Week 5 Final Assignment—Course Enrollment System Documentation

Alicia Piavis

CST499: Capstone for Computer Software Technology

Amr Elchouemi

6/7/2020

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Software Requirements Specification

1.1 Introduction

1.1.1 Purpose

This software requirements specification (SRS) document outlines the requirements for the Course Enrollment System build. The purpose of the Course Enrollment System is to allow students the ability to manage their school course schedule through a simple self-service website. This document refers to version 1.0 of the Course Enrollment System. The scope of this SRS encompasses the entirety of the initial build of the Course Enrollment System, including the basic features described in section 4, as well as other non-functional requirements described in section 5.

1.1.2 Document Conventions

This document is broken into 5 sections with headers for Introduction, Overall Description, External Interface Requirements, System Features, and Other Non-Functional Requirements. Sub-headers provide detail regarding the specifics of those sections.

1.1.3 Intended Audience and Reading Suggestions

This SRS is intended for UI/UX designers, software designers, architects, developers, project managers, testers, and other stakeholders involved in the execution of this project. It is suggested that all audience members read the document from start to finish in order to gain a thorough understanding of the purpose and requirements of the Course Enrollment System.

1.1.4 Product Scope

The goal of the Course Enrollment System is to allow a user to manage their school course schedule quickly and easily. The website will allow a user to register and create an account, log in, view their course schedule, search for available courses, register for a course, drop a course, and be added to the waitlist for a course if it is full. The development of this Course Enrollment System aligns with the

business objectives to increase enrollment, retain students, improve the overall student experience, and reduce the cost and overhead associated with the current manual course registration process.

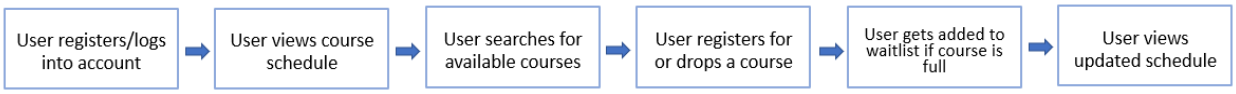
1.2 Overall Description

1.2.1 Product Perspective

The build for the Course Enrollment System is a new, self-contained product.

1.2.2 Product Functions

A user should be able to register and create an account, log in, view their course schedule, search for available courses, register for a course, drop a course, and be added to the waitlist for a course if it is full. In addition, the website should reflect that there is a maximum number of students allowed in a course, so if a course is full, the system should add the student to the waitlist. When a spot in a full course becomes available, the system should notify the student of the new availability.



1.2.3 User Classes and Characteristics

User classes for the Course Enrollment System include students, staff in the Registrar's Office, and administrators. The student class will need to be able to create an account, log in, view their course schedule, search for available courses, register for a course, and drop a course. The staff class will need to be able to add, update, and delete course offerings. Administrators will need to be able to access all functionalities, including assisting users in logging into their accounts and resetting their login information. Administrators should also be able to look up a student's schedule by student ID number. The primary user class for the Course Enrollment System is the student user class. In addition to the user classes, there will also be a Course class, which has attributes describing the course, and methods that control the actions that the Course class can take, such as creating and deleting enrollments.

1.2.4 Operating Environment

The Course Enrollment System needs to have cross-platform functionality, as well as be mobile responsive. The website should function on all major web browsers, including Chrome, Internet Explorer, Edge, Bing, Yahoo, and Safari. In addition, it should function properly on both Mac and Windows operating systems, as well as iOS and Android.

1.2.5 Design and Implementation Constraints

The Course Enrollment System needs to be built within 6 months, and must utilize a relational database. In addition, all passwords for user accounts must use encryption. In order to comply with the school's policies, the website must abide by ADA web accessibility guidelines (ADA, 2007), as well as FERPA guidelines. Furthermore, the system will be maintained by the school, so the design and build should support modifiability, maintainability, and testability.

1.2.6 User Documentation

In addition to the requested functionalities of the Course Enrollment System, the website will also include documentation aimed at users who need troubleshooting assistance or guidance in creating a user account. Furthermore, the site will have a chat bot, which allows users to request assistance when needed.

1.2.7 Assumptions and Dependencies

Constraints for the project include a budget of $100,000, a timeline of 6 months, and a project team of no more than eight, which includes designers, architects, database administrators, developers, testers, and project managers.

1.3 External Interface Requirements

1.3.1 User Interfaces

Each page of the Course Enrollment System should have the same header, footer, and navbar for consistency. Icons should all be selected from Font Awesome. Each page should allow a user to return

to the previous page without losing session information. All images should include alt tags, and users should be able to tab through a page. In addition, colors should be selected for usability and visual appeal, and fonts should be easily readable. There should not be ads that detract from the content, and the general page layout should be consistent. The home button should be clearly visible on every page. Furthermore, forms should provide users with clear feedback instructing them how to correct errors when they are made.

1.3.2 Hardware Interfaces

The Course Enrollment System should be accessible from all major devices including iOS and Android phones, iPads and other tablets, and both Mac and Microsoft products. The site will use HTTP to make requests and TCP to transfer resources from the web server to the client. The website should also support screen readers.

1.3.3 Software Interfaces

The website should be compatible with iOS, Android, Mac, and Windows operating systems. The build should also include an API that talks to the relational database in order to store and retrieve user and course data. In addition, the site should allow users to sign on with their school-provided email accounts.

1.3.4 Communications Interfaces

Users should receive email notifications upon registration for a course, or upon a course being dropped. They should also receive notification when a spot opens up for a course that they are currently on a waitlist for. In additional all personally identifiable information (PII) should be secure, and all passwords should be encrypted.

1.4 System Features

1.4.1 New User Can Create an Account

Description and Priority

- Priority: High
- Description: User should be able to create an account. This encompasses part of the basic functionality of the website. A user cannot view or manage their course schedule without creating an account.
- Benefit: 9

Stimulus/Response Sequences

- User gets prompted on the home page to log in or create an account
- User clicks on register
- User enters form data to create account
- User clicks submit
- User receives positive confirmation that account has been created or clear instructions regarding how to correct their input

Functional Requirements

- REQ-1: Website should present user with two buttons on home page (login and register)
- REQ-2: Website should present registration form when user clicks on “register”
- REQ-3: Data from form is validated, sanitized, processed, and sent to the database when a user hits “submit”
- REQ-4: Website presents user with informative messages if input does not meet requirements
- REQ-5: Website provides user with success message when account has been successfully created

1.4.2 Returning User Can Login

Description and Priority

- Priority: High
- Description: Returning user should be able to login. This encompasses part of the basic functionality of the website. A user cannot view or manage their course schedule without logging in.
- Benefit: 9

Stimulus/Response Sequences

- User gets prompted on the home page to log in or create an account
- User clicks on login
- User enters form data to login
- User clicks submit
- User receives positive confirmation that they have been logged in or clear instructions regarding how to correct their input

Functional Requirements

- REQ-1: Website should present user with two buttons on home page (login and register)
- REQ-2: Website should present login form when user clicks on “login”
- REQ-3: Data from form is validated, sanitized, processed, and sent to the database when a user hits “submit”
- REQ-4: Website presents user with informative messages if input does not meet requirements
- REQ-5: User credentials are verified against user information stored in the database

- REQ-6: Website provides user with success message when account has been successfully created

1.4.3 User Can Search for Available Courses

Description and Priority

- Priority: High
- Description: User should be able to search for available courses within a semester. This encompasses part of the basic functionality of the website. A user cannot register for a course without first being able to search for available courses.
- Benefit: 8

Stimulus/Response Sequences

- User clicks on “register for courses” link on “view course schedule” page
- User selects semester from drop down menu and hits submit
- User is presented with a list of search results

Functional Requirements

- REQ-1: Website should have a “register for courses” link on the “view course schedule” page
- REQ-2: Website should present user with drop down containing options for semesters
- REQ-3: Website should return search results when user hits submit
- REQ-4: Website should update search results when a new drop-down option is selected

1.4.4 User Can View Course Schedule

Description and Priority

- Priority: High

- Description: User should be able to view their course schedule. This encompasses part of the basic functionality of the website. A user cannot drop a course without being able to view their course schedule.
- Benefit: 9

Stimulus/Response Sequences

- User clicks on “view course schedule” in navigation bar
- User views course schedule

Functional Requirements

- REQ-1: Navigation bar has a link to the “view course schedule” page
- REQ-2: When link is clicked, user is redirected to page that presents their course schedule

1.4.5 User Can Register for a Course

Description and Priority

- Priority: High
- Description: User should be able to register for a new course. This encompasses part of the basic functionality of the website. A user cannot manage their course schedule if they cannot register for a course.
- Benefit: 9

Stimulus/Response Sequences

- User clicks on “register for courses” link on “view course schedule” page
- User selects semester from drop down menu and hits submit
- User is presented with a list of search results

- User checks the “register” checkbox next to all desired courses
- User hits submit button
- If there is availability in the course, user receives confirmation that they have successfully registered for the course
- If there is not availability in the course, user receives notification that they have been added to the waitlist for the course
- When the user confirms the confirmation message, they are redirected to the “view course schedule” page

Functional Requirements

- REQ-1: Website should have a “register for courses” link on the “view course schedule” page
- REQ-2: Website should present user with drop down containing options for semesters
- REQ-3: Website should return search results when user hits submit
- REQ-4: Website should offer checkboxes next to courses under “register” header
- REQ-5: Website should have submit button
- REQ-6: Website should collect data from table when submit button is clicked
- REQ-7: Website should display confirmation modal that user is either successfully registered, or that they have been added to the waitlist
- REQ-8: Website should redirect user to “view course schedule” page after they confirm the modal

1.4.6 User Can Drop a Course

Description and Priority

- Priority: High

- Description: User should be able to drop a course. This encompasses part of the basic functionality of the website. A user cannot manage their course schedule if they cannot drop a course.
- Benefit: 9

Stimulus/Response Sequences

- User clicks on “view course schedule” in navigation bar
- User views course schedule
- User checks the “drop” checkbox next to all desired courses
- User hits submit button
- User receives confirmation that they have successfully dropped the course
- When the user confirms the confirmation message, they are redirected to the “view course schedule” page

Functional Requirements

- REQ-1: Navigation bar has a link to the “view course schedule” page
- REQ-2: When link is clicked, user is redirected to page that presents their course schedule
- REQ-3: Website should offer checkboxes next to courses under “drop” header
- REQ-4: Website should have submit button
- REQ-5: Website should collect data from table when submit button is clicked
- REQ-6: Website should display confirmation modal that user successfully dropped courses
- REQ-7: Website should redirect user to “view course schedule” page after they confirm the modal

1.4.7 User Receives Notification When Course Is Available

Description and Priority

- Priority: Medium
- Description: User should be able to receive a notification when a course they are on the waitlist for becomes available.
- Benefit: 7

Stimulus/Response Sequences

- User logs into their account
- User sees a notification at the top of their account that says they have been moved from the waitlist of a course to registered
- User acknowledges the notification

Functional Requirements

- REQ-1: Website should allow a user to log in
- REQ-2: Website should present a notification at the top of the page once a user is logged in notifying them that they have been registered for a course that they were on the waitlist for

1.4.8 User Can Logout

Description and Priority

- Priority: High
- Description: User should be able to logout. This encompasses part of the basic functionality of the website. A user's information is not protected if they cannot logout of the course enrollment system.

- Benefit: 9

Stimulus/Response Sequences

- User clicks on logout in the navigation bar
- User is logged out of their account and directed back to the website home page

Functional Requirements

- REQ-1: Website should have “logout” link in the navigation bar
- REQ-2: Website should log a user out when the logout link is clicked
- REQ-3: Website should redirect the user to the home page once they are logged out

1.5 Other Nonfunctional Requirements

1.5.1 Performance Requirements

New users should be able to create an account in less than 60 seconds. When a user searches for a course, they should receive a list of search results in less than 3 seconds. Once a user hits the “submit” button to register for or drop a course, they should receive a confirmation message in under 3 seconds. Speed is of utmost importance, as delays may cause a poor user experience and may deter users from returning to the site in the future, causing more overhead in the Registrar’s Office for course enrollment.

1.5.2 Safety Requirements

All PII should be protected, and passwords should be encrypted. The system needs to follow FERPA guidelines. In addition, sessions should timeout after 5 minutes without activity, unless the user confirms that they want to continue the session.

1.5.3 Security Requirements

All PII should be protected and secure. Credentials should be verified before a user is given access to account information. In addition, passwords should be encrypted.

1.5.4 Software Quality Attributes

The Course Enrollment System should follow web accessibility guidelines outlined by ADA (n.d.). In addition, the site should be easy to use and navigate. The website should be reliable, as interruptions in service may deter users from returning. The site should be scalable to accommodate for increased enrollment. It is important that the site be modifiable, to allow for the addition of new features, and the product should be maintainable and testable, as maintenance will be the responsibility of the customer.

1.5.5 Business Rules

Users should not be allowed to view their schedule or register for/drop courses unless they are logged in. Administrators should have the ability to assist users in resetting credentials. They should also be able to search for student information and schedules using the student ID number. Finally, administrators should be able to retrieve a list of all students, enrollment, dropped courses, and course availabilities, for reporting purposes.

2. Testing Levels and UML Models

2.1 Introduction

It is imperative to test software throughout the development lifecycle in order to ensure a quality product. When performed effectively, testing can save costs, reduce development time, improve maintainability, improve code quality, identify and correct faults early in the process, increase robustness, and ensure that the customer is receiving a quality product that meets their needs and desires. The four levels of testing are component testing, integration testing, system testing, and acceptance testing (Spillner, Linz, & Schaefer, 2014). Component testing breaks software down into functional units, and tests whether or not these units meet the required specifications. Integration testing occurs when groups of related components are tested together to identify faults in the interfaces and interactions between components (GeeksForGeeks, n.d.). System testing “specifically focuses on

testing the functional and non-functional aspects of the software in more comprehensive manner including security, usability, performance and compatibility" (Suffian et.al., 2016). Acceptance testing ensures that the software meets the customer's needs, and works in a production-level environment.

2.2 Component Testing

In regards to the Course Enrollment System, component testing will occur for each of the different functionalities illustrated in the use case diagram in Figure 1. For example, the software will have components that carry out each of the different functions: displaying a student's schedule, dynamically generating the dropdowns that allow a student to search for courses, generating the search results based off of form input, adding a course to a student's schedule, dropping a course from a student's schedule, and adding a student to the waitlist for a course. All classes and functions should be thoroughly tested independently from one another. Component-level testing will also ensure that the code is maintainable, effective, robust, performant, and lacks faults.

2.3 Integration Testing

The Course Enrollment System will also need to undergo integration testing. Integration testing allows testers and developers the ability to "discover faults and bugs in the interaction between integrated components" (Ali et.al., 2018). Once all of the components mentioned above are thoroughly tested on their own, test cases will be generated to evaluate the collaboration of components. For example, Figure 4 illustrates the classes that are necessary in a sequence of events that allow a student to register for a course. While each of these classes might have passed testing at the component level, the real test is whether or not the control flow and data flow is successful across the interactions between components.

2.4 System Testing

After integration tests are complete, the Course Enrollment System will need to go through a series of system tests, which assess whether or not the website as a whole meets functional and non-functional

requirements. For example, does the website possess all of the functionalities described in the SRS? Is the Course Enrollment System website fast, reliable, available, secure, and maintainable? The testers and developers at this point need to view the system from a user perspective, and test the system in an environment that is as close to production-level as possible.

2.5 Acceptance Testing

Finally, the Course Enrollment System will go through acceptance testing. This process will involve students to ensure that the system provides them with all of the functionality they need to quickly search, register for, and drop classes. Administrators will also need to provide feedback regarding whether or not they can quickly and easily generate reports, and update course offerings. Acceptance testing will take place across different operating systems and devices to ensure that all users will have a positive user experience. Performing the four levels of testing (component, integration, system, and acceptance) will support the development team in delivering a high-quality product as efficiently and quickly as possible.

2.6 UML Models

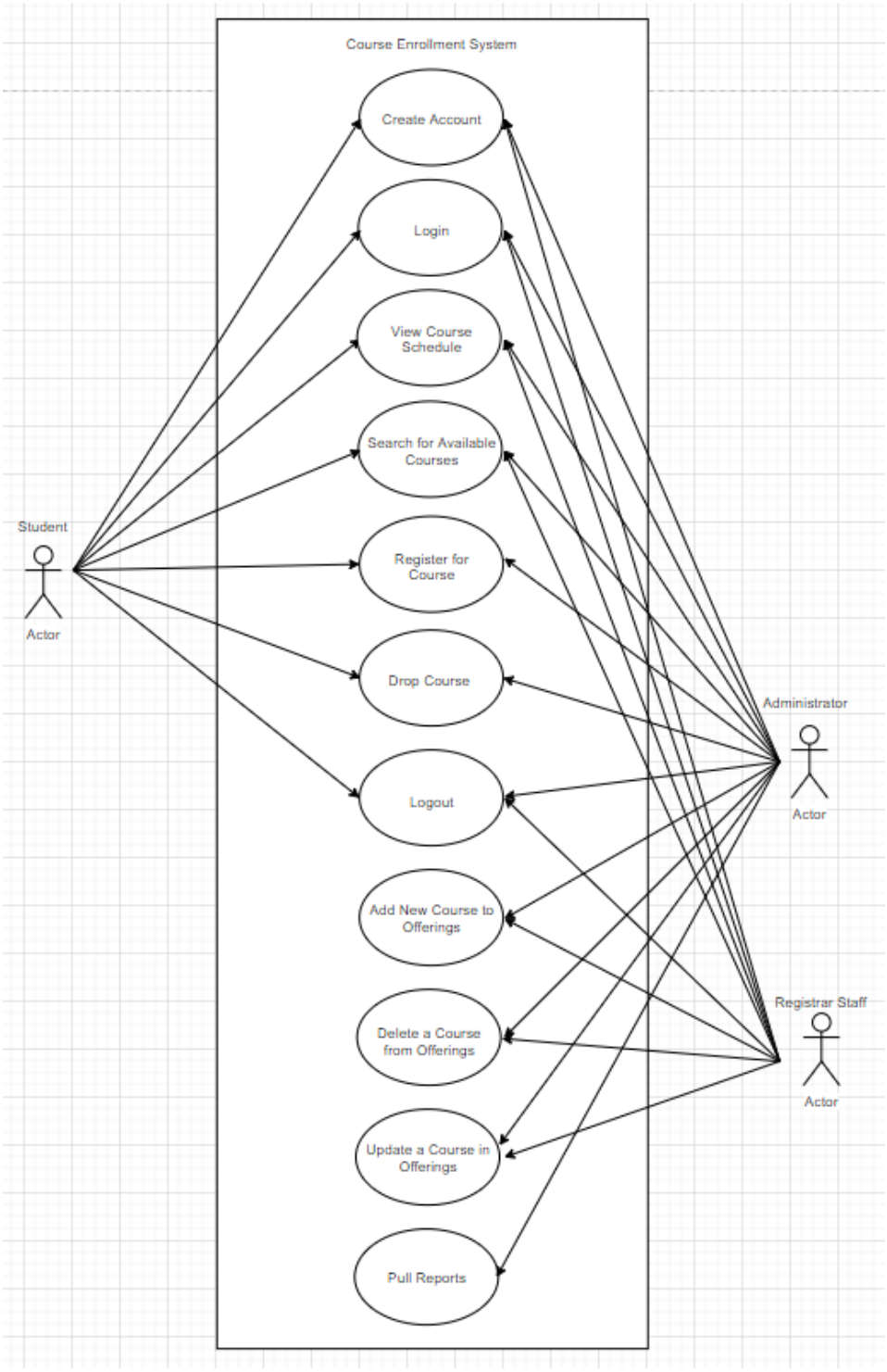


Figure 1. Use Case Diagram for Course Enrollment System

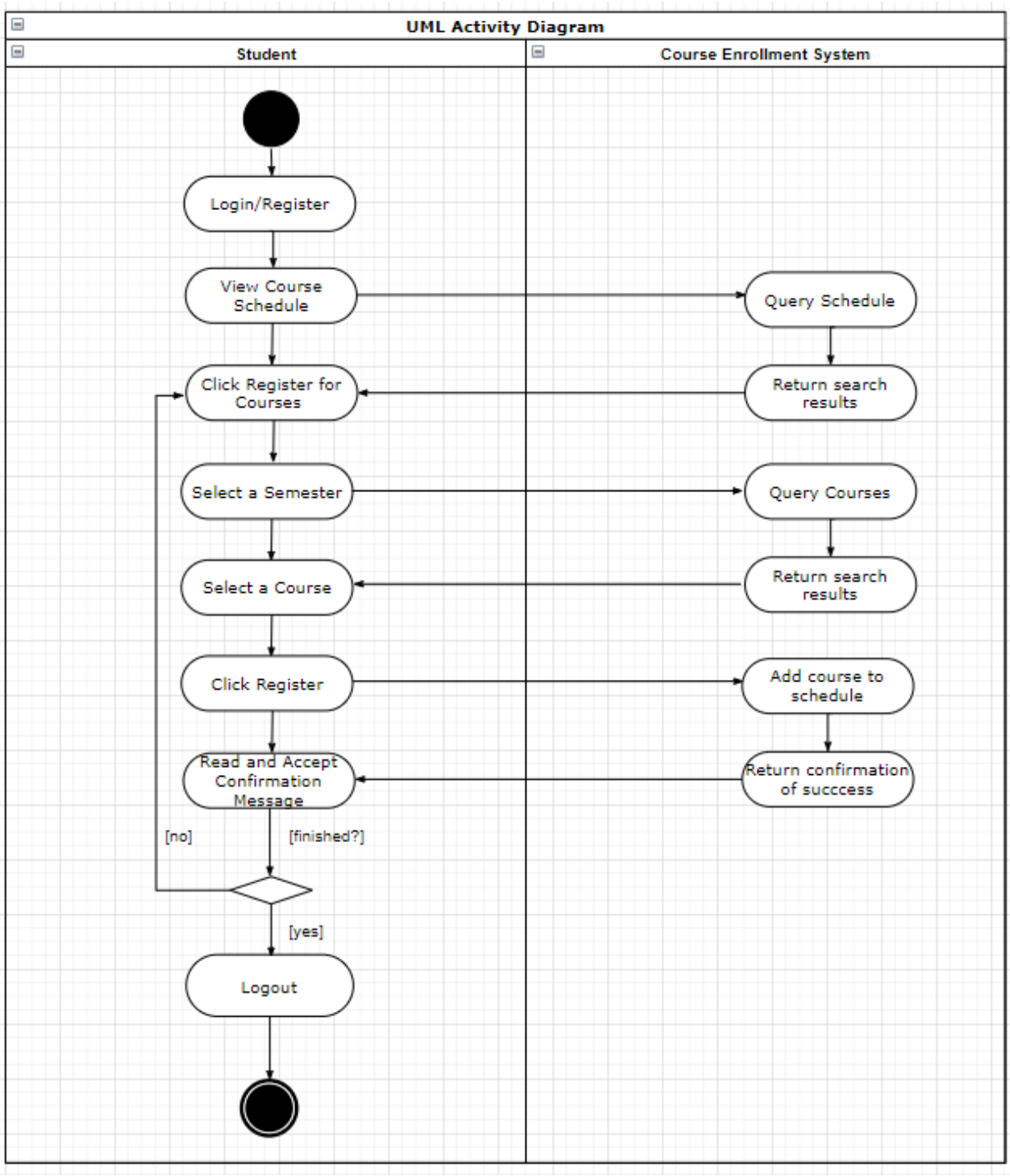


Figure 2. Activity Diagram for Course Enrollment System

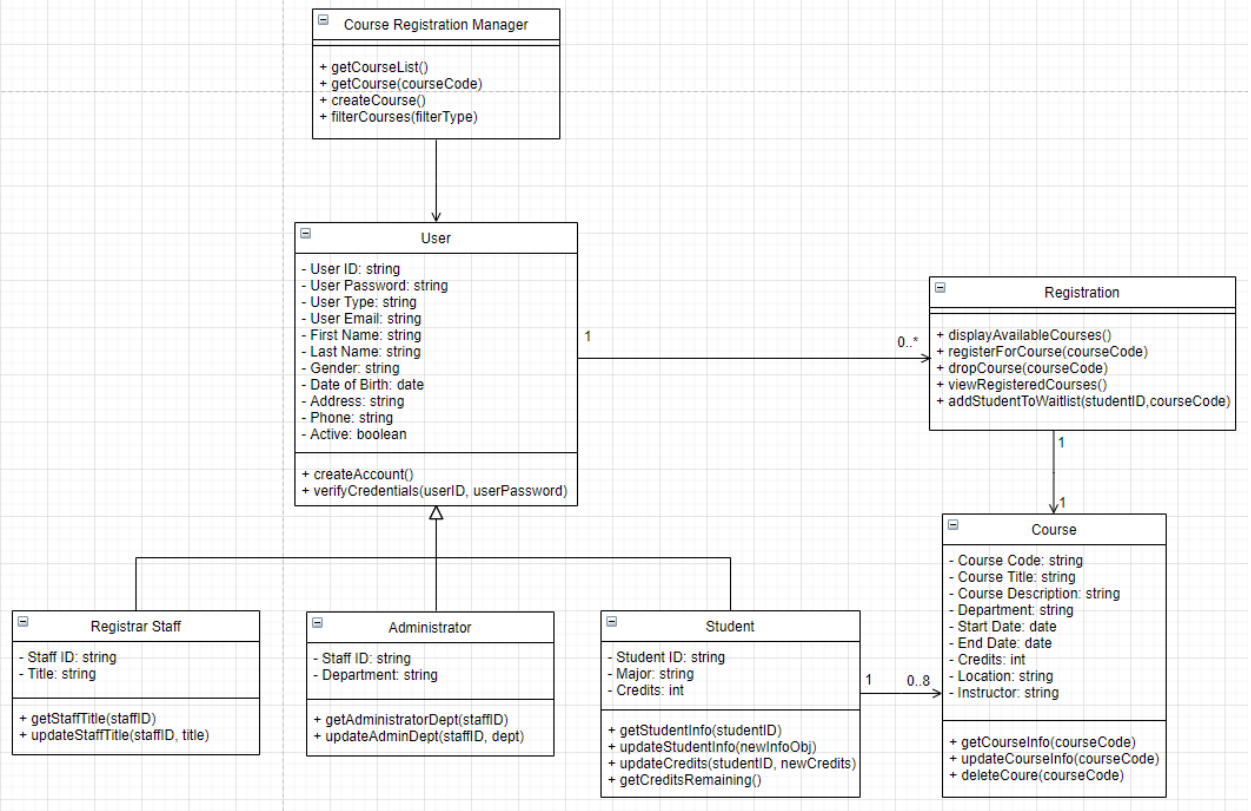


Figure 3. Class Diagram for Course Enrollment System

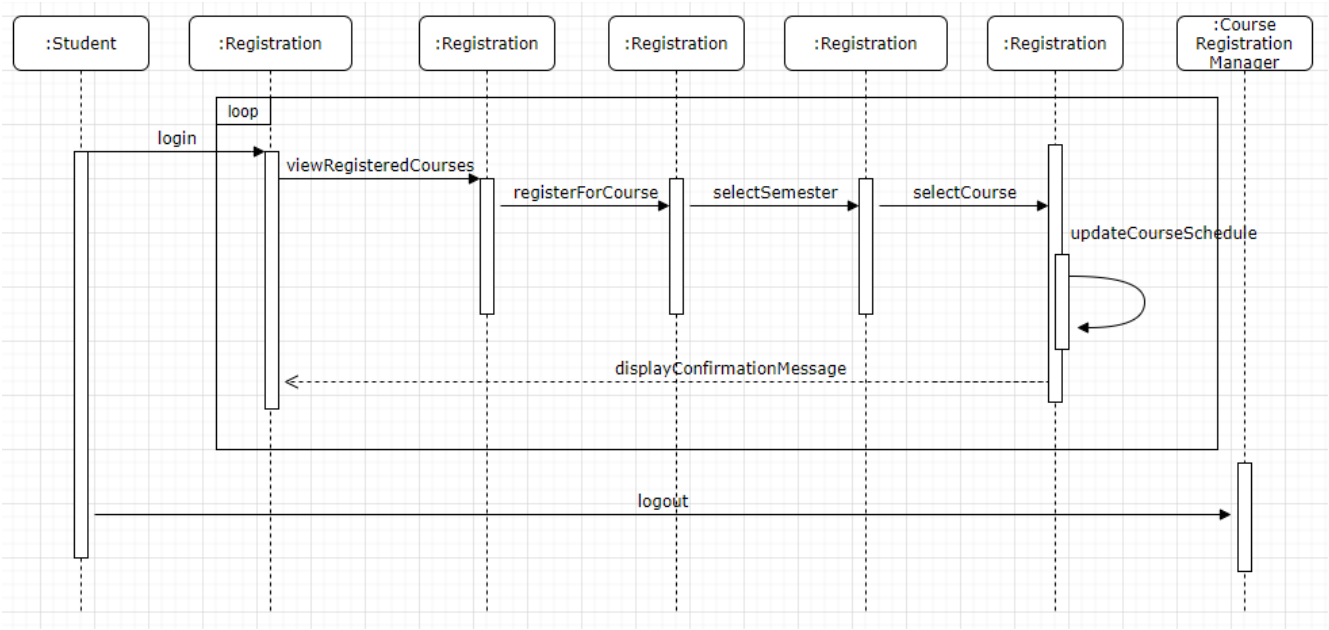


Figure 4. Sequence Diagram for Course Enrollment System

3. Test Strategies

3.1 Static Testing

3.1.1 Tools

The following tools will be used for static testing of the Course Enrollment System: an IDE or text editor, a compiler, and a linter. These tools will allow the developers and testers to identify syntax violations, cross-reference program elements, check data types, detect undeclared variables, detect dead code, and check interface consistency (Spillner, Linz, & Schaefer, 2014). Aside from detecting errors, the IDE and linter will also ensure that code is more readable and consistent since there will be multiple developers collaborating on the Course Enrollment System. The IDE and compiler will also assist developers in debugging.

3.1.2 Compliance to Conventions and Standards

According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Compliance to conventions and standards can also be checked with tools. For example, tools can be used to check if a program follows programming regulations and standards." Rules should be implemented within the IDE and linter mentioned above in order to enforce proper coding conventions and standards. These rules should be based on a style guide. For example, since the Course Enrollment System uses PHP, programmers and testers can utilize the MIT “PHP Code Style Guide” (Koschara, 2013) to ensure consistency and compliance.

3.1.3 Data Flow Analysis

Data flow analysis should be used to reveal defects in code for the Course Enrollment System. This type of analysis "checks the usage of every single variable" (Spillner, Linz, & Schaefer, 2014). This includes variables that are defined, referenced, and undefined. The data flow analysis will be used to reveal data

flow anomalies. "An anomaly is an inconsistency that can lead to failure but does not necessarily do so. An anomaly may be flagged as a risk" (Spillner, Linz, & Schaefer, 2014).

3.1.4 Control Flow Analysis

Control flow analysis will also be used to find defects. Control flow utilizes graphs to represent a sequence of statements to analyze program execution. These diagrams are used to visualize changes in the execution of the program (the control flow) caused by logic such as conditionals and loops. Examples of control flow graphs can be seen in Figure 1 below (GeeksForGeeks, n.d.). According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Due to the clarity of the control flow graph, the sequences through the program can easily be understood and possible anomalies can be detected."

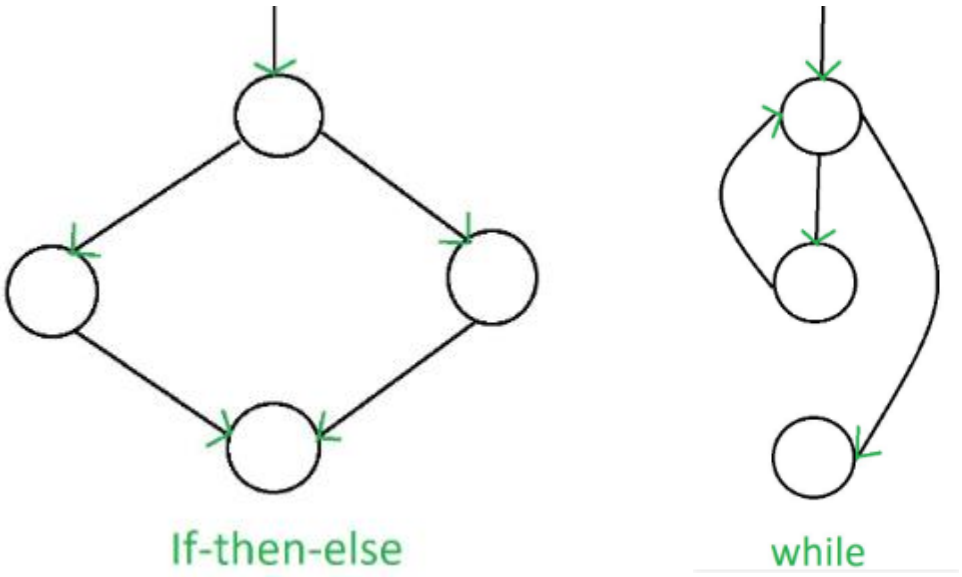


Figure 7. Example Control Flow Graphs (GeeksForGeeks, n.d.)

3.2 Dynamic Testing

3.2.1 Black Box Testing

Black box testing includes a group of testing techniques that aim to verify that a test object meets its specifications and that expected outputs result from a specified set of inputs. According to *Software*

testing foundations: A study guide for the certified tester exam (4th ed.) (2014), the goal of these strategies is "the verification of the functionality of the test object. It is indisputable that the highest priority is that the software work correctly. Thus, black box techniques should always be applied." Therefore, it is imperative that test strategies for the Course Enrollment System include black box techniques. The following methods will be used for testing the Course Enrollment System.

3.2.2 Equivalence Class Partitioning

This approach helps developers and testers systematically generate test cases based on domains for various inputs. "An equivalence class is a set of data values that the tester assumes are processed in the same way by the test object" (Spillner, Linz, & Schaefer, 2014). In other words, this strategy involves identifying the parameters that a function accepts, generating potential valid and invalid inputs for that parameter, creating an equivalence class that represents a group of similar inputs, selecting a representative input for each equivalence class, and then generating test cases that eliminate redundancy as much as possible (Spillner, Linz, & Schaefer, 2014).

3.2.3 Boundary Value Analysis

Boundary value analysis will be used in conjunction with equivalence class partitioning in order to generate test cases for the Course Enrollment System. This strategy focuses on testing inputs that lie on the boundaries between equivalence classes. "Boundary value analysis delivers a very reasonable addition to the test cases that have been identified by equivalence class partitioning. Faults often appear at the boundaries of equivalence classes. This happens because boundaries are often not defined clearly or are misunderstood" (Spillner, Linz, & Schaefer, 2014).

3.2.4. State Transition Testing

This testing method is beneficial in evaluating how a system reacts to transitions in state. For example, the Course Enrollment System involves a number of objects. These objects change and react to user

input. For example, Registration class begins in an empty state. When a student searches for courses and then selects a course that they would like to register for, the state of the object changes—it is no longer empty. This testing strategy will be applied to see how events related to a student searching, selecting, and adding a course to their schedule trigger changes in the state of different objects.

3.2.5 Use-Case-Based Testing

Use-case-based testing will be important in evaluating the functionality of the Course Enrollment System. This strategy involves testing the execution of paths aligned with each of the use cases illustrated in the use case diagram for the Course Enrollment System (in the design documents). For example, one of the use cases illustrates that a student should be added to the waitlist for a course if they try to register for a course that is already full. This means that at least one test case needs to be generated to evaluate whether or not the website implements that functionality correctly.

3.2.6 White Box Testing

White box testing is more granular than black box testing and directly involves evaluating the source code. As stated by *Analysis of statement branch and loop coverage in software testing with genetic algorithm* (2017), "White box testing involves executing a program and seeing which parts of it are executed." It is important to include these methods in analysis of the Course Enrollment System, since these tests are designed to evaluate branching logic, conditionals, and loops. This ensures that all paths of execution are evaluated in the test environment, so that the first time a line of code is being executed is not in production. The following white box techniques will be used to test the Course Enrollment System.

3.2.7 Statement Testing

Statement testing refers to testing each statement within a test object. The goal is 100% execution of the code. Statement testing can help identify dead statements (where code exists that cannot be

reached by any test case). It is important to track which lines of code have been executed to reduce redundancy and make testing more efficient (Spillner, Linz, & Schaefer, 2014).

3.2.8 Decision/Branch Testing

Decision/branch testing focuses on executing decisions in the code, rather than individual statements. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "In contrast to statement coverage, branch coverage makes it possible to detect missing statements in empty branches." As a result of this, "Decision/branch coverage usually requires the execution of more test cases than statement coverage" (Spillner, Linz, & Schaefer, 2014). This leads to more reliable code. Examples of decisions include IF statements, case statements, and loops. Control flow graphs can also be used in this type of testing to ensure that all test cases are considered.

3.2.9 Testing of Conditions

Condition testing evaluates more complex decisions influenced by multiple conditions in the code. This technique is important so that the results of all complex decisions are considered. Spillner, Linz, & Schaefer (2014) state, "Combinations of logical expressions are especially defect prone. Thus, a comprehensive test is very important."

3.2.10 Experience-Based Testing

Experience-based testing, also known as intuitive-based testing, will be used to supplement black box and white box techniques in the testing of the Course Enrollment System. Intuitive-based tests leverage the skills and experience of developers and testers to generate test cases that might otherwise be overlooked in systematic testing. In this approach, "The test cases are based on where faults have occurred in the past or the tester's ideas of where faults might occur in the future" (Spillner, Linz, & Schaefer, 2014).

4. Test Management Strategy

4.1 Test Teams

In order to achieve efficient development of the Course Enrollment System it is important to consider test teams and an approach to each level of testing. If at all possible, developers on the team should not test their own code. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "because there is a tendency to be blind to our own errors, it is much more efficient to let different people perform testing and development and to organize testing as independently as possible from development." For this reason, designated testers on the development team should perform component testing. In addition, a designated testing team within the project should perform integration testing. These testers may include individuals from the business or IT. System testing should also be performed by a designated team that includes specialists: "Especially in system testing, it is often necessary to extend the test team by adding IT specialists, at least temporarily, to perform work for the test team" (Spillner, Linz, & Schaefer, 2014). This will allow the system to be viewed and tested from multiple perspectives.

4.2 Test Roles

The following test roles should be assigned to individuals in order to conduct testing of the Course Enrollment System: test manager, test designer, test automator, test administrator, and tester (Spillner, Linz, & Schaefer, 2014). The test manager will require experience in software testing, quality management, project management, and personnel management. The test designer should have a skillset that includes test methods and specification, testing, and software engineering. They should also hold a degree in Computer Science. The test automator will require experience in testing, programming, scripting, test tools, and automation. The test administrator needs a skillset involving setting up and supporting test environments, as well as system administration and networking. The

tester should have experience following procedures, executing tests, reporting failures, and using test objects and testing tools (Spillner, Linz, & Schaefer, 2014).

4.3 Exit Criteria

Each test case should contain specific exit criteria in order to guide testers in determining when a test is considered to be complete. Exit criteria are important because "They prevent tests from ending too early, for example, because of time pressure or because of resource shortages" (Spillner, Linz, & Schaefer, 2014). After test cases are prioritized and classified based in risk, three metrics will be used to quantify when testing of the Course Enrollment System is complete. The first is based on graphing the number of defects discovered over time. The rate of defect finding should be high when testing first begins, and should taper off as the majority of defects are identified and managed. Testing will only stop once the problem find rate reaches 1 defect/1000 hours for high risk test cases, 1 defect/500 hours for medium risk test cases, and 1 defect/100 hours for low risk test cases. The second method that will be used is defect seeding and estimation. Testing of the Course Enrollment System will not stop until 100% of defects have been found in high risk test cases, 80% of defects have been discovered in medium risk test cases, and 70% of defects have been found in low risk test cases. For example, if a third-party seeds ten defects in the software amongst the others that were naturally occurring, then testing should stop when eight out of those ten defects have been found for medium risk test cases (Tsui, Karam, & Bernal, 2018). Finally, testing for the Course Enrollment System should only stop once 100% line coverage has been accomplished.

4.4 Test Estimated Effort

It is important to consider both the costs of testing, as well as the costs of undetected defects, when planning software testing. It is the test manager's responsibility to initiate test effort estimation during the planning phase to ensure that resources are assigned and distributed properly. Estimating test

effort for the Course Enrollment System will be conducted by basing estimates on data from former or similar projects, as this method provides the most reliable test effort estimates: “task-driven test effort estimation tends to underestimate the testing effort. Estimating based on experience data of similar projects or typical values usually leads to better results” (Spillner, Linz, & Schaefer, 2014).

4.5 Test and Risk

Risk-based prioritization will be implemented in order to ensure that the most significant defects within the Course Enrollment System are revealed as early as possible. This will prevent critical defects from having downstream affects while reducing the time and cost of handling critical defects that make it to production. While test cases can be prioritized based on multiple factors, including frequency of function use, visibility, priority of functional and non-functional requirements from the customer, severity, risk, and complexity, prioritization by risk is one of the best methods for selecting test cases. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), “Risk based prioritization of the tests ensures that risky product parts are tested more intensively and earlier than parts with lower risk. Severe problems (causing much corrective work or serious delays) are found as early as possible.”

4.6 Incident Reporting

Incident reporting will be used throughout the development and testing of the Course Enrollment System in order to document and manage incidents. Every defect that is determined to be significant and legitimate (not the result of a poorly designed test) should be documented. The template shown in Figure 5 (Spillner, Linz, & Schaefer, 2014) below will be used to document incidents. This will allow the communication of incidents to be consistent, and developers will be able to easily reproduce defects. Homès (2012) stresses the importance of clearly communicating defects: “Determining the impact in an understandable way for developers (data loss, functionality loss, software instability, etc.) and for

customers and the hierarchy (impacts in financial terms or in usability terms, noncompliance to requirements, etc.) enables a quick recognition of the anomaly."

The incident report will contain the following information: the tested software, test environment, tester's name, the class that contains the defect, prioritization of the defect, and information relevant to reproducing and locating the defect (Spillner, Linz, & Schaefer, 2014). Testers, developers, customers and users can report incidents. When corrections are made, the incident report should be updated in the database. This will help the project team track and manage incidents.

	Attribute	Meaning
Identification	Id / Number	Unique identifier/number for each report
	Test object	Identifier or name of the test object
	Version	Identification of the exact version of the test object
	Platform	Identification of the HW/SW platform or the test environment where the problem occurs
	Reporting person	Identification of the reporting tester (possibly with test level)
	Responsible de-veloper	Name of the developer or the team responsible for the test object
	Reporting date	Date and possibly time when the problem was observed
Classification	Status	The current state (and complete history) of processing for the report (section 6.6.4)
	Severity	Classification of the severity of the problem (section 6.6.3)
	Priority	Classification of the priority of correction (section 6.6.3)
	Requirement	Pointer to the (customer-) requirements which are not fulfilled due to the problem
	Problem source	The project phase, where the defect was introduced (analysis, design, programming); useful for planning process improvement measures
Problem description	Test case	Description of the test case (name, number) or the steps necessary to reproduce the problem
	Problem description	Description of the problem or failure that occurred; expected vs. actual observed results or behavior
	Comments	List of comments on the report from developers and other staff involved
	Defect correction	Description of the changes made to correct the defect
	References	Reference to other related reports

Figure 5. Incident Reporting Template (Spillner, Linz, & Schaefer, 2014)

4.7 Defect Classification

Defect classification will be implemented for prioritizing incidents related to the Course Enrollment System. The severity level will reflect the level of impairment caused by the defect. The following 5 levels of severity will be used: 1-Fatal, 2-Very Serious, 3-Serious, 4-Moderate, and 5-Mild. The criteria for these levels can be seen in Figure 6 (Spillner, Linz, & Schaefer, 2014) below. Levels of priority will also be implemented in order to identify how quickly a problem should be addressed. The following four levels of priority will be used: 1-Immediate, 2-Next Release, 3-On Occasion, and 4-Open. The criteria assigned to each of these priority levels can be seen in Figure 7 (Spillner, Linz, & Schaefer, 2014) below.

Class	Description
1 – FATAL	System crash, possibly with loss of data. The test object cannot be released in this form.
2 – VERY SERIOUS	Essential malfunctioning; requirements not adhered to or incorrectly implemented; substantial impairment to many stakeholders. The test object can only be used with severe restrictions (difficult or expensive workaround).
3 – SERIOUS	Functional deviation or restriction ("normal" failure); requirement incorrectly or only partially implemented; substantial impairment to some stakeholders. The test object can be used with restrictions.
4 – MODERATE	Minor deviation; modest impairment to few stakeholders. System can be used without restrictions.
5 – MILD	Mild impairment to few stakeholders; system can be used without restrictions. For example, spelling errors or wrong screen layout.

Figure 6. Severity Levels (Spillner, Linz, & Schaefer, 2014)

Priority	Description
1 – IMMEDIATE	The user's business or working process is blocked or the running tests cannot be continued. The problem requires immediate, or if necessary, provisional repair (→"patch").
2 – NEXT RELEASE	The correction will be implemented in the next regular product release or with the delivery of the next (internal) test object version.
3 – ON OCCASION	The correction will take place when the affected system parts are due for a revision anyway.
4 – OPEN	Correction planning has not taken place yet.

Figure 7. Priority Levels (Spillner, Linz, & Schaefer, 2014)

4.8 Configuration Management

Configuration management will be used throughout the development of the Course Enrollment System in order to track the version history of the project, as well as to allow multiple developers to make contributions without interfering with one another's work. Poor configuration management can lead to a number of avoidable problems such as developers overwriting one another's code, the inability to integrate components because of unknown versions, and difficulty testing because changes to a component are untraceable or testers don't know which test cases belong to which version of a test object (Spillner, Linz, & Schaefer, 2014). These circumstances can be avoided by implementing version management, configuration identification, incident and change status control, and configuration audits. All of these activities can be achieved through the use of configuration management tools. *System for Efficient Storage and Version Control of Arbitrary File Collections* (2020) discusses some popular version control options: "Modern file version control systems, such as git, mercurial, or svn implement a concept of revision or commit. This concept may be crucial to store program source code, as it allows users to view previous versions of stored files."

5. Test Tools

5.1 Introduction

Test tools can ease the burden of manual software testing. Because the Course Enrollment System is a new build, test tools should be acquired and utilized to increase test efficiency and reliability, reduce manual testing, and achieve load and performance testing (Spillner, Linz, & Schaefer, 2014). Before selecting test tools, a cost-benefit analysis should be conducted. Tools for testing the Course Enrollment System may include those for test management and control, test specification, static testing, dynamic testing, and non-functional testing.

5.2 Test Management and Control Tools

Tools that support test management and control allow testers and project managers to document, prioritize, list, and maintain test cases (Spillner, Linz, & Schaefer, 2014). A test management tool that may be considered for development of the Course Enrollment System is Jira. Test management tools can be used to ensure that there are test cases to address each software requirement. A test tool with this functionality is considered a requirements management tool. Test execution tools are another form of test management tool that execute test scripts automatically and document the results. Incident management tools track software defects and their resolutions. Configuration management tools track versions and builds that will be tested. Lastly, tool integration can be used to combine multiple test management tools into one (Spillner, Linz, & Schaefer, 2014). A cost-benefit analysis specific to the Course Enrollment System should be used to determine which tools will be the most beneficial to the project.

5.3 Test Specification Tools

Tools for the specification phase of testing should also be considered for the Course Enrollment System. These tools help test designers generate test data. There are four types of test generators: database-

based test generators, code-based test generators, interface-based test generators, and specification-based test generators. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), database-based test generators "process database schemas and are able to produce test databases from these schemas." These tools help generate test data by filtering through databases. An example of such a tool is DatProf, which supports Windows operating systems and generates synthetic data using most common database technologies (Software Testing Help, 2020). Code-based test generators use the source code to generate test data. While these tools can be helpful, they cannot detect faults caused by missing code. In addition, Spillner, Linz, & Schaefer (2014) state, "The use of code as a test basis for testing the code itself is in general a very poor foundation." However, code-based test generators can be helpful for regression testing. Interface-based test generators identify parameter domains, which can then be used for equivalence class partitioning and boundary value analysis to generate test cases (Spillner, Linz, & Schaefer, 2014). They are also helpful when testing API's and GUI's. Specification-based test generators synthesize test data from formal specifications or models.

5.4 Static Testing Tools

Static testing tools should be applied directly to the source code of the Course Enrollment System. Because they support the evaluation of source code, they can help identify defects early in the development process, before code is even executed. Review tools are a form of static testing that serve like checklists to help testers plan, execute, and evaluate code reviews (Spillner, Linz, & Schaefer, 2014). Static analyzers can also be used to identify areas in the code that are complex, risky, inconsistent, defect-prone, violate programming standards, or pose portability issues. These areas can then be refactored, and or prioritized for further testing. Model checkers may also be considered in order to analyze the UML models to check for "missing states, state transitions, and other inconsistencies in a model" (Spillner, Linz, & Schaefer, 2014). An example of a static analysis tool is Polyspace. This tool,

released in 2014 consists of two components: Polyspace Code Prover and Polyspace Bug Finder. "The former identifies every code instruction by applying all possible values of variables, while the later is used to perform static analysis" (Khalid, 2017).

5.5 Dynamic Testing Tools

Utilizing dynamic testing tools during development of the Course Enrollment System will help reduce the mechanical work involved with manual test execution. These tools provide the test object with input data, and record the output. Examples of dynamic testing tools are debuggers, test drivers and test frameworks, simulators, test robots, comparators, dynamic analyzers, and tools for coverage analysis (Spillner, Linz, & Schaefer, 2014). Debuggers can be found in popular text editors and IDE's. They allow testers and developers to execute a program one line at a time. A program can also be paused mid-execution to observe or manipulate variables. Test drivers and test frameworks are often used for component and integration tests, and are designed for specific programming languages and environments (Spillner, Linz, & Schaefer, 2014). They provide control over the test object. Simulators mirror the real system environment, but are used when the tests cannot be executed there. Test robots log manual inputs such as keyboard strokes or mouse clicks, store them as a script, and then can execute the script in order to replay that sequence of inputs in an automated fashion. These tools are especially helpful for regression testing. Comparators identify deviations from expected results. Dynamic analyzers evaluate memory usage and allocation, as well as identify "memory leaks, wrong pointer allocation, or pointer arithmetic problems" (Spillner, Linz, & Schaefer, 2014). Finally, coverage analysis tools track statement and branch coverage to ensure that as much code as possible is executed during testing. An example of a dynamic analysis tool is VectorCast. "VectorCAST ICover is used to test code coverage of source code through dynamic analysis" (Khalid, 2017).

5.6 Non-Functional Testing Tools

Non-functional testing tools should also be considered for assessing the quality attributes of the Course Enrollment System. Often, these attributes can be difficult to evaluate using manual methods alone. According to *Software testing foundations: A study guide for the certified tester exam (4th ed.)* (2014), "Load test tools generate a synthetic load (i.e., parallel data-base queries, user transactions, or network traffic." This allows testers to determine how a system will behave under more realistic conditions. Performance tests can also reveal the response time of a system. Load and performance tools are especially helpful when a system expects to experience high traffic, or when it needs to handle large numbers of parallel requests (Spillner, Linz, & Schaefer, 2014). They can also be helpful in identifying bottlenecks within the system. An example of a tool used for performance and load testing is Apache JMeter (Khandelwal, 2019). Aside from performance and load testing tools, other non-functional testing tools include those for testing security and those for assessing data quality. It is important to identify security vulnerabilities early on in development. As *Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools* (2018) states, "Developers cannot afford to believe that their security requirements during development are perfect and impenetrable, no matter how thorough their precautions might be." In addition to testing for security vulnerabilities, data quality assessment tools can ensure that data both before and after migration or conversion is correct and complete. Through the use of these testing tools, developers, testers, and project managers can produce an Course Enrollment System that is robust and reliable.

6. Implementation of Landing, Login, and Account Registration Pages

6.1 How to Run a PHP File in XAMPP

Once XAMPP has been installed (Mikoluk, 2013), you need to open the XAMPP Control Panel. When the XAMPP Control Panel is open, click "Start" next to the Apache module. This starts the Apache Web Server, which will serve the course enrollment system. Next, open a new browser tab or window, and

type in “http://localhost/home.php”. This will load the home page for the course enrollment system. If you receive an error message such as “Object Not Found”, the Apache Web Server may not be configured to serve the file from the correct file path (in other words, it is looking in the wrong directory for the file). If this is the case, open the XAMPP Control Panel, and click on “Config” next to the Apache Module. Click on “Apache (httpd.conf)”. Scroll down to DocumentRoot "C:/xampp/htdocs/", and <Directory "C:/xampp/htdocs/ ">, and change those file paths to the directory where the home.php file is actually located. Then save the configuration file and restart the Apache Web Server. You will also need to click start next to MySQL to connect to the MySQL database. Once Apache and MySQL are started, you should be able to run the website from your local machine.

6.2 Creating the MySQL Database Connection Class

The database connection for the course_enrollment_system was created using the procedural API method discussed in *Fundamentals of web programming (2nd ed.)* (2018). In this method, function calls are used to interact with the database. Therefore, within the Connect class and inside of the constructor function, a new mysql object was instantiated, and the credentials for connecting to the course_enrollment_system database (DBHOST, DBUSER, DBPASS, DBNAME) were passed in. A separate config.php file was created to store the database credentials. This is because the source code is actively being pushed to a remote repo in GitHub, but credentials for the database connection should be ignored. Therefore, a .gitignore file was created, which references the config.php file so that it is ignored when changes are committed and pushed to the remote repo. In addition to the constructor function, the Connect class also contains an executeQuery function that allows queries to be made to the database using “mysqli_query($con, $sql)”. Once the Connect.php class was written, a new instance of the class was created. This newConnection object has an attribute on it called $connection, which holds the value of the db connection. When the executeQuery function is invoked and the parameters

are passed in (the db connection and a SQL statement) the tables in the course_enrollment_system database can be queried.

6.3 Creating the Registration Page and Saving User Information to the Database

In order to create the registration page, a form was brought in from the bootstrap library. However, some of the spacing needed to be customized because it looked a bit off. For example, some of the fields, such as Email and Password, did not align on the outter edge of the container with some of the fields below them, such as Address and Address 2. Through use of the inspection tool, it was discovered that there was some padding on the outside of the form groups. In order to overwrite this style, a custom CSS file was created and brought into the registration page. Once the form was inserted into the page and the styling fixed, the form was modified to include all of the fields needed for the course enrollment system. In order to create the dropdown menu for the State field, some PHP logic was used to iterate over an array that dynamically generates the dropdown of State abbreviations. Once that was done, some code was added to ensure that user input receives some basic validation before being sent to the database. This was first tackled by adding a required attribute to each of the form inputs. After doing some research, some basic tools for PHP form validation were found on w3schools.com. Validation now includes use of the $_SERVER[“PHP_SELF”] variable, which, “sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user gets error messages on the same page as the form” (w3schools.com, n.d.). The htmlspecialchars() function was also implemented, which prevents hackers from injecting HTML or Javascript code (w3schools.com, n.d.). Finally, the input was further validated by trimming it (removing extra whitespaces), and removing backslashes.

In regards to submitting the user input to the database, a SQL query was used. The $_POST superglobal variable from the form submission was used to retrieve the data from each form field and then save it to a variable. Once the input from each field was saved to a variable, they were passed

through the test_input() function (w3schools.com, n.d.) for validation. Then, it was necessary to concatenate the data from the $address1, $address2, $city, $state, and $zip variables to create an address string, which would get submitted to the database. Finally, a SQL query was constructed to insert each of the values into the student table, and then the executeQuery() function on the Connect class was passed the database connection and the SQL query in order to insert the data.

6.4 Screenshots of Landing, Login, and Registration Pages

Figure 8. Landing (Home) Page Layout

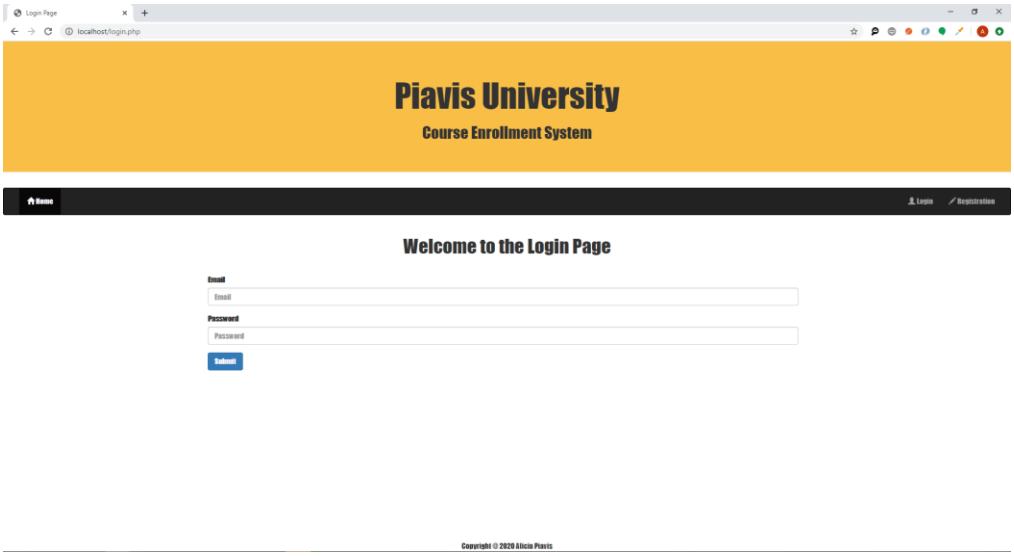


Figure 9. Login Page Layout

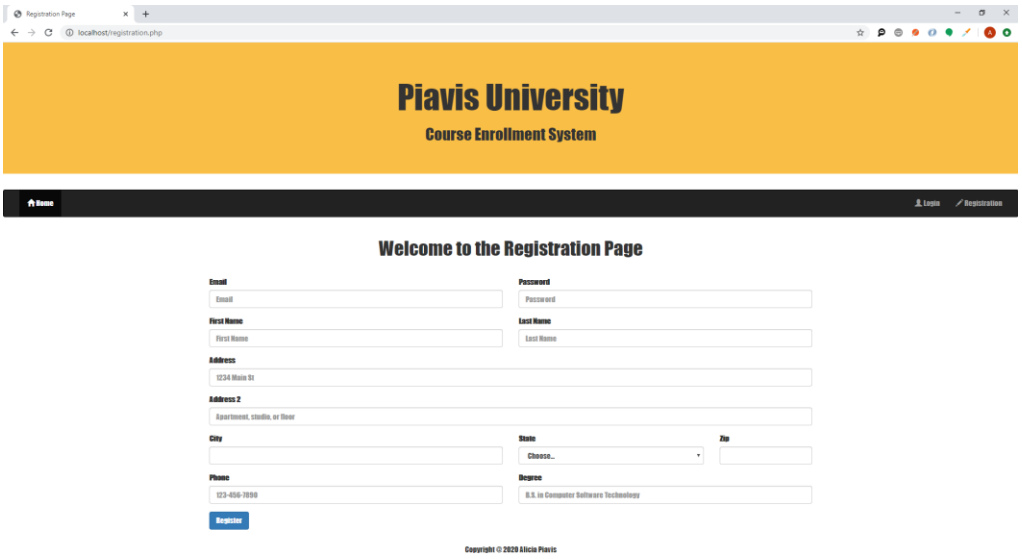


Figure 10. Account Registration Page Layout

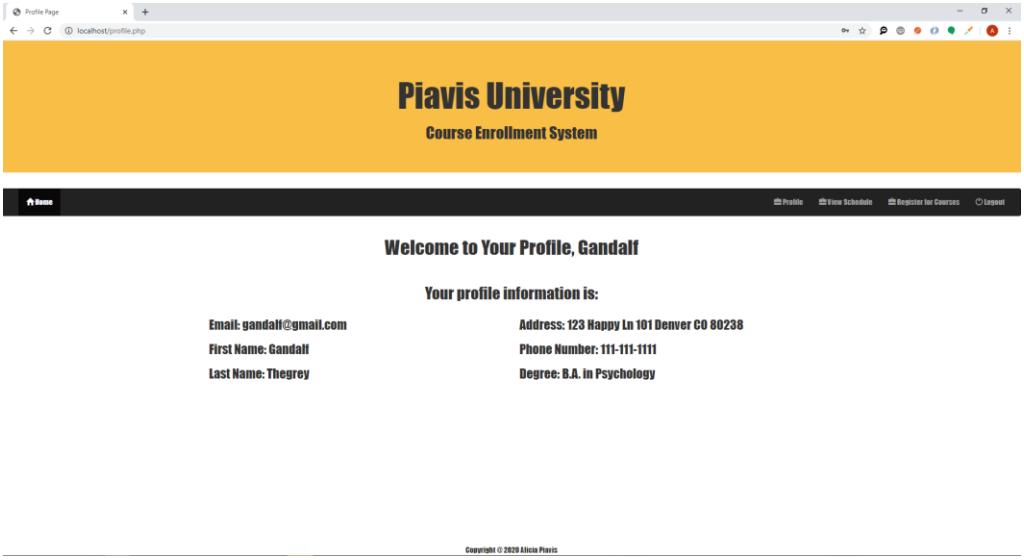


Figure 11. Profile Page Layout

7. Implementation of Course Registration Functionalities and MySQL Database Design

7.1 Overview of the Implementation Phase

The implementation phase was challenging due to time constraints. The full-stack application was built over about three weeks, but with availability on only about five days out of those three

weeks. The build required designing the database, generating mock data, seeding the database, writing all of our SQL queries, building out the front end, connecting the front end UI to the queries that run on the backend, learning more PHP, learning more Bootstrap, and learning how to write more complex queries in SQL. Because prior experience with PHP was minimal, a significant amount of research was required.

During those five or so days, the following parts of the build were completed: seeding the database, writing all of the queries for the backend of the application, building the UI for the Login, Profile, Register, Search Courses, and View Schedule pages, querying the database to pull a student's schedule from the database and display it on the page, building the UI for the "DROP" buttons to drop courses from the View Schedule page, building the UI to search for courses, dynamically generating dropdowns for semester and year based on a query that grabs those options from the database, dynamically generating the dropdown for course options based on the search criteria for semester and year that were submitted, checking if a student is already enrolled for a course, creating an enrollment when a student wants to add a new course to their schedule, checking if the course is already full, checking if the student is already on the waitlist, adding a student to the waitlist for a course if the course is full, updating the View Schedule page when a student adds a new course, building the functionality for a student to drop a course, automatically adding a student from the waitlist to a course when a different student drops the course, generating a notification to the student who was on the waitlist that they have been added to a course, adding a confirmation message that assures a student when they have successfully added a course, and updating the glyphicons next to the View Schedule and Register for Courses links on the navbar. With more time, two pieces of functionality that would have been built out further include implementing more validation on the account registration form, and

implementing a pop-up modal confirming that a student really wants to drop a course before that SQL query is executed.

7.2 Screenshots of View Schedule, Search Courses, and Add Course Pages

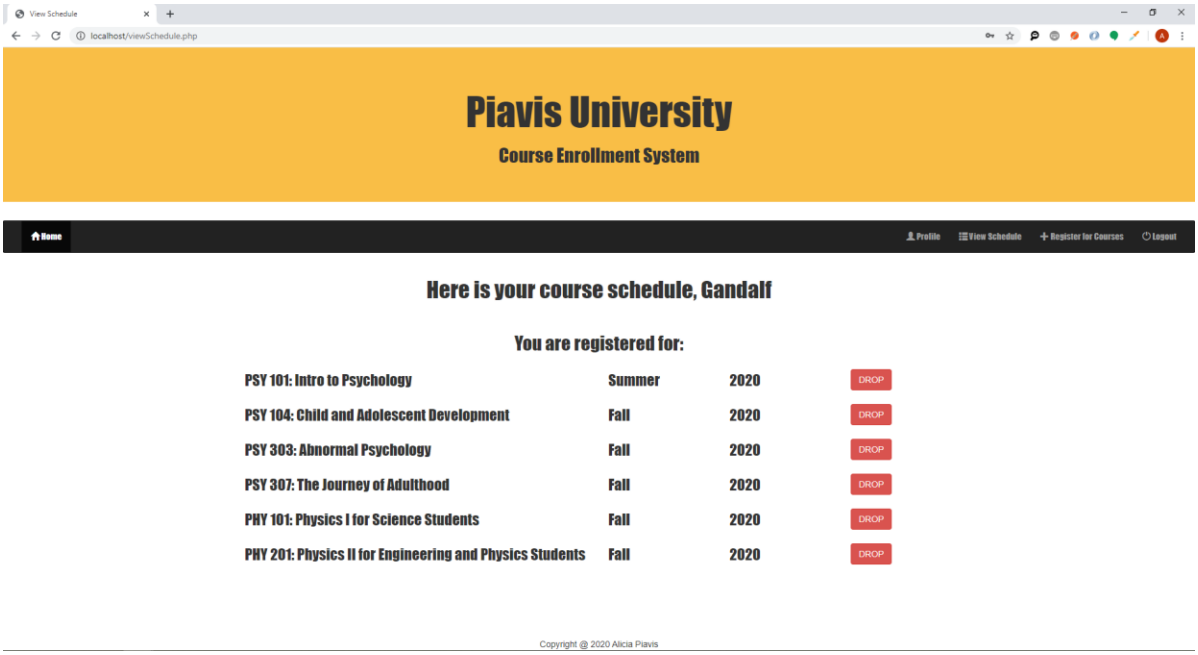


Figure 12. View Schedule Page Layout

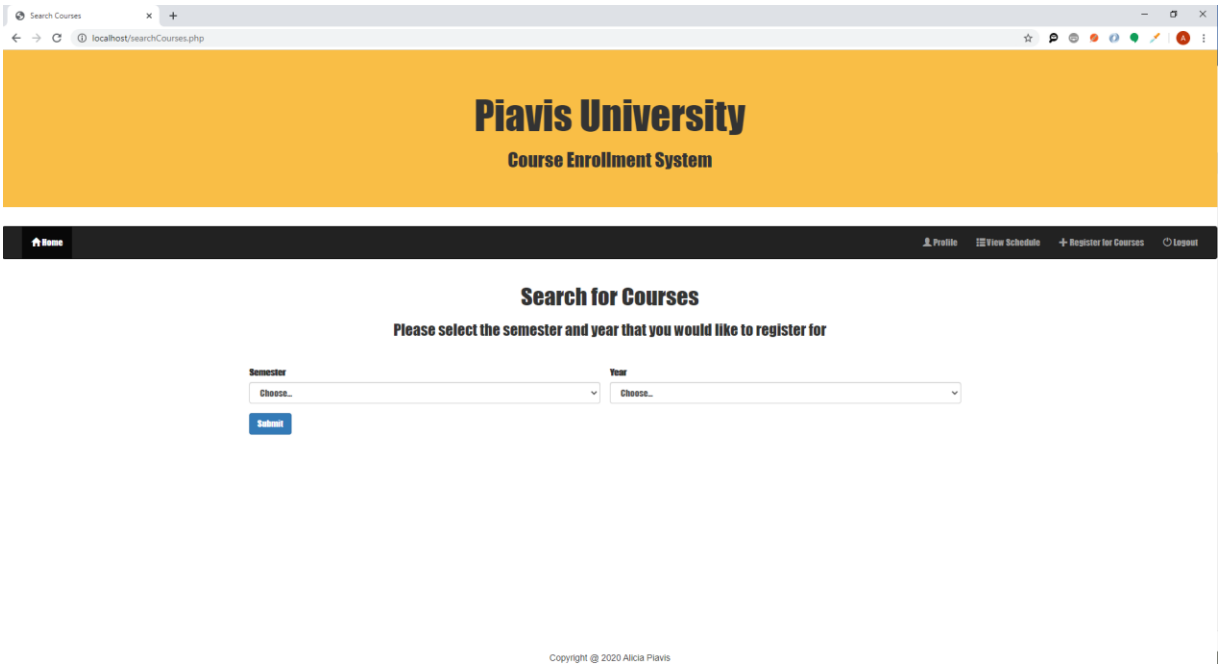


Figure 13. Search Courses Page Layout

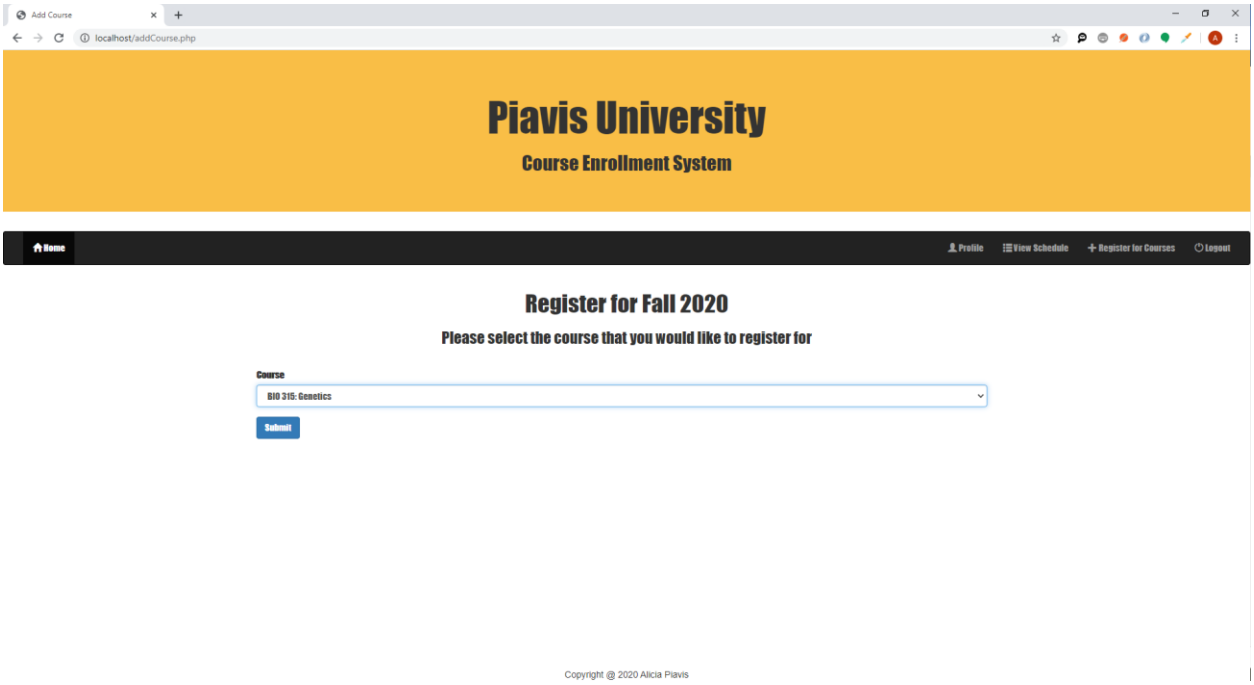


Figure 14. Add Course Page Layout

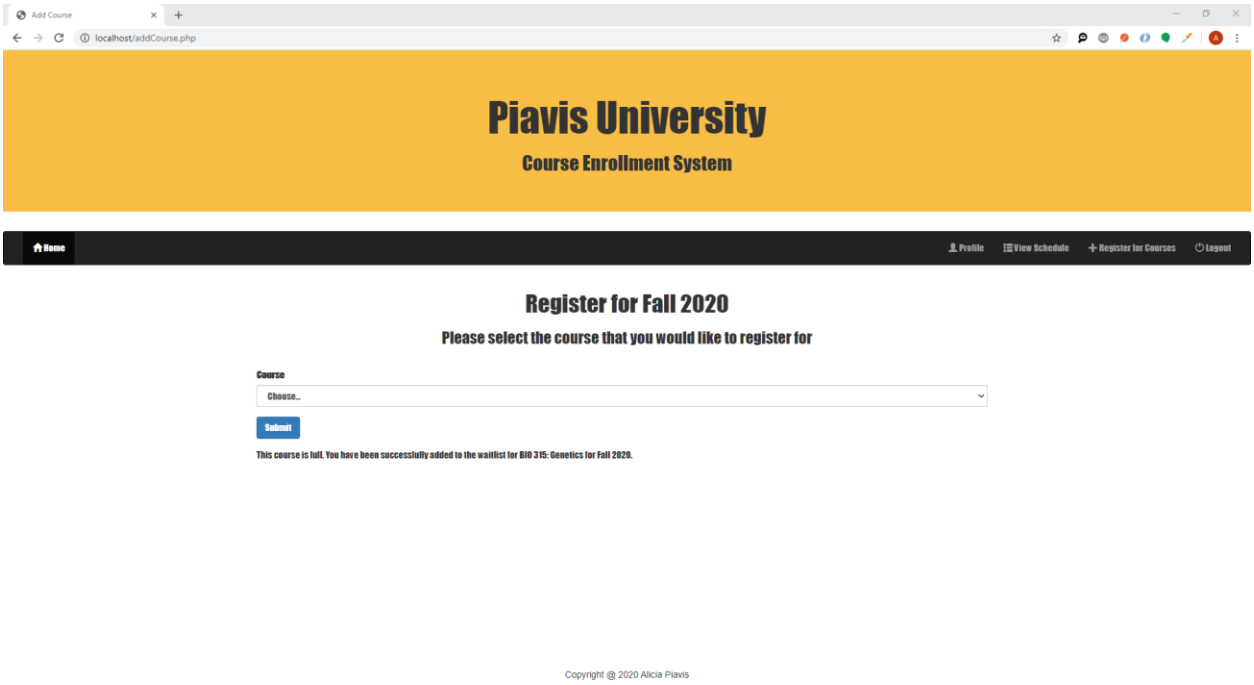


Figure 15. Notification Upon Being Added to the Waitlist for a Course

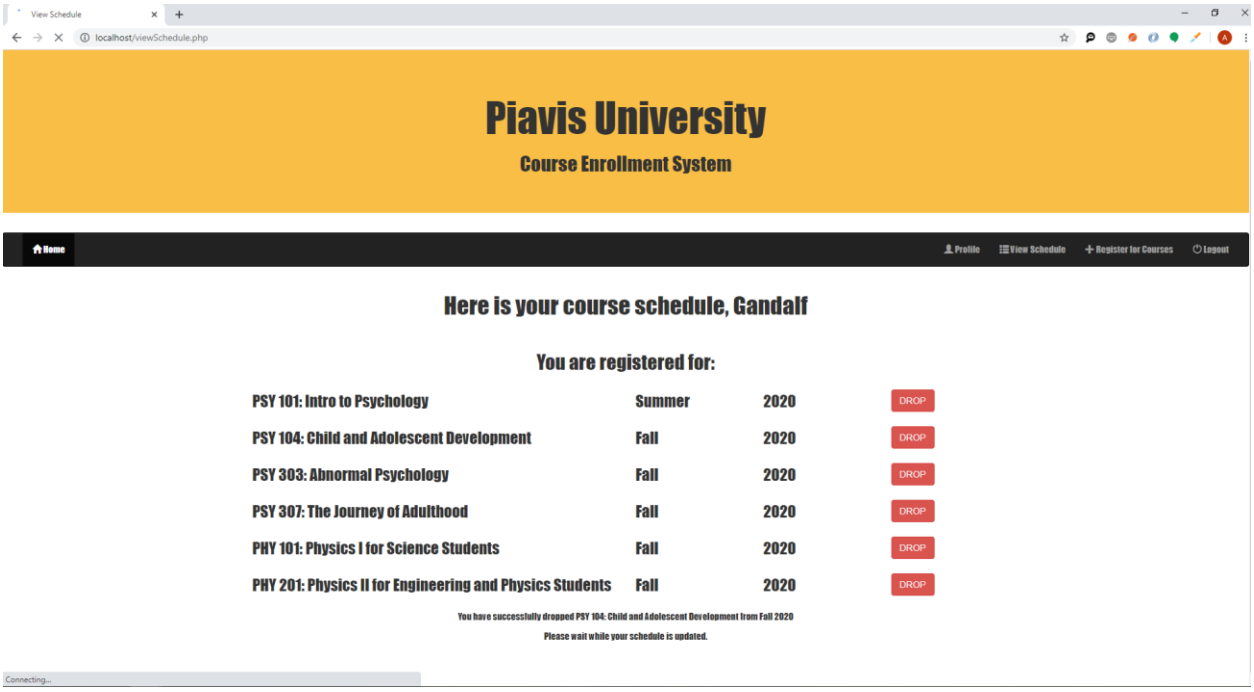


Figure 16. Confirmation That Course Has Been Dropped

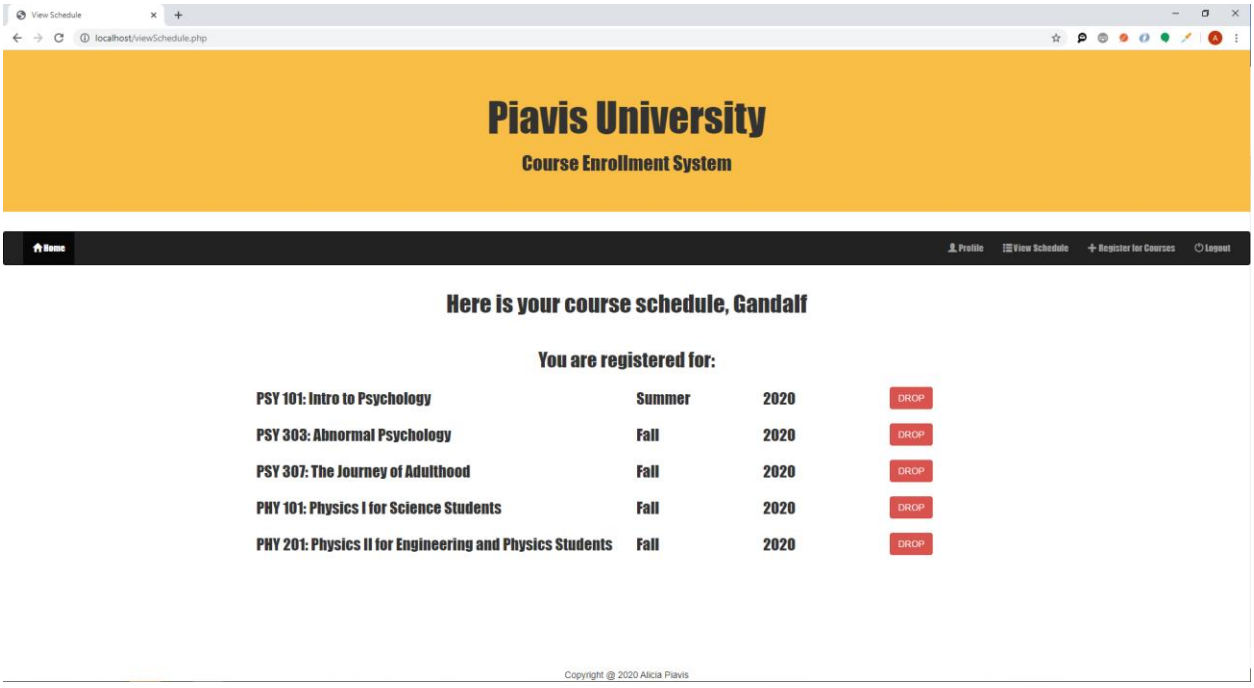


Figure 17. Updated Student Schedule Once Course Has Been Dropped

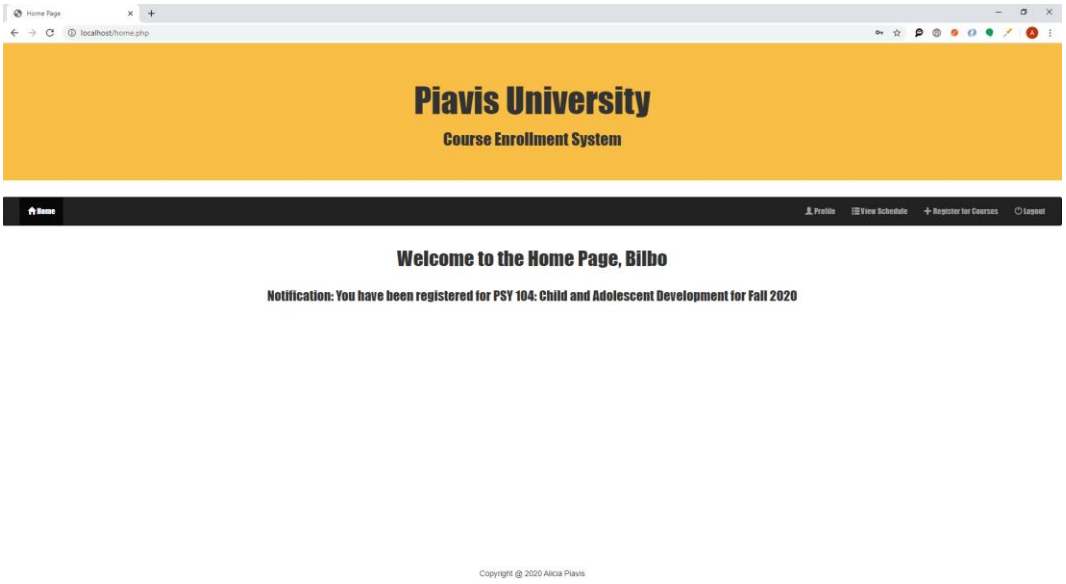


Figure 18. Notification When Student on a Waitlist Has Been Added to a Course

7.3 Screenshots Course Enrollment System Database Design

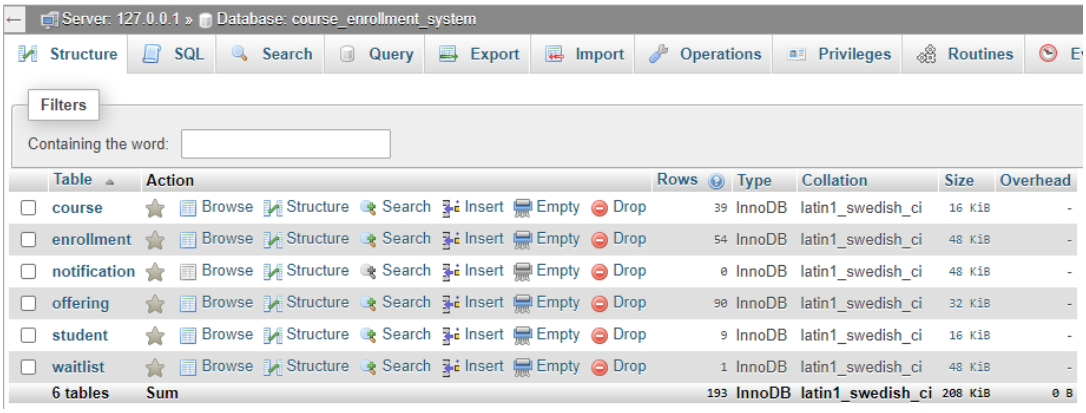
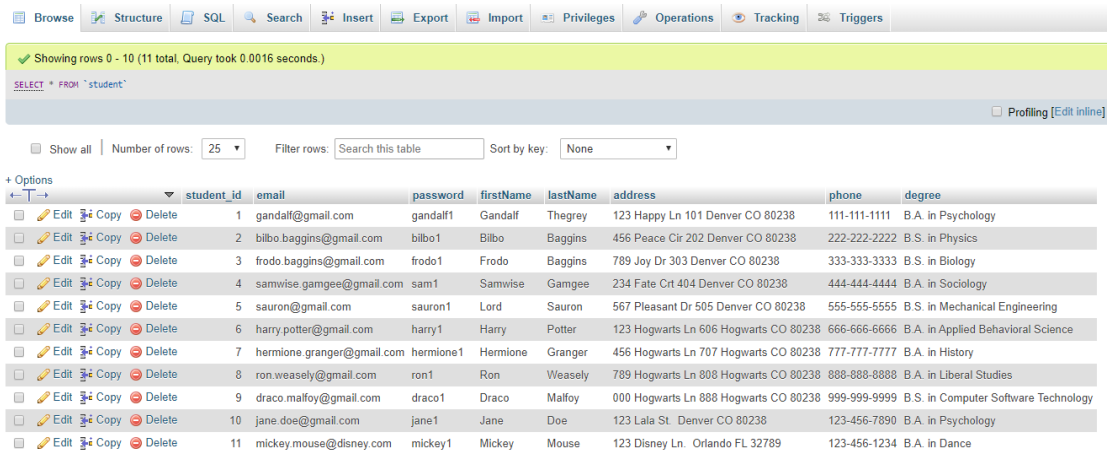


Figure 19. Course Enrollment System Database



Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Trac

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**student_id**	int(10)			No	*None*		AUTO_INCREMENT	Change Drop More
2	**email**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
3	**password**	varchar(20)	latin1_swedish_ci		No	*None*			Change Drop More
4	**firstName**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
5	**lastName**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
6	**address**	varchar(250)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
7	**phone**	varchar(13)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
8	**degree**	varchar(250)	latin1_swedish_ci		Yes	*NULL*			Change Drop More

Figure 20. “Student” Table in Course Enrollment System Database

Server: 127.0.0.1 » Database: course_enrollment_system » Table: course

Browse | Structure | SQL | Search | Insert | Export | Import | Privilege

Showing rows 0 - 24 (39 total, Query took 0.0013 seconds.)

```
SELECT * FROM `course`
```

1 > >> | Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

	course_id	courseName	maxStudents
Edit Copy Delete	1	PSY 101: Intro to Psychology	3
Edit Copy Delete	2	PSY 104: Child and Adolescent Development	3
Edit Copy Delete	3	PSY 301: Social Psychology	5
Edit Copy Delete	4	PSY 303: Abnormal Psychology	5
Edit Copy Delete	5	PSY 307: The Journey of Adulthood	3
Edit Copy Delete	6	PHY 101: Physics I for Science Students	3
Edit Copy Delete	7	PHY 201: Physics II for Engineering and Physics St...	3
Edit Copy Delete	8	PHY 303: Experimental Physics	5
Edit Copy Delete	9	PHY 310: Physics III Modern Essentials	5
Edit Copy Delete	10	BIO 120: Orientation to the Biology Major	3
Edit Copy Delete	11	BIO 312: Evolution	3
Edit Copy Delete	12	BIO 315: Genetics	3
Edit Copy Delete	13	BIO 332: General Ecology	5
Edit Copy Delete	14	BIO 341: Cell Biology	5
Edit Copy Delete	15	SOC 101: Intro to Sociology	3
Edit Copy Delete	16	SOC 120: Intro to Ethics & Social Responsibility	3
Edit Copy Delete	17	SOC 203: Social Problems	5
Edit Copy Delete	18	SOC 301: Identity & Social Inequality	3
Edit Copy Delete	19	MAE 101: Intro to ME Design	3
Edit Copy Delete	20	MAE 206: Engineering Statistics	5
Edit Copy Delete	21	MAE 208: Engr Dynamics	3
Edit Copy Delete	22	MAE 201: Engr Thermodynamics	3
Edit Copy Delete	23	MAE 308: Fluid Mechanics	3
Edit Copy Delete	24	ABS 200: Intro to Applied Behavioral Science	3
Edit Copy Delete	25	ABS 300: Psychological Assessment	5

Console | Check all | *With selected:* Edit Copy Delete Export

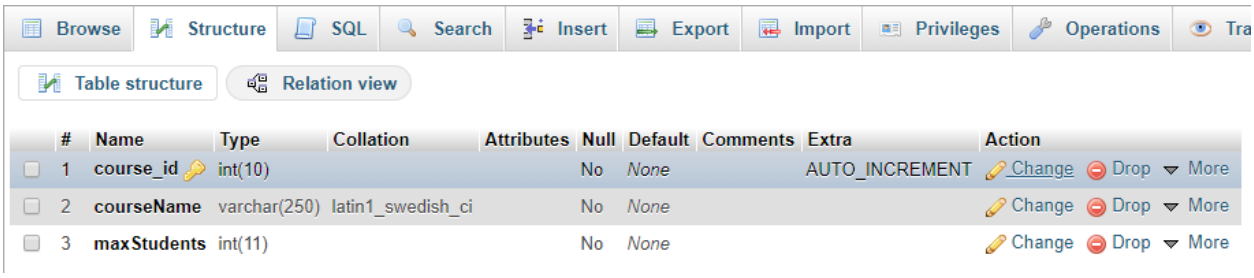
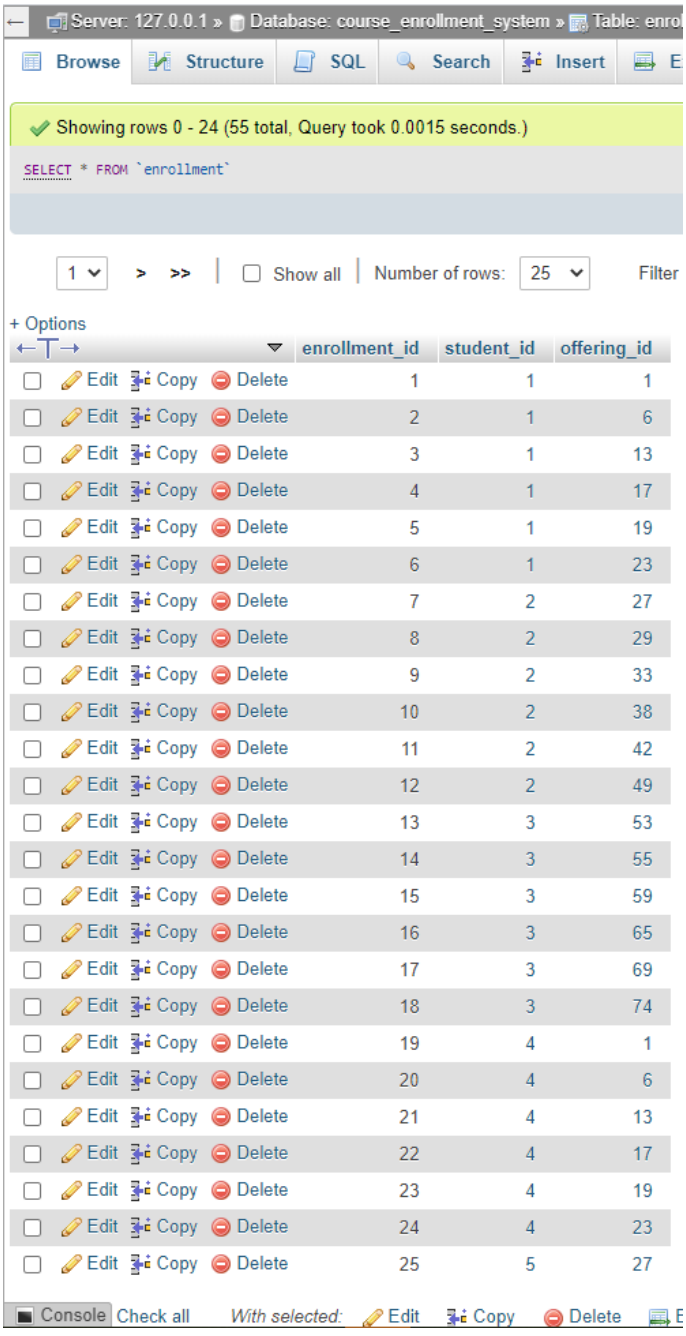


Figure 21. "Course" Table in Course Enrollment System Database



Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	enrollment_id	int(10)			No	None		AUTO_INCREMENT	Change Drop More
2	student_id	int(10)			No	None			Change Drop More
3	offering_id	int(10)			No	None			Change Drop More

Figure 22. “Enrollment” Table in Course Enrollment System Database

Server: 127.0.0.1 » Database: course_enrollment_system » Table: offering

Browse | Structure | SQL | Search | Insert | Exp

Showing rows 0 - 24 (90 total, Query took 0.0014 seconds.)

```
SELECT * FROM `offering`
```

1 > >> | Show all | Number of rows: 25 | Filter rov

+ Options

	offering_id	course_id	year	semester
Edit Copy Delete	1	1	2020	Summer
Edit Copy Delete	2	1	2020	Fall
Edit Copy Delete	3	1	2021	Spring
Edit Copy Delete	4	1	2021	Summer
Edit Copy Delete	5	1	2021	Fall
Edit Copy Delete	6	2	2020	Fall
Edit Copy Delete	7	2	2021	Spring
Edit Copy Delete	8	2	2021	Fall
Edit Copy Delete	9	3	2020	Summer
Edit Copy Delete	10	3	2021	Spring
Edit Copy Delete	11	3	2021	Summer
Edit Copy Delete	12	4	2020	Summer
Edit Copy Delete	13	4	2020	Fall
Edit Copy Delete	14	4	2021	Spring
Edit Copy Delete	15	4	2021	Summer
Edit Copy Delete	16	4	2021	Fall
Edit Copy Delete	17	5	2020	Fall
Edit Copy Delete	18	5	2021	Fall
Edit Copy Delete	19	6	2020	Fall
Edit Copy Delete	20	6	2021	Spring
Edit Copy Delete	21	6	2021	Fall
Edit Copy Delete	22	7	2020	Summer
Edit Copy Delete	23	7	2020	Fall
Edit Copy Delete	24	7	2021	Spring
Edit Copy Delete	25	7	2021	Summer

Console | Check all | With selected: Edit Copy Delete Exp

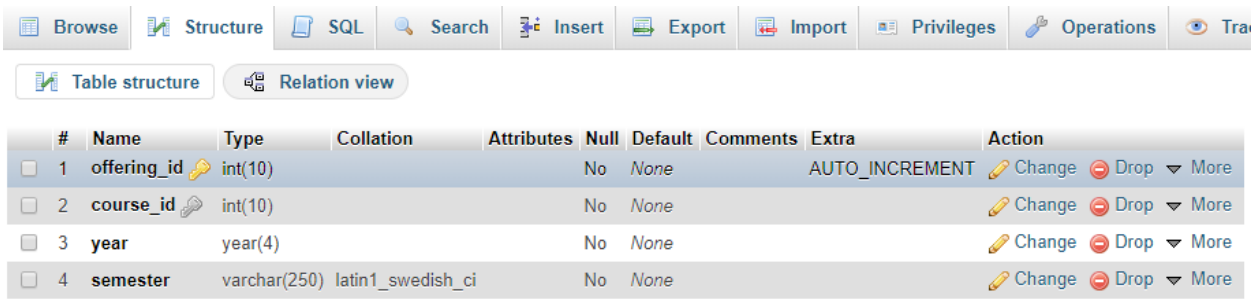
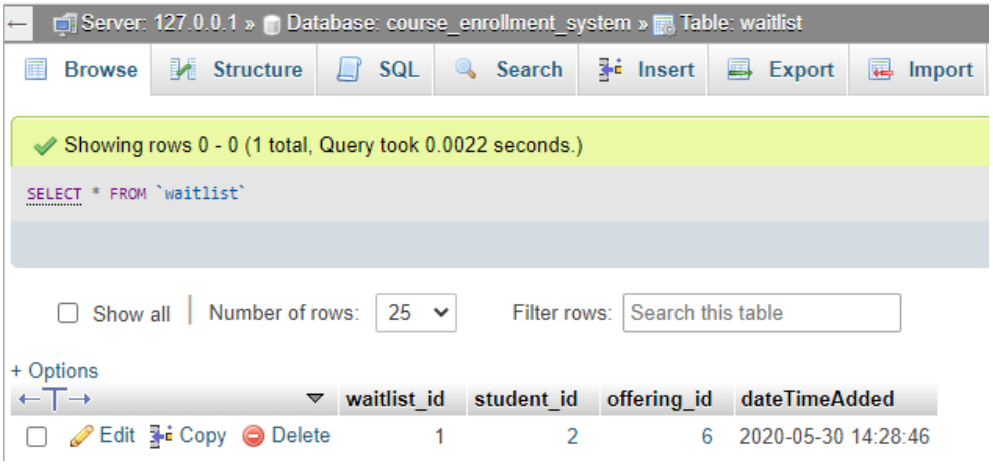


Figure 23. "Offering" Table in Course Enrollment System Database



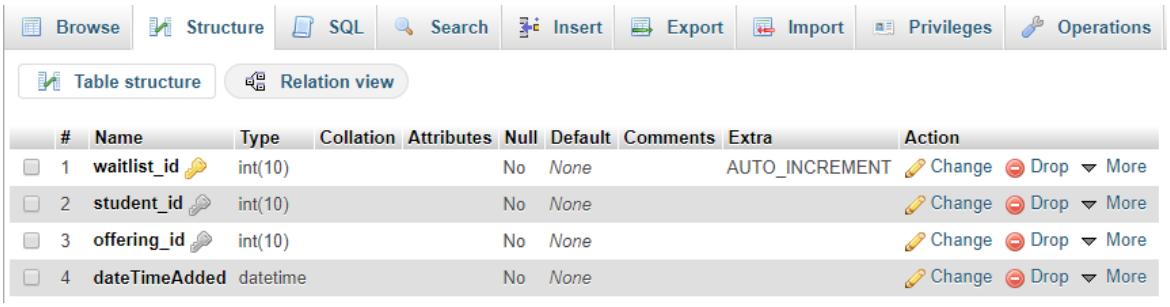


Figure 24. "Waitlist" Table in Course Enrollment System Database

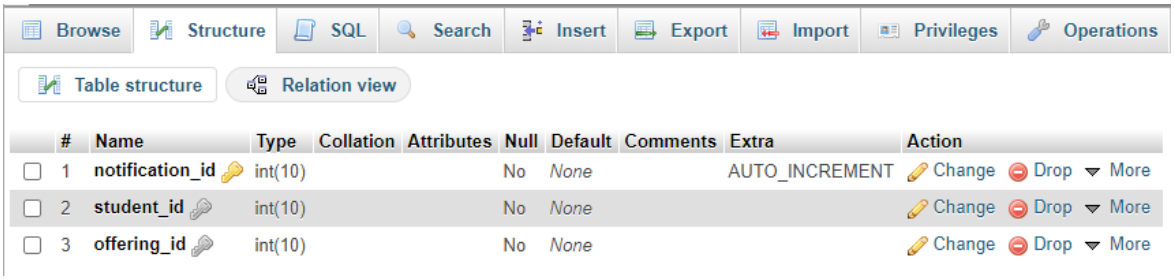


Figure 25. "Notification" Table in Course Enrollment System Database

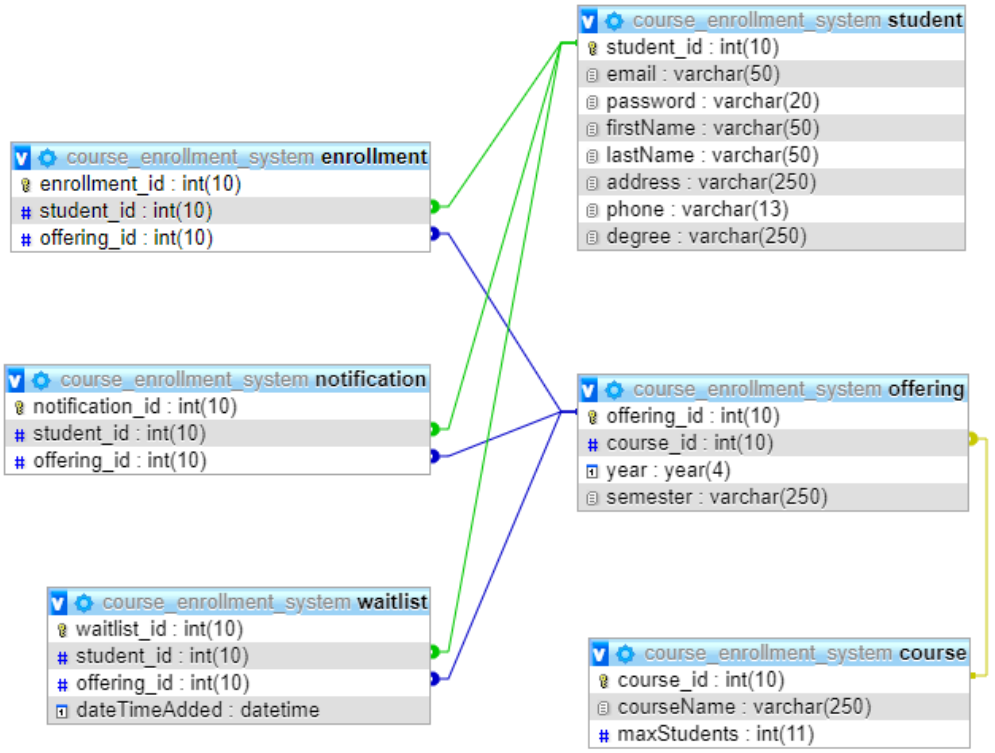


Figure 26. Design for Course Enrollment System Database

8. Resources

ADA. (2007). Website Accessibility Under Title II of the ADA. Retrieved April 5, 2020 from https://www.ada.gov/pcatoolkit/chap5toolkit.htm

Bahaweres, R. B., Zawawi, K., Khairani, D., & Hakiem, N. (2017). Analysis of statement branch and loop coverage in software testing with genetic algorithm. 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Electrical Engineering, Computer Science and Informatics (EECSI), 2017 4th International Conference On, 1–6. https://doi-org.proxy-library.ashford.edu/10.1109/EECSI.2017.8239088

Connolly, R., & Hoar, R. (2018). Fundamentals of web programming (2nd ed.). Retrieved from https://www.vitalsource.com

Dmitriev, S. O., Valter, D. A., & Kontsov, A. M. (2020). System for Efficient Storage and Version Control of Arbitrary File Collections. 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020 IEEE Conference Of, 295–298. https://doi-org.proxy-library.ashford.edu/10.1109/EIConRus49466.2020.9038922

Font Awesome. (n.d.) Font Awesome Icons. Retrieved April 6, 2020 from https://fontawesome.com/icons?d=gallery

GeeksForGeeks.com. (n.d.). Software Engineering: Control Flow Graph (CFG). Retrieved April 18, 2020 from https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/

Homès, B. (2012). Fundamentals of Software Testing, John Wiley & Sons, Incorporated. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ashford-ebooks/detail.action?docID=1120766.
Created from ashford-ebooks on 2020-04-25 14:33:08.

Khalid, R. (2017). Towards an automated tool for software testing and analysis. 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference On, 461–465. https://doi-org.proxy-library.ashford.edu/10.1109/IBCAST.2017.7868094

Khandelwal, A. (2019). Top 10 Non -Functional Testing Tools | Pros and Cons. Retrieved June 4, 2020 from https://testinggenez.com/non-functional-testing-tools/

Koschara, F. (2013). PHP Code Style Guide. MIT Sloan School of Management. Retrieved June 4, 2020 from https://mitsloan.mit.edu/shared/content/PHP_Code_Style_Guide.php

Mikoluk, K. (2013). XAMPP tutorial: How to use XAMPP to run your own web server [Blog post]. Retrieved from https://blog.udemy.com/xampp-tutorial

M. Parizi, R., Qian, K., Shahriar, H., Wu, F., & Tao, L. (2018). Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Computer Software and Applications Conference (COMPSAC), 2018 IEEE 42nd Annual, COMPSAC, 01, 825–826. https://doi-org.proxy-library.ashford.edu/10.1109/COMPSAC.2018.00139

Software Testing Help. (2020). Top 10 Best Test Data Generation Tools in 2020. Retrieved June 4, 2020 from https://www.softwaretestinghelp.com/test-data-generation-tools/.

Spillner, A., Linz, T., & Schaefer, H. (2014). Software testing foundations: A study guide for the certified tester exam (4th ed.). Rocky Nook.

Tsui, F., Karam, O., & Bernal, B. (2018). Essentials of software engineering (4th ed.). Jones & Bartlett Learning.

U.S. Department of Education. (n.d.). Family Educational Rights and Privacy Act (FERPA).

W3schools.com. (n.d.). PHP Form Validation. Retrieved March 9, 2020 from https://www.w3schools.com/php/php_form_validation.asp

Wiegers, K. E. (1999). Software requirements specifications for <project> [Template].

https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc

Appendix A: Glossary

ADA- Americans with Disabilities Act of 1990

HTTP- hypertext transfer protocol

PII- Personally Identifiable Information

SRS- Software Requirements Specification

TCP- transmission control protocol

UI- User Interface

UX- User Experience