Software Requirements Specification Document

Course Planner

Version 4

April 6th, 2022

Sivani Sayani, Mara Dimitroff, Usama Nadeem,
Awais Nadeem, Tom Freier

Submitted in partial fulfillment of the requirements of

IT 326 Software Engineering

Table of Contents

Revision History

Date	Description	Author	Comments
2/9/22	Version 1	Developers of CP	First Revision
3/2/22	Version 2	Developers of CP	Activity Diagrams
3/23/22	Version 3	Developers of CP	Class and SD
4/6/22	Version 4	Developers of CP	Use-case development

1. Introduction

Course planner aims to improve students' course scheduling. Users may choose courses based on skills, difficulty, and help to establish a better schedule based on student experiences. Students can create accounts, contribute to course skills/difficulty/textbook information, and plan their course schedule. A student driven course planner gives students autonomy over their success (better than blind administrative systems).

1.1 Purpose

The Course Planner allows students to plan their course schedule through a system that provides visual feedback of course options. The intended audience is any students studying at the School of IT at Illinois State University.

1.2 Scope

The Course Planner (C.P.) will be a website available for any students to use. Users will have the capability to track their scheduled courses by creating an account. Students will be allowed to view courses available at Illinois State University. A user will be able to share his/her course insights and view others' input on select courses.

The objective of the application is to improve overall student success. By receiving a visual representation of course availability and course insights, not only are students actively aware of academic offerings but are also saving costs in the process. The C.P. will increase students' autonomy, awareness, and knowledge of the opportunities available to them.

The C.P. merely serves for planning purposes. It will not be able to grant the services of the University's registrar's office such as enrolling/dropping courses, granting overrides, etc.

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 course card: A visual card (object in memory) displaying course information, suggestions (skills, difficulty, etc.), and credits.

1.3.2 course bank: Contains a collection of all available courses divided by department. It is unique for each school

1.3.3 planned: The courses planned by the student for future

1.3.4 enrolled: Includes the courses that the student is currently enrolled in

1.3.5 taken: Includes the courses and credits attained by the student.

1.3.6 C.P. : Stands for Course Planner, the name of the product.

1.4 Overview

The following document contains general descriptions that affect the product and its requirements. Requirements will be outlined in later sections, followed by the functional requirements of the C.P. Specific features of the software will be outlined, followed by non-functional requirements. Design constraints will be specified at the end.

2. General Description

Course Planner will allow users to create an account, create their courses plan, and add their desired courses from a predefined course bank to their course plan. Additionally, users may choose to rate courses based on skills, difficulty, and help to establish a better schedule based on student experiences. The benefits of our application include student success (skill insights, difficulty ratings), cost-saving (Textbook use), increased course availability, and increased student autonomy, awareness, and knowledge.

2.1 Product Perspective

For many students, Rate My Professor is used to plan courses for the upcoming year. However, the website does not provide meaningful information for planning classes that fit one's specific skill set and knowledge base. Furthermore, it is too broad and does not show skill-based course requirements. The C.P. is more tailored towards students at the School of IT at Illinois State University, thereby, providing a more thorough experience.

2.2 User Characteristics

A student enrolled in the School of IT at Illinois State University.

2.3 System Environment

A server hosting a Flask application exposing port 80 & 443 listening for HTTP/S fulfilled via webpages. The backend of the website will be supported by Python, an interpreted programming language. Additionally, the front-end UI will be handled via HTML, CSS, and JavaScript. We will be using SQLITE3 as our relation database which will support our data storage. Finally, the website will be hosted on PythonAnyWhere.com - an external hosting platform.

2.4 General Constraints

Items that limit the developer's options for designing the system include the following:

a) Webservers not available to host (PythonAnywhere.com is a freely available online hosting platform – since freeware we do not control access to resources, reliability, scale, security, etc.)

b) Restrictive port access (PythonAnywhere might deny our domain request).

c) Inability to scale (not enough resources)

d) Support limited number of users (limited hardware)

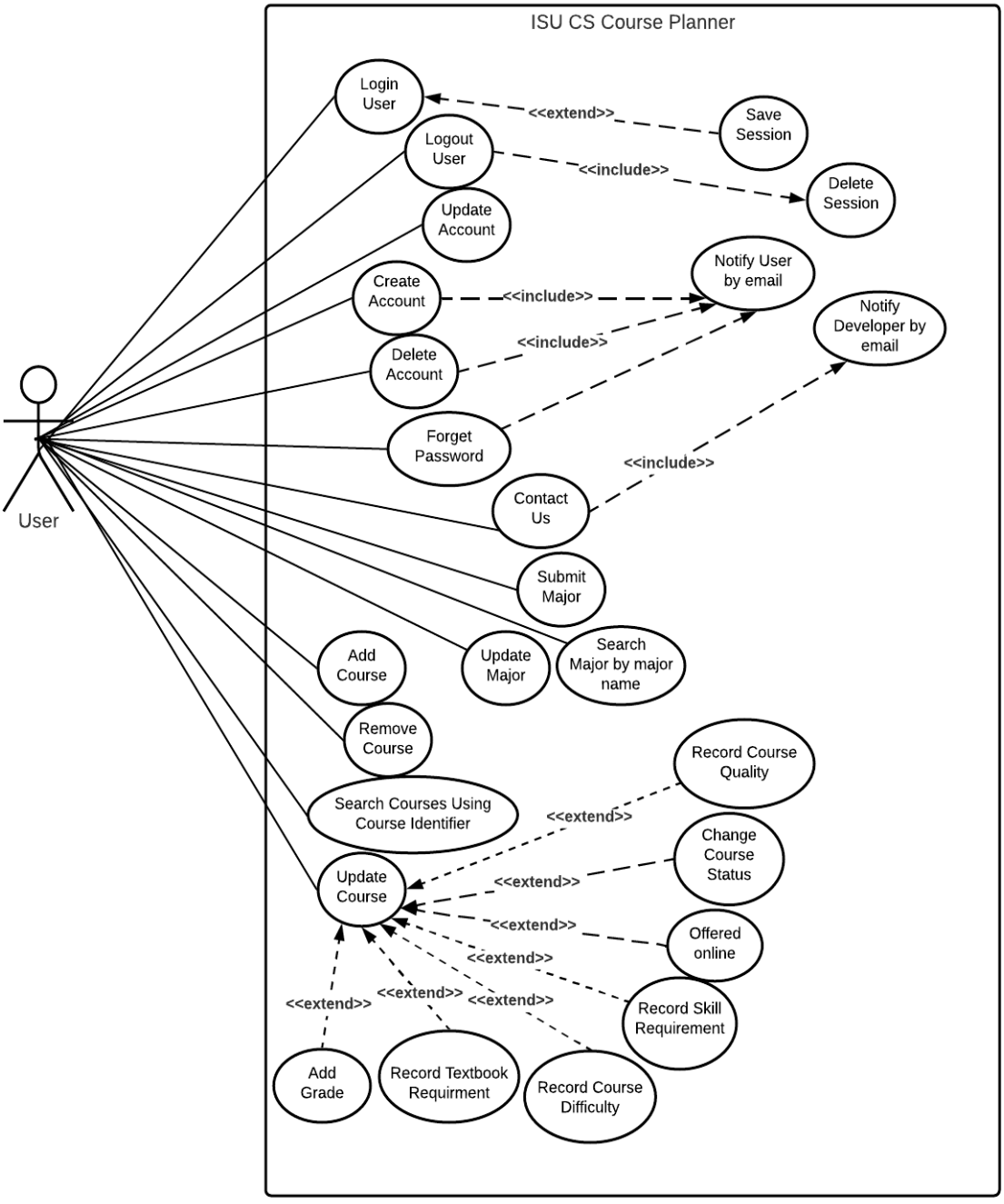
e) Inability to acquire a domain/IP (hosted locally on local network for example then).

2.5 Assumptions and Dependencies

Students should be aware of the registration process and course protocols or policies: This includes being aware of what prerequisites mean and other course/graduation requirements.

3. Specific Requirements

3.1 Functional Requirements


ISU CS Course Planner
User
Login User
Logout User
Update Account
Create Account
Delete Account
Forget Password
Contact Us
Submit Major
Update Major
Search Major by major name
Add Course
Remove Course
Search Courses Using Course Identifier
Update Course
Add Grade
Record Textbook Requirment
Record Course Difficulty
Record Skill Requirement
Offered online
Change Course Status
Record Course Quality
Save Session
Delete Session
Notify User by email
Notify Developer by email
<<extend>>
<<include>>

3.1.1 Create Account

a. Description – Creates an official account for our application: Here the user can sign up and create an account by giving the requested email address, password, and name.

b. Actor(s) – User

c. Trigger – User selects signup on homepage, provides requested information, and clicks sign-up.

d. Conditions

   I. Pre – User input valid and the account should not pre-exist within the database.

   II. Post – Account record added to database

e. Exceptions –

   I. If account could not be created due to an error in the process and the user is notified to try creating again and the page is loaded again.

   II. If the username field or password field were left empty by the user, the user is notified to fill all appropriate login credentials and the page is loaded again.

   III. If account exists under email, refuse account creation and redirect to login page.

3.1.2 Delete Account

a. Description – Removes a user's account from our application: Here the user must be logged in and provide the necessary authentication for this process.

b. Actor(s) – User

c. Trigger – User selects settings page, choses to select delete account.

d. Conditions

   I. Pre – The account must pre-exist within the database. User is logged in.

   II. Post – Account record deleted from the database.

e. Exceptions – If account could not be successfully deleted due to an error in the process and the user is notified to try again.

3.1.3 Update Account

a. Description – Updates user account information, such as: first name and last name.

b. Actor(s) – User

c. Trigger – User clicks on update account (redirects to settings page) and is free to change the first name and last name associated with their account.

d. Conditions

   I. Pre - The account must pre-exist within the database.

   II. Post – Account record updated in the database.

e. Exceptions –

   1. If the account could not be successfully updated due to an error in the process, then update fails, the user is notified, and the settings page is refreshed.

3.1.4 Login User

a. Description – Allows the user to enter the application after providing login information to their respective account.

b. Actor(s) – User

c. Trigger – User provides login credentials and hits login button on the login page.

d. Conditions

I. Pre – Existing user record in the database, available and accessible by the system.

II. Post – System is provided student id of the user and fetches/displays all relevant information on the redirected user dashboard.

e. Exceptions –

I. If login attempt could not be processed due to an error in the process the user is notified to try another login attempt.

II. If account does not exist, then notify user and redirect to signup page

3.1.5 Logout User

a. Description – Sign-outs the user from application and forgets context of the user.

b. Actor(s) – User

c. Trigger – User clicks log out button

d. Conditions

I. Pre – User must have account already logged into the system (I.e., system has context of student ID).

II. Post – System forgets context of the user.

e. Exceptions – If user context could not be forgotten, logout fails, user is notified, and page is refreshed.

3.1.6 Forget Password

a. Description – Allows users, who have forgotten their password, to retrieve it.

b. Actor(s) – User

c. Trigger – User clicks forgot password button

d. Conditions

I. Pre – Existing user record in the database with valid email available and accessible by the system.

II. Post – System sends email with the respective account password.

e. Exceptions –

I. If email does not exist within record, throw error notifying that this email does not exist in system record.

II. If mail could not be sent, notify user email failed to send.

3.1.7 Add Course

a. Description – Adds selected course to user's course list

b. Actor(s) – User

c. Trigger – User clicks the add course button on selected course

d. Conditions

I. Pre – Course must exit in database, user account must exist in database

II. Post – Course must be added to the user's course list and displayed

e. Exceptions – If course could not be successfully added to user's course list, then notify the user process failed & allow the user to try again.

3.1.8 Remove Course

a. Description – Removes selected course from user's course list.

b. Actor(s) – User

c. Trigger – User clicks the remove course button on selected course

d. Conditions

I. Pre – Course must exist in user's course list, user account must exist in database

II. Post – Course must be removed from user's course list

e. Exceptions – If there is an error while trying to remove the course from user's course list , inform the user & allow the user to try again.

3.1.9 Update Course

a. Description – Allows the user to update course information, such as rate the use of the textbook, the class difficulty, required skills, grade, online offerings, etc.

b. Actor(s) – User

c. Trigger – User clicks on class that they desire to update from their schedule

d. Conditions

I. Pre – User account must exist in database, Course must exist in database on in user's schedule

II. Post – Course details are updated in both the database and the user's account

e. Exceptions – If course could not be successfully updated due to an error in the process, then notify user and allow to try again.

3.1.10 Search Courses Using Course Identifier

a. Description – Provides user with a list of available matching requested courses with user provided keywords.

b. Actor(s) – User

c. Trigger – User enters desired course keyboard in search box and hits search.

d. Conditions

I. Pre – User account exists & logged in. The search string should also be valid.

II. Post – Display list/records matching string text/keywords.

e. Exceptions –

I. If it cannot find the desired course in database, show empty results.

II. If search fails, notify user and allow them to try again.

3.1.11 Change course status

a. Description – Allows the user to change the current status of a given course. Available choices: enrolled, taken, or planned.

b. Actor(s) – User

c. Trigger – User selects relevant course, clicks on it, and updates their current status for this course.

d. Conditions

I. Pre – User must be logged in and have course on their schedule.

II. Post – Update the current status of this respective course for user.

e. Exceptions – If course status update failed, then notify user and allow them to try again.

3.1.12 Record textbook requirement

a. Description – Allows user to indicate whether a textbook was required to complete the course regardless of the university requirement of the textbook.

b. Actor(s) – User

c. Trigger – User selects relevant course, clicks on it, and selects textbook checkbox and hit update.

d. Conditions

I. Pre – User must be logged in and have course under their account.

II. Post – Update course information on textbook use.

e. Exceptions – If textbook requirement fails to be recorded, notify the user through a warning.

3.1.13 Record skill requirement

a. Description – Allows user to indicate a skill that may be required to better succeed in the class.

b. Actor(s) – User

c. Trigger – User selects relevant course, clicks on it, and selects relevant skill used.

d. Conditions

I. Pre - User must be logged in and must have course under their account.

II. Post – Update course information on skills required/suggested.

e. Exceptions – If skill not available within selection, user may submit a request. If submitting skill fails, user is notified and prompted to try again.

3.1.14 Record course difficulty

a. Description – Allows the user to rate course difficulty on a scale from 1-5, with one being the lowest difficulty and five being the greatest difficulty.

b. Actor(s) – User

c. Trigger – User selects relevant course, clicks on it, selects the course difficulty scale, and hits submit.

d. Conditions

I. Pre – User must be logged in and have course under their account.

II. Post – Update course information with course difficulty rating.

e. Exceptions – If submitted difficulty fails to be recorded, notify the user through a warning. If user does not provide difficulty selection prompt them to provide required information.

3.1.15 Record course quality

a. Description – Allows the user to rate overall course quality/satisfaction on a scale from 1-5, with one being the poor and five being the great.

b. Actor(s) – User

c. Trigger – User selects relevant course, and records their course quality/satisfaction, and hits submit.

d. Conditions

I. Pre – User must be logged in and have course under their account.

II. Post – Update course information with course quality rating.

e. Exceptions – If submitted quality/satisfaction fails to be recorded, notify the user through a warning. If user does not provide selection, prompt them to provide required information.

3.1.16 Offered online

a. Description – Allows the user to indicate if the respective course is also offered online.

b. Actor(s) – User

c. Trigger – User selects relevant course, then selects checkbox to indicate offered online.

d. Conditions

I. Pre - User must be logged in and have course under their account.

II. Post – Update course information regarding if course is offered online.

e. Exceptions – If the system fails to record user submission, notify user.

3.1.17 Add grade

a. Description – Allows the user to add their respective letter grade for a course. User must select from the following available options: A,B,C,D,F.

b. Actor(s) – User

c. Trigger – User selects relevant course, clicks on it, and adds their respective grade.

d. Conditions

I. Pre User must be logged in and have course under their account.

II. Post – Update course information on average grade for class.

e. Exceptions – If the system fails to record user grade submission, notify user.

3.1.18 Save session

a. Description – Save the user session so user does not need to log in with credentials every time. The system issues a cookie on user's device and pulls that cookie to restore session.

b. Actor(s) – User

c. Trigger – User selects remember me box on login page and presses login button.

d. Conditions

I. Pre – User must have valid account; credentials must be correct.

II. Post – System creates, issues, and stores user session. Cookies used to store user information and can be used to retrieve user context the next time the user visits page.

e. Exceptions – If cannot create cookie, notify user. If user browser denies cookies, respect privacy and continue.

3.1.19 Delete session

a. Description – Clear user session and delete cookie.

b. Actor(s) – User

c. Trigger – User clicks log out button or clears cache/cookies/history of web browser.

d. Conditions

I. Pre – User must have logged in with "Remember me" box checked. User browser must be storing cookies.

II. Post – System deletes record of cookie and forgets all relevant cookie information.

e. Exceptions – If error while removing user's session, set session to None.

3.1.20 Notify user by email

a. Description – Sends email to user provided email.

b. Actor(s) – User

c. Trigger – User submits creates/delete account or forgets password.

d. Conditions

I. Pre – User account exists and actions taken

II. Post – Email notification sent to user inbox.

e. Exceptions – If email fails to send, inform user of failure.

3.1.21 Notify Developer by email

a. Description – Sends email to developers with various requests submitted by users, such as missing course, missing skill, etc.

b. Actor(s) – User

c. Trigger – Submits contact us form.

d. Conditions

I. Pre – User logged in, fills forms, and submits form.

II. Post – Format form and send email to developers.

e. Exceptions – If email fails to send, then show warning message to user.

3.1.22 Search Major by major name

a. Description – User can search for their respective major with keywords.
b. Actor(s) – User
c. Trigger – User provides input and hits search.
d. Conditions
    I. Pre – Account must be created/exist in database
    II. Post – User is displayed major offerings
e. Exceptions – If no such major exists, user is notified. If user enters invalid input, nothing is displayed.

3.1.23 Submit Major

a. Description – User can submit their desired major, so that the database can provide relevant courses.
b. Actor(s) – User
c. Trigger – User selects major from displayed major and hits submit.
d. Conditions
    I. Pre – Must be logged in & select major prior to submission.
    II. Post – Major selection should be recorded.
e. Exceptions – If major submission fails to record user's major, notify user and prompt to try again.

3.1.24 Update Major

a. Description – Allows the user to update/change their respective major.
b. Actor(s) – User
c. Trigger – Clicking on the update major button.
d. Conditions
    I. Pre – Must be logged in and already have selected a major prior to change.
    II. Post – Major update should be recorded, and appropriate course choices should be made available.
e. Exceptions – If the system fails to record major update, notify user via warning message.

3.1.25 Contact us

a. Description – Allows user to submit feedback to the developers, such as course request, skill request, etc.
b. Actor(s) – User
c. Trigger – User selects contact us button.
d. Conditions
    I. Pre – User account must be logged in.
    II. Post – User prompted to fillable & submittable feedback form.
e. Exceptions – If error during submission, notify user via warning message.

3.3 Non-Functional Requirements

3.3.1 Performance
We expect our application to handle multiple users and their individual planners. We also expect the application to be able to respond to the user in a reasonable amount of time.

3.3.2 Reliability
We ensure that the application can handle the various inputs provided by the user: Most of which are ensured in the form of selected field options rather than user typed. Also, we will thoroughly test the application to ensure quality software.

3.3.3 Availability
Preferably, users can always access the application. Additionally, the application should be able to at least support a single user at a time.

3.3.4 Security
The user must provide proper credentials before entering the application. The application also implements some error handling, such as account exits. Finally, we also use proper SQL standards to avoid potential SQL injections.

3.3.5 Maintainability
We plan to follow the SOLID and GRASP design principles to make it easy for software developers to add new code. Additionally, we use Git as our version control system to enable easy and proper collaborative code development.

3.3.6 Portability
The website should be accessible on a standard browser, such as Edge, Chrome, Firefox, etc.

4. Design & Development

4.1 Software Process Model
The process model we have chosen for the development of our application is Scrum. This is a model that will grow our application incrementally through iterations (Sprint 1, 2, and 3) using various phases. We thought this would be a perfect fit for our application as we can divide each sprint and follow the cycle of planning, design, implementation, testing, and evaluation beginning of the start of each. Each iteration/sprint will last 2 weeks long.

4.1.2.1 Plan Description

We plan to meet on Mondays and Wednesdays when class time is allotted to work on the project, from 2pm to 3:15pm. We also meet on Thursdays from 3:30pm to 5:00pm and Saturday in-between the hours of 12pm to 1:30 pm. We will adjust our meeting times throughout the sprints as the work shifts to independent work. Regarding the sprints, at beginning of each sprint, we will start by planning and designing the specifications designated on the sprint schedule. Then, we will work together and individually on our respective tasks. Finally, near the end of the sprint we will test the features and push them to master.

4.1.2.2 Plan Diagram

We meet bi-weekly on Thursdays from 3:30-5 pm for backlog refinement and Saturdays from 12-1:30 pm for sprint planning. Our sprint cycle is 2 weeks, beginning every other Sunday. We have standups scheduled on Mondays and Wednesday at 3:30. In true Agile fashion, we created a scrum board using Trello to keep track of our progress (see Figure 1).

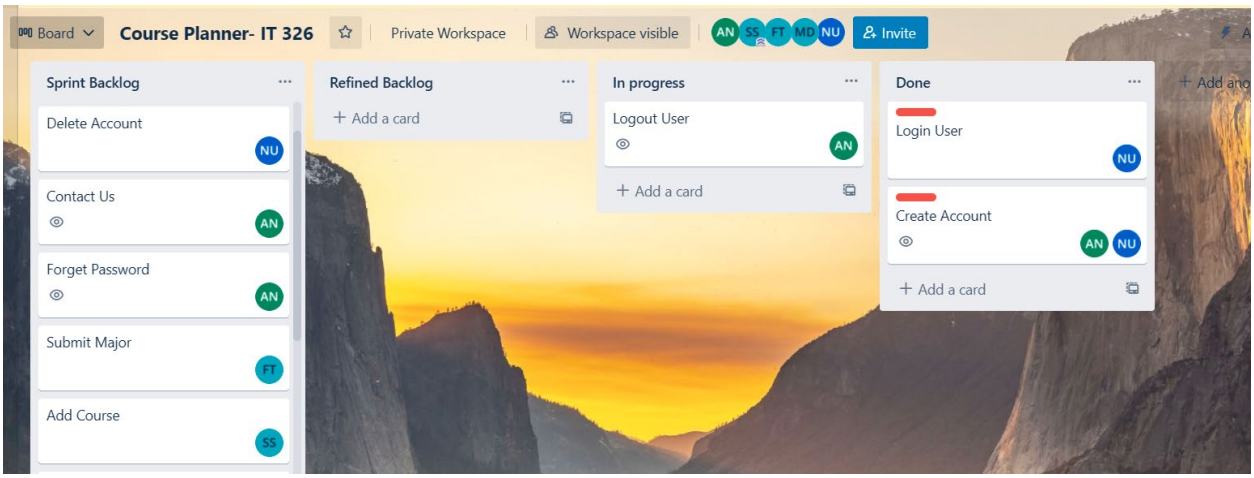


Figure 1: Scrum Board

4.2 Deliverable 1

4.2.1 Description

The first sprint mainly consists of planning future/upcoming work. We plan to diagram the activity and domain class diagrams. We plan to form a Trello board and assign responsibilities, deadlines, and hours. We have also assigned duties, product owner (Mara), scrum master (Savani), lead (Usama), developer (Tom & Awais). Additionally, we plan to get a basic infrastructure in place. This infrastructure consists of a working Flask application, virtual development environments setup, libraries finalized/installed/imported. Our plan is to get at least a multipage website, with basic user account functionality by the end of the sprint.

4.2.2 Satisfied Requirements

This sprint is going to fulfill the requirements 3.1.1 and 3.1.4.

4.2.3 Design

4.2.3.1 Class Diagram

The class diagram showed in Figure 2 captures the system interaction in terms of the User. Here we have the User class which contains the respective attributes along with methods that show case how a user can interact with the various other classes. Most importantly, the Course class, which contains course information, attributes, and course related operation. Additionally, we capture the Major, which contains major related information, attributes, and methods. Finally, to capture the storing/saving of information, we represent a singleton class Database. To capture the interaction between our database object and other objects/operations, we utilize an exposed interface Persistent that allows for queries, saves, updates to/from the database.

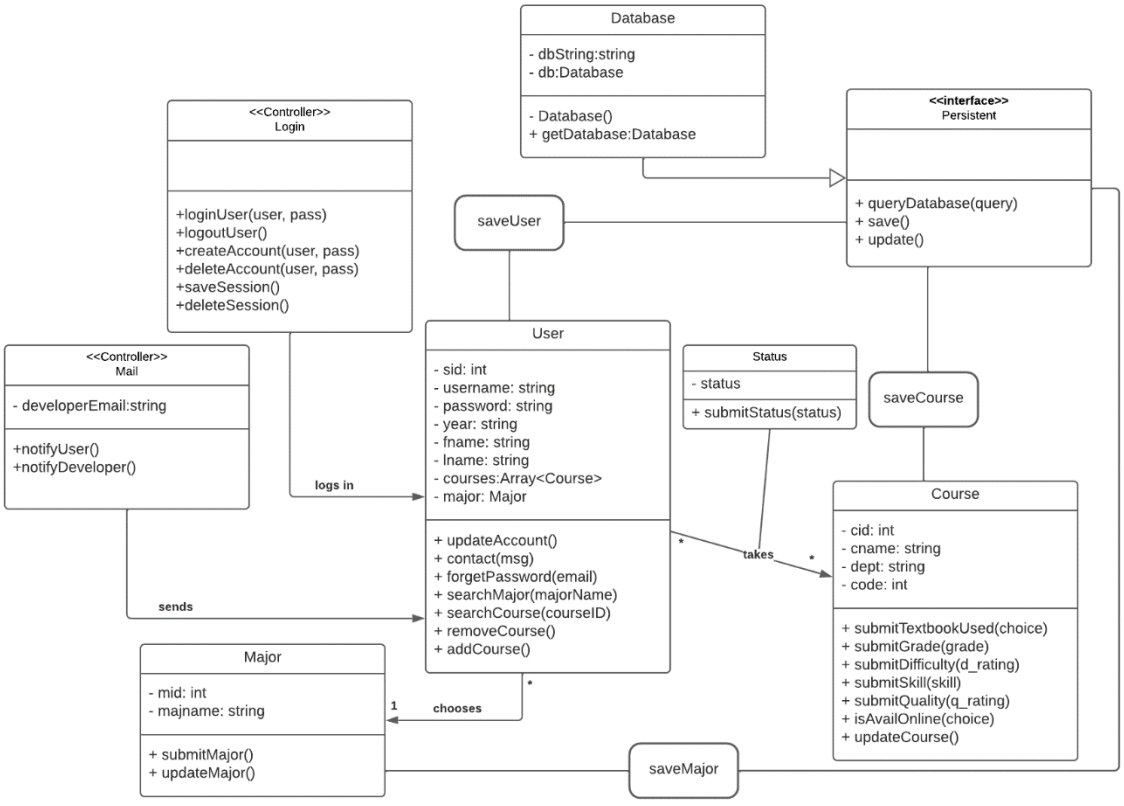


Figure 2: Course Planner Class Diagram

4.2.4 Activity

4.2.4.1 Activity Diagram

The activity diagram shown in Figure 3 explains the activities involved and their flow for logging out a user.

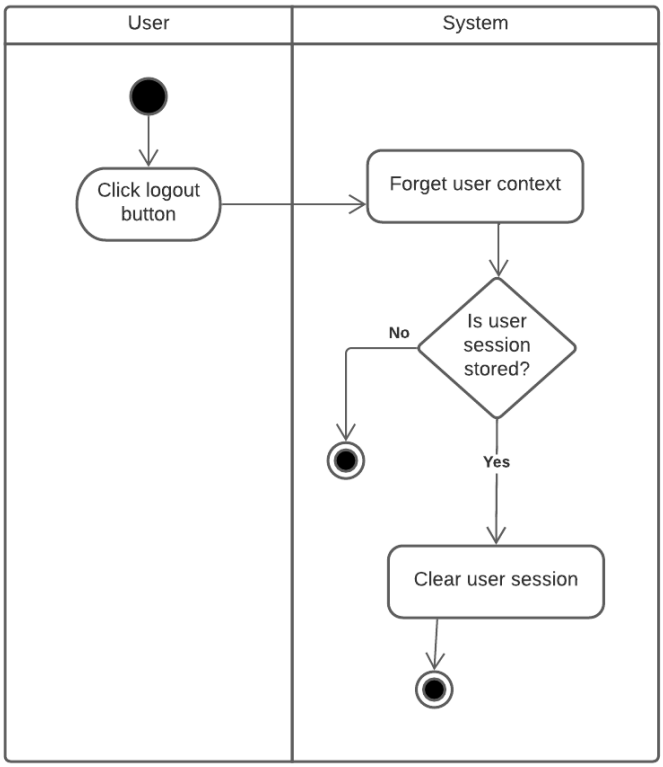


Figure 3: Logout User Activity Diagram

4.2.4.2 Activity Diagram

The activity diagram shown in Figure 4 explains the activities involved and their flow for creating an account.

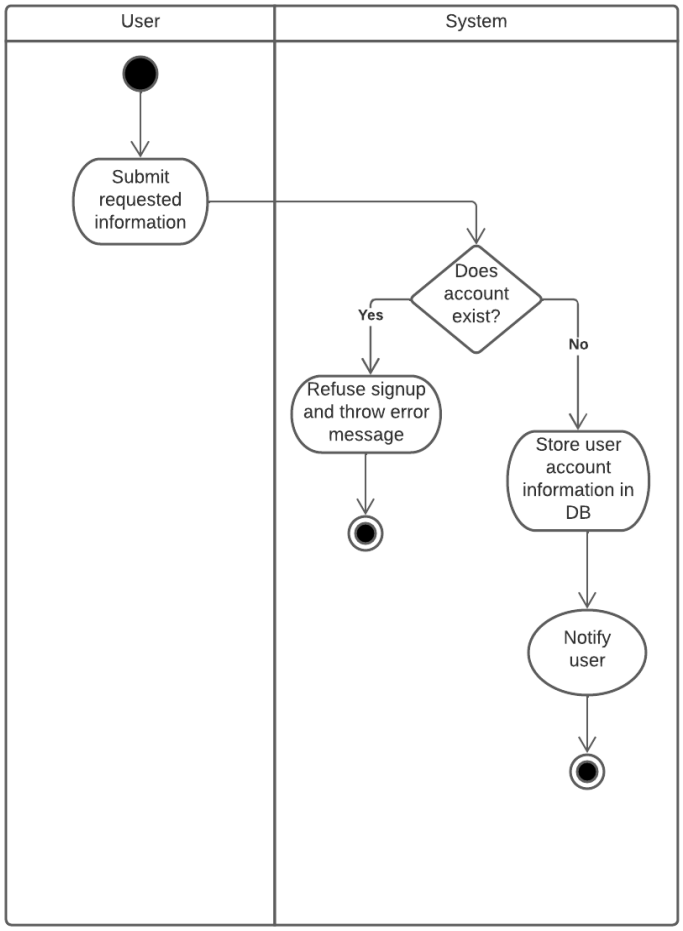


Figure 4: Create Account Activity Diagram

4.2.4.3 Activity Diagram

The activity diagram shown in Figure 5 explains the activities involved and their flow for recording a user's textbook requirement.

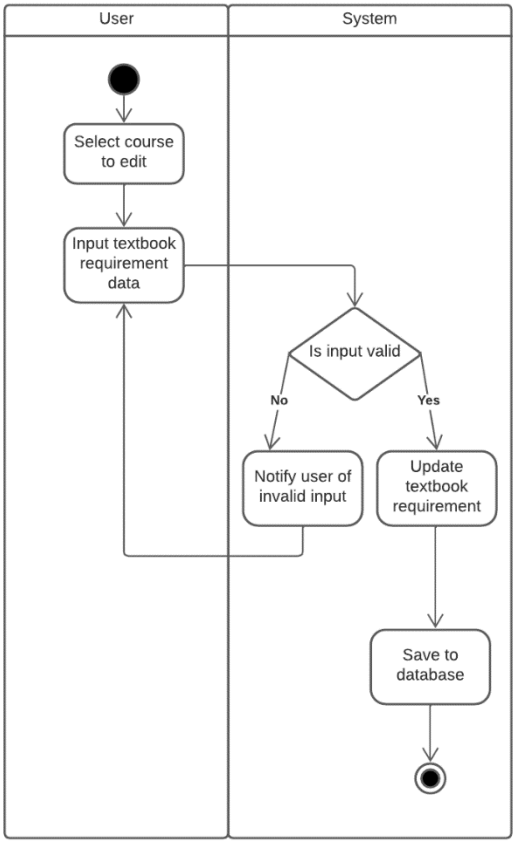


Figure 5: Record Textbook Requirement Activity Diagram

4.2.4.4 Activity Diagram

The activity diagram shown in Figure 6 explains the activities involved and their flow for recording a user's input on course quality

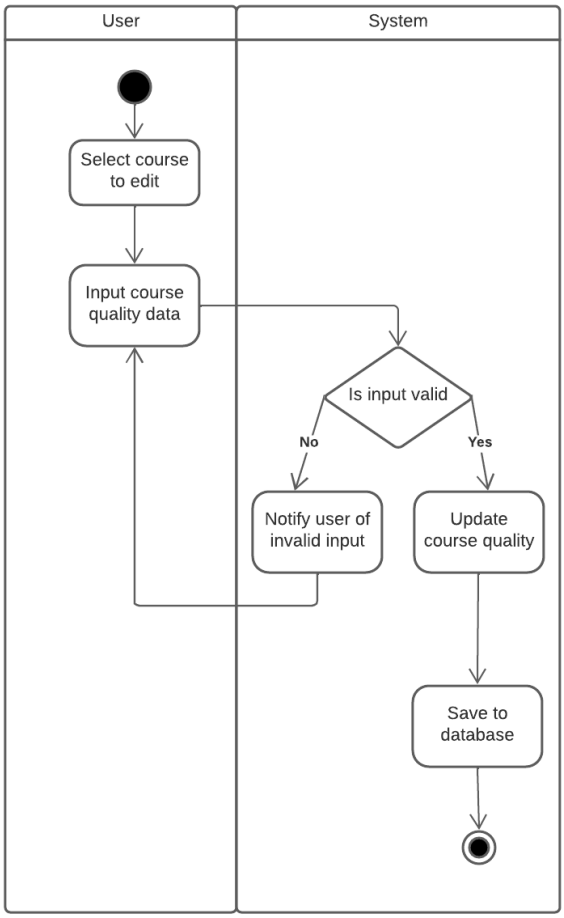


Figure 6: Record course quality Activity Diagram

4.2.4.5 Activity Diagram

The activity diagram shown in Figure 7 explains the activities involved and their flow for a user to delete their account.

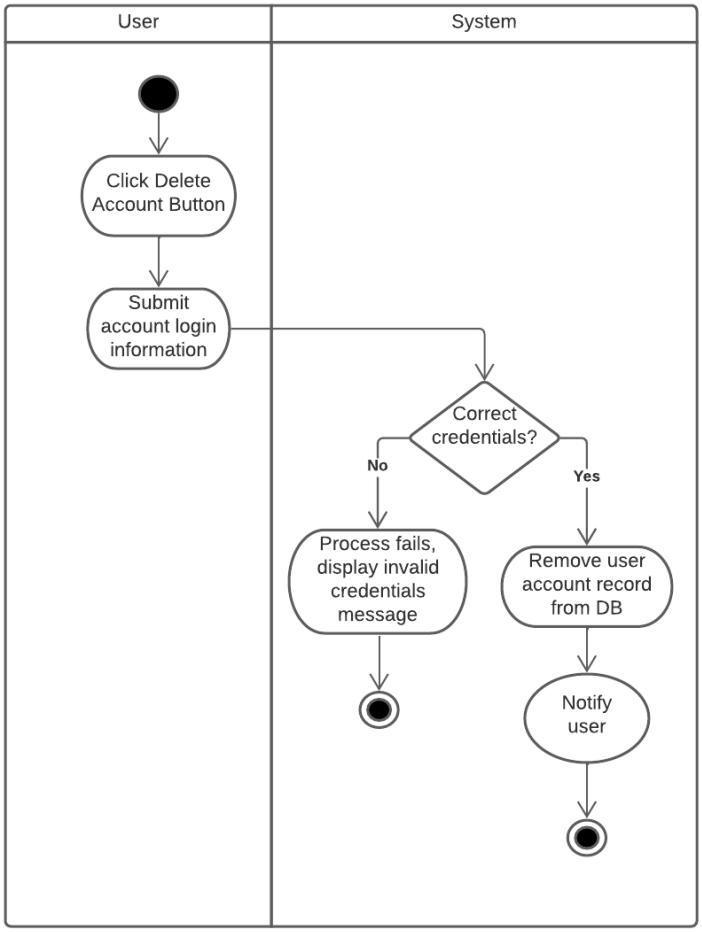


Figure 7: Delete Account Activity Diagram

4.2.4.6 Activity Diagram

The activity diagram shown in Figure 8 explains the activities involved and their flow for a user to use a major's name to search for a major.

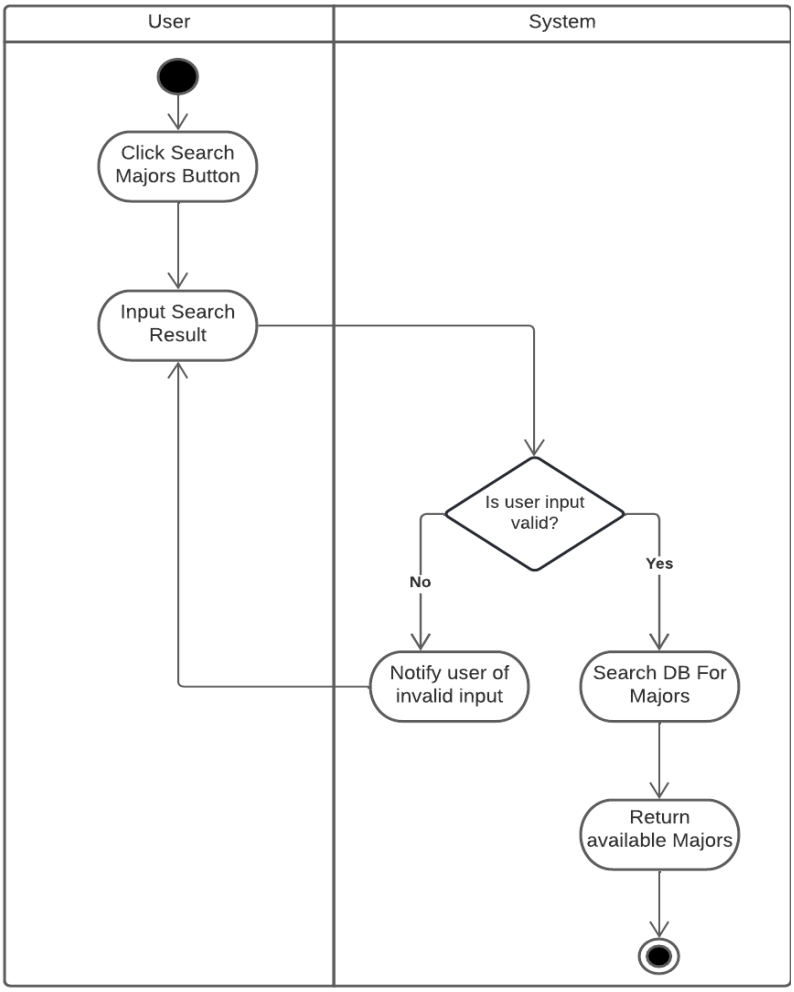


Figure 8: Search for Major using Major Name Activity Diagram

4.2.4.7 Activity Diagram

The following activity diagram shown in Figure 9 depicts the activities involved in the flow of the login of a user.

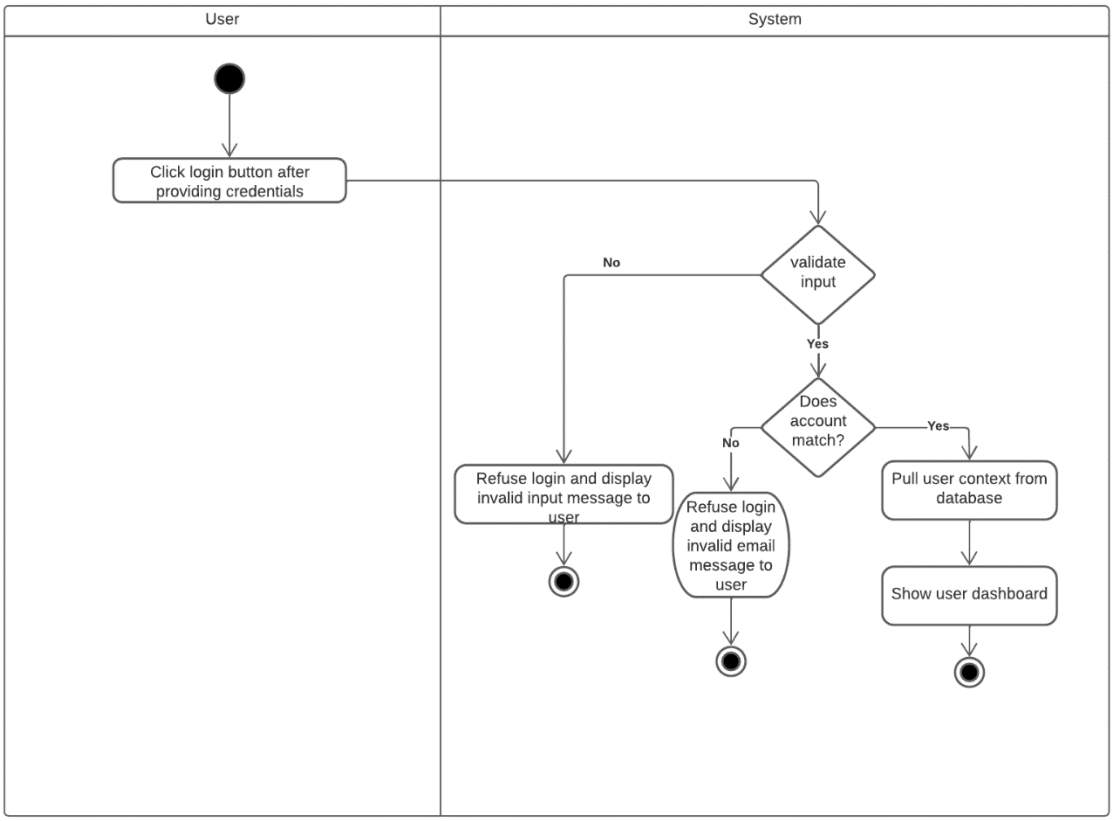


Figure 9: Login User Activity Diagram

4.2.4.8 Activity Diagram

The following activity diagram shown in Figure 10 depicts the activities involved in the flow of the user requesting to be reminded of their password.

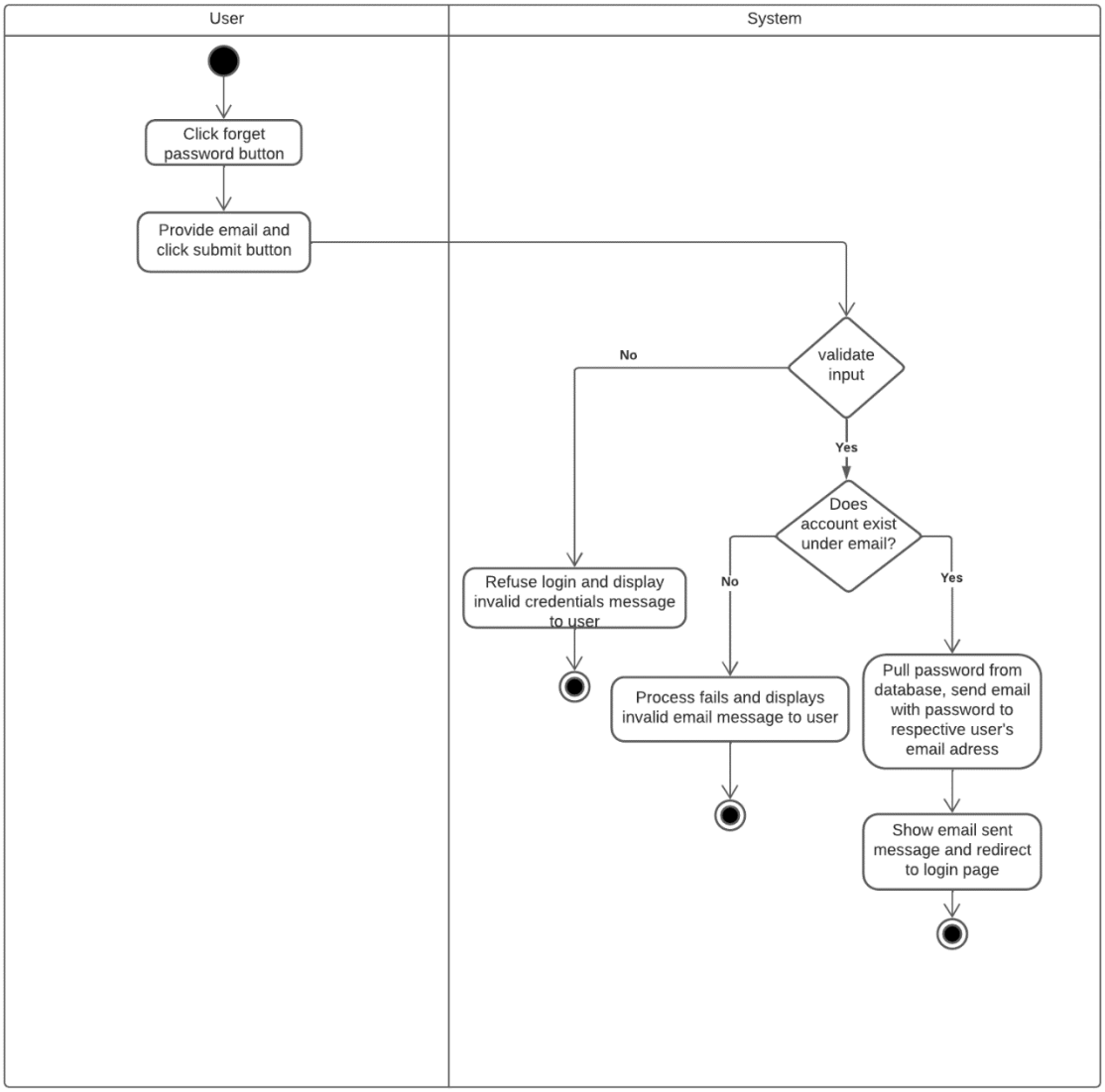


Figure 10: Forgot Password Activity Diagram

4.2.4.9 Activity Diagram

The following activity diagram shown in Figure 11 depicts the activities involved in the flow of the user requesting to add a course to their schedule.

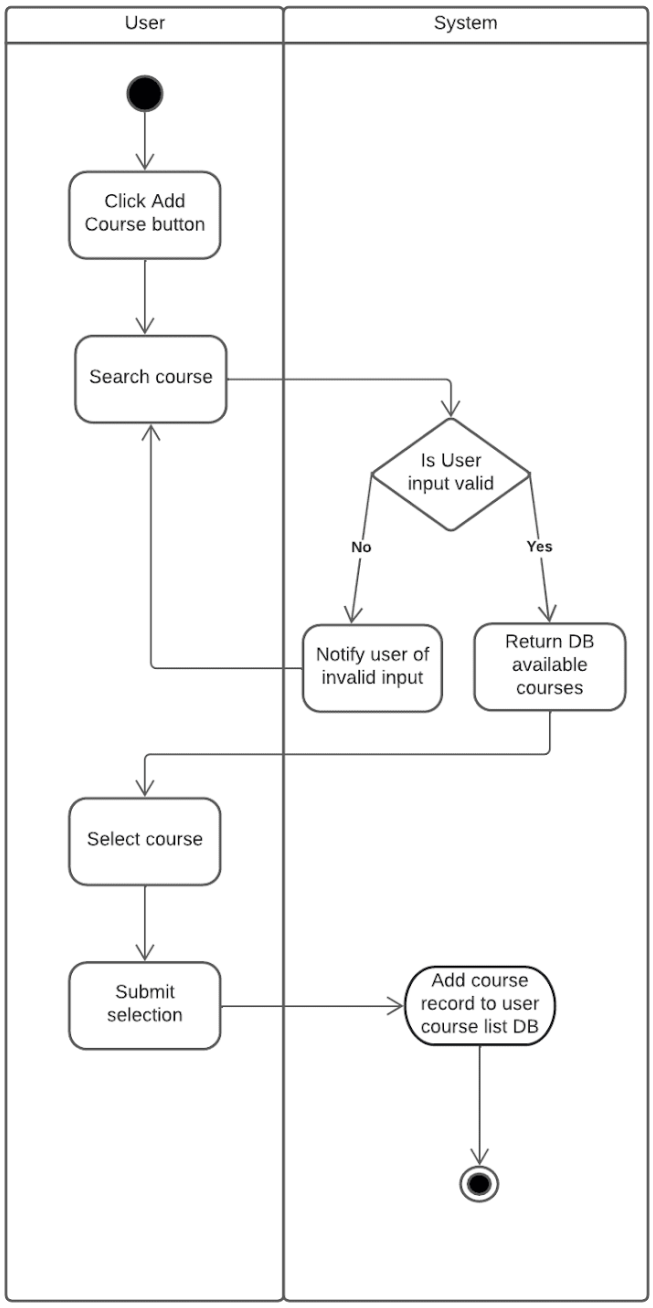


Figure 11: Add course Activity Diagram

4.2.4.10 Activity Diagram

The following activity diagram shown in Figure 12 depicts the activities involved in the flow of the user requesting to remove a course from their schedule.

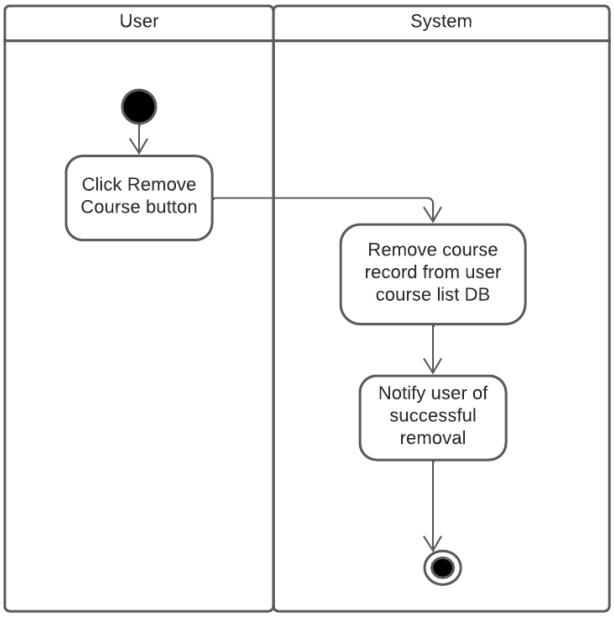


Figure 12: Remove Course Activity Diagram

4.2.5 Development

4.2.5.1 Description

The features we developed in the first sprint was our login and signup page for the application. We believe this sprint was a success and is vital to the later successful implementation of our application.

```python
@ app.route('/signup')  # Signup page
def signup():
    return render_template("signup.html")
```

```
{% extends "layout.html" %}
<!--EXTEND LAYOUT FILE-->

{% block body %}
<!--Code goes between body blocks-->
<link href="{{ url_for('static', filename='css/signin.css') }}" rel="stylesheet">

<body class="text-center">
    <main class="form-signin">
        <form method="post" action="/registered">
            <img class="mb-4" src="../static/images/1982830.jpg" alt="" width="60%" height="60%">
            <h1 class="h3 mb-3 fw-normal">Please sign up</h1>
            <div class="form-floating">
                <input name="email" type="email" class="form-control" id="floatingInput" placeholder="name@example.com">
            </div>
            <div class="form-floating">
                <input name="password" type="password" class="form-control" id="floatingPassword"
                    placeholder="Password">
            </div>
            <!-- <div class="checkbox mb-3">
                <label>
                    <input type="checkbox" value="remember-me"> Remember me
                </label>
            </div> -->
            <button class="w-100 btn btn-lg btn-primary" type="submit">Sign up</button>
        </form>
    </main>
</body>

{% endblock %}
```

```python
@ app.route('/login')  # Login page
def login():
    return render_template("login.html")
```

```
{% extends "layout.html" %}
<!--EXTEND LAYOUT FILE-->

{% block body %}
<!--Code goes between body blocks-->

<link href="{{ url_for('static', filename='css/signin.css') }}" rel="stylesheet">

<body class="text-center">
    <main class="form-signin">
        <form method="post" action="/validate">
            <img class="mb-4" src="../static/images/1982830.jpg" alt="" width="60%" height="60%">
            <h1 class="h3 mb-3 fw-normal">Login</h1>
            <div class="form-floating">
                <input name="email" type="email" class="form-control" id="floatingInput" placeholder="name@example.com">
            </div>
            <div class="form-floating">
                <input name="password" type="password" class="form-control" id="floatingPassword"
                    placeholder="Password">
            </div>
            <div class="form-floating">
                <a href="/forgot">Forogt Password</a>
            </div>
            <div class="checkbox mb-3">
                <label>
                    <input name="checkbox" type="checkbox" value="remember-me"> Remember me
                </label>
            </div>
            <button class="w-100 btn btn-lg btn-primary" type="submit">Sign in</button>
        </form>
    </main>
</body>


{% endblock %}
```

```python
@ app.route('/')
def index():
    global COURSES
    if "email" not in session:  # Check if session doesn't exist
        return render_template("index.html")

    # return render_template('dashboard.html', courses=COURSES)
    return render_template("mainpage.html", courses=COURSES)
```

```
{% extends "layout.html" %}
<!--EXTEND LAYOUT FILE-->

{% block body %}
<!--Code goes between body blocks-->
<!-- Bootstrap references -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
<link href="../static/css/signin.css" rel="stylesheet">

<body class="text-center">

    <main class="form-signin">
        <img class="mb-4" src="../static/images/1982830.jpg" alt="" width="60%" height="60%">
        <h1 class="h3 mb-3 fw-normal">Home Page</h1>
        <a type="submit" class="btn btn-primary" name="login" action="/login" href="/login">Login</a>
        <a type="submit" class="btn btn-primary" name="signup" action="/signup" href="/signup">SignUp</a>
    </main>
</body>

{% endblock %}
```

4.3 Deliverable 2

4.3.1 Description

The second sprint mainly consists of working on the application and creating some more design diagrams. Specifically, we plan to create the sequence diagrams to showcase the object interaction within the system. Also, regarding the application, we plan to focus on creating the database objects needed to store the required information. We also plan to create the basic login, signup, and forgot password logic for our initial screen. With these core component created, we will have the foundation needed to start building the remaining use cases.

4.3.2 Design

4.3.2.1 Class Diagram

The class diagram showed in Figure 13 captures the system interaction in terms of the User. Here we have the User class which contains the respective attributes along with methods that show case how a user can interact with the various other classes. Most importantly, the Course class, which contains course information, attributes, and course related operation. Additionally, we capture the Major, which contains major related information, attributes, and methods. Finally, to capture the storing/saving of information, we represent a singleton class Database. To capture the interaction between our database object and other objects/operations, we utilize an exposed interface Persistent that allows for queries, saves, updates to/from the database. We follow Model View Presenter design pattern with each domain class having a respective UI that accepts interaction from outside the system and a controller class that handles/controls these interactions. The controller is also responsible for making calls to the appropriate exposed database methods via the persistent interface. The domain classes interact intrinsically and with the logic of the controller allows the application to create, read, update, and delete courses, majors, and user accounts.

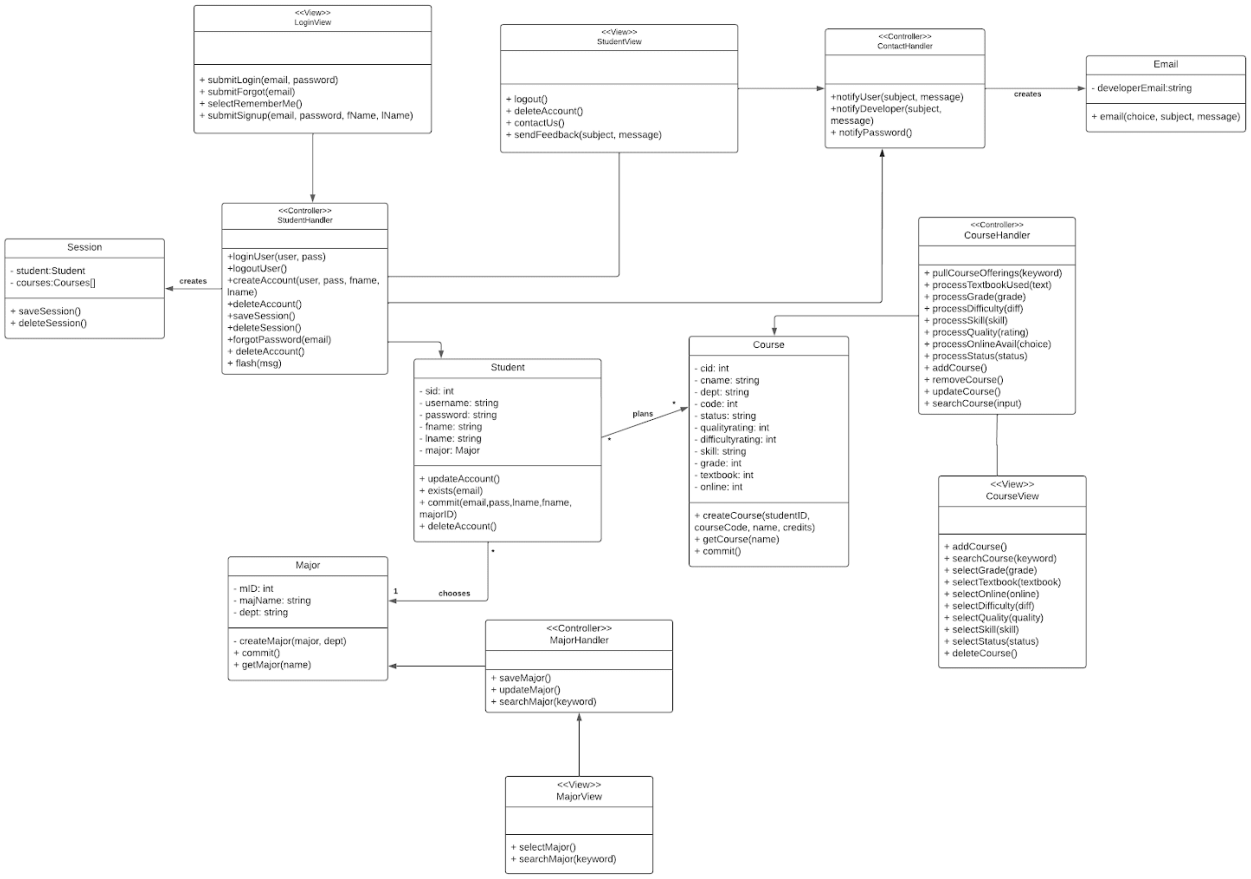


Figure 13: Course Planner Class Diagram

4.3.3 Sequence

4.3.3.1 Sequence Diagram

The sequence diagram shown in Figure 14 illustrates the messages exchanged for requesting a logout operation.

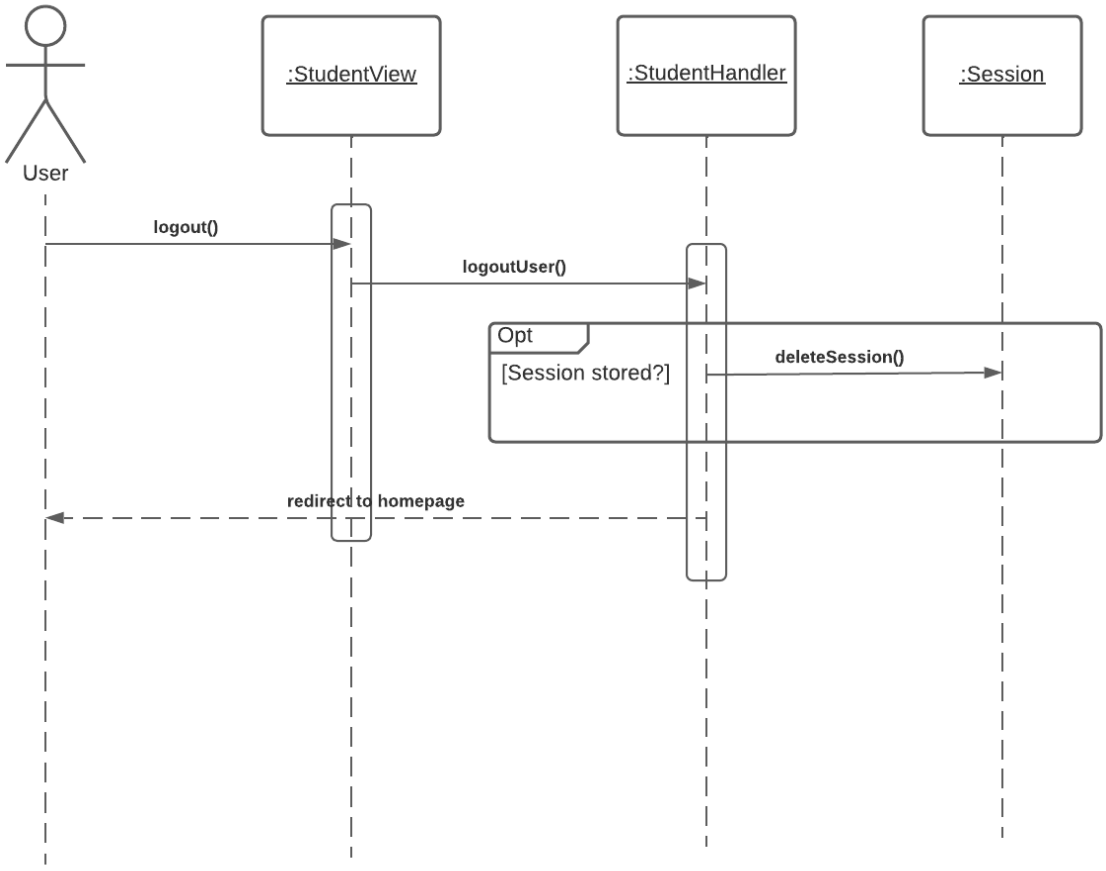


Figure 14: Logout Sequence Diagram

4.3.3.2 Sequence Diagram

The sequence diagram shown in Figure 15 illustrates the messages exchanged for requesting a create account operation.

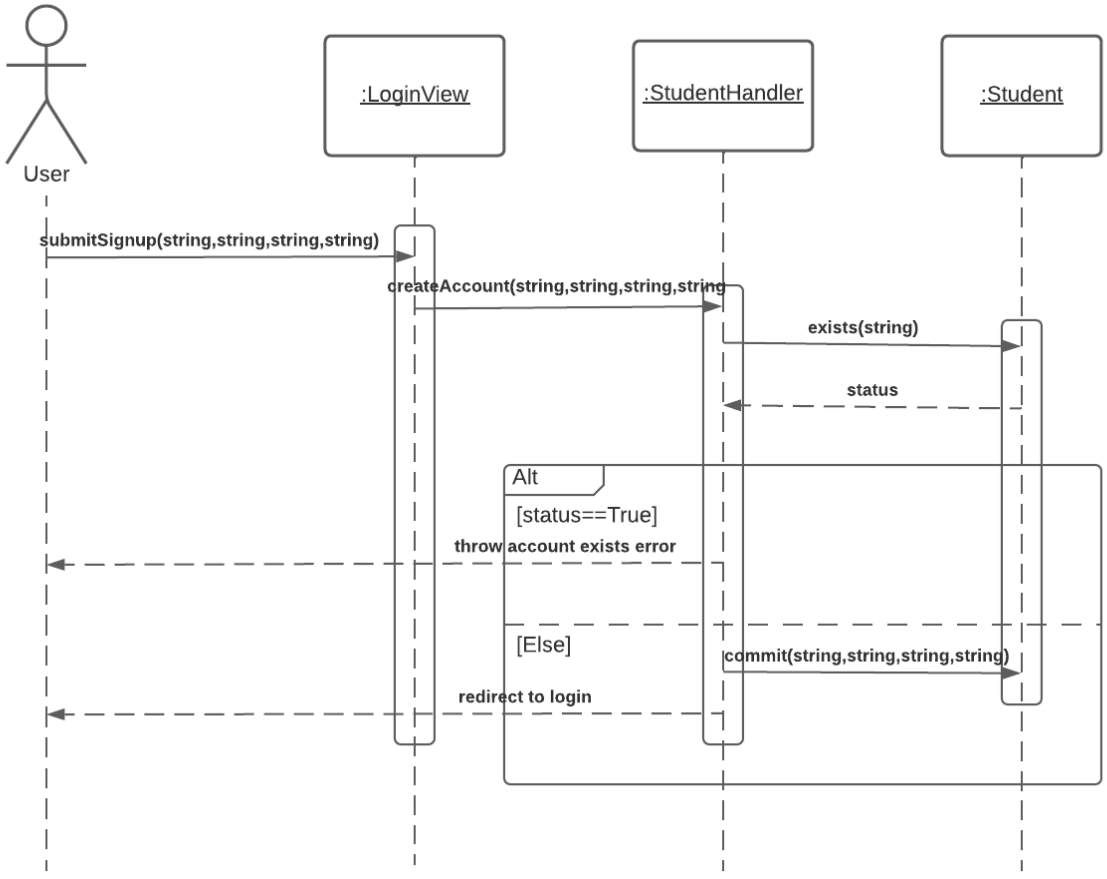


Figure 15: Create Account Sequence Diagram

4.3.3.3 Sequence Diagram

The sequence diagram shown in Figure 16 illustrates the messages exchanged for requesting a login operation.

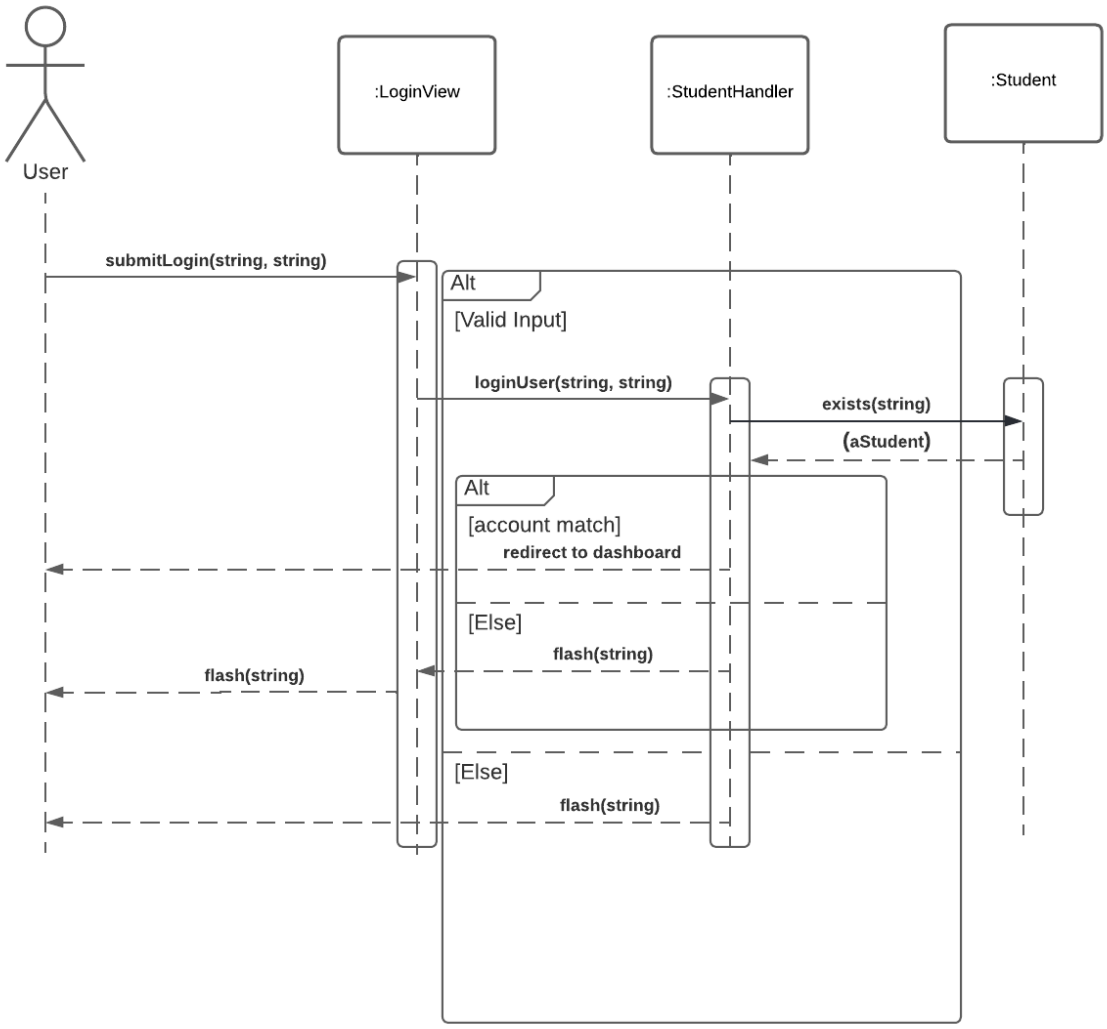


Figure 16: Login Sequence Diagram

4.3.3.4 Sequence Diagram

The sequence diagram shown in Figure 17 illustrates the messages exchanged for requesting a logout operation.

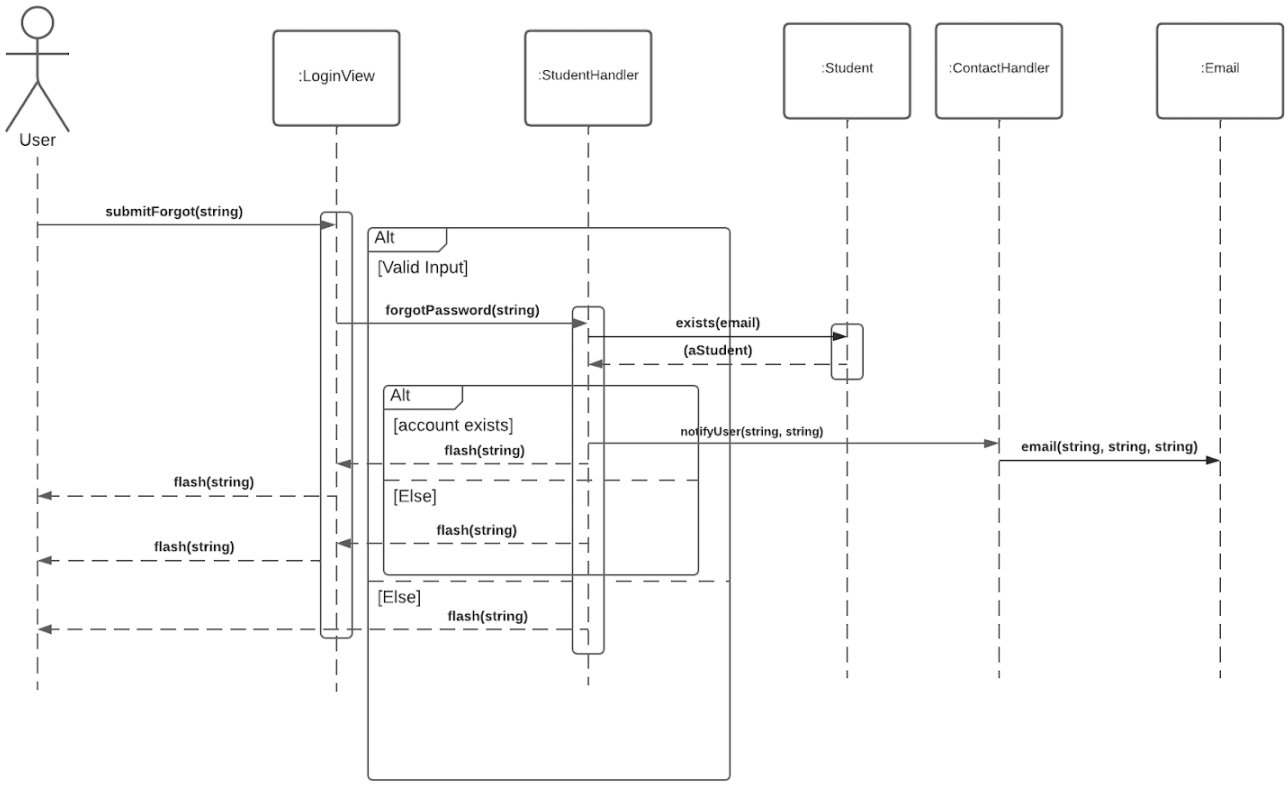


Figure 17: Forgot Password Sequence Diagram

4.3.3.5 Sequence Diagram

The sequence diagram shown in Figure 18 illustrates the messages exchanged for recording a user's course difficulty rating

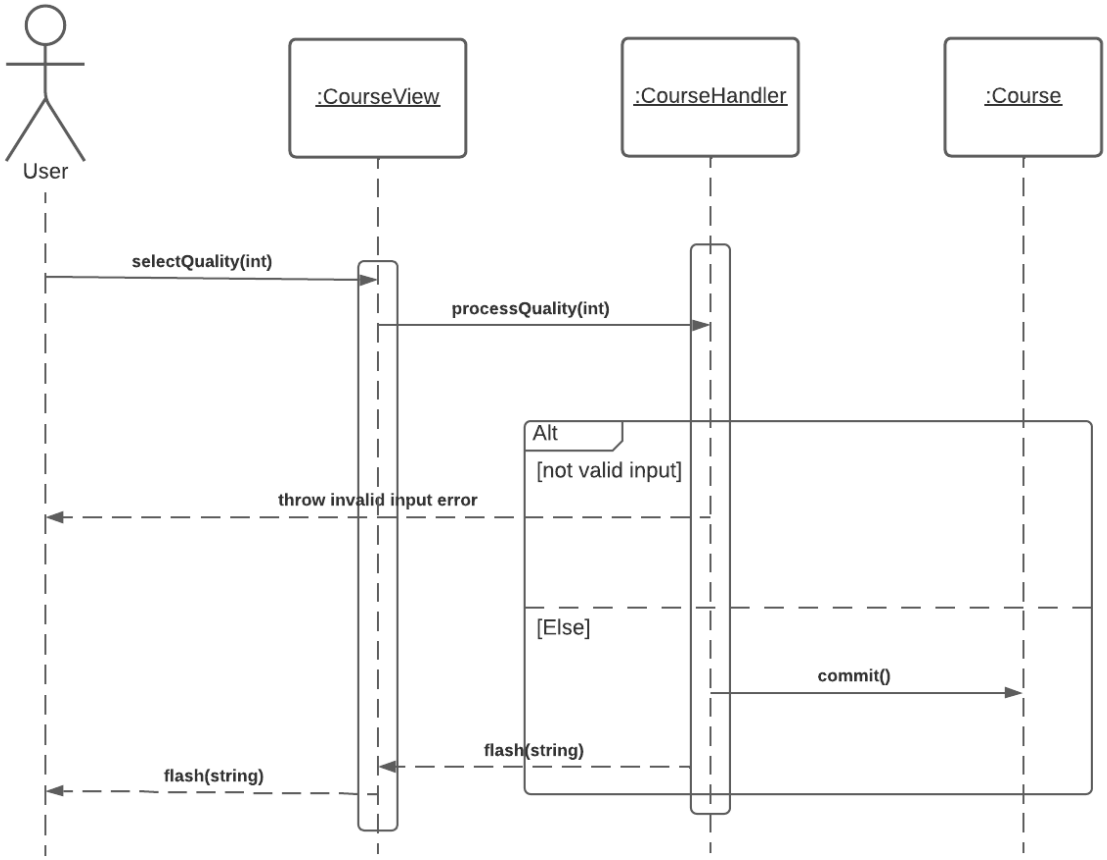


Figure 18: Record Course Difficulty Sequence Diagram

4.3.3.6 Sequence Diagram

The sequence diagram shown in Figure 19 illustrates the messages exchanged for recording a user's use of a textbook

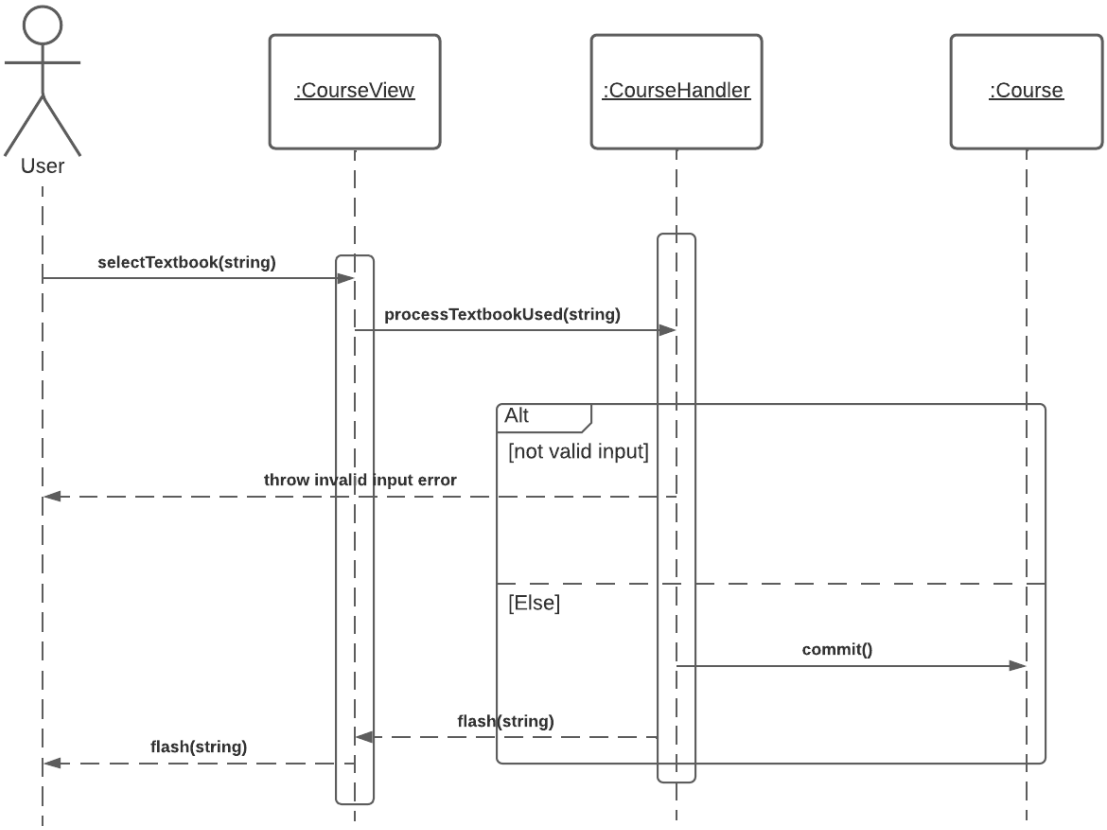


Figure 19: Record Textbook Requirement Sequence Diagram

4.3.3.7 Sequence Diagram

The sequence diagram shown in Figure 20 illustrates the process of searching for a course by name.

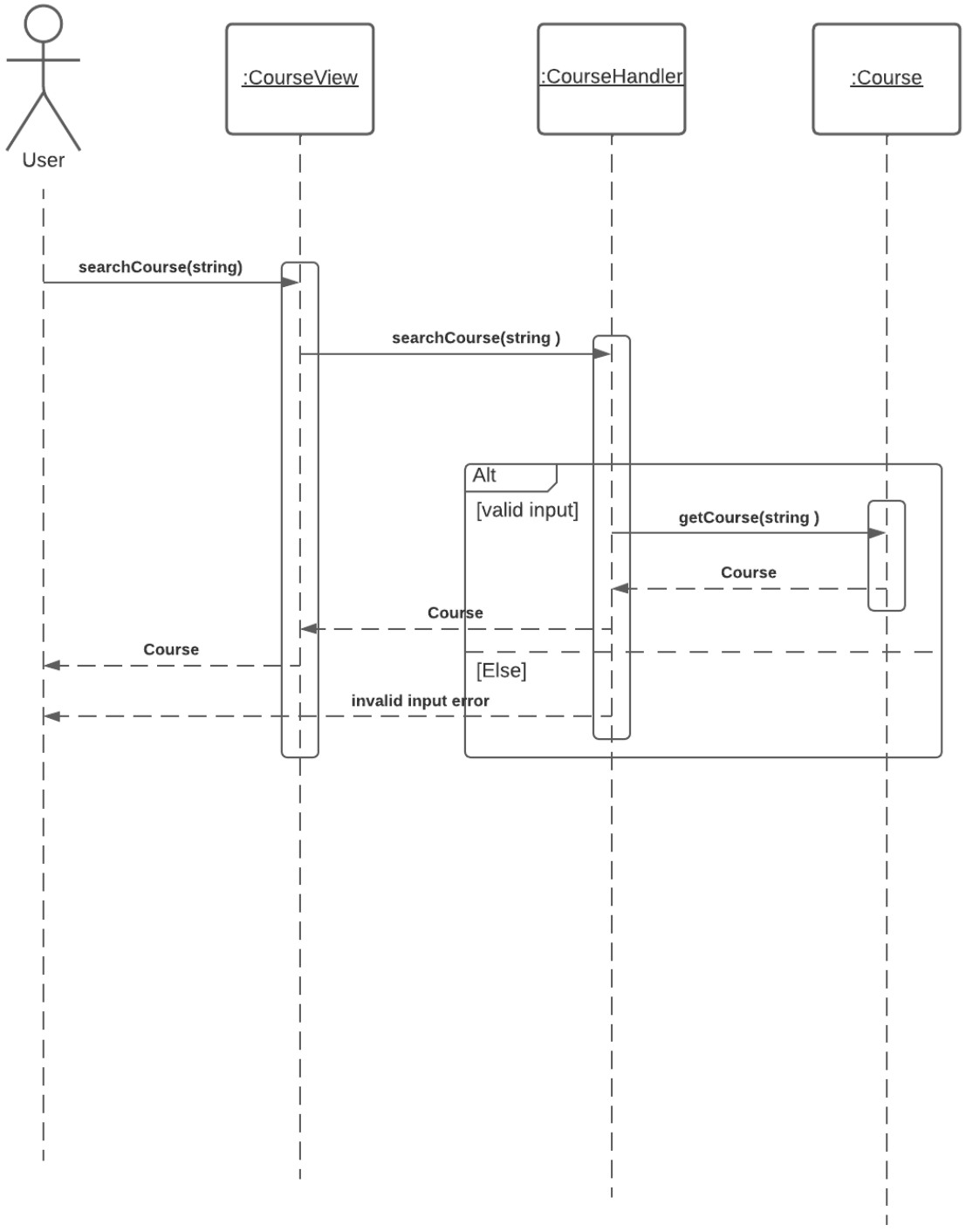


Figure 20: Search Course using course identifier Sequence Diagram

4.3.3.8 Sequence Diagram

The sequence diagram shown in Figure 21 illustrates how a user can submit feedback.

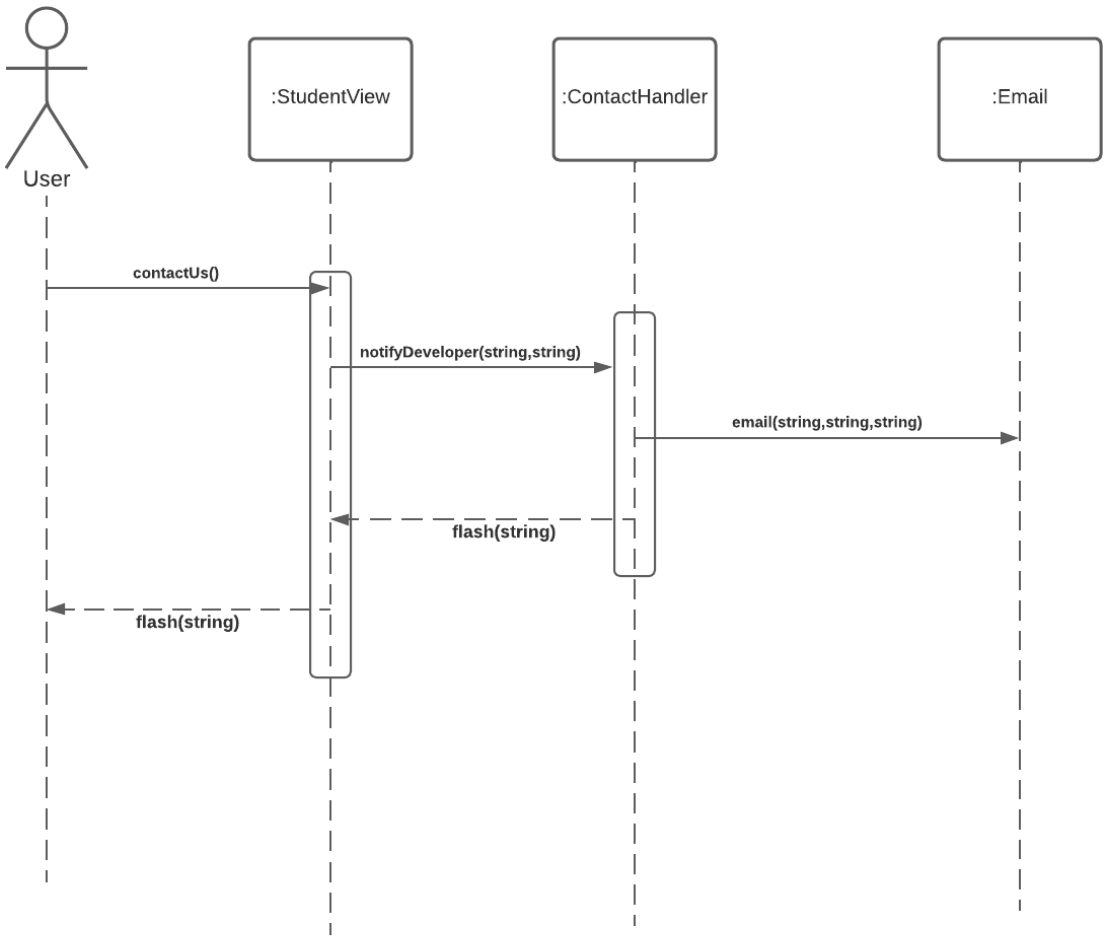


Figure 21: Contact Us Sequence Diagram

4.3.3.9 Sequence Diagram

The sequence diagram shown in Figure 22 illustrates the process of searching for a major by name.

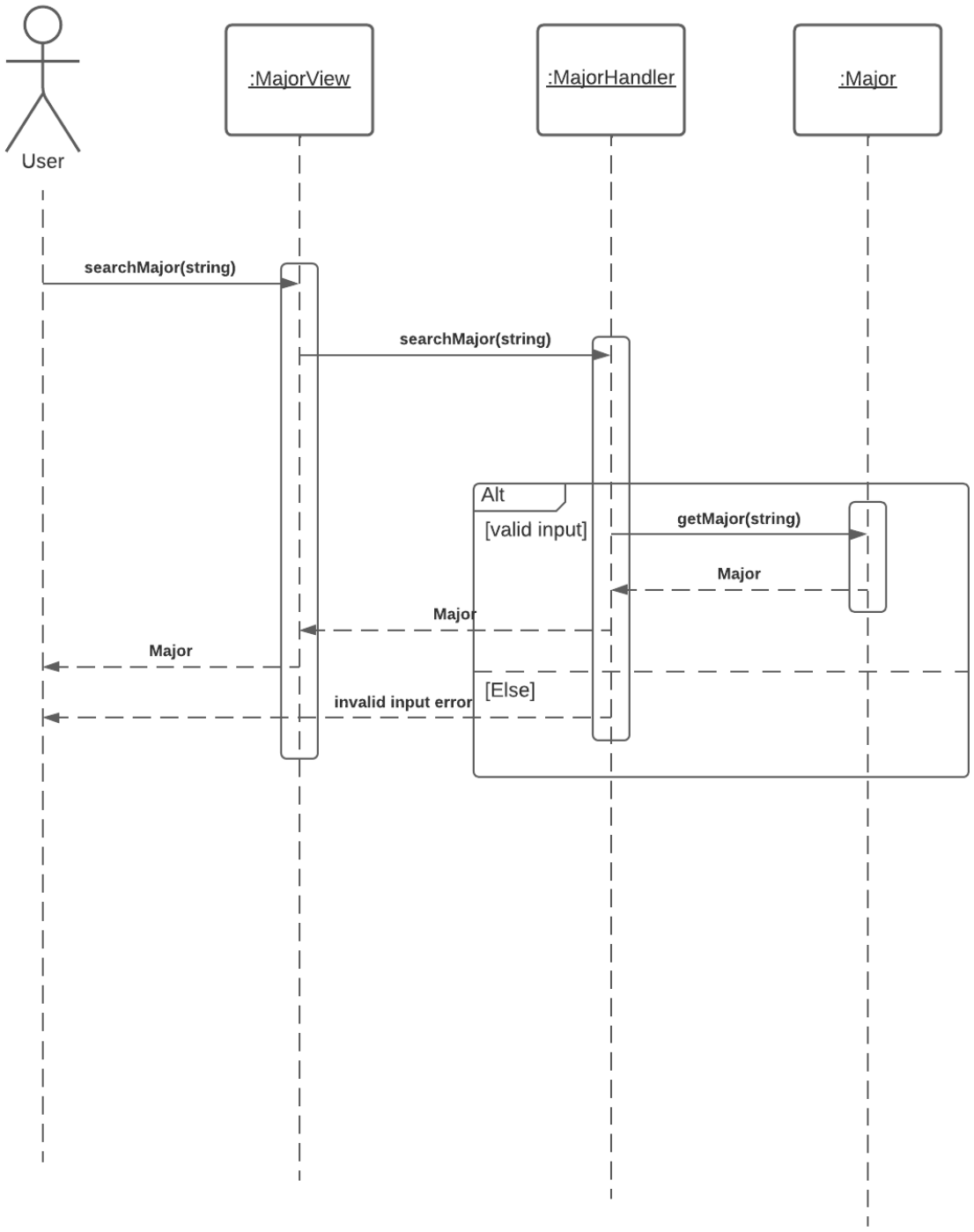


Figure 22: Search Major by major name Sequence Diagram

4.3.3.10 Sequence Diagram

The sequence diagram shown in Figure 23 illustrates how a user can remove their account.

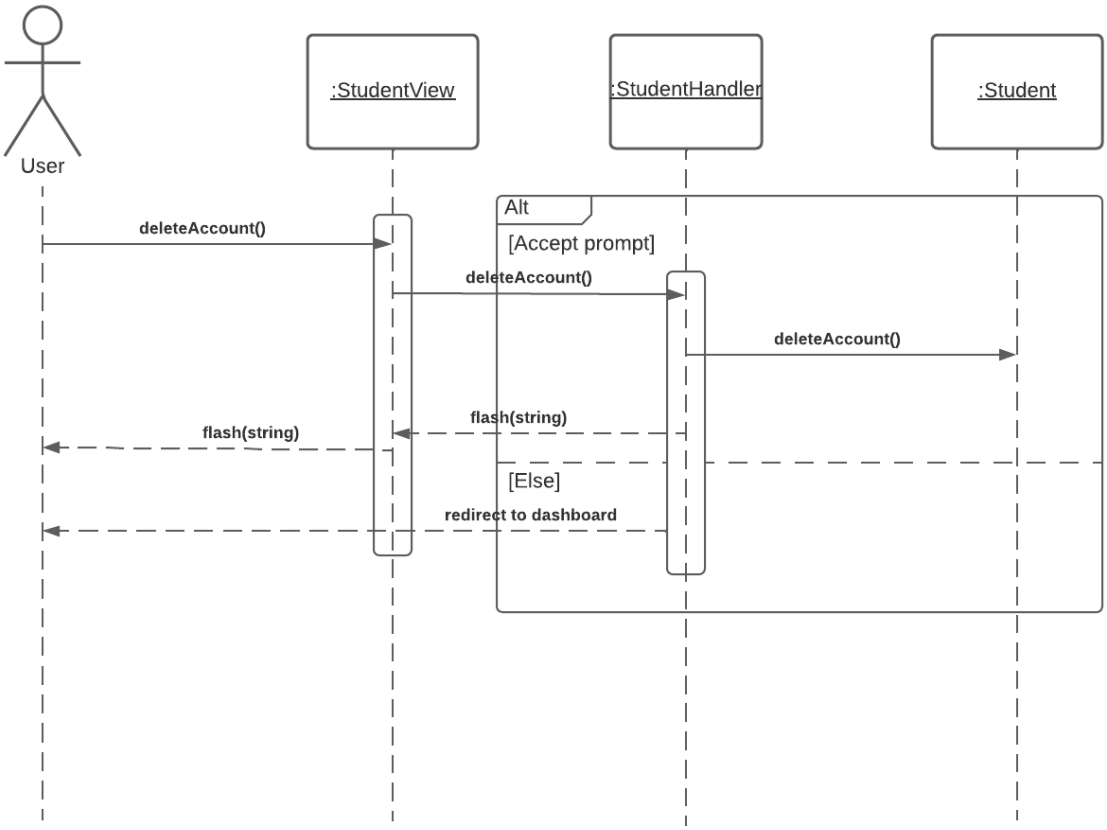


Figure 23: Delete account Sequence Diagram

4.3.4 Development

4.3.4.1 Description

The features we developed in the sprint were our required database objects, which will be used throughout the application. We also developed the logic for the sign up, login, and forgot password. We believe this sprint was a success and the developed database will be vital to the later use case implementations of our application.

```python
class Student(db.Model):
    __tablename__ = 'Student'
    sID = db.Column(db.Integer, primary_key=True)
    sEmail = db.Column(db.String(200), nullable=False)
    sFName = db.Column(db.String(200), nullable=True)
    sLName = db.Column(db.String(200), nullable=True)
    sPassword = db.Column(db.String(200), nullable=False)
    sMajorID = db.Column(db.Integer, nullable=False)
    # sMajorID =  db.Column("sMajorID", ForeignKey(
    #     'Major.mID'), nullable=False)
    def __init__(self, email, password, fname, lname, majorID=1):
        self.sEmail = email
        self.sPassword = password
        self.sFName = fname
        self.sLName = lname
        self.sMajorID = majorID

    def __repr__(self):
        return self.sID

    def setMajorID(self,id: int):
        self.sMajorID = id
```

```python
class Course(db.Model):
    __tablename__ = 'Course'
    cID = db.Column(db.Integer, primary_key=True)
    cCode = db.Column(db.String(10), nullable=False)  # IT383
    cName = db.Column(db.String(200), nullable=False)  # Operating Systems
    # Grade 5 = A 1 = F
    cGrade = db.Column(db.Float, nullable=True)
    # 1 = True 0 = False
    cTextbook = db.Column(db.Float, nullable=True)
    cOnline = db.Column(db.Float, nullable=True)
    cCredits = db.Column(db.Integer, nullable=False)
    # 1-5 scale 5 = most difficult
    cDifficulty = db.Column(db.Float, nullable=True)
    # skill suggestions
    cSkill = db.Column(db.String(200), nullable=True)
    # avg online
    cQuality = db.Column(db.Float, nullable=True)
    cStudentID = db.Column("cStudentID", ForeignKey(
        'Student.sID'), nullable=False)

    cStatus = db.Column(db.String(4), nullable=True)

    def __init__(self, studentID, code, name, credits):
        self.cStudentID = studentID
        self.cCode = code
        self.cName = name
        self.cCredits = credits
```

```python
class Major(db.Model):
    __tablename__ = 'Major'
    mID = db.Column(db.Integer, primary_key = True)
    mName = db.Column(db.String(50), nullable=False)# Computer Science
    cDept = db.Column(db.String(200), nullable=False)  # IT

    def __int__(self, majorName, dept):
        self.mName = majorName
        self.cDept=dept

    def __repr__(self):
        return self.mName
```

```python
class CourseBank(db.Model):
    cID = db.Column(db.Integer, primary_key=True)
    cDept = db.Column(db.String(200), nullable=False)  # IT
    cCode = db.Column(db.String(20), nullable=False)  # 383
    cName = db.Column(db.String(200), nullable=False)  # Operating Systems
    cCredits = db.Column(db.String(20), nullable=False)
    cDesc = db.Column(db.String(200), nullable=False)

    def __init__(self, dept,code,name,credits,desc):
        self.cDept = dept
        self.cCode = code
        self.cName = name
        self.cCredits = credits
        self.cDesc = desc

    def __repr__(self):
        return self.Dept+" "+self.cCode
```

Parent layout:

```
<!DOCTYPE html>
<html lang="en">

<head>



    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
        integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">



    <meta charset="UTF-8">
    <title>{% block title %} {% endblock %} </title>

    {% with messages = get_flashed_messages() %}

    {% if messages %}

    {% for message in messages %}

    <div class="alert alert-warning alert-dismissible fade show" role="alert">
        <strong>{{message}}</strong>
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>

    {% endfor %}

    {% endif %}
    {% endwith %}

</head>


<body>

    {% block body %} {% endblock %}



    <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
        integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
        crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
        integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
        crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
        integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6"
        crossorigin="anonymous"></script>

</body>

</html>
```

```python

@ app.route('/forgot', methods=["POST", "GET"])  # Login page
def forgot():
    if request.method == 'POST':
        user_email = request.form.get("email")
        exists = Student.query.filter_by(sEmail=user_email).first()
        if exists:
            message = Message(
                "Your forgotten password: " + exists.sPassword, recipients=[user_email])
            mail.send(message)
            flash("Email with password sucessfully sent")
            return redirect(url_for("login"))
        else:
            flash("Invalid email, please sign up")
            return redirect(url_for("signup"))
    else:
        return render_template("forgot.html")
```

```html
{% extends "layout.html" %}
<!--EXTEND LAYOUT FILE-->

{% block body %}
<!--Code goes between body blocks-->

<link href="{{ url_for('static', filename='css/signin.css') }}" rel="stylesheet">

<body class="text-center">
    <main class="form-signin">
        <form method="post" action="/forgot">
            <h1 class="h3 mb-3 fw-normal">Please enter email</h1>
            <div class="form-floating">
                <input name="email" type="email" class="form-control" id="floatingInput" placeholder="name@example.com">
            </div>
            <button class="w-100 btn btn-lg btn-primary" type="submit">Submit</button>
        </form>
    </main>
</body>

```

```python
@ app.route('/registered', methods=["POST"])  # Registered Page
def registered():
    user_email = request.form.get("email")
    user_pass = request.form.get("password")

    exists = Student.query.filter_by(sEmail=user_email).first()
    if not exists:
        usr = Student(email=user_email, password=user_pass, fname='', lname='')
        db.session.add(usr)
        db.session.commit()
        # return render_template("error.html", message="Account already exists")
    else:
        flash("Email already exists!")
        return redirect(url_for("login"))

    # db.execute("INSERT INTO student(sEmail, sPassword) VALUES(?,?)",
    #            user_email, user_pass)

    # Send confirmaton email
    message = Message(
        "You have been successfully registered! This is a confirmation email.", recipients=[user_email])
    mail.send(message)

    return render_template("login.html")  # Maybe redirect to mainpage
```

```python
@ app.route('/validate', methods=["POST"])
def validate():
    user_email = request.form.get("email")
    user_pass = request.form.get("password")

    global user
    global COURSES

    user = Student.query.filter_by(sEmail=user_email).first()

    if not user:
        flash("No user account for email")
        return redirect(url_for("signup"))

    if user.sPassword != user_pass:
        flash("Incorrect password!")
        return redirect(url_for("login"))

    # Add user session if checkbox true
    if request.method == "POST" and request.form.get("checkbox"):
        session.permanent = True
        session["email"] = request.form.get("email")

    # return render_template('dashboard.html', courses=COURSES)
    COURSES = Course.query.filter_by(cStudentID=user.sID).all()
    return render_template("mainpage.html", courses=COURSES)
```

4.4 Deliverable 3

4.4.1 Description

The third sprint mainly consists of wrapping up most of the use case implementations for our application. Some of the use cases developed are the following: Search major, select major, delete account, send feedback, notify developers via email, and search major. With the application mostly created, we can then move towards testing and finetuning the application as needed.

4.4.2 Development

4.4.2.1 Description

The features we developed in the sprint were the following: Search major, select major, delete account, send feedback, notify developers via email, and search major. We believe this sprint was a success as we successfully implemented many of the required features as noted in our use case diagram.

```python
@app.route('/deleteUser/')
def deleteUser():
    global user

    message = Message("Your account has been successfully deleted.", recipients=[user.sEmail])
    mail.send(message)

    db.session.delete(user)
    db.session.commit()

    flash("Account removed.")
    return redirect(url_for("logout"))
```

```python
@ app.route('/contactUs')  # Contact Us page
def contactUs():
    return render_template("contactUs.html")


@ app.route('/contacted', methods=["POST"])  # Contacted Page
def contacted():
    email_subject = request.form.get("subject")
    email_message = request.form.get("message")
    admin_email = "developerit326@gmail.com"

    message = Message ("""\
        Subject:  {subject}""".format(subject=email_subject), recipients=[admin_email])
    message.body = " Message: {message}".format(message=email_message)

    # Send confirmaton email
    mail.send(message)
    flash("Feedback sent!")
    return render_template("mainpage.html")  # Maybe redirect to mainpage
```

```python
def viewmajors():
    global user
    global COURSES

    majors = Major.query.all()

    curMajor = db.session.query(Major)\
        .filter(Major.mID == user.sMajorID).first()

    if not curMajor: curMajor="Undecided"

    return render_template("viewmajors.html", majors=majors, curMajor=curMajor)


# updates the major in the DB and displays a message reflecting that to that user
@ app.route('/updatemajor/<int:majorID>', methods=['GET', 'POST'])
def selectmajor(majorID):
    global user
    global COURSES

    updateUser = Student.query.filter_by(sID=user.sID).first()
    updateUser.sMajorID = majorID
    db.session.commit()

    flash("Major Successfully Updated")
    user = Student.query.filter_by(sID=user.sID).first()
    return render_template('mainpage.html', courses=COURSES)
```

```python
def createMajors():
    db.session.query(Major).delete()
    comSci = Major(mName='Computer Science', mDept="IT")
    db.session.add(comSci)

    cyberSec = Major(mName='Cyber Security', mDept="IT")
    db.session.add(cyberSec)

    infoTech = Major(mName='Information Technology', mDept="IT")
    db.session.add(infoTech)
    db.session.commit()

    majors = Major.query.all()
    for maj in majors:
        print(maj.mID, maj.mName, maj.mDept)
```

```html
<div class="text-center text-md-left">
    <a id="back-button" href="/mainpage" class="btn btn-secondary btn-sm" >Back</a>
</div>
<section class="mb-4">
        <!--Section heading-->
        <h2 class="h1-responsive font-weight-bold text-center my-4">Contact us</h2>
        <!--Section description-->
        <p class="text-center w-responsive mx-auto mb-5">Do you have any questions? Please do not hesitate to contact us directly. Our team will co
        a matter of hours to help you.</p>

        <div class="row">

         <div class= "col">
            <!--blank filler column-->
         </div>
            <!--Grid column-->
            <div class="col-6">
                    <form id="contact-form" name="contact-form" action="/contacted" method="POST">

                        <!--Grid row-->
                        <div class="row">
                            <div class="col-md-12">
                                <div class="md-form mb-0">
                                    <label for="subject" class="">Subject</label>
                                    <input type="text" id="subject" name="subject" class="form-control">
```

```
<div class = "container">
  <div class="text-left">
      <a id="back-button" href="/mainpage" class="btn btn-secondary btn-sm" >Back</a>
  </div>

  <h1 align="center">Major's Setting Page</12>
  <h2>Select Major</h2>
  <h4>Current Major: {{curMajor}}</h4>
  <input class="form-control" id="classInput" type="text" placeholder="Search...">
  <br>

  <form method="post" action="/viewmajors">
    <table class="table table-bordered table-striped">

      <thead>
        <tr>
          <th>Majors</th>
        </tr>
      </thead>
      <tbody id="myTable">
        {% for major in majors %}
        <tr>
            <td>{{major.mName}}</td>
            <td>
              <a id="majorSelect" href="/updatemajor/{{major.mID}}" class="btn btn-warning btn-xs" >Select</a>
```