Concordia University

Department of Computer Science

and Software Engineering

Software Process

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

Deliverable 2

March 23rd, 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.Grading Scheme

Section	Evaluation criteria (see instructions in the template for details)	Grading
all	*10 marks are allocated for excellence, professionalism and quality of work above and beyond the correct meeting of specifications..*	*/10*
1	*Presentation of the document*	*/5*
2	*Introduction of the document*	*/1*
3.1 *3.2*	*Validity and clarity of the architectural diagrams, as well as of the textual rationale.* *Validity, completeness, and clarity of description of each component interface.*	*/5* */4*
4.1 *4.2*	*Validity and clarity of the (UML) class diagrams or equivalent for each subsystem, as well as of the textual rationale.* *Validity and clarity and completeness of the class descriptions for each subsystem.*	*/8* */4*
5	*Validity and clarity of the dynamic design diagrams and contracts for each scenario.* *Compatibility of the scenarios with the components presented in section 2 and 3, as well as of the textual rationale.*	*/8*
6	*Revised cost estimation of each individual artifact, validity of explanation of cost estimation, total cost estimate*	*/2*
7	*Rapid Prototyping and Risk Report*	*/3*
Total		*/50*

DO NOT REMOVE THIS PAGE WHEN SUBMITTING YOUR DOCUMENT

2. INTRODUCTION

The purpose of the Apollo system is for a student enrolled in Concordia's Software Engineering program to plan their schedule for the upcoming semesters based off their academic history, and constraints given by course choices and time preferences.

The application has been scoped down to include only the student user. With this application, the student is able to create schedules based of their Software Engineering program option graduation requirements. The following four options are given as course sequences to Concordia students:

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

Once logged into Apollo, the student can generate schedules. The scheduler prompts the student first to filter through courses they would like to choose for their upcoming semesters. Once chosen, the student can then edit their course time preferences. By default the student has all-day availability set for their preferences.

Several settings for course time preferences are:

- All day
- Morning: 8h45 - 11h30
- Day: 11h45 - 17h30
- Evening: 17h45 - 22h00

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 requisites have been met
- Checking if the credit requirements are present
- Courses for which the student has an exemption

Once these constraints and course time conflicts has been verified, Apollo will generate the number of possible schedules that the student can choose from. The student can then save schedules they prefer to their account.

3. ARCHITECTURAL DESIGN

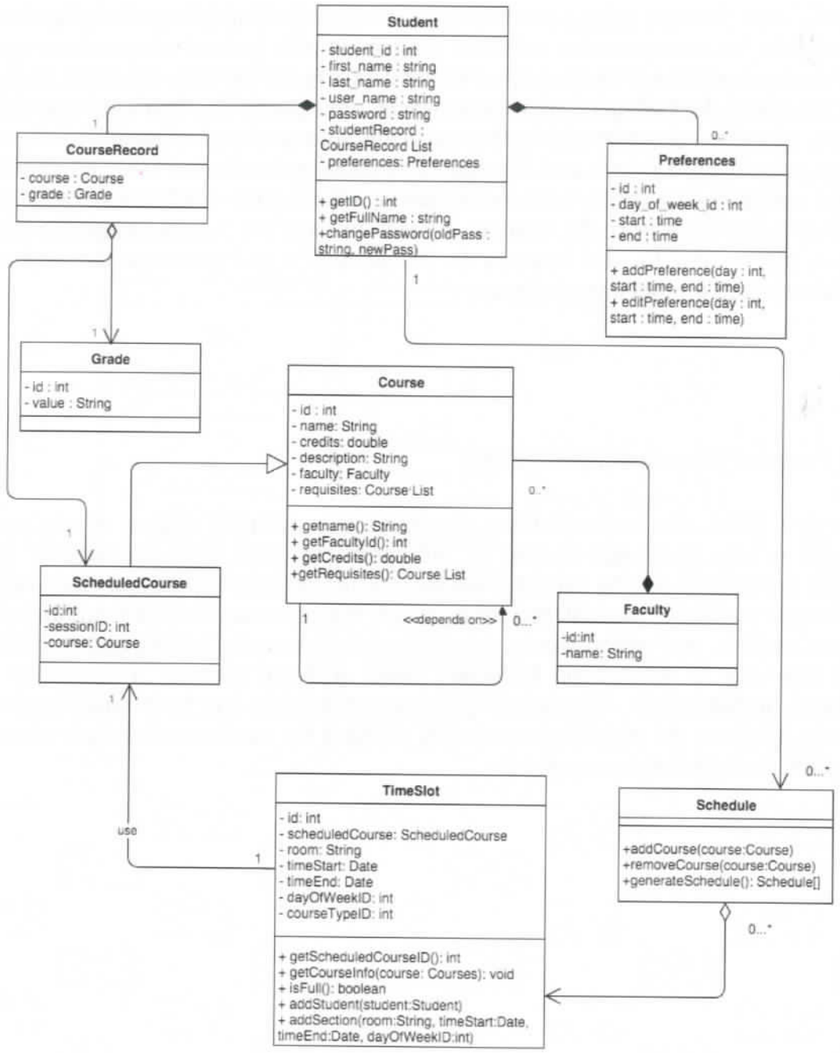
In this section a high-level description of the system and its modules is presented.

3.1 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.

3.1.1. 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.) Following the structure of this document (sequence diagrams are shown later), only the Class Diagram is used here, as shown the following image.



Class Diagram

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

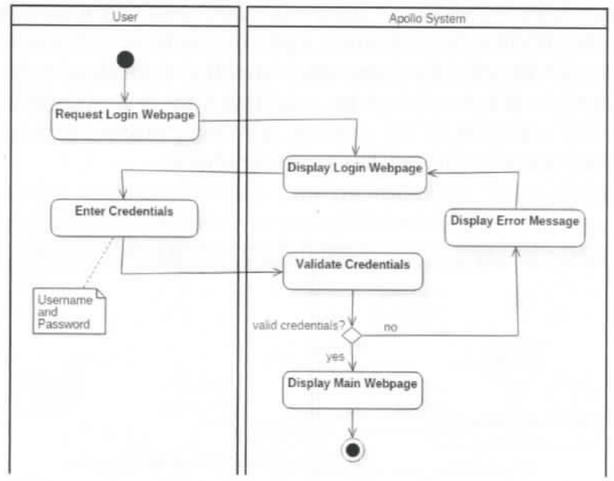
schedule. The Courses subsystem manages all the information regarding courses that are available. Courses provides a master course list with general information regarding each course available, including the Faculty it is offered by. Subsequently, Scheduled Courses would further take this information 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.

3.1.2. 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.

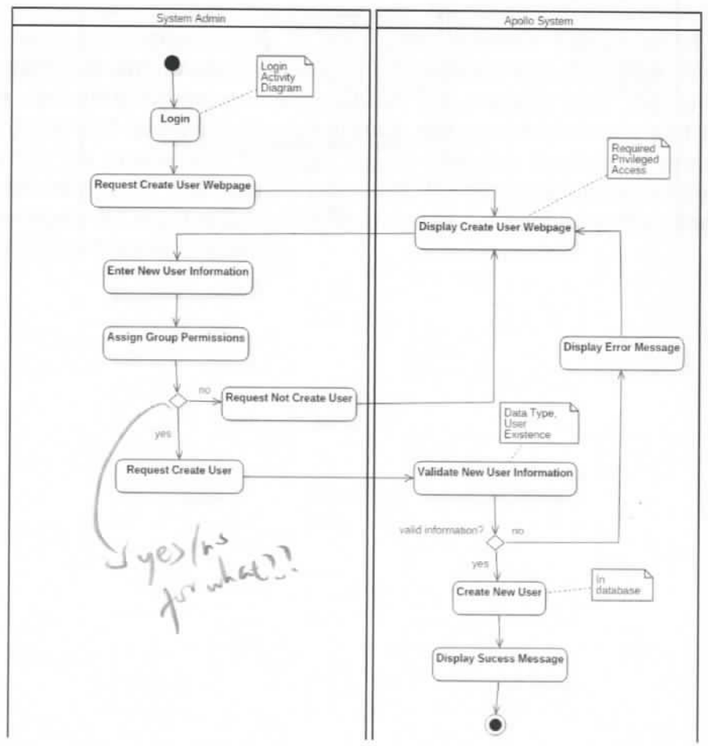
Activity Diagram "Login" (by All Users)

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)

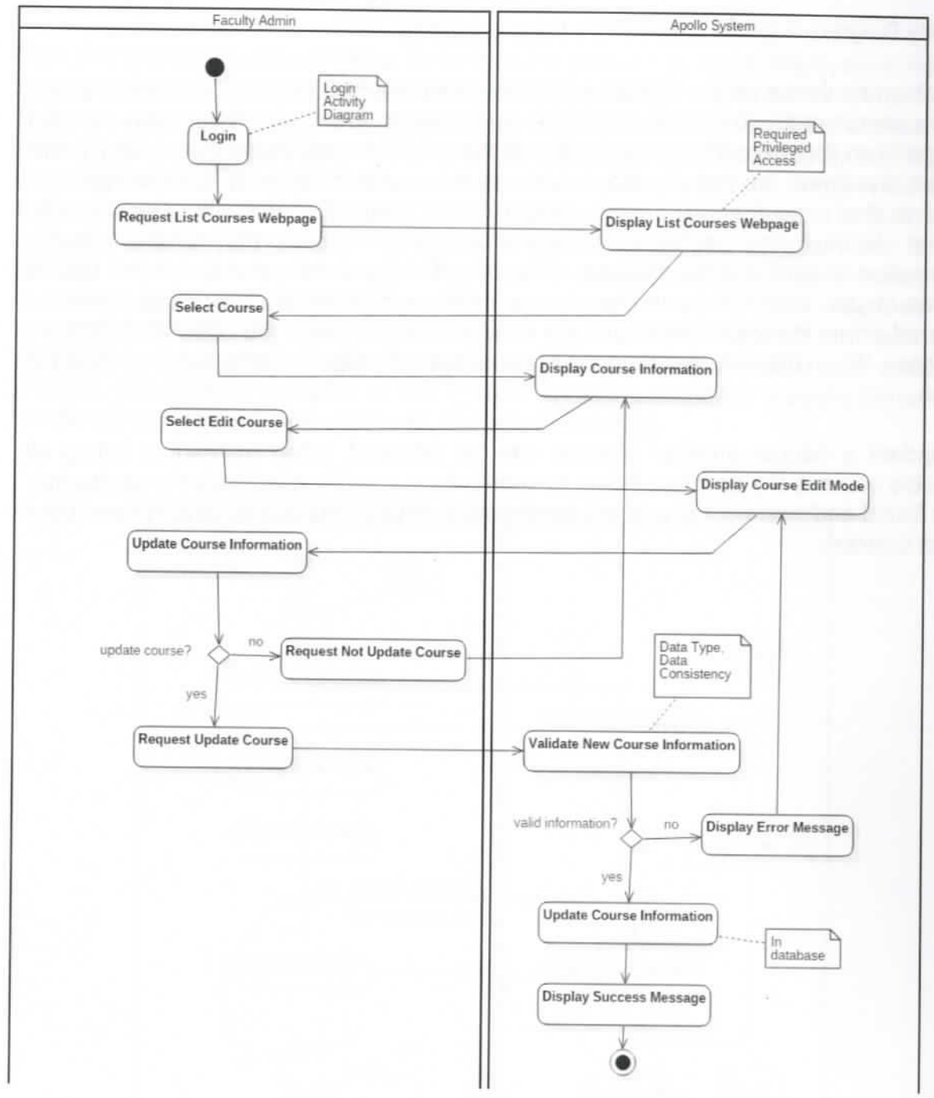
The activity diagram shown below describes the process by which the System Admin creates a user in the Apollo System. Like all the available options, it starts with the login of the user into the Apollo System, process previously described. Once the access has been granted, among all the available options, 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)

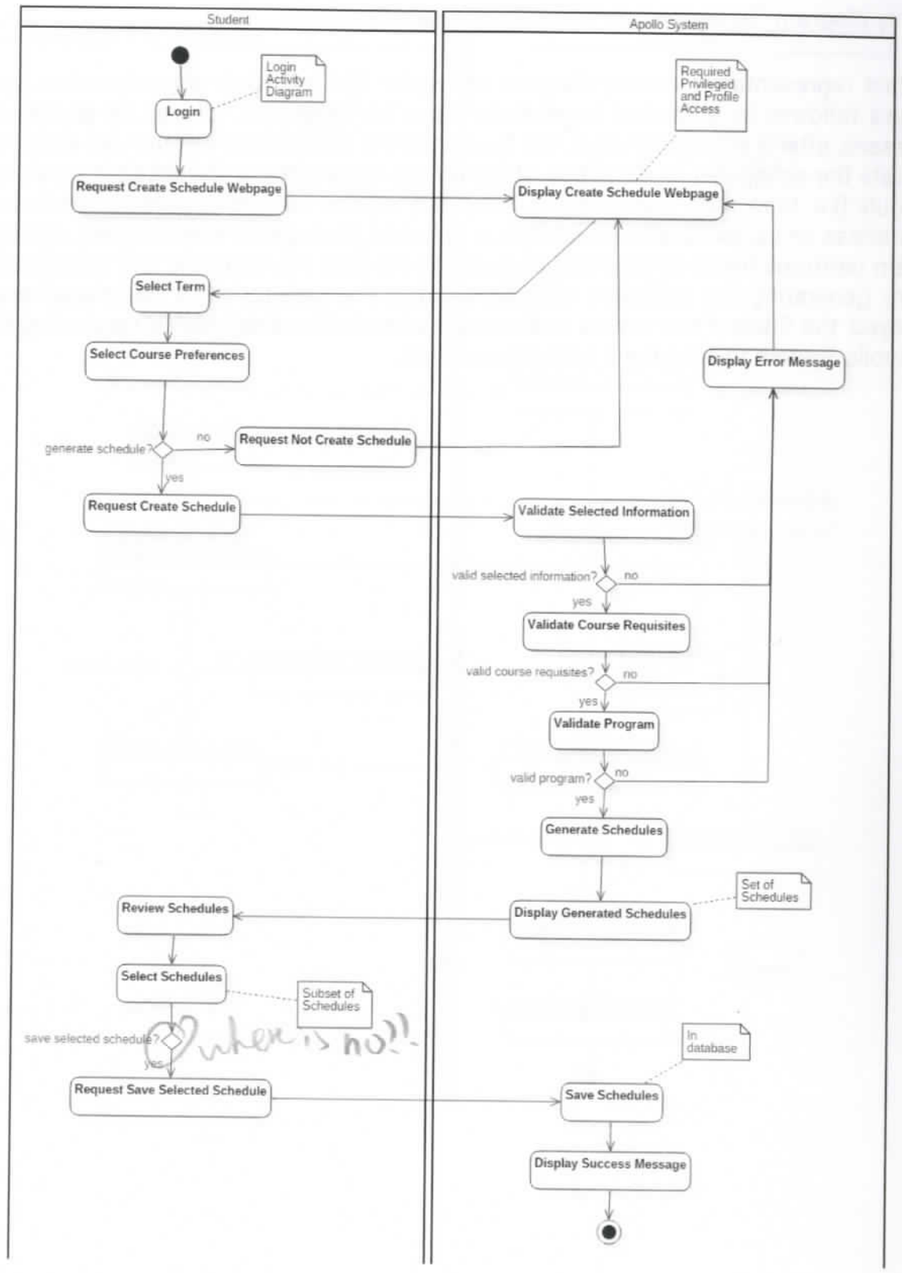
The diagram shown on the following page details the activities performed by a Faculty Admin user in order to update a course's information. After successful login, the user selects from the menu the option to list all the available courses in the system. From the list displayed, the Faculty Admin selects 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).

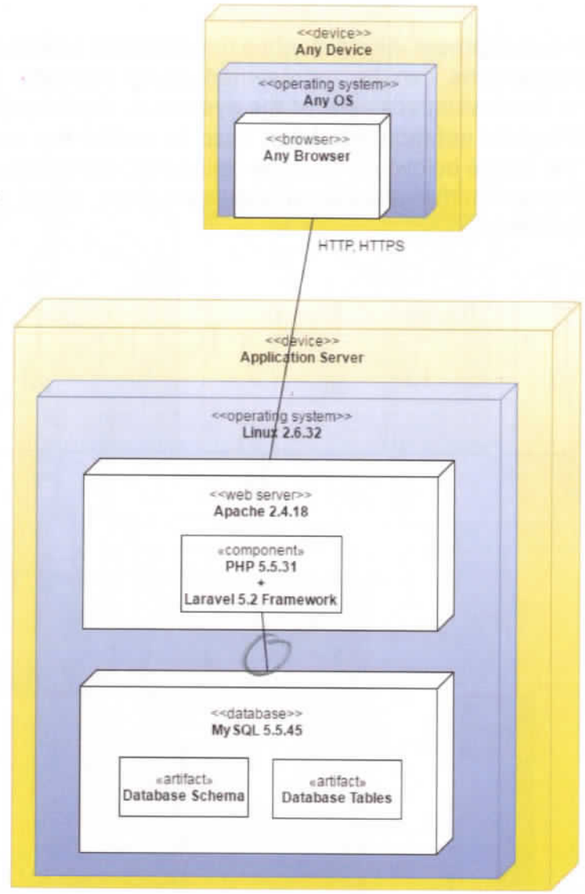


Activity Diagram "Generate Schedules" (by Student)

The last representative activity diagram shown on the following page 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). After that, he/she can decide either to continue the process or cancel it. 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. After the generation is completed and displayed, the Student can review and select his/her prefered schedules, requesting to the Apollo System to save them for further review.



3.1.3. Physical View: Deployment Diagram



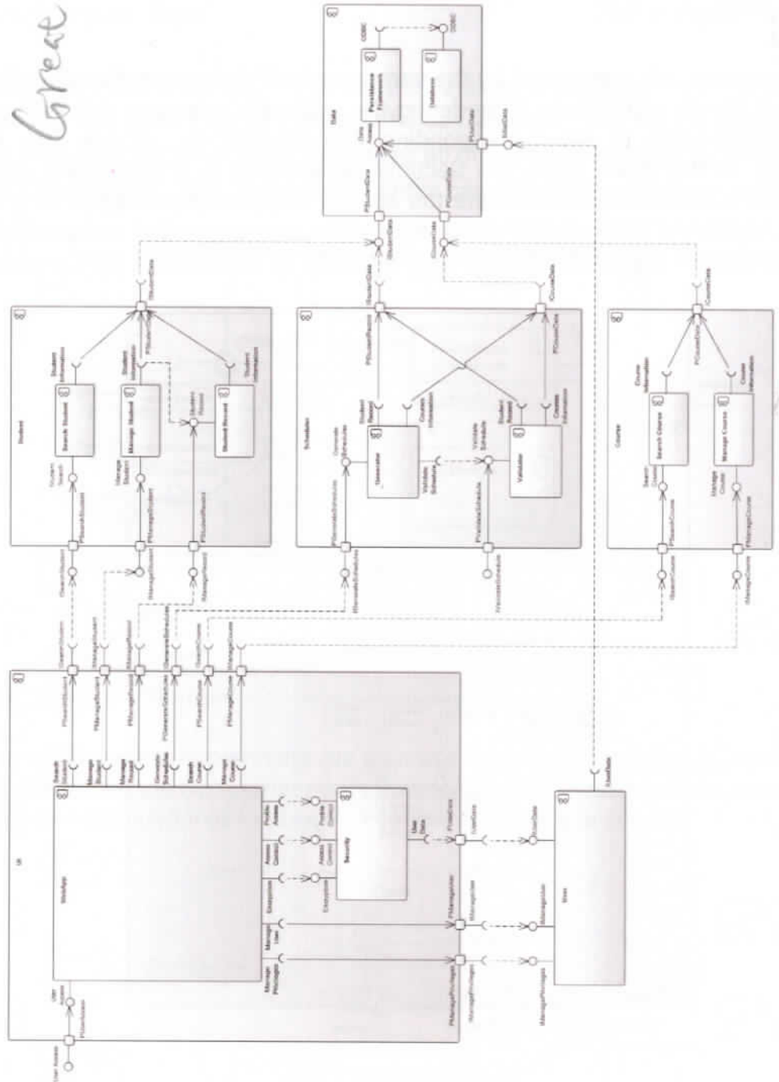
The deployment of our application is divided into two main physical nodes, as shown in the figure. The client node can be any device with any operating system, as the application is platform-independent. Clients interact with the application through a browser of their choice. The application server is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

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.

3.1.4. Development View: Component Diagram

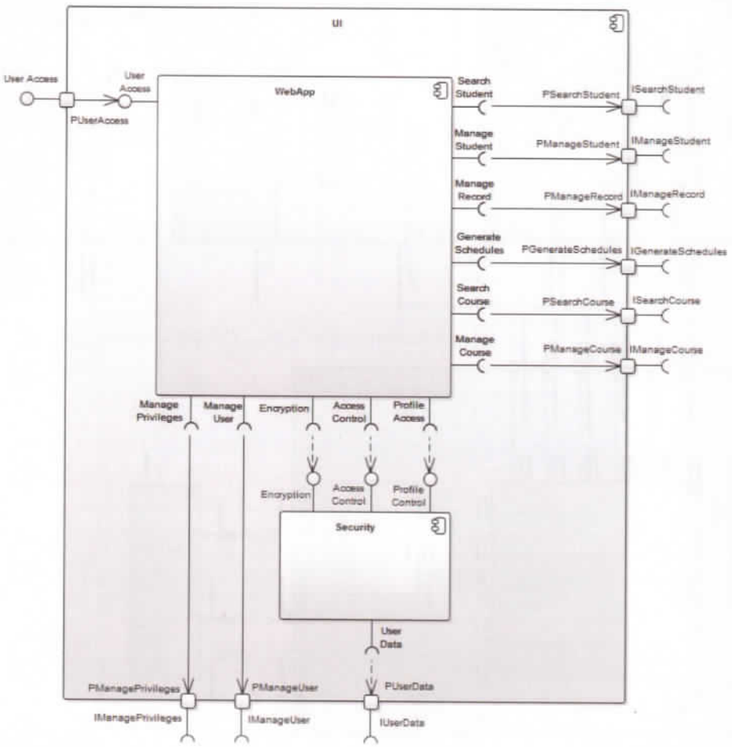
In the 4+1 AVM, the Development View describes the structural aspect of the system. In it modules and subsystems can be represented using Package and Component diagrams, making it the developer's view of the system. In the following sections the UML Component Diagram notation has been used to model the 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)

Great!



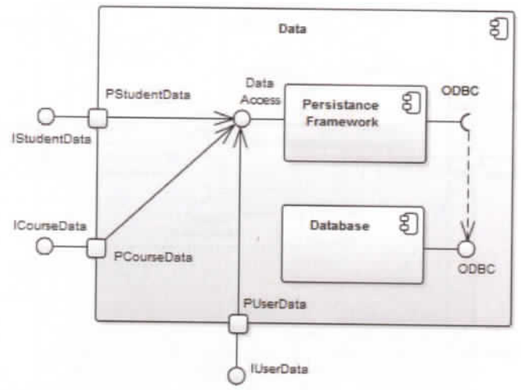
Component Diagram "UI"

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.



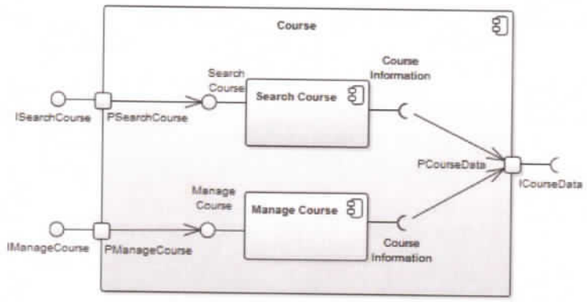
Component Diagram "Data"

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.



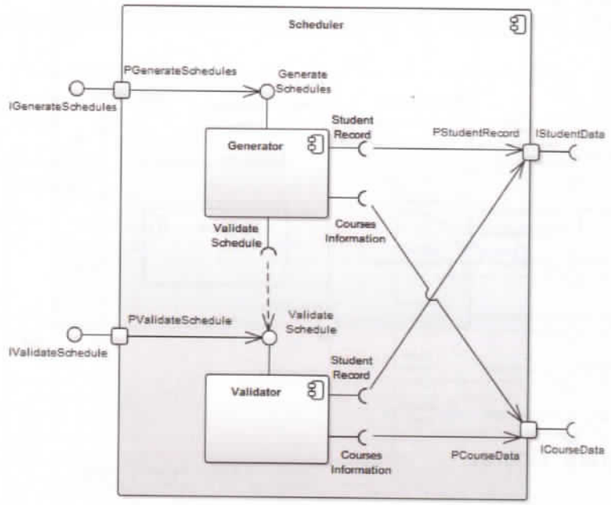
Component Diagram "Course"

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.



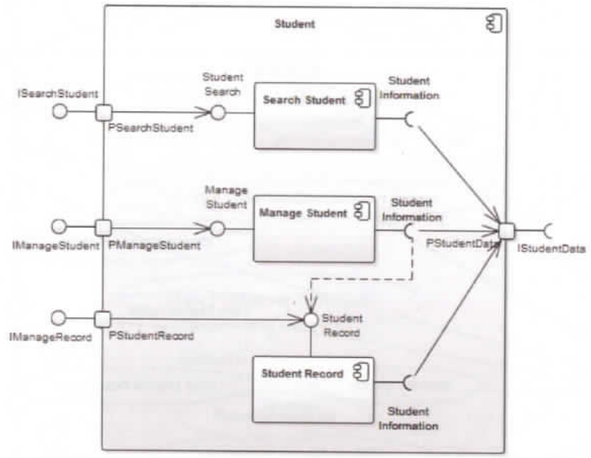
Component Diagram "Scheduler"

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.



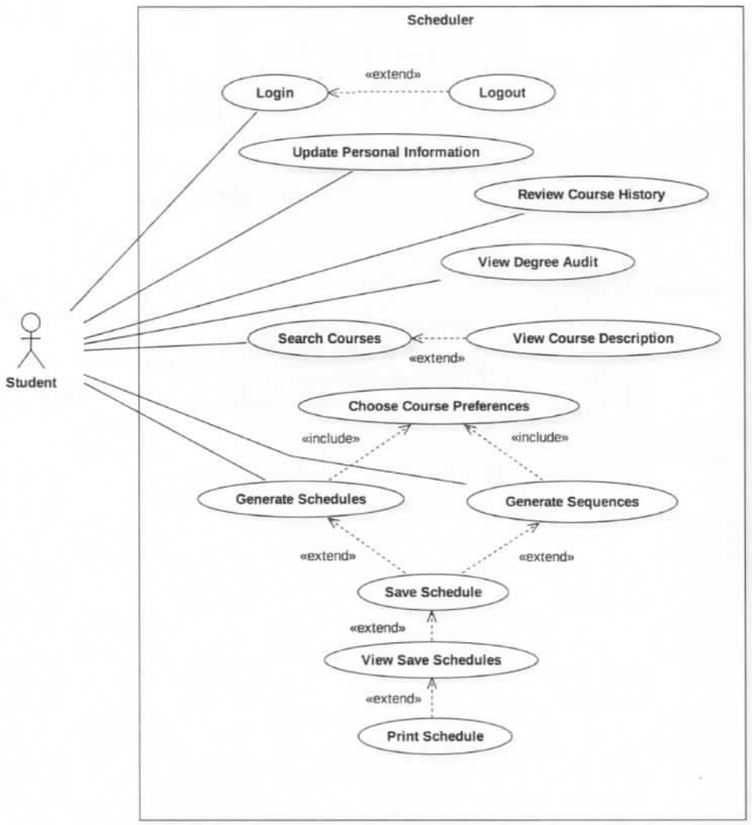
Component Diagram "Student"

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.



3.1.5. Scenarios: Use Case Diagram

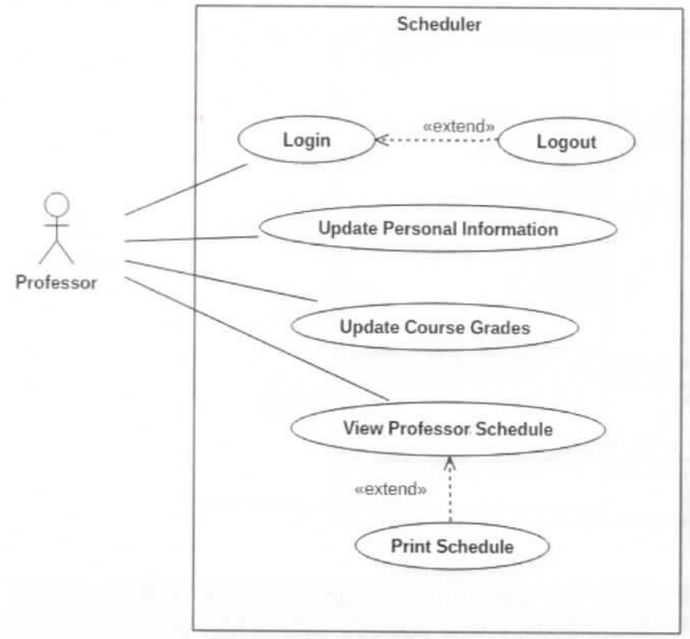
Student Scenarios



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.

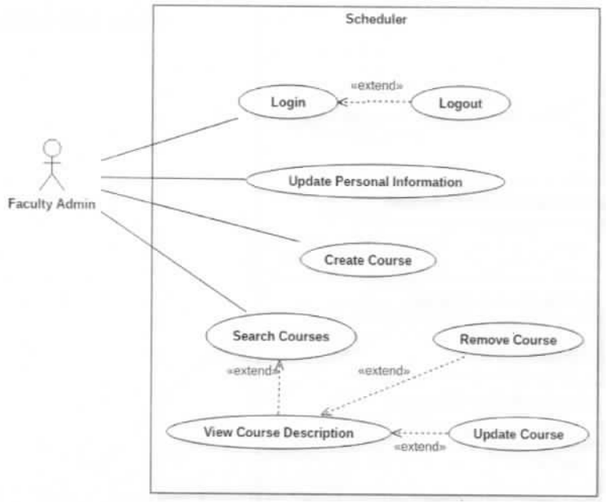
Professor Scenarios



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.

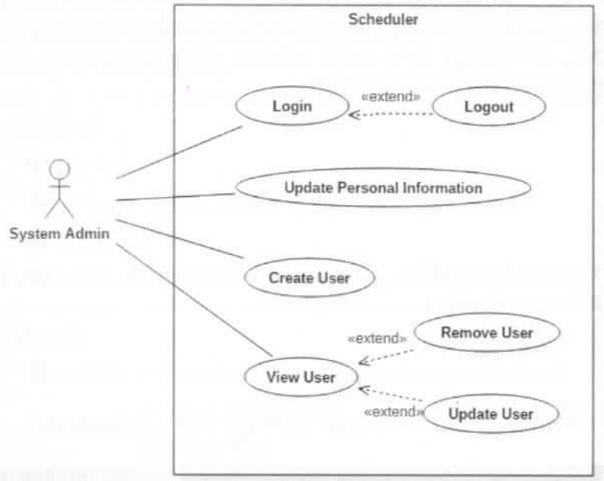
Faculty Administrator Scenarios



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.

The methods can be described with more details.

3.2 Subsystem Interfaces Specifications

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.

attributes are not needed here.

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

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

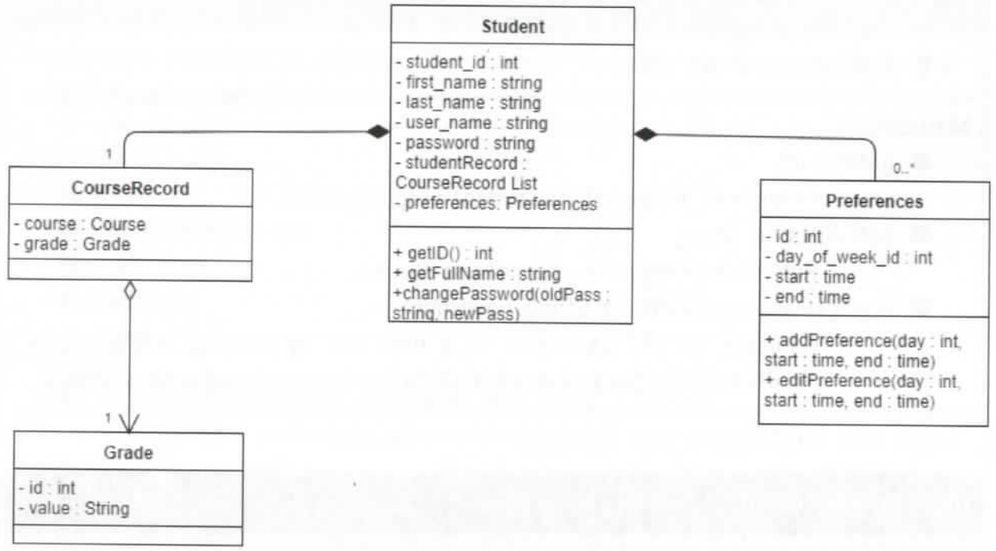
4. DETAILED DESIGN

4.1 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.

4.1.1 Student Subsystem

4.1.1.a Student Subsystem - Diagram



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.

4.1.1.b Student Subsystem - 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

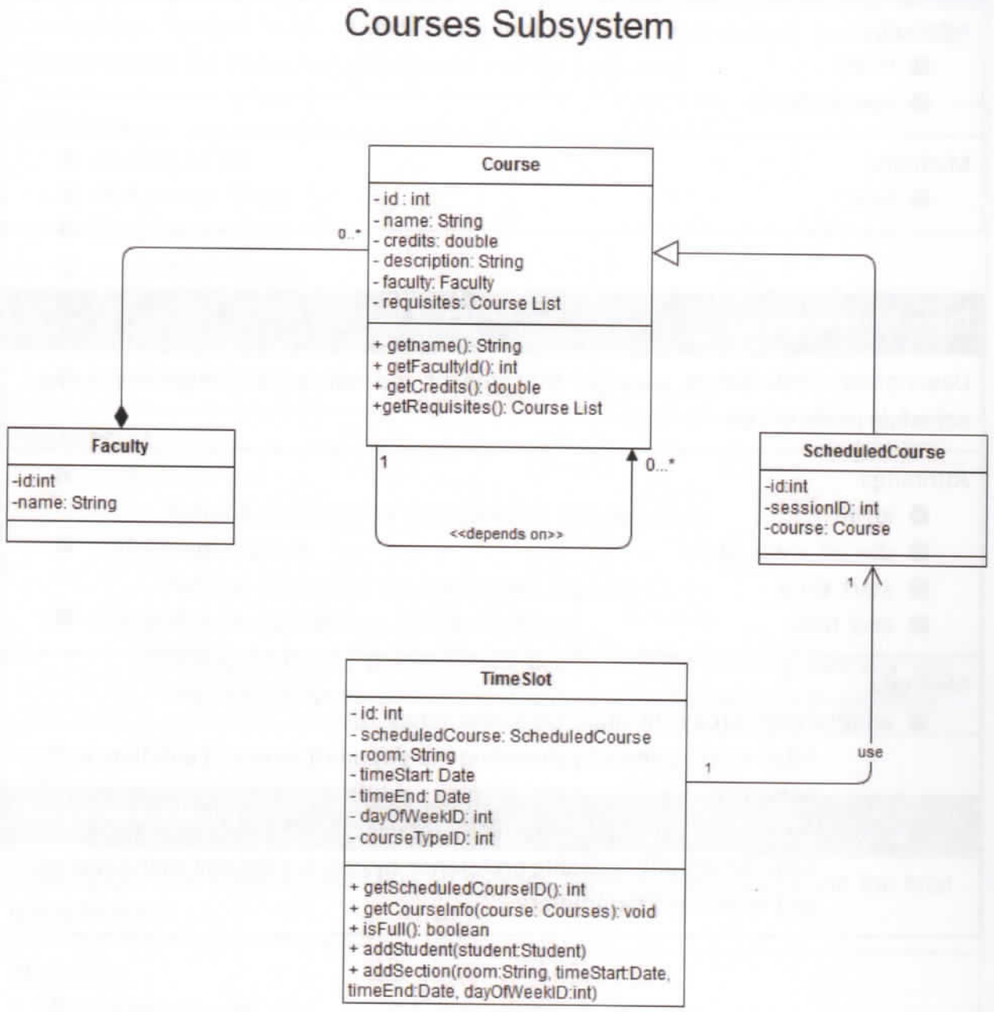
Class CourseRecord
Description: CourseRecord contains all courses taken by the student and the final grade they got
Attributes: ● Course: Course ● Grade: Grade
Methods: ● N/A

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

4.1.2 Courses Subsystem

4.1.2.a Courses Subsystem - Diagram



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.

4.1.2.b Courses Subsystem - 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

5. DYNAMIC DESIGN SCENARIOS

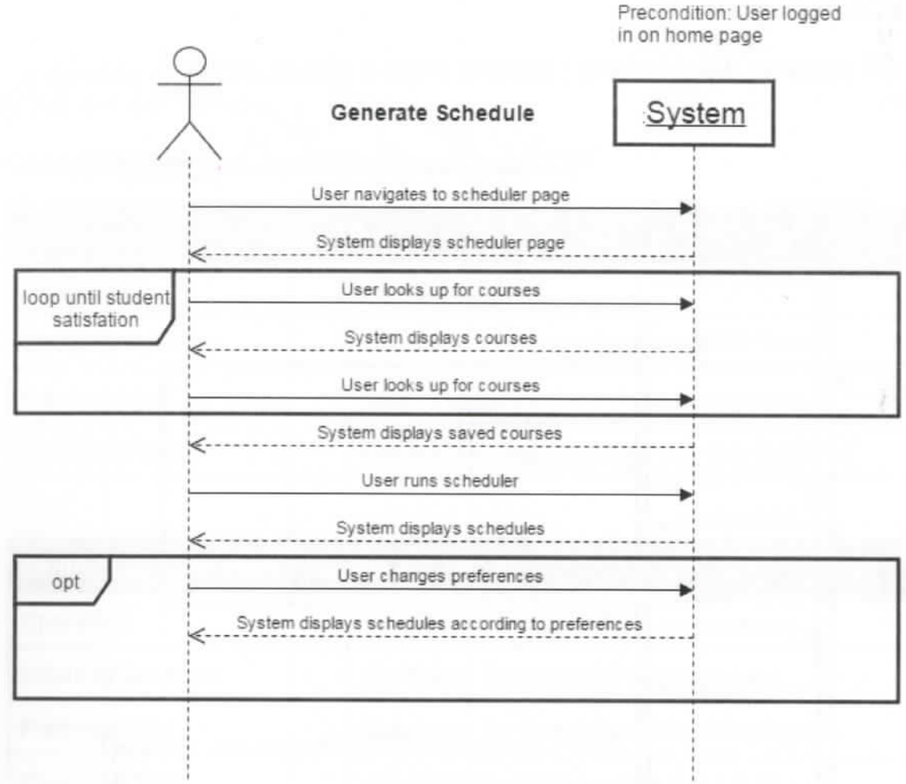
A full dynamic design of the use cases for "Generate Schedule" and "Save Preferences" use cases. This includes the system sequence diagram and the full sequence diagram for each use cases.

5.1 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.

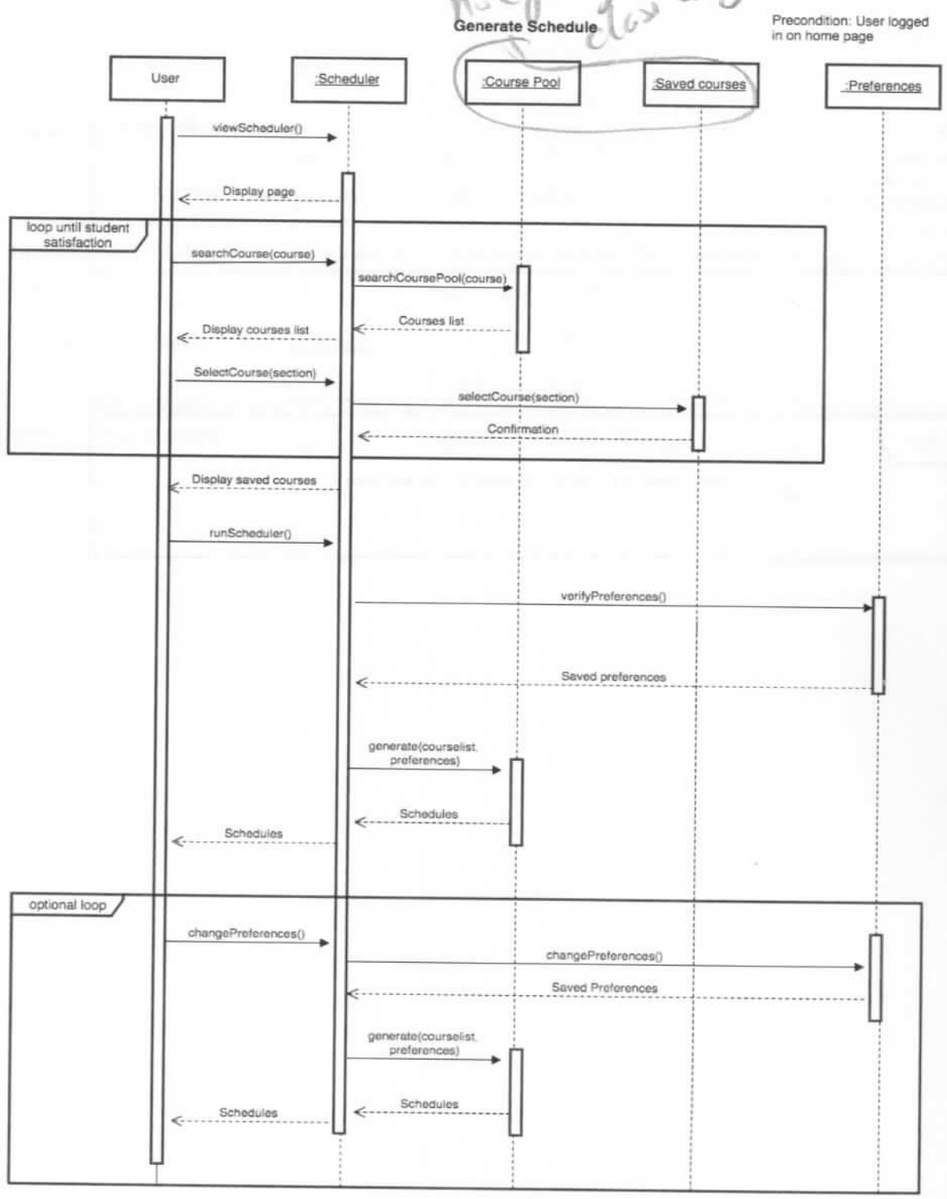
5.1.1 System Sequence Diagram

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



5.1.2 Sequence Diagram

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



5.1.3. 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

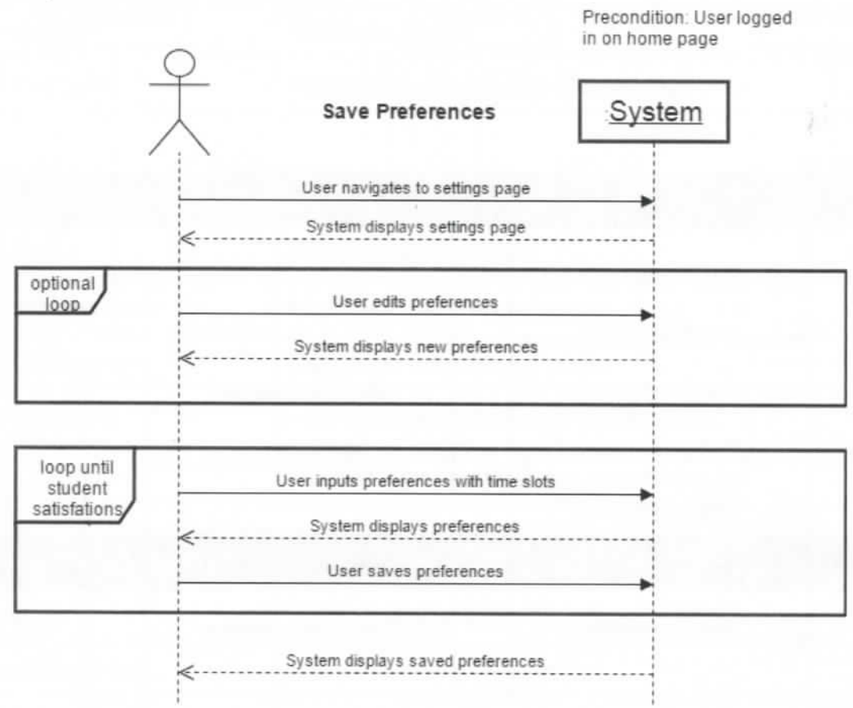
Contract C09.05: changePreferences	
Operation	changePreferences()
Cross References	Use Case: Generate Schedules
Preconditions	User has selected to edit preferences.
Postconditions	-An instance p of Preferences was created -p was associated with the new Preferences the user wanted -p modified the current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

5.2 Save Preferences

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

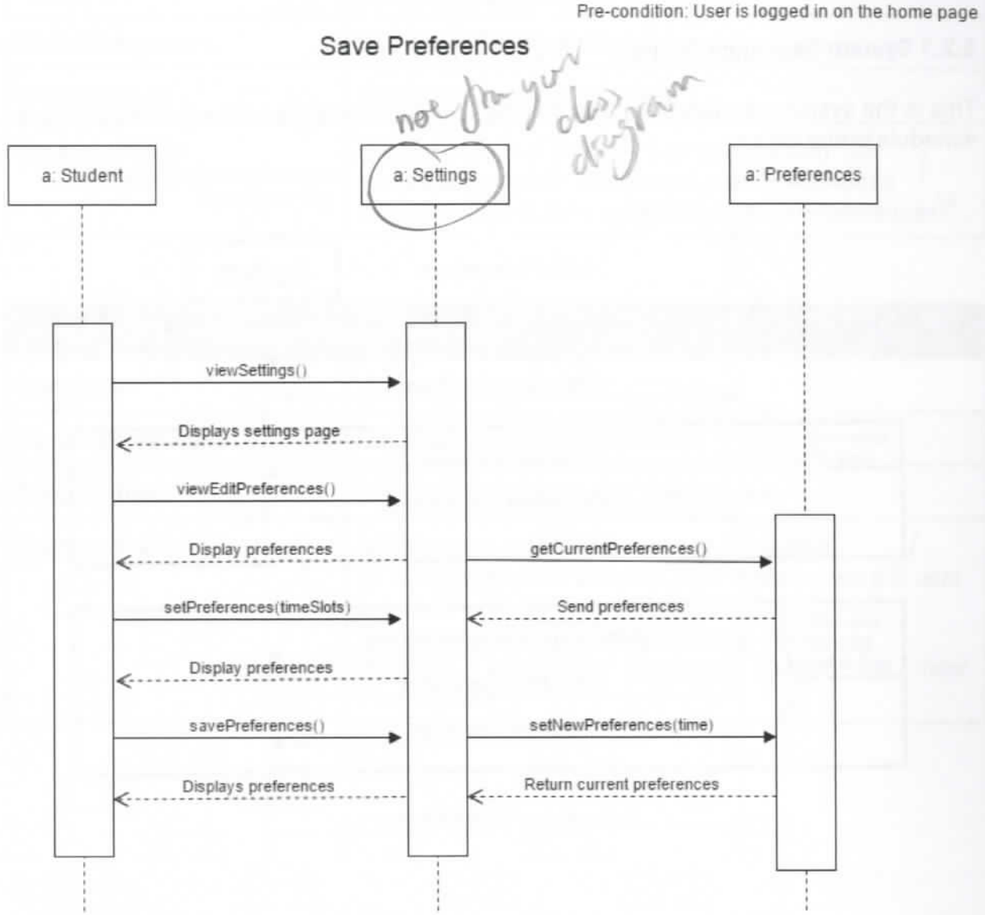
5.2.1 System Sequence Diagram

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



5.2.2 Sequence Diagram

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



5.2.3 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

6. 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

7. RISK

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.

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.

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.

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.

8. RAPID PROTOTYPING

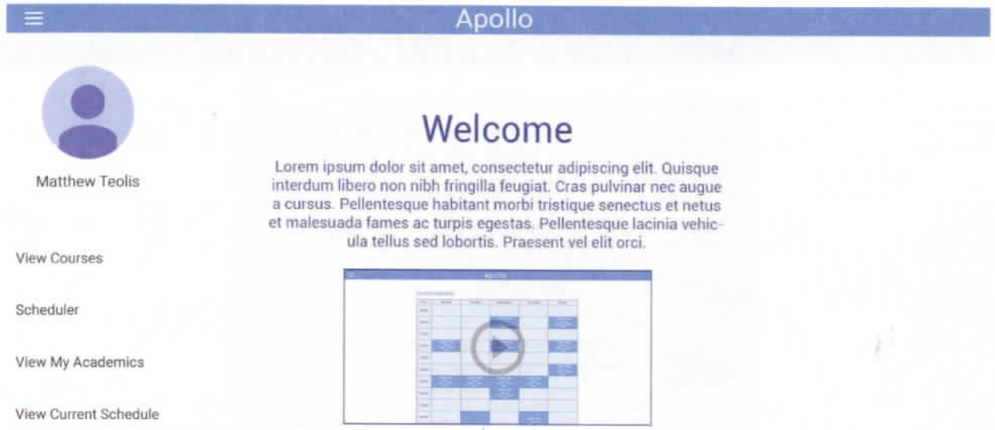
We have used the RESTful API using Laravel to connect the backend to the frontend. We can create, delete, list all courses and show specific courses and faculties. We display this information using Angular.js, and use the Angular Material framework for frontend.

8.1. Frontend Wireframe Design

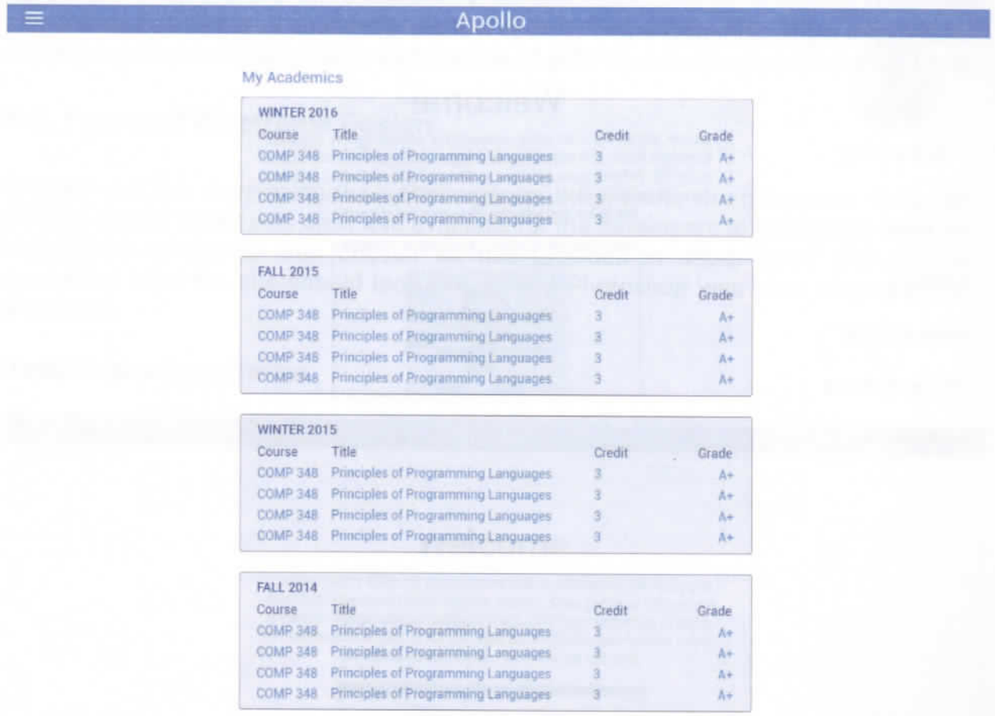
In order to begin developing the website pages, the website wireframe was made so that the design layout was clear and available for the developers to implement from. A high-fidelity wireframe was chosen as this provided in depth detail that closely resembles what the site should look like. Adobe Photoshop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

Home Page without Navbar

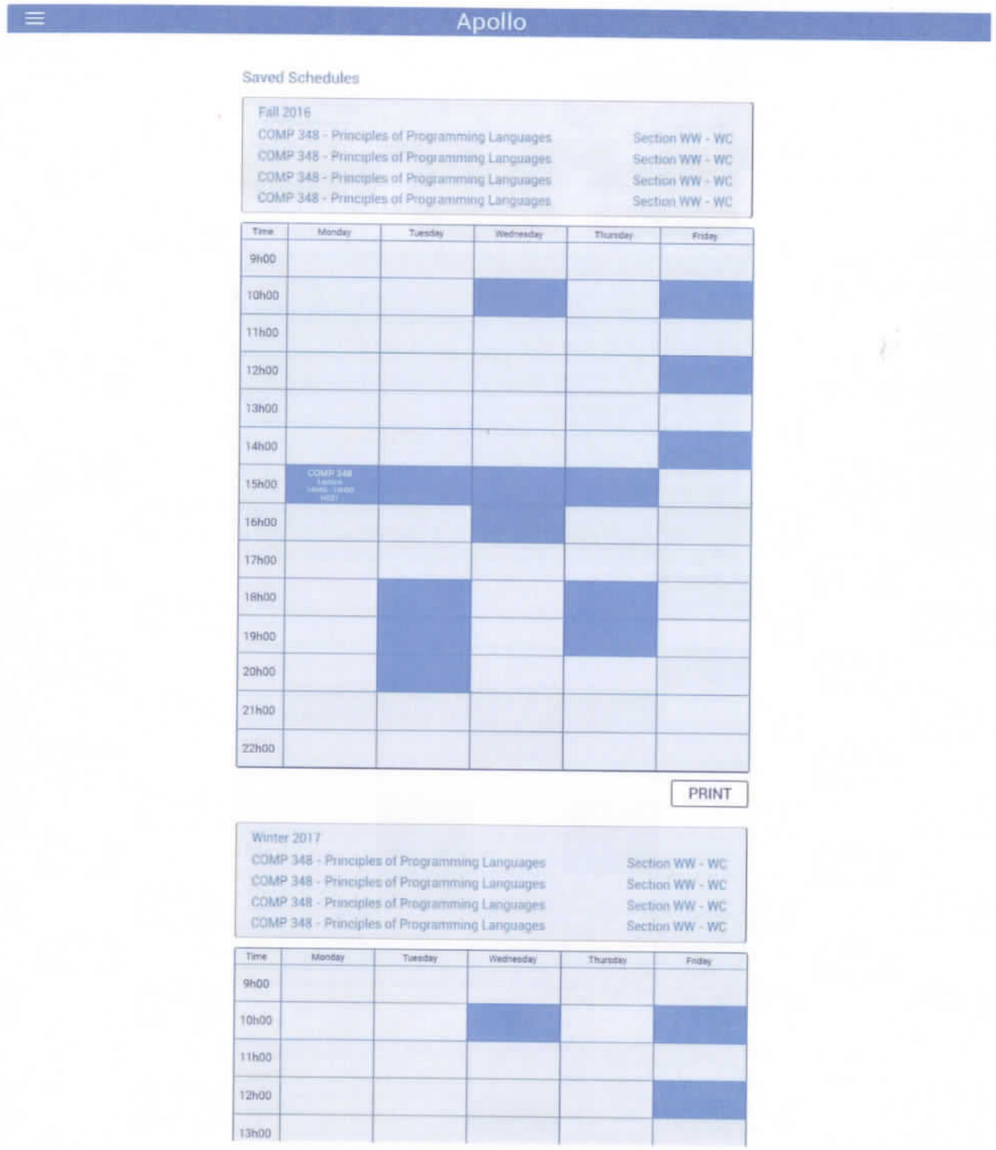
Home Page with Navbar



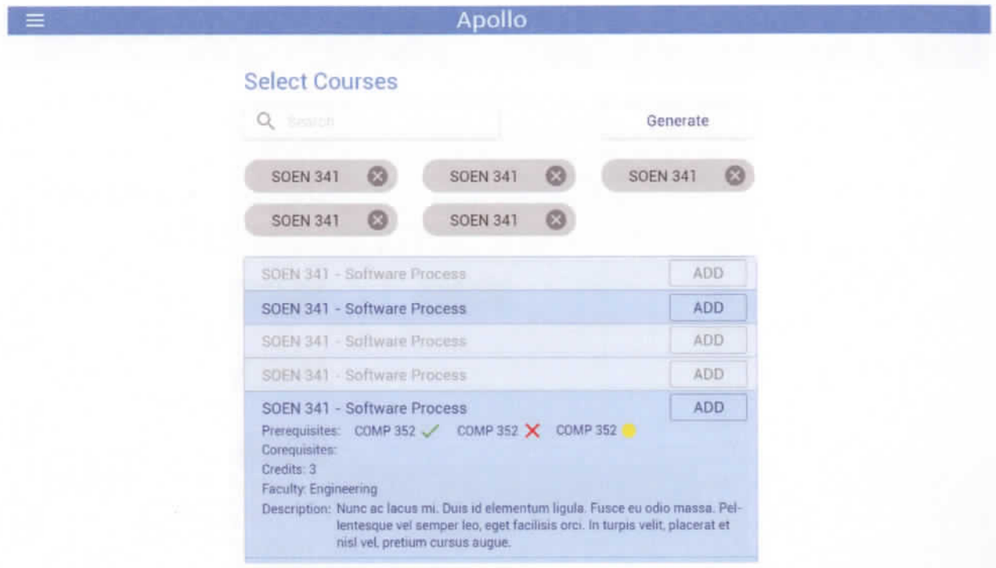
My Academics Page



Saved Schedules Page

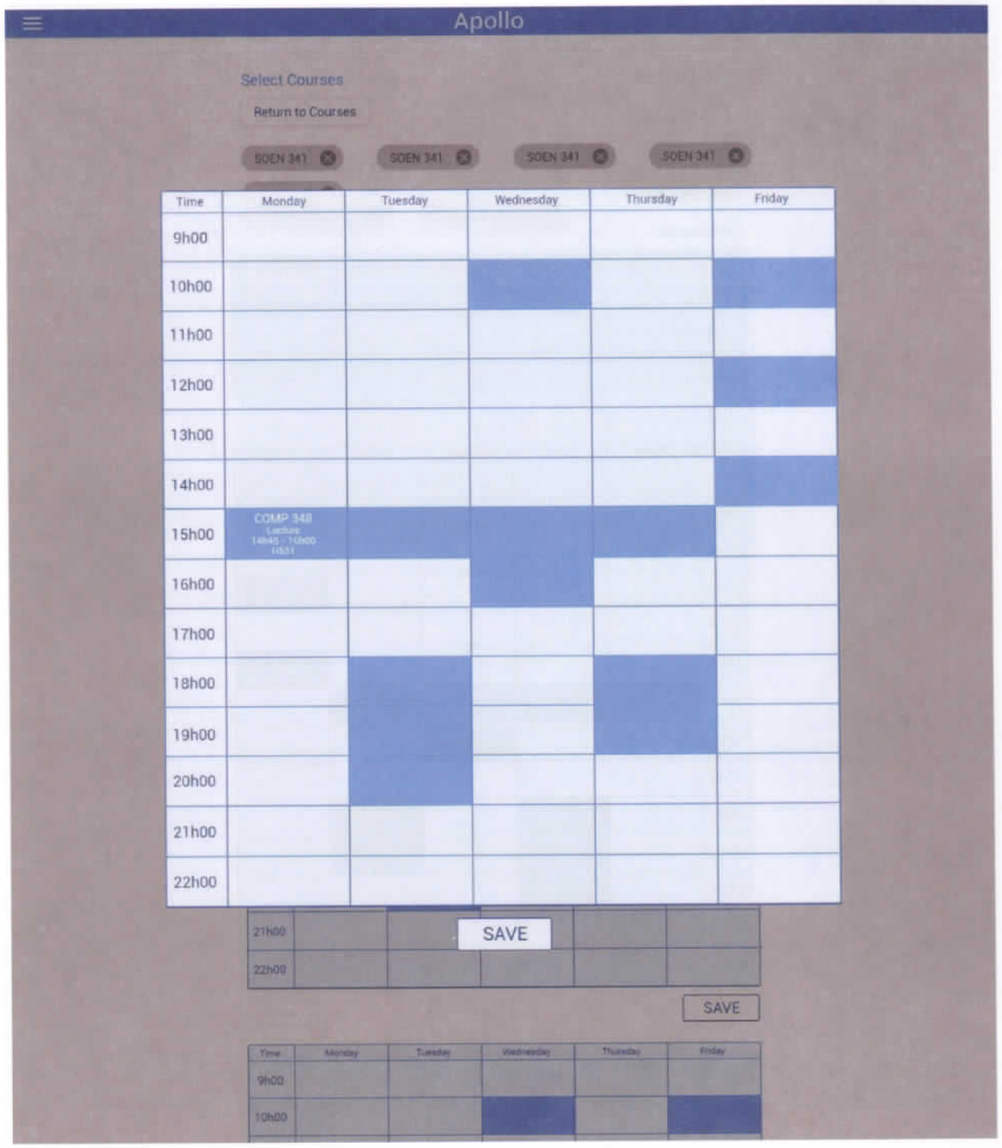


Select Courses Page

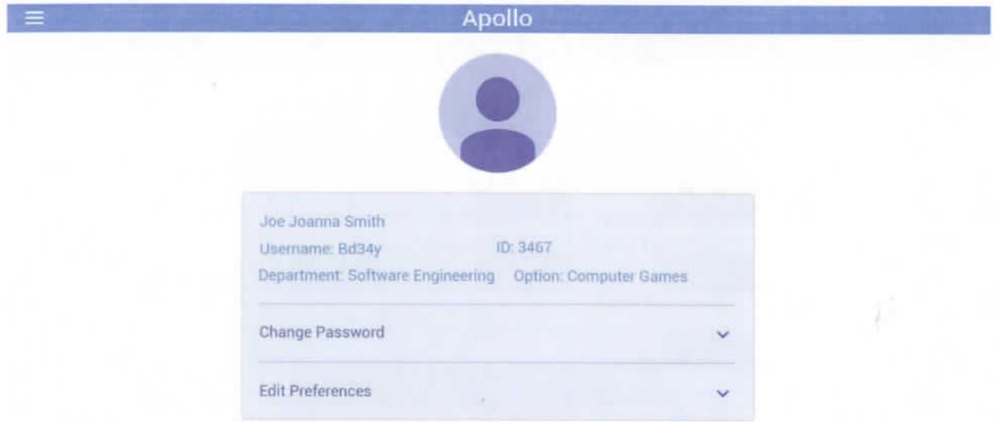


Generated Schedules Page

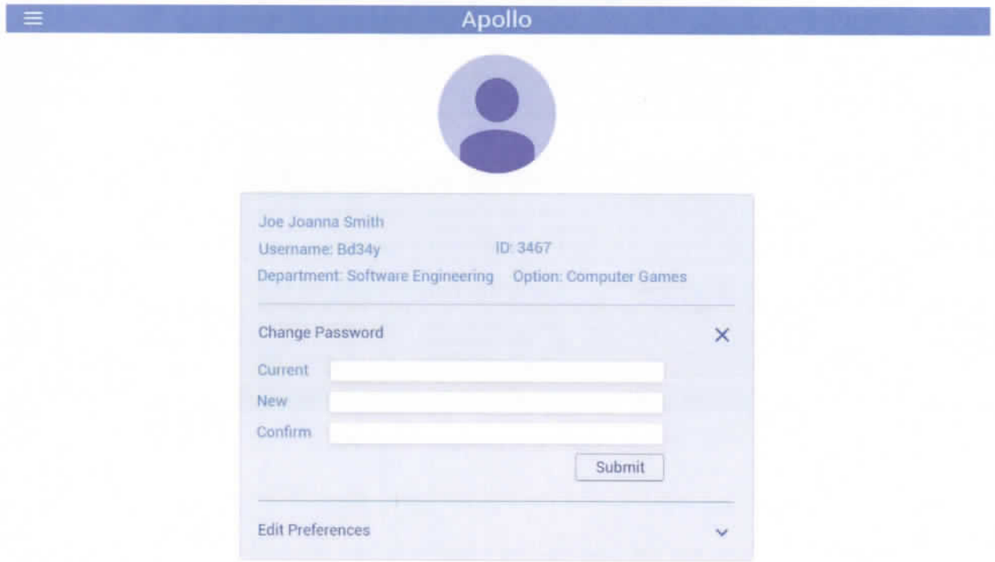
Generated Schedules Overlay View



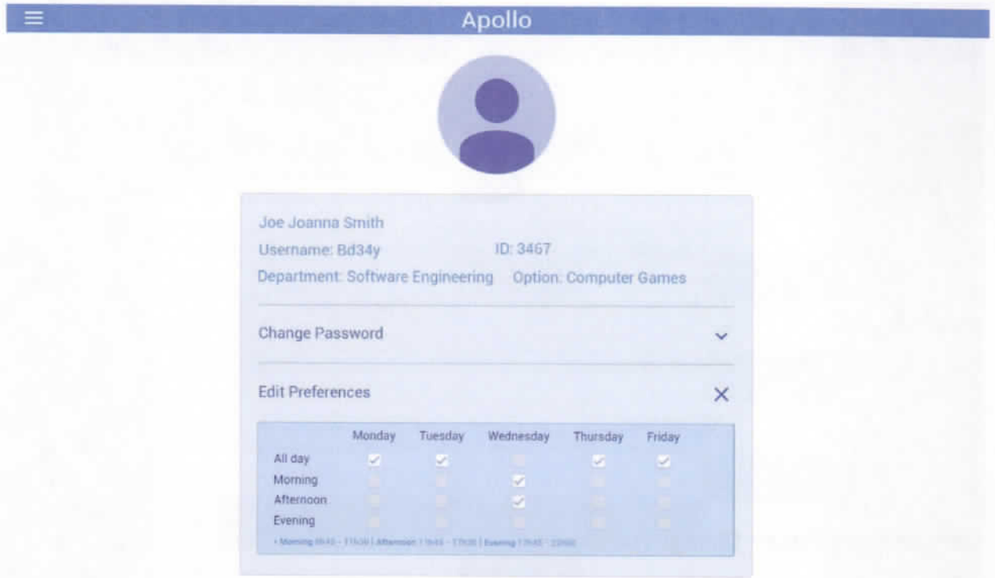
User Settings Page



User Settings Page - Change Password

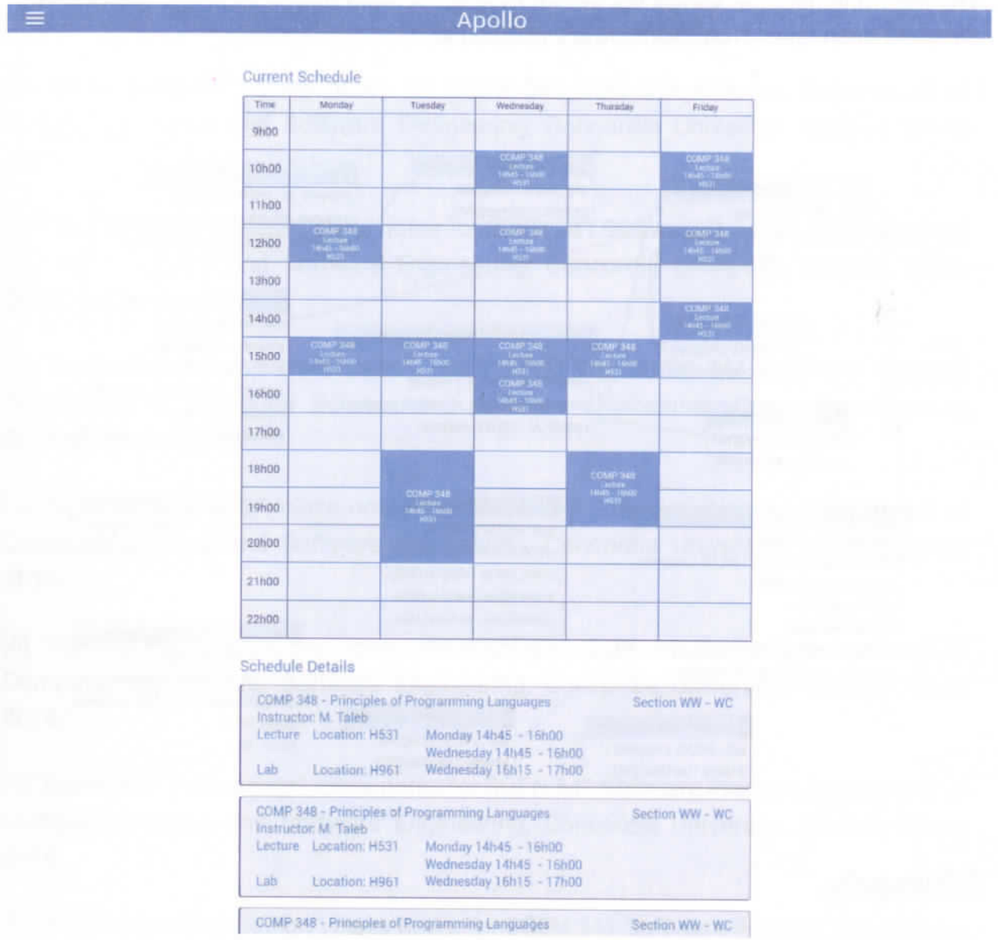


User Settings Page - Edit Preferences



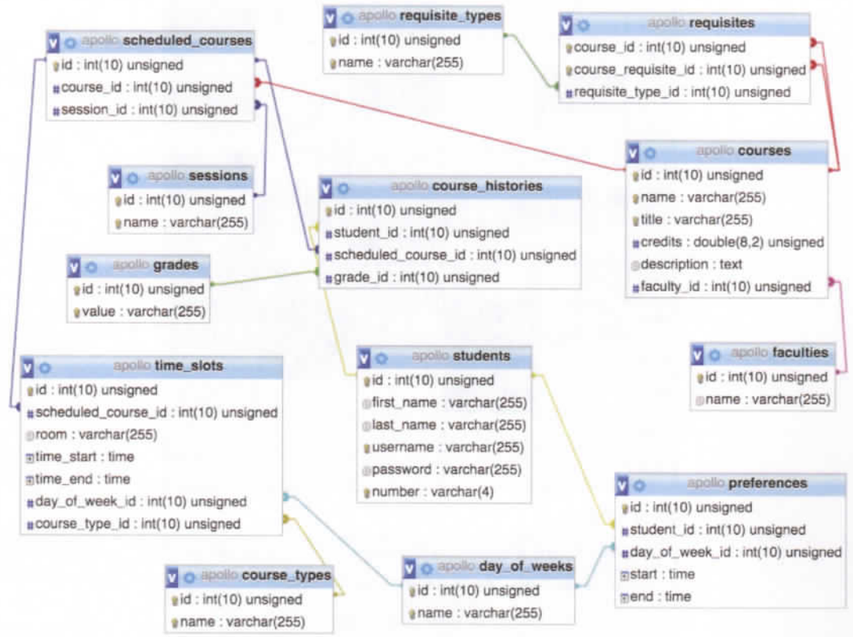
Current Schedule View

Note: This page has been scoped out from implementation.



8.2 Backend

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



8.3 Website

URL: http://apollo.matthewteolis.com/
Username: soen
Password: 341

9. Works Cited

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

[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/