# mytinerary

CONCORDIA UNIVERSITY

SOEN 341: SOFTWARE PROCESS

TEAM ANOTHER ONE MYTINERARY

---

# Project Scope and Plan Document

---


*Group Members*

Piratheeban ANNAMALAI
Laurendy LAM
Jacqueline LUO
Michael MESCHEDER
Andy NGUYEN
Kenny NGUYEN
Ronnie PANG
Eric PAYETTE
Alessandro POWER
James TALARICO
Pragas VELAUTHAPILLAI
Anhkhoi VU-NGUYEN


April 14, 2016

# Contents

PART

# 1

# SCOPE AND PLAN

# 1 Project Description

Mytinerary is a web application designed to create, adjust and optimize the schedules of Concordia students. The schedule is created by the students who add courses offered by their respective program. Prior to the creation of the schedule, the user must select the according school term. In addition, the program will check whether or not the students have the correct pre or co-requisites prior to adding the class to the schedule. What this program differs from the current Concordia schedule making, is the friendly user interface. One major example is that it displays what the current schedule looks like while the student adds courses. The reason behind this is that the student can add courses while viewing their available schedule times rather than going back and forth viewing what time space is free. The application will also prohibit students from adding courses that is out of their course sequence or do not possess the correct pre/co-requisite. This application is also usable by teachers alike. Teachers can post their term schedule which includes the courses being taught or their free hours. This is viewed by the students who then join the course.

# 2 Goals and Constraints

The project goals can be divided into two categories. The first consists of mandatory goals which must be implemented in order to meet the minimum project requirements. The second category includes goals which, while not absolutely necessary, would enhance the quality of the project and set Mytinerary apart from other program planners. These secondary goals were set by asking ourselves what features *we* would look for in a program planner. Additionally, a survey targeting 200 Concordia software engineering students was conducted to gauge interest in proposed features, including multilingual support and integration with Rate-My-Professor. The data from this survey are unfortunately still being processed and could not be included in the present document.

## 2.1 Functional Requirements

The following use cases summarize Mytinerary's functional requirements. Each use case will deal with two possible **Actors:** a student, who will use the system to generate course schedules, and a program director, who can modify course and section information.

Difficult and importance are scored on a scale of 1 to 10 with 10 denoting extremely high difficulty/importance.

**Use case:** Log in
**Actors:** Student, program director
**Goal:** Gain access to website services.
**Summary:** User logs into the website by providing their username and password.
**Trigger:** User enters their username and password into the respective fields.
**Basic course of events:**

1. System prompts the user to log in when the user accesses the website.
2. User enters their username and password.
3. System verifies the login information.
4. User is logged in.

**Exception paths:**

1. System prompts the user to log in when the user accesses the website.
2. User enters their username and password.
3. System determines that either the username or password were entered incorrectly.
4. User is informed of the failed login attempt and is asked to re-enter their user credentials.

**Postconditions:** User is logged in, meaning that they have access to restricted content, including the user's private data. This is accomplished by giving the user an authentication cookie.
**Difficulty:** 3
**Importance:** 10

**Use case:** Log out
**Actors:** Student, program director
**Goal:** Prevent future users of the browser section to have access to user's credentials.
**Preconditions:** User must be logged in.
**Summary:** User logs out of the system, preventing anyone who uses their browser afterwards from having access to the site under the user's name.
**Trigger:** User indicates intention to log off.
**Basic course of action:**

1. User indicates intention to log off.
2. User is logged off of the system.
3. User is directed to the website's login page.

**Postconditions:** User is logged off, meaning that they no longer have access to restricted content such as the user's private data. This is accomplished by having the system no longer recognize the previous authentification cookie.
**Difficulty:** 2
**Importance:** 10

**Use case:** Return to home page.
**Actors:** Student, program director.
**Goal:** Return to the website's home page.
**Precondition:** User must be logged in. That is, user must have the required authentication cookie.
**Summary:** User indicates their intention to return to the home page in order to access all of the site's facilities.
**Trigger:** User indicates intention to return to the home page.
**Basic course of events:**

1. User indicates intention to return to home page.
2. User is brought to home page.

**Difficulty:** 2
**Importance:** 8

**Use case:** View academic record
**Actors:** Student
**Goal:** View personal academic record.
**Preconditions:** User must be logged in and in the home page.
**Summary:** User indicates that they wish to view their academic record, at which point the system retrieves and displays their academic record.
**Trigger:** User indicates that they wish to view their academic record.
**Basic course of events:**

1. User indicates intention to view academic record.
2. Academic record is presented to user.

**Difficulty:** 2
**Importance:** 7

**Use case:** View student's academic record
**Actors:** Program director
**Goal:** View the academic record of a selected student.
**Precondition:** User must be a program director. Student must exist in the system's database.
**Summary:** Program director requests access to academic record of selected student, which is then displayed.
**Trigger:** Program director indicates intention to view a student's academic record.
**Basic course of events:**

1. User indicates intention to view academic record.
2. System prompts user to enter either the name of the student or the student's Concordia ID number.
3. System verifies that user is a program director.
4. System verifies that specified student exists.
5. System retrieves student's academic record and displays it to the user.

**Difficulty:** 3
**Importance:** 6

**Use case:** View course listing
**Actors:** Student, program director
**Goal:** View a list of available courses.
**Summary:** A list of all available courses, grouped by program and sorted in numerical order,

is presented to the user.
**Trigger:** User indicates intention to see course list.
**Basic course of events:**

1. User indicated intention to see course list.
2. System retrieves list of courses and displays them to user in appropriate order.

**Difficulty:** 2
**Importance:** 8

**Use case:** View academic progress
**Actors:** Student
**Goal:** View current progress towards completing degree in Software Engineering.
**Precondition:** User is logged in as a user. That is, user must have the required authentification cookie.
**Included Use case:** View course listing.
**Summary:** When accessing list of courses, system will present a summary of user's progress towards their Software Engineering degree.
**Trigger:** User indicates intention to see course list.
**Basic course of events:**

1. User indicates intention to see course list.
2. System retrieves list of courses (see **Use case:** View course listing).
3. System accesses student's academic record to determine courses that have been completed.
4. System compares completed courses versus required courses.
5. System displays course list to user along with information highlighting their academic progress.

**Difficulty:** 4
**Importance:** 6

**Use case:** View course details.
**Actors:** Student, program director.
**Goal:** View details of specified course.
**Precondition:** Course exists in the system's database.
**Summary:** Users enters a course number to view details such as location, number of credits, prerequisites, etc.
**Trigger:** User enters course number into appropriate field.
**Basic course of events:**

1. User indicates intention to view course details.
2. User enters course number into appropriate field.
3. System retrieves course information and displays it to user.

**Exception paths:**
**Course does not exist:**

1. User indicates intention to view course details.
2. User enters course number into appropriate field.
3. System does not find any course matching that course number.
4. User is informed that the specified course does not exist, and is re-prompted to enter a course number.

**Difficulty:** 3
**Importance:** 8

**Use case:** Modify course details.
**Actors:** Program director.
**Goal:** Modify the details of a course.
**Extended Use case:** View course details
**Precondition:** User is a program director. Course exists in the system's database.
**Summary:** User specifies a course to modify. Properties of the course that can be modified include location, time, professor, class capacity, etc.
**Trigger:** User enters course number into appropriate field.
**Basic course of events:**

1. User indicates intention to modify course details.
2. User enters course number into appropriate field.
3. System retrieves course information and displays it to user.
4. System displays fields where user can edit course information.
5. User enters new course information.
6. System checks that user is a program director.
7. System modifies course information based on user input.

**Exception paths:**
**Course does not exist:**

1. User indicates intention to modify course details.
2. User enters course number into appropriate field.
3. System does not find any course matching that course number.
4. User is informed that the specified course does not exist, and is re-prompted to enter a course number.

**User is not a program director:**

1. User indicates intention to modify course details.
2. User enters course number into appropriate field.
3. System retrieves course information and displays it to user.
4. System displays fields where user can edit course information.
5. User enters new course information.
6. System confirms that user is not a program director. An error message is displayed and the user is logged off the system. This should not happen in practice, as the user should not be given the opportunity to even attempt to modify course information.

**Difficulty:** 3
**Importance:** 7

**Use case:** View news feed
**Actors:** Student, program director
**Goal:** Present users with a news feed.
**Precondition:** User is logged in and on the home page.
**Summary:** User is presented with headlines of news around the world.
**Trigger:** User is on the home page.
**Basic course of events:**

1. User is on the home page and is presented with headlines of current news.

**Difficulty:** 5
**Importance:** 2

**Use case:** Set schedule preferences
**Actors:** Student
**Goal:** User sets their preferences for the type of schedule they want.
**Precondition:** User is logged in.
**Summary:** User sets their schedule preferences, which may include their available times, days they cannot go to school on, number of credits they wish to take, etc.
**Trigger:** User requests to edit schedule preferences.
**Basic course of events:**

1. User requests to edit schedule preferences.
2. System presents user with an interface to set preferences.
3. User uses this interface to set their preferences, which the system then saves.

**Postconditions:** User preferences are saved in the database.
**Difficulty:** 3
**Importance:** 9

**Use case:** Select course for schedule
**Actors:** Student
**Goal:** User selects a course they wish to be added to their next schedule.
**Precondition:** User is logged in and they are eligible to take the specified course.
**Included use cases:** View course listing OR view course details.
**Summary:** User adds a course to be included in their next schedule.
**Trigger:** User selects a course to be included in their next schedule.
**Basic course of events:**

1. User selects a course to be added, either by searching specifically for the course (see use case View course details) or by selecting the course in the list of available courses (see use case View course listing).
2. System verifies that user is eligible to take specified course.
3. Course is added to a list of courses to be added in the next schedule.

**Exception paths:**
**User cannot take specified course:**

1. User selects a course to be added, either by searching specifically for the course (see use case View course details) or by selecting the course in the list of available courses (see use case View course listing).
2. System determines that user is not eligible to take requested course.

3. An error message is displayed to the user, informing them of the reason why the course could not be added.

**Postcondition:** Specified course has been added to a list of courses that must be included in any generated schedules.
**Difficulty:** 4
**Importance:** 9

**Use case:** Generate schedules
**Actors:** Student
**Goal:** System creates a series of schedules that meet student's preferences.
**Precondition:** User is logged in.
**Summary:** System generates schedules that meet student's preferences, include courses that student requested, and further student's progress towards meeting the graduation requirements.
**Trigger:** User indicates they wish to generate schedules.
**Basic course of events:**

1. User indicates they wish to generate schedules.
2. System finds all possible schedules that meet user's preferences and incorporate user-selected classes.
3. The generated schedules are presented to the user.

**Postcondition:** The set of possible schedules are stored by the system in memory.
**Difficulty:** 6
**Importance:** 10

**Use case:** Select schedule
**Actors:** Student
**Goal:** User selects one of the system-generated schedules to be their actual schedule.
**Precondition:** User is logged in and has generated schedules.
**Extends use cases:** Generate schedules
**Summary:** User selects the schedule they find most desirable.
**Trigger:** User selects and commits a schedule.
**Basic course of events:**

1. User generates schedules.
2. User reviews schedules and selects one they find desirable.
3. System stores the chosen scheduled and updates user's record.

**Postcondition:** The selected schedule is stored in the system's database.
**Difficulty:** 4
**Importance:** 10

**Use case:** Change language
**Actors:** Student, program director
**Goal:** User changes the website language to French or English.
**Summary:** User selects a different language for the website's content to be delivered in.
**Trigger:** User selects a different language.
**Basic course of events:**

1. User selects a different language.
2. System records language change and delivers website content in new language.

**Postcondition:** The system records the user's language preferences and will deliver the website content in the user's chosen language.
**Difficulty:** 8
**Importance:** 3

**Use case:** View professor profile
**Actors:** Student
**Goal:** User views a professor's profile on Rate-my-Professor to determine whether they want to take course taught by them.
**Summary:** System will include easy access to a given professor's Rate-my-Professor profile.
**Trigger:** User selects the professor who's profile they wish to view.
**Basic course of events:**

1. User selects a professor.
2. System accesses ratemyprofessor.com, fetches professor's profile, and displays it to user.

**Difficulty:** 9
**Importance:** 2

Figure 1 graphically displays the above use cases and the relationships between them.

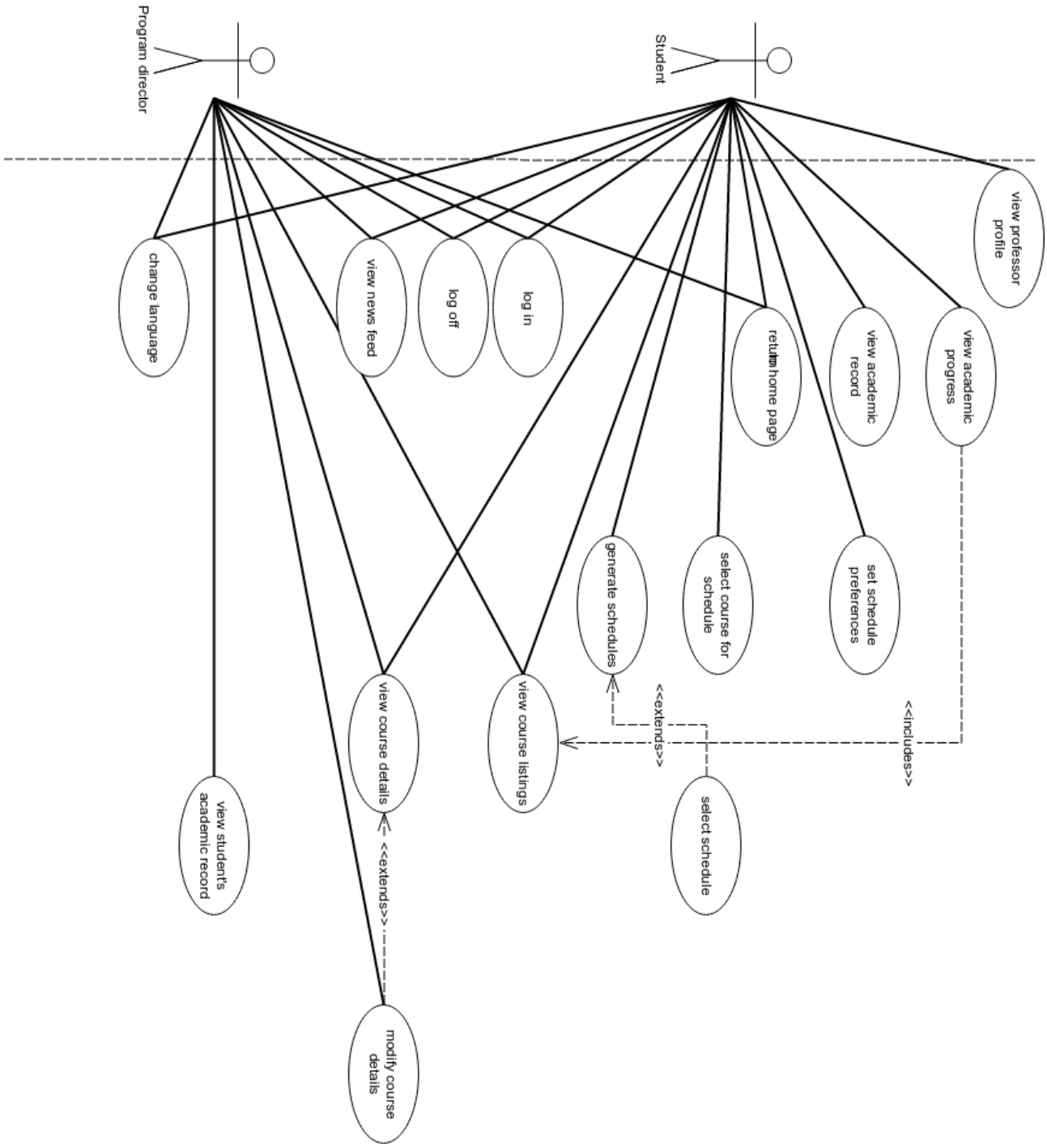



Figure 1: Use case diagram for the Mytinerary scheduling system

## 2.2 Domain Model

Figure 2 illustrates a domain model for our system. The principal domain objects are the student, the course section, and the scheduler itself. Students, who are users, have preferences and an academic record and interact with the scheduler. The scheduler in turn consults the student's academic record and preferences, the requirements of the program the student is enrolled in, and the available course sections offered in each semester to generate a list of tentative schedules for the student to select. Course sections consist possibly of a tutorial section and a lab and are in turn instances of the more general course object. Finally there are program directors, who are users of the system with the privilege to view and modify student academic records and manage course details. A detailed description of the principle domain objects is given below.

- **User:** A user has a login name and a password.
- **Student:** Students, who are users, have preferences and an academic record and interact with the scheduler. They can also view a semester and select a generated schedule.
- **Academic Record:** A student's academic record holds their program, option, academic standing, GPA, completed courses, etc.
- **Preferences:** Collection of time slots that the student cannot have classes during.
- **Semester:** A semester has a list of courses that are offered in that semester.
- **Scheduler:** Responsible for actually generating schedules for the students. Can view a student's preferences and academic record.
- **Schedule:** List of course sections.
- **Course:** Has a name, subject, credits, etc. Has a list of courses as prerequisites.
- **Section:** Is a course. Has additional details, such as room number, time slot, etc. Has tutorial and lab sections, if applicable. item **Program director:** Is a user. Manages course information, section information, lab information, and tutorial information. Can view student information, such as a student's academic record.

## 2.3 Constraints and Qualities

The non-functional requirements that Mytinerary will meet can be divided into three categories: security, performance, and usability.

**Security**

- Mytinerary shall ensure that login is protected using a strong hashing algorithm. This ensures that user data is kept safe if there is a partial database breach. The blowfish hashing function is provided by PHP's hashing function 'password_hash()' which generates a unique hash of a password. Unlike traditional md5 hash, hashing would produce two identical hashes if the same passwords was used. Using md5 would be valuable if a dictionary attack is performed.
- Login timeouts will be implemented to discourage brute-force attempts at compromising a user's account. Specifically, Mytinerary will mandate a five minute wait if a user's password was incorrectly entered 10 times.
- Mytinerary shall ensure that data transactions between the application and server are secure by the blowfish encrypting algorithm. The scheduler is one page user interface which means that when passing serialized data of PHP objects they are vulnerable to changes if the client decides to alter the string. Thus encrypting the string would prevent any alternation and stay consistent.
- Forms are validated automatically through CodeIgniter's form validation library. It filters html escape strings and cross site scripting (xss) attacks.
- Server-client communication will be done using the HTTPS protocol.

**Performance**

- A page shall load in under a second, as tested on a Concordia computer using Concordia University's network.
- The schedules shall be generated in under five seconds, as tested on a Concordia computer using Concordia University's network.

**Usability**

- The web application should be cross-platform compatible usable through various mobile, tablet, and desktop systems. Various browsers support like the Google Chrome, Safari and edge by Microsoft. With the help of normalize it will render HTML and CSS consistent across all web browser. Mytinerary will be enable to support Legacy web browsers with the help of Modernizer. A library writing in JavaScript aids the browser with unsupported features.
- There will be a minimum of ten preferences that a user can set in order to tailor the schedule generator to their individual needs.
- There will be a maximum of three mouse clicks that a user will have to make in order to use any of the website's functions. This ensures that Mytinerary is easy to use for users of varying computer skills.

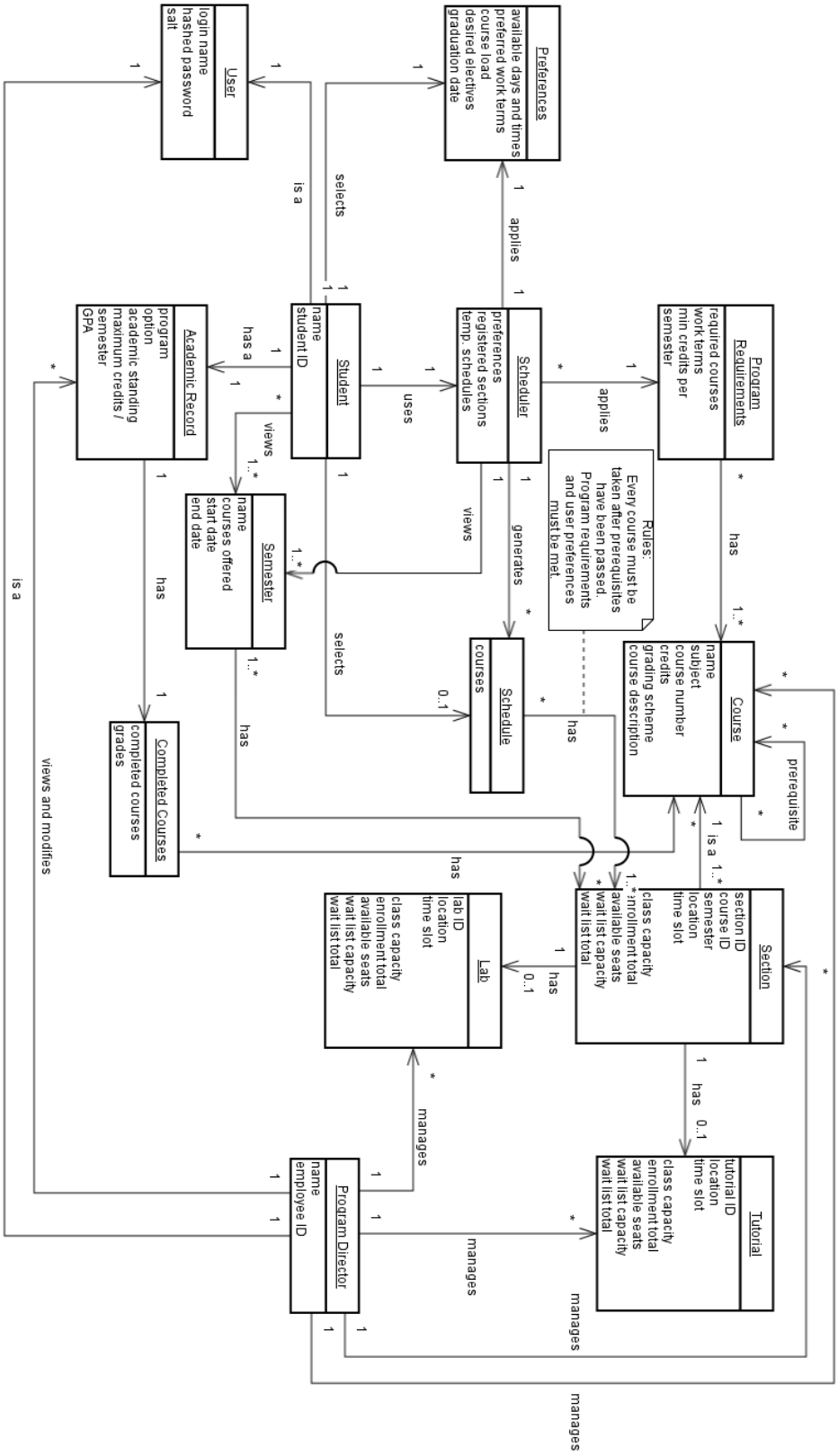



Figure 2: Domain Model for the Mytinerary scheduling system

# 3 Resource Requirements

## 3.1 Resource Evaluation

| **Member Name** | ***Laurendy Lam*** |
|---|---|
| **Role** | Programing Full-Stack, Programming Leader, Co-Team Leader |
| **Strengths** | – Web Design<br>– User interface<br>– Algorithm<br>– Team Work / Organization<br>– Database Design<br>– OOP |
| **Knowledge** | Java, C, C++, PHP, HTML5, CSS3, Javascript, SQL, Python, Ruby, Prolog, Lisp |
| **Experience** | – Personal website developer<br>– Design/Developed a restaurant website |
| **Availability** | 10-12 hours /week |

| **Member Name** | ***Alessandro Power*** |
|---|---|
| **Role** | Documentation Leader , Team Leader |
| **Strengths** | – Communication<br>– Team Work<br>– Object-oriented Programming |
| **Knowledge** | C++, Java, Python |
| **Experience** | Programming in Java for various class assignments. |
| **Availability** | 7 hours/week |

| **Member Name** | ***Anhkhoi Vu-Nguyen*** |
|---|---|
| **Role** | Programming Back-End |
| **Strengths** | – Object-oriented Programming<br>– Team Work<br>– Communication |
| **Knowledge** | Java, C++, HTML5, Prolog, PHP, Lisp, CSS3, Assembly , AspectJ |
| **Experience** | – Programming in Java for various class assignments |
| **Availability** | 6 hours/week |

| **Member Name** | ***Jacqueline Luo*** |
|---|---|
| **Role** | Testing |
| **Strengths** | – OOP with Java<br>– App development for Android devices |
| **Knowledge** | Java, C#, HTML , CSS3, PHP, JavaScript |
| **Experience** | Developed an app on Google Play Store |
| **Availability** | 6 hours/week |

| **Member Name** | ***James Talarico*** |
|---|---|
| **Role** | Documentation |
| **Strengths** | – Object-Oriented Design<br>– Communication and teamwork<br>– Document Writing |
| **Knowledge** | Java, C, C++ , Python , OCaml, HTML5, CSS3, MATLAB, Bash |
| **Experience** | – Developed a cluster search algorithm to parse Fermi LAT data for uncatalogued VHE sources.<br>– Configured McPHAC to run on Calcul Quebec's super computers. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Kenny Nguyen*** |
|---|---|
| **Role** | Documentation |
| **Strengths** | – Object-oriented Programming<br>– Team Work<br>– Communication |
| **Knowledge** | Java, PHP, HTML5, CSS3, MySQL, JavaScript |
| **Experience** | – Programmer at Industry Canada |
| **Availability** | 6 hours/week |

| **Member Name** | ***Michael Mescheder*** |
|---|---|
| **Role** | Programming Full-Stack |
| **Strengths** | – Client side<br>– Object-Oriented Programming<br>– Teamwork |
| **Knowledge** | Java, PHP, HTML5, CSS3, JavaScript |
| **Experience** | – Programming in Java for various class assignments. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Piratheeban Annamalai*** |
|---|---|
| **Role** | Programming |
| **Strengths** | – Server side<br>– Object-Oriented Programming<br>– Organization |
| **Knowledge** | Java, PHP, HTML5, CSS3, JavaScript |
| **Experience** | – Various programming assignments for class assignments involving Java, and Php. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Pragas Velauthapillai*** |
|---|---|
| **Role** | Testing |
| **Strengths** | – OOP<br>– Team Work |
| **Knowledge** | Java, PHP, HTML, CSS, Javascript |
| **Experience** | -Programming in Java for various class assignments. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Ronnie Pang*** |
|---|---|
| **Role** | Programming Full Stack |
| **Strengths** | – OOP<br>– Team Work |
| **Knowledge** | Java, PHP, HTML, CSS, Javascript |
| **Experience** | – Programming in Java for various class assignments. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Eric Payette*** |
|---|---|
| **Role** | Programming Full-Stack |
| **Strengths** | – Object-oriented programming<br>– Functional programming<br>– Logic programing<br>– Team Work |
| **Knowledge** | Java, PHP , HTML , CSS3, MySQL , JavaScript, Lisp, Prolog, AspectJ |
| **Experience** | – Programming for various class assignment. |
| **Availability** | 6 hours/week |

| **Member Name** | ***Andy Nguyen*** |
|---|---|
| **Role** | Documentation |
| **Strengths** | – Object-oriented Programming<br>– Team Work |
| **Knowledge** | Java, HTML5 , CSS3, JavaScript , PHP |
| **Experience** | – Programming in Java for class assignments<br>– Developed a Real-Estate website for a class project |
| **Availability** | 5 hours/week |

## 3.2 Technical Resources

### 3.2.1 Server

The web server used to host the developing project is the PHP built in web server provided by PHP which come with WAMP. It comes bundle of different other tools such as PHP-myAdmin, Apache and SQL buddy for local hosting.

### 3.2.2 Database

The hosting service for the database on a remote server is Heroku. To manage the database we are using dbForge studio for MySQL or PHPMyAdmin version 4.5.3 used with WAMP to run the app to access the database remotely.

### 3.2.3 Hardware

The average computer to host the working project is an Intel i5 or similar. 2 GB of ram and uses around 30 MB of storage space.

### 3.2.4 System

The operating system used to operate the program as very portable as long as it has PHP, Apache and a MySQL server install installed. Version and course control is handled by a private git repository.

### 3.2.5 Programming Languages

The language used to develop this project is PHP, JavaScript, and CSS. The framework used to develop the project is CodeIgniter v3.03 along with other user interface frameworks such as JQuery v3.03, Bootstrap 4, Normalize v3.0.2 and Google Fonts. The editors to develop the project are: notepad++ or PHPStorm by JetBrains. Refer to section 5.2 for a more extensive list of the technologies in use.

### 3.2.6 Communication

We are using slack to communicate, Google drive to share files and documents. and GitHub to upload finished documents. bug reporting on git hub. For the documentation we are using LaTeX. We use this to produce a nice compilation of all the documents collected throughout the project.

# 4 Scoping

Based on the resource constraints listed above, we will not be implementing the Rate-my-Professor integration unless time permits. Due to the lack of a publicly available API, integrating Mytinerary with Rate-my-Professor's services would be time-consuming to implement, and it is not essential to the project's success. Likewise, support for the French language will not be included. It can be safely assumed that any Concordia student can speak English sufficiently well to navigate an English-based program scheduler website.

# 5 Solution Sketch

## 5.1 Architecture

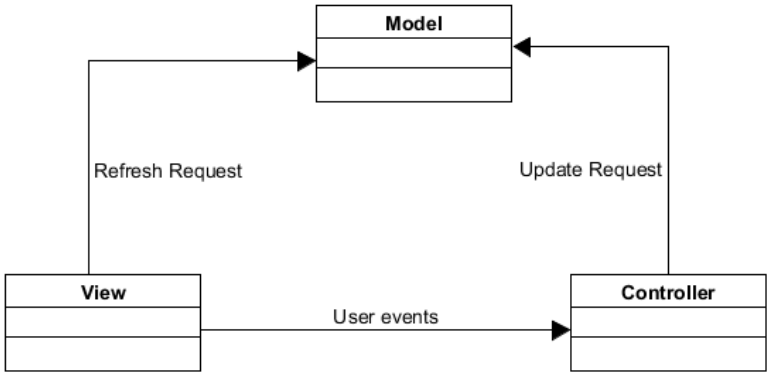



Figure 3: Diagram of the Model View Controller which Mytinerary is based on.

Mytinerary is based off of the Model-View-Controller (MVC) architecture. We are using MVC because it encourages a clear separation between the data and the methods that manipulate the data. The gives our program the advantage of allowing a front-end and back-end developer to work on the project at the same time, which out having to worry about their respective code interfering with one another.

As the name suggests, the Model-View-Controller pattern can be broken down into three layers: Model, View and Controller. These layers are what create an abstraction between components of the system as well as a layer of abstraction between the system and the user. A diagram of the interactions of the MVC can be seen in Figure 3. The three layers are described below.

**Model**
The Model is the component that has access to where the data used in the program is stored. The model acts as a sort of bridge between the view and controller component. It updates the View on pertinent changes that have been made, and allows the Controller to manipulate it's data content.

**View**
The View is where the information requested from the Model layer can be viewed. In our system, this would correspond to Mytinerary's HTML page. All data received by the View is given upon requesting it from the Model. This happens when a method from the Controller changes data in the Model. The Model then informs the View of these changes.

**Controller**
The job of the Controller is to take in any information that is passed by the user, and update the Model accordingly. The Controller does not always interact with the View directly, but instead changes the data controlled by the Model, which is then seen by the view. It can communicate directly with the view in order to change how the view presents the model. It is important to note that changes made by the Controller are initiated by user interactions.

## 5.2 Technologies in Use

### 5.2.1 Programming Languages

**PHP**
We decided to use PHP as the code base for Mytinerary. It is a quick and easy server side scripting language. PHP has a large community, which meant that most problems faced by developers would have preexisting solutions. Another important advantage is that PHP is part of the development platform WAMP, as described in section 5.2.3, which made installation easy. We decided to use the most recent version, PHP 7.02.

**JavaScript**
JavaScript is a high-level, dynamic and untyped programming language. Since JavaScript is commonly used for web development, it is supported by most browsers, including: Internet Explorer, Chrome, Firefox and Safari. In order to transfer data, we will be using JSON (JavaScript Object Notation) in place of the more common XML. This choice was motivated by the fact that JSON is more lightweight, making network transmissions and read/writes faster. JSON is also far more human-readable compared to XML. We used the stable release version ECMAScript 6.

**HTML**
HTML (Hyper Text Markup Language) is a markup language used for creating web pages. It is by far the most commonly used markup language, and therefore has the most cross-platform support. We used the most recent version HTML5, which is quickly rising in popularity.

**CSS**
CSS (Cascading Style Sheets) is a style sheet language used for creating a layout and styling a web page. Like JavaScript and HTML, CSS is one of the more common languages used in web development. CSS is used primarily to allow the separation of a web page's content from it's presentation; this makes code cleaner and more reusable. We are using CSS3, the most recent version.

**SQL**
SQL (Structured Query Language) is a special-purpose programming language used to ac-

cess and manipulate databases. We will be using it to manage our MySQL database (described in section 5.2.3, under MySQL). We chose SQL since it makes retrieving large amounts of records from databases quick and efficient it is the standard language for this task. We are using the most recent version, SQL:2011.

### 5.2.2 Libraries and Frameworks

**Bootstrap**
Bootstrap is a HTML, CSS, and JavaScript framework used for developing web pages. We chose Bootstrap as it is easy to use and has a large community with extensive documentation. Furthermore, Bootstrap supports the most popular browsers and as well as fixes some compatibility issues with CSS. We are using the most recent stable release, Bootstrap 3.3.6.

**CodeIgniter**
CodeIgniter is an open source PHP framework used in web development. It is loosely based on the Model-View-Controller pattern. We chose CodeIgniter as it is easy to use compared to other PHP frameworks and was the framework that our developers were most familiar with. We are using the most recent stable release, 3.0.4.

**AJAX**
AJAX (Asynchronous JavaScript and XML) is a set of client-side web development techniques used to create asynchronous Web applications. We chose AJAX over AJAJ (which uses JSON) for several reasons, the first being our programing team was more familiar with it. AJAJ also has a lack of documentation and is not able to perform two-way data transmission, so AJAX was the clear choice.

**jQuery**
jQuery is a JavaScript library which is designed to simplify client-side HTML. We chose jQuery as it is easy to use, especially compared to other JavaScript libraries. It is an extensive library that effortlessly supports AJAJ. We are using jQuery 1.12.0.

**Normalize.css**
Normalize.css is used to make built-in browser styling consistent across browsers. It also has a very extensive documentation and excellent support, which was very helpful for some of the members which were not familiar with the tool. We are using version 3.0.3.

**Moment.js**
Moment.js is a simple date and time library that enables us manipulate time and dates very easily. It comes with extensive documentation and a variety of features, unlike JavaScript's native date library.

### 5.2.3 Web server and Databases

**Apache**
Apache is an open-source web server software. We chose it because it is currently the most popular web server software and is the one we were most familiar with. We will be using the stable release, 2.4.18, which is the most recent version.

**Heroku**
Heroku is a Platform-as-a-Service (PaaS), which allows it's users to host a database on a remote server. We chose Heroku because other web hosting services were not working properly with Mytinerary. Heroku was user-friendly and had extensive documentation making it easy to learn.

**MySQL**
MySQL is a Database Management system which uses relational databases. We chose MySQL as it is easy to use, very popular and has excellent support. We also chose it because our programming team was very familiar with it. We will be using the most recent stable release, 5.7.10.

**PHPMyAdmin**
PHPMyAdmin is a web based tool that works to help the user handle the administration of MySQL. We chose it because of it's simplicity and our familiarity with software. We are using the most recent stable release version, 4.5.4.1.

**dbForge Studio**
dbForge Studio is a windows tool that is made for management and development of an SQL server. We chose this tool because are our familiarity with it. We are using the most recent version, 5.1.178.

**WAMPServer**
WAMPServer is a windows development environment which uses Apache, MySQL, and PHP. Since we had already decided on using PHP, Apache and MySQL for our project, using this tool to simplify the setup of our server was the obvious choice. We are using the most recent stable release version, 4.5.4.1.

### 5.2.4 IDEs

**Notepad++**
Notepad++ is a simple, lightweight text editor that has support for a large range of languages. We chose it because it is a very popular cross-platform text editor and our programming team was very familiar with it. We are using Notepad++ v6.8.8.

**PHPStorm**
PHPStorm is an IDE which handles PHP, HTML and JavaScript. We chose it because it is cross-platform and was the IDE our programming team was the most familiar with it. We are using version 10.0.3.

### 5.2.5 Documentation and Collaboration Software

**Slack**
Slack is an online messaging application that can be used on mobile or PC. We chose to use Slack since it made communication and collaboration more convenient and organized.

**GitHub**
GitHub is an online Git repository service. It is currently the most popular Git based system, and it virtually essential for version control in large programming project. GitHub is an excellent tool for organizing and collaborating, and is an industry standard.

**ShareLaTeX**
ShareLaTeX is an online LaTeX editor and compiler. We chose to use LaTeX because it's capability of producing high typographical quality documentation. We chose ShareLaTeX since it allowed the documentation team to collaborate easily and it they were already quite familiar with it.

**Gliffy**
Gliffy is an online web diagram editor with was used to make all UMLs and Domain Models in during the project. We chose it because it is easy to use and can produce high-quality diagrams.

# 6 Plan

## 6.1 Activities

The following are the main activities that have been or will be undertaken to reach the project goals.

### Planning Activities

**Division of Roles:** The purpose of this activity is to determine which team members will assigned to each of the project roles, which includes documentation, implementation, and testing. The result of this activity will be a list of proposed roles for each team member, which will be included in the first deliverable.

**Survey:** A survey of Concordia Software Engineering students will be conducted to gauge user interest in proposed features. The results of the survey will be included in the second deliverable.

**Setting Project Requirements:** The purpose of this activity is to fix the project requirements. These include mandatory features necessary to meet the minimum project requirements, as well as additionally features which would enhance Mytinerary's overall quality. These requirements will be collected to form a list of requirements, which if necessary will be scoped down.

**Risk Analysis:** Risk analysis will be undertaken to identify those events and circumstances that will put the project's success in jeopardy should they occur. The purpose of risk analysis is to assist us in initiating proactive measures to reduce the probability that these events take place. The risk analysis will be included in the second deliverable.

### Design Activities

**Create Domain Model:** A domain model consists of high-level overview of the domain level objects, their attributes, and the associations between them. The purpose of creating a domain model is to assist the high-level organization of the project. The domain model will be included in the first deliverable

**Specify Use Cases** The project's expected functional requirements will be organized into a series of use cases and graphically displayed in a use case diagram, which will be included in the second deliverable.

**Specify Non-functional Requirements:** As an extension of the *setting project requirements* activity and a complement to the *specify use cases* activity, a list of non-functional requirements will be made and included in the second deliverable.

**Resource Evaluation and Scoping:** The purpose of this activity is to determine the human and technical resources available to our project, which will be used to scope down the project goals if necessary. The resource evaluation and scoping will be included in the second deliverable.

**Create Solution Sketch:** The solution sketch presents an overview of the how we chose to implement the project requirements. This includes both the high-level project architecture as well as the specific technologies what will be used. The solution sketch will be included in the second deliverable.

**Plan:** The purpose of this activity is to create a schedule for the project to ensure that all requirements can be implemented in time for the final delivery date. The project plan will be included in the second deliverable.

## Implementation Activities

**Create GitHub Repository** The creation of a GitHub repository facilitates the integration of code and project documents that individual team members create.

**Setup Website Server:** The initialization of a web server on Heroku to host our website is a necessary step in the creation of prototypes and eventually the final product.

**Setup mySQL Database:** The system's database will have to be populated with course, student, and user information in order for the system to work effectively.

**Create Architecture Design Diagrams:** Architecture design diagrams, including the 4+1 Architectural View and UML diagrams, will be created to facilitate the detailed design of the system. These diagrams will be included in the third deliverable.

**Create and Test Prototypes 1, 2, and 3:** The system will be delivered in three prototypes, each adding increasing functionality. The delivery of a particular prototype will be followed by rigorous testing to ensure the prototypes meets the specified requirements. The third prototype will be tested with actual Concordia students (outside of our own group), and their feedback will be incorporated into the final delivery.

**Create and Test Final Delivery:** After the third prototype has been completed and tested, the final delivery will be assembled and tested before being pushed into production.

## 6.2 Artifacts

The list of artifacts this project has and will produce are tabulated below. The target date for each artifact, as well as the team members responsible for its creation, are included as well. Note that cost has *not* been included as each artifact should be free to produce, given that we are exclusively using free tools in our project.

| Artifact | Artifact Description | Target Date | Assigned to |
|---|---|---|---|
| List of Proposed Roles | Tentative list of the roles each team member has been assigned to. | Jan. 8 ✓ | All |
| Survey Results | Results of survey conducted by Concordia Software Engineering students on their interest in proposed features. | Feb. 12 ✓ | A. Power |
| Project Requirements | Tentative list of requirements for the project. Will be refined in the use cases and non-functional requirements artifacts. | Jan. 15 ✓ | All |
| Risk Analysis | List of possible events that would impede project progress, along with an estimate for their likelihood and projected severity. | Feb. 5 ✓ | J. Talarico |
| Domain Model | Domain model for project. | Jan. 10 ✓ | L. Lam, A. Power, J. Talarico |
| Deliverable 0 | System overview and team members document. | Jan. 13 ✓ | L. Lam, A. Power, J. Talarico |
| Use Cases and Use Case Diagram | Summary of functional requirements through use cases; graphical representation of the relationships between use cases through a use case diagram. | Feb. 2 ✓ | A. Power |
| Non-Functional Requirements | Non-functional requirements to be implemented. | Feb. 6 ✓ | K. Nguyen |
| Resource Evaluation | List of resources available – both human and technical. Scoping down of project requirements based on resource limitations. | Feb. 6 ✓ | A. Nguyen, A. Power |
| Solution Sketch | High-level overview of project implementation. Includes description of high-level system architecture and technologies used. | Feb. 2 ✓ | J. Talarico |
| Plan | Project schedule. Includes artifacts to complete and target dates for their completion. | Feb. 7 ✓ | A. Power |

| | | | |
|---|---|---|---|
| Deliverable 1 | Project scope and plan document. | Feb. 10 ✓ | P. Annamalai, A. Nguyen, K. Nguyen, A. Power, J. Talarico, A. Vu-Nguyen |
| GitHub repository | Remote git repository for hosting project files. | Jan. 8 ✓ | L. Lam |
| Web Server on Heroku | Webserver for hosting content of our website. | Feb. 7 ✓ | L. Lam, R. Pang |
| mySQL Database | Populated database with course, student, and user information. | Feb. 5 ✓ | J. Luo, P. Velauthapillai |
| Architecture Design Diagrams | Architecture design diagrams include the 4+1 Architectural View and UML diagrams. | Mar. 2✓ | P. Annamalai, A. Vu-Nguyen |
| Prototype 1 | First prototype. Features to include: student login, password hashing with blowfish algorithm, news feed, course searching, user setting of some preferences, student viewing of academic record, schedule generation, return to homepage by clicking Mytinerary logo. | Mar. 2 ✓ | P. Annamalai, L. Lam, J, Luo, M. Mescheder, R. Pang, E. Payette, P. Velauthapillai, A. Vu-Nguyen |
| Deliverable 2 | Project design document. | Mar. 8✓ | P. Annamalai, A. Nguyen, K. Nguyen, A. Power, J. Talarico, A. Vu-Nguyen |
| Prototype 2 | Second prototype. Features to include: program director login, ability for program director to modify courses, more preferences that student can set, full encryption of communication between server and client, ability of students to view academic progress. | Mar. 15 ✓ | P. Annamalai, L. Lam, J, Luo, M. Mescheder, R. Pang, E. Payette, P. Velauthapillai, A. Vu-Nguyen |

| | | | |
|---|---|---|---|
| Deliverable 3 | Project implementation and testing document. | Mar. 21✓ | P. Annamalai, A. Nguyen, K. Nguyen, A. Power, J. Talarico, A. Vu-Nguyen |
| Prototype 3 | Third prototype. Implement remaining features. | Mar. 25 ✓ | L. Lam, J, Luo, M. Mescheder, R. Pang, E. Payette, P. Velauthapillai |
| Final delivery | Incorporate any missing features from prototype 3, as well as bugs and deficiencies revealed through testing of prototype 3. | Apr. 1 ✓ | L. Lam, J, Luo, M. Mescheder, R. Pang, E. Payette, P. Velauthapillai |
| Final deliverable | Combination of previous deliverables. | Apr. 3 ✓ | P. Annamalai, A. Nguyen, K. Nguyen, A. Power, J. Talarico, A. Vu-Nguyen |

## 6.3 Risk Analysis

The perceived likelihood of the risks can be measured by Low, Medium or High. The effects of a risk can be broken up into four categories: insignificant, tolerable, serious and catastrophic.

| **Risk Type** | **Risk Description** | **Probability** | **Effects** |
|---|---|---|---|
| People | The group relies on one particular programmer to help with troubleshooting the more unfamiliar software. If he were to become seriously ill, or had to withdraw from the project due to personal reasons, the project's progress would slow dramatically. | Medium | Catastrophic. |
| Estimation | The project has been given 3 months to complete. The length of time allotted to each step must be properly estimated because if the deadline is not met, we will not be given an extension. | Medium | Catastrophic |

| | | | |
|---|---|---|---|
| Technology | Only one person is familiar with the CodeIgniter framework that is being used. Although it does not have a steep learning curve, it may take some programmers time before becoming fully familiar with the tool. This can cause the project to take more time than expected. | High | Serious |
| Technology | Not all our programmers are familiar with PHP. This could result in result in the project taking longer than expected, since some members cannot contribute to that portion. | Low | Serious |
| Organization | There are a large number of people collaborating in this project. If there is any hostility between members, the groups communication and cooperation may be negatively effected. | Low | Serious |
| Security | Our scheduler site uses HTTP, which is not a secure protocol. This means that anyone could potential intercept data being transmitted. | Low | Serious |
| Technology | Only two people in the documentation team are familiar with LaTeX. This may create a time bottle-neck since all of the writing done must be sent to two people, who can then properly format the documentation in LaTeX. | High | Tolerable |
| Organization | Communication between members is key to a seamless project. Some members may not be reachable for period of time, due to an unwillingness to cooperate. | Medium | Tolerable |
| Organization | If our scheduler receives an overwhelming number of requests users will experience high latency, which will negatively effect their experience. | Low | Tolerable |
| Technology | Our programmers collaborate using GitHub. It the service goes down temporarily, our project comes to a halt as no one can access the source code. | Low | Tolerable |
| Technology | All of our group communication is done through Slack. If the site goes down, members will not be able to communicate will one another. However, because there are many other communication platform, this is not serious. | Low | Insignificant |
| Technology | Our site is being hosted using Heroku. If the serivice were to go down, users would not have any way of accessing our scheduler. However, because there are many other hosting companies, this is not serious. | Low | Insignificant |

# 7 Prototyping

The following sections describes the features that will be implemented for the first prototype. The current status of this prototype can be view at http://mytinerary.herokuapp.com; the reader is encouraged to view the website in its current state. A test user with the username ”john_smith” and password ”password” can be used to explore the features of the site. The source code for the current prototype is included in Appendix A. Upon entering the website's URL the user is presented with a login page. The login page features a brief description of the current situation. For instance, any course issues or service downtime would be displayed here. The users also have the ability to view courses in any given semester without needing to log in. The login system security is handled by hashing the entered username and password using the Blowfish algorithm and sent to the database for server side validation. Once logged in, the users are sent to the default page with various options presented to them. Additionally, there is also a section containing relevant news that will updated on a regular interval. This section is currently created by using JSON and the New York Times API (may be modified later on).

The first selectable option is the student profile. This page shows a record of the student. It indicates which courses have been completed and the GPA. Also, this page is used to determine which prerequisites and co-requisites have been completed before a student can register for a class that has those specific prerequisites and co-requisites.

The second selectable option leads to the registration page. Here, students can search for courses in a selected semester in order to generate schedules. There is also an option to auto-generate schedules that is currently in development. Students can apply time preferences to generate their ideal schedule. Up to five different schedules can be displayed on the same page using client side JavaScript and jQuery to dynamically generate the schedules as the user selects or edits information. The students can then select which schedule they would prefer to have by comparing all of them. Students can add courses to the courses tab. However, in the event that they lack the necessary prerequisites or co-requisites, they will not be able to add the course onto the registered tab. This validation is handled by dynamically checking the students registered courses within the database. The courses shown on the right-hand side represent what the students are involved with. If the course name is within a green rectangle, then the student will be currently registered for that class. If the course name is within a yellow rectangle, then the student will be on a waiting list for that class. Finally, if the course name is within a red rectangle, then it indicates that the student has removed that class from the current semester.

At any time, users can click on the Mytinerary logo on the top left corner to be redirected to the home page. At the home page students can also view available courses as well as their current schedule. Mytinerary is cross-platform compatible in addition to featuring a responsive layout and automatic scaling. This is achieved using HTML5 and Bootstrap as the front-end framework. Any APIs to be used will also avoid being platform specific to ensure usability across all devices.

PART

# 2

# ARCHITECTURE AND DESIGN

# 8 Introduction

This document summarizes the design and behaviour of the Mytinerary web application. The overall architecture is described using a 4+1 diagram, which presents the design of the system through four views: a logical view, using a class diagram; a development view, using a component diagram; a physical view, using a deployment diagram; and an process view, using an activity diagram. The system was further divided into four subsystems: the authentication subsystem, the course management subsystem, the database subsystem, and the scheduler subsystem; a detailed design for each subsystem in the form of a class diagram is included in the document, along with a specification of the interfaces between subsystems.

The behaviour of the system was illustrating with a system sequence diagram and a sequence diagram for two use cases: the ”add course” use case and the ”generate schedules” use case. Each use case was further accompanied by operational contracts specifying the assumptions and guarantees that system operations can make.

The document concludes with an estimation of the cost of the project using the COCOMO software, as well as a summary of the current progress towards a working prototype, the testing of said prototype, and effects of the prototype on our risk assessment and scoping.

# 9 Architectural Design

## 9.1 Architecture Diagram

This section contains a 4+1 Architectural View depicting the high-level structure of the system. The 4+1 Architectural View contains a logical view, a development view, a physical view, and a process view. The four views above are illustrated using a small set of use cases, or scenarios.

Mytinerary has been divided into four subsystems: the Authentication subsystem, the Course Management subsystem, the Database subsystem, and the Scheduler subsystem. The interfaces between the four components are presented in section 2.2, and their detailed designs are illustrated and discussed in section 3.

### 9.1.1 Logical View

The logical view presented in Figure 4 consists of a class diagram of all architecturally significant classes in the system. Certain classes, such as the view and controller classes, were omitted from the diagram for simplicity's sake, in addition to certain methods whose existence are unnecessary for understanding the design of the system. Note that the classes shown below include both PHP classes as well as database objects.

The source code for the PHP classes can be found in Appendix A.

The classes shown in Figure 4 include

- **AcademicProgram:** Represents an academic program, such as Software Engineering. Contains basic information about the program, such as the program's name, faculty, and number of credits, as well as the required and optional courses.
- **AcademicRecord:** A database object for holding the academic record of a Student. It consists of an array of Course objects representing the courses a student has taken, as well as a Grade object for each course. The Grade class was not presented in this class diagram.
- **Course:** A class for representing a course. Contains the course's name, subject, course number, number of credits, and the course description, as well as the course's prerequisites (an array of Course objects) and sections available for the course (an array of Section objects). Methods include `getSections`, which returns all of the course's sections; `isAvailable(semester)`, which returns `True` if the course if offered in the specified semester; and `getPrerequisites(course)` and `arePrerequis-itesMet(student)`, which returns the course's prerequisites and returns whether the student has met those prerequisites, respectively.
- **Laboratory:** Represents a laboratory section for a course. Contains a lab instructor as well as an array of RoomBlock objects.

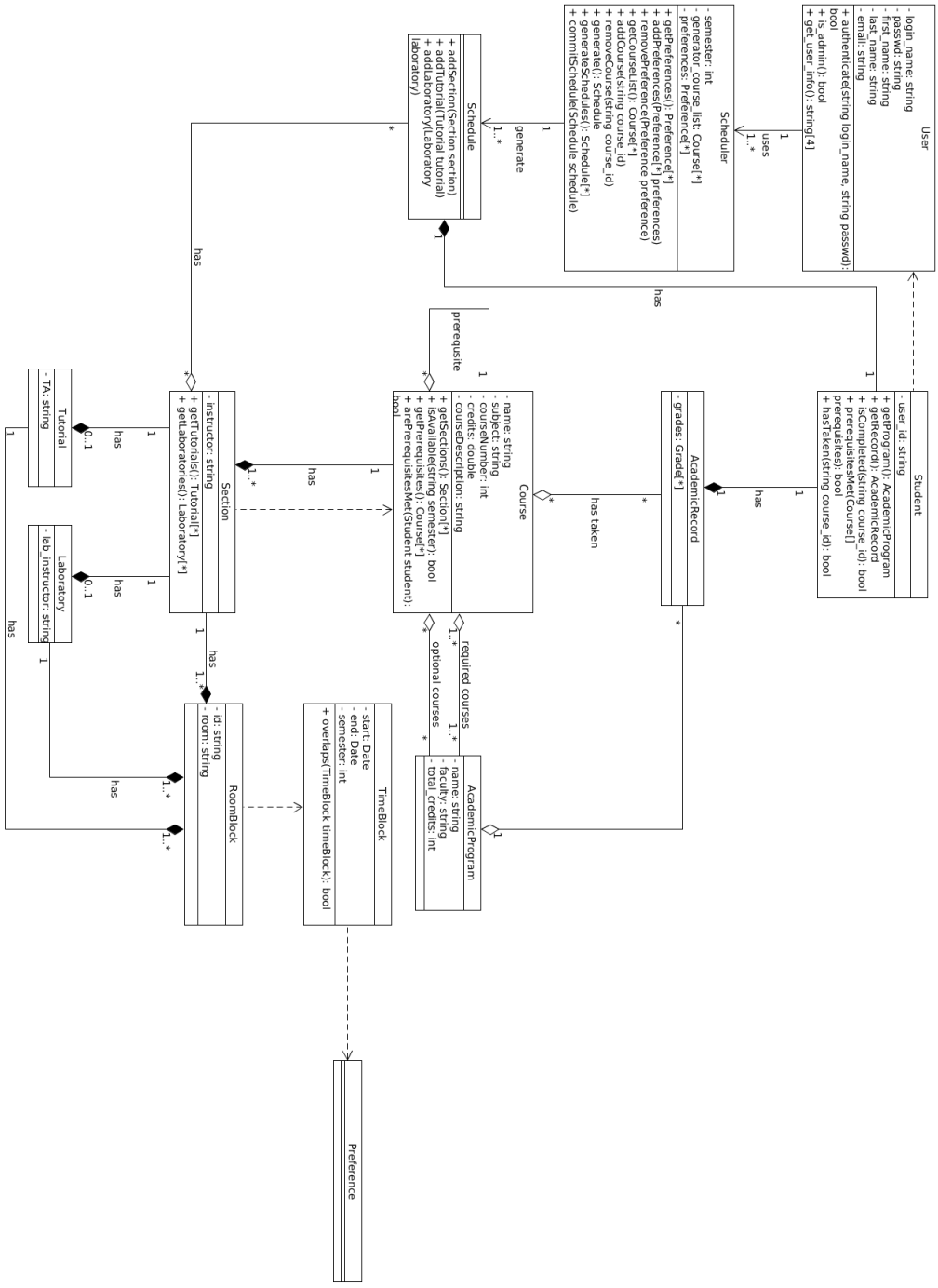



Figure 4: Logical view class diagram

- **Preference:** Represents a user's preference for times when they would *not* want to have a class. It inherits from and is a wrapper class of TimeBlock.
- **RoomBlock:** A class for representing a room being occupied for a period of time.

Inherits from TimeBlock.

- **Schedule:** Represents a student's schedule. In addition to having an array of Section objects, Schedule has three methods: `addSection(Section)`, which adds a Section object; `addTutorial(tutorial)`, which adds a Tutorial object; and `addLaboratory(laboratory)`, which add a Laboratory object.

- **Scheduler:** The class responsible for generating schedules. Contains an array of Preference objects to represent the user's preferences as well as an array of Course objects (`generator_course_list`) to represent the courses the user wishes to register in. Contains methods for retrieving and modifying the user's preferences, adding and removing courses from `generator_course_list`, generating schedules, and committing a schedule.

- **Section:** Class for representing a course section. Inherits from Course and includes an additional Instructor attribute, as well as a possible Tutorial and Laboratory object.

- **Student:** Represents a student. Inherits from User and adds a student ID attribute. Includes methods for accessing the student's academic record and program, as well as methods for checking whether the student has completed or taken a given course.

- **TimeBlock:** A class for representing a block of time in a week. Has a start and end time (represented with Date objects, not shown) as well as a semester. The method `overlaps` accepts another TimeBlock object and returns `true` if there is a time overlap between the two time blocks.

- **Tutorial:** Represents a tutorial section for a course. Contains a TA as well as an array of RoomBlock objects.

- **User:** System user. Attributes include login name, hashed password, first and last names, and email. The `authenticate` method logs the user in if the correct credentials have been passed. The `is_admin` method returns `True` if the user is a system administrator. System Administrator class has not been included in the class diagram for simplicity's sake.

### 9.1.2 Development View

The development view is presented in Figure 5 as a component diagram.

CodeIgniter is a PHP web framework used to create Mytinerary. It provides three major components encompassing the MVC architecture, models, views, and controllers. Along with those three major components CodeIgniter allows for other components that facilitates security through encryption and provides database management.

The Views component represents the core user interface that users will be seeing. In addition to that all the business logic handled by the Models component is connected through the views which allow the views to display all sub-components found in the models such

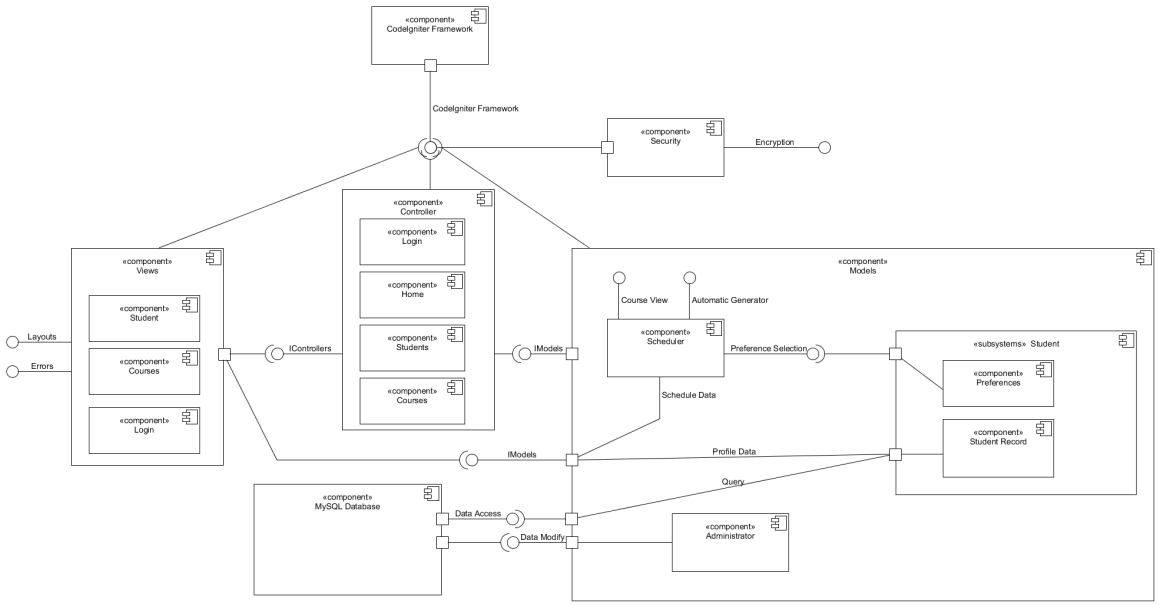



Figure 5: Development view component diagram

as the Scheduler, Student Profiles, and list of courses. The views component shall be routinely redirected to a new page by the controller which is why it also requires the Controller component.

The Controllers component comprises of several controller classes that correspond to their respective Views class. This component handles any user interaction with Mytinerary such as users navigating through tabs or more attempting to add new courses onto the schedule. In essence, the controllers redirect users to appropriate pages or dynamically update current pages according to user selections. Because the controller handles user selections it must require the models component to signal any potential changes within the business logic such as removing a course from the schedule.

The Models component is composed of several other sub-components that together form the core of Mytinerary's domain logic. These sub-components include the Scheduler, the Student, and the Administrator. The scheduler component contains all the current course information and possible schedules. In order to accomplish this it provides an automatic schedule generator taking into account a students' academic record in addition to their preferences. Therefore, in order to make the schedule, the scheduler needs to access the data information that comes from preferences entered by the student as well as his student record. The scheduler is also capable of simply displaying all available courses in a given semester.

The student component contains a student record and preferences. Note that the preferences component and the student record component are found within the student subsystem. This indicates that these components are different based on the user of the application. A separate administrator component is also defined to differentiate the role of a Student and

Administrator. Students may only query the database for data access while administrators can query and modify the database. For instance, administrators can add, remove, or update any class information while students may only view class listings. The administrators are also capable of directly accessing any student profile and manually altering their schedules. All user and course information are stored in the MySQL database component that is hosted both locally and within the cloud. As mentioned earlier, many of the models sub-components are linked to the views component in order to display the information.

An external component that requires the CodeIgniter framework to function is the security. The security provides encryption and secure access and logout to Mytinerary.

### 9.1.3 Physical View

The physical view is presented in Figure 6 as a deployment diagram.

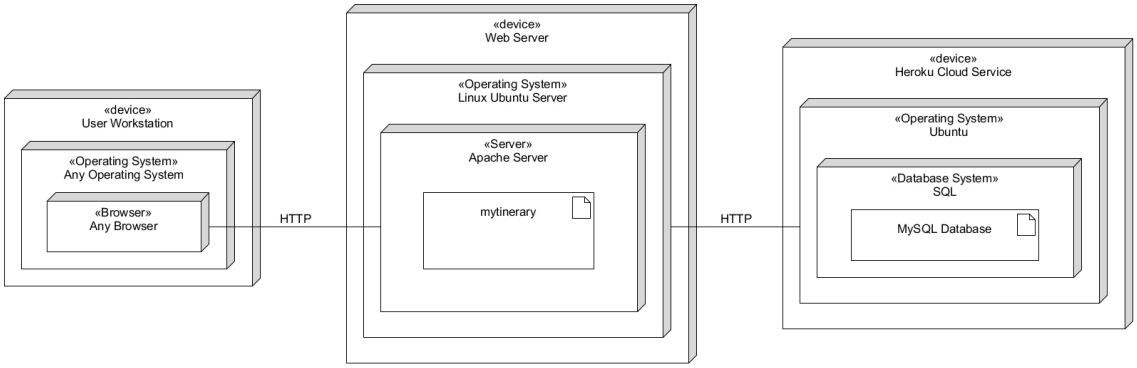



Figure 6: Physical view deployment diagram

The deployment diagram is composed of three main nodes: the user work station, the web server and the Heroku Cloud service. When looking at the left side of the diagram, the user work station is known as the first device. The user can be on any device with any operating system and he could use any browser to access the application through its association with the Apache Server. The node in the middle is the web server which is using the Linux Ubuntu Server as operating system. The Linux Ubuntu Server is an open source software that serves as the server as well as the cloud. The browser used by the student is connected to the Apache Server which is found within the Linux Ubuntu Server node and this server will provide access to the web site known as mytinerary. The node on the right side is the Heroku Cloud Service which stores all the information on the cloud. This device also uses the Ubuntu operating system. Note that the operating system has an association with the operating system from the web server, which is why both devices use the same operating system. Within the operating system of the right node, there is the database using SQL. Finally the SQL node holds the artifact known as MySQL database which stores all the information such as all the courses in the software engineering program as well as the students' record. All this information is stored in the cloud.

### 9.1.4 Process View

The process view is presented in Figure 7 as an activity diagram.

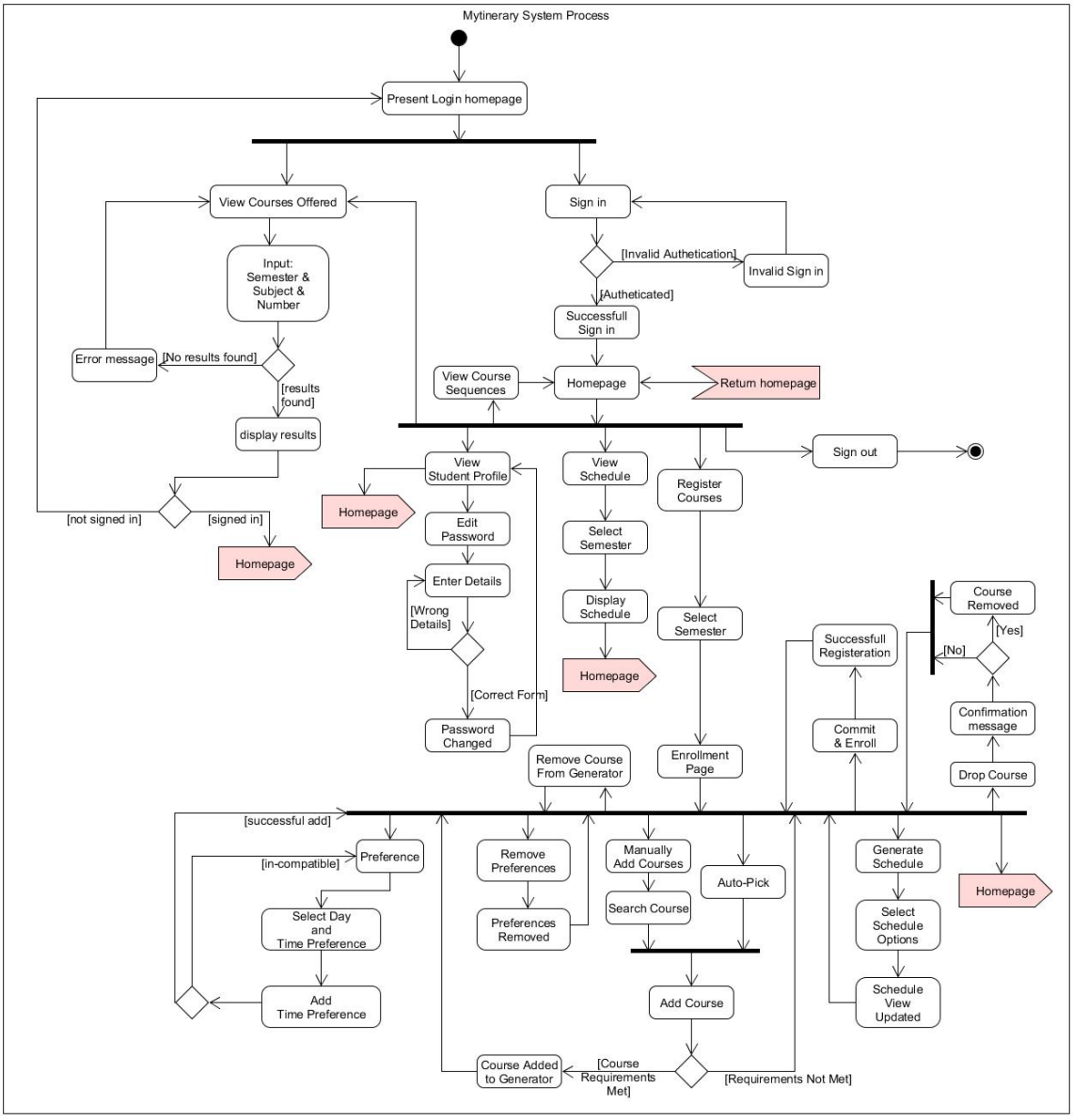



Figure 7: Process view activity diagram

The activity diagram describes precisely the order of the student's activities during registration. The activity diagram is similar to a flow chart of activities, and that demonstrates the capability of the system, as well as the results when errors may occur as caused by the users' decisions.

The Activity Diagram provided shows the system activity path from login until final registration.

After the user has successfully logged in, the user may decide on day and time preferences for their schedule. After selecting the time and day constraints, a list of courses based on the constraints will appear on the screen and the user may decide the courses they wish to be registered in.

Following this, the user may decide an auto selection or manual selection of their courses. If Auto selection is selected, the courses available based on their pre-requisites will be generated into the final schedule. The user may then modify these selection, if they would like. The next option is manual selection, where the user will enter the course number into the search box and choose a section. This selection will then go to check if the user has the prerequisite. If a required prerequisite is not satisfied, a message will appear and the user must choose another course. The user may drop their course, or modify the courses they chose at any given moment during their registration period. If the prerequisite is satisfied, the course will be added to the student's schedule.

Finally, the user just needs to commit their schedule and the final schedule is generated. The user may then proceed to log out or redo their schedule completely as all of this activity occurs on one page.

## 9.2 Scenarios

The use case diagram in Figure 8 illustrates a typical sequence of actions between objects and processes.

## 9.3 Subsystem Interfaces Specifications

### 9.3.1 IScheduler

The scheduler interface manages schedule generation as well as validation of conflicts, prerequisites and student preferences.

**Class Scheduler**

- addCourse(course_id)
  - *Description*: Adds course to scheduler.
  - *Valid Parameters:* Course id must correspond to existing course id in current semester.
  - *Return Type:* Void
- generateSchedules()
  - *Description*: Generates a set of possible schedules.
  - *Valid Parameters:* None

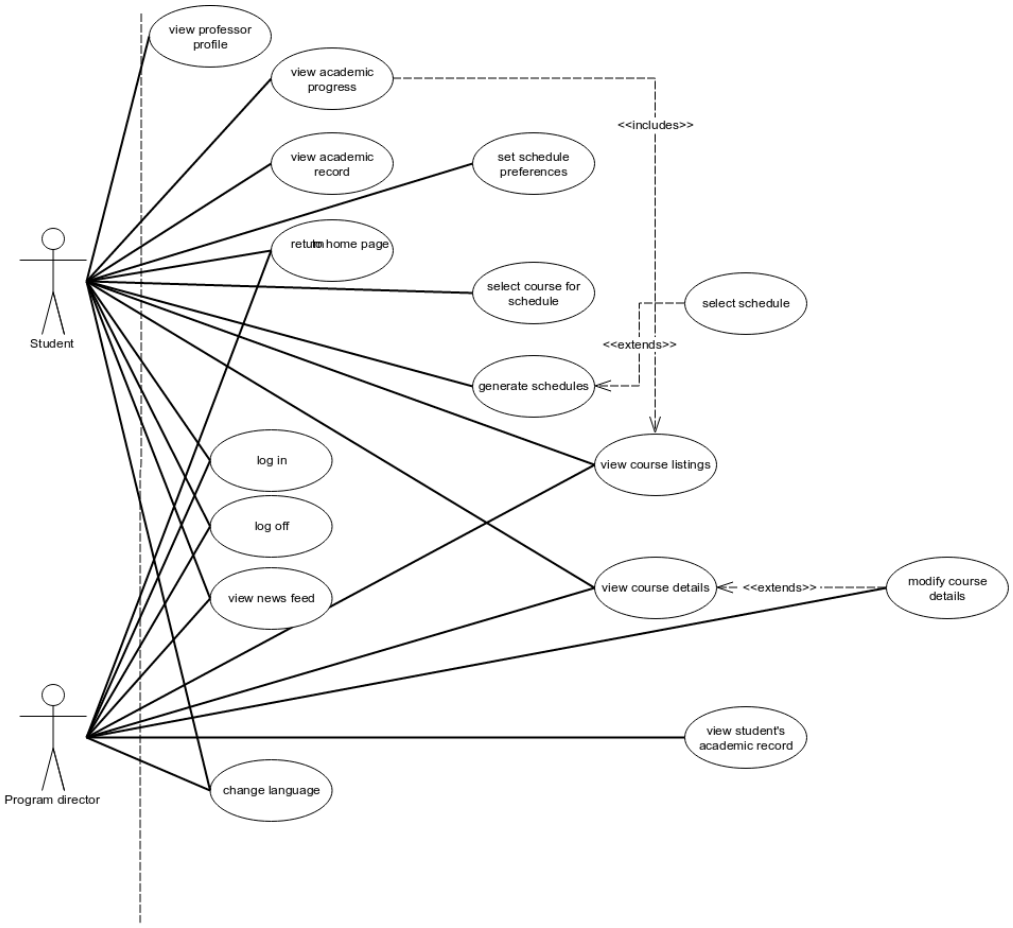



Figure 8: Use Case Diagram

  - *Return Type:* Array of Schedules

- generate(generator_course_list)
  - *Description:* Generates a schedule based off courses added.
  - *Valid Parameters:* Course List must correspond to available courses in current semester.
  - *Return Type:* Schedule

- commitSchedule(schedule)
  - *Description:* Replaces users current schedule with given schedule.
  - *Valid Parameters:* Schedule object must be generated from Scheduler class.

- *Return Type:* Void

- addPrefrences(Preference [*] )
  - *Description:* Adds selected preferences to current user preferences.
  - *Valid Parameters:* An array of preferences containing preference objects that correspond to list of available preferences.
  - *Return Type:* Void

### 9.3.2 IDatabase

The database interface connects the application and the database together. The interface allows for retrieval, insertion, and deletion of information within the database.

**Class DatabaseDriver**

- query(query_string)
  - *Description*: Returns database results depending on the query.
  - *Valid Parameters*: Query string must correspond to SQL query format.
  - *Return Type*: Return type varies according to Query.

### 9.3.3 IAuthentication

The authentication interface validates the login of users as well as providing security and restrictions for every user session or requests.

**Class User**

- authenticate(login_name, password)
  - *Description*: Validates user login.
  - *Valid Parameters*: Login name and password must be 6 to 255 characters long.
  - *Return Type*: Boolean

- is_admin(user_id)
  - *Description:* Determines if user is an administrator to grant more access.
  - *Valid Parameters:* User id must be 6 to 255 characters long.
  - *Return Type:* Boolean

**Class Login**

- index()

- *Description*: Validates if authentication method has succeeded by tracking the user's session. Redirects to homepage or an error message according to result.
- *Valid Parameters*: None
- *Return Type:* Void

**Class CI_Security**

- csrf_verify()
  - *Description:* Validates sessions and POST requests.
  - *Valid Parameters:* None
  - *Return Type:* Session object

### 9.3.4 ICourseManagement

The course management interface is responsible for handling the assignment of course tutorials, laboratories, sections, location, and time of all courses within any given semester. The interface also authenticates course requisites and availabilities.

**Class Schedule**

- addSection(section)
  - *Description*: Attaches a section to the scheduler.
  - *Valid Parameters*: Section object containing an instructor.
  - *Return Type*: Void
- addTutorial(tutorial)
  - *Description*: Attaches a tutorial to the scheduler.
  - *Valid Parameters*: Tutorial object containing a teaching assistant.
  - *Return Type*: Void
- addLaboratory(tutorial)
  - *Description*: Attaches a laboratory to the scheduler.
  - *Valid Parameters*: Laboratory object containing a lab instructor.
  - *Return Type*: Void

**Class Course**

- isAvailable(semester)

  - *Description*: Determines if a course is offered in given semester.
  - *Valid Parameters*: A valid semester within the scheduler.
  - *Return Type*: Boolean

- getPrerequisites()

  - *Description*: Obtains all perquisites of a given course.
  - *Valid Parameters*: None
  - *Return Type*: An array of course objects

- arePrerequisitesMet()

  - *Description*: Verifies if course perquisites are satisfied.
  - *Valid Parameters*: None
  - *Return Type*: Boolean

**Class TimeBlock**

- overlaps(Timeblock)

  - *Description*: Determines if given timeblock conflicts with the parameter timeblock.
  - *Valid Parameters*: Timeblock object with start time before end time.
  - *Return Type*: Boolean

# 10 Detailed Design

## 10.1 Detailed Design Diagram

### 10.1.1 Authentication Subsystem

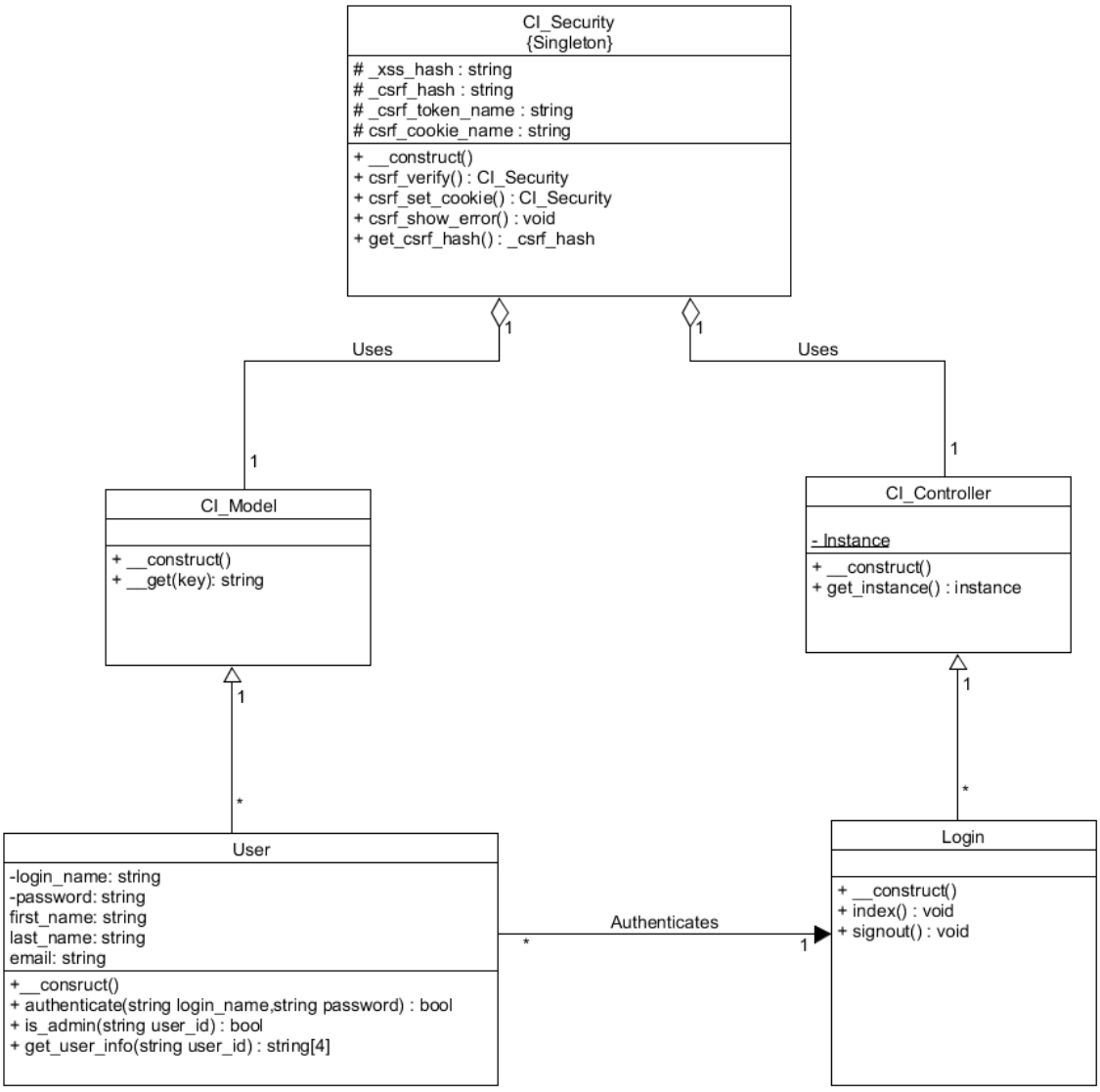



Figure 9: Authentication Subsystem Diagram

The authentication subsystem is responsible for validating user logins as well as providing security throughout the entire session and hashing sensitive information such as pass-

words. The user class is the starting point of the authentication subsystem. As it is found within the models it must inherit from the template class CI_Model. The user begins by entering their username and password in order to login to Mytinerary. This in turn calls the authenticate method which will take in the entered login name and password and verify them by querying the database. If the query results in one row and the internal password verify method returns true than the login is successful. During this time the User class will interact with the Login Controller class. If the authenticate method is successful the Login class will redirect the user to the index page using the index method as well as providing the user with a new session. The CI_Security is a singleton class that is the cornerstone of Mytinerary's secure session handling. It creates a unique hashed Cross Site Request Forgery protection cookie for every user session. Mytinerary also uses the BlowFish algorithm within the database to hash all sensitive data such as passwords or course details.

### 10.1.2 Course Management Subsystem

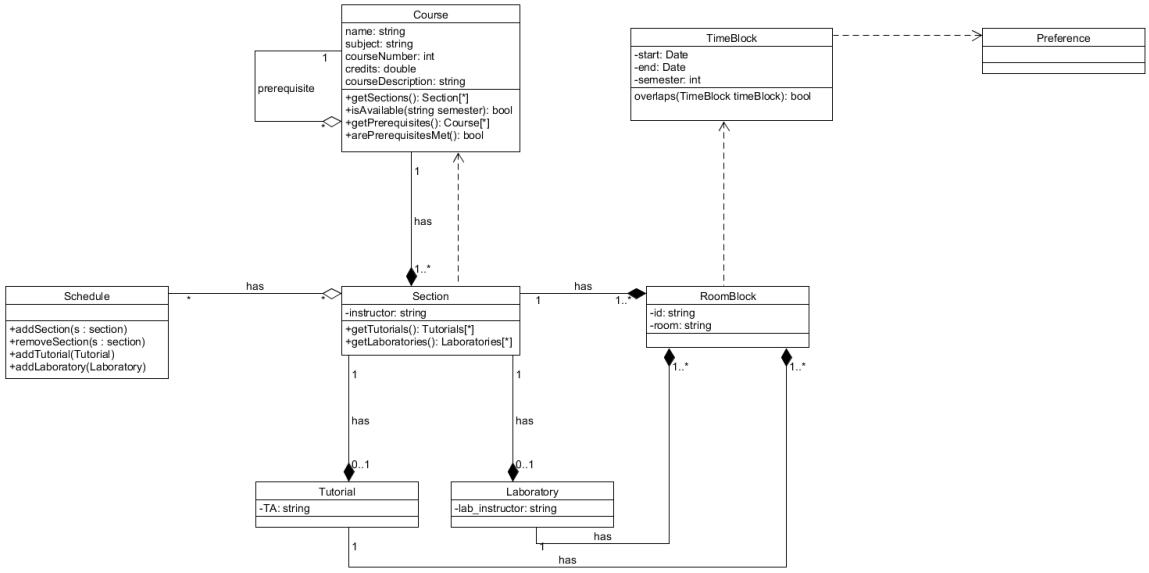



Figure 10: Course Management Subsystem Diagram

The course management subsystem is responsible for any and all operations that are related to the access of course information and the modification of student schedules. Each section objects inherits from its corresponding course object to knowall its information. Each course has at least one section. Also each course may have prerequisites which have to be met in order to be able to add it to a tentative schedule. Each section for a specific course may have a tutorial and a laboratory component. Note that each section, tutorial and laboratory all have one or more room blocks. This is necessary because each of those classes may have different time and room allocations during the week. The RoomBlock class inherits the

TimeBlock class to know at which time and during which semester the room is going to be occupied. The TimeBlock class inherits the preferences to know which times are preferred by the user. The current schedule may have different sections for which the user is already registered to.

### 10.1.3 Database Subsystem

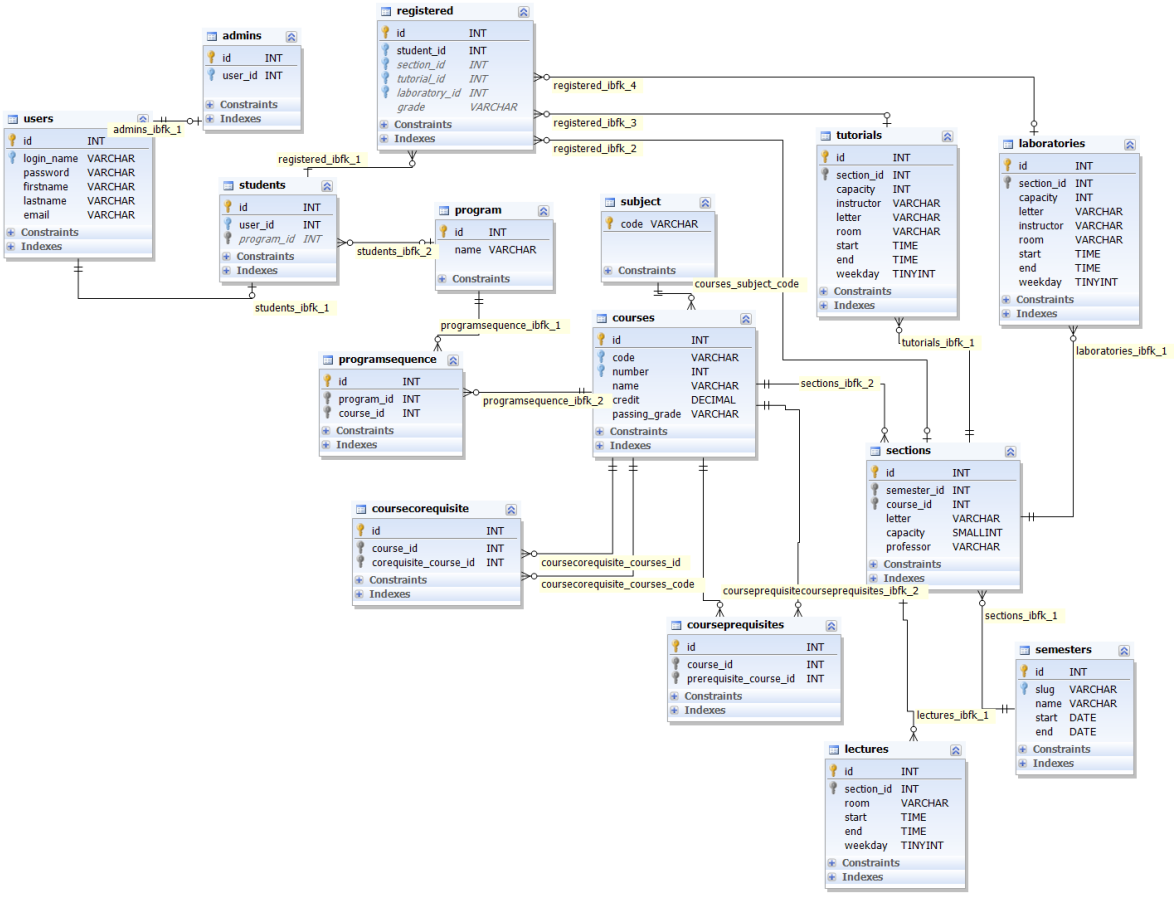



Figure 11: Database Subsystem Diagram

Within the database, it is important to note that the active user using the application will have a corresponding ID number. This is will be used during the user's process throughout their activity time. Thus any action will correspond to that user ID. When accessing the database, the user is detected to be either admin or students. The database however, is commonly used when the user is a student, thus allowing them to view/change their schedule from the database.

Going further into the student objects, it branches out into two separate blocks: registered and program. Registered block contains the list of the courses the student is currently

enrolled in. Furthermore, the sections block is used to represent the enrolled course's sections which include: lecture, tutorial and labs. The reason sections, tutorials and laboratories blocks are linked to the registered block is to achieve a quicker access to the saved course, as courses within the registered block will have an existing section ID which includes tutorial and labs ID if possible. This method is preferred over accessing the section block first followed by accessing the tutorial/labs sections when accessing the database.

The other path: program will allow the student to access the correct program sequence according to the corresponding program. This option is selected when the student wants to add a new course to their schedule. Following the program block is the programsequence block which contains the list of course within the program's sequence. Once in the course block, it branches out into a few other blocks: subject, coursecorequisite and courseprequisite blocks. The subject block returns the code of the course selected from the, ie: SOEN341. The courseprequisite and coursecorequisite blocks contain any possible pre/corequisite of that selected course.

Furthermore, the courses block is linked to the sections block, in order to finally insert the course into the schedule with the same ID as the original user.

### 10.1.4 Scheduler Subsystem

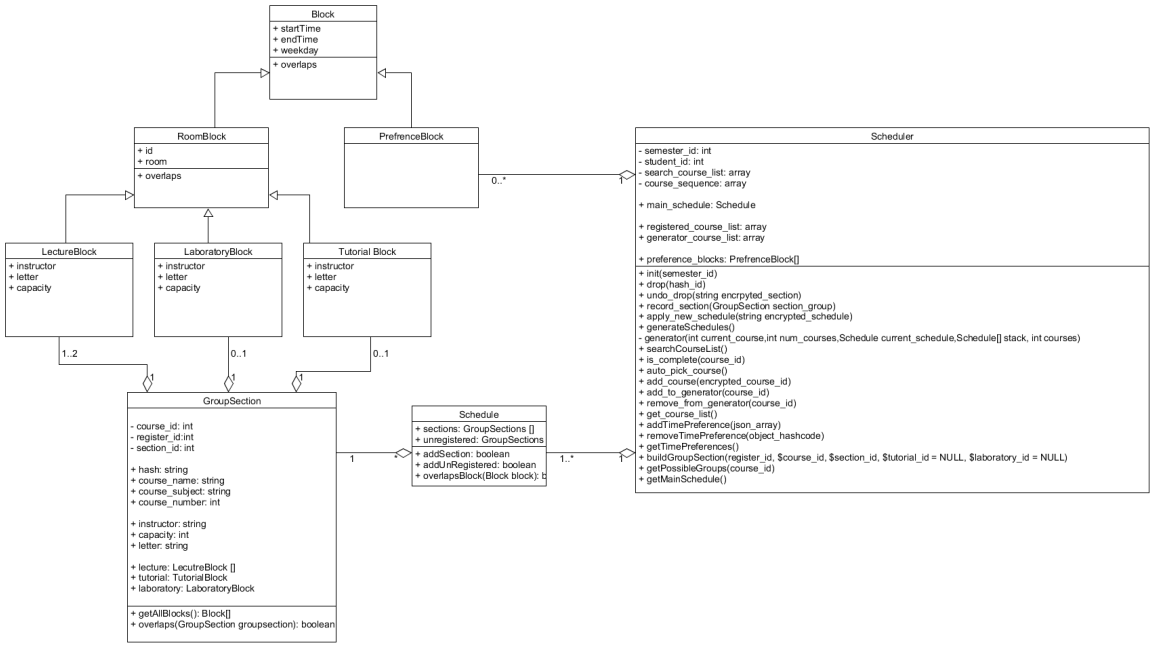



Figure 12: Scheduler Subsystem Subsystem Diagram

The scheduler subsystem is responsible for generating all possible schedules, given a list of courses the user wants to register for and a list of preferences that limit when a user can

take a course.

The student can view all preferences or change their personal preferences. The student can also view every course available, as well as sign up for courses and drop courses. Finally, the student can generate all possible schedules and the choose one as his finalized schedules.

## 10.2 Unit Descriptions

### 10.2.1 Authentication Subsystem

The following two classes are provided by the CodeIgniter framework that allow for a separation of concerns via the MVC pattern which promotes high cohesion and low coupling by allowing each module to work independently of one another. Please note these classes do not need to be defined but simply inherited from.
***CI_Model:*** Is a template class provided by CodeIgniter in which any class containing the domain logic will inherit from.

**Attributes:**

- None

**Methods:**

- `__construct()`: Class constructor initializing a default log message.
- `__get(key)`: Debugging method that returns an error message.

***CI_Controller:*** Is a template class provided by CodeIgniter in which any class that manipulates the user interface classes will inherit from.

**Attributes:**

- None

**Methods:**

- `__constructor()`: Class constructor loading respective view classes.
- `get_instance()`: Returns controller instance.

***Login*:** Is a controller class that redirects the user according the result of the User's authentication method.

**Attributes:**

- None

**Methods:**

- `__construct()`: Class constructor
- `index()`: Validates if authentication method has succeeded, sets session cookies, and redirects user to home page. Returns void.
- `signout()`: Destroys current user session and redirects to default login page. Returns void.

***User*:** Is a model class that directly interacts with the database containing all user information.

**Attributes:**

- `login_name`: string containing the users login name.
- `password`: string containing the users password.
- `first_name`: string containing the users first name.
- `last_name`: string containing the users last name.
- `email`: string containing the users email adress.

**Methods:**

- `__construct()`: Class constructor calling the parent constructor.
- `authenticate(login_name,password)`: This method takes in a two strings being the login name and password and queries the database to find a match. Note the strings are immediately hashed using the blowfish algorithm to match with the database elements. Returns a boolean.
- `is_admin(user_id)`: Checks if user is an administrator returning a Boolean used to grant different access rights compared to a regular student.
- `get_user_info(user_id)`: Returns user information from the database such as login name and email. Returns user information in the form of an array of strings.

***CI_Security:*** Is a singleton class provided by CodeIgniter in which every component of the MVC classes will use. The CI_Security class contains many attributes and methods however only a select few are relevant to the authentication subsystem. The CI_Security provides protection for sensitive information as well as secure session handling.

**Attributes:**

- `_xss_hash`: Randomly generated hash for protecting MyTinerary's URLs.
- `_csrf_hash`: Randomly generated hash for Cross Site Request Forgery protection cookie.
- `_csrf_token_name`: Token name for Cross Site Request Forgery protection cookie.
- `_csrf_cookie_name`: Cookie name for Cross Site Request Forgery protection cookie.

The xss and csrf hash allows for the protection of user sessions since the hash is randomly generated and provides secure access using either the token or cookie name. Please note that the usage of this class is all done internally within the CodeIgniter framework.

**Methods:**

- `__construct()`: Class constructor that will create the cookies and set the hash
- `csrf_verify()`: Provides security validation on POST requests, validates if URI has been whitelisted, as well as garbage collection on finished session arrays. Returns the session if verified otherwise returns null.
- `csrf_set_cookie()`: Modifies the internal cookie and returns it. Returns void.
- `csrf_show_error()`: Displays an error message when user of application attempts an action that is either not permitted or not secure. Returns void.
- `get_csrf_hash()`: Return the hash as a string.

### 10.2.2 Course Management Subsystem

***Schedule:*** is a views class that provides data about the courses the user wants to add.

**Attributes:**

- None

**Methods:**

- `addSection(section)` will temporarily add the section s, which is the parameter, to the schedule.
- `removeSection(section)` will remove the section s, which is the parameter, from the schedule.
- `addTutorial(tutorial)` will add the tutorial that is associated to the section (if any)
- `addLaboratory(laboratory)` will add the laboratory that is associated to the section (if any)

***Section:*** is a views class for a specific course. Depending on the course, it may have one or more sections.

**Attributes:**

- `instructor`: name of the instructor

**Methods:**

- `getTutorials()`: Tutorials[*] will return an array of tutorials for a specific section of a course.
- `getLaboratories()`: Laboratories[*] will return an array of laboratories for a specific section of a course.

***Tutorial:*** is a views class which represents the tutorial for a specific section of a course (if applicable.

**Attributes:**

- `TA`: name of the TA for the tutorial.

**Methods:**

- None

***Laboratory:*** is a views class which represents the laboratory for a specific section of a course (if applicable).

**Attributes:**

- `Lab_instructor`: string of the lab instructor's name.

**Methods:**

- None

***RoomBlock:*** is a views class which represents the room(s) in which a class is taking place.

**Attributes:**

- `id`: string representing the id of the room.
- `room`: string representing the name.

**Methods:**

- None

textbf*TimeBlock:* is a views class that represents the time for which the room is needed.

**Attributes:**

- `start`: starting date of the course
- `end`: ending date of the course
- `semester`: number representing the semester.

**Methods:**

- `overlaps(TimeBlock timeBlock)`: bool will look at timeBlock (which is the parameter) and checks if there is any conflict in terms of time. If there is a conflict, then the method will return true, otherwise false.

***Preference:*** is a views class which represents the user's desired options when generating a schedule.

**Attributes:**

- None

**Methods:**

- None

***Course:*** is a views class which represents a subject that needs to be completed in order to graduate. It has at least one section.

**Attributes:**

- `name`: string representing the name of the course
- `subject`: string representing the subject of the course
- `courseNumber`: a number representing the course.
- `credits`: a number representing the amount of credits the course is worth.
- `courseDescription`: string containing a description of the course

**Methods:**

- `getSections()` will return an array of different sections for a specific course.
- `isAvailable(string semester)` will take a string semester as parameter and check if the course is available for that semester. If it is available, the method will return true, otherwise it will return false.
- `getPrerequisites()` will return an array of different courses that are prerequisites for a specific course.
- `arePrerequisitesMet()`: bool will check if the prerequisites are met for a specific course. If the prerequisites are met, then the method will return true, else it will return false.

### 10.2.3 Database Subsystem

- `Users` : Contains a list of every users of this application.
- `Admins`: Contains list of every "admin"-type in the users table
- `Students`: Contains list of every "student"-type in the users table
- `Registered`: contains list of registered courses for a selected student
- `Program`: contains list of every program in Concordia
- `Programsequence`: list of courses for a selected program.
- `Courses`: Contains details of a selected general course. Details contain: code, co-requisite, prerequisite and section. This does not contain the availability of the class for a semester.

- `Subject`: Similar to program table, however contains the code rather than the name.
- `Courseprequisite`: contains list of courses that are pre-requisite of a selected course.
- `Coursecorequisite`: contains list of courses that are co-requisite of a selected course.
- `Sections`: contains a list of every lecture sections of selected course.
- `Semesters`: contains details of start and end date of a selected course in the selected semester
- `Lectures`: contains details about a selected lecture section (room #, start time, end time)
- `Tutorials`: contains a list or every possible tutorial sections of the selected lecture section (if exists) and their details.
- `Laboratories`: Contains a list of every possible laboratory section of the selected lecture section (if exists) and their details

### 10.2.4 Scheduler Subsystem

***Scheduler*:** Is a core class which handles the creation of potential schedules that the student can choose from. It also handles the committing of a finalized schedule.

**Attributes:**

- `semester:` is a number representing the semester the student is trying to add courses to.
- `generator_course_list:` is a list of all the sections for every course that the student wants to enroll in.
- `preferences:` is a list of preferences that limit when the student wants to take a course.

**Methods:**

- `getPreferences():` This method returns all of the available preferences that the student can select.
- `addPreferences(preferences):` This method adds a preference that the student has selected.

- `removePreference(preference)`: This method removes a preference that the student has selected.
- `getCourseList()`: This method returns every single course that is available.
- `addCourse(course_id)`: This method attempts to add a course, first checking if the prerequisites are met.
- `removeCourse(course_id)`: This method removes a course that the student has registered for.
- `generateSchedules()`: This method returns all possible schedules based on the users preferences.
- `generate(generator_course_list)`: This is a method used by *generateSchedules()*. It creates creates a single schedule that has not been created yet.
- `commitSchedule(schedule)`: This method applies the students selected schedule as his final schedule.

***Schedule*:** Is a class to represent a student's schedule. It contains a sequence of sections that the student has registered for.

**Attributes:**

- `student`: is a string representing the student that this particular schedule belongs to.
- `semester`: is a number representing the semester the student is trying to add courses to.
- `schedule_id`: is a number representing the schedule id.

**Methods:**

- `addSection(section)`: This method adds a section to the schedule class.
- `addTutorial(tutorial)`: This method adds a tutorial to the schedule class.
- `addLaboratory(labratory)`: This method adds a labratory to the schedule class.

# 11 Dynamic Design Scenarios

## 11.1 System Sequence Diagrams

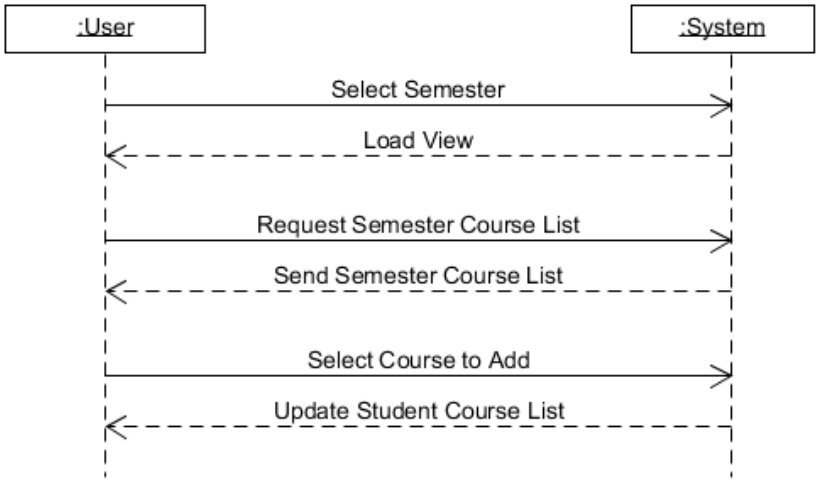



Figure 13: System Sequence Diagram for the *Add Course* Use Case.

The System Sequence Diagram in Figure 13 is for the *Add Course* Use Case. The User starts off by selecting a semester to add courses to. The system takes note of the chosen semester, and updates the user on options available for that semester. The user then requests a list of all courses available in that semester. The system responds by sending the user the desired course list. From this list, the user can select a course to add. The system then updates the list of courses the user wishes to register for.

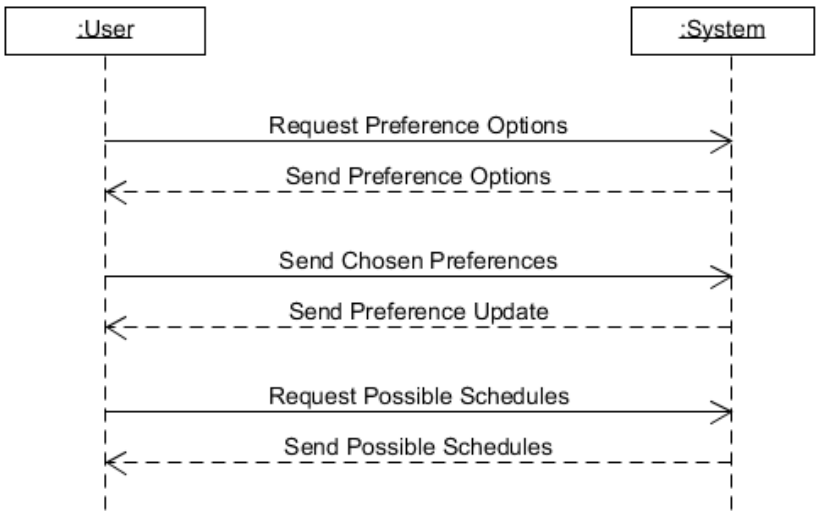



Figure 14: System Sequence Diagram for the *Generate Schedules* Use Case.

The System Sequence Diagram in Figure 14 is for the *Generate Schedule* Use Case. The user begins by setting their desired preferences for their schedule. This is done by first requesting all possible preference options. The system responds by providing all possible preference options. From the options, the user can select the preferences they want to apply to their schedule. The system then updates the users preferences, and lets the user know their currently chosen preference options. Finally, the student requests all possible schedules to be generated for the courses they wish to register for, consistent with the user-selected preferences. The system creates all possible schedules, and returns them to the user.

## 11.2 Sequence Diagrams

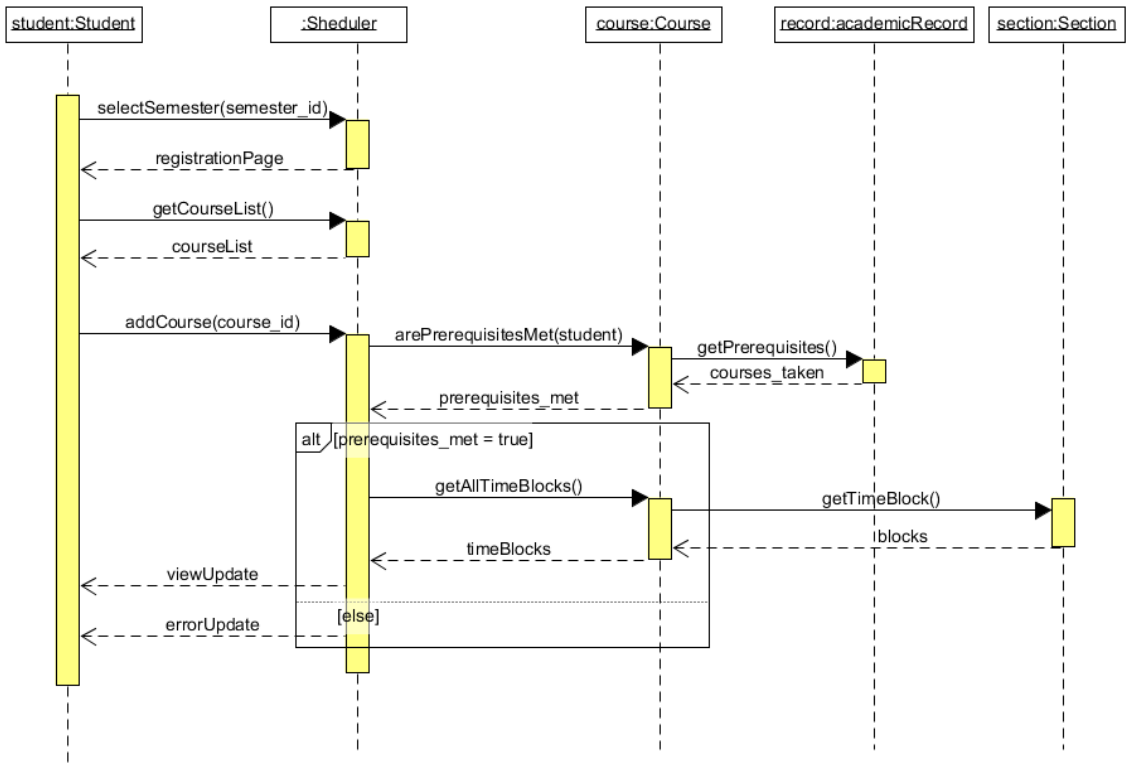



Figure 15: Sequence Diagram for the *Add Course* Use Case.

The Sequence Diagram in Figure 15 is for the *Add Course* Use Case. First, the student selects a semester he wants to add courses to, passing the selected semester to the scheduler. The scheduler takes note of the chosen semester, and updates the student on options available for that semester. The user then requests a list of all courses available in that semester. This information is all held within the scheduler class. The scheduler responds by sending the user the desired list of courses. From this list, the user can select a course to add. Selecting a course calls `addCourse(course_id)`. The scheduler then makes sure the student has completed all prerequisites for the course before adding it to the students course list. This is done by calling `arePrerequisitesMet(student)` on the course. If all prerequisites are met by the student, this function returns true, and all sections (timeBlocks in the diagram) are returned to `generator_course_list` in scheduler. `generator_course_list` is the list of all sections the schedule generator will use to create potential schedules for the student. Once the sections have been added, the scheduler lets the student know the course was successfully added. If the prerequisites are not met, the scheduler lets the student know there was an error.

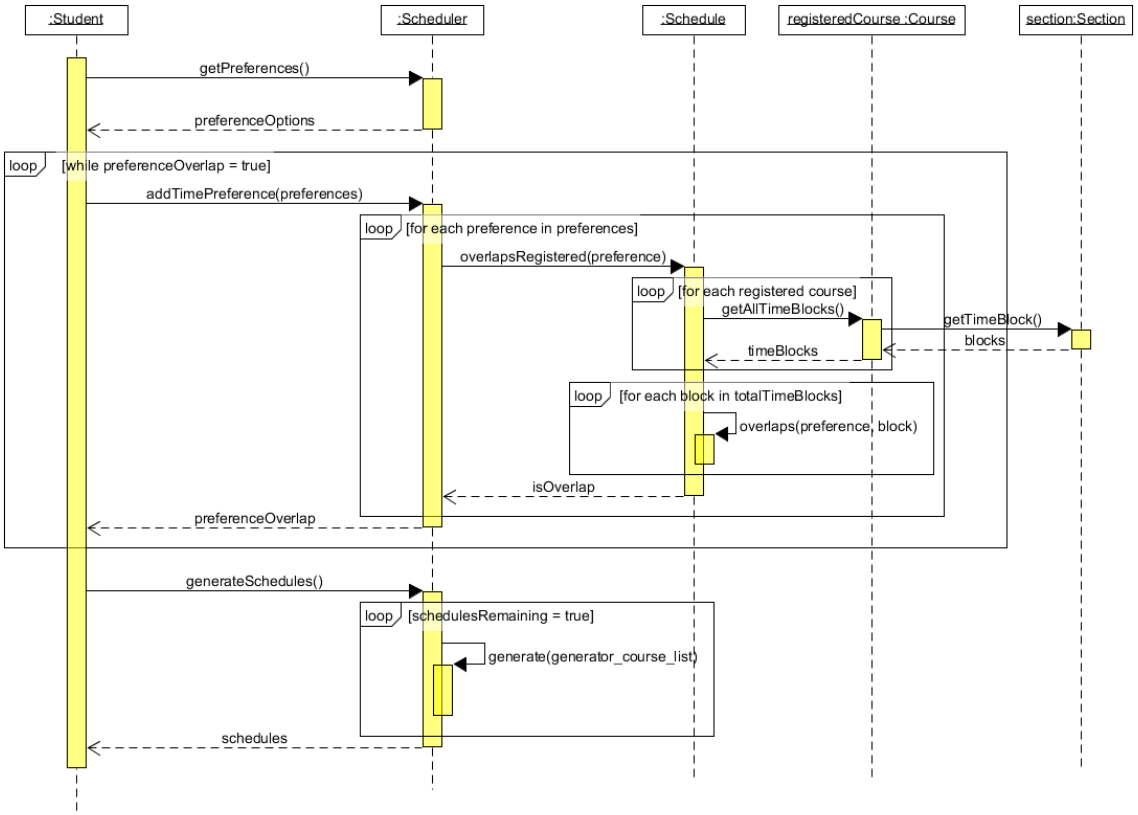



Figure 16: Sequence Diagram for the *Generate Schedules* Use Case.

The Sequence Diagram in Figure 16 is for the *Generate Schedule* Use Case. First, the student must set his preferences. These are limitations on the times when the student wants to have class. The student must first use `getPreferences()` to request all preference options from the scheduler, at which point the scheduler returns them.

The student then selects which preferences he wants, and adds them to the scheduler using `addTimePreference(preferences)` on Scheduler. This function will continue to be called until the student selects preferences that do not conflict with his registered classes, at which point `preferenceOverlap` will be set to false, and the loop will break.

For each of the selected preferences, the Scheduler checks if a particular preference conflicts with a section the student has already registered for. Scheduler does this by using `overlapsRegistered(preference)` on the Schedule class. This method starts by getting all of the time blocks for each section the student is registered for (by calling `getAllTimeBlocks` for each section the student is registered for). Every time block the student is registered for is put into the variable `totalTimeBlocks`. Finally, for each block in totalTimeBlocks the Schedule class uses `overlaps` to check if a preferences overlaps

with a registered time block. If even one of the sections overlap, `isOverlap` is true and so `preferenceOverlap` is true, meaning that the Scheduler will continue to ask the student for preferences. If every single `isOverlap` is false, then `preferenceOverlap` is false, the loop breaks and the preferences are added for the student.

Now that the preferences has been added, the student can use `generateSchedules` to generate all possible schedules. This is done by using `generator(generator_course_list)` where `generator_course_list` is a list of every section that corresponds to a course a student wants to add (Shown in Add Course System Diagram). `generate` is a black box method which creates one schedule that has not been created yet. The Scheduler loops, using `generate` until all possible Schedules have been created, at which point it returns them to the student.

## 11.3 Operational Contracts

Below are the operational contracts for the use case ”Add course to schedule”.

| Contract | CO1 - Select Semester |
|---|---|
| **Operation** | Select Semester |
| **Cross References** | UC: Add course to schedule |
| **Preconditions** | Student is logged in. There is a scheduler object. |
| **Postconditions** | • Scheduler’s semester attribute was modified. |

| Contract | CO2 - Request Semester Course List |
|---|---|
| **Operation** | Request Semester Course List |
| **Cross References** | UC: Add course to schedule |
| **Preconditions** | Semester has been selected. |
| **Postconditions** | • Course objects were created. • Association between those Course objects and Student was created. |

| Contract | CO3 - Select Course to Add |
|---|---|
| **Operation** | Select Course to Add |
| **Cross References** | UC: Add course to schedule |
| **Preconditions** | Course has not been taken by student and course’s prerequisites have been met. |
| **Postconditions** | • Scheduler’s generator_course_list attribute was modified to include requested course. |

Below are the operational contracts for the use case ”Generate schedules”.

| Contract | CO4 - Request Preference Options |
|---|---|
| **Operation** | Request Preference Options |
| **Cross References** | UC: Generate schedules |
| **Preconditions** | User has selected courses to add to their schedule. |
| **Postconditions** | • A Preferences instance was created. |

| Contract | CO5 - Send Chosen Preferences |
|---|---|
| **Operation** | Send chosen preferences |
| **Cross References** | UC: Generate schedules |
| **Preconditions** | Preference options have been sent. |
| **Postconditions** | • Preference objects were created. • Scheduler's preference attribute was modified to include these preferences. |

| Contract | CO6 - Request Possible Schedules |
|---|---|
| **Operation** | Request possible schedules. |
| **Cross References** | UC: Generate schedules |
| **Preconditions** | Preferences have been sent. |
| **Postconditions** | • Schedule objects were created. • An association between these Schedule objects and the user was created. |

# 12 Estimation

Estimates of the effort and cost required for each of the four modules were calculated using the COCOMO engine. The standard lines of code (SLOC) were difficult to obtain for the authentication subsystem and so the function point was calculated using the inputs shown in Figure 17. The function point could then be used to estimate the SLOC. The inputs for each module are shown in Figure 18, and output, which is the estimation according to COCOMO, is shown in Figure 19. All of the parameters used in the calculation were suggested values. Based on the output of the COCOMO engine, the most likely cost of integration is 455,957 CAD.

| Function Type | # of Function Points Low | Average | High | SubTotal |
|---|---|---|---|---|
| Inputs | 0 | 2 | 0 | 8 |
| Outputs | 0 | 1 | 0 | 5 |
| Files | 0 | 3 | 0 | 30 |
| Interfaces | 0 | 2 | 0 | 14 |
| Queries | 0 | 2 | 0 | 8 |
| Total Unadjusted Function Points | | | | 65 |
| Equivalent Total in SLOC | | | | 5915 |

Figure 17: Calculation of function point for the authentication module.

| X | Module Name | Module Size | LABOR Rate ($/month) | EAF | NOM Effort DEV | EST Effort DEV | PROD | COST | INST COST | Staff | RISK |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Authentication | F:2080 | 7000.00 | 1.00 | 8.1 | 8.1 | 256.8 | 56688.02 | 27.3 | 0.6 | 0.0 |
| | Course Management | S:1400 | 7000.00 | 1.00 | 5.5 | 5.5 | 256.8 | 38155.39 | 27.3 | 0.4 | 0.0 |
| | Database | S:12000 | 7000.00 | 1.00 | 46.7 | 46.7 | 256.8 | 327046.24 | 27.3 | 3.4 | 0.0 |
| | Scheduler | S:1250 | 7000.00 | 1.00 | 4.9 | 4.9 | 256.8 | 34067.32 | 27.3 | 0.4 | 0.0 |

Figure 18: Input to the COCOMO engine, along with suggested values.

Total Lines of Code: 16730

| Estimated | Effort | Sched | PROD | COST | INST | Staff | RISK |
|---|---|---|---|---|---|---|---|
| Optimistic | 43.6 | 12.2 | 383.3 | 305491.17 | 18.3 | 3.6 | |
| Most Likely | 65.1 | 13.8 | 256.8 | 455956.97 | 27.3 | 4.7 | 0.0 |
| Pessimistic | 97.7 | 15.8 | 171.2 | 683935.45 | 40.9 | 6.2 | |

Figure 19: Estimates according to the COCOMO engine.

The cost of documentation can be estimated using the formula $(Number\ of\ pages * 2) * 73.29\$(50€)$ [1], which is approximately 879,480 €or 1,289,643 CAD assuming that the final documentation will be 120 pages.

The cost of testing is typically 15-25% of integration cost [2]. Since integration cost is 455,957 CAD, testing should cost between 68,394 - 113,989 CAD, with a mean value of 91,191 CAD.

The sum of these values is the total cost which is estimated to be 1,836,791 CAD.

[1] Taken from: http://www.indoition.com/en/services/costs-prices-software-documentation.html
[2] Taken from: http://www.softwaretestinggenius.com/how-do-we-measure-the-costs-of-software-testing

# 13 Rapid Prototyping and Risk

Our current prototype satisfies most of the functional requirements listed in Deliverable 1, such as student login, schedule generation, preference setting, etc. The only functionality that is missing is the setting of time preferences by the student. An image of the prototype is shown in Figure 20.

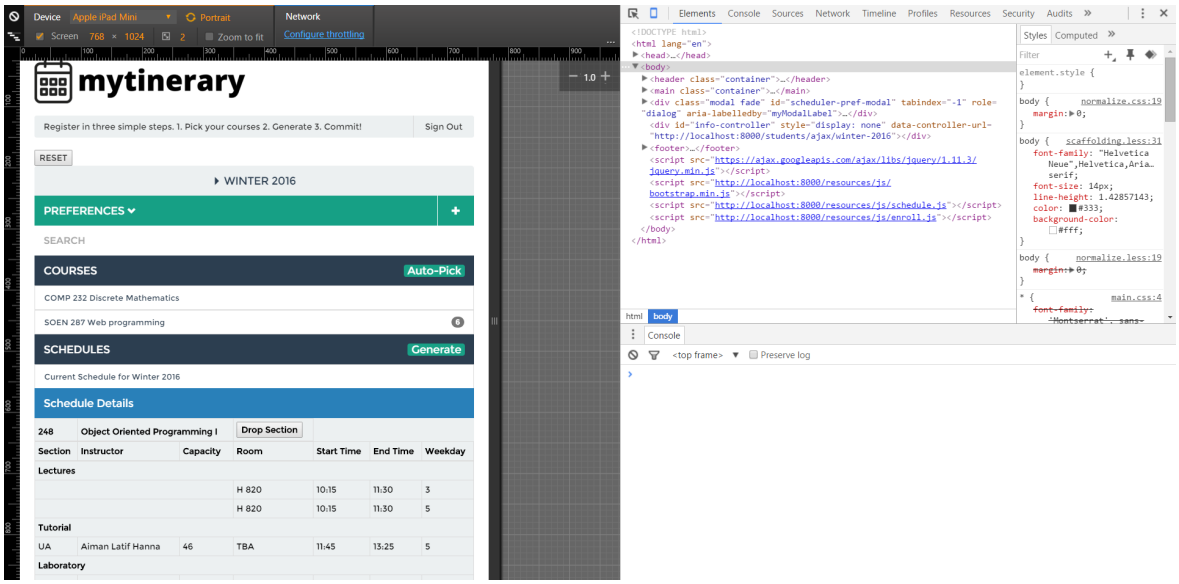



Figure 20: Mytinerary prototype. Image shows the adding of a course.

Although various part of the prototype have been manually tested, no drivers or stubs have been created yet. Unit testing of Mytinerary's modules will begin shortly using CodeIgnitor's built-in PHP unit testing suite, and the results of those tests will be included in Deliverable 3.

## 13.1 Effects of Prototypes on Risk

With our most recent prototype, we can see that none of the anticipated risks have manifested. For the people risk type, the most serious risk we faced was that our lead developer became ill or otherwise became incapable of working on the project. This did not occur, and presently almost all of the required code has been completed. For the estimation risk type, we are actually ahead of schedule, and we do not foresee having to scale down our functional requirements to meet impending deadlines. None of the technical risks that were raised actually occurred. Specifically, all frameworks and servers that were uses functioned as expected, and programmers quickly developed familiarity with their use. Team members never developed any hostility towards one another, and although there were the occasional

disagreement eventually all disputes were settled and a consensus reached on how to proceed. Overall, we felt no need to re-evaluate the risks presented in Deliverable 1.

## 13.2 Effects of Prototypes on Estimation and Scoping

Regarding the scoping, for the most recent prototype, we have implemented all the essential components and modules of our website, except to set the time preferences of the user. In fact, we believe that we have the time to implement functionalities that we previously scoped out of the functional requirements. Specifically, we will implement a feature that will permit a user to change the language between English and French. Additionally, if time permits we will integrate Mytinerary with RateMyProfessor to easily allow students to evaluate prospective professors before registering for a class.

PART

# 3

# TESTING AND DELIVERY

# 14 Introduction

The principal purpose of this document is to summarize the testing regiment that Mytinerary was subjected to. The testing can be divided into four categories: unit testing of individual classes; requirements testing, which verifies that Mytinerary satisfies all functional requirements as specified in deliverable 1; stress testing, which ensures that Mytinerary can cope with both expected and unexpected demands; and finally security testing to assuage clients that their personal information will remain confidential on Mytinerary.

In addition, this document contains thorough installation instructions for any system administrator or webmaster who wished to install Mytinerary on their server, as well as a user manual to assist individuals who will use our web application. Finally, a summary of the man hour costs of this project is included.

# 15 Testing Report

## 15.1 Test Coverage

### 15.1.1 Tested Items

**High Importance**
The following requirements are completely mandatory and the system could not perform the required functionality without them.

1. **Login:** The login item is significantly important to test as it is a core function of the system and must always function properly. Failure of the login procedure would mean the user could not get access to their profile thus being unable to use the system.

2. **Logout:** The logout item is of great importance as well and must also function properly at all times. Failure of the logout item would be very bad for account security as a new user could go onto the system using the previous user's information, accessing their courses and possibly changing their login information.

3. **Set Schedule Preferences:** The set schedule preferences item is extremely important as it directly aids the user with creating their schedule. If this function were to fail, the user would be unable to filter the auto generated schedules, making the user search through all the possible schedules rather than provide a schedule with the desired times.

4. **Select Course for Schedule:** Selecting courses for the schedule is required to function properly as this functionality is detrimental to the proper use of the system. Course selection failure would make generating a schedule impossible, thus failing the entire main functionality of the system. Selecting courses is therefore one of the top priorities of the system along with generating and selecting schedules.

5. **Generate Schedules:** Generating schedules is a top priority of the system and must always function properly. The main purpose of the entire system is to generate schedules for students, therefore the schedule generation itself is of utmost importance. Failure to generate schedules removes the main functionality of the system, making it impossible for users to create their schedules for the semester.

6. **Select Schedule:** Selecting a schedule must also always function properly and is another required function of the system, as it is the final step when users create and decide on their schedules. If this function fails, users will be unable to select the schedules they created, defeating the entire purpose of the system.

These high importance items require unit testing as load/unload are required to save user data and keep it secure, while the other items all provide functionality directly to the creation of schedules, the main purpose of the system. Login and logout will be used at least

once every time the system is used or more if multiple users want to switch. The schedule related items may not be used every time the user logs on to the system, but each of these are mandatory whenever a user wants to make a schedule. This will occur at least once per semester and can be used multiple times to either change their existing schedule or view potential future schedules. Having any of these items fail is extremely severe. If any of the schedule related items fails, it becomes either extremely tedious or impossible to create and select a schedule. If logging in fails, the user cannot even begin creating their schedule. If logging out fails, there is nothing stopping another user from maliciously creating a poor schedule without their consent.

**Low Importance**
The following requirements provide additional functionality and ease of use for the user but are not directly related to creating course schedules Return to Homepage: Returning to the homepage is an important item as the user can access four different pages from the home page. Although not critical due to the user's ability to go to the previous page, the return to homepage item should still function properly as an added feature that that allows for switching between pages more easily.

1. **View Academic Record:** The view academic record item should always work properly as it is accessed by the home page and allows the user to see their academic record by clicking "Student Profile". Failure of this item would block the user from accessing their own information including program, courses taken, and changing their password as well.

2. **View Course Listing:** The view course listing item must always function properly in order for the user to check courses in a semester as well as their different sections and times. Failure would give users Fno way to check whether the courses they need are being offered in a given semester or to plan their schedules ahead of time.

3. **View Academic Progress:** The view academic progress item is important as it helps the user see what courses he's taken in relation to what's being offered. This helps with planning the schedule and makes it easier for the user to see the progress in their current program. If this item fails, the system still works properly but it becomes more of a hassle for users to keep track of courses they have done and need to take.

4. **View Course Details:** Viewing course details must always function properly as the entire course listing page becomes practically useless without this item. The course listing shows the location and time of the courses selected; if the system fails to show these details, users will gain no information on the courses other than that they're offered.

5. **View News Feed:** The news feed feature displays recent news from the New York Times on the home page. Having this item fail doesn't affect the functionality of the system, but it should function properly regardless as an extra feature on the home page.

**Omitted**
The following requirements were scoped out and are no longer requirements within the system.

1. View Student Academic Record
2. Modify Course Details
3. Change language
4. View professor profile

### 15.1.2 Untested Items of Interest

The system was only tested using manual input, so the advantages of automatic testing were missed out on. For example, the system was not tested under a heavy workload. This could have been done by running a program generating many users and keeping track of performance and functionality. This would check if the system is capable of remaining functional instead of crashing during high stress periods, as well as confirmation of the non-functional requirements.

## 15.2 Test Cases

### 15.2.1 Unit Testing

**Class Block Unit Testing:**
Class block is one of the most important classes in the program, because it deals with time management of sections. It is the parent class that holds the time and the weekday information for a particular section. It has one function called overlaps which returns true or false if it overlaps other blocks. It is used by other classes to determine if a schedule is compatible with a certain section or preference time block and thus testing this class is important.

The test consisted of 4 tests that test the function as shown in Figure 21.

**Class Grade Unit Testing:**
The Grade Class is a class that compares grades vs passing grades of a certain course. It returns true if a grade is passed by a certain grade. In certain courses, grades may be passed with C- or D- which makes it hard if entire the class was a function. By separating the code of this function and incorporating this into class like this, managing letter grades is a lot easier. It is a important class and the testing unit can be shown in Figure 22.

**Time Table Schedule Generation:**

The schedule class written in JavaScript, displays view for the time grid and schedule of the retreived data. It was tested using the white-box method. A driver was written to test

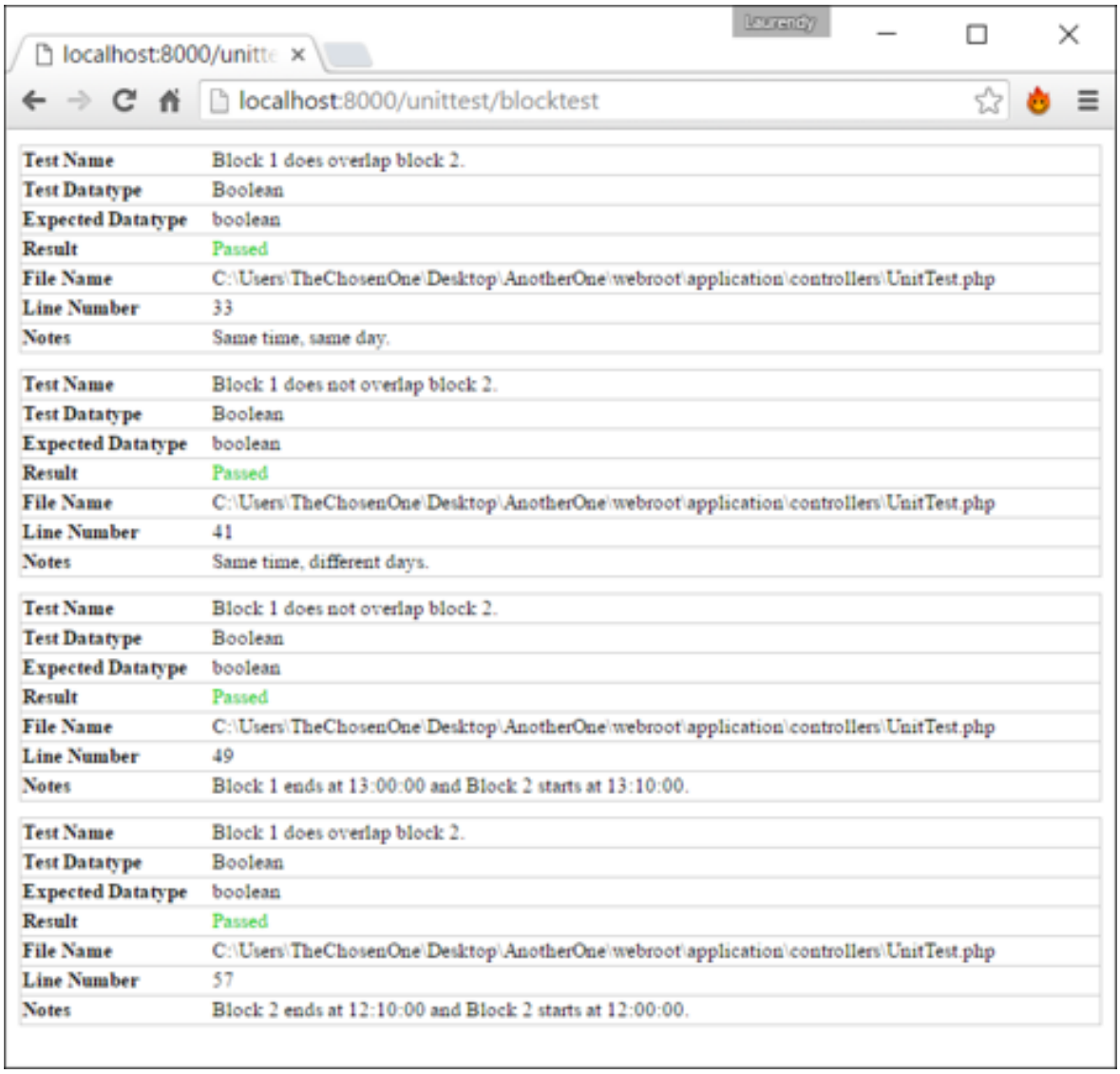
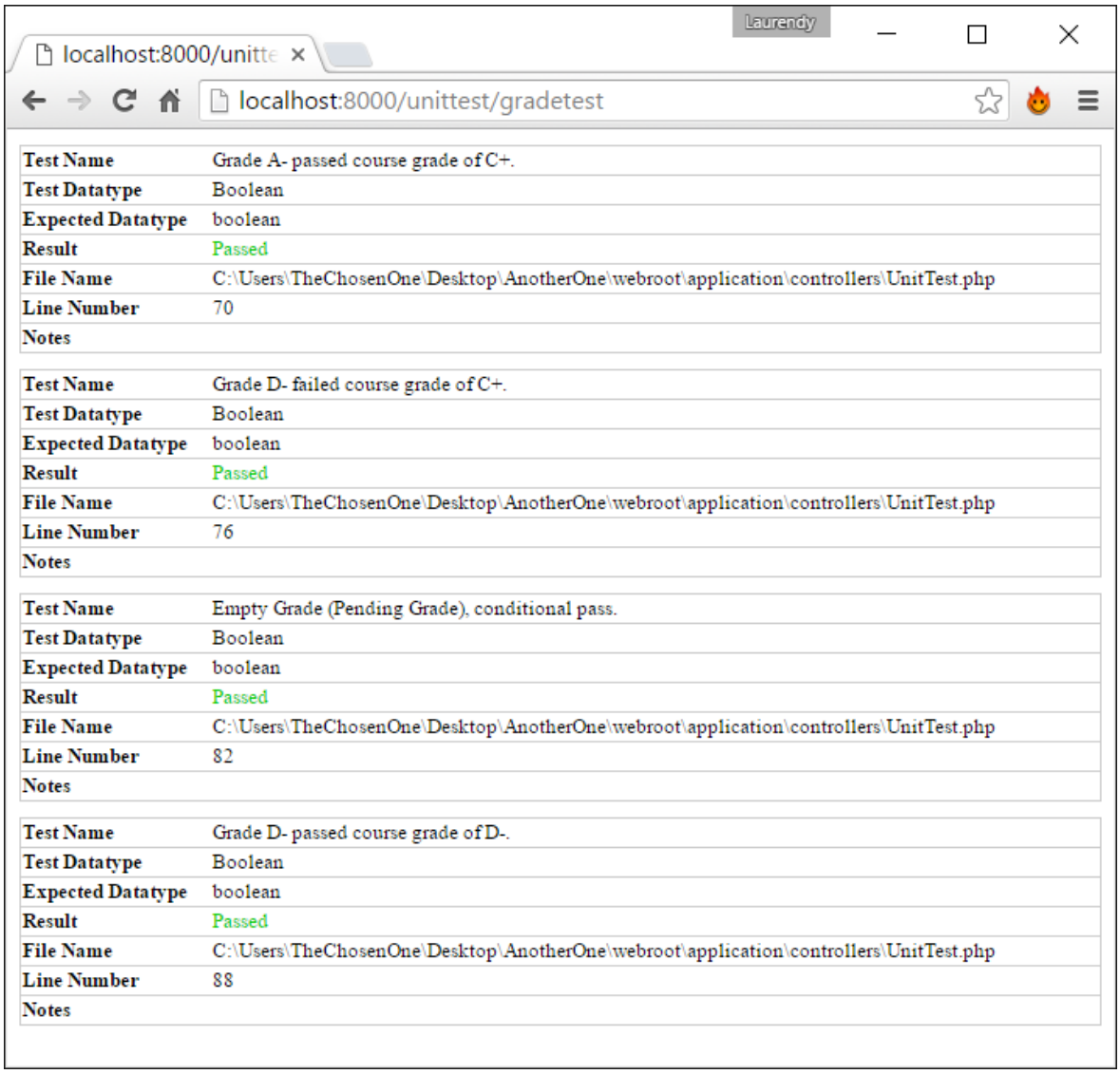



Figure 21: Class Block Unit Testing

for the class to see if it works properly when generating the time tables. To run the test, an expected html table string was compared with the generated table with over 9000 iterations with the same json data input.

The driver code for this test can be shown in Figure 23. To demonstrate the test, a JavaScript driver was used. On line 13 is the input data, and the expected html is on line 16. A variable num_errors keeps track of the number of errors the driver might output. A for-loop is used to check for consistency and validity. Between lines 21 to 23 is the schedule object being created that renders the html into the view, and in addition, return the html table. On line 24 is the comparison between the generated table vs the expected table. If the table




| | |
|---|---|
| **Test Name** | Grade A- passed course grade of C+. |
| **Test Datatype** | Boolean |
| **Expected Datatype** | boolean |
| **Result** | Passed |
| **File Name** | C:\Users\TheChosenOne\Desktop\AnotherOne\webroot\application\controllers\UnitTest.php |
| **Line Number** | 70 |
| **Notes** | |

| | |
|---|---|
| **Test Name** | Grade D- failed course grade of C+. |
| **Test Datatype** | Boolean |
| **Expected Datatype** | boolean |
| **Result** | Passed |
| **File Name** | C:\Users\TheChosenOne\Desktop\AnotherOne\webroot\application\controllers\UnitTest.php |
| **Line Number** | 76 |
| **Notes** | |

| | |
|---|---|
| **Test Name** | Empty Grade (Pending Grade), conditional pass. |
| **Test Datatype** | Boolean |
| **Expected Datatype** | boolean |
| **Result** | Passed |
| **File Name** | C:\Users\TheChosenOne\Desktop\AnotherOne\webroot\application\controllers\UnitTest.php |
| **Line Number** | 82 |
| **Notes** | |

| | |
|---|---|
| **Test Name** | Grade D- passed course grade of D-. |
| **Test Datatype** | Boolean |
| **Expected Datatype** | boolean |
| **Result** | Passed |
| **File Name** | C:\Users\TheChosenOne\Desktop\AnotherOne\webroot\application\controllers\UnitTest.php |
| **Line Number** | 88 |
| **Notes** | |

Figure 22: Class Grade Unit Testing

is not the same as the generated table then num_errors will increment by one. Once the test has completed, the JavaScript console will print the number of errors.

### 15.2.2 Requirements Testing

These are the tests for the use cases mentioned in the functional requirements section in Deliverable 1. (Section 2.1) Anything in gray or has a result of N/A is scoped out of our program.

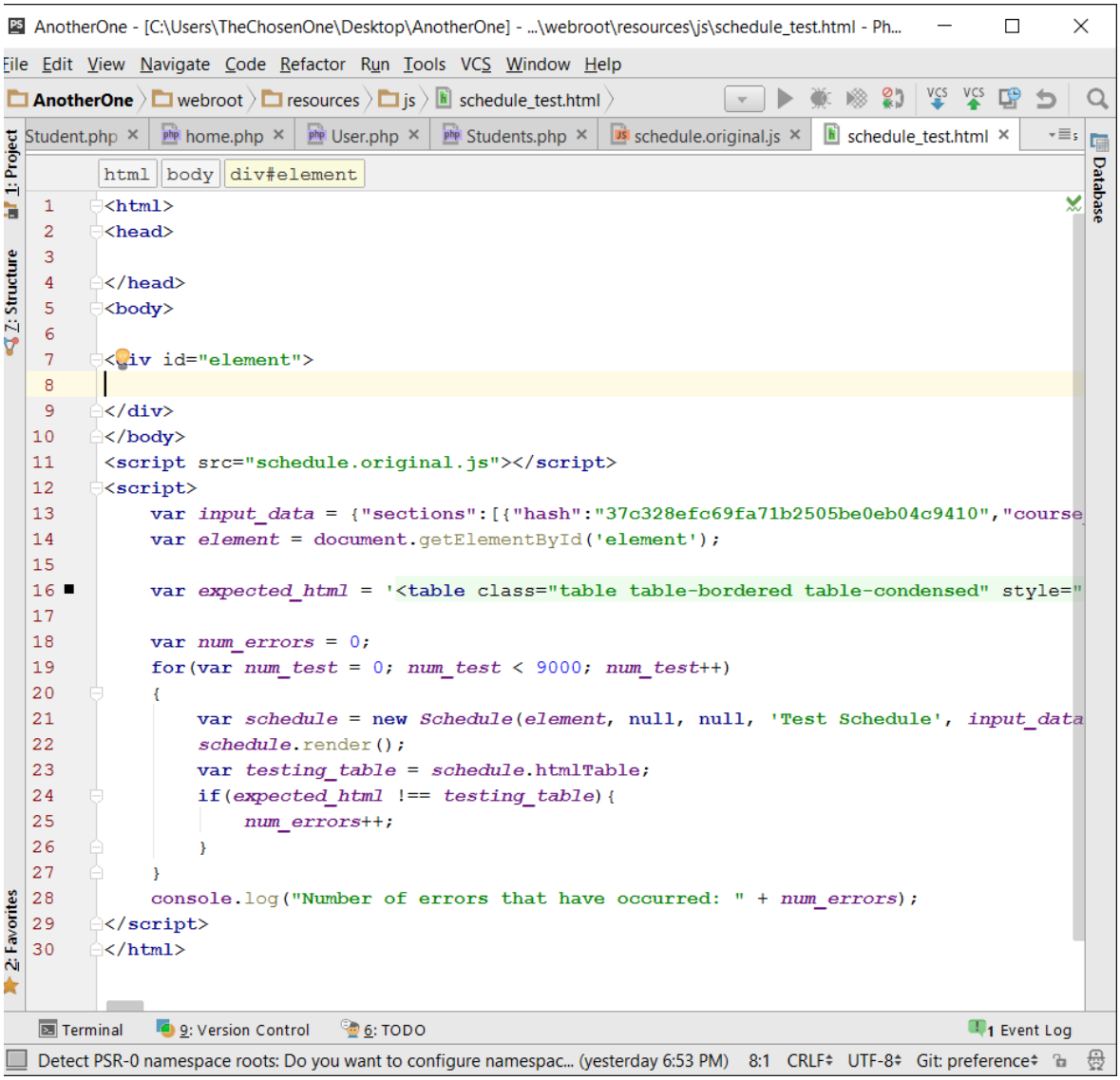



Figure 23: Front End Scheduler Driver Code

| Login | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 1.01 | User inputs valid username and password | Username: ”john_smith” Password: ”password” | User is logged in and homepage is displayed | Pass |
| 1.02 | User inputs valid username and invalid password | Username: ”john_smith” Password: ”cheese” | User remains on login page and a message displays: ”Did not match any records. Try again” | Pass |
| 1.03 | User inputs invalid username and valid password | Username: ”quinnNv” Password: ”password” | ”Did not match any records. Try again” | Pass |
| 1.04 | User inputs invalid username and invalid password | Username: ”quinnNv” Password: ”cheese” | ”Did not match any records. Try again” | Pass |
| 1.05 | User inputs un-matching username and password | Username: ”dwayne_johnson” Password: ”password” | ”Did not match any records. Try again” | Pass |
| 1.06 | User inputs all caps version of username and valid password | Username: ”JOHN_SMITH” Password: ”password” | User is logged in and homepage is displayed | Pass |
| 1.07 | User inputs valid username and all caps version of valid password. | Username: ”John_smith” Password: ”PASSWORD” | ”Did not match any records. Try again” | Pass |
| 1.08 | User tries to do SQL injection | Username: ”john_smith” Password: ” ‘ or 1 = 1–” | ”Did not match any records. Try again” | Pass |

| Logout | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 2.01 | User is logged in and attempts to log out | Click on ”Sign Out” button | User is logged out and login page is displayed | Pass |
| 2.02 | User is not logged in and attempts to log out | None | ”Sign out” option not displayed at login page | Pass |

| Return to Homepage | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 3.01 | User is on student profile page and clicks on logo to return to homepage | Click on Logo | User is redirected to the home page | Pass |
| 3.02 | User is on course registration page and clicks on logo to return to homepage | Click on logo | User is redirected to the home page | Pass |
| 3.03 | User is on search page and clicks on logo to return to homepage | Click on logo | User is redirected to the home page | Pass |
| 3.04 | User is on view schedule page and clicks on logo to return to homepage | Click on logo | User is redirected to the home page | Pass |

| View academic record | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 4.01 | User goes to student profile page to view academic record | Click on ”Student Profile” tab | Student profile page is loaded and academic record is displayed | Pass |

| View student academic record | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 5.01 | Admin goes to student profile page to view academic record | Click on ”Student Profile” tab | Student profile page is loaded and academic record is displayed | N/A (Scoped out) |

| View course listing | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 6.01 | User is on homepage and clicks on view available courses tab. | Click on ”View Available Courses” tab | User is brought to the search page. | Pass |
| 6.02 | User is on search page, changes nothing and clicks on search button. | None | All available courses for fall 2016 will be displayed. | Pass |
| 6.03 | User changes the semester and nothing else. | Pick ”Summer 2014” from drop down menu. | All available courses for summer 2014 will be displayed. | Pass |
| 6.04 | User enters a valid course code and leaves number blank. | Subject: ”soen” Number: ”” | All available SOEN classes for fall 2016 are displayed. | Pass |
| 6.05 | User enters invalid course code. | Subject: ”abab” | ”Warning! No results were found!” | Pass |
| 6.06 | User enters valid course code (in the software engineering sequence) and invalid course number. | Subject: ”soen” Number: ”1” | ”Warning! No results were found!” | Pass |
| 6.07 | User enters a valid number and leaves course subject blank. | Subject: ”” Number: ”341” | ”Warning! The Course Code field is required.” | Pass |
| 6.08 | User enters an invalid number. | Number: ”1” | ”Warning! The Course Code field is required.” | Pass |
| 6.09 | User enters an invalid course subject and a valid number. | Subject: ”abba” Number: ”341” | ”Warning! No results were found!” | Pass |
| 6.10 | User enters a valid course subject and a valid number. | Subject: ”soen” Number: ”341” | The given course will be displayed. | Pass |
| 6.11 | From the search result page, return to search page. | Click on ”New Search” link | The search page is displayed. | Pass |

| View academic progress | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 7.01 | User's academic progress is displayed on the front page. | None | Course sequence is displayed and courses the user has already taken are highlighted in green. | Pass |
| 7.02 | Correctly highlights courses user can take. | None | Courses the user can take are highlighted in yellow. | Pass |

| View course details | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 8.01 | Details such as location and time are displayed for each course in the search results | None | Details are displayed. | Pass |

| Modify course details | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 9.01 | Admin enters valid course times. | "5:45" | Allotted time of the course is changed | N/A |
| 9.02 | Admin enters invalid course times. | "11:11" | "That input is invalid. Please enter something else." | N/A |
| 9.03 | Admin enters valid classroom details. | "H-415" | Classroom details are changed. | N/A |
| 9.04 | Admin enters invalid classroom details. | "H-999" | "That input is invalid. Please enter something else." | N/A |
| 9.05 | Admin enters valid professor. | "Aiman Hanna" | Professor is changed. | N/A |
| 9.06 | Admin enters invalid professor. | "Kahless the Unforgettable" | "That input is invalid. Please enter something else." | N/A |

| View news feed | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 10.01 | News feed is properly displayed on home page. | None | There is breaking news displayed on the home page. | Pass |
| 10.02 | User clicks on the breaking news link. | Click on ”Gunfire in Republic of Congo as Police Stations Attacked” news link | User is redirected to that news story on the nytimes website. | Pass |

| Set schedule preferences | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 11.01 | User adds time to preference scheduler | Wednesday 10:00 AM to 3:00 PM | Wednesday 10:00 AM to 3:00 PM ‘Successfully added preference!’ | Pass |
| 11.02 | User tries to add the wrong values to the time preference in the scheduler | Wednesday 3:00 PM to 10:00 AM | ‘Error: End time earlier than beginning time!’ | Pass |

| Select course for schedule | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 12.01 | User goes to registration page. | Click on ”Register Now” tab | Enrollment page is loaded. | Pass |
| 12.02 | User enters a valid course in search bar and clicks on it. | Search: ”Comp 248” | Result will be added under ”Courses”. | Pass |
| 12.03 | User enters an invalid course in the search bar. | Search: ”Cool 101” | No options can be selected. Nothing happens when enter is pressed. | Pass |
| 12.04 | User enters a course that is already under Courses | Search: ”Comp 248” | Search result will be displayed. Selecting the option will cause a pop up message: ”Course already listed.” | Pass |
| 12.05 | User enters a course for which they do not have a prerequisite. | Search: ”Comp 249” | Pop up. ”Prerequisite not complete: [course code(s) and name(s)]” | Pass |
| 12.06 | User adds a 6th course. | Search: ”Phys” | Pop up. ”Exceeds the maximum number of courses per semester, 5.” | Pass |
| 12.07 | User enters a number corresponding to a course number in the search bar. | Search: ”201” | Classes with that number in their course code will be displayed. | Pass |
| 12.08 | User tries the auto-pick option. | Click ”Auto-Pick” button. | A random class will be added to course. | Pass |

| Generate schedules | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 13.01 | Drop down list of schedules are properly displayed. | Click ”Generate” button. | List of schedules. | Pass |
| 13.02 | User has courses which have time conflicts. | Courses selected previously. Click ”Generate” button. | ”Generator found 0 results! Try adding one course at a time.” | Pass |
| 13.03 | User is following sequence and has four classes in courses. | Courses selected previously. Click ”Generate” button. | List of all possible schedules | Pass |
| 13.04 | User clicks button when no new courses are added. | Click ”Generate” button. | ”Generator found 0 results! Try adding one course at a time.” | Pass |

| Select schedule | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 14.01 | Schedule details and time grid are properly displayed. | Select ”Schedule #1” | Courses’ course codes, sections, times and classrooms will be displayed. | Pass |
| 14.02 | Details properly change when choosing different schedules. | Select ”Schedule #2” | Appropriate details such as sections, times and classroom change. | Pass |
| 14.03 | Click button when no schedules are selected. | Click on ”Commit & Enroll” button | Nothing happens. | Pass |

| Change language | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 15.01 | User clicks on the French language option | Click on ”FR” | Language is changed from English to French | N/A |
| 15.02 | User clicks on the English language option | Click on ”EN” | Language is changed from French to English | N/A |

| View professor profile | | | | |
|---|---|---|---|---|
| **Test ID** | **Description** | **Input** | **Expected Result** | **Result** |
| 16.01 | User views a professor's rating via "Rate-my-Professor" | Click on a professor's name | Professor's "Rate-my-Professor" profile is displayed | N/A |

### 15.2.3 Stress Testing

Mytinerary is expected to be able to deliver its functionality at all points in time. However, extreme situations may occur that hinder the applications ability to deliver the functionality as expected. One of these situations may include the registration period where the application will be expecting a multitude of unique users attempting to use many of the applications features. A couple of the features that will certainly be under stress of high demand include searching for courses, generating schedules, and dropping courses. Because these are extremely important actions in particular, any server downtime here would be intolerable. Since this is an inevitable extreme situation, one solution would be to use an automated test script written in any common scripting language such as Python or Perl. This automated script will be used across a network of several computers to simulate thousands of unique students using all of the applications features across multiple browsers and tabs. The goal of these tests will be to pinpoint exactly where and when the system performance dips below the expected benchmarks and assign solutions according to the tests. An obvious solution would be to simply increase the server capacity by a specific amount according to the results of the automated tests.

**Stress Testing Results**
The following results was obtained by JMeter, an open source load testing tool that can simulate simultaneous user activities on many different types of servers. It works by sending multiple user requests to a target server to test the functionality and performance of the resources by collecting data. The collected data can then be used to calculate statistical information about Mytinerary, such as, reports, tables, graph, etc. The following table is the summary result of Mytinerary's performance from stress testing.

### 15.2.4 Security Testing

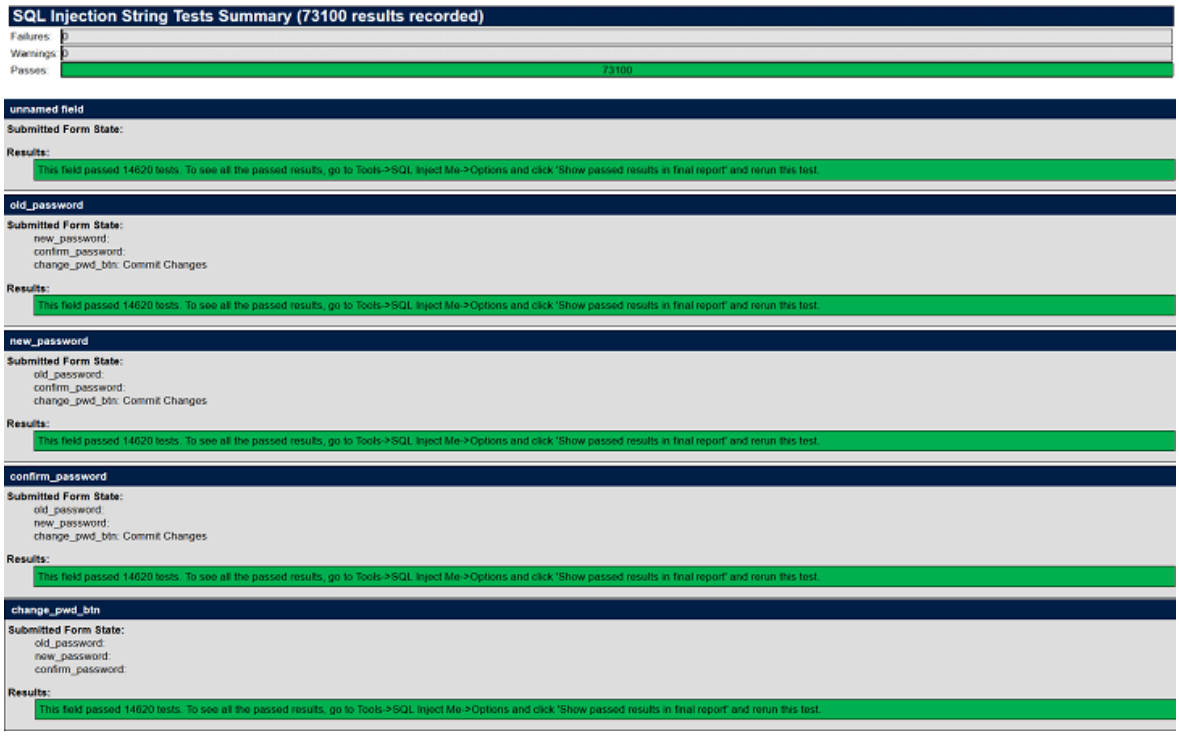
**SQL Injection**
SQL injection was tested by SQL Inject Me. This tool was run on Login, Profile and View Available Courses page. The tool works by injecting SQL statements into the html forms and executing it to receive error messages from the database which is then outputted to the rendered HTML page. These errors could contain privacy information to modify or manipulate the application that shouldn't be accessed. These attacks don't actually harm the application but can obtain vital information to find entries for an attack. The test results below shows that no failures or warnings occurred during the simulation SQL Injection.

| Stress Testing with 0.0% Errors | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Action** | **# Samples** | **Average** | **Min** | **Max** | **Std. Dev.** | **Throughput** | **Avg. Bytes** |
| Login | 400 | 24255 | 1248 | 33527 | 10207.55 | 6.1 / sec | 8495 |
| | 1 | 1193 | - | - | - | 50.2 / min | 8495 |
| View Profile | 400 | 27912 | 1232 | 39222 | 12109.16 | 5.4 / sec | 442 |
| | 1 | 1194 | - | - | - | 47.9 / min | 442 |
| Generate Schedule | 400 | 27030 | 1240 | 36986 | 11311.46 | 5.6 / sec | 442 |
| | 1 | 1886 | - | - | - | 31.7 / min | 442 |
| View Schedule | 400 | 25631 | 1192 | 37422 | 11324.34 | 5.7 / sec | 442 |
| | 1 | 1189 | - | - | - | 49.1 / min | 442 |
| View Courses | 400 | 32209 | 1220 | 44905 | 13083.14 | 4.5 / sec | 4237 |
| | 1 | 2300 | - | - | - | 49.1 / min | 4237 |
| Logout | 400 | 26261 | 1188 | 36924 | 11109.81 | 5.7 / sec | 542 |
| | 1 | 1307 | - | - | - | 44.0 / min | 542 |

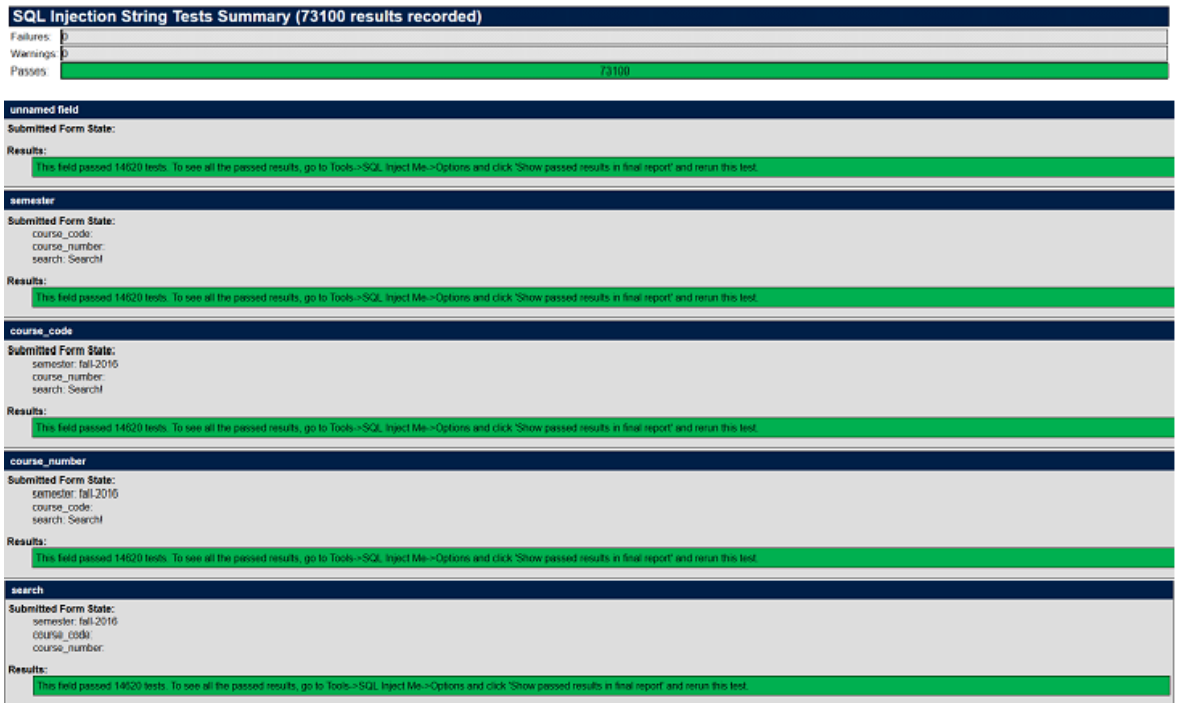
**Login**


Security Compass
Test Results
SQL Injection String Tests Summary (43860 results recorded)
Failures: 0
Warnings: 0
Passes: 43860
SQL Injection String Test Results
login_name
Submitted Form State:
password:
signin_btn: Sign In
Results:
This field passed 14620 tests. To see all the passed results, go to Tools->SQL Inject Me->Options and click 'Show passed results in final report' and rerun this test.
password
Submitted Form State:
login_name:
signin_btn: Sign In
Results:
This field passed 14620 tests. To see all the passed results, go to Tools->SQL Inject Me->Options and click 'Show passed results in final report' and rerun this test.
signin_btn
Submitted Form State:
login_name:
password:
Results:
This field passed 14620 tests. To see all the passed results, go to Tools->SQL Inject Me->Options and click 'Show passed results in final report' and rerun this test.

## Registration

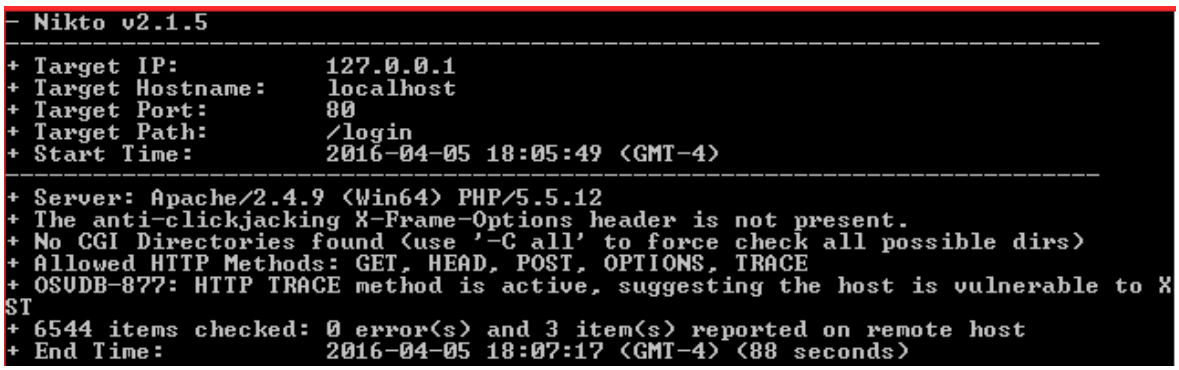


## View Available Courses



**Nikto Test Results**

Nikto was used to scan web servers for any dangerous files/CGI's, to check if the server was up to date and to check for outdated versions. The screenshot shot below displays no major issues.




**Dictionary Attack**

Finally, a Python script was written which simulated a dictionary attack against the user "john_smith", which can be seen in Appendix A. Unfortunately, because the imaginary user used the weak password "password", his account was compromised almost instantly. Based on the result of this test, we have concluded that 1) password strength must be evaluated before a password is set; and 2) a timeout system should be in place to prevent brute-forcing of an account.

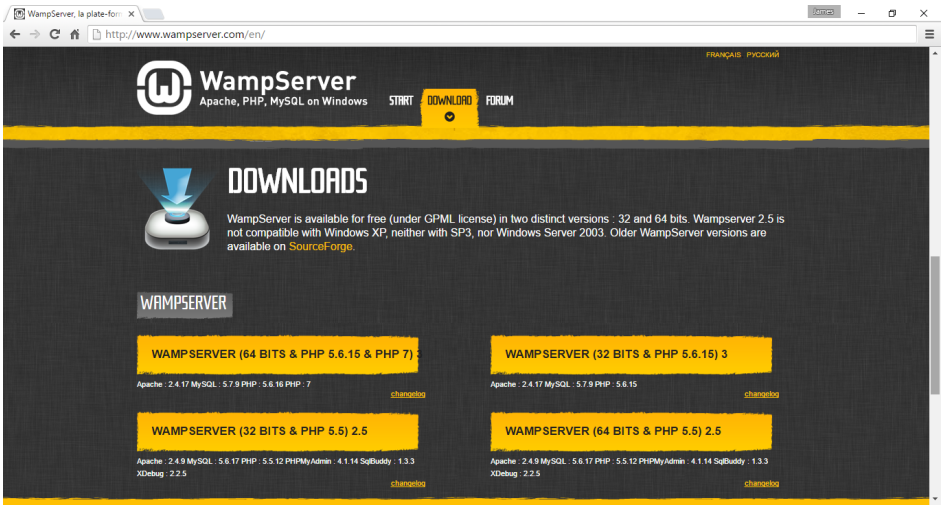
# 16 System Delivery

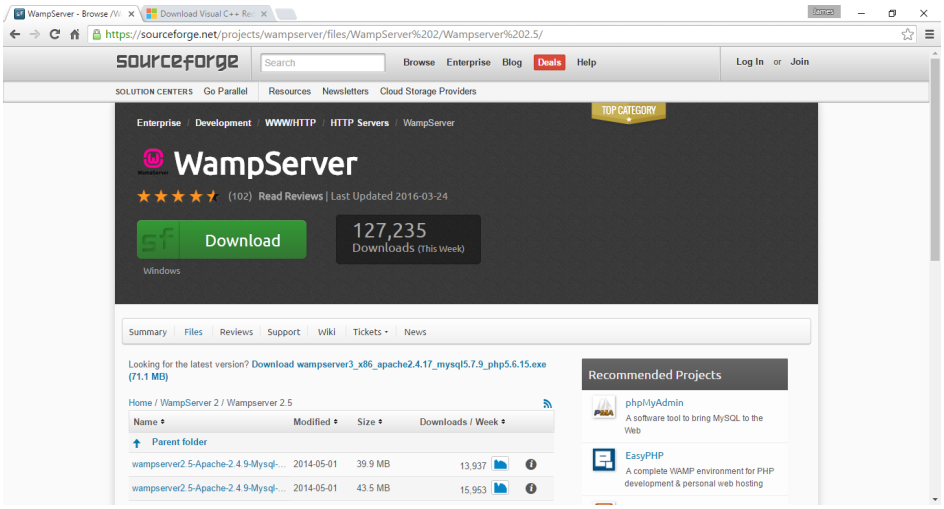
## 16.1 Installation Manual

In order to successfully install Mytinerary on your server, you must have the Windows operating system version 8 or later installed as well as Visual Studio version 2012 exactly.

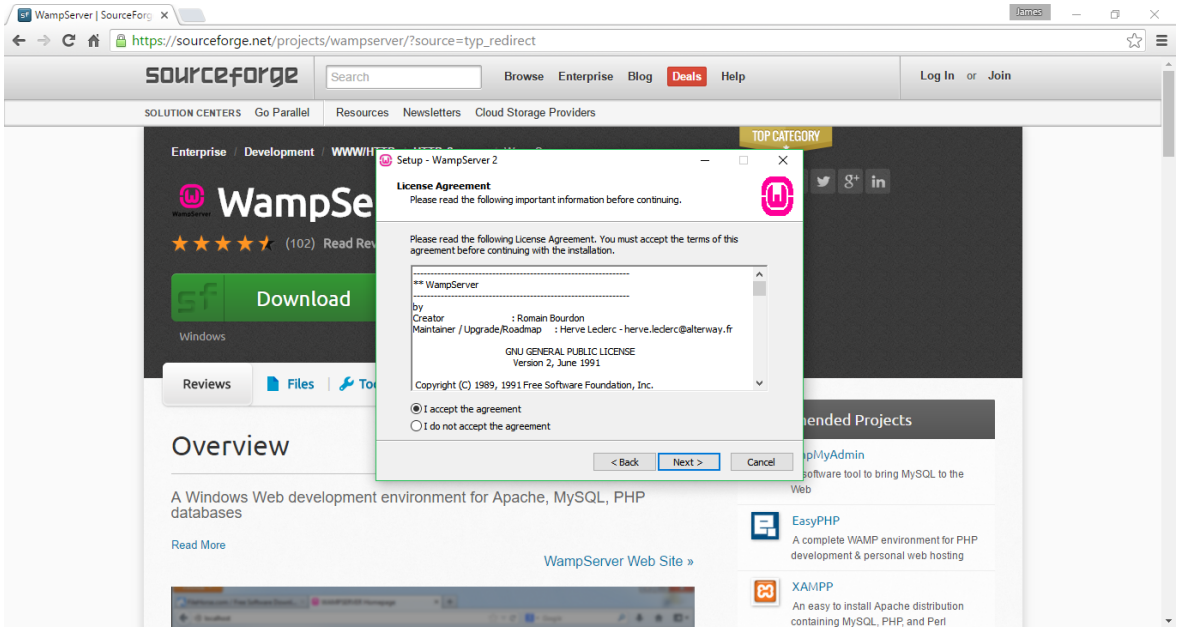
Please complete the following steps exactly to install Mytinerary on your server:

1. Download WAMP server by visiting http://wampserver.com/en, clicking on the link "WAMP SERVER (64 BITS & PHP 5.5) 2.5" in the downloads section, and following the provided instructions. If the domain is not available, you can download WAMP server directly from SourceForge through the URL https://sourceforge.net/projects/wampserver/files/WampServer%202/Wampserver%202.5.

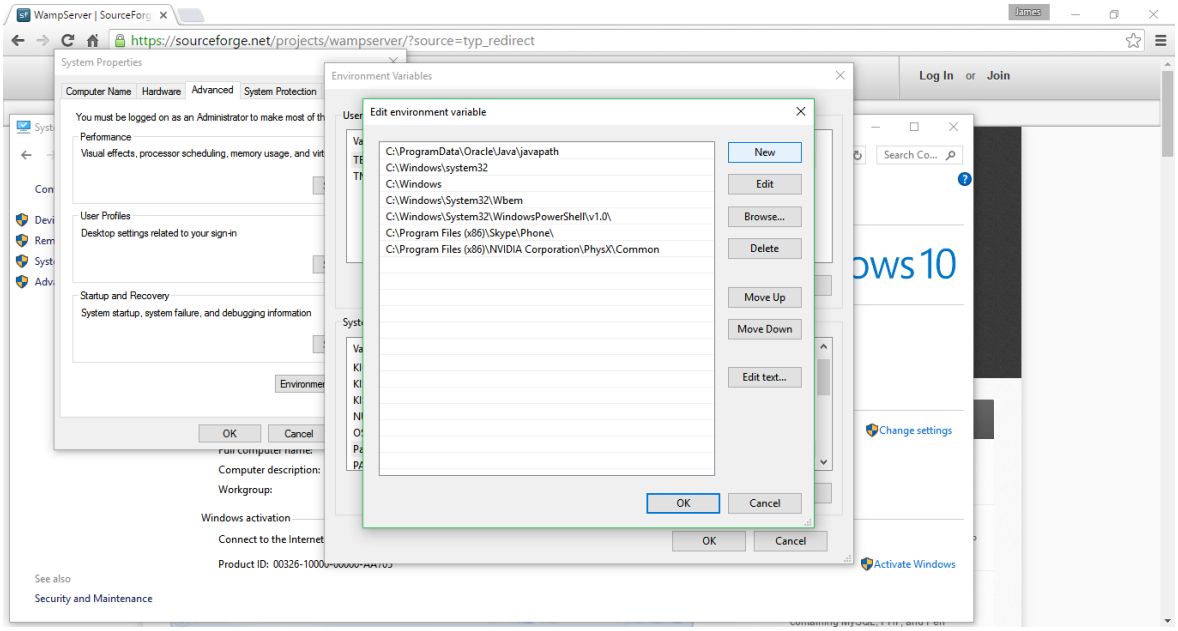


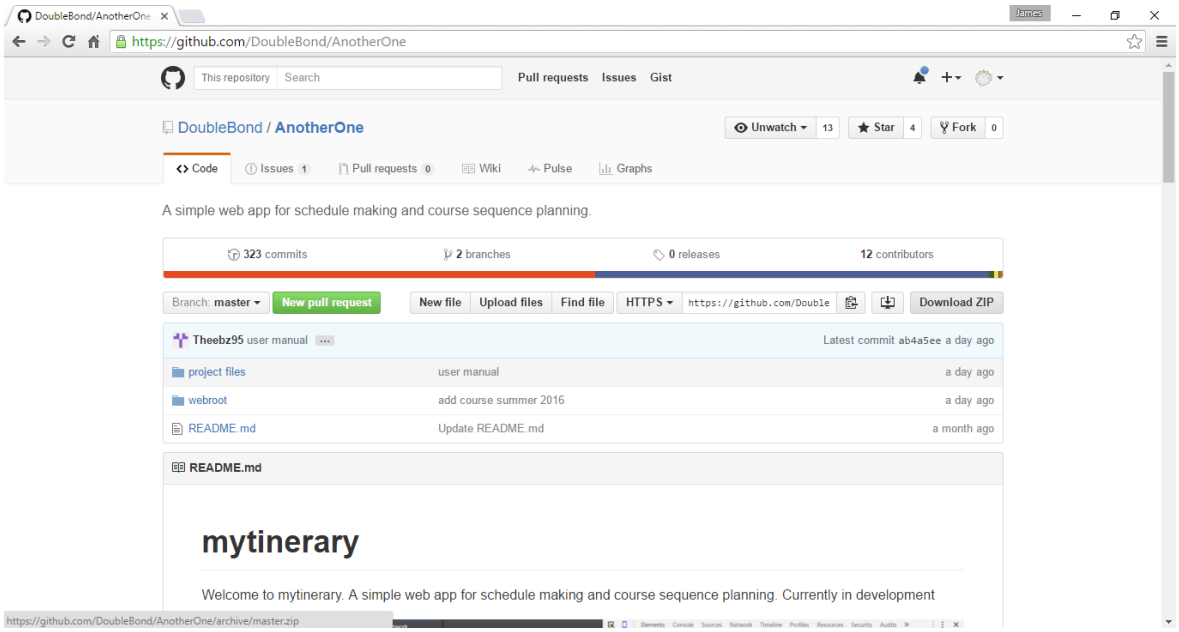
2. Install the WAMP server by running the downloaded executable, which will open an installation wizard. Accept the terms of agreement a select a destination folder for the WAMP server and complete the installation. It is important to take note of the destination folder as it will be used later in the installation process. It is not necessary to run the WAMP server after the installation completes.



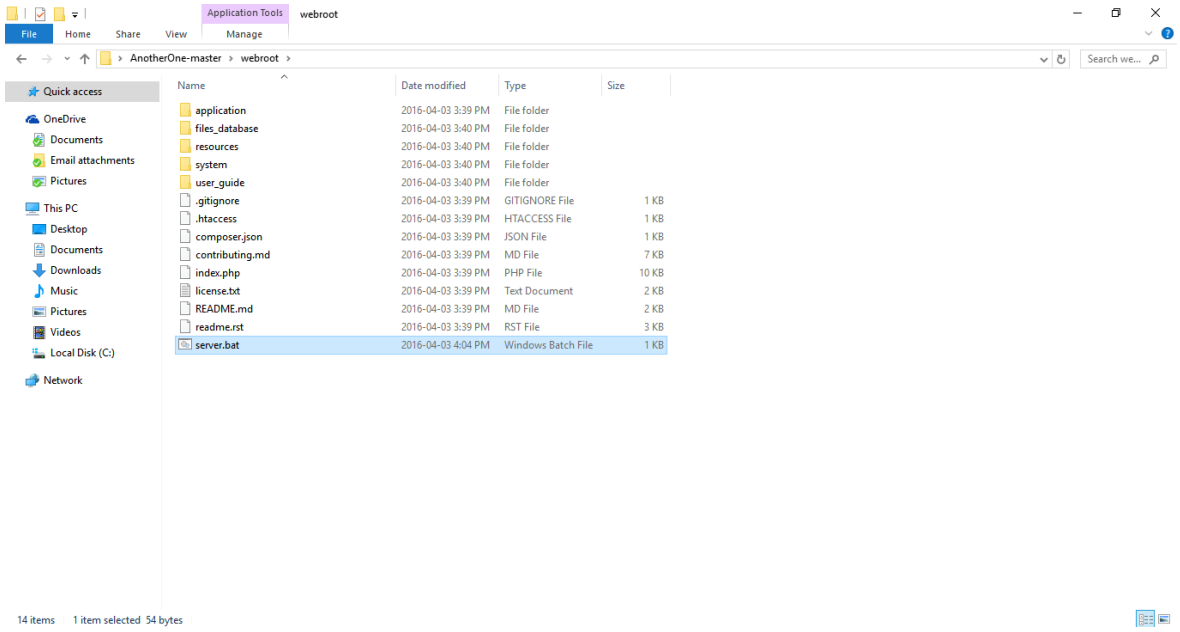
3. The folder containing the PHP interpreter must be added to your PATH environmental variable. Open the advanced system settings and click on the environmental variables. Under ”System variables”, add ”¡destination folder¿/wamp/bin/php/php5.5.12” to the PATH variable, with ¡destination folder¿ replaced by the destination folder of the WAMP server in step 3.



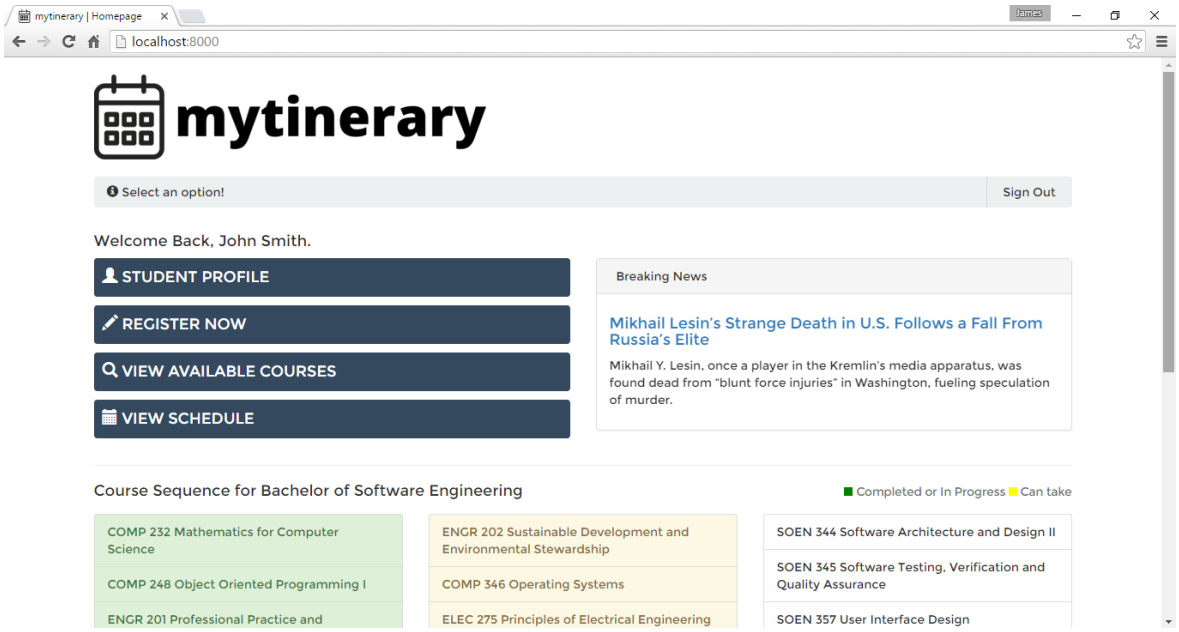
4. Visit https://github.com/DoubleBond/AnotherOne and download and unzip the project's ZIP file.



5. To start the Mytinerary server, enter the webroot folder and run “server.bat”. This will open a controlling terminal window which, if closed, will stop the server from running.



6. Open a Web browser of your choice and visit ”localhost:8000” to access Mytinerary. You should a window similar to what is shown below.



## 16.2 User's Manual

The user's manual consists of clear instructions on how to use the system. The instructions explain how to enroll in a course, view the schedule, view the student record, view all offered courses and the student's course sequence progress.

### 16.2.1 Login Screen

When initially accessing the website, the user will see the login page. The user can provide login information in order to access the homepage to manage his/her account or view the courses provided by the school. Steps on how to log in the application:

1. Enter the login name provided by the school.
    - An error 'Did not match any records. Try again' may appear if the login name is not found in the database.
2. Enter the password provided by the school.
    - An error 'Did not match any records. Try again' may appear if the wrong password is entered.
3. If a user successfully logs in, the homepage will appear as shown in figure 2.

To access the the courses provided by the school without logging in, simply click 'COURSE' on the top right corner as shown in Figure 24.

### 16.2.2 Home Page

When a user is logging in the homepage, it will load the corresponding homepage to the type of user: admin or student. Once logged in, the student has 5 different options from which he can select: 1) Student Profile, 2) Register Now, 3) View Available Courses, 4) View Schedule and 5) Sign Out. Administrators have 2 different options:, 1) edit courses, 2) manage sections and 3) manage students. The course sequence displays the courses in order the student should follow, according to the their program. In this case, Bob Smith is enrolled in Software Engineering as shown in Figure 25.

### 16.2.3 The Course Sequence

The courses shown within a green box indicates that the student has already registered for those courses. The courses shown within a yellow box indicates the courses that the student is allowed to register for, i.e., the student has already met the prerequisites for those courses, and therefore can add those courses. The courses shown within a white box cannot be added to the schedule because the student has not met the prerequisites for those courses

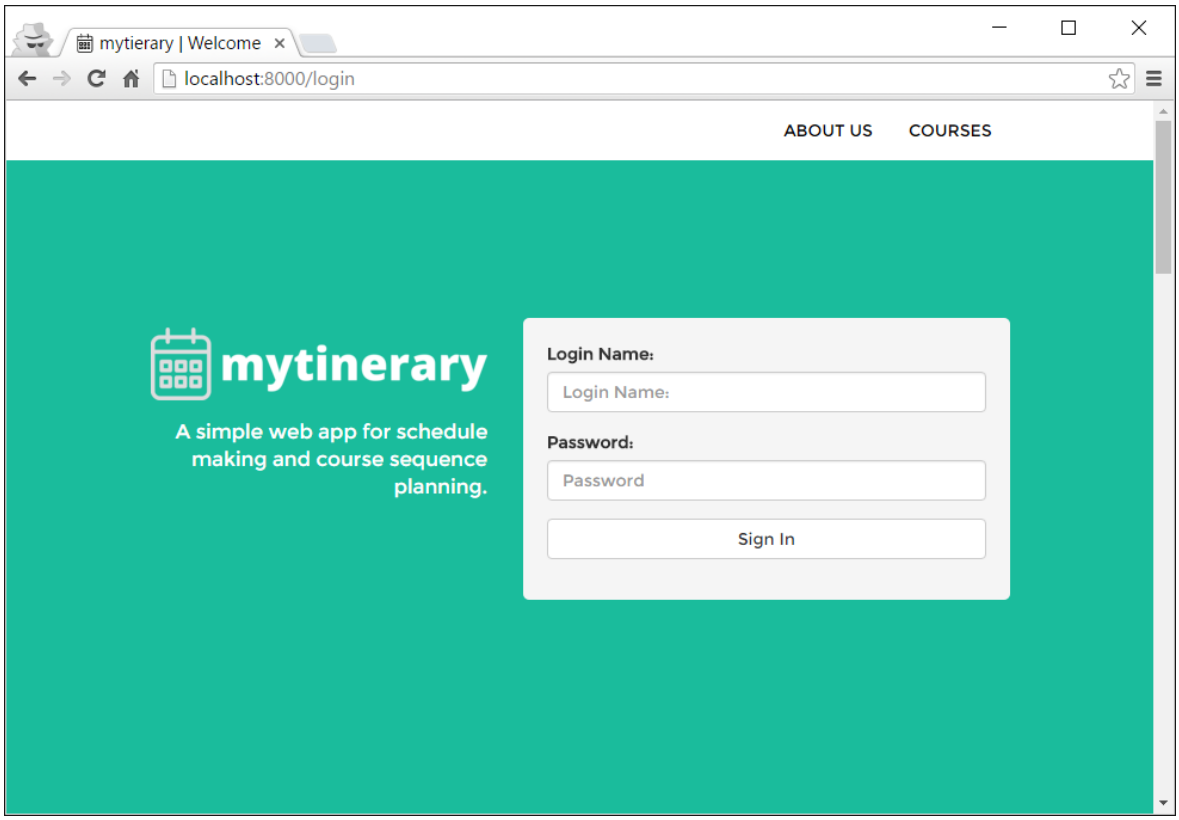



Figure 24: Login Page

### 16.2.4 Student Profile Page

The Student Profile page shows the student's academic record. Upon selecting this tab, students will be able to see their personal info such as student ID and their program. In addition, the page will display a list of courses that the students has completed and the classes they are currently registered in. Courses are grouped relative to the semester taken and the grade for the courses are displayed. While on this page students may also edit their password.

### 16.2.5 How to change the current user's password

To reset the user's password the student must access the student's profile page to gain access to the reset password form as shown in Figure 26.

Steps to changing the current password.

1. Enter the old password.
    - Note: The message 'The old password seems to be incorrect' will appear if the old password does not match against the current password.

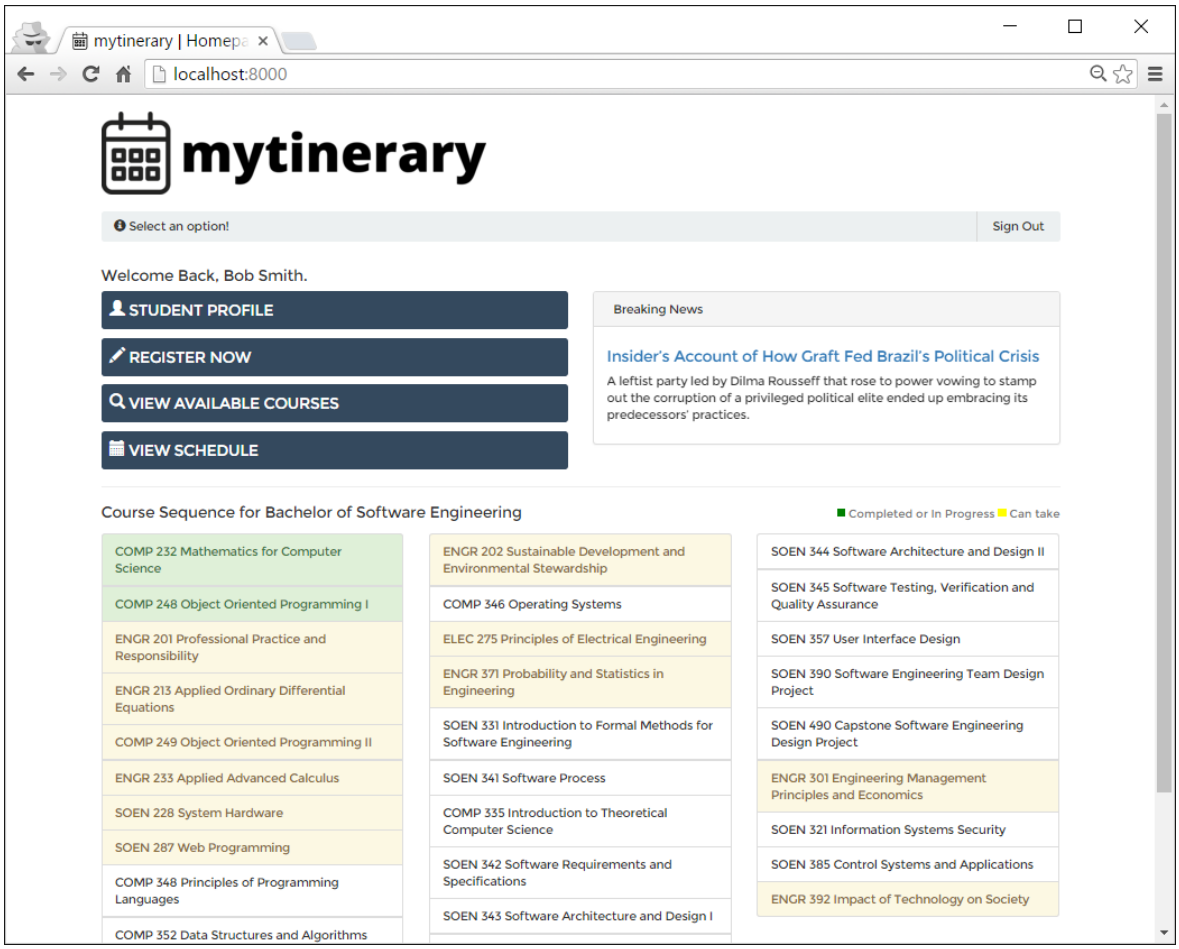



Figure 25: Home Page

2. Enter the new password.
    - The new password must contain 8 characters or more.
3. Confirm the new password.
    - The confirmation password must match the new password field.
4. A message 'Successfully updated the password!' will appear if the fields are entered correctly.

### 16.2.6 Registering for a Semester

When clicking on the "Register Now" tab, a list of available semesters will be displayed as shown in Figure 27. Students may click on any of the given semesters to begin registering courses for that semester. Once a semester has been selected students will be directed to

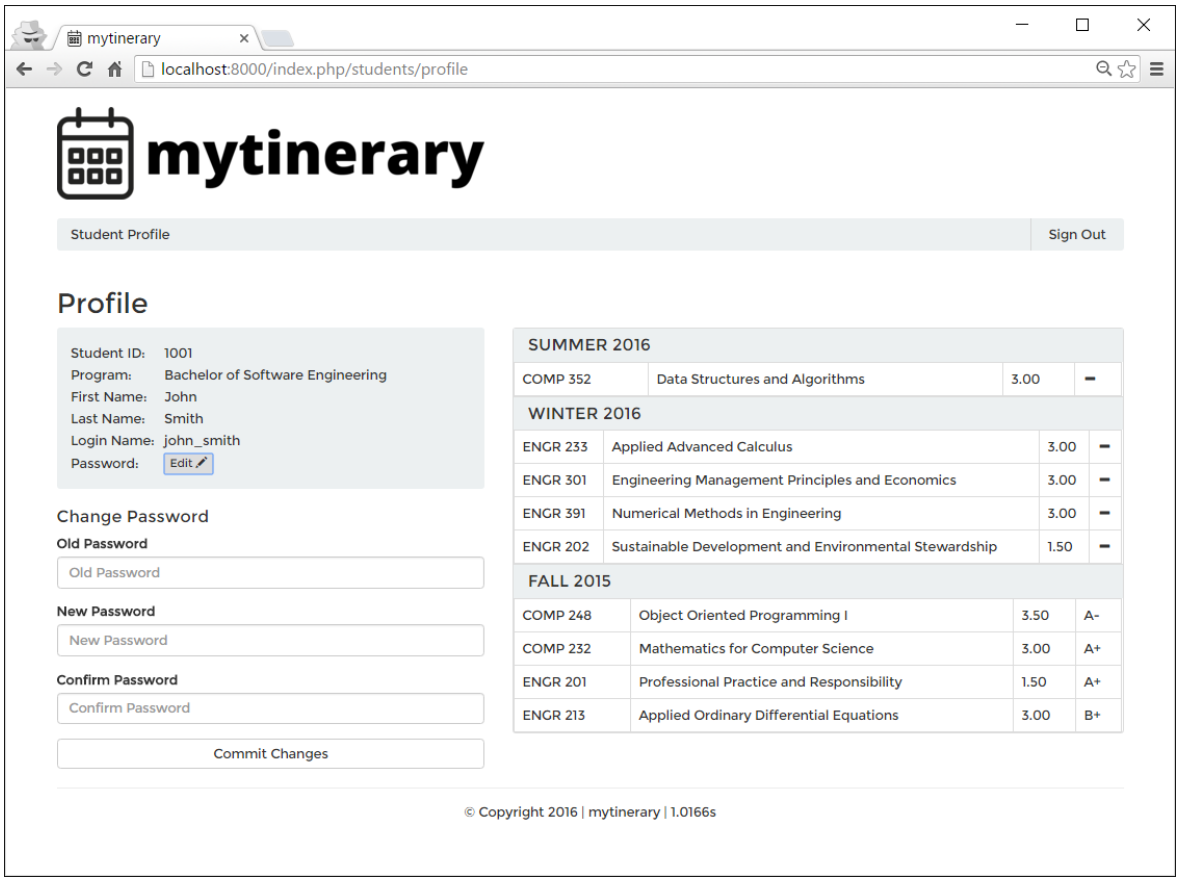



Figure 26: Student Profile Page

the registration page. On this page, students will see their schedule details, selected courses, possible schedules and their preferences.

### 16.2.7 Preferences in the Scheduler

Preferences are used for telling the scheduler that a time interval is prohibited from containing a class. This feature will mostly likely be used when a student desires a day off or during a time that the student might have work or extracurricular activity. The students will be able to add a restricting time by selecting a weekday that will prevent the scheduler to generate schedules that includes the selection. These preferences can be applied on a part of the day or on the whole day.

### 16.2.8 How to add Preferences to the Scheduler

To add a time preference.

1. Click the + symbol on the preferences bar on the far right to have the modal appear as

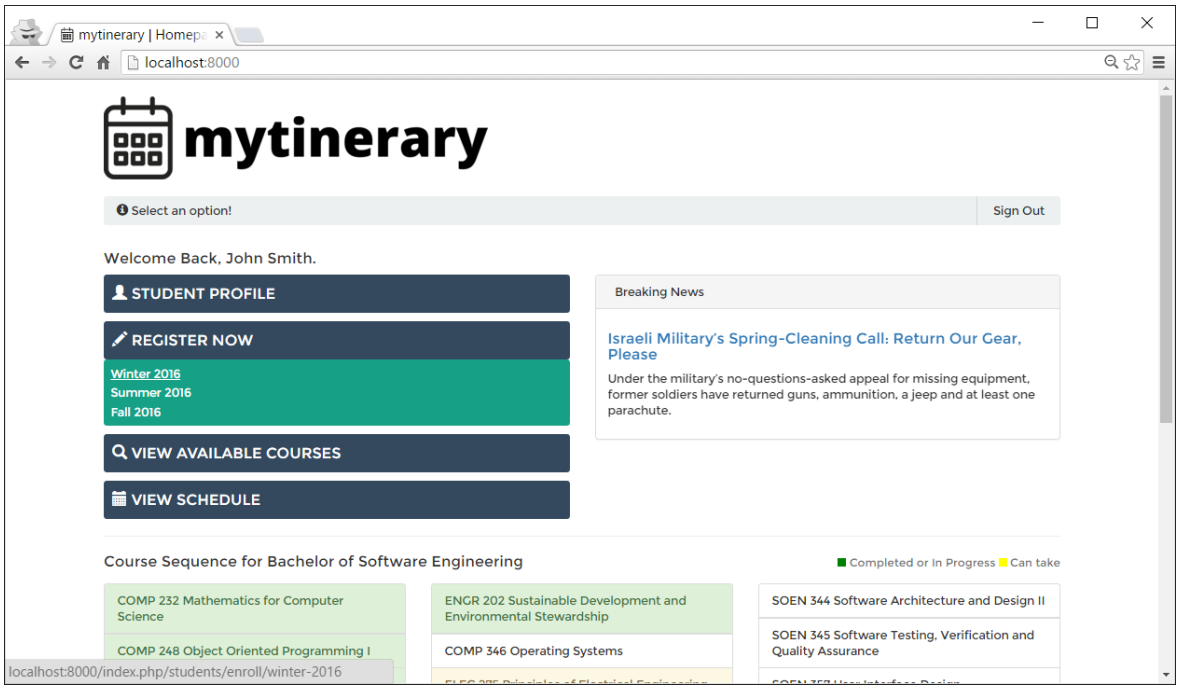



Figure 27: Main Menu

shown in Figure 28.

2. Select a weekday.
3. (Optional) Check 'restrict the whole day' if you would not like to have a class in that day.
4. If 'restrict the whole day' is checked then simply click apply and skip to step 6
5. Select the time interval in which a time is preferred to prevent a class from be added.
   - The ending time must be later than the starting time.
   - The intervals must be within a day.
6. Click save and the form will disappear and the time interval will be added to the list.

### 16.2.9 How to Remove a Time Preference From the Scheduler

1. If the time preference bar is collapsed, simply click on the bar to reveal the list of restricted times.
2. Click on the time preferences you want to remove.
   - **Note:** Clicking on one, will remove the preference immediately.

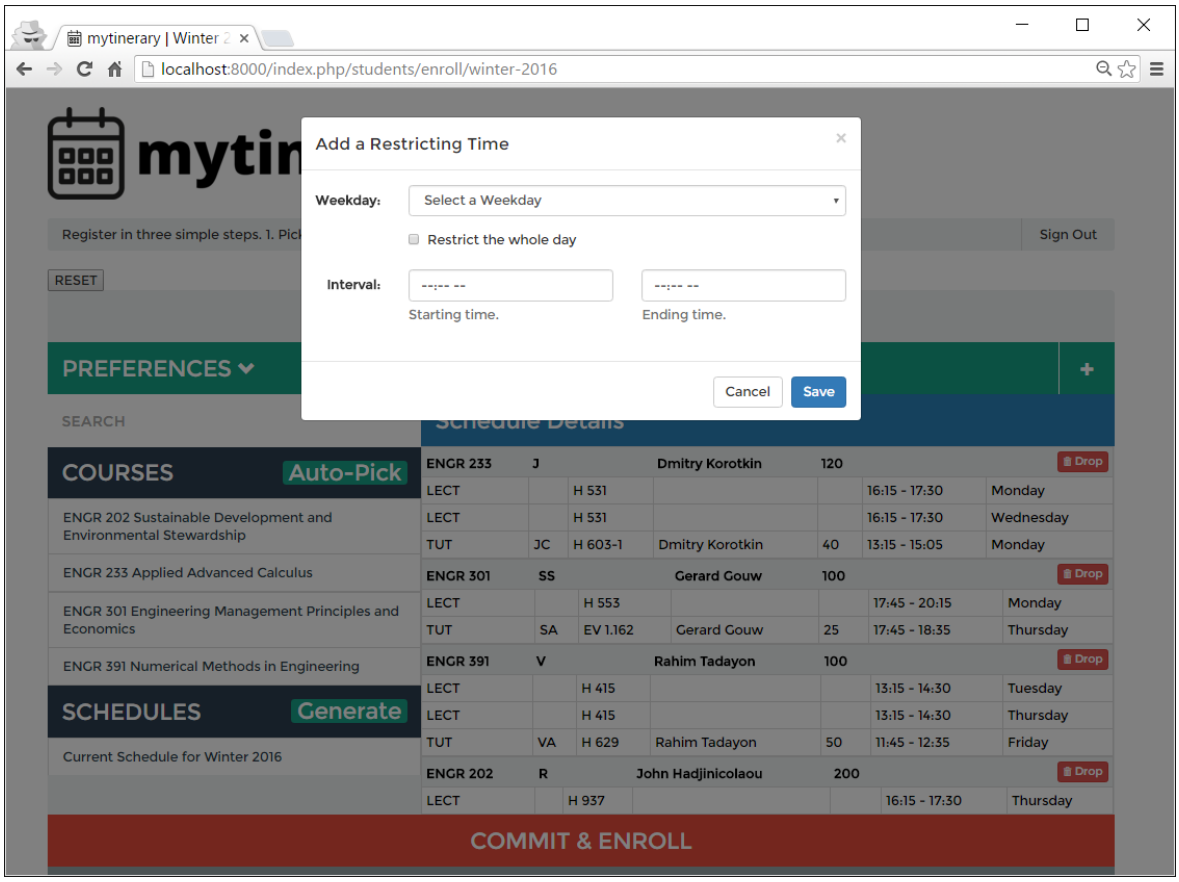



Figure 28: Preference Modal

### 16.2.10 Adding Courses to the Generator with Searching and Auto-Pick

A search box is presented below the preferences section as shown in Figure 6 which allows the user to search for any course and add it the generator list. The generator list is a list of courses that a student wishes to enroll in. The schedule generator will generate schedules from that course list. Note that the search box can accept general inputs, for instance entering SOEN will return all SOEN courses. Below the search box is a list of courses for which the student has selected. The auto-pick option allows the system to automatically select courses for the student by prioritizing courses within the course sequence. Note:

1. Courses that do not meet prerequisites can not be added to the course list.
2. Corequisite courses must be enrolled before actually adding the course that requires it.
3. A limit of 5000 possible schedules limits the generator from generating too many schedules. Shows 'Exceeds the max number of calculations' message.
4. If the course is already in the list, it is impossible to add it to the schedule.

5. The number next to the course is the number of possible combinational sections available for that course in that semester.

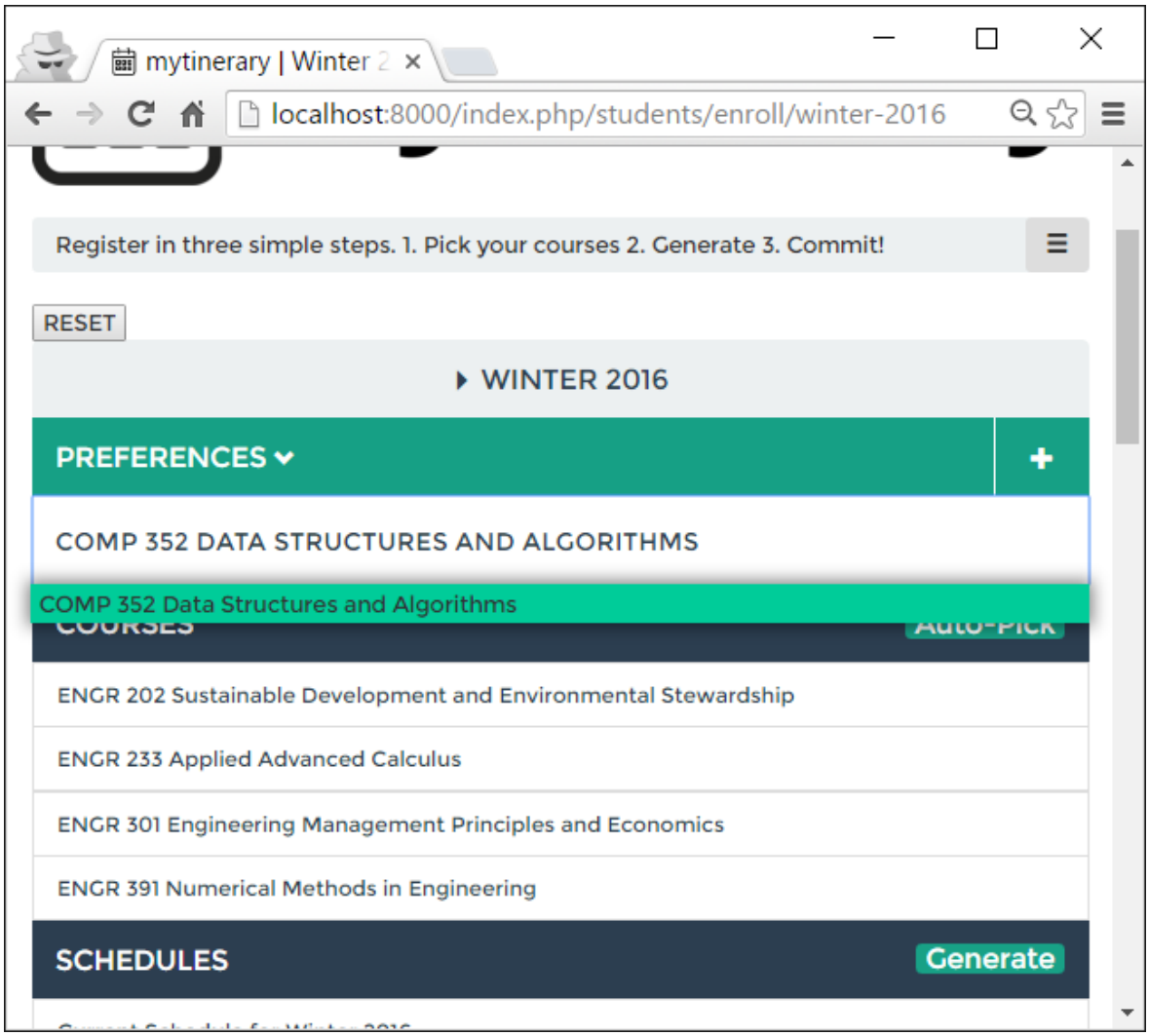



Figure 29: Search Box

### 16.2.11 How to Add a Course to the Generator with Search

1. As shown in Figure 29: Type in the course name.
    - If the course does not appear. The course does not exist or is not available that semester.
2. Select the course so that it turns green and it will be added to the list as shown in Figure 30.

- The courses that are in the generator list are highlighted yellow.

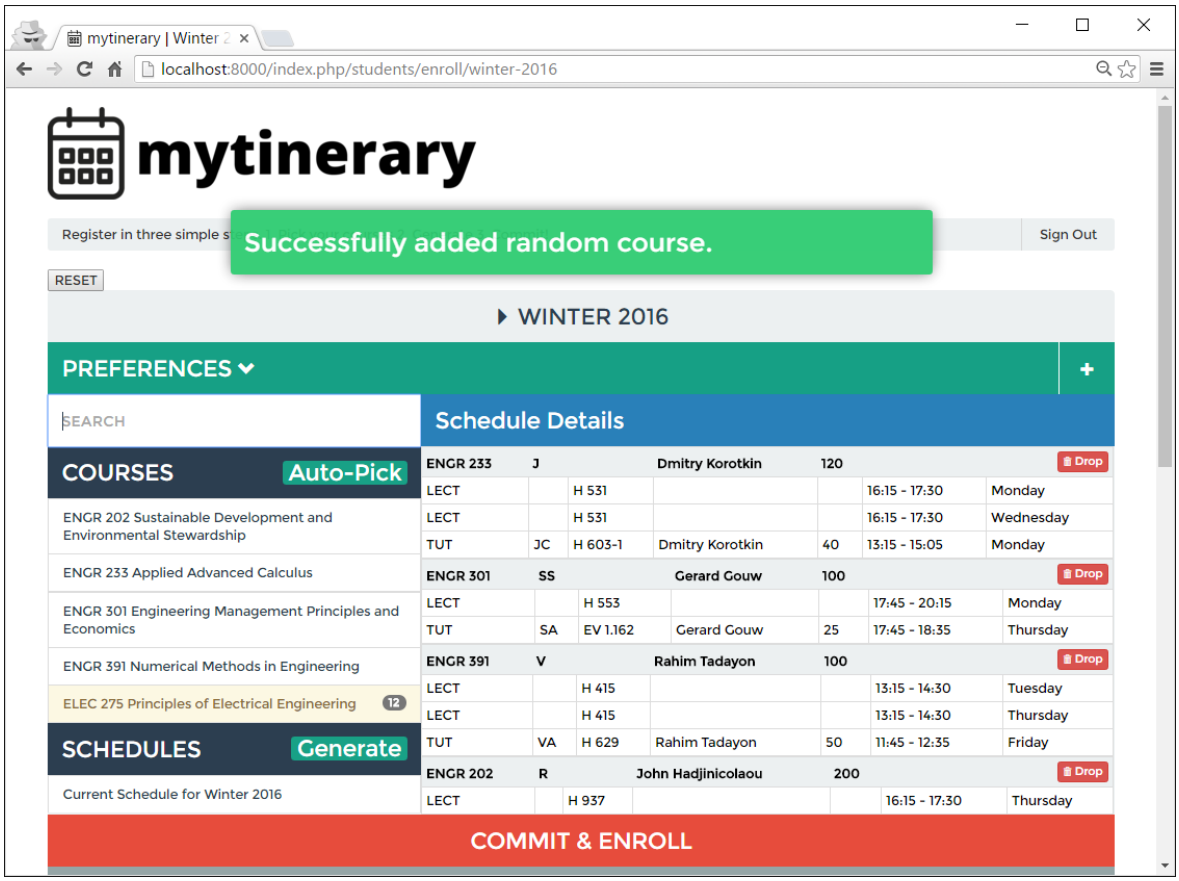



Figure 30: Addition Of Course

### 16.2.12 How to Add a Course to the Generator with Auto-pick

1. Click the Auto-Pick Button
   - One course will be automatically added to the course list.
   - The course added corresponds to the course on the course sequence list.
2. The course will appear in the course list highlighted in yellow as shown in Figure 30.

### 16.2.13 How to Remove the Course from the Generator List

1. Click on the course you would like to remove as shown in Figure 31. It will turn red.

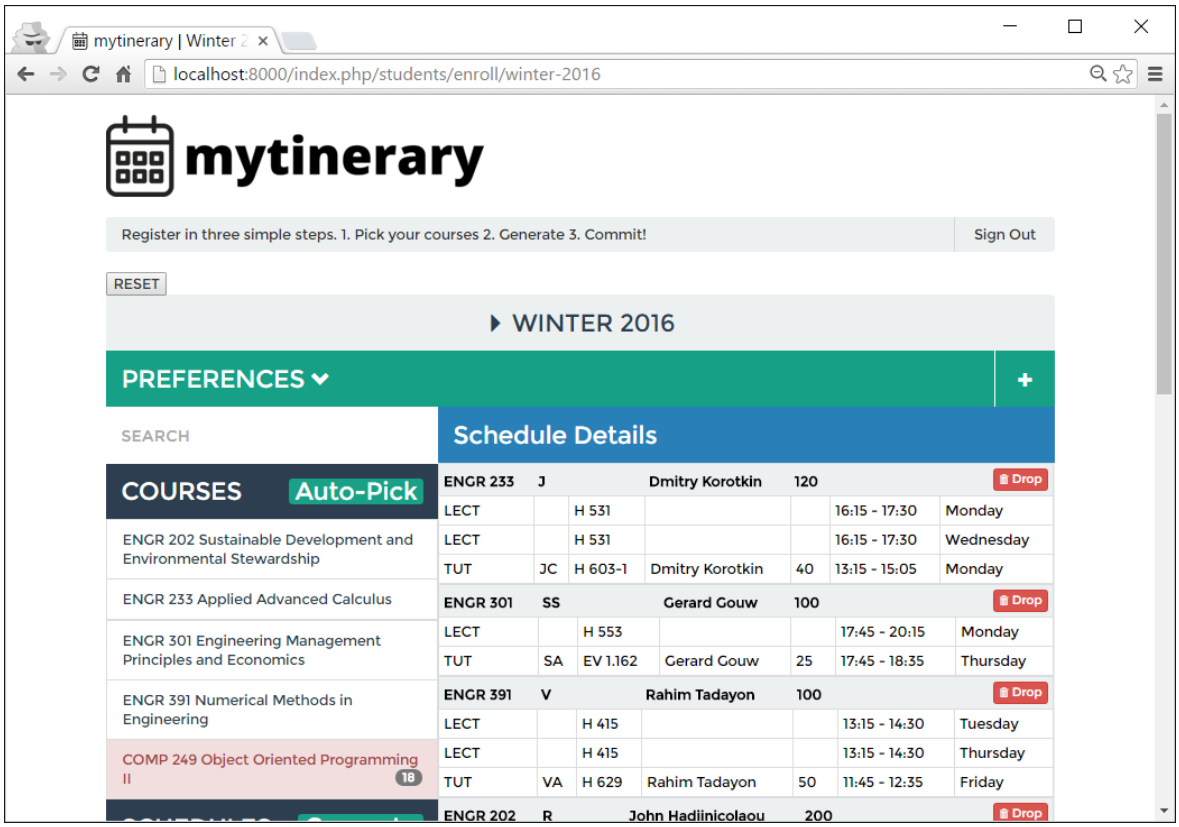



Figure 31: Removal of Course

### 16.2.14 Schedule Generation

Clicking the generate button will generate a set of all possible schedules. It uses the course list which was previously added. This feature is limited to generating up to 5000 schedules per request. This is because one course may have many section combinations. The number of possible schedules increases very quickly when we consider several courses. Once the generator has generated all the possible schedules, taking into account the current schedule and the preferences, as shown in Figure 32, the student can view all of the generated schedules and commit the preferred one by selecting it.

### 16.2.15 How to Generate Schedules

1. Before clicking the schedules, a list of courses must be added to the generator.
    - Refer to the previous section on how to add courses to the generator.
2. Click generate.
    - A list of schedules will be generated.

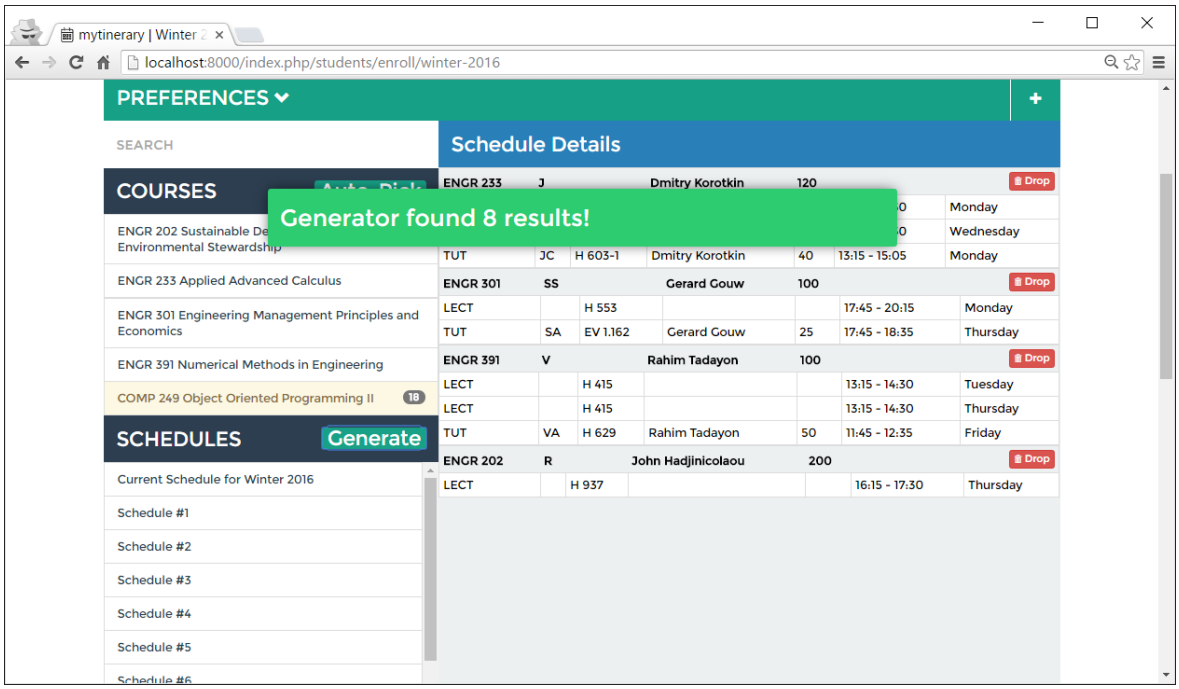



Figure 32: Schedule Generation

- If zero schedules were shown. Try adding one course at a time, and click generate. There is a possibility that a course is not compatible with the current schedule.
- Tip: To navigate through the list of schedules quickly use the left and right keys.

### 16.2.16 Enrolling a Schedule

1. Generate a list of schedules.
2. Click on the schedule preferred.
3. Click commit and confirm.
    - Note: the courses highlighted in yellow correlate to courses that the student will be enrolled into.
    - Once complete, everything should turn green.

The schedule Details contains a list of all courses that the student is currently registered in and courses that students have added to their tentative schedules. If a course is highlighted yellow, then it is temporarily added to the schedule. The courses will also display the lecturer, room location, class hours, as well as the class capacity of each course. The courses in yellow will only be added to the schedule once the user clicks on Commit & Enroll. Once the courses have been enrolled, the student will have the option to drop the course.

### 16.2.17 Schedule Timetable

Under the Commit & Enroll button, there is a schedule timetable that shows the courses. If the course is in a yellow box, then it indicates that the student is temporarily adding that course. If the course is in a green box then it is already enrolled for the student. Also, there is a reset button which would reset the user interface if there are glitches.

The schedule timetable has two versions. The main schedule and the generated schedules. The main schedule is always the first schedule to appear when accessing the registration page. Then there are the generated schedules that appear after the main schedule. The user can click to select the schedule to enroll in as shown in Figure 33. In that figure, the student has selected the 4th schedule. The sections in yellow will be enrolled in if the user presses the 'Commit & Enroll' button.

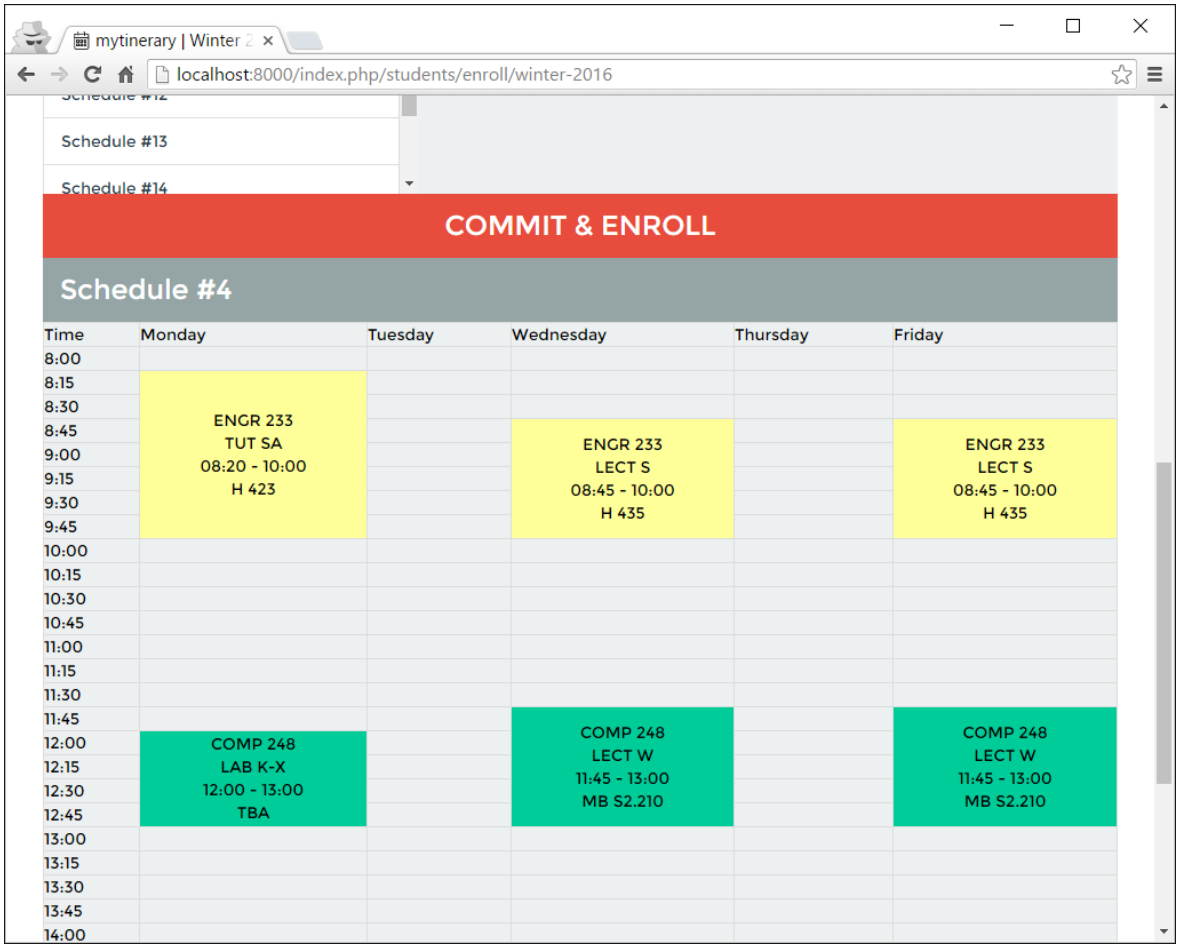



Figure 33: Schedule Timetable

### 16.2.18 Dropping a Course

Dropping a course is done in the schedule details section. On every course section there is a drop button that will drop that course if clicked. Note: Dropping a course can only be done within the current schedule after first committing and enrolling a schedule. It is also possible to undo dropping a class or several classes in case the user has changed their mind. The user may re-enroll in the courses by clicking the undo button. Note: If the user decides to add a course to the generate list. The 'UNDO' button will disappear to prevent the user from adding the course more than once.

### 16.2.19 How to Drop a Course Section

1. Look for the drop button located on the right side of the section course as shown in Figure 34.

2. Click 'Drop'.


Figure 34: Schedule Details Area

### 16.2.20 How to Undo a Drop

1. To undo a drop. Click on the 'UNDO' button as shown in Figure 35.

- The drop button will only be shown on that windows instance.
- Visiting other pages will remove this option.
- Adding a course to the generator list will remove the 'UNDO' button

2. If successful the course section will be re-enterred on the main schedule.

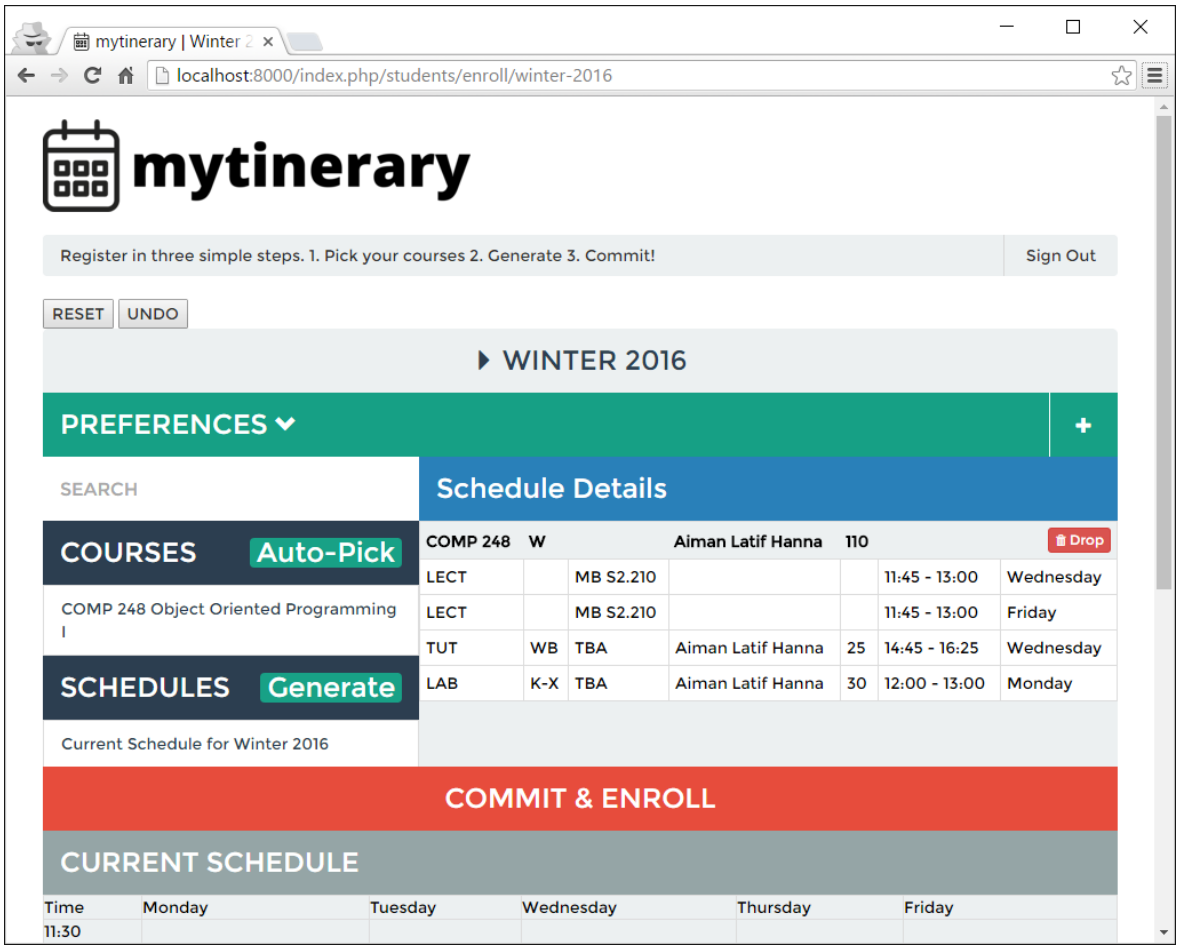



Figure 35: Undo Button

### 16.2.21 View Available Courses

The View Available Courses page would allow the user to verify if a certain course is available for a particular semester as shown in Figure 36. Once the user clicks on that option, they will be able to select a semester, enter the course's subject and the course number. Upon clicking the search button all available sections and course details such as room location and class capacity will be displayed to the user. In the event that a course is not available in the semester or does not exist, an error message will appear: 'No results found'.

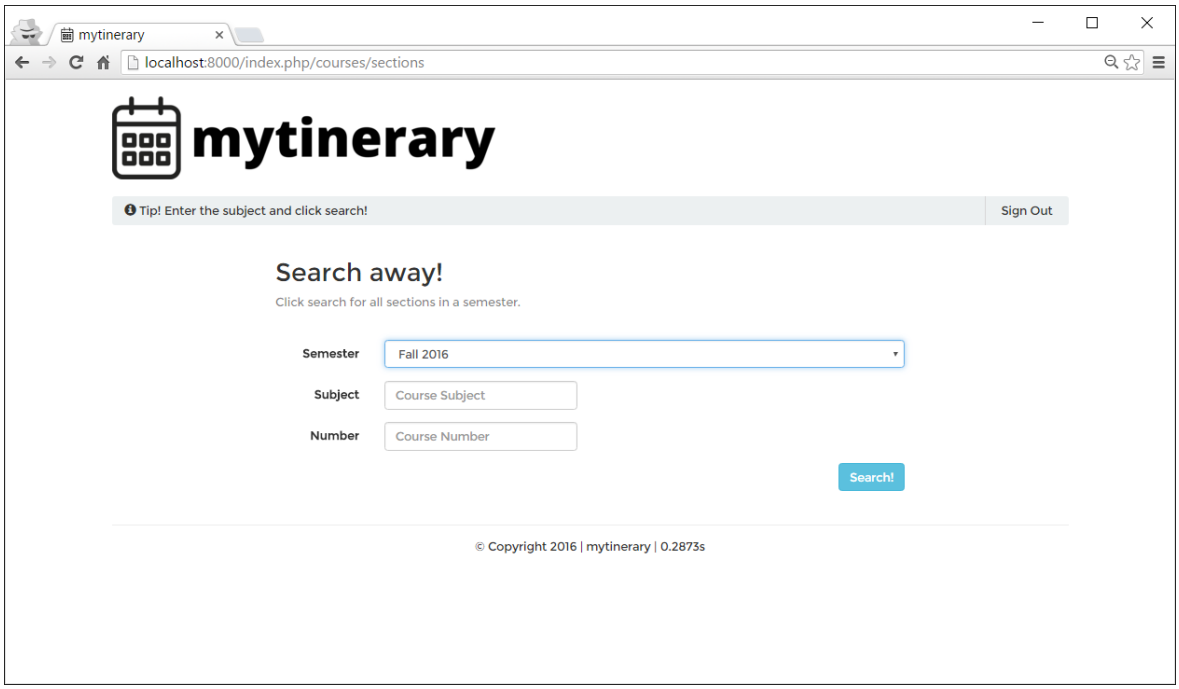



Figure 36: Search Page

### 16.2.22 How to Search for Course Details

1. From the home page. Click on the 'View Courses' box.
2. Select the semester.
    - Click search to view all courses in that semester.
3. Enter the subject and number of the desired course.
4. Click search.
    - An error message 'No results found' may appear if not results were found in that semester.

### 16.2.23 Viewing the Final Schedule

When clicking on the view schedule option in the main menu, a list of available semesters will be displayed. Students may click on any of the given semesters to view their schedule for the selected semester in a timetable format. Note that for this option, the system does not consider any of the temporary classes added by the student.

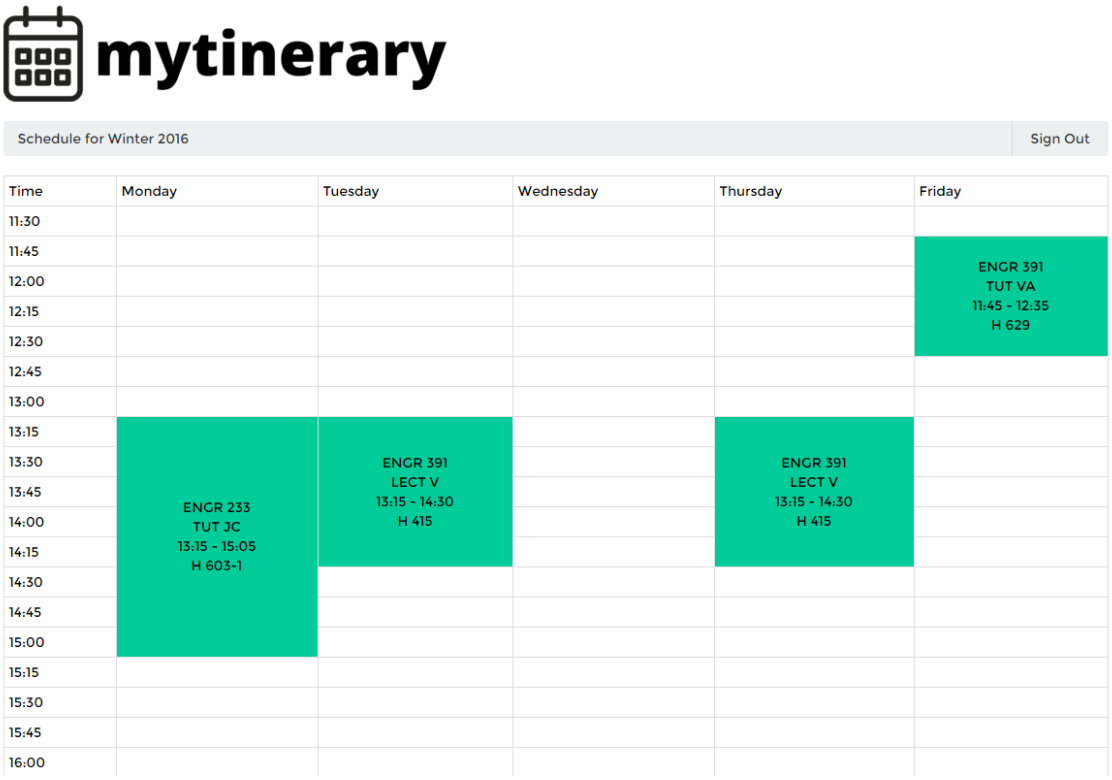



Figure 37: View Schedule for Winter 2016

### 16.2.24 How to View the Final Schedule

1. Sign in the application.
2. Click on the 'View Schedule' box.
3. Select an available semester and view the schedule.
    - If no schedule is available, a message will be displayed, saying that there is nothing to display.
    - Warning: If the course is online. It will not appear. Please refer to the registration page to view all the details.

# 17 Final Cost Estimate

The table below details the final cost of all components of the project measured in man hours. Ultimately there were no financial costs for the project as all tools and resources used were available as open source.

| Component | Man Hours |
|---|---|
| Initial domain model | 5 |
| Proposing roles | 1 |
| Completion of deliverable 0 | 2 |
| **Total:** | **8** |

| Component | Man Hours |
|---|---|
| Listing of functional requirements | 3 |
| Use cases and use case diagram | 3 |
| Update of domain model | 1 |
| Listing of non-functional requirements | 2 |
| Evaluation of available and needed resources | 2 |
| Project Scoping | 1 |
| High-level system architecture | 3 |
| Listing of technologies used | 2 |
| Create timetable and specify artifacts | 1 |
| Assign activities | 0.5 |
| Create a Gantt chart | 1 |
| Identify and assess risks | 2 |
| **Total:** | **22.5** |

| Component | Man Hours |
|---|---|
| 4+1 diagram | 7 |
| Subsystem interface specification | 2.5 |
| Detailed unit description | 4 |
| Dynamic design scenarios | 4 |
| Estimation | 2 |
| **Total:** | **19.5** |

| Component | Man Hours |
|---|---|
| Unit testing | 5 |
| Requirements testing | 3 |
| Stress testing | 2 |
| Security testing | 2 |
| Installation guide | 2 |
| User manual | 2 |
| Cost revaluation | 1 |
| **Total:** | **17** |

| Component | Man Hours |
|---|---|
| Login Page | 3 |
| Home Page | 8 |
| Profile Page | 2 |
| Register Now Page | 127 |
| View Schedule Page | 2 |
| View Courses Page | 5 |
| **Total:** | **139** |