16 – Register.aspx Code

Register.aspx

```
<%@ Page Title="Register" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeBehind="Register.aspx.cs" Inherits="UniTechCollegeSystem.Pages.Register" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div class="container mt-5">
        <h2 class="mb-4 text-center text-primary">Create Your Account</h2>

        <asp:Label ID="lblMessage" runat="server" CssClass="text-danger mb-3 d-block" />

        <!-- Full Name -->
        <div class="mb-3">
            <asp:Label ID="lblFullName" runat="server" Text="Full Name" />
            <asp:TextBox ID="txtFullName" runat="server" CssClass="form-control" />
        </div>

        <!-- Email -->
        <div class="mb-3">
            <asp:Label ID="lblEmail" runat="server" Text="Email" />
            <asp:TextBox ID="txtEmail" runat="server" TextMode="Email" CssClass="form-control" />
        </div>

        <!-- Password -->
        <div class="mb-3">
            <asp:Label ID="lblPassword" runat="server" Text="Password" />
            <asp:TextBox ID="txtPassword" runat="server" TextMode="Password" CssClass="form-control" ClientIDMode="Static"
                onfocus="showPasswordHint()" onblur="hidePasswordHint()" oninput="validatePassword()" />

            <!-- New feature: Password strength indicator -->
            <asp:Label ID="lblPasswordStrength" runat="server" CssClass="mt-2 d-block" />
            <!-- End of new feature -->

            <!-- Password hint (original) enhanced with requirements guide -->
            <span id="passwordHint" style="display:none; color:blue; font-size: 0.9em;">
                Password must be at least 8 characters, include 1 capital letter, 1 special character (!@#$%^&*(),.?":{}|<>), and 1 number.
            </span>
            <!-- New feature: Password requirements guide -->
            <div class="mt-2" id="passwordRequirements" style="display:none;">
                <p><strong>Password Requirements:</strong></p>
                <ul style="list-style-type: none; padding-left: 0;">
                    <li id="reqLength">At least 8 characters</li>
                    <li id="reqCapital">At least 1 capital letter</li>
                    <li id="reqSpecial">At least 1 special character (!@#$%^&*(),.?":{}|<>)</li>
                    <li id="reqNumber">At least 1 number</li>
                </ul>
            </div>
            <!-- End of new feature -->
        </div>

        <!-- Register button -->
        <asp:Button ID="btnRegister" runat="server" Text="Register"
            CssClass="btn btn-primary w-100" OnClick="btnRegister_Click" />
    </div>
```

Figure 68 - Structuring the registration page for users

The code creates the structure of the registration page where the new user can create an account with the full name, email address, and password. It has useful features such as the hint for the password to assist the user in setting a secure password and a strength meter so that the process of registering is simple, user-friendly, and secure.

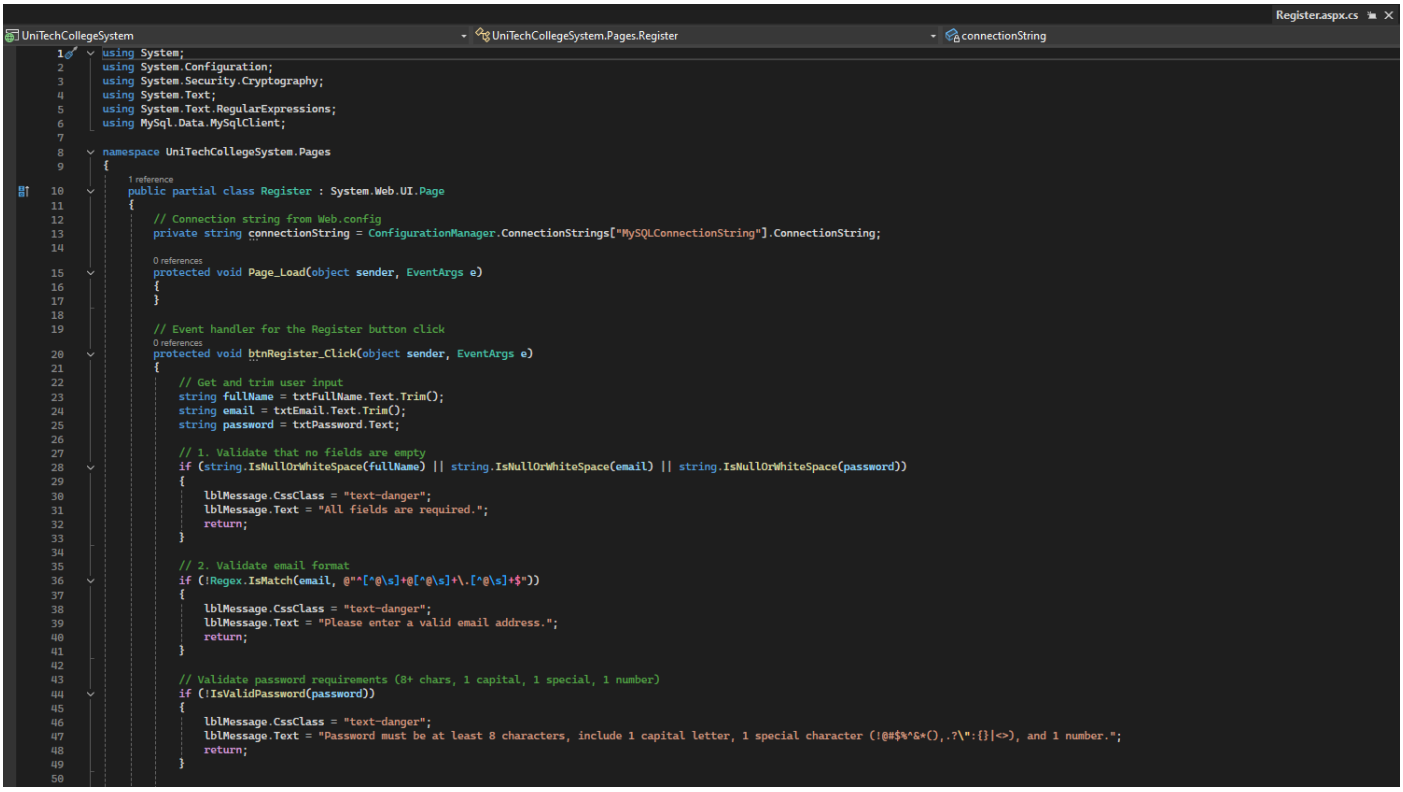
Register.aspx

```
    <!-- JavaScript for hint toggle and password validation -->
    <script type="text/javascript">
        function showPasswordHint() {
            document.getElementById("passwordHint").style.display = "inline";
            document.getElementById("passwordRequirements").style.display = "block";
        }

        function hidePasswordHint() {
            document.getElementById("passwordHint").style.display = "none";
            document.getElementById("passwordRequirements").style.display = "none";
        }

        function validatePassword() {
            const password = document.getElementById('txtPassword').value;
            const strengthLabel = document.getElementById('<%= lblPasswordStrength.ClientID %>');

            // Regex for requirements
            const lengthCheck = password.length >= 8;
            const capitalCheck = /[A-Z]/.test(password);
            const specialCheck = /[!@#$%^&*(),.?":{}|<>]/.test(password);
            const numberCheck = /[0-9]/.test(password);

            // Update requirements guide
            document.getElementById('reqLength').style.color = lengthCheck ? 'green' : 'red';
            document.getElementById('reqLength').innerHTML = lengthCheck ? '✔ At least 8 characters' : '✖ At least 8 characters';
            document.getElementById('reqCapital').style.color = capitalCheck ? 'green' : 'red';
            document.getElementById('reqCapital').innerHTML = capitalCheck ? '✔ At least 1 capital letter' : '✖ At least 1 capital letter';
            document.getElementById('reqSpecial').style.color = specialCheck ? 'green' : 'red';
            document.getElementById('reqSpecial').innerHTML = specialCheck ? '✔ At least 1 special character' : '✖ At least 1 special character (!@#$%^&*(),.?":{}|<>)';
            document.getElementById('reqNumber').style.color = numberCheck ? 'green' : 'red';
            document.getElementById('reqNumber').innerHTML = numberCheck ? '✔ At least 1 number' : '✖ At least 1 number';

            // Calculate strength
            let criteriaMet = [lengthCheck, capitalCheck, specialCheck, numberCheck].filter(Boolean).length;
            if (password.length < 8 || criteriaMet <= 2) {
                strengthLabel.style.color = 'red';
                strengthLabel.textContent = 'Weak';
            } else if (criteriaMet >= 3 && password.length < 12) {
                strengthLabel.style.color = 'orange';
                strengthLabel.textContent = 'Medium';
            } else {
                strengthLabel.style.color = 'green';
                strengthLabel.textContent = 'Strong';
            }
        }
    </script>
</asp:Content>
```

Figure 69 - Password hint and password strength done in JavaScript

This JavaScript provides the form with a password hint and instant feedback on the password strength. It verifies whether the password is fulfilling the core requirements for security like length, capital, special character, and number and also updates the advice in real time so that users have an easier time creating secure passwords.

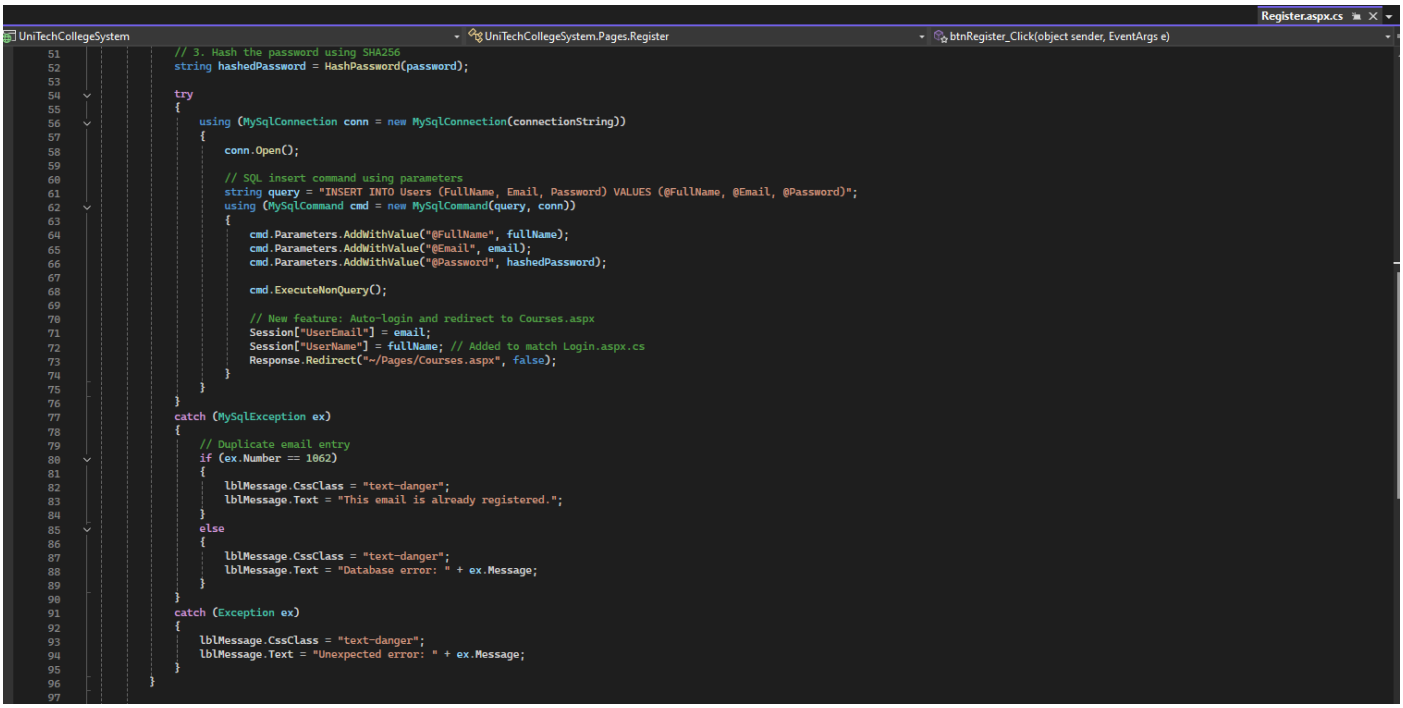
Appendix 17 – Register.aspx.cs Code



```
Register.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Register | connectionString

using System;
using System.Configuration;
using System.Security.Cryptography;
using System.Text;
using System.Text.RegularExpressions;
using MySql.Data.MySqlClient;

namespace UniTechCollegeSystem.Pages
{
    1 reference
    public partial class Register : System.Web.UI.Page
    {
        // Connection string from Web.config
        private string connectionString = ConfigurationManager.ConnectionStrings["MySQLConnectionString"].ConnectionString;

        0 references
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        // Event handler for the Register button click
        0 references
        protected void btnRegister_Click(object sender, EventArgs e)
        {
            // Get and trim user input
            string fullName = txtFullName.Text.Trim();
            string email = txtEmail.Text.Trim();
            string password = txtPassword.Text;

            // 1. Validate that no fields are empty
            if (string.IsNullOrWhiteSpace(fullName) || string.IsNullOrWhiteSpace(email) || string.IsNullOrWhiteSpace(password))
            {
                lblMessage.CssClass = "text-danger";
                lblMessage.Text = "All fields are required.";
                return;
            }

            // 2. Validate email format
            if (!Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$"))
            {
                lblMessage.CssClass = "text-danger";
                lblMessage.Text = "Please enter a valid email address.";
                return;
            }

            // Validate password requirements (8+ chars, 1 capital, 1 special, 1 number)
            if (!IsValidPassword(password))
            {
                lblMessage.CssClass = "text-danger";
                lblMessage.Text = "Password must be at least 8 characters, include 1 capital letter, 1 special character (!@#$%^&*(),.?\":{}|<>), and 1 number.";
                return;
            }
```

Figure 70 - C# code for handling the server side of registering

The following is the C# code for handling the server-side process of registering. It checks that the fields are filled, is the right format for the email, and that the password is secure. If they are appropriate, the information may safely proceed to create the user accounts. It has been meant to keep the user input structured and safe, which matches the project goal to deliver a usable and secure registrations option.



```
Register.aspx.cs
UniTechCollegeSystem | UniTechCollegeSystem.Pages.Register | btnRegister_Click(object sender, EventArgs e)

            // 3. Hash the password using SHA256
            string hashedPassword = HashPassword(password);

            try
            {
                using (MySqlConnection conn = new MySqlConnection(connectionString))
                {
                    conn.Open();

                    // SQL insert command using parameters
                    string query = "INSERT INTO Users (FullName, Email, Password) VALUES (@FullName, @Email, @Password)";
                    using (MySqlCommand cmd = new MySqlCommand(query, conn))
                    {
                        cmd.Parameters.AddWithValue("@FullName", fullName);
                        cmd.Parameters.AddWithValue("@Email", email);
                        cmd.Parameters.AddWithValue("@Password", hashedPassword);

                        cmd.ExecuteNonQuery();

                        // New feature: Auto-login and redirect to Courses.aspx
                        Session["UserEmail"] = email;
                        Session["UserName"] = fullName; // Added to match Login.aspx.cs
                        Response.Redirect("~/Pages/Courses.aspx", false);
                    }
                }
            }
            catch (MySqlException ex)
            {
                // Duplicate email entry
                if (ex.Number == 1062)
                {
                    lblMessage.CssClass = "text-danger";
                    lblMessage.Text = "This email is already registered.";
                }
                else
                {
                    lblMessage.CssClass = "text-danger";
                    lblMessage.Text = "Database error: " + ex.Message;
                }
            }
            catch (Exception ex)
            {
                lblMessage.CssClass = "text-danger";
                lblMessage.Text = "Unexpected error: " + ex.Message;
            }
        }
```

Figure 71 - Storing new user information to the database

The following code safely stores the new user information to the database. It encrypts the password with SHA256, then stores the user's name, the user's email address, and the encrypted password. Upon successful registration, the user is logged in and immediately redirected to the courses page. It also validates the existence of the user's email address to avoid duplication across different records. It also handles possible error occurrences, making the process user-friendly and secure key achievement of the task.

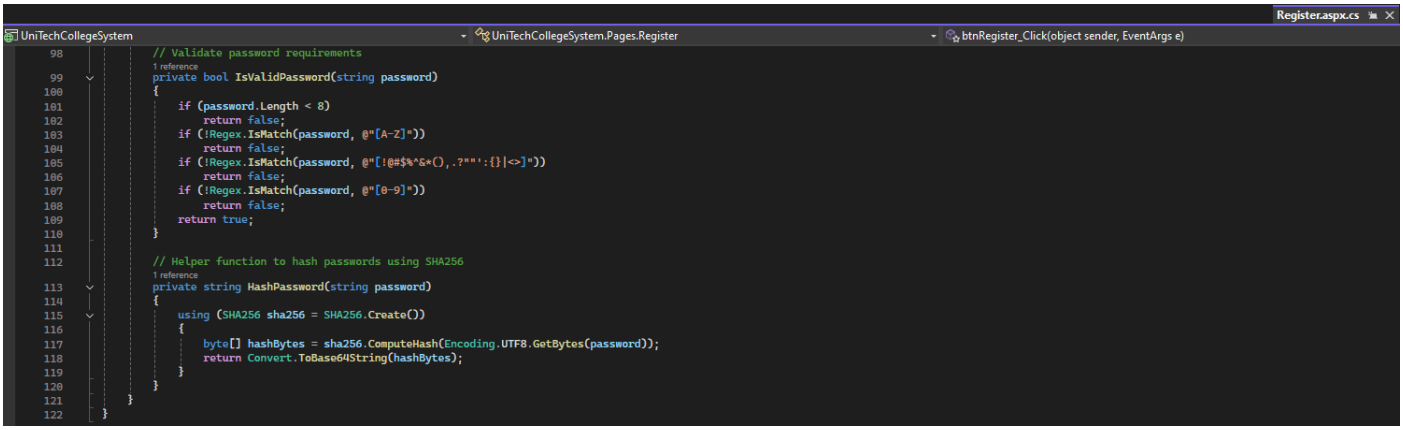


Figure 72 - Functions for IsValidPassword and HashPassword

This component of the code has two essential tasks. The first method, IsValidPassword, examines the password fulfils specified criteria: the password must comprise 8 letters, at least one capital character, at least one special character, and at least one number. The second approach, HashPassword, properly safeguards the password using the SHA256 algorithm before to the hash being added to the database. These two techniques both certify the password to be secure and true to be used in the registration procedure.

Appendix 18 – MySQL Workbench Database Design Code

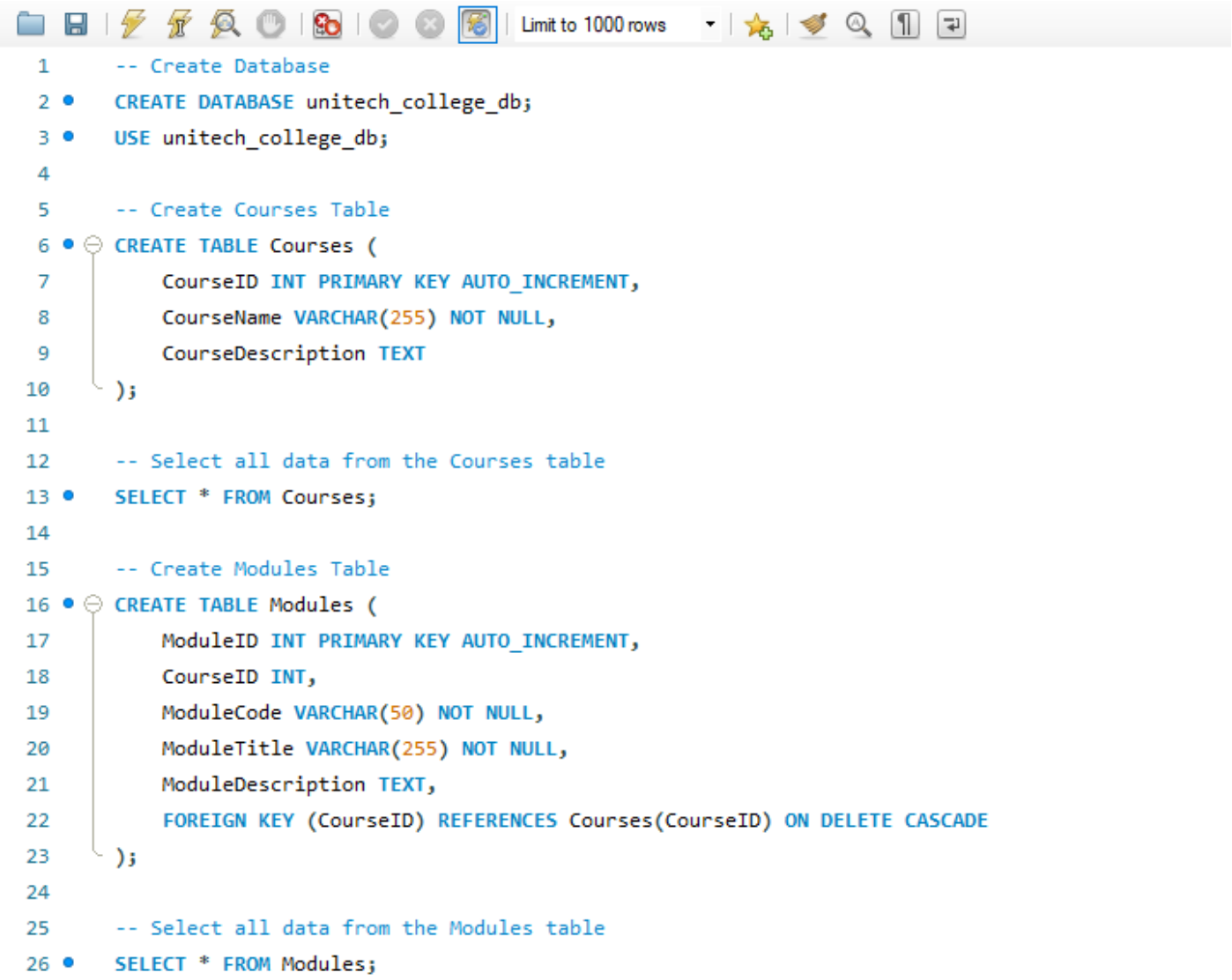
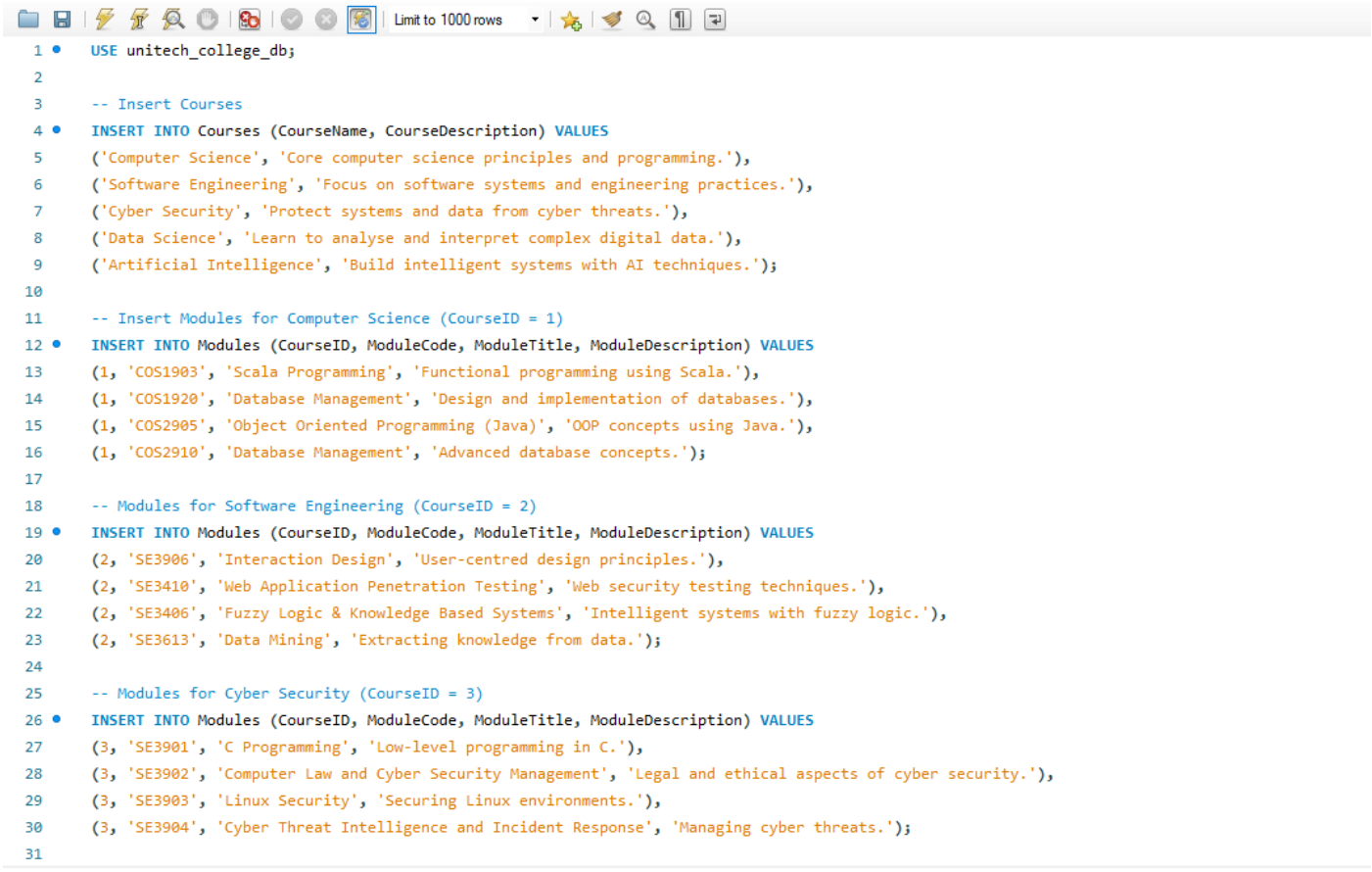


Figure 73 - SQL Database Schema for Courses and Modules Tables

Figure 73 depicts SQL database schema for a sample educational institution, Unitech College. The example illustrates constructing a database named unitech_college_db and generating two tables connected to each other, i.e., Courses and Modules. The Courses database consists of a CourseID (auto-incrementing integer as a primary key), CourseName (VARCHAR field having a maximum length of 255 characters, not null), and CourseDescription (TEXT field). A SELECT statement is demonstrated for bringing all courses from the Courses table.

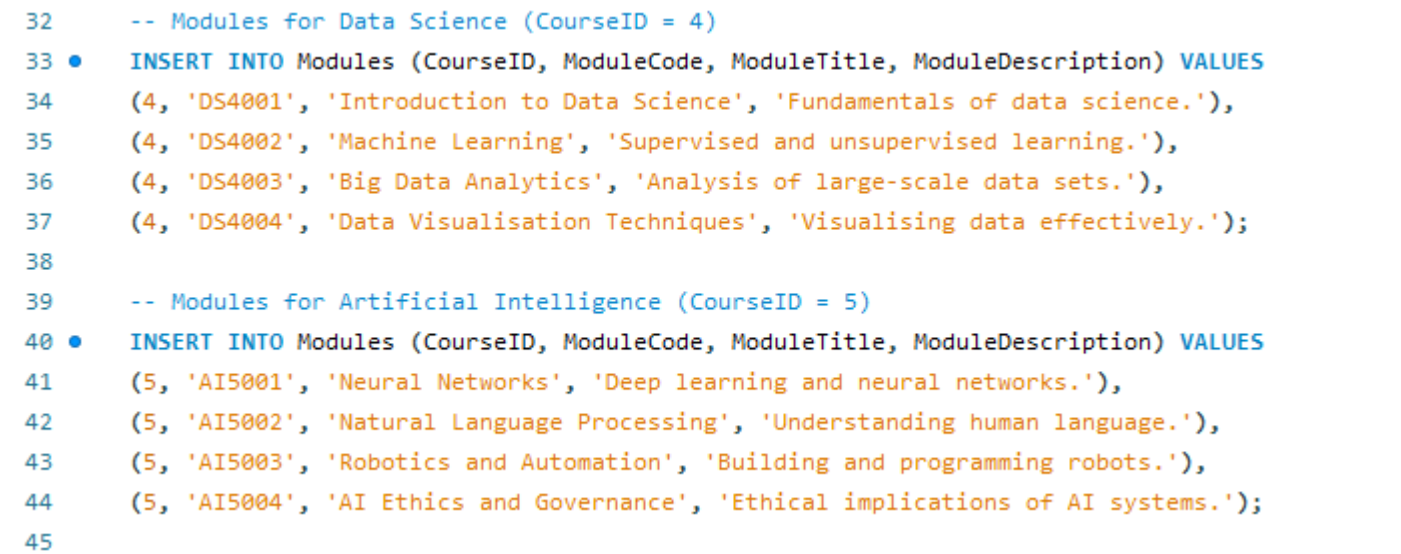
Modules consists of a number of crucial fields. ModuleCode, a required string field with a length restriction of 50, and ModuleTitle, a vital field with a length maximum of 255, are both needed fields. ModuleDescription is a writing type that enable more thorough writing. ModuleID, a unique, auto-incrementing integer, identifies each module, while being the principal key. The fourth one, CourseID, works as a foreign key referencing to Courses, with a cascade erasure mechanism guaranteeing linked modules are erased in case a course is finished. To obtain every entry from the Modules table, a SELECT query is provided. This schema facilitates a relational framework under which modules belong to certain courses, maintaining data integrity and cascading deletion when a course is deleted.



```
USE unitech_college_db;

-- Insert Courses
INSERT INTO Courses (CourseName, CourseDescription) VALUES
('Computer Science', 'Core computer science principles and programming.'),
('Software Engineering', 'Focus on software systems and engineering practices.'),
('Cyber Security', 'Protect systems and data from cyber threats.'),
('Data Science', 'Learn to analyse and interpret complex digital data.'),
('Artificial Intelligence', 'Build intelligent systems with AI techniques.');

-- Insert Modules for Computer Science (CourseID = 1)
INSERT INTO Modules (CourseID, ModuleCode, ModuleTitle, ModuleDescription) VALUES
(1, 'COS1903', 'Scala Programming', 'Functional programming using Scala.'),
(1, 'COS1920', 'Database Management', 'Design and implementation of databases.'),
(1, 'COS2905', 'Object Oriented Programming (Java)', 'OOP concepts using Java.'),
(1, 'COS2910', 'Database Management', 'Advanced database concepts.');

-- Modules for Software Engineering (CourseID = 2)
INSERT INTO Modules (CourseID, ModuleCode, ModuleTitle, ModuleDescription) VALUES
(2, 'SE3906', 'Interaction Design', 'User-centred design principles.'),
(2, 'SE3410', 'Web Application Penetration Testing', 'Web security testing techniques.'),
(2, 'SE3406', 'Fuzzy Logic & Knowledge Based Systems', 'Intelligent systems with fuzzy logic.'),
(2, 'SE3613', 'Data Mining', 'Extracting knowledge from data.');

-- Modules for Cyber Security (CourseID = 3)
INSERT INTO Modules (CourseID, ModuleCode, ModuleTitle, ModuleDescription) VALUES
(3, 'SE3901', 'C Programming', 'Low-level programming in C.'),
(3, 'SE3902', 'Computer Law and Cyber Security Management', 'Legal and ethical aspects of cyber security.'),
(3, 'SE3903', 'Linux Security', 'Securing Linux environments.'),
(3, 'SE3904', 'Cyber Threat Intelligence and Incident Response', 'Managing cyber threats.');
```

Figure 74 - Insert Data into the Courses and Modules Tables

Figure 74 provides a list of SQL insert statements for populating the Courses and Modules tables in the unitech_college_db database. Five courses, Computer Science, Software Engineering, Cyber Security, Data Science, and Artificial Intelligence, each given a brief description, are inserted into the Courses table. Specific modules corresponding to these courses through CourseID are inserted into the Modules table. Computer Science (CourseID = 1) modules are Scala Programming, Database Management, Object-Oriented Programming (Java), and Advanced Database Concepts. Modules for Software Engineering (CourseID = 2) are Interaction Design, Web Application Penetration Testing, Fuzzy Logic & Knowledge Based Systems, and Data Mining. Modules for Cyber Security (CourseID = 3) are C Programming, Computer Law and Cyber Security Management, Linux Security, and Cyber Threat Intelligence and Incident Response.



```
-- Modules for Data Science (CourseID = 4)
INSERT INTO Modules (CourseID, ModuleCode, ModuleTitle, ModuleDescription) VALUES
(4, 'DS4001', 'Introduction to Data Science', 'Fundamentals of data science.'),
(4, 'DS4002', 'Machine Learning', 'Supervised and unsupervised learning.'),
(4, 'DS4003', 'Big Data Analytics', 'Analysis of large-scale data sets.'),
(4, 'DS4004', 'Data Visualisation Techniques', 'Visualising data effectively.');

-- Modules for Artificial Intelligence (CourseID = 5)
INSERT INTO Modules (CourseID, ModuleCode, ModuleTitle, ModuleDescription) VALUES
(5, 'AI5001', 'Neural Networks', 'Deep learning and neural networks.'),
(5, 'AI5002', 'Natural Language Processing', 'Understanding human language.'),
(5, 'AI5003', 'Robotics and Automation', 'Building and programming robots.'),
(5, 'AI5004', 'AI Ethics and Governance', 'Ethical implications of AI systems.');
```

Figure 75 - Additional Module Data Insertion for Data Science and Artificial Intelligence

Figure 75 is an ongoing SQL insertion into the Modules table in the unitech_college_db database, focussing particularly on Data Science and Artificial Intelligence courses.

The Data Science program (CourseID = 4) begins with Introduction to Data Science (DS4001) to develop the framework for dealing with data processes. Machine Learning (DS4002) addresses approaches effective in both controlled as well as unstructured learning. Big Data Analytics (DS4003) covers large-scale data administration, and Data Visualisation Techniques (DS4004) delivers an education on exhibiting information.

Likewise, the Artificial Intelligence course (CourseID = 5) comprises Neural Networks (AI5001) to master deep learning theories, Natural Language Processing (AI5002) to comprehend languages, and Robotics and Automation (AI5003) to study hands-on robotic design. Finally, it incorporates AI Ethics and Governance (AI5004) to investigate responsible AI development.

This information completes the relational aspect through associations between these modules and their respective courses.

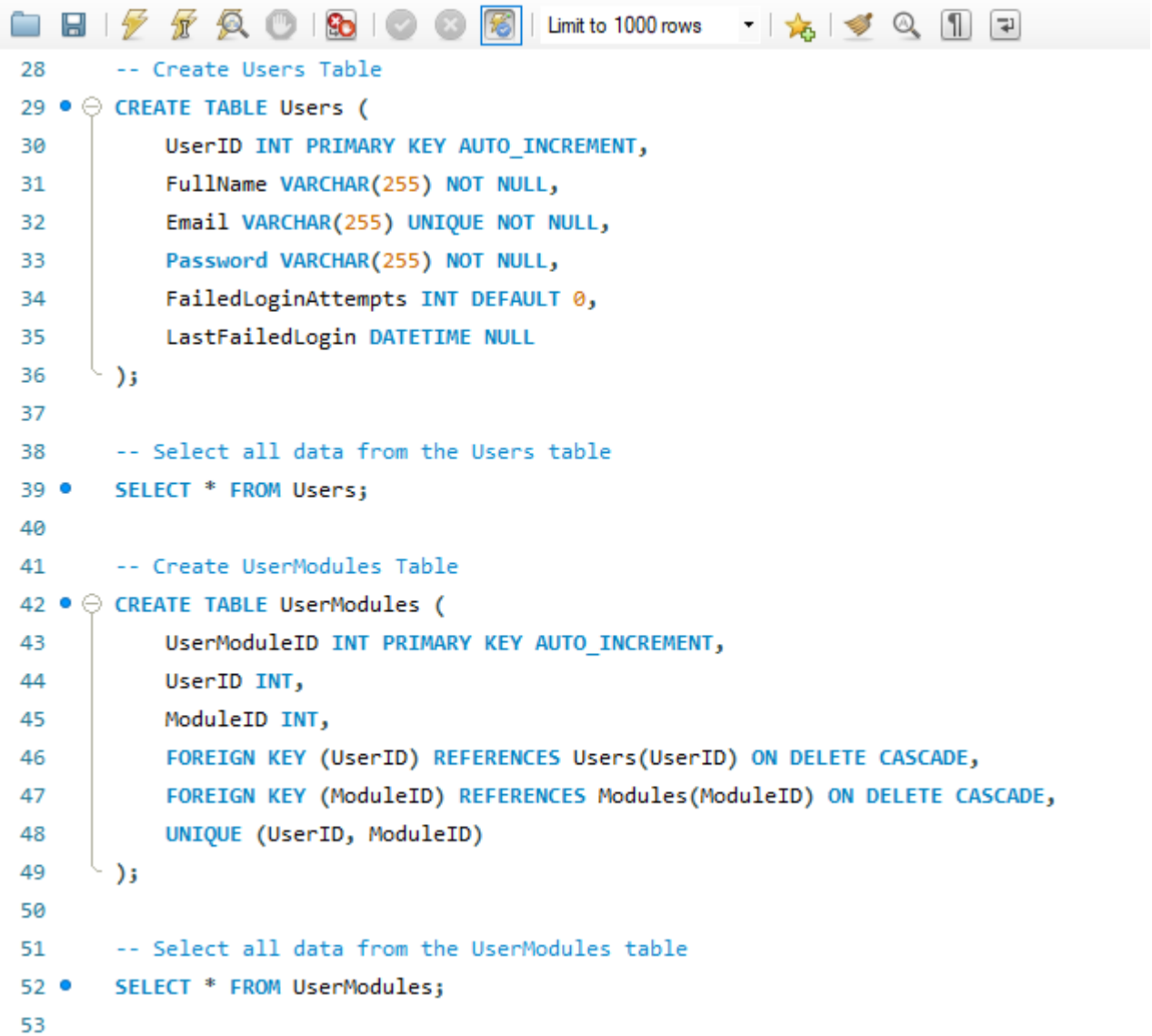


Figure 76 - SQL Database Schema for User and UserModules Tables

Figure 76 shows SQL schema for Users and UserModules table in unitech_college_db database. There are a few fields in the Users table to record account details. The UserID functions as a major key and is an autonumber. FullName is a mandatory text field with a length limit of 255. Email is a unique, non-nullable text field with a maximum of 255. The Password is a mandatory string field with a length restriction of 255. In order to assure protection against login, we have a FailedLoginAttempts field sets to 0, and LastFailedLogin to retain a date/time for the last unsuccessful login, with an option to leave this one empty. A SELECT statement reads all from this table. The UserModules table, which assigns users to modules, has UserModuleID (auto-

incrementing as a primary key), UserID and ModuleID (integers as foreign keys referencing Users and Modules, ON DELETE CASCADE), and a UNIQUE constraint between UserID and ModuleID pair. A SELECT statement is given for retrieving all from UserModules for supporting a relational structure for user-module enrolments.

```
54      -- Create Enrollments Table (User registering for Courses)
55      CREATE TABLE Enrollments (
56          EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
57          UserID INT,
58          CourseID INT,
59          FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE,
60          FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE
61      );
62
63      -- Select all data from the Enrollments table
64      SELECT * FROM Enrollments;
65
```

Figure 77 - SQL Database Schema for Enrolments

Figure 77 demonstrates the SQL configuration for the Enrollments table within the unitech_college database, which holds information about which users are enrolled in a given course. The EnrollmentID is an auto-incrementing number acting as a primary key, while UserID and CourseID are both integer foreign key fields accessing the Users and Courses columns, respectively. The two reference limitations both exploit the ON erase CASCADE rule to promptly erase connected data in cases where a user or a course is eliminated. A SELECT statement is given for selecting all Enrollments table records. This arrangement facilitates a relational relationship between users and courses they have enrolled for, maintaining data integrity through cascading deletion when corresponding records are deleted.