Concordia University

Department of Computer Science

and Software Engineering

Software Process

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

Deliverable 3

April 6TH, 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	*Validity and clarity of the architectural diagrams, as well as of the textual rationale.*	*/5*
3.2	*Validity, completeness, and clarity of description of each component interface.*	*/4*
4.1	*Validity and clarity of the (UML) class diagrams or equivalent for each subsystem, as well as of the textual rationale.*	*/8*
4.2	*Validity and clarity and completeness of the class descriptions for each subsystem.*	*/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.

The purpose of the following document is to test the Apollo application under several situations under which it will be expected to perform accordingly. This also includes the amount of stress the application can handle, as well as how secure the application is. Installation instructions for the application, as well as a user manual on how the system functions will also be provided. Additionally, the final estimation and cost for the development of the software is documented as well.

3. 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.

3.1 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.

3.1.1 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

3.1.2 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.

3.2 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.

3.2.1 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[].

```
describe("Course selection unit tests", function(){
    it("adds a chip when a the addCourse function is called with a course as an argument", function(){
        var course = {
            "id": 1041,
            "name": "Object-Oriented Programming II",
            "number": "249",
            "credits": "3.00",
            "description": "No description of the course.",
            "faculty": {
                "id": 28,
                "name": "COMP"
            },
            "prerequisites": [],
            "corequisites": []
        };
        $scope.addCourse(course);

        expect($scope.selectedCourses[0]).toEqual(course)
    });
});
```


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.

```
describe("Schedule Generator function unit tests", function(){
    it("returns the duration of the class in 15 minute increments", function(){
        var course = [{
                        "faculty": "COMP",
                        "classNum": "249",
                        "title": "Object-Oriented Programming II ",
                        "section": "D",
                        "type": "LEC",
                        "day": "Monday",
                        "timeBegin": "2:45PM",
                        "timeEnd": "6:00PM",
                        "room": "H 420 SGW",
                        "semester": "Fall 2016",
                        "classid": 7469
                    }];
        //2:45 PM to 6:00 PM is 3 hours and 15 minutes, which is 13 X 15 minutes.
        expect($scope.duration[0]).toEqual(13)
    });
});
```

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.

Function *convert12hTo24h(time)*

```
function convert12hTo24h(time){
    var time12 = time.split(':')
    var hour12 = Number(time12[0])
    if((time12[1].indexOf("PM") != -1) && hour12 != 12){
        var hour24 = hour12 + 12
    }
    else{
        var hour24 = hour12
    }
    //round minutes to the nearest 15 minute increment
    var minutes12 = Number(time12[1].slice(0,2))
    if((minutes12 > 0) && (minutes12 < 15)){
        minutes12 = 0
    }
    if((minutes12 > 15) && (minutes12 < 30)){
        minutes12 = 15
    }
    if((minutes12 > 30) && (minutes12 < 45)){
        minutes12 = 30
    }
    if(minutes12 >= 45){
        minutes12 = 45
    }
    var time24 = hour24 + ":" + minutes12
    if(minutes12 == 0){
        time24 = time24 + '0'
    }
    return time24
}
```

this needs function to be tested not the test!

This function converts 12 hours to 24 hours. This would be useful for our course generator since, we would need it to place our courses in the right brackets of the schedule, corresponding to the right time.

Pre-Condition: enter a double number
Post-Condition: returns the time in 24 hours.

this is not a unit test.
this is simulated requirement test

GET course request

```
$scope.courses;
$http({method: 'GET', url: 'http://apollo.matthew.172.31.98.213.xip.io/api/course', headers: {
    'Access-Control-Allow-Origin': '*'}
}).then(function (res) {
    $scope.courses = res.data;
});
```

This function is very important, since it gets all the data from the website and puts it into *courses*. It is a HTTP request that has the method GET to populate the data to courses.

Pre-condition: function requests from website to get data
Post-condition: *courses* gets all the data

Function *classAt (time, day)*

```
$scope.classAt = function(time, day){
    var daySliced = day.slice(0, 2)
    for(var i = 0; i<$scope.course.length; i++){
        if($scope.course[i].day.indexOf(daySliced) > -1){
            var courseHourBegin = Number(convert12hTo24h($scope.course[i].timeBegin).split(':')[0])
            var courseHourEnd = Number(convert12hTo24h($scope.course[i].timeEnd).split(':')[0])
            var timeHour = Number(time.split(':')[0])
            //console.log('course hours starts at: ' + courseHourBegin +' and ends at: ' + courseHourEnd + 'and th
            //check for the hour
            if((courseHourBegin <= timeHour) && (timeHour <= courseHourEnd)){
                if((courseHourBegin != timeHour) && (courseHourEnd != timeHour)){
                    return true
                }
                //check for the minutes
                var courseMinuteBegin = Number(convert12hTo24h($scope.course[i].timeBegin).split(':')[1])
                var courseMinuteEnd = Number(convert12hTo24h($scope.course[i].timeEnd).split(':')[1])
                var timeMinutes = Number(time.split(':')[1])
                if((courseHourBegin == timeHour) && (courseMinuteBegin <= timeMinutes)){
                    return true
                }
                if((courseHourEnd == timeHour) && (courseMinuteEnd >= timeMinutes)){
                    return true
                }
            }
        }
    }
    return false
}
```

this is the easiest the easiest function not the test!

This function provides the scheduler with information in which the class is there at the time and day. It iterates through the array and looks if the class is there.

Post-Condition: enter the time and day in function classAt.
Pre-Condition: returns true if the class is there, returns false if it is not.

3.2.2 Requirements Testing

need > actual input!
so when something is wrong,
someone can replicate your test
to see where's wrong.

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

grease.

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

3.2.3 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 to the server, the average connection duration was approximately 1000 milliseconds. This means that the Apollo applications can serve approximately 100 people per second, and 6000 people per minute.

Connection Setup	TIME
Queueing	6.54 ms
Stalled	7.84 ms
DNS Lookup	0.20 ms
Initial connection	68.78 ms
Request/Response	**TIME**
Request sent	0.11 ms
Waiting (TTFB)	516.95 ms
Content Download	223.61 ms
Explanation	**824.14 ms**

3.2.4 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 <script>alert("spam !")</script&gt, preventing the website from displaying an alert window.

4. System Delivery

4.1 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 IDE. We recommend using PHPStorm by JetBrains.
  - https://www.jetbrains.com/phpstorm/
- 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.
  - 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.
- 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.

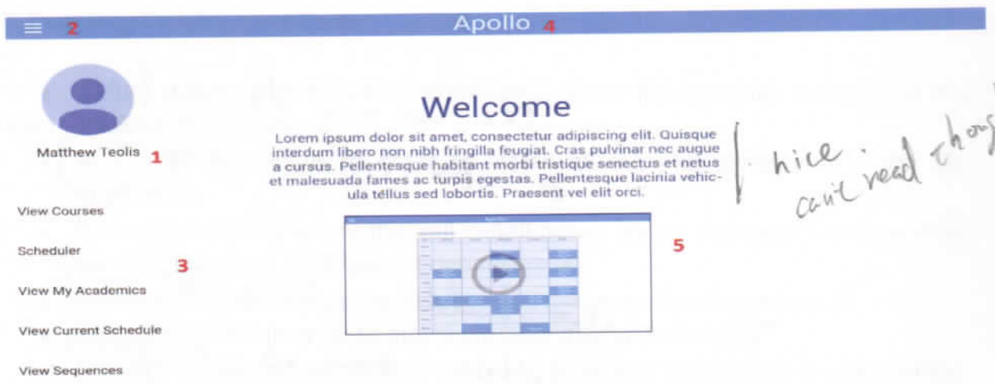
4.2 User Manual

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:

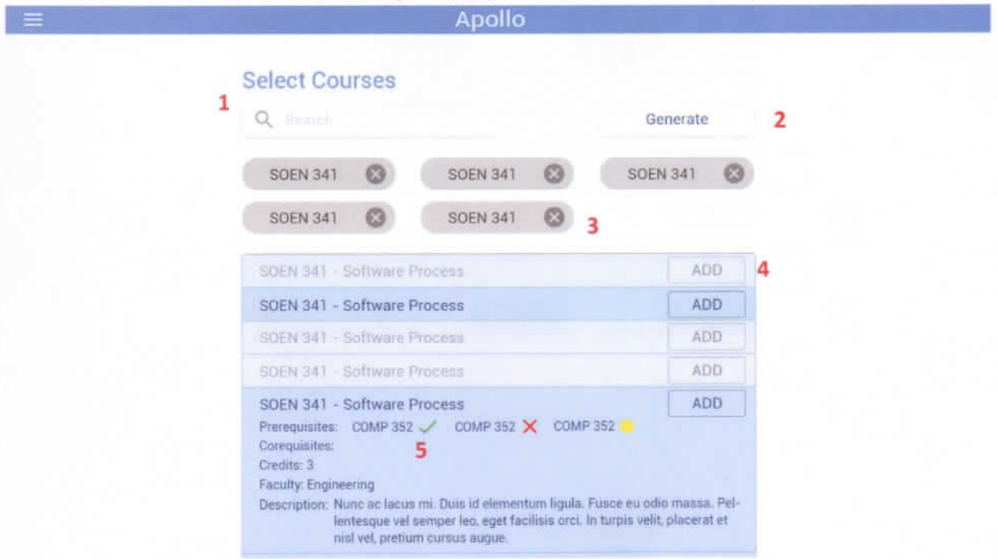
1) **User Setting**
   The user can click on his avatar or name to display their profile information or change his/her password and preferences.
2) **Toggle Sidebar**
   The toggle sidebar is used to hide/show the sidebar. By default, the sidebar is displayed on large devices such as computer screens or tablets, and it is hidden on smaller devices such as smartphones.
3) **Sidebar**
   The sidebar is used to navigate to various other sections of the application.
4) **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
5) **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.
2) **Generate**
   The Generate button, once pressed, will display the generated schedules using the selected classes to the student. (See "Generated Schedules" below)
3) **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.
4) **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.
5) **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.

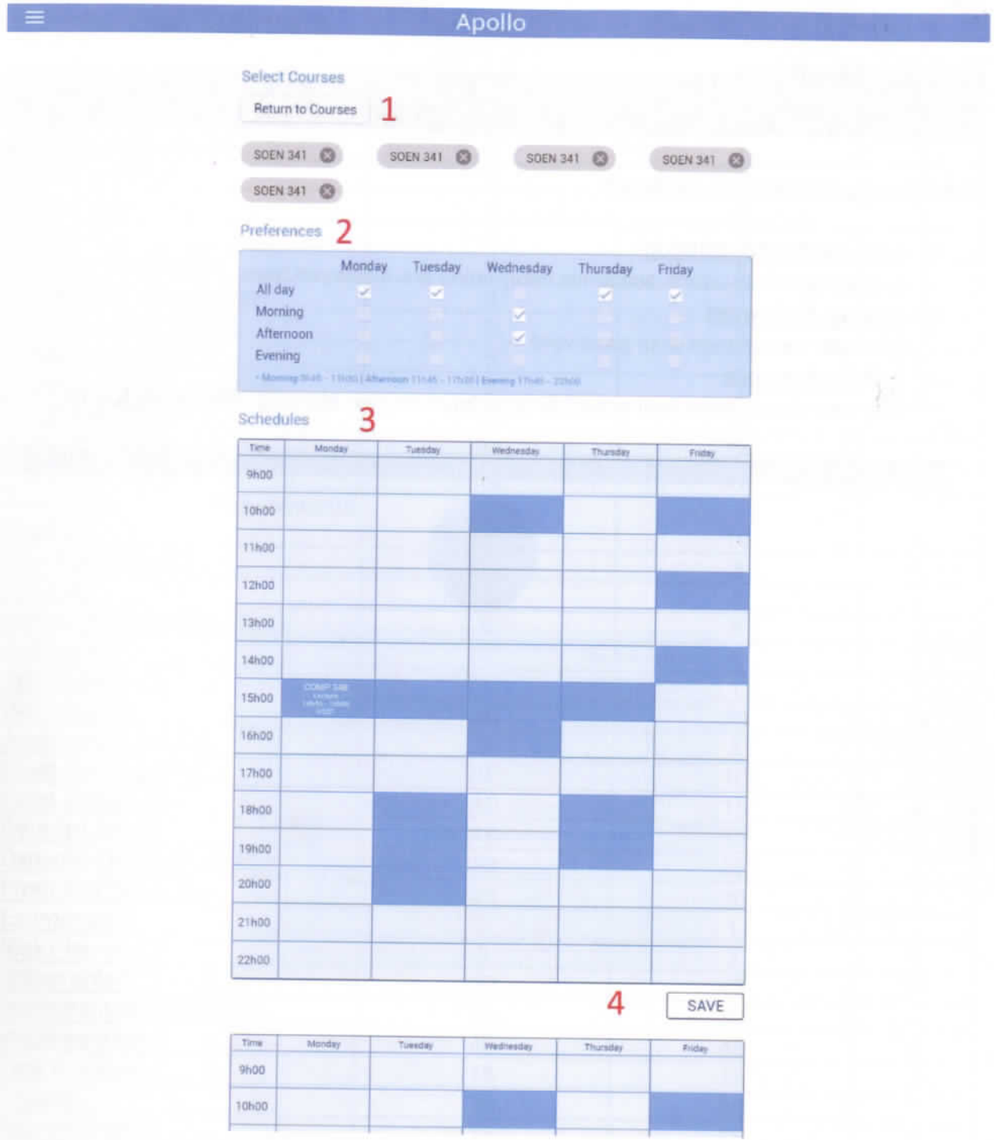


Generated Schedules

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**
   a) 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**
   a) 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**
   a) Every possible schedule combinations will be displayed as a table.
4) **Save Schedule**
   a) 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.

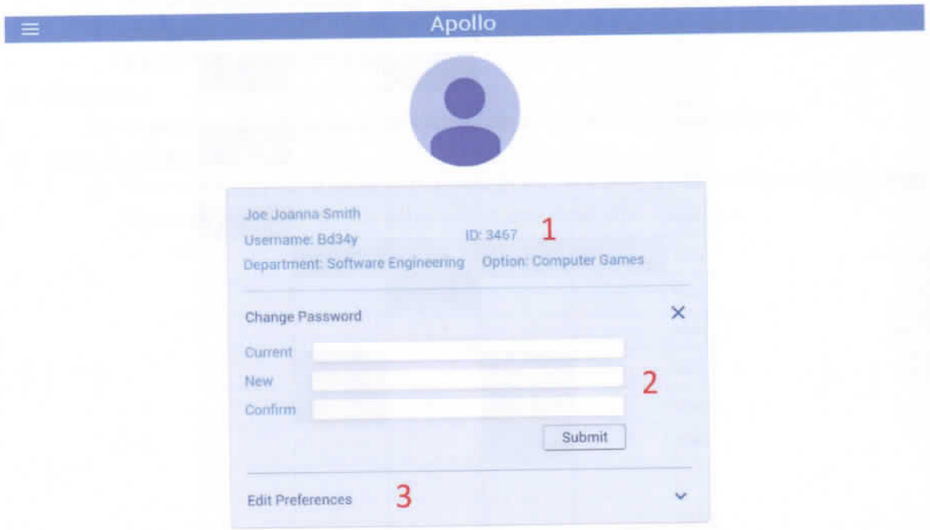


Edit Preferences

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.
2) **Change Password**
   The user can change their password.
3) **Edit Preferences**
   The user can change their scheduling preferences (ex. Morning, Afternoon, Night).



5. Final Cost Estimate

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**

6. Works Cited

[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