Department of Computer Science & Software Engineering

Software Process: SOEN341/4S---2016

Scheduler Builder

The Schedulers	
Emili Vasseva	27526741
Sean Marcoux	27511876
Bruce Edouard Brazier	27419562
Dias Marat	27277911
Le Vinh Dang	26844987
Adriel Fabella	27466005
Gabriele Bavaro	27399103
Ying-Chen Chu	27415710
Alex Eladas	27041462
Salma Aly	27176414
Adil Hssaini	24832396
Nicolas Frazer-McKee	27068956

Table of Contents

Part I: Project Scope & Plan Document....1
1. Introduction....2
2. Project Description....2
3. Goals & Constraints....3
3.1 Functional Requirements....3
3.1.1 Login....3
3.1.2 Registration....3
3.1.3 Account Management....3
3.1.4 Record Management....3
3.1.5 Course Selection....4
3.1.6 Schedule Preferences....4
3.1.7 Schedule Generation....4
3.1.8 Logout....4
3.1.9 Administrator....4
3.2 Domain Model....26
3.3 Constraints & Qualities....28
3.3.1 Product....28
3.3.2 Organizational....29
3.3.3 External....29
4. Resource Evaluation....30
4.1 Human resources....30
4.2 Technical Resources....34
4.2.1 Login....34
4.2.2 Registration....34
4.2.3 Account Management....35
4.2.4 Login....35
4.2.5 Registration....35
5. Scoping....35
5.1 Scoped In....36
5.2 Scoped Out....36
6. Solution Sketch....37
6.1 Architecture....37
6.1.1 Server Side....38

6.1.2 Client Side 39
6.2 Technology in Use 39
6.2.1 Programming Languages 39
6.2.2 Framework 40
6.2.3 Integrated Development Environments/Editors 40
6.2.4 Source Code and Revision Management 40
6.2.5 Collaboration Software 41
7. Plan 42
7.1 Activities, Artifacts & Activities Assignments 42
7.1.1 Deliverable 0: System Overview 42
7.1.2 Deliverable 1: Requirements, Scope and Plan 42
7.1.3 Deliverable 2: Design 44
7.1.4 Deliverable 3: Testing 46
7.1.5 Final Deliverable: Complete Report 48
7.2 Project Estimates 48
7.2.1 Basis for Estimates 48
7.3 Schedule 49
7.4 Risks 50
7.4.1 Knowledge of Frameworks and Programming Languages 50
7.4.2 Time 50
7.4.3 Security 50
7.4.4 Schedule Errors 50
7.4.5 Teamwork 50
8. Prototyping 51
8.1 Mock-Ups 51
8.1.1 Login Page 51
8.1.2 Preference Page 51
8.1.3 Schedule Page 52
8.1.4 Account Management Page 53
8.2 Prototype 53
8.2.1 Login Page 54
8.2.2 Preference Page 55
Part II: Architecture & Design 56
1. Introduction 57
2. Architectural Design 57

2.1 Architectural Diagram 58
2.1.1 Logical View 58
2.1.2 Development View 60
2.1.3 Physical View 61
2.1.4 Process View 62
2.1.5 Scenarios 65
2.2 Subsystem Interfaces Specifications 68
2.2.1 ManageCourses 68
2.2.2 PreferenceSettings 69
2.2.3 UserVerifications 69
2.2.4 AdminDataBaseModifications 71
2.2.5 CourseAvailability 72
2.2.6 PreferenceImplementation 74
2.2.7 ScheduleGeneration 75
3. Detailed Design 77
3.1 Detailed Design Diagram 77
3.1.1 Student Generate Schedule Subsystem 78
3.1.2 Student Manage Courses Subsystem 80
3.1.3 Student Manage Preferences Subsystem 83
3.1.4 Admin Manage Course Database Subsystem 84
3.1.5 Admin and Student Account Authentication Subsystem 86
3.1.6 Admin and Student Managing Account Information Subsystem 87
3.2 Unit Descriptions 88
4. Dynamic Design Scenarios 98
4.1 Generate Schedule 98
4.1.1 Full Use Case 98
4.1.2 System Sequence Diagram 99
4.2 Set Needed Course 100
4.2.1 Full Use Case 100
4.2.2 System Sequence Diagram 101
4.3 Set Preferences 103
4.3.1 Full Use Case 103
4.3.2 System Sequence Diagram 103
4.4 Add Course to Program 104
4.4.1 Full Use Case 104

4.4.2 System Sequence Diagram 105
5. Estimation 107
6. Rapid Prototyping and Risk 108
6.1 Risk 108
6.2 Front-End Work 109
6.3 Back-End Work 111
6.3.1 Front-End and Back-End Communication Issues 111
6.4 Added Technology 111
6.4.1 Twitter Typeahead 111
Part III: Testing & Delivery 113
1. Introduction 114
2. Testing Report 114
2.1 Test Coverage 114
2.1.1 Tested Items 115
2.1.2 Untested Items of Interest 116
2.2 Test Cases 117
2.2.1 Unit Testing 117
2.2.2 Requirements Testing 124
2.2.3 Stress Testing 136
2.2.4 Security Testing 143
3. System Delivery 144
3.1 Installation Manual 144
3.2 User Manual 154
4. Final Cost Estimate 162
5. Programming Specifications 165
5.1 Open-Source Technology Used 165
5.1.1 Twitter Typeahead 165
5.1.2 JQuery Week Calendar 166
5.1.3 React Bootstrap 167
5.2 Encryption & Security 168
5.3 Server Calls 168

Department of Computer Science & Software Engineering

Software Process: SOEN341/4S---2016

Part I:

Project Scope and Plan Document

The Schedulers	
Emili Vasseva	27526741
Sean Marcoux	27511876
Bruce Edouard Brazier	27419562
Dias Marat	27277911
Le Vinh Dang	26844987
Adriel Fabella	27466005
Gabriele Bavaro	27399103
Ying-Chen Chu	27415710
Alex Eladas	27041462
Salma Aly	27176414
Adil Hssaini	24832396
Nicolas Frazer-McKee	27068956

1. Introduction

Undergraduate students of the Software engineering program have the time consuming task of completing their schedule for each semester. This requires a planned out process to make sure they follow their sequence while not having any conflicts with the courses. This is usually done by trying numerous times to have a working schedule, as in most cases, the registration doesn't go as smoothly as one desires. The preferences in the schedule comes as an afterthought as it is quite complicated to compare all the courses and specifically register for certain times and days.

The Scheduler application is a web application looking to automate all these decisions for the Software engineering students. This provides an effective and pleasant to generate a schedule for their remaining years of study, all the while following their sequence and appealing to their preferences. This will be done through a web-based interface taking where students will have to sign up and input their information.

This document goes through the whole development process of the system. It contains the requirements of the system, the planning, the design phases and architecture, the implementation of the system and finally the testing. Estimations of the development's cost is also provided.

2. Project Description

The following document describes the fundamental scope and plan of a group project to be done in the Software engineering course, Software Process (341). The purpose of this project is to create an interactive multi-year schedule builder for software engineering students. In addition, being a team project, it emphasizes on the value of team cooperation and organization. The team name is "the Schedulers" and is composed of 12 students, having specific roles of front-end programmer, back-end programmer and documenters.

The document provides a foundation to follow when implementing the system. It contains the system's functionality, meaning requirements and behaviors, and the resources and technologies used to achieve the end goal.

The overall goal of this project is to create a system that can generate a schedule. The application should allow a user to log in to the network after signing up. It shall also invite the user to add his/her classes and display his/her preferences, for example: choosing only night classes. The system shall create a schedule, based on the inputs of the user, to be displayed on screen.

3. Goals and Constraints

3.1 Functional Requirements

3.1.1. Login

1.1 The system shall allow users to login using valid login credentials.

3.1.2. Registration

2.1 A new user shall be able to create an account on the scheduler system.

~~2.2 A user shall be able to view the default schedule prior to registration~~

3.1.3. Account Management

3.1 The system shall allow the user to view all account information within 2 seconds.

3.2 The user may modify mutable account information.

3.3 The system may generate a confirmation message describing account information changes.

3.4 The system shall update the user's database with the account information changes.

3.5 The user shall be able to reset their password.

3.1.4. Record Management

4.1 Upon first time login, the system shall allow the user the creation of a student record, based on the set of courses previously taken.

4.2 The user shall be able to update and modify the information provided initially as well as any information entered subsequently.

4.3 The system shall store the information entered during the creation or the modification of the record once the student commits to saving.

~~4.4 The system shall discard all entered information if the user's session is terminated or interrupted prior to committing to save~~.

4.5 The system shall allow the user to view the saved student record at any time during an active session.

~~4.6 The user may delete the existing student record at any time during an active session, and may recreate a new one~~.

4.7 The system shall allow the existence of one and only one student record per user account.

4.8 The system shall be able to generate a list of the courses taken based on the number of semesters finished by the student.

4.9 The user shall be able to add update their taken classes based on the auto generated list of taken courses.

3.1.5. Course Selection

5.1 The system shall allow registered ~~and unregistered~~ users alike to view the list of all courses and their respective descriptions and schedules.

5.2 The system shall allow the user to add needed courses for the generation of a schedule.

5.3 The system shall allow the user to remove previously added needed courses.

3.1.6. Schedule Preferences

6.1 The system shall allow the user to set his preferences.

6.2 The system shall allow the user to modify his previously set preferences.

3.1.7. Schedule Generation

7.1 The system shall allow the user to generate a schedule according to the courses added with or without the preferences

7.2 The system shall allow the user to modify options concerning the generation of the schedule

~~7.3 The system shall allow the user to save a copy of the generated schedule~~

~~7.4 The system shall allow the user to delete saved schedules~~

~~7.5 The system shall allow the user to print a generated schedule~~

7.6 The system shall allow the user to view a generated schedule

~~7.7 The system shall allow the user to change the section for an added course.~~

3.1.8. Logout

8.1 The user shall be able to logout from the system and close the current session.

3.1.9. Administrator

9.1 The administrator shall be able to add one or more courses to all programs.

9.2 The administrator shall be able to edit all courses.

9.3 The administrator shall be able to remove one or more courses from all programs.

9.4 The administrator shall be able to add one or more sections to all courses.

9.5 The administrator shall be able to edit all sections.

9.6 The administrator shall be able to delete one or more sections from all courses.

~~9.7 The system shall save any and all modifications performed by the administrator, only if the commit to changes operation terminates successfully.~~

~~9.8 The system shall discard all changes if the administrator's session is terminated unexpectedly.~~

Figure 1: *Administrator, Student, and Public User UCD*

Figure 2: *Administrator Use Case Diagram*

Name:	Login	Author	Salma Aly
Identifier:	UC1	Version:	2.0
Date Created:	Jan 30, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student or Administrator		
Goal:	To access the user's profile in the scheduler system		
Summary:	The user enters their username and password which will be validated by the system in order to give access to the user and retrieve their information.		
Related use-cases:			
Preconditions	1. The user has successfully landed on the login page 2. The user is not already logged in 3. The user has already signed up (created an account) on the system.		
Trigger:	User prompts system to validate login information		
Basic Flow:	1. User enters username and password and indicates the wish to enter by prompting for log in. 3. System determines validity of username and password. 4. System redirects user to preferences page		
Post-Conditions:	**Success:** The user is logged in and has access to their profile		
Minimum Guarantee:	1. The user can still have access to the login page 2. The system remains protected from unauthorized access. **Failure:** System fails to authenticate user and displays an error message.		
Risk Assessment:	Low		

Name:	Logout	Author	Salma Aly
Identifier:	UC2	Version:	2.0
Date Created:	Jan 30, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student or Administrator		
Goal:	To end and sign-out from the current session		
Summary:	Using the sign-out option, the user attempts to terminate and log out from the current session.		
Related use-cases:	UC1 - Login		

Preconditions	The user is already logged in
Trigger:	User selects Sign out option
Basic Flow:	1. The user indicates the wish to sign out by selecting the option. 2. System terminates the existing session 3. System redirects user to login page
Post-Conditions:	**Success:** Current session is terminated and user is successfully logged out
Minimum Guarantee:	The user's last activity is saved. **Failure:** System fails to terminate the session and an error message is displayed.
Risk Assessment:	Low

Name:	Reset Password	Author	Salma Aly
Identifier:	UC3	Version:	3.0
Date Created:	Jan 30, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student		
Goal:	To reset password		
Summary:	The system sends the user an email with a password reset link that allows them to login and access the change password section of the system.		
Related use-cases:	UC19		
Preconditions	1.User already registered in the system 2.User has provided a valid email address		
Trigger:	The user initiates a password reset		
Basic Flow:	1. User indicates the wish for a password reset by selecting the "Forgot Password" option. 2. System prompts to enter username 3. User inputs username and selects send email from the pop-up menu. 4. System verifies if account is associated with a valid email address. 5. System displays notification informing user that a confirmation email was sent to the email address associated with the account. 6. User confirms reception by clicking on the reset password link enclosed in the received email. 7. System redirects user to account information page & prompts for new password. 8. System updates account information with new password via UC19.		
Post-Conditions:	**Success:** Password is changed successfully.		
Minimum Guarantee:	The user can view the login page with reset password option. **Failure:** The password remains unchanged and an alert is sent to the user.		
Risk Assessment:	Low		

Name:	Signup	Author	Salma Aly
Identifier:	UC4	Version:	2.0
Date Created:	Jan 30, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Public User		
Goal:	To create an account on the scheduler system		
Summary:	The user signs up to create an account on the system by filling information to setup their account.		
Related use-cases:			
Preconditions	The user has successfully landed on the Login/Signup page		
Trigger:	User activates the "Sign up" process		
Basic Flow:	1.User indicates wish to sign up by selecting the create account option 2. System displays new window prompting for username, email, password & password confirmation. 3. User enters personal information. 4. System verifies integrity, completeness & non-duplicity of provided information. 6. System adds user to list of registered users. 7. System redirects user to main preferences page.		
Post-Conditions:	**Success:** The User has been added to the list of registered users on the system		
Minimum Guarantee:	The user is redirected to the Login/Signup page **Failure:** The user account is not created and an appropriate error message is displayed.		
Risk Assessment:	Low		

~~Name:~~	~~View Default Schedule~~	~~Author~~	~~Salma Aly~~
~~Identifier:~~	~~UC5~~	~~Version:~~	~~2.0~~
~~Date Created:~~	~~Feb 2, 2016~~	~~Last Modified:~~	~~April 12, 2016~~
~~Importance:~~	~~2/5~~		
~~Actor(s):~~	~~Student or Public User~~		
~~Goal:~~	~~To view the default generated schedule for in sequence student~~		
~~Summary:~~	~~Students have the option of viewing the default schedule created by the scheduling system without the need to login or signup to the system.~~		
~~Related use-cases:~~			
~~Preconditions~~	~~The user has successfully accessed the landing page.~~		

Trigger:	The user loads a generic automated schedule
Basic Flow:	1. User indicates wish to view the available default schedule. 2. System displays the default schedule
Post-Conditions:	**Success**: The user successfully viewed the default schedule
Minimum Guarantee:	Landing page is still accessible to the user with all system functionalities intact. **Failure**: System fails to generate schedule and error message is displayed.
Risk Assessment:	Low

Name:	Modify Schedule	Author	Nicolas Frazer-McKee
Identifier:	UC6	Version	1.1
Date Created:	8/2/2016	Last Modified	9/2/2016
Importance:	5/5		
Actor(s):	Student		
Goal:	Access modification feature for current schedule		
Summary:	The Student may edit their current schedule.		
Related use cases:	UC9, UC10, UC15, UC18.		
Preconditions	1. User has been authenticated 2. Schedule has been generated		
Trigger:	User chooses to edit current schedule		
Basic Flow:	1. User indicates wish to modify schedule via UC9 and/or UC10, UC15, UC18. 2. System updates account with new schedule.		
Post-Conditions:	**Success:** The modified schedule information is saved in the system		
Minimum Guarantee:	The system lets the user view account information **Failure:** The System cannot alter the user's schedule information		
Risk:	The information may not save properly.		

Name:	Auto Generate List of Taken courses	Author	Salma Aly
Identifier:	UC7	Version:	2.0
Date Created:	Feb 7, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student		
Goal:	To View the list of courses that should have been completed within the number of semesters the user has finished.		
Summary:	User enters the number of semesters finished so far and the scheduler generates the list of courses that correspond to the semesters taken based on the course sequence.		
Related use-cases:			
Preconditions	1. The user has a valid account on the system 2. The user is logged in to their account 3. The user has finished at least one semester		
Trigger:	User prompts system to validate login information		
Basic Flow:	1. User indicates the wish to auto generate the list of taken courses. 2. System prompts for number of semesters taken. 3. User enters corresponding number and selects “Generate Course List”. 4. System displays pop-up prompting for confirmation. 5. User provides confirmation. 6. System generates list of courses. 7. System adds courses to list of taken courses.		
Post-Conditions:	**Success:** A list of courses corresponding to the number of semesters finished appears.		
Minimum Guarantee:	All other account information remain unchanged. **Failure:** System fails to complete task & an error message is displayed.		
Risk Assessment:	Low		

Name:	Set Taken Courses	Author	Adil Hssaini
Identifier:	UC8	Version:	1.0
Date Created:	Feb 4, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Registered User / Administrator		
Goal:	To add a course to the list of previously taken courses be stored by the system		
Summary:	The user provides the system with a set of previously taken courses. The system stores that information and uses it to create a record for the specified account.		

Related use-cases:	
Preconditions	1. User is logged on. 2. System has access to the list of taken courses associated with the account.
Trigger:	User activates the “Set Taken Courses” process
Basic Flow:	1. User indicates the wish to add a course by selection the option. 2. System prompts for course name or number. 3. User enters course information. 4. User commits to by selecting the “add class” option. 5. System verifies non-duplicity of course in the list of taken courses. 6. System generates corresponding course name/number to entry if either is missing. 7. System adds course to list of taken courses.
Post-Conditions:	**Success:** Student record is updated successfully.
Minimum Guarantee:	All other system data, configurations and functionalities remain unchanged. **Failure:** System fails to process task and error message is displayed.
Risk Assessment:	Low

Name:	Set Needed Courses	Author:	Ying-Chen Chu
Identifier:	UC9	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Add a needed course to the schedule		
Summary:	Select a course to be added to the list of courses for the schedule generator.		
Related use-cases:	UC1		
Preconditions	1. User has been authenticated 2. Course requirements are met		
Trigger:	User selects the set needed course process.		
Basic Flow:	1. User indicates the wish to add a course by selecting a course from the needed courses list. 2. User commits by selecting the “add class” option. 3. System checks for eligibility and time conflict and adds course to list of needed courses.		
Post-Conditions:	**Success:** Course is added to the user's list of courses		
Minimum Guarantee:	Previous state of the system remains unchanged. **Failure:** System fails to process task and displays an error message.		
Risk Assessment:	Low		

Name:	Delete Needed Courses	Author:	Ying-Chen Chu
Identifier:	UC10	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Student, Administrator		
Goal:	Remove a course from the schedule		
Summary:	Remove a course from the list of selected courses for the schedule generator.		
Related use-cases:	UC1		
Preconditions	1. User has been authenticated 2. System is able to access to the list of needed courses' menu.		
Trigger:	User activates the delete needed course process.		
Basic Flow:	1. User indicates wish to delete a course from the list of taken courses. ~~2. System prompts for confirmation to delete.~~ ~~3. User commits to deleting~~ 4. System removes the selected course from the list & updates the account.		
Post-Conditions:	**Success:** Course is removed from the user's list of courses		
Minimum Guarantee:	All other account information remain unchanged. **Failure**: System fails to process task and displays an error message		
Risk Assessment:	Low		

Name:	Delete Taken Courses	Author	Adil Hssaini
Identifier:	UC11	Version:	1.0
Date Created:	Feb 4, 2016	Last Modified:	Feb 9, 2016
Importance:	2/5		
Actor(s):	Registered User / Administrator		
Goal:	To delete an existing course from the student record.		
Summary:	The user deletes the existing student record associated with account. The system has no record associated with the account afterwards.		
Related use-cases:			
Preconditions	1. User is logged on. 3. System has accessed the list of taken courses' menu.		
Trigger:	User activates the delete taken courses process		

Basic Flow:	1. User indicates wish to delete a course by selecting the corresponding option. ~~2. System prompts for confirmation to delete.~~ ~~3. User commits to deleting~~. 4. System responds by removing the course from the list of taken courses.
Post-Conditions:	**Success:** Student record is deleted successfully.
Minimum Guarantee:	All other system data, configurations and functionalities remain unchanged. **Failure:** System fails to process task and displays an error message.
Risk Assessment:	Low

Name:	Set Preferences	Author	Ying-Chen Chu
Identifier:	UC12	Version:	3.0
Date Created:	2015-02-03	Last Modified:	2015-02-08
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Input the schedule preferences to the system		
Summary:	Set the schedule preferences and save them to the system.		
Related use-cases:	UC1		
Preconditions	User has been authenticated		
Trigger:	Previous state of the system remains unchanged.		
Basic Flow:	1. User indicates the wish to set schedule preferences.by selecting the desired “Day Off” and “Preferred Time” from drop down menus. 2. System registers selections & displays schedule preferences.		
Post-Conditions:	**Success:** The user’s preferences are saved to the system.		
Minimum Guarantee:	Previous state of the system remains unchanged. **Failure:** System fails to process the task, and an error messages is displayed.		
Risk Assessment:	Low		

Name:	Generate Schedule	Author	Ying-Chen Chu

Identifier:	UC13	Version:	3.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Generate a schedule		
Summary:	Based on the course selection list and or without the list of preferences, the system generates schedule(s) for the selected courses.		
Related use-cases:	UC8, UC9, UC12		
Preconditions	1. User has been authenticated. 2. Schedule preferences already set.		
Trigger:	User activates the "Generate Schedule" process		
Basic Flow:	1. User indicates the wish to generate a schedule.by selecting the “Build Schedule” option. 2. System verifies entries in lists of taken courses, needed courses & schedule preferences saved to the system via UC8, UC9 & UC11 and displays schedule.		
Post-Conditions:	**Success:** Schedule with the selected courses is generated.		
Minimum Guarantee:	Previous state of the system remains the same. **Failure:** System fails to generate a schedule and an error message displays		
Risk Assessment:	High		

~~Name:~~	~~Save Schedule~~	Author	Ying-Chen Chu
~~Identifier:~~	~~UC14~~	Version:	~~2.0~~
~~Date Created:~~	~~2015-02-03~~	Last Modified:	~~April 12, 2016~~
~~Importance:~~	~~5/5~~		
~~Actor(s):~~	~~Student, Administrator~~		
~~Goal:~~	~~Save a generated schedule~~		
~~Summary:~~	~~Save a schedule generated by the system by keeping a copy of the schedule on the system, accessible to the user.~~		
~~Related use-cases:~~	~~UC "**View Schedule**" and UC "**Save Schedule**"~~		
~~Preconditions~~	~~1. User has been authenticated~~ ~~2. Schedule(s) were generated~~		
~~Trigger:~~	~~Users activates the "save schedule" process.~~		
~~Basic Flow:~~	~~1. User indicates wish to save the schedules by selecting the option.~~		

	2. System responds with the outcome of the operation.
Post-Conditions:	**Success:** Saved schedules are accessible to the user from the system
Minimum Guarantee:	Previous state of the system remains the same. **Failure:** System fails to save the schedule, and an error message displays.
Risk Assessment:	Low

Name:	Delete Schedule	Author	Ying Chen Chu
Identifier:	UC15	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Delete a saved schedule		
Summary:	Delete a schedule from the list of saved schedules		
Related use-cases:	UC "**Modify Schedule**"		
Preconditions	1. User has been authenticated 2. Schedule has been generated.		
Trigger:	Users activates "delete schedule" process.		
Basic Flow:	1. User indicated wish to delete the schedule by selecting the option. 2. System responds with the outcome of the operation.		
Post-Conditions:	**Success:** Deleted schedules are no longer saved, nor accessible through the system.		
Minimum Guarantee:	Previous state remains the same: schedule is not deleted. **Failure:** System fails to delete schedule and an error message is displayed.		
Risk Assessment:	Low		

Name:	Print Schedule	Author	Ying Chen Chu

Identifier:	UC16	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	3/5		
Actor(s):	Student, Administrator		
Goal:	Print a saved or generated schedule		
Summary:	Print a schedule generated by the system. The schedule has either just been generated or can be accessed through the list of saved schedules.		
Related use-cases:	UC "**View Schedule**"		
Preconditions	1. User has been authenticated. 2. Schedule has been generated.		
Trigger:	Users activates "print schedule" process.		
Basic Flow:	1. User indicates wish to print the schedule by selecting the option. 2. System responds with the outcome of the operation.		
Post-Conditions:	**Success:** The schedule is printed successfully.		
Minimum Guarantee:	Previous state of the system remains unchanged: schedule remains the same. **Failure**: Failure to print the schedule, and a failure message is displayed.		
Risk Assessment:	Medium		

Name:	View Schedule	Author	Ying-Chen Chu
Identifier:	UC17	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	View a saved schedule		
Summary:	View a schedule from the list of saved schedules		
Related use-cases:	UC "**Generate Schedule**"		
Preconditions	1. User has been authenticated 2. At least one schedule in the list of saved schedules		
Trigger:	Users activates "view schedule" process.		
Basic Flow:	1. User indicates wish to view schedule by selecting the option. 2. System responds with the outcome of the operation.		
Post-Conditions:	**Success:** The user is able to view the selected schedule.		

~~Minimum Guarantee:~~	~~Previous state of the system remains the same~~ ~~**Failure:** System fails to display the schedule, and an error message is displayed.~~
~~Risk Assessment:~~	~~Low~~

~~Name:~~	~~Change Section~~	~~Author~~	~~Ying Chen Chu~~
~~Identifier:~~	~~UC18~~	~~Version:~~	~~2.0~~
~~Date Created:~~	~~2015-02-03~~	~~Last Modified:~~	~~April 12, 2016~~
~~Importance:~~	~~5/5~~		
~~Actor(s):~~	~~Student, Administrator~~		
~~Goal:~~	~~Change course section~~		
~~Summary:~~	~~Change the course section for a selected course from the list of selected courses.~~		
~~Related use cases:~~	~~UC "**Modify Schedule**"~~		
~~Preconditions~~	~~1. User has been authenticated~~ ~~2. At least one course has been selected~~		
~~Trigger:~~	~~User activates the "change section" process.~~		
~~Basic Flow:~~	~~1. User indicates wish to change course section.~~ ~~2. System displays list of available sections~~ ~~2. User selects new section from the list.~~ ~~3. System responds with outcome of operation.~~		
~~Post-Conditions:~~	~~**Success:** The user is removed from the previous section and added to the new section.~~		
~~Minimum Guarantee:~~	~~Schedule remains the same~~ ~~**Failure:** System fails to process task and an error message is displayed.~~		
~~Risk Assessment:~~	~~Medium~~		

Name:	Edit Account Information	Author	Adil Hssaini

Identifier:	UC19	Version:	2.0
Date Created:	Feb 4, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Registered User / Administrator		
Goal:	To edit an existing account information.		
Summary:	The user adds or modifies entries into the existing account information. The system stores the newly entered information and updates account.		
Related use-cases:			
Preconditions	1. User is logged on. 2. System has access to the account menu.		
Trigger:	User activates the “Edit Account Information” process.		
Basic Flow:	1. User indicates wish to edit the account Information. 2. System prompts for new username, password & Email. 3. User enters desired new information. 4. User commits by saving. 5. System verifies validity, integrity & non duplicity of provided information. 6. System updates account with new information.		
Post-Conditions:	**Success:** Student record is updated with the new information successfully.		
Minimum Guarantee:	All other system data, configurations and functionalities remain unchanged. **Failure:** System fails to process task and displays an error message.		
Risk Assessment:	Low		

~~Name:~~	~~Drop course~~	~~Author:~~	~~Ying-Chen Chu~~
~~Identifier:~~	~~UC20~~	~~Version:~~	~~2.0~~
~~Date Created:~~	~~2015-02-03~~	~~Last Modified:~~	~~April 12, 2016~~
~~Importance:~~	~~5~~		
~~Actor(s):~~	~~Student, Administrator~~		
~~Goal:~~	~~Remove a course from the schedule~~		
~~Summary:~~	~~Remove a course from the list of selected courses for the schedule generator.~~		
~~Related use-cases:~~	~~UC "**Modify Schedule**"~~		
~~Preconditions~~	~~-User has been authenticated~~ ~~-At least one course has been selected~~		
~~Trigger:~~	~~User activates the "drop course" process.~~		
~~Basic Flow:~~	~~1. User indicates the wish to drop a course~~		

	2. System prompts for delete confirmation. 3. User confirms delete process, 3. System removes course from user's list of courses.
Post-Conditions:	**Success:** Course is removed from the user's list of courses
Minimum Guarantee:	Schedule remains the same **Failure**: System fails to process task and displays an error message
Risk Assessment:	Low

Name:	Modify Preferences	Author	Ying-Chen Chu
Identifier:	UC21	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Modify the schedule preferences saved to the system		
Summary:	Access the current preferences and change any of the preferences from the list of preferences. Then, save this new list of preferences.		
Related use-cases:	UC12		
Preconditions	1. User has been authenticated 2. A schedule preference was set		
Trigger:	User modifies and inputs new preferences		
Basic Flow:	1. User indicates the wish to modify preferences. 2. System prompts for day off and time of day 3. User inputs new preferences. 4. System updates user's preferences with new information.		
Post-Conditions:	**Success:** The user's modified preferences are saved to the system.		
Minimum Guarantee:	Previous preferences remain unchanged. **Failure:** System fails to modify the preferences, and an error message is displayed.		
Risk Assessment:	Low		

Name:	Update Taken Courses	Author	Adil Hssaini

~~Identifier:~~	~~UC22~~	~~Version:~~	~~1.0~~
~~Date Created:~~	~~Feb 4, 2016~~	~~Last Modified:~~	~~April 12, 2016~~
~~Importance:~~	~~4/5~~		
~~Actor(s):~~	~~Registered User / Administrator~~		
~~Goal:~~	~~To update an existing student record.~~		
~~Summary:~~	~~The user adds or modifies entries into the student record. The system stores the newly entered information and updates record.~~		
~~Related use-cases:~~			
~~Preconditions~~	~~1. User is logged on.~~ ~~3. System has the record menu.~~		
~~Trigger:~~	~~User activates the "Update Taken Courses" process~~		
~~Basic Flow:~~	~~1. User selects the "edit" feature.~~ ~~2. System prompts to enter information.~~ ~~3. User inputs information.~~ ~~4. User commits to saving.~~ ~~5. System responds with outcome of the operation.~~		
~~Post-Conditions:~~	~~**Success:** Student record is updated successfully.~~		
~~Minimum Guarantee:~~	~~All other system data, configurations and functionalities remain unchanged.~~ ~~**Failure:** System fails to process task and displays an error message~~		
~~Risk Assessment:~~	~~Low~~		

Name:	Add Course to Program	Author	Adil Hssaini
Identifier:	UC23	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Administrator		
Goal:	To add a new course to a specific program.		
Summary:	The Administrator updates the list of required courses for a specific program by adding a new course.		
Related use-cases:	UC1		
Preconditions	1. Actor is logged on as administrator. 2. System has accessed the program menu. 3. Admin has accessed to the add course main menu		
Trigger:	Administrator activates the "Add Course to Program" process.		

Basic Flow:	1. Administrator initiates the add a course process by entering and submitting the course Name, ID, Code, Semester, Description, & Number of Credits. 2. Administrator enters and submits Section name, Id, Room, Semester, Type, day, Beginning Time, End Time, Course ID, Course Cd. 3. System updates course information with additional information provided.
Post-Conditions:	**Success:** Course is added successfully to the program listing.
Minimum Guarantee:	List of courses in the program stored by the system will not be affected. **Failure:** The system fails to process the task and displays an error message.
Risk Assessment:	Low

Name:	Edit Course	Author	Adil Hssaini
Identifier:	UC24	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Administrator		
Goal:	To edit a course's description or information.		
Summary:	The administrator modifies the existing description or details of a course.		
Related use-cases:			
Preconditions	1. Actor is logged on as administrator. 2. System has accessed the course menu.		
Trigger:	Administrator activates the “Edit Course” process.		
Basic Flow:	1. Administrator indicates wish to edit a course from the list by selecting the option. 2. System prompts for new course description and prerequisites information. 3. Administrator enters new information. 4. Administrator commits to editing the course. 5. System update course information with new entries.		
Post-Conditions:	**Success:** Course is edited successfully.		
Minimum Guarantee:	All other courses present in the list of courses required for the program will not be affected. **Failure:** The system fails to process the task and displays an error message.		
Risk Assessment:	Low		

Name:	Delete Course	Author	Adil Hssaini

Identifier:	UC25	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Administrator		
Goal:	To remove a course from a program listing.		
Summary:	The Administrator deletes a course from the list of required courses for a specific program.		
Related use-cases:			
Preconditions	1. Actor is logged on as administrator. 2. System has accessed the course menu.		
Trigger:	Administrator activates the "Delete Course" process.		
Basic Flow:	1. Administrator indicates wish to delete a course from the list by selecting the option. ~~2. System prompts for delete confirmation.~~ ~~3. Administrator commits to deleting the course.~~ 4. System removes course from program listing.		
Post-Conditions:	**Success:** Course is deleted successfully.		
Minimum Guarantee:	All other courses present in the list of courses required for the program will not be affected. **Failure:** The system fails to process the task and displays an error message.		
Risk Assessment:	Low		

Name:	Add Section	Author	Adil Hssaini
Identifier:	UC26	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Administrator		
Goal:	To add a course section.		
Summary:	The administrator adds a section to a course in the list of required courses for a program.		
Related use-cases:	UC – **Edit Course**		
Preconditions	1. Actor is logged on as administrator. 2. System has accessed the course menu.		
Trigger:	Administrator activates the "Add Section" process.		

Basic Flow:	1. Administrator indicates wish to add a section to the course. 2. System prompts for section Name, Location, Day, time, Course Name & Semester. 3. Administrator enters new information. 4. Administrator commits by saving new information. 6. System checks for duplicity & location conflicts. 5. System adds new section to the course
Post-Conditions:	**Success:** Section is created successfully.
Minimum Guarantee:	All courses and sections previously stored by the system will not be affected. **Failure:** System fails to complete the task and an error message is displayed.
Risk Assessment:	Low

Name:	Edit Section	Author	Adil Hssaini
Identifier:	UC27	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Administrator		
Goal:	To edit information or details of a section.		
Summary:	The administrator modifies the existing details of a section.		
Related use-cases:	UC – **Edit Course**		
Preconditions	1. Actor is logged on as administrator. 2. The system has accessed the section menu.		
Trigger:	Administrator activates the “Edit Section” process.		
Basic Flow:	1. Administrator indicates wish to edit a section in an existing course. 2. System prompts for new time and location of the section. 3. Administrator inputs new information. 4. Administrator commits to editing the section. 5. System updates the section with the new information.		
Post-Conditions:	**Success:** Section details are modified successfully.		
Minimum Guarantee:	All courses and sections previously stored by the system will not be affected. **Failure:** System fails to complete the task and an error message is displayed.		
Risk Assessment:	Low		

Name:	Delete Section	Author	Adil Hssaini
Identifier:	UC28	Version:	2.0
Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5		
Actor(s):	Administrator		
Goal:	To delete a course section.		
Summary:	The administrator deletes a course section from the list of available sections.		
Related use-cases:			
Preconditions	1. Actor is logged on as administrator. 2. System has accessed to the section menu.		
Trigger:	Administrator activates the “Delete Section” process.		
Basic Flow:	1. Administrator indicates wish to delete a section from an existing course. ~~2. System prompts for confirmation to delete.~~ ~~3. Administrator commits to deleting the section.~~ 4. System removes the section from the list of available sections.		
Post-Conditions:	**Success:** Section is deleted successfully.		
Minimum Guarantee:	All courses and other sections previously stored by the system will not be affected. **Failure:** System fails to complete the task and an error message is displayed.		
Risk Assessment:	Low		

3.2 Domain Model

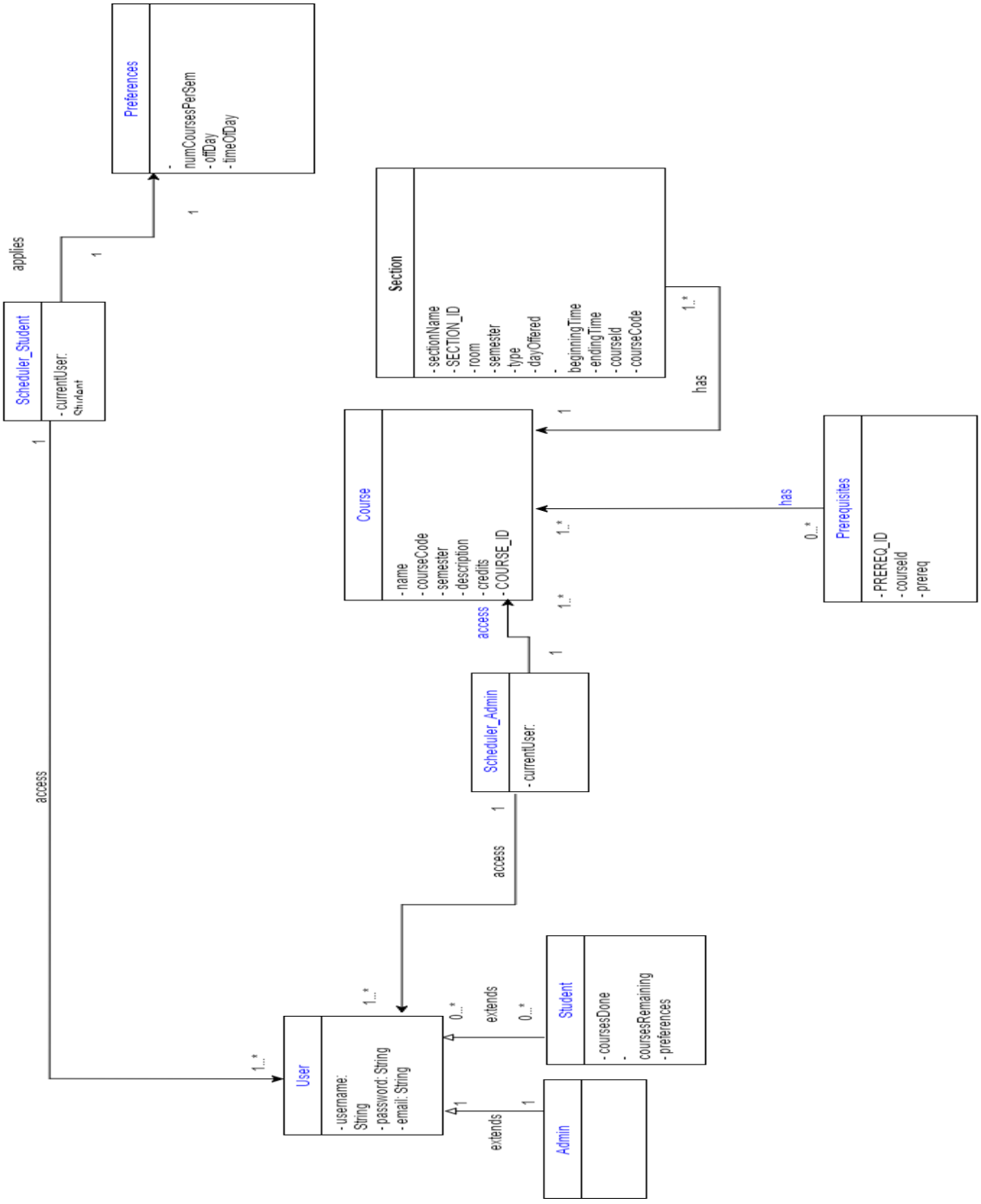


Figure3: *Domain Model*

DLO 1: **User**: This represents a real world user of the scheduler system. The user can be a registered student or an administrator of the system. A user has a user name, email and password.

DLO1.1: **Student**: This represents a student who has an account in the system and can use the schedule generator. A user can specify to the system the courses they have completed, the courses needed and their preferences in creating the schedule.

DLO1.2: **Admin:** This represents an administrator of the scheduler system. The admin can add, delete or modify courses and section to the scheduler system.

DLO2: **Preferences:** Preferences are a set of conditions that the student specifies prior to attempting to generate the schedule. preferences should be taken into account by the scheduler system during generating the schedule to the student.

DLO3: **Course:** Represents any course in the software engineering department. Courses can be accessed by an admin who can add, delete or modify courses. A course has sections, and can have a set of prerequisites. A course can be uniquely identified by an ID.

DLO4 **Section:** A section represents a real section that students attend. As section always needs to be related to one specific course. It can be either a lecture, lab or tutorial. It has specific start time and end time and is given on certain week days.

DLO5: **Prerequisites**: Represent courses that prerequisites to other courses. The system should make sure that a student has satisfied all the prerequisites of a certain course priors to adding it to their schedule.

DLO6: **Scheduler_Student**: This is responsible for allowing students to add or drop the courses that they need in their schedule. Moreover, it generates the schedule to students using the courses added and the preferences that the students specify.

3.3 Constraints & Qualities

Non Functional Requirements

3.3.1. Product

1.1 Security

The system shall be developed and coded with security at the forefront of concerns. The scheduler is implemented with React as a web application hosted by an Apache Web Server, and therefore provides default configurations as well as custom configurations that significantly help reduce XSS vulnerabilities and prevent attacks such as information leakage or PHP injections.

1.2 Privacy

To enforce and ensure privacy, practices such as session locks and expiry logouts shall be enforced. An encryption algorithm will also be implemented to provide an additional layer of abstraction to registered user's data.

1.3 Maintainability

The scheduler has several features that render its maintenance process more flexible than similar products. The use of React is in line with the nature of the system based on data that changes overtime. In addition, JavaScript being a very flexible and powerful language, makes expansions and additional options easily feasible as opposed to the standard directives or templates.

1.4 Compatibility

- The system shall be compatible with all mainstream platforms browsers.
- The system shall work on innate as well as on virtual machines.
- The system shall be accessed worldwide unless restricted by recipients' service providers or network settings.

1.5 Portability

The scheduler is not linked to any specific database. It is designed to operate as a public interface enabling access to multiple universities' programs. Options to link to selected databases are possible and are taken into account by the design.

1.6 Performance

The scheduler guarantees operations execution with minimal complexity. Coding aims at optimizing hardware and therefore reducing the system's and user devices' response time.

3.3.2. Organizational

2.1 Development

The system is designed from a prototype; a basic functional prototype will be developed and used in order to evaluate the Feasibility of requirements and strength of the design.
The system shall pose full documentation for requirements and design models. The documentation will be broken down and separated into sections in order to provide a work breakdown structure.

2.2 Operational

The system is designed to function with various sizes of databases: as long as the appropriate databases respect the current SQL schematic for tables and relationships, the database can be changed. Furthermore, the web scheduler shall possess a model view controller design pattern.
A model shall be used for the object oriented back-end to manage data. This model can respond to requests from the front end view and the overall controller. This controller will direct user input and general management directives in order to change the state of the model and its data.

2.3 Environmental

The system requires internet for all user access: the web application can only be utilized through a web browser on a device with internet access. The system requires access to databases for user information. This information is stored in SQL databases that must be accessed to perform all basic operations (login, see schedule).

3.3.3. External

3.1 Regulatory

The Schedulers' team shall conform to Concordia University's Academic Code of Conduct. The web application will also respect trademarks and intellectual property, and the supporting documentation will cite and reference all works used.

3.2 Legislative

For accounting purposes, the scheduler's team's captains will keep a time record for meetings. A time log record for each team meeting, created by a team leader, will be uploaded to the supporting documents for the Scheduler's team. This shall be added to the individually kept team member records and logs for time spent on each section: Individual members will record the time spent on each assigned sections.

4. Resource Evaluation

4.1 Human Resources

Team Member	**Nicolas Frazer-McKee**
Role	Documentation
Knowledge	Javascript, PHP, HTML, CSS, Java, c++, SQL
Experience	- Personal web page editing and hosting - Formal documentation writing as scientific papers
Strengths	- Formal writing - Diagram creation
Availability	Evenings during the week

Team Member	**Le Vinh Dang**
Role	Back-End Programmer
Knowledge	SQL, C++, Ruby
Experience	- Teamwork experience - Object-oriented programming - Work experience as software developer
Strengths	Debugging - Problem solving - SQL - Object Oriented
Availability	Any Time

Team Member	**Dias Marat**
Role	Back-End Programmer
Knowledge	Java, Javascript, HTML5, CSS3, PHP, Arduino, Objective C, C#
Experience	- Developed IPhone Applications and video games for several years - Developed server side applications using NodeJS and worked on backend previously - Team projects
Strengths	- Debugging - Problem solving - Mobile Development and Web Development
Availability	Any time

Team Member	**Bruce Edouard Brazier**
Role	Back-end Lead
Knowledge	Javascript, PHP, HTML, CSS, Java, Bash,SQL
Experience	- 1 work term - Work/Personal Experience maintaining a website (Client/Server side) - Work experience working with Software Deployments and Scripting
Strengths	- Troubleshooting - Process Automation - Team coordination
Availability	Any time

Team Member	**Sean Marcoux**
Role	Front-end Lead
Knowledge	React, Javascript, PHP, HTML, CSS, Java
Experience	- 1 work term - Work experience developing React application in teams - Work experience developing improvements and bug fixes for a Java, eclipse RCP project
Strengths	- Developing refined user experiences - Debugging - Problem Solving - Java + React
Availability	6 hours a week

Team Member	**Adriel Fabella**
Role	Front-end programmer
Knowledge	Javascript, PHP, SQL , HTML, CSS, Java.
Experience	- Part-time web designing and developing - Volunteer experience with PHP/ MySQL - Team Projects
Strengths	- Web developing - Communication skills - Problem solving
Availability	Any time

Team Member	**Adil Hssaini**
Role	Design & Development
Knowledge	C++, java, PHP, SQL, HTML, CSS
Experience	- Software maintenance experience - Prior involvement in large projects
Strengths	- Time and resource management - Versatile background - Team organization
Availability	4 to 6 hours per week

Team Member	**Emili Vasseva**
Role	Team leader and documenter
Knowledge	Javascript, PHP, SQL , HTML, CSS, Java, C++, Prolog, Lisp, AspectJ
Experience	- Contractual web developer using HTML, CSS, JavaScript, PHP, MySQL - Currently developing a small game in Unity - Team Projects
Strength	- Web development - Problem solving - Leadership - Object-oriented programming
Availability	Any time

Team Member	**Ying-Chen Chu**
Role	Documentation
Knowledge	Java, C++, Ruby, Ruby on rails, HTML, SQL, Arduino
Experience	- Work term as software developer - Developed a simulation framework - Team projects
Strengths	- Object-oriented programming - Problem solving
Availability	6 hours per week

Team Member	**Alex Eladas**
Role	Documentation and Corrector
Knowledge	C++, HTML,CSS
Experience	- 1 work term - Worked on designing and implementing a new process. - Team projects
Strengths	- Object-oriented programming - Problem solving - Analytical Skills
Availability	Anytime

Team Member	**Gabriele Bavaro**
Role	Front-end programmer
Knowledge	JavaScript, HTML, PHP, SQL, CSS, Java, C++, Python
Experience	Created numerous websites through WordPress and other web tools Helped program for a mars rover robot for Space Concordia Team projects
Strengths	Web development Problem solving Communication and teamwork skills
Availability	Any time

Team Member	Salma Aly
Role	Documenter
Knowledge	C++, Java, Python
Experience	- Teamwork experience - Object-oriented programming - Work experience as software developer - Documentation
Strengths	- Debugging - Problem solving - Testing - Object Oriented Programming
Availability	5 hours/Week

4.2 Technical Resources

The following section has been divided into several subsections which discuss technical resources associated with documentation, programming languages and software, hardware, operating systems, communication and management.

4.2.1. Documentation

The software applications that are being used for editing and reviewing the source documentations and codes are Google Docs/Drive and Microsoft word. Google Docs is a flexible program that allows team members to brainstorm and work concurrently on a document in real-time. Google Drive is used to store Google docs files in a shared folder accessible by all team members. Microsoft Word was used for more individual documentation from team members and for greater flexibility in organizing documented information. Adobe Reader was utilized in order to render documents into a format that could be read by all team members and their affiliates. WireFrame was used to design and showcase the front end interfaces and pages that would become part of the completed product.

4.2.2. Programming Languages and Software

The web server used to store the database is the WampServer64 2.5. The WAMP server, which stands for Windows, Apache, MySQL and PHP, provides those programs for use. With the WampServer64 version 2.5, it will support the Apache: 2.4.9, the MySQL: 5.6.17, the PHP: 5.5.12, the PHPMyAdmin: 4.1.14, the SqlBuddy: 1.3.3, and the XDebug: 2.2.5. The programming languages which will be used to construct the final product are HTML, PHP, JavaScript and CSS.

4.2.3. Hardware

In addition to the above programs, laptops and desktops will be used to install and carry the WAMP database and other software that will be required for the project. The laptops are not of a uniform variety but instead come from a wide range of manufacturers. They are DELL, HP and MAC laptops and desktops. In addition to containing the above server, the laptops and desktops also contain React, a JavaScript software used to develop the front end of the final product.

4.2.4. Operating Systems

In order to facilitate uniformity amongst team members the team laptops/desktops must have the following minimum requirements:

- 150 GB of memory
- 4 GB of RAM
- Intel Premium 4 or AMD Athlon x64
- WI-FI internet access
- Windows 7/Linux operating systems
- Headsets, earphones and speaker setups (to allow for discussions through skype)

4.2.5. Communication and Management

All code and document files related to the project are sorted and stored on Github under the repository Schedule-Builder through the use of Github accounts. All team members have access to Github folders, files and their content. To facilitate communication between team members, Facebook, Slack and Skype are used for holding discussions and meetings.

5. Scoping

In order to fulfill the requirements in section 3.1 and 3.3, Team leaders with more experience were assigned to each of the 3 sub-teams; front end-html and web design, back end-OOP with PHP, and documentation- further work breakdown structure. This breakdown allowed to streamline parts of the project efficiently; early in the project, one type of user was removed as it was judged to be unnecessary and detracted from the main user while adding a fair amount of complexity: Professor. The removal of this user simplified the system by rendering a request feature obsolete. This represented a significant difficulty for the programming teams and was ultimately useless.

The resulting system's full scope as a web application to be used for Software Engineering students is highlighted by the following lists of included (scoped in) and excluded (scoped out) features that extend the minimum requirements.

5.1 Scoped In

The system will allow 3 types of users: Public Users, Students and Admins, two of which were scoped in:

- *Students*: they can opt to add a set classes through the application, and consequently will be provided with recommended schedules for their coming semesters, with their time preferences taken into account. After that, they can choose the most fitting schedule for themselves for the next semester, and the system will display the rest of the courses as a sequence that is matched with the chosen schedule.
- *Admins*: they have the capability of adding and removing students & courses in and from the system. The admins can also modify all courses' information such as names, IDs, lecture time & location.

The functionality of resetting the password will be implemented in the application. The users will provide their usernames and emails to get an email from the system to change the password.
The prerequisite and co-requisite courses which are added into the system are controlled by admins. The admin can specify which course is a prerequisite to another. Generating schedules is also one of the features offered to students can do in the system.

5.2 Scoped Out

We have decided to scope out some of functionalities that we believe will be too expensive to implement and would not satisfy the demands of the clients.

- UC5: View Default Schedule without Creating an Account
  The public users are not implemented in the system since if a student wants to generate a schedule, we will need his or her information such as the major to implement the schedule. Therefore, viewing the schedule without creating an account is not possible
- UC6: Modify a generated schedule
  Modifying a generated schedule will not be done in the system.
- UC10: Delete needed courses
- UC11: Delete taken courses

- UC14: Save a generated schedule
- UC15: Delete a saved schedule
- UC16: Print a generated schedule
- UC17: View previously saved schedule

  Because we do not provide the ability to save the generated schedule, the user cannot view previously saved schedule.
- UC18: Change a section in the schedule.

  The generated schedule will meet all the demands from the user. Therefore, changing a section in the schedule will not possibly be done in the system.

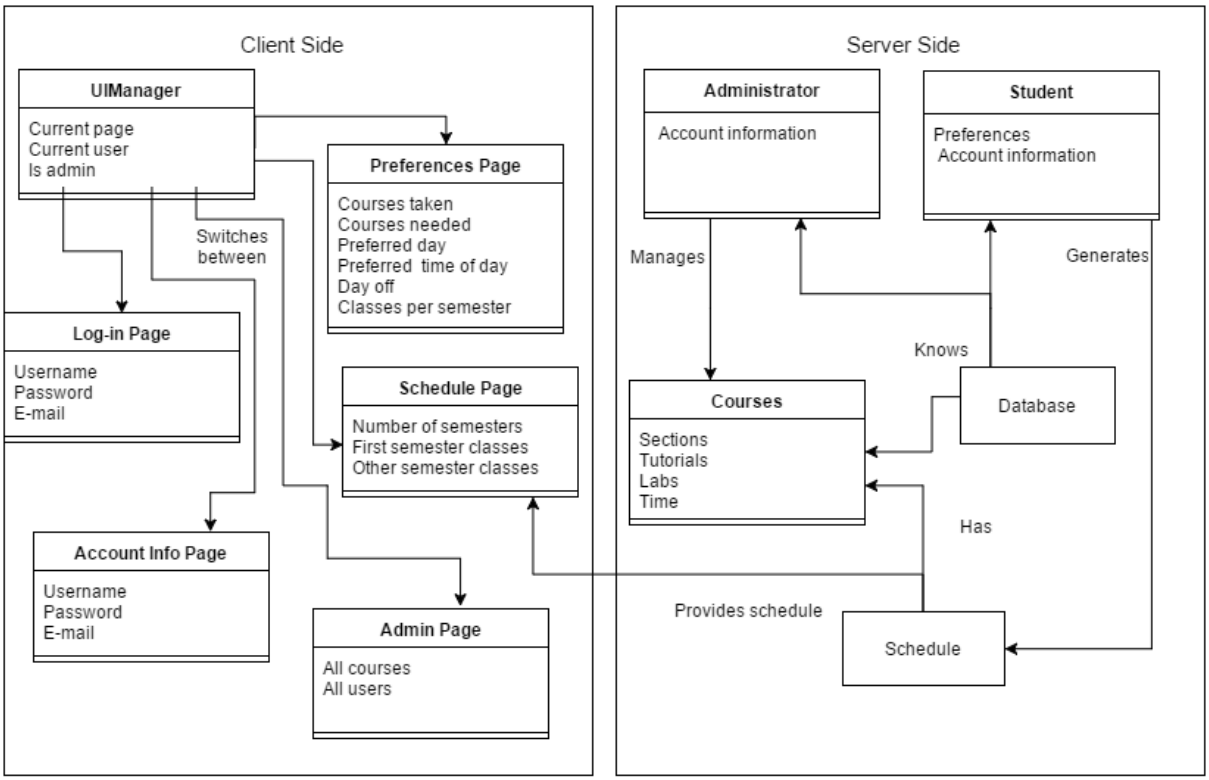
- UC20: Drop a Course
- UC21: Modify Preferences
- UC22: Update Taken Courses

- We remove the generating multiple recommended schedules from the UI because generating one schedule with all the following courses would be enough for students.
- UC25: Delete Course
- UC28: Delete Section

6. Solution Sketch

6.1 Architecture

Unlike the standard MVC architecture, the model, the view and the controller are not taken care by the same framework. In our architecture, the view is handled using React and data manipulation as well as database queries, are handled by Laravel. This means that the application is divided between client side and the server side. The client side handles everything to do with the view (everything the user sees and interacts with) while the server side handles everything else.



6.1.1. Server Side

For the server side, the components are the users (students, admin), the schedule, the database and the courses. The database contains information about the students, the administrators and the courses. When the users modify their preferences and their information, it is updated in the database by a query. The students interact with the schedule component when they generate their schedule based on their preferences. The schedule component then fetches their preferences and generate the appropriate schedule. The administrator component can manage the courses and their properties. After the required information is gathered through the user page, the appropriate courses are then modified with a database query. Finally, the schedule provides the client side with the data that is to be displayed on the pages such as the student schedule or the full course sequence. This setup allows us to control the information of the students and the administrators. Ensuring that when they are needed, they can be accessed through the database. This will also facilitate the process of generating the algorithm since the components are independent and will be easier to manipulate.

6.1.2. Client Side

The main components for the client side are the UIManager and the components for each page. These will all be React components. The UIManager will be the necessary main React component and it will handle switching between all of the pages and hold the data that is common to all of them: the active user and if that user is an admin. This structure is the best way to handle the UI because switching between pages will be as simple as changing which component is being rendered. It also allows simple communication between pages through the UIManager.

The page components are the log-in, preferences, account info, schedule, and admin pages. The log-in page needs to keep track of any input the user enters, which are username, password, and e-mail (if the user is registering for the first time). The account info page needs the same info, but this needs to be the information obtained from the server. The preferences page is where the user sets the courses they've taken, the courses they still need to take, and the preferences they have for their schedule. The preferences page component will keep track of all this info and will obtain any of it from the server if the user already input preferences in the past. From the preferences page, the user can click a button to build the schedule. This will generate the schedule on the server side and return the schedule information on the schedule page. The first semester classes here is a separate variable because these classes need the additional information of time, classroom, section, and teacher. The remaining semesters will simply be a list of classes for their recommended course sequence. Finally, the admin page will obtain a list of all courses and a list of all users registered in the database, allowing the admin to edit them.

6.2 Technologies in Use

6.2.1. Programming Languages

1. **HTML:**

   HTML is a computer and markup language that allows to create web sites and web documents. This language will consist of the very backbone of the website, when it comes to filling up the webpage with text and dialogs.

2. **CSC:**

   CSC is the language within the markup language that allows to manipulate the design of the web document, meaning positioning, color and overall presentation.

3. **JavaScript:**

   JavaScript is the main client side programming language used for creating interactive websites. JavaScript support is built right into all the major browsers and can support object

oriented programming. JavaScript will be the main dynamic language used by the front end team and everyone in the team possesses experience working with JavaScript.

4. **React:**

React is an open source JavaScript library which contains a template language and some function hooks to efficiently render HTML. React manages all UI updates when data has been changed and will update only those changed data. This is efficient because the user can tell how a component will render by looking at one source file. A program flow does not need to be traced which can be efficient when working in a big team.

5. **PHP:**

PHP is a server side scripting programming language used for web development. It is very well documented and can support objected oriented programming. PHP will be used for the backend and most of the team members have experience with PHP.

6.2.2. Framework

Laravel:

Laravel is an open source PHP web application framework that allows rapid development of web applications. Laravel uses MVC architecture and has features such as module package manager, template engine, database seeding, routes, authentication, and object oriented design. This will provide clean and manageable code.

6.2.3. Integrated Development Environments/Editors

PhpStorm:

PhpStorm is an Integrated Development Environment for Windows and Mac OS that allows developers to code their projects in PHP. It has syntax highlighting, plugins, different types of frameworks supported such as Symfony, Laravel, CakePHP, built in support for databases, version control, debugging and testing. PhpStorm increases the productivity of developers.

6.2.4. Source Code and Revision Management

Git:

Git is a source code management system used for software development. It allows developers to save different versions of their projects at different points in time and compares

them to one another. Git allows developers to contribute to a repository (project) even if the developer is not connected to the Internet. It stores a local copy of the project on the local repository and changes made on the local repository can be pushed to the main repository. This promotes organization and maintains previous versions of the project.

6.2.5. Collaboration Software

1. **GitHub:**

Github is a website that hosts Git repositories and has all the functionalities of Git. It provides bug tracking, feature requests and wikis for projects. It is used in this project as it allows efficient collaboration between developers.

2. **Google Docs:**

Google Docs is an online word processor that allows individuals to edit and collaborate on documents in real-time. It is free and can be accessed by anybody.

3. **Draw.io:**

Draw.io is a software application that provides tools to draw domain models, UML, use cases diagrams and etc. It allows collaboration between individuals and can be used as a plugin to Google Drive.

4. **MockFlow – Wireframe pro:**

MockFlow – Wireframe pro is an application used to create the mockup of the website. It allows collaboration where the whole team contributes to creating the backbone using the available widgets and elements and to display comments and reactions.

5. **Skype:**

Skype is a free video chat application that allows users to do video conference calls and exchange documents. For this project it is used between sub sections of the teams because it is more efficient.

7. Plan

The following section depicts the tentative schedule that will be applied during the rest of the project. Each table represents an activity and its artifact and their respective number of hours of execution. Furthermore, each activity has been assigned to various team members.

7.1 Activities, Artifacts & Activities Assignments

7.1.1. Deliverable 0 - System Overview

Due date: January 13th, 2016

The purpose of this deliverable is to familiarize ourselves with the project, and therefore create a domain model on how the software should behave.

Activity:	**Team Assignments**
Purpose	Assigning roles to the team in terms of their preferences and strength/weaknesses and electing a team leader
Artifact #1	Team members list
Description	List with the name and role of each team member
Combined total work hours	1
Due date	January 8th 2016
Participants	Emili, Sean, Dias, Bruce,

Activity	**System Definition**
Purpose	A concise description of the software to be developed with its purpose, functions and its classes of users.
Artifact #1	Domain Model
Description	The principal entities and their relationships. Not including any methods.
Combined total work hours	4
Due date	January 9th -January 11th 2016
Participants	Salma, Ying-Chen, Adriel, Gabriele, Le Vinh, Alex

7.1.2. Deliverable 1: Requirements, Scope and Plan

Due date: February 10th, 2016

The purpose of this deliverable is to work on the basic structure (UCD, DM and basic architecture), to create a plan for the project, as well as creating a small prototype.

Activity	Defining Requirements
Purpose	To describe the functionality of the system in terms of processing each user actions. Defining the main functions of The Scheduler that take place when generating an output
Artifact #1	Use Case Diagram
Description	A diagram explaining the interactions between the actors and functions of the system and showing the relationship between the use cases.
Combined total work hours	10
Artifact #2	Use Cases
Description	A complete list of all the use cases included in the system.
Combined total work hours	12
Artifact #3	Domain Model
Description	Updated domain model containing the attributes and associations between each class objects.
Combined total work hours	4
Due date	January 22th February 7th 2016
Participants	Salma, Adil, Ying-Chen, Nick

Activity	Architecture
Purpose	A preliminary description of the high-level structure showing the early version of the proposed solution and the reasons leading up to this design.
Artifact #1	Non-Functional Requirements
Description	The constraints the system will undoubtedly meet throughout its development.
Combined total work hours	10
Due date	January 29th-February 7th 2016
Participants	Bruce, Sean,

Activity	Resources
Purpose	Evaluating the experience and knowledge each team member can bring to the project. Presenting the list of the available technologies for the project.
Artifact #1	Technologies used
Description	A list of the different hardware, software or any other tool that could be used for the system's development.
Combined total work hours	2
Due date	February 6th-February 7th 2016
Participants	Gabriel, Adriel

Activity	Planning
Purpose	Describing every activity and documentation to be completed throughout the development of the system
Artifact #1	Estimation
Description	A time and cost estimation for the completion of the project
Combined total work hours	2
Artifact #2	Schedule
Description	A diagram showcasing the timetable for each main phases. (Gantt Chart)
Combined total work hours	2
Artifact #3	Risks
Description	A list of the various risks that could be encountered during the development of the system
Combined total work hours	3
Due date	February 4th-February 8th
Participants	Emili, Alex

Activity	Prototyping
Purpose	An early version of the system proving that the technologies used are proper for the project
Artifact #1	Working framework
Description	An initial design of the system that describes its main functions.
Combined total work hours	10
Artifact #2	Server Connection
Description	An initial call to the servers implemented in the prototype demonstrating the information storage
Combined total work hours	15
Due date	February 6th February 9th 2016
Participants	Sean, Bruce, Le Vinh, Dias, Adriel, Gabriel

7.1.3. Deliverable 2: Design

Due date: March 9th, 2016

The purpose of this deliverable is to develop the full structure and design of the software, and create a rapid prototype out of these.

Activity	Detailed Architecture
Purpose	Overall structure of the system
Artifact #1	4+1 Architectural View
Description	High-level structure of the system, composed of 5 views: logical view, process view, development view, physical view and scenarios. Used to describe a large system into multiple subsystems.
Combined total work hours	10
Artifact #2	Subsystems Interface Specifications/Module Interface Specifications
Description	Description of each subsystems meant to complete specific services, and their parameters (invalid/valid values) passed in functions.
Combined total work hours	25
Start/End dates	February 11th - February 29th
Participants	Sean, Bruce, Emili, Nick

Activity	Detailed Design
Purpose	Complete class description of each subsystem
Artifact #1	UML Class Diagram
Description	Connection between classes of each subsystem
Combined total work hours	12
Artifact #2	Dynamic Design Scenarios
Description	2 dynamic design of 2 uses cases (using at least 3 system operations). This includes system sequence, operational contracts, and sequence diagrams.
Combined total work hours	6
Artifact #3	Estimation
Description	Estimated cost for integration, testing and documentation for each module.
Combined total work hours	7
Start/End dates	February 24th - March 8th
Participants	Adil, Alex, Salma, Ying-Chen

Activity	Rapid Prototyping
Purpose	Programming of the prototype designed using the architecture and design description.
Artifact #1	Rapid Prototyping report
Description	Listing and commenting on classes/modules/drivers used for the rapid prototype.
Combined total work hours	22
Artifact #2	Testing
Description	Testing code and report of the rapid prototype
Combined total work hours	15
Artifact #3	Risks
Description	Update of the risks, cost estimate and scoping from the deliverable 1 to deliverable 2
Combined total work hours	4
Start/End dates	February 24th - March 8th
Participants	Dias, Adriel, Gabriele, Le Vinh

7.1.4. Deliverable 3: Testing

Due date: April 6th, 2016

The goal of this prototype is to finalize the programming with the respect test report. Furthermore, an instruction manual and a final cost estimate has to be documented.

Activity	Final prototype
Purpose	Final prototype of the a fully working software
Artifact #1	Test report and instructions manuals
Description	1. Test report on the entire making of the system 2. Instruction manual for future users
Combined total work hours	60
Start/End dates	March 10th - March 31st
Participants	Sean, Gabriele, Adriel, Le Vinh, Bruce, Dias

Activity	Testing Report
Purpose	Final report on all testing done on the final product
Artifact #1	Test coverage
Description	1. Listing of all tested items, and why. 2. Identification of 5 classes/methods and why they were tested.
Combined work hours	20
Artifact #2	Test cases
Description	1. Two mid-level units tests, with their respective test cases and descriptions. 2. Requirements testing and their test cases 3. Test cases of potential extreme system usages, and their respective description 4. Test testing regarding the security of the system
Combined work hours	15
Start/End dates	March 10th - April 1st
Participants	Emili, Adil, Salma, Nick

Activity	System delivery
Purpose	Instructions on the system
Artifact #1	Installation Manual
Description	Step by step instructions on how to install the system.
Combined work hours	6
Artifact #2	Users Manual
Description	Step by step instructions on how to use the system.
Combined work hours	6
Start/End dates	April 1st - April 5th
Participants	Alex, Ying-Chen

Activity	Final cost estimate
Purpose	Final coverage on the total amount of hours and money spent on the project
Artifact #1	Working hours
Description	Final coverage on the number of hours put into the project by each person.
Combined work hours	10
Artifact #2	Cost
Description	Final coverage on the costs spend on each individual work and for technological resources
Combined work hours	4
Start/End dates	April 1st - April 5th
Participants	Ying-Chen, Alex

7.1.5 Final Deliverable: Complete Report

Due date: April 13th, 2016

This section is the final delivery, consisting of finalizing the report.

Activity	Finalization of the deliverable
Purpose	Completion of the project by submitting a complete and corrected report.
Artifact #1	Final report
Description	Assembling and correction over the report and its content.
Total work hours	50
Start/End dates	April 7th - April 12th
Participants	Entire team
Estimated Total: 355 Hours	

7.2 Project Estimates

7.2.1 Basis for estimates

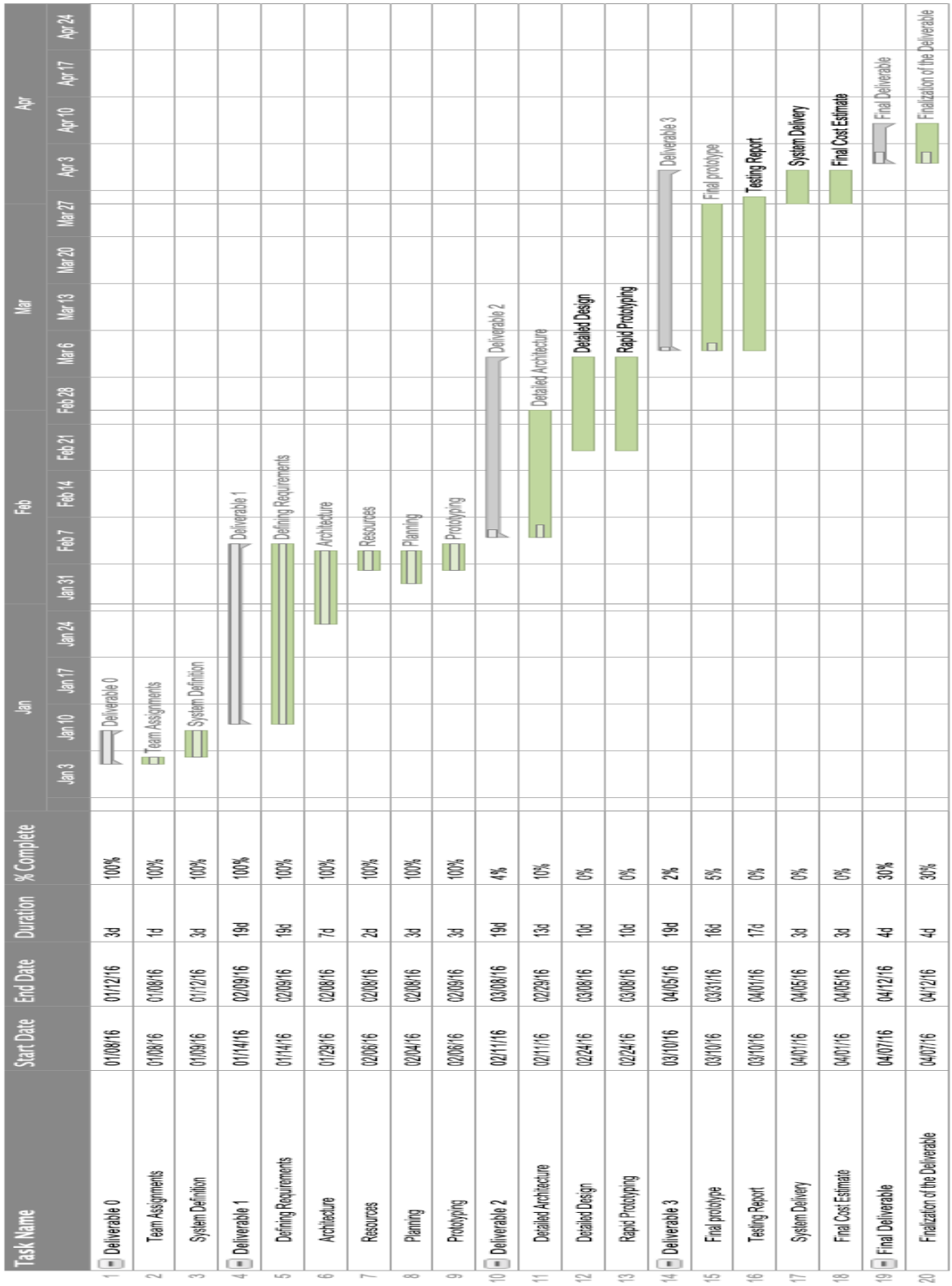
The basis for each artifact estimation came from analyzing the deliverables to be completed for the project. By breaking down all the sections, evaluating the difficulty of each tasks and considering the number of participants, the approximate working hours were calculated and added. The estimation will be revised later into the project if an important problem arises, which could delay the whole working process.

Assuming that the software engineers involved in the development are paid at an average rate of $25/hour.

Estimated Cost for the full project. At an hourly rate of $25/hour.

Hardware	
Computers, Servers:	**$0**
Software:	
Software/Technologies used:	**$0**
Software development/Documentation	**$8875**
Total:	**$8875**

7.3 Schedule


Task Name
Start Date
End Date
Duration
% Complete
Jan
Feb
Mar
Apr
Jan 3
Jan 10
Jan 17
Jan 24
Jan 31
Feb 7
Feb 14
Feb 21
Feb 28
Mar 6
Mar 13
Mar 20
Mar 27
Apr 3
Apr 10
Apr 17
Apr 24
1 Deliverable 0 01/08/16 01/12/16 3d 100%
2 Team Assignments 01/08/16 01/08/16 1d 100%
3 System Definition 01/09/16 01/12/16 3d 100%
4 Deliverable 1 01/14/16 02/09/16 19d 100%
5 Defining Requirements 01/14/16 02/09/16 19d 100%
6 Architecture 01/29/16 02/08/16 7d 100%
7 Resources 02/06/16 02/08/16 2d 100%
8 Planning 02/04/16 02/08/16 3d 100%
9 Prototyping 02/06/16 02/09/16 3d 100%
10 Deliverable 2 02/11/16 03/08/16 19d 4%
11 Detailed Architecture 02/11/16 02/29/16 13d 10%
12 Detailed Design 02/24/16 03/08/16 10d 0%
13 Rapid Prototyping 02/24/16 03/08/16 10d 0%
14 Deliverable 3 03/10/16 04/05/16 19d 2%
15 Final prototype 03/10/16 03/31/16 16d 5%
16 Testing Report 03/10/16 04/01/16 17d 0%
17 System Delivery 04/01/16 04/05/16 3d 0%
18 Final Cost Estimate 04/01/16 04/05/16 3d 0%
19 Final Deliverable 04/07/16 04/12/16 4d 30%
20 Finalization of the Deliverable 04/07/16 04/12/16 4d 30%
Deliverable 0
Team Assignments
System Definition
Deliverable 1
Defining Requirements
Architecture
Resources
Planning
Prototyping
Deliverable 2
Detailed Architecture
Detailed Design
Rapid Prototyping
Deliverable 3
Final prototype
Testing Report
System Delivery
Final Cost Estimate
Final Deliverable
Finalization of the Deliverable

7.4 Risks

7.4.1 Knowledge of frameworks and programming languages.

The laravel framework for PHP and React.js for javascript represent a risk, since only a few members of the team are familiar with their use and their learning curve could play in some scheduling issues. In others works, developing the system could longer than anticipated. In order to remedy this potential problem, experienced programmers in the languages will be put in charge to help and assist the other members of the team

7.4.2 Time

A single semester may not be enough to complete the whole system. Some key features may be left due to the final deadlines, which would hinder the usability of the Scheduler and won't be trustworthy enough to generate a full schedule.

7.4.3 Security

There will be a server storing all the user data and completing all the requests. Since this server is going to be made public, it represents a huge security risk and will have to be thoroughly examined.

7.4.4 Schedule Errors

The generated schedule has a possibility of not being valid if the system is not implemented perfectly. This would cause students to graduate in an extra semester or year.

7.4.5 Teamwork

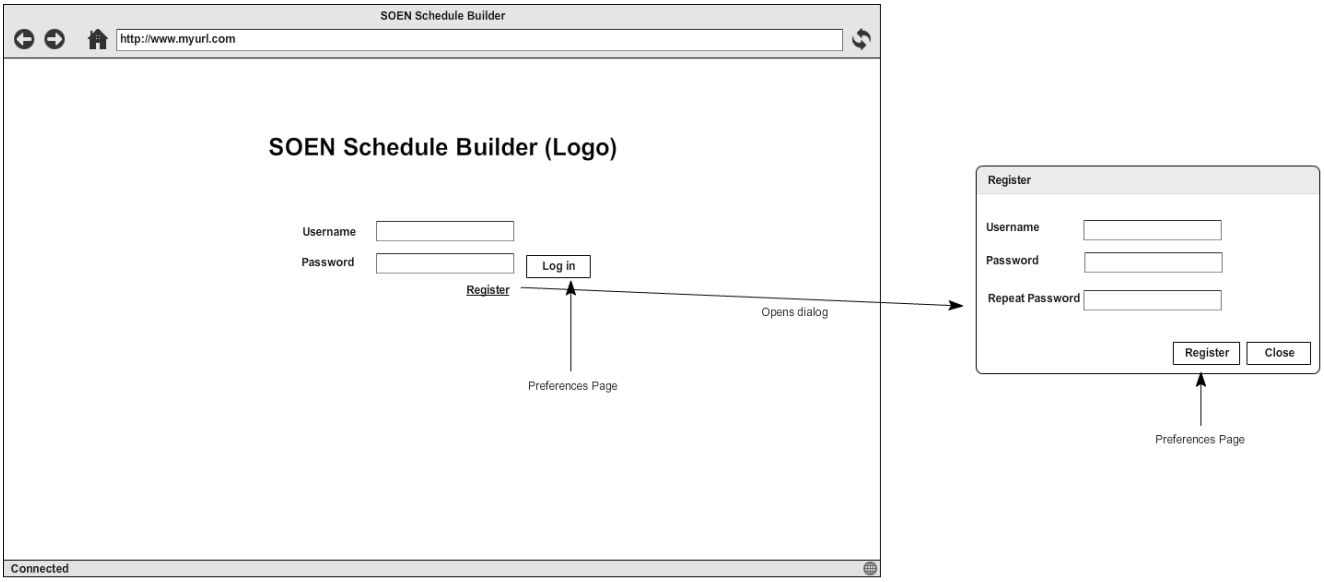
This is a time consuming project and all of the team members have responsibilities outside of this work. This causes a limit of time dedication for the project due to assignments or other engagements. Also, since the team is pretty large and for most it is the first time working together, there might struggles to communicate which causes a risk to the quality of the final system.

8. Prototyping

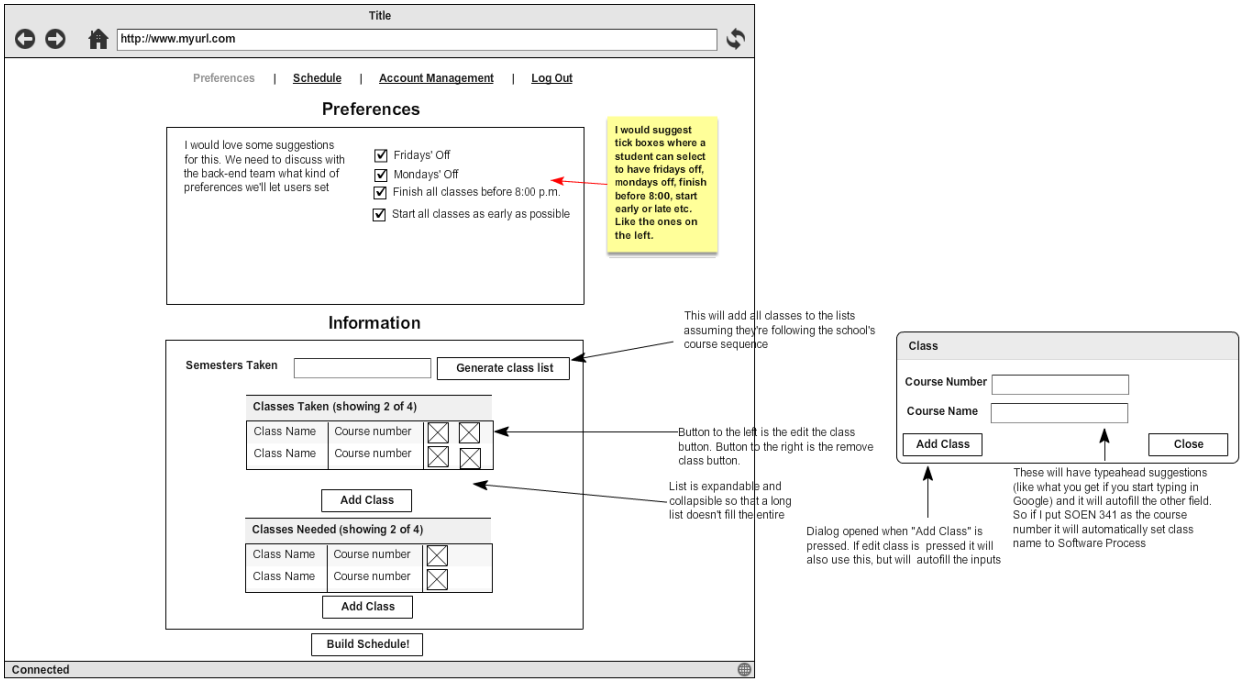
8.1 Mock-Ups

To create quick, simple designs of the website we used an online tool called MockFlow.

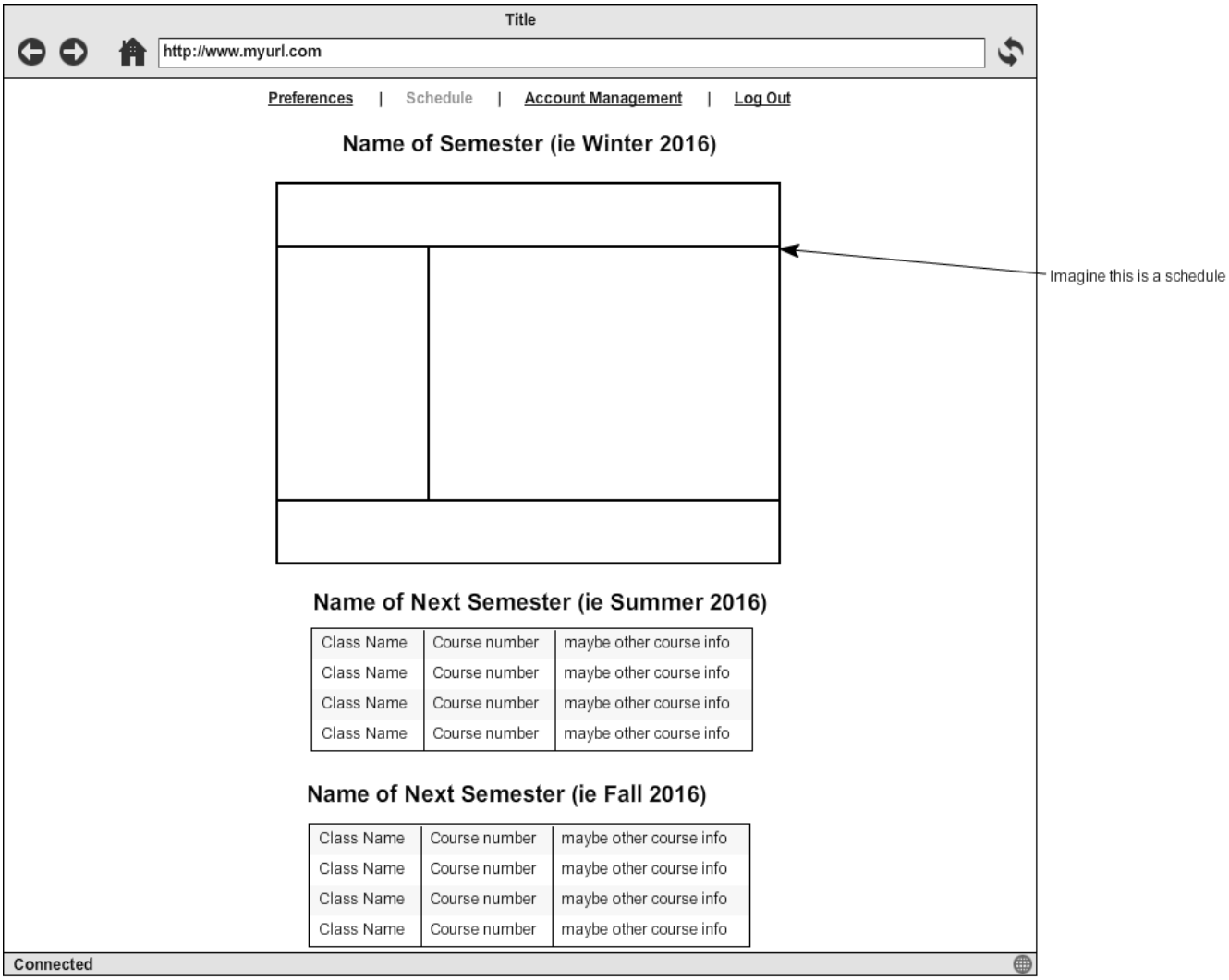
8.1.1 Log-In Page



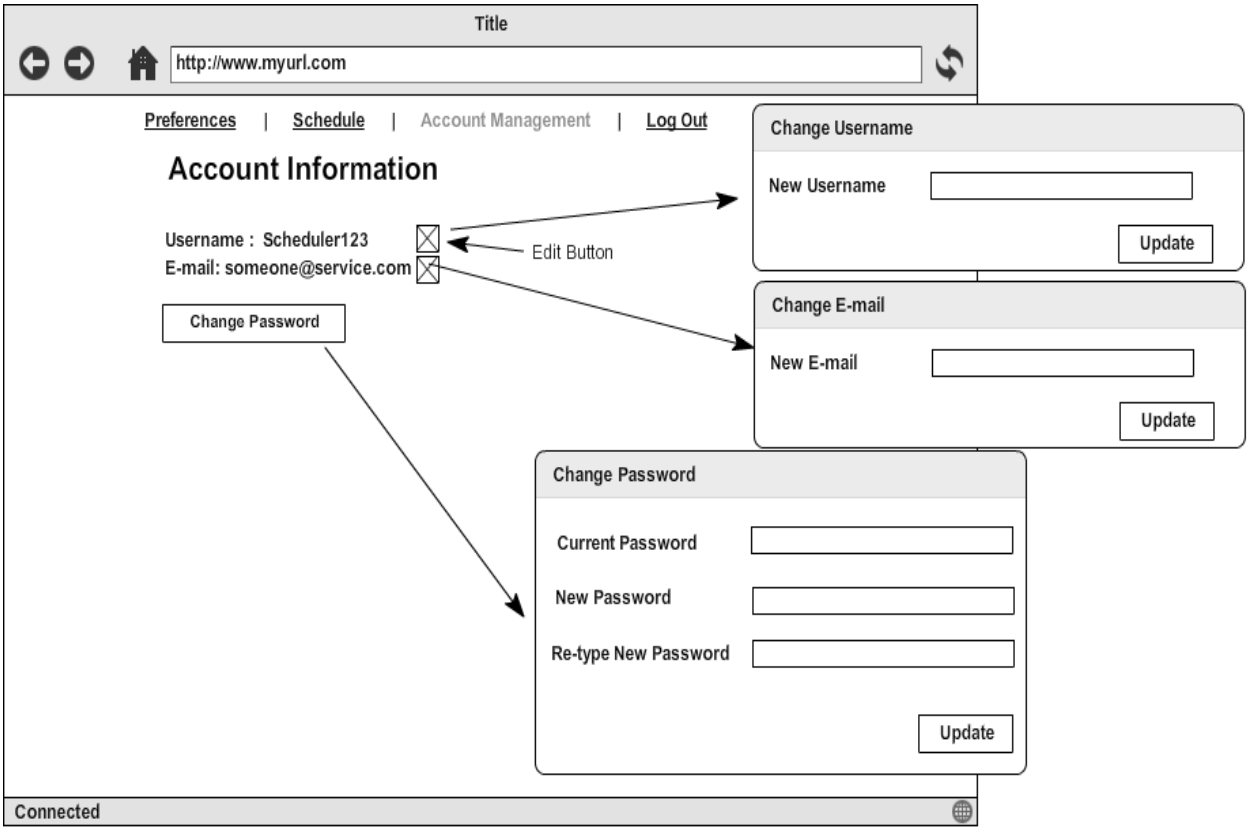
8.1.2 Preference Page



8.1.3 Schedule Page



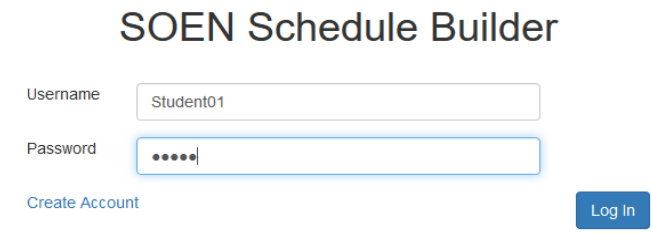
8.1.4 Account Management Page

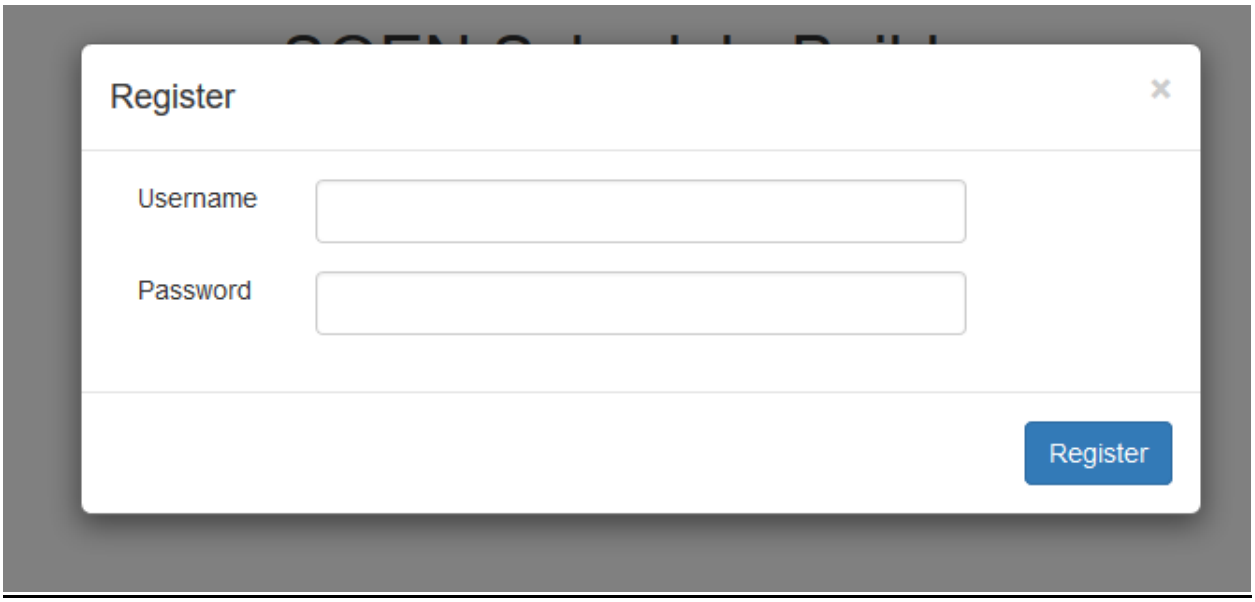


8.2 Prototype

To prove that the different frameworks work together, we decided to just have registration and log-in functionality, with the basic structure of the UI for the first two pages. This allows for some basic communication between the front and back end.

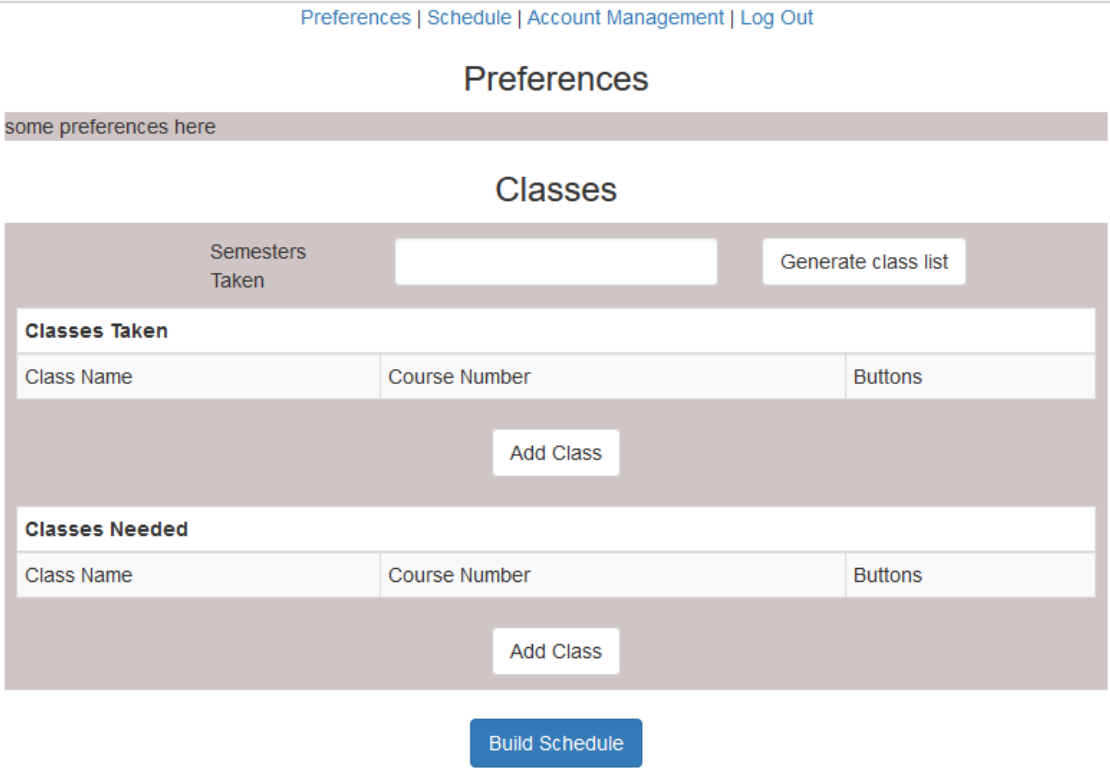
8.2.1 Log-In Page





The registration dialog on the log-in page

8.2.2 Preference Page



Part II:
Architecture & Design

1. Introduction

The present document provides a detailed description of the architecture and design of the scheduling system developed by team schedulers. The architecture of the system is represented through a "4+1" view that comprises logical, development, physical, and process views, in addition to scenarios that explain how actors, processes, and objects interact with each other.

Specification of the interactions between software interfaces and the components is discussed in the Subsystem interfaces specifications section, which describes how exchanges such as parameters passing, or function calls lead to service fulfillment. The design of each subsystem is subsequently detailed with class diagrams used to explain the internal structure of the subsystems, as well as a list of all attributes and functions in each class.

Dynamic design scenarios are depicted in the present document via a dynamic design of four use cases, namely: setpreferences, generateSchedule, addCourseinDB, and resetPasssword. While the first two uses cases are student specific, the third only pertains to the administrator, and the fourth is common between the two aforementioned actors.

A review of the cost estimate, prototyping, and risks comes at the end of the document to justify the validity of the overall software development process.

2. Architectural Design

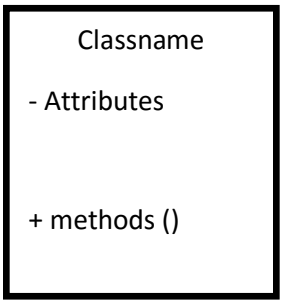
The "4+1 Architecture View" will be used in this section to show a detailed and updated version of the scheduling system, based on what has been developed and designed in deliverable 1. The "4+1 Architecture View" includes Logical, Development, Process, Physical and Scenarios Views. In the Logical View, we are going to present all the classes and their functions in the Class Diagram, and the Component Diagram will be shown in the Development View. Moreover, the Activity Diagram will be used in the Process view. And to implement the overview of the interaction between the system with the client, the Deployment Diagram will be presented in the Physical view. Lastly, the Use Cases Diagram will be shown in the Scenarios to help the clients understand more about the product.

2.1 Architecture Diagram

2.1.1 Logical View: Class Diagram Description

A class diagram is a static structure diagram which describes the system based on displaying its classes, their attributes, their methods, and the relationship between each class. The class diagram provides a good representation on the implementation of the software since it shows the types being modeled. Simply put, the class diagram of the system can be altered or refined.

A class in a class diagram is represented by a block divided into 3 sections.

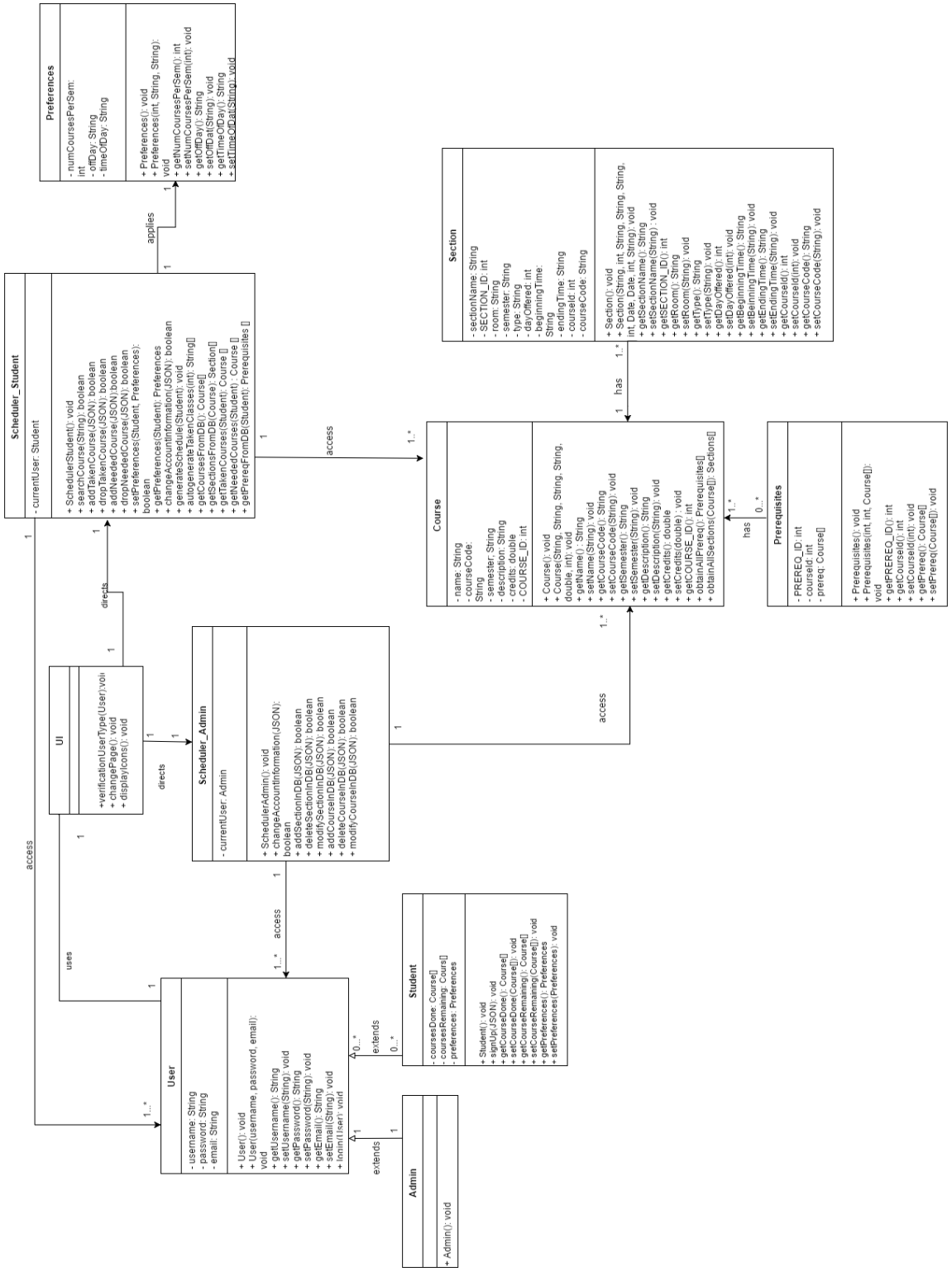


The block contains the **name of the class**, the **attributes** and the **methods**.

The **attributes** of a class describe the properties and characteristics that its object can have, while the **methods** pertain to the main actions and functions that the class can do. The signs, such as - and + represent accessibility level of a certain field or method in object oriented-programming. None the less, there also exists other types of accessibility levels such as default. It is conventional to display the attributes as private fields and the methods of public access.

The class diagram of the project portrays the system *Scheduler* as the main bridge to access functionality. The way for a certain user to access the Scheduler is to successfully login. From there, the UI will then redirect the user to the main page. The user interface also manages everything the client needs, meaning changing of pages, accepting requests or displaying information.

Class Diagram



2.1.2 Development View

The development view describes our scheduling system from our programmer's perspective and the management of our software. It is illustrated by the component diagram which consists of components and links that show the dependencies between them.

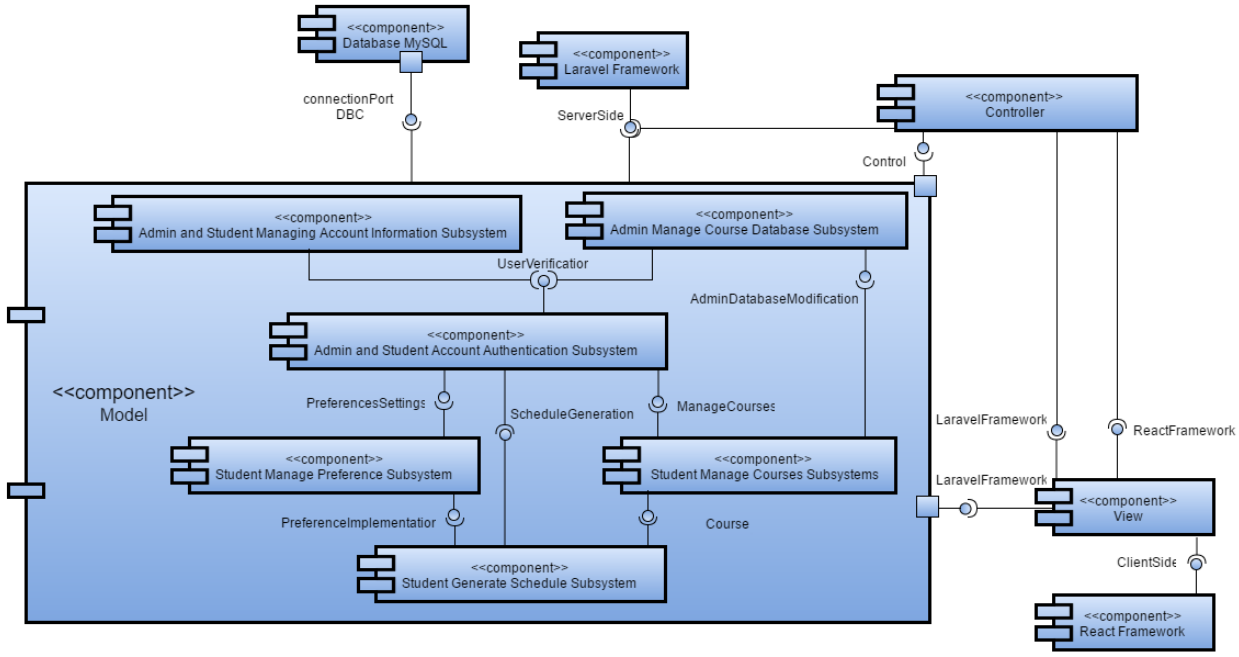


Figure 1: *Component Diagram*

The main component of our system consists of the model, view and controller. Since these 3 components are not taken by the same framework, we have specified on the component diagram that the view is handled using React and that data manipulation and database queries are handled by Laravel. Most importantly, in our model, we have defined our subsystems to be Admin and Student Managing Account Information Subsystem, Student Manage Courses Subsystems, Student Manage Preference Subsystem, Student Generate Schedule Subsystem, Admin and Student Account Authentication Subsystem, and Admin Manage Course Database Subsystem. The different subsystems interact with each other through interfaces, with the open end of the link being a required interface and the lollipop being the provided interface. Also, the model is connected to a MySQL database.

In our component diagram the two frameworks, Laravel for server side model and controller, and react for the client side view, are described. The components of the model view controller system interact as following: The model requires connection to the SQL database through Database Connection port and control instructions provided by the controller. The view requires object oriented and control information in order to present the user with the system on a browser. The controller requires input from the user through the view.

Within the model, the database management subsystem can provide the user credentials in order to manage the account subsystem and allow a confirmed admin to modify the database. The account subsystem provides preferences and course management subsystems for students. Each of these subsystems (course management and preferences) each provide their own criteria to a schedule generator that will use the inputs in order to provide a schedule.

2.1.3 Physical View

Deployment diagram will be used to illustrate the physical view of the system. The application server where the Web server and the Database server are located. The Web server and the Database server will interact with each other to get the requested information. MySQL is a tool to get the information stored in the Database server.

The client component can use any devices such as Apple device, Windows Device or Linux device that come with a web browser like Mozilla Firefox, Google Chrome, Safari or Internet Explorer to connect to the server.

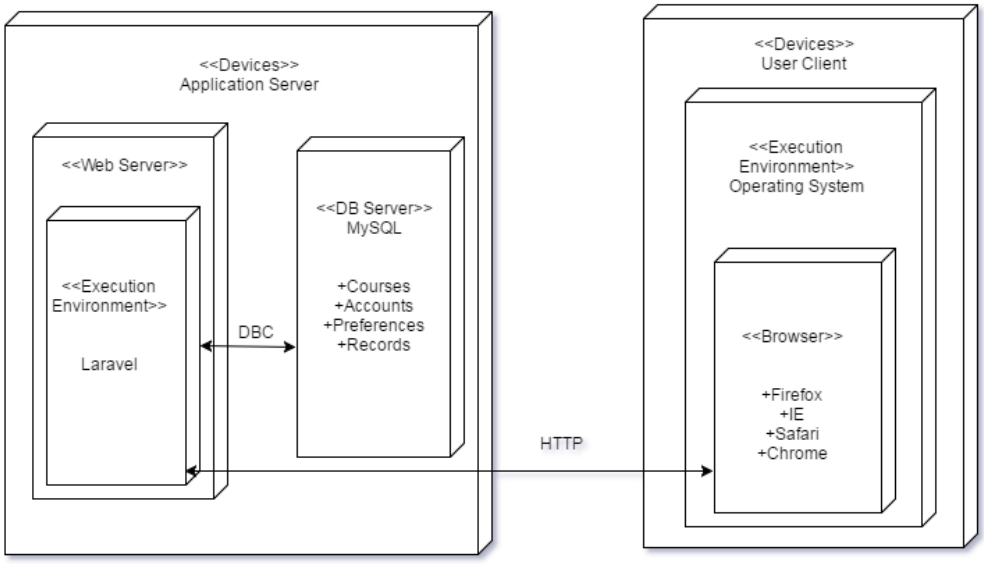


Figure 2: *Physical View: Deployment Diagram*.

2.1.4 Process View

The purpose of these diagrams are to show the path of system when actions and decisions are made by Users or Admins order to complete a specific task. In this case, showing the steps to generate a schedule and update account information are shown for the Student diagram. As for the admin diagram, representing the activities required in order to modify the database or to change a student's information can be observed.

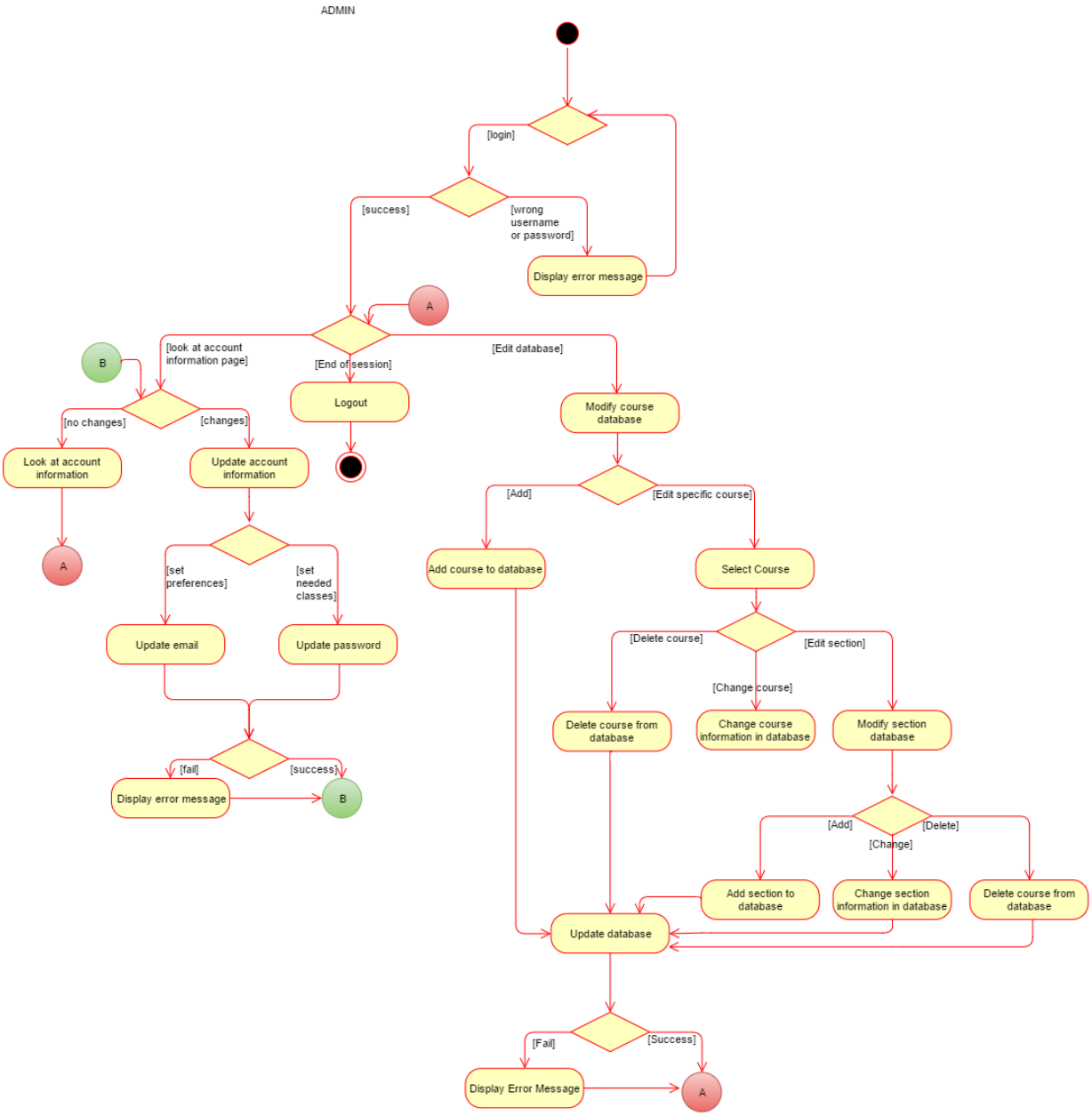


Figure 3: *Student Activity diagram*

The first activity diagram demonstrates the generation of a schedule by a User using The Scheduler system. Upon entering the website, the student is prompted with the decision to login using a username and password or create a new account. If the users are on their first visit, they will choose to sign up, otherwise, they can simply login. If the users input wrong data, an error message will be displayed. Once all the required information are given, the Scheduler opens the menu page, giving the choice to, have a look at the schedule, access the preferences and classes, open the account information page or logout.

If the users opt to look at their preferences and classes, they can either select to update them or to simply display them. If the former is chosen, the user will be prompted to set their preferences, classes and the needed classes. An error message will be displayed if anything went wrong and the system will bring back the users to update the preferences again, otherwise the choices will be saved or the schedule can be generated, which brings the user to the schedule display page.

In the event that the student decides to look at the schedule, it will either provide the user with an option to generate a schedule or it will display an error due to conflicting data, such as time conflicts for courses. After generating the schedule, the user can take a look at it and then choose to go back to the menu page.

The last option available on the menu page is the account information. If the users decide to choose this option, they can either update or simply display their account information. The information that are to be saved for the account are the email, username and password. A display error will be seen if there were any problems. Once the users are done using the Scheduler, they can logout.

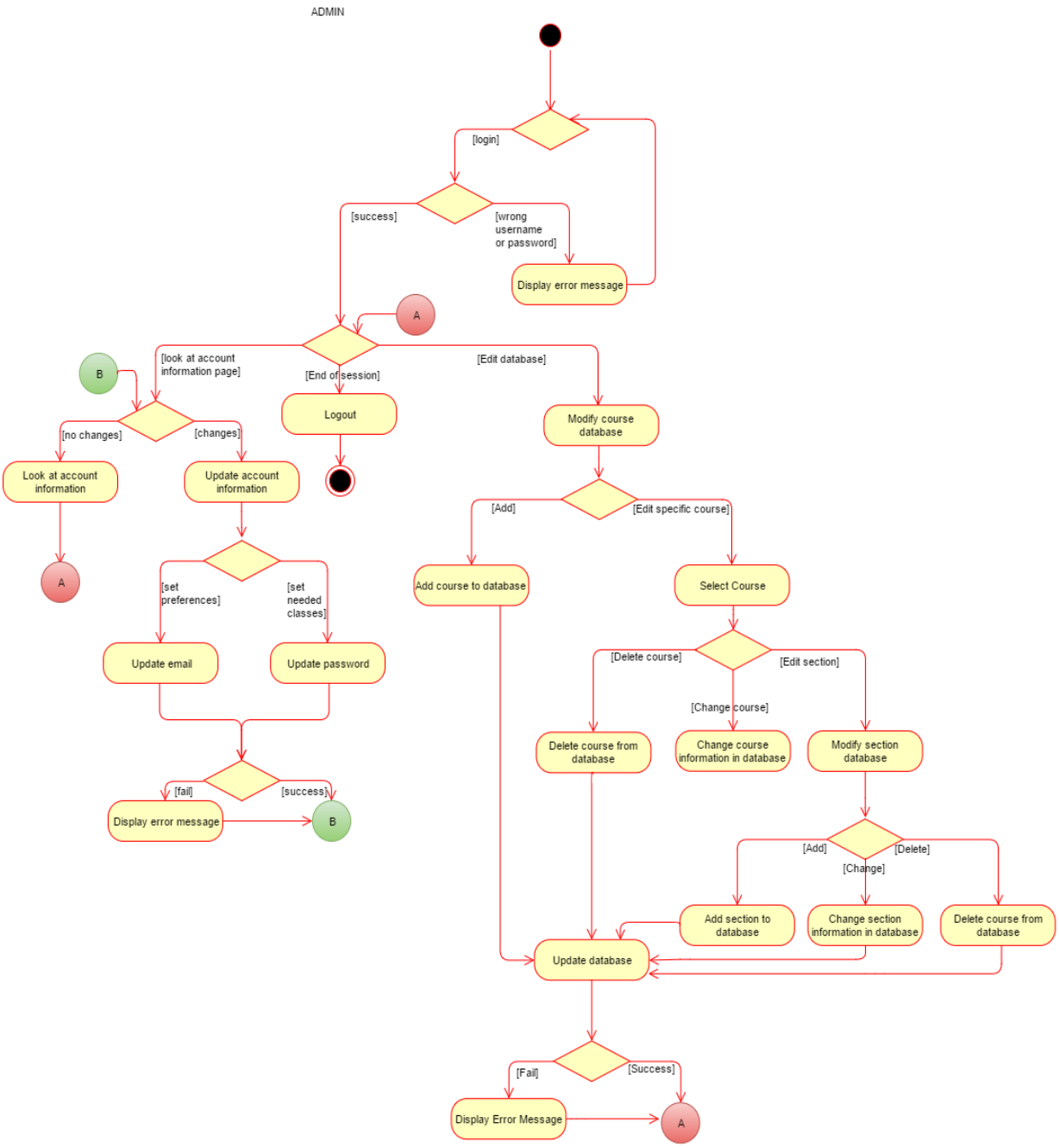


Figure 4: *Admin Activity diagram*

The second activity diagram shows the Admin's rights over the Scheduler system's database. When accessing the system, the admins also have to login. Successfully signing in to the website, brings up the menu page. The choices here are to open the account information page, which contains the students account information, to modify the course database and to logout.

In order to modify the database, the scheduler either requests to add a course or to edit a specific course. Choosing to select a course requires the admin to either delete it from the database, alter its description and information or modify the sections. The last option consists of adding deleting and changing information of the course section. After finishing working on the database, all the changes are saved.

For the account information page, this is used for any student that is unable to access the Scheduler for any reason and requires the assistance of an Admin. The Admin can look at the student's information and update their email or password. Upon modifying their login information, an error will be displayed if anything went wrong. If the changes were successful, the admin will be brought back to the account information page. Once the admins are done using the system, they can simply logout.

2.1.5 Scenarios

In the "4+1" architecture, the "1" stands for the *scenarios*. This section describes and illustrates the interactions between objects and process, or better said, between actors and use cases. The use case diagrams represent the functionalities and requirements of the system, such as generating a schedule, managing the account information, etc.

The system three types of users: public users, students, and admin. The *public users* can sign up into the system, and become a *student*, therefore interact with the system. As said before, the system goals is to generate a schedule out of the *student* preferences, course choices for the current semester and his own personalized list of taken classes. On the other side, the *admin* does not have the same functionalities as the *student* and a *public user*. First, an admin cannot sign in, as his/her account is already in the database. Second, the *admin*'s system goal is to mainly manipulate the course and section database. This includes adding, deleting, and modifying data. However, the *admin* and the *student* still shares some functionalities; both can login, edit their account (except the *admin* cannot change his username), reset their password if forgotten, login and logout.

Figure 5*: Admin Use Case Diagram*

Figure 6: *Student Use Case Diagram*

2.2 Subsystem Interfaces Specifications

The subsystem interfaces consist of: ManageCourses, PreferenceSettings, UserVerification, CourseAvailabilty, PreferenceImplementation, ScheduleGeneration. Each of the interfaces connects two subsystems together. In this section, each of these interfaces will be described and the function calls exchanged between its subsystems will be given along with the specific description of the parameters passed.

2.2.1 ManageCourses

The manage courses interface joins the provided interface of Admin and Student Managing Account Information Subsystem with the required interface of the Student Manage Courses Subsystems.

Classes involved	**Scheduler_Student, Student, Course**
List of Methods	1. addTakenCourses(JSON): boolean **Implemented in Class**: Scheduler Student **Description**: Method to add courses to the list of taken courses. **Input Parameter(s)**: serialized Course -course to be added **Return Type**: boolean 2. getTakenCourses(Student): Course[] **Implemented in Class**: Scheduler Student **Description**: Method to get the list of courses taken. **Input Parameter(s)**: User of type Student -user making inquiry **Return Type**: an array of type Course 3. addNeededCourses(JSON): boolean **Implemented in Class**: Scheduler Student **Description**: Method to add courses to the list of needed courses. **Input Parameter(s)**: serialized Course -courses to be aded **Return Type**: boolean 4. getNeededCourses(Student): Course[] **Implemented in Class**: Scheduler Student **Description**: Method to get the list of courses needed. **Input Parameter(s)**: User type Student -user making inquiry **Return Type**: an array of type Course

	5. autogenerateTakenClasses(int): String[] **Implemented in Class**: Scheduler Student **Description**: Method to generate a list of taken courses. **Input Parameter(s)**: int -number of classes t **Return Type**: an array of Strings -names of courses 6. getCoursesFromDB(int): void **Implemented in Class**: Scheduler Student **Description**: Method to access courses in database. **Input Parameter(s)**: none **Return Type**: N/A

2.2.2 PreferencesSettings

The preference settings interface joins the provided interface of Admin and Student Managing Account Information Subsystemwith the required interface of the Student Manage Preference Subsystem

Classes involved	**User Student, Preferences, Scheduler_Student**
List of Methods	1. setPreferences(User, Preferences): boolean **Implemented in Class**: Scheduler Student **Description**: Method to set the preferences for a user of type Student **Input Parameter(s)**: object of type Preferences and User of type Student -current user and their new preferences **Return Type**: boolean 2. getPreferences(User): Preferences **Implemented in Class**: Scheduler Student **Description**: Method to get the preferences **Input Parameter(s)**: User of type Student -current user and their preferences **Return Type**: An object of type Preferences

2.2.3 UserVerification

The UserVerification interface is provided by the user account in order to manage the database. The credentials entered in the UI are compared to the User's in order to establish a valid login type.

Classes involved	User
List of Methods	1. getName() : String **Implemented in Class**: User **Description**: Method to access the name of a user. **Input Parameter(s)**: N/A **Return Type**: String 2. getPassword() : String **Implemented in Class**: User **Description**: Method to access the password of a user. **Input Parameter(s)**: N/A **Return Type**: String 3. getEmail() : String **Implemented in Class**: User **Description**: Method to access the email of a user. **Input Parameter(s)**: N/A **Return Type**: String 4. login(Student) : void **Implemented in Class**: User **Description**: Method to access the student object associated with a user account. **Input Parameter(s)**: object of type Student **Return Type**: void

Verification of user login type is used to ensure proper access to the database.

Classes involved	UI
List of Methods	1. verificationUserType(User) : void **Implemented in Class**: UI **Description**: Method to access the user's dynamic type Student/Admin. **Input Parameter(s)**: object of type User -user to be verified **Return Type**: void

Enables an Admin to modify any User's information, allow Student to modify its own information.

Classes involved	**Scheduler_Student, Student_Admin**
List of Methods	1. changeAccountInformation(User) : boolean **Implemented in Class**: Scheduler Admin/Student **Description**: Method to modify a user's account information. **Input Parameter(s)**: object of type User -user who is verified as either Admin or Student. **Return Type**: boolean

2.2.4 AdminDataBaseModifications

This interface allows an administrator to edit the courses and sections list in the database. This is not accessible for a student because of the UserValidation interface.

Classes involved	**Scheduler_Admin, User Admin, Courses**
List of Methods	1. getCoursesFromDB(): void **Implemented in Class**: Scheduler Admin **Description**: Method to access all Courses in database. **Input Parameter(s)**: N/A **Return Type**: void 2. addCourseInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to add a Course in database. **Input Parameter(s)**: serialized Course -to be added **Return Type**: boolean 3. dropCourseInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to remove a Course from database. **Input Parameter(s)**: serialized Course -to be removed **Return Type**: boolean 4. modifyCourseInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to modify an existing Course in database. **Input Parameter(s)**: serialized Course -to be modified

	Return Type: boolean 5. modifySectionInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to modify an existing section of a course in database. **Input Parameter(s)**: serialized Section -to be modified **Return Type**: boolean 6. addSectionInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to add a Section for a Course in database. **Input Parameter(s)**: serialized Section -to be added **Return Type**: boolean 7. dropSectionInDB(JSON) : boolean **Implemented in Class**: Scheduler Admin **Description**: Method to remove a section for a course. **Input Parameter(s)**: serialized Section -to be removed **Return Type**: boolean

2.2.5 CourseAvailability

The Course availability Interface generates, from a list of Courses that a student may take, a list of course Sections that could match together in a schedule. This interface provides viewing of courses list, not modification.

The Scheduler Student provides a list of courses needed and their sections. This list is solely based on which course could be taken for the schedule; semester and prerequisites are the only constraints.

Classes involved	**Scheduler_Student, User Student, Course**
List of Methods	1. getTakenCourses(Student) : Course[] **Implemented in Class**: Scheduler Student **Description**: Method to access the list of courses taken. **Input Parameter(s)**: User of type Student -who has taken the courses **Return Type**: array of objects of type Course 2. getNeededCourses(Student) : Course[]

	Implemented in Class: Scheduler Student **Description**: Method to access the list of needed courses. **Input Parameter(s)**: User of type Student -who needs the courses **Return Type**: array of objects of type Course 3. getCoursesFromBD() : void **Implemented in Class**: Scheduler Student **Description**: Method to access the list of all courses. **Input Parameter(s)**: none **Return Type**: N/A 4. getSectionsForCourse(String) : Section[] **Implemented in Class**: Scheduler Student **Description**: Method to access the list of sections for a course. **Input Parameter(s)**: String -name of course **Return Type**: array of objects type Section

The Course attributes are accessed in order to create the schedule, more importantly they provide a list of Sections.

Classes involved	**Course, Section**
List of Methods	1. getName(): String **Implemented in Class**: Course **Description**: Method to access the name of a course. **Input Parameter(s)**: N/A **Return Type**: string 2. getNumber(): int **Implemented in Class**: Course **Description**: Method to access the number of a course. **Input Parameter(s)**: N/A **Return Type**: int 3. getPrereqs(): Courses[] **Implemented in Class**: Course

	Description: Method to access the list prerequisite courses of a course. **Input Parameter(s)**: N/A **Return Type**: array of objects of type Course -sections not initialized 4. getLectures(): Section[] **Implemented in Class**: Course **Description**: Method to access the list of lectures for a course. **Input Parameter(s)**: N/A **Return Type**: array of objects of type Course Section 5. getTutorial(): Section[] **Implemented in Class**: Course **Description**: Method to access the list of tutorials for a course. **Input Parameter(s)**: N/A **Return Type**: array of Course objects of type Section 6. getLabs(): Section[] **Implemented in Class**: Course **Description**: Method to access the list labs of a course. **Input Parameter(s)**: N/A **Return Type**: array of Course objects of type Section

Sections are contained within Courses, stored in the arrays lecture, tutorial and Lab -from which they get their section type. Sections provide availability and more precise scheduling information.

Classes involved	**Section**
List of Methods	1. getSemester(): String **Implemented in Class**: Section **Description**: Method to access the semester in which the section is taught. **Input Parameter(s)**: N/A **Return Type**: string

2.2.6 PreferenceImplementation

The preference implementation interface provides the student preferences in order to generate a schedule with the current student preferences.

Classes involved	Scheduler_Student, User
List of Methods	1. getPreferences(Student) : Preferences **Implemented in Class**: Scheduler **Description**: Method to access the preferences of a User of type Student. **Input Parameter(s)**: User type Student -who will access their preferences **Return Type**: object of type Preferences

2.2.7 ScheduleGeneration

The schedule generation interface is a combination of the course availability and the preferences of a student. A schedule is generated from a list of sections for the courses to be taken that are offered during the semester and have times corresponding with preferences. The Schedulegeneration then chooses sections from the lists that fulfil the requirements to be added to a schedule until it is filled. This schedule can then be displayed with section information - times and classrooms. In order to modify the schedule, preferences or courses to be taken can be modified to generate a new schedule.

Classes involved	Scheduler_Student, Student
List of Methods	1. generateSchedule(Student) : void **Implemented in Class**: Scheduler **Description**: Method to produce a schedule for a semester according to a list of courses and preferences. **Input Parameter(s)**: object of type Student **Return Type**: void.

More Information is required in order to draw a schedule: the course names, number and section.

Classes involved	Course, Section

List of Methods	1. getName(): String **Implemented in Class**: Course **Description**: Method to access the name of a course. **Input Parameter(s)**: N/A **Return Type**: string 2. getNumber(): int **Implemented in Class**: Course **Description**: Method to access the number of a course. **Input Parameter(s)**: N/A **Return Type**: int 3. getCredits(): double **Implemented in Class**: Course **Description**: Method to access the number of credits of a course. **Input Parameter(s)**: N/A **Return Type**: double 4. getPrereqs(): Courses[] **Implemented in Class**: Course **Description**: Method to access the list prerequisite courses of a course. **Input Parameter(s)**: N/A **Return Type**: array of objects of type Course -sections not initialized 5. getLectures(): Section[] **Implemented in Class**: Course **Description**: Method to access the list of lectures for a course. **Input Parameter(s)**: N/A **Return Type**: array of objects of type Course Section 6. getTutorial(): Section[] **Implemented in Class**: Course **Description**: Method to access the list of tutorials for a course. **Input Parameter(s)**: N/A **Return Type**: array of Course objects of type Section 7. getLabs(): Section[] **Implemented in Class**: Course

	Description: Method to access the list labs of a course. **Input Parameter(s)**: N/A **Return Type**: array of Course objects of type Section

Still more Information is required in order to draw a schedule: the section IDs, Times, Types, Classrooms.

Classes involved	**Section**
List of Methods	1. getID(): String **Implemented in Class**: Section **Description**: Method to access the id of a section. **Input Parameter(s)**: N/A **Return Type**: string 2. getTime(): String **Implemented in Class**: Section **Description**: Method to access the time of a section. **Input Parameter(s)**: N/A **Return Type**: string 3. getClassroom(): String **Implemented in Class**: Section **Description**: Method to access the location of a section. **Input Parameter(s)**: N/A **Return Type**: stringA **Return Type**: string

3. Detailed Design

3.1 Detailed Design Diagram

The following section describes a detailed description of the concerning the class diagram of the system. The scheduler contains a total of 10 classes. The classes *Student*, *Admin* and *User* represent essentially the user. *Student* and *Admin* are subclasses of the *User* class. This corresponds to the possibility of a student and an admin to log in to the system. The *User* class interacts with the *UI* class where this class simply manages the information to be displayed. For the classes *Scheduler_Admin* and *Scheduler_Student*, depending on the type of user, the *UI* will redirect that user the appropriate *Scheduler*

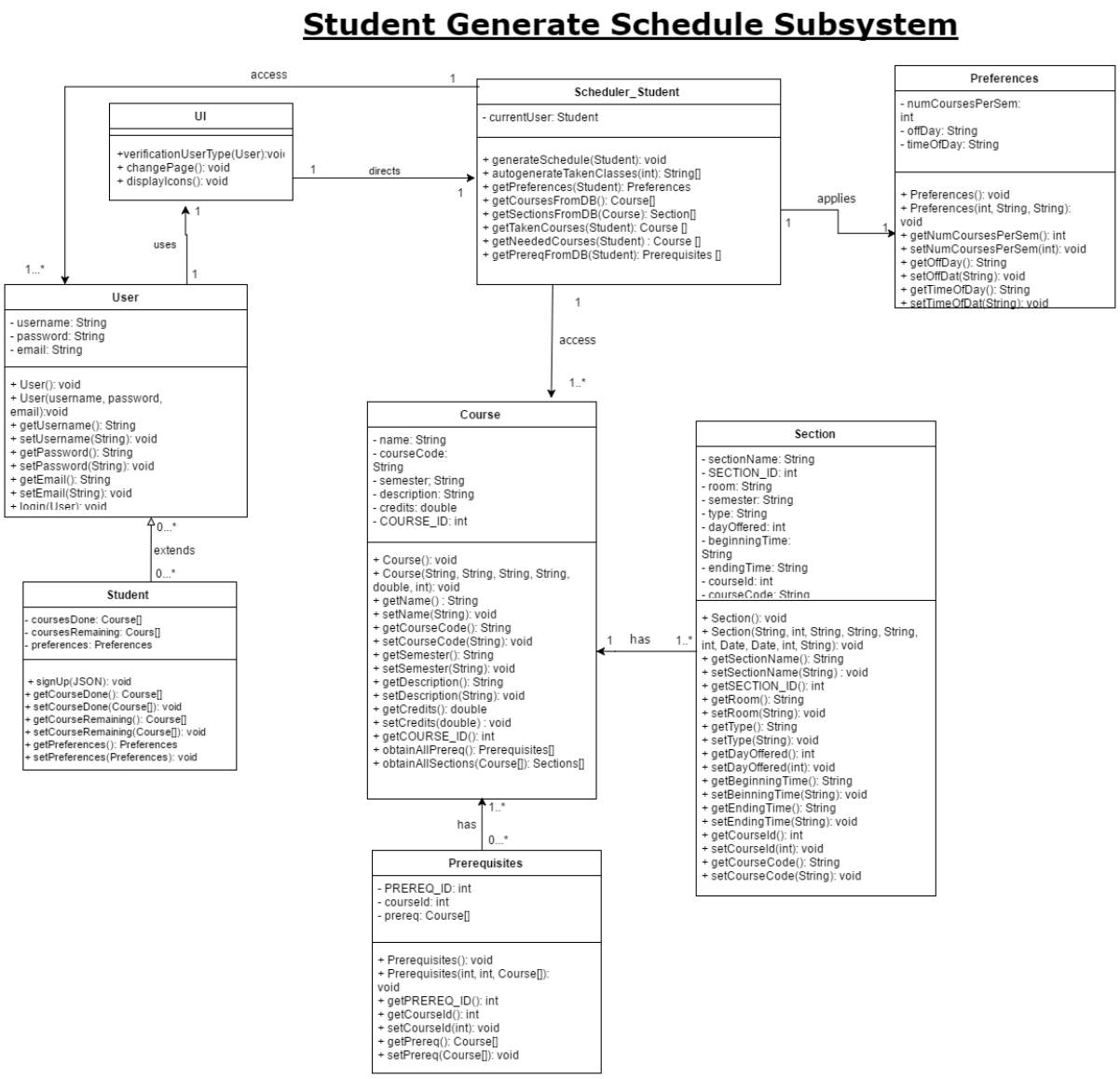
system. These classes are the core of the system since they provide functionality with the help of the other classes *Preferences, Classes, Sections & Prerequisites.*

3.1.1 Student Generate Schedule Subsystem

The *Student_Scheduler* has a method called *generateSchedule(Student)* which generates a schedule based on the courses, preferences and sections. A method called *autogenerateTakenClasses(int)* shall automatically display the courses that are added even before finalization. The subsystem contains *Preferences, Course, Section, Prerequisites* and *Scheduler_Student* as classes. Each of these classes communicate between each other to generate schedule. *Preferences, Courses, Section* and *Prerequisites* classes are only composed of getters and setters for their attributes. For more details, refer section 4.2. Student generated schedule is provided as part of the schedule generation component, the method listed below are elaborated:

Classes Involved	**Preferences, Course, Section, Prerequisites, Scheduler_Student**
Method(s) Implemented	generateSchedule(Student): void **Implemented in Class**: Scheduler_Student **Description**: Method that generates schedule for Student **Input Parameter(s)**: Student **Return Type**: void getTakenCourses(Student): Course [] **Implemented in Class**: Scheduler_Student **Description**: Method that retrieves taken courses from database **Input Parameter(s)**: Student **Return Type**: void getNeededCourses(Student): Course [] **Implemented in Class**: Scheduler_Student **Description**: Method that returns needed courses **Input Parameter(s)**: Student **Return Type**: Course [] getPreferences(Student): Preferences **Implemented in Class**: Scheduler_Student **Description**: Method that returns preferences of a user **Input Parameter(s)**: Student

	Return Type: Preferences getPrereqFromDB(Student): Prerequisites[] **Implemented in Class**: Scheduler_Student **Description**: Method that returns the prerequisites of all classes **Input Parameter(s)**: Student **Return Type**: Prerequisites[] autogenerateTakenClasses(int): String [] **Implemented in Class**: Scheduler_Student **Description**: Method that autogenerates taken classes **Input Parameter(s)**: int **Return Type**: String [] getCoursesFromDB(): Course[] **Implemented in Class**: Scheduler_Student **Description**: Method that returns all courses from the sequence of courses from the database **Input Parameter(s)**: - **Return Type**: Course[] getSectionsFromDB(Course): Section[] **Implemented in Class**: Scheduler_Student **Description**: Method that returns all sections of a specific course **Input Parameter(s)**: Course **Return Type**: Section []

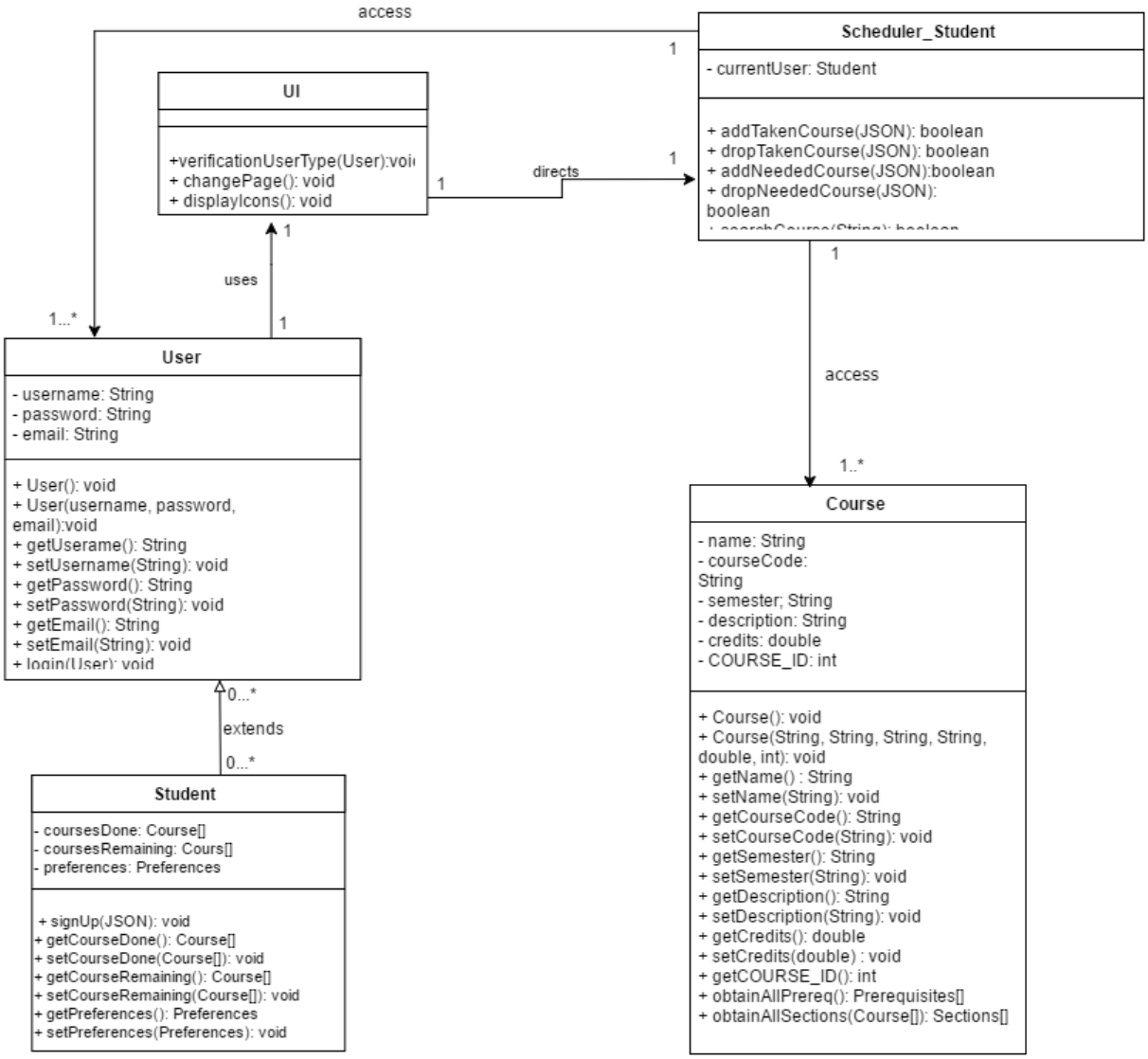


3.1.2 Student Manage Courses Subsystem

This subsystem portrays the path for a student to manage his course load. First, the course is searched with *searchCourse(String)* method. The scheduler possesses methods such as *addNeededCourse(JSON)*, *dropTakenCourse(JSON)*, which allows for the user to simply add courses to his/her schedule or simply drop them. The methods *dropTakenCourse(JSON)* and *addTakenCourse(JSON)* are functions where the user himself creates his own transcript. The project and specifications defines the courses taken in the previous semesters as inputs to be put into the database.

Classes Involved	Scheduler_Student
Method(s) Implemented	addTakenCourse(JSON): boolean **Implemented in Class**: Scheduler_Student **Description**: Method that adds a taken course in the scheduler **Input Parameter(s)**: JSON **Return Type**: boolean addNeededCourse(JSON): boolean **Implemented in Class**: Scheduler_Student **Description**: Method that adds a needed course that has not been taken yet in the scheduler **Input Parameter(s)**: JSON **Return Type**: boolean dropTakenCourse(JSON):boolean **Implemented in Class**: Scheduler_Student **Description**: Method that remove taken course requested by the student and returns a boolean **Input Parameter(s)**: JSON **Return Type**: boolean dropNeededCourse(JSON):boolean **Implemented in Class**: Scheduler_Student **Description**: Method that remove a course that is needed by the student and returns a boolean **Input Parameter(s)**: JSON **Return Type**: boolean searchCourse(String):boolean **Implemented in Class**: Scheduler_Student **Description**: Searches for a course through the database and returns true if course exists, false otherwise. **Input Parameter(s)**: String **Return Type**: boolean

Student Manage Courses Subsystem



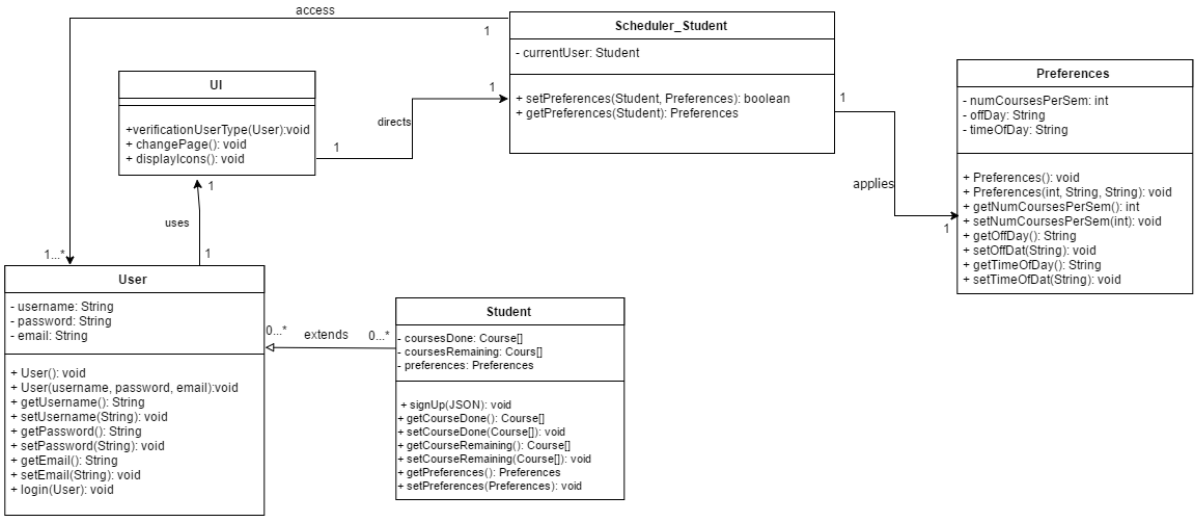
3.1.3 Student Manage Preferences Subsystem

For this subsystem, the scheduler contains two methods called *getPreferences(Student)* and *setPreferences(Student, Preferences)* inside class *Scheduler_Student* where the user who is a student can set and get the preferences. The *Preference* class contains attributes such as *timeOfDay*, *offDay* and *numCoursesPerSem*.

Class *Preferences* is mainly composed of getters and setters for its 3 attributes. For more details, please refer section 3.2.

Classes Involved	**Scheduler_Student, Preferences**
Method(s) Implemented	setPreferences(Student,Preferences):boolean **Implemented in Class**: Scheduler_Student **Description**: Method that returns a boolean for preferences modification **Input Parameter(s)**: Student and Preferences **Return Type**: boolean getPreferences(Student):Preferences **Implemented in Class**: Scheduler_Student **Description**: Method that returns preferences of a user **Input Parameter(s)**: Student **Return Type**: Preferences

Student Manage Preferences Subsystem



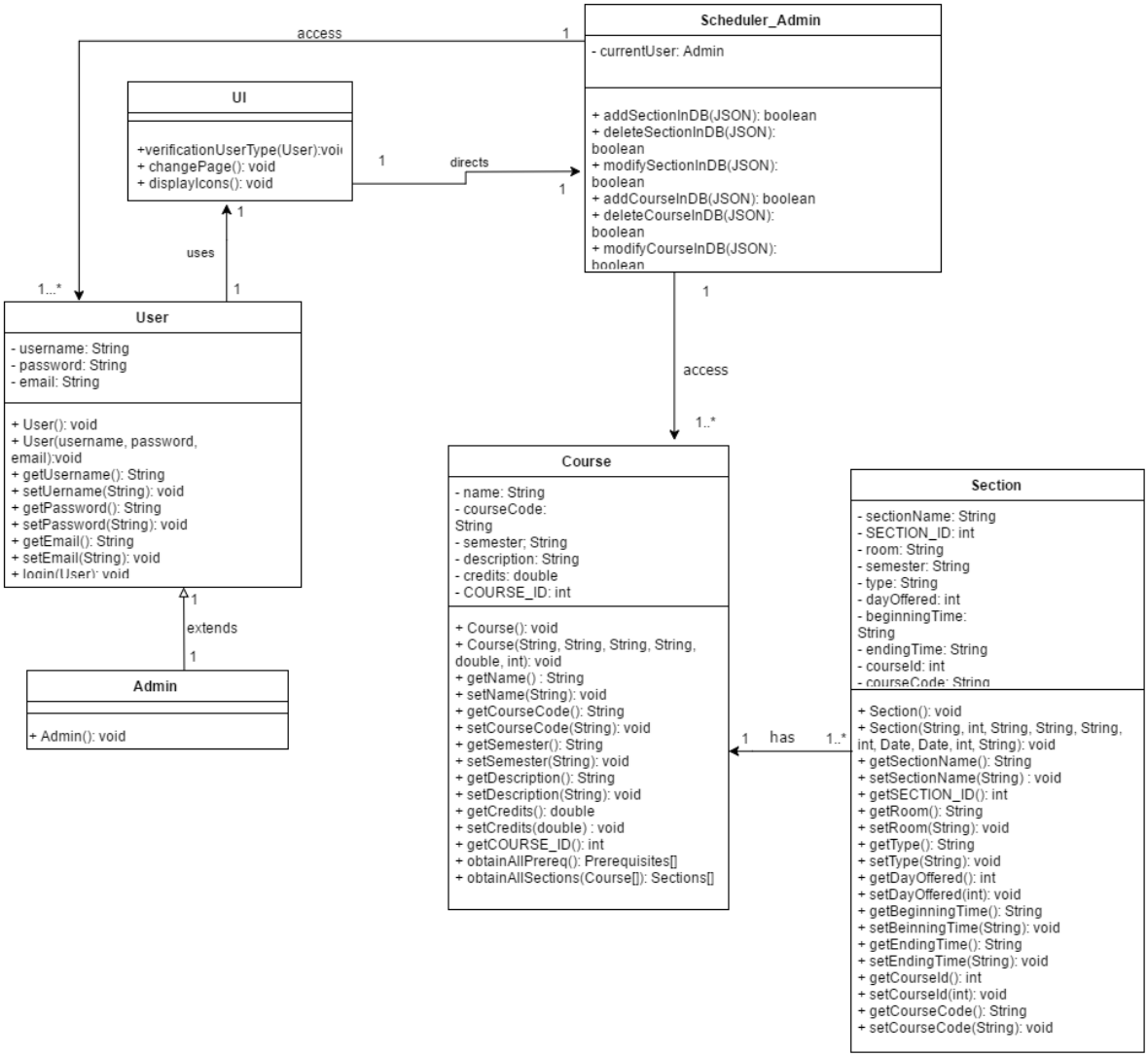
3.1.4 Admin Manage Course Database Subsystem

This subsystem portrays the relationship between the privileges of the admin and the database. The admin will set all the courses and sections that students can take through the *Scheduler_Admin* class. This system contains methods that manipulate the database such as *addSectionInDB()*, *modifyCourseInDB()*. *Course* and *Section* classes are only composed of getters and setters. For more details, please refer to section 3.2.

Classes Involved	**Scheduler_Admin, Course, Section,**
Method(s) Implemented	addSectionInDB(JSON): boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that adds a section in the database **Input Parameter(s)**: JSON **Return Type**: boolean dropSectionInDB(JSON): boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that removes a section from the database **Input Parameter(s)**: JSON **Return Type**: boolean modifySectionInDB(JSON):boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that modifies section for a course inside database **Input Parameter(s)**: JSON **Return Type**: boolean addCourseInDB(JSON):boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that adds course inside database **Input Parameter(s)**: JSON **Return Type**: boolean dropCourseInDB(JSON):boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that removes course from database

	Input Parameter(s): JSON **Return Type**: boolean modifyCourseInDB(JSON):boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that modifies the course inside database **Input Parameter(s)**: JSON **Return Type**: boolean

Admin Manage Course Database Subsystem



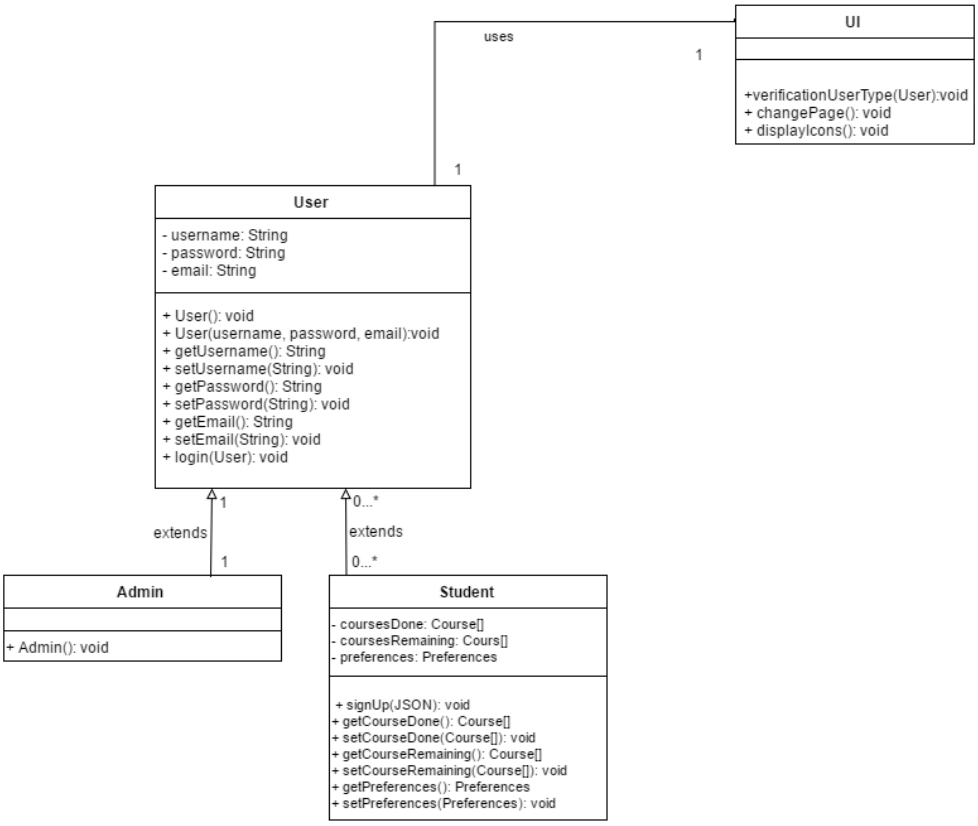
3.1.5 Admin and Student Account Authentication Subsystem

This subsystem has one functionality which is to verify which type of user is accessing the system(student/admin). The *UI* class represents the bridge between the systems and will direct the user to their corresponding Scheduler system depending on their type(student/admin). It is important to note both *admin* and *student* can login, but only a *student* can sign up, as no one can register as an *admin* and have access to the admin page.

User and *Student* are both composed of attributes, and each class has getters and setters for those attributes. For more details, please refer section 3.2.

Classes Involved	**Student, Admin, User, UI**
Method(s) Implemented	login(User): void **Implemented in Class**: User **Description**: Method that allows user to login. For security purposes, hashing will be used in order to avoid potential breaches and unwanted access. **Input Parameter(s)**: User **Return Type**: void signUp(JSON): void **Implemented in Class**: Student **Description**: Method that signs up the student **Input Parameter(s)**: JSON **Return Type**: void verificationUserType(User): void **Implemented in Class**: UI **Description**: Method that checks the user type and redirects to their respective scheduler according to their type **Input Parameter(s)**: User **Return Type**: void

Admin and Student Account Authentication Subsystem



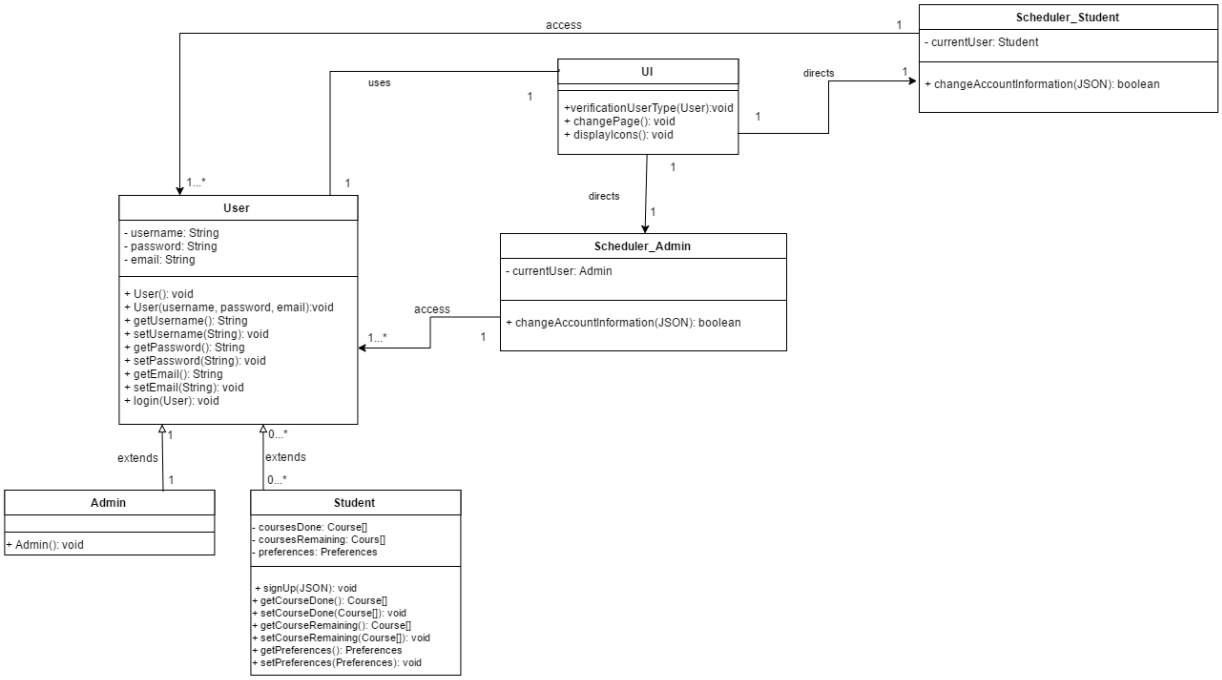
3.1.6 Admin and Student Managing Account Information Subsystem

In this subsystem, both Scheduler systems(admin/student) contain a method that allows for the user to change its account information: *changeAccountInformation()*. This subsystem simply permits the user to change his/her information and characteristics inside the *User* and *Student* classes, such as *username*, *password*, *email* for both *Student* and *Admin*, but also *courseDone*, *coursesRemaining* and *preferences* inside of *Student*. For more details regarding the setters and getters of this attributes, please refer to section 3.2.

Classes Involved	**Student, User, Admin, Scheduler_Student, Scheduler_Admin**
Method(s) Implemented	changeAccountInformation(JSON): boolean **Implemented in Class**: Scheduler_Admin **Description**: Method that changes the account information of the admin. It returns true if the changes are successful, false otherwise.

	Input Parameter(s): JSON **Return Type**: boolean changeAccountInformation(JSON): boolean **Implemented in Class**: Scheduler_Student **Description**: Method that changes the account information of a student. It returns true if the changes are successful, false otherwise. **Input Parameter(s)**: JSON **Return Type**: void

Admin and Student Manage Account Information Subsystem



3.2 Unit Descriptions

Class Name	**Admin**
Description	A type of user inherited from the User class representing the Admin.
Attribute(s)	N/A
Operation(s)	Ø Admin(): void o Creates a preset Admin object.

Class Name	Student
Description	A type of user inherited from the User class representing the Student.
Attribute(s)	• coursesDone : Course[] • coursesRemaining : Course[] • preferences : Preferences
Operation(s)	Ø Student(): void o Creates a preset Student object. Ø signUp(JSON_File : JSON): Boolean o Registers the student into the system for the first time. Ø getCourseDone(): Course[] o Return an array of already taken/done courses for the student Ø setCourseDone(coursesDone: Course[]): void o Changes the array of already taken/done courses for the student Ø getCourseRemaining(): Course[] o Return an array of remaining/needed courses for the student Ø setCourseRemaining(coursesRemaining: Course[]): void o Changes the array of remaining/needed courses for the student Ø getPreferences(): Preferences o Returns student's preferences Ø setPreferences(preferences: Preferences): void o Changes the preferences of the student

Class Name	User
Description	The generic model of the user of the system from which Student and Admin inherent from.
Attribute(s)	• username : String • password: String • email : String
Operation(s)	Ø User (username : String, password : String, email : String): void o Generates a new account for a new user.

	Ø User (): void o Creates a preset User object. Ø getName(): String o Returns the user's username. Ø setName(newUsername): String o Allows the users to change their username Ø getPassword(): String o Returns the user's password. Ø setPassword(oldpassword : String, newpassword : String): void o Allows the users to the change their password Ø getEmail(): String o Returns the user's email. Ø setEmail(newEmail : String): void o Changes the user's email. Ø login(StudentObj: Student): void o Provides the student user access to the system.

Class Name	**UI**
Description	Provides the front end user interface (GUI) to the user.
Attribute(s)	N/A
Operation(s)	Ø VerificationUserType(UserObj : User): void o Determines whether a user is a Student or Admin and only gives them access to parts of the system that they are allowed to interact with. Ø changePage(): void o Changes the page displayed to User based on User's interaction with the system. Ø displayIcons(): void o Displays different icons depending on page being displayed to User such as buttons and checkboxes.

Class Name	Preferences
Description	Manages and stores the preferences the Student User selects if any. Used by Scheduler_Student.
Attribute(s)	● numCoursesPerSem : int ● offDay : String ● timeofDay : String
Operation(s)	Ø Preferences(): void o Initializes preferences for a Student. Ø Preferences(): void o Generates the preferences of the Student User. Ø getNumCoursesPerSem(): int o Returns the number of courses a student wishes to take per semester. Ø setNumCoursesPerSem(numCourses : int): void o Student sets the number of courses they wish to take per semester. Ø getOffDay(): String o Returns the selection of days the Student selected to try and have off in a semester. Ø setOffDay(dayOff : String): void o Student sets the days they wish to have no classes. Ø getTimeOfDay(): String o Displays the time of the day the Student wishes to not have classes at. Ø setTimeOfDay(timeOfDay : String): void o Sets the time of day Student would like to have classes at.

Class Name	Course
Description	The object model of an academic course inside the system from which Section is derived. Used by Scheduler_Student to construct a Schedule for the Student User.
Attribute(s)	● name : String ● courseCode: String

	• semester : String • Description: String • credits: double • COURSE_ID: int
Operation(s)	Ø Course(): void o Creates an object of type Course and initializes all attributes. Ø Course(name: String, code: String, semester: String, desc: String, credits: int, id: int): void o Creates and object of type Course and sets all attributes. Ø getName(): String o Returns name of the course Ø setName(newName: String): void o Changes the name of the course Ø getCourseCode(): String o Returns name of the course code Ø setCourseCode(code: String): void o Changes the name of the course code Ø getDescription(): String o Returns the description of a course. Ø setDescription(desc: String): void o Changes the description of the course Ø getCredits(): double o Returns the number of credits for the Course. Ø setCredits(credit: double): void o Changes the number of credits for the Course. Ø getCOURSE_ID (): int o Returns a final and unique id for a course, a number between 1 and ~140). This attribute cannot be modified. Ø obtainAllPrereq (): Prerequisites[] o Returns an array of all prerequisites and corequisites from the DB. Ø obtainAllSections (neededCourses: Course[]): Section[] o Returns an array of all sections from the needed classes.

Class Name	Section
Description	Contains the information of a particular section for a given Course which can be accessed and manipulated by an Admin, and also utilized to generate the schedule.
Attribute(s)	● sectionName: String ● SECTION_ID: int ● room: String ● semester: String ● type: int ● dayOffered: int ● beginningTime: String ● endingTime: String ● courseId: int ● courseCode: String
Operation(s)	Ø Section(): void o Creates an object of type Section and initializes all attributes. Ø Section(name: String, id: int, room: String, semester: String, type: int, days: int, beginning: String, end: String, courseID: int, courseCode: String): void o Creates an object of type Section and sets all attributes Ø getSectionName(): String o Returns the section name . Ø setSectionName(name: String): void o Changes the name of a section. Ø getSECTION_ID(): String o Returns a final and unique id for a section, a number between 1 and ~400). This attribute cannot be modified. Ø getRoom(): String o Returns the classroom number. Ø setRoom(room: String): void o Changes the classroom/location of a section. Ø getSemester(): String o Returns and displays the semester the Section is being offered. The

	possibilities are Fall, Winter, Summer. Ø setSemester(sem:String): void o Changes the semester that a section is offered at. The possibilities are Fall, Winter, Summer. Ø getType(): String o Returns and displays the type of the Section (Lab, Tutorial or Lecture). Ø setType(newType: String): void o Changes the type of the Section (Lab, Tutorial or Lecture). Ø getDayOffered(): int o Returns the days at which the section is offered. 1 represents Monday, 2 represents Tuesday, 24 represents Tuesday-Thursday, and so on. Ø setDayOffered(day: int): void o Changes the day(s) at which the section is offered. Ø getBeginningTime(): String o Returns the time at which a section begins, in the hh:mm:ss format. Ø setBeginningTime(time: String): void o Changes the beginning time of a section. Ø getEndingTime(): String o Returns the time at which a section ends, in the hh:mm:ss format. Ø setEndingTime(time: String): void o Changes the ending time of a section. Ø getCourseId(): int o Returns the ID of the course. Ø setCourseId(id: int): void o Changes the id of a course. Ø getCourseCode(): String o Returns the coruse code of a course. Ø setCourseCode(code: String): void o Changes the course code of a section.

Class Name	Prerequisites
Description	Contains all prerequisites and corequisites of a specific course. This is only used in generating a schedule.
Attribute(s)	• PREREQ_ID: int • courseId: int • prereq: Course[]
Operation(s)	Ø Prerequisites(): void o Creates an object of type prerequisites and initializes all attributes. Ø Prerequisites(prereqId: int, courseId: int, prereq: Course[]): void o Creates an object of type Prerequisites and sets all attributes Ø getPREREQ_ID(): String o Returns a final and unique id for one set of prerequisites for one course, a number between 1 and ~140). This attribute cannot be modified. Ø getCourseId(): int o Returns the id number of the course. Ø setCourseId(id: int): void o Changes the id number of the course. Ø getPrereq(): Course[] o Returns and displays the semester the Section is being offered. The possibilities are Fall, Winter, Summer. Ø setPrereq(prereqs: Course[]): void o Changes the set of courses that a prerequesites/corequisites for a specific course.

Class Name	Scheduler_Admin
Description	The object model that allows only an Admin User to modify all aspects of the database.
Attribute(s)	• currentUser: Admin
Operation(s)	Ø changeAccountInformation(UserObj : User): boolean

	o Allows an Admin User to change their account information. Ø addSectionInDB(JSON_File : JSON): boolean o Adds a new Section object to the database. Ø dropSectionInDB(JSON_File : JSON): boolean o Removes a Section object from the database. Ø modifySectionInDB(JSON_File : JSON): boolean o Admin modifies the information of a Section object inside the database. Ø addCourseInDB(JSON_File : JSON): boolean o Adds a new Course object to the database. Ø dropCourseInDB(JSON_File : JSON): boolean o Removes a Course object from the database. Ø modifyCourseInDB(JSON_File : JSON): boolean o Admin modifies the information of a Course object inside the database. Ø SchedulerAdmin(): void o Generates a new Scheduler_Admin object which allows an Admin User to modify the database.

Class Name	**Scheduler_Student**
Description	The object model that creates a schedule for the Student based on information taken from the Preferences, Course, Section and Student classes.
Attribute(s)	• currentUser: Student
Operation(s)	Ø addTakenCourse(JSON_File : JSON): boolean o Updates database JSON file with record of Courses taken by Student. Ø addNeededCourse(JSON_File :JSON): boolean o Updates database JSON file with record of Courses that Student User needs to take. Ø dropTakenCourse(JSON_File : JSON): boolean o Removes a Course that was taken by Student from database JSON file. Ø dropNeededCourse(JSON_File : JSON): boolean o Removes a Course that Student needs to take from JSON file. Ø getTakenCourses(UserObj : Student): Course [ ]

	o Returns an array of Course that the Student has taken. Ø getNeededCourses(UserObj : Student): Course [ ] o Returns and displays an array of Course that Student needs to take. Ø changeAccountInformation(JSON_File : JSON): boolean o Student User may change certain information on their account. Ø generateSchedule(UserObj : Student): void o Generates and displays a new schedule to the Student based on information from Preferences, database, Course and Section. Ø autoGenerateTakenClasses(numj : int): String [ ] o Generates and displays to Student classes that have already been taken. Ø getCoursesFromDB(): Course [ ] o Retrieves all Courses from the database. Ø getSectionsFromDB(CourseObj : Course): Section [ ] o Retrieves all Course Section(s) from the database. Ø getPreferences(UserObj : Student): Preferences o Retrieves all of Student's selected preferences. Ø setPreferences(UserObj : Student): boolean o Student can set their preferences and are told whether action was successful or not. Ø getPrereqFromDB(UserObj : Student): Prerequisites [] o Returns all prerequesites of all courses Ø SchedulerStudent(): void o Default constructor. Ø searchCourse(courseCode: String): boolean o Returns true if course exist in database, false otherwise.

4. Dynamic Design Scenarios

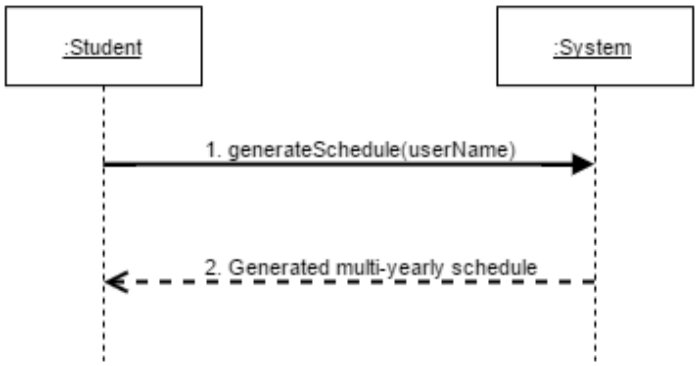
4.1 Generate Schedule

Generate schedule, as the name says, consists of creating a schedule for a student actor by using his/her preferences, list of taken classes and list of needed classes.

4.1.1 Full Use Case

Name:	Generate Schedule	Author	Ying-Chen Chu
Identifier:	UC13	Version:	3.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Generate a schedule		
Summary:	Based on the course selection list and or without the list of preferences, the system generates schedule(s) for the selected courses.		
Related use-cases:	UC8, UC9, UC12		
Preconditions	1. User has been authenticated. 2. Schedule preferences already set.		
Trigger:	User activates the "Generate Schedule" process		
Basic Flow:	1. User indicates the wish to generate a schedule.by selecting the “Build Schedule” option. 2. System verifies entries in lists of taken courses, needed courses & schedule preferences saved to the system via UC8, UC9 & UC11 and displays schedule.		
Post-Conditions:	**Success:** Schedule with the selected courses is generated.		
Minimum Guarantee:	Previous state of the system remains the same. **Failure:** System fails to generate a schedule and an error message displays		
Risk Assessment:	High		

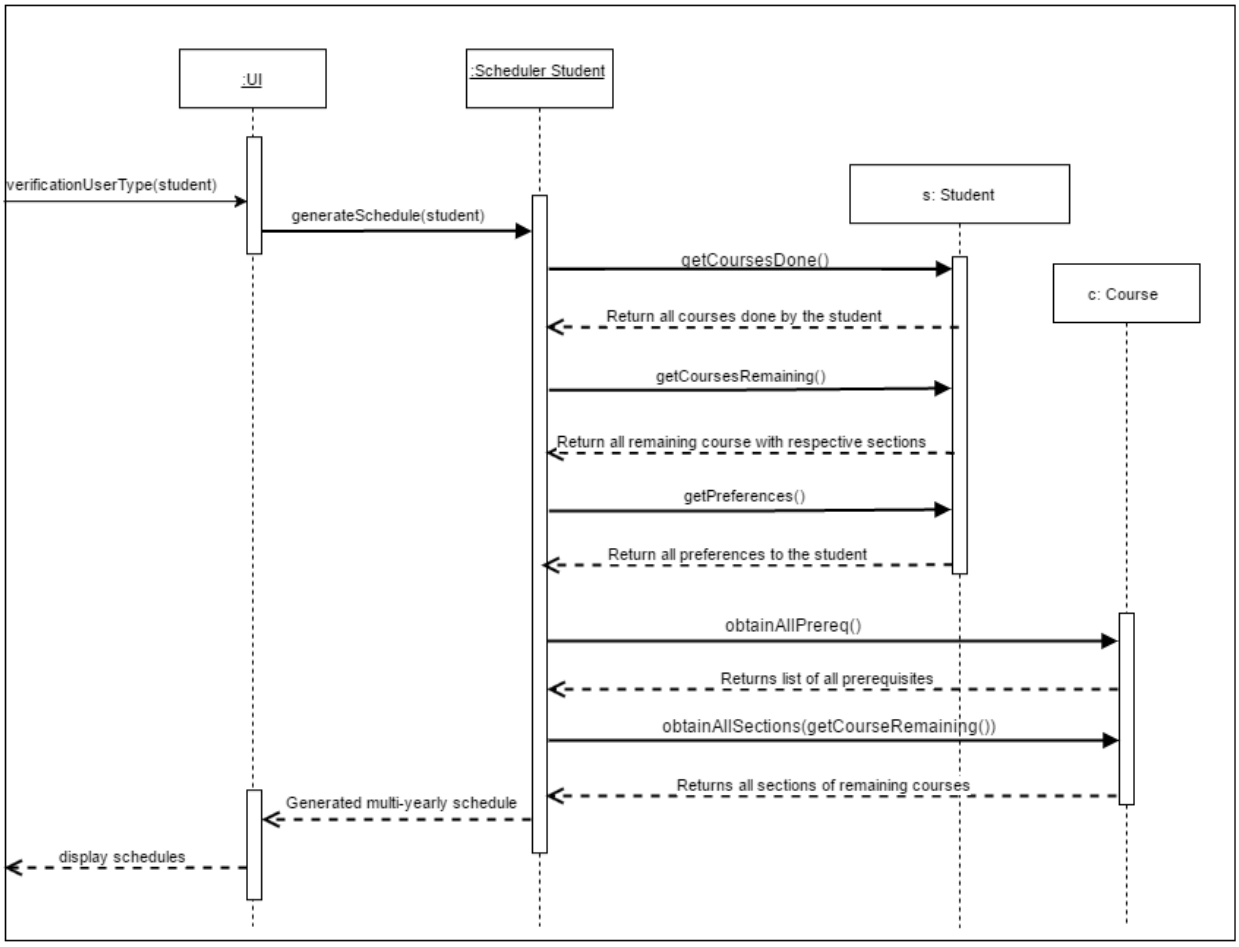
4.1.2 System Sequence Diagram



1. generateSchedule*(userName)* Operating Contracts & Respective Sequence Diagram

Name	Contract 1.1 Generate Schedule
Operation	generateSchedule(String)
Cross Reference	UC13 Generate Schedule
Pre conditions	-User is logged into the system -User has set needed/taken courses, and preferences
Post conditions	N/A

Sequence Diagram for contract 1.1



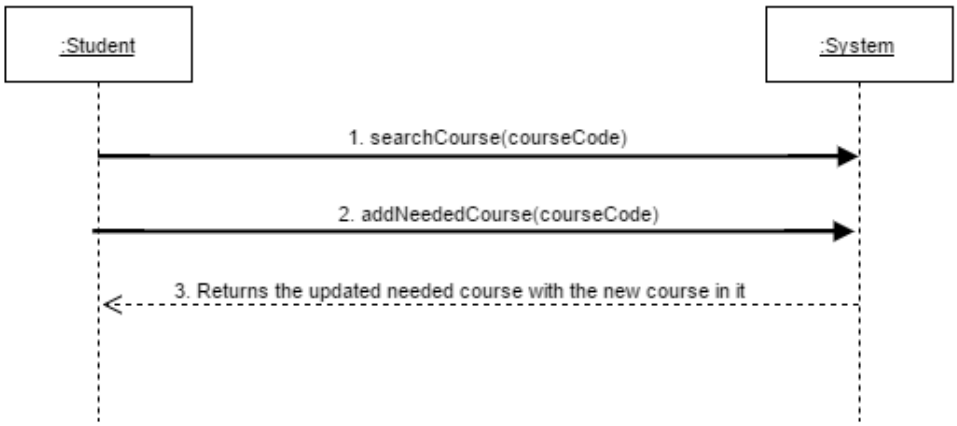
4.2 Set Needed Course

Set needed course consists of adding a course the student wants to the list. The course has to be first searched. If it is found, then it can be added to the list.

4.2.1 Full Use Case

Name:	Set Needed Courses	Author:	Ying-Chen Chu
Identifier:	UC9	Version:	2.0
Date Created:	2015-02-03	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Add a needed course to the schedule		
Summary:	Select a course to be added to the list of courses for the schedule generator.		
Related use-cases:	-		
Preconditions	1. User has been authenticated 2. Course requirements are met		
Trigger:	User selects the set needed course process.		
Basic Flow:	1. User indicates the wish to add a course by selecting a course from the needed courses list. 2. User commits by selecting the “add class” option. 3. System checks for eligibility and time conflict and adds course to list of needed courses.		
Post-Conditions:	**Success:** Course is added to the user's list of courses		
Minimum Guarantee:	Previous state of the system remains unchanged. **Failure:** System fails to process task and displays an error message.		
Risk Assessment:	Low		

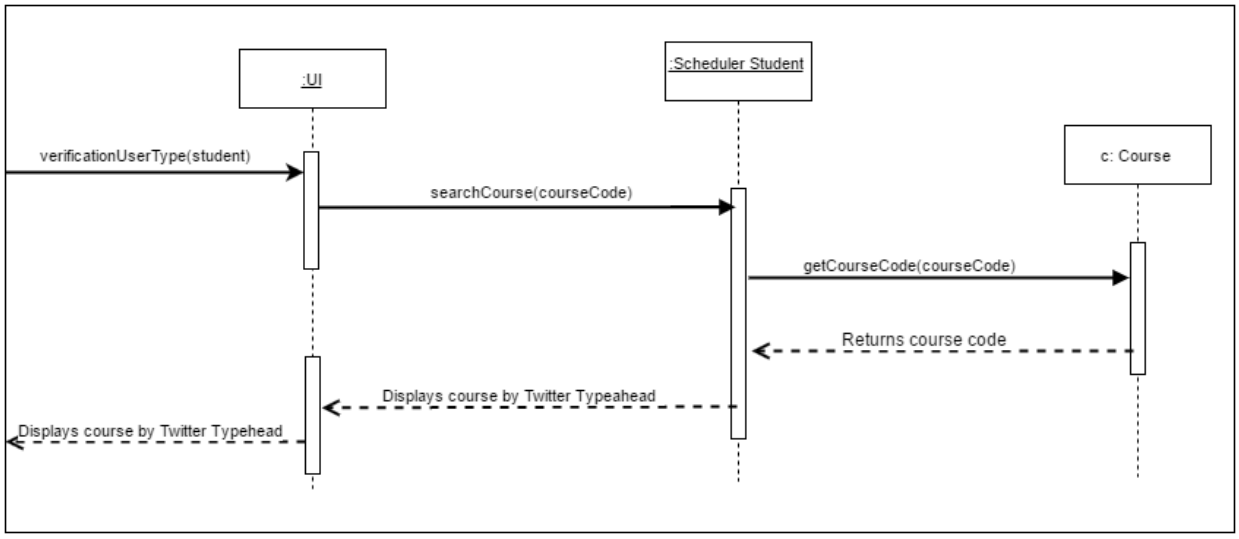
4.2.2 System Sequence Diagram



1. *searchCourse(courseCode)* Operating Contracts

Name	Contract 2.1 Search course in database
Operation	searchCourse(String)
Cross Reference	UC9 Set Needed Courses
Preconditions	-User is logged in.
Postconditions	-Association is made between inputted course code and course code inside the system.

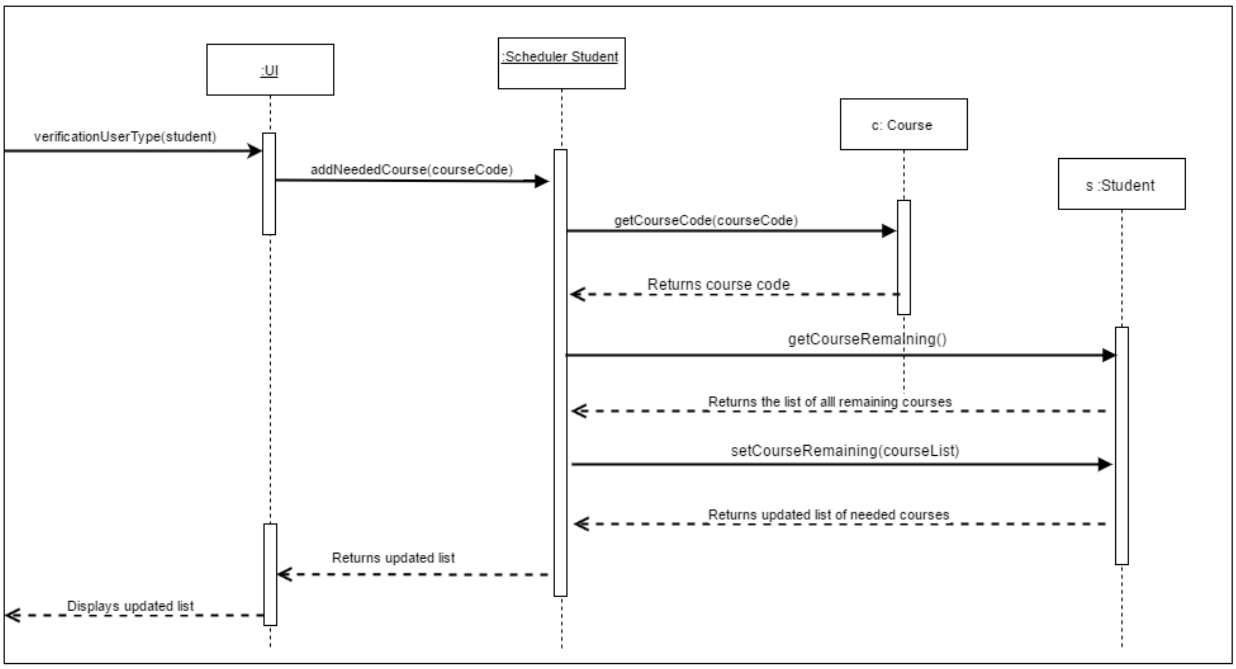
Sequence diagram for contract 2.1



2. *addNeededCourse(courseCode)* Operating Contracts

Name	Contract 2.2 Add needed course
Operation	addNeededCourse(String)
Cross Reference	UC9 Set Needed Courses
Pre conditions	-User is logged-in into the system -Inputted course exists into the system
Post conditions	-Instance of course has been created and added to the array of needed course of the student.

Sequence diagram for contract 2.2



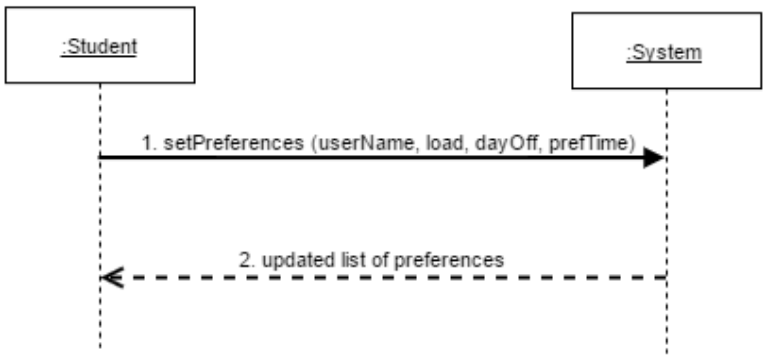
4.3 Set Preferences

Set preferences consists of saving the students preferences into the database, so they can be used later while generating the schedules.

4.3.1 Full Use Case

Name:	Set Preferences	Author	Ying-Chen Chu
Identifier:	UC12	Version:	3.0
Date Created:	2015-02-03	Last Modified:	2015-02-08
Importance:	5/5		
Actor(s):	Student, Administrator		
Goal:	Input the schedule preferences to the system		
Summary:	Set the schedule preferences and save them to the system.		
Related use-cases:	-		
Preconditions	User has been authenticated		
Trigger:	Previous state of the system remains unchanged.		
Basic Flow:	1. User indicates the wish to set schedule preferences.by selecting the desired “Day Off” and “Preferred Time” from drop down menus. 2. System registers selections & displays schedule preferences.		
Post-Conditions:	**Success:** The user’s preferences are saved to the system.		
Minimum Guarantee:	Previous state of the system remains unchanged. **Failure:** System fails to process the task, and an error messages is displayed.		
Risk Assessment:	Low		

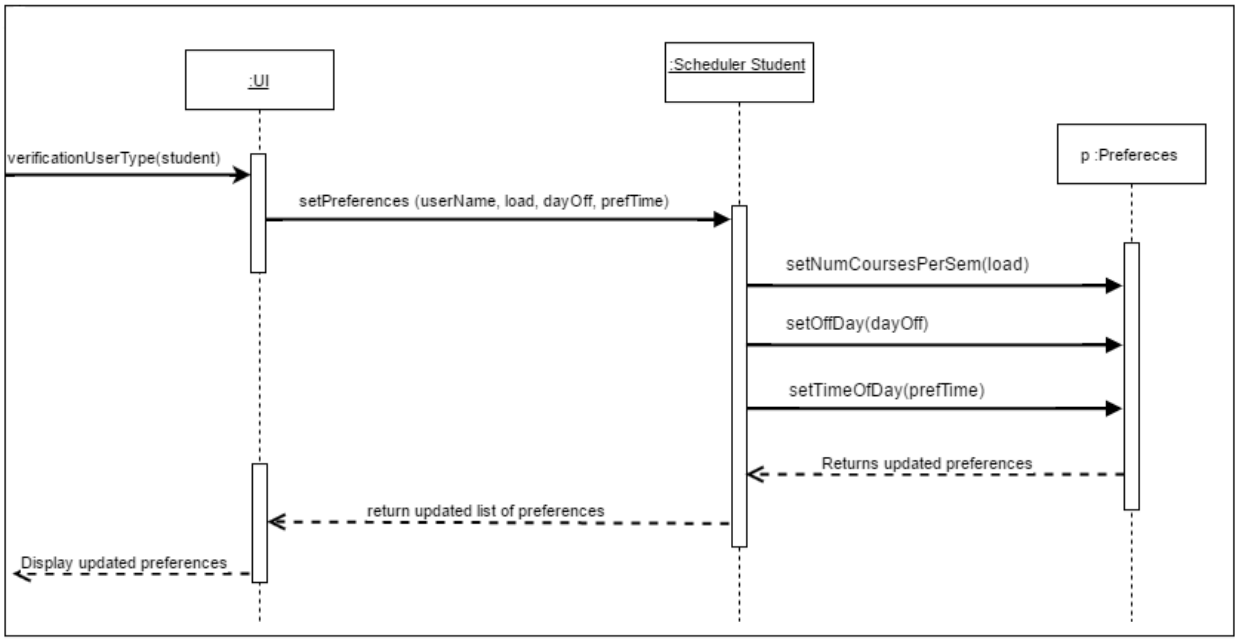
4.3.2 System Sequence Diagram



1. *setPreferences (userName, load, dayOff, periodDay)* Operating Contracts

Name	Contract 3.1 Set Prefrences
Operation	setPreferences(String, int, String, String)
Cross Reference	UC12 Set Preferences
Pre conditions	-Student logged into the System
Post Conditions	-Attributes of preferences are modified and saved . - Preferences are associated with the user.

Sequence Diagram for Contract 3.1



4.4 Add Course to Program

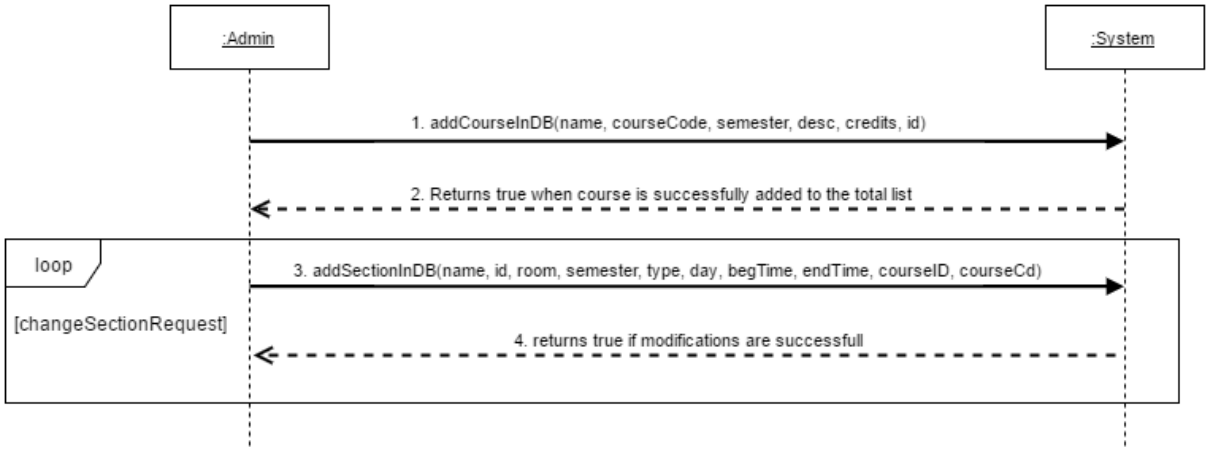
Add course to program consists of adding a course to the entire database of courses by the admin. By adding a course, the admin has to add the corresponding sections for lab, tutorial and lecture to this course.

4.4.1 Full Use Case

Name:	Add Course to Program	Author	Adil Hssaini
Identifier:	UC23	Version:	2.0

Date Created:	Feb 2, 2016	Last Modified:	April 12, 2016
Importance:	5/5		
Actor(s):	Administrator		
Goal:	To add a new course to a specific program.		
Summary:	The Administrator updates the list of required courses for a specific program by adding a new course.		
Related use-cases:			
Preconditions	1. Actor is logged on as administrator. 2. System has accessed the program menu. 3. Admin has accessed to the add course main menu		
Trigger:	Administrator activates the "Add Course to Program" process.		
Basic Flow:	1. Administrator initiates the add a course process by entering and submitting the course Name, ID, Code, Semester, Description, & Number of Credits. 2. System verifies non-duplicity of information and adds supplied course information to the list of program courses. 3. Administrator enters and submits Section name, Id, Room, Semester, Type, day, Beginning Time, End Time, Course ID, Course Cd. 4. System updates course information with additional information provided.		
Post-Conditions:	**Success:** Course is added successfully to the program listing.		
Minimum Guarantee:	List of courses in the program stored by the system will not be affected. **Failure:** The system fails to process the task and displays an error message.		
Risk Assessment:	Low		

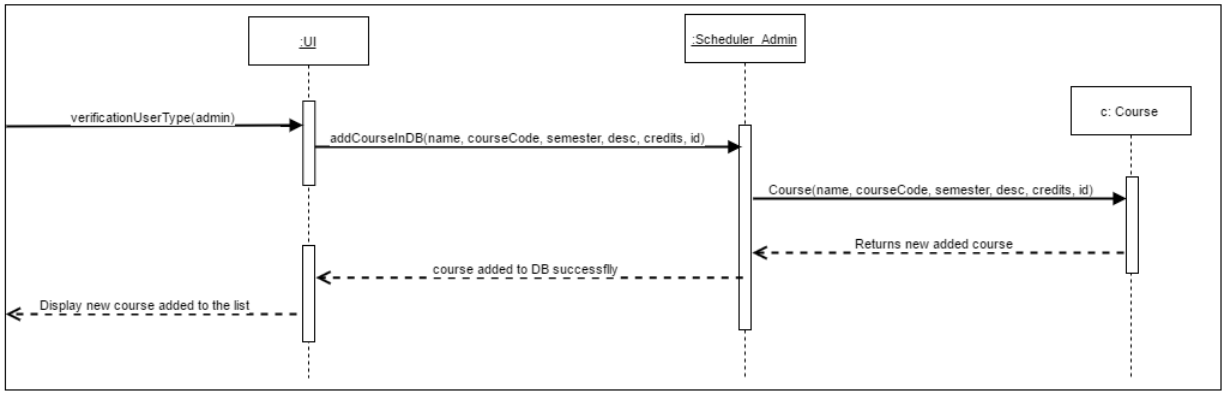
4.4.2 System Sequence Diagram



1. *addCourseInDB(name, courseCode, semester, desc, credits, id)* Operation Contracts

Name	Contract 4.1 Add course to database
Operation	addCourseInDB(JSON)
Cross Reference	UC23 Add Course to Program
Preconditions	-Admin is logged in. -Admin has prompted the *Add Course* section of the page.
Postconditions	-Instance of a course is created in the course database.

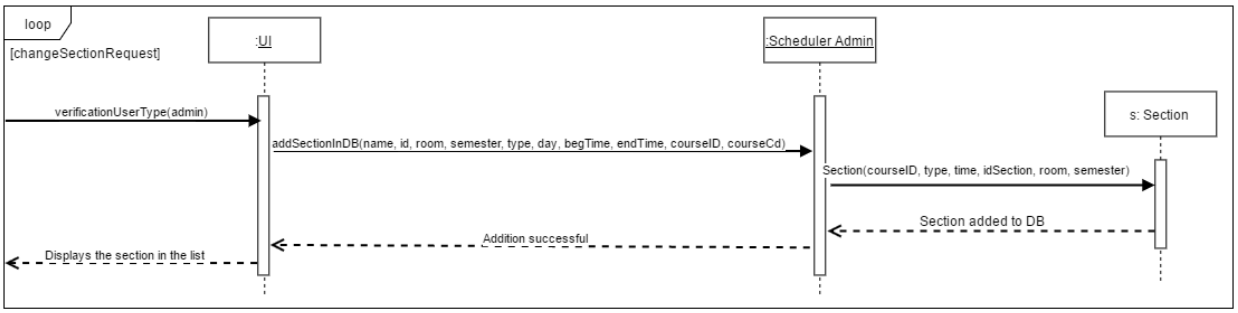
Sequence Diagram for Contract 4.1



2. *addSectionInDB(name, id, room, semester, type, day, begTime, endTime, courseID, courseCd)* Operation Contracts

Name	Contract 4.2 Add section in database
Operation	addSectionInDB(JSON)
Cross Reference	UC23 Add Course to Program
PreConditions	-User Logged in as administrator -Course already exists
Post Conditions	-New instance of section created into the section database.

Sequence Diagram for Contract 4.2



5. Estimation

Few corrections needed to made to the of deliverable 2, mainly due to the unexpected amount of work involved with the different designs.

Previously Incurred Costs:

	Deliverable 0	Deliverable 1
Total Hours	5	70
Cost Estimate ($25/hr)	$125	$1750

The cost estimate for deliverable 2 revealed itself to be slightly higher than previously forecasted as detailed in the table below.

Artifact		Estimated Cost in Hours	Final Cost in Hours
4+1 Architectural View	**Logical**	**10**	**10**
	Development		**3**
	Physical		**3**
	Process		**5**
	Scenarios		**1**
Subsystem Interface Specification		**25**	**20**
UML Class Diagram		**12**	**10**
Dynamic Design Scenario		**6**	**10**
Estimation		**7**	**5**
Rapid Prototyping Report		**22**	**25**

Testing	**15**	**15**
Risks	**4**	**4**
Total Hours	**101**	**111**
Cost Estimate ($25/hr)	**$2525**	**$2775**

The cost of the following tasks is not expected to be higher than what was estimated in the first deliverable. The total amount of hours calculated for the entire project was 352 hours, amounting to the estimated cost of $8800. Due to the 10 hours increase incurred during the realization of the second deliverable, the new total number of hours rises to 362hrs, which corresponds to a total project estimate of $9050.

6. Rapid Prototyping and Risk

6.1 Risk

Testing the functionality of a system during development contributes greatly in evaluating potential risks. And having an early understanding of the undertaken endeavor, helps in maximizing productivity by imposing certain benchmarks.

Framework and programming languages:

Despite the fact that almost all members in the team are proficient in programming, the use of a new and unfamiliar framework might pose a risk when it is time to implement compound and complicated functionalities. And while embarking in a learning experience during development was a collective decision and a calculated risk, the overhead on the more experienced could possibly become substantial.

Time constraints:

Our original cost estimate fell a little short as the project necessitated more effort than originally anticipated. It becomes demanding to compensate for the lack of time with effort as submission deadlines grow nearer.

Security:

The robustness of the system has alleviated most concerns pertaining to security; suffice to say that crashing the system during testing was barely made possible by a team member who became dedicated to overloading the system and over a long period of time. We do not perceive security as a high risk nevertheless it is always a very present factor in all design decisions.

Team interaction:

The risk that comes with a large pool of ideas is that it is may deter from a focused approach. While innovative ideas are brought forth because of the variety in a team, it is possible to spend production time debating the pros and cons of one view over another, which could be accentuated when multiple views are difficult to reconcile.

6.2 Front-End Work

All the major pages were completed for the prototype, except for the schedule page, which will be completed after the scheduling algorithm is in place.

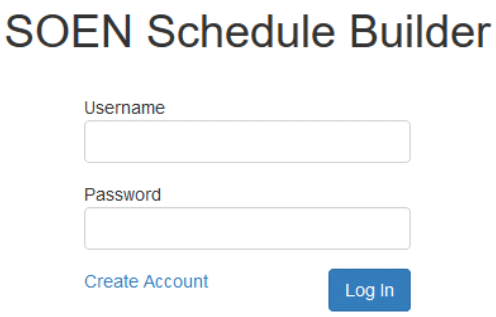


Figure 7: *The login page*

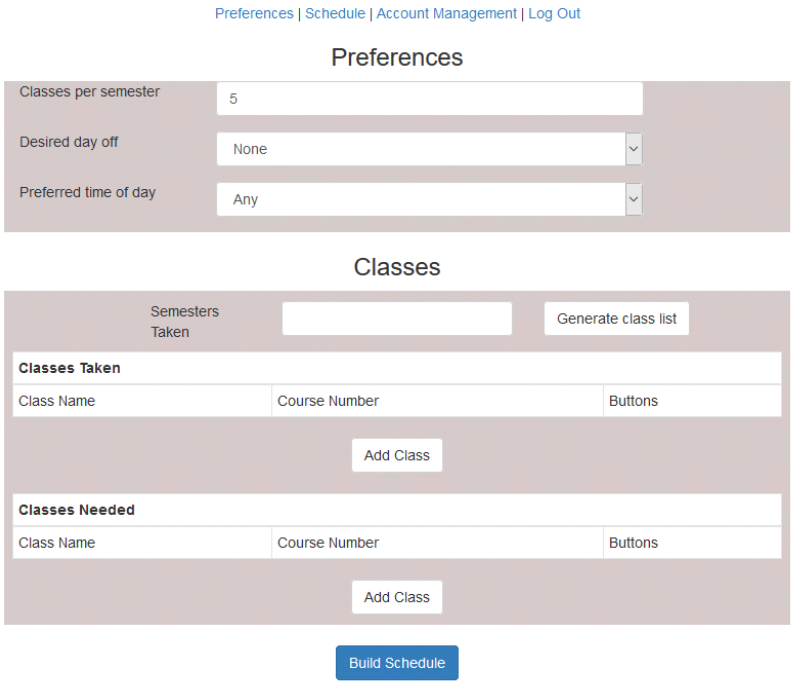


Figure 8: *The preferences/classes page*

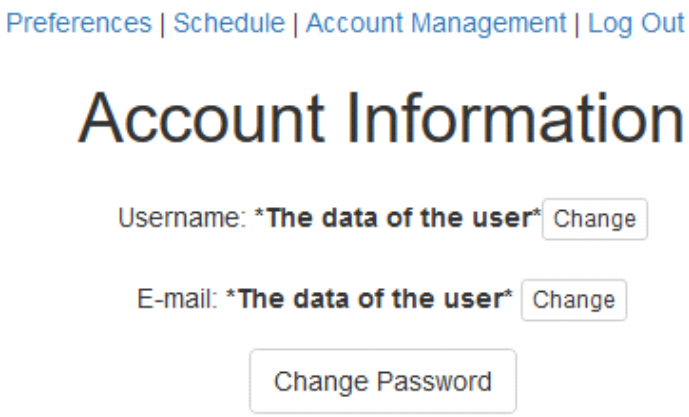


Figure 9: *The account page*

The front-end work progressed as expected without any scope changes. React continues to be a simple and elegant solution for the front-end that only gets complicated when it comes to communicating with the back-end.

6.3 Back-End Work

For the prototype, the plan was to get a working interface that interacted with the database. As planned, one of the first steps was to design and set up the database that was to be used. A document explaining how to create and populate the Mysql database was also shared with the team. The second step was to work with the front-end team to link the set of pages with the databases through server calls. The register and the log in modules were working. However, they were not using the Laravel framework. Incorporating the current pages in the MVC model and the Laravel folder architecture as well as using the features of the framework (such as routing) were the last step of the prototype implementation. As a confirmation, the application was shared with the team and implemented on their end to test the modules that were implemented.

6.3.1 Front-end and back-end communication issues

There were a lot of issues with the usage of React and Laravel. React was initially designed for use with NodeJS, where the Javascript can be rendered on the server, but this is not an option when our back-end is running with a PHP framework. Since we are using React generate the views (this is operation is done client-side) and Laravel to handle the architecture, this makes it complicated for Laravel to manage the views for instance. While the original prototype did prove React renders the pages flawlessly, we had not yet implemented them with the framework and its architecture. Our initial prototype was merely testing log-in authorization and registration. Once Laravel came into play, this complicated the project. These issues did not affect our design decisions.

6.4 Added technology

6.4.1 Twitter Typeahead

We have implemented Twitter's typeahead component to improve the user experience. It is a jQuery open source text component that provides auto-completion suggestions as the user types (much like the google search bar). We will be using it to help the user add their needed and taken courses. The list of courses given in the auto-complete will be retrieved from the server.

This will not only make a better user experience, but make it less likely for the user to input a course that doesn't exist by accident.

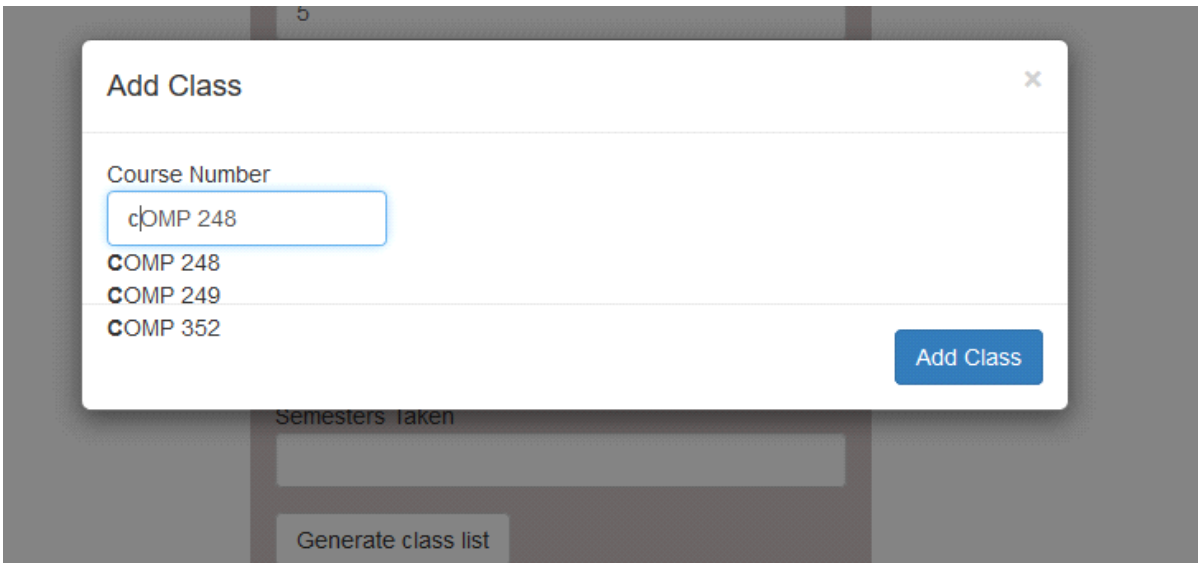


Figure 10: Twitter Typehead

Part III:
Testing & Delivery

1. Introduction

The present deliverable details the testing and delivery processes that were undertaken to the completion of the scheduling system. In its first part it deals with the testing coverage in which all test items along with the test cases that were applied on them will be listed, as well as an explanation as to why testing is required. Two testable units are identified. The first one is the authentication which includes the log in and registration units, and the second one is the "manage Preferences" unit. Both were selected because of their priority and relevance to the overall system. A list of test cases is included for each unit in addition to the code to the stubs and drivers used. Extra tests are performed to show how the system is to be used and what system reactions are to be expected.

Stress and Security testing follow to evaluate the robustness and security levels of the system. Those tests shed light on the potential extreme situations of system usage and on its resistance to security concerns such as SQL injection attacks.

The report extensively explains the installation process. A manual listing what is needed and how it is to be installed is made available, and it guarantees a successful and functional system upon completion of the provided steps. It also describes how the system is to be used along with all its available features.

The report concludes with a revised cost estimate that compares what was previously forecasted and what was actually incurred.

2. Testing Report

2.1 Test Coverage

The following section covers the testing phase of the project. Testing is a key concept when it comes to the delivery of a particular product. The reason being is that it allows for the workers to assess the early problems in the software and to ensure that the requirements of top priority are managed, since these functionalities may endanger the overall software. The section is divided into different test categories:

- Tested Items
- Untested Items
- Unit testing
- Requirement testing

Tested Items: Defined as the features that were already tested. It results into two possible outcomes: pass or fail.

Untested Items: Defined as the features that were not already tested due to a specific reason which results into not prioritizing that particular matter.

Unit Testing: Defined as testing a small part of the software, in other words, a small part of the code is evaluated to see if it works.

Requirements: Defined as the essential functionality that a user, student and admin must be able to accomplish with the software at hand.

2.1.1 Tested items

The two units which were tested on consist of the authentication unit and the preference unit.

Authentication: This was immediately tested since the requirement of signing up and logging forms the basis of the whole use of the software. This consists of ensuring that the user is successfully directed to his/her appropriate page. This item is the number one priority because this functionality allows access to the Scheduler.

Manage Preferences: This unit was tested because the integration of preferences demands to be of creative thinker when generating the schedule. This is of great importance because the system's requirement is to take into consideration the things that the student wants and testing solely on effectively containing, changing and deleting the preferences proves to be of big priority.

Stress and Security: This is an important key because the project, assuming it will be used by a certain client for a big academic institution, provides access to personal information. These tests are of importance since they provide a safety margin. Without protected and secure access, the software would suffer and, nonetheless the clients, since its functionality is overshadowed by not being secured and properly stressed.

Student & Admin		
Feature	**Use Case**	**Related Test Cases**
Login/Logout	**UC1, UC2**	**1.1, 1.2, 1.3, 1.4, 2.1**
Edit account information	**UC5**	**19.1, 19.2, 19.3, 19.4, 19.5, 19.6, 19.7, 19.8**

Admin		
Feature	**Use Case**	**Related Test Cases**
Add course to program	**UC13**	**23.1, 23.2, 23.3**
Delete course	**UC14**	**25.1, 25.2**
Edit course	**UC15**	**24.1, 24.2**
Add section	**UC16**	**26.1, 26.2, 26.3, 26.4**
Edit section	**UC17**	**27.1, 27.2**
Delete section	**UC18**	**28.1**

Student		
Feature	**Use Case**	**Related Test Cases**
Sign up	**UC3**	**4.1, 4.2, 4.3, 4.4**
Set preferences	**UC6**	**12.1, 12.2, 12.3 12.4**
Set needed courses	**UC7**	**9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7**
Delete needed courses	**UC8**	**10.1**
Set taken courses	**UC9**	**8.1, 8.2, 8.3, 8.4**
Delete taken courses	**UC10**	**11.1**
Generate schedule	**UC11**	**13.1, 13.2, 13.3**
View auto-generated list of taken courses	**UC12**	**7.1, 7.2, 7.3, 7.4**

2.1.2 Untested Items of Interest:

Remaining server calls: There are server calls for obtaining all courses, sections, and user information as well as calls for updating all of that information and calls for generating the schedule. These calls can be unit tested the same way the other calls were. They are important to test because the user interface relies on the server calls behaving as expected. Whether they succeed or fail, they must do it in the way the UI would expect.

The user interface: This was user tested, but it was not tested programmatically because there is no official React unit testing framework. There are some open source frameworks that we did not look too much into because they are very new and are neither robust nor heavily tested. Unit testing a UI is also much more complicated than testing a single function that handles data. If we were to unit test, we would use one of the open source React unit testing frameworks and edit it as needed. It is important to test the UI because this is the user facing portion of the application. If this does not perform as expected, it is immediately apparent to the user and does not look professional. User testing does not always pick up on all of the bugs the same way a unit test does, so unit testing the UI would thoroughly verify that it performs as expected.

Student & Admin	
Reset Password	Too complex
Confirm email	Too complex
Delete schedule	Not implemented
Change section	Does not add anything to software
View default schedule	Does not add anything
Save schedule	Not implemented
Print schedule	Does not add anything to software
View schedule	Not implemented

Unit	Tested?
Login/Sign up	Yes
Manage courses	No
Manage preferences	Yes
Manage account information	No
Manage course DB	No
Generate Schedule	No

2.2. TEST CASES

2.2.1 Unit Testing

For the unit testing we chose to test several important server calls. These are crucial because it is how the client side and server side of our application communicate. Without these working bug-free the application would not be able to do much.

All of the server calls interact with the database in some way, either to retrieve data or to update it. In order for testing to work consistently we needed a database that would be constant. For this we made a much smaller database purely for testing purposes. When running tests locally, this database will be used instead of the official one. That way we will know exactly what information is supposed to be stored in the database and can expect consistent results from our tests.

Laravel has an official unit testing framework, but it expects all of the mvc to be used. Since we replaced the view with React, this framework was not ideal. Instead we have our own testing script, coded in javascript that runs all of our tests. Neither the testing script nor the database will be included in the release version of our application, since they are only for testing.

For development we created a javascript object, realServerBridge, which we used to abstract all of the server calls. We used this to make the server calls when testing.

User class- login method Test:

Our login server call is as follows:

URL	Purpose	Type	Input	Returns
/login	Login verification	POST	{username:'', password:''}	{"success":"true","username": "JASONB","isAdmin":"true"}

It expects a JSON object including a username string and a password string. It will return a JSON object with a string holding the success state, the username, and a string saying whether or not the user is an admin. The test cases are as follows:

Description	Input	Expected output that was tested
Call is made with a username that exists in the database with the correct password for that user	{username: 'User', password:'password'}	success='true' username='User' isAdmin='false'
Call is made with a username that does not exist in the database (sending an empty username and password is treated the same way as this)	{username:'notauser', password:'password'}	success='false'
Call is made with existing user, but the wrong password	{username:'User', password:'notthepassword'}	success='false
Call is made with the admin's username and password	{username:'Admin', password:'password'}	success='true' username='Admin' isAdmin='true'

We tested by making the server calls and then verifying the output we received. No set up or tear down was needed to keep the test consistent.

Scheduler_Student Class- setPreferences method Test

Users can specify preferences for their schedule, such as a desired course load or a day of the week they want free. The server call to update the preferences for a user is as follows:

URL	Purpose	Type	Input	Returns
/editpreferences	Edit user's preferences	POST	{username:'',cload:",dayoff:",preftime:"}}	{"success":"true","username":"JasonB","courseload":"5","dayoff":"Monday","preftime":"Mornings"}

It expects a JSON object including strings for the username, course load, day off, and preferred time and will return a JSON object including the success state, as well as the info that was sent. Our client side of our project can handle invalid data entered for preferences by replacing it with default information, so for testing we were not concerned with that. The test cases are as follows:

Description	Input	Expected result
Call is made with a username that exists	{username:'Jason', cload:'5', dayoff:'Monday', preftime:'Mornings'}	Data updated in the database
Call is made with a username that is not in the database	{username:'notauser', cload:'5', dayoff:'Monday', preftime:'Mornings'}	Output: success='false'

For the first test case we had to also use the server call to retrieve the preferences in order to check that the information had actually been updated.

The methods for both cases expect a username in the cookies, so this had to be done as set up for the cases. This had to be removed for the tear down and the preferences had to be set back to blank values, so the tests could be consistent.

Student class- signUp method Test:

Registration was important to test. If this didn't work it could cause security issues or could make it so that people could not even sign up for the site and could never access it. The server call is as follows:

URL	Purpose	Type	Input	Returns
/register	Register user	POST	{username:'', email:'', password:''}	{"success":"false","username":"SprinkKing","error":"usernametakenalready"}

It expects a JSON object containing strings for the username, e-mail, and password and will return a JSON object containing strings for the success state, the username, and an error message. The success will be false only if a user with that username already exists. The error will be blank if the call was successful.

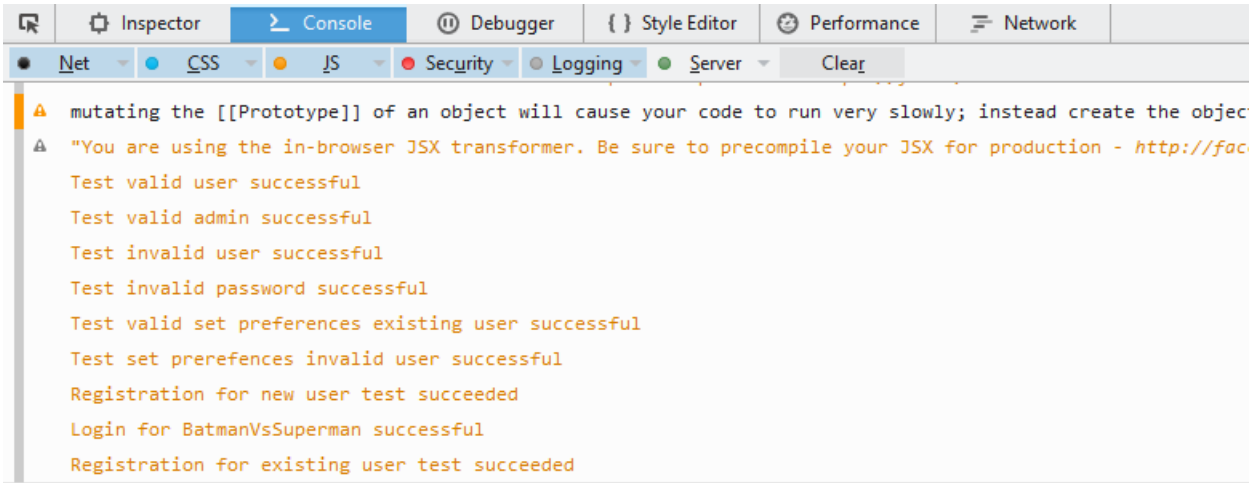
The test cases are as follows:

Description	Input	Expected output that was tested
Call is made with a new username	{username:'BatmanVsSuperman', email:'justiceleague@gmail.com', password:'password'}	success='true' Also the following login call should succeed, showing that the user was added in the database
Call is made with an existing username	{username:'Jason', email:'jason@hotmail.com', password:'password'}	success=false

No set ups were needed. For tear down we had to remove the BatmanVsSuperman user from the database, so the test would work again next time.

Running the tests

All of the tests succeeded. Here was the console after they ran:



Unit Testing Code

For any server calls that expect information back from the server, a callback method must be sent to the server bridge object. This is due to the asynchronous nature of AJAX calls. When the data is finally received, it will be sent as the only argument to that method and then it can be tested. This is why we send functions as an argument for the server bridge methods.

```
var serverBridge=realServerBridge;
//LOGIN TESTS
var response = serverBridge.login('User', 'password', function(data) {
    if(data.success=='true'&&data.username=='User'&&data.isAdmin=='false') {
        console.log('Test valid user successful');
    }
    else {
        console.log('Test valid user failed');
    }
});
response = serverBridge.login('Admin', 'password', function(data) {
    if(data.success=='true'&&data.username=='Admin'&&data.isAdmin=='true') {
        console.log('Test valid admin successful');
    }
    else {
        console.log('Test valid admin failed');
    }
});
response = serverBridge.login('notauser', 'password', function(data) {
    if(data.success=='false') {
        console.log('Test invalid user successful');
    }
    else {
        console.log('Test invalid user failed');
    }
});
response = serverBridge.login('User', 'notthepassword', function(data) {
    if(data.success=='false') {
        console.log('Test invalid password successful');
    }
    else {
        console.log('Test invalid password failed');
    }
});
```

```
//SET PREFERENCES TESTS
//Set up
var testPref = {"courseLoad":"5", "day":"Monday", "time":"Mornings"};
var response = serverBridge.editPreferences(testPref);
var cookie = cookieManager.addCookie("username", "Jason", 1);
//Test
serverBridge.getUserPrefs(function(data){

    if (data.courseload == testPref.courseLoad && data.dayoff == testPref.day && data.preferredTime == testPref.time){
        console.log('Test valid set preferences existing user successful');
    }
    else{
        console.log('Test valid set preferences existing user failed');
    }

});
//Tear down
serverBridge.editPreferences({"courseLoad": "", "day" : "", "time" : ""});
cookieManager.removeCookie("username");

//Set up
cookieManager.addCookie('username', 'notauser', 1);
//Test
var response = serverBridge.editPreferences(testPref, function(data) {
    if(data.success=='false') {
        console.log('Test set prerefences invalid user successful');
    }
    else {
        console.log('Test set prerefences invalid user failed');
    }
});
//Tear down
cookieManager.removeCookie("username");
```

```
//REGISTRATION TESTS
response = serverBridge.register('BatmanVsSuperman', 'justiceleague@gmail.com', 'password', function(data){
    if(data.success == "true"){
        console.log("Registration for new user test succeeded");
    }
    else{
        console.log("Registration for new user test failed");
    }
});
var response = serverBridge.login('BatmanVsSuperman', 'password', function(data) {
    if(data.success=='true'&&data.username=='User'&&data.isAdmin=='false') {
        console.log('Login for BatmanVsSuperman successful');
    }
    else {
        console.log('Login for BatmanVsSuperman failed');
    }
});

response = serverBridge.register('Jason', 'jason@hotmail.com', 'password', function(data){
    if(data.success == "false"){
        console.log("Registration for existing user test succeeded");
    }
    else{
        console.log("Registration for existing user test failed!");
    }
});
```

2.2.2 Requirements testing

The tables below show the result of black box testing for the functional requirement of the schedule generator system. Cells highlighted in green indicate a passed test while the ones in red indicates a failure. Failures are currently being investigated to be fixed for the final deliverable v1.2
Cases that were scoped out and therefore not tested are indicated in the last table

UC1	Login					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
1.1	**User inputs valid username and password (matching inputs)**	**Username: “student1” Password: “a1b3c3d4”**	**User is redirected to student main page**	**PASS**		
1.2	**User inputs a valid password and admin username**	**Username: “Jason” Password: “Password1”**	**User is redirected to admin main page**	**PASS**		
1.3	**User inputs invalid username and/or password**	**Username: “student1” Password: “abc”**	**"Incorrect password and/or username"**	**PASS**		
1.4	**User tries to login with one of the fields empty**	**Username: “student1” Password:**	**"Incorrect password and/or username"**	**PASS**		

UC2	Logout					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
2.1	**User clicks on "logout" button**	**N/A**	**User is redirected to student main page**	**PASS**		

UC4	Sign up					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
4.1	**User inputs valid username, email and password.**	**Username: “student1” Password: “a1b2c3d4” Email: student1@gmail.com**	**Account is created and the user is redirected to the preferences page in their new account**	**PASS**		
4.2	**User inputs a valid password and username, but an invalid email.**	**Username: “student1” Password: “a1b2c3d4” Email: student1**	**Error Message displayed asking user to enter a valid email**	**PASS**	**1**	**fixed for v1.2**
4.3	**User inputs a valid password and email, but an invalid username.**	**Username: “a” Password: “a1b2c3d4” Email: student1@gmail.com**	**Error Message displayed asking user to input a username with 4-6 characters**	**PASS**		
4.4	**User inputs "admin" as a username, with a valid password and email.**	**Username: “admin” Password: “a1b2c3d4” Email: student1@gmail.com**	**Error message appears notifying user that the username entered is invalid**	**PASS**		

UC7	View auto-generated list of taken courses					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
7.1	**User inputs a valid number of semesters taken (between 1 and 7) and selects “generate course list”**	**-Number of semesters: 1 -Select “generate course list”**	**A pop-up asking for confirmation to generate the course list is displayed.**	**PASS**		
7.2	**User selects “Yes” in**	**-Select “Yes”**	**The pop-up**	**PASS**		

	the pop-up prompting for confirmation to generate the course list		**closes and courses are auto-generated and added to the list of taken courses**			
7.3	**User selects "No" in the pop-up prompting for confirmation to generate the course list**	**-Select "No"**	**The pop-up closes and the input to "semesters taken" is blank.**	**PASS**		
7.4	**User inputs an invalid number of semesters taken (not between 1 and 7) selects "generate course list"**	**Number of semesters: 0**	**An error message is displayed and the "generate course list" option doesn't do anything.**	**PASS**		

UC8	**Set taken courses**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
8.1	**User selects the "add class" option for the taken courses list**	**Select "Add class"**	**User is presented with a pop-up box prompting for an input of a course number or an input of a course name**	**PASS**		
8.2	**User inputs a valid (present in the database) course number and selects "add class"**	**Course number: SOEN 341** **Select "Add class"**	**The course name corresponding to the course number is automatically generated. The pop-up box closes, and the**	**PASS**		

			inputted course is added to the list of taken courses			
8.3	**User inputs a valid (present in the database) course name and selects "add class"**	**Course name: Software Process** **Select "Add class"**	**The course number corresponding to the course name is automatically generated. The pop-up box closes, and the inputted course is added to the list of taken courses**	**PASS**	**2**	**Fixed v1.1**
8.4	**User inputs a valid (present in the database,) course number and/or course name already in the list of taken courses and selects "add class"**	**Course name: SOEN 341** **Select "Add class"**	**An error message is displayed and selecting the "add class" option doesn't do anything.**	**PASS**		

UC9	**Set needed courses**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
9.1	**User selects the "add class" option for the needed courses list**	**Select "Add class"**	**User is presented with a pop-up box prompting for an input of a course number or an input of a course name**	**PASS**		
9.2	**User inputs a valid (present in the**	**Course number: SOEN 341**	**The course name corresponding to**	**PASS**	**3**	**Fixed v1.1**

	database, does not conflict with another course time and the prerequisites are met) course number and selects "add class"	**Select "add class"**	**the course number is automatically generated. The pop-up box closes, and the inputted course is added to the list of needed courses**			
9.3	**User inputs a valid (present in the database, does not conflict with another course time and the prerequisites are met) course name and selects "add class"**	**Course number: Software Process** **Select "add class"**	**The course number corresponding to the course name is automatically generated. The pop-up box closes, and the inputted course is added to the list of needed courses**	**PASS**		
9.4	**User inputs a course number or course name that conflicts with the time of another course in the list of needed courses and selects "add class"**	**Course number: COMP 249** **Select "add class"**	**An error message is displayed and selecting the "add class" option doesn't do anything.**	**PASS**	**4**	**Fixed v1.2**
9.5	**User inputs a course number or course name for which the prerequisites are not met and selects "add class"**	**Course number: SOEN 422** **Select "add class"**	**An error message is displayed and selecting the "add class" option doesn't do anything.**	**PASS**	**5**	**Fixed v1.2**
9.6	**User inputs an invalid (not present in the database) course**	**Course number: SOSO 123**	**An error message is displayed and**	**PASS**		

	number and/or course name and selects “add class”	Select “add class”	selecting the “add class” option doesn’t do anything.			
9.7	User inputs a valid course number and/or course name already in the list of needed courses and selects “add class”	Course number: SOEN 341 Select “add class”	Course is added to the list of needed classes.	PASS		Fixed v1.1

UC10	Delete needed course					
ID	Description	Input	Expected Output	Result	Bug ID	Comments
10.1	User selects the delete icon of a course in the list of needed courses	Select the delete icon	The course is removed from the list of needed courses	PASS		

UC11	Delete taken course					
ID	Description	Input	Expected Output	Result	Bug ID	Comments
11.1	User selects the delete icon of a course in the list of taken courses	Select the delete icon	The course is removed from the list of taken courses	PASS		

UC12	Set Preferences					
ID	Description	Input	Expected Output	Result	Bug	Comments

					ID	
12.1	User selects his desired day off to be a specific day (Monday, Tuesday, Wednesday, Thursday, or Friday) from the drop down menu	Select “Monday”	The selected day is displayed as the desired day off	PASS		
12.2	User selects his desired day off to be “none” from the drop down menu	Select “none”	“none” is displayed as the desired day off	PASS		
12.3	User selects his preferred time of the day to be either “mornings”, afternoons” or “evenings” from the drop down menu	Select “mornings”	The selected preferred time of the day is displayed as the preferred time of the day	PASS		
12.4	User selects his preferred time of the day to be “Any” from the drop down menu	Select “any”	“Any” is displayed as the preferred time of the day	PASS		

UC13	Generate Schedule					
ID	Description	Input	Expected Output	Result	Bug ID	Comments
13.1	User selects the “build schedule” option	Select “build schedule”	User is directed to the schedule page, and multiple schedules are generated according to the preferences and needed courses.	PASS	6	Fixed v1.2

13.2	User selects an arrow on the schedule page to see a different generated schedule	Select an arrow	Another generated schedule is displayed on the schedule page	PASS	7	Fixed v1.1
13.3	User selects “select this schedule” from the schedule page	Select “select this schedule”	User is directed to the sequence page. The selected schedule is displayed along with the user’s course sequence	PASS	8	Fixed v1.2

UC19	Edit Account Information					
ID	Description	Input	Expected Output	Result	Bug ID	Comments
19.1	User changes the username by inputting a new valid username	Username: “studnetx”	Name is changed in the account management page	PASS		
19.2	User tries to change the username by inputting the same current user name	UsernameOld: “studnet1” UsernameNew: “studnet1”	Error Message notifying the user that the entered user name is the same as the old one	PASS	9	Fixed v1.1
19.3	User changes the username with a new user name that is too short (less than 4 characters)	UsernameOld: “studnet1” UsernameNew: “s1”	Error message notifying the user that the username is too short	PASS		
19.4	Change current email with a new valid email	OldEmail: s1@gmail.com NewEmails:s2@gmail.com	New email appears on the user account information	PASS		

19.5	**Change current email with one that has invalid format**	**New Email: s123**	**Error message appears asking the user to input a valid email address.**	**PASS**	**10**	**Fixed for V1.2**
19.6	**Change Password by entering the correct current password and new password with 8-16 characters and retype the new password correctly**	**oldPassword:"a1b2c34d" NewPasword: "123abc—"**	**Password is changed**	**PASS**		
19.7	**Change Password by inputting the same current password as the new one**	**oldPassword:"a1b2c34d" NewPasword: a1b2c34d"**	**Error message appears notifying user to enter a new password**	**PASS**		
19.8	**Change password by entering the correct current one , entering a valid new one but mismatch in retyping the new one**	**oldPassword:"a1b2c34d" Retype: 1249gjt NewPasword: a1b2c5jug**	**Error message notifying the user that the passwords do not match**	**PASS**		

UC23	Add Course to Program					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**

23.1	**Add a new course to list of courses offered**	**Soen549**	**Course is added to the list of courses displayed**	**PASS**		
23.2	**Add a course that already exists in the list of courses**	**Soen341**	**Error message appears and Course List is not changed**	**PASS**		

UC24	**Edit Course**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
24.1	**Edit Course Description**	**Semester: Winter**	**Course description is changed**	**PASS**		

UC25	**Delete Course**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
25.1	**Delete an existing course from course list**	**Click on delete icon**	**Course is removed from the course list**	**PASS**		
25.2	**Delete a course with ID not in the list**	**Click on delete icon**	**Error message appears and Course List is not changed**	**PASS**		

UC26	**Add Section**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**

26.1	**Create new section with valid inputs**	**Name:UI, Location: H-563 Day:MW Time:16:15-17:30,Course Name:SOEN341, Semester: W2016**	**New Section is added to the list of sections of the required course**	**PASS**		
26.2	**Create Section without inputting all the data**	**H-563 Day:MW**	**Error message appears notifying the user to complete the data required**	**PASS**		
26.3	**Add Section with existing section name**	**Name SOeEN341**	**Error Message appears notifying the admin that section already exists**	**PASS**		
26.4	**Add section with the same date, time and location of an existing section**	**Name:UF, Location: H-563 Day:MW Time:16:15-17:30,Course Name:SOEN341, Semester: W2016**	**Error Message appears to notify admin that the section cannot be created**	**PASS**		

UC27	**Edit Section**					
ID	**Description**	**Input**	**Expected Output**	**Result**	**Bug ID**	**Comments**
27.1	**Edit the time of an existing section (ex: SOEN 341, Time:18:00-19:00)**	**Time 15:00-17:00**	**Section time is changed; new time appears in the section information**	**PASS**	**10**	**fixed for v1.2**
27.2	**Edit the location of an existing section**	**H -511**	**Section location is changed; new location appears in the section**	**PASS**	**11**	**fixed for v1.2**

			information			

UC28	Delete Section					
ID	Description	Input	Expected Output	Result	Bug ID	Comments
28.1	Delete an existing section from the list of sections of a course	Click on delete button	Section is removed from the list	PASS	12	Fixed v1.1

Requirements Scoped out and therefore not tested:

Use Case Number	Description	Test Result
UC3	Reset Password	N/A
UC 5	View Default Schedule without creating an account	N/A
UC 6	Modify a generated schedule	N/A
UC 14	Save a generated schedule	N/A
UC 15	Delete a saved schedule	N/A
UC 16	Print a generated schedule	N/A
UC 17	View previously saved schedule	N/A
UC 18	Change a section in the schedule	N/A
UC20	Drop Course	N/A
UC21	Modify Preferences	N/A
UC22	Update taken course	N/A

2.2.3 Stress Testing

The system described throughout the scheduler's deliverables is a system which, by the very nature of its usage, will experience varying degrees of client traffic: There will be varying volume of users logging in to the system during the day versus the middle of the night or the dates of class registration versus other days between or in the middle of any given semester. This volume of users will tend to be condensed during the same higher traffic times (i.e. during the day of course registration) and more spread out during other times. These periods of heavy load on the system can be simulated and exaggerated beyond normal operation the system with a script efficiency tool. The resulting stress test can be applied to a system in order to determine the robustness, response and availability of a correct behavior for any system that may experience variable operation. The generic test is a black box test that can be applied to any system algorithm. However, the particular programming language and methods are a good indication

For the purpose of this deliverable, Apache Benchmark (ab) along with a PHP code in order console in order to repeat multiple htpwas used as a dynamic software verification and validation method in order to produce repeatable testing with quantitative request times (connect, processing, waiting, and total), transferred bytes (total, HTML, and document) for a given number of requests to be performed and the number of requests that occur simultaneously. A greater number of requests may represent simply a longer period of time (a week vs an hour). However, if a greater number of these requests occur at the same time (concurrency) than what is normal, then we are implementing a stress test in which we can observe several of the stress related defects.

The dynamic apachebenchmark test the PHP code and the server by attributing a number –n of requests and a value –c for the amount of simultaneous requests to be performed on every component of the server. This test however simulates a single device sending requests; in order to simulate many more devices, each with their own number set of requests, a PHP code was added to the apachebenchmark tool in order to multitask a number –r of url request repetition performed by each client and their value –c for the list of simultaneous clients.

Thus we have –c concurrent clients requesting the url's –r times each making each their –n requests with –c multiples.

This test is much more likely to lead to a failure of the code and an incident once that code runs causing an incident that may or may not be handled because of the larger number of simultaneous and total requests.

The PHP classes are: ezab and abrunner

```
class eZAB
{
   static $version = '0.3-dev';
   static $defaults = array(
      // 'real' options
      /// How much troubleshooting info to print. <, 3 and above prints response codes (404, 200, etc.), 2
          /// and above prints warnings and info."
      /// Real life testing seem to tell a different story though...
      'verbosity' => 1, // -v verbosity
      'children' => 2, // -c concurrency  Number of multiple requests to make
      'tries' => 10, // -n requests     Number of requests to perform
      'timeout' => 0, // -t timelimit    Seconds to max. wait for responses
      'auth' => false,
      'proxy' => false,
      'proxyauth' => false,
      'target' => '',
      'keepalive' => false,
      'head' => false,
      'interface' => '',
      'respencoding' => false,
      'httpversion' => CURL_HTTP_VERSION_NONE,
      'cookies' => array(),
      'skippercentiles' => false,
      'extraheaders' => array(),

      // 'internal' options
      'childnr' => false,
      'parentid' => false,
      // the actual script path (self)
      'self' => __FILE__,
      'php' => 'php',
      'outputformat' => 'text',
      'haltonerrors' => true,
      'command' => 'runparent' // allowed: 'helpmsg', 'versionmsg', 'runparent', 'runchild'
   );


class ABRunner
{
   static $version = '0.1-dev';
   static $defaults = array(
      // 'real' options
      'label' => '',
      'server' => 'http://localhost', // Server hostname (the prefix for urls below).
      'urls' => 'index.php', // List of urls to test. Use double quotes around, separate them with spaces
```

```
    'urlsfile' => '',
    'repetitions' => 1000, // The number of times each client requests each url
    'concurrencies' => '100 10', // List of concurrent clients to use
    'dognuplot' => false,
    'doaggregategraph' => false,
    'ab' => 'ab',
    'summary_file' => 'summary.txt',
    'output_dir' => 'test_logs',
    'sleep' => 1,

    // 'internal' options
    'verbosity' => 4,
    'self' => __FILE__,
    'outputformat' => 'text',
    'haltonerrors' => true,
    'command' => 'runtests',
    'abopts' => array()
);
```

Failure to meet response time requirements.

The System cannot be evaluated without completing a quantitative process to measure the response. This shall follow with more explanation.

Failure to run using particular configurations of hardware, operating systems and external libraries.

-libraries, and proper server are a critical issue in the installation and proper functioning of the Laravel framework. The absence of key libraries, PHP 5.4, PHP composer, Apache and MySQL within a proper server database are critical to the functioning of the system. The system will simply not exhibit the functionality given by the library –without warning.

Failure to gracefully handle resource shortage.

The Laravel framework implemented with a Heroku engine has the ability allow the database manager to scale in and out as well as automatic scaling to accommodate a varying number of clients in order to avoid downtime.

Failure to make resources available when they are no longer required.

A failure in the responsiveness of the SQL database can result in a cascade of failed requests and may result in a failure of the apache software and reboot incident –this is due to No code limiting the frequency of requests by any one client once a failed request has been observed.

Failure to fully recover from its own failure state or that of a related system.

And inappropriate SQL query can result into a failure of apache's ability to respond to incoming requests which could overload incident of the system.

A failure in the PHP Laravel or composer plugins can result in a failure to interpret the code other than a string.

Testing with ApacheBenchmark:

Single instances of a client requesting a specific number –n of times, of which –c are concurrent.

In the following tables, there are 4 different instances of these simple stress tests:

First the concurrency level was changed:

Server Hostname	Document Length (bytes)	Server Software
schedule-heroku.herokuapp.com	2 036	Apache

Concurrency level	Time for tests (ms)	Complete Requests	Failed Requests
100	15 807	500	0

Total transferred (bytes)	HTML transferred (bytes)	Transfer rate (Kbytes/sec)
1 104 000	1 018 000	68.21

Connection Times (ms)				
	min	mean	median	max
Connect	22	31	30	815
Processing	60	2663	2937	3010
waiting	54	1479	1473	2999
total	92	2694	2967	3784

Time to a percent of completion (ms)		
50	75	90
2 967	3 008	3 034

Server Hostname	Document Length (bytes)	Server Software
schedule-heroku.herokuapp.com	2 036	Apache

Concurrency level	Time for tests (ms)	Complete Requests	Failed Requests
500	17 995	500	0

Total transferred (bytes)	HTML transferred (bytes)	Transfer rate (Kbytes/sec)
1 104 000	1 018 000	59.91

Connection Times (ms)				
	min	mean	median	max
Connect	24	36	30	3030
Processing	50	10 382	10 412	17 918
waiting	50	10 374	10 404	17 915
total	80	10 418	10 444	17 949

Time to a percent of completion (ms)		
50	75	90
10 444	14 218	16 511

And Secondly the Complete requests was changed more drastically:

Server Hostname	Document Length (bytes)	Server Software
schedule-heroku.herokuapp.com	2 036	Apache

Concurrency level	Time for tests (ms)	Complete Requests	Failed Requests
20 000	303 427	100 000	0

Total transferred (bytes)	HTML transferred (bytes)	Transfer rate (Kbytes/sec)
220 800 000	203 600 000	68.06

Connection Times (ms)				
	min	mean	median	max
Connect	20	32	30	3 036
Processing	4 202	570 449	626 140	663 901
waiting	1 319	317 062	317 014	651 050
total	4 234	570 481	626 170	663 931

Time to a percent of completion (ms)		
50	75	90
626 170	635 745	657 250

Reduced request by a factor of 10.

Server Hostname	Document Length (bytes)	Server Software
schedule-heroku.herokuapp.com	2 036	Apache

Concurrency level	Time for tests (ms)	Complete Requests	Failed Requests
10 000	313 216	10 000	0

Total transferred (bytes)	HTML transferred (bytes)	Transfer rate (Kbytes/sec)
2 204 000	2 036 000	68.84

Connection Times (ms)				
	min	mean	median	max
Connect	20	31	30	3 032
Processing	1 1142	159 379	161 661	312 704
waiting	355	158 748	161 003	312 347
total	1 174	159 410	161 689	312 738

Time to a percent of completion (ms)		
50	75	90
161 689	237 593	312 738

These tables only represent one instance of a client requestings –they would be variable depending on the server current capacity and load. These tests give very little indication as to how the system could be pushed past it's limits.

In order to attempt this true stress testing, the PHP classes that multiply the number of imaginary clients and their requests were used to get and average responsiveness. The variables, -n, -r and –c were set in order to recreate 1000 students 10 or 1 at a time, each with 100 requests that are made 1 at a time.

It Should be noted that the server host was changed to http;//localhost:8000 –which reduced the connect times to virtually 0ms.

Start Time: Wed, 06 Apr 2016 10:55:38 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000
Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"
This is ABRunner, Version 0.1-dev

Start Time: Wed, 06 Apr 2016 12:36:22 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000
Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"
This is ABRunner, Version 0.1-dev

Start Time: Wed, 06 Apr 2016 12:37:45 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000
Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 10000
Command: ab -n 10000 -c 10 "http://localhost:8000/index.php"
This is ABRunner, Version 0.1-dev

Start Time: Wed, 06 Apr 2016 12:44:17 +0200

Testing http://localhost:8000/index.php, concurrency: 100, iterations: 100000
Command: ab -n 100000 -c 100 "http://localhost:8000/index.php"
This is ABRunner, Version 0.1-dev

Start Time: Wed, 06 Apr 2016 12:47:04 +0200

Testing http://localhost:8000/index.php, concurrency: 1, iterations: 1000
Command: ab -n 1000 -c 1 "http://localhost:8000/index.php"

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 10000

End Time: Wed, 06 Apr 2016 13:02:12 +0200

This is a total time of 2: 07: 26 for 3.2 Million requests: an average Of 419 requests every second. For the purpose of this large test the more detailed time keeping was turned off in order to save resources.

A shorter test was then used with no concurrency –this results in fewer requests per second, however a much smaller time to do each request.

Start Time: Wed, 06 Apr 2016 13:22:37 +0200

Testing http://localhost:8000/index.php, concurrency: 1, iterations: 100
Command: ab -n 100 -c 1 "http://localhost:8000/index.php"
Requests per second: 12.03 [#/sec] (mean)
Time per request: 83.128 [ms] (mean)
Failed requests: 0

Testing http://localhost:8000/index.php, concurrency: 10, iterations: 1000
Command: ab -n 1000 -c 10 "http://localhost:8000/index.php"

End Time: Wed, 06 Apr 2016 13:24:13 +0200

These tests give a brief view into the resource done by the server in order to give the best responsiveness in each situation.

2.2.4 Security Testing

SQL and HTML injection can be used to verify the security of a system and its capacity to leak important system information about the structure of the object oriented HTML/PHP or of the database schema; such as the exact HTML/PHP structure, Files, Extensions, Updates or even Database structure and contents. This is clearly an outcome to be avoided: thus a Laravel PHP framework was implemented in order to prevent eventual security breaches –especially by injection. The higher level abstraction of the frameworks helps to achieve this.

If this nikto is directly implemented into the server location, then many information is readily available:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>perl nikto.pl -h https://schedule-heroku.herokuapp.com/
Can't open perl script "nikto.pl": No such file or directory

C:\WINDOWS\system32>cd\Users\Nicolas\Documents\GitHub\nikto\program

C:\Users\Nicolas\Documents\GitHub\nikto\program>perl nikto.pl -h https://schedul
e-heroku.herokuapp.com/
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          23.23.231.101
+ Target Hostname:    schedule-heroku.herokuapp.com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /C=US/ST=California/L=San Francisco/O=Heroku, Inc./
CN=*.herokuapp.com
                   Ciphers:  ECDHE-RSA-AES128-GCM-SHA256
                   Issuer:   /C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCer
t SHA2 High Assurance Server CA
+ Start Time:         2016-04-05 12:46:54 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache
+ Retrieved via header: 1.1 vegur
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user a
gent to protect against some forms of XSS
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined
.
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type
+ Server banner has changed from 'Apache' to 'Cowboy' which may suggest a WAF. l
oad balancer or proxy is in place
+ Server leaks inodes via ETags, header found with file /robots.txt, fields: 0x1
8 0x52fb5ba3254c0
+ Server is using a wildcard certificate: *.herokuapp.com
+ Allowed HTTP Methods: GET, HEAD
+ OSVDB-3092: /web.config: ASP config file is accessible.
```

However, if the system is properly uploaded and the nikto is ran from outside the system, then no web server is detected at all.

3. System Delivery

3.1 Installation Manual

This installation manual will explain to an administrator how to install the scheduler system on a local hosting server. It is also possible to do it on a shared server, meaning the website would be accessible via the internet. However, this requires two things: a domain name and a shared server capable of running Laravel, such as https://www.fortrabbit.com/. Both cost a certain amount of money. However, the installation manual will only show how to install the project locally on the computer. The following installation is performed on Windows 10, but it is also possible to do it on other OS, with slightly modifications.

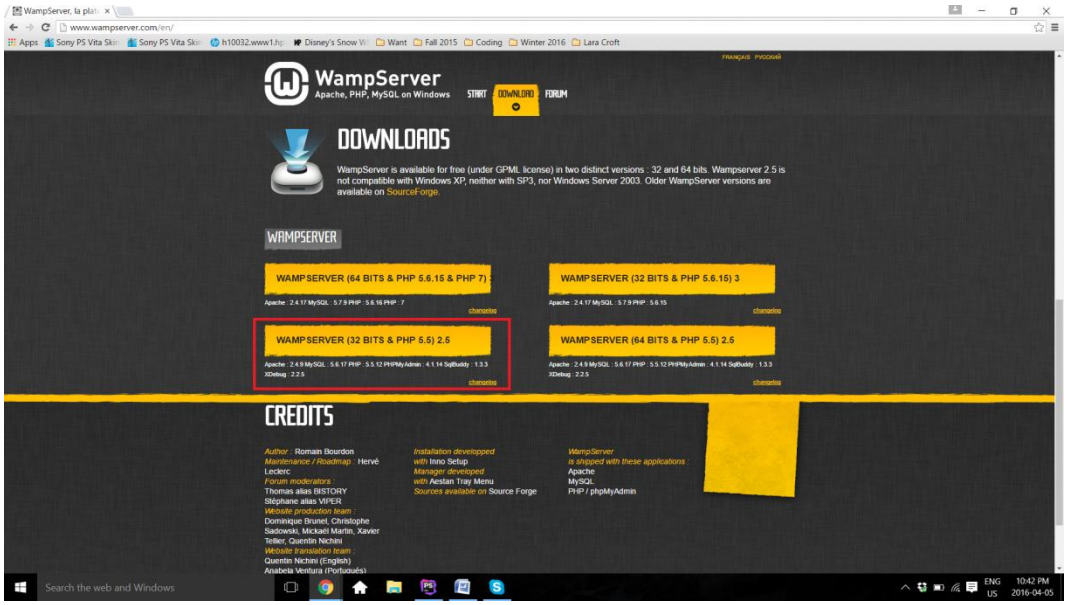
4 software are mainly required, with the scheduler project:

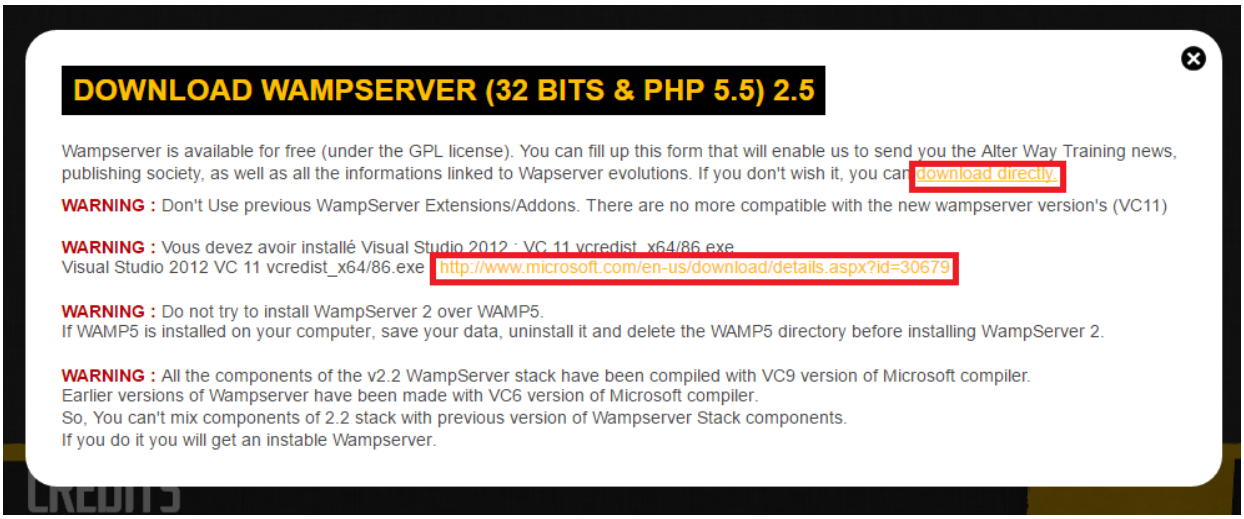
- WampServer
- Visual Studio
- Composer
- MySQL
- Scheduler Project from GitHub

Step 1: WampServer Installation

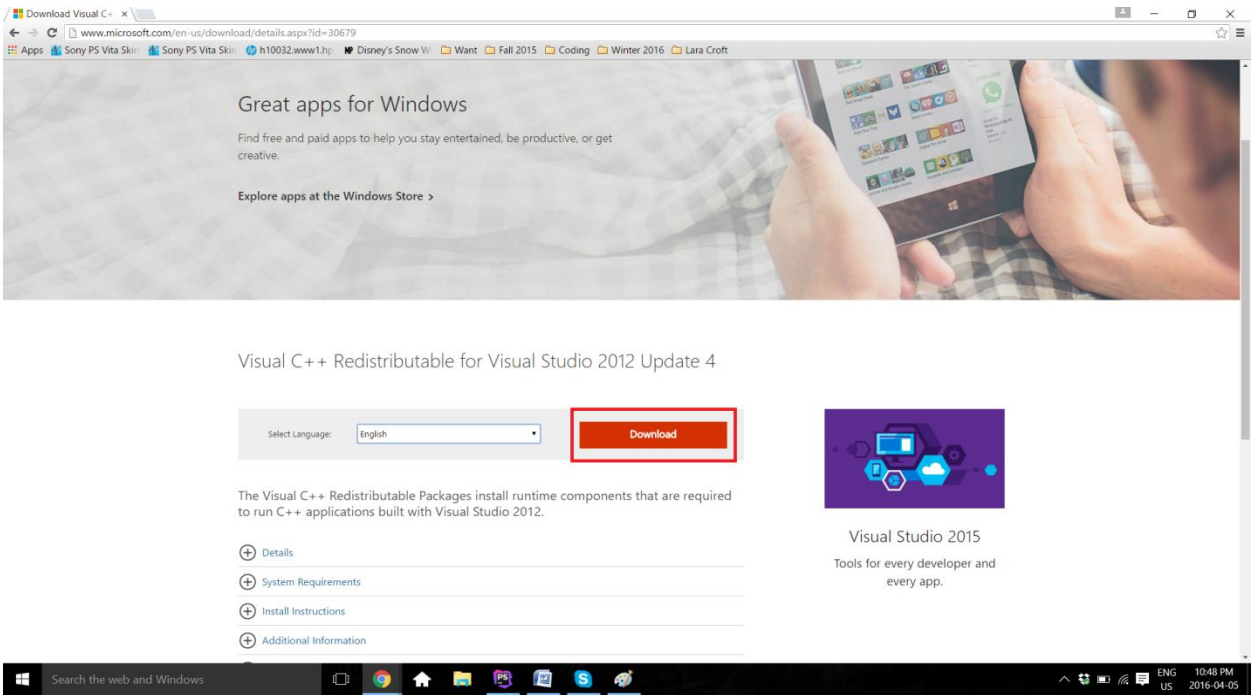
The first step consist of downloading and installing WampServer. WampServer contains all the needed softwares to run a PHP program on the local host: PHP, Apache and MySQL. It is recommended to download WampServer 2.5, containing especially Apache 2.4.9, MySQL 5.6.17, PHP 5.5.12. The scheduler has been coded in PHP 5.5, therefore it is preferred to use that version in case syntax modifications have been done in the newer versions of PHP 7. Furthermore, this package works both on 32-bit and 64-bit computers. The website to this download is http://www.wampserver.com/en/.

Note: Another alternative to WampServer is XAMPP. The only difference between both is that XAMPP is offered on multiple operating systems, such as iOS and Linux, in addition to Windows. WampServer is only offered for Windows.

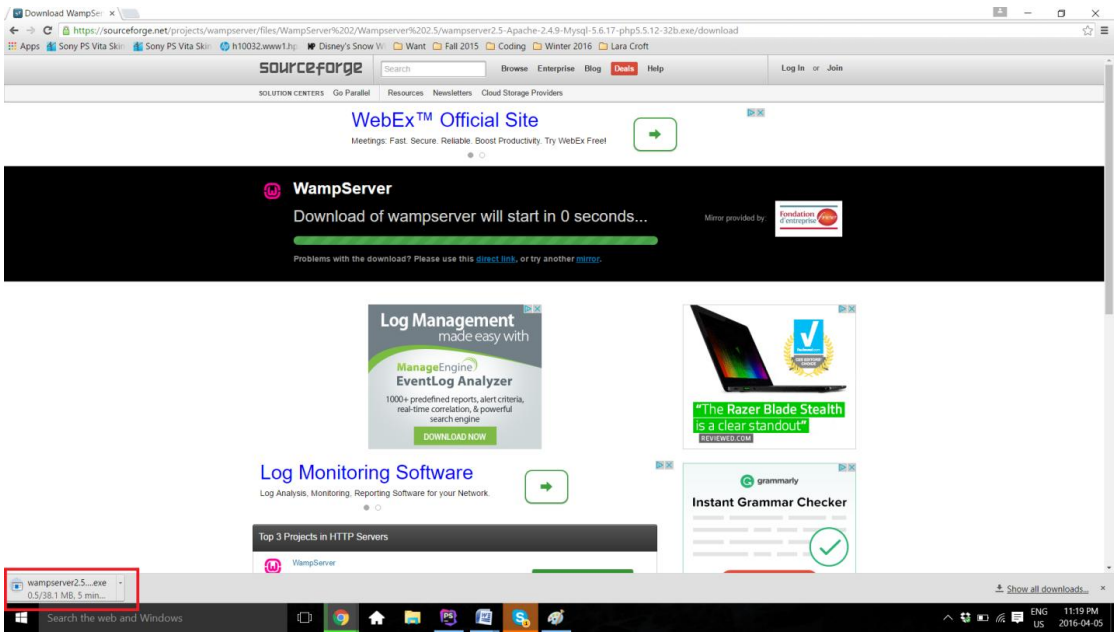




As shown here, there is a couple of warnings regarding WampServer. It is especially important to pay attention to second **warning**. In order for WampServer to work, Visual Studio is required. This is because Apache works along with Visual Studio. If Visual Studio 2012 is not installed on the computer, the administrator should click on the link https://www.microsoft.com/en-us/download/details.aspx?id=30679 found in the same warning. This opens a Microsoft download page for the correct version of Visual Studio. Once downloaded, Visual Studio can be installed. No special modification is required, therefore it is only necessary to follow the instructions displayed on Visual Studio installation window and click on **Next** a couple of times until the installation has begun, and proceeded successfully. It is important to note that there is a paid version of Visual Studio, but it is not required as the free version works equally well for the purpose of Apache.



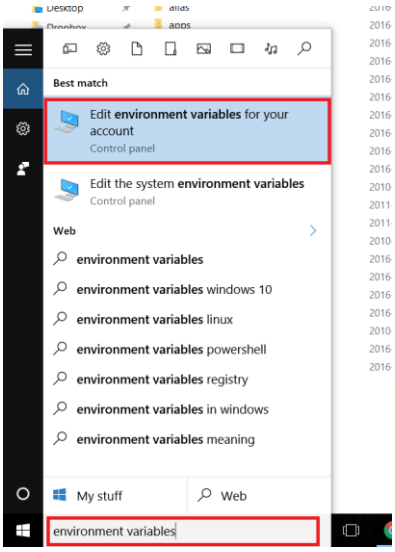
Once Visual Studio is set, the administrator can now go back to the WampServer website and click **download directly**. A new page from the website SourceForge will open, containing the file. A few seconds has to pass for the download to start. On Google Chrome, the download will be shown on the bottom left of the web page. After the download is completed, it is only sufficient to click on the file and the installation will begin. Otherwise, as any other downloads, the file has to be located to wherever downloaded files are usually saved (it is usually in the **Download** directory).

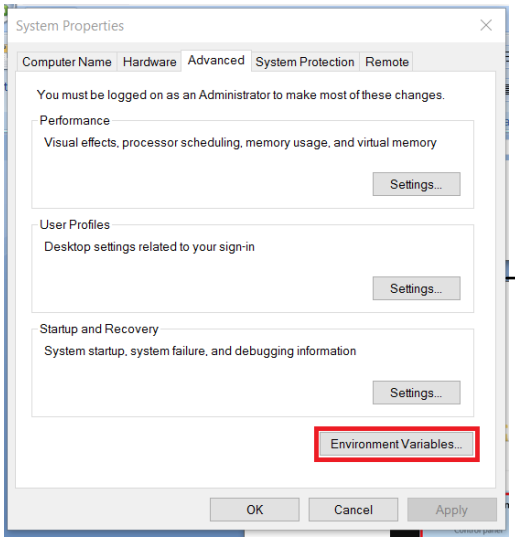


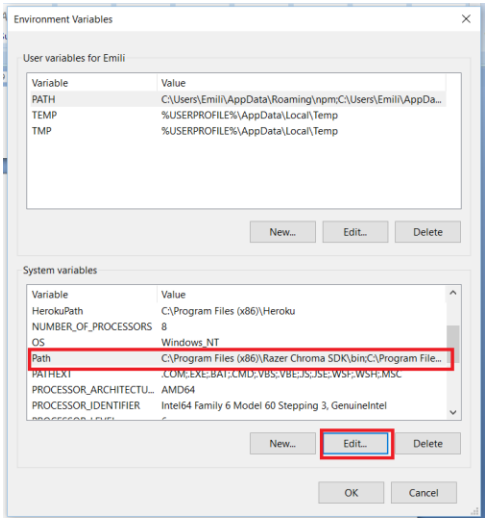
Once again, installing WampServer does not require any particular specifications or changes on the automatic procedure. The only thing required from the administrator is to keep track of where the software is installed. In this example, the software has been saved under **C:\wamp**.

Step 2: Create the PHP and MySQL Path in the Environment Variables.

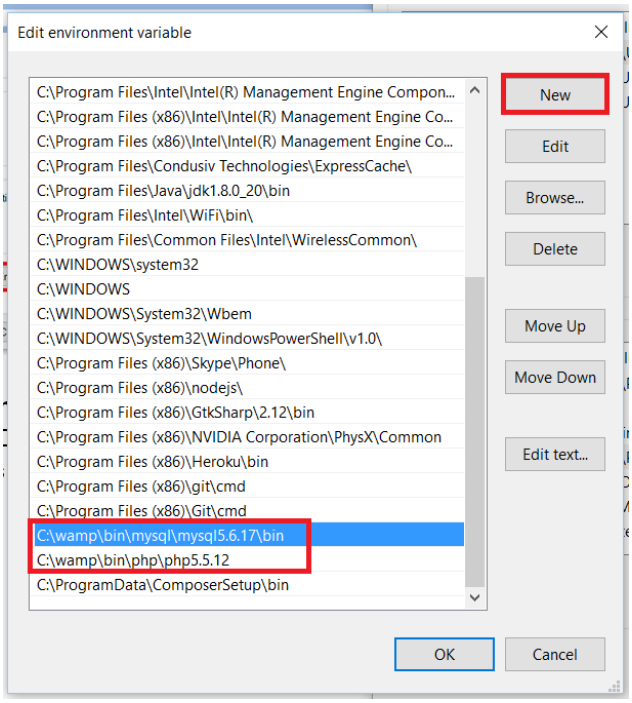
This step enables the Command Prompt to have access to the PHP and MySQL, therefore being able to manipulate and launch the software from there later on in the procedure. First, it is possible to find the Environment Variables by typing the name in the **Search** available on the computer.





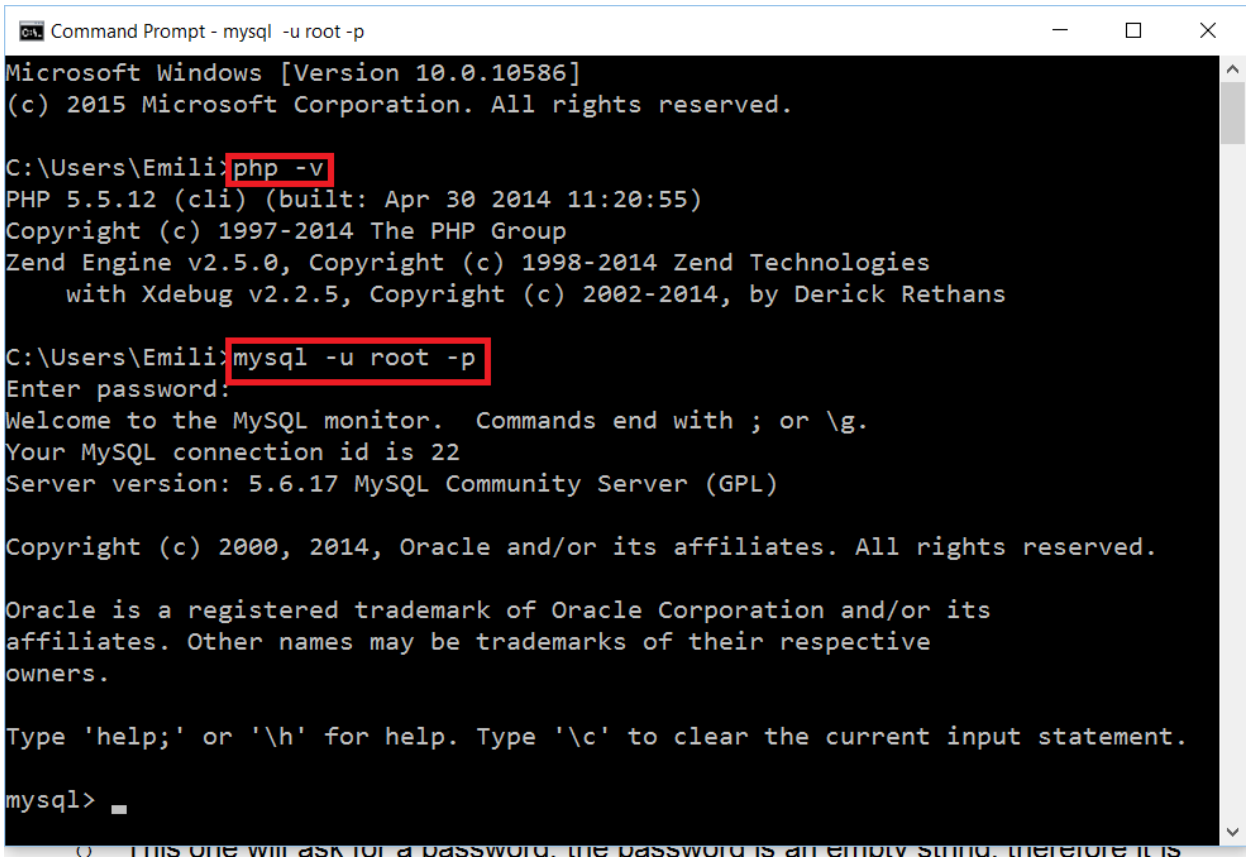


The control panel of the **System Properties** opens, under the **Advanced** section. **Environment Variables..** can be found there. Once clicked, another window called **Environment Variables** will open. In the **System variables**, **Path** has to be located, and then edited. Two new variables have to be added. The first one is the path to the Wamp's MySQL bin file. The file has to be located into the computer, as it is **C:\wamp\bin\mysql\mysql5.6.17\bin** in this case. The second one is the path to the Wamp's PHP file, as it is **C:\wamp\bin\php\php5.5.12** in this case. Once everything is set, the administrator has to click **OK** on all three windows, therefore saving the changes and closing the **Control Panel**.



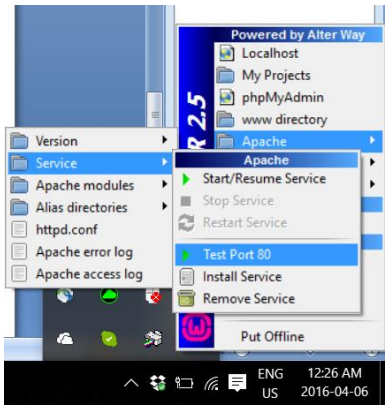
To make sure the path worked, the following commands have to be typed on the Command Prompt and the display should be similar to the following image, without any error message:

- **php -v**
- **mysql -u root -p**
    - This one will ask for a password, the password is an empty string, therefore it is only necessary to click on **Enter** on the keyboard.



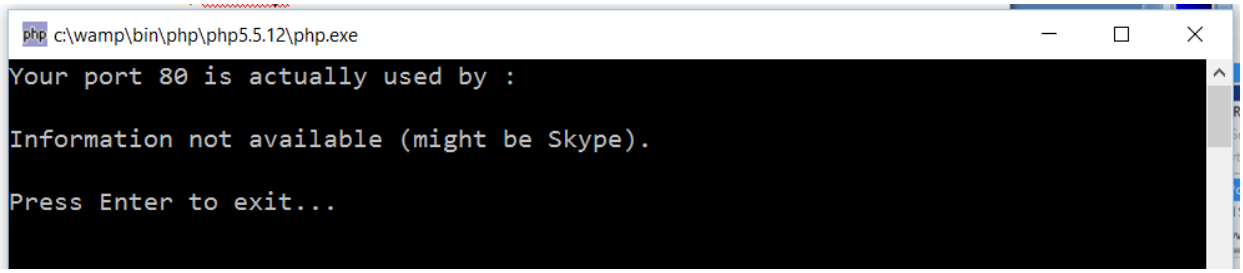
Step 3: Turn on WampServer and Test the Connection

First, Wamp has to be turned on. If the color does not turn green after clicking **Start All Services** and the color is stuck at orange, it is most likely because **port 80**. WampServer, as well for XAMPP, uses **port 80** for Apache. Therefore, this error is probably caused because another application is already using **port 80**. It is possible to test the connection by following this path on the Wamp application: **Apache > Service > Test Port 80**.



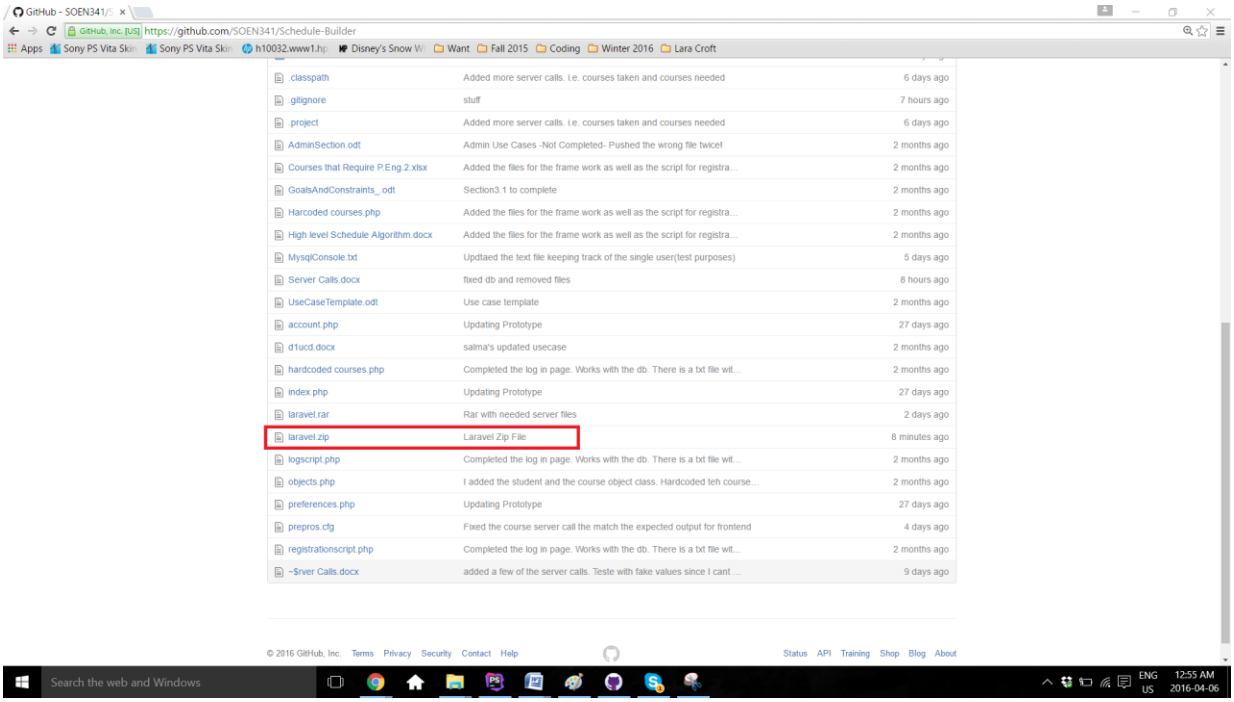
This is going to display an error if the port is really not available for Apache. As shown here, the error is usually caused by Skype being on at the same time, as both Apache and Skype use the same port. If it is

the case, Skype has to be completely closed. It is then only sufficient to **Restart All Services** on Wamp, and the icon should turn green.



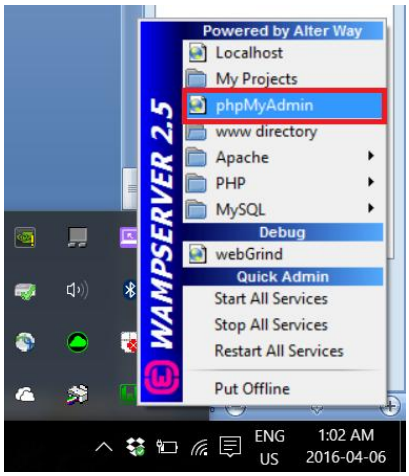
Step 4: Download the project from GitHub

The project has to be now retrieved from GitHub. This can be done from the following link: https://github.com/SOEN341/Schedule-Builder. As it is a zip file, the following has to be unzipped. The file can then be placed at any desired location.

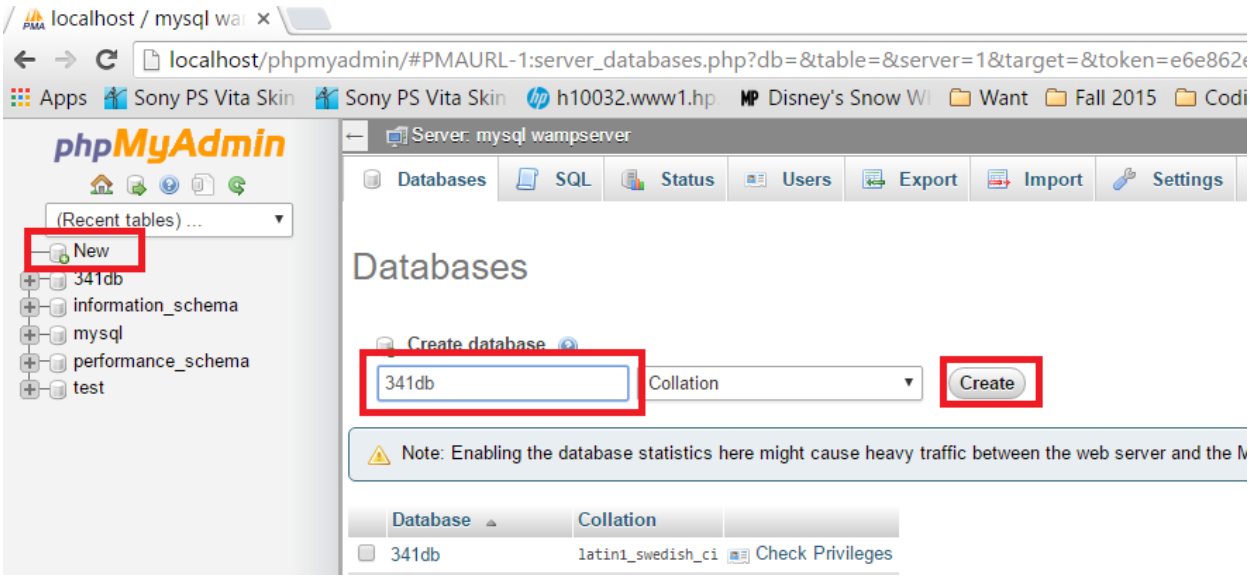


Step 5: Set Database

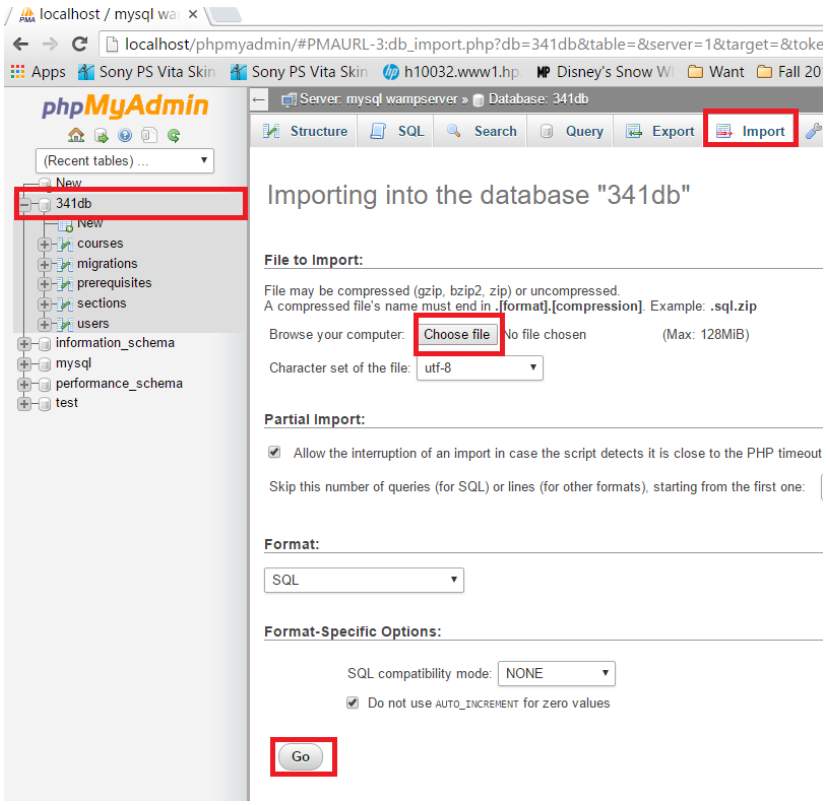
In order for the software to work, it has to be connected to the Database. From the **Laravel** file downloaded from GitHub, there is a file called **finaldbfile.sql**. This is the entire database of the project, containing all courses, sections, users information. This database has to be uploaded on MySQL on Wamp. To do this, **phpMyAdmin** has to be opened via Wamp.



phpMyAdmin will then open on the default browser. Then, the database has to be created, by clicking on **New**. The name of the database is **341db**.

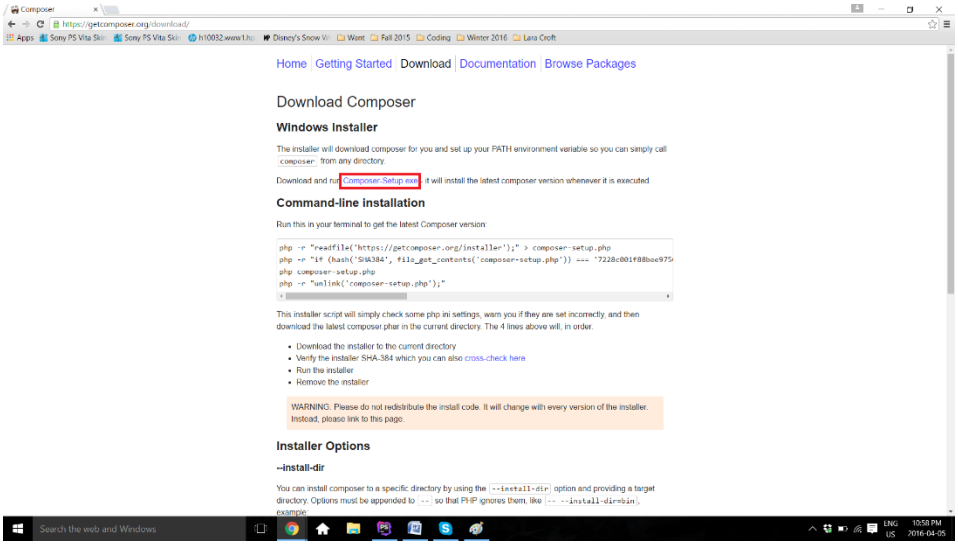


Afterwards, **finaldbfile.sql** has to be imported onto the same **341db** database. On **phpMyAdmin**, 341db has to be selected, then **Import**, then **Choose file**. Navigate to the **Laravel** file and select **finaldbfile.sql**. Once the upload done, this section can be finalized by clicking **Go**. A green confirmation message will be displayed, saying the import has been successful.



Step 6: Download and Install Compositor

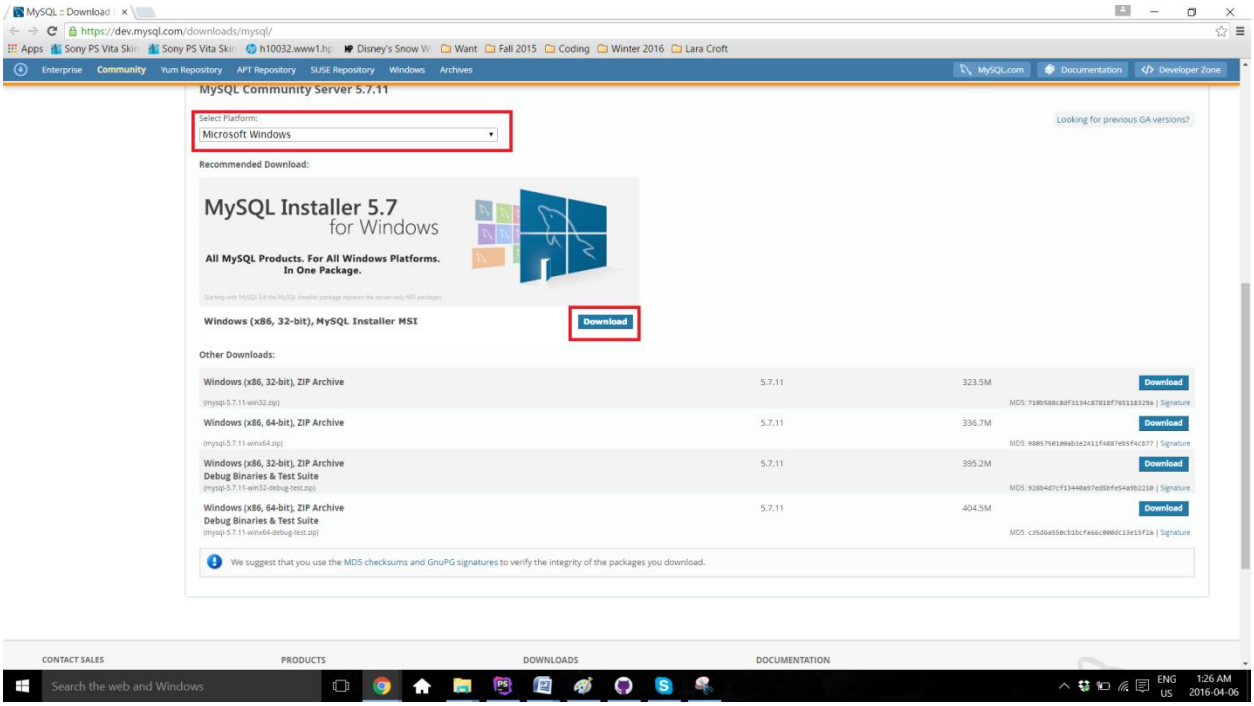
Compositor is a dependencies manager for PHP. This means that Compositor will handle all the needed libraries and files, in order to ease the development of the project. It can be download from https://getcomposer.org/download/. It is once again only necessary to follow the installation instructions. However, it is important keep track of where the Compositor gets the php.exe file from. It has to be from Wamp. If it is not the case, relocate the path to the **php.exe** inside of Wamp. The path is **C:\wamp\bin\php\php5.5.12\php.exe** for this case.



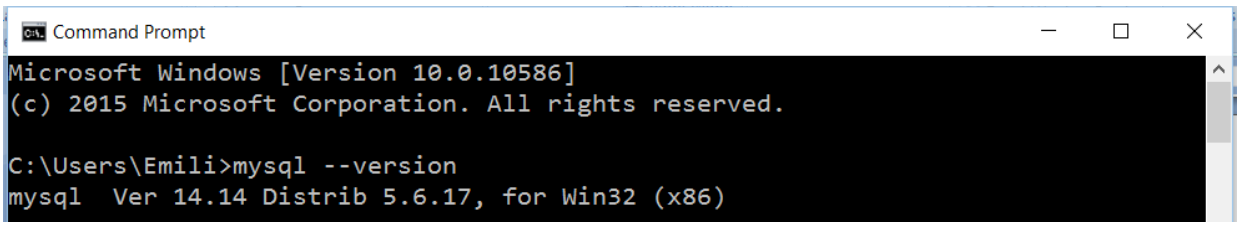
Step 7: Download and Install MySQL

Even though MySQL has been downloaded using Wamp, there seem to be some connection problem from Wamp if the software is not directly inside its **www** directory. To solve this problem, the original MySQL software is installed again, which helps to install MySQL correctly across the whole system. This allows the project to see the available MySQL throughout the system.

The software can be downloaded from https://dev.mysql.com/downloads/mysql/. The download can be found under **Community Server**. The desired OS can be picked (Windows in this case). **MySQL Installer 5.7 for Windows** is the desired version.

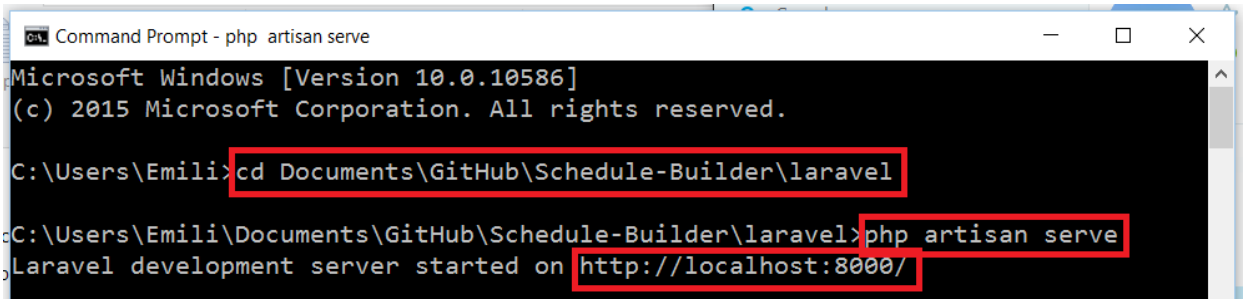


As any other past installation, it is only necessary to follow the instructions provided by the installer. Once installed, it is recommeded to test if MySQL is setup correctly, by typing **mysql --version** on **Command Prompt**.

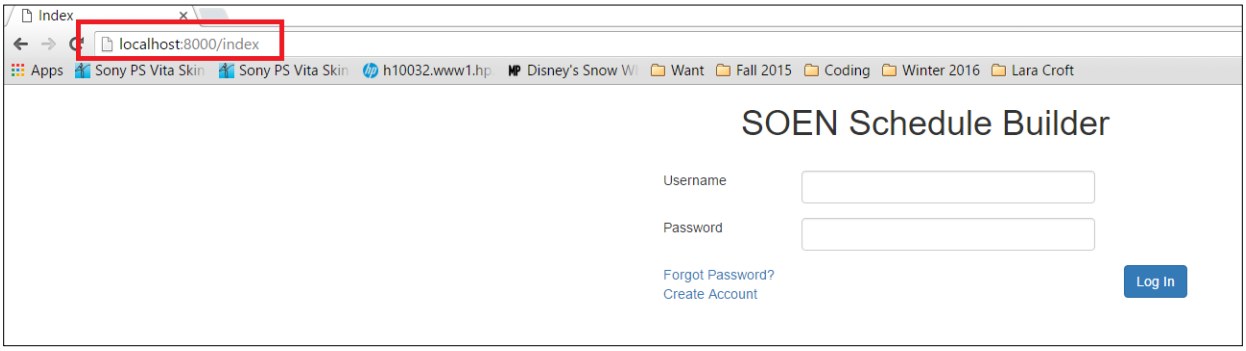


Step 8: Run Scheduler Project

Once everything is set up and Wamp is turned on and green, the project can now run. To do so, **Command Prompt** is used. The first command to type is **cd *project location***. To navigate backward in folders, the command **cd..** does the trick. Once in the file of the project, the last command is executed: **php artisan serve**. The output **Laravel development server started on http://localhost:8000/** should be displayed.



Now, it is only necessary to open a browser, and type **localhost:8000** in the URL bar.



3.2 User Manual

The Scheduler is a system used by undergraduate software engineering students at Concordia University, which generates a class schedule based on the student's record. It can be accesses from: https://schedule-heroku.herokuapp.com/index.

Logging In

Upon entering the website, a username and password will be needed to identify the user and allow access to the Scheduler. Returning users can simply login, while new users will be required to create a new account using the button "Create Account".

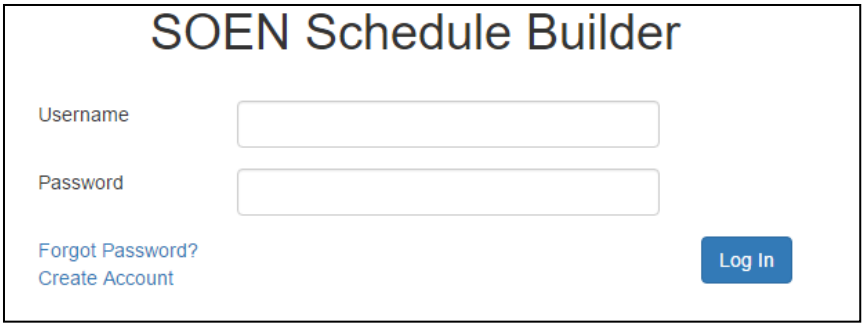


Figure 1: *Home Page*

Resetting the Password

If one of the returning users forgot their password, they can press the "Forgot Password?" link, which will allow them to reset their password by receiving a temporary one through e-mail. Successfully logging in will grant access to the menu page.

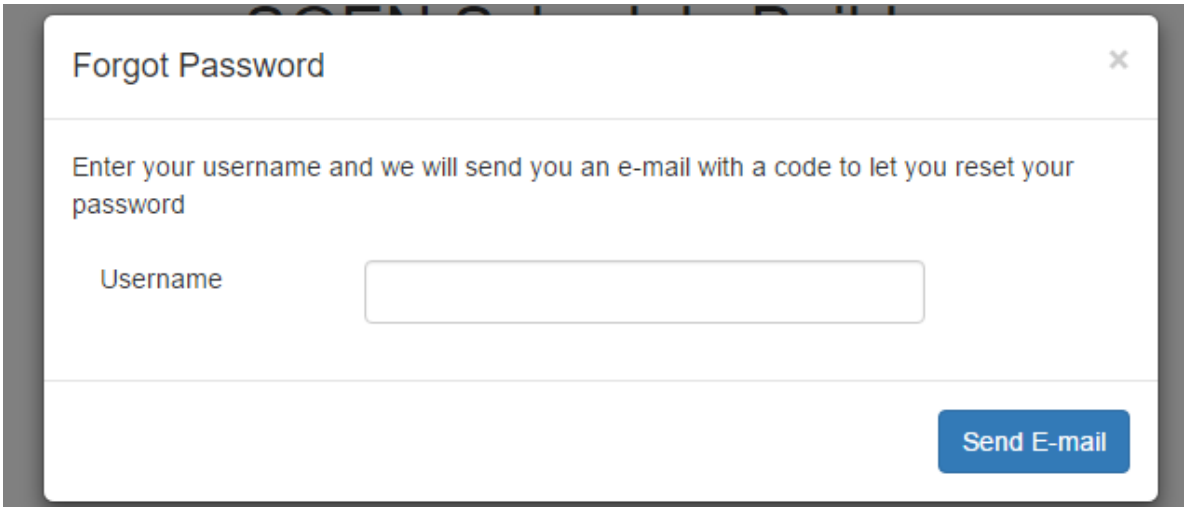


Figure 2: *Resetting Password*

Account Creation

When first time users click on "register", the screenshot below will be shown. They will simply have to follow the instruction and fill out each box with the required information. Once this is done, the menu page will open.

Figure 3: *Signing up*

The menu page is accessed after entering a valid username and password or after creating an account. This is where students make all their decisions concerning their schedule.

Here are the different actions users can choose in the menu page:

Preferences:

In this page, the users will first see when and where they would like to take courses. As seen in the screenshot below, the choices of having a day of the week off, taking classes in the morning afternoon or at night and the number of classes to take in the semester are all preferences to be set.

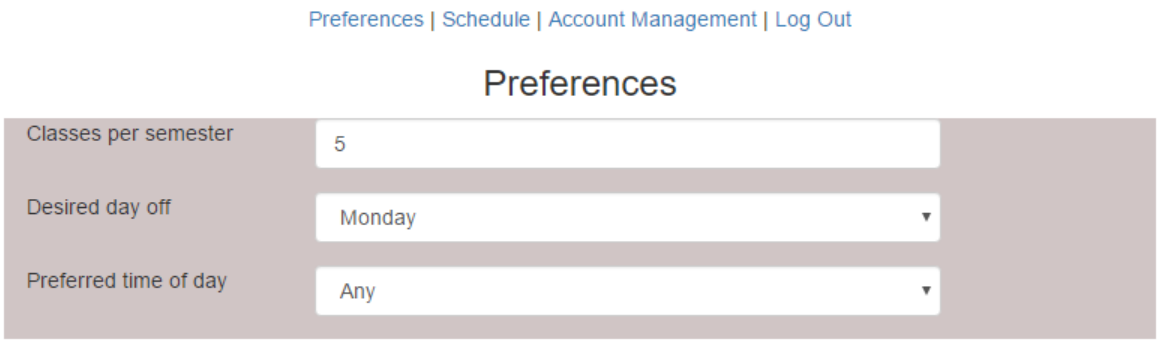


Figure 4: *Modifying preferences*

Adding classes

The preferences page also allows the students to input their taken courses as well as the needed courses.

One way to add the taken classes by choosing the semester already completed and generating a course list of all the courses that were offered. The user can then simply choose the courses he or she already completed.

The other method is by using the "Add Class" button and manually writing the course name number.

For the needed courses, the same concept is used. Using the "Add Class", courses can be saved. Each of the saved courses can be edited or removed using the two icons located next to the course numbers.

Using the "build schedule" button seen at the bottom of the page, schedules will be generated and can be accessed in the "Schedule" page.

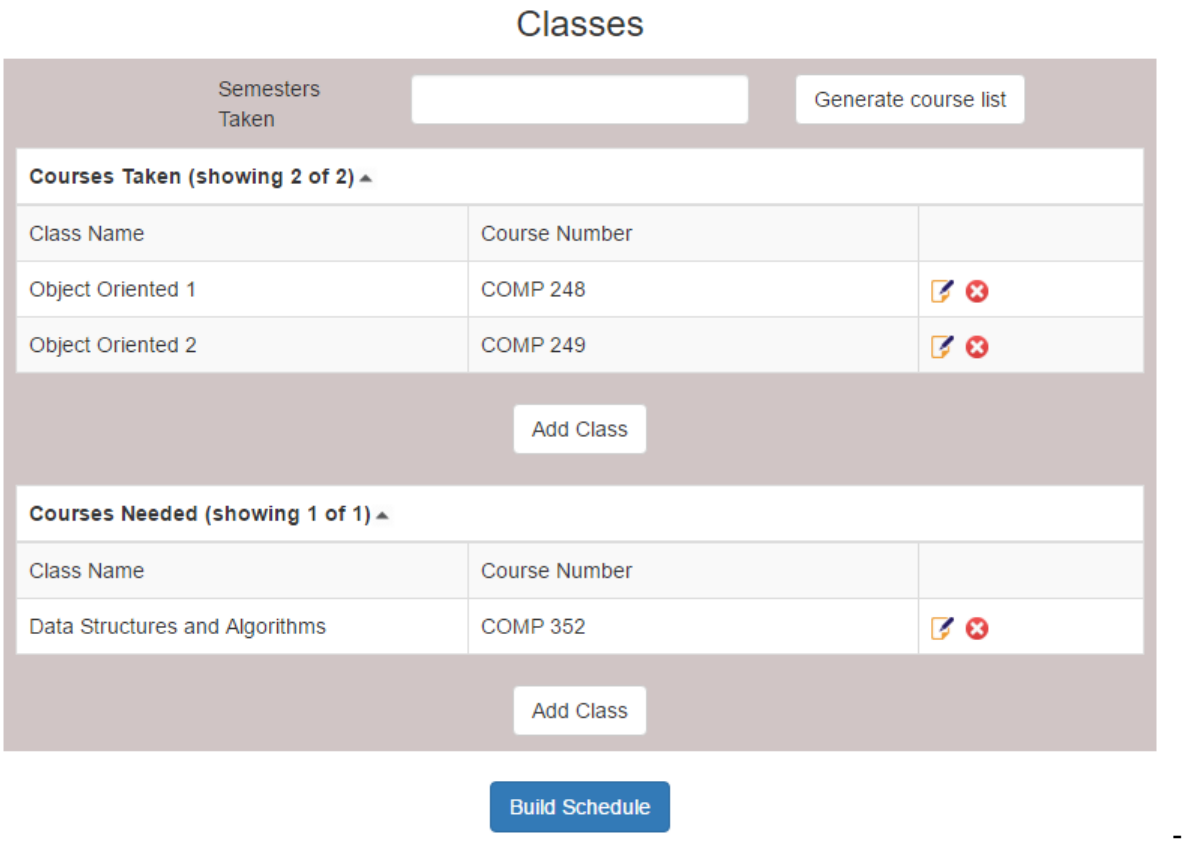


Figure 5: *Adding courses*

Account Management:

By clicking on the account information link, users can modify their username, email or password (see figure 6).

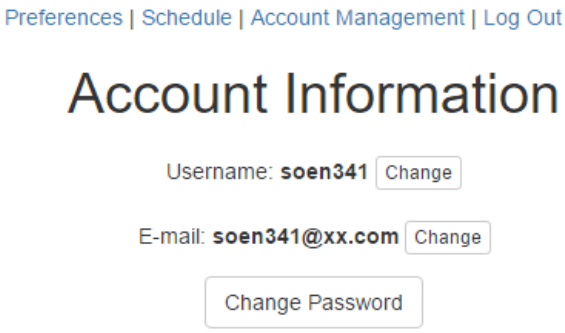


Figure 6: Account information

Schedule:

The schedule page is where all the possible schedules corresponding to user's preferences are going to be generated. It is to be noted that the Scheduler needs the user to input all their taken and needed classes in order to generate a schedule.

The various schedules can be navigated by using the arrows at the top. When a schedule is deemed acceptable, it can be selected using the "Select This Schedule" button. Upon selecting a schedule, the sequence for the rest of the academic years will be created giving a guideline to follow (see figures below).

Figure 7: *Schedule*

Year 2 Fall	
Class Number	**Course Name**
COMP 348	Principles of Programming Languages
COMP 352	Data Structures and Algorithms
ENCS 282	Technical Writing and Communication
ENGR 202	Sustainable Development and Environmental Stewardship

Year 2 Winter	
Class Number	**Course Name**
COMP 346	Operating Systems
ELEC 275	Principles of Electrical Engineering
ENGR 371	Probability and Statistics in Engineering
SOEN 331	Introduction to Formal Methods for Software Engineering
SOEN 331	Software Process

Figure 8: *Full sequence*

Logging out

To log out, the “log out button” has to be pressed. This will exit the system bring the user back to the home page.

3.3 ADMIN MANUAL

The Admins access the website from the same domain name: https://schedule-heroku.herokuapp.com/index.

Logging Page

Upon entering the website, a username and password will be demanded. Admins should have a special login username and password to enter the website specifically designed for them.

If the password is forgotten, by clicking on “Forgot Password”, it can easily be recovered.

Successfully logging in brings the Admins to the main page of the website.

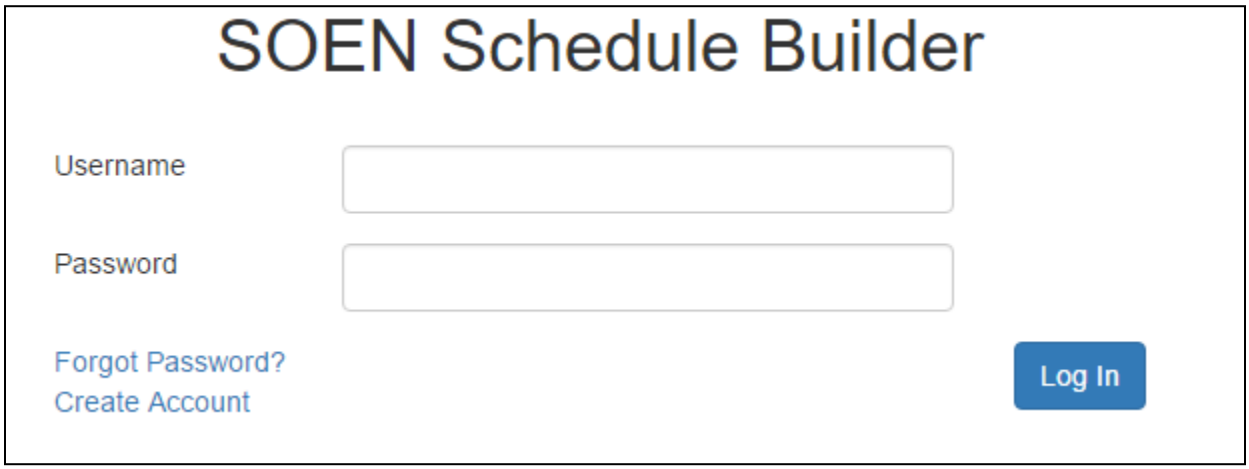


Figure 9: Home Page

The Admins can add or edit courses present in the database. Entering this page provides the Admins with a whole list of the courses.

Adding a Course

In order to add a course, the button “add class” is to be pressed. This will provide the Admins with a form to fill. More specifically, entering the course number, name and the semester.

Adding/Modifying Information and Sections

Next to courses seen on the page, there is a small icon showing a paper and a pencil. By pressing it, it allows the admins to change the descriptions of the course or its number. Also, by clicking on the course name, the different sections of the course are displayed. The same paper and pencil icon can be pressed to modify the time of the sections and the classroom.

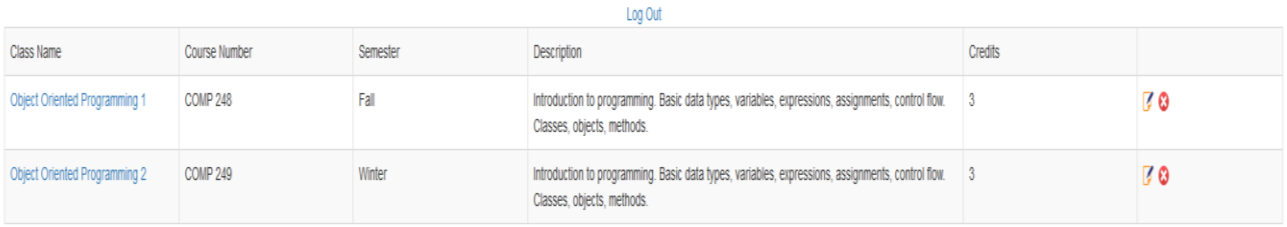


Figure 10: Course list

Figure 11: Class Sections

Log Out

To log out, Admins can press the “log out” button at the top of the page.

4. Final Cost Estimate

A number of design and testing aspects were overlooked during our initial estimate. In the first deliverable, tasks such as creating the database, and ensuring that the system would be able to perform the necessary read and write operations were slightly underestimated. Testing was also.

The total project cost estimate as calculated to date is detailed in the table below. A side by side comparison between the original estimated cost and the actual cost illustrates the difference in hours that our team needed to make adjustments for.

Artifact		Estimated Cost in Hours	Final Cost in Hours
Deliverable 0		**5**	**5 ($125)**
Deliverable 1		**70**	**70 ($1750)**
Deliverable 2			
4+1 Architectural View	**Logical**	**10**	**10**
	Development		**3**
	Physical		**3**
	Process		**5**
	Scenarios		**1**
Subsystem Interface Specification		**25**	**20**
UML Class Diagram		**12**	**10**
Dynamic Design Scenario		**6**	**10**
Estimation		**7**	**5**
Rapid Prototyping Report		**22**	**25**
Testing		**15**	**15**
Risks		**4**	**4**
Total Hours		**101**	**111**
Cost Estimate ($25/hr)		**$2525**	**$2775**
Deliverable 3			
Database Creation		**20**	**20**
Web Interface Design		**20**	**16**
Database interaction & Server calls		**20**	**14**
Unit Testing		**20**	**30**
Requirement Testing		**20**	**20**
Stress Testing		**8**	**10**
Security Testing		**7**	**7**
Installation		**Unaccounted for**	**10**
Installation Manual		**6**	**12**
User's Manual		**6**	**8**
Administrator manual		**Unaccounted for**	**2**
Estimation		**4**	**3**
Total Hours		**129**	**152**
Cost Estimate ($25/hr)		**$3225**	**$3800**

Deliverable 4			
Deliverable 1 revision	Use Cases	10	4
	Domain Model		4
	Scoping		3
	Risks		2
Deliverable 2 Revision	Class Diagram	10	3
	Component Diagram		3
	Subsystems		4
	Deployment Diagram		4
	Sequence Diagram		4
	Contracts		3
Deliverable 3 Revision	Testing Table	10	4
	Requirement Testing		5
	Instructions		4
	Unit		3
Coding	Front End	20	10
	Back End		10
Total Hours		50	70
Cost Estimate ($25/hr)		$1000	1750

Apart from deliverables 0 & 1; where the cost estimate was accurate, all other deliverables came with an added value. The second deliverable registered a 10 hours cost increase; which was calculated to amount to $250.

An additional cost of 23 hours has been incurred during the realization of the third deliverable bumping the cost from an estimated $3225 to an actual $3800. Moreover, the 50 hours estimate that was projected for the realization of the final deliverable was also exceeded, when our team had to invest an additional 20 hours.

By summing the costs incurred from the subtotals in the table above, we can summarize the cost estimate margin of error in the table below:

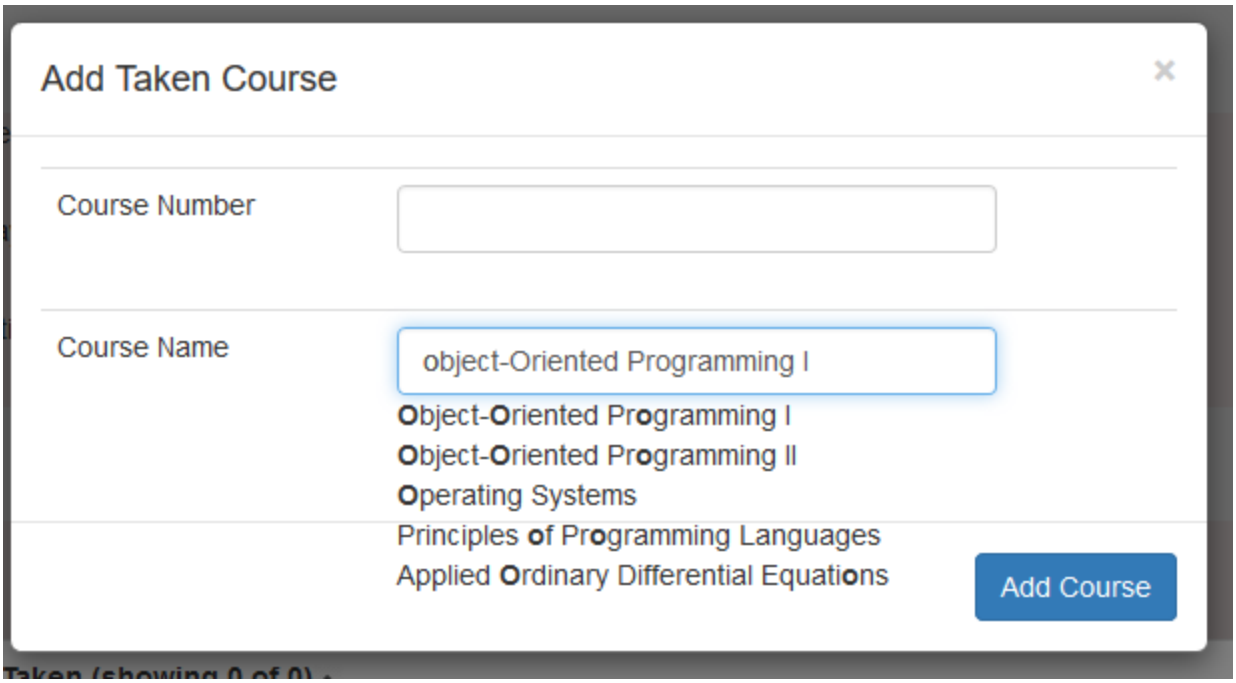
	Estimated Cost in Hours	Final Cost in Hours
Deliverable 0	**5**	**5**
Deliverable 1	**70**	**70**
Deliverable 2	**101**	**111**
Deliverable 3	**129**	**152**
Deliverable 4	**50**	**70**
Total Number of Hours	**355**	**408**
Total Cost($25/Hour)	**8875**	**10200**
Estimation Error	**13%**	

5. Programming Specifications

5.1 Open-Source Technology Used

5.1.1. Twitter Typeahead

We have implemented Twitter's typeahead component to improve the user experience. It is a jQuery open-source text component that provides auto-completion suggestions as the user types (much like the google search bar). We use it to help the user add their needed and taken courses. The list of courses given in the auto-complete is retrieved from the server. This not only makes a better user experience, but makes it less likely for the user to input a course that doesn't exist by accident.

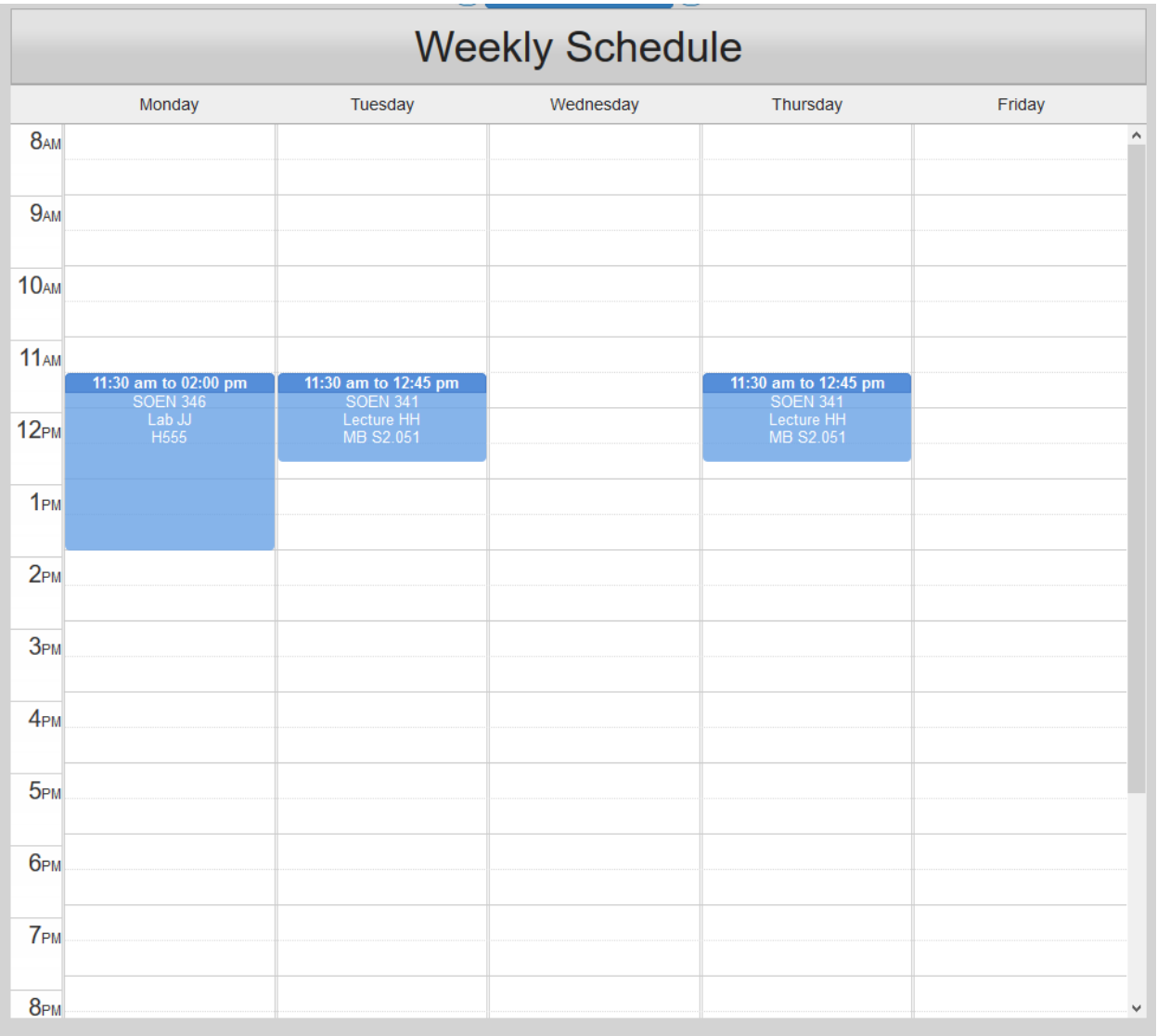


Twitter Typeahead

This component can be found at: https://twitter.github.io/typeahead.js/

5.1.2. JQuery Week Calendar

We use the jQuery Week Calendar to display the schedule once it has been generated. This majorly simplified this part of the UI. The component did need some customization for our uses, however, since originally it allows users to create, edit, and remove items on the schedule. It also displays items on a calendar with dates. None of these were things we wanted, so we had to customize the component to make it closer to what we wanted.



The Customized jQuery Week Calendar

This component can be found at: http://www.jqueryrain.com/2012/04/jquery-week-calendar-with-demo/

5.1.3. React Bootstrap

Bootstrap is a package of component that can be used for any web UI. It includes buttons, form objects, tables, dialogs, and many more commonly used UI components. They were made to have a list of components of the same style that behave the same across all major browsers. React Bootstrap is a package of the Bootstrap components that were made to work with React. They helped make our pages look consistent and professional as well as work across all major browsers without too much extra work.

React Bootstrap can be found at: https://react-bootstrap.github.io/

5.2 ENCRYPTION & SECURITY

Bcrypt is used to encrypted passwords. It is a key derivation function that has a salt to protect from table attacks. As well bcrypt can be changed to a slower encryption which allows for stronger encryption. Bcrypt will be using crypt blowfish hashing that has a salt followed by two digit cost parameter and 22 characters from the alphabet.

In this encryption, username will be used as salt. This will prevent rainbow attack and will make it difficult to break it.

A salt is a random data that is used as an additional input to a hashed password. A new salt is randomly generated for each password.

The number 14 before the $ sign is the cost parameter; it decides how intensive the hashing is. According to php documentation, the range is from 04 to 31.

5.3 Server Calls

The following is the fully documented list of server calls used in the product. It lists the call, the type of call, the expected data to be sent with the call, and the expected format of the returned data.

These contracts were important to make sure the front and back end teams could work independently. Once these server calls were decided on, each team could work independently, programming their side to work according to these calls. The front-end team was able to fake these server calls using some hardcoded values until the back-end team had finished programming them. This allowed the front-end team to make everything work and then they just had to replace the server calls with the real calls when they were finished.

URL	Purpose	Type	Input	Returns
/index	Login/Home Page	GET	None	The login/home page
/preferences	Preferences Page	GET	None	The preferences page
/schedule	Schedule Page	GET	None	The schedule page
/sequence	Course Sequence Page	GET	None	The course sequence page
/account	Account Page	GET	None	The account page
/admin	Admin Courses Page	GET	None	The admin courses page
/admincourse	Admin Edit Courses Page	GET	None	The admin edit courses page
/courses	Get all courses in DB	POST	None	[{"courseId":"1","courseCode":"COMP 248","semester":"Fall","descri

				ption":"Introduction to programming. Basic data types, variables, expressions, assignments, control flow. Classes, objects, methods.","name":"Object-Oriented Programming I","credits":"3}] //goes on forever
/sections	Get all sections for all courses in DB	POST	None	^ Similar output
/sectioncourse	Get all section in DB for the specified course	POST	$courseId=$_POST('courseId');	^ Similar output
/userprefs	Get preferences for a user	POST	$username=$_POST['username'];	{"success":"true","username":"SprinkKing","courseload":"4","dayoff":"Monday","preferredTime":"Afternoons"}
/needed	Get needed courses for a user	POST	$username=$_POST['username'];	String that will be parsed {"List":[{"name":"oop", "number":"248"},{"name":"oop2", "number":"249"}]}
/taken	Get taken courses for a user	POST	$username=$_POST['username'];	String that will be parsed {"List":[{"name":"oop", "number":"248"},{"name":"oop2", "number":"249"}]}
/scheduler	Generate schedule	POST	$username=$_POST['username'];	
/addadmincourse	Add a course to the DB	POST	$json=$_POST['json']; //$json='{"section":"UI-X","sectionId":"1","classroom":"H-4","semester":"Winter","type":"Lab","dayOffered":"5","beginTime":"13:35:00","endTime":"14:35:00","courseId":"2","courseCode":"COMP 248","sectionNum":"1"}';	False: if course with that id already exists True: otherwise {"success":"true","courseID":"142","Course":"{\"courseId\":\"null\",\"courseCode\":\"COMP 2489895\",\"semester\":\"Summer\",\"description\":\"Salt level critical, abort !\",\"name\":\"Object-Oriented Programming I\",\"credits\":\"3\"}"}
/addadminsection	Add a section to the DB	POST	$json=$_POST['json']; //$json='{"section":"UI-X","sectionId":"1","classroom":"H-4","semester":"Winter","type":"Lab","dayOffered":"5","beginTime":"13:35:00","endTime":"14:35:	{"success":"true","SectionID":"352","Section":"{\"section\":\"UI-GR\",\"sectionId\":\"null\",\"classroom\":\"H-4\",\"semester\":\"Winter\",\"type\":\"Lab\",\"dayOffered\":\"5\",\"beginTime\":\"13:35:00\",\

			00","courseId":"2","courseCode":"COMP 248","sectionNum":"1"}';	"endTime\":\"14:35:00\",\"courseId\":\"2\",\"courseCode\":\"COMP 248\",\"sectionNum\":\"1\"}"}
/removeadmincourse	Remove a course from the DB	POST	//$courseId=$_POST['courseId']; //course id	N/A
/removeadminsection	Remove a section from the DB	POST	//$sectionId=$_POST['sectionId']; //section id	N/A
/editcourse	Edit course information	POST	$json=$_POST['json']; //$json='{"courseId":"1","courseCode":"COMP 248","semester":"Fall","description":" Critical amount of salt atteined","name":"Object-Oriented Programming I","credits":"3"}';	{"success":"true","Course":"{\"courseId\":\"1\",\"courseCode\":\"COMP 248\",\"semester\":\"Fall\",\"description\":\"Introduction to programming. Critical amount of salt atteined\",\"name\":\"Object-Oriented Programming I\",\"credits\":\"3\"}"}
/editsection	Edit section information	POST	$json=$_POST['json']; //$json='{"section":"UI-X","sectionId":"1","classroom":"H-905","semester":"Winter","type":"Lab","dayOffered":"5","beginTime":"13:35:00","endTime":"14:35:00","courseId":"2","courseCode":"COMP 248","sectionNum":"1"}';	{"success":"true","Section":"{\"section\":\"UI-GR\",\"sectionId\":\"1\",\"classroom\":\"H-420\",\"semester\":\"Winter\",\"type\":\"Lab\",\"dayOffered\":\"5\",\"beginTime\":\"13:35:00\",\"endTime\":\"14:35:00\",\"courseId\":\"2\",\"courseCode\":\"COMP 248\",\"sectionNum\":\"1\"}"}
/editpreferences	Edit user's preferences	POST	$username=$_POST['username'];//username $cload=$_POST['cload'];//courseload $dayoff=$_POST['dayoff'];//time off $preftime=$_POST['preftime'];//pref time	{"success":"true","username":"JasonB","courseload":"5","dayoff":"Monday","preftime":"Mornings"}
/editneededcourses	Edit user's needed courses	POST	$old=$_POST['username']; $json=$_POST['json']; // $json='{"List":[{"name":"oop", "number":"248"},{"name":"oop2", "number":"249"}]}';	{"success":"true","username":"user17","List":"{\"List\":[{\"name\":\"Operating Systems\",\"number\":\"COMP 346\"},{\"name\":\"Principles of Electrical Engineering\",\"number\":\"ELEC 275\"},{\"name\":\"Probability and Statistics in

				Engineering\",\"number\":\"ENG….. }]}"}
/edittakencourses	Edit user's taken courses	POST	{username:'', json: '{"List":[{"name":"oop", "number":"248"},{"name":"oop2", "number":"249"}]}'}	{"success":"true","username": "user17","List":"{\"List\":[{\"name\":\"Mathematics for Computer Science\",\"number\":\"COMP 232\"},{\"name\":\"Object-Oriented Programming I\",\"number\":\"COMP 248\"},…….{name\":\"Sustainable Development and Environmental Stewardship\",\"number\":\"ENGR 202\"}]}"}
/edituser name	Change username for user	POST	$old=$_POST['old']; // old username $new=$_POST['new']; //new username	False: new username exists already True: otherwise result"=>"true","username"=>"$new")
/editemail	Change e-mail for user	POST	$old=$_POST['old'];//username $new=$_POST['new'];//new email	("success"=>"true","username"=>"$old","email"=>"$new") False: username not found
/editpassword	Change password for user	POST	$username=$_POST['old']; //username $newPassword=$_POST['new']; //new password	"success"=>"true","password"=>"$new"));
/email	Get user's email	POST	$username=$_POST['username'];//username	E-mail as a String "result":"good","username":"JasonB","email":"newEnail@email.com"}
/login	Login verification	POST	$username=$_POST['username']; $password=$_POST['password'];	{success:bool, isAdmin:bool} {"success":"true","username": "JASONB","isAdmin":"true"}
/register	Register user	POST	$username=$_POST['username']; $password=$_POST['password']; $email=$_POST['email'];	False: username exists already True: otherwise {"success":"false","username": "SprinkKing","error":"usernametakenalready"}