Executive Summary

At the University of Victoria (UVic), course registration is a difficult and stressful process for students. Although registration is somewhat straightforward for those individuals that manage to follow their curriculum throughout their degree, it can become an enormous headache for students who get behind in their courses due to illness, poor grades, or other reasons. The web applications provided by the University and third parties for registration have a number of problems that Puzzle would like to address by providing a new website that serves as a complete solution for student's degree planning needs.

There is a website developed by a past student at UVic called ScheduleCourses.com which serves as a significant improvement to the University provided site. It allows students to select a list of courses, and then shift the course times selected around in order to make an ideal schedule. It provides a dynamic visualization of your timetable which is very helpful for mitigating conflicting course times.

The most significant issue with ScheduleCourses.com is that there is no simple way to view how your selection of courses in a particular semester will affect your final goal of completing your degree program. Often, students will fail to register in a prerequisite during a particular term. This will cause them to be unable to take required courses in the future, putting them further behind in their degree. Similarly, courses are often offered in specific terms, and an unaware student may need to wait a year to take a particular course. Our system will attempt to take into account a user's program requirements and provide a means of visualizing how their selected registration will affect their final graduation date, and future schedule requirements.

This is a fairly large project which will require a lot of data entry regarding various program requirements at UVic. As such, Puzzle will develop a proof of concept that shows how our course scheduling system will work with the Software Engineering program at UVic. If this concept is satisfactory, it will then be possible to move forward with the other programs offered at the school.

S2a Conceptual Design Document

Graduate!

University Degree Planner

Created by: Puzzle

Authors:
Brendan Heal
Jonah Rankin
Ali Nobari
Spencer Mandrusiak

Table of Contents

1.0 Project Summary

The Graduate! web application will help students at the University of Victoria plan their degree by providing students with tools to create custom tailored schedules, which accommodates each individual's specific needs. Many University programs have highly specific degree requirements and the University of Victoria is no different. Just Right Showers has expressed interest in commissioning a system that will assist UVic students by planning their entire degree. Puzzle's Graduate! system will help users by allowing them to customize certain aspects of the schedule, such as having no evening classes or only four classes in a semester, to accommodate for the busy lifestyle of a university student. This will give students more freedom when planning out their degree and also help them keep track of what they have or have not completed. Another way Graduate! benefits students is it will automatically regenerate a new schedule if a student marks that they have failed a course; therefore the student will not have to deal with the burden of rearranging their schedule if such an unfortunate event occurs.

1.1 Important Features

Graduate! has many important features such as:

- Generate a schedule for a specific degree/program. Users maintain one program schedule. To facilitate schedule experimentation, users will be able to undo changes. All changes a user makes are immediately saved.
- Allow users to customize aspects of the schedule such as number of courses per term, time preferences, and specify any work terms to avoid classes being scheduled during that time
- Make automatic adjustments to the schedule if a user makes alterations, changes their preferences, or fails a course
- Allow a user to manually enter desired courses such as when students have not declared their program
- Adheres to University restrictions on courses such as term offerings, pre and co-requisites, maximum credits per term, etc.
- Allow user to view, update and save the schedule to their profile, as well as make modifications at a later date

What kind of undo feature are planning to put in? Will it support multiple undos?

1.2 Hardware

Our software will run on all major operating systems including: Mac OS X, Linux, and Windows. Graduate! will be designed to run on any modern HTML5 compliant browser including: Chrome, Firefox, Safari, and Internet Explorer.

1.3 Performance

Graduate! is expected to be available at least 95% of the time in a year (approximately 18 days of down time) and leave users without access to their schedules for no more than 48 hours at a time. While Graduate! is not safety critical software, it is important that scheduling services be available for users and that course registration will not be compromised because of a failure of the Graduate! system.

The course scheduling algorithm is expected to complete in less than 5 seconds, and during processing it must provide the user with feedback.

We think that 95% is a good amount of availibility, but can you tailor it so that there is as little down time as possible (~99% availibility) in crucial registration periods at the University? (such as the beginning and ending of semesters)

At Puzzle we value our users privacy and the final product will utilize industry standard encryption and security practices to protect user information.

2.0 User Interaction

2.1 Application Views

2.1.1 Front Page

The Front Page is the landing page for the application with options to register, login, and get more information about Graduate!.

2.1.2 Login Page

The *Login Page* contains a form for users to enter their Email and Password. The **Cancel** button returns the user to their previous page — most commonly the Front Page. The Login button proceeds with login validation. If the submitted credentials are valid the user is taken to the *Program Management Page*. The *Login Page* may alternately appear as a modal window over-top the previous view.

2.1.3 Program Management Page

The *Program Management Page* is the home view for the application. From this view a logged-in user will be able to:

- Review their schedule including completed classes, the current semester and future semesters.
- Add, modify, and remove courses
- Define program preferences
- Auto-generate schedules based on their preferences.

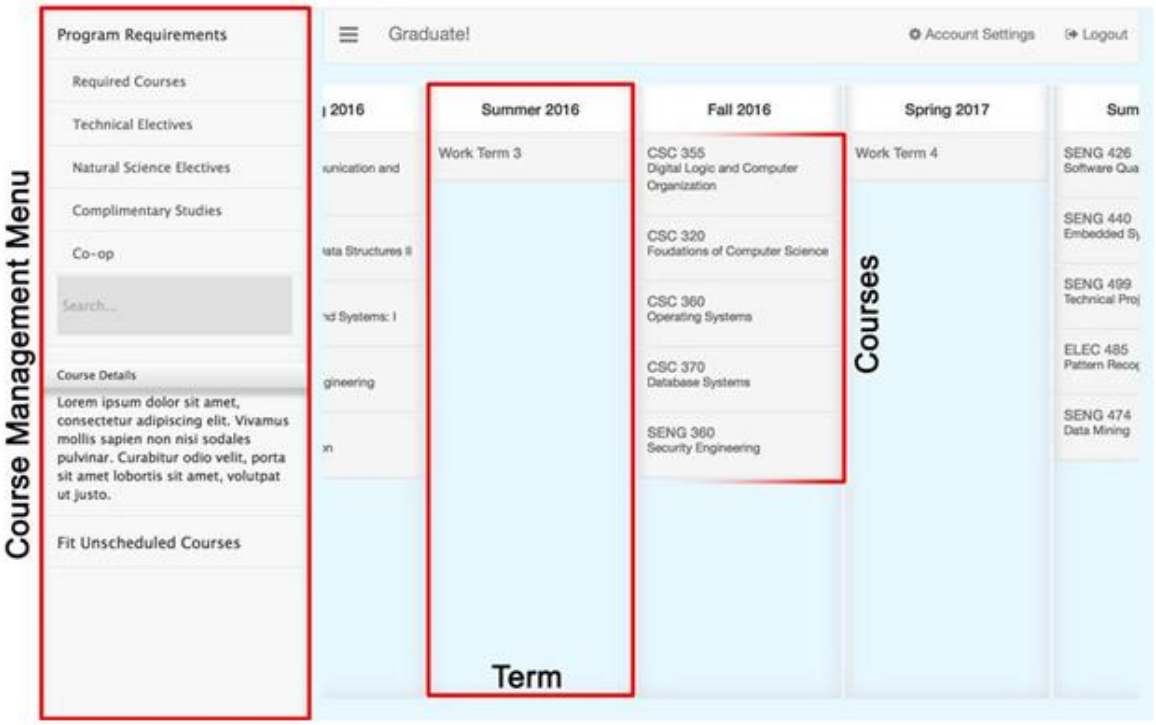

Figure 1: Mock-up of Program Management View

2.2 Use Case Diagram

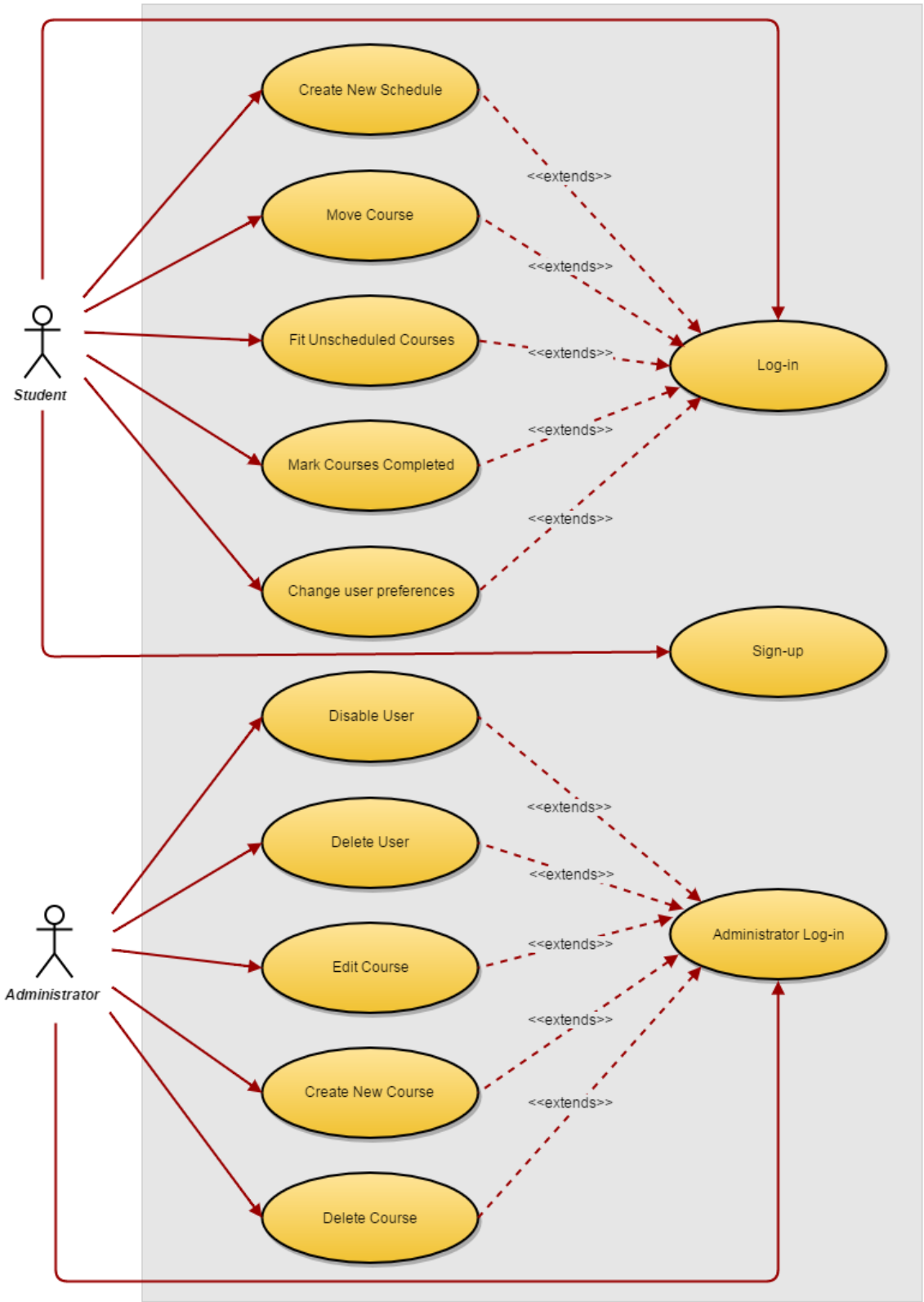

Figure 2: Use Case Diagram

2.3 User Use Cases

2.3.1 Registration

When a user first accesses the web application they arrive at the *Front Page*; users have the option to *Login* or *Register*. When the registration button is pressed the registration process begins:

1. The user enters their account information including: Email, Password, and University. After entering their information, the user presses the **Next** button to proceed to the second registration step.
2. The user specifies their course and program information. First, the user selects their program. Choosing a program populates the required courses list. Optionally, the program can be unspecified and all classes must be added manually. Second, the user adds other courses, as well as specifying electives.[1] Third, the user marks all the courses in the list that they have completed. After all program information has been entered, the user finalizes account registration by pressing the **Finish Registration** button.[2,3]
3. After finishing the registration process, the user is redirected to the *Program Management Page*.[4]

[1] It is not required for users to specify all electives to register. Initially electives will be specified as generic classes such as *Technical Elective* or *Complementary Study*. In order to schedule an elective, it must be a specific course. This can be done after the registration process.

[2] Users will have the option to modify program details and add or remove courses after registration.

[3] The course and program specification process may be simplified through integration with a user's UVic account. This would allow for automatic retrieval program information and completed courses.

[4] Account confirmation emails are not a high priority for initial prototypes. They will be implemented if time permits. If account confirmation is implemented, users will be sent to the login page and notified that they must check their email and confirm their account.

Error Handling

Incomplete Registration: If a user attempts to register for Graduate! without filling out all required sections of the form, they will not be able to register. A notification will pop-up informing the user that they have not filled out all required sections. They will continue to receive this notification until they have filled in the missing sections.

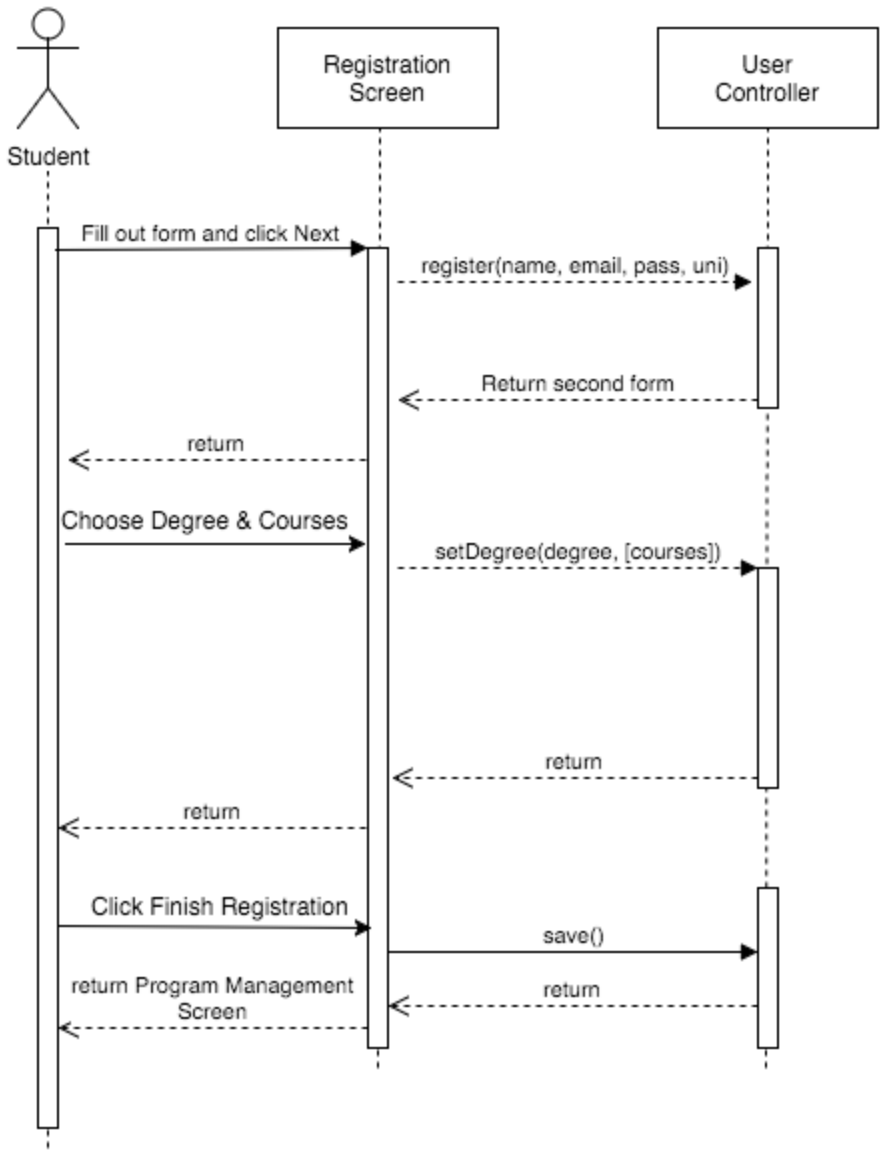

Figure 3: Registration Sequence Diagram

2.3.2 Login

When arriving at the applications *Front Page*, pressing the **Login** button brings the user to the *Login Page*. On the *Login Page,* the user enters their *email* and *password.* They then press the **Login** button, which redirects them to the *Program Management Page*. Alternatively, on the *Login Page* the **Cancel** button returns the user to the *Front Page*.

Sample Interaction: Jane has just arrived at the *Front Page* of Graduate!. Since she already has an account, Jane selects the **Login** button, bringing her to the *Login Page.* Jane enters her email and password, followed by pressing the **Login** button. Once authorized, Jane is brought to the *Program Management Page*.

Error Handling

Incorrect Login: If a user attempts to login with an invalid username or password, they will be denied access to Graduate!. A notification will be displayed informing the user that the username or password they entered is invalid.

Cancel: If the user selected the login button and does not want to login, they may select **Cancel** to return to the front page.

2.3.3 Mark Courses Complete

Once registration has been successfully completed, Graduate! will automatically mark courses as completed based on the user's transcript. Graduate! will update completed courses each time the user logs in to the application.

Sample Interaction: Jay has already started using Graduate! and has a generated schedule. Because Jay has just finished his term and passed his courses, one of which is "SENG 321", he navigates to the *Program Management Page*. When Jay arrives at the page, he selects **View Schedule** and notices that Graduate! has already marked "SENG 321" as complete.

2.3.4 Alternate - Mark Courses Complete

Upon registration, the user will be prompted to select all courses which they have already completed. Alternatively, the user may go to the *Program Management Page* and select **Add Completed Course**, where they will search for the appropriate course. Once they have found the desired course, the user marks a checkbox indicating the course has been completed.

Sample Interaction: Jay has already started using Graduate! and has a generated schedule. Because Jay has just finished his term and passed his courses, one of which is "SENG 321", he navigates to the *Program Management Page*. When Jay arrives at the page, he selects the **Add Completed Course** button, which displays a list of all courses relevant to Jay's degree. Jay searches for "SENG 321" and selects the checkbox indicating he has completed the course.

Error Handling

Add incorrect course: If a user marks a course as complete that should not be completed, the user must simply uncheck the box, indicating the course is not completed.

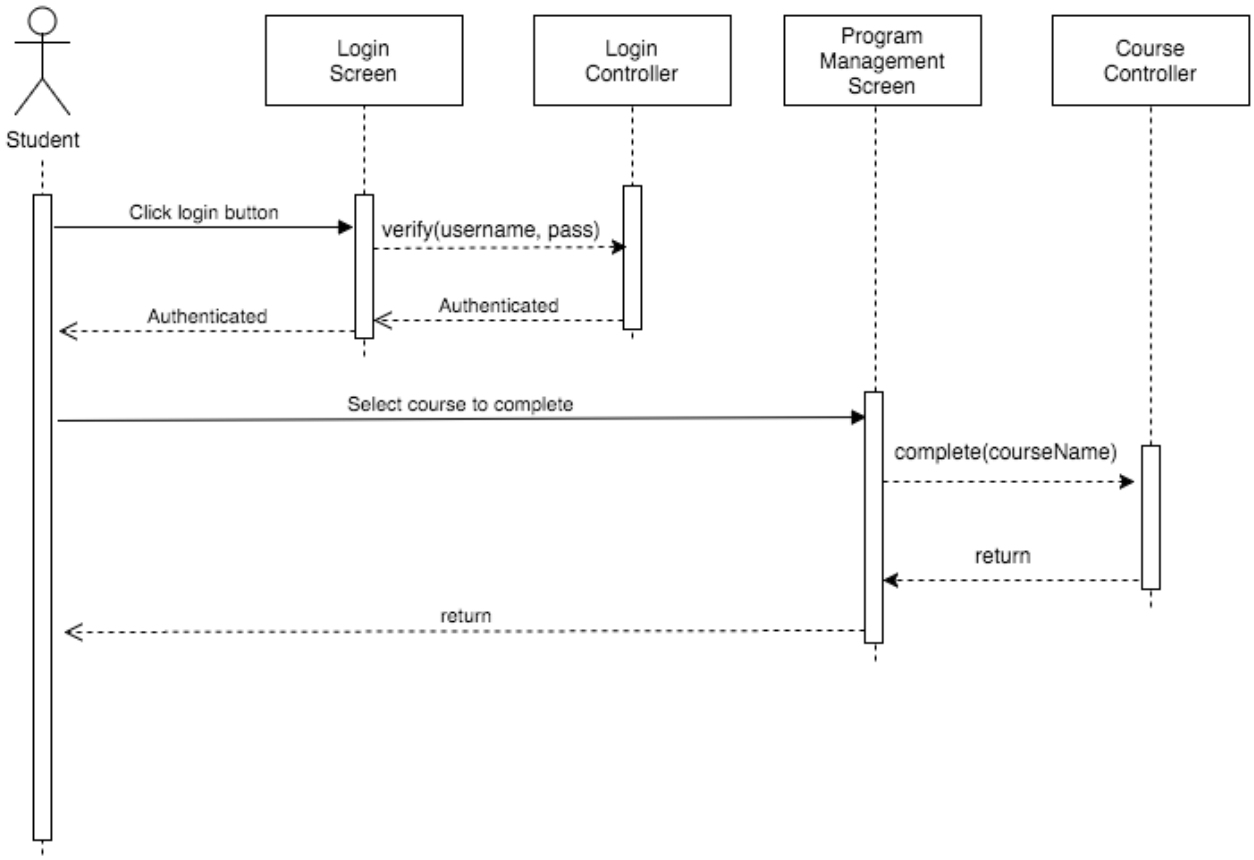

Figure 4: Mark Course Complete Sequence Diagram

2.3.5 Add Courses to Schedule

A registered user will be able to add courses to their schedule. On the *Program Management Page*, the user opens the *Course Management Menu*. From the *Course Management Menu* the user can select courses as specified by their program requirements or by search. To add courses to a term the user drags the course from the *Course Management Menu* to the desired term on the *Program Management Page*.

Initially the *Program Management Page* will contain no courses. On screen prompts will direct first time users to the *Course Management Menu*. The prompts for first time users will introduce them to the basic functions of Graduate! including:

- The Course Management Menu
- Selecting electives
- How to add courses to a term
- How to move courses between terms
- How to use the Fit Unscheduled Courses option to automatically generate a schedule according to specific preferences

In the case where a student adds a course when and its prerequisite is not scheduled[1], the system will display clear visual identification[2].

[1] Prerequisite conflicts may occur for courses that have multiple unmet prerequisites. In the case where the prerequisites do not apply to fulfilling degree requirements a notification will appear. For example: A computer science student wants to take POLI 410 which has the prerequisites POLI 310A and POLI 310B. A notification will inform the users that if they take POLI 410 the prerequisites may not be applicable to the completion of their degree.

[2] Clear identification may include a red tint, exclamation icon, and/or pop-up notification.

Sample Interaction: Jay has recently finished registering for Graduate!. When he arrives at the *Program Management Page*, after logging in, a pop-up window directs his attention to the Course Management Menu button, he clicks it and the Course Management Menu appears. Jay wants to add MATH 122 and CSC 225 to the same term as his friends so he drags them from the Menu to the Summer 2017 term.

2.3.6 Move Courses

On the *Program Management Page*, courses can be shifted between semesters by drag-and-drop. The user will be notified graphically if they are moving a course to an incorrect semester. This notification will appear for:

1. Missing prerequisites
2. Course not offered during the semester[1]

[1] Criteria 2 depends on the release of course schedules — usually 4-8 months ahead of their offering. If a course is moved beyond this date, the system will use the historical data to determine if the course is likely to be offered. A notification will also appear informing the user that offerings are subject to change.

*Sample Interaction:*Jane decides she no longer wants to take “CSC 225” in the summer semester. Thinking ahead, Jane determines the only other available semester for this course is in the fall. Once Jane arrives at the *Program Management Page,* she drags “CSC 225” from her summer semester schedule to her fall semester schedule. Graduate! verifies the course is available during this semester, and that Jane has the proper prerequisites. Upon success, Graduate! automatically saves the new schedule.

What about the case where a student enters a course that they have prerequisites for and it's offered the semester they want to move it to, but it conflicts with one of courses already in that semester? For example, trying to move CSC 225 in a semester where you already have CSC 226.

Our suggestion is that the program will automatically move the course to the next available time slot within parameters given.

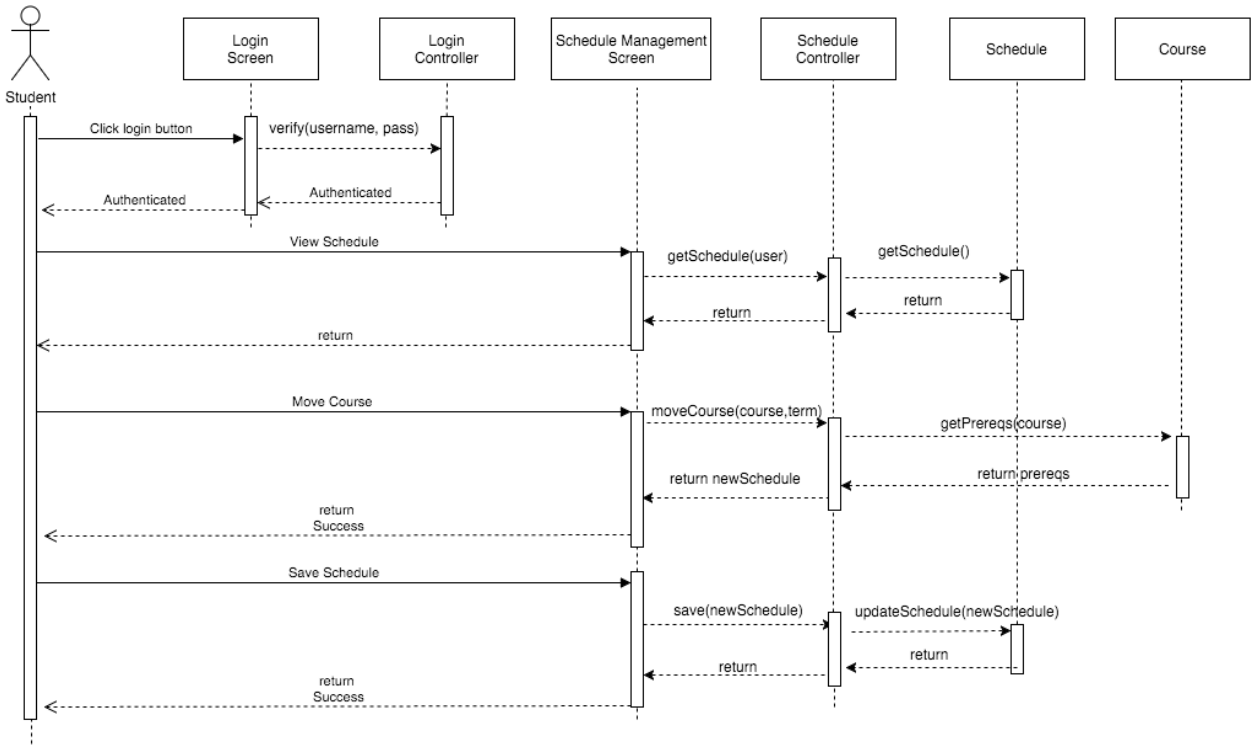

Figure 5: Move Courses Sequence Diagram

2.3.7 Fit Unscheduled Courses

The schedule generation performed by Graduate! will fit existing unscheduled courses to the existing program schedule. To fit their unscheduled courses a user first navigates to the *Program Management Page*. In the Unscheduled Courses panel, the user clicks the **Fit Courses** button. A pop-over notification will inform the user that schedule generation is being performed. When complete, the notification will update allowing the user to undo the changes made. Fitted courses will be temporarily tinted[1] so that users can see the changes to their schedule.

In the case that a schedule cannot be generated, such as if the criteria is too restrictive, the user will be informed that only a partial match was achieved. The notification will detail the criteria which was and was not considered and users will have to ability to undo any changes made. As with successful generation, courses added to the schedule will be temporarily tinted[1].

[1] Tinted courses will exist while the schedule notification is visible. Once the user closes the notification all courses will appear normally.

Sample Interaction: Jane has recently completed all of the information required to create a schedule. There are a few courses which she knows she wants to take in the coming semester because her friends are also taking them. From the Unscheduled Courses panel she drags the courses she wants to take into her program schedule. For her remaining unscheduled courses Jane presses the Fit Courses button. A notification pop-up up saying her courses are being scheduled, in a few seconds the notification informs her that a match was found. She can easily

observe the changes to her schedule by seeing the tinted courses. Jane likes her schedule, and she closes the notification.

2.3.8 Setting Generation Preferences

When a user clicks the **Fit Unscheduled Courses** button a settings window will appear. On this window the user will be able to specify their target graduation date and their desired numbers of courses per term. Users will be able to prioritize their preferences, prioritization will assist the course generation algorithm in finding partial matches. After all settings are specified a user can generate their schedule by pressing the **Generate Button**. User's close the preferences window by pressing **Cancel**.

A user's preferences will be retained and reused the next time a schedule is generated.

Sample Interaction: Jay has been using Graduate! for some time now. Jay decides for his next semester he does not want more than four classes, so he navigates to the *Project Management Page* and opens the *Course Management Menu*. Jay presses **Fit Unscheduled Courses** button then sets the "Maximum Courses per Term" option to "4" and changes it to the top priority. He then clicks **Generate**, and a schedule with only four courses per term is generated.

2.4 Administrator Use Cases

2.4.1 Administrator Login

When arriving at the applications *Front Page*, pressing the **Login** button brings the administrator to the *Login Page*. On the *Login Page,* the administrator enters their *email* and *password.* They then press the **Login** button, which redirects them to the *Administrator Page*. Alternatively, on the *Login Page* the **Cancel** button returns the administrator to the *Front Page*.

Sample Interaction: Steve has just arrived at the *Front Page* of Graduate!. Since he already has an administrator account, Steve selects the **Login** button, bringing him to the *Login Page.* Steve enters his email and password, followed by pressing the **Login** button. Once authorized, Steve is brought to the *Administrator Page*.

Error Handling

Incorrect Login: If an administrator attempts to login with an invalid username or password, they will be denied access to Graduate!. A notification will be displayed informing the administrator that the username or password they entered is invalid.

2.4.2 Disable User

Once the administrator has successfully logged in, they select **View Users**. This will display a list of all current users in the system. The administrator then scrolls through the list to find the username desired. Alternatively, the administrator may simply enter the desired username into the search box in the *View Users* section. Once the user has been found, the administrator selects the profile, followed by **Disable User**.

Sample Interaction: Steve is an administrator for Graduate!, therefore when he logs in, he has all options that an administrator has. Steve is directed by his boss to disable the user with the username “user123”. He selects **View Users**, and types “user123” into the search box. Steve selects the profile, and disables it by pressing **Disable User**.

2.4.3 Delete User

Once the administrator has successfully logged in, they select **View Users**. This will display a list of all current users in the system. The administrator then scrolls through the list to find the username desired. Alternatively, the administrator may simply enter the desired username into the search box in the *View Users* section. Once the user has been found, the administrator selects the profile, followed by **Delete User**. They will be prompted with, “Are you sure?”. The administrator will select **Yes**.

Sample Interaction: Since Sarah is an administrator, when she logs in she has administrator options. Sarah’s boss has instructed her to delete the user with the username “userSmith”. She selects **View Users**, and enters “userSmith” into the search box. Sarah then selects the user profile, and selects **Delete User**. Graduate! prompts her with a notification saying, “Are you sure?”. Sarah selects **Yes**.

2.4.4 Create New Course

After successfully logging in to Graduate!, an administrator will select **View Courses**. In the upper right hand corner, they will select **Add Course**. This will display a form for the administrator to fill out with fields such as *Course Name, Course Description, Prerequisites,* and *Co-requisites*. Upon completion of filling out the form, they will save the information by clicking **Finish & Save**. Alternatively, if the administrator does not want to save the filled in information, they will select **Cancel**.

Sample Interaction: Sarah discovers the course “SENG 321” has not yet been added to the Graduate! system. She first logs into her administrator account, and selects **View Courses**. Once the page loads, she selects **Add Course**. Sarah fills out the required information for the course and looks it over for any errors. Since she is satisfied with her work, she selects **Finish & Save**.

There were no requirements for making an administration role for this software. We can't see of any reason why a person would need to disable or delete a user. Please justify why this feature is needed.

As for adding and editing courses, it seems like your project tries to do two approaches simultaneously: one where the design is completely integrated with UVic's systems, and the other where it takes a more third-party approach.

If the system is completely integrated, there would not be a need for the system to have this administration built in, as the University would handle it on their end. (They also wouldn't need the delete/disable tools, as your account would be tied in with your University account).

If the system is being designed with the third-party approach, we can see why the Edit/Create/Delete courses functions would be useful.

Please decide on which approach you would like to pursue, instead of trying to do both at the same time. We suggest the third-party approach, with the option to integrate with UVic systems in the future.

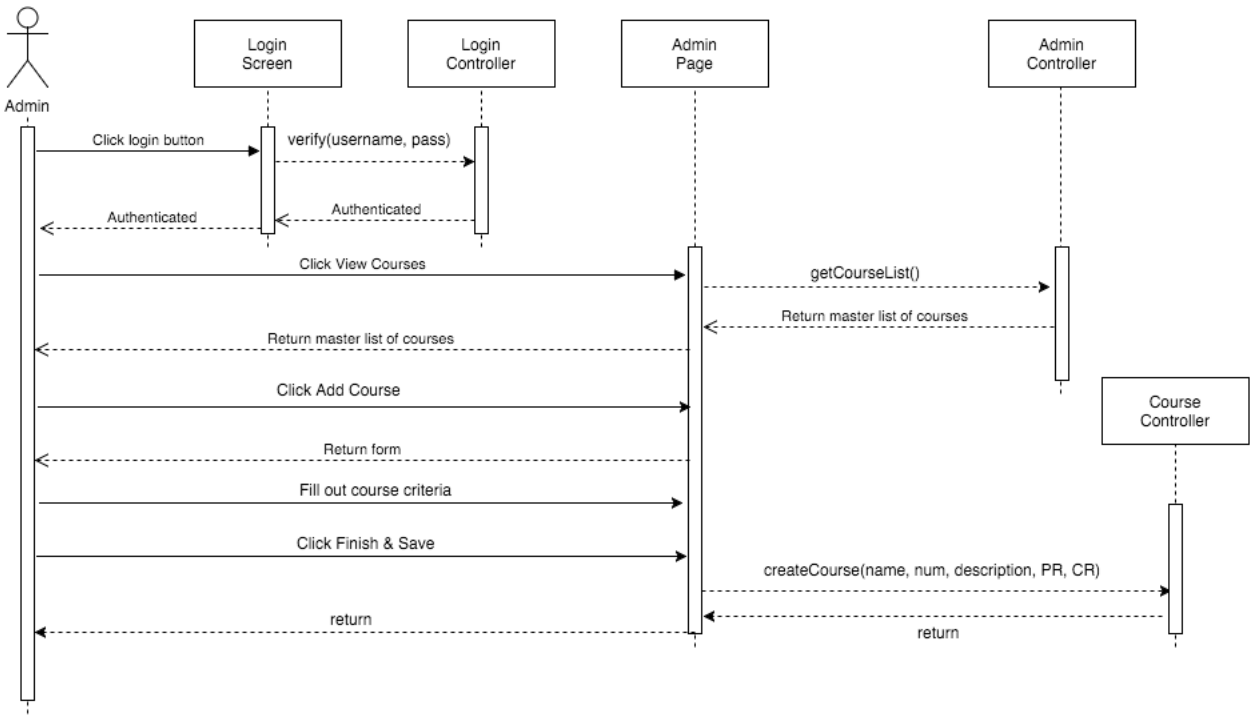

Figure 6: Create New Course Sequence Diagram

2.4.5 Edit Course

Once the administrator has successfully logged in, they select **View Courses**. This will display a list of all courses in the Graduate! system. The administrator can either scroll through the list of courses until they find the desired course, or they can simply type the name of the course into the search box on the *View Courses* page. The administrator selects the appropriate course, followed by **Edit Course**. When the administrator is finished editing, they select **Save**.

Sample Interaction: Steve notices an error in the spelling of the course name “SNG 321”, so he logs into his account with administrator privileges. Steve selects **View Courses** and scrolls through the list until he finds “SNG 321”. He selects the course, followed by **Edit Course**. Steve then corrects the error and changes the course name to “SENG 321”. Now that Steve has finished his corrections, he finishes by selecting **Save**.

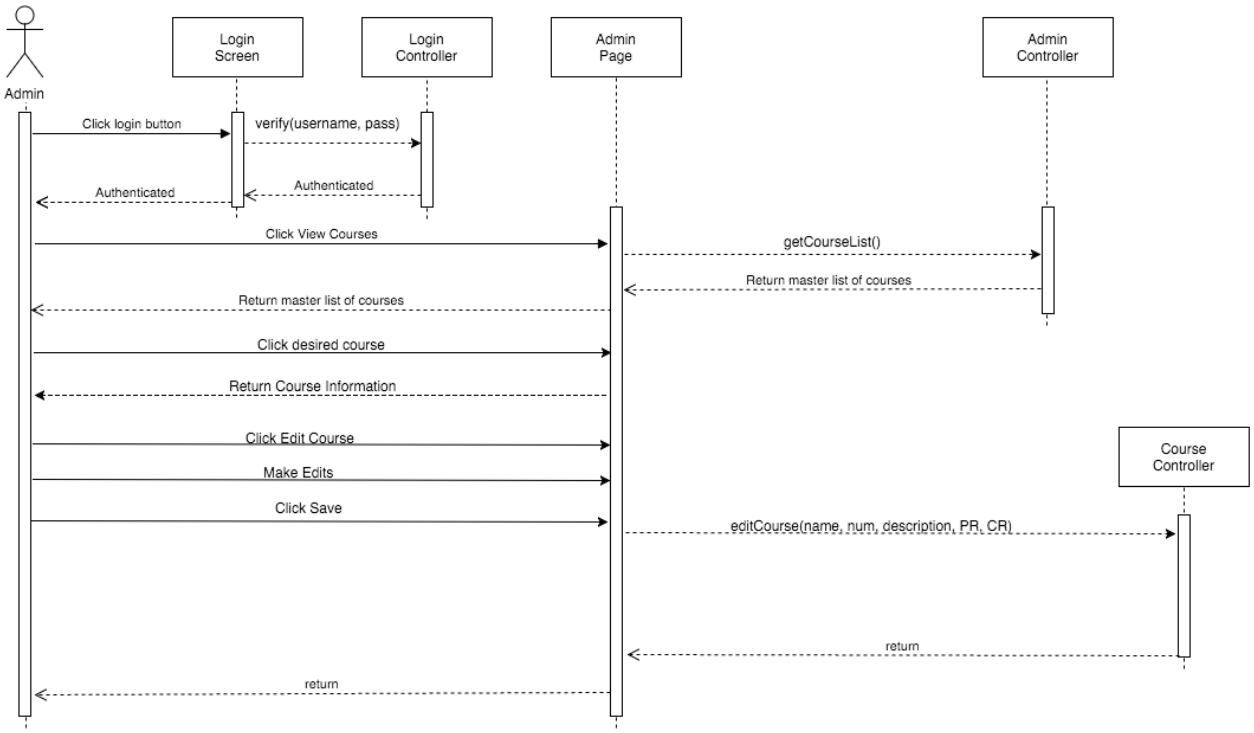

Figure 7: Edit Course Sequence Diagram

2.4.5 Delete Course

Once the administrator has successfully logged in, they select **View Courses**. This will display a list of all courses in the Graduate! system. The administrator can either scroll through the list of courses until they find the desired course, or they can simply type the name of the course into the search box on the *View Courses* page. The administrator selects the appropriate course, followed by **Delete Course**. They will be prompted with, “Are you sure?”. The administrator will confirm by selecting **Yes.**

Sample Interaction: While going through the course list, Sarah notices that “ECON 103C” is still listed as an available course. Because Sarah knows this course is no longer offered at the University of Victoria, she logs into her account with administrator privileges. She selects **View Courses** and types “ECON 103C” into the search box. Sarah then selects the course, and clicks **Delete Course**. Graduate! prompts her with a notification saying, “Are you sure?”. Sarah confirms by selecting **Yes**.

Error Handling

Deleting an active course: An error should warn the administrator that they are about to delete a course that is being used by students and is as well offered by the University. If the administrator sees fit, they can delete the course from the database.

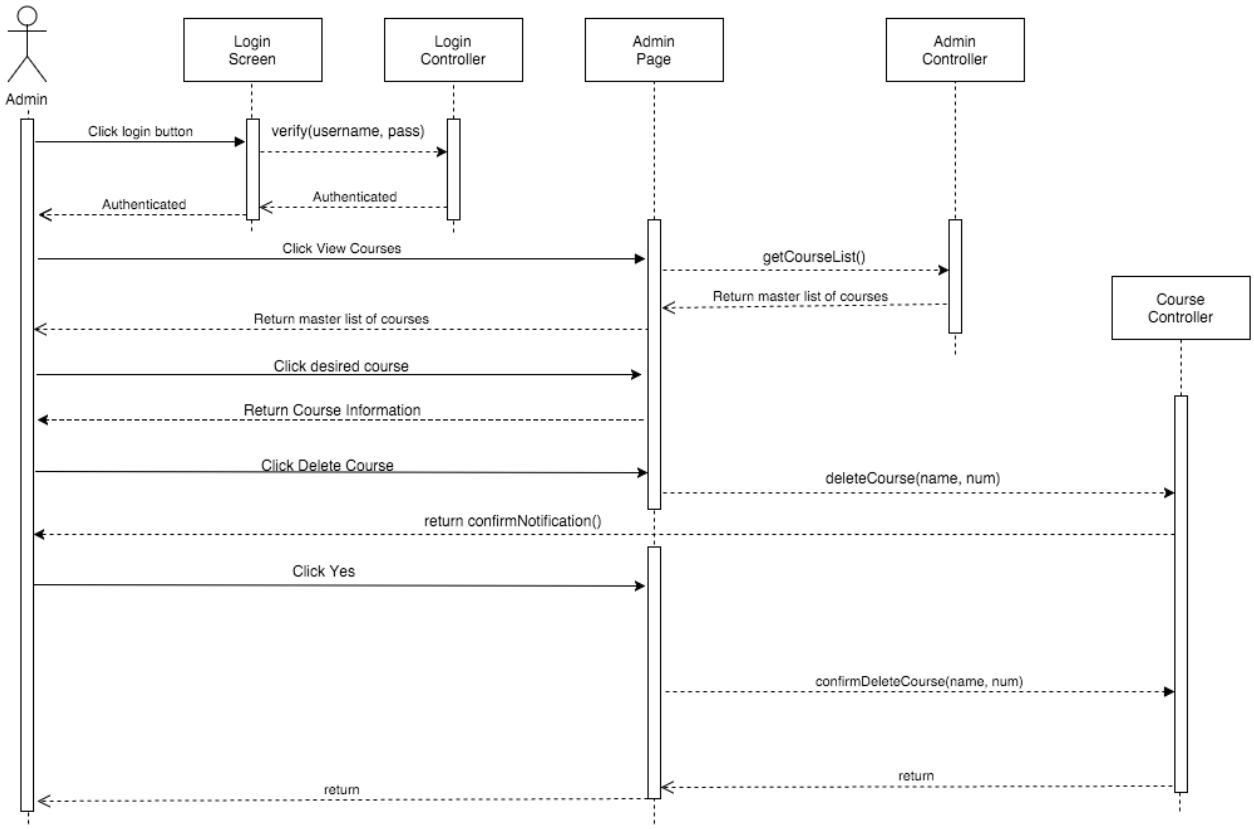

Figure 8: Delete Course Sequence Diagram

2.5 Glossary

Framework: A collection of software libraries that provide generic functionality that can be customized or extended. A framework provides a fast way to implement a commonly needed function.

Bootstrap: A HTML, CSS, JavaScript framework that facilitates fast and clean user interface design

Operating System: System software that manages computer hardware and software resources and provides common services for computer programs (reference: https://en.wikipedia.org/wiki/Operating_system)

Use Case: A list of steps defining the interaction between a role and a system, to achieve a goal (reference: https://en.wikipedia.org/wiki/Use_case)

Web Application: A client-server software application in which the client (or user interface) runs in a web browser (reference: https://en.wikipedia.org/wiki/Web_application)

3.0 Management Plan

3.1 Feature Breakdown

The core features of the Graduate! application is its course scheduling algorithm, intuitive user interface, user authentication, course and user data storage, course information collection, and Netlink integration.

3.1.1 Course Scheduling Algorithm

To obtain an "optimal" program schedule multi-stage processing will be performed. The stage generates options that satisfy University constraints, course prerequisites and terms in which courses are available. The remaining stages generate schedules which satisfy the constraints provided by the user such as graduation data and courses per term. User specified constraints will be processed in the order specified. Users will be able to prioritize their preferences (refer to Change User Preferences use case). If the algorithm cannot satisfy all the provided constraints the system will provide a schedule satisfying their highest priority constraints (the user will be notified that some preferences could not be satisfied). If there would be multiple schedules satisfying all constraints the first schedule found will be returned (If all preferences are satisfied the algorithm will determine that the schedule is "optimal" and in the interest of performance the result will be returned immediately).

How does the user state which constraints are the highest priority?

3.1.1.1 Implementation Details

A vertex graph will be constructed using a table of the courses that need to be scheduled, the courses that are already scheduled, and their prerequisites. By traversing this graph in topological order a list of valid course schedules will be created in such that no course is taken before its prerequisite. Next, the list will be processed such that courses are available only in the terms in which they are offered, while still maintaining the prerequisite constraint. Variations of the schedules will be examined ensuring that user preferences, as prioritized are satisfied. If a schedule is generated that satisfies all preferences it will be returned. If no schedule satisfies all preferences the system will backtrack until a schedule is found which satisfies the user's highest priority schedules.

If there are multiple possible schedules that match all constraints, will there be a way to view the different options? For example, in ScheduleCourses, every time a user says to generate a schedule, it randomly selects which one to show.

3.1.2 Intuitive Interface Design

The creation of an easy to learn and use interface is critical in the creation of this application. To achieve this goal the Graduate! will utilize modern web application features such as drag-and-drop modification of schedules. Beyond the use of modern tool providing the user with constant feedback will provide the information as to whether or not they are performing valid actions when modifying their schedule. If a user attempts to reorder a course such that it is in the term before its prerequisite there will be clear indication on screen that the action cannot be completed and the courses will revert to their previous valid position. It will be possible for users to override course prerequisites so that department course accommodations can be input.

3.1.3 User Interface Layout

To maximize the screen area a side, or top, menu bar will provide the user with quick access to additional features such as the selection of new courses, the modification of schedule preferences, and changes to account information. Refinement of the user interface will occur through rapid prototyping utilizing styling framework tools provided by bootstrap.

3.1.4 Data Collection/Scraping

The data for both the courses and their pre-requisite/co-requisite can be collected from the UVic databases. The information collected also includes the term in which the courses are offered in and the lab sections that come with it. The data is then transferred onto the personal server, where the course scheduler will be retrieving the data from.

3.1.5 Database Creation

A database will be developed created that stores all the relevant information for the Graduate! application. The database will need to store the collected program requirements in order to associate the courses based on prerequisites, corequisites ,labs, electives and alternates. Similarly, the course offerings will be stored based on the particular semesters they are offered. User information and their saved schedule will also be stored in the database.

3.1.6 Netlink Integration

Using your Netlink account, the service is able to view the courses that a user has already taken, or is enrolled in. This way, it can better tailor the remainder of the degrees scheduling to the user. After determining the courses the user has both taken and is enrolled in, those courses can be removed from the overall group of courses that are going to be used to create the schedule for the user. This removes redundancies in what courses the user has to take. As well, pre-requisites can be determined for what courses are going to be scheduled, in case the user took some courses out of the average ordering of the courses they were supposed to take.

3.1.7 User Authentication

Users will initially register on the site and provide some required information to the Graduate! application. They will then be able to login with their selected username and password. All sensitive user information will be stored in a safe, encrypted form. The login page will provide password recovery.

3.2 Possible Implementations

There are a number of technologies suitable for prototyping the Graduate! web application.

3.2.1 Deployment

The application will be deployed using Heroku (www.heroku.com). The service provides an excellent support for multiple web development frameworks and facilitates rapid prototyping. Heroku provides free-of-charge small-scale databases and simple deployment using Git.

3.2.2 Persistent Storage

Heroku provides integrated add-ons for a variety of database management systems. Given the experience of the development team, PostgreSQL or MongoDB will be used.

3.2.3 Server-side Technologies

The requirements for the back-end of the Graduate! web application can be fulfilled by Ruby on Rails, Django, or Node.js. All of the candidate frameworks provide the routing, templating, database object relational mapping, and REST api integrations which are critical to the development of Graduate!.

3.2.4 Client-side Technologies

To create an engaging user interface, the application will rely on a JavaScript library such as Angular, JQuery, or D3. Additionally, to rapidly develop a clean interface, Bootstrap will be employed.

Angular is an extensive front-end web development framework supporting templates and routing. Angular may not be necessary given the choice of Server-side technology.

Implementing a drag-and-drop interface is highly desirable; it will make moving courses between semesters more intuitive and make the application friendly for mobile users. There are several technologies for displaying interactive graphics including SVG (scalable vector graphics), HTML5 Canvas, and JQueryUI.

The use of SVG is particularly promising because they are highly dynamic and would provide the ability to display custom schedules as an interactive graph. To interact with SVG graphics JQuery can be used, but alternative libraries such as D3 (https://d3js.org/) may be better suited as it is specifically designed for the creation of interactive graphics.

3.3 Minimal System

This section describes the minimal system that will be provided by Puzzle by the end of the term. Puzzle is going to develop a proof of concept of the Graduate! application, that shows how it will work for a student taking the Software Engineering program at the University of Victoria. Once the proof of concept is complete, adding other programs should be a simple matter of data entry. As there is only a month of development time available, it may not be possible to develop a fully functioning scheduling application by the project deadline. As such, Puzzle will focus on implementing solutions to the major weaknesses that are present in the current systems.

ScheduleCourses.com shows how a timetable builder system can be developed for a given semester, so our minimal system will not include such a feature. Rather, our program will leverage the program requirements provided in order to develop a plan for which courses will be taken in which semester, without the specific times being evaluated. This decision was made in part because it is difficult to develop and is already available, and because at this time, the University of Victoria only provides specific times for courses at most eight months in advance. This does mean that our suggested schedule will occasionally result in a course conflict, but at this time there is not a way to avoid this.

Our program will focus on providing students with a simple way to manage their program and view how their scheduling choices affect their course load and final graduation date. It will make it easy for students to fit electives in their schedule, and provide a list of possible electives available that can be taken in a given semester. This allows students to work out a schedule that gives them the electives they desire, as opposed to taking whichever elective happens to fit their schedule. The University provides information about which courses are offered in which terms. Our minimal system will also include an authentication system for users to register and login to the system. Each user will be able to select their program and maintain a saved version of their personalized schedule.

Netlink Integration will not be included in the minimal system. Rather than automatically updating based on a student's completed courses, users will manually select their completed courses.

3.3.1 Summary

Graduate!'s main focus is to make planning a student's degree easier and more customizable, while pertaining to the University's restrictions such as prerequisites and term offerings. In the initial proof of concept, only the Software Engineering program at the University of Victoria will be included. The minimal system provided will include a web application that supports user authentication and dynamic schedule creation and visualization based on course offerings provided by the University.

3.4 Team Structure and Organization

The various components of the project have been broken down into tasks that will be taken on by individual team members of our group. The tasks include everything necessary in order to deliver the minimal system, but will be carried out with the final system in mind.

3.4.1 Web Application Development - Brendan Heal

This task involves the design and development of the web application including the selection of the database, front-end and back-end frameworks, configuration and deployment.

Phase 1: System Design

The web application will be designed with the selected front and back-end frameworks in mind.

Phase 2: Development and Deployment

A basic user authentication will be developed and deployed early so that the remaining development can occur.

3.4.2 Database Creation - Ali Nobari

This task involves the collection of data from the University of Victoria and creation of a database for the web application.

Phase 1: Data Collection

Two important pieces of information will be collected from the University of Victoria's website:

1. The program requirements for the Software Engineering program.

2. The course offerings by term.

Ideally, a scraper will be implemented that can access information from UVic's calendar page.

Phase 2: Database Creation

A database will be created which allows comprehensive access to the data collected as well as user information and user schedules.

3.4.3 User Interface Development - Jonah Rankin

A user interface will be developed according to the client's requirements.

Phase 1: Prototyping

User interface prototypes will be developed and presented to the client. Upon satisfaction with the user interface proposed, phase 2 will begin.

Phase 2: Interface Implementation

The views will be implemented using the selected front-end framework.

3.4.4 Course Scheduling Algorithm - Spencer Mandrusiak

An algorithm will be implemented to solve the course schedule optimization problem.

Phase 1: Research

The course scheduling algorithm will be researched in order to ensure an optimal solution is selected.

Phase 2: Implementation

The course scheduling algorithm will be implemented and rigorously tested.

4.0 Conceptual Design

4.1 Application Flow Diagram

The application flow diagram is a representation of the various views that will be included in the system and the links between them. The grey boxes are tasks that can be achieved from the Program Management page. This diagram is useful for understanding navigation through the Graduate! web application.

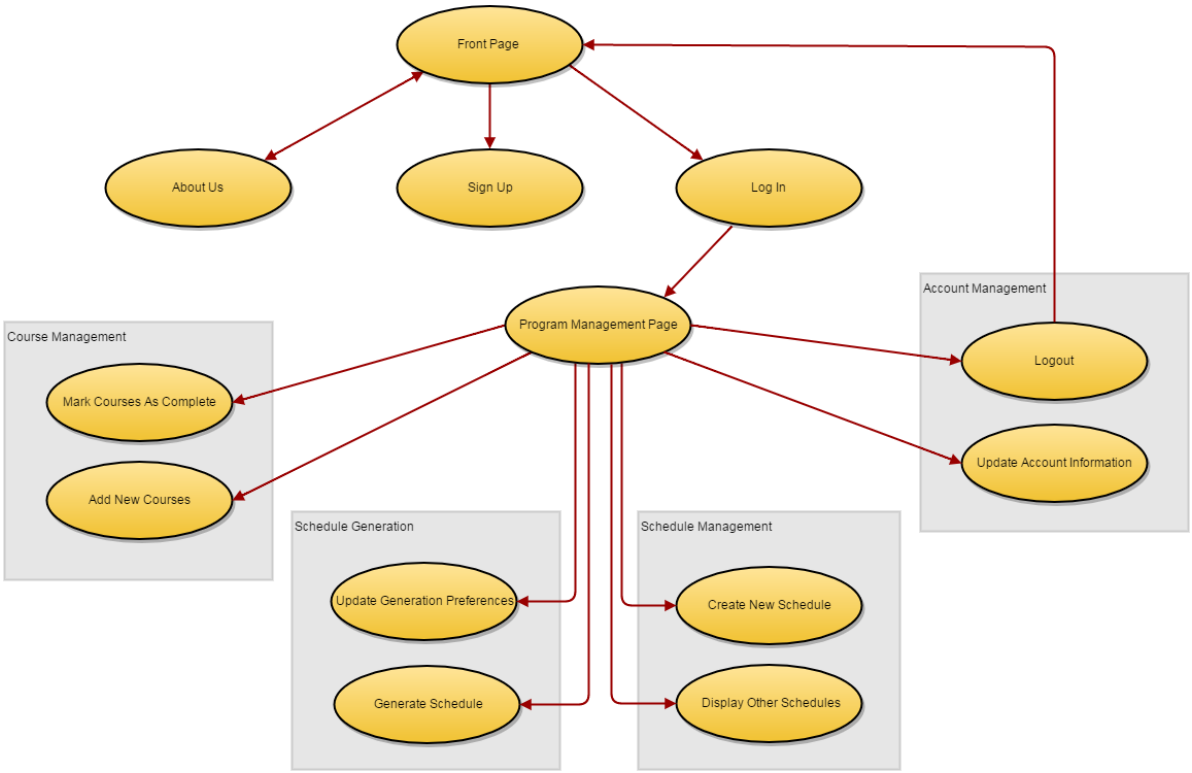

Figure 9: Application Flow Diagram

4.2 Activity Diagram

This activity diagram serves as a high level representation of workflow through the application. It describes the flow of various user actions throughout the Graduate! application.

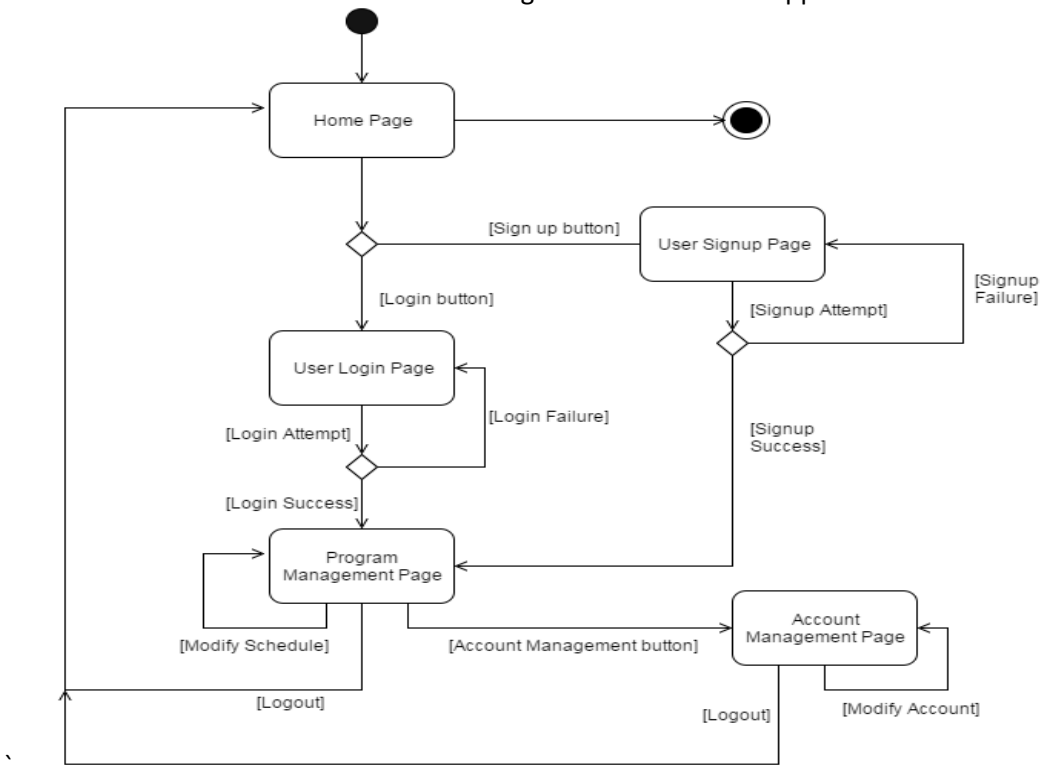

Figure 9: Activity Diagram

4.3 Program Management Page Statechart Diagram

This statechart diagram is a detailed description of the Program Management Page. It shows the behaviour of the page depending on the action that is performed by the user, while using the web application. It can be seen as a complete description of the Program Management Page's functionality.

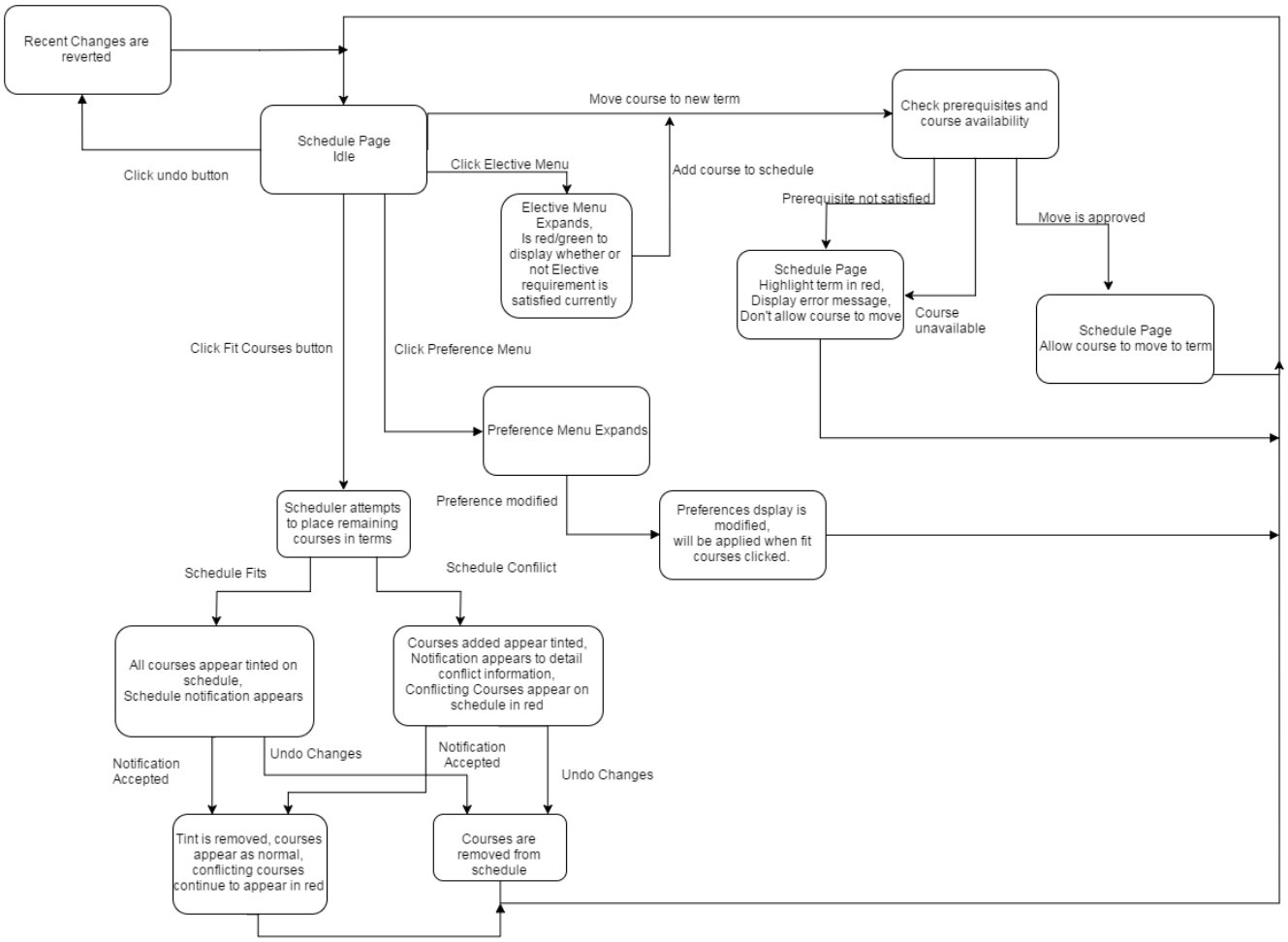

Figure 10: State Diagram for Program Management Page

4.4 Class Diagram

The class diagram is a representation of the classes used in Graduate, and their inter-relationships, functions and attributes. In order to achieve the desired Model-View-Controller(MVC) architecture, a number of design patterns will be used.

4.4.5 Observer Pattern

The observer pattern defines two acting classes, the subject and the observer. The subject maintains a list of dependent observers, and notifies them when a state change occurs. In this implementation of MVC, all views will be observers, and the models will be subjects. Therefore, when a model is changed, it will notify all dependent views of the change, and they will be updated accordingly.

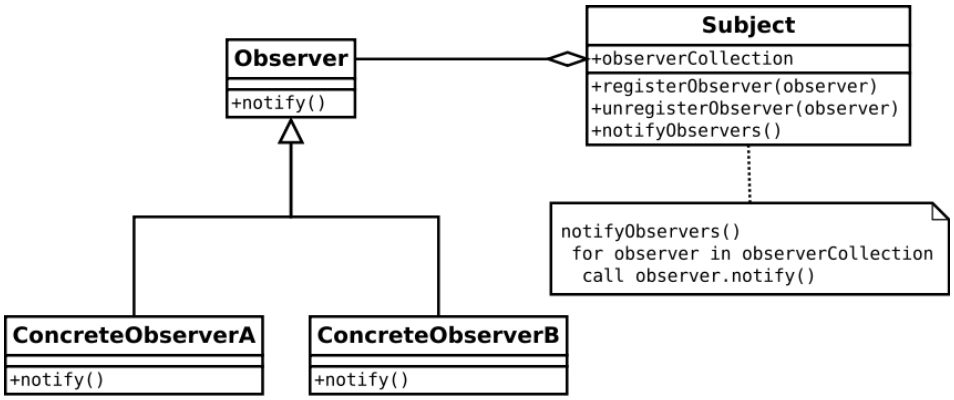

Figure 11: Observer Pattern Example

4.4.6 Façade Pattern

The façade pattern defines a façade object, which acts as a simplified interface to a large body of code. In order to facilitate the management of the various system models in our design, a model façade will be included which acts as an interface for accessing the models in the database. Therefore, the model façade contains a representation of the entire database. Controllers will make changes to the model facade, and these changes will be saved to the database.

4.4.7 Singleton Pattern

The singleton pattern ensures that prevents instantiation of a class. This ensures that only a single instance of a singleton class can exist. In order to ensure a consistent state for the model façade, it will be a singleton. This means that all users will interact with the same instance of the model façade.

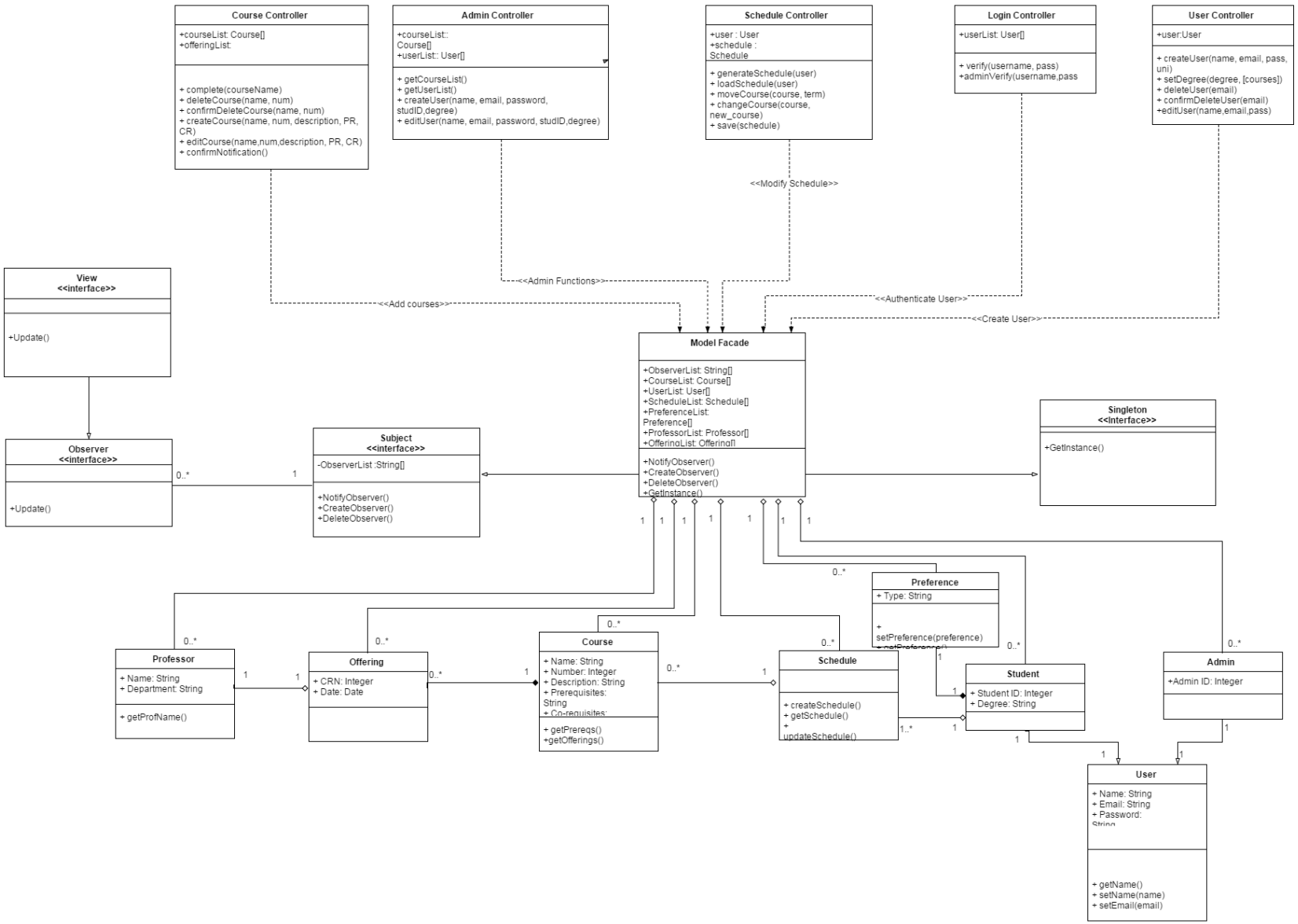

Figure 11: Graduate! Class Diagram