Software Requirements Specification

for

Course Enrollment System

Version 2.0 approved

Prepared by Jongho So

|  |  |
| --- | --- |
| Instructor: | Joseph Rangitsch |
| Course: | CST499 Capstone for Computer Software Technology  (CSF2420A) |
| Date: | June 17, 2024 |

Contents

Revisions iii

1 Introduction 1

1.1 Document Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 2

1.4 Product Scope 3

2 Overall Description 5

2.1 Product Perspective 5

2.2 Product Functionality 8

2.3 Users and Characteristics 11

2.4 Operating Environment 14

2.5 Design and Implementation Constraints 16

2.6 User Documentation 19

2.7 Assumptions and Dependencies 22

3 External Interface Requirements 24

3.1 User Interfaces 24

3.2 Hardware Interfaces 27

3.3 Software Interfaces 30

3.4 Communications Interfaces 33

4 System Features 36

4.1 New User Registration 36

4.2 User Login 39

4.3 Course Listing and Enrollment 42

4.4 Course Waitlist Management 45

4.5 Enrollment Cancellation 48

5 Other Non-functional Requirements 52

5.1 Performance Requirements 52

5.2 Safety Requirements 52

5.3 Security Requirements 53

5.4 Software Quality Attributes 53

5.5 Business Rules 54

6 Other Requirements 54

6.1 Glossary 54

6.2 Analysis Models 54

6.3 To Be Determined List 55

6.4 UML Diagrams 55

6.5 Test Strategy. 60

6.6 how to run a PHP file in XAMPP. 62

6.7 MySQL database and database connection custom class. 63

6.8 landing page, login page, and registration page 66

6.9 Create tables within the MySQL database. 91

6.10 list classes, add and delete classes from the schedule . 93

References and Acknowledgments 104

**Revisions**

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| 1.0 | Jongho Son | Requirements added | 05/18/2024 |
| 2.0 | Jongho Son | Adding PHP Code and explanation | 06/15/2024 |

# Introduction

## Document Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the requirements for the Online Course Registration System. This system will allow users to register new accounts, log in, enroll in courses, manage waitlists, and cancel enrollments. The goal is to provide a user-friendly platform for students and administrative users to manage course registrations efficiently.

## Document Conventions

This document adheres to a set of standard conventions to maintain clarity, readability, and consistency. Each major section title is bolded and numbered sequentially to provide a clear structure and easy navigation. Subsections are also bolded and numbered within their respective sections to further organize the document into manageable parts.

Requirements are presented in a structured format, where each requirement is individually listed and assigned a unique identifier for easy reference. The priority of each requirement is specified, indicating its relative importance in the overall system development. This prioritization helps stakeholders and developers focus on the most critical aspects first.

Additionally, technical terms and acronyms are defined upon their first use and listed in the glossary section to ensure common understanding. External references are cited in APA format to maintain academic integrity and allow readers to locate the original sources for further information. Visual elements such as tables and figures are numbered and captioned, providing visual aids to support the textual descriptions.

The document uses consistent font styles and sizes to differentiate between headings, subheadings, and body text. Bullet points and numbered lists are employed to break down complex information into more digestible parts. Emphasis is added where necessary through the use of bold, italics, and underlining to highlight key points or actions.

## Intended Audience and Reading Suggestions

This document is intended for a diverse audience, including developers, project managers, testers, documentation writers, and other stakeholders involved in the development and maintenance of the Online Course Registration System. Each group has specific sections of the document that will be most relevant to their roles and responsibilities (Oye, N. D., & Salleh, M. 2012).

Developers: Developers should focus on sections 3 through 5, which detail the functional and non-functional requirements. These sections provide the technical specifications, system features, and performance criteria that will guide the coding and implementation of the system. Developers will find the detailed requirements, interface specifications, and design constraints particularly useful for understanding what needs to be built and how it should function.

Testers: Testers should also concentrate on sections 3 through 5. These sections outline the criteria against which the system will be tested, including functional requirements, performance benchmarks, and security considerations. Testers will use this information to develop test cases and ensure that the system meets all specified requirements and performs reliably under expected conditions.

Project Managers: Project managers should review the entire document to gain a comprehensive understanding of the project scope, objectives, and requirements. Sections 1 and 2 will provide project managers with the overall context and high-level description of the system, including the product scope, user characteristics, and operating environment. This knowledge will assist in project planning, resource allocation, and progress monitoring.

Documentation Writers: Documentation writers should also review the entire document to ensure they are familiar with all aspects of the system. Understanding the detailed requirements and system features will enable them to create accurate and comprehensive user manuals, help guides, and other support materials. They should pay particular attention to sections 2 and 6, which describe the overall system and any additional requirements or constraints.

Other Stakeholders: Other stakeholders, such as business analysts, user representatives, and client stakeholders, may find value in different sections depending on their specific interests and needs. Sections 1 and 2 provide a high-level overview and context, which will be beneficial for understanding the system's purpose, scope, and intended use.

## Product Scope

The Online Course Registration System is designed to provide a comprehensive solution for managing student course registrations. The primary goal of the system is to streamline the process of course enrollment, making it efficient and user-friendly for students, faculty, and administrative staff. The system will offer a range of functionalities that cover the entire lifecycle of course registration and management.

Student Registration and Profile Management: The system will enable students to create new user accounts by registering with a unique ID and password. Upon registration, students will be able to create and manage their profiles, which will include essential information such as name, phone number, email address, and other relevant details. This profile information will be used for authentication and communication purposes.

Course Availability and Enrollment: Students will have access to a detailed list of available courses for each semester (spring, summer, and fall). The system will allow students to browse and search for courses based on various criteria such as course name, code, and department. Each course listing will display key information including course description, schedule, instructor, and the maximum number of students that can enroll. Students can then enroll in their selected courses directly through the system.

Waitlist Management: In cases where a course reaches its maximum enrollment capacity, the system will provide a waitlist functionality. Students who wish to enroll in a full course can add themselves to the waitlist. The system will automatically manage the waitlist, ensuring that students are notified and given the opportunity to enroll when a spot becomes available due to cancellations or drops.

Enrollment Management: Students will have the ability to view their current enrollments and make changes as needed. This includes the option to drop courses they are enrolled in. When a student drops a course, the system will automatically notify the first student on the waitlist, allowing them to enroll in the newly available spot. This dynamic management ensures optimal course occupancy and provides students with fair access to desired courses.

User Authentication and Security: The system will handle user authentication to ensure that only authorized users can access the course registration features. This includes secure login mechanisms using unique IDs and passwords. The system will employ robust security measures to protect user data and ensure privacy. All sensitive information will be encrypted, and the system will comply with relevant data protection regulations.

Notifications and Communication: The system will provide automated notifications to keep students informed about their enrollment status. Notifications will be sent for successful registrations, waitlist updates, and enrollment changes. These notifications can be configured to be delivered via email or through the system’s internal messaging platform. This ensures that students are always aware of their registration status and any important updates.

Administrative and Faculty Features: In addition to student functionalities, the system will offer features for faculty and administrative staff. Faculty members will be able to manage course details, including updating course information and adjusting enrollment limits. Administrative users will have access to a comprehensive dashboard for overseeing the entire registration process, managing user accounts, and generating reports on enrollment statistics.

Scalability and Flexibility: The system is designed to be scalable to accommodate growing numbers of students and courses. It will be flexible enough to integrate with other university systems and adapt to changes in registration policies or procedures. This ensures that the system can evolve with the institution’s needs and continue to provide efficient service.

# Overall Description

## Product Perspective.

The Online Course Registration System is envisioned as a standalone application specifically designed to provide a seamless and efficient course registration experience for students, faculty, and administrative staff. This system will be accessible through web browsers, ensuring that users can interact with it from any location with an internet connection, using various devices such as desktops, laptops, tablets, and smartphones.

Accessibility and User Interface: The system’s user interface will be web-based, providing a consistent and user-friendly experience across different web browsers including Chrome, Firefox, Safari, and Edge. The interface will be designed with modern web standards and best practices to ensure compatibility and responsiveness, adapting to various screen sizes and resolutions. This ensures that users can access the system conveniently, whether they are using a desktop computer in a library or a mobile device on the go (Almasri, H., & El-Sayed, M. 2016).

Backend Database Management: At the core of the system is a robust backend database that will manage all critical data related to user accounts, course information, and enrollment records. The database will be designed to handle a large volume of transactions and data entries, ensuring reliability and performance. It will store detailed information about each course, including course codes, titles, descriptions, schedules, and instructor details. Additionally, the database will maintain user profiles, enrollment statuses, and waitlist positions.

Data Integrity and Security: The backend database will employ advanced data integrity mechanisms to ensure that all information remains accurate and consistent. This includes the use of constraints, triggers, and stored procedures to manage data relationships and enforce business rules. Security measures will be in place to protect sensitive data, including encryption of personal information and secure authentication protocols to prevent unauthorized access.

Interoperability and Integration: Although designed as a standalone application, the system will be built with interoperability in mind, allowing it to integrate with other institutional systems such as Student Information Systems (SIS), Learning Management Systems (LMS), and administrative databases. This will enable seamless data exchange and synchronization, reducing duplication of efforts and ensuring that all systems have up-to-date information. For instance, course enrollment data from the registration system can be automatically reflected in the LMS, allowing students to access course materials and participate in online classes without additional steps (Yusof, N. M., & Wahid, N. 2014).

Scalability and Performance: The system will be designed to scale efficiently, accommodating an increasing number of users and courses as the institution grows. The architecture will support load balancing and distributed computing to handle peak usage times, such as during registration periods. Performance optimization techniques, including database indexing and query optimization, will ensure that the system responds quickly to user actions, providing a smooth and efficient experience.

Modularity and Flexibility: The system will be modular in design, allowing for easy updates and enhancements. New features and functionalities can be added without disrupting existing operations. For example, the system can be extended to support additional types of notifications, new user roles, or integration with third-party services. This flexibility ensures that the system can adapt to evolving requirements and continue to meet the needs of the institution.

User Experience and Support: The system will prioritize a positive user experience, with intuitive navigation, clear instructions, and helpful prompts. Comprehensive user support will be available, including online help documentation, FAQs, and a support ticket system. Training materials and tutorials will be provided to help users familiarize themselves with the system’s features and functionalities.

Compliance and Standards: The system will adhere to relevant industry standards and regulations, such as data protection laws (e.g., GDPR, FERPA) and accessibility standards (e.g., WCAG). This ensures that the system is not only secure and reliable but also accessible to all users, including those with disabilities (Naik, S. 2017).

## Product Functions.

The Online Course Registration System encompasses a wide range of functions designed to facilitate a smooth and efficient registration process for students, as well as providing necessary administrative tools for faculty and staff. The primary functions of the system include:

User Registration with Unique ID and Password: The system allows new users to create an account by registering with a unique user ID and password. During the registration process, users will provide essential personal information, such as their name, email address, and phone number. The system will ensure that each user ID is unique, preventing duplicate registrations. A confirmation email may be sent to validate the user’s email address, adding an extra layer of security and ensuring the accuracy of user information.

User Login and Profile Management: Once registered, users can log in to the system using their unique ID and password. Upon successful login, users will have access to their personal dashboard where they can manage their profile information. This includes updating personal details, changing passwords, and managing contact information. The profile management feature ensures that user data is kept current and secure.

Viewing Available Courses by Semester: The system provides a comprehensive course catalog that users can browse to view available courses for each semester (spring, summer, and fall). The course listings will include detailed information such as course titles, descriptions, schedules, prerequisites, instructor details, and the maximum number of students allowed to enroll. Users can search for courses using various filters, such as course code, department, and keywords, making it easy to find relevant courses that fit their schedules and academic needs.

Enrolling in Courses and Managing Enrollments: Users can enroll in courses directly through the system. The enrollment process involves selecting desired courses from the catalog and adding them to their schedule. The system will check for any prerequisites and ensure that the course is not already at maximum capacity. Once enrolled, users can view and manage their current enrollments through their dashboard. This includes the ability to view course details, check enrollment status, and receive updates on any changes.

Waitlist Management for Full Courses: For courses that reach their maximum enrollment capacity, the system offers a waitlist feature. Users who wish to enroll in a full course can add themselves to the waitlist. The waitlist management system tracks the order in which users join the list and notifies them automatically when a spot becomes available. This feature ensures that students have a fair chance of enrolling in popular courses and helps maximize course utilization.

Enrollment Cancellation and Automatic Notification to Waitlisted Users: Users can cancel their enrollment in a course through their dashboard. When a user drops a course, the system automatically updates the course availability and notifies the first user on the waitlist, if any, about the newly available spot. This automated notification system ensures efficient management of course enrollments and reduces administrative overhead. Users on the waitlist will receive an email or system notification informing them of the opportunity to enroll, along with instructions on how to complete the process.

Additional Functionalities:

* Course Feedback and Ratings: After completing a course, users can provide feedback and ratings. This helps future students make informed decisions and allows instructors to improve their course offerings.
* Academic Advising Integration: The system can integrate with academic advising tools, allowing students to plan their course schedules in alignment with their degree requirements and career goals.
* Reports and Analytics: Administrative users have access to detailed reports and analytics on course enrollments, waitlist statuses, and user activity. This data can help in making informed decisions regarding course offerings and resource allocation.
* Notifications and Alerts: The system provides various notifications and alerts to keep users informed about important events, deadlines, and changes. This includes reminders for registration deadlines, alerts for enrollment status changes, and notifications for new course offerings.
* User Support and Help: Integrated user support features, such as FAQs, tutorials, and a helpdesk, assist users in navigating the system and resolving any issues they may encounter. An automated chatbot may also be available to provide instant assistance for common queries.

## User Classes and Characteristics.

The Online Course Registration System is designed to accommodate various user classes, each with specific roles and access levels tailored to their needs. The primary user classes and their characteristics are as follows:

**Students:**

* **Role and Access:**
  + Students are the primary users of the system. They have the ability to register for a new account, log in, and manage their course enrollments.
* **Functions:** 
  + Account Management: Students can create and update their personal profiles, change passwords, and manage contact information. This ensures that their profile details are current and accurate.
  + Course Enrollment: Students can browse the course catalog, view detailed course information, and enroll in courses for the upcoming semester. They can also drop courses if needed and manage their current enrollments through their dashboard.
  + Waitlist Management: When a desired course is full, students can add themselves to a waitlist. They will be notified automatically when a spot becomes available, ensuring they have a fair chance to enroll.
  + Notifications and Alerts: Students receive notifications about important events such as registration deadlines, course availability updates, and enrollment status changes. These alerts help students stay informed and make timely decisions regarding their course registrations.
  + Feedback and Support: Students can provide feedback on courses they have completed, which helps improve future offerings. They also have access to support features such as FAQs, tutorials, and a helpdesk for assistance with system navigation and issue resolution.

**Faculty:**

* **Role and Access:**
  + Faculty members use the system to manage course details and monitor student enrollments. They have elevated access compared to students but do not have full administrative privileges.
* **Functions:** 
  + Course Management: Faculty can create and update course information, including course descriptions, schedules, prerequisites, and enrollment limits. This ensures that course details are accurate and up-to-date.
  + Enrollment Monitoring: Faculty can view the list of students enrolled in their courses and manage enrollments as needed. They can approve or deny special enrollment requests and handle any enrollment-related issues.
  + Communication: Faculty can send notifications and announcements to students enrolled in their courses. This feature facilitates effective communication and helps keep students informed about course-related updates.
  + Reports and Analytics: Faculty have access to basic reports and analytics on course enrollments, helping them understand student engagement and course demand. This data can be used to make informed decisions about future course offerings and adjustments.

**Administrators:**

* Role and Access:
  + Administrators have the highest level of access within the system. They oversee all functionalities, manage user accounts, and ensure the system operates smoothly and efficiently.
* **Functions:** 
  + User Management: Administrators can create, update, and delete user accounts for students, faculty, and other administrative staff. They can assign roles and permissions, ensuring that users have appropriate access levels.
  + System Configuration: Administrators can configure system settings, including security protocols, data backup schedules, and notification preferences. This ensures the system remains secure and reliable.
  + Course and Enrollment Management: Administrators have the ability to oversee the entire course registration process, including managing course catalogs, setting enrollment policies, and monitoring course capacities. They can also handle escalated issues related to course enrollments and waitlists.
  + Reporting and Analytics: Administrators have access to comprehensive reports and analytics on all aspects of the system, including user activity, course enrollments, and waitlist statuses. These insights help in strategic planning and decision-making to improve the overall effectiveness of the registration process.
  + Support and Maintenance: Administrators are responsible for maintaining the system, addressing technical issues, and providing support to users. They can access logs and system diagnostics to troubleshoot problems and ensure the system operates smoothly.
  + Compliance and Security: Administrators ensure that the system complies with relevant data protection regulations and institutional policies. They manage security settings, monitor for potential security breaches, and implement measures to safeguard sensitive information.

## Operating Environment

The Online Course Registration System is designed to operate seamlessly across multiple platforms and devices, ensuring broad accessibility and usability. Key aspects of the operating environment include:

Web Browsers: The system will be accessible through major web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and others that support modern web standards. This ensures that users can access the system using their preferred browser without compatibility issues. The web application will leverage HTML5, CSS3, and JavaScript to provide a responsive and interactive user interface (Panda, D., & Pati, B. 2015).

Operating Systems: The system will support multiple operating systems to accommodate diverse user preferences and institutional requirements. Specifically, the system will be compatible with (Oriental Journal of Computer Science and Technology, n.d.):

* Windows: Versions 10 and 11, ensuring compatibility with the latest security and feature updates from Microsoft.
* macOS: Versions 11 (Big Sur) and 12 (Monterey), supporting the latest Apple devices and operating system features.
* Linux: Popular distributions such as Ubuntu, Fedora, and Debian, providing flexibility for users who prefer open-source solutions.

Device Compatibility: While the primary interface is web-based, the system will be designed to be responsive, ensuring usability across various devices. This includes (Springer, n.d.):

* Desktops and Laptops: The system will be fully functional on standard desktop and laptop screens, providing a robust and detailed interface for managing course registrations and administrative tasks.
* Tablets and Smartphones: The responsive design will ensure that users can access the system on tablets and smartphones, allowing for convenient use on-the-go. The interface will adapt to different screen sizes to maintain usability and readability.

Server Environment: The backend of the system will be hosted on secure, reliable servers that can handle high volumes of traffic, especially during peak registration periods. The server environment will include (IEEE Xplore, n.d.):

* Web Server: Hosting the application code and serving web pages to users. Common choices include Apache or Nginx.
* Database Server: Storing all user data, course information, and enrollment records. A robust relational database management system (RDBMS) such as MySQL or PostgreSQL will be used to ensure data integrity and performance.

Network Requirements: The system will require a stable internet connection for users to access the web application. It will support standard HTTP and HTTPS protocols to ensure secure communication between users and the server.

Security and Compliance: The operating environment will include measures to ensure data security and compliance with relevant regulations. This includes encryption for data in transit and at rest, secure authentication mechanisms, and regular security audits. Compliance with data protection regulations such as GDPR (General Data Protection Regulation) and FERPA (Family Educational Rights and Privacy Act) will be ensured.

Development and Testing: During development and testing, the system will be evaluated across various browsers, operating systems, and devices to ensure compatibility and performance. This comprehensive testing approach will help identify and resolve any potential issues, ensuring a smooth user experience upon deployment.

## Design and Implementation Constraints

The design and implementation of the Online Course Registration System are subject to several constraints to ensure its effectiveness, security, and scalability. These constraints are crucial for maintaining system integrity, protecting user data, and accommodating the needs of a growing user base.

Data Security and Integrity (ResearchGate, n.d.):

* Encryption: All sensitive information, including user credentials, personal details, and course enrollment data, must be encrypted both in transit and at rest. The system will utilize industry-standard encryption protocols such as TLS (Transport Layer Security) for data in transit and AES (Advanced Encryption Standard) for data at rest. This ensures that data is protected from unauthorized access and potential breaches.
* Access Control: Strict access control measures must be implemented to ensure that only authorized users can access specific functionalities and data. Role-based access control (RBAC) will be used to assign permissions based on user roles (e.g., students, faculty, administrators). This minimizes the risk of unauthorized access and data manipulation.
* Data Integrity: Mechanisms to ensure data integrity must be in place, including validation checks, transaction management, and auditing trails. The system will use database constraints, triggers, and stored procedures to enforce data consistency and prevent corruption.

Scalability:

* Horizontal and Vertical Scaling: The system must support both horizontal scaling (adding more servers) and vertical scaling (upgrading existing servers) to handle an increasing number of users and growing volumes of course data. This flexibility is essential for accommodating peak usage periods, such as registration times.
* Load Balancing: To ensure high availability and optimal performance, load balancing techniques will be employed. This involves distributing incoming traffic across multiple servers to prevent any single server from becoming a bottleneck.
* Efficient Database Management: The database design must be optimized for performance and scalability. This includes proper indexing, query optimization, and partitioning to ensure that the system can efficiently handle large datasets and complex queries.

Compliance and Regulatory Constraints (Academia.edu, n.d.):

* Data Protection Regulations: The system must comply with relevant data protection regulations, such as the General Data Protection Regulation (GDPR) in Europe and the Family Educational Rights and Privacy Act (FERPA) in the United States. Compliance involves implementing policies for data handling, user consent, data anonymization, and breach notification.
* Accessibility Standards: The system must adhere to accessibility standards such as the Web Content Accessibility Guidelines (WCAG) to ensure that it is usable by individuals with disabilities. This includes providing features like screen reader support, keyboard navigation, and text alternatives for non-text content.

Development and Maintenance Constraints (ResearchGate, n.d.):

* Technology Stack: The choice of technology stack (programming languages, frameworks, databases, etc.) is constrained by the need for reliability, maintainability, and community support. The system will be built using widely adopted technologies with active developer communities to ensure ongoing support and updates.
* Integration with Existing Systems: The system must integrate seamlessly with existing institutional systems, such as Student Information Systems (SIS) and Learning Management Systems (LMS). This requires careful planning of data exchange protocols, API compatibility, and synchronization mechanisms.
* Development Timeline and Budget: The project is constrained by a fixed development timeline and budget. This requires efficient project management practices, including clear milestones, resource allocation, and risk management to ensure that the system is delivered on time and within budget.

User Experience Constraints (Springer, n.d.):

* Performance Requirements: The system must meet specific performance criteria to ensure a smooth user experience. This includes fast page load times, responsive interactions, and minimal downtime. Performance optimization techniques, such as caching and asynchronous processing, will be employed to achieve these goals.
* User Feedback and Iteration: User feedback will be an essential part of the design and implementation process. The system must be designed to allow for iterative improvements based on user feedback and changing requirements. This involves implementing a feedback mechanism and maintaining a flexible development approach to accommodate updates and enhancements.

## User Documentation

Comprehensive user documentation is essential for ensuring that all users can effectively navigate and utilize the Online Course Registration System. The documentation will be designed to cater to the needs of different user classes, including students, faculty, and administrators, and will cover all aspects of system usage. Key components of the user documentation will include:

User Manuals:

* Student User Manual: This manual will provide detailed instructions for students on how to register for an account, log in, view available courses, enroll in courses, manage waitlists, and cancel enrollments. It will include step-by-step guides with screenshots to illustrate each process clearly.
* Faculty User Manual: The faculty manual will cover how to manage course details, monitor student enrollments, send notifications, and use reporting features. It will also provide guidance on handling common issues that may arise during course management.
* Administrator User Manual: This manual will provide in-depth instructions for administrators on managing user accounts, configuring system settings, overseeing the registration process, generating reports, and maintaining system security. It will include advanced troubleshooting tips and best practices for system administration.

Online Help:

* Help Center: An online help center will be available within the system, offering searchable articles and guides on various topics. Users can quickly find answers to their questions by entering keywords or browsing through categorized sections.
* Contextual Help: Throughout the system, contextual help buttons (e.g., tooltips and help icons) will provide users with immediate assistance and explanations for specific features and functions. This will help users understand how to use different parts of the system without needing to refer to the manual constantly.

Frequently Asked Questions (FAQs):

* Comprehensive FAQs: A dedicated FAQ section will address common questions and issues that users may encounter. This will include general inquiries, technical problems, and specific scenarios related to course registration, waitlist management, and user account settings.
* Continuous Updates: The FAQ section will be regularly updated based on user feedback and emerging issues, ensuring that it remains relevant and helpful.

Tutorials:

* Video Tutorials: A series of video tutorials will be created to provide visual and auditory guidance on using the system. These tutorials will cover basic and advanced features, making it easier for users to follow along and understand the processes.
  + Student Tutorials: Topics will include account registration, course enrollment, waitlist management, and profile updates.
  + Faculty Tutorials: Topics will cover course management, enrollment monitoring, and communication tools.
  + Administrator Tutorials: Topics will include user account management, system configuration, and generating reports.
* Interactive Tutorials: Interactive tutorials will be available for new users to practice key tasks within a guided environment. These tutorials will provide real-time feedback and step-by-step instructions, helping users gain hands-on experience with the system.

Support and Training:

* User Support Portal: A support portal will be available for users to submit tickets and request assistance from the support team. This portal will include a ticketing system to track issues and provide timely responses.
* Live Chat and Email Support: Users will have access to live chat and email support for real-time assistance with urgent issues or complex questions.
* Training Sessions: Regular training sessions will be conducted for new users, particularly during the initial rollout of the system. These sessions will be available as live webinars or recorded sessions that users can access at their convenience.

Documentation Maintenance:

* Regular Updates: The user documentation will be regularly reviewed and updated to reflect any changes or enhancements to the system. This ensures that the information remains accurate and useful over time.
* User Feedback Integration: Feedback from users will be actively sought and integrated into the documentation. This helps identify areas where users may be experiencing difficulties and allows for the continuous improvement of the support materials.

## Assumptions and Dependencies

The successful operation and implementation of the Online Course Registration System rely on several assumptions and dependencies. Understanding these assumptions and dependencies is crucial for planning, development, and deployment to ensure the system functions as intended and meets user expectations.

Assumptions:

* Internet Connectivity: It is assumed that all users have access to a reliable internet connection to interact with the system. The system's web-based nature requires continuous internet access for real-time data retrieval, course registration, and other functionalities.
* User Proficiency: It is assumed that users have basic computer literacy and familiarity with web browsing. This includes the ability to navigate web pages, fill out online forms, and manage personal account settings.
* Institutional Policies: It is assumed that the institution has policies and procedures in place for course registration, data privacy, and security that align with the system's functionality. This includes user authentication processes, data protection measures, and compliance with relevant regulations.
* Regular Maintenance: It is assumed that the system will undergo regular maintenance to ensure optimal performance and security. This includes software updates, database backups, and server maintenance.
* Support Infrastructure: It is assumed that there is an adequate support infrastructure in place to assist users with technical issues and system navigation. This includes helpdesk support, documentation, and training resources.

Dependencies:

* Web Servers: The system depends on robust web servers to host the application and serve web pages to users. These servers must be capable of handling high traffic volumes, particularly during peak registration periods. Common choices for web servers include Apache and Nginx, which provide stability and performance.
* Databases: A reliable and scalable database management system (DBMS) is essential for storing and retrieving user data, course information, and enrollment records. The system relies on databases such as MySQL, PostgreSQL, or similar RDBMS solutions to ensure data integrity and efficient query processing.
* Cloud Hosting Services: To ensure scalability and availability, the system depends on cloud hosting services. Cloud providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure offer the infrastructure needed to deploy and scale the application efficiently. Cloud hosting provides flexibility, disaster recovery options, and the ability to handle fluctuating workloads.
* Authentication Services: The system depends on secure authentication mechanisms to verify user identities and protect sensitive information. This includes integration with authentication services that support secure login processes, password management, and multi-factor authentication (MFA).
* Third-Party APIs: The system may rely on third-party APIs for additional functionalities such as email notifications, SMS alerts, and payment processing (if applicable). These APIs must be reliable and secure to ensure seamless integration and operation.
* Development Tools and Frameworks: The system's development depends on various tools and frameworks that facilitate coding, testing, and deployment. This includes integrated development environments (IDEs), version control systems (e.g., Git), and web development frameworks (e.g., React, Angular, Django).
* Compliance and Regulatory Dependencies: The system must comply with relevant data protection and privacy regulations, such as GDPR (General Data Protection Regulation) and FERPA (Family Educational Rights and Privacy Act). Compliance requires adherence to specific guidelines and standards, impacting system design and data handling practices.
* Institutional IT Infrastructure: The system depends on the institution’s existing IT infrastructure for integration and support. This includes network configurations, firewalls, and other security measures that ensure the system operates within the institutional environment without conflicts or vulnerabilities.

# External Interface Requirements

## User Interfaces

The Online Course Registration System will feature a comprehensive web-based interface designed to facilitate user interactions and provide a seamless user experience. The interface will be intuitive, responsive, and accessible, ensuring that all users can easily navigate and utilize the system’s functionalities. Key components of the user interface include:

Registration Forms:

* New User Registration: The registration form will guide new users through the process of creating an account. It will collect essential information such as name, email address, phone number, and desired user ID. The form will include validation checks to ensure data accuracy and uniqueness of the user ID. Helpful tooltips and instructions will assist users in completing the form correctly.
* Profile Management: Users will have access to a profile management page where they can update their personal information, change passwords, and manage contact details. This page will provide clear instructions and immediate feedback on changes.

Login Screens:

* User Login: The login screen will allow registered users to access their accounts by entering their user ID and password. The interface will include options for password recovery and multi-factor authentication (MFA) to enhance security. A "Remember Me" feature will offer convenience for frequent users.
* Guest Access: While full functionality requires login, certain features like browsing the course catalog may be available to guest users. This encourages engagement and exploration before committing to registration.

Course Listings:

* Browse and Search: The course listing page will provide a comprehensive catalog of available courses. Users can browse courses by category, semester, department, or instructor. Advanced search functionality will allow users to filter courses based on various criteria such as course title, code, and schedule.
* Course Details: Each course listing will include detailed information such as course description, prerequisites, schedule, instructor information, and enrollment capacity. Users can view these details by clicking on a course title.

Enrollment Management Pages:

* Course Enrollment: The enrollment page will enable users to enroll in selected courses. It will display real-time availability and provide options to join waitlists if courses are full. Users can view their current enrollments and add or drop courses with a few clicks.
* Waitlist Management: For courses that are full, users can add themselves to a waitlist. The interface will show their position on the waitlist and provide notifications when spots become available. Users can also remove themselves from the waitlist if needed.
* Schedule Overview: Users will have access to a schedule overview page that displays all enrolled courses in a calendar format. This visual representation helps users manage their time and avoid scheduling conflicts.

Notifications and Alerts:

* Enrollment Notifications: Users will receive notifications for successful enrollments, waitlist updates, and course changes. These alerts will appear on the dashboard and can also be sent via email or SMS based on user preferences.
* Deadline Reminders: The system will provide reminders for important deadlines such as registration dates, drop/add periods, and waitlist expirations. These reminders help users stay informed and take timely actions.

User Dashboard:

* Personal Dashboard: Each user will have a personalized dashboard that provides an overview of their account status, current enrollments, and pending actions. The dashboard will include quick links to frequently used features such as course search, profile management, and support resources.
* Admin Dashboard: Administrators will have access to a more comprehensive dashboard that includes system analytics, user management tools, and reporting features. This interface will allow administrators to monitor system performance and make informed decisions.

Support and Help:

* Help Center: An integrated help center will provide users with access to FAQs, tutorials, and troubleshooting guides. Contextual help icons will be available throughout the interface to assist users with specific features.
* Live Support: Users will have the option to access live support via chat or email for immediate assistance with any issues or questions.

Accessibility Features:

* Accessibility Options: The interface will include features to support users with disabilities, such as screen reader compatibility, keyboard navigation, and adjustable text sizes. Compliance with WCAG (Web Content Accessibility Guidelines) will ensure that the system is usable by all individuals.

Responsive Design:

* Mobile and Tablet Compatibility: The interface will be designed to be fully responsive, ensuring that it works well on a variety of devices, including desktops, laptops, tablets, and smartphones. This allows users to access the system from anywhere and at any time.

## Hardware Interfaces

The Online Course Registration System is designed to interact seamlessly with various hardware interfaces to ensure a smooth and efficient user experience. The system’s hardware compatibility considerations include:

User Devices:

* Keyboards and Mice: The primary mode of interaction for users will be through standard input devices such as keyboards and mice. The system will support all common keyboard layouts and mouse configurations, enabling users to navigate, enter data, and interact with the system’s features effectively.
* Displays: The system will be compatible with a wide range of display devices, including standard desktop monitors, laptop screens, and external displays. The user interface will be designed to be responsive, adapting to different screen sizes and resolutions to provide an optimal viewing experience.

Desktop and Laptop Compatibility:

* Operating Systems: The system will support devices running on popular operating systems, including Windows, macOS, and Linux. This ensures that users can access the system from their preferred devices without any compatibility issues.
  + Windows: The system will be compatible with Windows 10 and Windows 11, ensuring it functions correctly on the latest versions and updates of the Windows operating system.
  + macOS: Compatibility with macOS versions 11 (Big Sur) and 12 (Monterey) will be ensured, providing support for Apple desktop and laptop devices.
  + Linux: The system will support popular Linux distributions such as Ubuntu, Fedora, and Debian, catering to users who prefer open-source operating systems.

Peripheral Devices:

* Printers: The system will provide support for printing functionalities, allowing users to print course schedules, enrollment confirmations, and other important documents. Compatibility with standard printers connected via USB or network will be ensured.
* Scanners: Although not a primary interface, the system will support document scanning for administrative purposes, such as scanning enrollment forms or identification documents. Compatibility with common scanner models will be considered.
* External Storage Devices: Users may need to upload documents such as identification proofs or prerequisite certifications. The system will support interaction with external storage devices like USB flash drives and external hard drives for uploading and downloading relevant documents.

Mobile Devices:

* Touchscreens: For users accessing the system via tablets and smartphones, the interface will support touchscreen interactions. This includes touch-based navigation, tapping, swiping, and other gestures commonly used on mobile devices.
* On-Screen Keyboards: Mobile and tablet users will interact with the system using on-screen keyboards. The system will ensure compatibility with these virtual keyboards, providing a smooth data entry experience.

Assistive Technologies:

* Screen Readers: The system will be compatible with screen readers used by visually impaired users. This includes ensuring that all interactive elements are accessible and properly labeled, allowing screen readers to convey the necessary information to the user.
* Braille Displays: For users who rely on Braille displays, the system will ensure compatibility by providing text-based output that can be converted by these devices.
* Alternative Input Devices: The system will support alternative input devices such as trackballs, joysticks, and specialized keyboards used by individuals with motor impairments.

Hardware Requirements for Servers:

* Web Server: The backend infrastructure will require robust web servers to host the application and handle user requests efficiently. These servers must have adequate processing power, memory, and storage to support high traffic volumes and data loads.
* Database Server: A reliable database server is essential for managing user data, course information, and enrollment records. The server must be capable of handling large datasets and providing quick query responses to ensure a smooth user experience.
* Cloud Infrastructure: If hosted on cloud platforms, the system will depend on virtual servers provided by cloud service providers. These virtual servers must be configured to meet the performance, scalability, and security requirements of the system.

## Software Interfaces

The Online Course Registration System will utilize multiple software interfaces to ensure smooth and efficient interactions between different components of the system. These interfaces will enable the system to perform various functions such as user interactions, data management, and notifications. Key software interfaces include:

Web Browsers:

* Compatibility: The system will be accessible through all major web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. Ensuring compatibility with these browsers will provide a consistent user experience across different platforms.
* HTML/CSS/JavaScript: The frontend of the system will be built using HTML, CSS, and JavaScript. These technologies will ensure that the user interface is responsive, interactive, and visually appealing. The use of modern web development frameworks and libraries (e.g., React, Angular, Vue.js) may be employed to enhance the user experience.
* AJAX and RESTful APIs: Asynchronous JavaScript and XML (AJAX) will be used to enable dynamic content updates without requiring full page reloads. RESTful APIs will facilitate communication between the frontend and backend, allowing the system to fetch and display data efficiently.

Backend Database:

* Database Management System (DBMS): The system will interact with a robust DBMS such as MySQL, PostgreSQL, or Microsoft SQL Server. The database will store critical data, including user accounts, course information, enrollment records, and waitlist details.
* SQL Queries: Structured Query Language (SQL) will be used to perform database operations such as data retrieval, insertion, updating, and deletion. The system will employ efficient query optimization techniques to ensure quick data access and manipulation.
* ORM Frameworks: Object-Relational Mapping (ORM) frameworks like Hibernate (for Java), Entity Framework (for .NET), or Sequelize (for Node.js) may be used to simplify database interactions and enhance development efficiency.

Email Server:

* SMTP Protocol: The system will use the Simple Mail Transfer Protocol (SMTP) to send email notifications to users. These notifications will include account confirmations, enrollment updates, waitlist notifications, and other important alerts.
* Email Templates: Predefined email templates will be used to ensure consistency and professionalism in all communications. These templates will be customizable to accommodate different types of notifications and branding requirements.
* Email Service Providers: The system may integrate with third-party email service providers such as SendGrid, Amazon SES, or Mailgun to handle large volumes of email efficiently and ensure reliable delivery.

Authentication Services:

* OAuth and SAML: For enhanced security, the system may integrate with authentication protocols like OAuth and Security Assertion Markup Language (SAML). These protocols will enable secure user authentication and single sign-on (SSO) capabilities, allowing users to log in using their institutional credentials or third-party accounts (e.g., Google, Microsoft).
* Password Management: The system will incorporate secure password management practices, including hashing and salting of passwords, to protect user credentials. Integration with password recovery and reset services will provide users with a seamless experience in case they forget their passwords.

APIs and External Integrations:

* RESTful APIs: The system will expose RESTful APIs to enable integration with other institutional systems such as Student Information Systems (SIS) and Learning Management Systems (LMS). These APIs will allow for data exchange and synchronization, ensuring that all systems have up-to-date information.
* External Services: Integration with external services for additional functionalities, such as payment gateways for fee processing or SMS gateways for sending text message notifications, will be facilitated through API interfaces.

Logging and Monitoring:

* Logging Services: The system will interface with logging services to record system events, user activities, and errors. These logs will be crucial for debugging, performance monitoring, and ensuring system reliability.
* Monitoring Tools: Integration with monitoring tools such as Prometheus, Grafana, or New Relic will enable real-time tracking of system performance, resource utilization, and potential issues. This will help maintain system health and optimize performance.

Security Interfaces:

* Encryption Libraries: The system will use encryption libraries to secure data in transit and at rest. Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols will be used for data transmission, while database encryption mechanisms will protect stored data.
* Firewall and Intrusion Detection: The system will interface with firewall and intrusion detection systems to safeguard against unauthorized access and cyber threats. These security measures will be integral to maintaining data integrity and user privacy.

## Communications Interfaces

The Online Course Registration System will utilize several communication interfaces to ensure effective and secure data exchange between the system's components and users. The primary communication protocols and methods include (Panda, D., & Pati, B. 2015):

HTTP/HTTPS Protocols:

* HTTP (HyperText Transfer Protocol): The system will use HTTP for basic web communication. HTTP is the foundation of data communication for the web, allowing the system to transmit and receive information between the client (user's web browser) and the server.
* HTTPS (HyperText Transfer Protocol Secure): To ensure secure data transmission, the system will use HTTPS. HTTPS encrypts data using SSL/TLS protocols, protecting sensitive information such as user credentials, personal data, and enrollment records from being intercepted or tampered with during transmission. The use of HTTPS will be enforced for all communications to guarantee privacy and data integrity.

RESTful APIs:

* REST (Representational State Transfer): The system will employ RESTful APIs for communication between the frontend and backend. RESTful APIs use standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources. This approach ensures a stateless, scalable, and flexible architecture, enabling efficient data exchange and integration with other systems.
* API Security: RESTful APIs will be secured using authentication mechanisms such as OAuth2, API keys, or JWT (JSON Web Tokens). These methods will ensure that only authorized users and systems can access the APIs, providing an additional layer of security.

WebSockets:

* Real-Time Communication: For features requiring real-time updates, such as live notifications of enrollment status changes or waitlist availability, the system may use WebSockets. WebSockets provide a persistent connection between the client and server, allowing for instantaneous two-way communication and reducing latency.

Email Communication:

* SMTP (Simple Mail Transfer Protocol): The system will use SMTP to send email notifications to users. This includes account verification emails, enrollment confirmations, waitlist notifications, and other alerts. SMTP ensures reliable and efficient email delivery.
* Email Encryption: To protect the content of email communications, the system will employ email encryption standards such as STARTTLS, which upgrades plain text connections to encrypted connections using SSL/TLS.

SMS Communication:

* SMS Gateway Integration: For sending SMS notifications, the system will integrate with an SMS gateway service. This allows users to receive real-time alerts on their mobile devices for critical updates such as enrollment confirmations and waitlist movements.
* Secure Transmission: The SMS gateway will ensure secure transmission of messages, adhering to relevant security standards and best practices.

Internal Messaging:

* In-System Notifications: The system will include an internal messaging platform for sending notifications directly within the user interface. This ensures users are promptly informed about important events and updates without relying solely on external communication methods.
* Notification Management: Users will have options to customize their notification preferences, choosing between email, SMS, or in-system notifications based on their preferences.

Data Synchronization:

* Scheduled Synchronization: The system will periodically synchronize data with external systems, such as Student Information Systems (SIS) and Learning Management Systems (LMS), using secure data transfer methods. This ensures that all systems have up-to-date and consistent information.
* Real-Time Data Sync: For critical data that requires immediate updates, the system will use real-time data synchronization methods. This may involve API calls or webhooks to ensure timely data exchange.

Firewall and Network Security:

* Secure Network Configuration: The system's network configuration will include firewalls and intrusion detection systems to protect against unauthorized access and cyber threats. These security measures will monitor and control incoming and outgoing network traffic based on predetermined security rules.
* VPN (Virtual Private Network): For secure remote access by administrators and support staff, the system may use VPNs. VPNs encrypt internet connections, ensuring that data transmitted over public networks remains secure.

Backup and Recovery Communication:

* Automated Backups: The system will perform automated backups of critical data, transmitting backup files to secure storage locations. These backups will be encrypted and stored in multiple geographic locations to ensure data availability in case of a disaster.
* Data Recovery Protocols: Communication protocols for data recovery will be established to enable quick and efficient restoration of data from backups in the event of data loss or corruption.

# System Features

## New User Registration

Description and Priority

* Description: The New User Registration feature allows users to create a new account with a unique ID and password. This is a critical function as it enables users to access the system and its various features. Ensuring a smooth and secure registration process is essential for user engagement and satisfaction.
* Priority: High

Stimulus/Response Sequences

* Stimulus: The user navigates to the registration page via a link on the homepage or login screen.
* Response: The system displays the registration form, prompting the user to enter required details such as name, phone number, email address, and desired user ID and password.
* Stimulus: The user fills in the registration form and submits it by clicking the "Register" button.
* Response: The system validates the entered information, checking for completeness and correctness. It ensures that the user ID is unique and not already in use.
* Stimulus: If the validation is successful, the system creates the user account and stores the user information securely in the database.
* Response: The system sends a confirmation email to the provided email address, welcoming the new user and confirming the successful creation of the account. The user is then redirected to the login page or their new dashboard.

Functional Requirements

* REQ-1: The system must prevent duplicate user IDs.
  + Rationale: Ensuring unique user IDs prevents conflicts and maintains the integrity of user accounts.
  + Implementation: The system will check the database for existing IDs before accepting a new registration.
* REQ-2: User profiles must include name, phone number, email address, and other necessary information.
  + Rationale: Collecting comprehensive user information allows for personalized communication and better service delivery.
  + Implementation: The registration form will include fields for all required information, with validation to ensure completeness and accuracy.
* REQ-3: The system must validate email addresses to ensure they are in the correct format and unique.
  + Rationale: Valid email addresses are essential for communication and account recovery.
  + Implementation: The system will use regex patterns for format validation and check for uniqueness in the database.
* REQ-4: Passwords must meet security criteria (e.g., minimum length, complexity).
  + Rationale: Strong passwords enhance security and protect user accounts from unauthorized access.
  + Implementation: The system will enforce password policies requiring a combination of letters, numbers, and special characters.
* REQ-5: The system must provide feedback to users for any errors or missing information in the registration form.
  + Rationale: Clear feedback helps users correct mistakes and complete the registration process successfully.
  + Implementation: The system will display error messages next to the relevant form fields.
* REQ-6: The system must send a confirmation email upon successful registration.
  + Rationale: Confirmation emails verify the provided email address and welcome new users to the system.
  + Implementation: The system will generate and send an email to the user's address upon successful account creation.
* REQ-7: The system should allow users to opt-in for additional communication or newsletters during registration.
  + Rationale: Opt-in options facilitate user engagement and ensure compliance with communication regulations.
  + Implementation: The registration form will include checkboxes for users to subscribe to newsletters or additional updates.

## User Login

Description and Priority

* Description: The User Login feature allows registered users to access their accounts by entering their user ID and password. This is a crucial function as it provides authenticated access to the system's features and ensures secure user interactions.
* Priority: High

Stimulus/Response Sequences

* Stimulus: The user navigates to the login page via a link on the homepage or registration confirmation email.
* Response: The system displays the login form, prompting the user to enter their user ID and password.
* Stimulus: The user enters their credentials and clicks the "Log In" button.
* Response: The system verifies the entered credentials against the stored data in the database.
* Stimulus: If the credentials are correct, the system grants access and redirects the user to their dashboard or the last accessed page.
* Response: If the credentials are incorrect, the system displays an error message indicating the login failure and prompts the user to try again.

Functional Requirements

* REQ-1: The system must authenticate user credentials securely.
  + Rationale: Secure authentication is essential to protect user accounts and prevent unauthorized access.
  + Implementation: The system will use secure hashing algorithms (e.g., bcrypt) to store passwords and will validate credentials using these secure methods. All authentication processes will occur over HTTPS to ensure encrypted communication.
* REQ-2: The system must handle login failures gracefully and provide feedback.
  + Rationale: Providing clear feedback helps users understand why their login attempt failed and guides them in taking corrective actions.
  + Implementation: The system will display specific error messages (e.g., "Incorrect user ID or password") when login attempts fail. It will also provide options for password recovery and support contact information for further assistance.
* REQ-3: The system should implement account lockout mechanisms after multiple failed login attempts.
  + Rationale: Account lockout mechanisms protect against brute force attacks and unauthorized access attempts.
  + Implementation: After a predefined number of failed login attempts (e.g., 5 attempts), the system will temporarily lock the account and notify the user via email. The lockout duration and conditions for unlocking will be specified.
* REQ-4: The system must support multi-factor authentication (MFA) for enhanced security.
  + Rationale: MFA provides an additional layer of security by requiring users to verify their identity through multiple means.
  + Implementation: The system will offer MFA options such as SMS codes, email verification, or authenticator apps. Users will be prompted to set up MFA during registration or through their account settings.
* REQ-5: The system must provide a "Remember Me" option for users who wish to stay logged in.
  + Rationale: The "Remember Me" option enhances user convenience by allowing users to remain logged in across sessions.
  + Implementation: The system will securely store session tokens with appropriate expiration times and implement mechanisms to revoke tokens if necessary.
* REQ-6: The system must allow users to recover their passwords securely.
  + Rationale: Password recovery is crucial for user accessibility and account management.
  + Implementation: The login page will include a "Forgot Password" link, prompting users to enter their registered email address. The system will send a password reset link or code to the email, enabling users to set a new password.
* REQ-7: The system must log successful and failed login attempts for security monitoring.
  + Rationale: Logging login attempts helps in identifying and responding to potential security threats.
  + Implementation: The system will maintain logs of all login attempts, including timestamps, user IDs, and IP addresses. These logs will be accessible to administrators for security audits and monitoring.
* REQ-8: The system must provide secure session management.
  + Rationale: Secure session management ensures that user sessions are protected and terminated appropriately.
  + Implementation: The system will use secure session cookies with attributes such as HttpOnly and Secure flags. Sessions will be timed out after a period of inactivity, and users will be prompted to re-authenticate as needed.

## Course Listing and Enrollment

Description and Priority

* Description: The Course Listing and Enrollment feature allows users to view available courses for each semester and enroll in them if slots are available. This is a crucial function as it directly supports the primary goal of the system: facilitating course registration for students.
* Priority: High

Stimulus/Response Sequences

* Stimulus: The user logs into the system and navigates to the course listing page.
* Response: The system displays options for selecting a semester (e.g., Spring, Summer, Fall).
* Stimulus: The user selects a specific semester from the dropdown or list.
* Response: The system retrieves and displays a list of available courses for the selected semester, including details such as course title, code, description, schedule, instructor, and the number of available slots.
* Stimulus: The user browses through the list and selects courses they wish to enroll in by clicking on an "Enroll" button next to each course.
* Response: The system checks the availability of slots in the selected courses.
* Stimulus: If slots are available, the user confirms their enrollment by clicking a "Confirm Enrollment" button.
* Response: The system updates the enrollment status in real-time, deducts the available slots, and adds the user to the course roster.
* Stimulus: If slots are not available, the system prompts the user to join the waitlist or choose a different course.
* Response: The system displays a confirmation message for successful enrollments and updates the user's schedule.

Functional Requirements

* REQ-1: The system must display course availability accurately.
  + Rationale: Accurate display of course availability ensures that users can make informed decisions about their enrollments.
  + Implementation: The system will query the database in real-time to retrieve current enrollment numbers and available slots, updating the course listings accordingly.
* REQ-2: The system must update enrollment status in real-time.
  + Rationale: Real-time updates prevent over-enrollment and ensure that users receive immediate feedback on their enrollment actions.
  + Implementation: The system will use AJAX or similar asynchronous techniques to update the user interface without requiring a full page reload. Enrollment transactions will be processed atomically to maintain data integrity.
* REQ-3: The system must provide detailed course information.
  + Rationale: Providing comprehensive course details helps users choose the right courses and understand their commitments.
  + Implementation: Course listings will include detailed information such as course title, code, description, prerequisites, schedule, instructor name, and contact information.
* REQ-4: The system must allow users to filter and search courses.
  + Rationale: Filtering and search functionalities enhance the user experience by enabling users to find courses that match their criteria quickly.
  + Implementation: The system will provide filtering options (e.g., by department, time, instructor) and a search bar to find courses based on keywords.
* REQ-5: The system must handle conflicts and prerequisites.
  + Rationale: Managing conflicts and prerequisites ensures that users can only enroll in courses for which they are eligible and that do not overlap with their existing schedule.
  + Implementation: The system will check for schedule conflicts and prerequisite requirements before allowing enrollment. Users will be notified of any issues and provided with options to resolve them.
* REQ-6: The system must support waitlist functionality for full courses.
  + Rationale: Waitlists provide a fair and organized method for users to join full courses when spots become available.
  + Implementation: If a course is full, the system will offer users the option to join a waitlist. The system will manage the waitlist order and notify users automatically when a spot opens up.
* REQ-7: The system must provide enrollment confirmation and notifications.
  + Rationale: Clear confirmation and notifications keep users informed about their enrollment status and any changes.
  + Implementation: Upon successful enrollment, the system will display a confirmation message and send an email notification to the user. Notifications for waitlist status changes and schedule updates will also be sent as needed.
* REQ-8: The system must allow users to view and manage their enrolled courses.
  + Rationale: Providing a centralized view of enrolled courses helps users manage their schedules and make adjustments as needed.
  + Implementation: Users will have access to a dashboard displaying their current enrollments, with options to drop courses or add new ones, subject to availability.
* REQ-9: The system must maintain data integrity during high-traffic periods.
  + Rationale: Ensuring data integrity during peak usage times (e.g., start of registration periods) is crucial for a reliable user experience.
  + Implementation: The system will employ load balancing, database transaction management, and performance optimization techniques to handle high traffic efficiently.

## Course Waitlist Management

Description and Priority

* Description: The Course Waitlist Management feature allows users to add themselves to a waitlist if a course is full. This ensures that students still have a chance to enroll in their desired courses when spots become available. The system manages the waitlist, keeping it updated and notifying users as soon as slots open up.
* Priority: Medium

Stimulus/Response Sequences

* Stimulus: The user attempts to enroll in a course that has reached its maximum enrollment capacity.
* Response: The system displays a message indicating that the course is full and offers the option to join the waitlist.
* Stimulus: The user opts to join the waitlist by clicking the "Join Waitlist" button.
* Response: The system adds the user to the waitlist, maintaining the order based on when users joined. The user receives a confirmation message that they have been added to the waitlist.
* Stimulus: When a spot becomes available in the course, the system identifies the next user on the waitlist.
* Response: The system sends a notification to the user, informing them that a slot has become available and providing instructions on how to enroll.
* Stimulus: The notified user follows the instructions to enroll in the course.
* Response: The system updates the course enrollment and removes the user from the waitlist.

Functional Requirements

* REQ-1: The system must maintain an accurate waitlist.
  + Rationale: Ensuring the accuracy of the waitlist is crucial for fair and efficient course enrollment.
  + Implementation: The system will use a database table to track waitlist entries, including the user's position on the waitlist, timestamp of joining, and course details. The waitlist will be dynamically updated to reflect any changes.
* REQ-2: The system must notify users of available slots promptly.
  + Rationale: Prompt notifications ensure that users have sufficient time to act on the available slot and complete their enrollment.
  + Implementation: The system will use automated email and/or SMS notifications to inform users immediately when a slot opens up. Notifications will include detailed instructions on how to complete the enrollment process.
* REQ-3: The system must allow users to view their waitlist status.
  + Rationale: Providing transparency about waitlist status helps users manage their expectations and make informed decisions.
  + Implementation: Users will have access to a dashboard where they can view their current position on the waitlist for each course, along with estimated wait times if applicable.
* REQ-4: The system must handle multiple waitlists for different courses simultaneously.
  + Rationale: Efficient management of multiple waitlists is necessary to accommodate user preferences for various courses.
  + Implementation: The system will support multiple waitlists by maintaining separate entries for each course, ensuring that updates and notifications are specific to the relevant course.
* REQ-5: The system must support user removal from waitlists.
  + Rationale: Allowing users to remove themselves from waitlists gives them control over their enrollment preferences and prevents unnecessary notifications.
  + Implementation: Users will be able to remove themselves from a waitlist through their dashboard. The system will update the waitlist accordingly and notify the next user in line if a spot becomes available.
* REQ-6: The system must manage waitlist capacity limits.
  + Rationale: Managing the number of users on a waitlist ensures that it remains functional and manageable.
  + Implementation: The system will set a maximum capacity for each waitlist based on course policies. If a waitlist reaches its capacity, the system will inform users and prevent additional entries.
* REQ-7: The system must ensure data integrity during waitlist updates.
  + Rationale: Maintaining data integrity is essential for accurate and reliable waitlist management.
  + Implementation: The system will use transaction management techniques to ensure that updates to the waitlist (e.g., additions, removals, notifications) are processed atomically, preventing data inconsistencies.
* REQ-8: The system must log waitlist activities for audit purposes.
  + Rationale: Logging waitlist activities helps in auditing and resolving disputes related to waitlist management.
  + Implementation: The system will maintain logs of all waitlist activities, including user additions, removals, notifications sent, and enrollments from the waitlist. These logs will be accessible to administrators for review and analysis.

## Enrollment Cancellation

Description and Priority

* Description: The Enrollment Cancellation feature allows users to cancel their enrollment in a course. This feature is important for users who need to adjust their schedules and ensures that available slots can be filled by other interested students.
* Priority: Medium

Stimulus/Response Sequences

* Stimulus: The user logs into the system and navigates to their dashboard or enrolled courses page.
* Response: The system displays a list of courses in which the user is currently enrolled.
* Stimulus: The user selects the courses they wish to cancel by clicking on the "Cancel Enrollment" button next to each course.
* Response: The system prompts the user to confirm the cancellation to prevent accidental unenrollment.
* Stimulus: The user confirms the cancellation.
* Response: The system processes the cancellation, updates the course's available slots, and removes the user from the course roster.
* Stimulus: The system identifies users on the waitlist for the canceled course.
* Response: The system sends notifications to the waitlisted users, informing them of the newly available slot and providing instructions for enrollment.

Functional Requirements

* REQ-1: The system must handle enrollment cancellations efficiently.
  + Rationale: Efficient handling of cancellations ensures that available slots are quickly made available for other students and maintains accurate enrollment records.
  + Implementation: The system will use database transactions to process cancellations, ensuring that updates to enrollment records and available slots are executed atomically.
* REQ-2: The system must notify waitlisted users of available slots automatically.
  + Rationale: Automatic notifications keep the waitlist process transparent and prompt, ensuring that users have a fair chance to enroll in newly available slots.
  + Implementation: The system will send automated email and/or SMS notifications to waitlisted users as soon as a slot becomes available. Notifications will include instructions for completing the enrollment process.
* REQ-3: The system must provide a confirmation prompt before finalizing cancellations.
  + Rationale: A confirmation prompt prevents accidental cancellations and ensures that users are aware of their actions.
  + Implementation: Before processing a cancellation, the system will display a confirmation dialog asking the user to confirm their decision. The cancellation will only proceed if the user confirms.
* REQ-4: The system must update the user's schedule immediately after cancellation.
  + Rationale: Immediate updates to the user's schedule help maintain accurate records and inform users of their current enrollment status.
  + Implementation: Once a cancellation is processed, the system will refresh the user's dashboard to reflect the change in their schedule, showing the removed course.
* REQ-5: The system must log all cancellation activities for audit purposes.
  + Rationale: Logging cancellation activities helps in auditing and resolving any disputes related to enrollment changes.
  + Implementation: The system will maintain logs of all cancellation activities, including user details, course information, timestamps, and actions taken. These logs will be accessible to administrators for review and analysis.
* REQ-6: The system must ensure that canceled slots are reallocated to the waitlist in the correct order.
  + Rationale: Fair reallocation of slots ensures that waitlisted users are given opportunities based on their position in the waitlist.
  + Implementation: The system will automatically allocate newly available slots to waitlisted users in the order they joined the waitlist, maintaining the correct sequence.
* REQ-7: The system must support bulk cancellations for administrators.
  + Rationale: Allowing administrators to perform bulk cancellations can be useful for managing course changes, schedule adjustments, or administrative decisions.
  + Implementation: Administrators will have the option to select multiple users or courses for bulk cancellation through an administrative interface. The system will process these cancellations efficiently and update all relevant records.
* REQ-8: The system must provide feedback to users on the success or failure of the cancellation process.
  + Rationale: Clear feedback ensures that users are informed about the outcome of their actions and can take further steps if necessary.
  + Implementation: After processing a cancellation request, the system will display a message indicating whether the cancellation was successful or if there were any issues that need to be addressed.

# Other Non-functional Requirements

## Performance Requirements

The system must exhibit minimal processing and loading times to ensure a seamless and efficient user experience. Key actions such as user registration, login, and course enrollment should be completed within seconds. Specifically, user registration, including data validation and account creation, should be processed within 3-5 seconds, while login and authentication should be completed within 2-3 seconds. The display of available courses for a selected semester should load within 3 seconds, and actions related to course enrollment, including the confirmation and updating of the user’s schedule, should be completed within 3-5 seconds. Similarly, adding or removing users from the waitlist, processing enrollment cancellations, and reallocating slots to waitlisted users should all be performed within 3-5 seconds to maintain system efficiency and user satisfaction. The system should also handle peak loads and concurrent access by multiple users without significant performance degradation, employing scalable infrastructure, load balancing, and performance optimization techniques (Naik, S. 2017).

## Safety Requirements

To ensure the well-being of users, the system should inform them of potential health risks associated with prolonged system use, such as eye strain or repetitive strain injuries. This can be achieved by displaying periodic reminders for users to take breaks, stretch, and rest their eyes. Additionally, providing tips on ergonomic setups and healthy usage habits can further promote safe interaction with the system (Oye, N. D., & Salleh, M. 2012).

## Security Requirements

The system must prioritize the security of sensitive user data through robust encryption methods. All sensitive information, including user credentials, personal details, and enrollment records, must be encrypted both in transit and at rest. The system will use SSL/TLS protocols for data transmission and AES encryption for stored data. User authentication must be secure and reliable, employing strong password policies, multi-factor authentication (MFA), and secure hashing algorithms like bcrypt for password storage. Additionally, the system should implement measures to prevent unauthorized access, such as account lockout mechanisms after multiple failed login attempts and secure session management practices (Almasri, H., & El-Sayed, M. 2016).

## Software Quality Attributes

The system should be designed to be highly available, reliable, and maintainable. High availability ensures that the system is accessible to users whenever needed, with minimal downtime. This can be achieved through redundancy, failover mechanisms, and robust disaster recovery plans. Reliability is crucial for maintaining user trust, requiring the system to perform consistently and correctly under various conditions. Maintainability is essential for allowing easy modifications and updates to the system. The design should support modularity, allowing individual components to be updated or replaced without affecting the entire system. Clear documentation, well-structured code, and comprehensive testing practices will facilitate maintainability and enable efficient troubleshooting and enhancements.

## Business Rules

Administrative users have the rights and privileges to manage system settings and user accounts, ensuring they can perform tasks such as creating, updating, and deleting user profiles, configuring system parameters, and generating reports. This administrative control is essential for maintaining the system's integrity and operational efficiency. In contrast, general users, including students and faculty, do not have the permissions to modify system settings, course data, or other users’ information. Their interactions with the system are limited to viewing course listings, enrolling in courses, managing their own accounts, and accessing related functionalities. This separation of roles and permissions ensures that system integrity is maintained and that sensitive operations are restricted to authorized personnel only.

# Other Requirements

## Appendix A: Glossary

API: Application Programming Interface

GUI: Graphical User Interface

HTTP: HyperText Transfer Protocol

REQ: Requirement

SRS: Software Requirements Specification

## Appendix B: Analysis Models

* Include any relevant diagrams or models here.

## Appendix C: To Be Determined List

* Any items that need further clarification or decision-making.

## A computer screen shot of a computer program Description automatically generatedUML Diagrams

(FIG 1. Course Enrollment System Class Diagrams)

The ER diagram represents a learning management system where a Learner registers for both ClassRegistration and CourseRegistration, with details of enrollment recorded. A CourseRegistration links Learner to a Course, which belongs to a Program. Similarly, ClassRegistration links Learner to a Class, which is scheduled within a Semester. Each Class has a ClassSchedule detailing the timing and location, and this schedule is associated with a Tutor who teaches the class. The diagram illustrates the interconnectedness of learners, courses, programs, classes, schedules, and tutors within the system.

A diagram of a system

Description automatically generated with medium confidence(FIG 2. Course Enrollment System Sequence Diagrams)

The sequence diagram illustrates the student registration process on a website. It begins with the student clicking the register tab, prompting the website to request and display the registration page from the web server. The student enters their information and submits it, which is then sent by the website to the web server. The web server forwards this information to the admin for verification. If the information is invalid, the admin sends an error message back to the web server, which then relays it to the website for the student to correct and re-enter. If the information is valid, the admin confirms it and instructs the database to save the new student information. Finally, an acceptance email is sent to the student, confirming their successful registration.

A screenshot of a computer

Description automatically generated

(FIG 3. Course Enrollment System Activity Diagrams)

A computer screen shot of a diagram

Description automatically generatedThe process begins with the student selecting a class within the system. The system first checks if the class is full. If the class is full, the process ends here. If the class is not full, the system then checks if the student has met the prerequisites for the class. If the prerequisites are not met, the student can request an override from the faculty. The faculty reviews the request and, if approved, overrides the prerequisite requirement. Once the prerequisites are satisfied or overridden, the system proceeds to add the class to the student's schedule. The system then updates the class schedule and enrollment records accordingly, completing the process.

(FIG 4. Course Enrollment System State Diagrams)

The enrollment process starts with a class being proposed. Once proposed, the class moves to the scheduled state. From there, it can become open for enrollment, allowing students to register. If the class reaches capacity, it transitions to a full state but can revert back to open for enrollment if spots become available. Alternatively, the class may close to enrollment, ending the registration period. Once closed, the class proceeds to being taught, signifying the start of the instructional period. After classes end, the class transitions to the final exams state. Throughout this process, a class can be cancelled at any point or students can drop the class, returning it to the initial state. Finally, once the final exams are completed, the class transitions to a closed state, marking the end of the enrollment process.

A computer screen shot of a diagram

Description automatically generated

(FIG 5. Course Enrollment System Use Case Diagram)

In the college registration system, a **Student** can perform various actions, including registering for courses, viewing available courses, dropping courses, and quitting college. Students can also log in to view and edit their profile, including their social status and phone number, and see their schedule, which they can further edit. They have access to information about their advisor as well. On the other hand, **Management Staff** can accept student registrations, add or delete courses, and view student information. When a student registers, the system includes login functionality to verify their credentials, and any errors encountered are handled appropriately. This use case diagram outlines the key functionalities available to students and management staff within the college registration system, highlighting the interactions and dependencies among various actions.

## Test Strategy

Component Testing

Component testing, also known as unit testing, involves testing individual components or modules of the software in isolation from the rest of the system. The primary goal is to verify that each component functions as intended independently of other components. In the context of the Online Course Registration System, component testing would include testing individual features such as the user registration process, login functionality, course listing retrieval, course enrollment, waitlist management, and enrollment cancellation. This testing scope includes validating the correctness of code logic, error handling, and data processing within each component. Tools commonly used for component testing include unit testing frameworks such as JUnit for Java, NUnit for .NET, or PyTest for Python. For example, ensuring the user registration form correctly validates user input, checking that the login function properly authenticates users, and verifying that the course enrollment function correctly updates the database are essential aspects of component testing (Lisa Crispin and Janet Gregory, 2009).Integration Testing

Integration testing involves testing the interaction between different components or modules of the system to ensure they work together correctly. The goal is to identify issues that may arise when integrating various parts of the system. In the Online Course Registration System, integration testing would cover scenarios such as the interaction between the user interface and the backend database, the communication between the enrollment module and the waitlist module, and the integration of third-party services like email notifications. This testing scope includes ensuring that modules communicate and cooperate as expected, detecting interface defects, and validating the integration of external systems. Tools for integration testing include Postman for API testing, Selenium for web interface testing, and continuous integration (CI) tools like Jenkins. Examples include verifying that a user who registers receives a confirmation email, ensuring that enrolling in a course updates both the course roster and the user's schedule, and checking that waitlist notifications are sent when a spot becomes available (Paul C. Jorgensen, 2013).System Testing

System testing involves testing the complete and integrated system to validate that it meets the specified requirements. This level of testing assesses the overall behavior, performance, and reliability of the entire system in a simulated production environment. For the Online Course Registration System, system testing would cover end-to-end scenarios such as user registration through to course enrollment and cancellation, as well as performance testing under various loads. The testing scope includes validating functional and non-functional requirements, including performance, security, usability, and compliance with regulations. Tools for system testing include LoadRunner for performance testing, QTP/UFT for automated functional testing, and security testing tools like OWASP ZAP. Examples include simulating multiple users accessing the system concurrently to test performance and load handling, verifying that the system meets security requirements for data protection, and checking the overall user experience for ease of navigation and accessibility (Mark Fewster and Dorothy Graham, 1999).

Acceptance Testing

Acceptance testing involves testing the system against the user requirements and business processes to determine if it is ready for deployment. The goal is to ensure that the system fulfills its intended purpose and meets the expectations of end users and stakeholders. In the context of the Online Course Registration System, acceptance testing would be performed by actual users (e.g., students, faculty, administrators) to validate that the system supports all necessary workflows and provides a satisfactory user experience. The testing scope includes ensuring that the system meets business requirements, user needs, and regulatory compliance. Acceptance testing can be facilitated through user acceptance testing (UAT) tools and platforms like TestRail, as well as manual testing by end users. Examples include students testing the complete registration process, including account creation, course search, enrollment, and waitlist management; faculty verifying their ability to manage course details and communicate with students; and administrators testing user management, reporting, and system configuration features (Rex Black, Erik van Veenendaal, and Dorothy Graham, 2012).

## How to run a PHP file in XAMPP.

A screenshot of a computer

Description automatically generatedTo run a PHP file in XAMPP:

1. Install XAMPP: Download and install XAMPP from the official website.
2. Start Apache and MySQL: Open the XAMPP Control Panel and start Apache and MySQL.
3. Place PHP Files: Copy your PHP files into the htdocs directory inside your XAMPP installation folder (e.g., C:\xampp\htdocs).

A screenshot of a computer

Description automatically generated

1. Access PHP File in Browser: Open a web browser and enter <http://localhost/index.php> to run your PHP file.

## MYSQL DATABASE AND DATABASE CONNECTION CUSTOM CLASS.

**Create the MySQL database and tables.**CREATE DATABASE Course\_Enrollment\_portal;

CREATE TABLE tblUser ( id INT AUTO\_INCREMENT PRIMARY KEY, email VARCHAR(255) NOT NULL, password VARCHAR(255) NOT NULL, firstName VARCHAR(50), lastName VARCHAR(50), address TEXT, phone VARCHAR(20), GPA DECIMAL(1, 1), Major VARCHAR(30));A screenshot of a computer

Description automatically generated

The SQL code creates a new MySQL database called Course\_Enrollment\_portal and a table named tblUser within it. The tblUser table has multiple columns: id (a primary key with auto-increment), email, password, firstName, lastName, address, phone, GPA, and Major. Each column is defined with its data type and constraints. For example, email and password are required fields with a maximum length of 255 characters, while GPA is a decimal number with up to 1 digit and 1 decimal places (Ullman 2016).

**Discuss the MySQL database functions that you used and the steps you took to create the database connection custom class.**

**PHP - DatabaseHandler Class**

The DatabaseHandler class handles database connections and queries. It sets the database connection parameters and uses PDO for connecting to the database. The connect method establishes the connection and handles any connection errors. The executeSelectQuery method prepares and executes SELECT queries with optional parameters, while the executeQuery method handles non-SELECT queries (INSERT, UPDATE, DELETE) and includes error handling for query execution (Nixon 2018).

**<?php**

**class** DatabaseHandler **{**

**private** **$host** **=** 'localhost'**;**

**private** **$db\_name** **=** 'Course\_Enrollment\_portal'**;**

**private** **$username** **=** 'root'**;**

**private** **$password** **=** ''**;**

**private** **$conn;**

**public** **function** connect**()** **{**

**$this->**conn **=** **null;**

**try** **{**

**$this->**conn **=** **new** PDO**(**"mysql:host=" **.** **$this->**host **.** ";dbname=" **.** **$this->**db\_name**,** **$this->**username**,** **$this->**password**);**

**$this->**conn**->exec(**"set names utf8"**);**

**}** **catch(**PDOException **$exception)** **{**

**echo** "Connection error: " **.** **$exception->**getMessage**();**

**}**

**return** **$this->**conn**;**

**}**

**public** **function** executeSelectQuery**($sql,** **$params** **=** **[])** **{**

**$stmt** **=** **$this->**conn**->**prepare**($sql);**

**foreach** **($params** **as** **$key** **=>** **$val)** **{**

**$stmt->**bindParam**($key,** **$val);**

**}**

**$stmt->**execute**();**

**return** **$stmt;**

**}**

// Added method for executing non-select queries like INSERT, UPDATE, DELETE

**public** **function** executeQuery**($sql,** **$params** **=** **[])** **{**

**try** **{**

**$stmt** **=** **$this->**conn**->**prepare**($sql);**

**foreach** **($params** **as** **$key** **=>** **$val)** **{**

// Use bindValue() to accommodate both bound variables and literals

**$stmt->**bindValue**($key,** **$val);**

**}**

**$stmt->**execute**();**

**return** **$stmt;**

**}** **catch(**PDOException **$exception)** **{**

**echo** "Execution error: " **.** **$exception->**getMessage**();**

**return** **null;**

**}**

**}**

**}**

**?>**

## LANDING PAGE, LOGIN PAGE, AND REGISTRATION.

**Landing Page (index.php)**

A screen shot of a computer

Description automatically generated

The Home Page script includes PHP error reporting and sets up the HTML head with Bootstrap and jQuery. It includes the master.php file for common elements like the navigation bar. The main content is centered and introduces the CST499 Course Enrollment System with a welcome message and a brief description of its services. The page concludes with a footer.

**<?php**

**error\_reporting(E\_ALL** **&** **~E\_NOTICE);**

**?>**

<!DOCTYPE html>

<html **lang**=**"en"**>

<head>

<title>Home Page</title>

<meta **charset**=**"utf-8"**>

<meta **name**=**"viewport"** **content**=**"width=device-width, initial-scale=1"**>

***<!-- Using HTTPS for all external resources to avoid mixed content issues -->***

<link **rel**=**"stylesheet"** **href**=**"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"**>

<script **src**=**"https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"**></script>

<script **src**=**"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"**></script>

</head>

<body>

**<?php** **require** 'master.php'**;** **?>**

<div **class**=**"container text-center"**>

<h1 **class**=**"display-4"**>Welcome to the <strong **class**=**"text-primary"**>CST499 Course Enrollment System</strong></h1>

<p **class**=**"lead"**>Empowering Your Success Academic Journey with Innovative Course Enrollment</p>

</div>

**<?php** **require\_once** 'footer.php'**;** **?>**

</body>

</html>

**PHP - Master Template**

The PHP code begins by configuring error reporting and starting a session if it hasn't already started. It sets various session parameters for enhanced security, such as using only cookies and ensuring they are HTTP-only and secure (when using HTTPS). The code then checks if a user is logged in by verifying if $\_SESSION['user\_id'] is set (Welling and Thomson 2016).. If the user is logged in, a welcome message is displayed; otherwise, it remains empty. The HTML structure includes a navigation bar and links to different pages, adjusting the displayed links based on the user's login status.

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

if (session\_status() == PHP\_SESSION\_NONE) {

    // Set session cookie parameters for security

    ini\_set('session.cookie\_httponly', '1');

    ini\_set('session.cookie\_secure', '1'); // Enable this if you are using HTTPS

    ini\_set('session.use\_only\_cookies', '1');

    // Start the session

    session\_start();

}

// Check if the username session variable is set and not empty

if (!empty($\_SESSION['user\_id'])) {

    $welcomeMessage = "Welcome #" . htmlspecialchars($\_SESSION['user\_id']) . " You are logged in";

} else {

    $welcomeMessage = '';

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"/>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>

<body>

    <div class="jumbotron">

        <div class="container text-center">

        <h1><strong class="text-primary">CST499 Course Enrollment System</strong><h1>

        <h1>Student Portal</h1>

            <?php if ($welcomeMessage !== ''): ?>

                <p><?= $welcomeMessage; ?></p>

            <?php endif; ?>

        </div>

    </div>

    <nav class="navbar navbar-inverse">

    <div class="container-fluid">

        <div class="navbar-header">

            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">

                <span class="icon-bar"></span>

                <span class="icon-bar"></span>

                <span class="icon-bar"></span>

            </button>

        </div>

        <div class="collapse navbar-collapse" id="myNavbar">

            <ul class="nav navbar-nav">

                <li class="active"><a href="index.php"><span class="glyphicon glyphicon-home"></span> Home</a></li>

                <li><a href="About Us.php"><span class="glyphicon glyphicon-exclamation-sign"></span> About Us</a></li>

                <li><a href="Contact Us.php"><span class="glyphicon glyphicon-earphone"></span> Contact Us</a></li>

            </ul>

            <ul class="nav navbar-nav navbar-right">

                <?php if (!empty($\_SESSION['user\_id'])): ?>

                    <li><a href="logout.php"><span class="glyphicon glyphicon-log-out"></span> Logout</a></li>

                    <li><a href="profile.php"><span class="glyphicon glyphicon-user"></span> Profile</a></li>

                <?php else: ?>

                    <li><a href="login.php"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>

                    <li><a href="registration.php"><span class="glyphicon glyphicon-registration-mark"></span> Registration</a></li>

                <?php endif; ?>

            </ul>

        </div>

    </div>

</nav>

**PHP - Footer Template**

The footer template sets up the HTML structure for a fixed-bottom footer. It includes Bootstrap for styling and jQuery for functionality. The footer content is centered and includes a copyright notice, with the year dynamically updated if necessary.

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Use HTTPS for CDN links to prevent mixed content warnings -->

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.6.0/css/bootstrap.min.css">

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.6.0/js/bootstrap.bundle.min.js"></script>

</head>

<body>

<div class="fixed-bottom">

    <div class="container text-center">

        <!-- Update the copyright year if necessary -->

        Copyright © 2024

    </div>

</div>

</body>

</html>

**PHP - Registration Script**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The registration script includes the DatabaseHandler.php file and sets up error reporting. It processes form submissions by capturing user input, hashing the password, and inserting the data into the tblUser table (Murach and Harris 2017). If the registration is successful, it displays a success message; otherwise, it shows an error. The HTML part includes a form for users to enter their registration details, with fields for email, password, name, address, phone, GPA, and major.

<?php

require\_once 'DatabaseHandler.php';

error\_reporting(E\_ALL & ~E\_NOTICE);

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

    // Capture form fields

    $email = $\_POST['email'];

    $password = password\_hash($\_POST['password'], PASSWORD\_DEFAULT); // Hash the password for security

    $firstName = $\_POST['firstName'];

    $lastName = $\_POST['lastName'];

    $address = $\_POST['address']; // New field

    $phone = $\_POST['phone']; // New field

    $GPA = $\_POST['GPA']; // New field

    $Major = $\_POST['Major']; // New field

    $db = new DatabaseHandler();

    $conn = $db->connect();

    // Update SQL statement with new fields

    $sql = "INSERT INTO tblUser (email, password, firstName, lastName, address, phone, GPA, Major) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    // Prepare statement

    $stmt = $conn->prepare($sql);

    // Bind parameters and execute

    if ($stmt->execute([$email, $password, $firstName, $lastName, $address, $phone, $GPA, $Major])) {

        echo "<div class='alert alert-success'>Registration successful!</div>";

    } else {

        echo "<div class='alert alert-danger'>Registration failed. Please try again.</div>";

    }

}

?>

<!DOCTYPE html>

<html>

<head>

    <title>Registration Form</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 500px;

            margin: 50px auto;

            background: #fff;

            padding: 20px;

            border-radius: 8px;

            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

        }

        h2 {

            color: #333;

            margin-bottom: 20px;

        }

        .form-group {

            margin-bottom: 15px;

        }

        .form-group label {

            display: block;

            color: #666;

            margin-bottom: 5px;

        }

        .form-control {

            width: 100%;

            padding: 10px;

            border-radius: 4px;

            border: 1px solid #ddd;

            box-sizing: border-box;

        }

        .btn-default {

            background-color: #5cb85c;

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            font-size: 16px;

        }

        .btn-default:hover {

            background-color: #4cae4c;

        }

        .btn-back {

            background-color: #6c757d; /\* Bootstrap's default gray \*/

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            font-size: 16px;

        }

        .btn-back:hover {

            background-color: #5a6268; /\* A darker shade of gray for hover effect \*/

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Register</h2>

        <form action="" method="post">

            <div class="form-group">

                <label for="email">Email:</label>

                <input type="email" class="form-control" id="email" name="email" required>

            </div>

            <div class="form-group">

                <label for="pwd">Password:</label>

                <input type="password" class="form-control" id="pwd" name="password" required>

            </div>

            <div class="form-group">

                <label for="firstName">First Name:</label>

                <input type="text" class="form-control" id="firstName" name="firstName" required>

            </div>

            <div class="form-group">

                <label for="lastName">Last Name:</label>

                <input type="text" class="form-control" id="lastName" name="lastName" required>

            </div>

            <div class="form-group">

                <label for="address">Address:</label>

                <input type="text" class="form-control" id="address" name="address" required>

            </div>

            <div class="form-group">

                <label for="phone">Phone:</label>

                <input type="tel" class="form-control" id="phone" name="phone" required>

            </div>

            <div class="form-group">

                <label for="GPA">GPA:</label>

                <input type="number" step="0.01" class="form-control" id="GPA" name="GPA" required>

            </div>

            <div class="form-group">

                <label for="Major">Major:</label>

                <input type="text" class="form-control" id="Major" name="Major" required>

            </div>

            <button type="submit" class="btn btn-default">Submit</button>

            <button type="button" onclick="window.location.href = 'index.php'" class="btn btn-default btn-back">Go Back</button>

        </form>

    </div>

</body>

</html>

A screenshot of a login screen

Description automatically generated**PHP - Login Script**

The login script starts a session and includes the DatabaseHandler.php file. It processes login requests by verifying the provided username and password against the database records (Simplilearn n.d.). If the credentials are correct, it sets session variables for the user and redirects to the home page. If not, it displays an error message. The HTML part includes a form for users to enter their login credentials, with styling and validation.

<?php

session\_start();

require\_once 'DatabaseHandler.php'; // Include the DatabaseHandler class

$dbHandler = new DatabaseHandler();

$conn = $dbHandler->connect();

if ($\_SERVER["REQUEST\_METHOD"] == "POST" && $conn) {

    $user\_username = $\_POST['username'];

    $user\_password = $\_POST['password'];

    $stmt = $dbHandler->executeSelectQuery("SELECT \* FROM tbluser WHERE email = :email", [':email' => $user\_username]);

    if ($stmt && $stmt->rowCount() == 1) {

        $user = $stmt->fetch(PDO::FETCH\_ASSOC);

        if (password\_verify($user\_password, $user['password'])) {

            $\_SESSION['user\_id'] = $user['id'];

            $\_SESSION['username'] = $user['username'];

            header("Location: index.php");

            exit;

        } else {

            $error\_message = "Invalid username or password.";

        }

    } else {

        $error\_message = "Invalid username or password.";

    }

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Login</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

            margin: 0;

            padding: 0;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

        }

        .login-container {

            background-color: #fff;

            padding: 40px;

            border-radius: 10px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            width: 300px;

        }

        h2 {

            text-align: center;

            color: #333;

            margin-bottom: 20px;

        }

        .form-group {

            margin-bottom: 15px;

        }

        label {

            display: block;

            margin-bottom: 5px;

            color: #666;

        }

        input[type="text"],

        input[type="password"] {

            width: 100%;

            padding: 10px;

            border: 1px solid #ddd;

            border-radius: 5px;

            box-sizing: border-box;

        }

        button,

        .back-button { /\* Added .back-button class for styling \*/

            width: 100%;

            padding: 10px;

            color: white;

            border: none;

            border-radius: 5px;

            cursor: pointer;

            font-size: 16px;

            margin-top: 10px; /\* Add some space between buttons \*/

        }

        button {

            background-color: #5cb85c; /\* Changed color to blue \*/

        }

        button:hover,

        .back-button:hover { /\* Added hover effect for .back-button \*/

            opacity: 0.8;

        }

        .back-button { /\* Specific styles for the back button \*/

            background-color: gray; /\* Changed background color to gray \*/

        }

        .error {

            color: #ff0000;

            margin-bottom: 15px;

            text-align: center;

        }

    </style>

</head>

<body>

    <div class="login-container">

        <h2>Login</h2>

        <?php if(isset($error\_message)): ?>

            <p class="error"><?php echo $error\_message; ?></p>

        <?php endif; ?>

        <form action="login.php" method="post">

            <div class="form-group">

                <label for="username">Username:</label>

                <input type="text" name="username" id="username" required>

            </div>

            <div class="form-group">

                <label for="password">Password:</label>

                <input type="password" name="password" id="password" required>

            </div>

            <button type="submit">Login</button>

            <button type="button" onclick="window.location.href = 'index.php'" class="back-button">Go Back</button>

        </form>

    </div>

</body>

</html>

**PHP - Logout Script**

The logout script starts a session, clears all session variables, destroys the session, and redirects the user to the home page. This ensures that the user is fully logged out and no session data remains.

<?php

session\_start(); // Start the session.

// Unset all session variables.

$\_SESSION = array();

// Destroy the session.

session\_destroy();

// Redirect to the login page.

header("Location: index.php");

exit;

?>

**PHP - Profile Page**

A screen shot of a computer

Description automatically generated

The profile page script starts a session and includes the DatabaseHandler.php file. It checks if a user is logged in and fetches the user's information from the database. The HTML structure displays the user's profile details such as email, name, address, phone, GPA, and major, with an option to change the password. The page uses Bootstrap for styling and includes the navigation bar and footer from master.php.

<?php

session\_start();

require\_once 'DatabaseHandler.php'; // Include the DatabaseHandler class

// Check if the user is not logged in, then redirect to login page

if (!isset($\_SESSION['user\_id'])) {

    header('Location: login.php');

    exit;

}

$dbHandler = new DatabaseHandler();

$conn = $dbHandler->connect();

// Assuming the user's ID is stored in the session

$user\_id = $\_SESSION['user\_id'];

// Fetch user information from the database

$stmt = $dbHandler->executeSelectQuery("SELECT email, firstName, lastName, address, phone, GPA, Major FROM tbluser WHERE id = :id", [':id' => $user\_id]);

if ($stmt && $stmt->rowCount() > 0) {

    $user = $stmt->fetch(PDO::FETCH\_ASSOC);

} else {

    // Handle error or redirect if user data cannot be fetched

    echo "User data not found.";

    exit;

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Profile Page</title>

    <!-- Ensure you link to the CSS files for Bootstrap and Font Awesome if you're using them -->

    <link rel="stylesheet" href="path/to/bootstrap.min.css">

    <link rel="stylesheet" href="path/to/font-awesome.min.css">

    <link rel="stylesheet" href="path/to/your/custom/styles.css">

    <!-- Alternatively, you can include the style directly here -->

    <style>

        .profile-card {

            background-color: #fff;

            border-radius: 8px;

            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

            padding: 20px;

            margin: 20px auto;

            width: 80%;

            max-width: 600px;

        }

        .profile-title {

            color: #333;

            margin-bottom: 20px;

        }

        .profile-detail {

            margin: 10px 0;

            color: #555;

            font-size: 16px;

        }

        .profile-detail i {

            color: #777;

            width: 30px;

            text-align: center;

            margin-right: 10px;

        }

        .detail-value {

            font-weight: bold;

            color: #333;

        }

    </style>

</head>

<body>

<?php include 'master.php'; ?>

<div class="container text-center">

    <h1>Welcome to your Profile</h1>

    <div class="profile-card">

    <div class="container text-left">

        <p class="profile-detail"><i class="fa fa-envelope"></i>Email: <span class="detail-value"><?= htmlspecialchars($user['email']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-user"></i>First Name: <span class="detail-value"><?= htmlspecialchars($user['firstName']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-user"></i>Last Name: <span class="detail-value"><?= htmlspecialchars($user['lastName']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-home"></i>Address: <span class="detail-value"><?= htmlspecialchars($user['address']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-phone"></i>Phone: <span class="detail-value"><?= htmlspecialchars($user['phone']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-dollar-sign"></i>GPA: <span class="detail-value"><?= htmlspecialchars($user['GPA']) ?></span></p>

        <p class="profile-detail"><i class="fa fa-id-card"></i>Major: <span class="detail-value"><?= htmlspecialchars($user['Major']) ?></span></p>

        <div class="change-password-button">

            <a href="change\_password.php" class="btn btn-warning">Change Password</a>

        </div>

    </div>

</div>

<?php include 'footer.php'; ?>

</body>

</html>

**PHP - Change Password**

A screenshot of a computer

Description automatically generated

The Change Password script starts a session and includes the DatabaseHandler.php file to handle database connections. It checks if a user is logged in and redirects them to the login page if not. Upon form submission via POST, it verifies the current password, compares the new passwords, hashes the new password if they match, and updates it in the database (Techgeekbuzz n.d.). The script provides feedback on the success or failure of these operations. The HTML part includes a form for users to enter their current and new passwords.

<?php

session\_start();

require\_once 'DatabaseHandler.php'; // Include the DatabaseHandler class

// Ensure the user is logged in

if (!isset($\_SESSION['user\_id'])) {

    header('Location: login.php');

    exit;

}

// Initialize the DatabaseHandler

$dbHandler = new DatabaseHandler();

$conn = $dbHandler->connect();

if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

    $currentPassword = $\_POST['current\_password'];

    $newPassword = $\_POST['new\_password'];

    $confirmNewPassword = $\_POST['confirm\_new\_password'];

    $userId = $\_SESSION['user\_id'];

    // Fetch the current hashed password from the database

    $stmt = $dbHandler->executeSelectQuery("SELECT password FROM tbluser WHERE id = :id", [':id' => $userId]);

    if ($stmt && $stmt->rowCount() > 0) {

        $user = $stmt->fetch(PDO::FETCH\_ASSOC);

        $hashedPassword = $user['password'];

        if (password\_verify($currentPassword, $hashedPassword)) {

            if ($newPassword === $confirmNewPassword) {

                // Hash the new password

                $newHashedPassword = password\_hash($newPassword, PASSWORD\_DEFAULT);

                // Update the password in the database

                $updateStmt = $dbHandler->executeQuery("UPDATE tbluser SET password = :password WHERE id = :id", [':password' => $newHashedPassword, ':id' => $userId]);

                if ($updateStmt) {

                    echo "<p>Password changed successfully.</p>";

                } else {

                    echo "<p>Error updating password.</p>";

                }

            } else {

                echo "<p>New passwords do not match.</p>";

            }

        } else {

            echo "<p>Current password is incorrect.</p>";

        }

    } else {

        echo "<p>User not found.</p>";

    }

}

?>

<!-- HTML content follows -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Change Password</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f2f2f2;

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 400px;

            margin: 50px auto;

            background-color: #fff;

            padding: 20px;

            border-radius: 8px;

            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

        }

        h2 {

            margin-top: 0;

        }

        .form-group {

            margin-bottom: 20px;

        }

        label {

            display: block;

            margin-bottom: 5px;

            font-weight: bold;

        }

        input[type="password"] {

            width: 100%;

            padding: 10px;

            border: 1px solid #ccc;

            border-radius: 4px;

            box-sizing: border-box;

        }

        button {

            padding: 10px 20px;

            background-color: #007bff;

            color: #fff;

            border: none;

            border-radius: 4px;

            cursor: pointer;

            transition: background-color 0.3s;

        }

        button:hover {

            background-color: #0056b3;

        }

        .goback-btn {

            background-color: black;

            color: white;

        }

        .goback-btn:hover {

            background-color: #333;

        }

    </style>

</head>

<body>

<div class="container">

    <h2>Change Password</h2>

    <form action="change\_password.php" method="post">

        <div class="form-group">

            <label for="current\_password">Current Password:</label>

            <input type="password" name="current\_password" required>

        </div>

        <div class="form-group">

            <label for="new\_password">New Password:</label>

            <input type="password" name="new\_password" required>

        </div>

        <div class="form-group">

            <label for="confirm\_new\_password">Confirm New Password:</label>

            <input type="password" name="confirm\_new\_password" required>

        </div>

        <button type="submit">Change Password</button>

        <button type="button" onclick="window.location.href = 'index.php'" class="back-button goback-btn">Go Back</button>

    </form>

</div>

</body>

</html>

A screenshot of a computer

Description automatically generated

**PHP - About Us Page**

A screen shot of a computer

Description automatically generated

The About Us page starts with PHP error reporting and includes necessary HTML head elements, including Bootstrap and jQuery for styling and functionality (Edureka n.d.). It includes the master.php file, which contains common elements like the navigation bar. The page content features a section with information about the CST499 Course Enrollment System, highlighting its mission and services. It also provides a button to navigate to the Contact Us page. The page ends by including a footer.

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <title>About Us</title>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"/>

    <!-- Using HTTPS for all external resources to avoid mixed content issues -->

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>

<body>

<?php require 'master.php'; // This includes your navigation bar and any other repeated elements ?>

<section class="container my-5 p-5 text-center bg-light rounded shadow-lg">

    <header>

        <h1 class="mb-4 display-4">About Us</h1>

    </header>

    <article>

        <p class="lead">

            Here at <strong class="text-primary">CST499 Course Enrollment System</strong>, we specialize in simplifying academic enrollment processes. Founded in 2024, our mission is to streamline course registration through innovative technology. Our team of experts is dedicated to providing cutting-edge solutions, ensuring seamless integration and improved efficiency for your educational institution.

        </p>

    </article>

    <div class="mt-4">

        <a href="Contact Us.php" class="btn btn-primary btn-lg">Get in Touch</a>

    </div>

</section>

<?php require\_once 'footer.php'; // This includes your footer ?>

</body>

</html>

**PHP - Contact Us Page**

A screen shot of a computer

Description automatically generated

The Contact Us page includes PHP error reporting and sets up the HTML head with Bootstrap and jQuery. It includes the master.php file for common elements and contains a section that provides contact information for the CST499 Course Enrollment System. The contact details include phone numbers, email addresses, and the office location. The page is designed with a structured layout using Bootstrap's grid system, and it concludes with a footer (Code with Faraz n.d.).

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <title>Contact Us</title>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"/>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

</head>

<body>

<?php require 'master.php'; // This includes your navigation bar and any other repeated elements ?>

<div class="container">

    <h1 class="text-center">Contact Us</h1>

    <p class="text-center">At <strong class="text-primary">CST499 Course Enrollment System</strong>, we're always ready to assist you. Whether you have questions, need support, or want more information about our course enrollment services, we're here to help. </p>

    <div class="row">

        <div class="col-md-6">

            <h2>Get in Touch</h2>

            <p><strong>Phone:</strong> +1 (555) 123-4567<br>

            Available from 9:00 AM to 6:00 PM, Monday to Friday (Eastern Time)</p>

            <p><strong>Email:</strong> info@cst499coursesystem.com<br>

            We'll get back to you as soon as possible.</p>

            <p><strong>Office Location:</strong> 123 Main Street, Suite 100, Cityville, ST 12345<br>

            Visit us for a personal consultation during our office hours.</p>

        </div>

        <div class="col-md-6">

            <h2>Customer Support</h2>

            <p>Our team is committed to providing you with the highest level of service. For any support-related inquiries, please contact us.</p>

            <p><strong>Support Hotline:</strong> +1 (555) 987-6543</p>

            <p><strong>Support Email:</strong> support@cst499coursesystem.com</p>

        </div>

    </div>

    <div class="row">

        <div class="col-md-12">

            <h2>Feedback and Inquiries</h2>

            <p>Your feedback is invaluable to us. For feedback, suggestions, or specific inquiries, please reach out. We're always looking to improve and serve you better.</p>

        </div>

    </div>

</div>

<?php require\_once 'footer.php'; // This includes your footer ?>

</body>

</html>

## CREATE TABLES WITHIN THE MYSQL DATABASE.

**Create your tables within the MySQL database.**

Course\_enrollment\_portal already have the tblUser table created. Next, let's create the tables needed for courses and enrollment:

CREATETABLEtblCourse **(**

course\_id INTAUTO\_INCREMENT PRIMARYKEY**,**

course\_name **VARCHAR(**100**)** NOTNULL**,**

course\_number **VARCHAR(**10**)** NOTNULL**,**

instructor\_name **VARCHAR(**100**),**

credits **INT,**

semester **VARCHAR(**50**)**

**);**

**phpMyAdmin 
Recent Favorites 
_e New 
course enrollment_portal 
New 
tblcourse 
tblenro"rnent 
employee_portal 
information schema 
mysql 
newusers_portal 
online_food ordering_database 
performance_schema 
Browse 
E SQL 
Structure 
Search Insert 
Export 
Import 
MySQL retumed an empty result set (i.e. zero rows). (Query took 0.0001 seconds.) 
SELECT * FROM •tblcourse• 
Profiling [ Edit inline I [ Edit] [ Explain SQL ] Create PHP code ] Refresh I 
course id course name course number instructor name credits semester 
Query results operations 
Create view 
Bookmark this SQL query 
D Let every user access this bookmark 
Label: 
phpmyadmin 
Bookmark this SQL query **

CREATETABLEtblEnrollment **(**

enrollment\_id INTAUTO\_INCREMENT PRIMARYKEY**,**

**user\_id** **INT,**

course\_id **INT,**

FOREIGNKEY **(user\_id)** REFERENCEStblUser**(**id**),**

FOREIGNKEY **(**course\_id**)** REFERENCEStblCourse**(**course\_id**)**

**);**

**phpMyAdmin 
Recent Favorites 
_e New 
course enrollment_portal 
New 
tblcourse 
tblenro"rnent 
tbluser 
employee_portal 
information schema 
mysql 
newusers_portal 
online_food ordering_database 
performance_schema 
phpmyadmin 
Browse 
E SQL 
Structure 
Search Insert 
Export 
Import 
Privile 
MySQL retumed an empty result set (i.e. zero rows). (Query took 0.0002 seconds.) 
SELECT * FROM tbIenroIIment• 
Profiling [ Edit inline I [ Edit] [ Explain SQL] [Create PHP code ] 
enrolhent id user id course id 
Refresh ] 
Query results operations 
Create view 
Bookmark this SQL query 
Label: 
Bookmark this SQL query 
D Let every user access this bookmark **

**Display database structure**

A screenshot of a computer

Description automatically generated

Below The SQL code provided consists of two main parts: inserting course information into a table called tblCourse and enrolling a user in courses via another table called tblEnrollment.

**INSERT** **INTO** tblCourse **(**course\_name**,** course\_number**,** instructor\_name**,** credits**,** semester**)** **VALUES**

**(**'Introduction to Computer Science'**,** 'CS101'**,** 'Dr. Smith'**,** 3**,** 'Fall 2024'**),**

**(**'Data Structures'**,** 'CS201'**,** 'Dr. Johnson'**,** 4**,** 'Spring 2024'**),**

**(**'Algorithms'**,** 'CS301'**,** 'Dr. Brown'**,** 4**,** 'Fall 2024'**),**

**(**'Database Systems'**,** 'CS401'**,** 'Dr. White'**,** 3**,** 'Spring 2024'**),**

**(**'Operating Systems'**,** 'CS501'**,** 'Dr. Green'**,** 4**,** 'Fall 2024'**),**

**(**'Computer Networks'**,** 'CS601'**,** 'Dr. Black'**,** 3**,** 'Spring 2024'**),**

**(**'Machine Learning'**,** 'CS701'**,** 'Dr. Gray'**,** 3**,** 'Fall 2024'**),**

**(**'Artificial Intelligence'**,** 'CS801'**,** 'Dr. Violet'**,** 4**,** 'Spring 2024'**),**

**(**'Cybersecurity'**,** 'CS901'**,** 'Dr. Gold'**,** 3**,** 'Fall 2024'**),**

**(**'Software Engineering'**,** 'CS1001'**,** 'Dr. Silver'**,** 4**,** 'Spring 2024'**);**

**INSERT** **INTO** tblEnrollment **(user\_id,** course\_id**)** **VALUES**

**(**3**,** 1**),** ***-- Enroll user with id=3 in 'Introduction to Computer Science'***

**(**3**,** 3**);** ***-- Enroll user with id=3 in 'Algorithms'***

## LIST, ADD and DELETE CLASSES FROM THE SCHEDULE.

**Add new line in the master php**

<li><a href="my\_courses.php"><span class="glyphicon glyphicon-book"></span> My Courses</a></li>

**Home Page 
Google Notion 
localhost/ 127.0.0.1 / course_e 
localhost/index.php 
a GPT 
Gemini 
Claude 
D 
Magoosh Papago YouGIish 
iconfinder 
drawio 
Koyrfin 
E*TRADE 
TradingView 
V ETF Database: 
ETF Research Center 
ASULibrary 
UAGC 
My CPETracker 
Replit 
G 
All Bookmarks 
Home 
O About us 
Contact us 
CST499 Course Enrollment System 
Student Portal 
Welcome #3 You are logged in 
Welcome to the CST499 Course Enrollment System 
Empowering Your Success Academic Journey with Innovative Course Enrollment 
Copyright 0 2024 
add my coursc 
Profile Il My Courses C* Logout 
1:56 PM 
6/8/2024 **This line of code creates a navigation link in the form of a list item that, when clicked, will take the user to the my\_courses.php page. It includes a book icon from Bootstrap's Glyphicon set and the text "My Courses" to inform the user about the purpose of the link. This link is part of the conditional navigation menu that only shows if the user is logged in (i.e., if the $\_SESSION['user\_id'] is not empty).

A screen shot of a computer program

Description automatically generated

**PHP – My Courses**

1. **Initial Setup**:

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

session\_start();

require 'DatabaseHandler.php'; // Include your DatabaseHandler class file

This section sets up error reporting to ignore notices, starts the session, and includes the DatabaseHandler.php file, which likely contains the database connection logic.

1. **User Authentication**:

// Check if the user is logged in

if (empty($\_SESSION['user\_id'])) {

    header('Location: login.php');

    exit;

}

This checks if the user is logged in by verifying if $\_SESSION['user\_id'] is set. If not, it redirects to the login page.

1. **Database Connection**:

$user\_id = $\_SESSION['user\_id'];

$db = new DatabaseHandler();

$conn = $db->connect();

It retrieves the user ID from the session and creates a new instance of DatabaseHandler to establish a database connection.

1. **Fetch Current Classes**:

// Fetch current classes for the logged-in user

$query = "SELECT c.course\_id, c.course\_name, c.course\_number, c.instructor\_name, c.credits, c.semester

          FROM tblCourse c

          JOIN tblEnrollment e ON c.course\_id = e.course\_id

          WHERE e.user\_id = :user\_id";

$params = [':user\_id' => $user\_id];

$current\_classes = $db->executeSelectQuery($query, $params);

This query retrieves the current classes the user is enrolled in by joining the tblCourse and tblEnrollment tables.

A screen shot of a computer

Description automatically generated

1. **Fetch Available Classes**:

// Fetch all available classes

$query = "SELECT course\_id, course\_name, course\_number, instructor\_name, credits, semester

          FROM tblCourse WHERE course\_id NOT IN

          (SELECT course\_id FROM tblEnrollment WHERE user\_id = :user\_id)";

$available\_classes = $db->executeSelectQuery($query, $params);

The provided SQL query retrieves a list of all available classes that a specific user has not yet enrolled in. The query selects course details including the course ID, course name, course number, instructor name, credits, and semester from the tblCourse table. It excludes any courses that are already associated with the user in the tblEnrollment table. This exclusion is achieved by using a subquery that selects course IDs from tblEnrollment where the user\_id matches the given user, ensuring that only non-enrolled courses for that user are fetched. The $params array likely contains the user\_id parameter necessary for the query's execution (CampCodes, n.d.).A screenshot of a computer

Description automatically generated

1. **Handle Adding a Class**:

// Handle adding a class

if ($\_SERVER['REQUEST\_METHOD'] == 'POST' && isset($\_POST['add\_class\_id'])) {

    $class\_id = $\_POST['add\_class\_id'];

    $query = "INSERT INTO tblEnrollment (user\_id, course\_id) VALUES (:user\_id, :course\_id)";

    $params = [':user\_id' => $user\_id, ':course\_id' => $class\_id];

    $db->executeQuery($query, $params);

    header('Location: my\_courses.php');

    exit;

}

The provided PHP code handles the process of adding a class to a user's enrollment. When the form is submitted via a POST request and includes the add\_class\_id field, the script extracts the class ID from the POST data. It then constructs an SQL query to insert a new record into the tblEnrollment table with the specified user ID and course ID. The $params array includes the user ID and class ID for the query execution. The executeQuery method is called to perform the insertion. After successfully adding the class, the user is redirected to the my\_courses.php page, and the script terminates to ensure the redirection occurs immediately (SourceCodester, 2021).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Handle Deleting a Class**:

// Handle deleting a class

if (isset($\_GET['delete\_class\_id'])) {

    $class\_id = $\_GET['delete\_class\_id'];

    $query = "DELETE FROM tblEnrollment WHERE user\_id = :user\_id AND course\_id = :course\_id";

    $params = [':user\_id' => $user\_id, ':course\_id' => $class\_id];

    $db->executeQuery($query, $params);

    header('Location: my\_courses.php');

    exit;

}

The provided PHP code handles the process of deleting a class from a user's enrollment. When the script detects a GET request with the parameter delete\_class\_id, it extracts the class ID from the query string. It then constructs an SQL query to delete the corresponding record from the tblEnrollment table where the user ID and course ID match the provided values. The $params array includes the user ID and class ID for the query execution. The executeQuery method is called to perform the deletion. After successfully removing the class, the user is redirected to the my\_courses.php page, and the script terminates to ensure the redirection occurs immediately (ITSourceCode, 2021).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **HTML Structure**:

?>

<?php include 'master.php'; ?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"/>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

    <title>My Courses</title>

</head>

<body>

    <div class="container">

        <h2>My Current Classes</h2>

        <ul class="list-group">

            <?php while ($row = $current\_classes->fetch(PDO::FETCH\_ASSOC)): ?>

                <li class="list-group-item">

                    <strong><?= htmlspecialchars($row['course\_name']) ?> (<?= htmlspecialchars($row['course\_number']) ?>)</strong>

                    - Instructor: <?= htmlspecialchars($row['instructor\_name']) ?>, Credits: <?= htmlspecialchars($row['credits']) ?>, Semester: <?= htmlspecialchars($row['semester']) ?>

                    <a href="my\_courses.php?delete\_class\_id=<?= $row['course\_id'] ?>" class="btn btn-danger btn-sm pull-right">Delete</a>

                </li>

            <?php endwhile; ?>

        </ul>

        <h2>Available Classes</h2>

        <ul class="list-group">

            <?php while ($row = $available\_classes->fetch(PDO::FETCH\_ASSOC)): ?>

                <li class="list-group-item">

                    <strong><?= htmlspecialchars($row['course\_name']) ?> (<?= htmlspecialchars($row['course\_number']) ?>)</strong>

                    - Instructor: <?= htmlspecialchars($row['instructor\_name']) ?>, Credits: <?= htmlspecialchars($row['credits']) ?>, Semester: <?= htmlspecialchars($row['semester']) ?>

                    <form method="post" action="my\_courses.php" class="pull-right">

                        <input type="hidden" name="add\_class\_id" value="<?= $row['course\_id'] ?>">

                        <button type="submit" class="btn btn-success btn-sm">Add</button>

                    </form>

                </li>

            <?php endwhile; ?>

        </ul>

    </div>

</body>

</html>

This section includes the HTML and Bootstrap CSS/JS for styling. It lists the user's current classes and available classes, allowing them to add or delete classes through forms and buttons.

**Full Script:**

<?php

error\_reporting(E\_ALL & ~E\_NOTICE);

session\_start();

require 'DatabaseHandler.php'; // Include your DatabaseHandler class file

// Check if the user is logged in

if (empty($\_SESSION['user\_id'])) {

    header('Location: login.php');

    exit;

}

$user\_id = $\_SESSION['user\_id'];

$db = new DatabaseHandler();

$conn = $db->connect();

// Fetch current classes for the logged-in user

$query = "SELECT c.course\_id, c.course\_name, c.course\_number, c.instructor\_name, c.credits, c.semester

          FROM tblCourse c

          JOIN tblEnrollment e ON c.course\_id = e.course\_id

          WHERE e.user\_id = :user\_id";

$params = [':user\_id' => $user\_id];

$current\_classes = $db->executeSelectQuery($query, $params);

// Fetch all available classes

$query = "SELECT course\_id, course\_name, course\_number, instructor\_name, credits, semester

          FROM tblCourse WHERE course\_id NOT IN

          (SELECT course\_id FROM tblEnrollment WHERE user\_id = :user\_id)";

$available\_classes = $db->executeSelectQuery($query, $params);

// Handle adding a class

if ($\_SERVER['REQUEST\_METHOD'] == 'POST' && isset($\_POST['add\_class\_id'])) {

    $class\_id = $\_POST['add\_class\_id'];

    $query = "INSERT INTO tblEnrollment (user\_id, course\_id) VALUES (:user\_id, :course\_id)";

    $params = [':user\_id' => $user\_id, ':course\_id' => $class\_id];

    $db->executeQuery($query, $params);

    header('Location: my\_courses.php');

    exit;

}

// Handle deleting a class

if (isset($\_GET['delete\_class\_id'])) {

    $class\_id = $\_GET['delete\_class\_id'];

    $query = "DELETE FROM tblEnrollment WHERE user\_id = :user\_id AND course\_id = :course\_id";

    $params = [':user\_id' => $user\_id, ':course\_id' => $class\_id];

    $db->executeQuery($query, $params);

    header('Location: my\_courses.php');

    exit;

}

?>

<?php include 'master.php'; ?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"/>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

    <title>My Courses</title>

</head>

<body>

    <div class="container">

        <h2>My Current Classes</h2>

        <ul class="list-group">

            <?php while ($row = $current\_classes->fetch(PDO::FETCH\_ASSOC)): ?>

                <li class="list-group-item">

                    <strong><?= htmlspecialchars($row['course\_name']) ?> (<?= htmlspecialchars($row['course\_number']) ?>)</strong>

                    - Instructor: <?= htmlspecialchars($row['instructor\_name']) ?>, Credits: <?= htmlspecialchars($row['credits']) ?>, Semester: <?= htmlspecialchars($row['semester']) ?>

                    <a href="my\_courses.php?delete\_class\_id=<?= $row['course\_id'] ?>" class="btn btn-danger btn-sm pull-right">Delete</a>

                </li>

            <?php endwhile; ?>

        </ul>

        <h2>Available Classes</h2>

        <ul class="list-group">

            <?php while ($row = $available\_classes->fetch(PDO::FETCH\_ASSOC)): ?>

                <li class="list-group-item">

                    <strong><?= htmlspecialchars($row['course\_name']) ?> (<?= htmlspecialchars($row['course\_number']) ?>)</strong>

                    - Instructor: <?= htmlspecialchars($row['instructor\_name']) ?>, Credits: <?= htmlspecialchars($row['credits']) ?>, Semester: <?= htmlspecialchars($row['semester']) ?>

                    <form method="post" action="my\_courses.php" class="pull-right">

                        <input type="hidden" name="add\_class\_id" value="<?= $row['course\_id'] ?>">

                        <button type="submit" class="btn btn-success btn-sm">Add</button>

                    </form>

                </li>

            <?php endwhile; ?>

        </ul>

    </div>

</body>

</html>

References and Acknowledgments

Oye, N. D., & Salleh, M. (2012). Implementation of Online Course Registration System in University of Maiduguri. *International Journal of Computer Applications*, 59(10), 1-8. doi:10.5120/9600-4097

Almasri, H., & El-Sayed, M. (2016). A Web-Based Online Registration System for Academic Institutes. *International Journal of Advanced Computer Science and Applications*, 7(2), 212-217. doi:10.14569/IJACSA.2016.070230

Yusof, N. M., & Wahid, N. (2014). Design and Development of an Online Course Registration System for the School of Science and Technology at Universiti Malaysia Sabah. *Journal of Computing*, 4(7), 146-152. doi:10.1109/ICSESS.2014.6933542

Naik, S. (2017). Online Course Registration System. *IOSR Journal of Computer Engineering*, 19(2), 52-56. Retrieved from IOSR Journals

Panda, D., & Pati, B. (2015). A Study on Online Student Course Registration System with Special Reference to Online Admission System. *International Journal of Innovative Research in Computer and Communication Engineering*, 3(5), 4103-4109. doi:10.15680/IJIRCCE.2015.0305041

Oriental Journal of Computer Science and Technology. (n.d.). Development of Online Student Course Registration System. Retrieved from [https://www.computerscijournal.org](https://www.computerscijournal.org/)

Springer. (n.d.). Towards Efficient Distance Studies: Online Course Management System. In Advances in Web-Based Learning. Retrieved from [https://link.springer.com](https://link.springer.com/)

IEEE Xplore. (n.d.). Design and Implementation of the Online Course Registration System at Tsinghua University. Retrieved from [https://ieeexplore.ieee.org](https://ieeexplore.ieee.org/)

ResearchGate. (n.d.). Online Course Registration System Report. Retrieved from [https://www.researchgate.net](https://www.researchgate.net/)

Academia.edu. (n.d.). Design and Implementation of a Mobile Students' Course Registration Platform. Retrieved from [https://www.academia.edu](https://www.academia.edu/)

Crispin, L., & Gregory, J. (2009). Agile testing: A practical guide for testers and agile teams. Addison-Wesley.

Jorgensen, P. C. (2013). Software testing: A craftsman’s approach (4th ed.). Auerbach Publications.

Fewster, M., & Graham, D. (1999). Software test automation: Effective use of test execution tools. Addison-Wesley.

Black, R., van Veenendaal, E., & Graham, D. (2012). Foundations of software testing: ISTQB certification (2nd ed.). Cengage Learning EMEA.

Ullman, L. (2016). PHP for the web: Visual quickstart guide (5th ed.). Peachpit Press.

Nixon, R. (2018). Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5 (5th ed.). O'Reilly Media.

Welling, L., & Thomson, L. (2016). PHP and MySQL web development (5th ed.). Addison-Wesley Professional.

Murach, J., & Harris, R. (2017). Murach's PHP and MySQL (3rd ed.). Mike Murach & Associates.

Simplilearn. (n.d.). How to run a PHP using XAMPP. Retrieved from <https://www.simplilearn.com>

Techgeekbuzz. (n.d.). How to run PHP program in XAMPP? [A Complete Guide]. Retrieved from <https://www.techgeekbuzz.com>

Edureka. (n.d.). How to run a PHP program in XAMPP? Step-by-step guide. Retrieved from <https://www.edureka.co>

Code with Faraz. (n.d.). Step by step guide: Connecting PHP to MySQL database with XAMPP. Retrieved from <https://www.codewithfaraz.com>

CampCodes. (n.d.). Online student enrollment system in PHP MySQL free source code. CampCodes. Retrieved June 8, 2024, from <https://www.campcodes.com/projects/php/online-student-enrollment-system-in-php-mysql-free-source-code>

SourceCodester. (2021, July 7). Online student enrollment system using PHP/MySQLi free source code. SourceCodester. Retrieved June 8, 2024, from <https://www.sourcecodester.com/php/online-student-enrollment-system-php-mysqli-free-source-code.html>

ITSourceCode. (2021, July 7). Enrollment system project with source code using PHP/MySQL. ITSourceCode. Retrieved June 8, 2024, from https://itsourcecode.com/free-projects/php-projects/enrollment-system-project-with-source-code-using-php-mysql