**Concordia University**

**Department of Computer Science**

**and Software Engineering**

**Software Process**

**SOEN 341/4 --- Winter 2016 --- Section S**

**Deliverable 4**

**April 13th, 2016**

**Project Name: Apollo**

**Team Name: Athena**

|  |  |
| --- | --- |
| **Team Member Information** | |
| **Name** | **SID** |
| Philippe Abou Kasm | 27305133 |
| Wahab Ahmed | 21311980 |
| Sabrina Ashraff | 29533400 |
| Francis Bouchard | 26786812 |
| Clozzy-Mailey Chavez | 26610315 |
| Ricardo Cortés | 27734107 |
| Liuai Hatter | 25976618 |
| Jian Huang | 26772862 |
| Anna Rogozin | 27494939 |
| Matthew Teolis | 40005332 |

# Table of Contents

### 

|  |  |
| --- | --- |
| **1. Introduction ………………………………………………………………………...** | **4** |
| **2. Goals and Constraints ……………………………………………………………..** | **5** |
| 2.1 . Functional Requirements …………………………………………………… | 5 |
| 2.2 . Domain Model ……………………………………………………………….. | 23 |
| 2.3. Constraints and Qualities …………………………………………………… | 23 |
| **3. Resource Evaluation ……………………………………………………………….** | **26** |
| 3.1. Human Resources ……………………………………………………………. | 26 |
| 3.2. Technical Resources ………………………………………………………... | 30 |
| **4. Scoping ………………………………………………………………………………** | **31** |
| **5. Solution Sketch ……………………………………………………………………..** | **32** |
| 5.1. Architecture …………………………………………………………………... | 32 |
| 5.2. Technologies in Use ………………………………………………………..... | 33 |
| **6. Plan …………………………………………………………………………………..** | **36** |
| 6.1. Activities ……………………………………………………………………... | 36 |
| 6.2. Artifacts ………………………………………………………………………. | 37 |
| 6.3. Project Estimates ……………………………………………………………. | 43 |
| 6.4. Project Re-estimation Scenarios ………………………………………….... | 45 |
| 6.5. Activities Assignments …………………………………………………….... | 46 |
| 6.6. Schedule ……………………………………………………………………..... | 47 |
| 6.7. Risks …………………………………………………………………………… | 48 |
| **7. Prototyping ………………………………………………………………………….** | **49** |
| **8. Architectural Design ……………………………………………………………...** | **54** |
| 8.1. Architecture Diagram ………………………………………………………. | 54 |
| 8.2. Subsystem Interfaces Specifications …………………………………….. | 69 |
| **9. Detailed Design …………………………………………………………………...** | **74** |
| 9.1. Detailed Design Diagram …………………………………………………... | 74 |
| 9.2. Unit Descriptions ……………………………………………………………. | 75 |
| **10. Dynamic Design Scenarios ………………………………………………………** | **81** |
| **11. Estimation ………………………………………………………………………….** | **88** |
| **12. Risks ………………………………………………………………………………..** | **90** |
| **13. Rapid Prototyping ………………………………………………………………...** | **91** |
| **14. Testing Report …………………………………………………………………….** | **102** |
| 14.1. Test Coverage ……………………………………………………………….. | 102 |
| 14.1.1. Tested Items ………………………………………………………….. | 102 |
| 14.1.2. Untested Items of Interest …………………………………………... | 107 |
| 14.2. Test Cases …………………………………………………………………... | 109 |
| 14.2.1. Unit Testing ……………………………………………………………. | 109 |
| 14.2.2. Requirements Testing ………………………………………………... | 111 |
| 14.2.3. Stress Testing ……………………………………………………….... | 122 |
| 14.2.4. Security Testing ………………………………………………………. | 123 |
| **15. System Delivery …………………………………………………………………..** | **125** |
| 15.1. Installation Manual …………………………………………………………. | 125 |
| 15.2. User’s Manual ……………………………………………………………….. | 126 |
| **16. Final Cost Estimate ……………………………………………………………….** | **132** |
| **References ……………………………………………………………………………..** | **134** |

### 

### 

# 1**.** INTRODUCTION

The purpose of this system is for a student enrolled in the Software Engineering program to plan their schedule for the entire duration of the program.

The application will have the following class of users:

* Students
* Professor
* Faculty Administrator
* System Administrator

With this application, the student will also be able to define their program option in Software Engineering from amongst the following four options:

* General
* Computer Games
* Web Applications
* Real-Time and Embedded Systems

Once the student has defined their program option, a master course list will be provided with all the courses the student is required to take. From there, a student will be able to select the courses they wish to take each semester for the duration of the program.

Several settings for preferences will be featured, such as:

* Creating the course schedule based on the course sequence provided by the department for each option
* Creating the course schedule without depending on the suggested sequence
* Selecting time preferences for courses, either morning, afternoon, or night

Once all these preferences are selected by the student, the Apollo application will evaluate the student record based on the following criteria:

* The overall academic record of the student
* Checking if each course’s prerequisites have been met
* Checking if the credit requirements are present
* Courses for which the student has an exemption
* Assuring if the courses are taught by an accredited engineer

Once all these checks have been made, Apollo will generate the number of possible schedules that the student can choose from, and the student can then select the schedule they prefer.

# 2**. GOALS AND CONSTRAINTS**

In this section, all the features that the Apollo application will implement will be discussed, as well as the goals which will be achieved. This will be explained by presenting the functional requirements of the application in terms of the users which will be expected to use it, and use case models to explain each feature. A domain model diagram is provided to give an overall view of how the various entities within the software will interact. Finally, constraints which will be anticipated in the building of this application are discussed.

## **2.1. Functional Requirements**

1. User

1.1. All users can log in

1.2. All users can log out

1.3. All users can update their personal information

1. System Administrator

2.1. System administrator can create new users

2.1.1. System administrator can grant new users specific permission

2.2. System administrator can view existing users

2.3. System administrator can update existing users’ information

2.3.1. System administrator can change users’ permissions

2.4. System administrator can remove existing users from the system.

1. Faculty Administrator

3.1. Faculty administrator can create new courses

3.1.1. Faculty administrator can assign requisites and requirements to courses

3.2. Faculty administrator can view existing courses

3.3. Faculty administrator can update existing courses’ information

3.3.1. Faculty administrator can change course requisites and requirements

3.4. Faculty administrator can remove existing courses from the system

1. Professor

4.1. Professor can view his or her course schedule

4.2. Professor can update students’ grades

1. Student

5.1. Student can view his or her current schedule

5.2. Student can review his or her course history

5.3. Student can search for courses

5.3.1. Student can view a course’s description

5.4. Student can generate schedules

5.4.1. Student can choose course preferences to generate schedules

5.4.2. Student can save generated schedules

5.4.3. Student can view saved schedules

5.4.4. Student can finalize a schedule

5.5. Student can print a schedule

### 2.1.2. Use Case Descriptions

|  |  |
| --- | --- |
| **UC01 - Login** | |
| Risk Assessment | 5/5 |
| Importance | 5/5 |
| Actor(s) | Student, Professor, Faculty Admin, System Admin |
| Description | Validate user credentials and allow user specific privileges to the system. |
| Basic Flow | 1. The User open the login webpage. 2. The User provide the required credentials. 3. The System validate the credentials. 4. Once validated, the System display the main webpage, along with authorized options. |
| Pre-condition | User is not logged in. |
| Post-condition | User is logged in. |

|  |  |
| --- | --- |
| **UC02 - Logout** | |
| Risk Assessment | 5/5 |
| Importance | 3/5 |
| Actors | Student, Professor, Faculty Admin, System Admin |
| Description | Terminate the session of the user interacting with the system, no longer provide the user his or her personal data. |
| Basic Flow | 1. The User selects the logs out option. 2. The System delete the User’s session. 3. Once deleted, the System displays the login webpage. |
| Pre-condition | User is logged in. |
| Post-condition | User is logged out. |

|  |  |
| --- | --- |
| **UC03 - Update Personal Information** | |
| Risk Assessment | 5/5 |
| Importance | 5/5 |
| Actors | Student, Professor, Faculty Admin, System Admin |
| Description | Update user profile information. |
| Basic Flow | 1. The User selects the “Personal Information” option. 2. The System displays the “Personal Information” webpage. 3. The User performs the desired changes to the personal information. 4. The User saves the newly entered information. 5. The System perform validation routines. 6. Once validated, the System save the the new information. |
| Pre-condition | User have personal information. |
| Post-condition | User’s information is updated. |

|  |  |
| --- | --- |
| **UC04 - Review Course History** | |
| Risk Assessment | 2/5 |
| Importance | 4/5 |
| Actors | Student |
| Description | Review student’s course history which includes the semesters taken and student’s grades. |
| Basic Flow | 1. The Student selects the “Course History” option. 2. The System displays the “Course History” webpage. 3. Once displayed, the Student have access to his course history |
| Pre-condition | Student have course history. |
| Post-condition | Student sees course history. |

|  |  |
| --- | --- |
| **UC05 - View Degree Audit** | |
| Risk Assessment | 2/5 |
| Importance | 2/5 |
| Actors | Student |
| Description | Review list of completed courses and remaining courses to be taken in order to graduate. |
| Basic Flow | 1. The Student selects the “Degree Audit” option. 2. The System displays the “Degree Audit” webpage. 3. Once displayed, the Student have access to his individual program information. |
| Pre-condition | Student have a program and course information. |
| Post-condition | Student sees his program advances. |

|  |  |
| --- | --- |
| **UC06 - Choose Course Preferences** | |
| Risk Assessment | 2/5 |
| Importance | 3/5 |
| Actors | Student |
| Description | Edit preferences, add constraints such as class time preference and semester preferences. |
| Basic Flow | 1. The Student selects the “Course Preferences” option. 2. The System displays the “Course Preferences” webpage. 3. The Student performs the desired changes to the selected courses. 4. The Student saves the newly entered information. 5. The System perform validation routines on modified courses. 6. Once validated, the System save the updated information. |
| Pre-condition | Student has been enrolled. |
| Post-condition | Student’s course information is updated. |

|  |  |
| --- | --- |
| **UC07 - Search Courses** | |
| Risk Assessment | 1/5 |
| Importance | 5/5 |
| Actors | Student, Faculty Admin |
| Description | Search for a specific course by course name or by department. |
| Basic Flow | 1. The User selects the “Search Course” option. 2. The System displays the “Search Course” webpage. 3. The User enters information into the search field. 4. The User requests to view courses matching query. 5. The System display the courses found that match with the search criteria. |
| Pre-condition | User is logged and have appropriate privileges. |
| Post-condition | User sees list of course matching his criteria. |

|  |  |
| --- | --- |
| **UC08 - View Course Description** | |
| Risk Assessment | 1/5 |
| Importance | 4/5 |
| Actors | Student, Faculty Admin |
| Description | See course instructor, sections, session time, requisites. |
| Basic Flow | 1. On any course, the User selects the “View Course” option. 2. The System displays the “View Course” webpage for the selected course. 3. The User have access to detailed information about the course. |
| Pre-condition | User is logged in and course is existent. |
| Post-condition | User sees the course information. |

|  |  |
| --- | --- |
| **UC09 - Generate Schedules** | |
| Risk Assessment | 1/5 |
| Importance | 5/5 |
| Actors | Student |
| Description | User can view the different semesterly schedules offered to him or her. |
| Basic Flow | 1. The Student selects the “Generate Schedule” option. 2. The System displays the “Generate Schedule” webpage. 3. The Student chooses the term and course preferences. 4. The Student requests to generate schedules. 5. The System validate course requisites and program. 6. Once validated, the system displays the generated schedules. 7. The User review and save selected schedules. |
| Pre-condition | Student is logged in and has chosen the time frame for which to display schedules and entered his/her preferences. |
| Post-condition | One or more schedules are generated according to Student’s specifications. |

|  |  |
| --- | --- |
| UC10 - Generate Sequences | |
| Risk Assessment | 1/5 |
| Importance | 5/5 |
| Actors | Student |
| Description | User can view the different course sequences offered for their remaining semesters until graduation. |
| Basic Flow | 1. The Student selects the “Generate Sequences” option. 2. The System displays the “Generate Sequences” webpage. 3. The Student chooses course preferences. 4. The Student requests to generate course sequences. 5. The system displays the generated sequences. |
| Pre-condition | Student is logged in and has entered their preferences. |
| Post-condition | One or more sequences are generated according to Student’s specifications. |

|  |  |
| --- | --- |
| **UC11 - Save schedule** | |
| Risk Assessment | 1/5 |
| Importance | 4/5 |
| Actors | Student |
| Description | User can save generated schedule. |
| Basic Flow | 1. User chooses the schedule he wants to save 2. The system asks for confirmation 3. The student confirms 4. The schedule is saved |
| Pre-condition | User is logged in and sees at least one generated schedule. |
| Post-condition | Schedule is saved to user profile. |

|  |  |
| --- | --- |
| **UC12 - View Saved Schedules** | |
| Risk Assessment | 1/5 |
| Importance | 3/5 |
| Actors | Student |
| Description | Student views previously saved schedules or sequences that were generated from scheduler. |
| Basic Flow | 1. Student runs scheduler 2. Student saves schedule(s) 3. Student requests to view their saved schedules 4. System looks up for the saved schedules 5. System displays student’s saved schedules |
| Pre-condition | Student is logged in. |
| Post-condition | Student sees saved schedules. |

|  |  |
| --- | --- |
| **UC13 - View Student Schedule** | |
| Risk Assessment | 1/5 |
| Importance | 5/5 |
| Actors | Student |
| Description | Student sees their current semester schedule. |
| Basic Flow | 1. Student chooses to view schedule 2. System displays current schedule |
| Pre-condition | Student is logged in. |
| Post-condition | Student sees current schedule. |

|  |  |
| --- | --- |
| **UC14 - Print Schedule** | |
| Risk Assessment | 1/5 |
| Importance | 1/5 |
| Actors | Student |
| Description | User prints their current schedule, generated schedule, or saved schedule. |
| Basic Flow | 1. User chooses a schedule 2. User chooses a print option 3. User gets a confirmation about print option 4. Chosen schedule is printed |
| Pre-condition | User is logged in and has finalized a schedule. |
| Post-condition | User’s schedule is printed. |

|  |  |
| --- | --- |
| **UC15 - Create Course** | |
| Risk Assessment | 3/5 |
| Importance | 5/5 |
| Actors | Faculty Admin |
| Description | Faculty Admin creates a new course. |
| Basic Flow | 1. User navigates to create course page 2. User enters all course information 3. System asks for confirmation 4. User saves new course |
| Pre-condition | Faculty Admin is logged in and has permission to add course. |
| Post-condition | Faculty Admin creates a course in the system. |

|  |  |
| --- | --- |
| **UC16 - Remove Course** | |
| Risk Assessment | 5/5 |
| Importance | 4/5 |
| Actors | Faculty Admin |
| Description | Faculty Admin removes a course. |
| Basic Flow | 1. User searches a course 2. User finds course to remove 3. System asks for confirmation 4. User removes the course |
| Pre-condition | Faculty Admin is logged in and has permission to remove course. |
| Post-condition | Course is removed. |

|  |  |
| --- | --- |
| **UC17 - Update Course** | |
| Risk Assessment | 2/5 |
| Importance | 5/5 |
| Actors | Faculty Admin |
| Description | User edits the description of a course. |
| Basic Flow | 1. User enters information for a course 2. The system looks up for the course and displays 3. User updates information about the course 4. The system confirms |
| Pre-condition | User is logged in. |
| Post-condition | The specified course’s information is updated. |

|  |  |
| --- | --- |
| **UC18 - Create User** | |
| Risk Assessment | 4/5 |
| Importance | 5/5 |
| Actors | System Admin |
| Description | System Admin creates a user of the system. |
| Basic Flow | 1. System Admin enters information for a new user (with specific permissions) 2. The system asks for confirmation 3. System creates the new user |
| Pre-condition | System Admin is logged in. |
| Post-condition | A new user is created. |

|  |  |
| --- | --- |
| UC19 - Remove User | |
| Risk Assessment | 4/5 |
| Importance | 5/5 |
| Actors | System Admin |
| Description | System Admin removes a user from the system. |
| Basic Flow | 1. System Admin selects a user to remove 2. System finds the user and displays 3. System asks for confirmation 4. System removes selected user |
| Pre-condition | System Admin is logged in. |
| Post-condition | The user is removed from the system. |

|  |  |
| --- | --- |
| **UC20 - Update User** | |
| Risk Assessment | 4/5 |
| Importance | 5/5 |
| Actors | System Admin |
| Description | System Admin updates a user’s information. |
| Basic Flow | 1. User puts in information about the user 2. The system looks up and shows the user information 3. System Admin modifies information of a selected user 4. The system asks for confirmation 5. System updates the user’s information |
| Pre-condition | System Admin is logged in. |
| Post-condition | The user information is updated. |

|  |  |
| --- | --- |
| **UC21 - View User** | |
| Risk Assessment | 1/5 |
| Importance | 4/5 |
| Actors | System Admin |
| Description | View system user information. |
| Basic Flow | 1. System Admin logs in 2. System Admin searches for a user 3. System looks up in the user database 4. System Admin views user information |
| Pre-condition | System Admin is logged in and finds user. |
| Post-condition | System Admin sees user information. |

|  |  |
| --- | --- |
| **UC22 - Update Course Grades** | |
| Risk Assessment | 4/5 |
| Importance | 5/5 |
| Actors | Professor |
| Description | User enters the grades of all students of a class. |
| Basic Flow | 1. User searches for a course 2. The system displays that course and its grade list 3. User inputs the grades for the course 4. The system asks for confirmation 5. Course grades are updated |
| Pre-condition | User is logged in and is the professor of the selected course. |
| Post-condition | Students’ grades in the selected class are updated. |

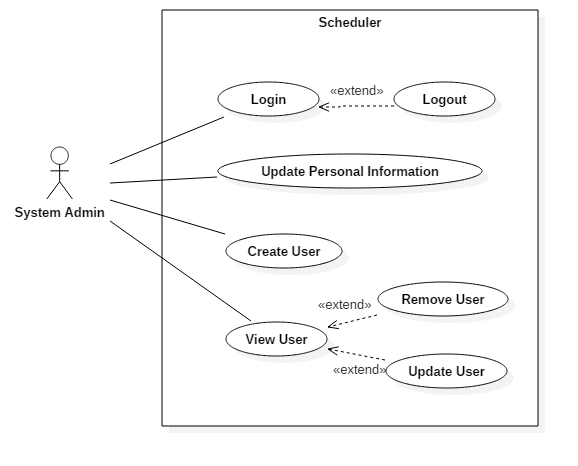
|  |  |
| --- | --- |
| **UC23 - View Professor Schedule** | |
| Risk Assessment | 1/5 |
| Importance | 3/5 |
| Actors | Professor |
| Description | Professor views the schedule of the course they are teaching. |
| Basic Flow | 1. User navigates to view schedule page 2. System displays professors schedule |
| Pre-condition | User is logged in and is a professor. |
| Post-condition | Professor sees their current schedule. |

### 

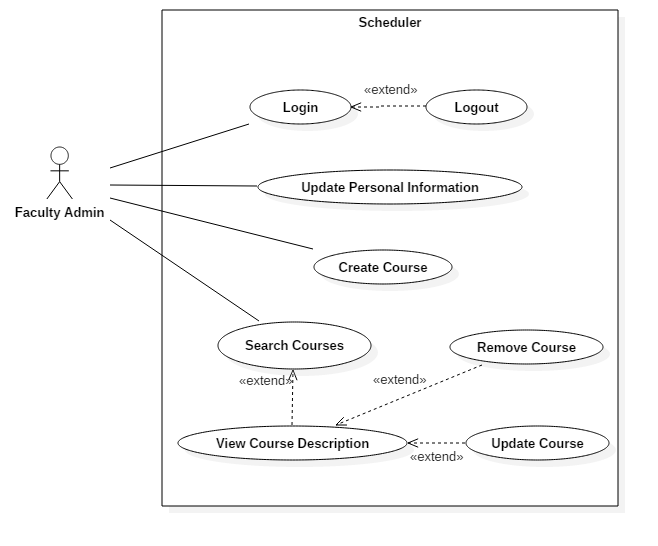
### 

### 2.1.3. Use Case-Models

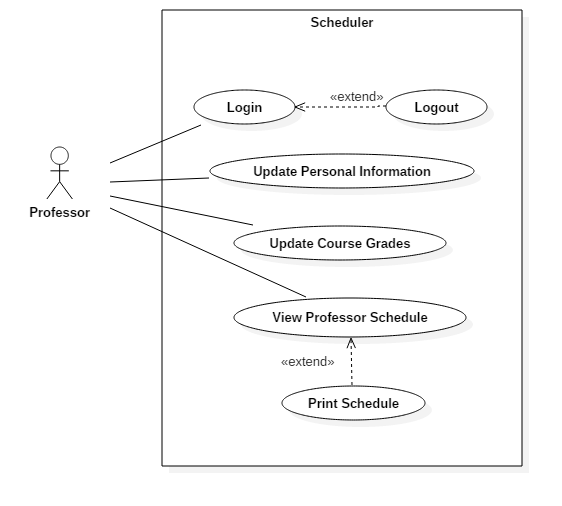
**System Administrator**



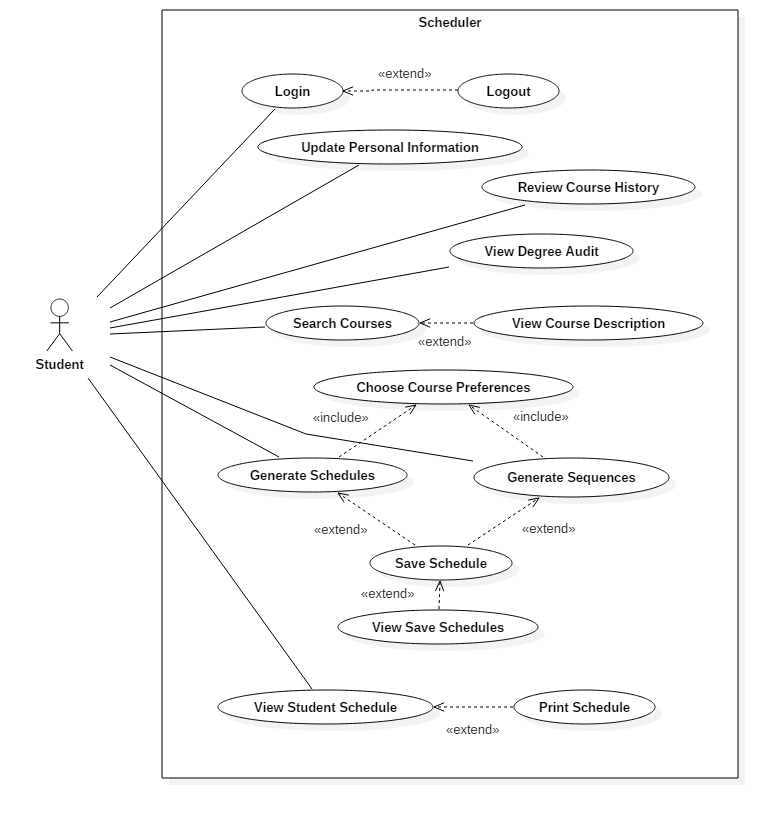
**Faculty Administrator**



**Professor**



**Student**

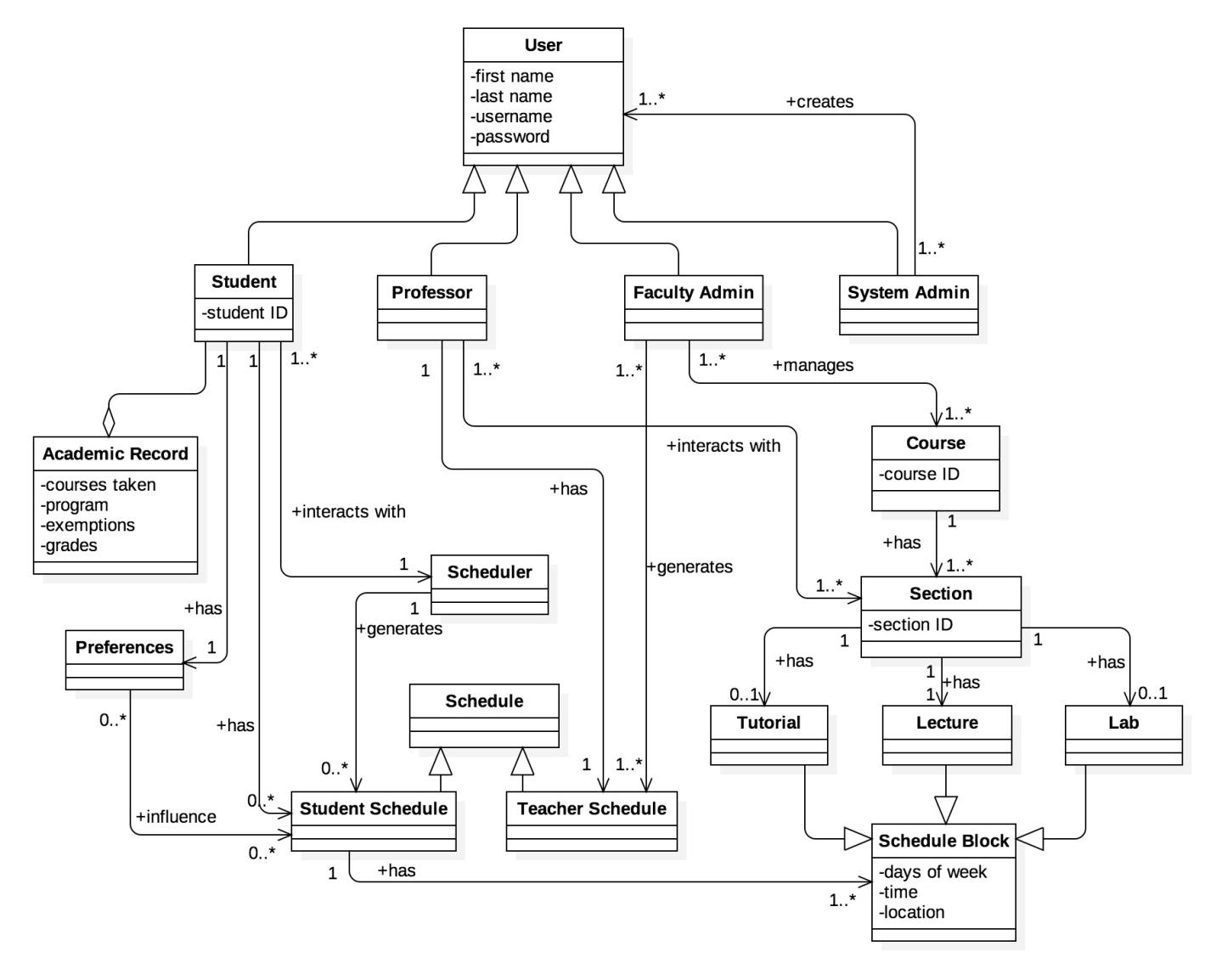


## 

## 

## **2.2. Domain Model**

Using recommended standard, the Domain Model for this project was done using UML Class Diagram notation. The concepts introduced on it were identified from the context of an educational institution, ergo they can be applied to high school, college or university environment. The main focus of this model was the relationship between the roles (i.e. Student, Professor, Faculty Admin) and the course scheduling processes. As it is required, this is a real world representation of the context, not a software modeling. Beside scheduling, the model include student’s records and resource allocation.



## 

## **2.3. Constraints and Quality**

Based on ISO/IEC 9126 [1] standard, the following quality criteria should been address in the design:

### 2.3.1. **Functionality**

The constraints described in this section address features that extend functionality to technical aspects.

1. Security
   1. The access to the system will be granted only by providing correct credentials (i.e. username and password)
   2. The password must have a length between 8 to 16 characters alphanumeric characters.
   3. The communication protocol must be encrypted (i.e. the system will be access by HTTPS)
   4. iv. The user’s password will be stored encrypted in the database, using SHA-2
   5. After 10 minutes of inactivity, the system shall automatically logout the user.
   6. The system must provide the options to manage users and permissions.
2. Interoperability
   1. The system must have the option to download the information in PDF and CSV format.
   2. The system must have the option to export the schedule to Outlook and Google calendars.
   3. The system must be have the option to be integrated (embedded) to the current University portal, if required (SSO).
   4. The system must have the option to be integrated to standard directory protocol (LDAP)
3. Normative
   1. The system’s information must be archived for a period of 4 years.
   2. The system must complain the current Quebec normative for educational web applications.

### 2.3.2. **Reliability**

* 1. Availability
     1. Since it is not mission critical, the system must have an annual uptime of 99.9% (i.e. the maximum annual downtime must be 87.6 hours)
  2. Recoverability
     1. On case of system failure, the maximum restore time must be 4 hours.
     2. On case of database failure, the maximum information lost must be 1 day.

### 2.3.3. **Usability**

* 1. Understandability
     1. The system’s web user interface should be designed so the users do not require any other assistance (manual, external applications) to operate it (user friendly)
     2. The system must be offered in English and French. I must have to option to change its language.
  2. Accessibility
     1. To grant access to senior people, the minimum system’s font size will be 14px, along with soft, no high contrast colors.
     2. To grant access to blind people, the system should have menu voice assistance and screen readers (visually impaired user's technology).

### 2.3.4. **Efficiency**

* 1. Time
     1. The system’s response time will be less than 3 seconds in all operations.
  2. Resources
     1. The system must designed to run on medium size server (4-8 Cores, 16-32 GB RAM)

### 2.3.5. **Maintainability**

* 1. Scalability
     1. The system must be able to support 5.000 users, and allow a grow up to 20.000 users.
     2. The system must be able to serve 500 concurrent users.
  2. Support
     1. To help support, technical documentation about the system use, design, installation, configuration and troubleshooting must be created, and the source code must be documented.
  3. Testability
     1. Evidence about the test cases performed during its development must be provided.

### 2.3.6. **Portability**

* 1. Adaptability
     1. The system must adapt its interface for both laptop/desktop browser and mobile devices with screens 8 inch or bigger.
     2. The system must work on the latest versions of Chrome and Firefox.
     3. The system must be build using modern web standards (HTML5, CSS3).

# 

# 

# 3**. RESOURCE EVALUATION**

In this section, the resources available in the building of the Apollo application will be provided. Human resources, in terms of each member’s capabilities and their expected contributions in the building of this software will be looked at, as well as the technical resources available to the team.

## **3.1. Human Resources**

|  |
| --- |
| Team Member: **Philippe Abou Kasm** |
| **Primary Role**: Documentation/Testing |
| **Capabilities**: I took COEN 243/244, which I have knowledge in object oriented programming in C++. I also took COEN 352, which I have knowledge in data structures and algorithms and familiarity with Java programming. Finally, I took COEN 311, with basic knowledge in assembly language programming. I have basic knowledge in HTML. |
| **Availability:** I am willing to contribute 5 hours per week meeting included. I am also available sometimes on weekends. |

|  |
| --- |
| Team member: **Wahab Ahmed** |
| **Primary Role**: Documentation |
| **Capabilities**: I took SOEN 287 a year ago, which we learnt HTML, PHP, JS, and briefly touched databases for the final website assignment. I worked as a mechanical CAD drafter at some Toronto based company. |
| **Availability**: I can contribute 5 hours to the project weekly, meetings included. I am available to meet any time that doesn't clash with my class schedule, including weekends. |

|  |
| --- |
| Team member: **Sabrina Ashraff** |
| **Primary Role:** Documentation/Testing |
| **Capabilities**:I did SOEN 287, which I learned HTML5, CSS, Javascript and PHP. I did two co-op internships as an analyst programmer at CMC Electronics, customizing their customer resource management application to fit their needs as well as writing up documentation on where the customizations were made and creating instruction guides for non-technical users. |
| **Availability**:I can contribute around 4 hours per week, including meeting. I may miss class occasionally as I am currently searching for another internship under the co-op program. Weekend availability may vary as I work on weekends. |

|  |
| --- |
| Team Member : **Francis Bouchard** |
| **Primary Role**: Programmer |
| **Capabilities**: I took SOEN 287 (Web programming class) where we learned various languages such as HTML, CSS, Javascript and PHP. I had a 4 months of co-op internship as a web developer at UQAM. I also developed RESTful APIs to replace legacy applications. I have experience in Node.js and Angular. I have good understanding of various web development design patterns. |
| **Availability**: I can commit to 4 hours per week on this project including meetings. I may miss classes and meetings occasionally because of job interviews as I am currently looking for a second co-op internship for the summer. |

|  |
| --- |
| Team member: **Clozzy-Mailet Chavez** |
| **Primary Role**: Documentation |
| **Capabilities**:I took COEN 243/244/352, which I learned object oriented programming in C++, and a bit of Java, respectively. I also too ENGR 290/301, which I learned about project management, technical writing and technical presentation in a team environment. I have basic knowledge in HTML. |
| **Availability**:I can commit to about 5 hours per week, meetings included. |

|  |
| --- |
| Team Member: **Ricardo Cortes** |
| **Primary Role:** Documentation |
| **Capabilities**: I am a Electronic/Computer Engineering undergraduate and I am currently doing my M. Eng. Software Engineering. I took SOEN 6471, which teaches advanced software architectures, took SOEN 6481, a class on system requirement specifications, and SOEN 6461, which is software design methodologies. I have several years of work experience, in which I had performed different roles. I have had the opportunity to work in several project, at different phases, both development (new) and maintenance (existing), in different industries, with different methodologies. I can contribute to the project with my experience. I am good with Java, Design Patterns, Architectural Patterns. |
| **Availability**: I am available Monday to Friday only, about 4 hours per week, including meetings. |

|  |
| --- |
| Team Member: **Liuai Hatter** |
| **Primary Role**: Programmer/Leader |
| **Capabilities**:I took SOEN 287, but have basic knowledge of PHP. I took ENGR 301 (Eng. Management Principles and Economics), which I learned MS Project, work breakdown schedule, and Gantt charts. I am a graduate from Bachelor of Fine Arts and have worked as a graphic designer (all for web) for a tech company. |
| **Availability**: I would say about 5 hours a week (meetings included). I am also available to meet during weekdays, evenings, and sometimes weekends. |

|  |
| --- |
| Team Member: **Jian Huang** |
| **Primary Role**: Programmer |
| **Capabilities**: I took SOEN 287, which I learnt HTML5, CSS, Javascript and a bit of PHP. I also took ENGR 301, which is a management course for engineering, where I learnt a lot of processes. |
| **Availability**: I can contribute to as many house (5 or more hours) as possible to the project as I only have assignments to do, as long as it does not interfere with my class schedule. |

|  |
| --- |
| Team Member: **Anna Rogozin** |
| **Primary Role**: Programmer |
| **Capabilities**: I completed the Computer Science Technology program at Dawson College, where I had a few web programming classes similar to SOEN 287. I have gained experience through internships in the past and currently am on an internship dealing with machine-to-machine communication. I’m familiar with different design patterns and approaches to software development, which can be useful for this project. |
| **Availability**: I can contribute up to 5 hours per week, meeting included, but I prefer in the evening or during the weekend. |

|  |
| --- |
| Team Member: **Ramy Sandouk** |
| **Primary Role**: Documentation |
| **Capabilities**:I am a second year student in computer engineering. I took COEN 243, COEN 244, COEN 352 and COEN 311, which I learned C++, Data Structures and Computer Organization, respectively. |
| **Availability**:I can contribute 4 to 5 hours per week. |

|  |
| --- |
| Team Member: **Matthew Teolis** |
| Primary Role: **Programmer** |
| **Capabilities**: Iobtained my DEC in the Computer Science Technology program at Vanier College. I have a fairly good understanding of how the RESTful API design pattern works. I've worked with PHP, Git, MVC pattern, and the Laravel framework for one year in the workforce, including my internship time. I am a good and efficient coder |
| **Availability**:I am available 5 hours per week, including meeting, minimum. |

## **3.2. Technical Resources**

The following technical resources will be used in the development of the Apollo application:

* Multiplatform (windows, mac, linux)
* Frontend: angular material framework (to implement the material design)
* Backend: PHP with laravel framework, database with MySQL, data structures are with JSON.
* IDE: PHP Storm, Notepad++
* Gulp
* WAMP / MAMP / XAMPP

In terms of collaboration and communication between team members, the following technologies will be used:

* Github
* Slack
* Google Drive

# 

# 

# 

# 4**. SCOPING**

After careful evaluation of the resources available to develop the application, some features that had been initially been anticipated to be included have been left out. The following requirements have been scoped out of the project:

**1. System Administrator**

    1.1. System administrator can create new users

        1.1.1. System administrator can grant new users specific permissions

    1.2. System administrator can view existing users

    1.3. System administrator can update existing users’ information

        1.3.1. System administrator can change users’ permissions

    1.4. System administrator can remove existing users from the system

**Reason:** Unable to implement due to time constraints

**2. Faculty Administrator**

    2.1. Faculty administrator can create new courses

        2.1.1. Faculty administrator can assign requisites and requirements to courses

    2.2. Faculty administrator can view existing courses

    2.3. Faculty administrator can update existing courses’ information

        2.3.1. Faculty administrator can change course requisites and requirements

    2.4. Faculty administrator can remove existing courses from the system

**Reason:** Unable to implement due to time constraints

**3. Professor**

    3.1. Professor can view his or her course schedule

    3.2. Professor can update students’ grades

**Reason:** Unnecessary for general functionality of the system

**4. Student**

    4.1. Student can generate schedules

        4.1.4. Student can finalize a schedule

**Reason:** Unnecessary for general functionality of the system

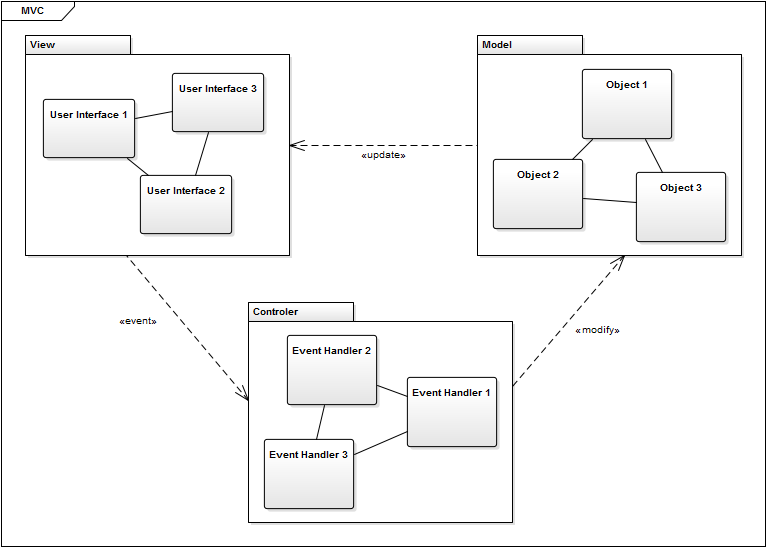
# 

# 

# 5**. SOLUTION SKETCH**

## 

## **5.1. Architecture**



The Apollo project will require the use of the Model-View-Controller (MVC) architecture design pattern. This pattern suits best for the development of our system because it contains distant separation between domain, the actions based on user inputs, and the presentation. This will ease up the development phase as programmers will be able to work on three different classes at the same time. The model will be the one that manages the behaviour of data, responds to information about the state, and responds to instructions to change state. To enforce this pattern, we will be using Laravel, a framework for developing web applications. The view, on the other hand, is a visual representation of the model. It will get information from the model and display it appropriately. The view will therefore need to know the semantics of the attributes of the model. We will be using Angular.js, a JavaScript framework, as well as Angular Material framework to implement a beautiful design for the user to interact with. The controller handles user interaction. The Controller will thus accept user inputs and convert this data into commands for the model or view. The controller can send commands to modify the state of the model and can also send commands to update the view that is associated with the model. As seen in the diagram, this is exactly what is happening. The user first sees the view and desires to make some change. Using the controller, the user sends commands to modify the model. Once the model is modified, it sends information to update the view when necessary. The user can then see what is updated in the view. The cycle continues.

All the communication between the backend and the front end will be done in a RESTful web service format. This means that the application communicates by exchanging resources back and forth using specific HTTP requests. For example, when someone wishes to store a course in the database, the user will create a JSON with all the course’s data inside. Then, the POST http method will be used on the url <http://apollo.matthewteolis.com/api/v1/courses>, with the json inside the payload. This will be interpreted by Laravel and stored in the database.

## **5.2. Technologies in Use**

### 5.2.1. PHP (Hypertext PreProcessor)

PHP is a web scripting language. The application is a web application which explains why PHP has been chosen as the language. The server will be running off of PHP scripts, in conjunction with Laravel (which is a PHP framework). PHP is an easy to learn and a very powerful language. It will make the project development quick and easy.

### 5.2.2. Laravel

Laravel is an open source PHP framework. It follows the MVC pattern using their own blade compiler. This allows developers to quickly deploy web projects. Laravel's Eloquent, which is uses the Object-relational mapping (ORM) technique, makes database manipulations really easy because it can access the database as if they were their own objects. Creating and populating databases to different machines is also easily done with the database migration commands they offer, they made their own database version control. Laravel has the RESTful design pattern integrated with the resource controller, therefore you can create a new REST URL in 2 lines.

### 5.2.3. **JetBrains** **PHPStorm**

The reason we chose to go with PHPStorm is because it has a lot of integrations with the IDE. The IDE has integrations such as Git, Vagrant, Composer, debug with a live server, has good code sniffing and intellisense, use the terminal (right in the IDE), Gulp, NodeJS, Laravel support, make SSH connections, even use RESTful requests. PHPStorm is very powerful tool that makes the development process faster and easier.

### 5.2.4. Gulp

Gulp is a minifying application. Gulp will be obfuscating and minifying all css and javascript files in the project. There is also a watch feature that automatically minifies and combines the javascript and css files when the user saves the file, which makes it quick for development.

### 5.2.5. **Angular.js**

AngularJS is an open source javascript framework. It binds data to HTML with expressions and extends HTML attributes with directives. Angular’s code is downloaded and included in the html files.

Angular will be used to perform the REST API calls on the front-end of our applications. It will communicate with the Laravel back-end. Angular Material framework will be used to implement the material design, this will give our website a easy to use, beautiful user interface.

### 5.2.6. **HTML**

HTML (HyperText Markup Language) is a markup language used to display content from the web. Any web app must use HTML to be able to interact with the user. HTML uses tags to describe every element on the web page.

### 

### 5.2.7. **JSON**

JSON stands for Javascript Object Notation. It is a popular format used to store and exchange data. It is an alternative to the XML format. We will be using this format because it is easier to use and more readable than XML. All the information we collect and send to the user will be stored in a JSON.

### 5.2.8. **Git**

Git is a code repository and Github is a git hosting service. All code and documentation is maintained using Github. Using it for documentation with Google Drive is redundant; but not every team member is familiar with Github, therefore at this point in the project documentation is maintained on both systems. The benefits of Github as a repository for documentation and code is that issues can be made, team members assigned to the issue, and the issue can be tracked until resolved. Then the code or documentation can be pushed onto the repository by the issue handler.

### 5.2.9. **Google Drive**

Google Drive is a repository for documents. It does not offer issue tracking but has the advantage of being easy to learn. All documents can be edited by a member of the team.

### 5.2.10. **Slack**

Slack is a cloud based collaboration software. It enables the creation of many channels and allows for group and individual communications. Channels can also be linked to other tools and software. For example, a #calendar channel was created where group meetings are organized and scheduled; it is also directly linked to the group's Google Calendar. The #calendar channel on Slack is updated and group members alerted anytime the Google Calendar is changed. There are also a #general channel where aspects of the project are discussed, a #useful channel where important links and reminders are posted. Channels can also be made for subgroups, such #softwareteam for the programming sub-group. Alerts can specified by each team member: to receive an email notification at every update in Slack, or only for personal messages, or notification after only a certain period of time. There is also a Slack desktop app which can notify users directly.

### 5.2.11. **MySQL**

MySQL is an open-source relational database management system (RDBMS). It is widely used for both small and large web-based applications. We will be using MySQL due to its high-performance and reliability. Its cross-platform support suits the many operating systems that we will use. In addition, MySQL combined with PHP (our chosen programming language) allows rapid web application development.

### 5.2.12. **Apache Server**

The Apache HTTP Server is a powerful open-source tool that provides HTTP services and integrates well with PHP. It is a secure, efficient, and extensible web server that is easy to manage and is available for all the platforms used in our project.

### 5.2.13. **XAMPP**

XAMPP is a web server solution pack that stands for cross-platform Apache HTTP Server MySQL PHP Perl. It is a portable development environment that introduces consistency to a team working on multiple platforms and that allows testing on local machines. XAMPP includes the previously mentioned PHP, MySQL, and Apache Server technologies, thus enabling us to decrease development setup time.

# 

# 

# 6**. PLAN**

## **6.1. Activities**

|  |  |
| --- | --- |
| Research | |
| Description | Research for technologies in use (eg: frameworks, design patterns, design specifications etc). Requirements research (eg: classrooms, requisites, professors). |
| Artifacts | 15, 26 |

|  |  |
| --- | --- |
| Discussion | |
| Description | Multiple weekly meetings (in person and online). Assigning various tasks to team members and discussing solutions and approaches to our application. |
| Artifacts | 1, 2, 4, 5, 12, 13, 28 |

|  |  |
| --- | --- |
| Planning & Designing | |
| Description | Planning out how to implement the application design (eg: drawing database architecture, domain model, uml diagrams, frontend design, dynamic design scenarios etc.) |
| Artifacts | 1, 3, 4, 5, 20, 21 |

|  |  |
| --- | --- |
| Technical Configuration | |
| Description | Setting up our framework, version control, slack communication, google drive. Setting up the frontend and backend communication. Setting up the database. |
| Artifacts | 6, 7 |

|  |  |
| --- | --- |
| Data Input | |
| Description | Extracting course information and inputting course data into the database. |
| Artifacts | 9 |

|  |  |
| --- | --- |
| Implementation | |
| Description | Importing the framework modules. Creating RESTful API. Implementing all features. |
| Artifacts | 10, 14, 18, 19 |

|  |  |
| --- | --- |
| Testing & Debugging | |
| Description | Creating a test plan. Testing application for functionality and checking if features are working correctly. Verifying bugs are fixed. |
| Artifacts | 23, 24 |

|  |  |
| --- | --- |
| Documentation | |
| Description | Documenting all planning, specifications, design, and issues encountered. Creating the user manual. Documenting source code. |
| Artifacts | 3, 8, 11, 16, 17, 20, 22, 25, 27, 28 |

## 

## **6.2. Artifacts**

|  |  |
| --- | --- |
| 1 | Member Organization |
| Team members subdivided themselves into groups of programmers and documenters according to skill set. | |

|  |  |
| --- | --- |
| 2 | Initial Discussion |
| The initial discussion involves the gathering of ideas and information around the project. Initial software requirements and designs were proposed. Team members shared their key strengths in order to participate effectively in the process. A brainstorming of the structural and functional requirements of the scheduler lead to the establishment of a foundation for the software development process. | |

|  |  |
| --- | --- |
| 3 | Deliverable 0 |
| Deliverable 0 consisted of all the elements that were discussed in the initial discussion. A description for the project was provided, along with the domain model diagram and a description of the areas within the domain model. An initial list of the team members and the tasks they were interested in accomplishing regarding the application was also drawn up. | |

|  |  |
| --- | --- |
| 4 | Scheduling of Tasks |
| This artifact involved assigning specific tasks to each member of the team regarding the project. Meeting times for each subgroup working on a certain task were also arranged and set up, as well as dates as to when each task should be expected to be completed. | |

|  |  |
| --- | --- |
| 5 | Resource Evaluation |
| Establishing everyone's technical capacities and the amount of time each individual can spend working on the project. Collected every technical resource available to the team including software tools and computers. | |

|  |  |
| --- | --- |
| 6 | Framework Configuration |
| Downloading and installing laravel on each programmer's machine. Importing the various libraries needed to make run the application through the use of composer. Importing the database. | |

|  |  |
| --- | --- |
| 7 | Collaboration Tools |
| Setting up every team member with all collaboration tools. Creating a slack team to set up meetings, to discuss and to communicate useful information. Creating a Google Drive for all team members to contribute documents, in order to work on collaboratively. Creating a GitHub organization for all teams members to access the repository which includes documents and codes. This allows to keep track of any revisions and issues that occur. | |

|  |  |
| --- | --- |
| 8 | Domain Model Design |
| Creating the domain model diagram to document the key concepts and terminology of the system. It consists of domain level objects, their attributes and associations. | |

|  |  |
| --- | --- |
| 9 | Course Database |
| Adding to the course database the course titles, professors, time slots, requisites, descriptions, credits and faculty of each course required for the software engineering degree. Inputting the course lecture, tutorial, and lab time slots. | |

|  |  |
| --- | --- |
| 10 | Prototype |
| Creating a live prototype to allow team members to input course information to the database. The prototype will demonstrate the integration of laravel and angular.js frameworks for our application. | |

|  |  |
| --- | --- |
| 11 | Use Case Tables & Diagrams |
| The use case tables describe in detail every possible user (actor) interaction with the system. The use case diagrams visualize these interactions, grouped by actors. This artifact is important for the identification of functionalities of the system. | |

|  |  |
| --- | --- |
| 12 | Constraints Identification |
| This artifact involves analyzing and classifying any constraints that the team might have. It includes design, hardware, performance, security, or external constraints. The identified constraints are then described in detail. | |

|  |  |
| --- | --- |
| 13 | Scoping |
| The scoping artifact describes the functionalities that are removed from the final system design due to limited time or resources. The functionalities may include features, goals, or requirements and qualities of the system. For each scoped out functionality, a clear reason is given. | |

|  |  |
| --- | --- |
| 14 | Initial Front End Planning & Design |
| Establishing design patterns to follow when creating the user interface. Choosing a color palette. Drawing mock-ups. | |

|  |  |
| --- | --- |
| 15 | Estimation & Risk Planning |
| Carefully identifying risks that the project can have so team members can take pre-caution to avoid those risks. Estimating the total cost of the project after carefully estimating the cost of production of each artifact. Also estimating the time for each task and scheduling accordingly. | |

|  |  |
| --- | --- |
| 16 | Deliverable 1 Documentation |
| Deliverable 1 consisted of the initial project description and domain model which was provided in deliverable 0. In addition to that, after several team meetings we defined the goals of our project in the document along with all the constraints, some of which were scoped down later on. Human and technical resources were discussed and tasks were split accordingly. An early overview of our solution and plan along with a prototype was provided in this document. | |

|  |  |
| --- | --- |
| 17 | Architecture Design & Diagrams |
| An updated and detailed version of the architectural design. It includes a description of the reasons behind our design, the changes we made from the previous design in Deliverable 1, and all the components (such as function calls and description of parameters) included in our new design. This will be presented in the form of UML diagrams for our complete description of the system design. | |

|  |  |
| --- | --- |
| 18 | Frontend Implementation |
| Development of the frontend of our system, the part where the user can interact with. | |

|  |  |
| --- | --- |
| 19 | Backend Implementation |
| Development of the backend of our system, which includes the server, the application and the database. | |

|  |  |
| --- | --- |
| 20 | UML Class Diagrams |
| Create Unified Modelling Language (UML) class diagrams for all classes of program. Responsible person who meet with developing team. | |

|  |  |
| --- | --- |
| 21 | Dynamic Design Scenarios |
| A full dynamic design of use cases including system sequence designs, operational contracts and sequence diagrams. | |

|  |  |
| --- | --- |
| 22 | Deliverable 2 Documentation |
| Deliverable 2 consists of a much more structured and detailed design for our system, including better UML class diagrams. It also includes an updated, and more a detailed version of the architectural design from the previous one. Dynamic design diagrams are also well put together. This deliverable will help the programmers give a more organised overview of the system, which they can report on its rapid prototypes and examine the effects on estimates, risks and scopes. | |

|  |  |
| --- | --- |
| 23 | Test Planning & Testing |
| Meet with the programmers and discuss the program’s possible weaknesses, compile a list. Meet with the testing team and plan out use cases specific to weakness discussed with developers; also plan other cases covering all other aspects the testing team can derive. Compile use cases, test them on program and record success, failures, and comments. | |

|  |  |
| --- | --- |
| 24 | Improvements |
| Implement risk management for all problems found during testing; treat problems found as identified risks. Implement risk analysis (prioritizing), risk planning (mitigation, avoidance, and reprogramming), and create document for risk monitoring to be part of system administrator’s user manual. | |

|  |  |
| --- | --- |
| 25 | User Manual |
| Create User manual for end user and system administrator. | |

|  |  |
| --- | --- |
| 26 | Final Cost Estimate |
| A final listing of all components of all phases of the project, including the hours cost per person for each of the components of each phase. | |

|  |  |
| --- | --- |
| 27 | Deliverable 3 Documentation |
| Deliverable 3 consists of documenting the testing process, including unit, requirement, stress and security testing. Includes the user manual, and final cost estimate. | |

|  |  |
| --- | --- |
| 28 | Project Documentation & Presentation |
| This includes the final and complete documentation of the project. A presentation is given to demonstrate the fully functional application. | |

## 

## 

## **6.3. Project Estimates**

|  |  |
| --- | --- |
| Tasks | Project Estimate (Hours) |
| Member Organisation | 1 |
| Initial Planning | 2 |
| Deliverable 0 | 4 |
| Scheduling | 2 |
| Resource Evaluation | 5 |
| Technical Configuration | 3 |
| Domain Model Design | 3 |
| Architecture Planning | 5 |
| Initial Architecture Design | 3 |
| Database Setup | 24 |
| Rapid Prototype | 24 |
| Data Input | 15 |
| Functional Requirements Planning | 10 |
| Constraint planning | 5 |
| Scoping | 5 |
| Initial Front End Planning | 5 |
| Estimation 1 | 1 |
| Risk Planning 1 | 1 |
| Deliverable 1 | 30 |
| Architecture Design | 5 |
| Front End Planning | 10 |
| Front End Design | 10 |
| Detailed Design | 10 |
| Dynamic Design Scenarios | 20 |
| Front End Prototype | 30 |
| Estimation 2 | 1 |
| Risk Planning 2 | 3 |
| Deliverable 2 | 30 |
| Implementation Planning | 15 |
| Implementation | 40 |
| Test Planning | 15 |
| Testing | 30 |
| Improvement | 15 |
| Testing | 10 |
| User Manual | 20 |
| Final Cost Estimate | 2 |
| Deliverable 3 | 40 |
| Project Documentation | 80 |
| Deliverable 4 | 15 |
| **TOTAL** | **549** |

## **6.4. Project Re-estimation Scenarios**

### 6.4.1. **Front End Prototype**

Re-estimation of the hours costed may occur due to unforeseen circumstances that may occur in the creation of the prototype. Some parts of may take more time than anticipated if it does not function the way it is intended, or if we run into bugs in initial stages. Similarly, it may require less time than anticipated if everything runs smoothly.

### 6.4.2. **Implementation**

Re-estimation of hours costed may occur while implementing features and writing code. This may include initial bugs that may immediately occur and must be resolved, as well as running into problems implementing certain functionalities that may not have been predicted. Similarly, it may cost less time if everything runs smoothly without issue.

### 6.4.3. **Testing**

Re-estimation may be necessary as the duration of the testing phase depends on how well the initial application functions. If there are more problems and bugs than anticipated, the testing phase may take longer as more tests will be required to point out any malfunctioning areas.

### 6.4.4. **Improvement**

Re-estimation may be necessary as the improvement phase is largely dependent on how well the initial application functions. It may require longer if more problems and bugs than anticipated are found and need to be fixed.

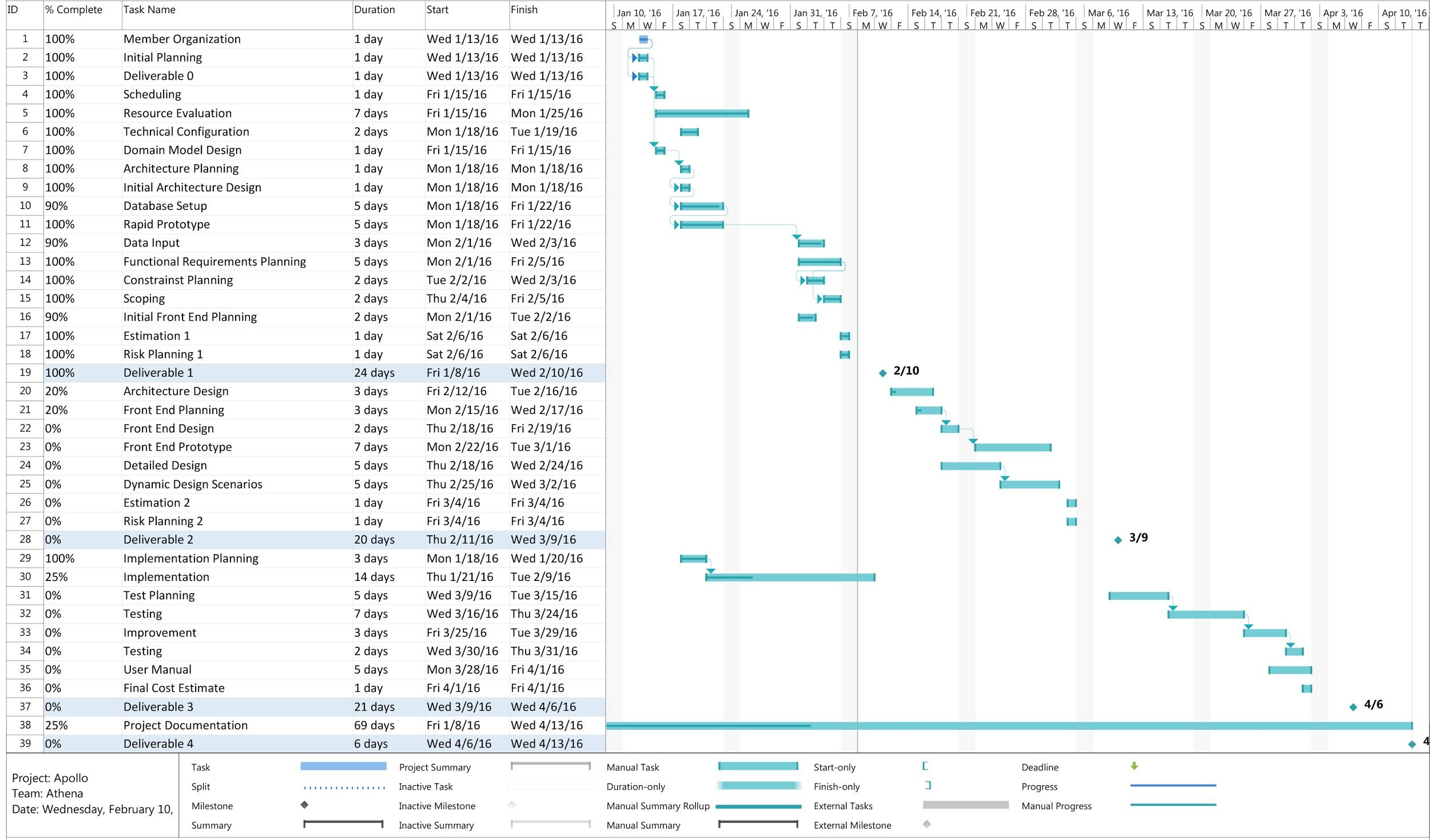
## **6.5. Activities Assignment**

|  |  |
| --- | --- |
| **Team Member** | **Activities Assignment** |
| Philippe Abou Kasm | Data input, Plan (7.1, 7.2, 7.4, 7.5), Documentation, Testing |
| Wahab Ahmed | Plan (7.3, 7.6), Documentation, Testing |
| Sabrina Ashraff | Presentation, Project description, Plan (7.1, 7.2, 7.4, 7.5), Programming |
| Francis Bouchard | Solution Sketch (6.2), Plan (7.1, 7.2, 7.4, 7.5), Prototyping, Programming (Front-end) |
| Clozzy Chavez | Data input, Plan (7.1, 7.2, 7.4, 7.5), Presentation, Documentation, Testing |
| Ricardo Contés | Goals & Constraints, Scoping, Documentation |
| Liuai Hatter | Lead, Goals & Constraints, Scoping, Plan (7.1, 7.2, 7.4, 7.5), Presentation, Programming (Front-end) |
| Jian Huang | Solution Sketch (6.1), Plan (7.3, 7.6), Programming |
| Anna Rogozin | Goals & Constraints, Scoping, Solution Sketch (6.2), Programming (back-end) |
| Ramy Sandouk | Solution Sketch (6.2), Documentation, Testing |
| Matthew Teolis | Solution Sketch (6.2), Prototype, Data input, Programming (front-end & back-end) |

## 

## 

## **6.6. Schedule**



## **6.7. Risks**

### 6.7.1. **Use of an unfamiliar framework**

* Not everyone in the team is knowledgeable about the Laravel framework being used for the project.
* Amateur developers might face problems while extending code and classes, as it is a new platform for most of the developers to deal with.
* Because the team is working with a new framework, the scheduling part of the project is a hard task to do and learning how to work with the framework could delay the assigned tasks.

### 6.7.2. **Different levels of programming knowledge**

* Not everyone in the team comes from a software engineering background. Different levels of experience with the programming language could result in inability to understand complex pieces of code for some team members.
* Not everyone necessarily follows commenting guidelines. One developer might have coded something without proper documentation about it which could be confusing for other developers working on the same project resulting in delays and other problems.

### 6.7.3. **Schedule generating issues**

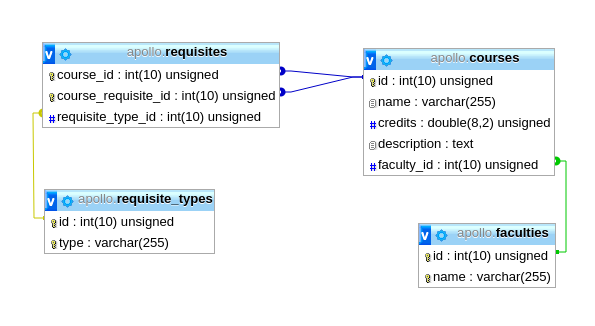
* Students might end up with a course in their schedule that they shouldn’t be allowed to take according to university’s rule and regulations. For example, a student is registered for a regular Fall/Winter year, and he fails a course in fall which is a prerequisite for some other course in winter (or gets a D+ or lower in some 200 level course). With this issue the student could still be enrolled in the course in winter even after failing its prerequisite which could violate the university’s regulations.The scheduler might show a specific class full before its full capacity has been reached, possibly resulting in graduation delay for many students. For example, if a class like SOEN 342 or 343 is shown full when it is actually is not, a student will have to wait one full academic year to be able to take it as it's offered only once every year.

### 6.7.4. **Security issues**

* Security is a huge problem when it comes to storing confidential data about the users in a database. Security measures such as having a secured server to handle the information and much testing is needed before the final release of our software.

# 

# 7**. PROTOTYP**ING

We are using a RESTful API using laravel to connect the backend to the frontend. We can create, delete, list all courses and show specific courses and faculties. We display this information using Angular.js, and the Angular Material framework. Angular send AJAX requests to the backend and receives a JSON containing the information requested. We have a working database for the courses, prerequisites and faculties. The following diagram shows the structure of the current database for the prototype:

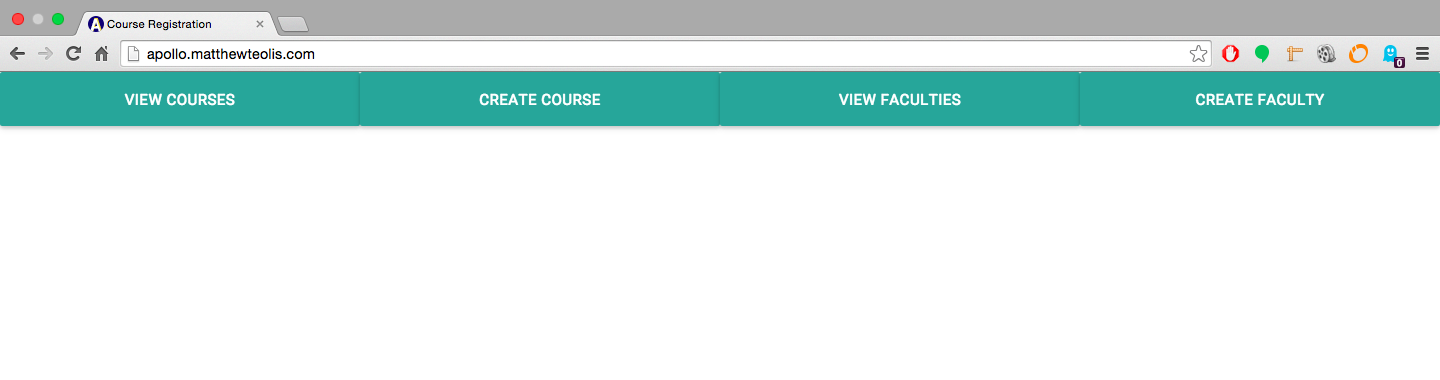
The prototype website has been used in order to populate all the courses. The data has been taken from Concordia’s course list.

**URL**: <http://apollo.matthewteolis.com/>

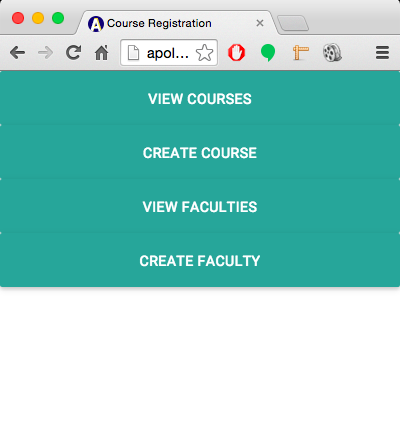
**Username**: soen

**Password**: 341

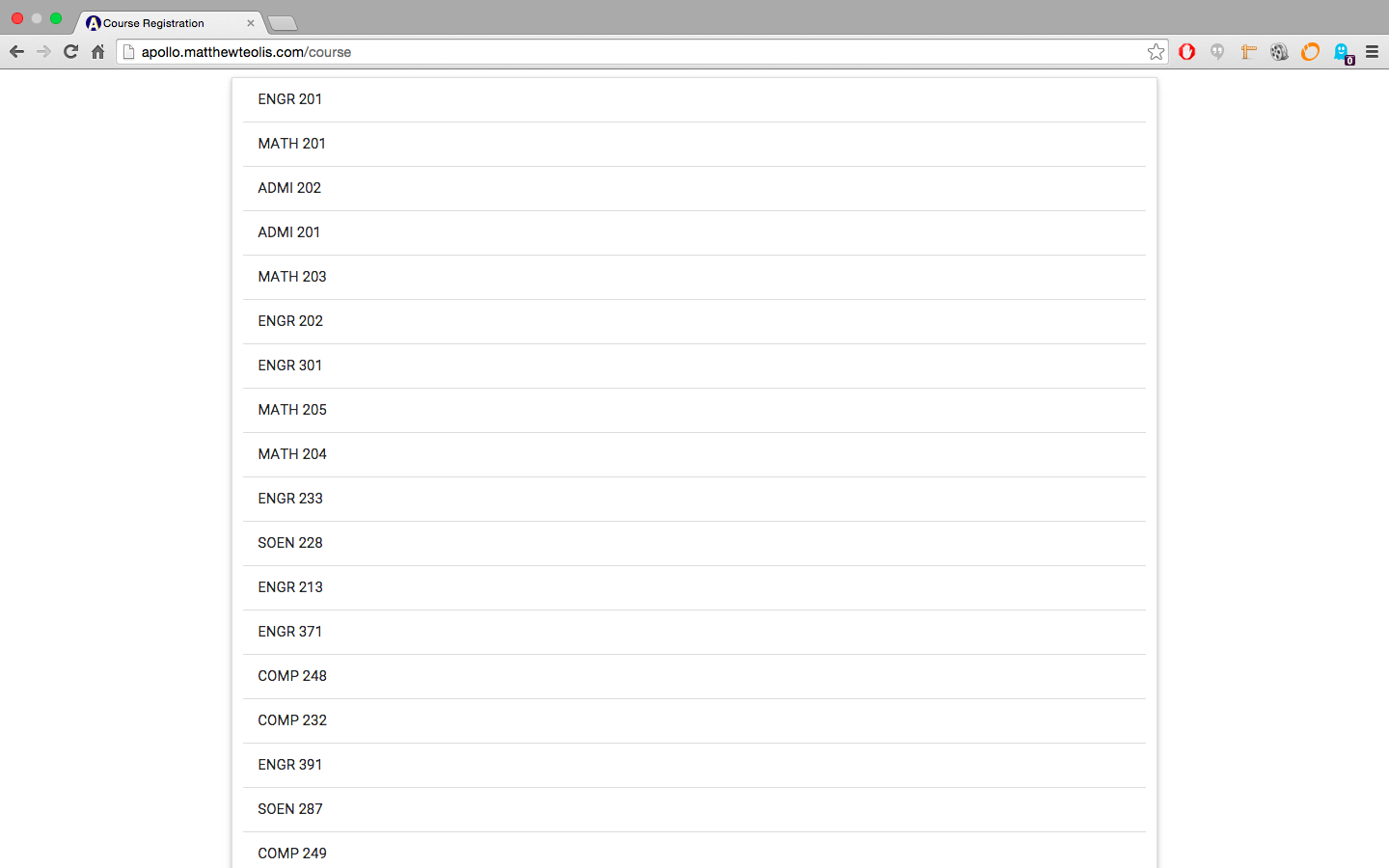
**Full Width Screenshot**



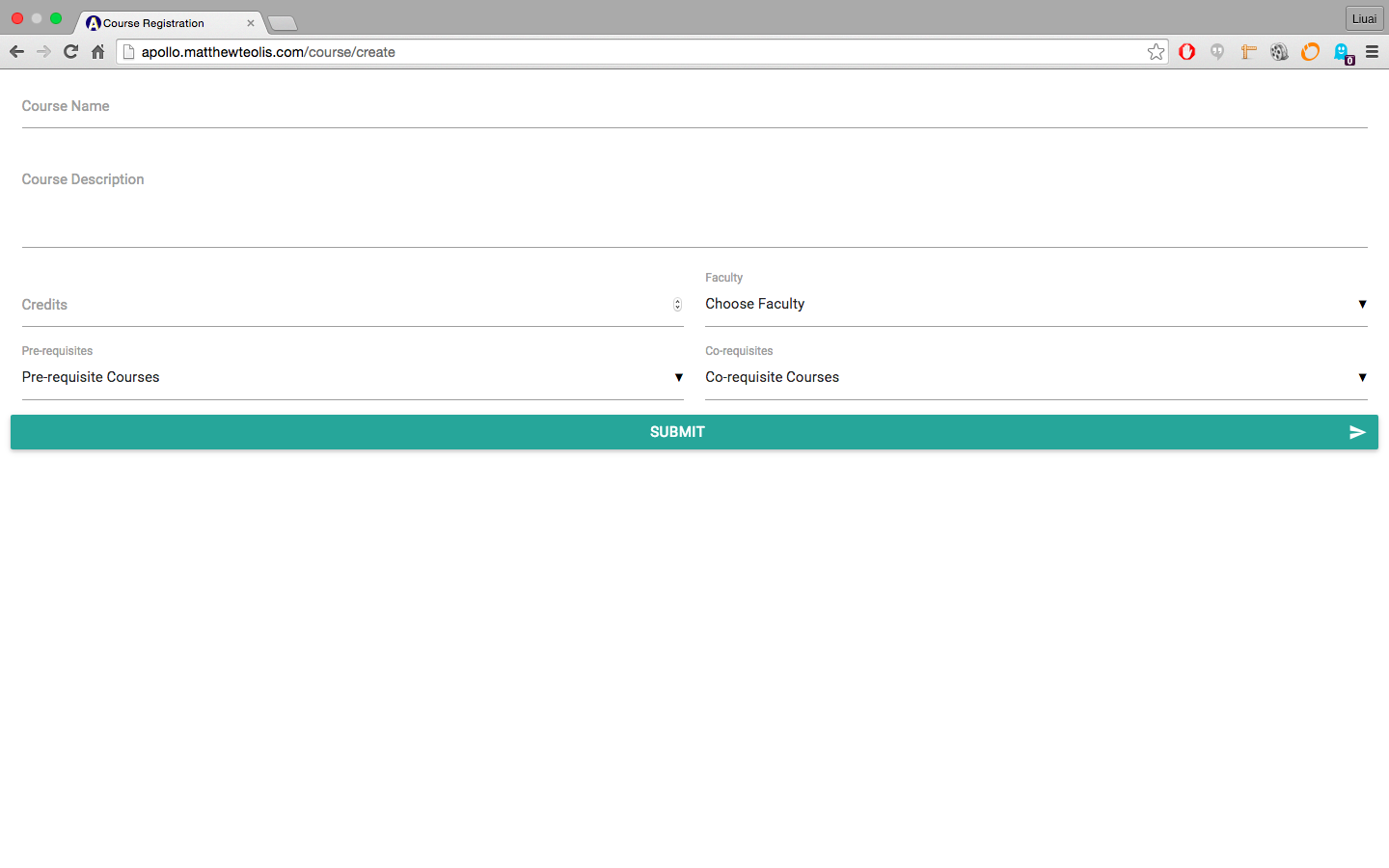
**Mobile Width Screenshot**



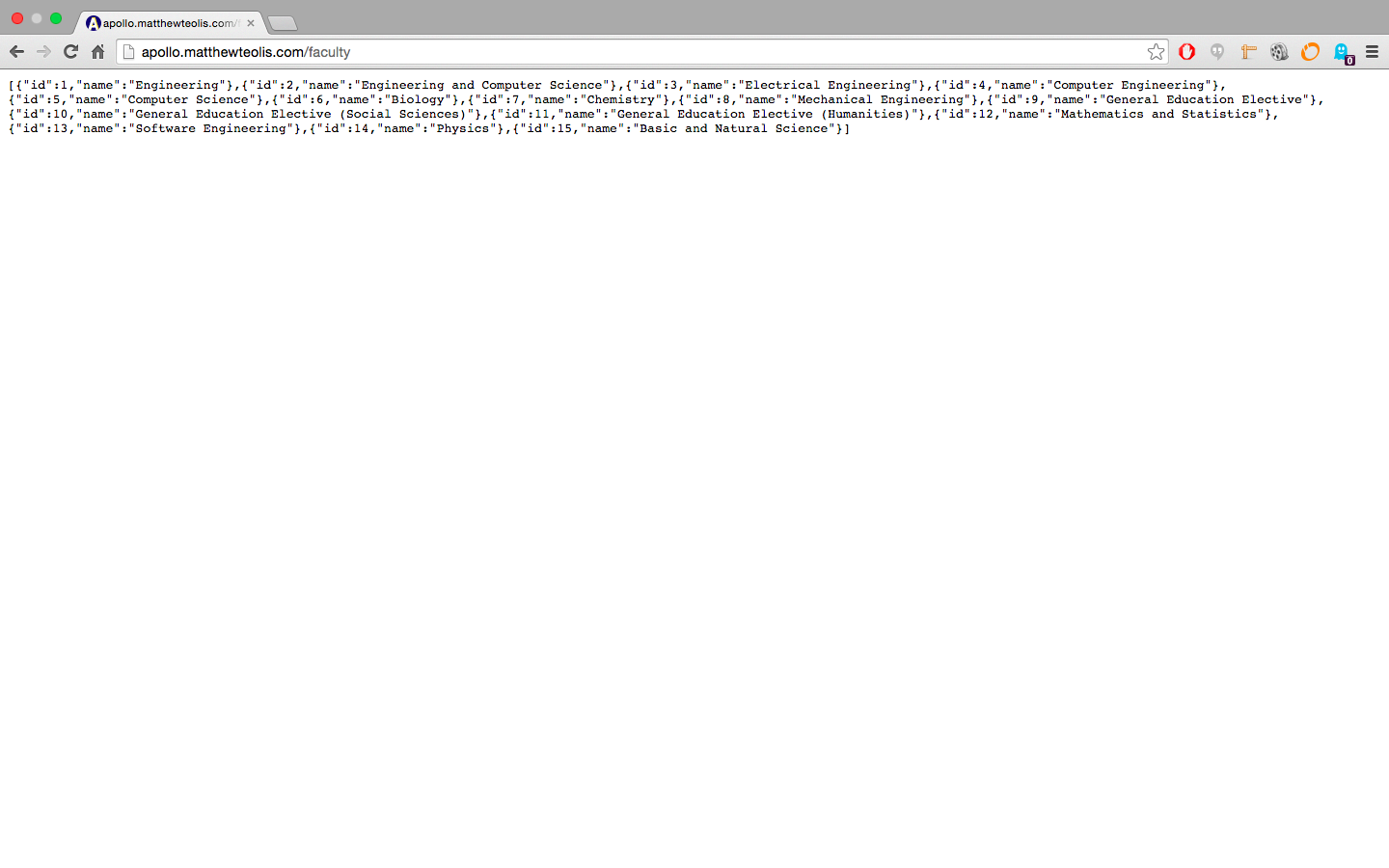
**View Courses Page**



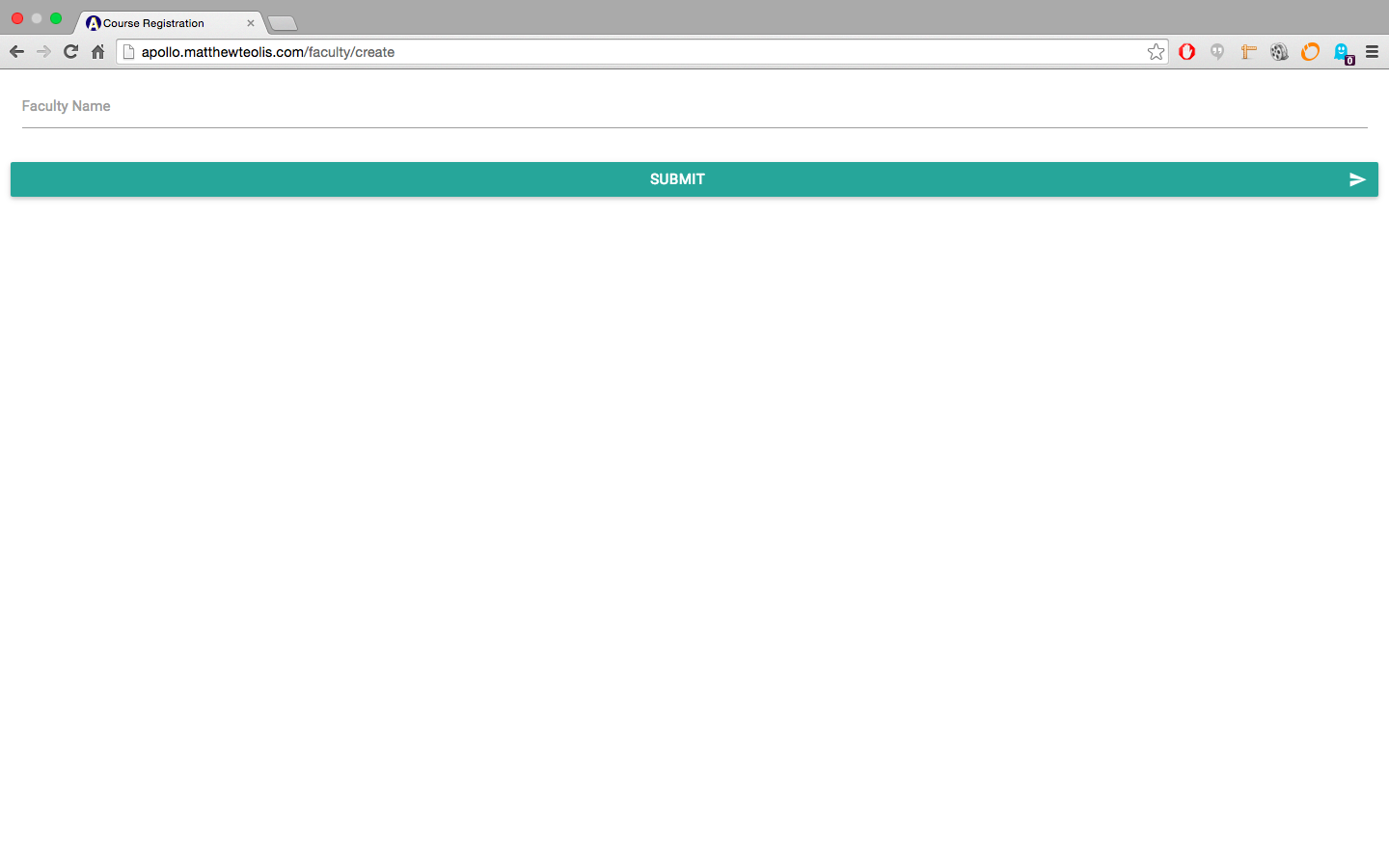
**Create Course Page**



**View Faculties Page**



**Create Faculty Page**

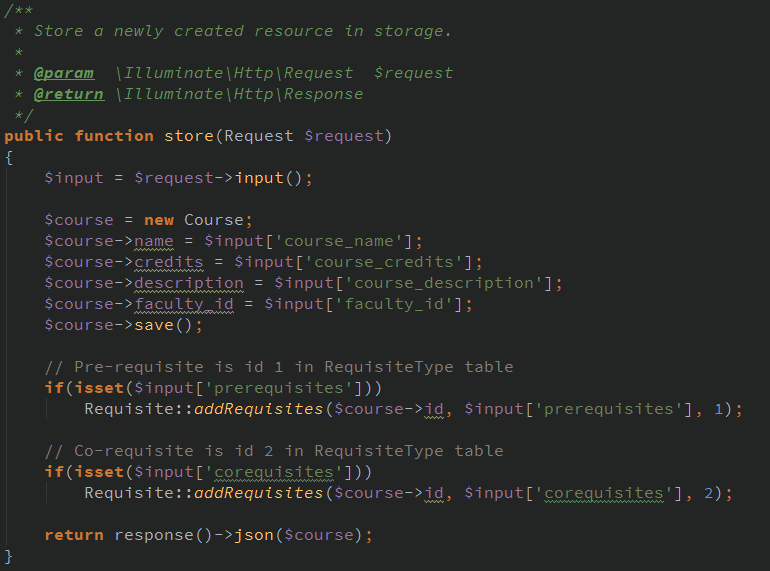


**RESTful Request Table**

|  |  |  |
| --- | --- | --- |
| **Verb** | **Path** | **Action** |
| **POST** | api/v1/course | Inserts a new course in the database. |
| **GET** | api/v1/course | Retrieves all courses from the database. |
| **GET** | api/v1/course/{id} | Retrieves a specific course with a matching id parameter. |
| **PUT** | api/v1/course/{id} | Updates a specific course with new attributes. |
| **DELETE** | api/v1/course/{id} | Deletes a specific course with a matching id parameter. |

NOTE: Please refrain from altering the data.

Storing a new course to the database:



Using the RESTful API conventions, listed in the previous RESTful request table , the store method listens for a “POST” request at the “api/v1/course” route.

Course Model:



The course model access the database directly. By default it will access the “courses” table because the name of the model is “Course” (it adds an “s” at the end of the model name). Therefore, creating new courses and saving them to the database can be done easily.

# 

# 

# 

# 8**. ARCHITECTURAL DESIGN**

In this section a high-level description of the system and its modules is presented.

## **8.1. Architecture Diagram**

In this section the design of the system will be described using the 4+1 Architectural View Model (AVM). This model describe the architecture of a system based on multiple complementary views, considering the viewpoints of different stakeholders of the system (e.g. developer, architect, user, etc.). This view model propose to have a physical, logical, development, and process view, supported by different UML diagrams, plus selected use cases.

### 8.1.1. **Logical View: Class Diagram**

In the 4+1 AVM, the Logical View describes the functional aspect of the system. In other words, what the system should provide in terms of service to its users. To represent this view the several UML Diagram can be used (e.g. class, object, sequence, communication, state, etc.). In this section the Class Diagram is described.

The Class Diagram presented here is an overall view of the subsystems that make up the Apollo application. In this model, we have two subsystems which are the Student and Courses subsystems. The Student subsystem is responsible for handling all the information that is pertinent to a student. This includes general information about a Student, such as their name and login information, their Course Record, which keeps track of all courses completed and the grade received, and their Preferences in regards to their schedule, such as at what time they prefer to have their courses placed in their schedule. The Courses subsystem manages all the information regarding courses that are available. Courses provides a master course list with general information regarding each course available, including the Faculty it is offered by. Subsequently, Scheduled Courses would further take this information and check for courses available per session, and Time Slots would further take this information and specify information such as which room it is in and the time and sections available. The Schedule class accesses information from both subsystems, utilizing the preferences and student information available from the Student subsystem, and the courses available from the Courses subsystem to add courses to the schedule and generate the appropriate schedules based on this information.

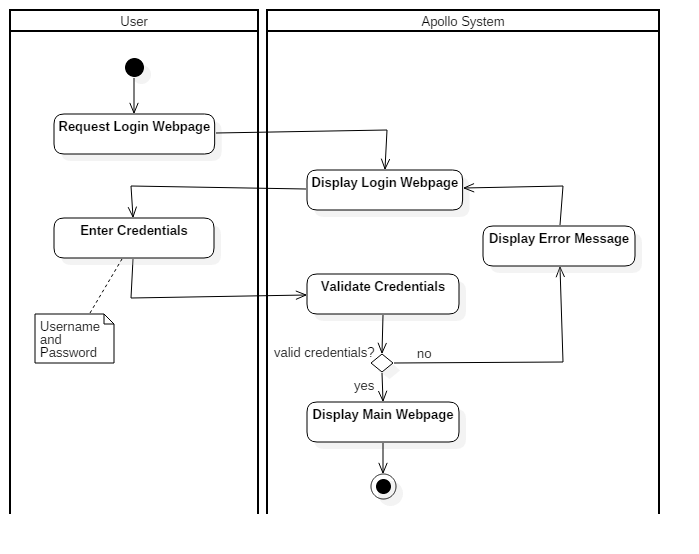
FullClassDIagram.png

### 8**.1.2. Process View: Activity Diagram**

In the 4+1 AVM, the Process View describes the dynamic aspect of the system. Although it was proposed mainly to represent dynamic behaviour of the system (runtime aspects), it can be used as well for interactions with the users. When referring to software elements, typical concerns about the process view are communication between objects and components, integration, concurrency issues and transactions. When referring to interaction between users and the system, it concerns about functional processes. In this design document, the latter has been addressed. In the following sections the most representative process for each user has been described using UML Activity Diagram notation.

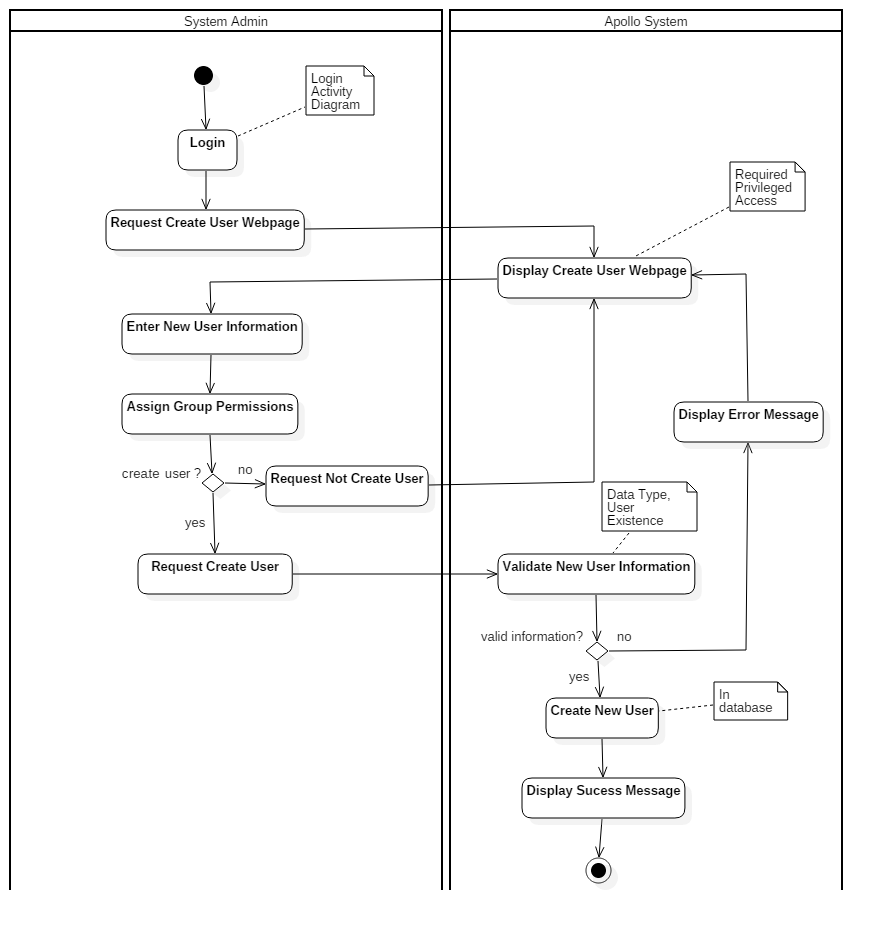
#### **Activity Diagram “Login” (by All Users)**

The activity diagram shown below depicts the actions performed by any user that attempts to login into the Apollo System. Since the use of the system requires privileged access, prior to displaying any option, the system requires the user to provide correct credentials (i.e. username and password). Once the access has been granted, the system displays profile information (i.e. personal data and allowed actions) associated with the user logged in.



#### **Activity Diagram “Create User” (by System Admin)**

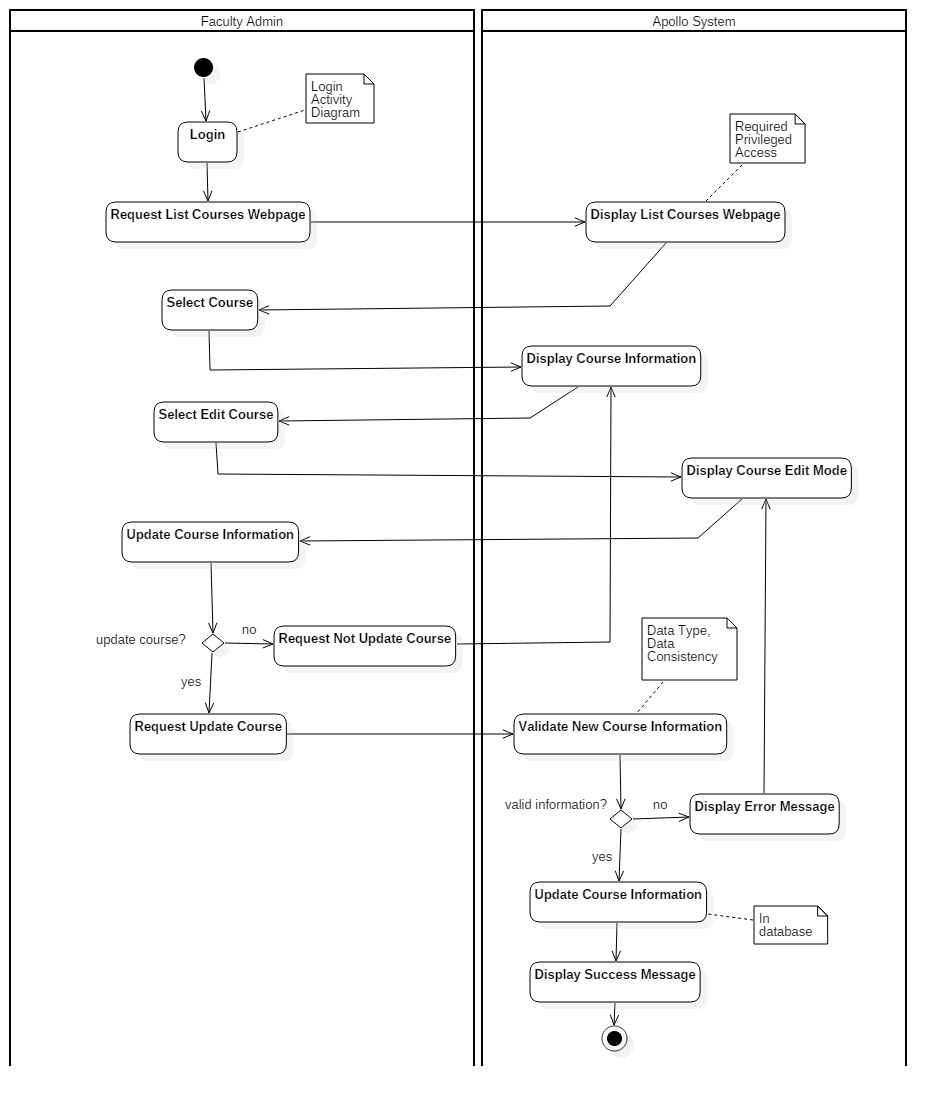
The activity diagram shown below describes the process by which the System Admin creates a user in the Apollo System. Like all the available options, it starts with the login of the user into the Apollo System, process previously described. Once the access has been granted, among all the available options, the user request to display the Create User web page, option only available for users with System Admin privileges, as shown in the diagram. When the web page has been displayed, the new user information is entered along the group permissions. At this point the System Admin can either proceed with the creation or cancel it. If proceed is selected, some standard verification is performed by the Apollo System (like user uniqueness). If the validations are successful, the system creates the new user into the database, and that action is informed to the System Admin. It can be said that this is a standard user creation process, conveniently adopted in the system (by using standard process some non functional requirements are improved, like maintainability).



#### **Activity Diagram “Update Course” (by Faculty Admin)**

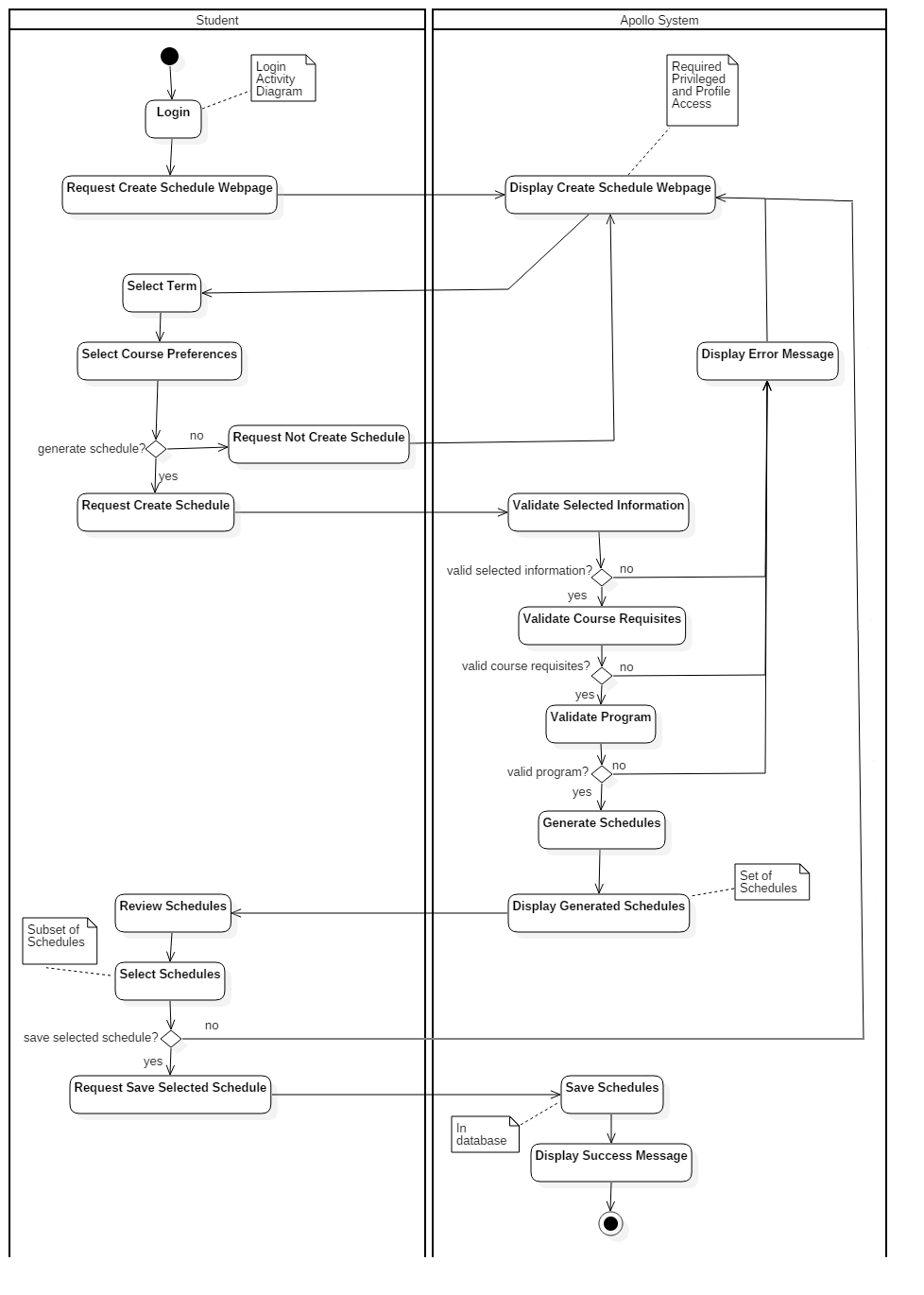
The diagram shown on the following page details the activities performed by a Faculty Admin user in order to update a course’s information. After successful login, the user selects from the menu the option to list all the available courses in the system. From the list displayed, the Faculty Admin selects the course of interest to be reviewed in detail. In that page, the user can select to update it. When this action is requested, the system displays the course in edition mode, which means that all the editable information is available for changes. When the Faculty Admin concludes the update actions, it can select to save the change or discard them. If update the course is selected, some standard validations are conducted before save the information into the database. When the changes are saved, a success message is displayed to confirm the action.

To update a course another process can be followed, when instead of listing all available courses the user can select to search for a specific one. Once the course has been found and opened, the process continues exactly as the one described here (from Select Course).



#### **Activity Diagram “Generate Schedules” (by Student)**

The activity diagram shown below describes the process followed by a Student to generate his or her schedules. Like all the previous processes, after a successful login, the request to the system displays the web page to generate the schedules. In that page, students can select their preferences to create a schedule (i.e. time, term and courses). When continue is selected, thus create the schedule, Apollo System performs first a serie of required validations (e.g. course grade and requisites) before generating the different schedules.



### 8.1.3. **Physical View: Deployment Diagram**

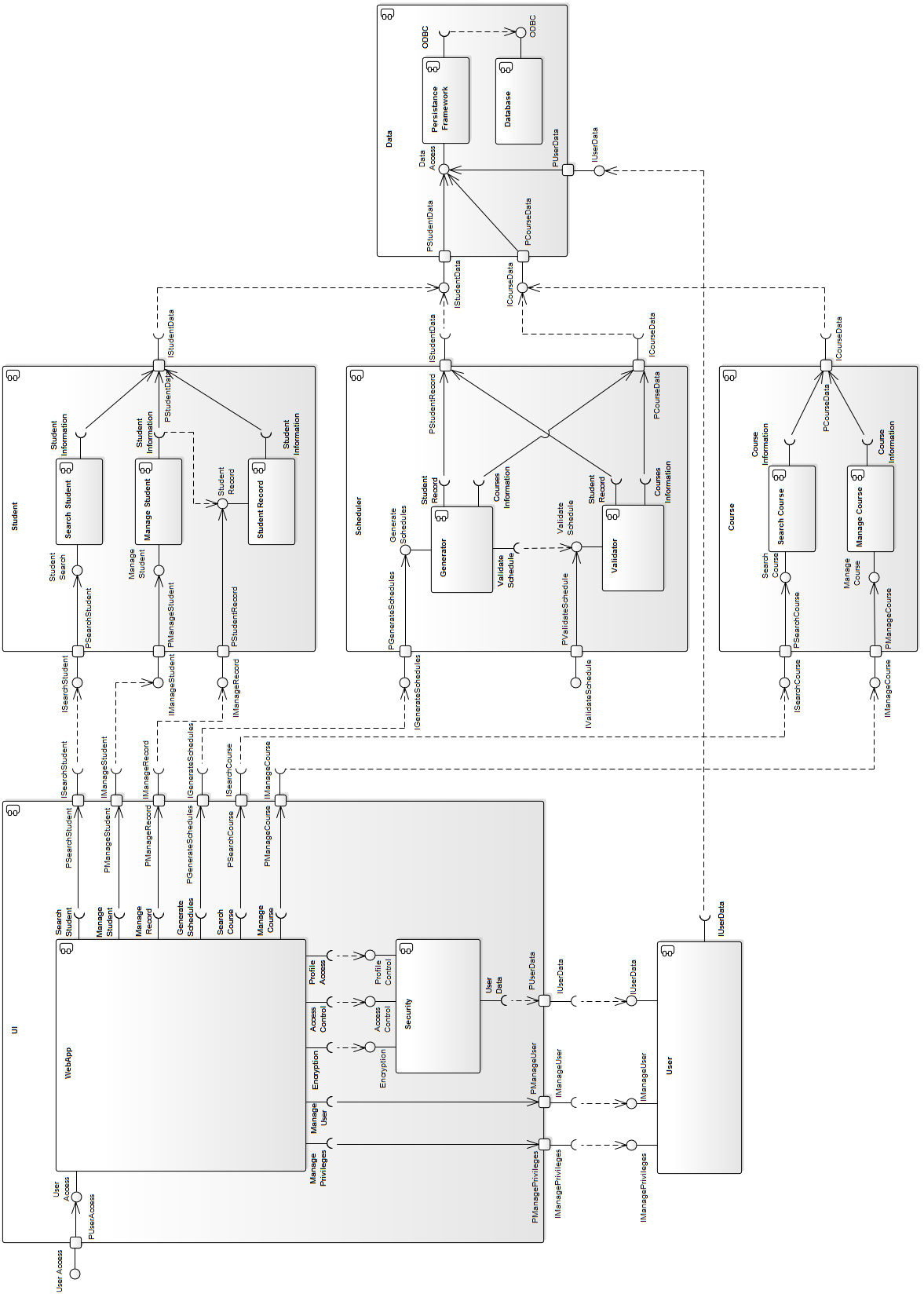
The deployment of our application is divided into two main physical nodes, as shown in the figure. The client node can be any device with any operating system, as the application is platform-independent. Clients interact with the application through a browser of their choice. The application server is based on the Apache web server and the MySQL database.

deploymentDiagram.png

The client communicates with the web server through HTTP and HTTPS. Client requests are received by the Apache server which then interacts with the database schema and tables, and sends responses back to the clients. The Laravel PHP Framework is used to handle manipulations of the database, by using a RESTful approach of controlling resources.

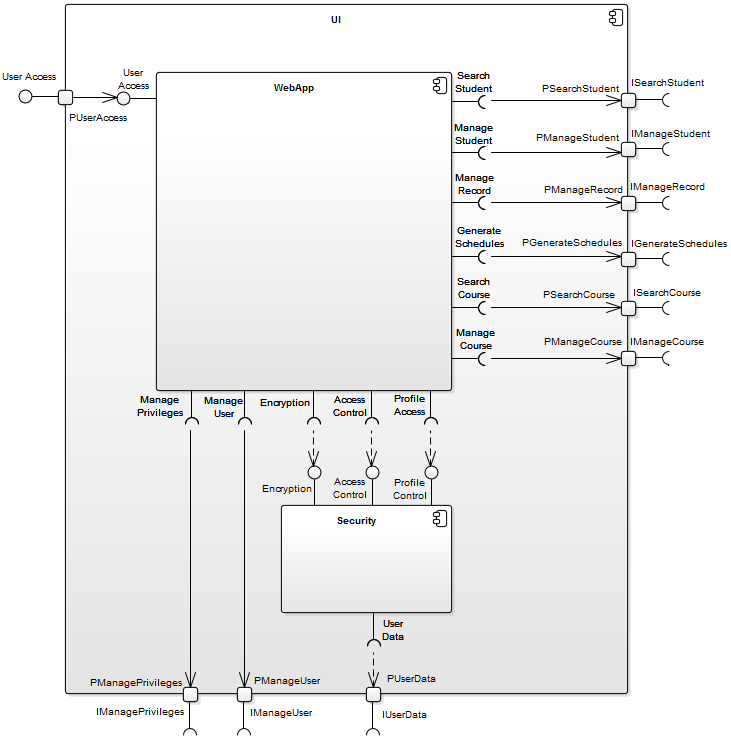
### 8.1.4. **Development View: Component Diagram**

In the 4+1 AVM, the Development View describes the structural aspect of the system. In it modules and subsystems can be represented using Package and Component diagrams, making it the developer’s view of the system. In the following sections the UML Component Diagram notation has been used to model the component of the system, as shown the image below. As it can be seen all the interfaces dependency are properly depicted (except for the validation schedule interface, which is required only in the scheduler component)



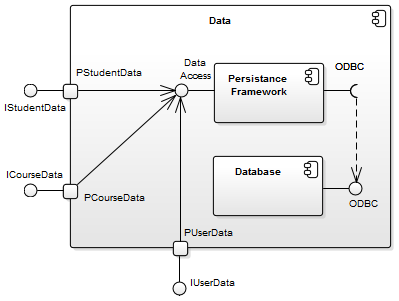
#### Component Diagram “UI”

The UI Component is composed by the components WebApp and Security. The main purpose of the UI component is rendering the web user interface, delegating all the user’s requirements to the corresponding components (i.e. it only has required interfaces). In the other hand, the Security component is continuously invoked to encrypt the information, and validate the access and profile privileges. The Security component provide three interfaces for encryption, access control and profile control. The UI Component represent the view and controller of the Apollo’s MVC architecture.



#### Component Diagram “Data”

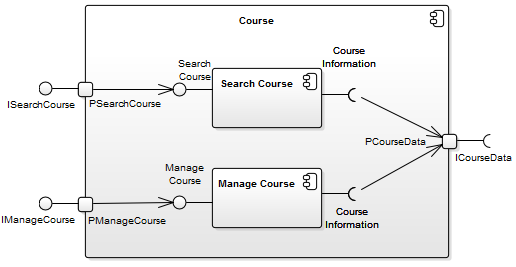
The main purpose of the Data Component is to provide persistence features to the system’s data. It is composed by the persistence framework used in the system and a database. Although it provide three interfaces, all of them are mapped to one interface to access the system’s date. This component is part of the Model in the Apollo’s MVC architecture.



##### 

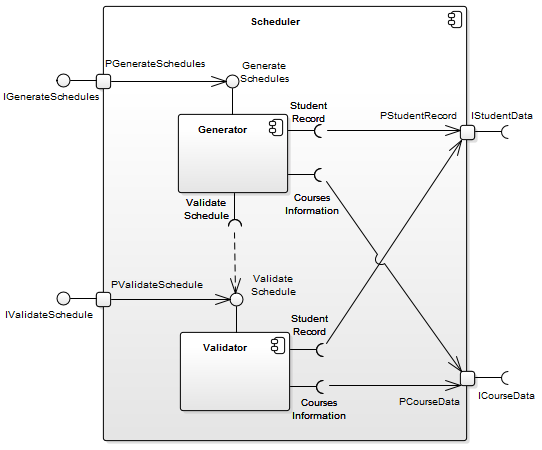
#### Component Diagram “Course”

The Course Component implements the course’s search and manage features, which are at the same time the two components that conform it. These features are available through the two provided interface search course and manage course.

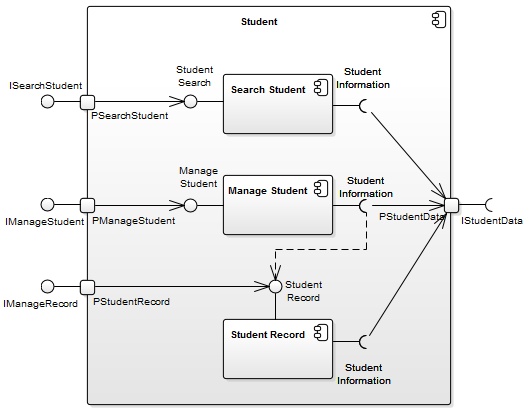


#### Component Diagram “Scheduler”

The Scheduler Component is responsible for the generation and validation of the schedules. To do so it provides two interfaces, but only one is used. To generate and validate the schedules it is required to access information of both the student and courses, so two interfaces are required too.



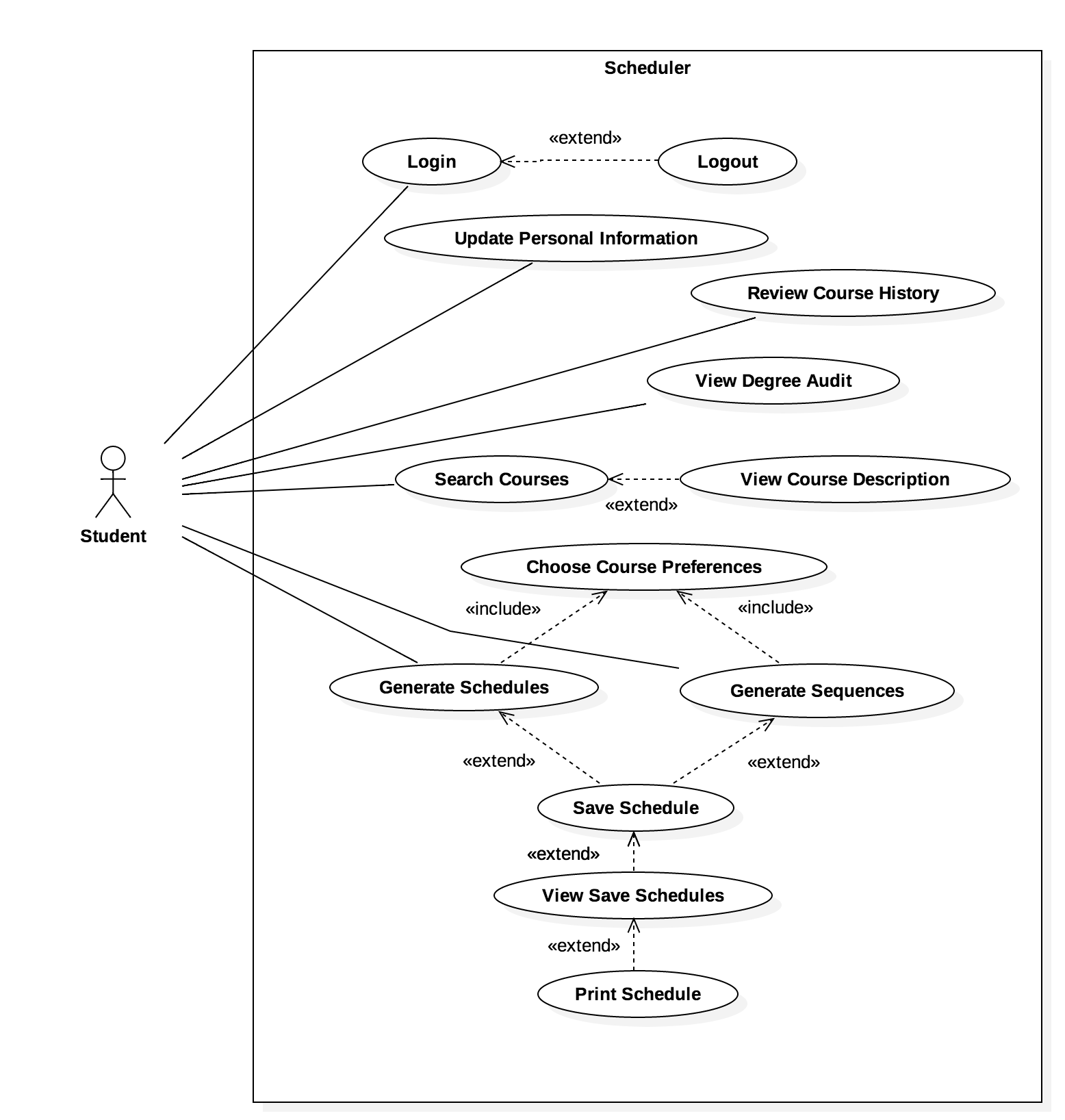
Component Diagram “Student”  
  
The Student component implement the student search and manage features, as well its records. To do so it provide three interfaces, and only one is required in order to access the student information.



### 

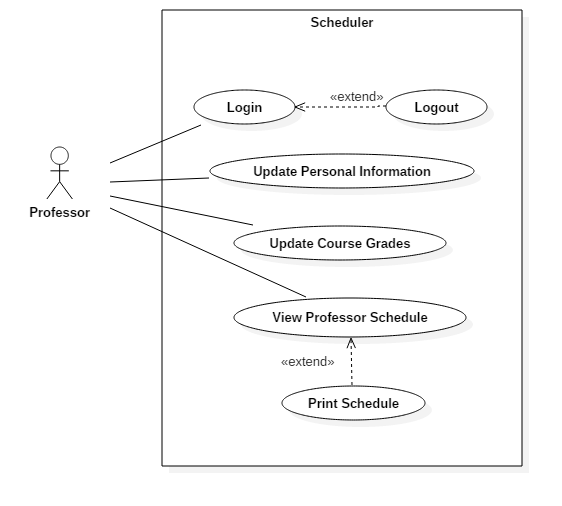
### 8.1.5. **Scenarios: Use Case Diagram**

**Student Scenarios**



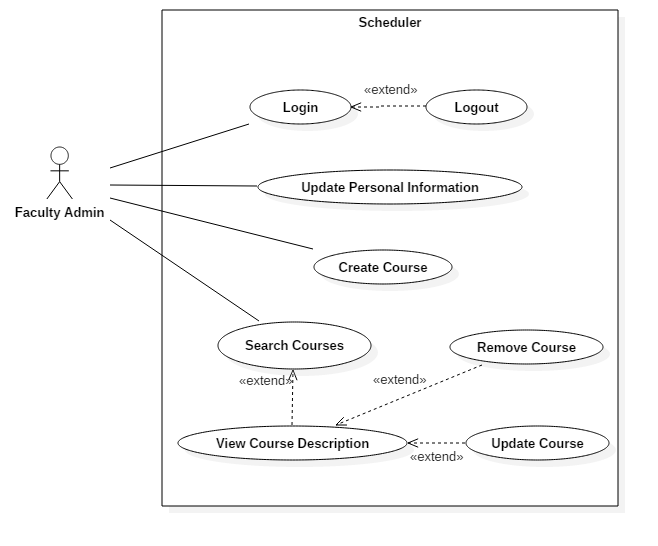
The student constitutes a user of the system who can interact with the scheduler application in all the common scenarios shown in the diagram. The student has to log in to the application and then potentially log out of it.   
  
Once logged in, students can update their personal information, review their course history and degree audit, search for courses, and generate schedules (or entire sequences of schedules) based on their preferences. The students can also save the generated schedules, view and print them.

**Professor Scenarios**



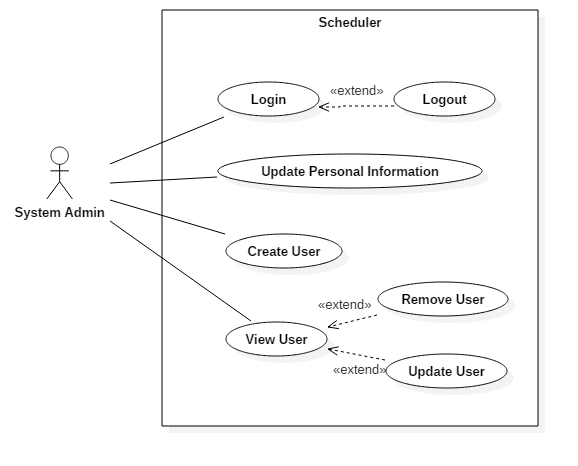
The professor must also log in to use the Apollo system, and can later log out. Professors can update their personal information, update grades in a course they instruct, and view and print their personal schedules.  
  
Note: scoped out since it is not necessary for overall functionality of the system.

**Faculty Administrator Scenarios**



The Faculty Admin user logs into the system and can later log out of it. This user can update his or her personal information, create courses, search for courses and view, update, or remove the courses found.   
  
Note: scoped out due to time constraints.

**System Administrator Scenarios**



The System Admin, as any other user, has to log in to use the Apollo System and can log out. These administrators can update their personal information, create new users, view existing users and update or remove them.  
  
Note: scoped out due to time constraints.

## **8.2. Subsystem Interfaces Specification**

The following subsystem interface specifications provide an overview of the messages exchanged between components. Each interface specification describes the function calls, parameters used, and rang of invalid and valid values. Please assume that accessor, mutator, and constructor methods are included.

### 8.2.1. Student Subsystem

#### **ISearchStudent**

This has been scoped out from implementation. This allows professors and faculty admin to search for a student and input grades.

#### **IManageStudent**

This allows user to edit student information such as user settings and preferences.

|  |
| --- |
| **Class Student** |
| Description: Student class provides information about the student, including the course record, the name, the preferences and the password |
| Attributes:   * student\_id: int * first\_name: String * last\_name: String * user\_name: String * password: String * studentRecord: CourseRecord List * Preferences: Preferences |
| Methods:   * getID(): int   Returns an integer of the student’s identification   * getFullName: String   Returns a String of the full name’s student   * changePassword(oldPass: String, newPass)   Changes an old password to a new one by passing oldPass and newPass in the parameters and updating the password with newPass |

|  |
| --- |
| **Class Preferences** |
| Description: Preferences provides the student the ability to change his or her schedule preferences |
| Attributes:   * id: int * day\_of\_week\_id: int * start: time * end: time |
| Methods:   * addPreference(day: int, start: time, end: time)   Adds a preference, by providing the day, start time and end time in the parameters   * editPreference(day: int, start: time, end: time)   Edits an already available preference by proving the day, start time and end time in the parameters |

#### **IManageRecord**

This has been scoped out of implementation. This allows for faculty admin to edit a student academic record.

|  |
| --- |
| **Class CourseRecord** |
| Description: CourseRecord contains all courses taken by the student and the final grade they got |
| Attributes:   * Course: Course * Grade: Grade |
| Methods:   * N/A |

|  |
| --- |
| **Class Grade** |
| Description: Grade manages the grades received by the student for all courses taken |
| Attributes:   * Id: int * Value: String |
| Methods:   * N/A |

### 8.2.2. Course Subsystem

#### **ISearchCourse**

This allows users to search for courses being offered. This functionality is handled by the AngularJS framework.

|  |
| --- |
| **Class ScheduledCourse** |
| Description: ScheduledCourse is a general class that provides courses scheduled for each session. |
| Attributes:   * id: int * sessionID: int * course: Course |
| Methods:   * N/A |

#### **IManageCourse**

This has been scoped out from implementation. This allows system admins and faculty admins to adjust course information.

|  |
| --- |
| **Class Course** |
| Description: Course will provide the system with the courses available for the student. |
| Attributes:   * id: int * name: String * credits: double * description: String * faculty: Faculty * requisites: Course List |
| Methods:   * getname(): String   Returns a string of the name of the course   * getFaculty(): int   Returns an integer of the faculty of the course   * getCredits(): double   Returns a double of the credits of the course   * getRequisites(): Course List   Returns a Course List of the requisites of a given course |

|  |
| --- |
| **Class Faculty** |
| Description: Faculty provides the name of the faculty that courses will utilize. |
| Attributes:   * id: int * name: String |
| Methods:   * N/A |

|  |
| --- |
| **Class TimeSlot** |
| Description: TimeSlot provides specific information for a course offered in one session. |
| Attributes:   * id: int * scheduledCourse: ScheduledCourse * room: String * timeStart: Date * timeEnd: Date * dayOfWeekID: int * courseTypeID: int |
| Methods:   * getScheduledCouseID(): int   Returns an integer of a Schedule Course for its identification   * getCourseInfo(course: Courses): void   Returns a void in which its parameters is course, where it provides a list of  Information about the course   * isFull(): Boolean   Returns a Boolean to specify if the course is full or not   * addStudent(student: Student)   Adds a student in the selected course, using a parameter student   * addSection(room: String, timeStart:Date, timeEnd:Date, dayOfWeekID:int)   Adds a desired section of the course, by providing the room, the time start, the time end and the day of the week |

# 9**. DETAILED DESIGN**

## **9.1. Detail Design Diagrams & Unit Description**

The following section will detail the subsystems currently employed in our system by the use of class diagrams, as well discuss the rationale behind the classes contained in each subsystem and their function within the system as a whole.

### 9.1.1. Student Subsystem



The student subsystem manages all the information in regards to the student user. The Student class stores all general information regarding a student, such as their name and login information. From this class we have the Student Record class which stores the student’s academic history including Grade, and the Preferences class which will store the student’s overall preferences regarding how they wish their schedule to be made. In designing this subsystem, the idea was to have all the information pertinent to each student available.

**Unit Descriptions**

|  |
| --- |
| **Class Student** |
| Description: Student class provides information about the student, including the course record, the name, the preferences and the password |
| Attributes:   * student\_id: int * first\_name: String * last\_name: String * user\_name: String * password: String * studentRecord: CourseRecord List * Preferences: Preferences |
| Methods:   * getID(): int   Returns an integer of the student’s identification   * getFullName: String   Returns a String of the full name’s student   * changePassword(oldPass: String, newPass)   Changes an old password to a new one by passing oldPass and newPass in the parameters and updating the password with newPass |

|  |
| --- |
| **Class CourseRecord** |
| Description: CourseRecord contains all courses taken by the student and the final grade they got |
| Attributes:   * Course: Course * Grade: Grade |
| Methods:   * N/A |

|  |
| --- |
| **Class Grade** |
| Description: Grade manages the grades received by the student for all courses taken |
| Attributes:   * Id: int * Value: String |
| Methods:   * N/A |

|  |
| --- |
| **Class Preferences** |
| Description: Preferences provides the student the ability to change his or her schedule preferences |
| Attributes:   * id: int * day\_of\_week\_id: int * start: time * end: time |
| Methods:   * addPreference(day: int, start: time, end: time)   Adds a preference, by providing the day, start time and end time in the parameters   * editPreference(day: int, start: time, end: time)   Edits an already available preference by proving the day, start time and end time in the parameters |

### 9.1.2. Courses Subsystem



The course subsystem manages all the information regarding the courses available.The Courses class manages the master course list, and includes general information such as the the name of each course, the course description, the amount of credits it is worth, and the requisites for each course. The Faculty class organizes which courses are available per faculty. The ScheduledCourse class utilizes the information in the master course list and lists all the courses available for each session. Time Slot stores all the pertinent information regarding all the courses that are available in a session, such as if the course is full, the times it is offered in the session in question, as well as the sections available for each course. The rationale behind these choices is that the master course list’s purpose serves to keep track of all courses offered by the program and faculty, and store information that remains unchanged about each course, such as the amount of credits it is worth, the prerequisites for the course, amongst other general information. Subsequently, the Scheduled Courses class would take care of checking which classes are offered in which session, while Time Slots would further utilize this information, and store what sections are available, what time they are offered at, and if the class is full.

#### **Unit Descriptions**

|  |
| --- |
| **Class Course** |
| Description: Course will provide the system with the courses available for the student. |
| Attributes:   * id: int * name: String * credits: double * description: String * faculty: Faculty * requisites: Course List |
| Methods:   * getname(): String   Returns a string of the name of the course   * getFaculty(): int   Returns an integer of the faculty of the course   * getCredits(): double   Returns a double of the credits of the course   * getRequisites(): Course List   Returns a Course List of the requisites of a given course |

|  |
| --- |
| **Class Faculty** |
| Description: Faculty provides the name of the faculty that courses will utilize. |
| Attributes:   * id: int * name: String |
| Methods:   * N/A |

|  |
| --- |
| **Class ScheduledCourse** |
| Description: ScheduledCourse is a general class that provides courses scheduled for each session. |
| Attributes:   * id: int * sessionID: int * course: Course |
| Methods:   * N/A |

|  |
| --- |
| **Class TimeSlot** |
| Description: TimeSlot provides specific information for a course offered in one session. |
| Attributes:   * id: int * scheduledCourse: ScheduledCourse * room: String * timeStart: Date * timeEnd: Date * dayOfWeekID: int * courseTypeID: int |
| Methods:   * getScheduledCouseID(): int   Returns an integer of a Schedule Course for its identification   * getCourseInfo(course: Courses): void   Returns a void in which its parameters is course, where it provides a list of  Information about the course   * isFull(): Boolean   Returns a Boolean to specify if the course is full or not   * addStudent(student: Student)   Adds a student in the selected course, using a parameter student   * addSection(room: String, timeStart:Date, timeEnd:Date, dayOfWeekID:int)   Adds a desired section of the course, by providing the room, the time start, the time end and the day of the week |

# 

# 

# 10**. DYNAMIC DESIGN SCENARIOS**

A full dynamic design of the use cases for “Generate Schedule” and “Save Preferences” use cases. This includes the system sequence diagram and the full sequence diagram for each use cases.

## **10.1. Generate Schedule**

The generate schedule use case, is the scenario when a student user wishes to create a schedule. The student navigates to the scheduler page and chooses the courses he or she wishes to take. The scheduler will create the schedule when the student runs the scheduler.

### 10.1.1. **System Sequence Diagram**

This is the system sequence diagram for the scenario where the student wishes to generate a schedule.



### 10.1.2. Sequence Diagram

This is the sequence diagram for the scenario where a student wishes to generate a schedule.

generateSchedule_sequenceDiagram.png

### 

### 10.1.3. **Contracts**

Generate schedule contracts go with use case UC09.

|  |  |
| --- | --- |
| **Contract C09.01: viewScheduler** | |
| Operation | viewScheduler() |
| Cross References | Use Case: Generate Schedules |
| Preconditions | User is logged in. |
| Postconditions | Generate Schedules page was displayed. |

|  |  |
| --- | --- |
| **Contract C09.02: searchCourse** | |
| Operation | searchCourse(course) |
| Cross References | Use Case: Generate Schedules |
| Preconditions | User is on the Generates Schedule page. |
| Postconditions | -An instance c of Course was created.  -c was associated with the Course the user searched for.  -Course list based on search was displayed. |

|  |  |
| --- | --- |
| **Contract C09.03: selectCourse** | |
| Operation | selectCourse(section) |
| Cross References | Use Case: Generate Schedules |
| Preconditions | User is on the Generate Schedules page. |
| Postconditions | -An instance c of Course was created.  -c was associated with the Course selected by the user  -Course selected was displayed |

|  |  |
| --- | --- |
| **Contract C09.04: runScheduler** | |
| Operation | runScheduler() |
| Cross References | Use Case: Generate Schedules |
| Preconditions | Courses have been selected. |
| Postconditions | -An instance p of Preferences was created  -p was associated with current user Preferences  -Multiple schedules displayed based on Preferences |

|  |  |
| --- | --- |
| **Contract C09.05: changePreferences** | |
| Operation | changePreferences() |
| Cross References | Use Case: Generate Schedules |
| Preconditions | User has selected to edit preferences. |
| Postconditions | -An instance p of Preferences was created  -p was associated with the new Preferences the user wanted  -p modified the current Preferences stored  -Multiple schedules displayed based on new Preferences |

## **10.2. Save Preferences**

# 

The save preferences use case describes the scenario where the student wishes to make modifications to his or her scheduling preferences.

### 10.2.1. **System Sequence Diagram**

This is the system sequence diagram for the scenario where the student wishes to save schedule preferences.

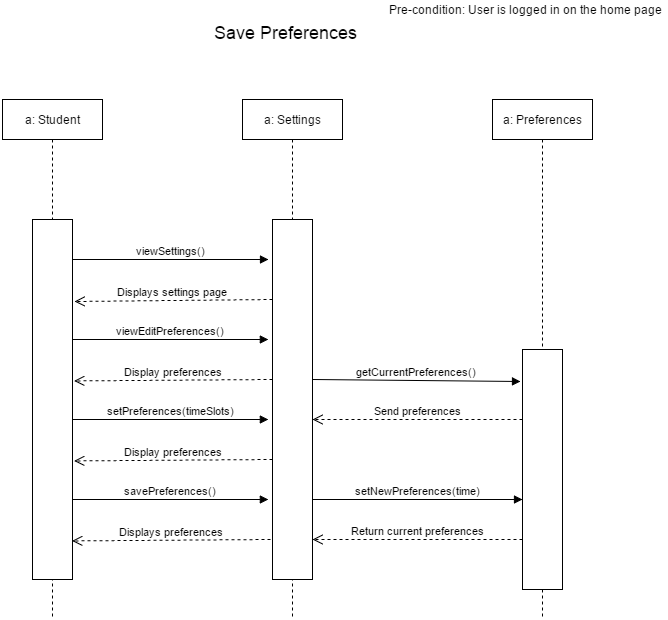
# 

### 

### 

### 10.2.2. **Sequence Diagram**

This is the sequence diagram that describes the scenario when a student wishes to modify his or her scheduling preferences.



### 

### 10.2.3. **Contracts**

Save preferences contracts go with use case UC06.

|  |  |
| --- | --- |
| **Contract C06.01: viewSettings** | |
| Operation | viewSettings() |
| Cross References | Use Case: Save Preferences |
| Preconditions | User is logged in. |
| Postconditions | User Settings page is displayed. |

|  |  |
| --- | --- |
| **Contract C06.02: viewEditPreferences** | |
| Operation | viewEditPreferences() |
| Cross References | Use Case: Save Preferences |
| Preconditions | User is on the settings page. |
| Postconditions | - A Preference instance p was created.  - p was associated with the current user preferences.  - Current user preferences was displayed. |

|  |  |
| --- | --- |
| **Contract C06.03: setPrefs** | |
| Operation | setPrefs(Preference: timeSlots) |
| Cross References | Use Case: Save Preferences |
| Preconditions | User has selected to edit preferences. |
| Postconditions | - A Preference instance timeSlots was created  - timeSlots modified information to new user preferences  - New preferences were displayed |

|  |  |
| --- | --- |
| **Contract C06.04: savePrefs** | |
| Operation | savePrefs() |
| Cross References | Use Case: Save Preferences |
| Preconditions | User is on edit preferences |
| Postconditions | Modified user preferences have been saved |

# 

# 

# 11**. ESTIMATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tasks** | **Previous Cost (Hours) for all members** | **Revised Cost (Hours) for all members** | **Duration (Days)** |
| Member Organisation | 1 | 1 | 2 |
| Initial Planning | 2 | 2 | 1 |
| Deliverable 0 | 4 | 4 | 1 |
| Scheduling | 2 | 2 | 1 |
| Resource Evaluation | 5 | 5 | 1 |
| Technical Configuration | 3 | 3 | 2 |
| Domain Model Design | 3 | 3 | 1 |
| Architecture Planning | 5 | 5 | 2 |
| Initial Architecture Design | 3 | 3 | 2 |
| Database Setup | 24 | 24 | 1 |
| Rapid Prototype | 24 | 24 | 1 |
| Data Input | 15 | 15 | 1 |
| Functional Requirements Planning | 10 | 10 | 3 |
| Constraint planning | 5 | 5 | 1 |
| Scoping | 5 | 5 | 1 |
| Initial Front End Planning | 5 | 5 | 3 |
| Estimation 1 | 1 | 1 | 1 |
| Risk Planning 1 | 1 | 1 | 1 |
| Deliverable 1 | 30 | 30 | **24** |
| Architecture Design | 5 | 5 | 3 |
| Front End Planning | 10 | 10 | 3 |
| Front End Design | 10 | **15** | 4 |
| Detailed Design | 10 | 10 | 2 |
| Dynamic Design Scenarios | 20 | **25** | 4 |
| Front End Prototype | 30 | 30 | 2 |
| Estimation 2 | 1 | 1 | 1 |
| Risk Planning 2 | 3 | 3 | 1 |
| Deliverable 2 | 30 | **40** | **30** |
| Implementation Planning | 15 | **20** | 4 |
| Implementation | 40 | **50** | 5 |
| Test Planning | 15 | 15 | 5 |
| Testing | 30 | **45** | 2 |
| Improvement | 15 | **30** | 2 |
| Testing | 10 | **15** | 1 |
| User Manual | 20 | 20 | 2 |
| Final Cost Estimate | 2 | 2 | 1 |
| Deliverable 3 | 40 | **60** | **20** |
| Project Documentation | 80 | **90** | 7 |
| Deliverable 4 | 15 | **20** | **10** |
| **TOTAL** | 549 | **654** | 84 |

# 

# 

# 12**. RISKS**

## **12.1. Use of an unfamiliar framework**

As identified in the previous deliverable as a risk, this issue has been a hindrance in achieving tasks on time. Integrating the two frameworks (Laravel and Angular) that the team members have chosen to work with has been a problem and team members are still working to cope with the issue and get more familiar with Angular framework which is new to most of the team members. We minimized the risks as much as possible by automating as many tasks as we could such as script injection and installation of the various dependencies.

## **12.2. Different levels of programming knowledge**

The team has managed this problem well. Tasks related to documentation were assigned to people with low programming knowledge and programming tasks were assigned to team members who have good knowledge in programming. Slack is used for communication between team members if there is any confusion about any task. Although there is problem in integrating the two frameworks together, overall it has been easy to implement the design using the programming language and frameworks that we have chosen.

## **12.3. Schedule generating issues**

Keeping this issue in mind as previously mentioned, our team has managed to minimize this risk as low as possible by listing all the courses and their pre-requisites down and applying constraints where needed to avoid this issue.

One problem the team has encountered was acquiring the times and sections for every class given in Concordia. Several approaches were proposed such as creating a form and manually entering the information for every class, or creating a script that would scrape the information we need from the Student Academic Services website.

## **12.4. Security Issues**

The design hasn’t affected the previously assessed security risk. This risk will always exist even after a successful implementation of our project as handling personal data about students in a database always carries some risk.

# 

# 

# 

# 13**. RAPID PROTOTYPING**

We have used the RESTful API using Laravel to connect the backend to the frontend. We can create, delete, list all courses and show specific courses and faculties. We display this information using Angular.js, and use the Angular Material framework for frontend.

**13.1. Frontend Wireframe Design**

In order to begin developing the website pages, the website wireframe was made so that the design layout was clear and available for the developers to implement from. A high-fidelity wireframe was chosen as this provided in depth detail that closely resembles what the site should look like. Adobe Photoshop was used to create the wireframe.

### 13.1.1. **Home Page without Navbar**



### 

### 

### 

### 

### 

### 

### 

### 

### 13.1.2. **Home Page with Navbar**



### 13.1.3. **My Academics Page**

### apollo_myAcademics.jpg

### 

### 13.1.4. **Saved Schedules Page**



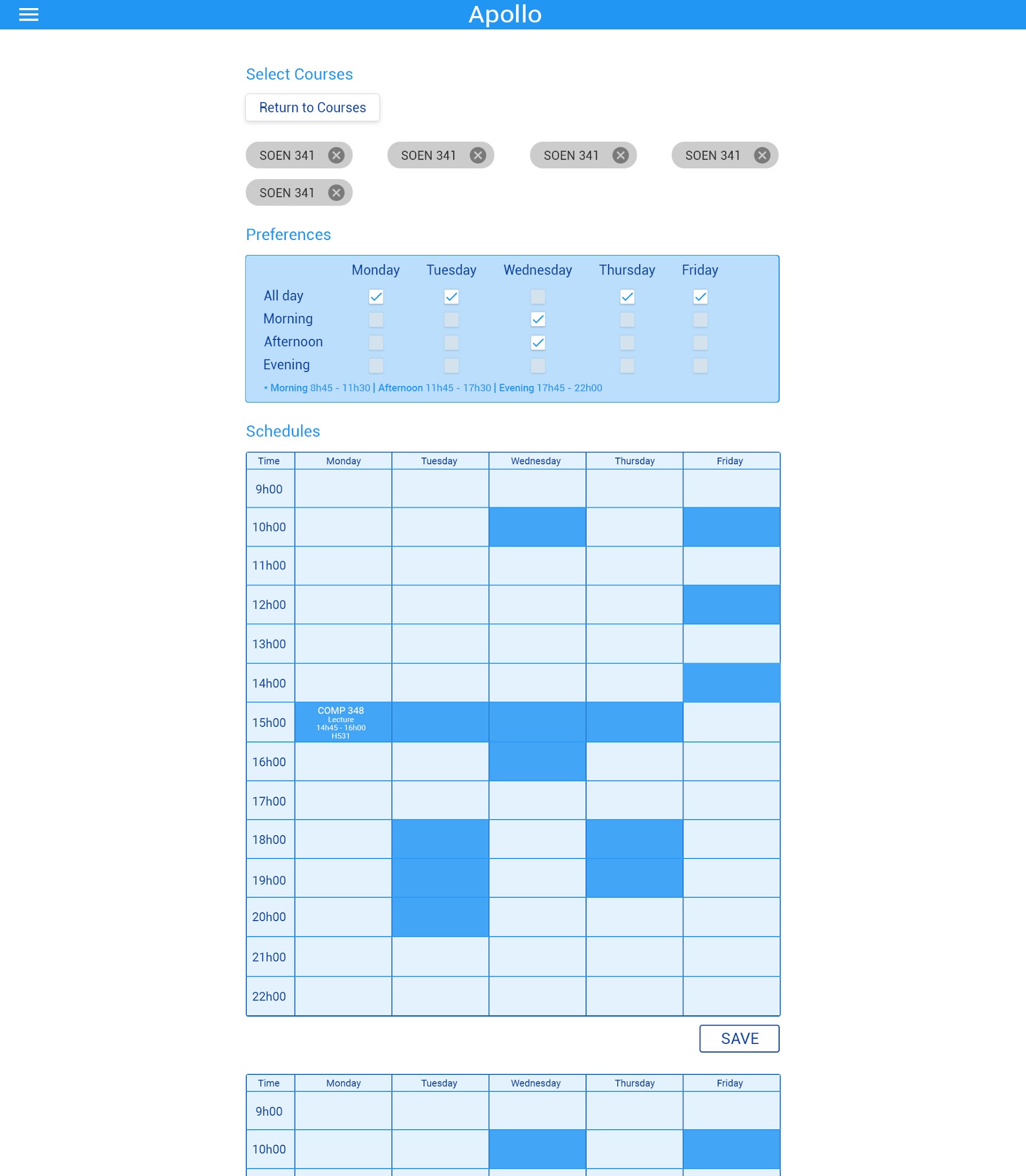
### 

### 13.1.5. **Select Courses Page**



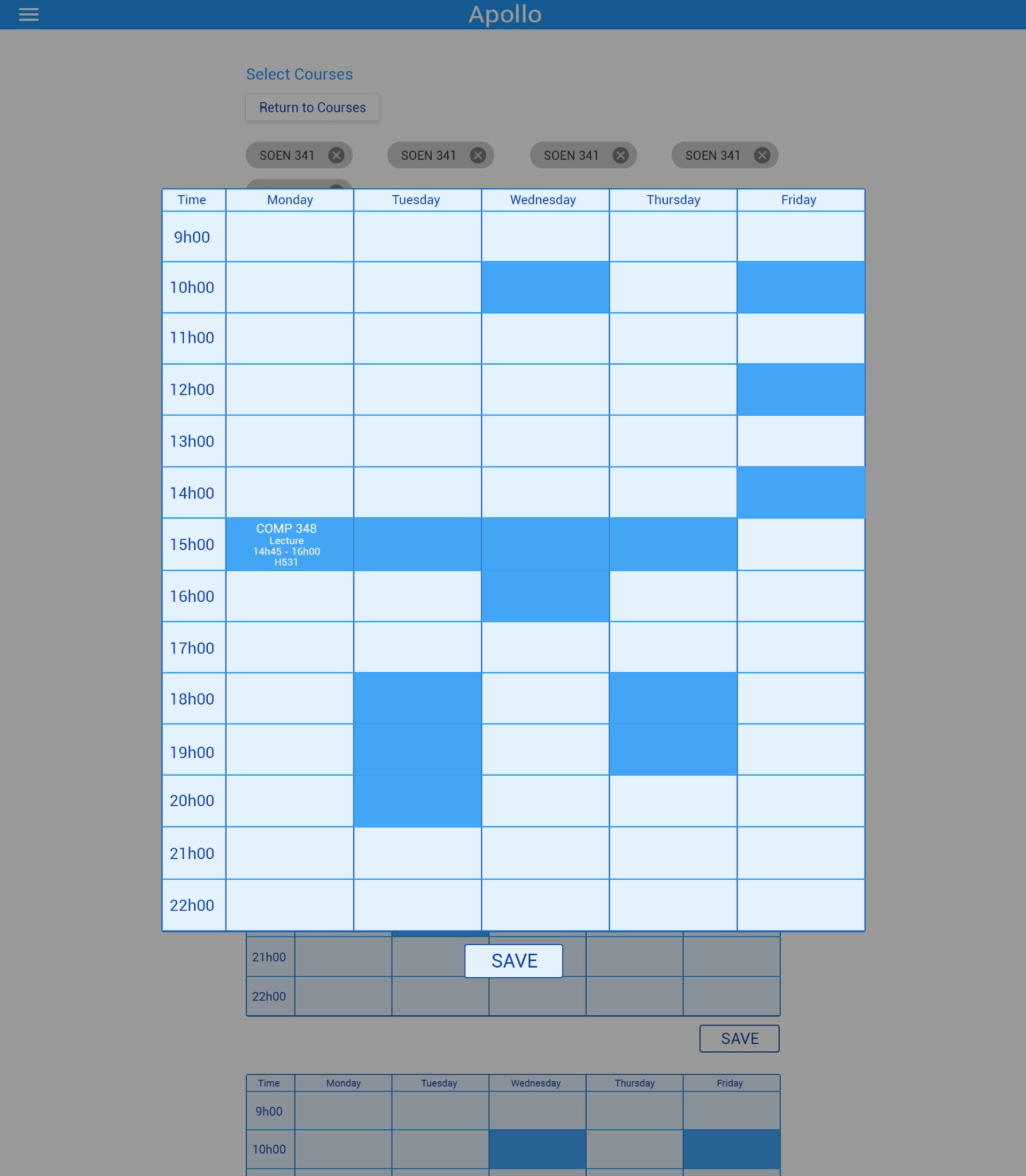
### 

### 13.1.6. **Generated Schedules Page**

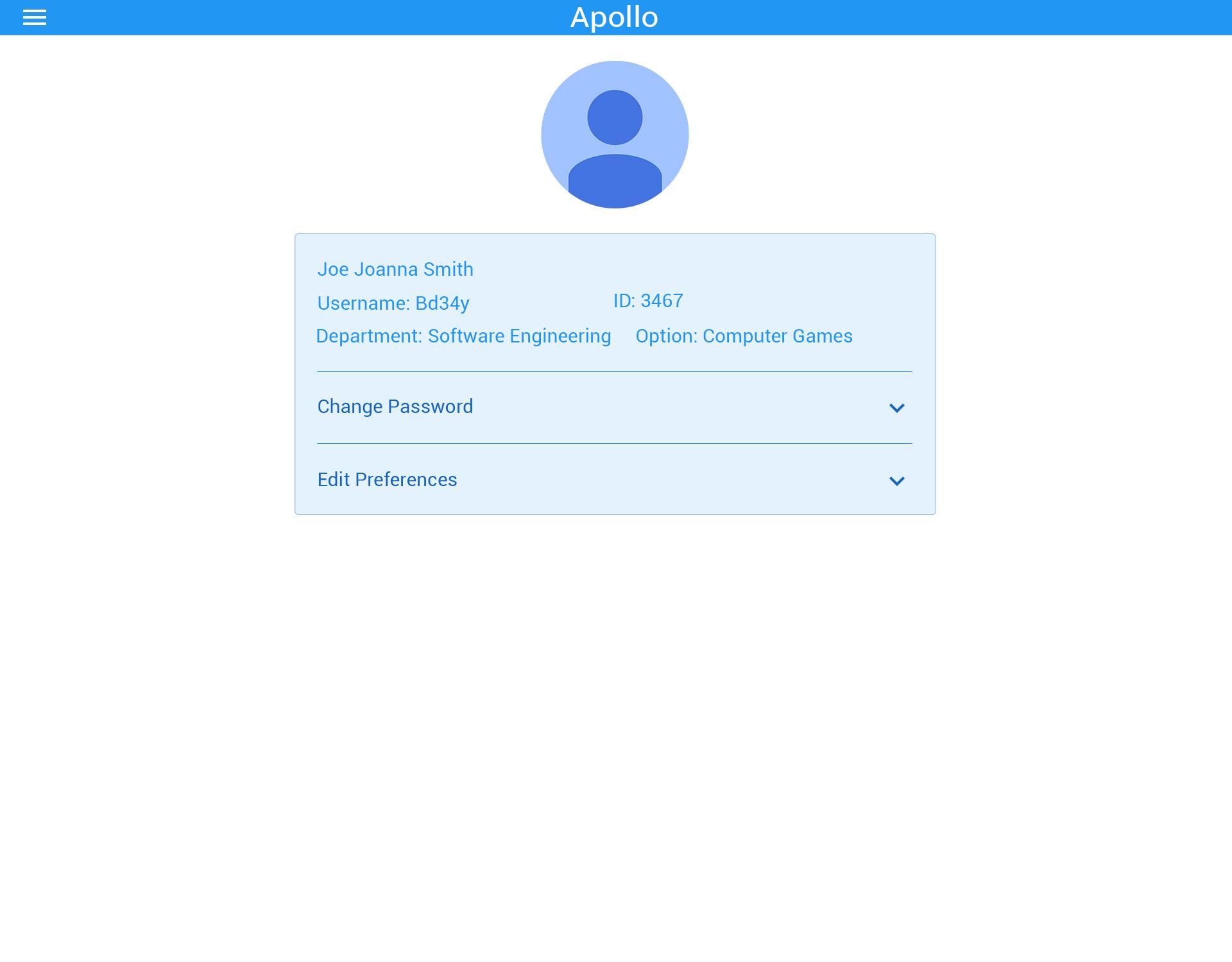


### 

### 13.1.7. **Generated Schedules Overlay View**

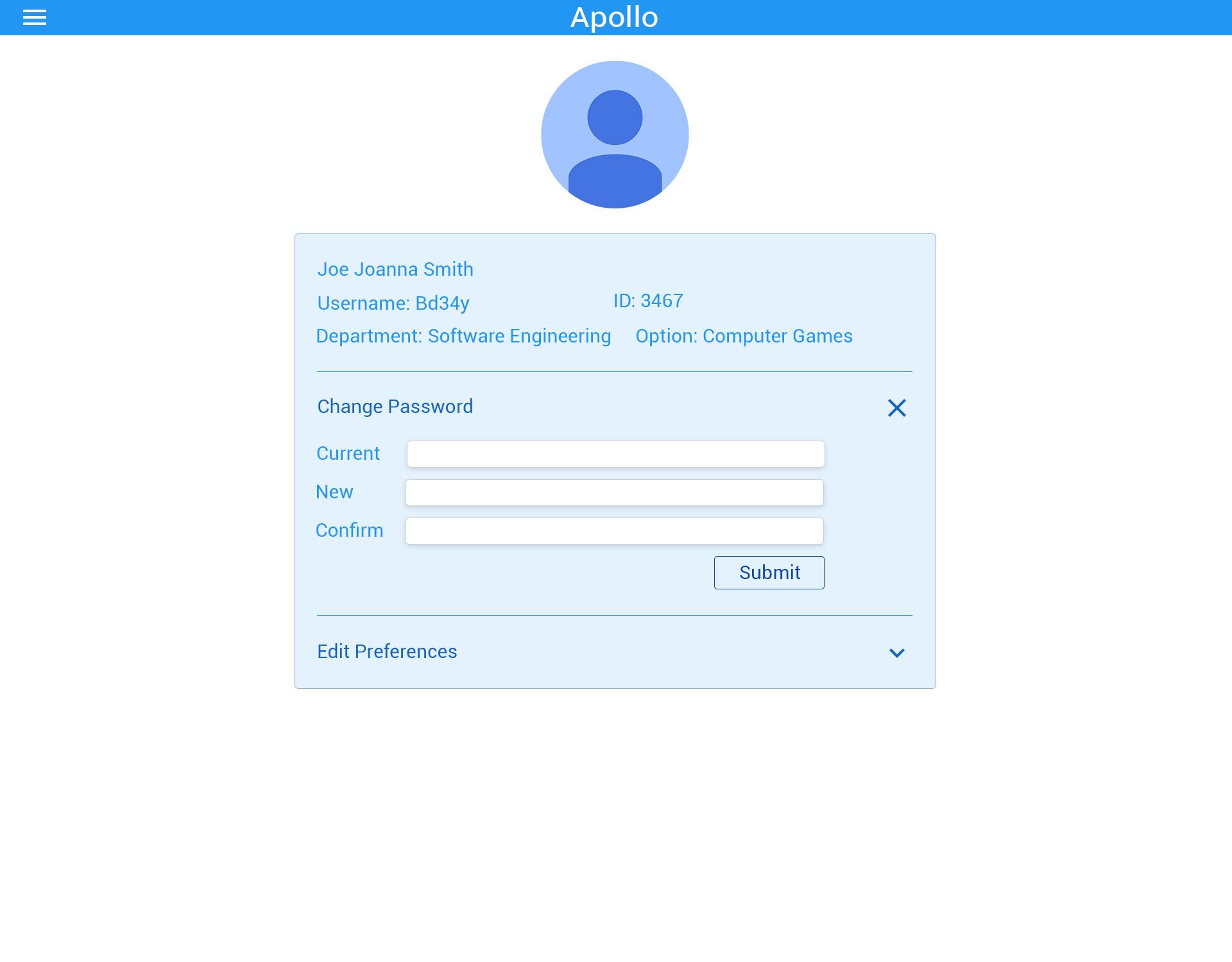


### 13.1.8. **User Settings Page**



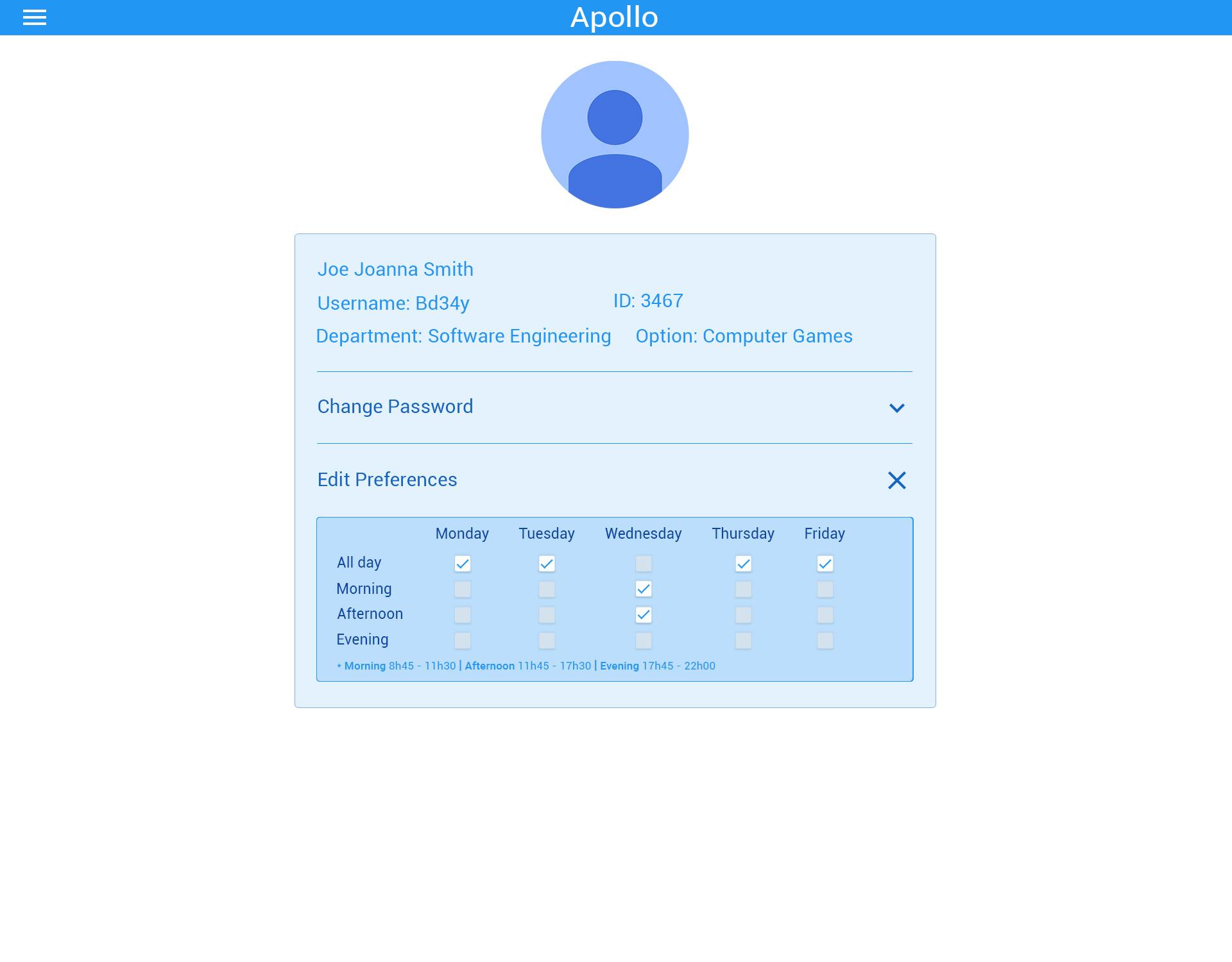
### 

### 13.1.9. **User Settings Page - Change Password**



### 

### 13.1.10. **User Settings Page - Edit Preferences**



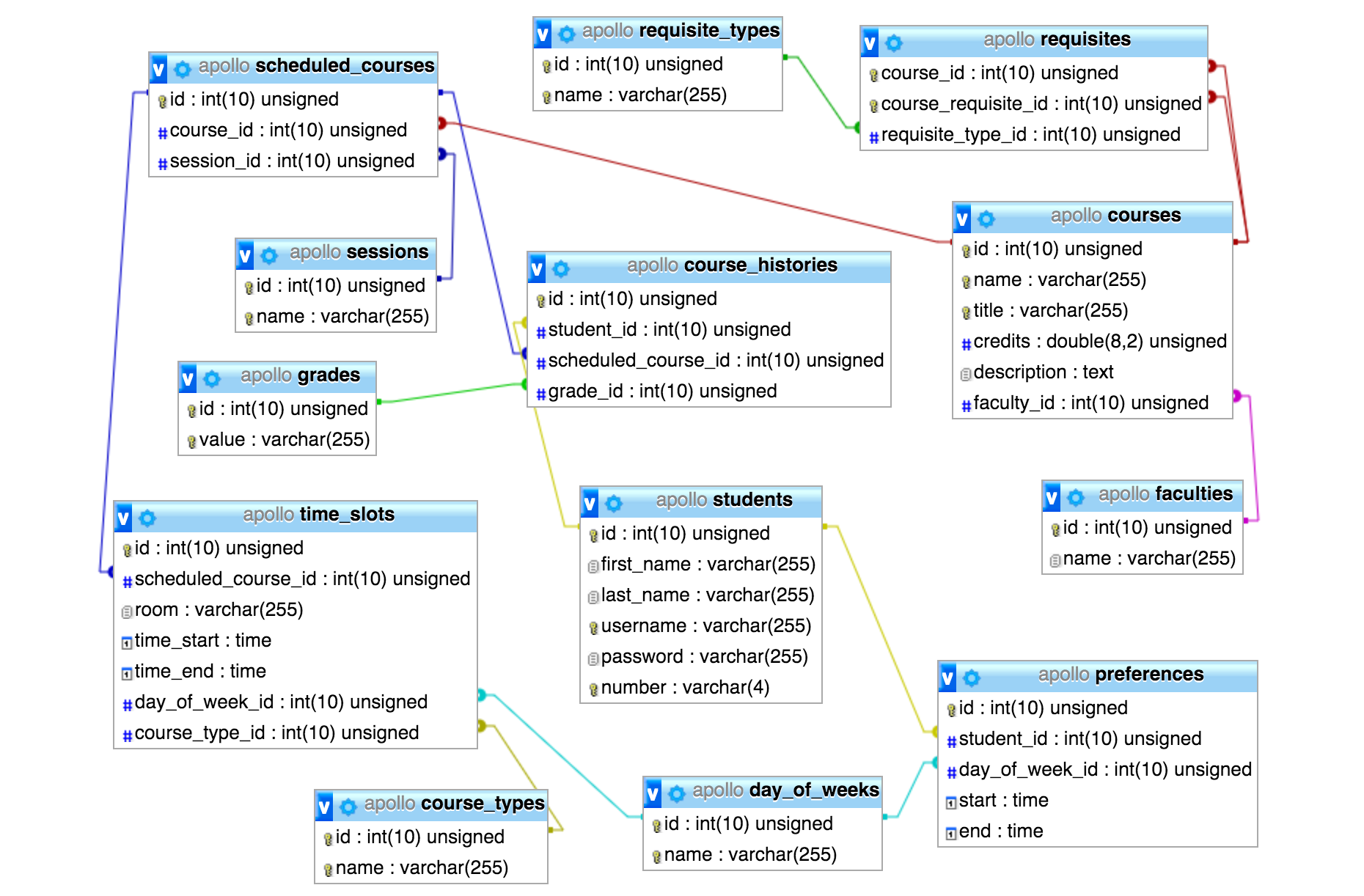
### 13.1.11. **Current Schedule View**

Note: This page has been scoped out from implementation.



## **13.2. Backend**

The following diagram shows the structure of the current database of the website. The data has been taken from Concordia’s course list.



## **13.3. Website**

**URL**: <http://apollo.matthewteolis.com/>

**Username**: soen

**Password**: 341

# 

# 

# 14**. TESTING REPORT**

This section contains all relevant information on tests that were performed on the Apollo scheduler. The first sub-section describes the breadth of the testing that was performed on the web application. The following sub-section describes the testing procedures performed on the software.

## **14.1. Testing Coverage**

This section describes what was tested on the web-application. It incorporates the description of tested items and untested items of interest. A list of tested items are included with their respective test case and the importance of testing each item. A list of untested items of interest are included with their explanation on how it could be tested and the importance of their testing.

### 14**.1.1. Tested Items**

The items which were tested were based on the functional requirements listed in Deliverable 1, Section 3.1. However, some of the requirements specified in Deliverable 1 were scoped out due to time constraints, and scoping down the type of users to include only the student user. Priority was thus given to core requirements needed for the application to function as needed.

The following features were maintained within the application and were subsequently tested as well.

|  |  |  |  |
| --- | --- | --- | --- |
| Login | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC01 | High | This feature was tested as it is necessary for a user to be able to enter the application using their own credentials, else it would be rendered useless as the user cannot enter the system. | 1.1, 1.2, 1.3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Logout | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC02 | High | This was tested as once a user is done utilizing the application, they should be able to log out of the application. If not, there is a security risk as well as tying up resources with the system. | 2.1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Choose Course Preferences | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC06 | High | This feature was tested as a user should be able to set when they would prefer their courses to occur, or if they do not care for the timing. This information is required to generate the appropriate schedules. | 6.1, 6.2, 6.3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Search Courses | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC07 | High | This feature was tested as a user has to absolutely be capable of searching for the course they want in order to add it to their schedule, else making a schedule would be impossible. | 7.1, 7.2 |

|  |  |  |  |
| --- | --- | --- | --- |
| View Course Description | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC08 | Low | This feature was tested as a student should be able to view the description of the course they are taking. However, its priority is not that high as it would not hinder the process of adding a course to the schedule. | 8.1, 8.2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Generate Schedule | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC09 | High | This feature was tested as a student should absolutely be able to generate a schedule based on the courses they have selected for the semester in question, else the core function of the application would be rendered useless. | 9.1, 9.2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Generate Sequence | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC10 | High | This feature was tested as a student should be able to generate the entire sequence of their program based on the courses they chose per semester, and this is a core functionality of the system. | 10.1, 10.2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Save Schedule | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC11 | High | This feature was tested as a user should be able to save all the schedules they have generated, else they would not be able to retain any of the generated schedules. | 11.1 |

|  |  |  |  |
| --- | --- | --- | --- |
| View Saved Schedule | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC12 | High | This feature was tested as a user should be able to view all the schedules they have saved, and thus be able to choose which schedule is to their preference. | 12.1, 12.2 |

|  |  |  |  |
| --- | --- | --- | --- |
| View Student Schedule | | | |
| Related Use Case | Priority | Comments | Related Test Cases |
| UC13 | High | This feature was tested as a user should be able to view their current schedule, else it would make choosing courses for the next semesters difficult and compromise the convenience of the system as a result. | 13.1 |

Additionally, unit testing was done on on two functions chosen within the Apollo application. The functions chosen were addCourse(course), which serves to add a course to the schedule, and duration, which determines the length of a class in a day.

Requirement testing was also undertaken to check the functionality of each use case that was included in the Apollo system, as listed above. Each was checked with test inputs to ensure that everything was functioning as intended, and any bugs detected were noted down in order to take care of it.

### 14**.1.2. Untested Items of Interest**

Below is a list of omitted features that have been scoped out from the functional requirements outlined in Deliverable 1, Section 3.1. Since they were omitted, they were not included in the application, nor were they tested. Had they been included, they would have been subjected to requirements testing as were the included tested items, and tested to see if each feature fails or passes.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Related Use Case | Related Test Cases | Reason for Omission |
| Update Personal Information | UC03 | 3.1, 3.2, 3.3, 3.4, 3.5 | Omitted due to time constraints |
| Review Course History | UC04 | 4.1 | Omitted due to time constraints |
| Update User | UC20 | 20.1, 20.2 | Omitted due to scoping out System Admin |
| View Degree Audit | UC05 | 5.1 | Omitted due to time constraints |
| Print Schedule | UC14 | 14.1, 14.2 | Omitted due to time constraints |
| Create Course | UC15 | 15.1, 15.2 | Omitted due to scoping out Faculty Admin |
| Remove Course | UC16 | 16.1, 16.2 | Omitted due to scoping out Faculty Admin |
| Update Course | UC17 | 17.1 | Omitted due to scoping out Faculty Admin |
| Create User | UC18 | 18.1 | Omitted due to scoping out System Admin |
| Remove user | UC19 | 19.1, 19.2 | Omitted due to scoping out System Admin |
| View User | UC21 | 21.1, 21.2 | Omitted due to scoping out System Admin |
| Update Course Grades | UC22 | 22.1 | Omitted due to scoping out Proffesor user |
| View Professor Schedule | UC23 | 23.1, 23.2 | Omitted due to scoping out Proffesor user |

Additionally, due to time constraints, we were unable to test some important features of our website, especially in terms of unit testing. For the purposes of this report, unit testing was only carried out on 2 units. However, the application would benefit from writing stubs and drivers for each function, class, and subsystem within the system. As a result, it would become much more apparent where code could be improved or is failing, and could be corrected, providing for a more robust system as a result. The unit tests for all the functions within the application would be carried out in a similar manner to the tests carried out for the two test units contained within this report.

Another test of interest would be supplementary security tests. Currently, the laravel framework used for this application has its own mechanisms to prevent SQL injections from occurring. However, it would be useful to provide additional security testing using other tools so as to ensure the application is completely secure, as the database contains a lot of private information about its users that should not be compromised.

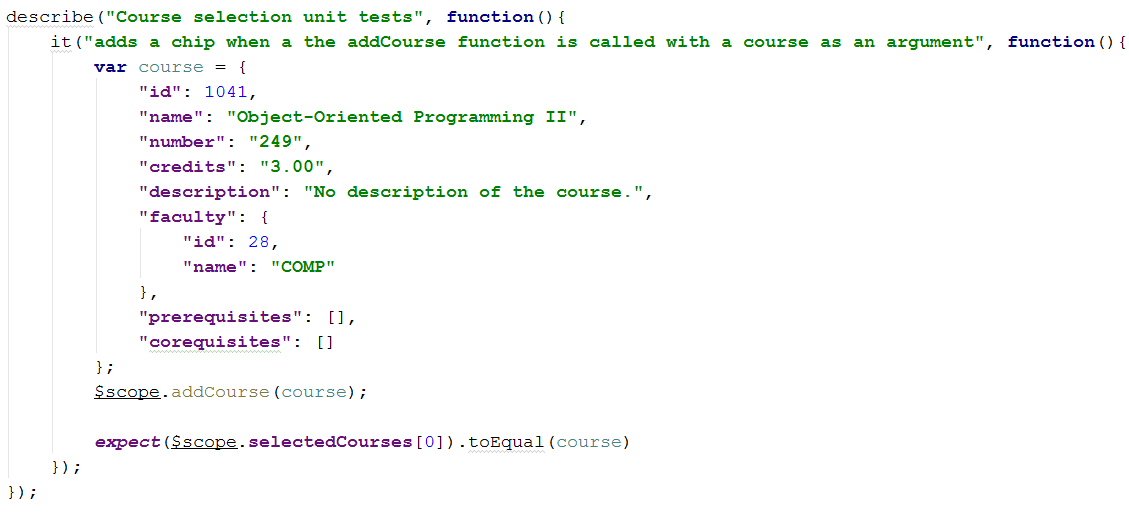
## **14.2. Test Cases**

This section describes all the test cases that are applied to the desired tested items in different aspects of our system, by using different testing techniques, such as creating a testing class. This section includes other subsections which provide additional testing perspective. The tests would be well formatted and well reproducible with expected results, to truly understand them.

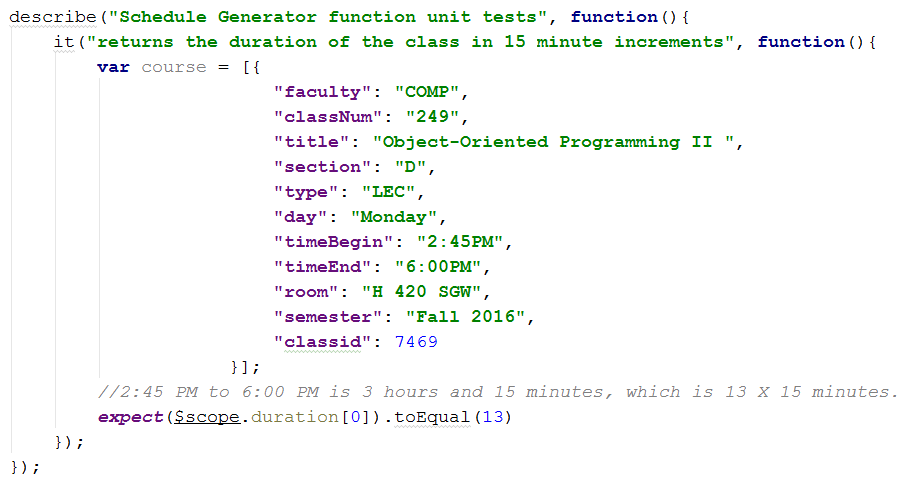
### 14**.2.1. Unit Testing**

### **Function addCourse(course)**

For unit testing, we test the functionality for one function at a time. Here, we provided two function form the class generated-schedule. The first function tests if the addcourse actually adds the selected course. It thus creates a chip interface is modeled by the array $scope.selectedcourses[ ].



**Function duration**  
  
This functions tests if the duration of the course corresponds well to an increment of 15 minutes. Since our scheduler is divided into increments of 15 minutes, a class of, for example, an hour will take 4 x 15 minutes.



In this test case, we check if a class from 2:45 PM to 6:00 PM is in fact equal to 13 x 15 minutes. This helps the *Scheduler* decide the row span of the cell, corresponding to that course.

### 

### 14**.2.2. Requirements Testing**

The following includes a list of test cases from concrete system usage scenarios and the expected system response.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC01 - Login** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 1.1 | Validate user credentials and allow user specific privileges to the system. | User is logged in. | | Pass |  |  |
| 1.2 | System gets an invalid username/password. | “ Please enter a valid username or password” | | Fail | 2,3 |  |
| 1.3 | System gets an input that is not in the database | “ Your username is not in the database” | | Fail | 2,3 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC02 - Logout** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 2.1 | Terminate the session of the user interacting with the  system, no longer provide the user his or her personal  data. | User is logged out. | | Pass |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC03 - Update Personal Information** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 3.1 | User can change his/her username. | User’s username is changed | | N/A |  | This function has been scoped out |
| 3.2 | User can change his/her password | User’s password i changed | | N/A |  | This function has been scoped out |
| 3.3 | User can change his/her address | User’s address is changed | | N/A |  | This function has been scoped out |
| 3.4 | User can change his/her email address | User’s email address is changed | | N/A |  | This function has been scoped out |
| 3.5 | User can change his/her phone number | User’s phone number is changed | | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC04 - Review Course History** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 4.1 | Review student’s course history which includes the semesters taken and student’s grades. | User sees course history. | | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC05 - View Degree Audit** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 5.1 | Review list of completed courses and remaining  courses to be taken in order to graduate. | User sees course audit list. | | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC06 - Choose Course Preferences** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 6.1 | Class time preference | Class time preferences are saved. | | Pass |  |  |
| 6.2 | Class day preference | Day preferences are saved | | Pass |  |  |
| 6.3 | Semester preference | Semester preferences are saved. | | Pass |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC07 - Search Courses** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 7.1 | User inputs valid course number and id. | The searched course is displayed along with its course code and name, and its description. | | Pass |  |  |
| 7.2 | User inputs invalid course number or id. | “The course specified is not in our directory. Please try again.” | | Pass |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC08 - View Course Description** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 8.1 | See course instructor, sections, session time,  requisites. | Course information is displayed. | | Pass |  |  |
| 8.2 | There is no course description for a course. | “No course description” | | Pass |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC09 - Generate Schedules** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 9.1 | User can view the different semesterly schedules  offered to him or her. | One or more schedules are generated  according to user’s specifications. | | Pass |  |  |
| 9.2 | Successfully checks for requirements before generating | Checks if the student has satisfied all prerequisite requirements | | Fail | 9 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC10 - Generate Sequences** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 10.1 | User can view the different course sequences offered  for their remaining semesters until graduation. | One or more sequences are generated  according to user’s specifications. | | N/A |  | This function has been scoped out |
| 10.2 | Checks if all preceding semesters fulfill the prerequisite requirements for the upcoming semester. | Checks for each semester if its requirements have been fulfilled in the previous semesters. | | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC11 - Save schedule** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 11.1 | User can save generated schedule. | Schedule is saved to user profile. | | Pass |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **UC12 - View Saved Schedules** | | | | | | |
| ID | Description | Expected Output | | Result | Bug ID | Comments |
| 12.1 | Student views previously saved schedules. | Student sees saved schedules. | | Fail | 12 |  |
| 12.2 | Student views current generated schedules. | Student sees generated schedules. | | Pass |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC13 - View Student Schedule** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 13.1 | Student sees their current semester schedule. | Student sees current schedule. | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC14 - Print Schedule** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 14.1 | User prints their current schedule, generated schedule, or saved schedule. | User’s schedule is printed. | N/A |  | This function has been scoped out |
| 14.2 | User has no schedule | “No schedule to print” | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC15 - Create Course** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 15.1 | Faculty Admin creates a new course. | Course is added | N/A |  | This function has been scoped out |
| 15.2 | The course does not exist | “The course does not exist in the directory | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC16 - Remove Course** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 16.1 | Faculty Admin removes a course. | Course is removed. | N/A |  | This function has been scoped out |
| 16.2 | The course does not exist | “The course does not exist in the directory” | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC17 - Update Course** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 17.1 | User edits the description of a course. | The specified course’s information is updated. | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC18 - Create User** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 18.1 | System Admin creates a user of the system. | A new user is created. | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC19 - Remove User** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 19.1 | System Admin removes a user from the system. | The user is removed from the system. | N/A |  | This function has been scoped out |
| 19.2 | The user doesn’t exist | “The user does exist in the database” | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC20 - Update User** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 20.1 | System Admin updates a user’s information. | The user information is updated. | N/A |  | This function has been scoped out |
| 20.2 | The user doesn’t exist | “The user does not exist in the database | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC21 - View User** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 21.1 | Admin searches for a valid student id or name | System Admin sees user information. | N/A |  | This function has been scoped out |
| 21.2 | Admin searches for an invalid student id or name | “No user found” | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC22 - Update Course Grades** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 22.1 | User is logged in and is the professor of the selected course. | Students’ grades in the selected class are updated | N/A |  | This function has been scoped out |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UC23 - View Professor Schedule** | | | | | |
| ID | Description | Expected Output | Result | Bug ID | Comments |
| 23.1 | Professor views the schedule of the course they are teaching. | Professor sees their current schedule. | N/A |  | This function has been scoped out. |
| 23.2 | No courses scheduled for the professor | “No courses to be taught this semester” | N/A |  | This function has been scoped out |

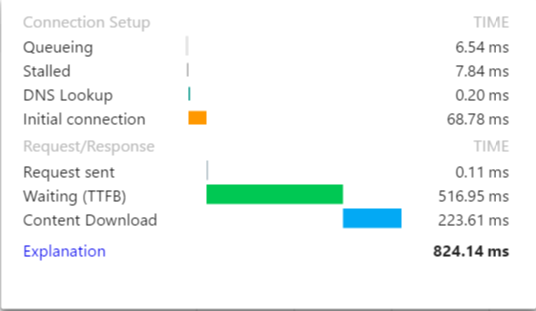
### 14.2.3. Stress Testing

# 

Extreme situations that can affect the application may cause the performance of the software to suffer as a result. In terms of the Apollo application, the situation under which the application can come under stress is when it is subjected to a heavy load in terms of the network in question. As a result, the stress test for the Apollo application would require checking the network capacity and the number of connections it can handle without affecting its performance. Additionally, the times under which the Apollo application may come under high stress can also be anticipated, as it will be expected that more users will attempt to simultaneously connect to the application during times of the year when it is time to schedule for courses for the following semesters.

The Apollo application is currently hosted on GoDaddy. With over 61 million domains under its management, it is the world’s largest ICANN-accredited registrar. With its capability to handle a very large volume of users and connections, having the application hosted on GoDaddy has ensured that the application can handle a high volume of simultaneous connections without affecting the performance of the software, as well as an even greater volume of viewers as GoDaddy only considers a user to be connected while they are actively downloading content from the application [1].

Since the application is hosted on GoDaddy, the information regarding how many connections the application can maintain is available on the GoDaddy account. It revealed that the application can handle 100 concurrent connections without affecting its performance. After testing an API call (a GET courses request) to the server, the average connection duration was approximately 820 milliseconds. This means that the Apollo applications can serve approximately 122 people per second, and 7310 people per minute. Concordia had 3,289 registered students in the engineering department in 2015[12]. We expect our application to be able to run well even when it is heavily used.



### 14.2.4. Security Testing

# 

SQL injection refers to when an injection attack where an attacker can execute malicious SQL statements that can thus control a web application's database. Hence, by accessing a website’s database without permission, the attacker can modify or even erase important data about the website. This, however, is not a problem, since we are using the Laravel framework.

Laravel’s object-related mapping uses parameter binding to avoid SQL injections. Parameter binding protects from SQL injections by ensuring that attackers can’t bypass the query data, which could modify the query’s intent. Let’s consider an example, if in the instance form field we are asked to supply an email-address, but instead we write: ‘abc@example.com’ or 1=1. 1=1 syntax is a simple logic that always evaluates to true. When this is used with “or”, all records will be returned from the table form. Let’s look at another example. If, for instance, we supply an email address again and we write: drop table users, like this: ‘abc@example.com’ ; drop table users; , there would be a dangerous error in the database. If the account is responsible for executing the application queries and has the “drop” privilege, it could destroy and erase all data from the users table. However, Laravel's parameter binding can remove these problems. When it is used, the whole input will be quoted, thus the 1=1 or drop table users will not be executed.

Another attack could be Cross-Site Scripting. Laravel will automatically escape any HTML entities passed along with a new variable, using {{}}. What happens is that if, for example, an attacker passes the following string: My list <script>alert("spam!")</script>. If this is saved in the database without any filters, then an alert display window will show up without the admin’s consent. However, since Laravel provides an escape tag, using {{}}, then it would render a string like this: My list &lt;script&gt;alert("spam !")&lt;/script&gt, preventing the website from displaying an alert window.

# 

# 

# 15**. SYSTEM DELIVERY**

## **15.1. Installation Manual**

The Apollo schedule generator is used online and is compatible with Google Chrome, Firefox, Safari, and Explorer browsers. There is no need to download and install any other software (apart from an internet browser) in order to run the system. However, the client may use the Apollo files to install the system on their local server.

### 

### 1. **Application Requisites**

* Download and install a PHP Environment. We recommend using XAMPP.
  + <https://www.apachefriends.org/download.html>
* Download, install, and update Composer.
  + <https://getcomposer.org/>
* Clone Apollo repository to grab Laravel framework and system files into htdocs.
  + xampp>htdocs
  + <https://github.com/ApolloSoen341/apollo-website>
* NodeJS is needed in order to download other development tools such as Bower, Gulp and it is needed to run the web scraper.
  + <https://nodejs.org/en/download/>

### **2. Installing Development Tools**

* Before running the following commands, from the terminal, navigate to the application’s directory.
* Run composer install in order to install all the php dependencies for the application.
* Run npm install -g bower this will install bower on the computer, which will be used to download all javascript dependencies.
* Run bower install in order to download all the javascript dependencies.
* Run npm update in order to download Gulp and laravel-elixir.
* Run gulp in order to combine all the resource scripts and styles into one minified file.

### 

### **3. Configuration Files**

* Rename the .env.example to .env. The file is located in the root directory.
* Inside this file, rewrite the database configs as the following:
  + DB\_HOST=localhost
  + DB\_DATABASE=apollo
  + DB\_USERNAME={USERNAME}
  + DB\_PASSWORD={PASSWORD}
  + where {USERNAME} & {PASSWORD} are the values in order to connect to the database.
* Run the following command, to generate a key for the application:
  + php artisan key:generate

### **4. Database Creation**

* Create the database with mysql using the following command:
  + CREATE DATABASE apollo;
* Running the following code, in the terminal or cmd (assuming php is installed), will create the database, with populated values:
  + php artisan migrate --seed

### **6. Start & Stop**

* Use the xampp control panel to install and run the Apache and MySQL.

## **15.2. User’s Manual**

## **Getting Started**

On any preferred browser, navigate to the URL <http://apollo.matthewteolis.com/>

The homepage will appear as displayed on Figure 1. From there the user may choose to navigate to the following options:

1. **User Setting**

The user can click on his avatar or name to display their profile information or change his/her password and preferences.

1. **Toggle Sidebar**

The toggle sidebar is used to hide/show the sidebar. By default, the sidebar is displayed on large devices such as computer screens or tablets, and it is hidden on smaller devices such as smartphones.

1. **Sidebar**

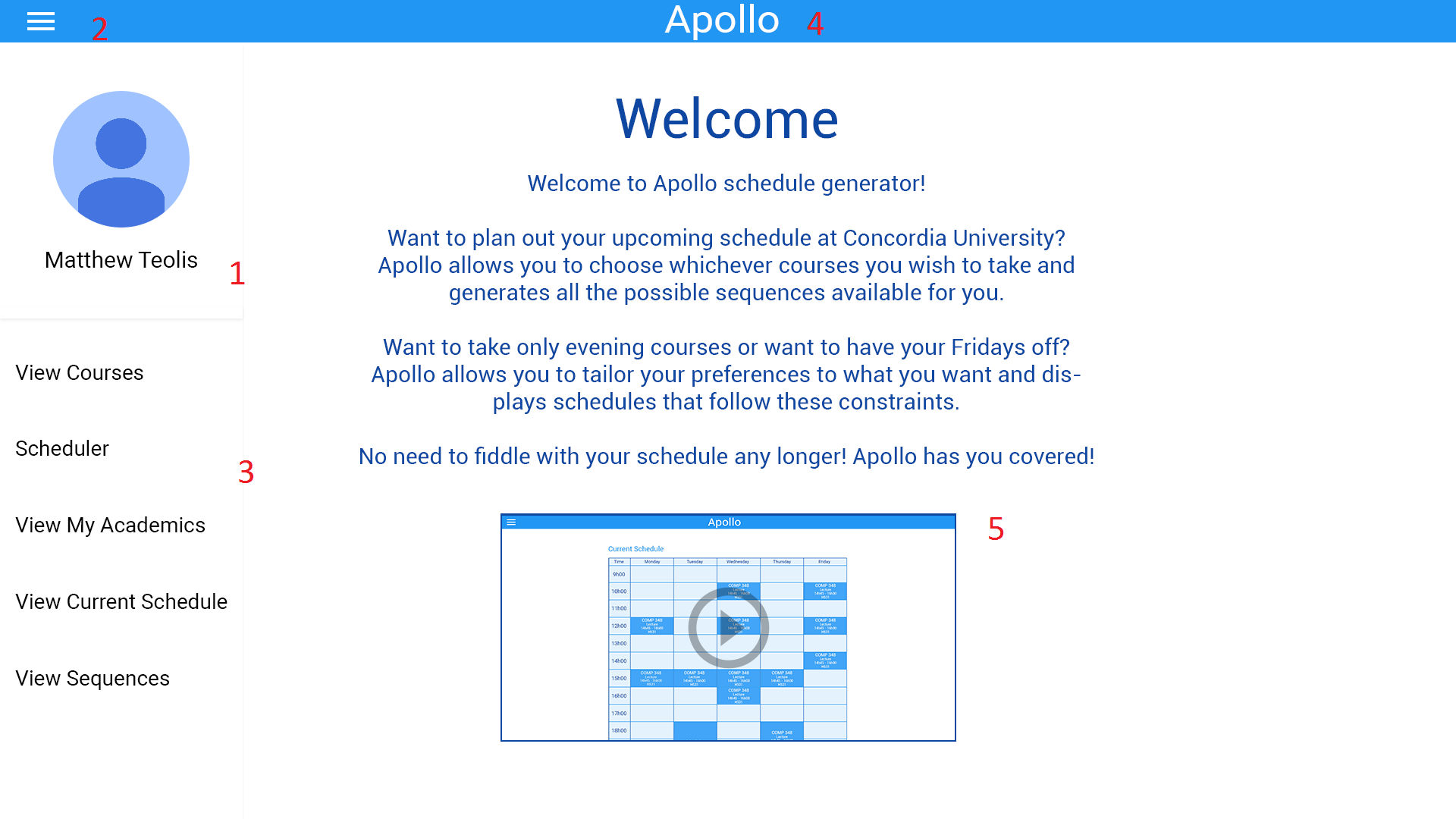
The sidebar is used to navigate to various other sections of the application.

1. **Application Home Button**

At all times, the user can click on the Apollo home button on the toolbar at the top of the page to navigate back to the homepage

1. **How-To Video**

There is an embedded Youtube video on the homepage providing a detailed walkthrough of the application.



## **Selecting Courses**

After navigating to “View Courses” page from the sidebar, the user will be able to select the courses he/she wishes to enroll in.

The user will have the following options:

1. **Search Courses**

The search bar is used to search for classes offered by concordia. The classes are filtered dynamically so the classes that match the search query appear as the user types.

1. **Generate**

The Generate button, once pressed, will display the generated schedules using the selected classes to the student. (See “Generated Schedules” below)

1. **Selected Course Chip**

When the student selects a course, a chip matching the course will appear under the search bar. These chips represent the set of courses the user wishes to generate the schedule with. Courses can be easily removed from the set by clicking on the X in course’s chip.

1. **Course List**

The course list contains the courses offered by Concordia. The list changes dynamically to match the search query as the student types inside the search bar.

1. **Course Details**

Details about the selected course is displayed when the user clicks on the course’s box. This includes the course description, requisites, credits and faculty.

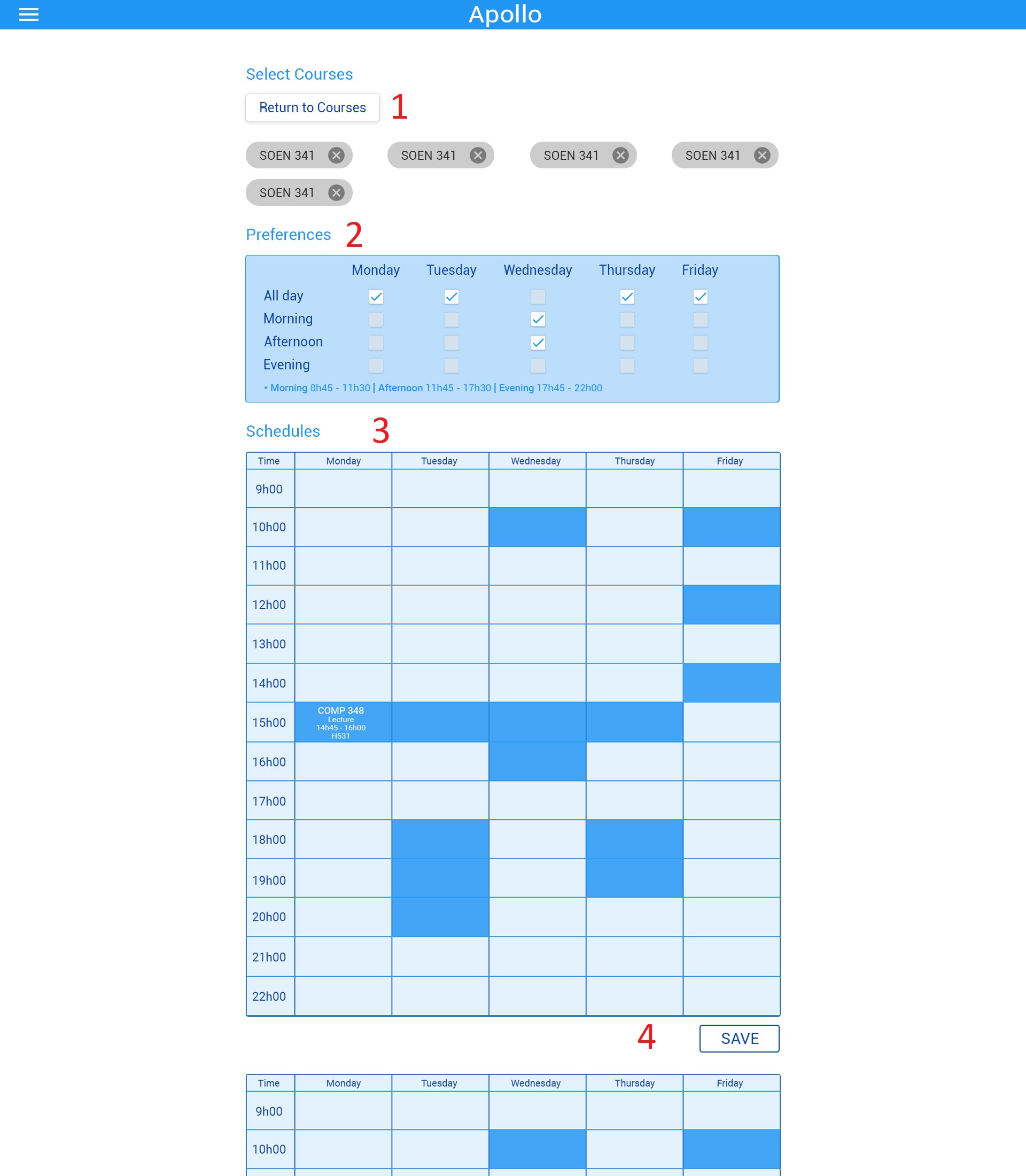


## **Generated Schedules**

Once the user clicks on the “Generate” button, the page containing containing all the schedules created by the scheduler will be displayed.

The following elements are shown:

1. **Return to Courses**
   1. If the user wishes to make changes to the selected courses, the “Return to Courses” button will return the user to the previous screen.
2. **Schedule Preferences**
   1. The user can indicate their scheduling preferences. By default, “All Day” is selected, indicating that courses can be scheduled any time of day. The user can change their preferences and the scheduler will only include the schedules that match the preferences specified.
3. **Schedule**
   1. Every possible schedule combinations will be displayed as a table.
4. **Save Schedule**
   1. This button saves a selected schedule into the user’s “Saved Schedules” page. The user can refer to the schedule at any time after saving it.



## **Edit Preferences**

Once the user navigates to their profile page, they can change their password or preferences.

The following elements are displayed:

1. **User Profile Information**

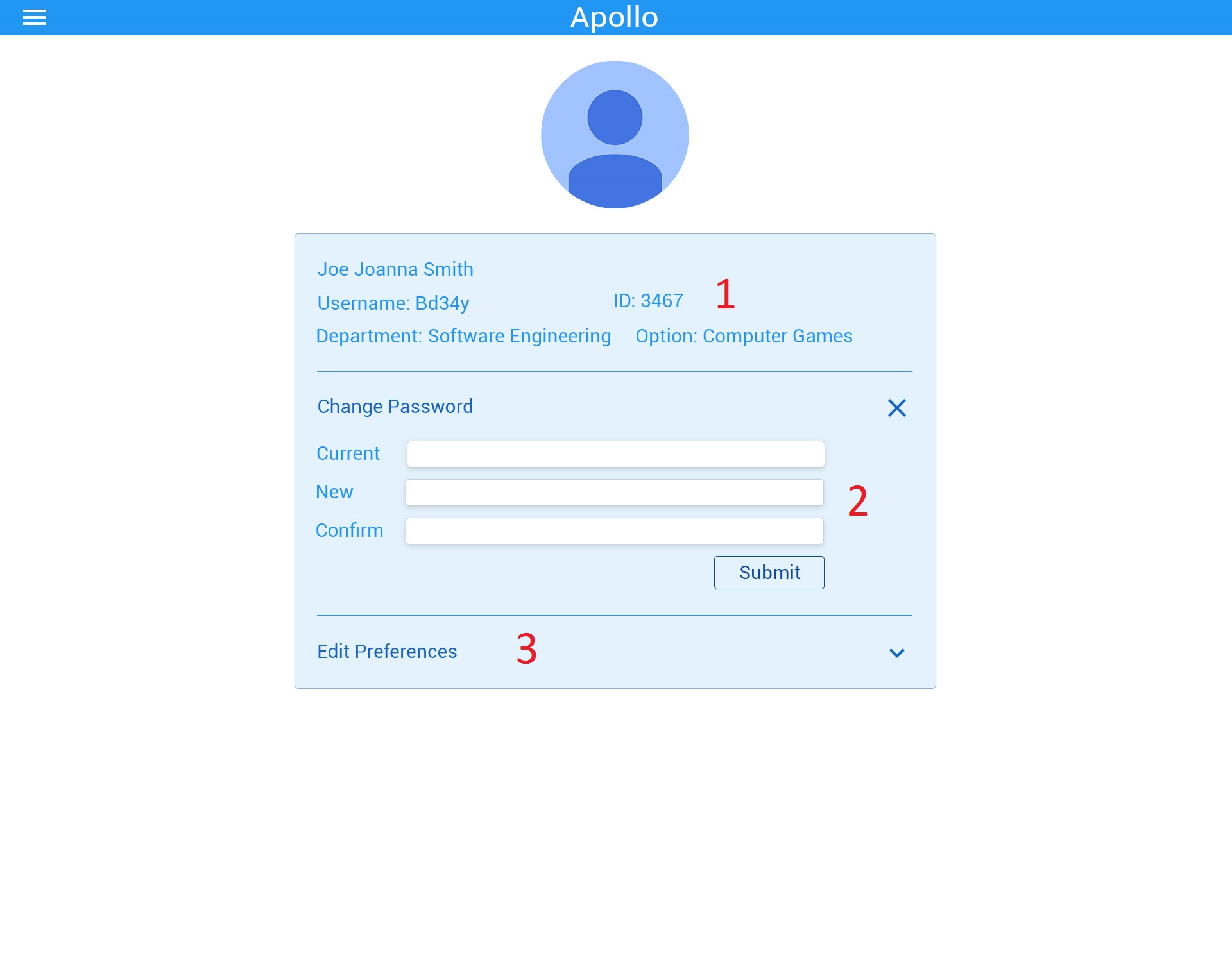
Details about the user’s academic information are displayed here.

1. **Change Password**

The user can change their password.

1. **Edit Preferences**

The user can change their scheduling preferences (ex. Morning, Afternoon, Night).



# 

# 

# 16. FINAL COST ESTIMAT**E**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tasks** | **Previous Cost (Hours) for all members** | **Revised Cost (Hours) for all members** | **Duration (Days)** |
| Member Organisation | 1 | 1 | 2 |
| Initial Planning | 2 | 2 | 1 |
| Deliverable 0 | 4 | 4 | 1 |
| Scheduling | 2 | 2 | 1 |
| Resource Evaluation | 5 | 5 | 1 |
| Technical Configuration | 3 | 3 | 2 |
| Domain Model Design | 3 | 3 | 1 |
| Architecture Planning | 5 | 5 | 2 |
| Initial Architecture Design | 3 | 3 | 2 |
| Database Setup | 24 | 24 | 1 |
| Rapid Prototype | 24 | 24 | 1 |
| Data Input | 15 | 15 | 1 |
| Functional Requirements Planning | 10 | 10 | 3 |
| Constraint planning | 5 | 5 | 1 |
| Scoping | 5 | 5 | 1 |
| Initial Front End Planning | 5 | 5 | 3 |
| Estimation 1 | 1 | 1 | 1 |
| Risk Planning 1 | 1 | 1 | 1 |
| Deliverable 1 | 30 | 30 | 24 |
| Architecture Design | 5 | 5 | 3 |
| Front End Planning | 10 | 10 | 3 |
| Front End Design | 10 | 15 | 4 |
| Detailed Design | 10 | 10 | 2 |
| Dynamic Design Scenarios | 20 | 25 | 4 |
| Front End Prototype | 30 | 30 | 2 |
| Estimation 2 | 1 | 1 | 1 |
| Risk Planning 2 | 3 | 3 | 1 |
| Deliverable 2 | 30 | 40 | 30 |
| Implementation Planning | 15 | 20 | 4 |
| Implementation | 40 | 45 | 5 |
| Test Planning | 15 | 15 | 5 |
| Testing | 30 | 30 | 2 |
| Improvement | 15 | 15 | 2 |
| Testing | 10 | 10 | 3 |
| User Manual | 20 | 20 | 2 |
| Final Cost Estimate | 2 | 2 | 2 |
| Deliverable 3 | 40 | 45 | 20 |
| Project Documentation | 80 | 80 | 7 |
| Deliverable 4 | 15 | 20 | 10 |
| **TOTAL** | **549** | **589** | **88** |

# 

# 

# REFERENCES

[1] GoDaddy, “How many viewers can view my site at once?” 2016. [Online]. Available: <https://ca.godaddy.com/help/how-many-visitors-can-view-my-site-at-once-3206#cpanel>

[2] “Domain Model”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[3] “The Software Process”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[4] “Requirements and User Case”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[5] “Risk Management”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[6] “Activity Planning”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[7] “Estimation Techniques”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[8] “Architecture and Design”, class notes for SOEN 341 Software Process, Department of Computer Science and Software Engineering, Concordia University, Quebec, Winter 2016.

[9] T. Otwell. *Laravel* [Framework]. Available: https://laravel.com/

[10] Brat Tech LLC, Google. *AngularJS* [Framework]. Available: https://angularjs.org/

[11] ISO/IEC 9126-1:2001 Software engineering - Product Quality, [Online]. Available: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749>

[12] "Fast Facts." Fast Facts. Concordia University, 6 Aug. 2015. Web.

# 