**Requirements Analysis Document**

Security in Software Engineering Case Study

CSCI 4712 Senior Capstone Project

Spring 2022

Augusta University

Augusta, GA

Team Members

Table of Contents

[1 Introduction 3](#_Toc100014685)

[1.1 Scope of System 3](#_Toc100014686)

[2 Requirements of Systems 4](#_Toc100014687)

[2.1 Functional Requirements 4](#_Toc100014688)

[2.2 Non-Functional Requirements 5](#_Toc100014689)

[2.3 Use Cases 6](#_Toc100014690)

[2.4 Use Case Description 7](#_Toc100014691)

[3 Sequence Diagrams 17](#_Toc100014692)

[3.1 Login Use Case 17](#_Toc100014693)

[3.2 Logout Use Case 18](#_Toc100014694)

[3.3 Startup Use Case 19](#_Toc100014695)

[3.4 CreateAccount Use Case 20](#_Toc100014696)

[3.5 UserSearch Use Case 21](#_Toc100014697)

[3.6 EditAccount Use Case 22](#_Toc100014698)

[3.7 CourseSearch Use Case 23](#_Toc100014699)

[3.8 CourseEnroll Use Case 24](#_Toc100014700)

[3.9 EnterGrade Use Case 25](#_Toc100014701)

[3.10 ForgotPassword Use Case 26](#_Toc100014702)

[4 Class Diagrams 27](#_Toc100014703)

[4.1 Control Subsystem 27](#_Toc100014704)

[5 Dataflow Diagrams 28](#_Toc100014705)

[5.1 Login Controller 28](#_Toc100014706)

[5.2 Logout Controller 29](#_Toc100014707)

[5.3 Startup Controller 30](#_Toc100014708)

[5.4 CreateAccount Controller 31](#_Toc100014709)

[6 Data Model 32](#_Toc100014710)

[6.1 ER Diagram 32](#_Toc100014711)

[7 User Interface Mockups 33](#_Toc100014712)

[7.1 Login 33](#_Toc100014713)

[7.2 Dashboard 34](#_Toc100014714)

[7.3 CourseSearch 37](#_Toc100014715)

[7.4 CourseEnroll 38](#_Toc100014716)

[7.5 EnterGrades 39](#_Toc100014717)

[7.6 CreateAccount 40](#_Toc100014718)

[7.7 UserSearch 42](#_Toc100014719)

[7.8 EditAccount 44](#_Toc100014720)

[7.9 ForgotPassword 45](#_Toc100014721)

# Introduction

## Scope of System

This web application simulates a university course management system. It supports three actors: Faculty, Student, and Admin. Faculty represents professors at the university who teach classes and enter grades. Student represents students of said professors, who enroll in courses and earn grades. Admin represents a single user who manages the assets of the system, namely user information.

The system supports these actors by allowing Students to search for and enroll in courses, allowing Faculty to enter grades for the courses, and allowing Admins to search, create, and edit user accounts.

The purpose of the application is to allow users to try to break the app, using various vulnerabilities and exploits. The software also resets itself to its initial settings following a session, allowing users to try various things without worrying about destroying their installation. The system is browser-based, but the server is local and runs on the client’s computer.

# Requirements of Systems

## Functional Requirements

* CourseEnroll – Student is able to register for a course.
* CourseSearch – Student is able to search for a course from the course catalog.
* CreateAccount – Admin can create accounts for all types of actors.
* ForgotPassword – All actors can receive password assistance in the event of a forgotten password.
* Login – All actors are required to login to access the system.
* Logout – All actors are required to logout to terminate their session. The system should automatically terminate a user session after a period of inactivity.
* EnterGrade – Faculty is able to enter student grades for a course that is assigned to them by uploading a grade report in CSV.
* EditAccount – Admin is able to edit the profile of any existing account.
* SearchUser – Admin can perform a search for existing users.
* Startup – This use case configures the application on startup and loads the initial welcome screen.

## Non-Functional Requirements

* Platform
  + Target operating system is Microsoft Windows
  + System is not web-based but should utilize a web browser
  + System should be implemented using PHP, HTML, CSS, SQLite, and Javascript
* Usability
  + System should implement a graphical user interface

## Use Cases

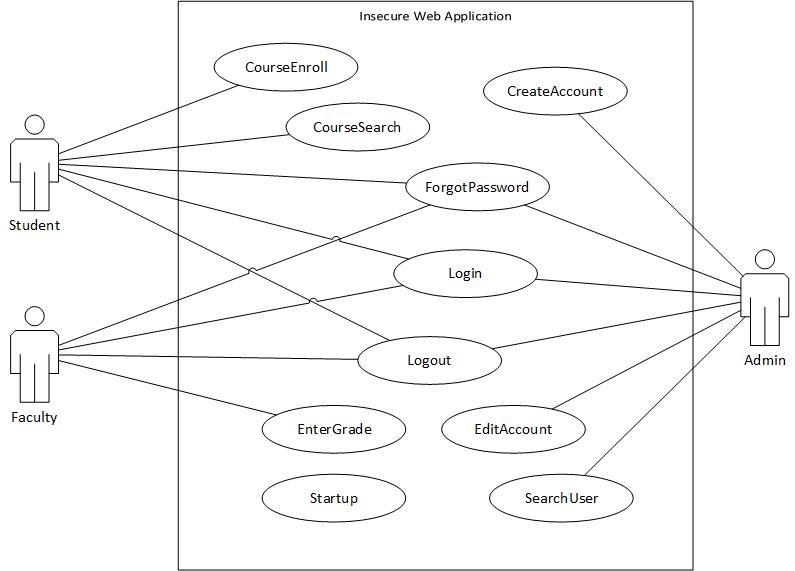


Figure 2.1: use case diagram

## Use Case Description

|  |  |
| --- | --- |
| *Use case name* | Login |
| *Participating*  *actors* | Initiated by Admin, Student, or Faculty |
| *Flow of events* | 1. The user fills out the login form by entering their username and password. The user submits the form. 2. **Application displays the respective user dashboard OR an error message if credentials are incorrect.** |
| *Entry condition* |  |
| *Exit condition* | * Correct dashboard is displayed * Login form is displayed. |
| *Security*  *requirements* | * Validate input data for format (Event # 1) * Validate input data for cross-site scripting (Event # 1) * Validate input data for SQL injection (Event # 1) * Validate user credentials (Event # 1) * Validate client-side view (Event # 2) * Use generic error message (Event # 2) * Use secure transport (Event # 1, 2) |

Figure 2.2: Login

|  |  |
| --- | --- |
| *Use case name* | LogOut |
| *Participating*  *actors* | Initiated by Student, Admin or Faculty |
| *Flow of events* | 1. The user selects the logout function from any form. 2. **Application terminates the user's session and displays the login form.** |
| *Entry condition* | * The user is logged in. |
| *Exit condition* | * Login form is displayed |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1) * Validate client-side view (Event # 2) * Use secure transport (Event # 1, 2) |

Figure 2.4: LogOut

|  |  |
| --- | --- |
| *Use case name* | Startup |
| *Participating*  *actors* |  |
| *Flow of events* | 1. This use case is initiated on launch of the application. 2. **The database is initialized with stored information from the configuration file and the login form is displayed.** |
| *Entry condition* |  |
| *Exit condition* | * Database is initialized and login form is displayed. |
| *Security*  *requirements* | * Store user credentials securely (Event # 2) |

Figure 2.5: Startup

|  |  |
| --- | --- |
| *Use case name* | CreateAccount |
| *Participating*  *actors* | Initiated by Admin |
| *Flow of events* | 1. Admin selects the create account function from the admin dashboard. 2. **Application displays the create account form if user is authorized.** 3. Admin selects an account type from a list. 4. **Application displays input boxes relevant to the selected account type. For all users, this includes first name, last name, date of birth, email, confirm email, password, confirm password, security question, and security answer. Faculty and Student have an additional input field for faculty rank and student year, respectively.** 5. Admin enters information into the form and submits it. 6. **Application creates an account corresponding to the information submitted in the form and displays the admin dashboard OR error message if the data is invalid.** |
| *Entry condition* | * Admin is logged in. |
| *Exit condition* | * Admin dashboard is displayed * Data is correctly stored |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 5) * Check user authorization (Event # 1, 5) * Validate input data for format (Event # 5) * Validate input data for cross-site scripting (Event # 5) * Validate input data for SQL injection (Event # 5) * Validate input data code injection (Event # 5) * Validate client-side view (Event # 2, 4, 6) * Use generic error message (Event # 6) * Store user credentials securely (Event # 6) * Use secure transport (Event # 1, 2, 5, 6) |

Figure 2.6: CreateAccount

|  |  |
| --- | --- |
| *Use case name* | UserSearch |
| *Participating*  *actors* | Initiated by Admin |
| *Flow of events* | 1. Admin selects the search user function from the admin dashboard. 2. **Application displays the user search form if user is unauthorized.** 3. Admin enters information into the form and submits it. 4. **The displays the user search form with results of the search OR error message if data is invalid.** |
| *Entry condition* | * Admin is logged in. |
| *Exit condition* | * User search form is displayed |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 3) * Check user authorization (Event # 1, 3) * Validate input data for format (Event # 3) * Validate input data for cross-site scripting (Event # 3) * Validate input data for SQL injection (Event # 3) * Validate input data code injection (Event # 3) * Validate client-side view (Event # 2, 4) * Use generic error message (Event # 4) * Use secure transport (Event # 1, 2, 3, 4) |

Figure 2.7: UserSearch

|  |  |
| --- | --- |
| *Use case name* | EditAccount |
| *Participating*  *actors* | Initiated by Admin |
| *Flow of events* | 1. Admin selects the edit account function from the search result form. 2. **Application displays the edit account form that is pre-populated with user’s account information if user is unauthorized.** 3. Admin enters information that needs to be updated for the selected account and submits the form. 4. **Application updates the selected account with the changed data and displays the user search form OR error message if data is invalid.** |
| *Entry condition* | * Admin is logged in. |
| *Exit condition* | * User search form is displayed * Data is correctly stored |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 3) * Check user authorization (Event # 1, 3) * Validate input data for format (Event # 3) * Validate input data for cross-site scripting (Event # 3) * Validate input data for SQL injection (Event # 3) * Validate input data code injection (Event # 3) * Validate client-side view (Event # 2, 4) * Use generic error message (Event # 4) * Use secure transport (Event # 1, 2, 3, 4) |

Figure 2.8: EditAccount

|  |  |
| --- | --- |
| *Use case name* | CourseSearch |
| *Participating*  *actors* | Initiated by Student |
| *Flow of events* | 1. Student selects the course search function from the student dashboard. 2. **Application displays course search form if user is unauthorized.** 3. Student searches for a course by selecting a semester, followed by a professor, a course name, and/or a course identifier. 4. **Application displays the course search form with list of courses that match the search criteria OR error message if data is invalid.** |
| *Entry condition* | * Student is logged in. |
| *Exit condition* | * Course search form is displayed. |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 3) * Check user authorization (Event # 1, 3) * Validate input data for format (Event # 3) * Validate input data for cross-site scripting (Event # 3) * Validate input data for SQL injection (Event # 3) * Validate input data code injection (Event # 3) * Validate client-side view (Event # 2, 4) * Use generic error message (Event # 4) * Use secure transport (Event # 1, 2, 3, 4) |

Figure 2.9: CourseSearch

|  |  |
| --- | --- |
| *Use case name* | CourseEnroll |
| *Participating*  *actors* | Initiated by Student |
| *Flow of events* | 1. Student selects the course enroll function from course search form. 2. **Application displays course enroll form, showing available sections for the selected course if user is unauthorized.** 3. Student selects a course section and submits the form. 4. **Application enrolls Student for that course and displays the course search form OR error message if data is invalid.** |
| *Entry condition* | * Student is logged in. |
| *Exit condition* | * Course search form is displayed. * Data is correctly stored |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 3) * Check user authorization (Event # 1, 3) * Validate input data for format (Event # 3) * Validate input data for cross-site scripting (Event # 3) * Validate input data for SQL injection (Event # 3) * Validate input data code injection (Event # 3) * Validate client-side view (Event # 2, 4) * Use generic error message (Event # 4) * Use secure transport (Event # 1, 2, 3, 4) |

Figure 2.10: CourseEnroll

|  |  |
| --- | --- |
| *Use case name* | EnterGrade |
| *Participating*  *actors* | Initiated by Faculty |
| *Flow of events* | 1. Faculty selects the enter grade function from the faculty dashboard. 2. **Application displays the enter grades form if user is unauthorized.** 3. Faculty enters a course identifier, and selects the upload file function. 4. **Application displays a file chooser for user to select a comma-separated value (CSV) file containing data for Student and Grade.** 5. Faculty selects a CSV file to upload, then submits the form 6. **Application updates the grades for the selected course using the data in the file and displays the faculty dashboard OR error message if data is invalid.** |
| *Entry condition* | * Faculty is logged in. |
| *Exit condition* | * Faculty dashboard is displayed. * Data is correctly stored |
| *Security*  *requirements* | * ­­Check user authentication (Event # 1, 5) * Check user authorization (Event # 1, 5) * Validate input data for format (Event # 3, 5) * Validate input data for cross-site scripting (Event # 3, 5) * Validate input data for SQL injection (Event # 3, 5) * Validate input data code injection (Event # 3, 5) * Validate file format and content (Event # 5) * Validate client-side view (Event # 2, 6) * Use generic error message (Event # 6) * Use secure transport (Event # 1 - 6) |

Figure 2.11: EnterGrade

|  |  |
| --- | --- |
| *Use case name* | ForgotPassword |
| *Participating*  *actors* | Initiated by Faculty, Admin, or Student |
| *Flow of events* | 1. User selects the forgot password function from the login form.  **2. Application displays the forgot password form for the user’s email address.**  3. User enters an email address and submits the form.  **4. Application checks if the email address exists and displays the security question on the security form OR displays error message if data is invalid.**  5. User enters their security answer and submits the form.  **6. Application checks that the security answer is correct and displays the password form for a new password to be entered by the user OR displays error message if data is invalid.**  7. User enters a new password and confirms they typed the password correctly in confirm password, then submits.  **8. Application updates the password in the database and displays the login form OR displays error message if data is invalid.** |
| *Entry condition* |  |
| *Exit condition* | * Login form is displayed * Data is correctly stored |
| *Security*  *requirements* | * Validate input data for format (Event # 3, 5, 7) * Validate input data for cross-site scripting (Event # 3, 5, 7) * Validate input data for SQL injection (Event # 3, 5, 7) * Validate input data code injection (Event # 3, 5, 7) * Validate client-side view (Event # 2, 4, 6, 8) * Use generic error message (Event # 4, 6, 8) * Store user credentials securely (Event # 8) * Use secure transport (Event # 1-8) |

Figure 2.11: ForgotPassword

# Sequence Diagrams

## Login Use Case

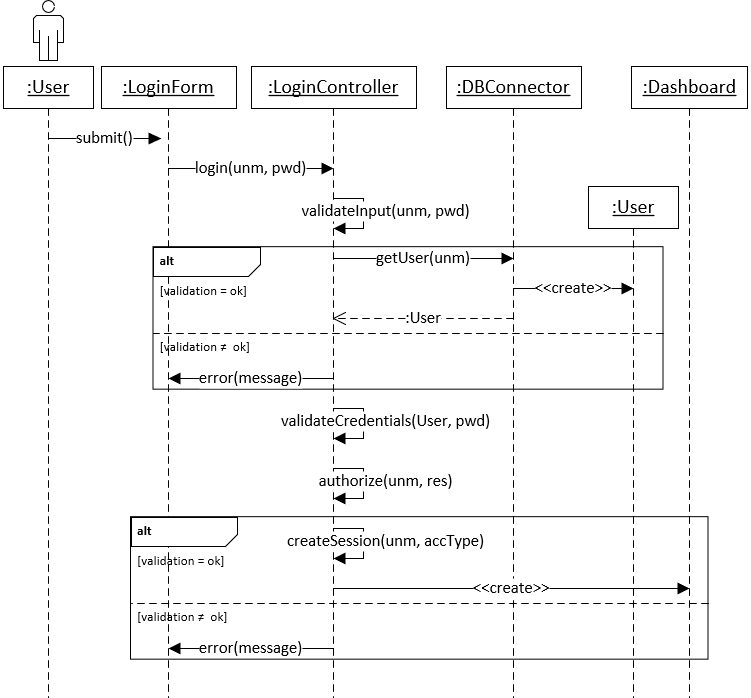


Figure 3.1: Interaction for login use case

## Logout Use Case

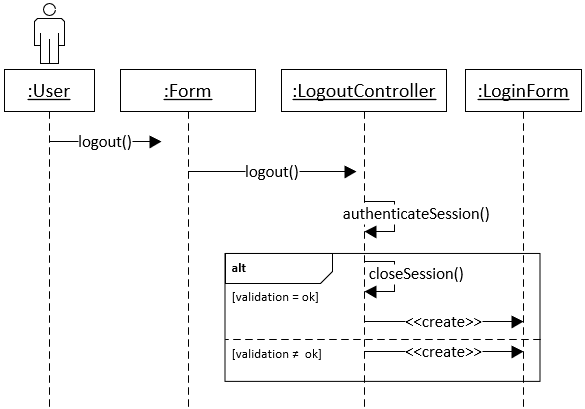


Figure 3.2: Interaction for logout use case

## Startup Use Case

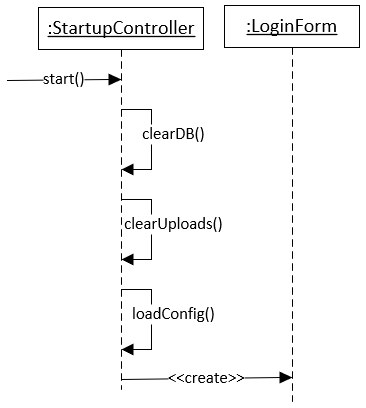


Figure 3.3: Interaction of startup use case

## CreateAccount Use Case

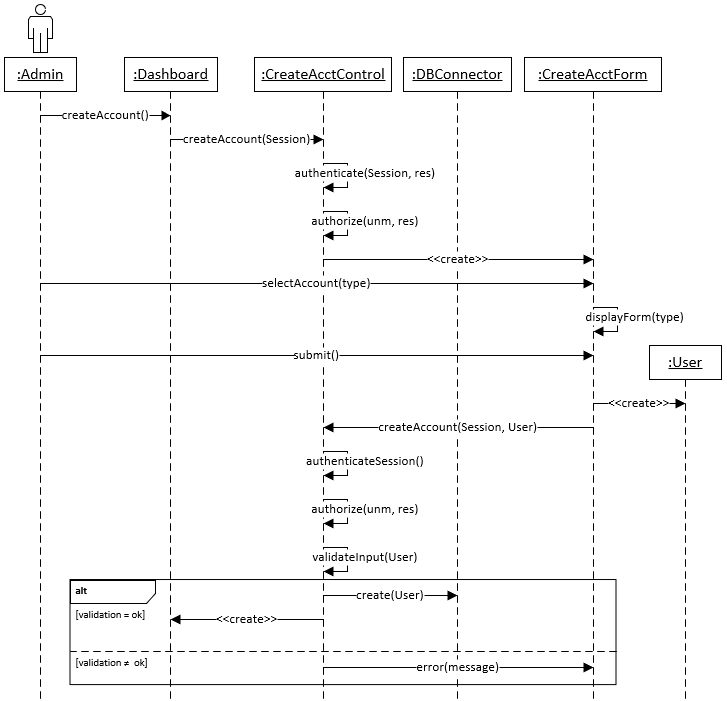


Figure 3.4: Interaction for create account use case (for valid admin session)

## UserSearch Use Case

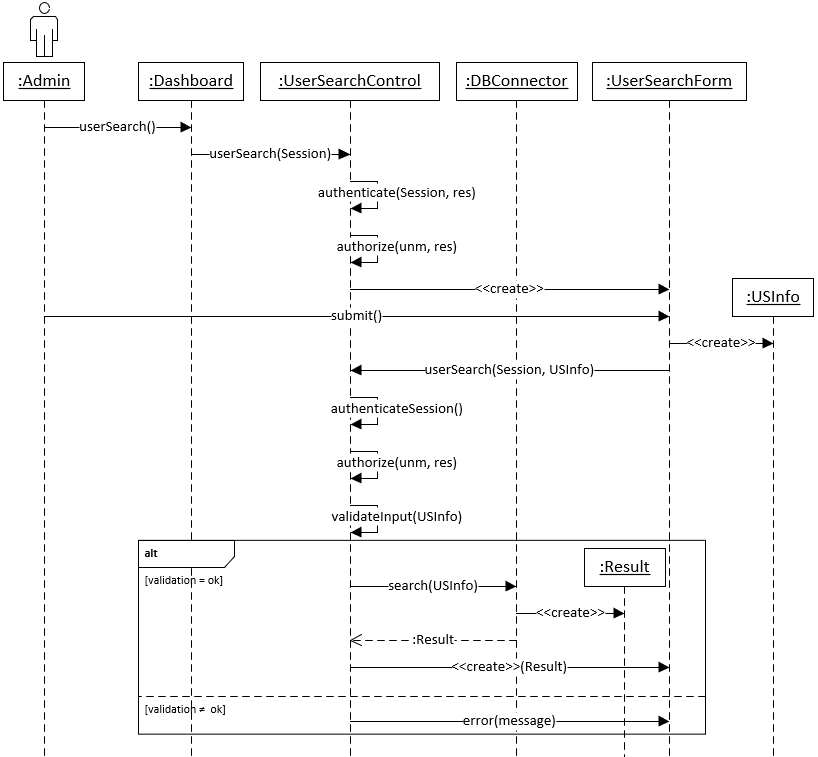


Figure 3.5: Interaction for user search use case (for valid admin session)

## EditAccount Use Case

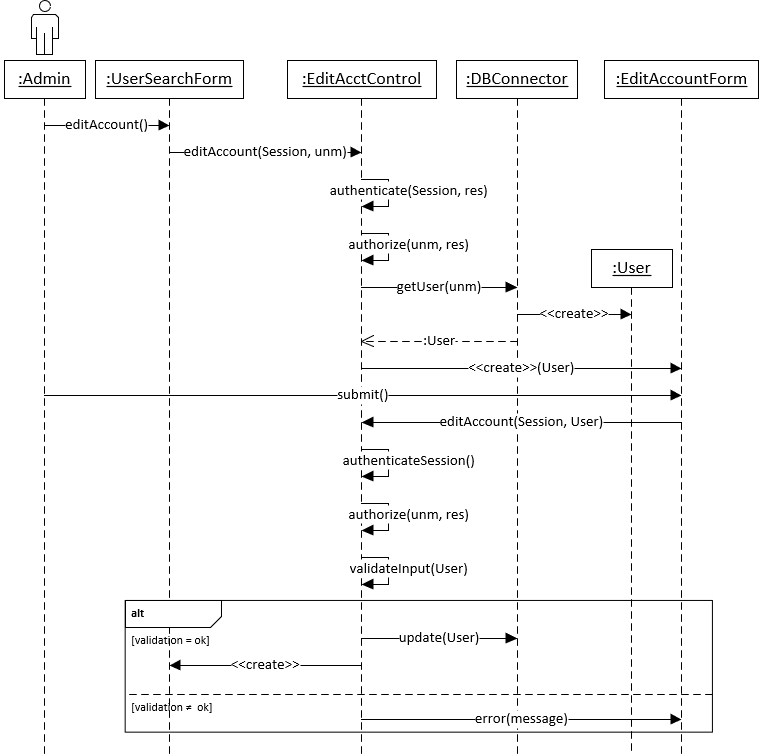


Figure 3.6: Interaction for edit account use case (for valid admin session)

## CourseSearch Use Case

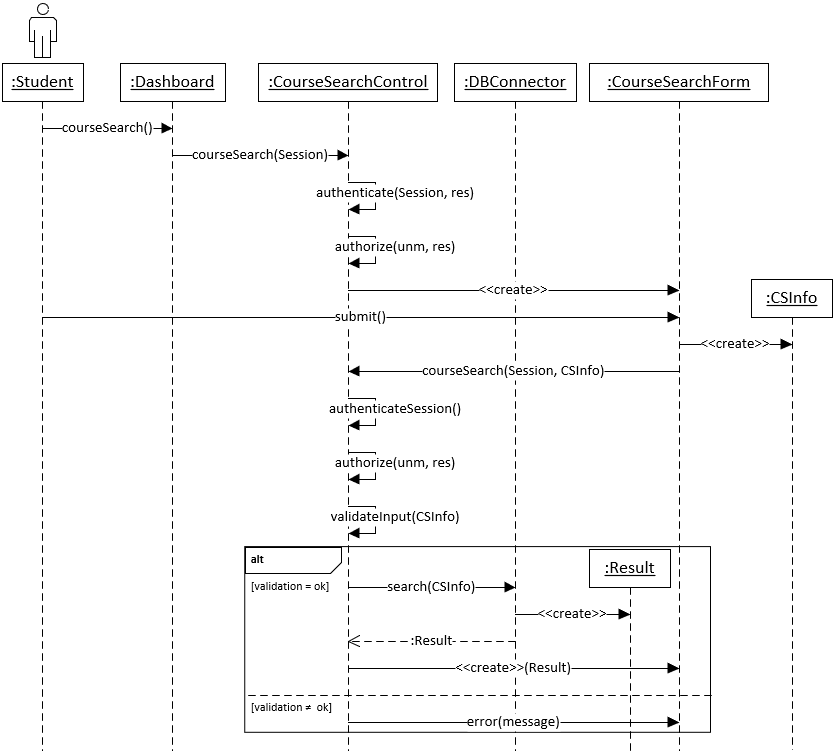


Figure 3.7: Interaction for course search use case (for valid student session)

## CourseEnroll Use Case

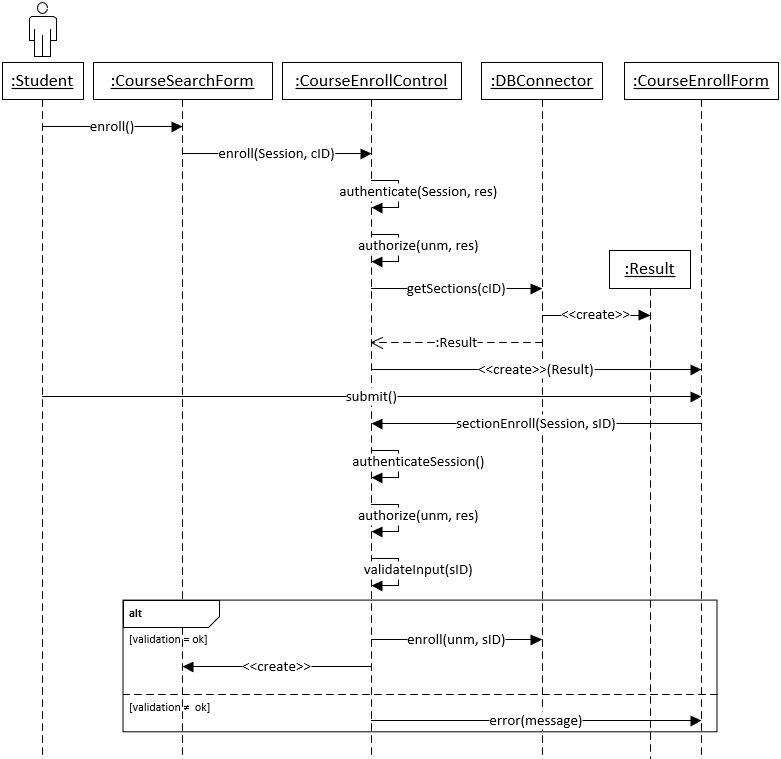


Figure 3.8: Interaction for course enroll use case (for valid student session)

## EnterGrade Use Case

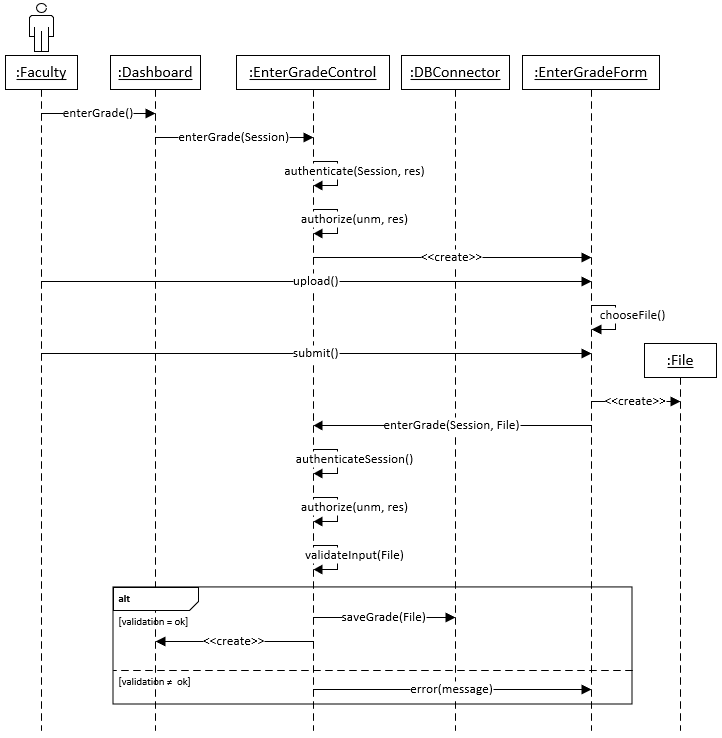


Figure 3.9: Interaction for enter grade use case (for valid faculty session)

## ForgotPassword Use Case

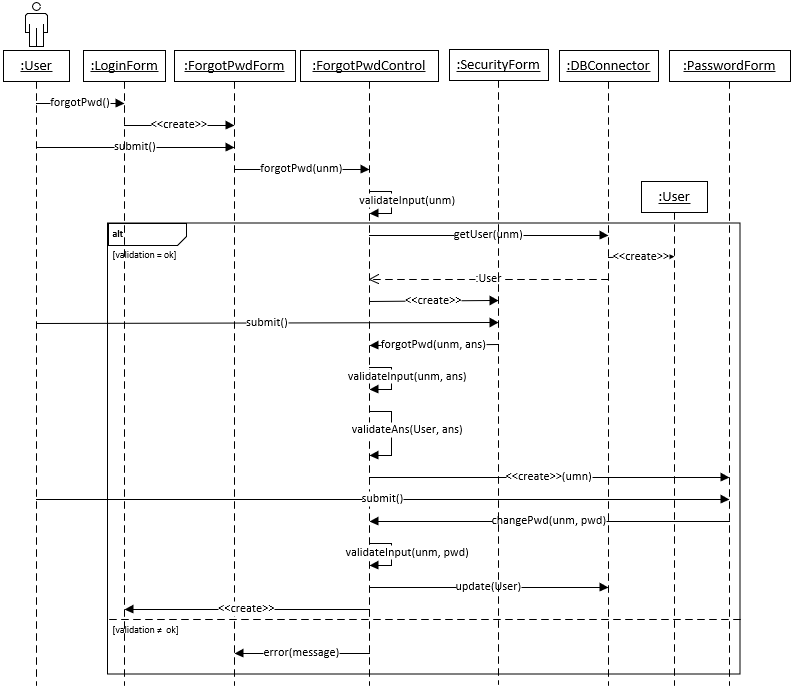


Figure 3.10: Interaction of forgot password use case

# Class Diagrams

## Control Subsystem

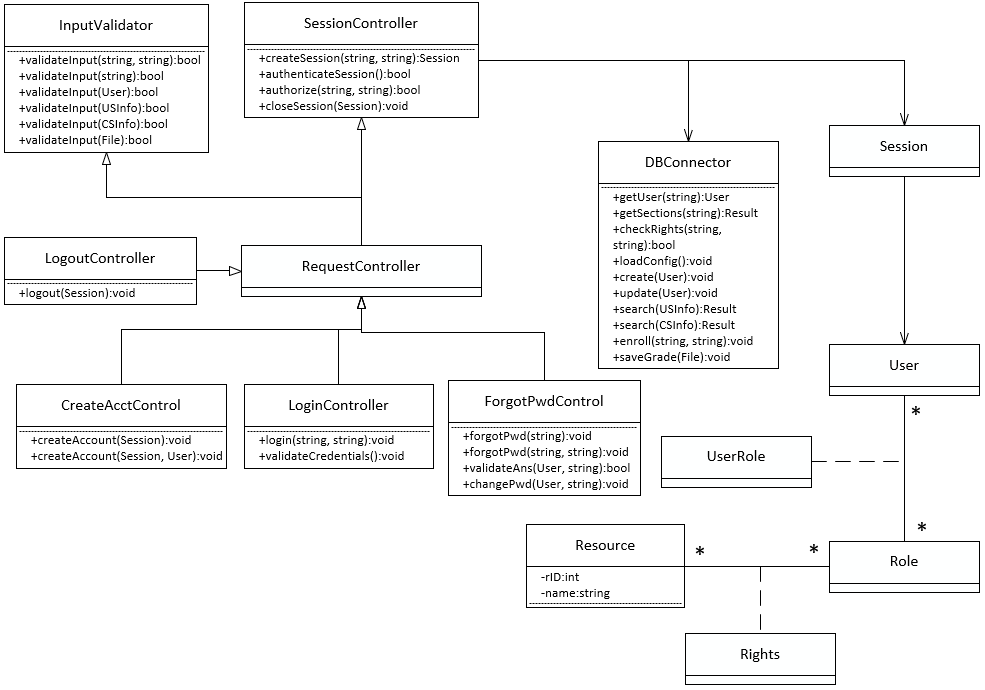


Figure 4.1: Object model for controllers

# Dataflow Diagrams

## Login Controller

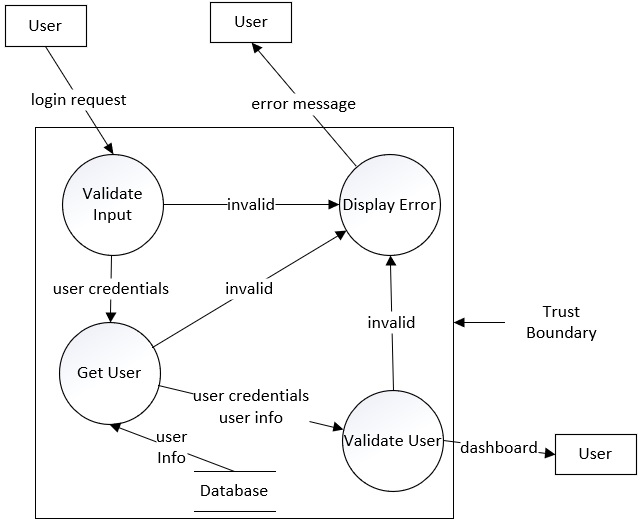


Figure 5.1: Data flow for login controller

## Logout Controller

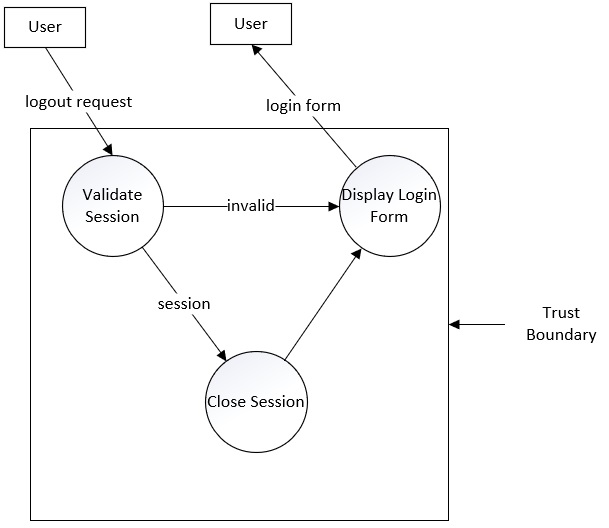


Figure 5.2: Data flow for logout controller

## Startup Controller

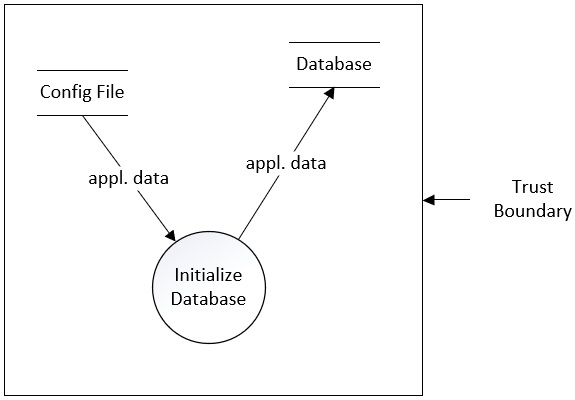


Figure 5.3: Data flow for startup controller

## CreateAccount Controller

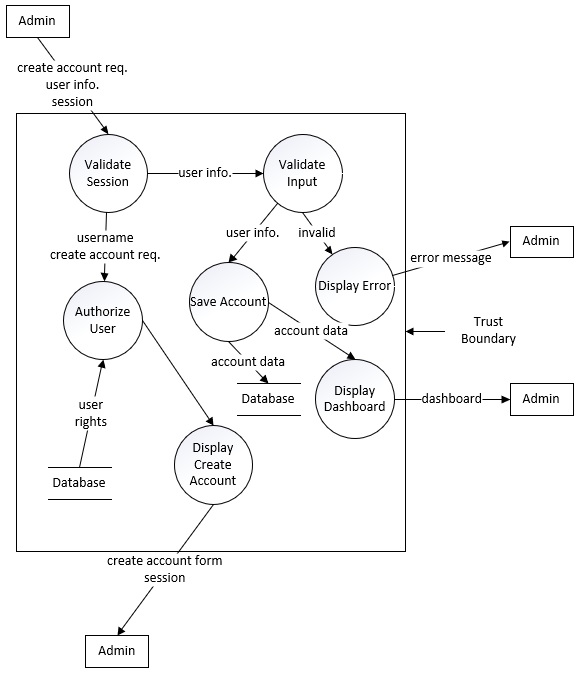


Figure 5.4: Data flow for create account controller

# Data Model

## ER Diagram

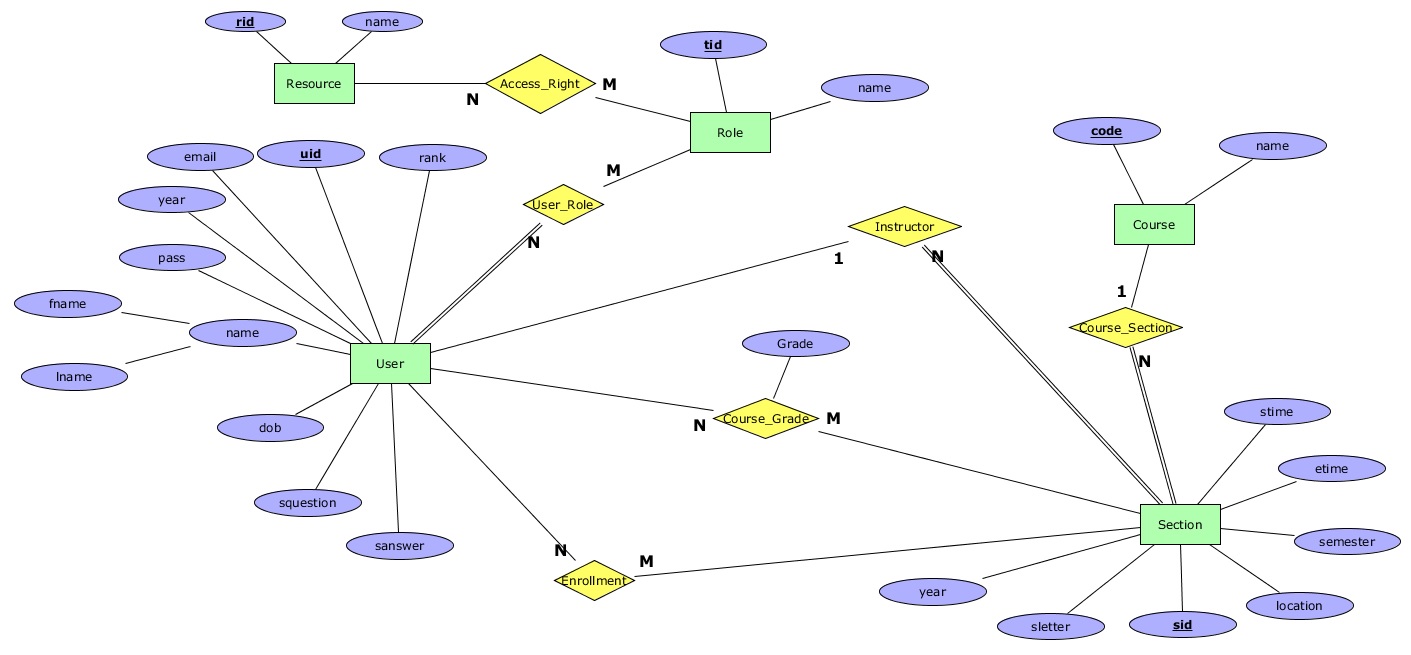


Figure 6.1: ER Diagram

# User Interface Mockups

## Login



Figure 7.1: Login

## Dashboard



Figure 7.2: Dashboard: StudentDashboard



Figure 7.3: Dashboard: FacultyDashboard



Figure 7.4: Dashboard: AdminDashboard

## CourseSearch



Figure 7.5: Course Search Form

## CourseEnroll



Figure 7.6: Course Enroll Form

## EnterGrades



Figure 7.7: EnterGrades

## CreateAccount



Figure 7.8: CreateAccount: CreateStudentAccount



Figure 7.9: CreateAccount: CreateFacultyAccount

## UserSearch



Figure 7.10: UserSearchForm



Figure 7.11: UserSearch: UserSearchFaculty

## EditAccount



Figure 7.12: EditAccount

## ForgotPassword



Figure 7.13: ForgotPassword



Figure 7.14: ForgotPasswordSecurityQuestion



Figure 7.15: ForgotPasswordChangePassword