Concordia University
Engineering and Computer Science

Software Process, SOEN 341/4 S, Winter 2016
Dr. Shang
Dr. Fancott
Mr. Morse

TimeTurner by team YAWD

Project Scope and Plan Document – Deliverable 1

Team members information	
Name	**SID**
Aline Koftikian	**27764162**
Bryce Drewery-Schoeler	**27283199**
Claudia Della Serra	**26766048**
Daniel Di Corpo	**26331602**
Dimitri Topaloglou	**29358269**
Erin Benderoff	**27768478**
Ideawin-Bunthy Koun	**26314155**
Kevin Yasmine	**27195346**
Lori Dalkin	**27738293**
Marc-Andre Leclair	**27754876**
Philip Lim	**27485506**
Ryan Lee	**27752504**

TABLE OF CONTENTS

1. Introduction

Every year, newly admitted students at Concordia University are delegated a course sequence based on their matriculated program. Students are required create a schedule for themselves based on this course sequence and follow it thoroughly until the end of their degree. New students doing this on their own for the first time, and even experienced students, may encounter a number of difficulties when making schedules to fit this sequence, such as struggling to meet proper course prerequisites and managing time conflicts amongst multiple courses. This process is often long, tedious, and takes away from valuable time students may instead spend studying, working, or engaged in extracurricular activities. The following project, developed by a group of 12 undergraduate Software Engineering students at Concordia University, was created to simplify this process.

TimeTurner will be a web application implemented by Concordia University in order to aid students in planning and creating a schedule of courses for the duration of their degrees. Important factors to be kept in mind throughout the creation of this software system are ease of use, efficiency, speed, and overall effectiveness of the application in making scheduling of classes quick, easy, and personalized to students' needs. Students will be given a sequence of classes recommended to them by their faculty, much the same as current students in the Faculty of Engineering and Computer Science are given at the start of their degrees. Based on this sequence and each student's preferences, personalized schedules will be generated for the students' entire degree, thus saving the students much time and energy.

2. Project Description

The proposed and outlined web application, known as TimeTurner, is designed to auto-generate a student's course sequence from their first semester up until the end of their degree. It takes into account user input preferences and and any previously completed courses or course prerequisites before creating this sequence. Preferences can be made by the student and include options such as night classes or having particular days off. The application will notify the user if a certain preference suggested results in an impossibility or conflict in the sequence. This sequence generator will be able to create a sequence at any point throughout the user's degree, if sudden change in circumstances were to arise.

The goal of this application is to simplify the method with which students may decide and schedule their courses. If a course must be redone, the generator can decide what other courses should move where in regards to the remaining courses to be completed, which can be done in seconds, rather than hours. It saves the time of the user, in a simple and efficient manner. Ultimately, the system's end goal will be to simplify a student's task of creating their own course schedule in order to allow students to redirect their time to other more important activities, thus making course registration much simpler, quicker, and easier.

3. Goals and Constraints

3.1 Functional Requirements

3.1.1 Login

This Use Case enables any user to log on onto the scheduler to access their own profile to then effectuate whatever they need to in order to accomplish what they initially needed to. The log in action requests the attention of the server to pull the information that the user requested. This is done by the user entering his personal information (ID in this case) and his password in the password box. This, if done correctly, will allow the user to see his account information such as his generated schedule.

3.1.2 Logout

When the user is done his work on his TimeTurner profile, he or she will be able to log out from their account, cutting all their connection with any data associated to their username. This means that the user will have to go through the log in action again if he or she wants to access his personal information such as schedule.

~~3.1.3 Create Course Sequence~~

~~Once a student is logged in, they may create a course sequence. This course sequence will prompt the student for preferences upon when they want their classes, how many classes they desire per semester, etc. For instance, this could include morning versus evening classes. The course sequence will include the following four years (assuming the user is a student who just started his or her undergraduate program). Once the course sequence is created, a student will be able to access it to generate a schedule.~~

3.1.4 Browse Course List

This use case allows the user to browse a list of all relevant courses from the course calendar. The user has to log into the system and select the 'Browse Course List' option. The system must display a page containing all the courses in order by name. The user can scroll down the page and view any courses he or she is interested in with their description.

~~3.1.5 View Course Sequence~~

~~This use case allows the user to view his or her program's course sequence according to the user's preferences or the generic course sequence if a schedule has not yet been generated. The user needs to log into the scheduler and select the 'View Course Sequence' option. The system will then display a page which contains the course sequence for the next four years that the student has to follow.~~

3.1.6 Generate Schedule

This use case allows the system to generate a schedule by first allowing the user to select his or her preferences. Once selected, the system will generate a schedule based on these preferences and the student's records, such as previously enrolled courses, prerequisites and co-requisites as well as currently enrolled courses. The schedule will also be kept inside the system once it has been generated.

3.1.7 View a Saved Schedule

This use case allows the user, a student, to view a previously generated schedule that was saved to the system, either during the current session or a previous session. For this use case, the user must be logged into the system and have at least one schedule saved to the system. First, the user views a list of their saved schedules and selects which one they wish to view. Then, the system displays that schedule for the user to review.

3.1.8 View Academic Record

This use case allows the user, a student, to view their academic record. The academic record includes all completed courses and courses in which the student is currently enrolled. The record displays the name, number of credits and grade (if completed) for each course. The use case requires that the user be logged into the system and have completed and/or be enrolled in at least one course. First, the user requests to view their academic record. The system then produces the record for the user to view.

3.1.9 Drop Course

This use case allows the user, a student, to remove a course from their generated schedule. It requires that the user be logged into the system and have generated a schedule containing one or more courses. First, the user selects which course they would like to remove from the schedule. Next, the system prompts the user to confirm if they are sure they wish to remove the course. Once the user confirms the removal, the system removes that course from the schedule and generates a modified schedule which does not contain the deleted course.

~~3.1.10 Add Course~~

~~This use case allows a student to search for and manually add a course to their schedule. The student searches for a course and selects a possible section from what is available to them. The system verifies that the student has completed all prerequisites required to take the course selected. If the student is eligible, the course will be added to the generated schedule.~~

3.1.11 Save Generated Schedule

This use case allows a student to save a schedule they have generated to their accounts. After generating a schedule, the student can request the system save their generated schedule. The system will then record the information pertaining to the desired schedule and save it with the user records. The student can then access this schedule at a later time.

~~3.1.12 Weekly View Of Schedule~~

~~The View Generated Schedule use case allows a student user to view their schedule in a weekly format. The time at which each class starts and ends will be displayed in a visibly appealing manner. Each class will have the teacher, room number, type of class, and course code displayed as well. The default schedule shown would be that of the current week.~~

3.1.13 Manage Courses (Administrator)

The Manage Courses use case allows the user, an Administrator, to make a change to an existing course in the course bank (list of all courses). Besides viewing course details, the changes possible can be to change the course code, change the course description (name), change the number of credits of the course, and delete the course. If no change has been done, the page will return to the previous course bank. If a modification is made, an updated course bank will be saved.

3.1.14 View A Course (Administrator)

The View a Course requirement allows the user, an Administrator, to view course details for a selected course from the course bank. This will produce a new page that loads the course details which include the course I.D., course code, and course description.

3.1.15 Update A Course (Adminsitrator)

The Update a Course requirement allows the user, an Administrator, to edit course details of a selected course from the course bank. The Administrator will be able to view the current values of course information and will be able to modify the values that they wish. The Administrator makes all necessary changes, and must confirm their choice to save the course details. The system will register these details and update the course bank accordingly.

3.1.16 Delete A Course (Administrator)

The Delete a Course requirement allows the user, an Administrator, to delete a course from the course bank. This requirement allows Administrators to remove any courses that are no longer being offered from the course bank. Once deleted, the system will update the course bank accordingly.

3.1.17 Create Course (Administrator)

The administrator wishes to create a class to be added to the database of courses. Upon viewing the course database, the administrator requests to create a course from the side menu. The system will prompt the administrator for the course information (code, description and amount of credits). Upon entering the course information, the system will record the course as a new course in the database and present the administrator with the course information as it has been recorded.

3.1.18 Browse Course (Administrator)

The Administrator wishes to view all of the courses in the course database. Upon logging in, he selects the course tab at the top of the page. The system will then present the administrator with a page listing all of the courses in the database. The administrator may search through the list for any courses he wishes to view. The side menu will present the administrator with options to manage or create courses.

3.1.19 Add User (Administrator)

The administrator wishes to create a user to be added to the database of users. Upon viewing the user database, the administrator requests to create a user from the side menu. The system must prompt the administrator to enter user information (username, first name, last name, net name and password) and to select the privilege of the user (student or administrator). As the administrator fills the form with valid information and submits it, the system must record the user as a new user in the database and present the administrator information about the newly added user.

3.1.20 Search For Users (Administrator)

The administrator wishes to search for one or many particular user(s) in the system. To do so, the administrator, through the user management page, specifies the search criteria and indicates that the search if ready to be performed. Upon completion of the search, the administrator will be able to view the result of the search. More precisely, a table of users matching the search criteria is displayed for the administrator.

3.1.21 View User (Administrator)

The administrator wants to view the profile of a user. In order to do so, the administrator indicates which user is to be displayed through a search result table containing one or many users. A page containing the detailed profile of the indicated user is then displayed for the administrator to consult.

3.1.22 Update User

The administrator needs to update some information about a user of the system. In order to accomplish that task, the administrator first indicates that a specific user needs to be updated. Editable fields corresponding to the user's information are then displayed for the administrator to change. Once the changes done, the administrator indicates that the changes have been completed. If the changes are valid, the updated profile of the user is displayed, otherwise, the administrator receives an error message.

3.1.23 Delete User

The administrator needs to delete a user from the system. To do so, the administrator needs to indicate which user is to be deleted, and confirm the action to be undertaken. Once completed, an updated table of users without the deleted user is displayed to the administrator.

3.1.24 Use Case Diagram

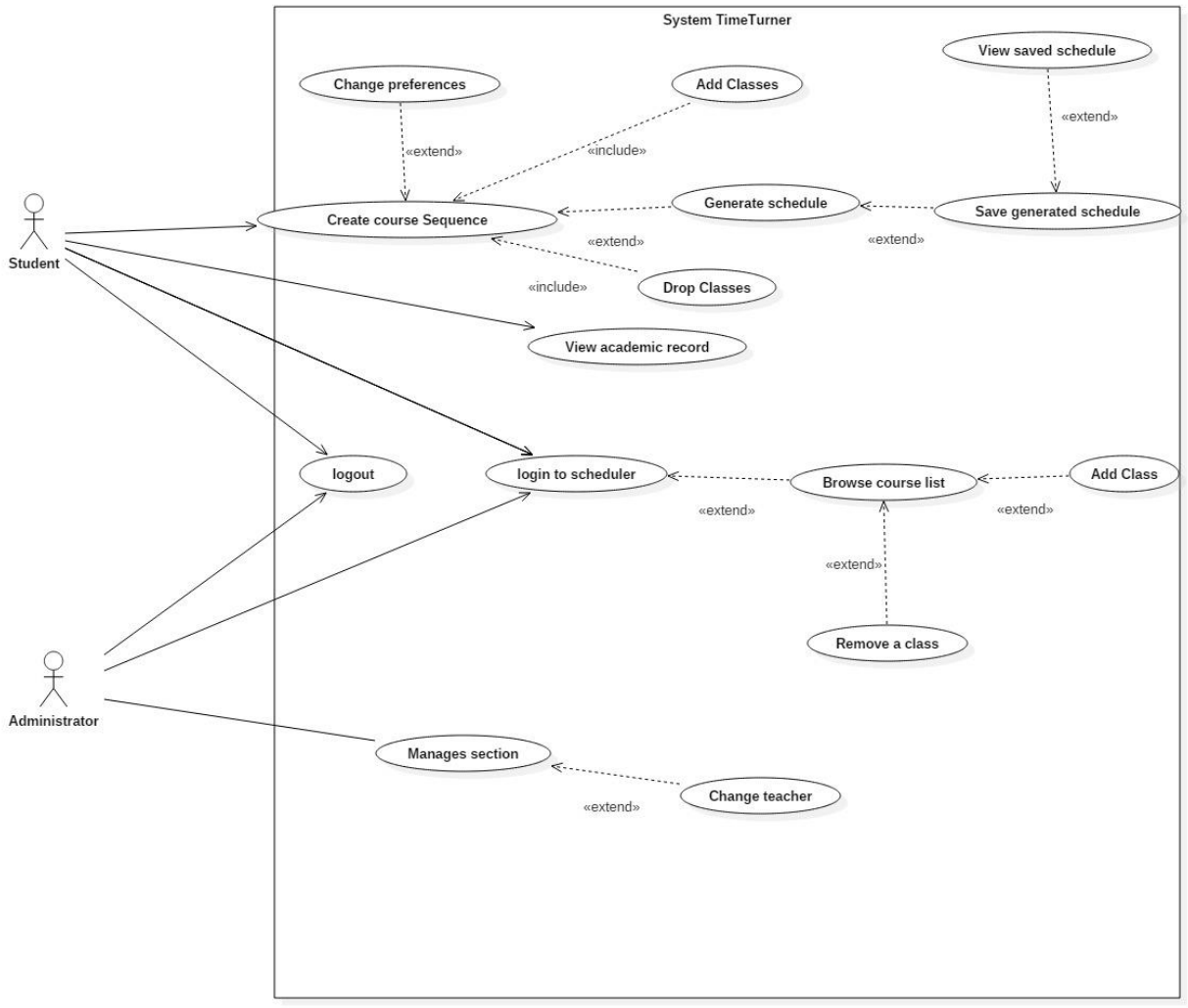


Figure 3.1.1 Use Case Diagram for use cases representing system requirements

3.1.25 Use Cases

Use Case ID:	UC1	
Use Case Name:	Login to TimeTurner	
Created By:	Marc-Andre Leclair	Last Updated By: Claudia Della Serra
Date Created:	January 25th, 2016	Last Revision Date: February 8, 2016
Actor(s):	Student, Administrator	
Goal/Actor Goals:	A student or Administrator wants to Login in their account	
Description/Summary:	The Student or Administrator wants to login into their account. The initial webpage contains a login box where the he or she can enter their username and password. The request is sent and it is check in the database whether or not the account exist or if the password is correct	
Preconditions:	❖ The user has an internet connection. ❖ The user has valid login credentials.	
Post-conditions:	The user is authenticated and logged in.	
Minimum Guarantee:	User will not log in	
Basic Flow:	1. Student enters its login information 2. System logs student into the system 3. Student is brought to the home page.	
Risk assessment:	Low	
Importance assessment:	5/5	

Use Case ID:	UC2	
Use Case Name:	Logout of Scheduler	
Created By:	Marc-Andre Leclair	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 17, 2016
Actor(s):	Student, Administrator	
Goal/Actor Goals:	A student or Administrator wants to Logout of their account.	
Description/Summary:	The Student or Administrator wants to logout of their account. He or she sends a request to the server to close the connection between themselves and the Time Turner.	
Preconditions:	The user is logged in their account	
Post-conditions:	The user is disconnected from the server and logged out.	
Minimum Guarantee:	User will remain logged in	
Basic Flow:	1. The user indicates that they wish to logout of the system. 2. The server terminates the connection. 3. The user is logged out through the webpage.	
Risk assessment:	Low	
Importance assessment:	5/5	

Use Case ID:	**UC3**	
Use Case Name:	Create Course Sequence	
Created By:	Marc-Andre Leclair	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	A student wants to create a course sequence	
Description/Summary:	The Student wants to create his or her course sequence. The system must provide the user with the option to add their own preferences in order to generate a sequence that is personalized to these preferences.	
Preconditions:	❖ The user is logged into the system ❖ The user is registered in a program.	
Post-conditions:	The user has a personalized course sequence for the future	
Minimum Guarantee:	No course sequence will be created	
Basic Flow:	1. Student requests to create a sequence. 2. The system must prompt the student to add his or her preferences to the sequence. 3. The student selects zero, one, or more preferences. 4. The system prompts the student to confirm their preferences 5. The student confirms his choices. 6. The course sequence is created and shown to the student.	
Risk assessment:	High	
Importance assessment:	5/5	

Use Case ID:	UC4	
Use Case Name:	Browse Course List	
Created By:	Ideawin-Bunthy Koun	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	Browse the list of courses from the course calendar.	
Description/Summary:	The user wishes to view available courses and access information pertaining to them. The system must display such information, including sections, times, and locations.	
Preconditions:	❖ User must be logged in as a Student. ❖ Course list must be available.	
Post-conditions:	A list of courses is displayed.	
Minimum Guarantee:	The system fails to display a list of courses and displays an error message to the user.	
Basic Flow:	1. Student selects the 'Browse Course List" feature. 2. System retrieves list of courses. 3. System displays list of courses. 4. User selects a course from that list. 5. System displays information about the selected course.	
Risk assessment:	Medium	
Importance assessment:	3/5	

Use Case ID:	**UC5**	
Use Case Name:	View Course Sequence	
Created By:	Ideawin Bunthy Koun	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	Allows the user to view their expected progression in the program.	
Description/Summary:	User can view a complete course sequence of the program the student is enrolled in. The Sequence will show the courses the Student is expected to take, and the order he or she is expected to take them in.	
Preconditions:	❖ User must be logged in as a Student ❖ The system has access to a course sequence.	
Post-conditions:	The system displays a course sequence.	
Minimum Guarantee:	The system fails to display a course sequence and displays an error message to the user.	
Basic Flow:	1. Student selects the option to view his or her course sequence. 2. System retrieves the sequence. 3. Course sequence is displayed to the student.	
Risk assessment:	Medium	
Importance assessment:	4/5	

Use Case ID:	UC6	
Use Case Name:	Generate Schedule	
Created By:	Ideawin-Bunthy Koun	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	Obtain a suggested schedule for the next four years.	
Description/Summary:	The system can generate a 4-year schedule for the user based on the student's preferences and constraints and on prerequisite and co-requisite policy.	
Preconditions:	❖ User must be logged in as a Student. ❖ The system must have access to course database. ❖ The system must have access to the student's records.	
Post-conditions:	The system generates a schedule with no overlaps.	
Minimum Guarantee:	The system fails to generate a schedule and displays an error message to the user.	
Basic Flow:	1. User selects the 'Generate Schedule' feature. 2. System retrieves list of preferences. 3. User selects his or her options and preferences. 4. System prompts user to confirm preferences. 5. System responds by displaying a schedule that meets the user's preferences and constraints.	
Risk assessment:	High	
Importance assessment:	5/5	

Use Case ID:	UC7	
Use Case Name:	View Saved Schedules	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to view a previously saved schedule.	
Description/Summary:	The user selects a schedule from a list of their previously saved schedules. The system must display this schedule for the user to review.	
Preconditions:	❖ User is logged into the system. ❖ User has previously generated one or more schedules ❖ User has saved at least one generated schedule	
Post-conditions:	The system displays the saved schedule	
Minimum Guarantee:	Saved schedules remain in the system unchanged.	
Basic Flow:	1. The user indicates their wish to view a saved schedule 2. The system prompts the user with a list of saved schedules 3. The user selects which semester's schedule to view 4. The system displays the chosen schedule	
Risk assessment:	Medium	
Importance assessment:	3/5	

Use Case ID:	UC8	
Use Case Name:	View Academic Record	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to view a list of his or her completed courses and grades for those courses.	
Description/Summary:	The system displays the user's academic record, which includes completed courses and courses currently being taken.	
Preconditions:	❖ User is logged into the system. ❖ User has completed at least one course and/or is presently enrolled in at least one course.	
Post-conditions:	The system displays the user's academic record.	
Minimum Guarantee:	The academic record remains in the system unmodified.	
Basic Flow:	1. The user requests to view their academic record 2. The system generates a list of the user's completed courses, as well as courses in which the user is currently enrolled.	
Risk assessment:	Low	
Importance assessment:	1/5	

Use Case ID:	UC9	
Use Case Name:	Drop Course	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to remove a course from a generated schedule.	
Description/Summary:	The user selects which course he or she would like to remove from the schedule. The system must delete this course from the schedule.	
Preconditions:	❖ User is logged into the system. ❖ User has generated a schedule containing at least one course. ❖ User has chosen to view a saved schedule	
Post-conditions:	The chosen course has been removed from the user's schedule.	
Minimum Guarantee:	The courses on the user's schedule remain unchanged.	
Basic Flow:	1. User selects a schedule they wish to view 2. System returns the desired Schedule 3. The user chooses a course from his or her schedule to view. 4. The system displays the course information 5. The user indicates they wish to remove the course from the schedule 6. The System prompts the user for confirmation. 7. The user confirms dropping of the course 8. The produces a modified schedule without the removed course.	
Risk assessment:	High	
Importance assessment:	4/5	

Use Case ID:	**UC10**	
Use Case Name:	Add Course	
Created By:	Lori Dalkin	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	Add a course to a generated schedule.	
Description/Summary:	Specific courses can be added to a schedule generated by a user. The user must search for the course and manually add it to their schedule.	
Preconditions:	❖ The user is logged in to their account ❖ A schedule has been generated	
Post-conditions:	The specified course is added to the schedule.	
Minimum Guarantee:	The user's schedule remains unchanged and no new course is added	
Basic Flow:	1. User requests to view schedule. 2. System displays the indicated schedule. 3. User indicates they wish to add a course 4. The system displays a search screen prompting the user to enter the name of the course 5. The user types in the name of the course they wish to add 6. The system displays a list of corresponding courses and sections 7. User selects the specific section of the course they wish to add. 8. The system prompts the user to confirm their selected course 9. The user confirms their wish to add the course 10. The system returns a modified schedule.	
Risk assessment:	Medium	
Importance assessment:	3/5	

Use Case ID:	UC11	
Use Case Name:	Save Generated Schedule	
Created By:	Lori Dalkin	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to save a schedule they have generated.	
Description/Summary:	A generated schedule can be saved in order to be accessed and viewed later on by the user. This schedule consists of courses the user is satisfied with and wishes to keep on their schedule.	
Preconditions:	❖ The user is logged on to their account ❖ A schedule has been generated containing at least one course.	
Post-conditions:	The system saves the generated schedule to the user's account.	
Minimum Guarantee:	The user's account remains unchanged.	
Basic Flow:	1. User indicated their wish to generate a schedule 2. The system prompts the user with a list of semesters for which they wish to generate a schedule 3. The user indicates which semester they wish to generate a schedule for 4. The system displays a generated schedule for the indicated semester 5. The user will indicates that they want to save the generated schedule. 6. The system will display a message telling the user that the schedule has been saved.	
Risk assessment:	Medium	
Importance:	4/5	

Use Case ID:	**UC12**	
Use Case Name:	View Weekly Schedule	
Created By:	Bryce Drewery Schoeler	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	A student wants to view weekly schedule.	
Description/Summary:	The student wishes to see their class schedule for the current week. The student will specify a week to be displayed. The system must display the schedule for that week.	
Preconditions:	❖ The student has been authenticated ❖ A schedule has been generated and saved	
Post-conditions:	The schedule is shown on screen.	
Minimum Guarantee:	The schedule fails to be displayed.	
Basic Flow:	1. Student requests to see a saved schedule. 2. The system prompts the student to select a week. 3. Student selects a week 4. The systems display the schedule for that week	
Risk assessment:	Medium	
Importance assessment:	3/5	

Use Case ID:	UC13	
Use Case Name:	Update Course	
Created By:	Aline Koftikian	Last Updated By: Claudia Della Serra
Date Created:	Marc 25th, 2016	Last Revision Date: April 7, 2016
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	An administrator wants to modify available courses	
Description/Summary:	The Administrator is given access to the course bank and can search for a specific course, view course I.D., edit course details and/or delete course.	
Preconditions:	❖ The user is authenticated ❖ There is at least one course in the course bank	
Post-conditions:	A course has been modified	
Minimum Guarantee:	No course will be modified	
Basic Flow:	1. Administrator requests to manage courses 2. System displays course bank 3. Administrator chooses a course to modify and what modification they would like to make 4. System prompts the user with a form to make the desired changes 5. Administrator confirms change 6. System displays the updated course bank	
Risk assessment:	4/5	
Importance assessment:	3/5	

Use Case ID:	UC14	
Use Case Name:	View A Course	
Created By:	Aline Koftikian	Last Updated By: Claudia Della Serra
Date Created:	April 10th, 2016	Last Revision Date: April 11, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	An administrator wants to view a course	
Description/Summary:	The Administrator wishes to view information about the course. The Administrator is given course details including course I.D., course code, and course description.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ There is at least one course in the course bank	
Post-conditions:	❖ Course details will be shown	
Minimum Guarantee:	The user will remain on the same page	
Basic Flow:	1. Administrator requests to manage courses 2. System displays the course bank 3. Administrator chooses a course to view 4. The system displays the course details on the page	
Risk assessment:	1/5	
Importance assessment:	3/5	

Use Case ID:	UC15	
Use Case Name:	View A Course	
Created By:	Aline Koftikian	Last Updated By: Claudia Della Serra
Date Created:	April 10th, 2016	Last Revision Date: April 11, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	An administrator wants to update a course	
Description/Summary:	The Administrator wishes to update information about the course. The Administrator is given course details to update including course I.D., course code, and course description.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ There is at least one course in the course bank	
Post-conditions:	❖ Course details will be shown	
Minimum Guarantee:	The user will remain on the same page	
Basic Flow:	1. Administrator chooses a course to update 2. System displays update fields that can be edited 3. Administrator inputs necessary changes 4. Administrator confirms change 5. System displays updated course bank	
Risk assessment:	1/5	
Importance assessment:	3/5	

Use Case ID:	UC16	
Use Case Name:	Delete A Course	
Created By:	Aline Koftikian	Last Updated By: Claudia Della Serra
Date Created:	April 10th, 2016	Last Revision Date: April 11, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	An administrator wants to delete a course	
Description/Summary:	The Administrator wishes to delete and entire course. The Administrator deletes a course permanently from the course bank.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ There is at least one course in the course bank	
Post-conditions:	❖ The course will no longer exist in the course bank	
Minimum Guarantee:	All courses in the course bank will remain unchanged	
Basic Flow:	1. Administrator requests to manage courses 2. System displays the course bank 3. Administrator chooses a course to delete and confirms deletion 4. System displays updated course bank which does not contain the course	
Risk assessment:	4/5	
Importance assessment:	3/5	

Use Case ID:	**UC17**	
Use Case Name:	Create Course	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	Marc 26th, 2016	Last Revision Date: April 8, 2016
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	Administrator wishes to add a course to the database of courses	
Description/Summary:	The Administrator wishes to add a course to the system's database of courses by entering its course code, the course description, and the amount of credits awarded by the course. A course may be a section, a tutorial, or a lab.	
Preconditions:	❖ The administrator has logged in. ❖ The administrator has viewed the list of courses in the database.	
Post-conditions:	The course is added to the database of courses	
Minimum Guarantee:	The database of courses will remain unaltered.	
Basic Flow:	1. Administrator requests to create a course 2. The system presents a form asking for the course id, the course description, and the amount of credits awarded by the course. 3. The Administrator enters the course information and presses submit 4. The system presents the course information, with the options to manage the course or view all courses in the database	
Risk assessment:	3/5	
Importance assessment:	5/5	

Use Case ID:	UC18	
Use Case Name:	Browse Courses	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	Marc 26th, 2016	Last Revision Date: April 8, 2016
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	Administrator wishes to view the database of courses	
Description/Summary:	The Administrator wishes to add view the system's database of courses. From here, the administrator may choose to add a course or manage courses.	
Preconditions:	❖ The administrator has logged in.	
Post-conditions:	The course database will be presented	
Minimum Guarantee:	The administrator will remain on the home page.	
Basic Flow:	1. Administrator requests to view the course database 2. The system presents a page listing all the courses in the database	
Risk assessment:	2/5	
Importance assessment:	4/5	

Use Case ID:	**UC19**	
Use Case Name:	Add User	
Created By:	Ideawin-Bunthy Koun	Last Updated By: Ideawin-Bunthy Koun
Date Created:	April 2nd, 2016	Last Revision Date: April 2nd, 2016
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	Administrator wishes to add a new user to the database of users.	
Description/Summary:	The Administrator wishes to add a user to the system's database of users by entering a username, first name, last name, net name, password and select the user's privilege as a student or an administrator.	
Preconditions:	❖ The Administrator has logged in. ❖ The Administrator has viewed the list of users in the database.	
Post-conditions:	The user database will be presented	
Minimum Guarantee:	The database of users remains unaltered.	
Basic Flow:	1. Administrator requests to add a user. 2. The system presents a page asking for the username, first, last and net names, password and an option to select the privilege. 3. The Administrator enters the course information and presses submit 4. The system displays the new user's information such as ID, username, first, last and net names.	
Risk assessment:	2/5	
Importance assessment:	4/5	

Use Case ID:	UC20	
Use Case Name:	Search for Users	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	Obtain access to data related to a user through search criteria.	
Description/Summary:	An administrator wishes to gain access to information related to a user through search criteria such as the user's ID, username, first name, last name and net name.	
Preconditions:	❖ User must be logged in as an Administrator.	
Post-conditions:	❖ A list of student(s) that fits the search criteria is created. ❖ The list is displayed as being editable to the administrator.	
Minimum Guarantee:	The system does not find any users matching the specified criteria and indicates that no results have been found.	
Basic Flow:	1. Administrator specifies the search criteria in available search fields. 2. System displays a list of editable users matching the search criteria indicated by the administrator.	
Risk assessment:	3/5	
Importance assessment:	2/5	

Use Case ID:	UC21	
Use Case Name:	View User	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	View information stored in the system about a user.	
Description/Summary:	An administrator wishes to view the profile summary that contains information about a user such as its ID, username, first name, last name, and net name.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ A user search result is displayed. ❖ The user the administrator is looking for is listed in the search result.	
Post-conditions:	❖ The administrator has access to consult information about the user	
Minimum Guarantee:	The set of information field is displayed but may contain empty fields.	
Basic Flow:	1. Administrator indicates that the information about a particular user displayed on a search result is to be consulted. 2. System retrieves information about the user and displays it in a table.	
Risk assessment:	3/5	
Importance assessment:	3/5	

Use Case ID:	UC22	
Use Case Name:	Update User	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	Update information related to a user.	
Description/Summary:	An administrator wishes to update information saved about a user. Those include the username, first name, last name, net name, password and privilege.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ A user search result is displayed. ❖ The user the administrator wishes to update is listed in the search result.	
Post-conditions:	❖ Information about the user is updated. ❖ Updated information is displayed to the administrator.	
Minimum Guarantee:	The user's profile fails to be updated and an error is displayed.	
Basic Flow:	1. Administrator indicates that the information about a specific user must be updated. 2. System displays the update fields filled with the current information about the selected user. 3. Administrator changes and saves the new information about the user. 4. System displays the updated profile of the user.	
Risk assessment:	3/5	
Importance assessment:	3/5	

Use Case ID:	UC23	
Use Case Name:	Delete User	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 9, 2016	Last Revision Date: April 9, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	Delete a user from the system.	
Description/Summary:	An administrator wishes to delete a user (its account, and all related information) from the system.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ A user search result is displayed. ❖ The user that the administrator wishes to delete is part of the displayed search result.	
Post-conditions:	❖ Targeted user is deleted. ❖ Updated table of users is generated. ❖ Updated table of users is displayed to the administrator.	
Minimum Guarantee:	The administrator cancels the deletion and the user remains on the system.	
Basic Flow:	1. Administrator indicates that a user must be deleted. 2. System displays a message to confirm the deletion of the user. 3. Administrator confirms the deletion. 4. System displays an updated table of users to the administrator.	
Risk assessment:	3/5	
Importance assessment:	3/5	

3.2 Domain Model

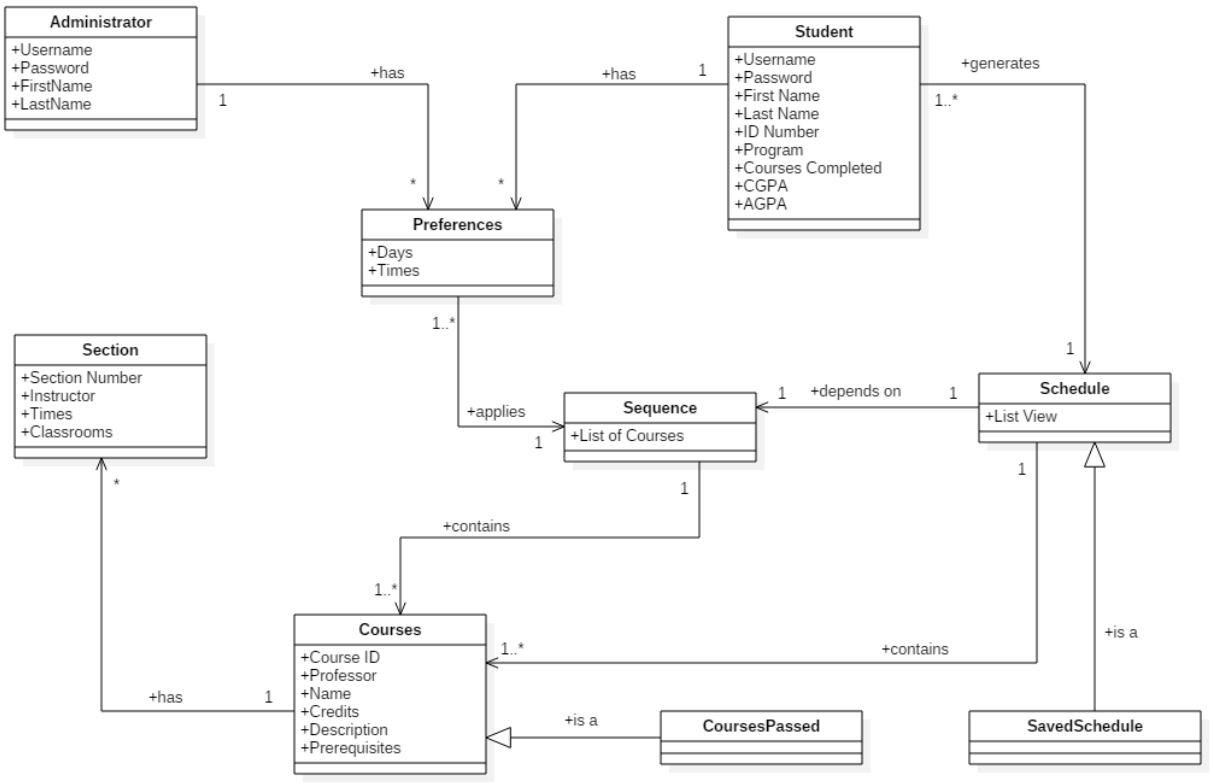


Figure 3.2.1. Domain Model

3.2.1 DLO 1 - Administrator

The administrator is one of the two types of users of the Scheduler application. The administrator is a privileged user as they will have higher access rights than the other type of user, which is the student. The administrator will have the rights to set/edit the capacity of a course section, to empty a course section, and in general manages a course section. The administrator is typically expected to be a course advisor or program coordinator of sorts. A single administrator has rights to as many sections as they want. This object has an ID number as an attribute, which serves as a unique identifier of the Administrator.

3.2.2 DLO 2 - Student

The student is the second type of user that will access the Scheduler application. The student is not a privileged user like the admin because they have limited access rights to the Scheduler. The student has preferences which will be what the user prefers their schedule to look like, and has access to the Scheduler object. The student has no direct access to the course sections. This object has the attributes of an ID number which is the Student's unique identifier, a Program which is expected to be Software Engineering or Computer Science, Courses Complete which is the courses that the individual Student has completed, a

CGPA which is the cumulative GPA from the start of the Student's studies, and a AGPA which is the Student's total GPA from the latest year.

3.2.3 DLO 3 Courses

The course is the class that can be taken by a student. A student's course sequence is composed of courses, as well as the schedule. Each course has at least one section, which is what a student picks when choosing a specific course group. The course object has several important attributes. Course ID is the specific identification number associated with the course in question. Professor is the assigned professor's identification. Name is the course name, Credits represents the number of credits that course presents, description is a summary of the course contents, and prerequisites are a list of courses that are prerequisites to the course, if any.

3.2.4 DLO 4 – Courses Passed

The courses passed object is an extension of the courses object, and it represents a course that a student has taken and completed, meaning that the student has received full credit for the course. It inherits the attributes of the course object.

3.2.5 DLO 5 - Section

The section of a course, managed by an administrator, is an entity representing a time slot at which students can attend a course given by an instructor in a certain classroom. There can therefore be multiple sections for the same course. A section holds as information its section number, the full name of the instructor teaching the course, the times and the classroom numbers at which the lectures, laboratories and tutorials are held.

3.2.6 DLO 6 - Preferences

The preferences of a student are filters to be applied when the course sequence of that student is generated. Those preferences include the times in a day at which the student prefers to have classes and a set of days on which the student wishes to have less classes. At least one preference is to be applied when the sequence is generated.

3.2.7 DLO 7 – Sequence

The sequence is the list of courses that the student must take. The sequence may either be the static sequence that every SOEN student takes, or an adapted sequence in case the student deviates from the original sequence. The sequence is generated by the scheduler based on the preferences that are set by the student.

3.2.8 DLO 8 – Schedule

The schedule is generated by the scheduler. It is comprised of several courses throughout the week. It can be viewed by a list view where all courses are shown one after another, or it can be shown on a weekly basis where the times of each class are represented visually on a table.

3.2.9 DLO 9: Saved Schedule

The saved schedules object is an extension of the schedule object. It represents the schedules that the user, a student, has saved under their account. It inherits the attributes that are presented in the schedule object of the model, mainly the list view attribute.

3.3 Constraints and Qualities

3.3.1 Security

Security is a crucial aspect of any software. One of the most important security requirements is to make sure there are no SQL injections that could breach and break our database. In order to assist in preventing SQL injections, we will be using prepared statements prior to query executing which is provided by PHP's internal libraries. The possibility of having the database breached could potentially expose user personal user information or perhaps destroy content. The exposure of passwords is a concern in case of breach as they should be hashed and salted upon entry. This would essentially diminish any chance hackers have to brute force password entries.

3.3.2 Performance

Performance is important especially when working with large data sets. The user must be able to navigate and use the site without having to wait for data generation. The application shall also ensure that database design is robust with the use of proper table indexing and storage engines and cache sizes. This will essentially allow for quicker SQL query executions that are expected to return data within 100 nanoseconds. The performance shall also account for larger data execution, such as the schedule generator with a performance factor speed of no more than 5 seconds.

3.3.3 Cross-Browser Compatibility

There are cases where web applications do not necessarily comply with all possible web browsers. For example, certain aspects of CSS and HTML might work for one browser but break in another. This application shall ensure that the CSS and HTML model will be compliant with the browser's boundaries. Ensuring browser compliance, the application shall be able to run on virtually any operating system that has any of the following browsers: Safari, Opera, Internet Explorer, Firefox and Chrome.

3.3.4 Ease of Use

The application shall be straightforward and lightweight in order to ensure that students can start working on their schedule planner immediately. The use of Ajax technology will also minimize content and page refreshes thereby allowing users to navigate without losing the scope of the data they were working on.

4. Resource Evaluation

4.1 Human Resources

Our team of Undergraduate Students is a strong team of 12 individuals with varying skills and strengths. Our team has a diverse background, both socially and in terms of past work experiences. We also possess a good gender ratio of 7 males and 5 females. With both knowledgeable leaders and dedicated team players, we work efficiently together while also complementing each other's strengths and weaknesses. Each member of our team brings their own outlook towards the project that will ultimately create a well-rounded project that can fit the needs of many. Below are the names of the 12 members along with the following information:

- The different skills and knowledge of languages they possess.
- Their experiences with past occupations, school projects and personal projects. Other relevant experiences that can add insight towards the project.
- Their primary role during the project.
- Their strengths that will help towards the success of the project.
- Their approximate availability per week, including both group meetings and personal work time.

Name	**Daniel Di Corpo**
Skills	C++, JAVA, PROLOGUE, LISP, PHP, HTML
Experience	Technical DEC in Comp. Sci., Image Programing Specialist at Matrox
Role	Project Team Lead, System Architecture and Distribution of Tasks
Strengths	Leadership, Team Player, Management skills, Communication
Availability	7 hours/week

Name	**Dimitri Topaloglou**
Skills	PHP, CSS, C++, C#, HTML, SQL, JAVASCRIPT, JQUERY, JAVA, PROLOGUE, CLOS, ASP.NET
Experience	Seven years of experience in web application development.
Role	Coding team Technical Lead, system designer, frontend + backend developer
Strengths	Advanced coder, quick learner, opened to new ideas
Availability	7 hours/week

Name	Claudia Della Serra
Skills	HTML, CSS, JAVA, JAVASCRIPT, PROLOGUE, LISP, CLOS, ASPECTJ, PYTHON
Experience	Developed 2 Android applications, certified tutor in English reading and writing, multiple years of experience in document editing.
Role	Documentation Team Lead, writing, editing and consolidation of documents
Strengths	Quick learner, team player, leadership and management skills
Availability	7 hours/week

Name	Aline Koftikian
Skills	JAVA, HTML5, CSS3, JAVASCRIPT, C#, PROLOG, ASPECTJ, COMMON LISP, PYTHON
Experience	Developed two mobile (Android) applications w/ teams.
Role	Documentation team member, in charge of diagrams + logo
Strengths	Team player, open minded, organized
Availability	7 hours/week

Name	Kevin Yasmine
Skills	JAVA, HTML5, CSS3, JAVASCRIPT, PROLOG, ASPECTJ, COMMON LISP, CLOS, Photoshop/Paint.Net
Experience	Class projects/assignments, extracurricular Android App Development using Android Studio.
Role	Documentation team member
Strengths	Team player, ability to see the big picture in large projects
Availability	7 hours/week

Name	Ideawin-Bunthy Koun
Skills	JAVA, HTML5, CSS3, PHP, JAVASCRIPT, PROLOG, ASPECTJ, COMMON LISP
Experience	Academic projects/assignments, attended two hackathons where Android Studio was used, developed Android applications.
Role	Coding team member as a Front-End developer
Strengths	Fast learner, eager to learn new technologies, team player
Availability	7 hours/week

Name	Philip Lim
Skills	JAVA, HTML5, CSS3, JAVASCRIPT, PHP, MYSQL, PROLOG, COMMON LISP
Experience	Completed an internship as a programmer analyst: involved Java programming using IntelliJ IDEA and documenting the code Developed an apartment renting website (academic project) Completed various academic assignments/projects in Java
Role	Documentation team member
Strengths	Flexible, active listener, team player
Availability	7 hours/week

Name	Ryan Lee
Skills	JAVA, HTML, CSS, JAVASCRIPT, PHP, PROLOG, LISP, C#, PYTHON, RUBY
Experience	Competed in 2 hackathon and made applications using java and android studio, implementing APIs provided by companies.
Role	Documentation team member
Strengths	Team player, Communicative abilities, problem solving
Availability	7 hours/week

Name	Lori Dalkin
Skills	PHP, CSS, HTML, JAVASCRIPT, JAVA, PROLOGUE, CLOS, ASPECTJ, PYTHON(basic), C#(basic)
Experience	Attended two hackathons where two android apps using java and android studio were developed.
Role	Coding team member. Back end and front end developer.
Strengths	Quick learner, enthusiasm for learning new languages, creative.
Availability	7 hours/week

Name	Bryce Drewery Schoeler
Skills	PHP, CSS, HTML, JAVASCRIPT, JAVA, PROLOGUE, ASPECTJ, PYTHON, C++, C#
Experience	Developed android apps at hackathons.
Role	Coding team member. Back end and database developer.
Strengths	Enthusiastic learner, Good grasp of theoretical computer science. Excellent at solving logical problems.
Availability	7 hours/week

Name	Erin Benderoff
Skills	PHP, CSS, HTML, JAVASCRIPT, JAVA, PROLOG, CLOS, ASPECTJ, PYTHON (beginner), MySQL
Experience	Made two Android applications at hackathons
Role	Coding team member. Mainly front end but will also do a bit of back end.
Strengths	Detail-oriented, motivated, fast learner, easy to get along with
Availability	7 hours/week

Name	Marc-Andre Leclair
Skills	PHP, CSS, HTML, JAVASCRIPT, JAVA, PROLOG, RUBY, C#, Visual Basic.net, MySQL
Experience	Created two android app at two different hackathons
Role	Coding team member. Mostly back end attending to php and the database but also some front-end
Strengths	Love to learn, strong team player, fast learner
Availability	8 hours/week

4.2 Technical Resources

Each developer's computer must have the same XAMPP version installed in order to align the Apache and PHP version as found in the production environment. This is to avoid any conflicts and bugs which can be caused due to incompatible versions of PHP or Apache's server core files.
XAMPP (64-bit environment) includes the following pre-installed software packages:

- Database: MySQL 5
- Web Server: Apache HTTP Server 2.0
- Primary language: PHP 6.0 and Python 2.7
- Database User Interface: phpMyAdmin

The production environment in question has the following infrastructure:

- CPU: Intel i7
- OS: Windows Server 2012
- RAM: 16GB
- DISK DRIVE: 500 GB (Solid State)
- Network Speed: 10 Mbps

The production environment is hosted on a team member's own computer. This decision was made in order to have a better overlook of the server and have complete control of the environment.

Main software packages used in the project include:

- Languages: HTML, CSS, PHP, JQuery, JavaScript, SQL
- IDE: PhpStorm 10
- Framework: Yii 1.1.16
- Version Control Systems: GitHub, Gitbash, GitHub Desktop
- Team Collaboration Tools: GitHub, Facebook Chat, Trello
- OS: Windows, Linux
- UML and Diagrams: StarUML
- Documentation tools: Google Drive, Microsoft Word
- Graphical Design Tool: Photoshop

5. Scoping

When the Project was first announced and the groups were formed, the team spent a lot of time discussing what the Scheduler should include. Many features came up, such as administrative privileges, manual manipulation of the sequence, etc. However, after working on the plan and blueprints of the project, it became clear that there are many requirements and constraints that would limit the additional features that we wanted to include.

5.1 Limitations

Taking into consideration section 3 and 4, our biggest limitation is time. Some features would be an asset to have, but given that the project should be developed in only a few weeks, it is impossible to include them all while maintaining Security, Performance, Cross-Browser Compatibility and Ease of Use (see section 3.3).

5.2 Scoped Out

5.2.1 Reimplemnting of Administrative Privileges

~~Initially, the team wanted to include Teachers and Administrators that are able to do different tasks, such as change class time, drop students, add students, change teachers, bypass prerequisites, etc., but given our time constraint and our limited experience as students, this aspect was scoped out. Such activities include~~

- ~~Managing a Section~~
- ~~Adding a Course to the Course List~~
- ~~Changing the Time of a Course~~
- ~~Etc.~~

A final decision to implement administrative capabilities was taken, in order to include the vital functionality of having a super user to control all other functionalities of the system. The administrator can perform some unique actions, such as manipulate users' accounts and courses in the system database. This was important to include, as new users and new courses must be able to be added, and users and courses must be able to be removed by anyone who may decide to implement this system as their scheduling system.

5.2.2 Manual Manipulation of Sequence

Another feature that was scoped out, is to allow students do add and drop individual classes. The team realized that the main function of the Scheduler is to automatically create a full, four-year schedule. We decided to optimize this function so that it runs well, instead of giving multiple – less efficient – ways of adding classes.

5.2.3 Creation of Sequence

This feature was scoped out due to time constraints and difficulty of implementation. If all users used the same hard coded sequence as a suggestion of what to follow, this provides a much easier way to implement the sequence. This is the implementation choice that we made.

5.2.4 Manually Adding a Class to a Saved Schedule

Being able to add a class to a schedule is an important functionality in any scheduling system. However, due to the time constraints and difficulties experienced in testing, the option to add courses to a schedule that has already been saved has been scoped out of the final version of this project. With more time, this functionality may have been added in, and is a good addition to any future versions of this system.

5.2.5 Weekly View of Schedules

The weekly view of schedules has been removed from the requirements of this project due to difficult integration of the scheduling widget into the framework.

6. Solution Sketch

6.1 Architecture

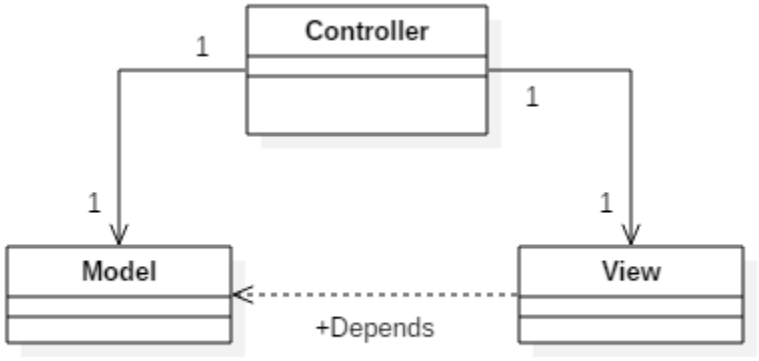


Figure 6.1.1 Model View Controller Architecture model

Time Turner's architecture will follow a Model-View-Controller architecture (MVC). This MVC architecture is best suited for this project because it isolates our application to 3 different layers, data, logic and presentation. With these clear separation of concerns, changing any of these components might not affect the other two components. This is a preferred for team based environments since all 3 components can be coded at the same time.

We have chosen to use the Yii Framework for its ease of use, modular components and its exceptional handling of applications with big traffic flow such as portals and content management systems. We have chosen this framework for its high speed of development and work, integration with Javascript, can handle multiple types of databases and convenient and flexible caching system which will help for caching validation.

A general work flow starts off with a user request. This is handles by the controller, which will separate what needs to be sent to the model component and the view component. Anything that needs data retrieval or sequence generation, will be sent to the model component, which is in conjunction with our domain model. Once the data has been retrieved, it is sent back to the controller to be packaged and sent to the viewer. Once the view component has all the information, it will generate, what will be, in the end, the display on the user's web browser. Then, once a new request is sent, the process will start over again.

6.1.1 Controller

This is the data of the MVC. With a web application, the user will make a request for a web page which will be handled by the controller. The controller will evaluate the request, sending any logic operations to the model component. Once the controller has collected all of the necessary information, it will send all its data to the view component, which will generate an HTML output to the user.

6.1.2 Model

This is the logic of the MVC. With the user information given from the controller, and with the use of the underlying logical data structure, such as our domain model, processes such as, generate sequence, can be utilized and sent back to the controller. One controller does not have exclusivity over the model component, since multiple pages may need the same information, many controllers can access the model component.

6.1.3 View

This is the presentation of the MVC. With the proper session handling from the controller and relevant data from the modeller, the viewer can generate an HTML output with CSS and Javascript, which will, in-turn, be viewed on the user's web browser.

6.2 Technologies in Use

6.2.1 Programming Languages

6.2.1.1 Server-side

6.2.1.1.1 PHP

PHP is the main server-side language that will be handling business logic and SQL database queries. PHP was chosen due to the syntax similarities of other major Object-Oriented programming languages, allowing for a quicker learning experience.

6.2.1.2 Client Side

6.2.1.2.1 JavaScript

JavaScript a multi-paradigm scripting language that is supported and natively embedded by all major browsers including Internet Explorer, Firefox, Safari and Chrome. It is mainly used for dynamic client side manipulation of the DOM and allows for a dynamic browsing experience.

6.2.1.2.2 jQuery

jQuery is essentially a JavaScript script library that allows programmers to write less lines of code in order to perform the same functionality in the native JavaScript environment. It allows for quick DOM manipulation with very few lines of code, animations, client side validation, form input handling and quick Ajax calls.

6.2.1.2.3 HTML (Hyper Text Markup Language)

HTML is the primary markup language used to create web layout and provide structure to web content. Of course, it is also natively supported in every major web browser.

6.2.1.2.4 CSS (Cascading Style Sheet)

CSS is used primarily for styling purposes that is supported by all major browsers with some detailed exceptions in portability. The essence of CSS is to separate styling from the markup language, such as HTML.

6.2.2 IDEs

6.2.2.1 PhpStorm 10

PhpStorm is a powerful IDE provided by JetBrains that is integrated with an extensive Git support system and provides direct database management without the use of any other third party application.

6.2.3 Database Management System

6.2.3.1 MySQL

MySQL is an open-source database management system that provides more than enough features required to for the scope of the web application. It also provides InnoDB storage engine access which supports transactions, enabling to undo database query commits.

6.2.4 Web Server

6.2.4.1 XAMPP (v1.8.1)

XAMPP 1.8.1 is a free open-source, cross-platform web server solution package which includes Apache HTTP Server, MySQL and PHP functionalities.

6.2.5 Source Code Revision Management

6.2.5.1 GitHub

GitHub is a popular source code revision management system that allows for developers to clone source code repositories and provide a non-destructive means of working in isolation from the original repository. In turn, this provides a better overview of which developer is working on which portion of the code, thereby providing a flexible version control workflow.

6.2.6 Deployment Software

6.2.6.1 DeployHQ

DeployHQ is an online service that allows automatic and manual deployment of sources code from any type of repository service such as GitHub. It is used to automatically deploy source code directly to the main hosting server from a selected GitHub branch of your choice in a seamless manner. The service is installed via a webhook service, integrated within GitHub and source code deployment occurs upon a commit done in the master branch. It also provides rollback options.

6.2.7 Team Collaboration

6.2.7.1 Trello

Trello is a free team collaboration tool that organizes task into board spaces. It provides an overview of what tasks need to be done, tracks due dates and assigns tasks to collaborators. It also provides a mobile app version which allows to push notifications to users' mobile when the project board was updated, allowing collaborators to respond to tasks accordingly.

6.2.8 Framework

6.2.8.1 Yii

Yii is a highly extensible, open source framework designed for web applications and is based on an MVC architecture. It shortens development time with the use of Gii, an incorporated tool that quickly creates on-the-fly templates for models, controllers, forms, modules, extensions and CRUD control actions and views. This allows developers to minimize repetitive tasks and enforces the MVC architecture according to Yii's specifications. It also comes with a pre-built web template to allow unfamiliar developers to get a quick start in learning how to properly create models, views and controllers.

7. Plan

7.1 Activities

Activity	Purpose/Description	Artifact(s) produced
7.1.1 Form and divide team	Forming a team allows the project to start. Dividing the team into a coding and documentation sub-teams allows members to focus on particular tasks. Such division requires the evaluation of the strengths and weaknesses of all members to determine their role.	1.4.1 Human Resources
7.1.2 Complete the system overview	The system overview consists of finding a name for the system, determining the main purposes and elements involved in it, and drawing a domain model of the system along with the description of each entity found in the domain model.	Deliverable 0 1.3.2 Domain Model
7.1.3 Write a project description	The project description introduces the entire document. It informs the reader about its goals, the information to be found in it, as well as the purposes and objectives of the project.	1.2 Project Description
7.1.4 Determine the functional requirements and draw use case diagrams accordingly	Determine the requirements that the system should have, including those that may later be scoped out, their relative importance and relative difficulty. Define the actions that must take place in the software and illustrate each requirement by the mean of a use case diagram.	1.3.1 Functional Requirements
7.1.5 Review and correct domain model	Make any necessary changes according to the feedbacks of Deliverable 0.	1.3.2 Domain Model
7.1.6 Find and describe constraints and qualities	Describe any design constraints, qualities and non-functional requirements that the system should meet.	1.3.3 Constraints and Qualities

7.1.7 Review and format human resources document	After the profile of each team member has been determined, the document should be revised and formatted correctly.	1.4.1 Human Resources
7.1.8 Determine technical resources	List the computer resources and tools available to complete the project.	1.4.2 Technical Resources
7.1.9 Reduce the amount of features to be realistic according to time and resources	Outline the scope of the software. List all features, goals and qualities that are scoped out, and provide an explanation for each element that is scoped out.	1.5 Scoping
7.1.10 Choose the architecture of the system	Explain the high-level architecture of the system, and give a design rationale. A diagram should accompany the explanation.	1.6.1 Architecture
7.1.11 Decide which technologies are relevant to the project development	List technologies that will be used in the project and give a rationale for each of them.	1.6.2 Technologies in Use
7.1.12 List and describe the main activities involved in the project	List all activities that produce at least one artifact and include a description for each activity.	1.7.1 Activities
7.1.13 List all artifacts	List and describe all artifacts that need to be produced for the project.	1.7.2 Artifacts
7.1.14 Estimate the cost and time of production	Estimate the cost and time of production of each artifact and sum them up.	1.7.3 Project Estimates
7.1.15 Delegate activities to different team members	Assign each activities of the project to a team member or to a group of team members.	1.7.4 Activities Assignments

7.1.16 Produce a schedule for the project	Create a schedule/timeline of all activities for the project using a Gantt chart. Start dates, due dates, names of activities, and participants are all information to be included in the schedule.	1.7.5 Schedule
7.1.17 Determine the risks of the project	Outline and explain the risks associated and presented by this project.	1.7.6 Risk
7.1.18 Produce a prototype report	Describe the work done in the development of the prototype. Explain how the chosen technologies are appropriate, and how the team members are comfortable using such technologies.	1.8 Prototyping
7.1.19 Ensure professionalism of the document	Assemble all parts of deliverable 1. Ensure the document is professional looking and well-organized.	Deliverable 1
7.1.20 Update the introduction for Deliverable 2	Update the introduction to give the reader an overview of the content found in Deliverable 2: Architecture and Design	2.2 Introduction to Part 2
7.1.21 Produce an architecture diagram	Produce a 4+1 Architectural View. Include a rationale for the design. Explain any differences between the old design and this one.	2.3.1 Architecture Diagram
7.1.22 Specify the interactions between the components of the software	Describe the interactions between the components of the software through their interfaces. Include the function calls, the description of the parameters and the range of accepted values of those parameters.	2.3.2 Subsystem Interfaces Specifications
7.1.23 Illustrate the internal structure of the system	Provide a graphical representation of the structure of each subsystem by means of a UML class diagram. A description of each class should be included.	2.4.1 Detailed Design Diagram
7.1.24 Describe each class in the subsystem	Provide the programmers with the descriptions of each class in the UML diagram along with any necessary detailed relevant to the development.	2.4.2 Unit Description
7.1.25 Produce dynamic design scenarios	Draw the dynamic design of two use cases.	2.5 Dynamic Design Scenario

7.1.26 Update project estimates	Update the cost of the project for each module and include the cost of integration, testing and documentation.	2.6 Estimation
7.1.27 Produce a report about the prototype and risk	List and describe all elements that have been implemented and describe the effect that those implementations had on the design decisions, risks, estimate and scope.	2.7 Rapid Prototyping and Risk
7.1.28 Ensure professionalism of the document	Assemble all parts of deliverable 2. Ensure the document is professional looking and well-organized.	Deliverable 2
7.1.29 Update introduction	Update the introduction so that it includes an overview of the content presented in deliverable 3.	3.2 Introduction
7.1.30 List all tested items	Create a list of all items that have been tested, the test cases that were used, and a rationale for the test.	3.3.1.1 Tested Items
7.1.31 List all untested items	Create a list of items that are to be tested, an explanation of why those items should be tested, and how they could be tested.	3.3.1.2 Untested Items
7.1.32 Unit testing report	Describe the test cases, the technique used, the code used and the result of the testing for two units.	3.3.2.1 Unit Testing
7.1.33 Requirements testing report	Provide a list of test cases for all tested requirements by means of scenario of system usage and system reaction.	3.3.2.2 Requirements Testing
7.1.34 Stress testing report	Describe situations of extreme system usage, the tests designed to evaluate the performance of the system under such condition, and the test results.	3.3.2.3 Stress Testing
7.1.35 Security testing report	Describe tests performed to protect the system against security threats such as SQL injections and automated tools.	3.3.2.4 Security Testing
7.1.36 Installation manual redaction	Guide any administrators to install and execute the software.	3.4.1 Installation Manual
7.1.37 User's manual redaction	Guide any users to use the system with all of its features.	3.4.2 User's Manual
7.1.38 Update cost estimate	Update the table of costs with all components of all phases.	3.5 Final Cost Estimate
7.1.39 Ensure professionalism of the document	Assemble all parts of deliverable 2. Ensure the document is professional looking and well-organized.	Deliverable 3

7.1.40 Correction of various part of the deliverables	Make all necessary changes in the deliverables for the final report.	Corrected deliverables
7.1.41 Assemble all deliverables	Put all corrected/updated deliverables together for the final report. Ensure professionalism of the document.	Final report

7.2 Artifacts

7.2.1 1.2. Project Description

The project description is an introduction to the entire project. It's a brief that allows the reader and the user of the system to understand the goal and purpose of the project. This document must be presented as an opening to the rest of the manual, and must include the idea, objective, background, approach, and result of the entire project.

7.2.2 1.3.1 Functional Requirements

Requirements are the tasks that are necessary for the completion of the system. They are the components of the system that must be documented, measured and tested in order to ensure the ideal conclusion to the project in question. The requirements that need to be specified for this system are functional and non-functional requirements. The functional requirements are the tasks that define what the system is trying to accomplish. These are the elementary actions that constitute the main goal of the system. They include the user viewing weekly schedule, adding classes, dropping classes, swapping, etc.

Each individual requirement must be presented as a use case. A use case is a description of the behavior of a system that explains the process and results obtained through interactions of the user with the system. The functional requirements must be known before writing use cases, because they are the details that compose the main actions and tasks of the system. The use cases are present in both list form and diagram form.

- Use Cases: The list-based use cases are full formal documents that list all use cases of the system in detail, with each property of the use case explained in full detail. These include the name of the use case, the general description, the actors, preconditions, postconditions, etc.
- Use Case Diagram: Use case diagrams are used to depict a visual representation of the use cases listed. The actors, which typically include the user or an external entity, are depicted using stick figures. The use cases themselves are drawn as circles with a name that describes the title of the action that can be initiated by the actors. There are lines connected from users to the use cases called connections, which describe the actor's interactions with those use cases.

7.2.3 1.3.2. Domain Model

The Domain Model is the model that describes all core concepts and an overview of the system at hand. Its main purpose is to list the objects of interest that form the system, and presents their attributes,

constraints, and relationships between them. The model is designed in UML format, with arrows describing relationships from one object to another, and ensuring that there are multiplicity values at both ends of the arrow denoting multiplicity factors of the relationship. The finalized domain model will come with a description that will provide a more detailed explanation of the model as well as each object present in the model.

7.2.4 1.3.3. Constraints and Qualities

The constraints and qualities segment describes the main standards of quality and non-functional requirements that are expected to be met by the system. The non-functional requirements are the general properties of the system. They are the constraints and qualities that will make up the system when in use. These include response times, maintenance times, security, etc. It's important that the constraints and qualities are as specific as possible, which means they must have concrete metric specifications wherever possible.

7.2.5 1.4.1. Human Resources

The human resources of a project is the group of individuals who compose the main workforce of the project. They are the source of labor of a project, and in the case of large projects, are typically divided into smaller groups for more organized work.

7.2.6 1.4.2. Technical Resources

The technical resources of a project are the computer resources that will be used to complete the project. This includes the computer programs, software, libraries, and websites that are available for use.

7.2.7 1.5. Scoping

The scoping section describes all elements, features, qualities, and goals that have been scoped out from the end result. Each scoped element must come with a description of what it is, in addition to a paragraph that describes why it was chosen to be scoped out. These scoped elements have been typically scoped out in the early phases of a project, where the requirements are being planned and discussed in between team members.

7.2.8 1.6.1. Architecture

The architecture of a project is the high-level system architecture that is planned for the complete project. It must be presented with a UML class diagram that describes the details and relationships between different components of the system, along with a detailed description of the mindset behind the architecture (also known as a design rationale). In addition, each module of the architecture must be accompanied with its own design rationale and responsibilities.

7.2.9 1.6.2. Technologies in Use

This segment must provide a technical description of all the technologies that will be used to build the project. These include programming languages, libraries, servers, databases, IDE and compilers, etc. For each technology listed, there must be a short description that explains the rational for the choice and use of the specific tool.

7.2.10 1.7.1. Activities

The activities section is a section that will list a step-by-step process of how the separate artifacts of the project will be produced in order to reach the completion of the project. Each activity must be presented with a clear purpose, description, and must produce at least one artifact in consequence.

7.2.11 1.7.2. Artifacts

An artifact is a by-product that is produced as a consequence of the development of a software project. The artifacts segment will list each document that will be produced in the project. Each artifact listed must come with a description that provides details about the role of that specific document in the completion of the project.

7.2.12 1.7.3. Project Estimates

The project estimates section is for providing a realistic estimate for the cost, as well as the schedule for the length of the project. This is done by summing the evaluated cost of each artifact that's produced throughout the conception of the project. Each cost must be backed up by a basis of the estimation.

7.2.13 1.7.4. Activities Assignments

The activities assignments are the specific assignments of tasks and activities that are made to each team member based on the members' skills, requirements, capacities, limitations and schedules.

7.2.14 1.7.5. Schedule

The schedule section must present all the project's deadlines, milestones, sprints, iterations and other target dates in the form of a diagram or table. A Gantt chart is the ideal way to present a schedule as it outlines the dates all important events from start to finish in a clean and organized fashion.

7.2.15 1.7.6. Risk

The risk section will present all elements of risk of the project, including both early factors and late factors of risk. Each risk must be accompanied by a description and a consequence should the risk occur. They may be presented in a table or list form.

7.2.16 1.8. Prototyping

The prototype section is a description of what work has been completed so far in terms of developing a prototype of the project. A prototype is an early model that is built before the final release of a system or device. The goal of this section is to decide whether the tools and technologies chosen for the completion of the project are appropriate for the final release.

7.2.17 2.2. Introduction to Part 2

The second introduction will introduce the user to the second large deliverable of the project. It is simply a brief description of what information is included in the entire document.

7.2.18 2.3.1. Architecture Diagram

The architecture diagram section is an improved and more detailed version of the architecture diagram presented in the previous deliverable. The system must be divided into at least two systems for this diagram. It must include a high-level description of the architecture of the system in addition to descriptions for each separate module included in the system, accompanied with the visual diagram.

7.2.19 2.3.2. Subsystem Interfaces Specification

The subsystem interfaces specification section is the section that will describe the specifications of the interactions between the software interfaces of the components of the project. These are most likely presented as the function calls and message passing that occur. They must be accompanied with descriptions of the parameters passed of the function calls, including possible valid and invalid ranges of values.

7.2.20 2.4.1. Detailed Design Diagram

The detailed design diagram is a diagram that describes the internal structure of the subsystems depicted in the project. This must be presented as a UML diagram, and must include descriptions that explain the rationale of the designs, in addition to any extra information that is not present in the diagram.

7.2.21 2.4.2. Unit Description

The unit description section is the section that lists each class that's present in the subsystem and accompanies them with a description of the purpose and function of the class. In addition, they are accompanied by all attributes and methods of the class. They may also be notes and reminders that are there to give any extra information for the programmers who are implementing the classes.

7.2.22 2.5. Dynamic Design Scenario

The dynamic design scenario is a list of two vital and substantial use cases of the system. These use cases must have at least 3 system operations, and the scenarios listed must include system sequence diagrams and operational contracts.

7.2.23 2.6. Estimation

The estimation section is a list of all modules that have been identified so far, accompanied by a new estimate for each one. The revision must also include the newly estimated costs of integration, testing, and additional documentation.

7.2.24 2.7. Rapid Prototyping and Risk

The rapid prototyping and risk section lists the prototypes of the system completed so far. The list must be accompanied by all information pertaining to the models, including classes, modules, drivers, frameworks, database, and other technologies used. Each prototype has a description of the effect of its model, in addition to the effects on the risks previously mentioned, the effects on the estimate, and the changes made in the scope because of them.

7.2.25 3.2. Introduction to Part 3

The third introduction will introduce the user to the third large deliverable of the project. It is simply a brief description of what information is included in the entire document.

7.2.26 3.3.1.1. Tested Items

The tested items section is a list of all items and components that have been tested so far, including the test cases that were applied during the process. Each tested item is accompanied with a description that explains the test case along with why the testing was necessary.

7.2.27 3.3.1.2. Untested Items

The untested items section is a list of all items and components that have yet to be tested, but are deemed necessary to be tested. They are accompanied with an explanation that highlights why it should be tested, along with how it would be tested.

7.2.28 3.3.2.1. Unit Testing

The unit testing section includes two mid-level testable units of the system, which could be modules, classes, or subsystems. Each test unit is accompanied by a list of test cases and the code for the drivers used. In addition, an explanation of the techniques and description of the test are provided, along with the results of the testing.

7.2.29 3.3.2.2. Requirements Testing

The requirements testing section is a list of test cases that accompany each tested requirement of the system. This includes a realistic scenario of the system usage during the test, along with the expected reaction of the system under the conditions of the test.

7.2.30 3.3.2.3. Stress Testing

The stress testing section is a description of the potential extreme situations of system usage. This is necessary to view the behavior of the system in unlikely but extreme conditions to ensure stability of the system as a whole. A design of tests is provided along with the description that verifies how the system performs under those extreme conditions.

7.2.31 3.3.2.4. Security Testing

The security testing section is a description of the tests that have been undergone to ensure maximum security of the system. This includes SQL injection attack resistance to avoid breach of user information.

7.2.32 3.4.1. Installation Manual

The installation manual is a report that is aimed for a competent administrator. It is a precise description of the steps to take to install (i.e. from a compressed file or disk) and execute the software. The reader is expected to install the system simply using the code and this manual provided.

7.2.33 3.4.2. User's Manual

The installation manual is a document that is aimed for the general user. It is a precise description of how to use the system, and it provides guidelines for all usable components present. All features of the system must be included and described in this document.

7.2.34 3.5. Final Cost Estimate

The final cost estimate is a table that neatly lists all components that are and have been present in the completion of the project. These estimates include documentation, design, implementation, data acquisition, and testing costs.

7.3 Project Estimates

7.3.1 Deliverable 0 Estimation

Task Names	Cost (Hours)
Domain Model	6
Identification of Technologies	12
Team Members	0.5
Implementation Discussion	5
DMO Descriptions	6
Total	**29.5**

7.3.2 Deliverable 1 Estimation

Task Names	Cost (Hours)
Requirements	10
Use Case Diagrams/Models	8
Resources	5
Scope	5
Architecture	7
Total	**35**

7.3.3 Deliverable 2 Estimation

Task Names	Cost (Hours)
Programming Architecture	100
Use Case Implementation	50
Design Discussion/Implementation	25
Writing up Test Cases	15
Total	**190**

7.3.4 Deliverable 3 Estimation

Task Names	Cost (Hours)
Test the Test Cases	30
Debugging/Optimizing code based on test cases	15
User Performance Test	20
Debugging/Optimizing code based on user tests	10
Total	**75**

7.3.5 Deliverable 4 Estimation

Task Names	Cost (Hours)
Final Optimizations	20
Compiling data into one file	30
Final Test on System	10
Total	**60**

7.3.6 Final Estimations

Deliverable Names	Cost (Hours)
Deliverable 0	29.5
Deliverable 1	35
Deliverable 2	190
Deliverable 3	75
Deliverable 4	60
Total	**389.5**

7.4 Activities Assignments

Activity	Assigned member(s)
7.1.1 **Form and divide team**	Claudia Della Serra Dimitri Topaloglou Philip Lim Ryan Lee Daniel Di Corpo Erin Benderoff Aline Koftikian Ideawin-Bunthy Koun Kevin Yasmine Marc-Andre Leclair Lori Dalkin Bryce Drewery-Schoeler
7.1.2 **Complete the system overview**	Claudia Della Serra Dimitri Topaloglou Philip Lim Ryan Lee Daniel Di Corpo Erin Benderoff Aline Koftikian Ideawin-Bunthy Koun Kevin Yasmine Marc-Andre Leclair Lori Dalkin Bryce Drewery-Schoeler
7.1.3 Write a project description	Ryan Lee

7.1.4 **Determine the functional requirements and draw use case diagrams accordingly**	Dimitri Topaloglou Bryce Drewery-Schoeler Ideawin-Bunthy Koun Erin Benderoff Lori Dalkin Marc Leclair Claudia Della Serra Daniel Di Corpo
7.1.5 **Review and correct domain model**	Aline Koftikian
7.1.6 **Find and describe constraints and qualities**	Dimitri Topaloglou
7.1.7 **Review and format human resources document**	Lori Dalkin
7.1.8 **Determine technical resources**	Dimitri Topaloglou
7.1.9 **Reduce the amount of features to be realistic according to time and resources**	Kevin Yasmine
7.1.10 **Choose the architecture of the system**	Daniel Di Corpo
7.1.11 **Decide which technologies are relevant to the project development**	Dimitri Topaloglou
7.1.12 **List and describe the main activities involved in the project**	Philip Lim
7.1.13 **List all artifacts**	Aline Koftikian
7.1.14 **Estimate the cost and time of production**	Daniel Di Corpo
7.1.15 **Delegate activities to different team members**	Philip Lim
7.1.16 **Produce a schedule for the project**	Ryan Lee
7.1.17 **Determine the risks of the project**	Dimitri Topaloglou Kevin Yasmine
7.1.18 **Produce a prototype report**	Lori Dalkin
7.1.19 **Ensure professionalism of the document**	Claudia Della Serra
7.1.20 **Update the introduction for Deliverable 2**	Ryan Lee

7.1.21 **Produce an architecture diagram**	Aline Koftikian Dimitri Topaloglou Daniel Di Corpo
7.1.22 **Specify the interactions between the components of the software**	Ideawin-Bunthy Koun Marc-Andre Leclair Philip Lim
7.1.23 **Illustrate the internal structure of the system**	Erin Benderoff Aline Koftikian
7.1.24 **Describe each class in the subsystem**	Kevin Yasmine Lori Dalkin Bryce Drewery-Schoeler
7.1.25 **Produce dynamic design scenarios**	Ryan Lee Claudia Della Serra
7.1.26 **Update project estimates**	Daniel Di Corpo
7.1.27 **Produce a report about the prototype and risk**	Dimitri Topaloglou
7.1.28 **Ensure professionalism of the document**	Claudia Della Serra
7.1.29 **Update introduction**	Ryan Lee
7.1.30 **List all tested items**	Claudia Della Serra
7.1.31 **List all untested items**	Ryan Lee
7.1.32 **Unit testing report**	Lori Dalkin Aline Koftikian
7.1.33 **Requirements testing report**	Erin Benderoff Philip Lim
7.1.34 **Stress testing report**	Ideawin-Bunthy Koun Kevin Yasmine
7.1.35 **Security testing report**	Bryce Drewery-Schoeler Marc-Andre Leclair
7.1.36 **Installation manual redaction**	Claudia Della Serra Dimitri Topaloglou Philip Lim Ryan Lee Aline Koftikian Kevin Yasmine
7.1.37 **User's manual redaction**	Daniel Di Corpo Claudia Della Serra Philip Lim Ryan Lee Aline Koftikian Kevin Yasmine

7.1.38 **Update cost estimate**	Daniel Di Corpo
7.1.39 **Ensure professionalism of the document**	Claudia Della Serra
7.1.40 **Correction of various part of the deliverables**	Claudia Della Serra Dimitri Topaloglou Philip Lim Ryan Lee Daniel Di Corpo Erin Benderoff Aline Koftikian Ideawin-Bunthy Koun Kevin Yasmine Marc-Andre Leclair Lori Dalkin Bryce Drewery-Schoeler
7.1.41 **Assemble all deliverables**	Claudia Della Serra

7.5 Schedule

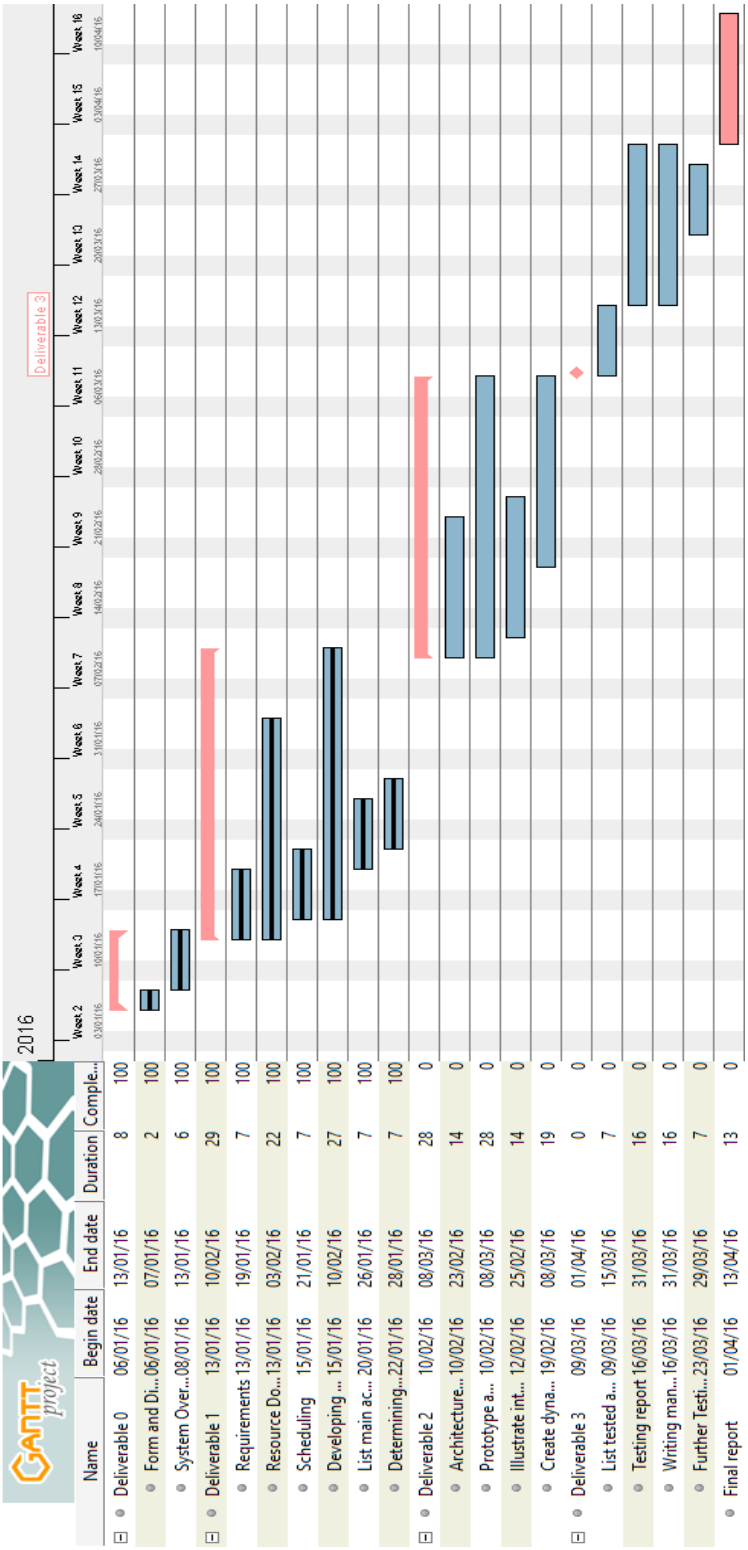


Figure 7.5.1 Gantt chart for first version of project deliverables

7.6 Risk

7.6.1 Language Familiarities

The main languages used in this project revolve around PHP, SQL, JavaScript and jQuery; not all team members may have much experience using such languages.The language unfamiliarity risks causing further deadline delays due to the learning curves involved in learning a new language. Debugging issues can also arise due to potential inexperience with a certain programming language.

Mitigation of such a risk will involve the use of extensive research on languages, as well as the aid of other team members in learning and passing on knowledge of programming languages.

7.6.2 Server Uptime

This production environment of this project is set up at a team member's own home. This could potentially cause a problem should a power outage occur or the server computer crashing. This can hinder deployment and may ultimately cause certain downtime until the issue is resolved.

7.6.3 Control Version System

The control version system in use is GitHub. However, GitHub requires an extensive learning curve to master and have full control over. This may cause unforeseen delays should there be an issue with the repository.

Mitigation of this risk will involve all team members becoming familiar with GitHub early on in the project. Working with their own repositories, learning how to add, commit, push, branch, and make pull requests on their own time, and learning how issues and milestones work will aid in reducing the potential of the risk occurring.

7.6.4 Time

One of the main concerns with a project this size is time constraint. With the given timeframe, it is uncertain whether there will be adequate time to complete the project in its entirety without the possibility of lacking certain functionality. Essentially, assessing some of the aforementioned risks, time is a common concern.

In order to attempt to mitigate such risks, tasks will be divided heavily amongst group members; given that the group consists of 12 members, dividing tasks up will serve to maximize time spent working on the project and time to completion of tasks.

Risk Table				
Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM
Server Uptime	Technology	Medium	1	**7.6.2**
Time	Estimation	High	1	**7.6.4**
Control Version System	People	High	2	**7.6.3**
Language Familiarities	People	Low	3	**7.6.1**

8. Prototyping

The prototype has three functioning pages, the login, the courses and the users page. The log in page allows the user to log in to the website only if their credentials check out. The website sends a request to the database to check if the username and password combination is valid. The current working pair of credentials is: Username= admin, Password= 1234. The update user page allows the user to change their information. This change is transmitted to the database and modifies the user's table. The course page allows the user to view a list of courses from the database. Currently all core class needed for the Software Engineering program are available to see on this list. The user may see information about ten classes at a time and can press the next and previous arrow to move through the list. In this prototype, all users created are admins. In a later iteration, the general user will have more restricted access to the database.

The Yii framework is used to construct this prototype. No members of the group were familiar with this framework thus building the prototype allowed the group members to learn the basics of this framework. This framework follows the MVC architecture format. Yii greatly simplified the processes of making classes to retrieve and modify information from the database by generating the model, view and controller class code in php. This prototype demonstrates that this framework is capable of performing all the specified project requirements.

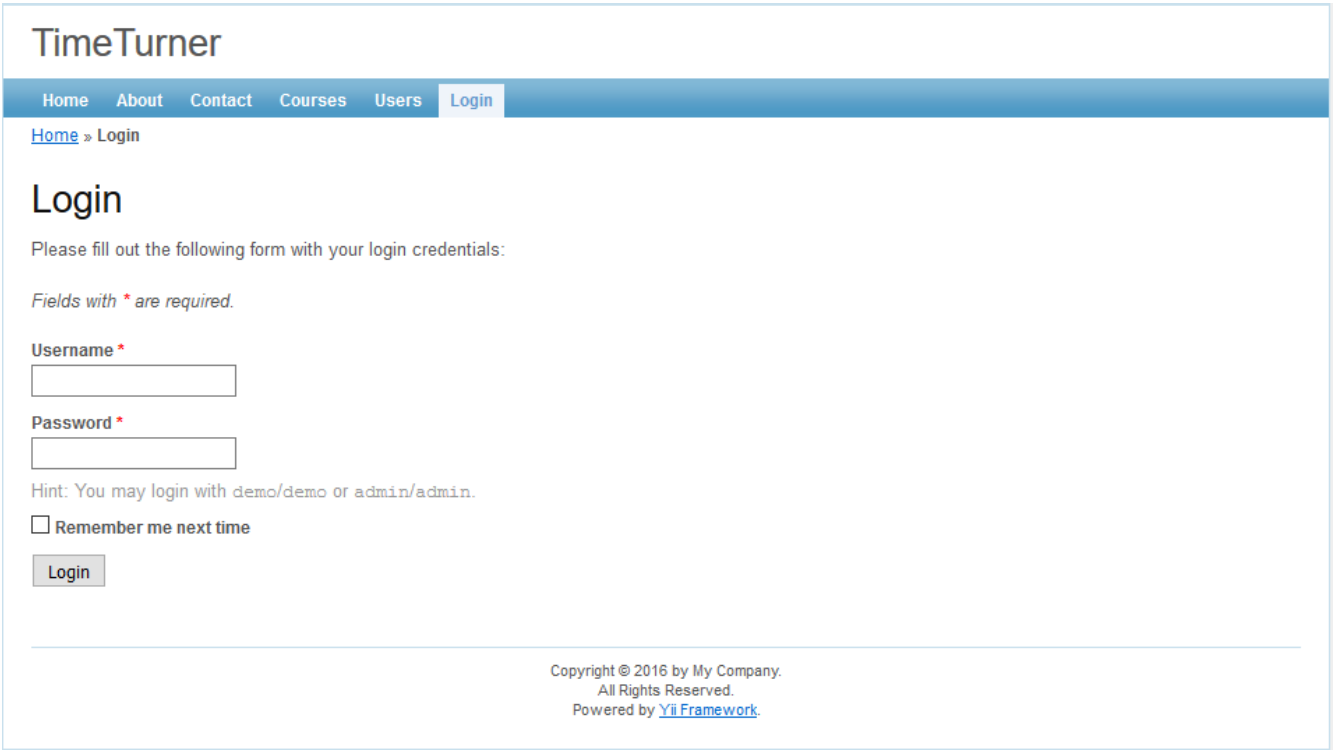


Figure 8.1 Early Prototype of login page for TimeTurner

Project Design Document

9. Introduction

The following section covers the design of TimeTurner in detail. This design is partitioned into different views and diagrams, starting with the 4+1 architecture view. This provides 5 individual views on the system which include logical, development, process, physical and scenario views. Following, will be descriptions of the subsystems, along with the classes included in each. Accompanying each class will be an in depth description of the class and their respective class diagrams. The final section includes all the dynamic design scenarios for each given situation defined by the use cases.

10. Architectural Design

10.1 Architecture Diagram (4+1 View)

10.1.1 Physical View

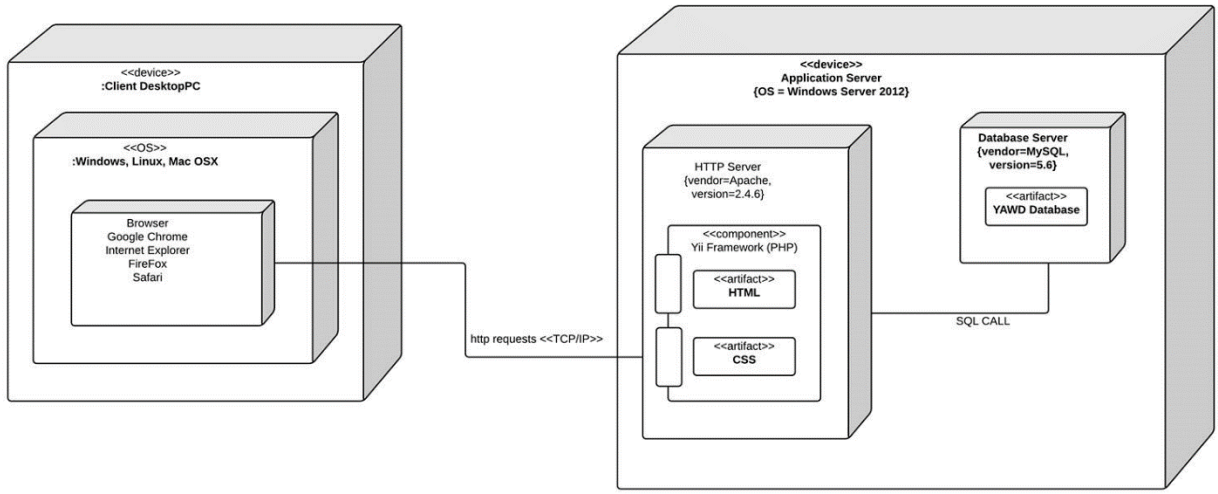


Figure 10.1.1 Deployment diagram representing the physical view

This diagram illustrates the physical deployment of the system. The system is a web-based application which uses a MVC architecture. The server component is located on the application server that runs Windows Server 2012. The server runs the Apache HTTP server supported by a MySQL database server. The database deploys seven tables, holding the information of saved users, all course, section, and subsection information, all prerequisites applying to those courses, the completed courses of each user, and each user's saved schedules.

The system itself is deployed by the Yii framework, running on the Apache server, which takes care of the model, view and controller modules. The view modules are also deployed by various AJAX, HTML, and CSS files to provide the user with an appealing interface. The framework works closely with the

database in order through SQL calls to keep all tables updated as users perform tasks on the system; this includes updating their list of courses completed and creation and upkeep of schedules.

The client component could be used on any device that has a supported web browser such as Google Chrome, Internet Explorer and Mozilla Firefox. Communication between client and server are done through HTTP request and responses.

10.1.2 Logical View

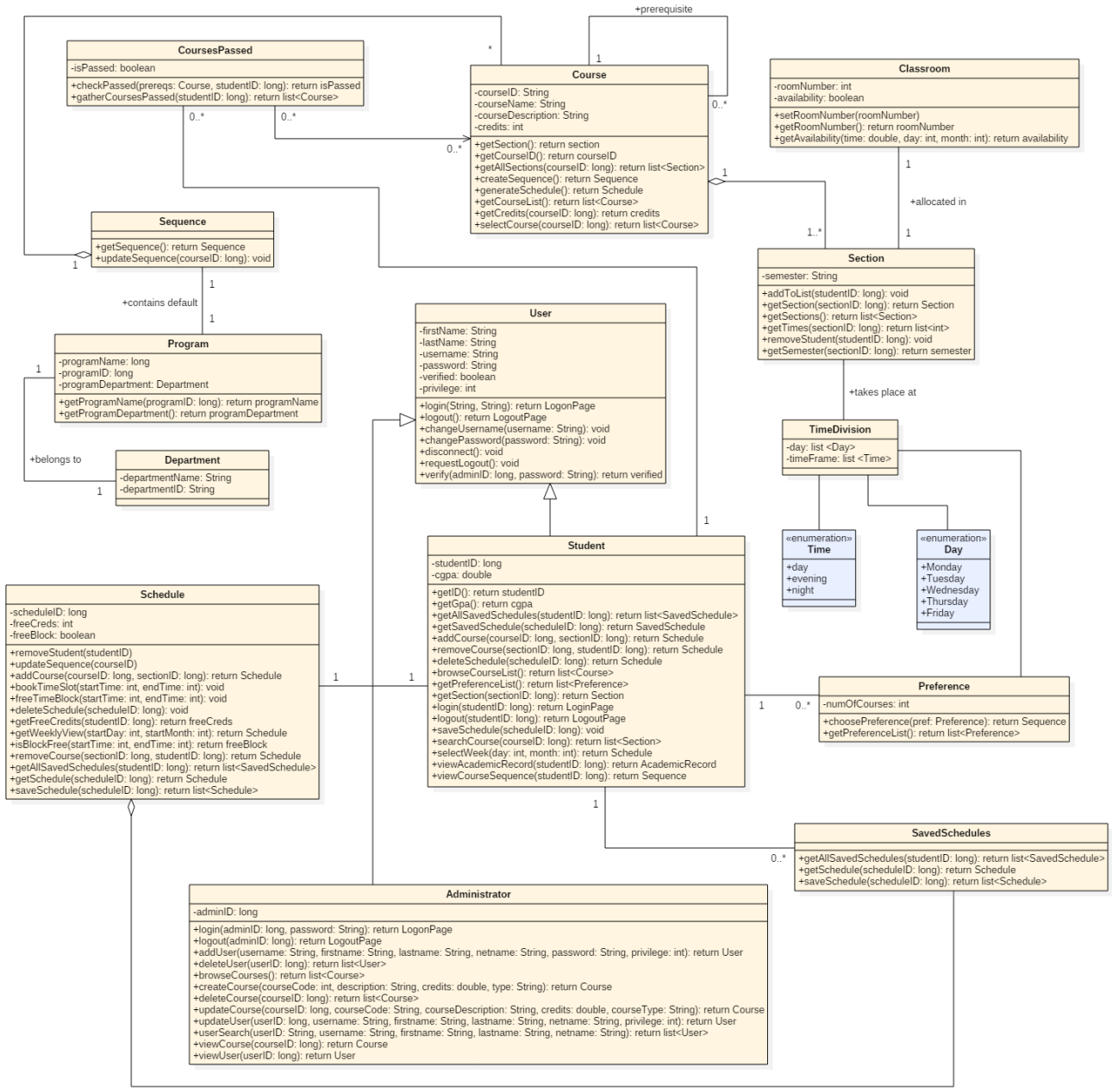


Figure 10.1.2 Class diagram representing the logical view

The logical view of the system's architecture is represented by a full class diagram. The Student and Administrator classes are subclasses of the User superclass, where they inherit the methods of the latter, and overwrite the login() and logout() functions. The User class is an aggregate of SavedUsers, since the SavedUsers is comprised of many Users.

A student has a number of saved schedules, represented by the SavedSchedules class, a single schedule, and a number of preferences, which are the students preferred times and days of the week for classes.

The Section class is also related to the TimeDivision class by being offered at those times. Sections are also taken place in classrooms, therefore explaining the relationship between the two classes. It's also related to the Course class through aggregation.

The Course class is an aggregation of the Section class, because a course is composed of many sections. A course has a number of prerequisites represented by the Prerequisites class, and a class CoursesPassed inherits from the Course class, which represents the courses from the Course class that have or have not been credited.

The Sequence class is an aggregation of the Course class, because a sequence is composed of courses. A program also contains its own default sequence of courses, which is shown with the relationship between the Program and Sequence classes. In addition, a program belongs to a department.

10.1.3 Process View

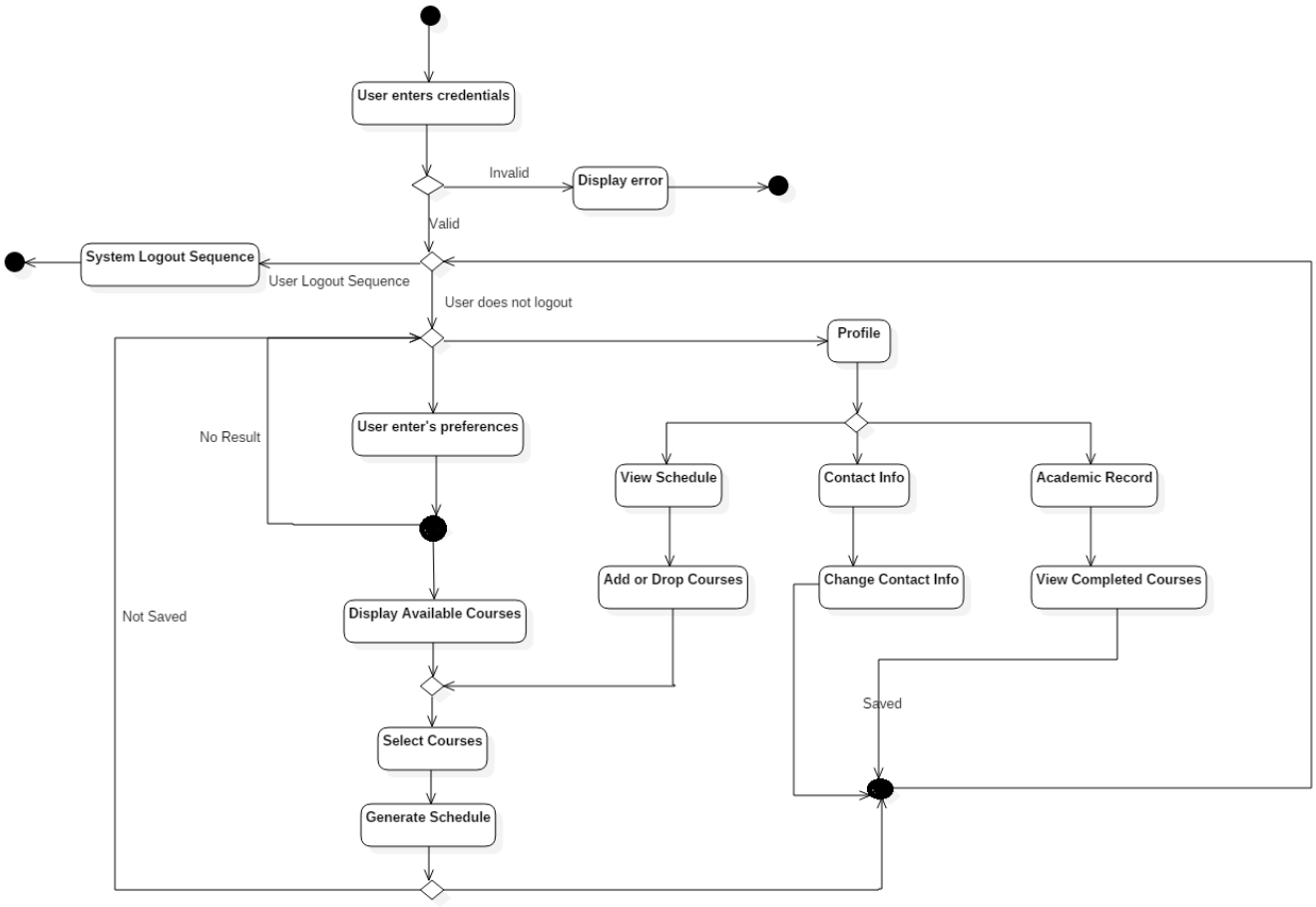


Figure 10.1.3 Activity diagram representing the process view

The main activity sequence in Time Turner shows the flow of events for a student course sequence generator and profile view. The flow of events start with the student user entering his credentials, which is verified. If the user enters invalid information, an error occurs, an error message is sent and the user is sent back to the login page.

From this point, the user has 3 options:

1. **Logout:** The user chooses to logout. The system will go through the proper sequence and return to the home page.
2. **Profile:** The user chooses to go to the profile page. Once there, the user has 3 options, contact info, academic record or view schedule. If contact info is chosen, the user is able to change his/her info and the system will return them to the home page. If academic record is chosen, the user is able to view completed courses and the system will send them back to the home page. If view schedule is chosen, the user is able to add and remove specific courses from their generated list. At this point, their choices will be verified and they must generate a new schedule.

3. **Generate Schedule:** The user chooses to enter preferences. Once the user has selected their prefered selection, the system will show available classes. If no result has been shown, the user must make a new selection for their preferences. If classes have been shown, the user is able to select their preferred lectures/tutorials/labs with checkboxes. Once selected, the system will generate a schedule and save it to the user's profile.

10.1.4 Development View

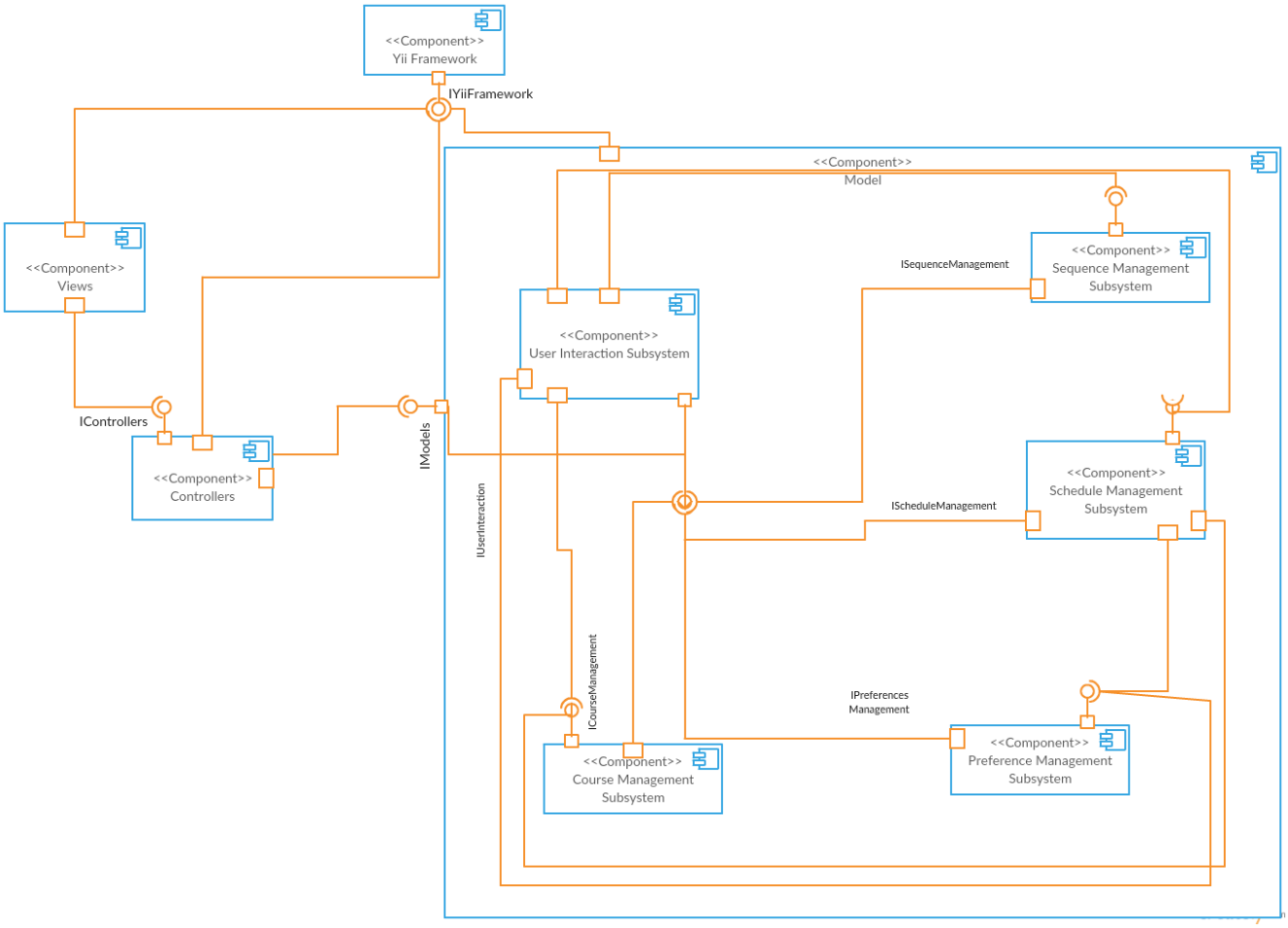


Figure 10.1.4 Component diagram representing the development view

Time Turner is broken down into three main components which are based on the MVC architecture. Each of these components are dependent on the Yii framework which provides services to the rest of the application.

The Views component is comprised of all the user interfaces for Time Turner and is the component in which the user first interacts with the system. The Controllers component is paired with an interface from the view component to provide access to the system.

The Models component is by far the most complex and is composed of 5 unique subsystems components. The point of interest to the various subsystems of the model component is the User Interaction Subsystem which is comprised of classes to give services to the user such as, viewing the course sequence and searching for courses. Classes of the User Interaction Subsystem will call upon methods from other

subsystems such as the Sequence Management Subsystem, to view the user's current sequence, Schedule Management Subsystem to generate schedules and register for courses, Preference Management Subsystem to record user's requirements and Courses Management Subsystem to view courses offered by the university. To complete these tasks, many of these subsystems must communicate with each other, such as the Schedule Management Subsystem which works with the Courses Management Subsystem in order to know which classes are available to the student. It is also necessary for the User Interaction Subsystem to give information about the user to the subsystems in order to ensure that tasks are completed properly.

10.1.5 Situational View (Use Cases)

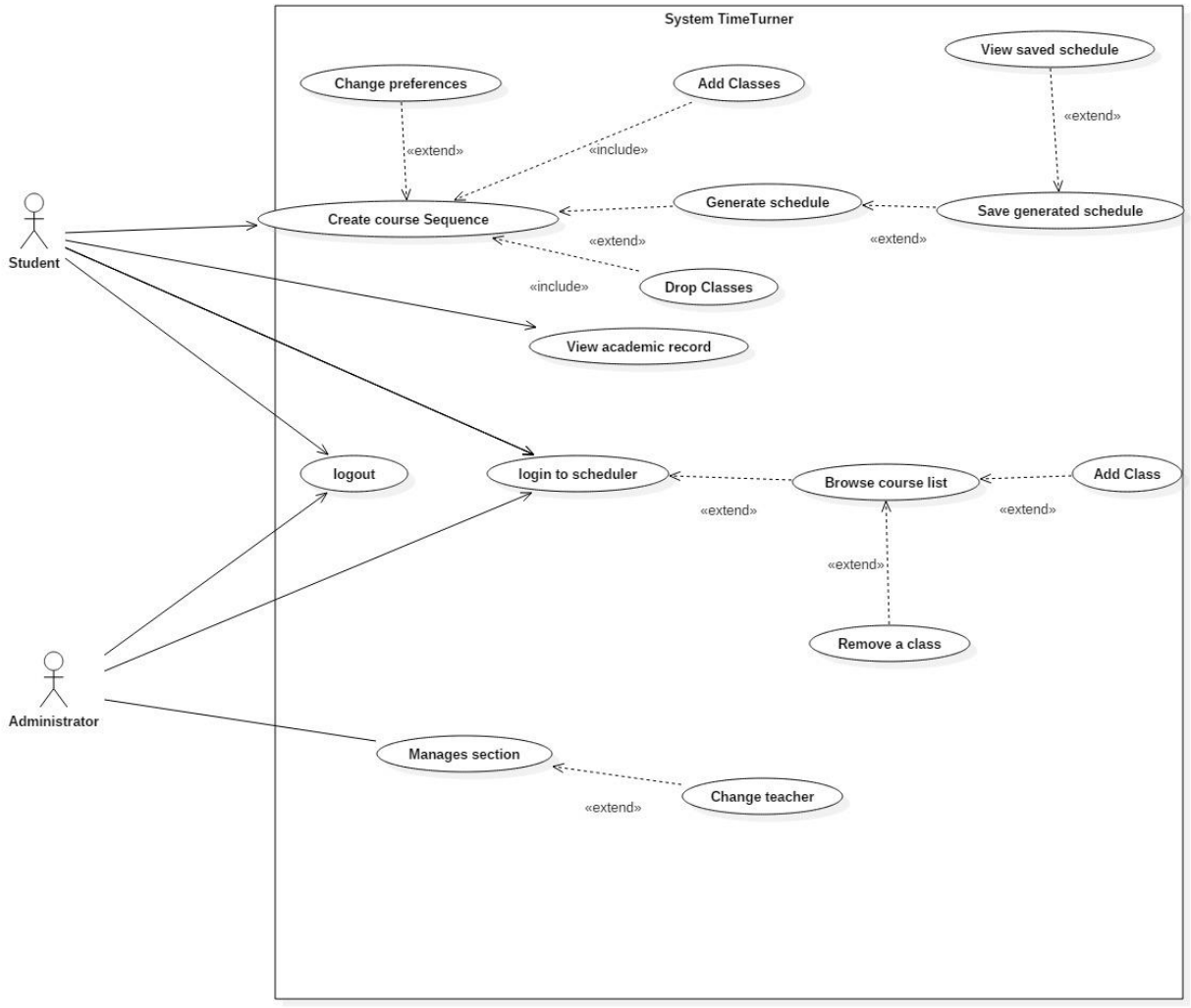


Figure 10.1.5 Use case diagram representing the situational view

The situational, or "+1", view of the system's architecture is represented by a full use case diagram. The use cases of the Student and Administator users are depicted in the diagram, represented by the ellipses.

The actions a student can perform are logging into the scheduler, creating their course sequence, viewing their academic record, and logging out of the scheduler. Creating the course sequence is comprised of several actions, including changing the preferences of the sequence, adding classes to the sequence,

dropping classes, and generating schedule. The generating schedule use case is an extension of the saving schedule use case, which is an extension of viewing the schedule use case.

The administrator ‘s actions include login, logout, and manage sections. Similar to the student’s use cases, the login use case is an extension of the browsing course list use case, which is an extension of adding and removing classes. In addition, the administrator manages a section, which includes the action of changing the section’s teacher.

10.2 Subsystem Interfaces Specifications

The system consists of five interfaces: **IUserInteraction**, **IScheduleManagement**, **ICourseManagement**, **IPreferenceManagement** and **ISequenceManagement**. Each of these interfaces provides a set of methods that components can publicly access. The use of the method calls are explained in further detail in dynamic design scenarios (See Section 5).

10.2.1 IUserInteraction

The user interaction interface provides basic methods that a user (administrator or student) will use to interact with the system. Those interactions include login and logout operations. This interface also allows a user to consult data related to courses and schedules, but no modifications on those data are allowed through this interface. The classes implementing the provided methods are the Student, Administrator and User classes.

10.2.1.1 Specifications for methods implemented by the Student class

Method
browseCourseList() list of Course objects
Specification
Description: Retrieves and returns a list of courses that a student can take.   ***Input parameter(s):***   none   ***Return type:***   list of Course objects

Method
getSavedSchedule(scheduleID:long) Schedule
Specification
Description: Given a schedule ID, retrieves and returns a specific saved schedule that represents the schedule of a semester.   ***Input parameter(s):***   scheduleID:long representing the schedule ID   ***Condition(s) for validity:***   scheduleID ≥ 0 ***Return type:***   Schedule

Method
login(studentID:long, password:String) LogonPage
Specification
Description: Logs a student into the system, provided that the student ID and password passed as parameters are valid. ***Input parameter(s):*** studentID:long representing the ID of the student to be logged in password:String representing the password entered by the user ***Conditions for validity:*** studentID ≥ 0 password is not empty ***Return type:*** LogonPage

Method
logout(studentID:long) LogoutPage
Specification
Description: Logs a student out of the system. ***Input parameter(s):*** studentID:long representing the ID of the student to be logged out ***Conditions for validity:*** studentID ≥ 0 ***Return type:*** LogoutPage

Method
searchCourse(courseID:long) list of Section objects
Specification
Description: Retrieves and returns information about a course, including the course's sections, given the course ID. ***Input parameter(s):*** courseID:long representing the course ID ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** list of Section objects

Method
selectWeek(day:int, month:int) Schedule
Specification
Description: Provides the caller with a weekly schedule view given the day and month as valid integer values for the targeted week. ***Input parameter(s):*** day:int representing a calendar day in the targeted week month:int representing the month of the targeted week ***Conditions for validity:*** $1 \leq day \leq 31$ $1 \leq month \leq 12$ ***Return type:*** Schedule

Method
viewAcademicRecord(studentID:long) AcademicRecord
Specification
Description: Allows to view the academic record of a student given the student's ID. ***Input parameter(s):*** studentID:long representing the ID of the student ***Conditions for validity:*** $studentID \geq 0$ ***Return type:*** AcademicRecord

Method
viewCourseSequence(studentID:long) Sequence
Specification
Description: Retrieves and returns the sequence of a student given the ID of the student. ***Input parameter(s):*** studentID:long representing the ID of the student ***Conditions for validity:*** $studentID \geq 0$ ***Return type:*** Sequence

10.2.1.2 Specifications for methods implemented by the Administrator class

Method
addUser(username:String, firstname:String, lastname:String, netname:String, password:String, privilege:int) User
Specification
Description: Adds a new user to the system, provided its username, first name, last name, net name, password and privilege code. Upon successful completion, the newly create user is returned. ***Input parameter(s):*** username:String representing the username of the user to be created firstname:String representing the first name of the user to be created lastname:String representing the last name of the user to be created netname:String representing the net name of the user to be created password:String representing the password of the user to be created privilege:int representing a code that indicates whether the new user is a student or an administrator ***Conditions for validity:*** username.length > 0 firstname.length > 0 lastname.length > 0 netname.length > 0 password.length > 5 privilege = 0 ⊕ 1 username does not exist netname does not exist ***Return type:*** User

Method
browseCourses() list of Course objects
Specification
Description: Returns a list of all courses offered by the university. ***Input parameter(s):*** none ***Return type:*** list of Course objects

Method
createCourse(courseCode:int, description:String, credits:double, type:String) Course
Specification
Description: Adds a new course along with its description into the system provided the course code, the description, the credits and the type of course. Returns the newly create course. ***Input parameter(s):*** courseCode:String representing the code of the course to be added description:String representing the description of the course to be added credits:double representing the number of credits allocated to the course type:String representing the type of the course ***Conditions for validity:*** courseCode ≥ 100 description is not empty credits ≤ 0 type is not empty ***Return type:*** Course

Method
deleteCourse(courseID:long) list of Course objects
Specification
Description: Deletes a course given its course ID and returns an updated list of courses. ***Input parameter(s):*** courseID:long representing the ID of the course to be deleted ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** list of Course objects

Method
deleteUser(userID:long) list of User objects
Specification
Description: Deletes a user given its user ID and returns an updated list of users. ***Input parameter(s):*** userID:long representing the ID of the user to be deleted ***Conditions for validity:*** userID ≥ 0 ***Return type:*** list of User objects

Method
login(adminID:long, password:String) LogonPage
Specification
Description: Logs an administrator into the system, provided that the admin ID and password passed as parameters are valid. ***Input parameter(s):*** adminID:long representing the ID of the administrator to be logged in password:String representing the password entered by the user ***Conditions for validity:*** adminID ≥ 0 password is not empty ***Return type:*** LogonPage

Method
logout(adminID:long) LogoutPage
Specification
Description: Logs an administrator out of the system. ***Input parameter(s):*** adminID:long representing the ID of the administrator to be logged out ***Conditions for validity:*** adminID ≥ 0 ***Return type:*** LogoutPage

Method
updateCourse(courseID:long, courseCode:String, courseDescription:String, credits:double, courseType:String) Course
Specification
Description: Given the ID of a course, allows to update the course information such as the course code, course description, credits allocated and type of course. Once completed, the updated course is returned. ***Input parameter(s):*** courseID:long representing the ID of the course to be updated courseCode:String representing the new code of the course credits:double representing the new number of credits allocated for the course courseType:String representing the new type of course ***Conditions for validity:*** courseID ≥ 0 credits ≥ 0 ***Return type:*** Course

Method
updateUser(userID:long, username:String, firstname:String, lastname:String, netname:String, privilege:int) User
Specification
Description: Given the ID of a user and a set of username, first name, last name, net name and privilege information, updates the user's information accordingly. Once updated, returns the updated user. ***Input parameter(s):*** userID:long representing the ID of the user to be updated username:String representing the new username to be set firstname:String representing the new first name to be set lastname:String representing the new last name to be set netname:String representing the new net name to be set privilege:int representing the new privilege level (student or administrator) to be set ***Conditions for validity:*** username.length > 0 firstname.length > 0 lastname.length > 0 netname.length > 0 privilege = 0 ⊕ 1 username does not exist netname does not exist ***Return type:*** User

Method
userSearch(userID:String, username:String, firstname:String, lastname:String, netname:String) list of User objects
Specification
Description: Based on filtering search criteria such as the user ID, the username, first name, last name and net name, this method returns a list of user corresponding to the search criteria. Note that the search criteria are based on a 'contains' relationship, and not 'exactly is'. ***Input parameter(s):*** userID:String representing the filtering criteria for the user ID, can be preceded by comparative operators username:String representing a part of, or the complete username to be filtered when looking for users firstname:String representing a part of, or the complete first name to be filtered when looking for users lastname:String representing a part of, or the complete last name to be filtered when looking for users netname:String representing a part of, or the complete net name to be filtered when looking for users ***Conditions for validity:*** none ***Return type:*** list of User objects

Method
viewCourse(courseID:long) Course
Specification
Description: Returns a course to be viewed by the administrator given its course ID. ***Input parameter(s):*** courseID:long representing the ID of the course to be viewed ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** Course

Method
viewUser(userID:long) User
Specification
Description: Returns the user corresponding to the valid given user ID. ***Input parameter(s):*** userID:long representing the ID of the user to be viewed ***Conditions for validity:*** userID ≥ 0 ***Return type:*** User

10.2.1.3 Specifications for methods implemented by the SavedUsers class

Method
isNameValid(name:String) boolean
Specification
Description: Verifies whether the given name, either first name or last name, is valid. For the name to be valid it must be non-empty. ***Input parameter(s):*** name:String representing the name to verify ***Conditions for validity:*** none ***Return type:*** Boolean

Method
isNetnameValid(netname:String) boolean
Specification
Description: Verifies whether the given net name is valid. For the net name to be valid, it must be non-empty and unique. Returns true if the username is valid, false otherwise. ***Input parameter(s):*** netname:String representing the netname to verify ***Conditions for validity:*** none ***Return type:*** boolean

Method
isPrivilegeValid(privilege:int) boolean
Specification
Description: Verifies whether the privilege level set to a user is valid. For the privilege level to be valid, it must be either 0 or 1 (student or administrator). Returns true if the privilege is valid, false otherwise. ***Input parameter(s):*** privilege:int representing the privilege level to verify ***Conditions for validity:*** none ***Return type:*** boolean

Method
isUsernameValid(username:String) boolean
Specification
Description: Verifies whether the given username is valid. For the username to be valid, it must be non-empty and unique. Returns true if the username is valid, false otherwise. ***Input parameter(s):*** username:String representing the username to verify ***Conditions for validity:*** none ***Return type:*** boolean

10.2.2 IScheduleManagement

The IScheduleManagement interface consists of a set of methods that allow another component to access and modify data about a schedule. The classes Schedule and SavedSchedules implement the provided methods.

10.2.2.1 Specifications for methods implemented by the Schedule class

Method
addCourse(courseID:long, sectionID:long) Schedule
Specification
Description: Adds a course given the course ID and the section ID to a student's schedule. ***Input parameter(s):*** courseID:long representing the ID of the course sectionID:long representing the ID of the section ***Conditions for validity:*** courseID ≥ 0 sectionID ≥ 0 ***Return type:*** Schedule

Method
bookTimeSlot(startTime:int, endTime:int)
Specification
Description: Sets time slots in a scheduled as occupied given the starting and ending time of the time block to book. ***Input parameter(s):*** startTime:int representing the time at which the time block starts endTime:int representing the time at which the time block ends ***Conditions for validity:*** 0 ≤ startTime ≤ endTime ***Return type:*** Void

Method
deleteSchedule(scheduleID:long)
Specification
Description: Deletes a schedule given its schedule ID. ***Input parameter(s):*** scheduleID:long representing the ID of the schedule to be deleted ***Conditions for validity:*** scheduleID ≥ 0 ***Return type:*** void

Method
freeTimeBlock(startTime:int, endTime:int)
Specification
Description: Frees a time block of a schedule given the start and end time of the block. ***Input parameter(s):*** startTime:int representing the time at which the time block starts endTime:int representing the time at which the time block ends ***Conditions for validity:*** $0 \leq startTime \leq endTime$ ***Return type:*** void

Method
getFreeCredits(studentID:long) int
Specification
Description: Returns the number of free credits a student is still allowed to take. ***Input parameter(s):*** studentID:long representing the ID of the student ***Conditions for validity:*** $studentID \geq 0$ ***Return type:*** int

Method
getWeeklyView(startDay:int, startMonth:int) Schedule
Specification
Description: Generates a weekly view of a schedule given the starting date. ***Input parameter(s):*** startDay:int representing the day on which the week starts startMonth:int representing the month in which the week is found ***Conditions for validity:*** $1 \leq startDay \leq 31$ $1 \leq startMonth \leq 12$ ***Return type:*** Schedule

Method
isBlockFree(startTime:int, endTime:int) boolean
Specification
Description: Verifies whether a time block defined by the given starting and ending time is free in a schedule. ***Input parameter(s):*** startTime:int representing the time at which the time block starts endTime:int representing the time at which the time block ends ***Conditions for validity:*** $0 \leq startTime \leq endTime$ ***Return type:*** boolean

Method
removeCourse(sectionID:long, studentID:long) Schedule
Specification
Description: Removes a given course section from the schedule of a student. ***Input parameter(s):*** sectionID:long representing the ID of the section to be removed studentID:long representing the ID of the student ***Conditions for validity:*** $sectionID \geq 0$ $studentID \geq 0$ ***Return type:*** Schedule

10.2.2.2 Specifications for methods implemented by the SavedSchedules class

Method
getAllSavedSchedules(studentID:long) list of Schedule objects
Specification
Description: Retrieves and returns all saved schedules of a student given its student ID. ***Input parameter(s):*** studentID:long representing the student ID ***Condition(s) for validity:*** $studentID \geq 0$ ***Return type:*** list of Schedule objects

Method
getSchedule(scheduleID:long) Schedule
Specification
Description: Retrieves and returns the schedule of a semester given the ID of the schedule. ***Input parameter(s):*** scheduleID:long representing the ID of the schedule ***Conditions for validity:*** scheduleID ≥ 0 ***Return type:*** Schedule

Method
saveSchedule(scheduleID:long) list of Schedule objects
Specification
Description: Saves a new schedule related to a student given the schedule ID, then returns an updated list of schedules that have been saved. ***Input parameter(s):*** scheduleID:long representing the ID of the schedule to be saved ***Conditions for validity:*** scheduleID ≥ 0 ***Return type:*** list of Schedule objects

10.2.3 ICourseManagement

The ICourseManagement interface allows to consult information about academic courses as well as the sections of each course. It also allows to perform modification on courses, such as adding and removing students from a course. The ICourseManagement interface is also responsible for creating course sequences and generating schedules. The Course, CoursesPassed, Section and Prerequisites classes are the ones implementing the provided methods.

10.2.3.1 Specifications for methods implemented by the Course class

Method
createSequence() Sequence
Specification
Description: Creates and returns a course sequence for a student. ***Input parameter(s):*** none ***Return type:*** Sequence

Method
generateSchedule() Schedule
Specification
Description: Generates and returns a student's schedule. ***Input parameter(s):*** none ***Return type:*** Schedule

Method
getAllSections(courseID:long) list of Section objects
Specification
Description: Returns a list containing all the sections of a course. ***Input parameter(s):*** courseID:long representing the ID of the course to be consulted ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** list of Section objects

Method
getCourseList() list of Course objects
Specification
Description: Returns a list containing all courses that a student can take. ***Input parameter(s):*** none ***Return type:*** list of Course objects

Method
getCredits(courseID:long) int
Specification
Description: Returns the number of assigned credits of a course. ***Input parameter(s):*** courseID:long representing the ID of the course ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** Int

Method
getPrerequisites() list of Prerequisite objects
Specification
Description: Returns a list of prerequisites for a course, or an empty list if no prerequisite courses are required for a course. ***Input parameter(s):*** none ***Return type:*** list of Prerequisite objects

Method
selectCourse(courseID:long) list of Section objects and Prerequisite objects
Specification
Description: Given the ID of the selected course, returns information including the sections and prerequisites of that course. ***Input parameter(s):*** courseID:long representing the ID of the course ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** (mixed) list of Section objects and Prerequisite objects

10.2.3.2 Specifications for methods implemented by the CoursesPassed class

Method
checkPassed(prereqs:Course, studentID:long) boolean
Specification
Description: Verifies that a student has passed a prerequisite course. Returns true if the student has passed, false otherwise. ***Input parameter(s):*** courseID:long representing the ID of the prerequisite course studentID:long representing the ID of the student to verify ***Conditions for validity:*** courseID ≥ 0 studentID ≥ 0 ***Return type:*** Boolean

Method
gatherCoursesPassed(studentID:long) list of Course objects
Specification
Description: Retrieves and returns a list of courses that a student has successfully completed. ***Input parameter(s):*** studentID:long representing the ID of the student ***Conditions for validity:*** studentID ≥ 0 ***Return type:*** list of Course objects

10.2.3.3 Specifications for methods implemented by the Section class

Method
addToList(studentID:long)
Specification
Description: Adds a student to the list of registered student for a section given the student ID. ***Input parameter(s):*** studentID:long representing the ID of the student ***Conditions for validity:*** studentID ≥ 0 ***Return type:*** void

Method
getSection(sectionID:long) Section
Specification
Description: Retrieves and returns information about a course given the section ID. ***Input parameter(s):*** sectionID:long representing the ID of the section ***Conditions for validity:*** sectionID ≥ 0 ***Return type:*** Section

Method
getSections() list of Section objects
Specification
Description: Retrieves and returns a list of section associated with a course. ***Input parameter(s):*** none ***Return type:*** list of Section objects

Method
getTimes(sectionID:long) list of int
Specification
Description: Returns a list of integers representing the starting and ending time of each time block that a section holds. ***Input parameter(s):*** sectionID:long representing the ID of the section ***Conditions for validity:*** sectionID ≥ 0 ***Return type:*** list of int

Method
removeStudent(studentID:long)
Specification
Description: Removes a student from the list of registered student of a course. ***Input parameter(s):*** studentID:long representing the ID of the student to be removed ***Conditions for validity:*** studentID ≥ 0 ***Return type:*** void

10.2.4 IPreferenceManagement

The preference management interface allows to access and modify data related to the scheduling preferences of a student. The only class implementing the methods provided by this interface is the Preferences class.

10.2.4.1 Specifications for methods implemented by the Preferences class

Method
choosePreference(pref:Preference) Sequence
Specification
Description: Adds a preference to the list of scheduling preferences of a student and returns the list of all chosen preferences. ***Input parameter(s):*** pref:Preference representing a chosen preference ***Conditions for validity:*** pref != null ***Return type:*** list of Preference objects

Method
getPreferenceList() list of Preference objects
Specification
Description: Retrieves and returns a list of preferences assigned to a student. ***Input parameter(s):*** none ***Return type:*** list of Preference objects

10.2.5 ISequenceManagement

The sequence management interface provides methods to access and modify the course sequence of a student. The only class implementing the provided methods is the Sequence class.

10.2.5.1 Specifications for methods implemented by the Sequence class

Method
getSequence() Sequence
Specification
Description: Returns an object representing a course sequence of a student. ***Input parameter(s):*** none ***Return type:*** Sequence

Method
updateSequence(courseID:long)
Specification
Description: Updates a course sequence after a course removal given the ID of the course removed. ***Input parameter(s):*** courseID:long representing the ID of the course removed ***Conditions for validity:*** courseID ≥ 0 ***Return type:*** void

11. Detailed Design

11.1 Detailed Design Diagram and Unit Descriptions

11.1.1 iUserInteraction

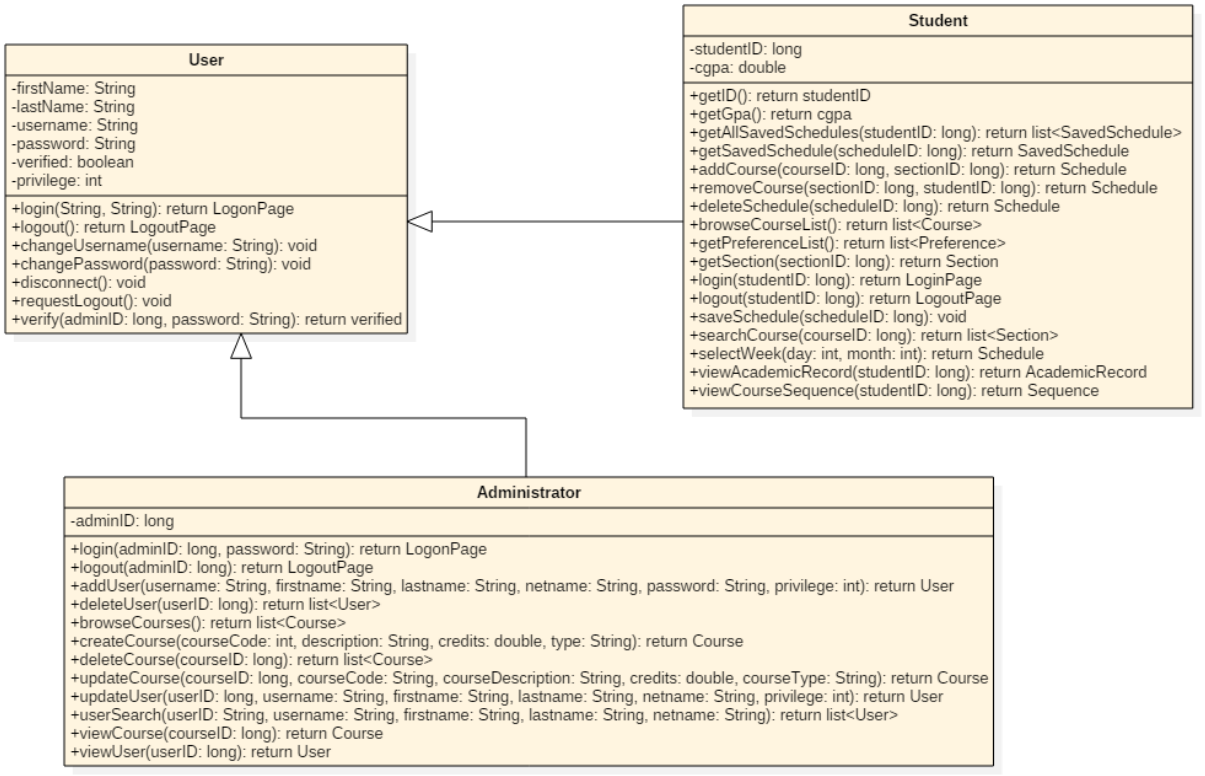


Figure 11.1.1 IUserInteraction subsystem class diagram

This subsystem is responsible to every interaction made between the User and the System, such as logging in, browsing the course list, setting preferences, adding courses, logging out, etc. The classes in this subsystem are the Student, Administrator and SavedUsers classes. Therefore, this subsystem represents the functions behind the user interface of the system.

Student Class
Description
This class models students, which are users that interact with the program directly.
Attributes
studentID: long the studentID is a unique identifier for every student cgpa: double the cgpa is the cumulative GPA of each student

Methods

- getAllSavedSchedules(studentID:long) list of Schedule objects
  returns a list of all the previous and upcoming schedules of a student
- getSavedSchedule(scheduleID:long) Schedule
  returns the saved schedule of a student
- addCourse(courseID:long, sectionID:long) Schedule
  adds a course with section sectionID to the student's sequence
- removeCourse(sectionID:long, studentID:long) Schedule
  removes a course from the student's sequence
- deleteSchedule(scheduleID: long) Schedule
  deletes the entire schedule
- browseCourseList() list of Course objects
  returns a list of all the available courses
- getPreferenceList() list of Preference objects
  returns the student's preferences
- getSection(sectionID:long) Section
  returns the section
- login(studentID:long, password:String) LogonPage
  gives the student access to the system
- logout(studentID:long) LogoutPage
  disconnects student from the system
- saveSchedule(scheduleID:long)
  saves changes to the schedule
- searchCourse(courseID:long) list of Section objects
  takes a course ID as input and returns the matched results
- selectWeek(day:int, month:int) Schedule
  returns a schedule of the selecteddate
- viewAcademicRecord(studentID:long) AcademicRecord
  returns the student's academic record
- viewCourseSequence(studentID:long) Sequence
  returns the student's sequence

Administrator Class
Description
Administrators are users that interact with the program directly. They log in, make changes in the schedule, then log out.
Attributes
adminID: long the adminID is a unique identifier for each administrator
Methods
❖ login(adminID:long, password:String) LogonPage ❖ logout(adminID:long) LogoutPage ❖ addUser(username:String, firstname:String, lastname:String, netname: String, password:String, priviledge:int) User ❖ creates and adds user to userlist ❖ deleteUser(userID:long) list<User> ❖ removes user with ID userID and returns the list of users ❖ browseCourses() list<Course> ❖ returns list of courses ❖ createCourse(courseCode:int, description:String, credits:Double, type:String) Course ❖ creates a course ❖ deleteCourse(courseID:long) list<Course> ❖ deletes course with ID courseID ❖ updateCourse(courseCode:int, description:String, credits:Double, type:String) Course ❖ changes a course information ❖ updateUser(username:String, firstname:String, lastname:String, netname: String, password:String, priviledge:int) User ❖ changes a user's information ❖ userSearch(userID:String, username:String, firstname:String, lastname:String, netname: String) list<User> ❖ searches for users with according information and returns a list ❖ viewCourse(courseID:long) Course ❖ returns course with ID courseID ❖ viewUser(userID:long) User ❖ returns user with ID userID

SavedUsers Class
Description
SavedUsers is the class that verifies if a user is legitimate or not, and allows it to logout.
Attributes
verified: Boolean this attribute is the state of the user which determines whether his login is successful or not
Methods
❖ disconnect() void disconnects the student from the system ❖ requestLogout() void makes a request to disconnect ❖ verify(adminID: long, password: String) verified verifies if the login info is correct

11.1.2 IScheduleManagement

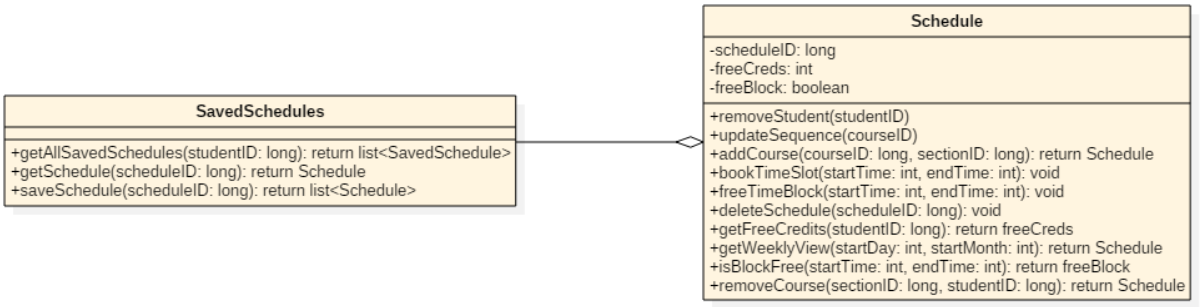


Figure 11.1.2 IScheduleManagement subsystem class diagram

The Schedule Management subsystem, consisting of the Schedule and SavedSchedules classes, is responsible for all schedule operations. In other words, whenever another one of the subsystems wants to modify or interact with a schedule or the list of schedules, they interact with this subsystem which in turn does the operations.

Schedule Class
Description
This class manages courses in a schedule, and ensures that courses don't overlap
Attributes
scheduleID: long every schedule has a unique long identifier freeCreds: int freeCreds is the amount of remaining free credits that the student can take in a semester freeBlock: boolean freeBlock is a Boolean which determines whether the time for a class is free or not

Methods
❖ removeStudent(studentID) removes the student ❖ updateSequence(courseID) updates the sequence after changes are made ❖ addCourse(courseID:long, sectionID:long) Schedule adds a course to the schedule ❖ bookTimeSlot(startTime:int, endTime:int) books a timeslot for a course to prevent overlap ❖ freeTimeBlock(startTime:int, endTime:int) frees a timeslot that was booked ❖ deleteSchedule(scheduleID:long) deletes a schedule ❖ getFreeCredits(studentID:long) int returns the amount of credits still available ❖ getWeeklyView(startDay:int, startMonth:int) Schedule returns a weekly view of the schedule ❖ isBlockFree(startTime:int, endTime:int) Boolean computes if the timeslot is available and returns true or false ❖ removeCourse(sectionID:long, studentID:long) Schedule removes a course from the schedule

SavedSchedules
Description
This class manages schedules as a whole. It could get a schedule, save it, or get a list off all schedules that are in the system.
Attributes
None
Methods
❖ getAllSavedSchedules(studentID:long) list of Schedule objects returns all schedules available in the system ❖ getSchedule(scheduleID:long) Schedule returns a single schedule that has the id scheduleID ❖ saveSchedule(scheduleID:long) list of Schedule objects saves changes made to schedules

11.1.3 ICourseManagement

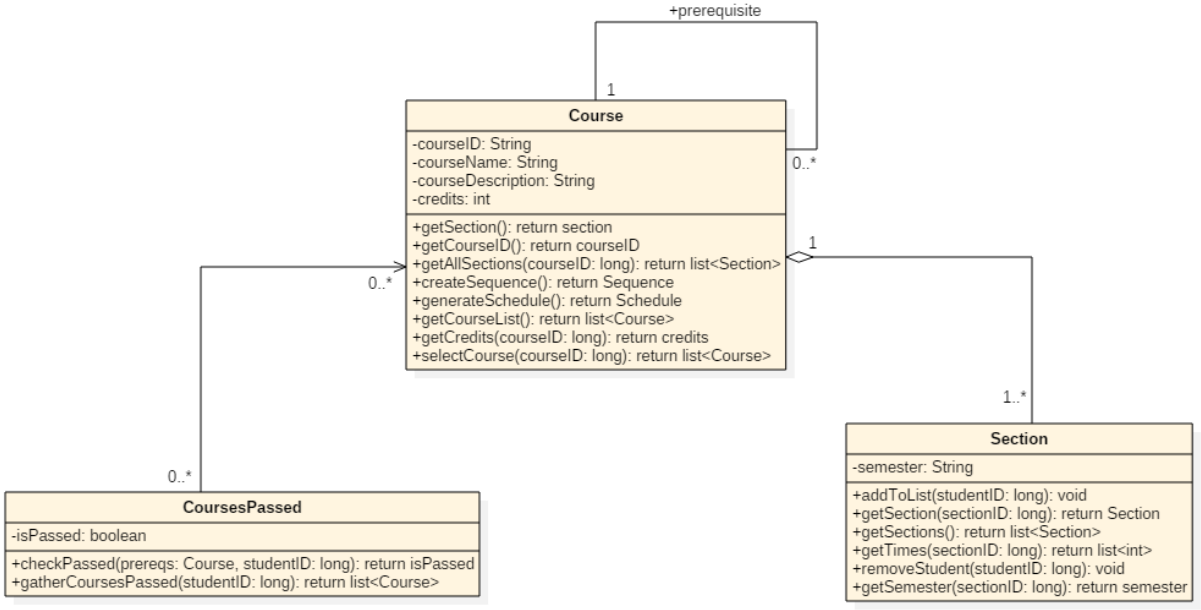


Figure 11.1.3 ICourseManagement subsystem class diagram

The Course, CoursesPassed, Section and Prerequisite classes make up this subsystem. Because the Courses are much more complex than a name (they contain sections, times, prerequisites, etc.), we cannot have a single class take care of everything. Instead, we have the Course Management subsystem which uses the above classes to modify and interact with the Courses.

Course Class
Description
The Course class contains all the information on the classes available.
Attributes
courseID: String every course has a unique String identifier courseName: String every course has a name courseDescription: String courses have a String description credits: int this attribute is an integer value of the credits the course is worth
Methods
❖ getSection() section returns a section of a class ❖ getCourseList() list of Course objects returns a list of available courses ❖ getAllSections(courseID:long) list of Section objects returns a list of all available sections ❖ createSequence() Sequence initializes a sequence ❖ generateSchedule() Schedule generates a schedule based on preferences ❖ getCourseList() list<Course> gets a list of courses in a schedule ❖ getCredits(courseID:long) int returns the credits of a course ❖ selectCourse(courseID:long) list of Section objects and Prerequisite objects selects a course that has the ID courseID

CoursesPassed Class
Description
This class is responsible for confirming and returning the passed courses.
Attributes
isPassed: Boolean this attribute is a Boolean value that determines if a course is already passed or not
Methods
❖ checkPassed(prereqs:Course, studentID:long) boolean checks if a course is passed by student with ID studentID ❖ gatherCoursesPassed(studentID:long) list of Course objects returns a list of passed courses for student with ID studentID

Section Class
Description
A course can have multiple sections; this class contains the different sections and is responsible for adding and removing students from the sections.
Attributes
semester: String the semester String is a value which determines which semester the course is in
Methods
❖ addToList(studentID:long) adds a student to a section ❖ getSection(sectionID:long) Section returns a section with the ID sectionID ❖ getSections() list of Section objects returns a list of available sections ❖ getTimes(sectionID:long) list of int returns the time of section with ID sectionID ❖ removeStudent(studentID:long) removes student with ID studentID from a section

Prerequisite Class
Description
Each course has a list of prerequisites. This class returns this list.
Attributes
None
Methods
❖ getPrerequisites() list of Prerequisite objects returns the prerequisites of a course

11.1.4 IPreferenceManagement

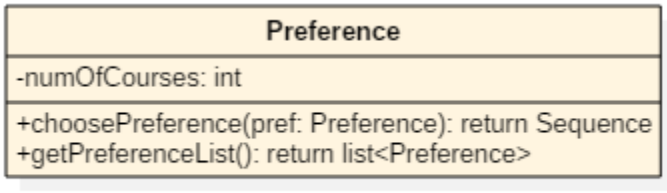


Figure 11.1.4 IPreferenceManagement subsystem class diagram

Though this subsystem only contains one class: the Preference class, tt is responsible for taking the students preferences and giving it to another subsystem to help create a schedule according to the student's needs. These preferences dictate the sections that are shown to the student in order to choose from to build their schedules.

Preference Class
Description
This class manages the user's preference and generates a sequence, and is capable of returning all the preferences in the system.
Attributes
numOfCourses: int numOfCourses is an integer attribute that determines how many courses a student wishes to take
Methods
❖ choosePreference(pref:Preference) Sequence applies the preferences set by the user and returns a sequence ❖ getPreferenceList() list of Preference objects gets the preferences set by the user

11.1.5 ISequenceManagement

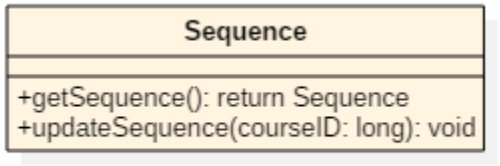


Figure 11.1.5 ISequenceManagement subsystem class diagram

This subsystem also contains one class, the Sequence class. It is responsible for returning the Sequence of a student to a user who wishes to view it, or to update the sequence by removing courses (when a student completes a course). The sequence dictates which classes a student may take in any given semester, and is updated when a student adds or drops a class on their schedule.

Sequence Class
Description
This class allows other classes to get the Sequence of a student, and the class also is capable of updating the sequence whenever a course is completed.
Attributes
None
Methods
❖ getSequence() Sequence returns a sequence of a student ❖ updateSequence(courseID:long) whenever a course is passed, this method is used to remove the course from the sequence

12. Dynamic Design Scenarios

12.1 Use Case 1: Login to TimeTurner

12.1.1 Fully Dressed Use Case

Use Case ID:	UC1	
Use Case Name:	Login to TimeTurner	
Created By:	Marc-Andre Leclair	Last Updated By: Claudia Della Serra
Date Created:	January 25th, 2016	Last Revision Date: February 8, 2016
Actor(s):	Student, Administrator	
Goal/Actor Goals:	A student or Administrator wants to Login in their account	
Description/Summary:	The Student or Administrator wants to login into their account. The initial webpage contains a login box where the he or she can enter their username and password. The request is sent and it is check in the database whether or not the account exist or if the password is correct	
Preconditions:	❖ The user has an internet connection. ❖ The user has valid login credentials.	
Post-conditions:	The user is authenticated and logged in.	
Minimum Guarantee:	User will not log in	
Basic Flow:	1. Student enters its login information 2. System verifies the login information 3. Student is logged in and brought to the home page.	
Risk assessment:	Low	
Importance assessment:	5/5	

12.1.2 5.1.2 System Sequence Diagrams

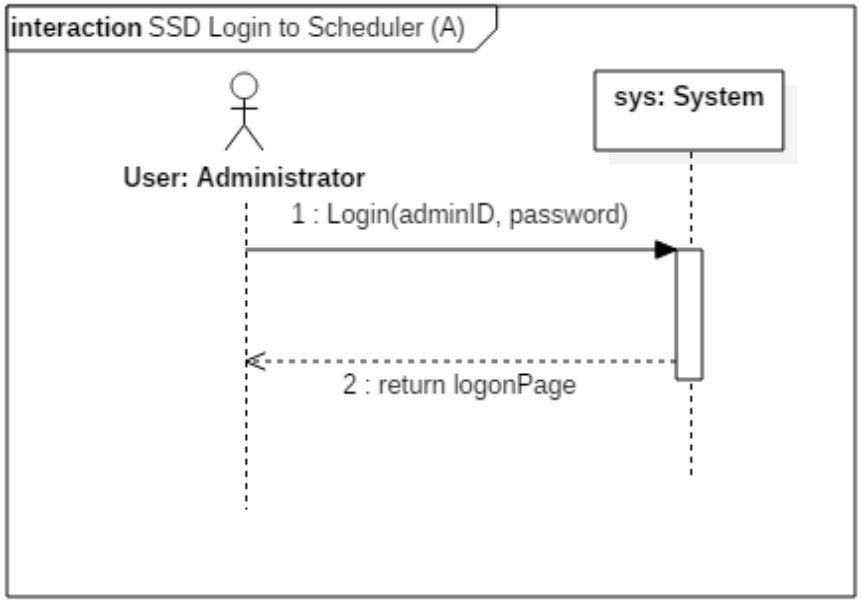


Figure 12.1.1 Administrator login system sequence diagram

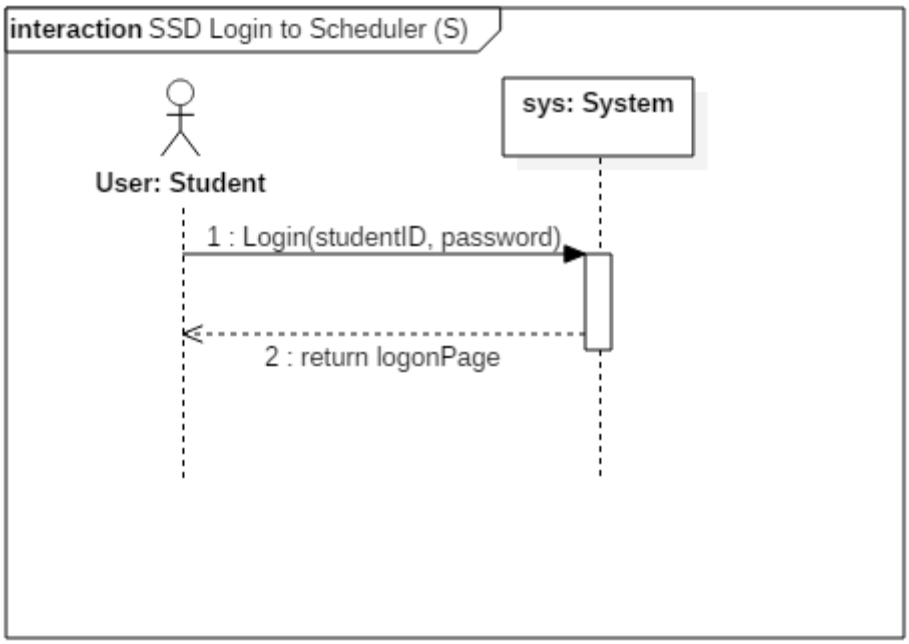


Figure 12.1.2 Student login system sequence diagram

12.1.3 5.1.3 Contract Diagrams

Contract 1.1	Login	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	login(adminID: long)	
Cross-reference:	UC1	
Preconditions:	❖ The user has an internet connection. ❖ The user has valid login credentials.	
Post-conditions:	❖ An Administrator is created. (instance creation) ❖ The Administrator object is associated to the client. (association formed)	

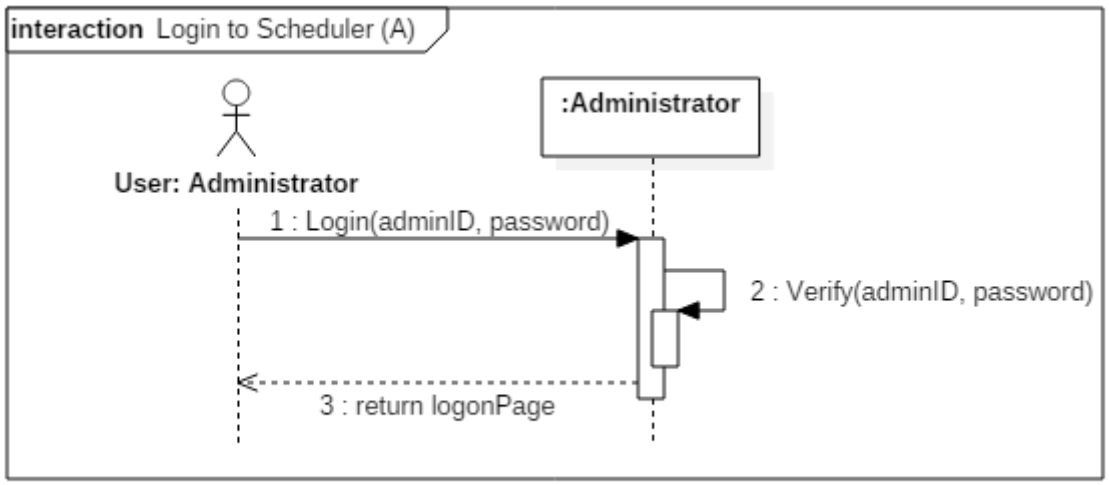


Figure 12.1.3 Administrator login contract diagram

Contract 1.2	Login	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	login(studentID: long)	
Cross-reference:	UC1	
Preconditions:	❖ The user has an internet connection. ❖ The user has valid login credentials.	
Post-conditions:	❖ A Student is created. (instance creation) ❖ The Student object is associated to the client. (association formed)	

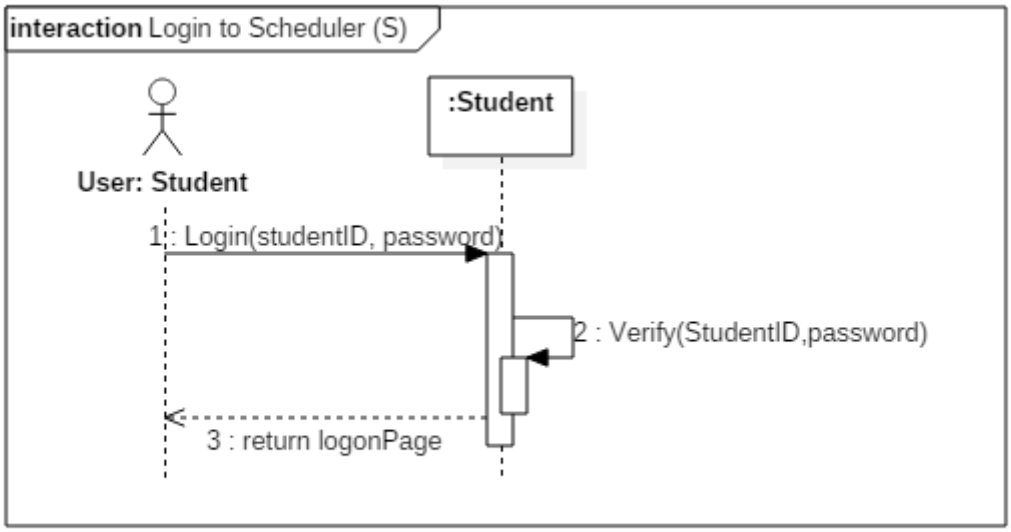


Figure 12.1.4 Student login contract diagram

12.2 Use Case 2 : Logout of TimeTurner

12.2.1 Fully Dressed Use Case

Use Case ID:	UC2	
Use Case Name:	Logout of Scheduler	
Created By:	Marc-Andre Leclair	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 17, 2016
Actor(s):	Student, Administrator	
Goal/Actor Goals:	A student or Administrator wants to Logout of their account.	
Description/Summary:	The Student or Administrator wants to logout of their account. He or she sends a request to the server to close the connection between themselves and the Time Turner.	
Preconditions:	The user is logged in their account	
Post-conditions:	The user is disconnected from the server and logged out.	
Minimum Guarantee:	User will remain logged in	
Basic Flow:	1. The user indicates that they wish to logout of the system. 2. The server terminates the connection. 3. The user is logged out through the webpage.	
Risk assessment:	Low	
Importance assessment:	5/5	

12.2.2 System Sequence Diagrams

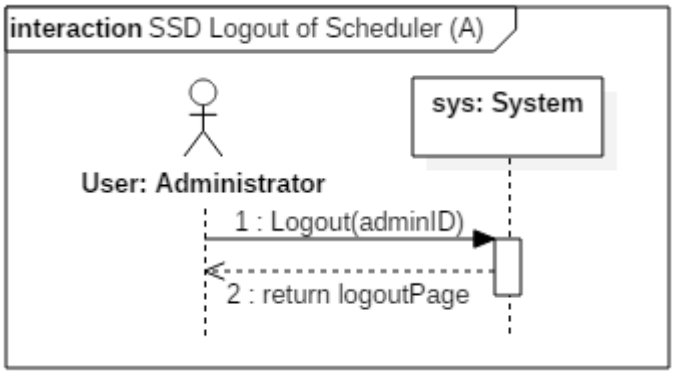


Figure 12.2.1 Administrator logout system sequence diagram

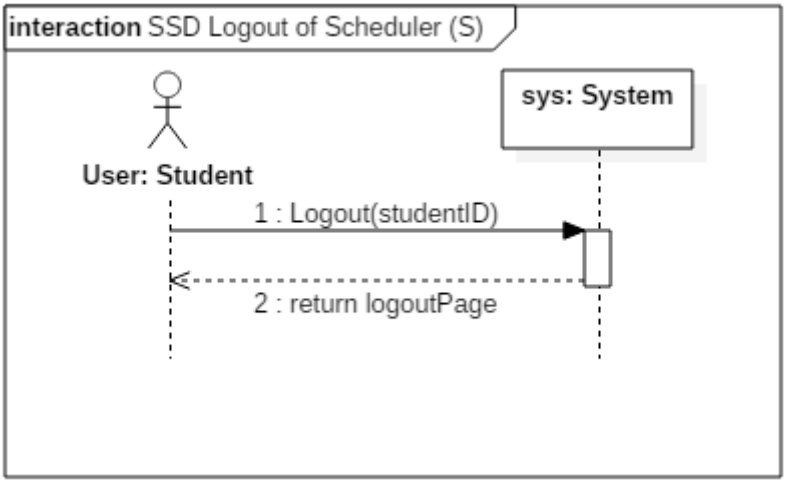


Figure 12.2.2 Student logout system sequence diagram

12.2.3 Contract Diagrams

Contract 2.1	Logout	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	logout(adminID: long)	
Cross-reference:	UC2	
Preconditions:	❖ The user is logged into their account.	
Post-conditions:	❖ The Administrator is deleted (instance Deletion) ❖ The user is disconnected from the Administrator object (association removed)	

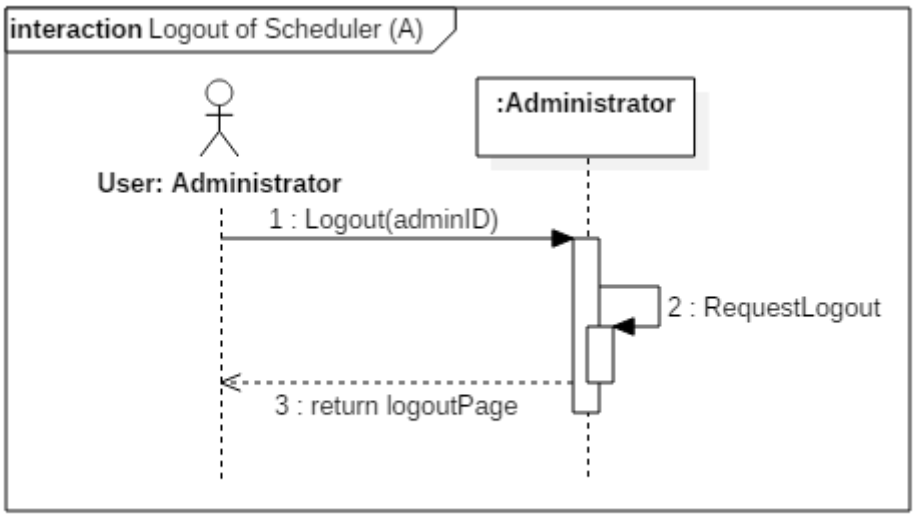


Figure 12.2.3 Administrator logout contract diagram

Contract 2.2	**Logout**	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	logout(studentID: long)	
Cross-reference:	UC2	
Preconditions:	❖ The user is logged into their account.	
Post-conditions:	❖ The Student is deleted (instance Deletion) ❖ The user is disconnected from the Student object (association removed)	

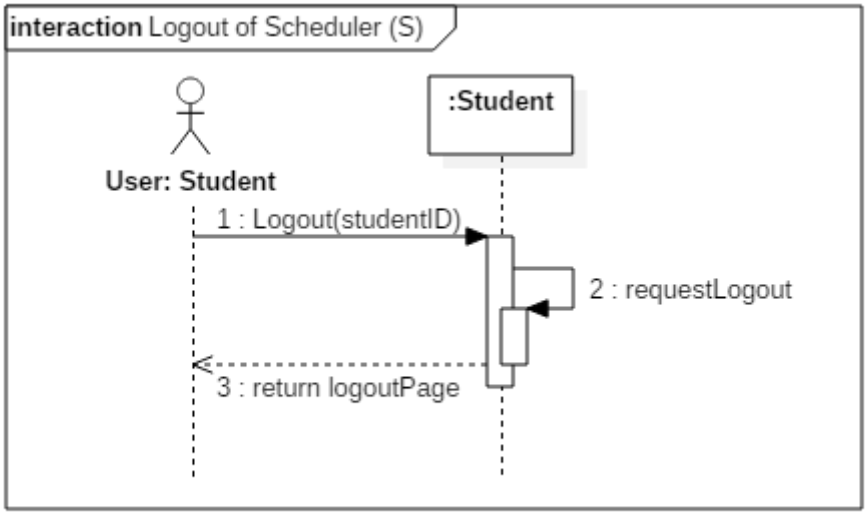


Figure 12.2.4 Student logout contract diagram

12.3 Use Case 4: Browse Course List

12.3.1 Fully Dressed Use Case

Use Case ID:	UC4	
Use Case Name:	Browse Course List	
Created By:	Ideawin-Bunthy Koun	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	Browse the list of courses from the course calendar.	
Description/Summary:	The user wishes to view available courses and access information pertaining to them. The system must display such information, including sections, times, and locations.	
Preconditions:	❖ User must be logged in as a Student. ❖ Course list must be available.	
Post-conditions:	A list of courses is displayed.	
Minimum Guarantee:	The system fails to display a list of courses and displays an error message to the user.	
Basic Flow:	3. Student selects the 'Browse Course List" feature. 4. System retrieves list of courses. 5. System displays list of courses. 6. User selects a course from that list. 7. System displays information about the selected course.	
Risk assessment:	Medium	
Importance assessment:	3/5	

12.3.2 System Sequence Diagram

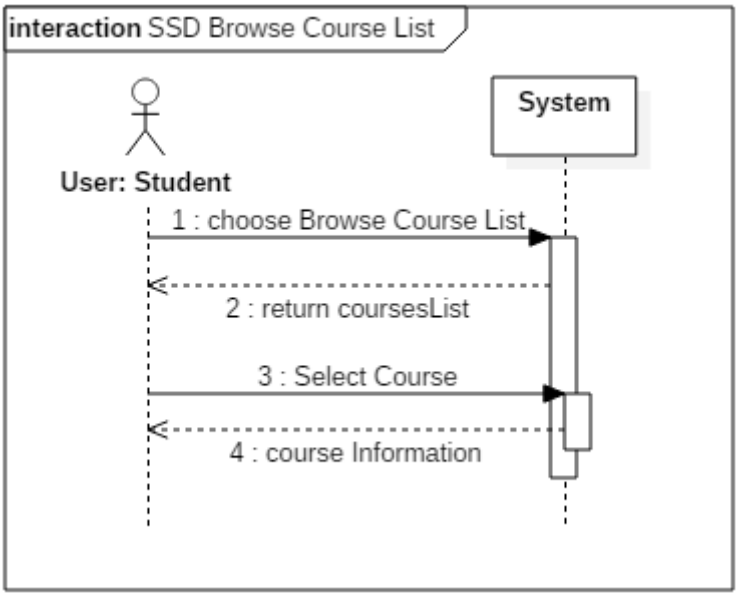


Figure 12.4.1 Browse course list system sequence diagram

12.3.3 Contract Diagrams

Contract 4.1	browseCourseList	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	getCourseList( )	
Cross-reference:	UC4	
Preconditions:	❖ User must be logged in as a Student. ❖ Course list must be available.	
Post-conditions:	❖ An instance of a list of all Course objects is created (instance creation) ❖ The list of all Course objects is associated to the user (association formed)	

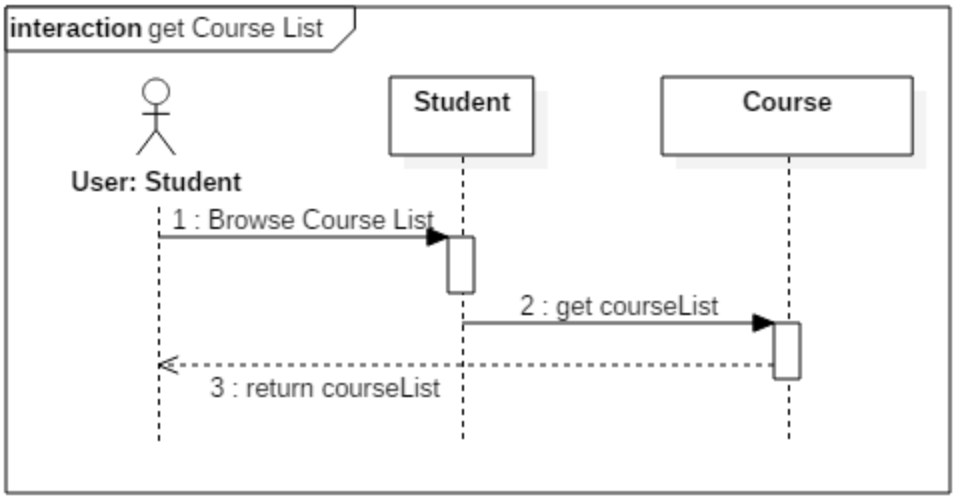


Figure 12.4.2 getCourseList contract diagram

Contract 4.2	**selectCourse**	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	selectCourse(courseID: long)	
Cross-reference:	UC4	
Preconditions:	❖ User must be logged in as a Student. ❖ Course list must be available.	
Post-conditions:	❖ An instance of a Section is created (instance creation) ❖ The Section object is associated to the user (association formed)	

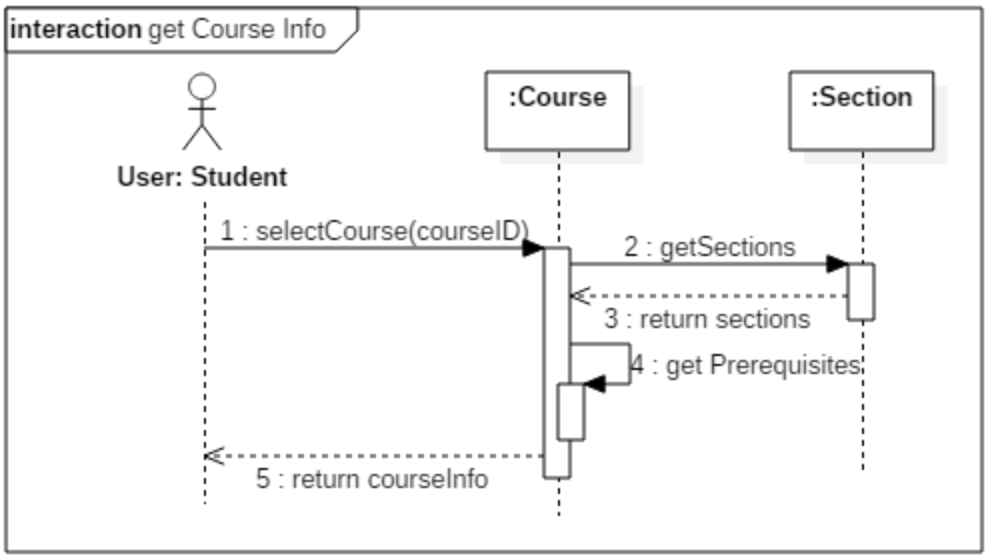


Figure 12.4.3 getCourseInfo contract diagram

12.4 Use Case 6: Generate Schedule

12.4.1 Fully Dressed Use Case

Use Case ID:	UC6	
Use Case Name:	Generate Schedule	
Created By:	Ideawin-Bunthy Koun	Last Updated By: Ryan Lee
Date Created:	January 25th, 2016	Last Revision Date: March 9, 2016
Actor(s):	Student	
Goal/Actor Goals:	Obtain a suggested schedule for the next four years.	
Description/Summary:	The system can generate a 4-year schedule for the user based on the student's preferences and constraints and on prerequisite and co-requisite policy.	
Preconditions:	❖ User must be logged in as a Student. ❖ The system must have access to course database. ❖ The system must have access to the student's records.	
Post-conditions:	The system generates a schedule with no overlaps.	
Minimum Guarantee:	The system fails to generate a schedule and displays an error message to the user.	
Basic Flow:	1. User selects the 'Generate Schedule' feature. 2. System retrieves list of preferences. 3. User selects his or her options and preferences. 4. System prompts user to confirm preferences. 5. System responds by displaying a schedule that meets the user's preferences and constraints.	
Risk assessment:	High	
Importance assessment:	5/5	

12.4.2 System Sequence Diagram

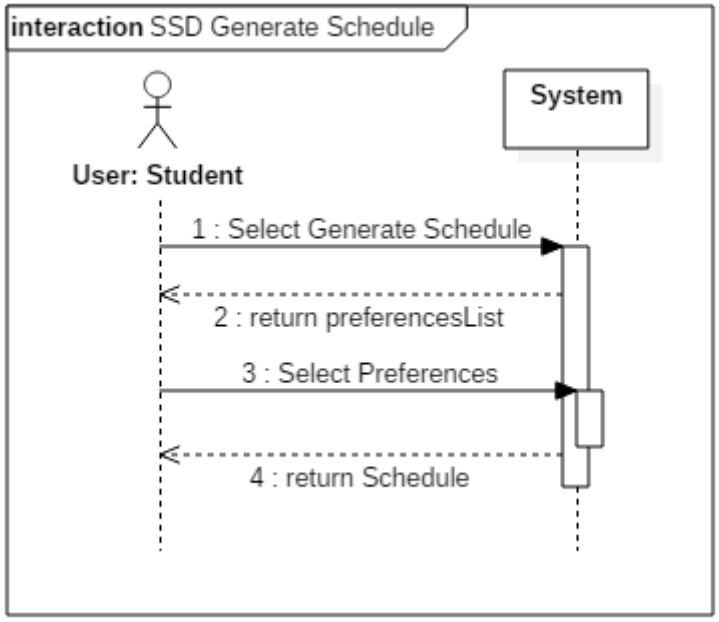


Figure 12.6.1 Generate Schedule System Sequence Diagram

12.4.3 Contract Diagrams

Contract 6.1	getPreferenceList	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	getPreferenceList()	
Cross-reference:	UC6	
Preconditions:	❖ User must be logged in as a Student. ❖ The system must have access to course database. ❖ The system must have access to the student's records.	
Post-conditions:	❖ A list of preferences is created (instance creation) ❖ The list of preferences is associated to the user (association formed)	

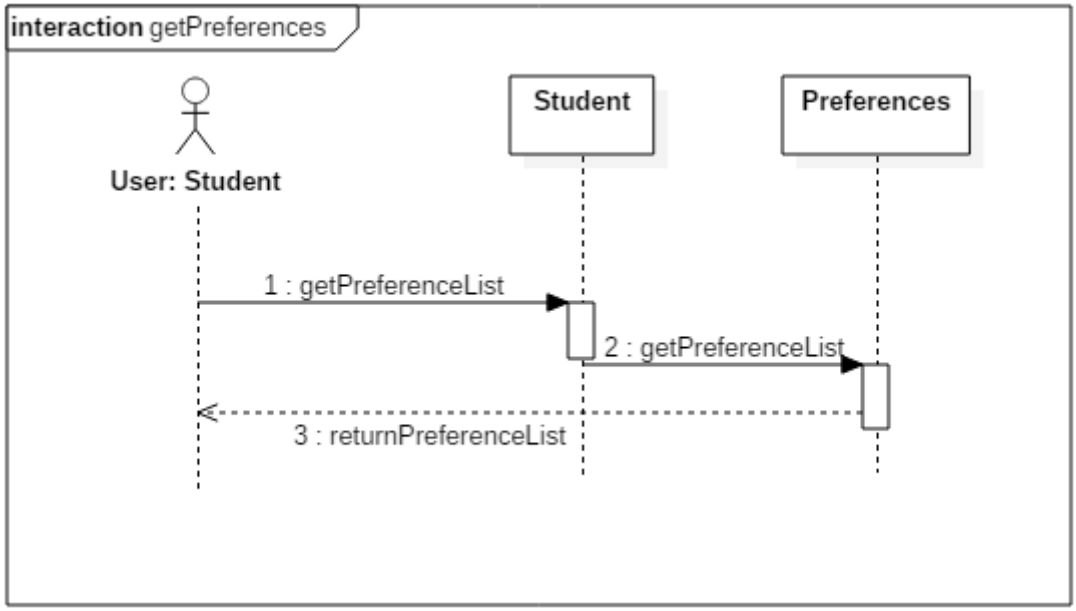


Figure 12.6.2 getPreferences contract diagram

Contract 6.2	choosePreferences	
Created By:	Ryan Lee	Last Updated By: Ryan Lee
Date Created:	March 17, 2016	Last Revision Date: March 17, 2016
Operation:	choosePreferences (studentID: long)	
Cross-reference:	UC6	
Preconditions:	❖ User must be logged in as a Student. ❖ The system must have access to course database. ❖ The system must have access to the student's records.	
Post-conditions:	❖ An instance of Schedule, schd, will be created (instance creation) ❖ Schd will be associated to the Student (association formed) ❖ Schd will be added to student's saved schedules. (attribute modification)	

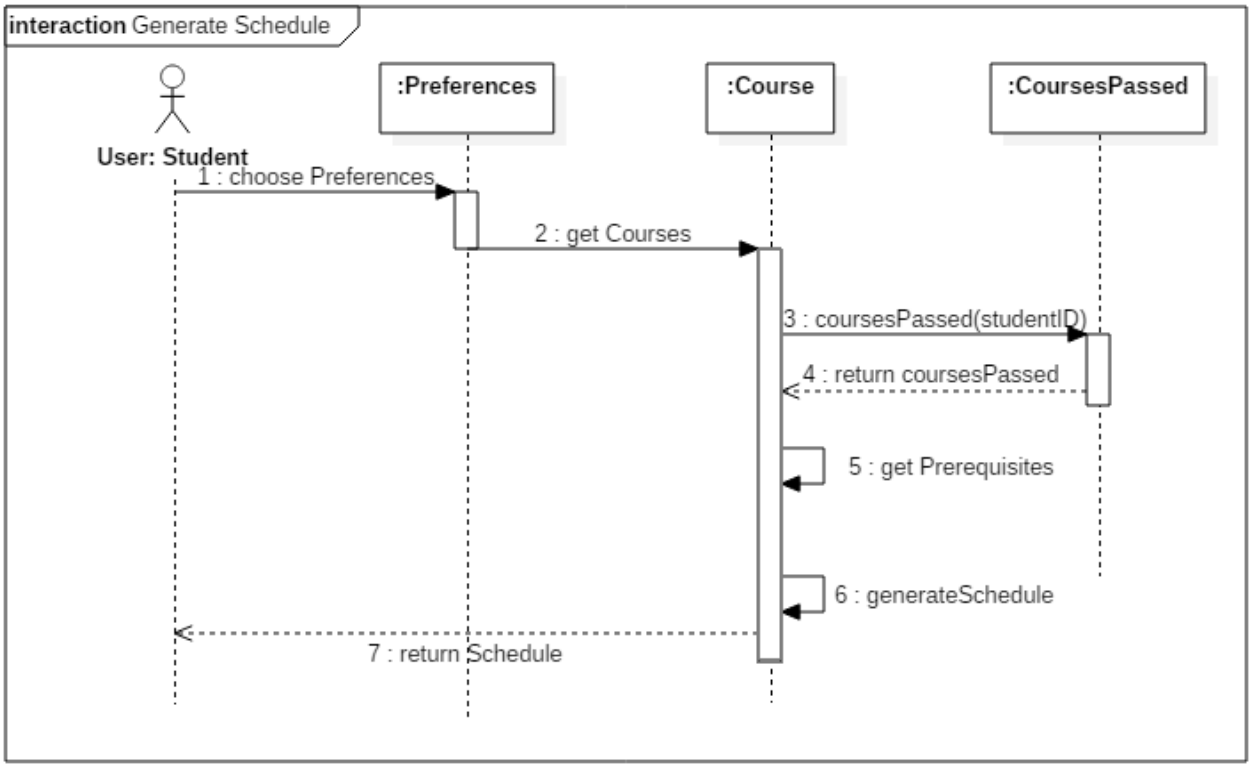


Figure 12.6.3 generateSchedule contract diagram

12.5 Use Case 7 – View Saved Schedules

12.5.1 Fully Dressed Use Case

Use Case ID:	UC7	
Use Case Name:	View Saved Schedules	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to view a previously saved schedule.	
Description/Summary:	The user selects a schedule from a list of their previously saved schedules. The system must display this schedule for the user to review.	
Preconditions:	❖ User is logged into the system. ❖ User has previously generated one or more schedules ❖ User has saved at least one generated schedule	
Post-conditions:	The system displays the saved schedule	
Minimum Guarantee:	Saved schedules remain in the system unchanged.	
Basic Flow:	5. The user indicates their wish to view a saved schedule 6. The system prompts the user with a list of saved schedules 7. The user selects which semester's schedule to view 8. The system displays the chosen schedule	
Risk assessment:	Medium	
Importance assessment:	3/5	

12.5.2 System Sequence Diagram

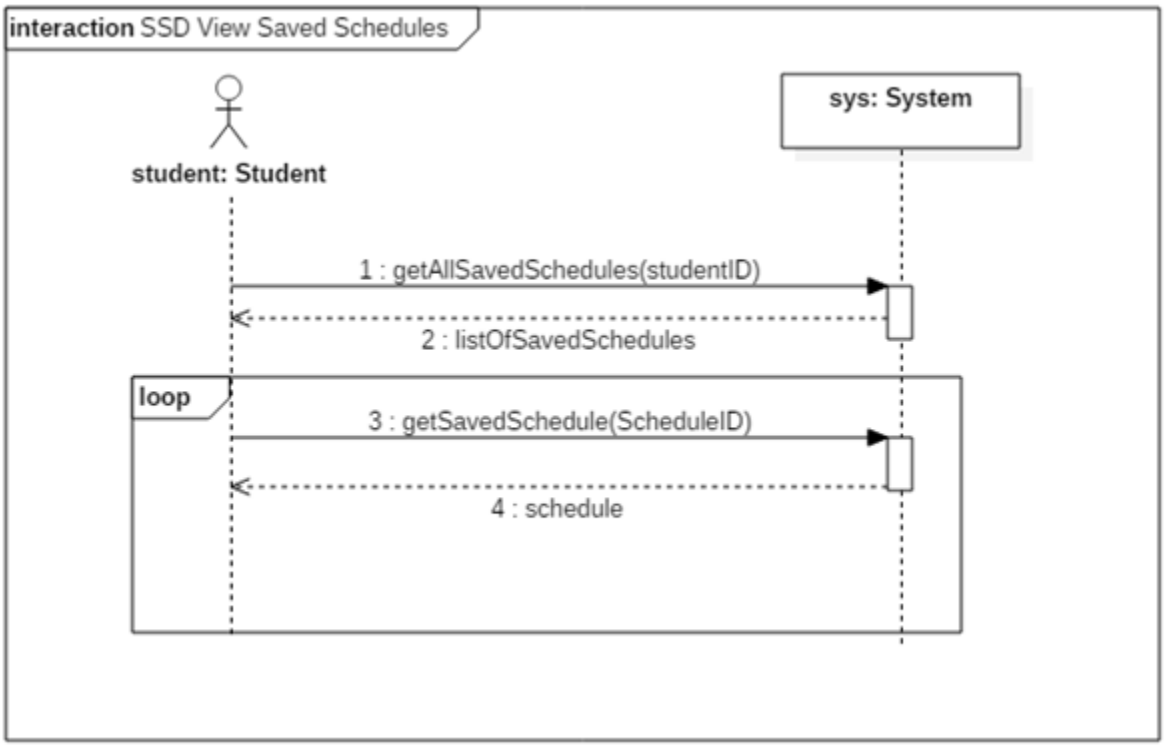


Figure 12.7.1 View Saved Schedules system sequence diagram

12.5.3 Contract Diagrams

Contract 7.1	getAllSavedSchedules	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	viewSavedSchedules(studentID: long) : list<SavedSchedule>	
Cross-reference:	UC7	
Preconditions:	❖ User is logged into the system. ❖ User has previously generated one or more schedules ❖ User has saved at least one generated schedule	
Post-conditions:	• A list of SavedSchedules is generated (instance creation) • The list is returned to the Student (association creation)	

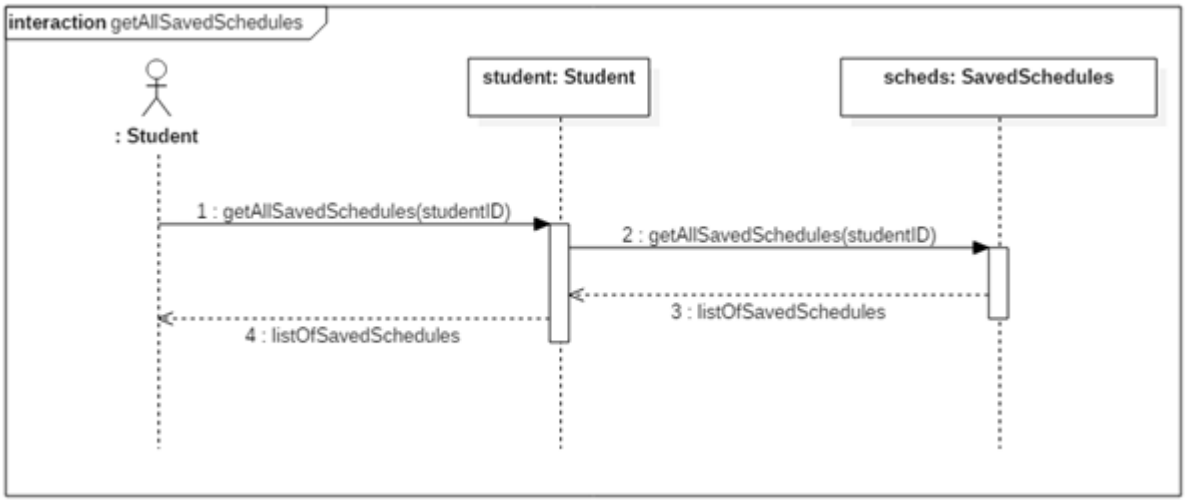


Figure 12.7.2 getAllSavedSchedules contract diagrams

Contract 7.2	**getSavedSchedule**	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	getSavedSchedule(scheduleID: long) : Schedule	
Cross-reference:	UC7	
Preconditions:	❖ User is logged into the system. ❖ User has previously generated one or more schedules ❖ User has saved at least one generated schedule	
Post-conditions:	• A previously saved schedule is retrieved (instance creation) • The saved schedule is returned to the Student (association Creation)	

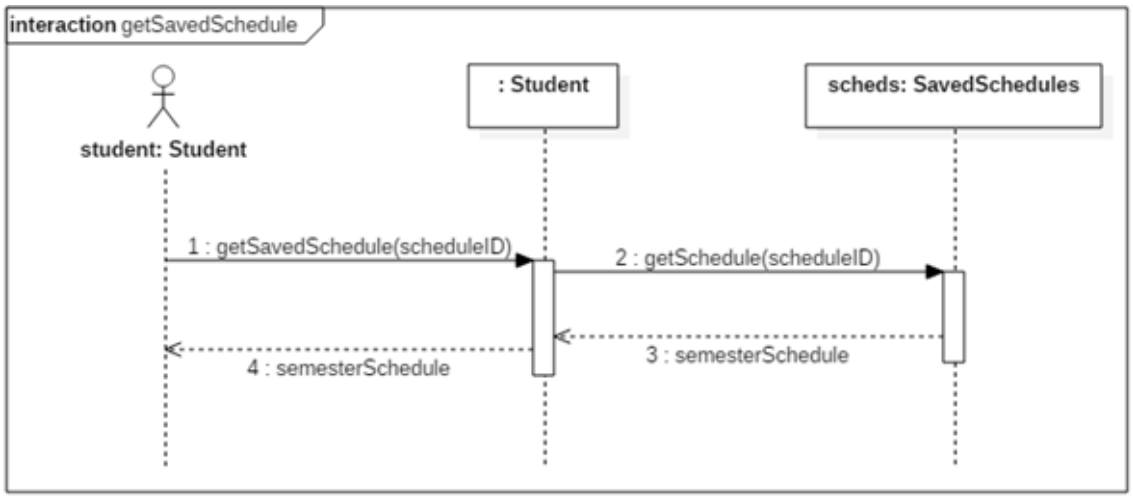


Figure 12.7.3 getSavedSchedule contract diagram

12.6 Use Case 8 – View Academic Record

12.6.1 Fully Dressed Use Case

Use Case ID:	UC8	
Use Case Name:	View Academic Record	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to view a list of his or her completed courses and grades for those courses.	
Description/Summary:	The system displays the user's academic record, which includes completed courses and courses currently being taken.	
Preconditions:	❖ User is logged into the system. ❖ User has completed at least one course and/or is presently enrolled in at least one course.	
Post-conditions:	The system displays the user's academic record.	
Minimum Guarantee:	The academic record remains in the system unmodified.	
Basic Flow:	6. The user requests to view their academic record 7. The system generates a list of the user's completed courses, as well as courses in which the user is currently enrolled.	
Risk assessment:	Low	
Importance assessment:	1/5	

12.6.2 System Sequence Diagram

Figure 12.8.1 View Academic Record System Sequence Diagram

12.6.3 Contract Diagrams

Contract 8.1	viewAcademicRecord	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	viewAcademicRecord (studentID: long) : list<Course>	
Cross-reference:	UC8	
Preconditions:	❖ User is logged into the system. ❖ User has completed or is presently enrolled in at least one course	
Post-conditions:	❖ A list of classes passed is dynamically created for the student (instance creation) ❖ The academic record is returned to the student (association creation)	

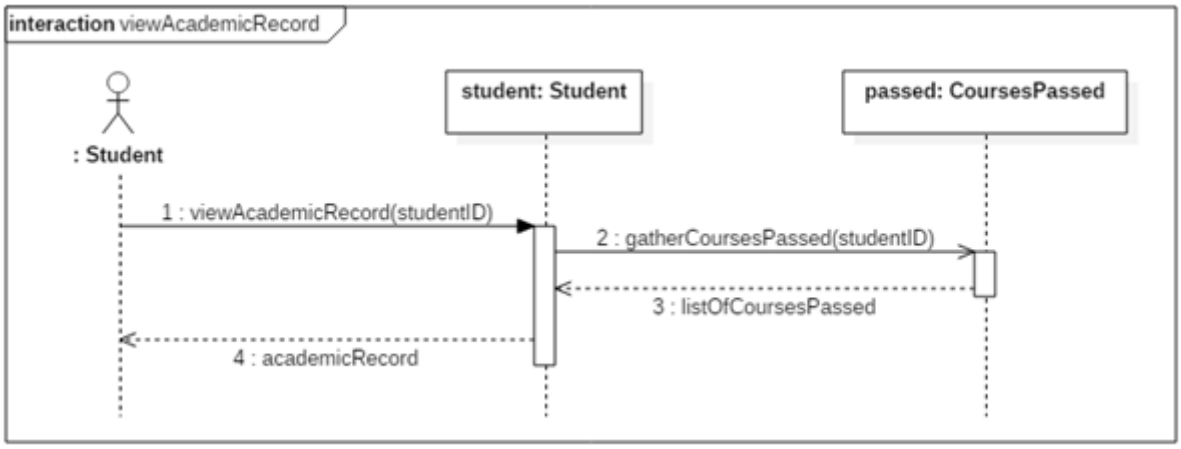


Figure 12.8.2 viewAcademicRecord contract diagram

12.7 Use Case 9 – Drop Course

12.7.1 Fully Dressed Use Case

Use Case ID:	UC9	
Use Case Name:	Drop Course	
Created By:	Erin Benderoff	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to remove a course from a generated schedule.	
Description/Summary:	The user selects which course he or she would like to remove from the schedule. The system must delete this course from the schedule.	
Preconditions:	❖ User is logged into the system. ❖ User has generated a schedule containing at least one course. ❖ User has chosen to view a saved schedule	
Post-conditions:	The chosen course has been removed from the user's schedule.	
Minimum Guarantee:	The courses on the user's schedule remain unchanged.	
Basic Flow:	1. User selects a schedule they wish to view 2. System returns the desired Schedule 3. The user chooses a course from his or her schedule to view. 4. The system displays the course information 5. The user indicates they wish to remove the course from the schedule 6. The System prompts the user for confirmation. 7. The user confirms dropping of the course 8. The produces a modified schedule without the removed course.	
Risk assessment:	High	
Importance assessment:	4/5	

12.7.2 System Sequence Diagram

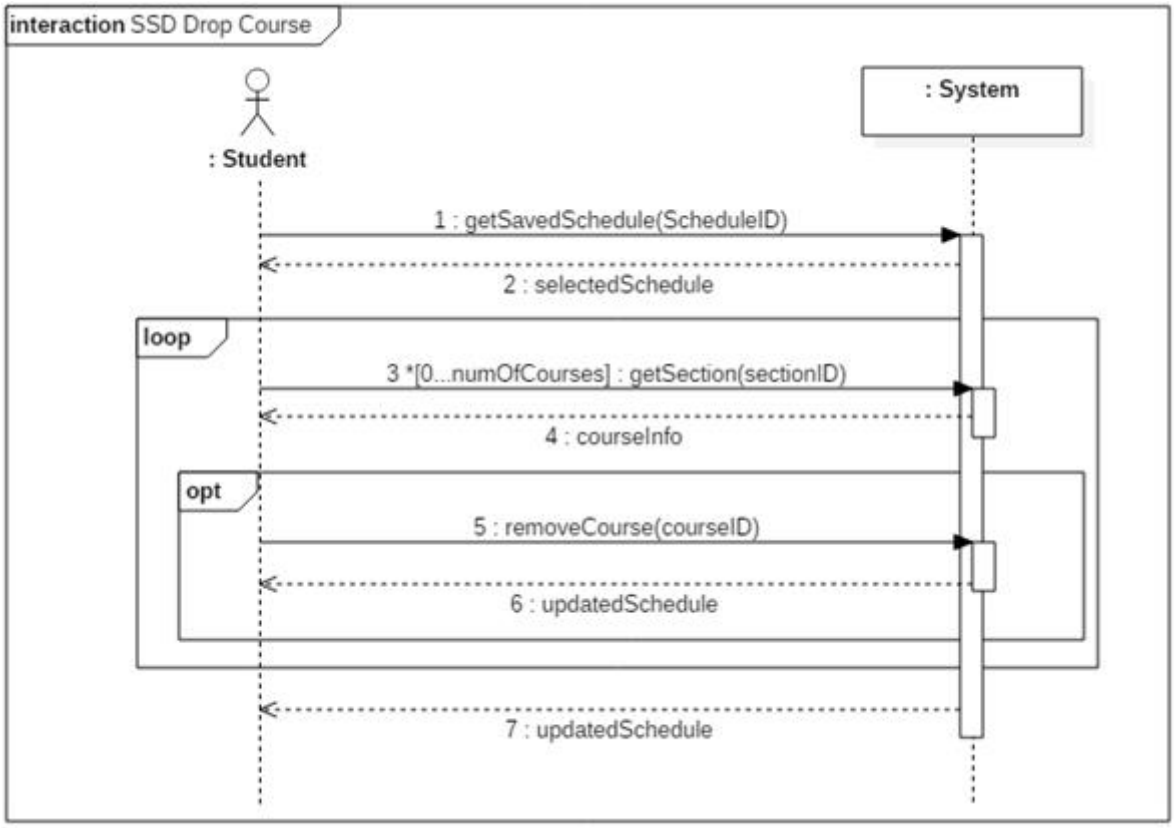


Figure 12.9.1 Drop Course system sequence diagram

12.7.3 Contract Diagrams

Contract 9.1	getSavedSchedule	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	getSavedSchedule(scheduleID: long)	
Cross-reference:	UC9, UC7, Contract 7.2	
Preconditions:	❖ User is logged into the system. ❖ User has generated and saved a schedule ❖ User has requested to view a list of saved schedules	
Post-conditions:	❖ An instance of a saved schedule is retrieved (instance creation) ❖ The schedule is returned to the Student (association creation)	

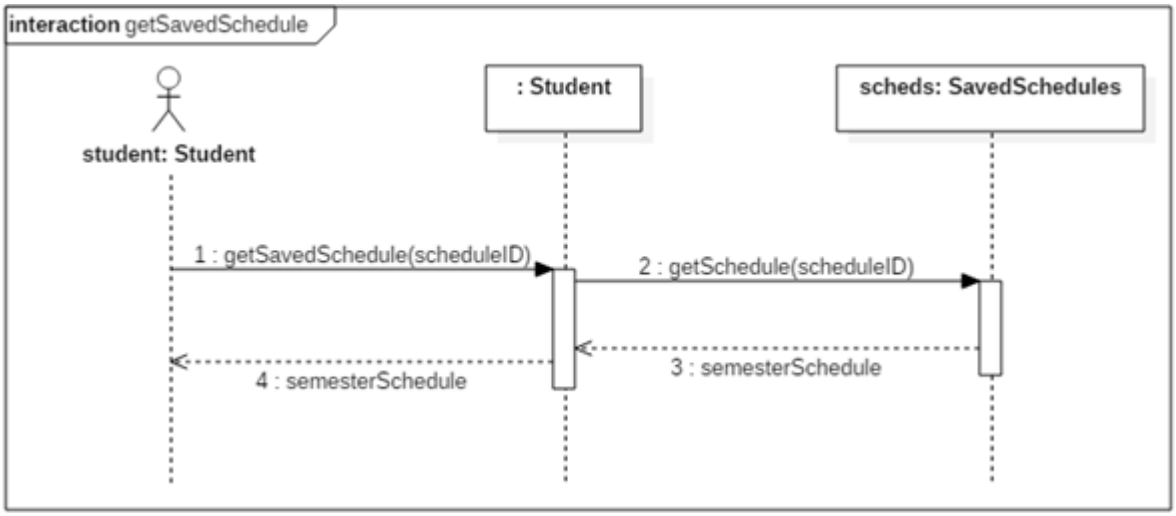


Figure 12.9.2 getSavedSchedule contract diagram

Contract 9.2	**getSection**	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	viewCourseInfo(sectionID: long) : Section	
Cross-reference:	UC9	
Preconditions:	❖ User is logged into the system. ❖ User has generated and saved a schedule ❖ User has requested to view a saved schedule	
Post-conditions:	❖ The desired Section object is created (instance creation) ❖ The Section is returned to the Student (association formed)	

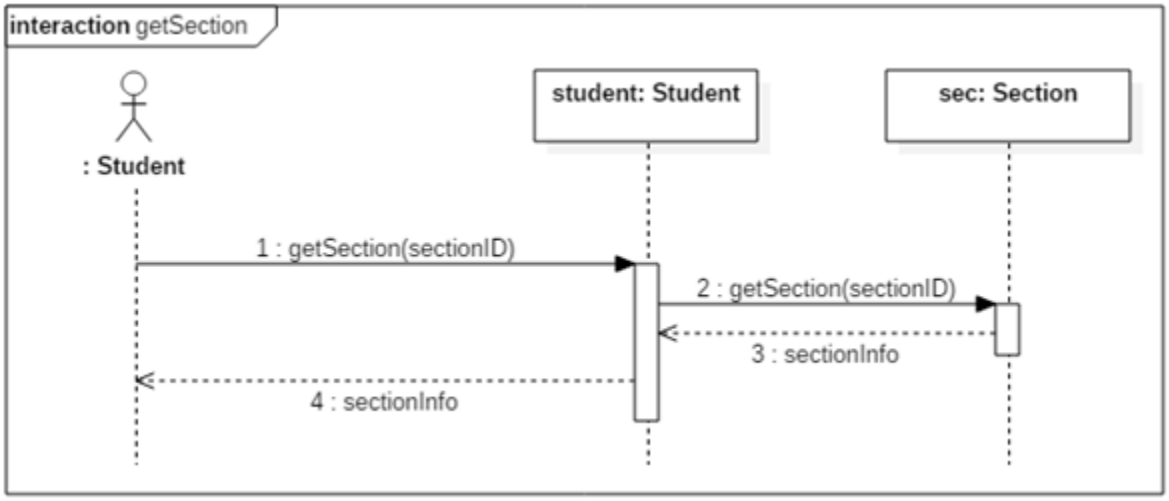


Figure 12.9.3 getSection contract diagram

Contract 9.3	**removeCourse**	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	removeCourse(sectionID: long, studentID: long) : Schedule	
Cross-reference:	UC9	
Preconditions:	❖ User is logged into the system. ❖ User has generated and saved a schedule ❖ User has requested to view a saved schedule ❖ User has requested to view course registration info	
Post-conditions:	❖ The student will be removed from the section's registration list (attribute modification, association removed) ❖ The course will be removed from the student's schedule (association modification) ❖ The student's course sequence will be updated (attribute modification)	

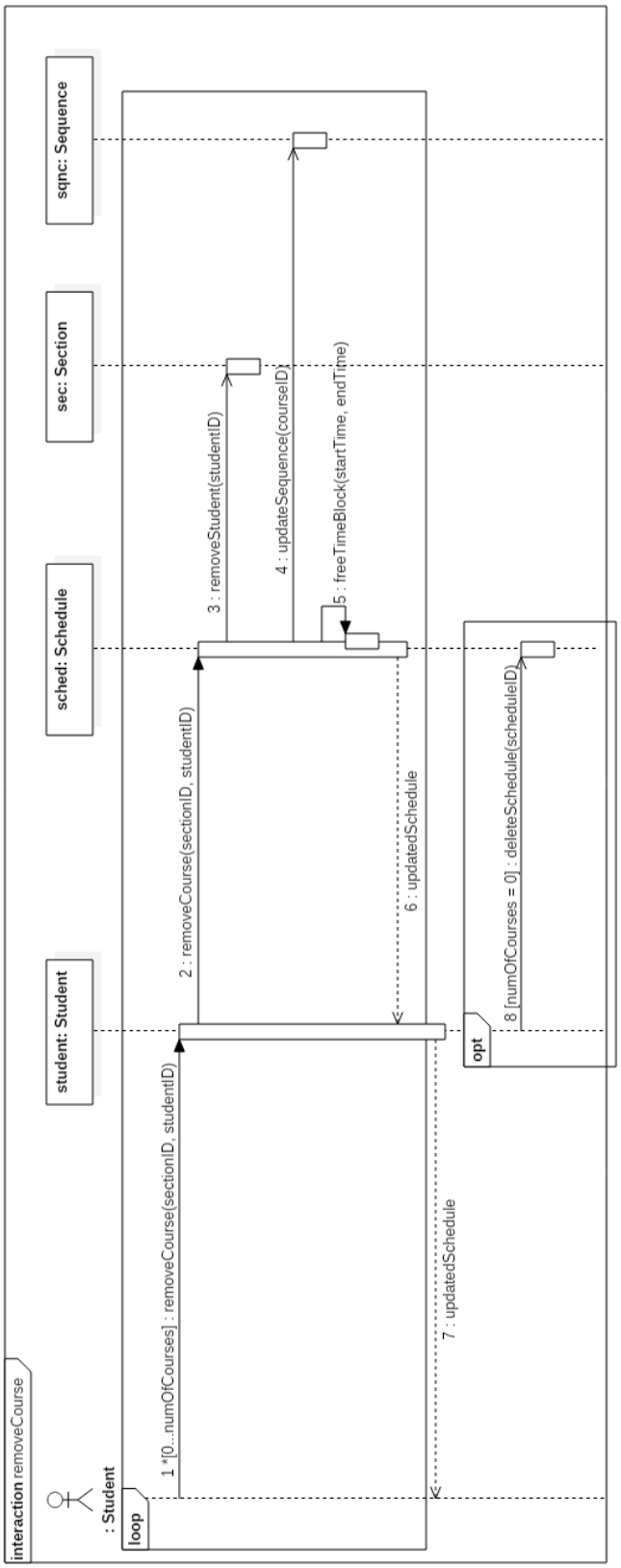


Figure 12.9.4 removeCourse contract diagram

12.8 Use Case 11 – Save Generated Schedule

12.8.1 Fully Dressed Use Case

Use Case ID:	UC11	
Use Case Name:	Save Generated Schedule	
Created By:	Lori Dalkin	Last Updated By: Claudia Della Serra
Date Created:	February 7, 2016	Last Revision Date: February 14, 2016
Actor(s):	Student	
Goal/Actor Goals:	The user wishes to save a schedule they have generated.	
Description/Summary:	A generated schedule can be saved in order to be accessed and viewed later on by the user. This schedule consists of courses the user is satisfied with and wishes to keep on their schedule.	
Preconditions:	❖ The user is logged on to their account ❖ A schedule has been generated containing at least one course.	
Post-conditions:	The system saves the generated schedule to the user's account.	
Minimum Guarantee:	The user's account remains unchanged.	
Basic Flow:	7. User indicated their wish to generate a schedule 8. The system prompts the user with a list of semesters for which they wish to generate a schedule 9. The user indicates which semester they wish to generate a schedule for 10. The system displays a generated schedule for the indicated semester 11. The user will indicates that they want to save the generated schedule. 12. The system will display a message telling the user that the schedule has been saved.	
Risk assessment:	Medium	
Importance:	4/5	

12.8.2 System Sequence Diagram

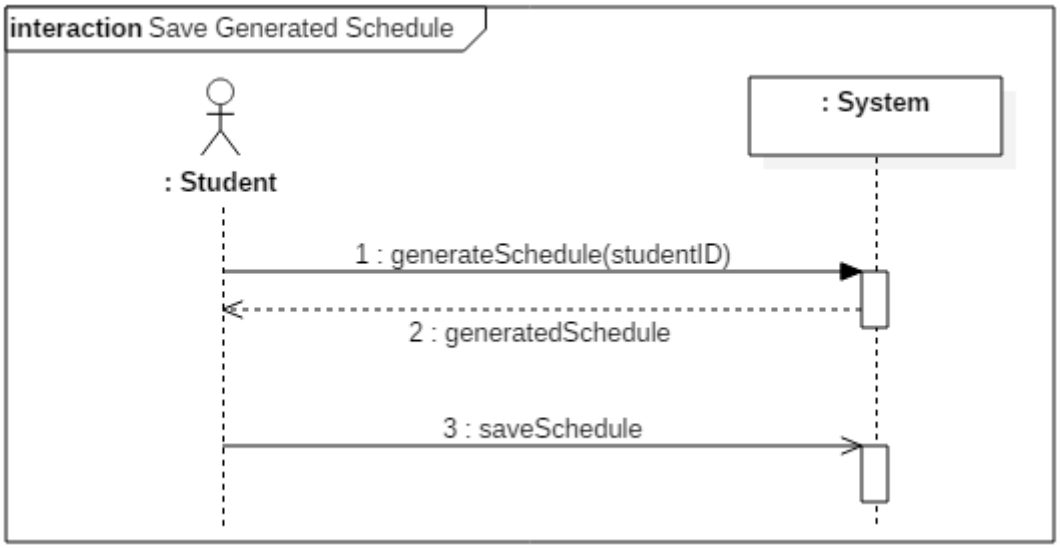


Figure 12.11.1 Save Generated Schedule system sequence diagram

12.8.3 Contract Diagrams

Contract 11.1	generateSchedule	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	generateSchedule(studentID)	
Cross-reference:	UC11, UC6, contract 6.2	
Preconditions:	❖ User is logged into the system. ❖ User is enrolled in a program	
Postconditions:	❖ A Schedule object is generated (Instance creation) ❖ The Schedule is associated to the student (association formed)	

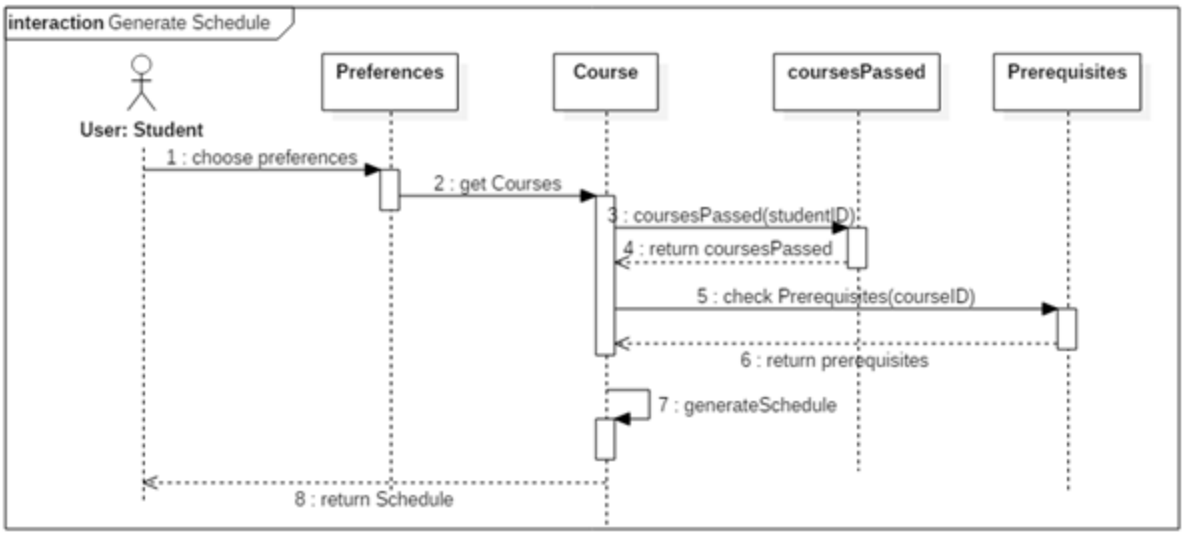


Figure 12.11.2 generateSchedule contract diagram

Contract 11.2	**saveGeneratedSchedule**	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	saveGeneratedSchedule(scheduleID): list<Schedule>	
Cross-reference:	UC11	
Preconditions:	❖ User is logged into the system. ❖ User is enrolled in a program ❖ User has generated a schedule	
Post-conditions:	❖ A schedule for that semester will be saved to the student's account (association formed) ❖ The schedule will be added to the list of saved schedules (attribute modification)	

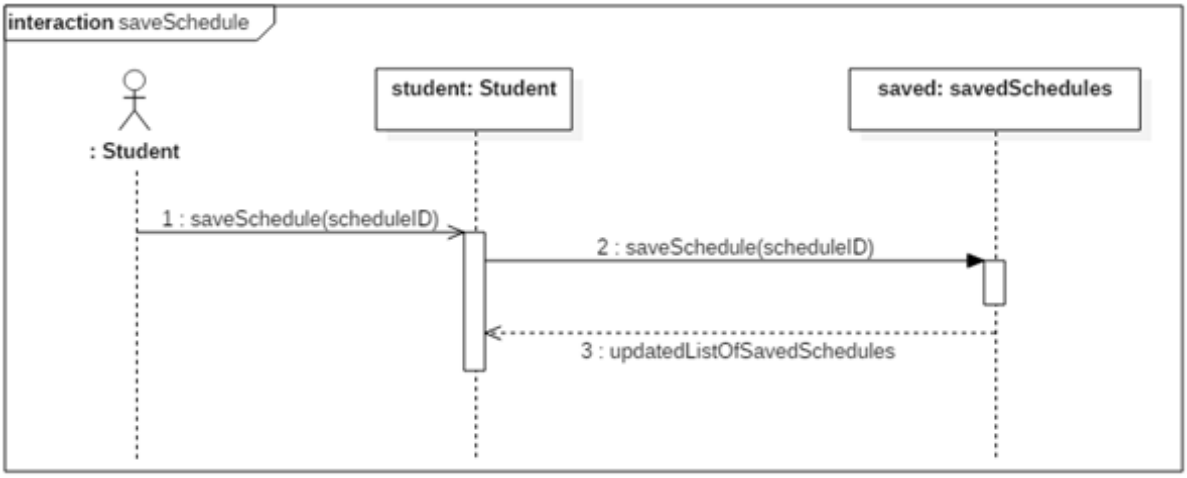


Figure 12.11.3 saveSchedule contract diagram

12.9 Use Case 14 - Create course

12.9.1 Fully Dressed Use Case

Use Case ID:	UC14	
Use Case Name:	Create Course	
Created By:	Claudia Della Serra	Last Updated By:
Date Created:	Marc 26th, 2016	Last Revision Date:
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	Administrator wishes to add a course to the database of courses	
Description/Summary:	The Administrator wishes to add a course to the system's database of courses by entering its course code, the course description, and the amount of credits awarded by the course. A course may be a section, a tutorial, or a lab.	
Preconditions:	❖ The administrator has logged in. ❖ The administrator has viewed the list of courses in the database.	
Post-conditions:	The course is added to the database of courses	
Minimum Guarantee:	The database of courses will remain unaltered.	
Basic Flow:	1. Administrator requests to create a course 2. The system presents a page asking for the course id, the course description, and the amount of credits awarded by the course. 3. The Administrator enters the course information and presses submit 4. The system presents the course information, with the options to manage the course or view all courses in the database	
Risk assessment:	3/5	
Importance assessment:	5/5	

12.9.2 System Sequence Diagram

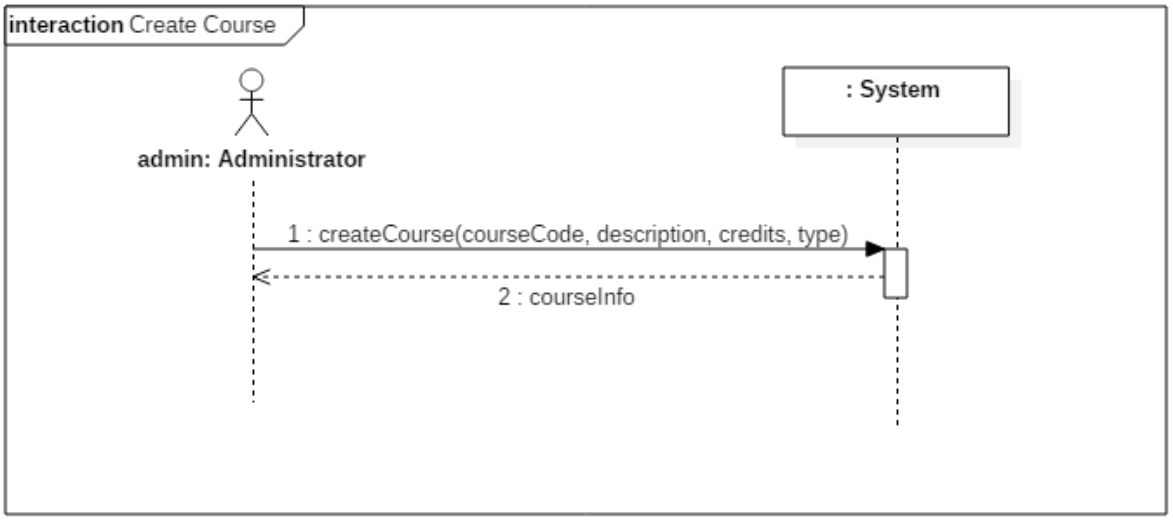


Figure 12.9.1 System Sequence Diagram for create course

12.9.3 Contract Diagrams

Contract 14.1	createCourse	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	createCourse(courseCode: String, description: String, credits: int, type: String) : Course	
Cross-reference:	UC14	
Preconditions:	❖ User is logged into the system as an Administrator. ❖ User has requested to create a course	
Post-conditions:	❖ A new Course is created (instance creation) ❖ The new Course is displayed to the Administrator (association formed)	

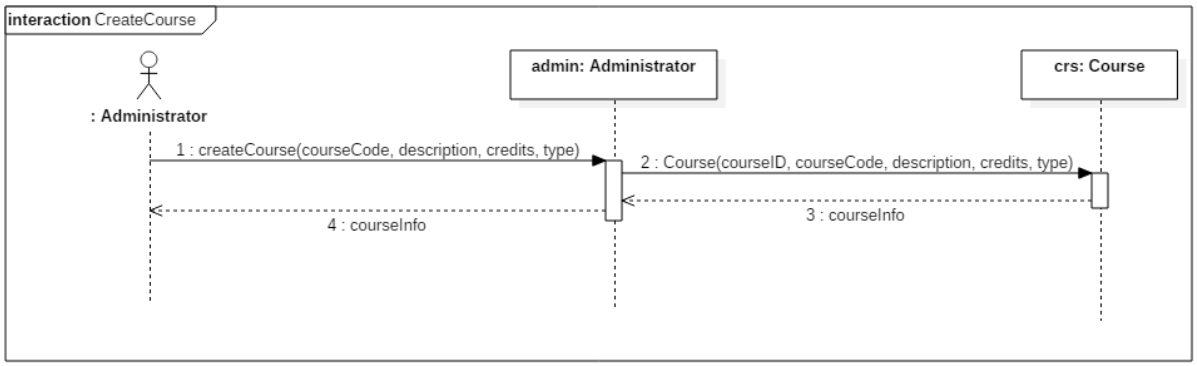


Figure 12.9.2 Sequence diagram for CreateCourse contract

12.10 Browse Courses (Administrator)

12.10.1 Fully Dressed Use Case

Use Case ID:	UC15	
Use Case Name:	Browse Courses	
Created By:	Claudia Della Serra	Last Updated By:
Date Created:	Marc 26th, 2016	Last Revision Date:
Actor(s):	Primary Actor(s): Administrator	
Goal/Actor Goals:	Administrator wishes to view the database of courses	
Description/Summary:	The Administrator wishes to add view the system's database of courses. From here, the administrator may choose to add a course or manage courses.	
Preconditions:	❖ The administrator has logged in.	
Post-conditions:	The course database will be presented	
Minimum Guarantee:	The administrator will remain on the home page.	
Basic Flow:	1. Administrator requests to view the course database 2. The system presents a page listing all the courses in the database	
Risk assessment:	2/5	
Importance assessment:	4/5	

12.10.2 System Sequence Diagram

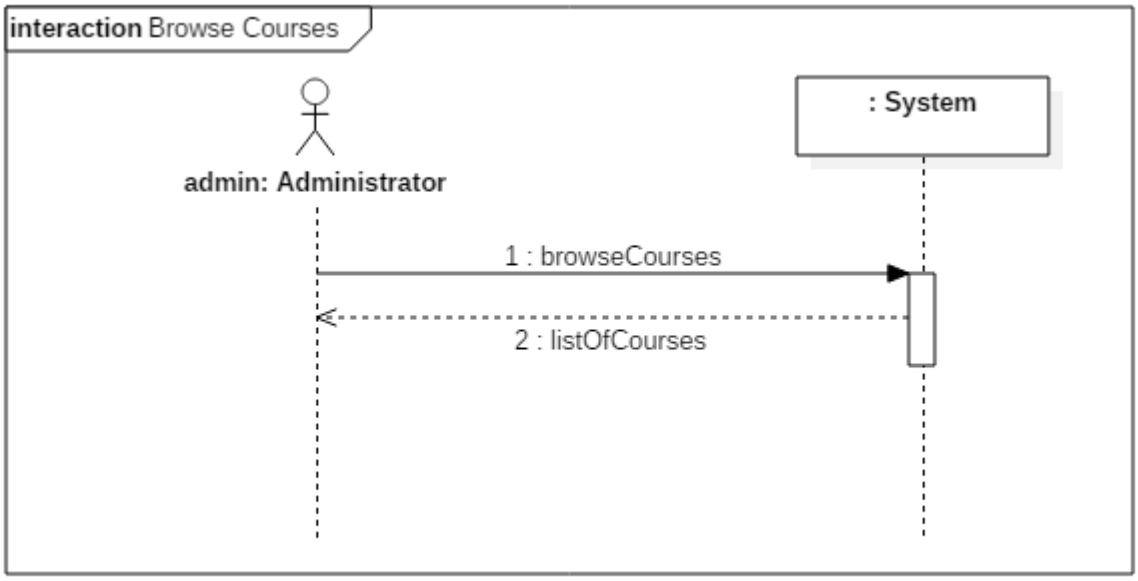


Figure 12.10.1 System Sequence Diagram for Browse Courses

12.10.3 Contract Diagrams

Contract 15.1	browseCourses	
Created By:	Claudia Della Serra	Last Updated By: Claudia Della Serra
Date Created:	March 2, 2016	Last Revision Date: March 2, 2016
Operation:	browseCourses() : list<Course>	
Cross-reference:	UC14	
Preconditions:	❖ User is logged into the system as an Administrator.	
Post-conditions:	❖ A new list of Courses is created (instance creation) ❖ The new list of Courses is displayed to the Administrator (association formed)	

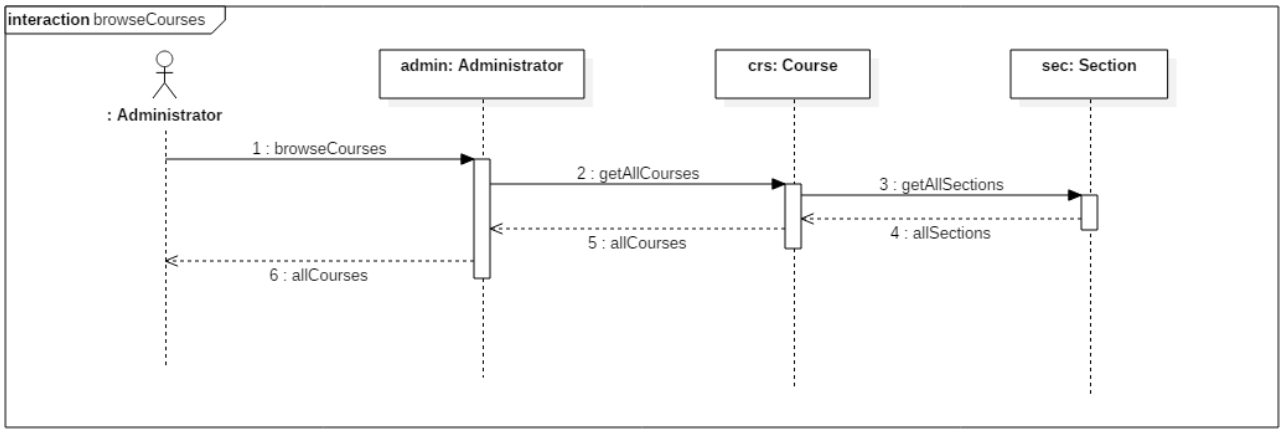


Figure 12.10.2 Sequence Diagram for browseCourses Contract

12.11 Use Case 17 – Search for Users

12.11.1 Fully Dressed Use Case

Use Case ID:	UC17	
Use Case Name:	Search for Users	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	Obtain access to data related to a user through search criteria.	
Description/Summary:	An administrator wishes to gain access to information related to a user through search criteria such as the user's ID, username, first name, last name and net name.	
Preconditions:	❖ User must be logged in as an Administrator.	
Post-conditions:	❖ A list of student(s) that fits the search criteria is created. ❖ The list is displayed as being editable to the administrator.	
Minimum Guarantee:	The system does not find any users matching the specified criteria and indicates that no results have been found.	
Basic Flow:	1. Administrator specifies the search criteria in available search fields. 2. System displays a list of editable users matching the search criteria indicated by the administrator.	
Risk assessment:	Medium	
Importance assessment:	2/5	

12.11.2 System Sequence Diagram

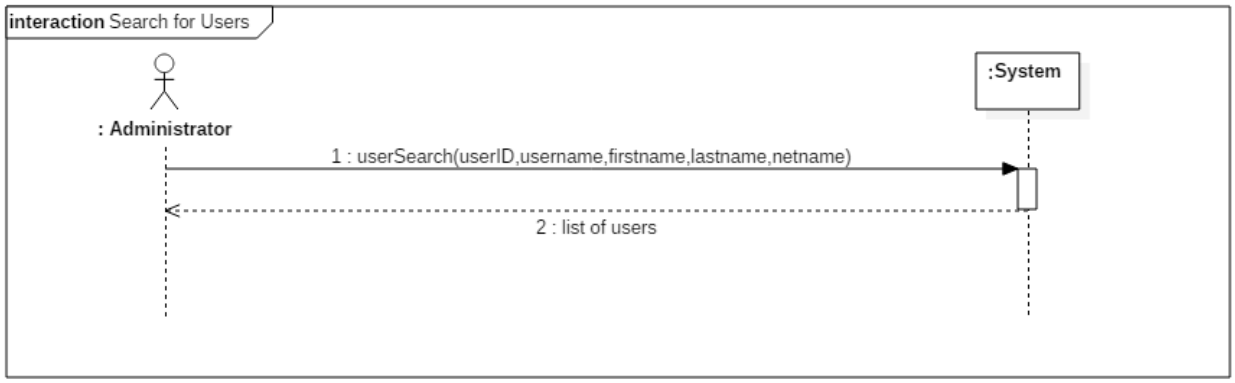


Figure 12.11.1 Search for Users System Sequence Diagram

12.11.3 Contract Diagram

Contract 17.1	userSearch	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Operation:	userSearch(userID, username, firstname, lastname, netname)	
Cross-reference:	UC17	
Preconditions:	❖ User is logged into the system as an Administrator.	
Post-conditions:	❖ A list of student(s) that fits the search criteria is created. (instance Creation) ❖ The administrator can access the list. (association formed)	

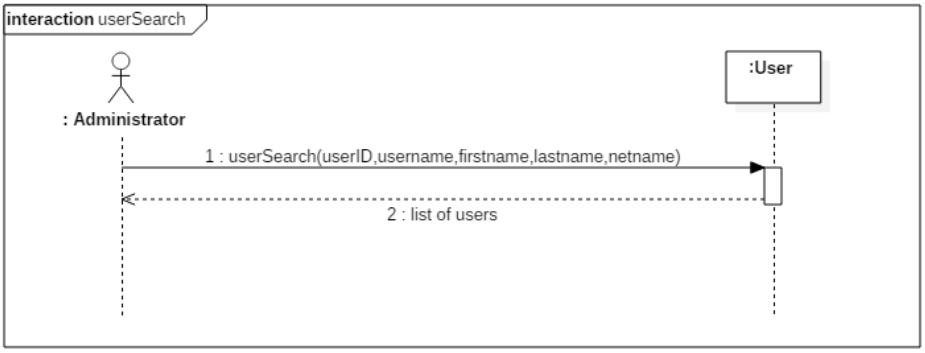


Figure 12.11.2 userSearch contract diagram

12.12 Use Case 18 – View User

12.12.1 Fully Dressed Use Case

Use Case ID:	UC18	
Use Case Name:	View User	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	View information stored in the system about a user.	
Description/Summary:	An administrator wishes to view the profile summary that contains information about a user such as its ID, username, first name, last name, and net name.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ A user search result is displayed. ❖ The user the administrator is looking for is listed in the search result.	
Post-conditions:	❖ The administrator has access to consult information about the user	
Minimum Guarantee:	The set of information field is displayed but may contain empty fields.	
Basic Flow:	1. Administrator indicates that the information about a particular user displayed on a search result is to be consulted. 2. System retrieves information about the user and displays it in a table.	
Risk assessment:	3/5	
Importance assessment:	3/5	

12.12.2 System Sequence Diagram

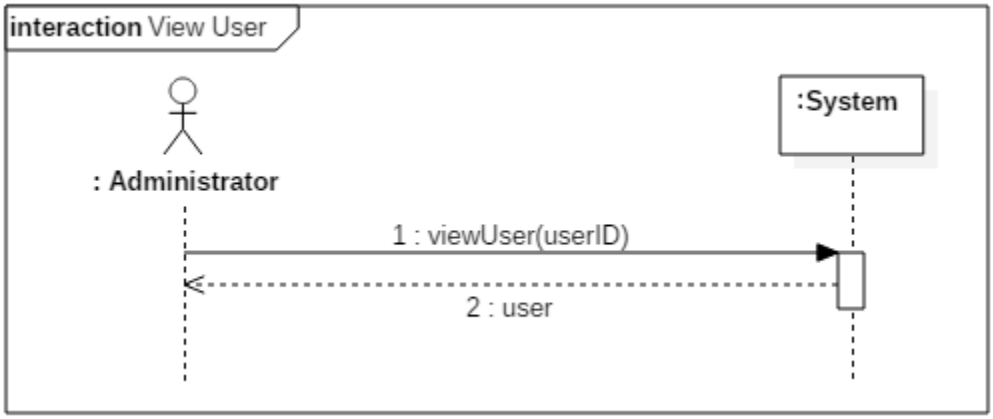


Figure 12.12.1 View User System Sequence Diagram

12.12.3 Contract Diagram

Contract 18.1	viewUser	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Operation:	viewUser(userID)	
Cross-reference:	UC18	
Preconditions:	❖ User is logged into the system as an Administrator. ❖ The ID of the user passed is a valid and existing user ID.	
Post-conditions:	❖ User corresponding to the passed user ID is created (instance creation) ❖ The User is made available for the administrator to consult. (association formed)	

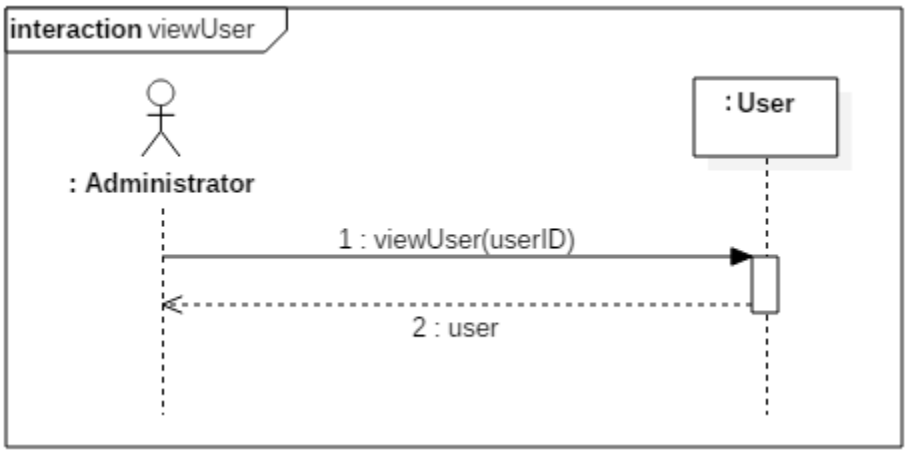


Figure 12.12.2 viewUser contract diagram

12.13 Use Case 19 – Update User

12.13.1 Fully Dressed Use Case

Use Case ID:	UC19	
Use Case Name:	Update User	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Actor(s):	Administrator	
Goal/Actor Goals:	Update information related to a user.	
Description/Summary:	An administrator wishes to update information saved about a user. Those include the username, first name, last name, net name, password and privilege.	
Preconditions:	❖ User must be logged in as an Administrator. ❖ A user search result is displayed.	
Post-conditions:	❖ Information about the user is updated. ❖ Updated information is displayed to the administrator.	
Minimum Guarantee:	The user's profile fails to be updated and an error is displayed.	
Basic Flow:	1. Administrator indicates that the information about a specific user must be updated. 2. System displays the update fields filled with the current information about the selected user. 3. Administrator changes and saves the new information about the user. 4. System displays the updated profile of the user.	
Risk assessment:	3/5	
Importance assessment:	3/5	

12.13.2 System Sequence Diagram

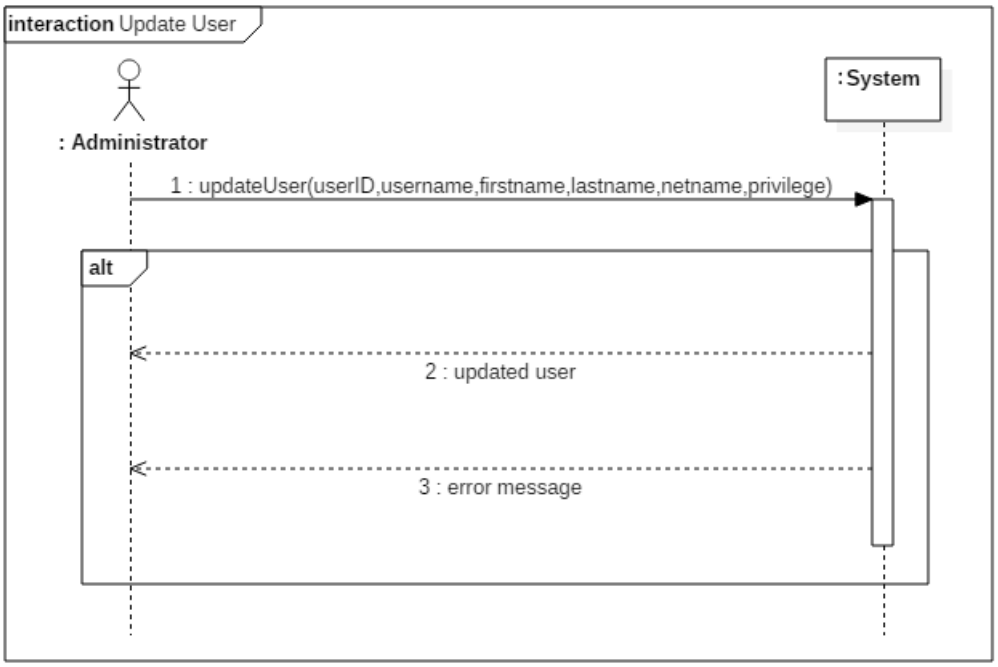


Figure 12.13.1 Update User System Sequence Diagram

12.13.3 Contract Diagram

Contract 19.1	updateUser	
Created By:	Philip Lim	Last Updated By: Philip Lim
Date Created:	April 2, 2016	Last Revision Date: April 4, 2016
Operation:	updateUser(userID,username,firstname,lastname,netname,privilege)	
Cross-reference:	UC19	
Preconditions:	❖ User is logged into the system as an Administrator. ❖ A search result is displayed ❖ The user that corresponds to the userID is displayed on the search result	
Post-conditions:	❖ The user's information is updated with the new one passed as parameters. (attribute Modification) ❖ Administrator has access to consult the updated user (association formed)	

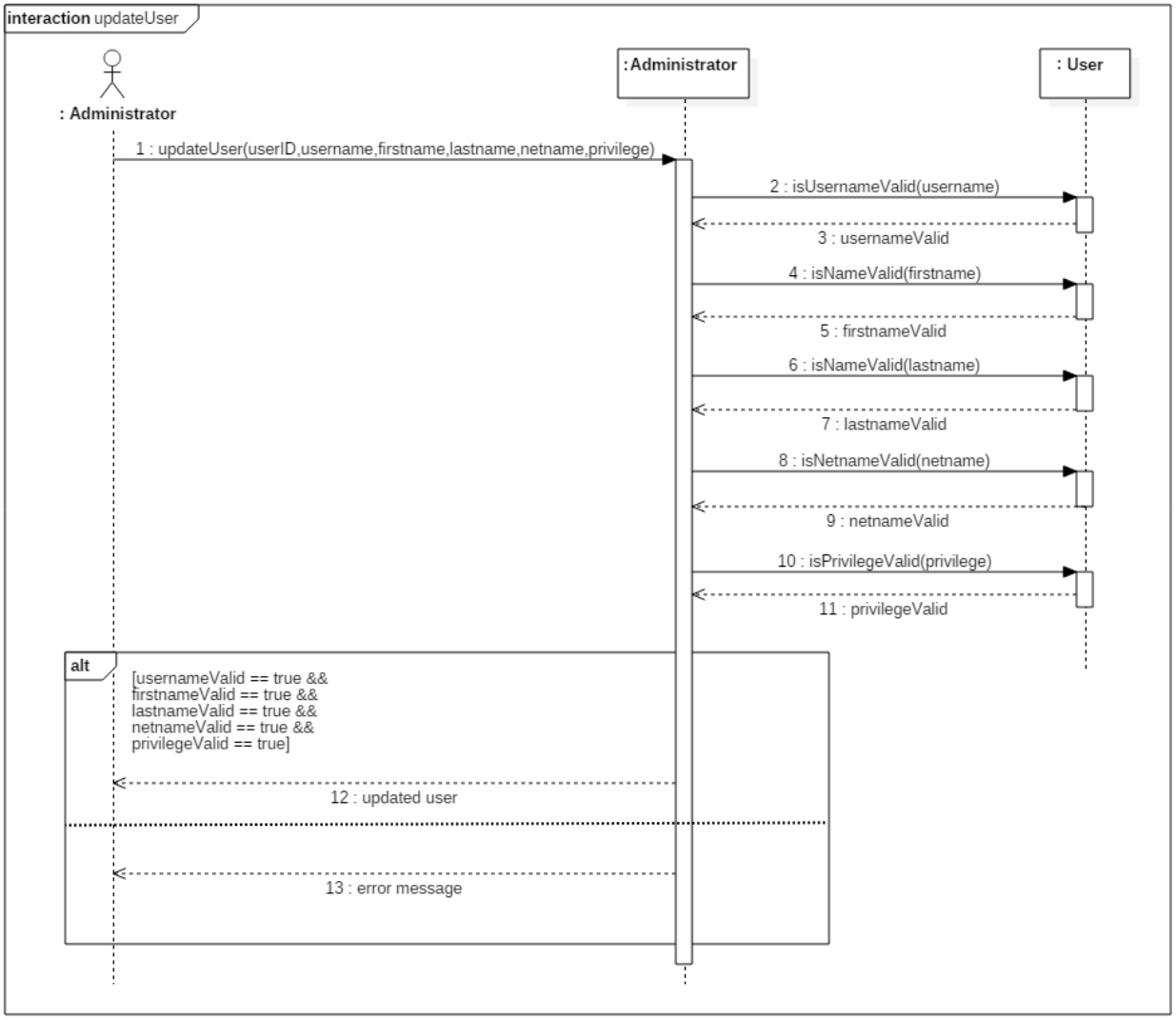


Figure 12.13.2 updateUser contract diagram

13. Estimation

The current cost and time estimation has been updated from previous estimations made in deliverable 1 by considering the estimated lines of code that will be produced in the implementation of the TimeTurner system. An estimated person-months amount was calculated, and then this amount applied to previous estimations made in conjunction with estimations based on time-logs taken from activities performed in the production of the Deliverable 2 document.

13.1 Function Point Estimation

13.1.1 Unadjusted Function Points

Function Type	Functional Complexity		Complexity Totals	Function Type Totals
ILF	0 Low	X 7	0	
	10 Average	X10	100	
	0 High	X15	0	100
EIF	0 Low	X 5	0	
	0 Average	X7	0	
	0 High	X10	0	0
EI	0 Low	X3	0	
	4 Average	X4	16	
	0 High	X6	0	16
EQ	0 Low	X 3	0	
	0 Average	X4	0	
	6 High	X6	36	36
EO	0 Low	X 4	0	
	5 Average	X5	25	
	0 High	X7	0	25
			Unadjusted Function Point Count	177
			SLOC/FP for PHP [1]	67
			Total SLOC	11859
			COCOMO II estimated person-months effort	43.8

[1] taken from https://www.cs.helsinki.fi/u/taina/ohtu/fp.html

Internal function points were calculated based on the database tables given for Users, User Schedules, Completed Courses, Courses, Sections, Subsections, and Prerequisites. External Outputs, External Queries

and External Inputs were calculated based on methods found in the above sections (see Section 3.2 Subsystem Interface Specification). Given the above estimated lines of code, this value was put into the COCOMO II engine, with the Software Scale Drivers modified as follows:

- Pecedentedness: Low
  - Due to the fact that not very much experience in programming a large system such as this one has been previously incurred by many of the team members.
- Architecture/Risk Resolution: high
  - Due to the fact that much planning in architecture and design has been undergone, and that risk assessment and analysis has been performed since the start of the project.
- Process Maturity: low
  - Due to the fact that the team has only begun using software process management, and does not have much experience in handling projects in this manner
- Development flexibility: high
  - Due to the fact that very vague requirements were given by instructors, thus the flexibility to implement these requirements, and create our own, was very high.
- Team cohesion: high
  - The team has had much experience working together on previous assignments, labs, etc. thus team cohesion is high and group work and communication are excellent.

13.2 Module Estimations

Given the estimated time of 43.8 person-months, and previous time estimates that were performed in the first version of the project, new time estimates have been made. Given that all team members are currently full-time students and working on other projects throughout the course of this project development, our person-months are based on a rough estimate of 4.85 hours per person per task; such an estimate is taken from the time logs of one team member working on the Deliverable 2 task of creating system sequence and contract diagrams (times were divided by 2 due to the fact that all system sequence diagrams were implemented rather than just 2 major ones). This results in a 424.86 person-hours estimate for the entire project. Thus, previous estimates are updated as follows:

13.2.1 6.3.4 Total Deliverable Estimates

Deliverable Names	Cost (Hours)	Revised Cost (Hours)
Deliverable 0	29.5	29.5
Deliverable 1	35	35
Deliverable 2	190	194
Deliverable 3	75	97
Deliverable 4	60	48.5
Total	**389.5**	**404**

13.3 Updated Gantt Chart

The following updated version of the initial Gantt chart shows the progress of the project and its current state of development.

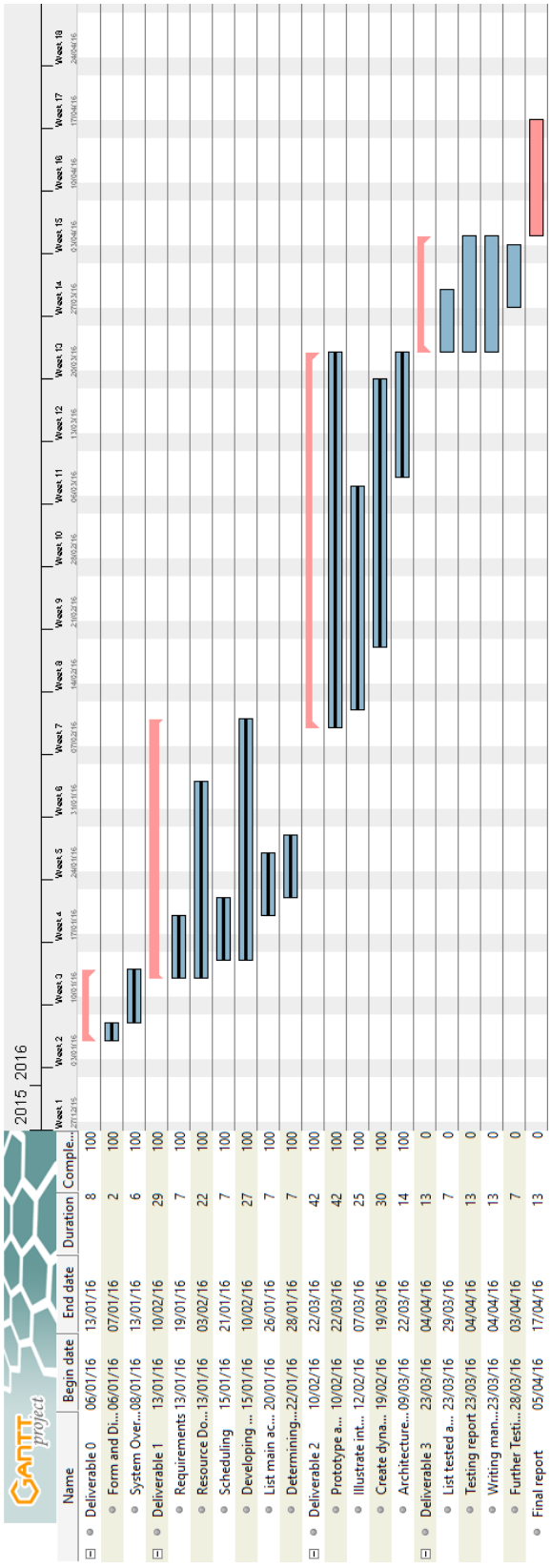


Figure 6.3.1 Updated Gantt Chart

14. Rapid Prototyping and Risk

In order to minimize risk, a quick prototype of the system is made. A quick mockup of the system, shown below, is produced to ensure that the project is headed in the right direction and reduce overall risk in the early development phase of the project.

The prototype has also helped us view how the final project will look like and has inspired us to come up with ideas on what a user should expect from a schedule planner in terms of features, layout and ease-of-use.

14.1 User Interface Mockup

14.1.1 Main Layout

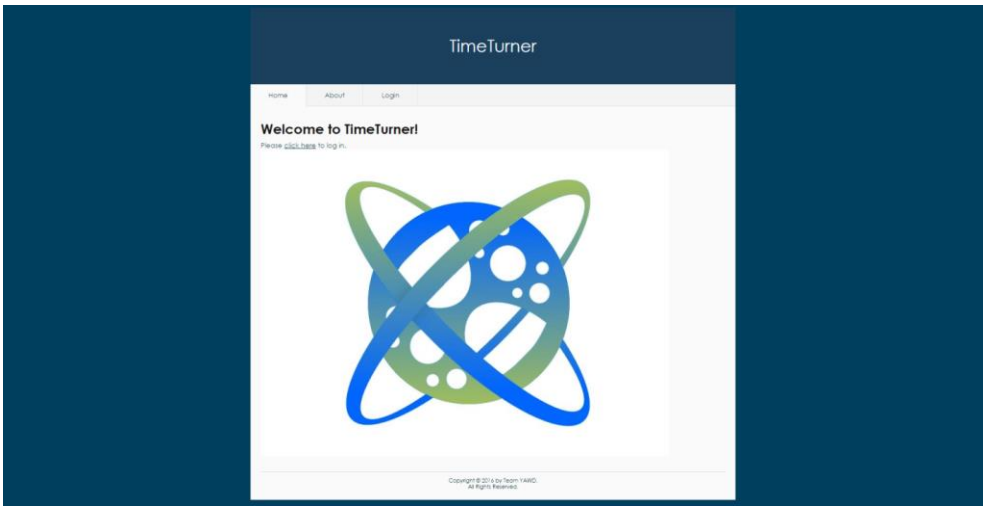


Figure 14.1.1. Main layout of the system

Upon accessing the TimeTurner website, the user is welcomed with a message and has the choice to login via the tab or by clicking the “click here” link. These will redirect the user to the Login page. The Yii framework was used to generate this layout, as it already contains a few themes.

14.1.2 Login Interface Mockup

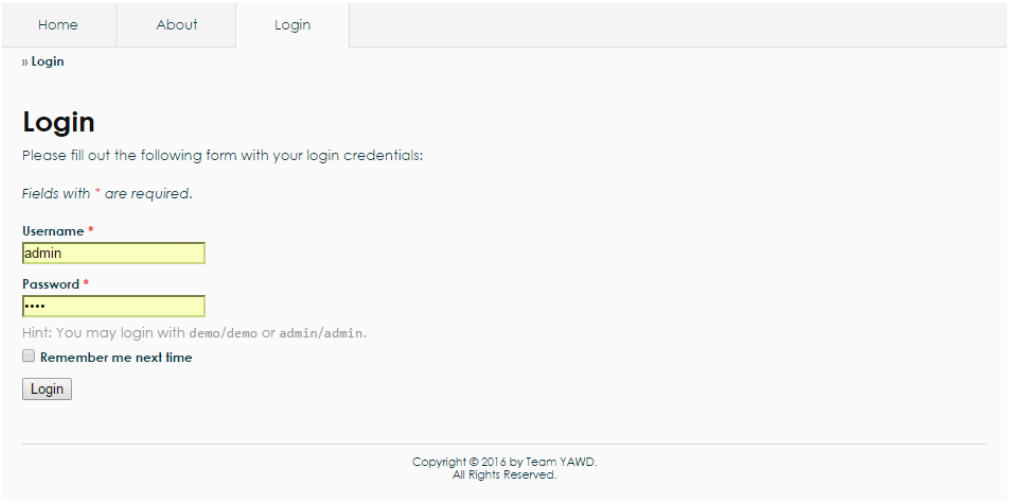


Figure 14.1.2. Login interface mockup

For the Login interface, there is an authentication form which has been generated by the Yii framework. The user may enter as an admin or as a student. A username and a password are required. Upon clicking the "Login" button, the system will query the database to check if the username and password are valid. The system will redirect to another page if the login is successful, else it will display an error message.

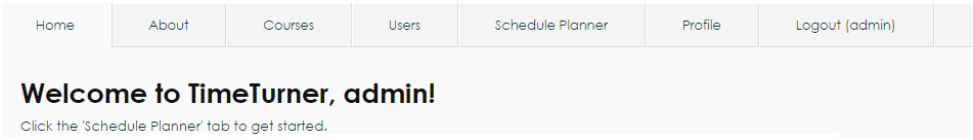


Figure 14.1.3. User menu for an administrator

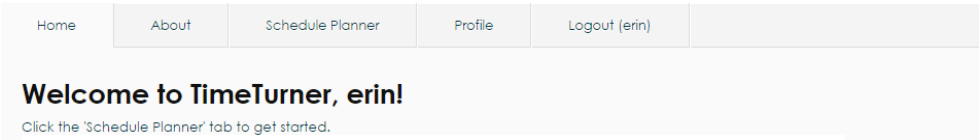


Figure 14.1.4. User menu for a student

If the administrator logs in, additional tabs appear on top. These are "Courses", "Users", "Schedule Planner", "Profile" as well as a "Logout" tab, as shown in Figure 3. If logged in as a student, the user cannot see the "Courses" and "Users" tabs, as shown in Figure 4, as these can only be modified by the administrator.

14.1.3 Generate Schedule Interface Mockup

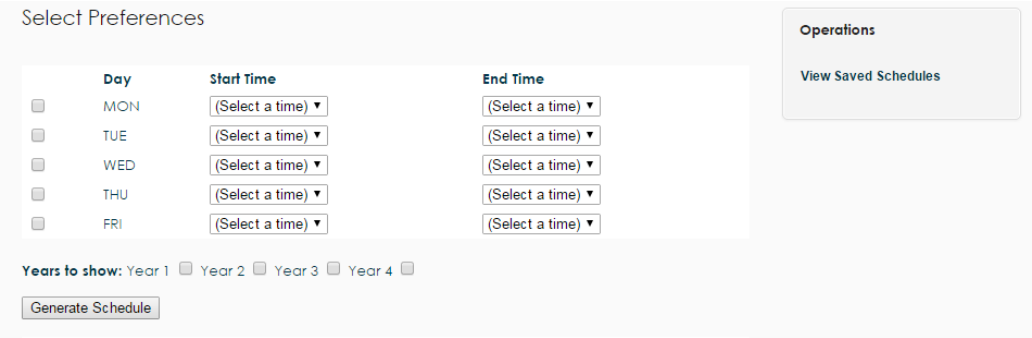


Figure 14.1.5. Generate schedule interface mockup

All users have a "Schedule Planner" tab, which upon clicking, redirects the user to the page as shown in Figure 5. The user can select their preferences by checking one or more days of the week and select a starting and ending time. The user has the option of choosing which years can be displayed. Additionally, the "Viewed Saved Schedules" operation on the right navigation bar can be selected to view schedules that were previously generated and saved.

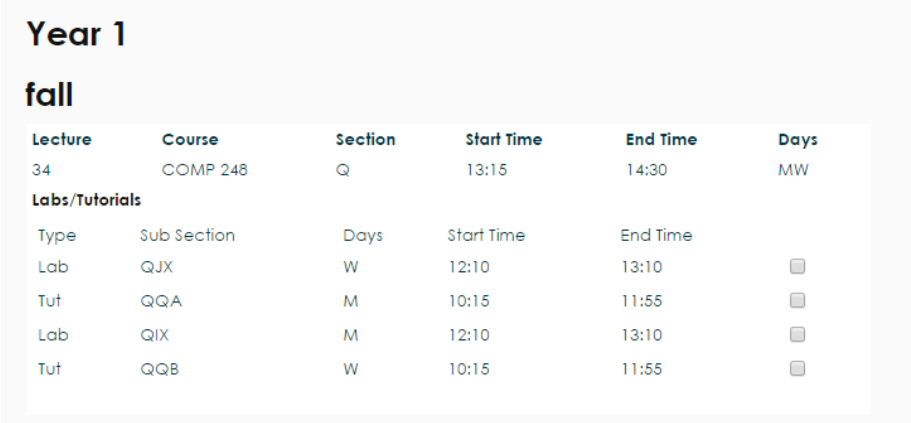


Figure 14.1.6. Classes sections view

Upon clicking the "Generate Schedule" button, classes are displayed along with their sections, as shown in Figure 6. The user may check the boxes next to the preferred sections.

14.2 Risk

14.2.1 Framework

One of the main risks involved in this project is the use of an unfamiliar framework. The current framework in use is Yii, which is based on MVC architecture in PHP. Due to the fact that most of the development team has never worked with a large framework, the learning curve tends to be very high. As such, possible delays might arise during development due to unforeseen problems that might come up while working with Yii. Though one of the major advantages of using this framework, is that it is loaded with features that make development much quicker in the long run; entire modules can be created on the fly with

little effort. Depending on the learning curve progress, Yii can either create a healthy environment for development or hurt production..

14.2.2 Time Constraint

As deadline approaches, the realization of some features of software will have to be scoped out; only certain core components will be placed in order to meet a good portion of the major requirements of the scheduler.

14.2.3 Control Version System

Since the control version system in use is GitHub, the development team opted to use PhpStorm 10, an IDE provided by JetBrains, since it is integrated with Git. This adds up to the learning curve, since the team members have to learn an unfamiliar program, as well as how it works with Git. Using a framework requires working with many files, thus the project needs to be carefully managed. At times, PhpStorm 10 would not synchronize properly with the project's repository on GitHub, which adds up to the time required to spend on learning and working on other components of the project. Therefore, learning how to properly use Git with PhpStorm can arise in possible delays during development due to team members' inexperience with them.

14.2.4 Server Uptime

During team meetings, the production environment occurs at school. Often times, the network does not work properly as many students and employees try to access Internet at the same time. This would cause a decrease in productive time. Team members also have their own production environment at home. However, if a power outage occurs, the server computer may crash; hindering deployment and causing further delays.

Project Testing and Delivery Document

15. Introduction

The following document covers the testing involved in the system. A full breakdown of each test case is provided, accompanied by the results. This is useful information in the installation procedure of this software for any users, which is shown alongside the user manual of this section. The types of testing includes unit testing, requirement testing and stress testing. The final topic included is the final cost estimation of the entire project.

16. Testing Report

16.1 Test Coverage

While it is ideal to test every single item and every possible requirement of a system, this is often impossible due to time and budget constraints. In this section, we describe which items of the system were tested, why it was important to test them and which tools were used for testing. We also describe which items were not tested, yet would have been tested given more time and/or money.

16.1.1 Tested Items

16.1.1.1 Unit Testing:

Unit testing focuses on testing small units of code at a time. It breaks the system down into its individual classes or methods and tests them each separately. Unit testing is important for making sure each piece of an application works as expected, so that these pieces can then be combined with other parts of the application in order to ultimately have a full working system.

Here, Unit testing was done using PHPUnit 5.3. We performed unit tests on two of the model classes in our system, described below:

Tested items:

- **LoginForm:** it was important to test the LoginForm class because this class is needed for users to access the system. Without access to the system, nothing else can be done. To test this unit, we created a PHPUnit test class called LoginFormTest, which consists of methods to test each function of the LoginForm class, as well as a setUp() method that creates a testable LoginForm object and provides sample input (username and password) for the form.

- **UserSchedule:** the UserSchedule model class was chosen for unit testing since it accesses the database and is critical for generating a schedule. To test this unit, we again made a PHPUnit test class, called UserScheduleTest with methods to test each function. The setUp() method creates a testable UserSchedule object and opens a connection to the database, and the tearDown() method closes the database connection after the test is run.

16.1.1.2 Requirements Testing:

Requirements testing breaks a system down based on its requirements, and tests each one of them to ensure they function as expected. It is important for checking which requirements are met and which are not met. For each different requirement, several test cases are performed with different possible scenarios or inputs. Here, each test case includes a test ID, description (describing the input), expected output, actual output and result. If the actual output matches the expected output, the result is a *pass*. If not, the result is a *fail*. For any failed tests, we can return to the source code used to implement this requirement and identify the bugs that caused it to fail. We tested the following requirements:

- **UC1:** Login
- **UC2:** Logout
- **UC4:** Browse Course List
- **UC6:** Generate Schedule
- **UC7:** View Saved Schedules
- **UC8:** View Academic Record
- **UC9:** Drop Course
- **UC11:** Save Generated Schedule
- **UC13:** Manage Courses
- **UC14:** Create Course
- **UC15:** View Courses
- **UC16:** Add User
- **UC17:** Search Users
- **UC18:** View User
- **UC19:** Update User
- **UC20:** Delete User
- **UC21:** Browse Users

16.1.1.3 Stress Testing:

Stress testing is used to test an application's performance under extreme conditions of usage, such as a high volume of users or a large data load. It is done in order to determine the system's robustness and reliability, and helps to identify issues that may only arise under stressful conditions.

There are a number of tools available for stress testing a web application that simulate extreme conditions. Here, we used Apache JMeter, a Java application that works by creating multiple threads that simultaneously access a given web server via HTTP requests. We tested stress conditions for both the login and schedule generation functions of the system.

16.1.1.4 Security Testing:

Security testing is crucial for any software system to ensure that the system can protect itself against malicious attacks. We used two tools to test security. The first was SQL Inject Me, a Firefox plugin that sends SQL injection strings through the forms of a webpage and checks for any database errors that may occur as a result. The second was Nikto, a web server scanner that uses a Perl script to perform multiple tests against a given server, including checking for dangerous files and checking for outdated versions.

16.1.2 Untested Items of Interest

16.1.2.1 Untested Units

For the unit tests, we were only able to test 2 of the 11 models in our system. This was due to tight time constraints, as it took a while to set up PHPUnit and write the test code. We chose to test the LoginForm and UserSchedule models because these classes are very important to the system's functionality; every user needs the LoginForm to access the system, and UserSchedule is needed to generate a schedule. With more time, we would have ideally performed unit tests on the other 9 models in our system as well as the 5 controller classes, listed below:

- Model classes:
  - CompletedCourses
  - ContactForm
  - Course
  - PreferenceForm
  - Prerequisite
  - Section
  - Subsection
  - User
  - UserSchedules
- Controller classes:
  - CourseController
  - ProfileController
  - SchedulerController
  - SiteController
  - UserController

16.1.2.2 Untested Requirements

For the requirements tests, we tested all requirements except the ones that were scoped out. With more time, we would have liked to implement these requirements and perform tests on them. The following requirements were not tested:

- **UC3:** Create Course Sequence
- **UC5:** View Course Sequence
- **UC10:** Add Course
- **UC12:** View Weekly Schedule

16.2 Test Cases

16.2.1 Unit Testing

16.2.1.1 Code for UserScheduleTest:

```
<?php
require_once('../../../protected/models/UserSchedule.php');
$yiit='../../../framework/yiit.php';
require_once($yiit);

class UserScheduleTest extends CDbTestCase {
    protected $fixture;
    protected $db;

    protected function setUp() {
        $this->db = new CDbConnection('mysql:host=conuscheduler.ddns.net;dbname=soen341','soen341','soen341');
        $this->db->active=true;
        CActiveRecord::$db=$this->db;
        $this->fixture = new UserSchedule;
    }

    protected function tearDown() {
        if($this->db)
            $this->db->active=false;
    }

    public function testTableName() {
        $expected = 'user_schedule';
        $results = $this->fixture->tableName();
        $this->assertEquals($expected,$results);
    }

    public function testRules() {
        $expected = array(
            array('scheduleID, courseID, sectionID, subsectionID, year', 'required'),
            array('scheduleID, courseID, sectionID, subsectionID, year', 'numerical', 'integerOnly'=>true),
            array('ID, scheduleID, courseID, sectionID, subsectionID, year', 'safe', 'on'=>'search'),
        );
        $results = $this->fixture->rules();
        $this->assertEquals($expected,$results);
    }

    public function testRelations() {

        $expected = array(
            'scheduleID' => array(UserSchedule::BELONGS_TO, 'UserSchedules', 'ID')
        );
        $results = $this->fixture->relations();
        $this->assertEquals($expected,$results);
    }

    public function testAttributeLabels() {
        $expected = array(
            'ID' => 'ID',
            'scheduleID' => 'Schedule',
```


```
            'courseID' => 'Course',
            'sectionID' => 'Section',
            'subsectionID' => 'Subsection',
            'year' => 'Year',
        );
        $results = $this->fixture->attributeLabels();
        $this->assertEquals($expected, $results);
    }

    public function testSearch() {
        $stub = $this->getMockBuilder('CDbCriteria')
            ->disableOriginalConstructor()
            ->getMock();
        $stub->expects($this->any())
            ->method('compare')
            ->will($this->returnValue('true'));
    }

    public function testModel() {
        $stub = $this->getMockBuilder('UserSchedule')
                ->disableOriginalConstructor()
                ->getMock();
        $stub->expects($this->any())
            ->method('model')
            ->will($this->returnValue($this->fixture));
        $this->assertNotNull($stub);
    }
}
```

Results:

UserScheduleTest passed 6/6 tests, shown in the figure below:

```
C:\xampp\php\php.exe C:\xampp\php\phpunit.phar --no-configuration UserScheduleTest C:\xampp\htdocs\ConUScheduler\protected\tests\unit\UserScheduleTest.php --teamcity
Testing started at 4:25 PM ...
PHPUnit 5.3.0 by Sebastian Bergmann and contributors.


Time: 1.19 seconds, Memory: 9.50Mb

OK (6 tests, 5 assertions)

Process finished with exit code 0
```


16.2.1.2 Code for LoginForm test:

```php
<?php

require_once('../../../protected/models/LoginForm.php');
require_once('../../../protected/components/UserIdentity.php');
$yiit='../../../framework/yiit.php';
require_once($yiit);

class LoginFormTest extends CTestCase
{
    protected $fixture;
    protected function setUp() {
        $this->fixture = new LoginForm;
        $this->fixture->username = "erin";
        $this->fixture->password = "soen341";
        $this->fixture->rememberMe = "0";
    }

    protected function tearDown() {
    }

    public function testRules() {
        $stub = array(array('username, password', 'required'),
            array('rememberMe', 'boolean'),
            array('password', 'authenticate'));
        $results = $this->fixture->rules();
        $this->assertEquals($stub,$results);
    }

    public function testAttributeLabels() {
        $stub = array('rememberMe'=>'Remember me next time',);
        $results = $this->fixture->attributeLabels();

        $this->assertEquals($stub, $results);
    }

    public function testAuthenticate() {
        $stub = $this->getMockBuilder('UserIdentity')
            ->disableOriginalConstructor()
            ->getMock();
        $stub->expects($this->any())
            ->method('authenticate')
            ->will($this->returnValue('true'));
    }

    public function testLogin_2() {
        $stub = $this->getMockBuilder('LoginForm')
            ->disableOriginalConstructor()
            ->getMock();
        $stub->expects($this->any())
            ->method('login')
            ->will($this->returnValue('true'));
        $this->assertEquals('true', $stub->login());
    }
}
```


Results:
LoginFormTest passed 4/4 tests, shown in the figure below:

```
C:\xampp\php\php.exe C:\xampp\php\phpunit.phar --no-configuration LoginFormTest C:\xampp\htdocs\ConUScheduler\protected\tests\unit\LoginFormTest.php --teamcity
Testing started at 5:24 PM ...
PHPUnit 5.3.0 by Sebastian Bergmann and contributors.


Time: 232 ms, Memory: 8.25Mb

OK (4 tests, 3 assertions)

Process finished with exit code 0
```

16.2.2 Requirements Testing

16.2.2.1 Login

UC1	**Login (Administrator)**			
Test ID	Description	Expected Output	Actual Output	Result
1.1	Username: Admin Password: 1234	Home Page	Home Page	**Pass**
1.2	Username: Admine Password: 1234	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**
1.3	Username: Admin Password: 12345	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**
1.4	Username: Admine Password: 12345	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**

UC1	**Login (Student)**			
Test ID	Description	Expected Output	Actual Output	Result
1.5	Username: Erin Password: soen341	Home Page	Home Page	**Pass**
1.6	Username: Erine Password: soen341	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**
1.7	Username: Erin Password: soen341	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**
1.8	Username: Erine Password: soen341	Incorrect Username or Password Error	Incorrect Username or Password Error	**Pass**

16.2.2.2 Logout

UC2	**Logout (Administrator)**			
Test ID	Description	Expected Output	Actual Output	Result
2.1	User clicks logout tab	Home Page	Home Page	**Pass**

UC2	**Logout (Administrator)**			
Test ID	Description	Expected Output	Actual Output	Result
2.2	User clicks logout tab	Home Page	Home Page	**Pass**

16.2.2.3 Create Course Sequence

~~**UC3**~~	~~**Create Course Sequence**~~			
~~Test ID~~	~~Description~~	~~Expected Output~~	~~Actual Output~~	~~Result~~

16.2.2.4 Browse Course List

UC4	**Browse Course List**				
Test ID	Description	Input	Expected Output	Actual Output	Result
4.1	Student requests to view course list	User clicks on "Course Browser" tab	List of all courses in the system is displayed	List of all courses in the system is displayed	**Pass**
4.2	Student clicks on a prerequisite course	User clicks on "COMP 248" link	Course information for that course is displayed	Course information for that course is displayed	**Pass**

16.2.2.5 View Course Sequence

~~**UC5**~~		~~**View Course Sequence**~~			
~~Test ID~~	~~Description~~	~~Input~~	~~Expected Output~~	~~Actual Output~~	~~Result~~

16.2.2.6 Generate Schedule

UC6	**Generate Schedule**				
Test ID	Description	Input	Expected Output	Actual Output	Result
6.1	Student leaves the preferences blank and generates the schedule for a given year.	[blank preferences] Year 1	All courses for each semester in each academic year selected is displayed.	All courses for each semester in each academic year selected is displayed.	**Pass**
6.2	Student selects a time preference without selecting the day checkbox, then generates schedule for a given year.	MON unchecked MON 10:15:00 to 23:30:00 Year 1	All courses for each semester in each academic year selected is displayed.	All courses for each semester in each academic year selected is displayed.	**Pass**
6.3	Student selects a day without specifying the time preferences for that day, then generates a schedule for a given year.	WED checked [blank] start/end times Year 2	No courses are displayed: 'no result'	No courses are displayed: 'no result'	**Pass**
6.4	Student selects a day and the times for that day as a preference, and generates the schedule for a given year.	M checked W checked M 8:45 to 13:30 W 8:45 to 13:30 Year 1	Only courses that match the time filter are shown, no result found if no classes match the time criteria.	COMP 248 COMP 249	**Pass**
6.5	Student selects a set of time preferences, but no year and generates the schedule.	M checked W checked M 8:45 to 13:30 W 8:45 to 13:30 No year	No courses are displayed: 'no result'.	No courses are displayed: 'no result'.	**Pass**
6.6	Student clicks 'generate schedule button' with a set of time preferences and for given years, and clicks 'validate schedule' without	M checked W checked M 8:45 to 13:30 W 8:45 to 13:30	Error message indicating to select tutorial and laboratory section pops up.	~~Empty error message pops up:~~ ~~[]~~	~~Fail~~ [older version] **Pass**

	choosing any tutorial and laboratory sections.	Year 1		Error: "No courses were chosen"	
6.7	Student choose a day, a set time preferences with the end time before the start time.	F checked F 10:30 to 8:45 Year 1,2,3,4	No courses are displayed: 'no result'	No courses are displayed: 'no result'	**Pass**

16.2.2.7 View Saved Schedules

UC7	**View Saved Schedules**			
Test ID	Description	Expected Output	Actual Output	Result
7.1	Student chooses to view saved schedules	List of saved schedules is displayed	List of saved schedules is displayed	**Pass**
7.2	Student chooses to view a specific saved schedule	Saved schedule is displayed	Saved schedule is displayed	**Pass**
7.3	Student chooses to delete a saved schedule	A prompt to confirm deletion is given, then after confirmation the saved schedule is removed	A prompt to confirm deletion is given, then after confirmation the saved schedule is removed	**Pass**

16.2.2.8 View Academic Record

UC8		**View Academic Record**			
Test ID	Description	Input	Expected Output	Actual Output	Result
8.1	Student requests to view Academic Record	Click on 'view academic record' link on profile tab	Page with a record of classes passed and the student's gpa is displayed	Page with a record of classes is displayed, but no classes on the record	**Pass**
8.2	Student adds a course to his academic record	Course code: COMP 248 Grade: A	The course has been added to the academic record and the student's GPA has been updated	The course has been added to the academic record and the student's GPA has been updated	**Pass**

8.3	Student adds a course to his academic record, but does not input a course code	Course code: Grade: B	Error: invalid course code	Error: invalid course code	**Pass**
8.4	Student adds a course to his academic record, but does not add a grade	Course code: COMP 249 Grade:	Error: invalid grade	Error: invalid grade	**Pass**
8.5	Student clicks add course button, but no course information is entered into the form	Course code: Grade:	Error: Invalid course code, invalid grade	Error: Invalid course code, invalid grade	**Pass**

16.2.2.9 Drop Course

UC9		**Drop Course**			
Test ID	Description	Input	Expected Output	Actual Output	Result
9.1	Student requests to drop a non-prerequisite course, and confirms the choice	Clicks "drop" button, followed by the "delete" button	Updated schedule is produced and displayed	Updated schedule is produced and displayed	**Pass**
9.2	Student requests to drop a non-prerequisite course, and cancels the confirmation dialog box	Clicks "drop" button, followed by the "cancel" button	Same schedule is displayed	Same schedule is displayed	**Pass**
9.3	Student requests to drop a course that is a prerequisite to another course saved later in the schedule, and confirms the choice	Clicks "drop" button	Message indicating that that course could not be dropped due to it being a prerequisite for a later course that's saved is displayed	The course is successfully dropped from the schedule, and the course requiring that prerequisite stays in the schedule	**Fail**

16.2.2.10 Add Course

~~UC10~~	~~Add Course~~				
~~Test ID~~	~~Description~~	~~Input~~	~~Expected Output~~	~~Actual Output~~	~~Result~~

16.2.2.11 Saved Generated Schedule

UC11	**Save Generated Schedules**			
Test ID	Description	Expected Output	Actual Output	Result
11.1	Student chooses to save a generated schedule	Saved schedule confirmation is displayed	Saved schedule confirmation is displayed	**Pass**
11.2	Student chooses to save a schedule in a semester that already has a saved schedule	Error message indicating the semester already has a schedule saved	Saved schedule confirmation is displayed	Fail

16.2.2.12 View Weekly Schedule

~~UC12~~	~~View Weekly Schedule~~				
~~Test ID~~	~~Description~~	~~Input~~	~~Expected Output~~	~~Actual Output~~	~~Result~~
~~12.1~~	~~Student requests to see a saved schedule.~~	~~-~~	~~Displays a prompt message to let student select a week.~~	~~Not implemented~~	~~Pending~~
~~12.2~~	~~From the prompted message of week selection, student selects a week.~~	~~-~~	~~Displays the student's schedule for the selected week.~~	~~Not implemented~~	~~Pending~~

16.2.2.13 Manage Courses (Administrator)

UC13		**Manage Courses**			
Test ID	Description	Input	Expected Output	Actual Output	Result
13.1	Administrator requests to manage courses	Clicks "manage courses" button	Course management page is displayed	Course management page is displayed	**Pass**
13.2	Administrator enters an existing course I.D. to the search bar	Course I.D. (existent)	Course with the given I.D. is displayed, along with an option to edit the course	Course with the given I.D. is displayed, along with the course code, course description, and an option to view, edit, or delete the course	**Pass**
13.3	Administrator enters a non-existent course I.D. to the search bar	Course I.D. (nonexistent)	Message indicating that a course with that I.D. does not exist	"No results found" message is displayed	**Pass**
13.4	Administrator enters an existent course code to the search bar	Course code (existent)	Course with the given code is displayed, along with an option to edit the course	Course with the given code is displayed, along with the course I.D., course description, and an option to view, edit, or delete the course	**Pass**
13.5	Administrator enters a non-existent course code to the search bar	Course code (nonexistent)	Message indicating that a course with that course code does not exist	"No results found" message is displayed	**Pass**
13.6	Administrator enters an existent course description to the search bar	Course description (existent)	Course with the given description is displayed, along with an option to edit the course	Course with the given description is displayed, along with the course I.D., course code, and an option to view, edit, or delete the course	**Pass**
13.7	Administrator enters a non-existent course description to the search bar	Course description (nonexistent)	Message indicating that a course with that description does not exist	"No results found" message is displayed	**Pass**

13.8	Administrator requests to delete a course and confirms dialog box	Clicks "X" icon, followed by "OK"	Updated course bank without the deleted course is produced and displayed	Updated course bank without the deleted course is produced and displayed	**Pass**
13.9	Administrator requests to delete a course and cancels dialog box	Clicks "X" icon, followed by "Cancel"	Same course bank including the chosen course is displayed with no changes	Same screen as before requesting course deletion is displayed	**Pass**
13.10	Administrator requests to update a course	Clicks "Pencil" icon	Course editing page is displayed	Course editing page is displayed	**Pass**
13.11	Administrator edits the course code of a course	Course code	Updated course bank with the new description is produced and displayed	Updated course details with the new course description is displayed	**Pass**
13.12	Administrator edits the number of credits of a course	Number of credits	Updated course bank with the new number of credits is produced and displayed	Updated course details is displayed, and the new number of credits is produced	**Pass**
13.13	Administrator edits the type of the course	Course type (drop-down choice)	Updated course bank with the new course type is produced and displayed	Updated course details is displayed, and the new course type is produced	**Pass**

16.2.2.14 Create Course

UC17		**Create Course**			
Test ID	Description	Input	Expected Output	Actual Output	Result
14.1	Administrator requests to create a course	Click on 'Create Course' link	Course creation page is displayed	Create Course page is displayed	**Pass**
14.2	Administrator enters a course code, description, amount of credits, and type	Course code: MATH 203 Description: Differential Calculus Credits: 3 Type: Basic Science	Course info is registered and displayed	View Course info is displayed	**Pass**
14.3	Administrator enters a description, credits, and type but no course code	Course code: Description: Differential Calculus Credits: 3 Type: Basic Science	Error message indicating the course ID field has been left blank	Please fix the following input errors: • Course Code cannot be blank.	**Pass**
14.4	Administrator enters a course code, credits, and type but no description	Course code: MATH 203 Description: Credits: 3 Type: Basic Science	Error message indicating the description field has been left blank	Please fix the following input errors: • Course Description cannot be blank.	**Pass**
14.5	Administrator enters a course code, description, and type but no credits	Course code: MATH 203 Description: Differential Calculus	Error message indicating the credits field has been left blank	Please fix the following input errors: • Credits cannot be blank.	**Pass**

		Credits: Type: Basic Science			
14.6	Administrator enters a course code, description, and credits but no type	Course code: MATH 203 Description: Differential Calculus Credits: Type:	Error message indicating the credits field has been left blank	Please fix the following input errors: • CType cannot be blank.	**Pass**

16.2.2.15 View Courses (Administrator)

UC18		**View Courses**			
Test ID	Description	Input	Expected Output	Actual Output	Result
15.1	Administrator requests to view courses	Click on 'Courses' tab	List of all courses in the system is displayed	List of all courses in the system is displayed	**Pass**
15.2	Administrator clicks the create course link	Click on 'create course' link	Create course page is displayed	Create course page is displayed	**Pass**
14.3	Administrator clicks the manage course link	Click on 'manage course' link	Manage courses page is displayed	Manage courses page is displayed	**Pass**
14.4	Administrator clicks on a course prerequisite	Click on 'COMP 248' link in COMP 249 course description	Course information for that course is displayed	Course information for that course is displayed	**Pass**

16.2.2.16 Add User

UC19	**Add User**				
Test ID	Description	Input	Expected Output	Actual Output	Result
16.1	Administrator requests to add a user	User clicks on "Create User" link	Create user page is displayed	Create user page is displayed	**Pass**
16.2	Administrator submits the form with empty fields	User clicks on "Create" button	Error message indicating that all fields have been left blank	Error: please fix the following input errors - Username cannot be blank - Firstname cannot be blank - Lastname cannot be blank - Net Name cannot be blank - Password cannot be blank	**Pass**
16.3	Administrator enters a firstname, lastname, netname, password, but no username, for a student.	User enters "Ryan" for Firstname, "Lee" for Lastname, "NoCellPhone" for Net Name, "soen341" for Password, keep "Student" as privilege and clicks on "Create" button	Error message indicating the Username field has been left blank	Error: please fix the following input errors - Username cannot be blank	**Pass**
16.4	Administrator enters a username, lastname, netname, password, but no firstname, for a student.	User enters "Cark" for Username, "Leclair" for Lastname, "M_Leclair" for Net Name, "soen341" for	Error message indicating the Firstname field has been left blank	Error: please fix the following input errors - Firstname cannot be blank	**Pass**

		Password, keep "Student" as privilege and clicks on "Create" button			
16.5	Administrator enters a username, firstname, netname, password, but no lastname, for a student.	User enters "Alien" for Username, "Aline" for Firstname, "Alien" for Net Name, "soen341" for Password, keep "Student" as privilege and clicks on "Create" button	Error message indicating the Lastname field has been left blank	Error: please fix the following input errors - Lastname cannot be blank	**Pass**
16.6	Administrator enters a username, firstname, lastname, password, but no netname, for a student.	User enters "Backrow" for Username, "Philip" for Firstname, "Lim" for Lastname, "soen341" for Password, keep "Student" as privilege and clicks on "Create" button	Error message indicating the Net Name field has been left blank	Error: please fix the following input errors - Net Name cannot be blank	**Pass**
16.7	Administrator enters a username, firstname, lastname, netname, but no password, for a student.	User enters "Pet" for Username, "Ideawin" for Firstname, "Koun" for Lastname, "Pet" for Net Name, keep "Student" as privilege and clicks on "Create" button	Error message indicating the Password field has been left blank	Error: please fix the following input errors - Password cannot be blank	**Pass**
16.8	Administrator enters a password of less than 6 characters	User enters "Kevyass" for Username, "Kevin" for Firstname,	Error message indicating the Password is too short	Error: please fix the following input errors	**Pass**

		"Yasmine" for Lastname, "Kevyass" for Net Name, "soen" for Password, keep "Student" as privilege and clicks on "Create" button		- Password is too short	
16.9	Administrator enters a username, firstname, lastname, netname, and a password for a student.	User enters "Foxy" for Username, "Lori" for Firstname, "Dalkin" for Lastname, "Foxy" for Net Name, "soen341" for Password, keep "Student" as privilege and clicks on "Create" button	User info is registered and displayed	User info is registered and displayed	**Pass**
16.10	Administrator enters a username, firstname, lastname, netname, and a password for an administrator.	User enters "Ern" for Username, "Erin" for Firstname, "Benderoff" for Lastname, "Ern" for Net Name, "soen341" for Password, select "Administrator" as privilege and clicks on "Create" button	User info is registered and displayed	User info is registered and displayed	**Pass**

16.2.2.17 Search Users

UC20	Search for Users				
Test ID	Description	Input	Expected Output	Actual Output	Result
17.1	Administrator leaves the search field blank and press 'Search' button.	[blank search field]	Displays all users without any filter.	Displays all users without any filter.	**Pass**
17.2	Administrator enters the ID in the ID search field of an inexistent user.	ID = 90	No user found.	No user found.	**Pass**
17.3	Administrator enters the ID in the ID search field of an existent user.	ID = 39	Displays a row corresponding to user with ID 39, ("backrow").	Displays a row corresponding to user with ID 39 ("backrow").	**Pass**
17.4	Administrator uses comparative operators in the ID search field.	ID <= 23	Displays all users with ID less or equal to 23 (19, 20 and 23).	IDs of users displayed: 19 20 23	Pass
17.5	Administrator enters a text in the username search field which is not found in any usernames.	Username = "zx"	No user found.	No user found.	**Pass**
17.6	Administrator enters an existing username in the username search field.	Username = "Claudic"	Displays a row corresponding to user "Claudic"	Displays row corresponding to user "Claudic"	**Pass**
17.7	Administrator enters a portion of an existing username in the username search field.	Username = "b"	Displays all users having "b" in their username ("BDS" and "backrow").	Displays all users having "b" in their username: "BDS" "backrow"	**Pass**
17.8	Administrator enters an existent username in the username search field without	Username = "aDmIn"	Displays a row corresponding to the user "admin".	Displays a row corresponding to the user "admin".	**Pass**

	respecting upper/lower cases of the actual username.				
17.9	Administrator enters a text in the first name search field which is not found in any first names.	First name = “!@#$”	No user found.	No user found.	**Pass**
17.10	Administrator enters an existing first name in the appropriate search field.	First name = “Dimitri”	Displays a row corresponding to the user 35, username being “dimitri”.	Displays a row corresponding to the user 35, username being “dimitri”.	**Pass**
17.11	Administrator enters a portion of an existing first name in the appropriate search field.	First name = “ri”	Displays all users having “ri” in their first name (Dimitri and Erin).	Displays all users having “ri” in their first name (Dimitri and Erin).	**Pass**
17.12	Administrator enters an existent first name in the appropriate search field without respecting upper/lower cases of the actual first name.	First name = “pHIlIp”	Displays a row corresponding to the user having as first name “Philip” (user #39).	Displays a row corresponding to the user having as first name “Philip” (user #39).	**Pass**
17.13	Administrator enters a text in the last name search field which is not found in any last names.	Last name = “Blizerian”	No user found.	No user found.	**Pass**
17.14	Administrator enters an existing last name in the appropriate search field.	Last name = “Koun”	Displays a row corresponding to the user having as last name “Koun” (user #36).	Displays a row corresponding to the user having as last name “Koun” (user #36).	**Pass**

17.15	Administrator enters a portion of an existing last name in the appropriate search field.	Last name = “min”	Displays a row corresponding to the user which last name contains “min” (admin).	Displays a row corresponding to the user which last name contains “min” (admin).	**Pass**
17.16	Administrator enters an existent last name in the appropriate search field without respecting upper/lower cases of the actual first name.	Last name = “tOpALoglOu”	Displays a row corresponding to the user having as last name “Topaloglou”, user #35.	Displays a row corresponding to the user having as last name “Topaloglou”, user #35.	**Pass**
17.17	Administrator enters a text in the net name search field which is not found in any net names.	Net name = “ “ (space)	No user found.	No user found.	**Pass**
17.18	Administrator enters an existing net name in the appropriate search field.	Net name = “p_lim”	Displays a row corresponding to the user having as net name “p_lim”, user #39.	Displays a row corresponding to the user having as net name “p_lim”, user #39.	**Pass**
17.19	Administrator enters a portion of an existing net name in the appropriate search field.	Net name = “_”	Displays all users having “_” in their net name: d_topal1 i_koun p_lim	Displays all users having “_” in their net name: d_topal1 i_koun p_lim	**Pass**
17.20	Administrator enters an existent net name in the appropriate search field without respecting upper/lower cases of the actual first name.	Net name = bRYcE	Displays a row corresponding to the user having as net name “Bryce”, user #20.	Displays a row corresponding to the user having as net name “Bryce”, user #20.	**Pass**

16.2.2.18 View User

UC21	View User				
Test ID	Description	Input	Expected Output	Actual Output	Result
18.1	Administrator clicks the magnifying glass button from a user's row.	Click on the magnifying glass corresponding to user 39.	Displays the profile of user 39, along with administrative operations.	Displays the profile of user 39, along with administrative operations.	**Pass**

16.2.2.19 Update User

UC22	Update User				
Test ID	Description	Input	Expected Output	Actual Output	Result
19.1	Administrator clicks the pencil button from a user's row.	Click on the pencil corresponding to user 39.	Displays the editable profile of the user 39 with current data filled in the editable profile.	Displays the editable profile of the user 39 with current data filled in the editable profile.	**Pass**
19.2	From a user's profile viewed by an administrator, the administrator clicks on the 'update user' operation.	Click on 'update user' from user 39.	Displays the editable profile of the user 39 with current data filled in the editable profile.	Displays the editable profile of the user 39 with current data filled in the editable profile.	**Pass**
19.3	Administrator attempts to save the updated profile with empty username.	Username is empty	Displays an error saying that the username cannot be blank.	Please fix the following input errors: Username cannot be blank.	**Pass**
19.4	Administrator attempts to save the updated profile with empty first name.	First name is empty	Displays an error saying that the first name cannot be blank.	Please fix the following input errors: Firstname cannot be blank.	**Pass**
19.5	Administrator attempts to save the updated profile with empty last name.	Last name is empty	Displays an error saying that the last name cannot be blank.	Please fix the following input errors: Lastname cannot be blank.	**Pass**

19.6	Administrator attempts to save the updated profile with empty net name.	Net name is empty	Displays an error saying that the net name cannot be blank.	Please fix the following input errors: Net Name cannot be blank.	**Pass**
~~19.7~~ Feature removed	~~Administrator attempts to save the updated profile with empty password.~~	~~Password is empty~~	~~Displays an error saying that the password cannot be blank.~~	~~Please fix the following input errors: Password cannot be blank.~~	~~**Pass**~~
19.8	Administrator attempts to save the updated profile with an already existing username.	Username = “admin”, while user is 39	Displays an error saying that the username is already taken.	Please fix the following input errors: Username "admin" has already been taken.	**Pass**
19.9	Administrator attempts to save the updated profile with an already existing net name.	Net name = “i_koun”, while user is 39	Displays an error saying that the net name is already taken.	Please fix the following input errors: Net Name "i_koun" has already been taken.	**Pass**
~~19.10~~ Feature removed	~~Administrator attempts to save the updated profile with a password which length is lower than six characters.~~	~~Password = “a”~~	~~Displays an error saying that the password is too short and that the minimum length is six characters.~~	~~Displays an error saying that the password is too short and that the minimum length is six characters.~~	~~**Pass**~~
19.11	Administrator attempts to save the updated profile with any combination of invalidity of test cases 19.3 to 19.10	Example: All blanks fields.	Displays a list of all invalidity to be corrected.	Please fix the following input errors: Username cannot be blank. Firstname cannot be blank. Lastname cannot be blank. Net Name cannot be blank.	**Pass**
~~19.12~~ Feature removed	~~Administrator hits the save button without changing any fields.~~	~~-~~	~~Updated profile is displayed, all information including password~~	~~Password has been changed~~	~~Fail~~

			~~remains unchanged.~~		
19.13	Administrator attempts to save the updated profile with an inexistent username.	Username = “inexistent”	Updated profile is displayed, all remaining information have remained unchanged.	~~Password has changed.~~	~~Fail~~
				Updated profile is displayed, all remaining information have remained unchanged.	**Pass**
19.14	Administrator attempts to save the updated profile with a new first name.	First name = “Ryce”	Updated profile is displayed, all remaining information have remained unchanged.	~~Password has changed.~~	~~Fail~~ [older version]
				Updated profile is displayed, all remaining information have remained unchanged.	**Pass**
19.15	Administrator attempts to save the updated profile with a new last name.	Last name = “Topoftheigloo”	Updated profile is displayed, all remaining information have remained unchanged.	~~Password has changed.~~	~~Fail~~ [older version]
				Updated profile is displayed, all remaining information have remained unchanged.	**Pass**
19.16	Administrator attempts to save the updated profile with an inexistent net name.	Net name = “inexistent”	Updated profile is displayed, all remaining information have remained unchanged.	~~Password has changed.~~	~~Fail~~ [older version]
				Updated profile is displayed, all remaining information have remained unchanged.	Pass
~~19.17~~ Feature removed	~~Administrator attempts to save the updated profile with a new password which length is greater than six.~~	~~Password = “jdbc123”~~	~~Updated profile is displayed, password for that user has been changed.~~	~~Updated profile is displayed, password for that user has been changed.~~	~~**Pass**~~

19.18	Administrator changes the privilege of the user from student to administrator.	Privilege = administrator on user #39	Updated profile is displayed, user is now administrator, and all remaining information remained unchanged.	~~Password has changed.~~	~~Fail~~ [older version]
				Updated profile is displayed, user is now administrator, and all remaining information remained unchanged.	**Pass**
~~19.19~~ Feature removed.	~~Administrator clicks on the password field but does not change anything, save the profile.~~	-	~~Update profile is displayed, user information and password have remained unchanged.~~	~~Password has been changed.~~	~~Fail~~

16.2.2.20 Delete User

UC23	**Delete User**				
Test ID	Description	Input	Expected Output	Actual Output	Result
20.1	Administrator hit the 'X' button on a row associated to a user in a table of users.	Hit 'X' on user 'test1'	Displays a confirmation message to delete the corresponding user.	Displays a confirmation message to delete the corresponding user.	**Pass**
20.2	From the profile page of a user viewed as an administrator, the administrator clicks the 'Delete User' operation.	Hit 'X' on profile page of user 'test1'	Displays a confirmation message to delete the corresponding user.	Displays a confirmation message to delete the corresponding user.	**Pass**
20.3	The administrator hits the cancel button on the confirmation message to delete a user.	Hit 'cancel' on confirmation message that appeared after hitting 'X' on user 'test1'	Confirmation message disappears, user remains on the system.	Confirmation message disappears, user 'test1' remains on the system.	**Pass**

20.4	The administrator hits the 'OK' button on the confirmation message to delete a user.	Hit 'OK' on confirmation message that appeared after hitting 'X' on user 'test1'	User 'test1' is removed from the system, the administrator is brought back to the user management page with an updated table of users.	User 'test1' is removed from the system, the administrator is brought back to the user management page with an updated table of users.	**Pass**

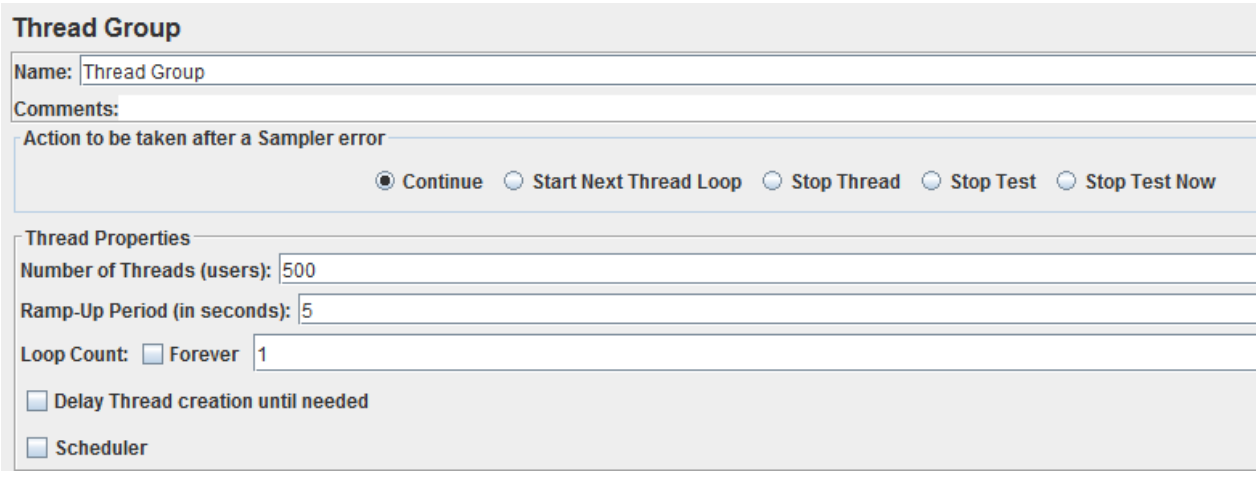
16.2.2.21 Browse Users

UC24	**Browse Users**			
Test ID	Description	Expected Output	Actual Output	Result
18.1	Admin selects user's tab and is populated with all the users	List of users are displayed on screen	List of users are displayed on screen	**Pass**

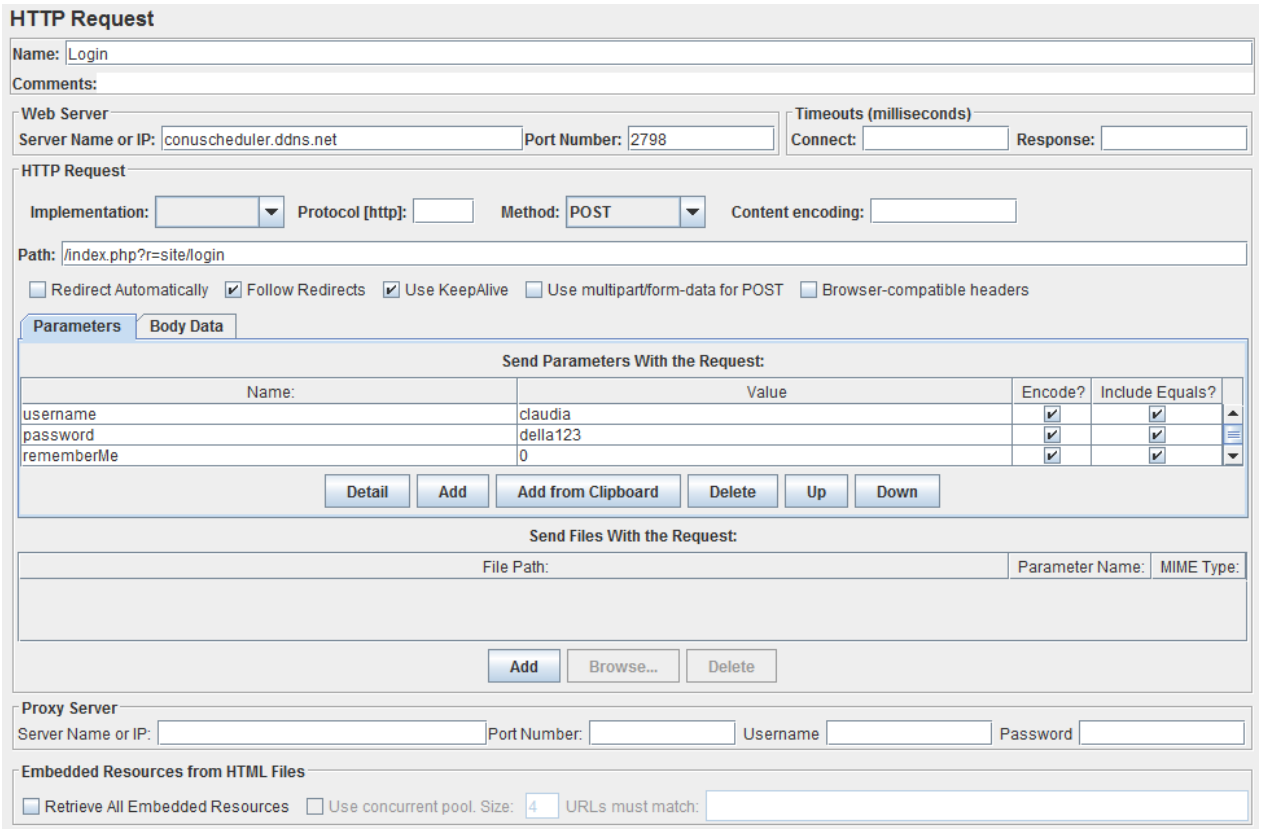
16.2.3 3.3.3 Stress Testing

We ran the JMeter tool with 500 simultaneous users and a 5-second ramp-up period (time to get all threads started). Each thread sent an HTTP request to login to the system and then another request to generate a schedule.

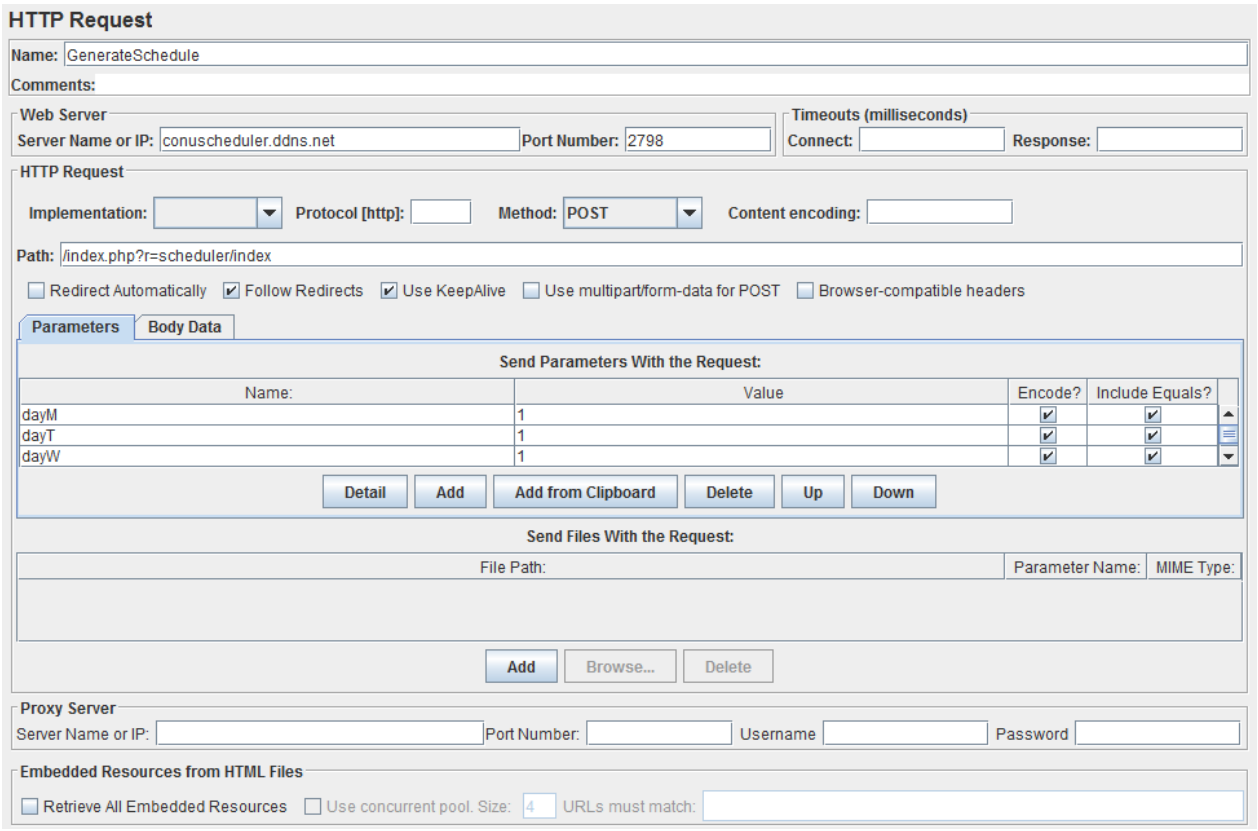
Thread group setup:



Login request setup:

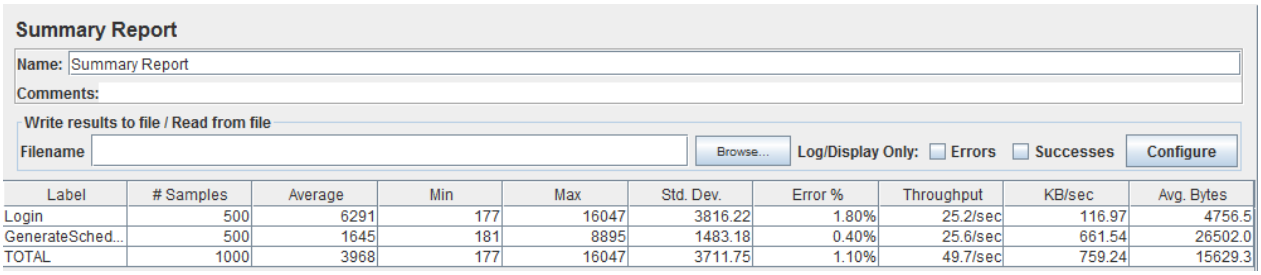


Schedule generation request setup:



Note: the *Parameters* table could not be expanded for the screenshot. The parameters were set up such that the all checkboxes of the preference form were checked off, all start times were set to '8:45:00', and all end times were set to '23:30:00', in order to generate a schedule with as many options as possible and therefore stress the system.

The test resulted in zero errors (response time out, system crash, etc), indicating that it can handle many simultaneous users without major problems. We obtained the following summary report and graph of response time:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login	500	6291	177	16047	3816.22	1.80%	25.2/sec	116.97	4756.5
GenerateSched...	500	1645	181	8895	1483.18	0.40%	25.6/sec	661.54	26502.0
TOTAL	1000	3968	177	16047	3711.75	1.10%	49.7/sec	759.24	15629.3

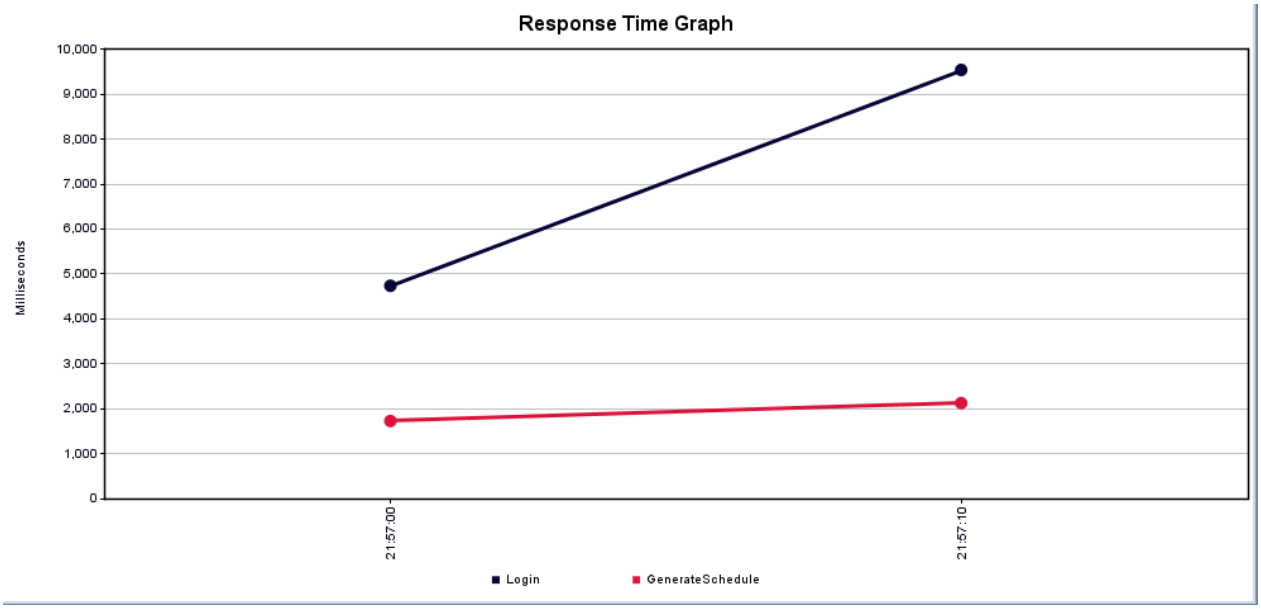


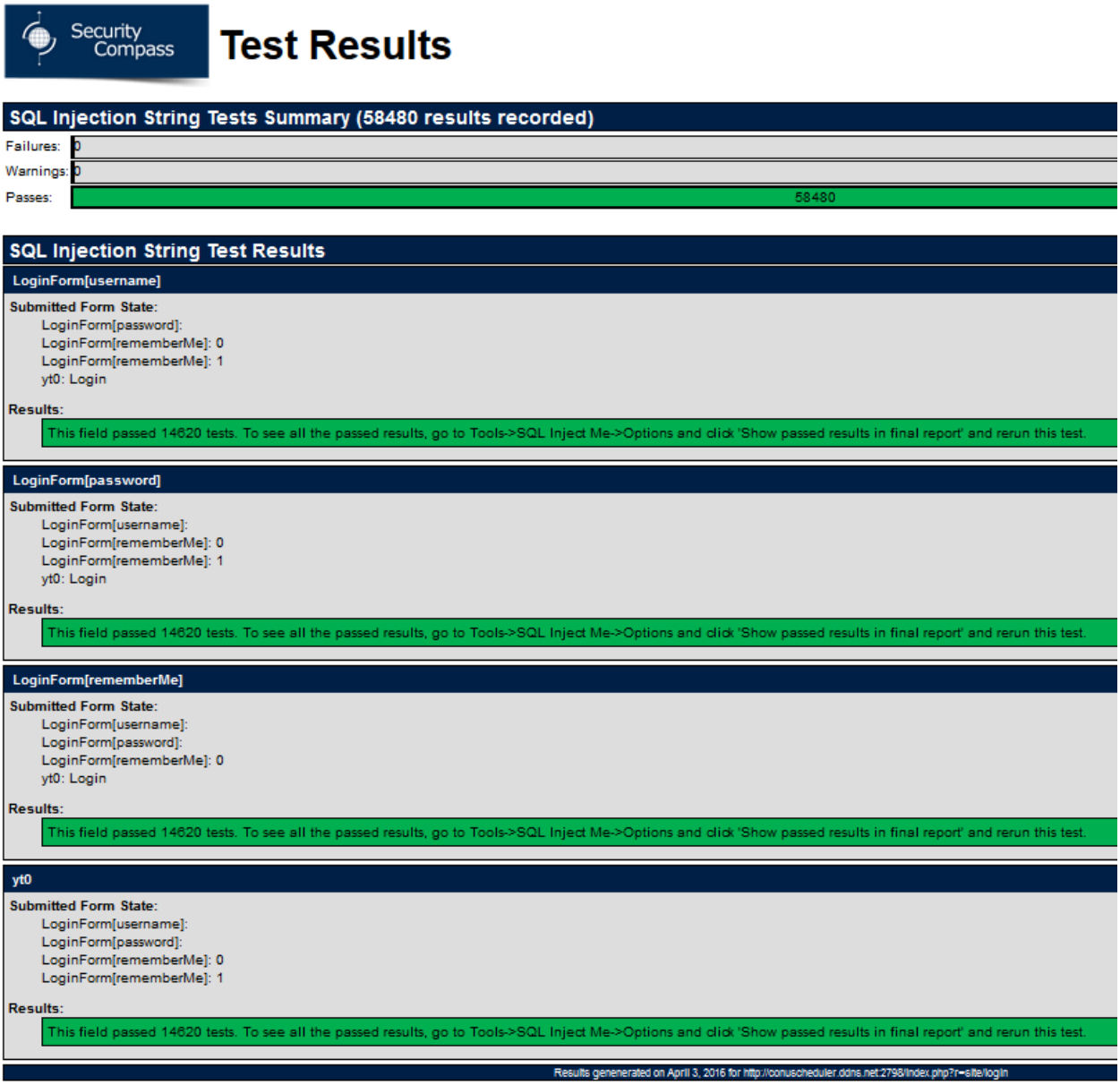
Figure 16.2.3.1. Response time graph

From the above graph, we can see that the response time to login to the system increases as more users try to access it at the same time. However, once logged in, the time to generate a schedule remains fast even when many users are making simultaneous requests.

16.2.4 Security Testing

16.2.4.1 SQL Inject Me LoginForm

The LoginForm passed all 58480 tests, as depicted in the figure below.



16.2.4.2 SQL Inject Me PreferenceForm

The PreferenceForm passed all 292400 tests, as depicted below.

Security Compass

Test Results

SQL Injection String Tests Summary (292400 results recorded)

Failures: 0
Warnings: 0
Passes: 292400

SQL Injection String Test Results

PreferenceForm[dayM]

Submitted Form State:

PreferenceForm[dayM]: 0
PreferenceForm[fromTimeM]:
PreferenceForm[toTimeM]:
PreferenceForm[dayT]: 0
PreferenceForm[dayT]: 1
PreferenceForm[fromTimeT]:
PreferenceForm[toTimeT]:
PreferenceForm[dayW]: 0
PreferenceForm[dayW]: 1
PreferenceForm[fromTimeW]:
PreferenceForm[toTimeW]:
PreferenceForm[dayJ]: 0
PreferenceForm[dayJ]: 1
PreferenceForm[fromTimeJ]:
PreferenceForm[toTimeJ]:
PreferenceForm[dayF]: 0
PreferenceForm[dayF]: 1
PreferenceForm[fromTimeF]:
PreferenceForm[toTimeF]:
PreferenceForm[year1]: 0
PreferenceForm[year1]: 1
PreferenceForm[year2]: 0
PreferenceForm[year2]: 1
PreferenceForm[year3]: 0
PreferenceForm[year3]: 1
PreferenceForm[year4]: 0
PreferenceForm[year4]: 1
yt0: Generate Schedule

Results:

This field passed 14620 tests. To see all the passed results, go to Tools->SQL Inject Me->Options and click 'Show passed results in final report' and rerun this test.

... (results not shown) ...

yt0

Submitted Form State:

PreferenceForm[dayM]: 0
PreferenceForm[dayM]: 1
PreferenceForm[fromTimeM]:
PreferenceForm[toTimeM]:
PreferenceForm[dayT]: 0
PreferenceForm[dayT]: 1
PreferenceForm[fromTimeT]:
PreferenceForm[toTimeT]:
PreferenceForm[dayW]: 0
PreferenceForm[dayW]: 1
PreferenceForm[fromTimeW]:
PreferenceForm[toTimeW]:
PreferenceForm[dayJ]: 0
PreferenceForm[dayJ]: 1
PreferenceForm[fromTimeJ]:
PreferenceForm[toTimeJ]:
PreferenceForm[dayF]: 0
PreferenceForm[dayF]: 1
PreferenceForm[fromTimeF]:
PreferenceForm[toTimeF]:
PreferenceForm[year1]: 0
PreferenceForm[year1]: 1
PreferenceForm[year2]: 0
PreferenceForm[year2]: 1
PreferenceForm[year3]: 0
PreferenceForm[year3]: 1
PreferenceForm[year4]: 0
PreferenceForm[year4]: 1

Results:

This field passed 14620 tests. To see all the passed results, go to Tools->SQL Inject Me->Options and click 'Show passed results in final report' and rerun this test.

Results generenated on April 3, 2016 for http://conuscheduler.ddns.net:2798/index.php?r=scheduler/index

16.2.4.3 Nikto results:

conuscheduler.ddns.net / 96.23.131.61 port 2798	
Target IP	96.23.131.61
Target hostname	conuscheduler.ddns.net
Target Port	2798
HTTP Server	Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7
Site Link (Name)	http://conuscheduler.ddns.net:2798/
Site Link (IP)	http://96.23.131.61:2798/

URI	/
HTTP Method	GET
Description	Cookie PHPSESSID created without the httponly flag
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	GET
Description	Retrieved x-powered-by header: PHP/5.4.7
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	GET
Description	The anti-clickjacking X-Frame-Options header is not present.
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	GET
Description	The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	GET
Description	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
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0

URI	/index
HTTP Method	GET
Description	Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives for 'index' were found: HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var
Test Links	http://conuscheduler.ddns.net:2798/index http://96.23.131.61:2798/index
OSVDB Entries	OSVDB-0

URI	
HTTP Method	HEAD
Description	Apache/2.4.3 appears to be outdated (current is at least Apache/2.4.18). Apache 2.2.31 is also current for the 2.x branch.
Test Links	http://conuscheduler.ddns.net:2798 http://96.23.131.61:2798
OSVDB Entries	OSVDB-0

URI	
HTTP Method	HEAD
Description	PHP/5.4.7 appears to be outdated (current is at least 5.6.17). PHP 5.5.25 and 5.4.41 are also current.
Test Links	http://conuscheduler.ddns.net:2798 http://96.23.131.61:2798
OSVDB Entries	OSVDB-0

URI	
HTTP Method	HEAD
Description	OpenSSL/1.0.1c appears to be outdated (current is at least 1.0.1q). OpenSSL 1.0.0o and 0.9.8zc are also current.
Test Links	http://conuscheduler.ddns.net:2798 http://96.23.131.61:2798
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	UEDEPNGC
Description	Web Server returns a valid response with junk HTTP methods, this may cause false positives.
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-0
URI	/
HTTP Method	TRACE
Description	HTTP TRACE method is active, suggesting the host is vulnerable to XST
Test Links	http://conuscheduler.ddns.net:2798/ http://96.23.131.61:2798/
OSVDB Entries	OSVDB-877
URI	/webalizer/
HTTP Method	GET
Description	Server leaks inodes via ETags, header found with file /webalizer/, fields: 0xf8a 0x4bdcd7fdd5680
Test Links	http://conuscheduler.ddns.net:2798/webalizer/ http://96.23.131.61:2798/webalizer/
OSVDB Entries	OSVDB-0
URI	/webalizer/
HTTP Method	GET
Description	/webalizer/: Webalizer may be installed. Versions lower than 2.01-09 vulnerable to Cross Site Scripting (XSS).
Test Links	http://conuscheduler.ddns.net:2798/webalizer/ http://96.23.131.61:2798/webalizer/
OSVDB Entries	OSVDB-682
URI	/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000
HTTP Method	GET
Description	/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
Test Links	http://conuscheduler.ddns.net:2798/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000 http://96.23.131.61:2798/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000
OSVDB Entries	OSVDB-12184
URI	/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42
HTTP Method	GET
Description	/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
Test Links	http://conuscheduler.ddns.net:2798/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42 http://96.23.131.61:2798/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42
OSVDB Entries	OSVDB-12184
URI	/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42
HTTP Method	GET
Description	/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
Test Links	http://conuscheduler.ddns.net:2798/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42 http://96.23.131.61:2798/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42
OSVDB Entries	OSVDB-12184
URI	/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42
HTTP Method	GET
Description	/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
Test Links	http://conuscheduler.ddns.net:2798/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42 http://96.23.131.61:2798/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42
OSVDB Entries	OSVDB-12184
URI	/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42
HTTP Method	GET
Description	/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
Test Links	http://conuscheduler.ddns.net:2798/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42 http://96.23.131.61:2798/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42
OSVDB Entries	OSVDB-12184
URI	/phpmyadmin/changelog.php
HTTP Method	GET
Description	Uncommon header 'x-ob_mode' found, with contents: 0
Test Links	http://conuscheduler.ddns.net:2798/phpmyadmin/changelog.php http://96.23.131.61:2798/phpmyadmin/changelog.php
OSVDB Entries	OSVDB-0
URI	/phpmyadmin/changelog.php
HTTP Method	GET
Description	Default account found for 'phpMyAdmin 127.0.0.1' at /phpmyadmin/changelog.php (ID 'admin', PW ''). Generic account discovered.
Test Links	http://conuscheduler.ddns.net:2798/phpmyadmin/changelog.php http://96.23.131.61:2798/phpmyadmin/changelog.php
OSVDB Entries	OSVDB-0
URI	/phpmyadmin/changelog.php
HTTP Method	GET
Description	/phpmyadmin/changelog.php: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
Test Links	http://conuscheduler.ddns.net:2798/phpmyadmin/changelog.php http://96.23.131.61:2798/phpmyadmin/changelog.php
OSVDB Entries	OSVDB-3092

URI	/phpmyadmin/ChangeLog
HTTP Method	GET
Description	/phpmyadmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
Test Links	http://conuscheduler.ddns.net:2798/phpmyadmin/ChangeLog http://96.23.131.61:2798/phpmyadmin/ChangeLog
OSVDB Entries	OSVDB-3092
URI	/icons/
HTTP Method	GET
Description	/icons/: Directory indexing found.
Test Links	http://conuscheduler.ddns.net:2798/icons/ http://96.23.131.61:2798/icons/
OSVDB Entries	OSVDB-3268
URI	/images/
HTTP Method	GET
Description	/images/: Directory indexing found.
Test Links	http://conuscheduler.ddns.net:2798/images/ http://96.23.131.61:2798/images/
OSVDB Entries	OSVDB-3268
URI	/images/?pattern=/etc/*&sort=name
HTTP Method	GET
Description	/images/?pattern=/etc/*&sort=name: Directory indexing found.
Test Links	http://conuscheduler.ddns.net:2798/images/?pattern=/etc/*&sort=name http://96.23.131.61:2798/images/?pattern=/etc/*&sort=name
OSVDB Entries	OSVDB-3268
URI	/icons/README
HTTP Method	GET
Description	/icons/README: Apache default file found.
Test Links	http://conuscheduler.ddns.net:2798/icons/README http://96.23.131.61:2798/icons/README
OSVDB Entries	OSVDB-3233
URI	/phpmyadmin/
HTTP Method	GET
Description	/phpmyadmin/: phpMyAdmin directory found
Test Links	http://conuscheduler.ddns.net:2798/phpmyadmin/ http://96.23.131.61:2798/phpmyadmin/
OSVDB Entries	OSVDB-0

Host Summary

Start Time	2016-04-04 01:31:31
End Time	2016-04-04 01:40:51
Elapsed Time	560 seconds
Statistics	8421 requests, 0 errors, 26 findings

Scan Summary

Software Details	Nikto 2.1.6
CLI Options	-h https://conuscheduler.ddns.net:2798/ -output html
Hosts Tested	1
Start Time	Mon Apr 4 01:21:27 2016
End Time	Mon Apr 4 01:40:51 2016
Elapsed Time	1164 seconds

17. System Delivery

17.1 Installation Manual

17.1.1 Description

With this product, provided to clients as an archived or zipped folder, only a few simple steps are needed to install this on a client's server. The zip folder includes the root folder of the web application, the database file and Xampp program.

17.1.2 System requirements

- Minimum 2gb of ram
- Windows Vista or newer
- Database included with Xampp,.db file included in zipped folder

17.1.3 Installation steps

Step 1: Extract the timeturner.zip folder to your preferred location.

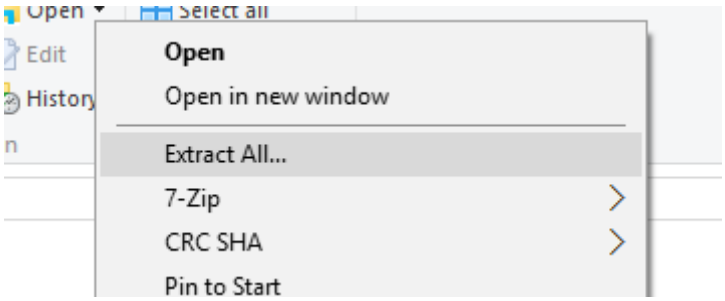


Figure 17.1.1. Folder Extraction

Step 2: Locate the 'Xampp' executable and install the program to your preferred location. No special changes made when installing, follow the installation steps

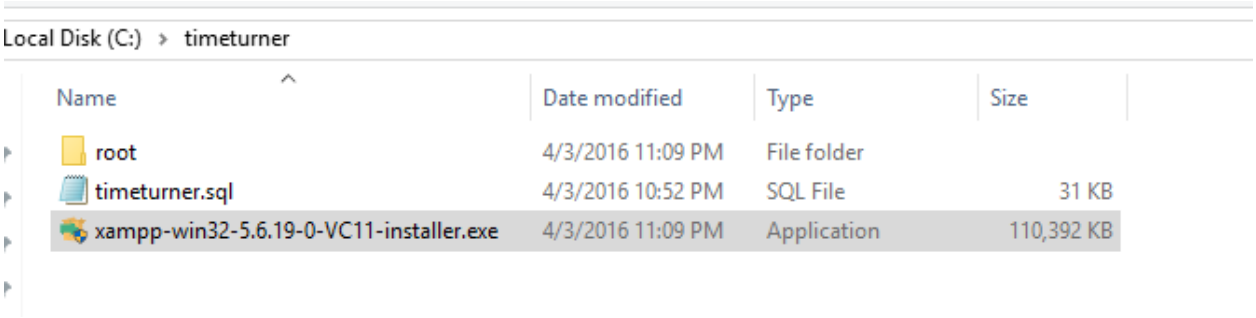


Figure 17.1.2 Xampp location

Step 3: Setup the Virtual Server

Step 3.1: Go to the installed 'Xampp' folder

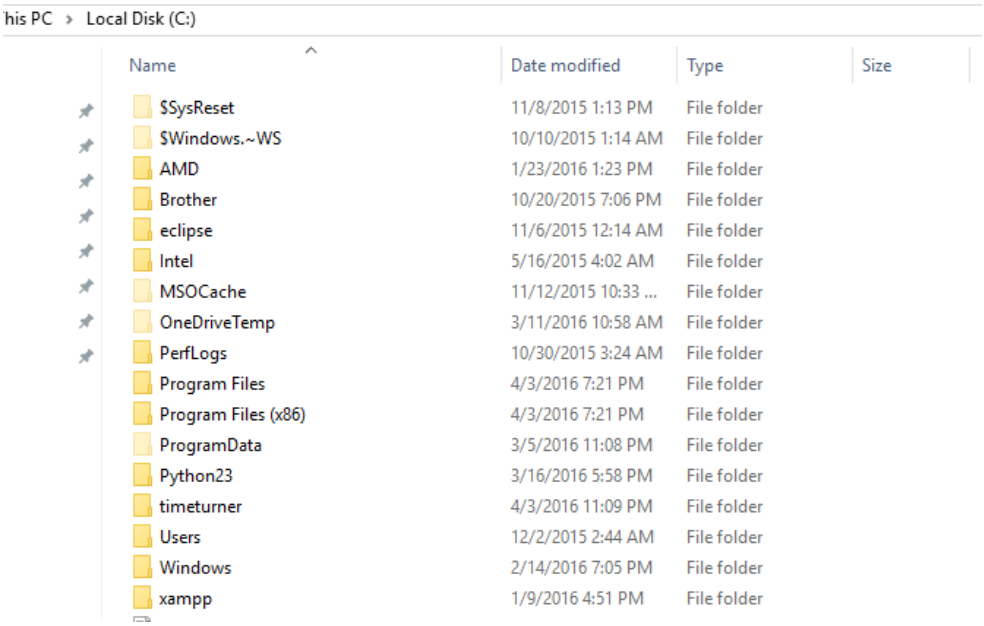


Figure 17.1.3. Xampp folder location (last folder)

Step 3.2: Go to the Apache folder

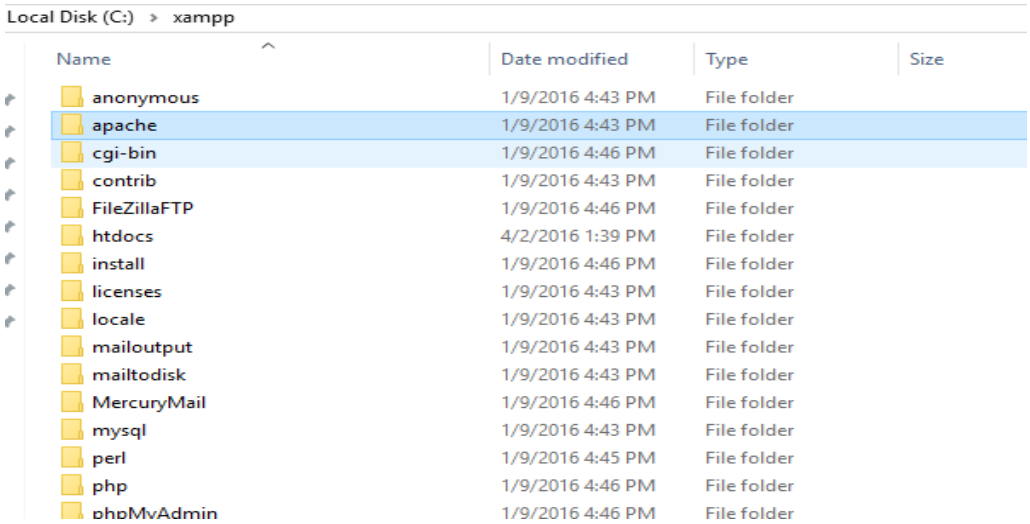


Figure 17.1.4 Apache folder location

Step 3.3: Locate conf folder in apache

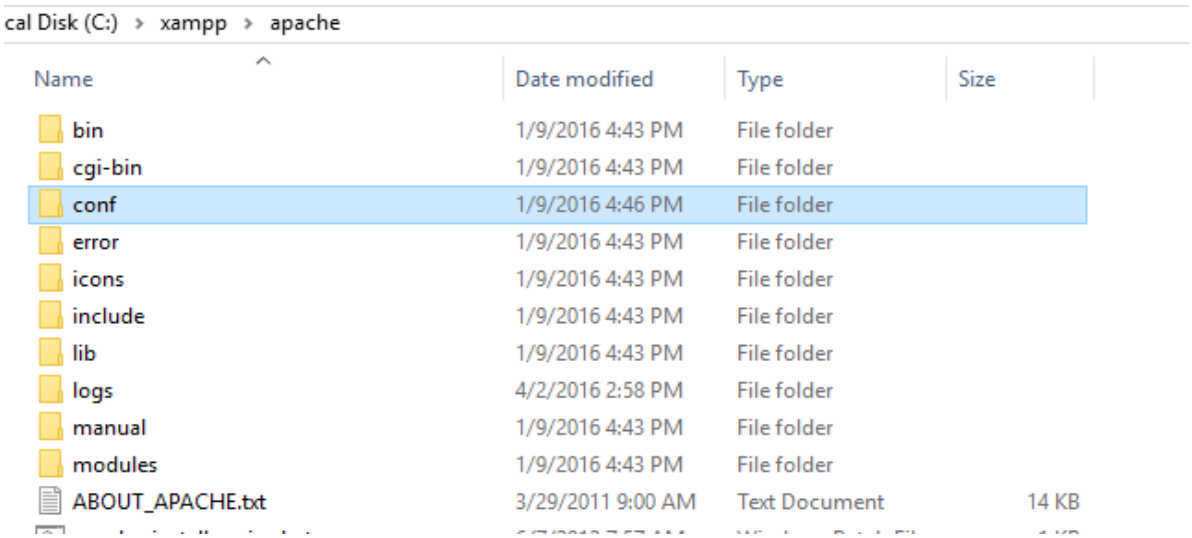


Figure 17.1.5 conf folder location

Step 3.4: Go to extra

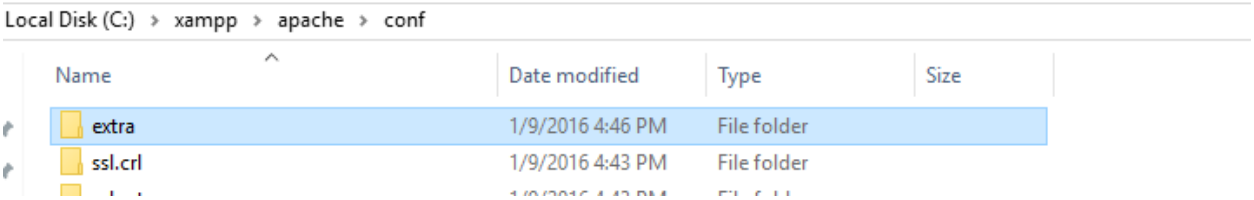


Figure 17.1.6. Extra folder location

Step 3.5: Open the httpd-vhost.conf file

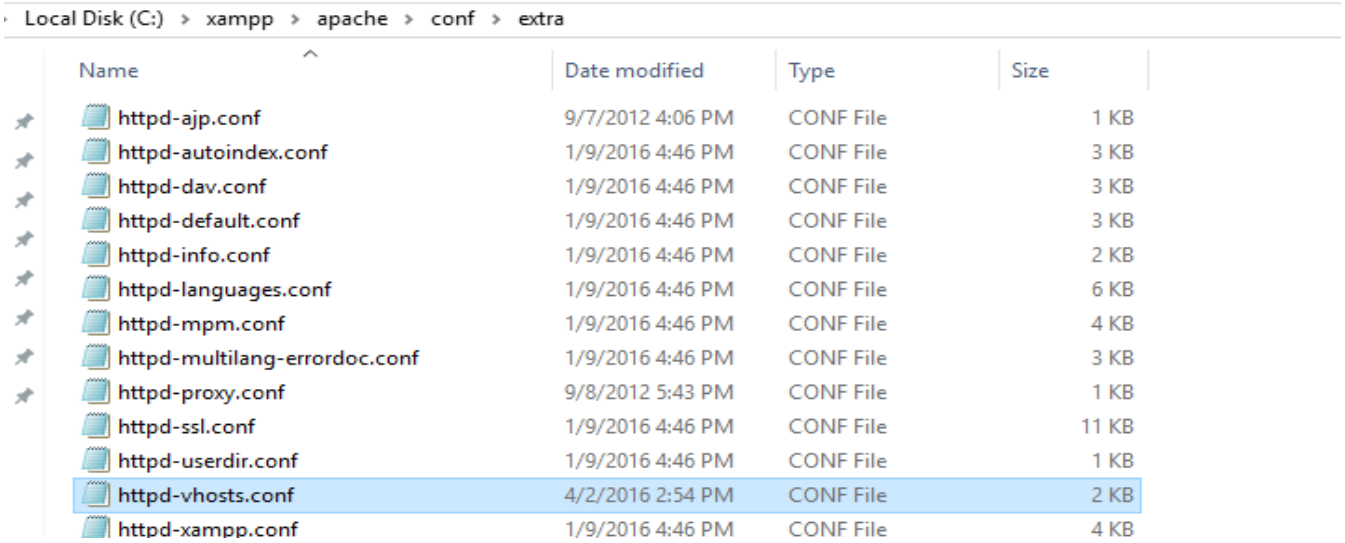


Figure 17.1.7 httpd-vhost.conf file

Step 3.6: Set up virtual server by including the appropriate lines of code provided and visualized in the next screen capture:

```
##<VirtualHost *:80>
    ##ServerAdmin postmaster@dummy-host.localhost
    ##DocumentRoot "C:/xampp/htdocs/dummy-host.localhost"
    ##ServerName dummy-host.localhost
    ##ServerAlias www.dummy-host.localhost
    ##ErrorLog "logs/dummy-host.localhost-error.log"
    ##CustomLog "logs/dummy-host.localhost-access.log" combined
##</VirtualHost>
```

Figure 17.1.8 Virtual server setup code

Remove the hashtags from all the lines. The ServerName should be the name of the domain that will be used to access the application via URL. The ServerAlias allows to access the domain through a 'www' protocol. For example, if the preferred domain is 'www.mytimeturner.com' the ServerName would be **mytimeturner** and the ServerAlias would be 'www.mytimeturner.com'. The DocumentRoot is the location of the web application root folder, which is provided in the zip folder of the application. The **root** folder provided in the zip file, should be extracted in the same location as the DocumentRoot location. Once provided with valid DocumentRoot location folder, extract the 'root' folder in that location.

Step 4: Redirecting windows hosts

Step 4.1: Once the virtual host is setup, the new domain name that was provided in the ServerAlias must be also setup in the Windows host file. To locate the Windows host file:

C:\Windows\System32\Drivers\etc\host.file

Step 4.2: Open the file and append the following lines:

127.0.0.1 www.mytimeturner.com

This simply tells the operating system to resolve the domain as the local machine. Of course, www.mytimeturner.com should match exactly what was setup in the Virtual Host setup ServerAlias name. Save the file.

NOTE: Do NOT continue installation after this step until conducting installation testing (see Section 4.1.2).

Step 5: Database Installation

Step 5.1: To install the database, a database file is provided in the ZIP folder called 'timeturner.db.' This file is to be extracted in phpmyadmin.

Step 5.2: At this point, we need to create a database in order to import the existing database file. Click on database, and create a database under the name: 'timeturner'

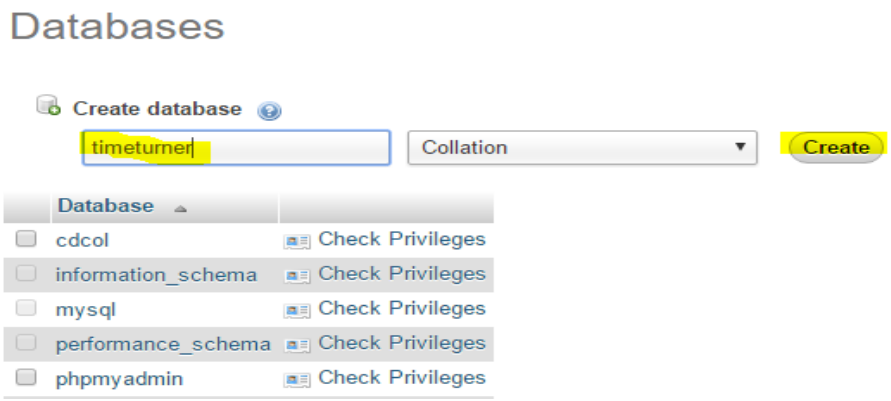


Figure 17.1.9 Create database

Step 5.3: Click on 'Create'. This will create a new database that will appear on the left hand column where the rest of the databases are located. From here, the actual database needs to be imported in order for the application to run.

Step 6: Database Extraction

Step 6.1: To extract the database, click on the database name 'timeturner' on the left and then click on import.

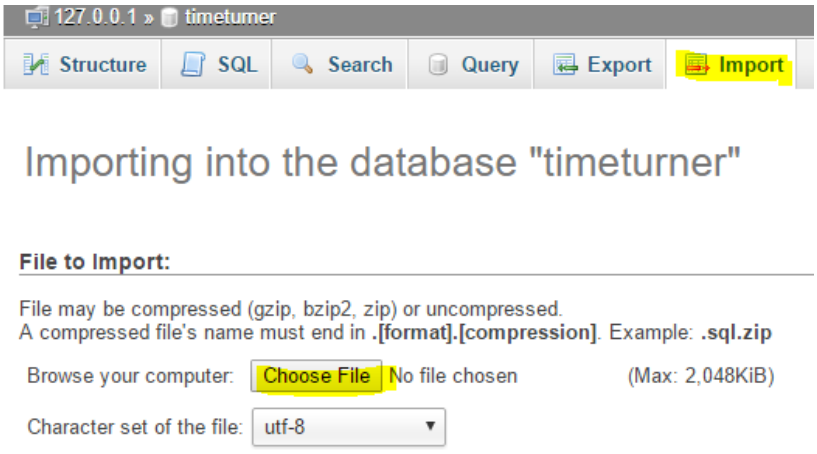


Figure 17.1.10 Database extraction

Step 6.2: Upload the 'timeturner.sql' file provided and then click on 'Go'. At this point, all the data tables should appear in the timeturner database:

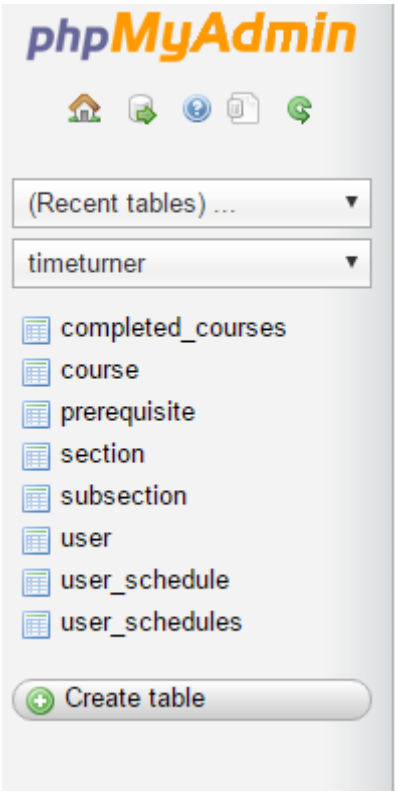


Figure 17.1.11 Database tables

Final Installation Testing

To run the application, open a web browser and type in the domain name provided earlier through ServerAlias, which in this case would be 'www.timeturner.com'

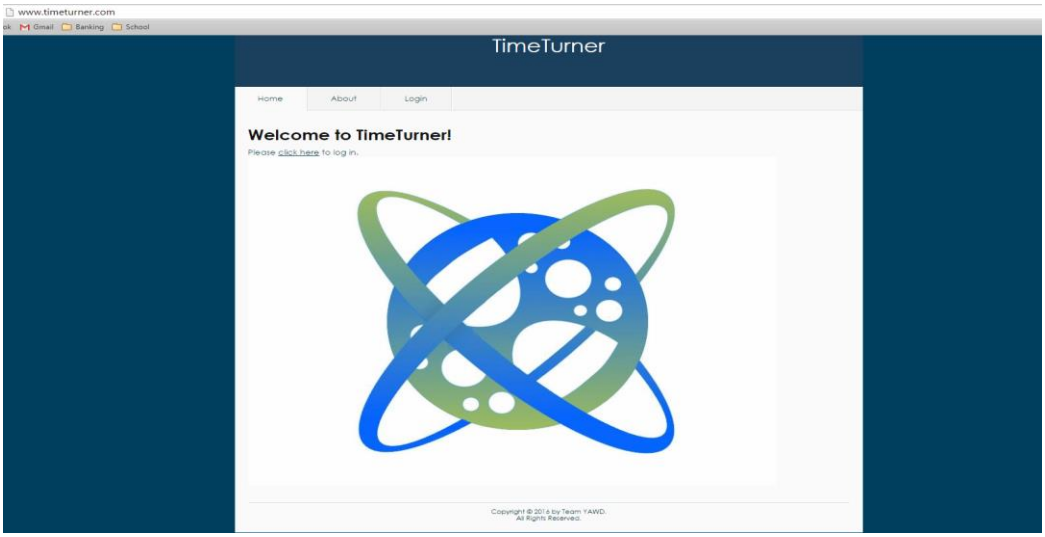


Figure 17.1.12 Final Installation screen

17.1.4 Virtual Server Installation Testing

In order to test if the virtual server was setup properly, locate the 'XAMPP' installation folder and locate the xampp-control.exe file:

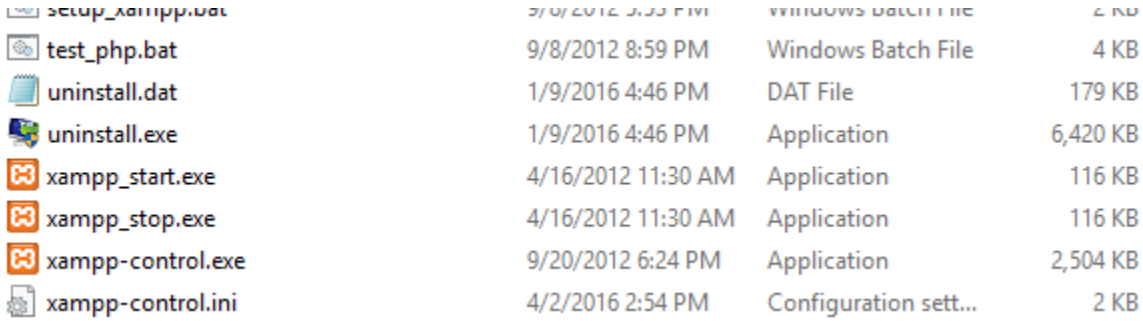


Figure 17.1.13 xampp-control.exe file

Once XAMPP runs, click on 'Start' on the Apache line. If the server was setup properly, the following should appear:

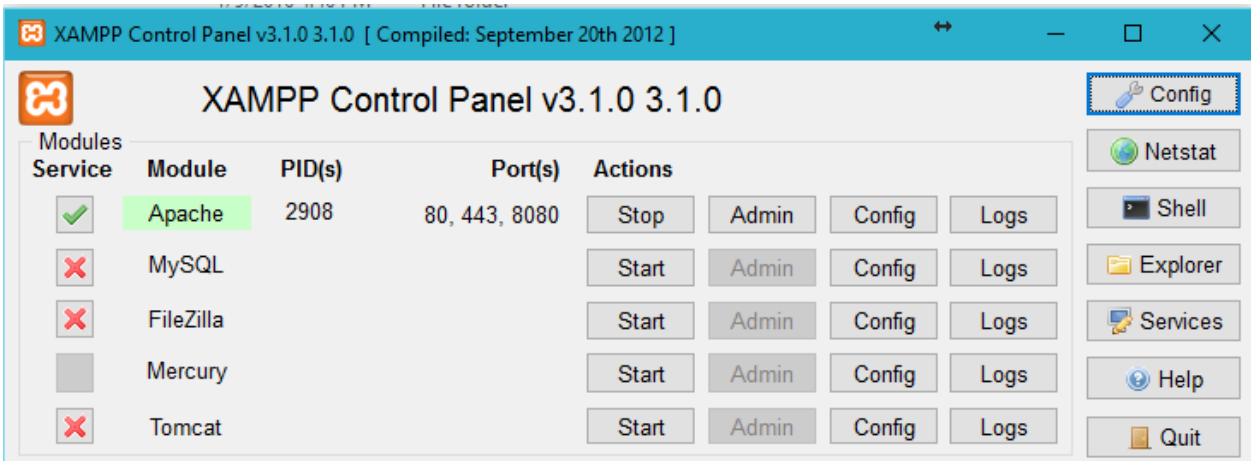


Figure 17.1.14 Proper running of Xampp server

A green indicator should appear on Apache.

Once the Apache server is setup, click on 'Start' for the MySQL line. Again, a green indicator light should highlight the MySQL. If the green light appears, click on 'Admin'. This will open up phpmyadmin, a GUI provided by XAMPP for the MySQL database.

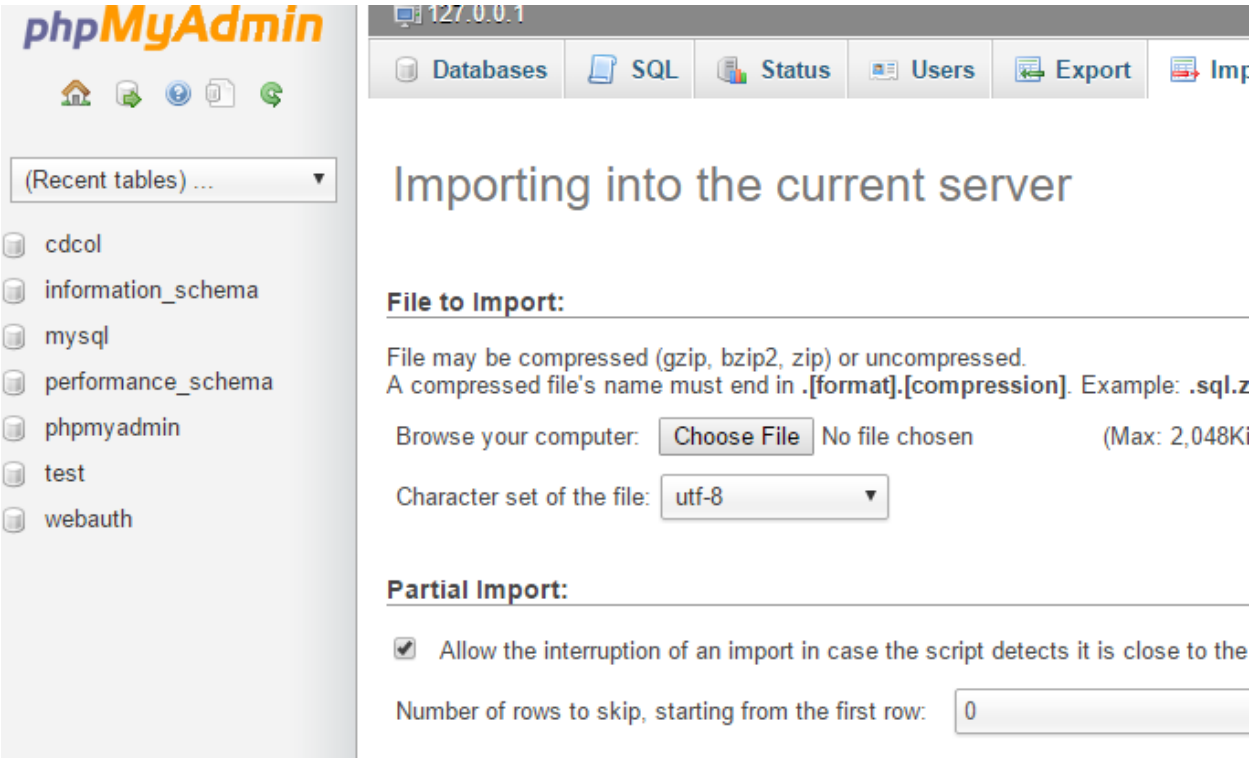


Figure 17.1.15 phpmyadmin GUI

17.2 User Manual

17.2.1 Student Users

17.2.1.1 Login

To log in, head to the 'Login' tab from the main screen as outlined below and fill out the user information (see Figure . Finally, press the 'Login' button at the bottom, and if you wish for the browser to remember the identification, select the "Remember me next time" checkbox.

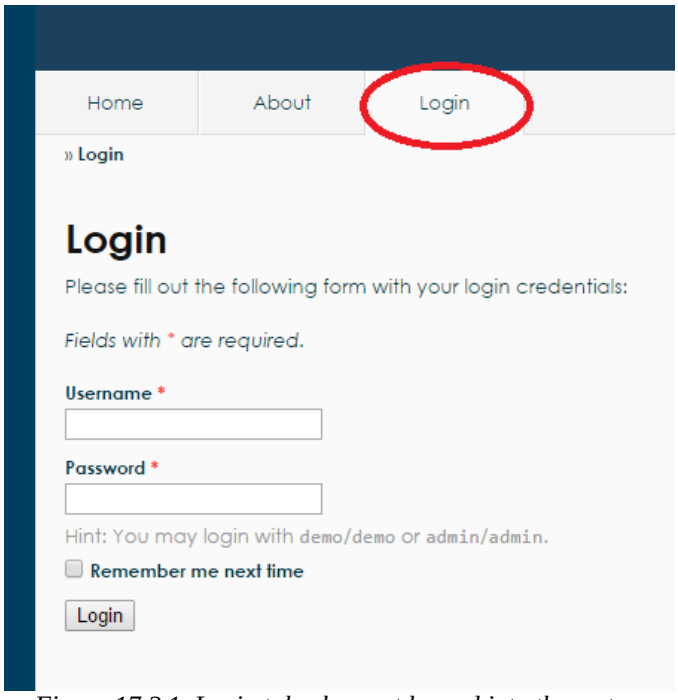


Figure 17.2.1. Login tab when not logged into the system

17.2.1.2 Logout

To log out, simply press the 'Logout' button as outlined below, and you will be redirected to the main page.

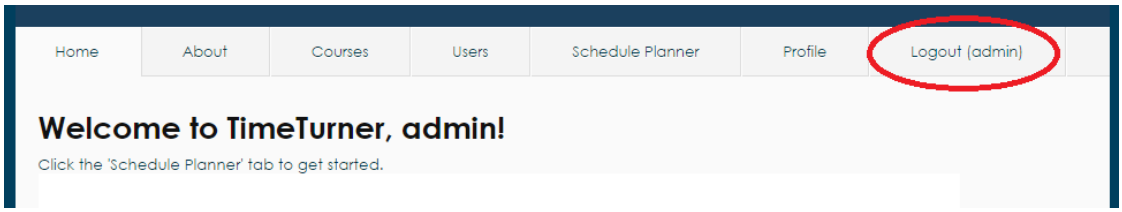


Figure 17.2.2 Logout Tab when logged into the system

17.2.1.3 Browse Course List

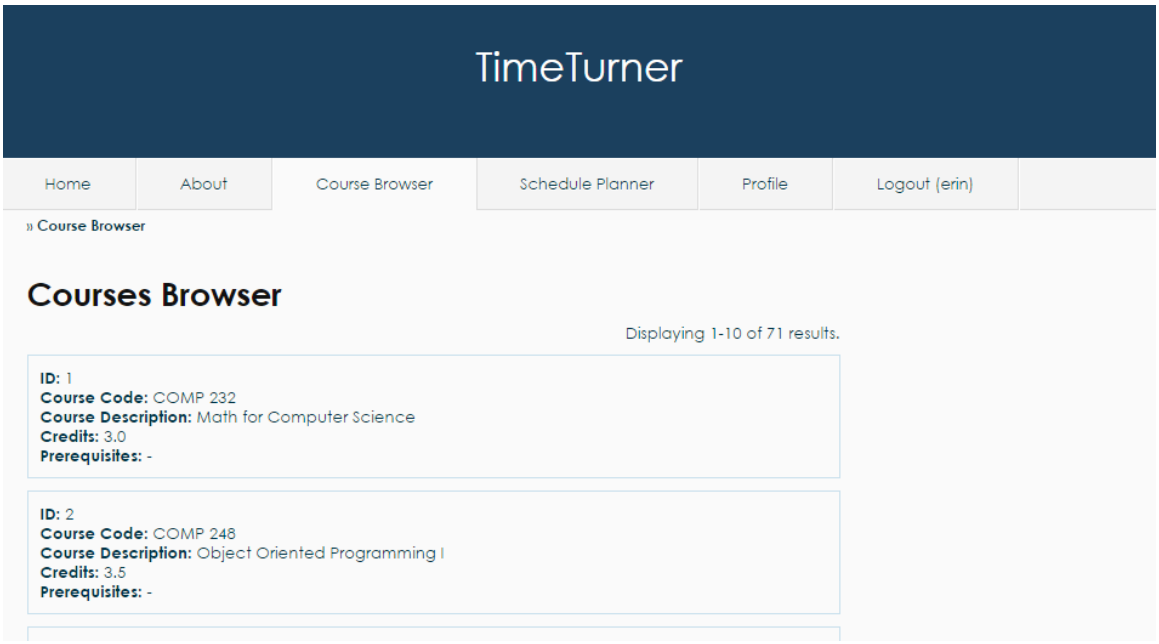


Figure 17.2.3. Browse Course List functionality

If the user wishes to view the list of all available courses, the user must click on the 'Course Browser' tab, and a list will be displayed in an interval of 10 courses (see Figure 17.2.3). To view more courses, the user may scroll down the page and click on the desired page or click 'Next' to view the next 10 courses (see Figure 17.2.4).

Figure 17.2.4. View more courses

17.2.1.4 Generate Schedule

Students can generate a schedule based on preferences. To do so, students may go to the 'Schedule Planner' tab from the home page after logging in (see Figure 17.2.5).

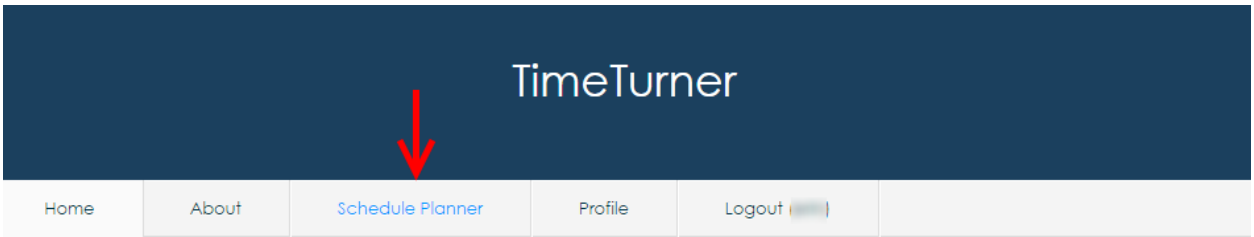


Figure 17.2.5 Accessibility to the 'Schedule Planner' tab from the home page.

Once the tab is selected, a set of preferences is displayed along with the 'Generate Schedule' button and a set of operations (see Figure 17.2.6).

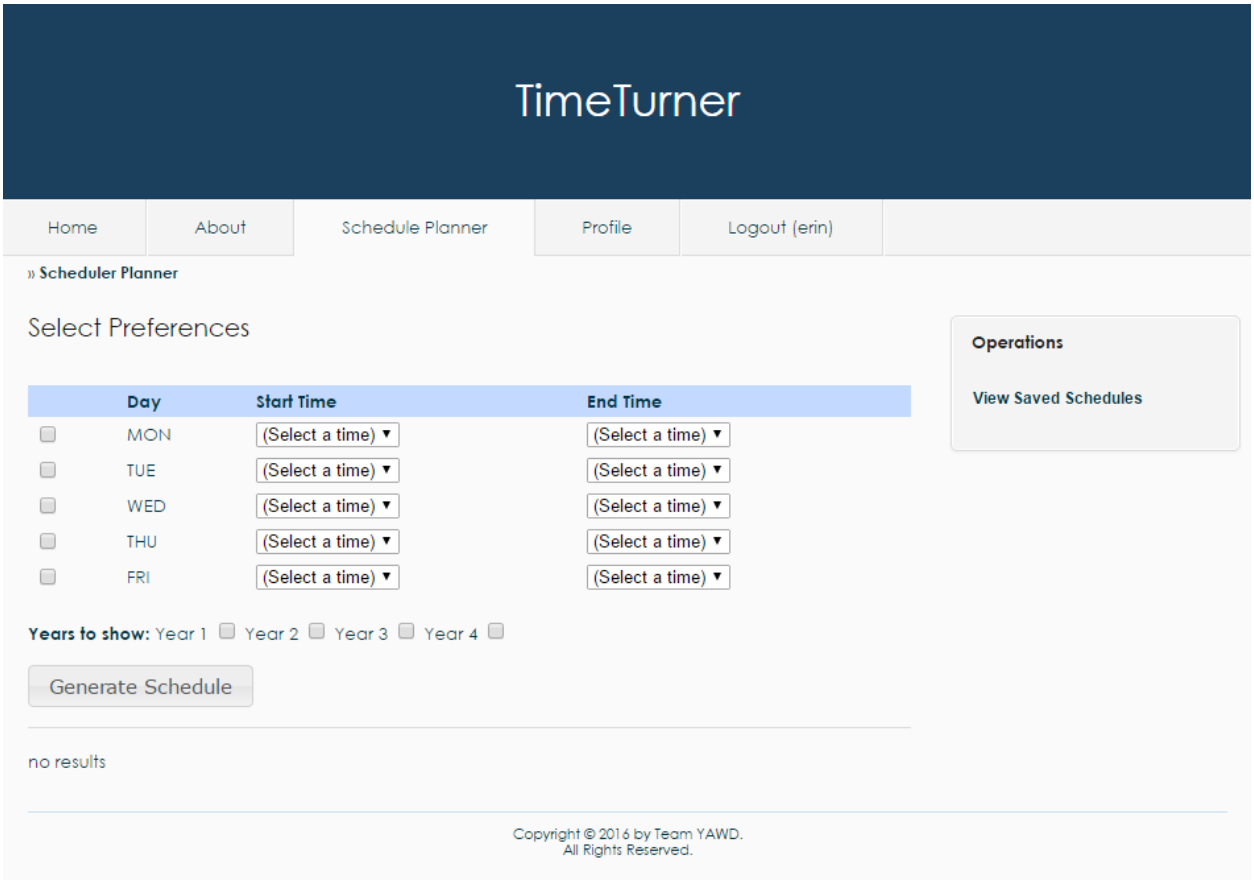


Figure 17.2.6. Set of preferences displayed on the 'Schedule Planner' tab.

The preferences include the starting and ending time at which the student would prefer having classes for each week day. Students can choose to apply the preference filters for the days desired by selecting the checkboxes corresponding to the day (see Figure 17.2.7).

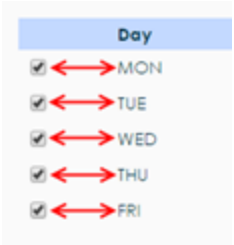


Figure 17.2.7. Checkboxes corresponding to the day to be filtered.

Students may then select the times at which they wish to start and end their courses by scrolling down and choosing the offered times from the drop-down menus (see Figure 17.2.8).

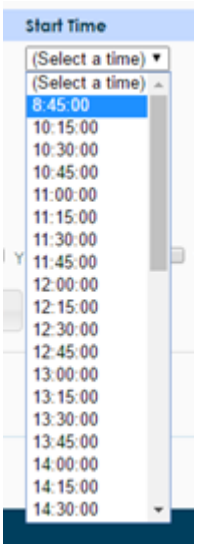


Figure 17.2.8. Drop-down menu to select time preferences.

Once the preferences are set, the year(s) to be displayed can be chosen by click the appropriate checkboxes (see Figure 17.2.9). The system will generate schedule based on the preferences only for the years selected.

Figure 17.2.9. Academic years to be displayed according to the set preferences.

The next step is then to generate the schedule by clicking the 'Generate Schedule' button. A set of tabs, each representing an academic year, will be displayed on the bottom of the page. Each tab is divided into semesters. Every semester division contains a list of proposed lectures, tutorials and laboratories for all courses in the sequence that correspond to the time blocks selected (see Figure 17.2.10).

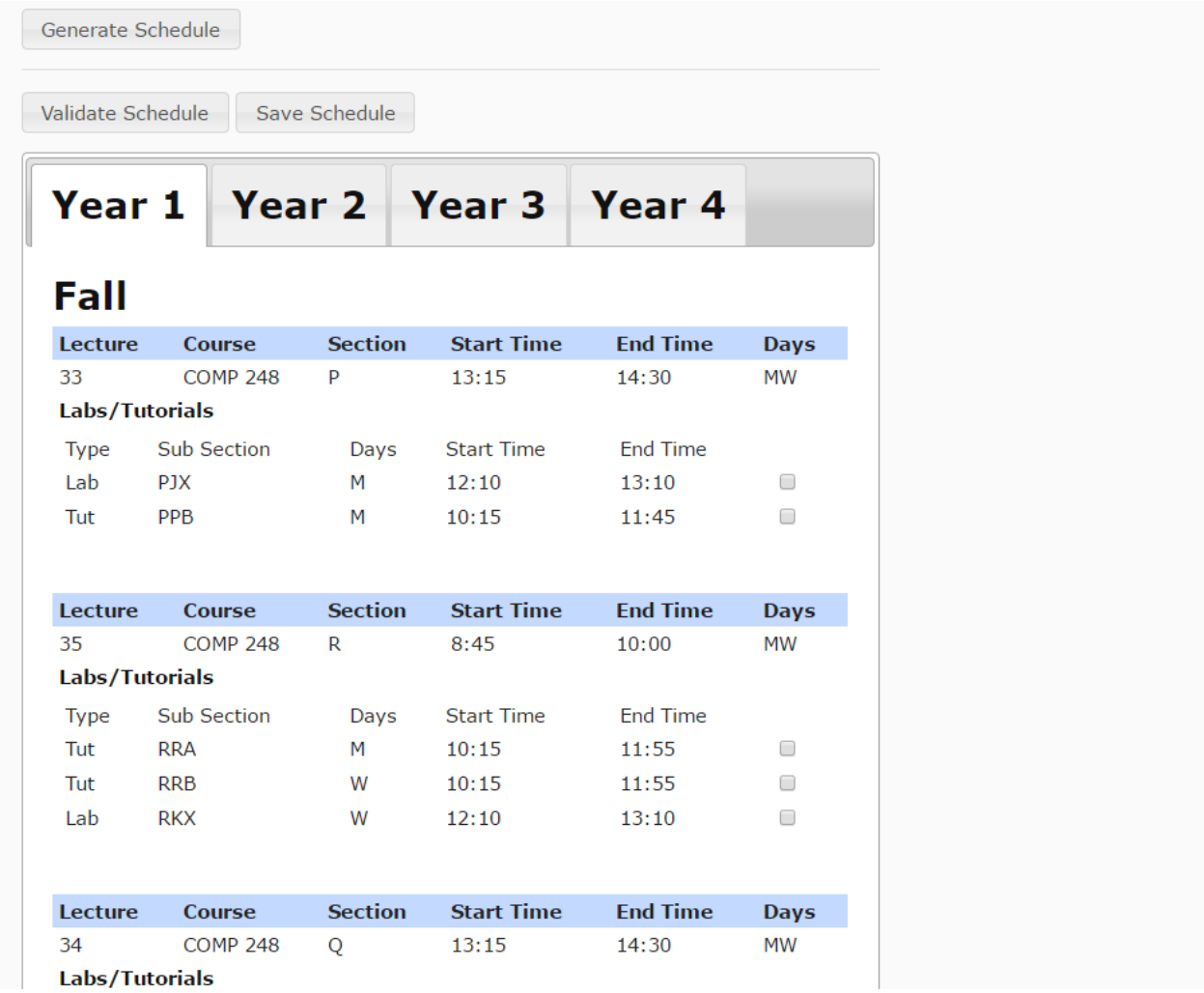


Figure 17.2.11. Tabs representing each academic year selected with courses for each semester.

All there is left to do is to choose the laboratory and tutorial times for each of the courses by selecting the checkbox corresponding to the desired sections (see Figure 17.2.12). Once sections are chosen, the student may click the 'Validate Schedule' button to ensure that the times are consistent and that no overlapping occurs. . Note that the checkboxes for tutorials and laboratories of other lecture sections of the same course have been disabled when one lecture section has been selected.

Lecture	Course	Section	Start Time	End Time	Days
33	COMP 248	P	13:15	14:30	MW

Labs/Tutorials

Type	Sub Section	Days	Start Time	End Time	
Lab	PJX	M	12:10	13:10	☑
Tut	PPB	M	10:15	11:45	☑

Lecture	Course	Section	Start Time	End Time	Days
35	COMP 248	R	8:45	10:00	MW

Labs/Tutorials

Type	Sub Section	Days	Start Time	End Time	
Tut	RRA	M	10:15	11:55	☐
Tut	RRB	W	10:15	11:55	☐
Lab	RKX	W	12:10	13:10	☐

Figure 17.2.12. Selection of the laboratory and tutorial sections

17.2.1.5 Save Generated Schedule

While on Schedule Planner tab (located at the top of the page (See figure 17.2.13)), the Student may wish to save a schedule they have generated.

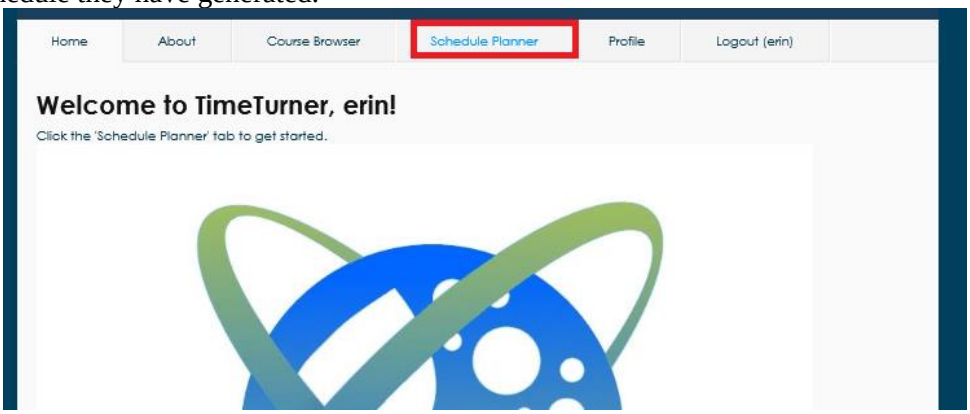


Figure 17.2.13. Schedule Planner tab

A schedule that adheres to the specifications requested in the preferences list must be generated firsts. After having generating the schedule for a specific year and selecting the section of choice, the schedule can be saved with the save schedule button (see Figure 17.2.14).

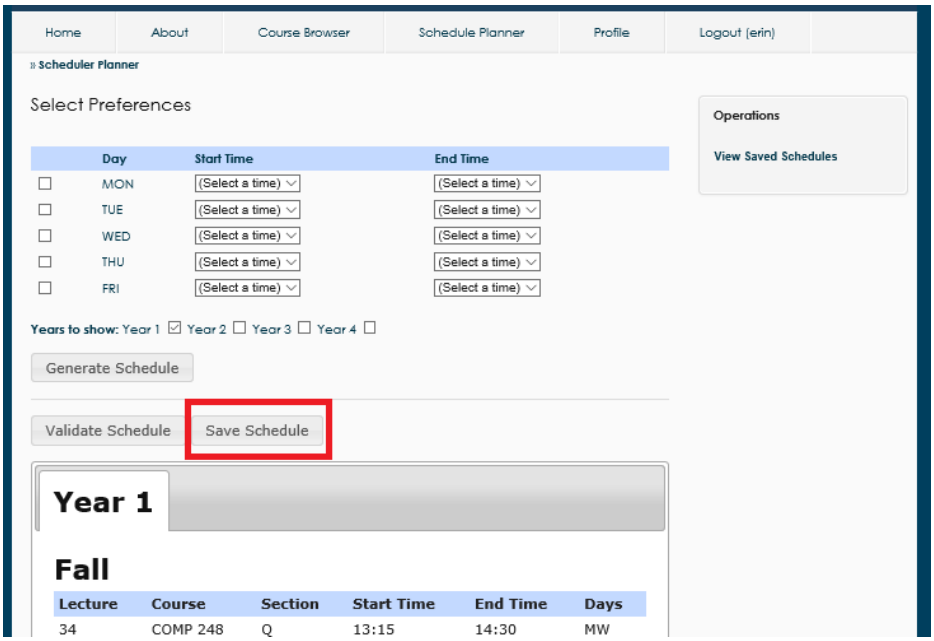


Figure 17.2.14. Save Schedule button after generating a schedule and choosing tutorial and lab sections

This will return a pop-up indicating confirmation that the schedule has been saved (see Figure 17.2.15).

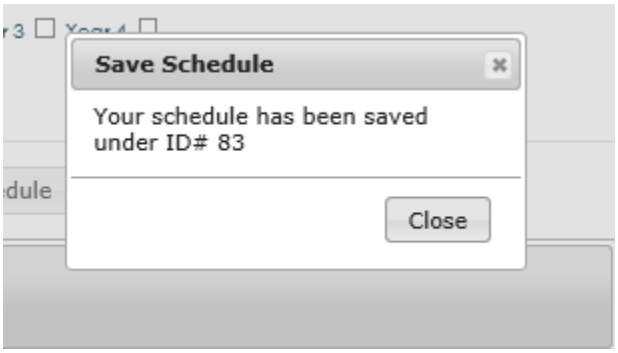


Figure 17.2.15. Confirmation of a Saved Schedule

17.2.1.6 View A Saved Schedule

Upon accessing the Scheduler Planner tab, generated a schedule and saved a schedule, the student may wish to view any schedules they have saved. In the operations box on the sidebar, there is a view saved schedules option (see Figure 17.2.16).

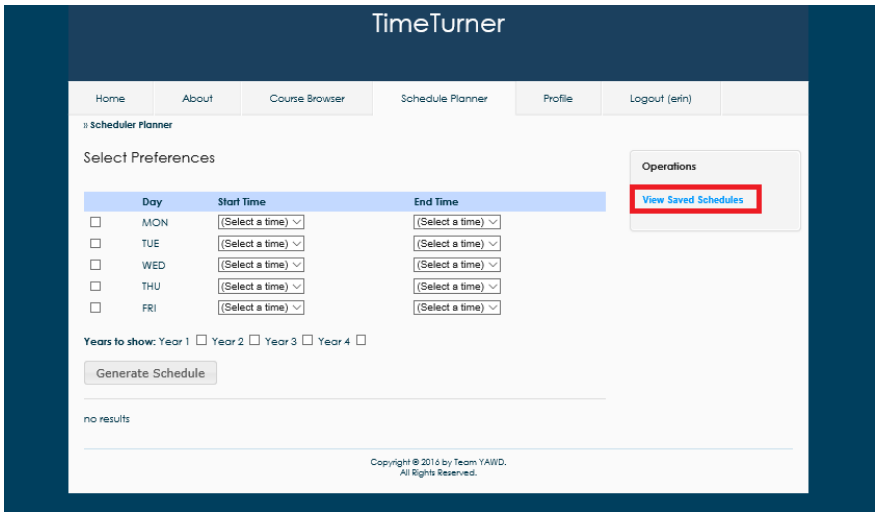


Figure 17.2.16. View Saved Schedule option

Upon clicking this option, a list of all saved schedules will be presented to the student. At this point they can view or delete a schedule (see Figure 17.2.17).

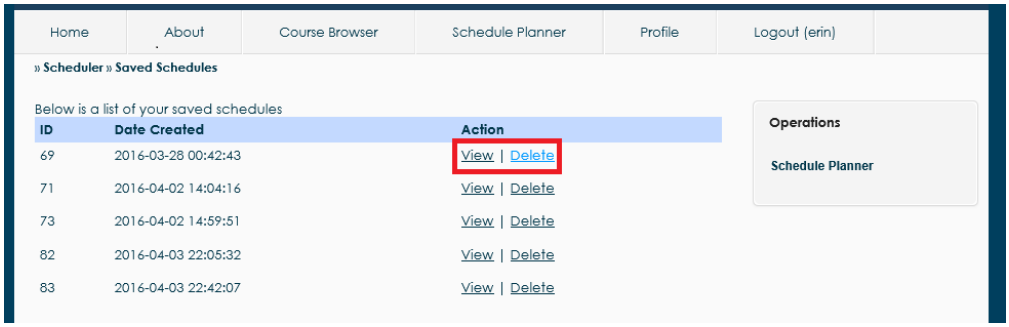


Figure 17.2.17. Options to view or delete a schedule

By clicking view, it returns the schedule for the given semester. By deleting it, the system asks for confirmation of deletion and then removes the schedule from the list of saved schedules (see Figure 17.2.18).

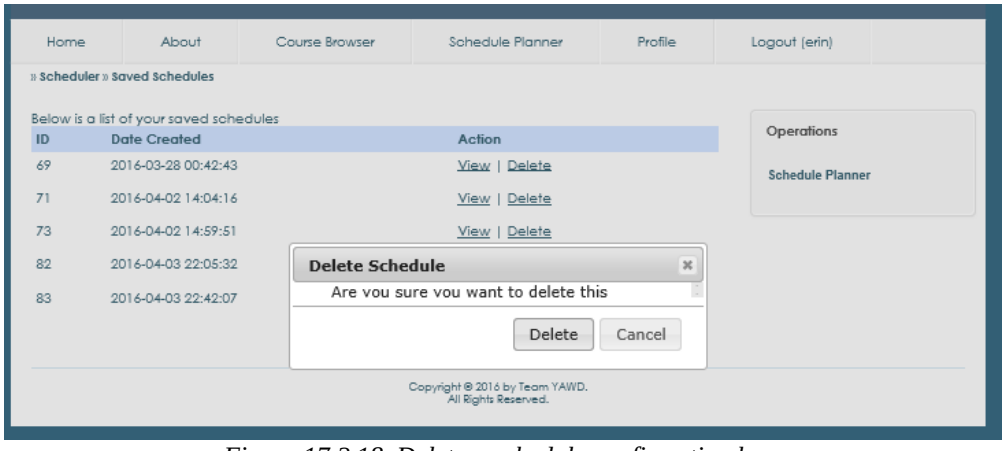


Figure 17.2.18. Delete a schedule confirmation box.

17.2.1.7 Drop Course

A student that is logged in the system and has at least one saved schedule containing at least one course has the option to drop a course. They can do so by clicking on the "View Saved Schedules" option in the Schedule Planner tab. The option appears on the right hand side of the Schedule Planner page (see Figure 17.2.19).

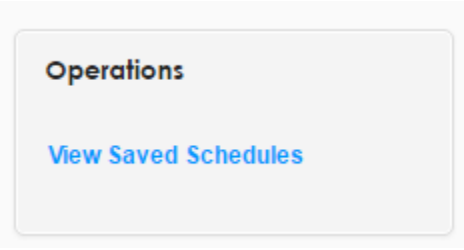


Figure 17.2.19. View Saved Schedules operation

A page displaying all currently saved schedules for the student is displayed. To choose the schedule from which a course will be dropped, the "View" option should be selected. It appears to the right of each saved schedule (See Figure 17.2.20).

Figure 17.2.20. View Saved Schedule action

The selected saved schedule will load, displaying all courses that are currently in that schedule. Each course has a "Drop" option to the right of the number of credits (see Figure 17.2.21).

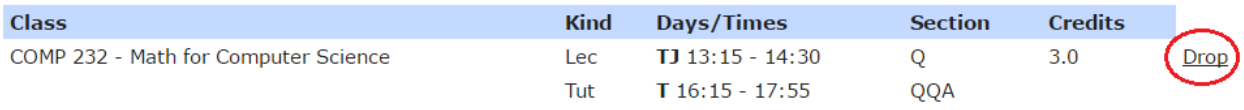


Figure 17.2.21. Drop Course action

A dialog box prompting the user to confirm their choice will appear after clicking the Drop button. To successfully drop the selected course, the "Delete" button on the left will confirm the deletion of that course from the selected saved schedule (see Figure 17.2.22). If the student changes their mind, the cancel button will cancel the operation and the course is not dropped from the schedule.

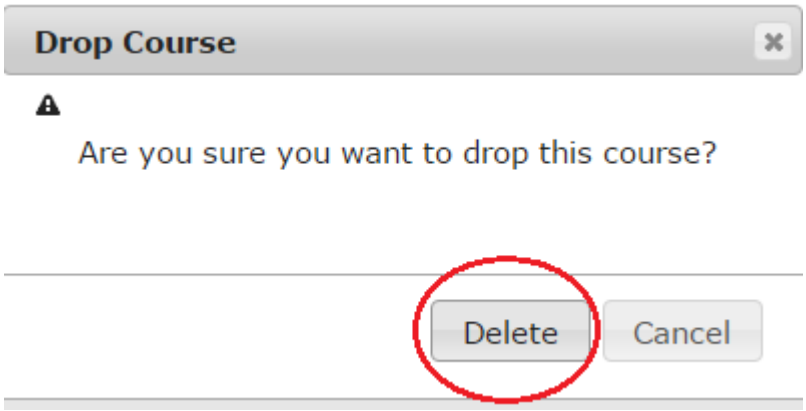


Figure 17.2.22. Confirmation dialog box

17.2.1.8 View Academic Record

A student wishing to view their academic record and/or GPA may do so from the 'Profile' tab (See Figure 17.2.23).

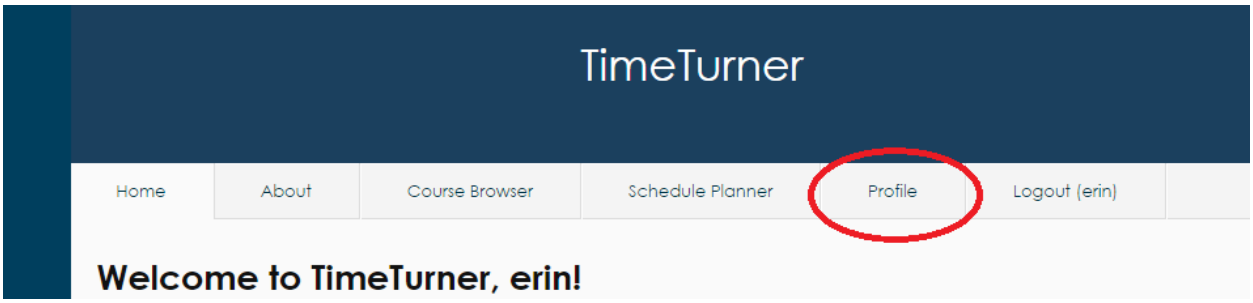


Figure 17.2.23. Profile Tab located second from right at the top of the screen

Once on the 'Profile' page, the student may click on the 'View Academic Record' displayed at the far right of the page (see Figure 17.2.24). Clicking this link will result in the display of all classes that the student has completed, as well as their grades for each course, and their GPA.

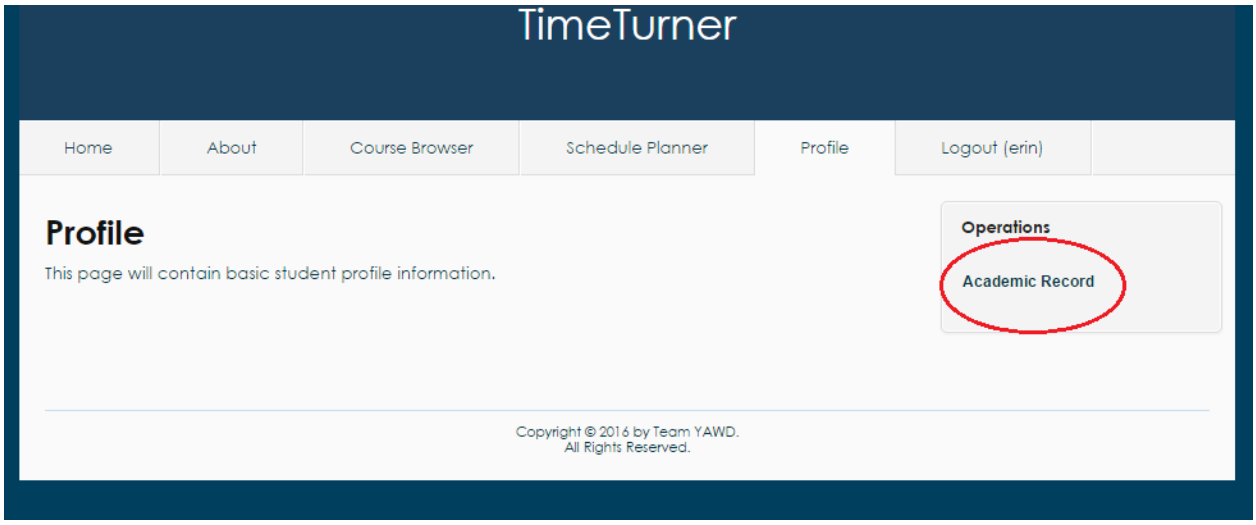


Figure 17.2.24. Academic Record link in 'Operations' on far right of 'Profile' screen

17.2.2 Administrator Users

17.2.2.1 Login

Administrators are simply privileged users. As such, logging in as an Administrator is performed much as logging in as a student. See section 17.2.1.1.

17.2.2.2 Logout

Administrators are simply privileged users. As such, logging in as an Administrator is performed much as logging in as a student. See section 17.2.1.2.

17.2.2.3 Manage Courses (Administrator)

An administrator that is logged in the system will have the option to manage courses. This option is found in the Courses tab, on the right hand side of the page (see Figure 17.2.25).

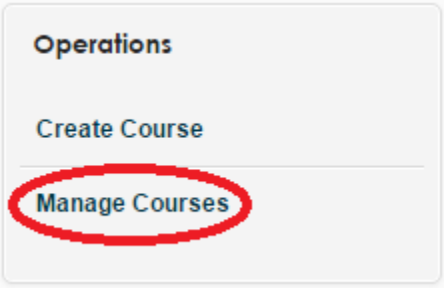


Figure 17.2.25. Manage Courses operation

Clicking the manage courses option will display the manage courses page. There are many possible actions for the administrator to take from here.

17.2.2.4 Search User

If the Administrator wishes to manage a specific course, the search bars above the table containing all the courses may be used to search for a course (see Figure 17.2.26), or the "Advanced Search" setting may be used (see Figure 17.2.27).

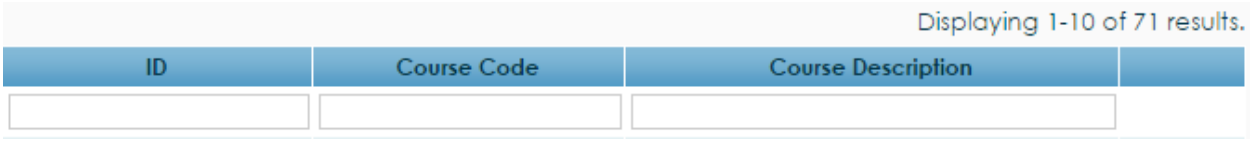


Figure 17.2.24. Course search bars

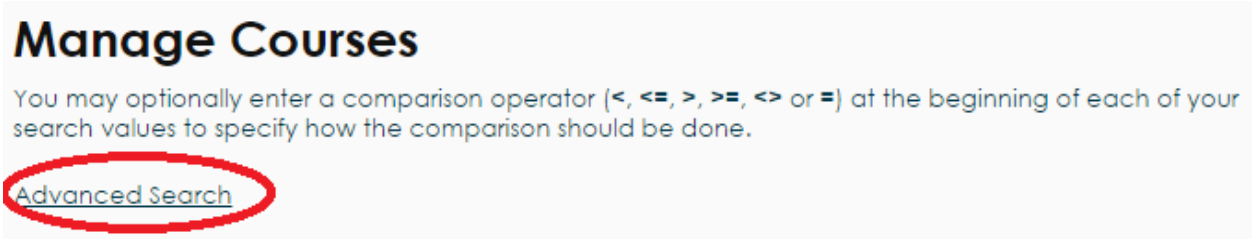


Figure 17.2.27. Advanced search setting

The advanced search setting will display two search bars on the page which will accept a course I.D. and/or course code to search the system (see Figure 17.2.28). Clicking the search button will perform the search.

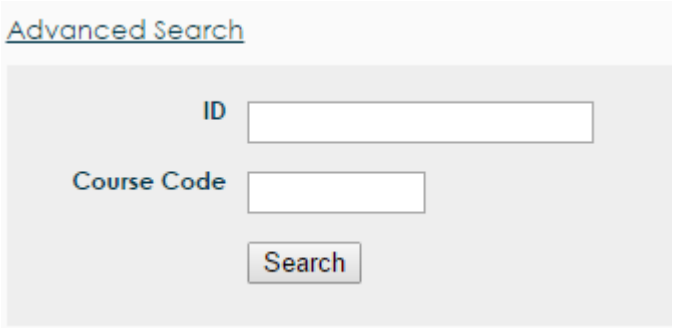


Figure 17.2.28. Advanced search bars

Once a course is searched and found, the administrator has the options of viewing, editing or deleting the course. These options can be seen on Figure 17.2.29, in the respective order.

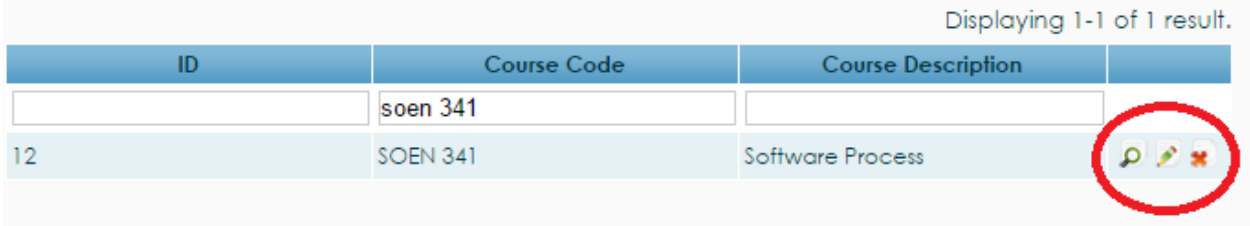


Figure 17.2.29. View, edit, and delete buttons

17.2.2.5 View Course

The view option will produce a page that displays the details of the course, including course I.D., course code, and course description (see Figure 17.2.30)

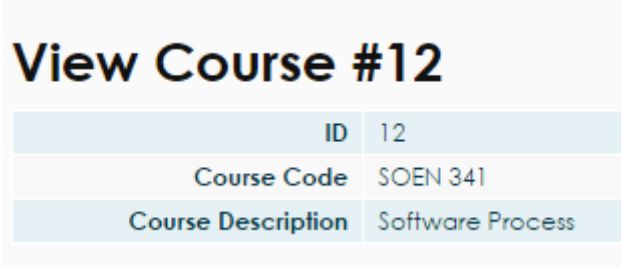


Figure 17.2.30. View course table

17.2.2.6 Edit Course

The edit function is selected by clicking the pencil icon to the right of the view button. Clicking the edit button will produce the update course page, which can edit the course code, description, number of credits, and type of the course (see Figure 17.2.31). Any changes made will be finalized by clicking the "Save" button below the editing boxes.

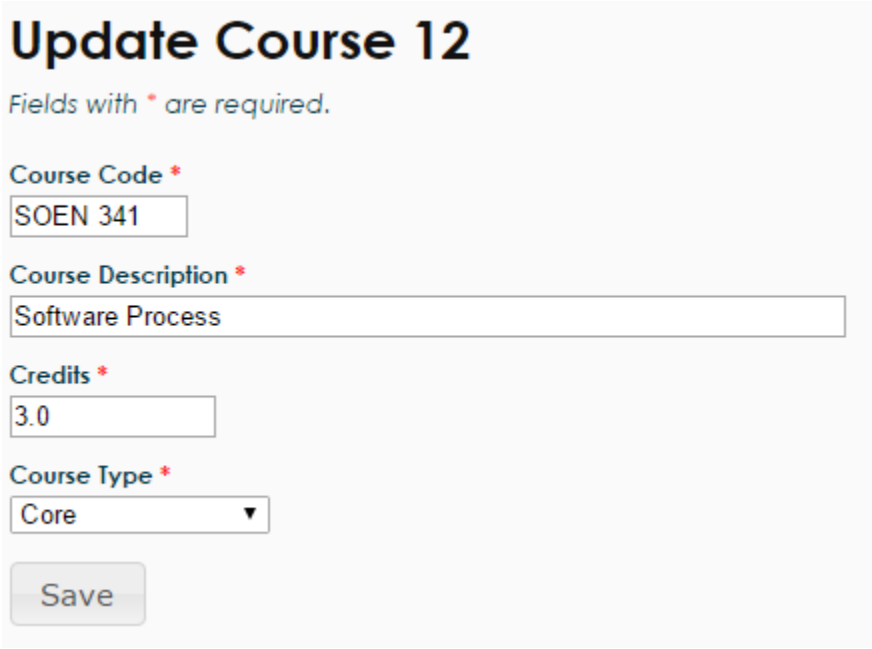


Figure 17.2.31. Update course boxes

17.2.2.7 Delete Course

The final operation that an administrator can do to manage a course is to delete a course, which can be performed by clicking the red "X" button to the right of the update course icon. Clicking the X icon will produce a dialog box that prompts the administrator to confirm their decision. To confirm their choice, the "OK" button should be pressed and the course will be deleted from the course bank (see Figure 17.2.32). If the administrator does not want to delete the course, they can simply press the cancel button or exit the dialog box.

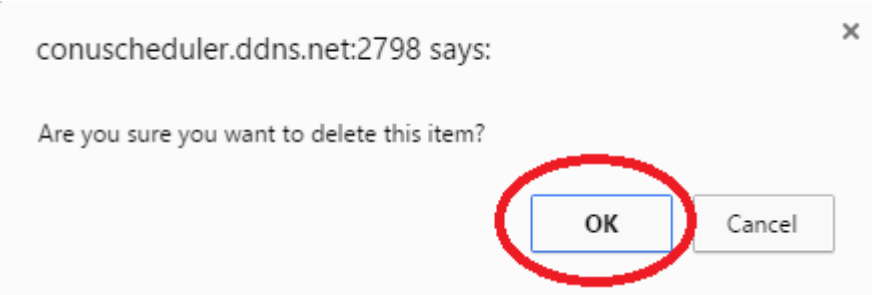


Figure 17.2.32. Confirmation dialog box for course deletion

17.2.2.8 Create Course

An administrator may wish to create a course to add to the database of courses that students can take. In order to do so, the administrator must first click on the 'Courses' tab to find themselves on the courses database page (see Figure 17.2.33). From here, the administrator must click the 'Create Course' link located on the far right of the page; this action will display the course creation page (see Figure 17.2.34).

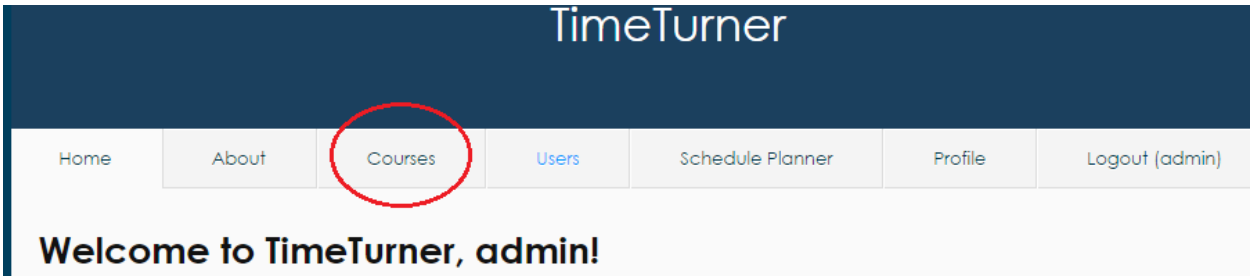


Figure 17.2.33. Courses tab on top of page when logged in as Administrator

Figure 17.2.34. Create course link on far right of Course Browser page

This page will ask for the course code, the course description, and the amount of credits awarded by the course (see Figure 17.2.35 for example input). Once the administrator enters the appropriate course information, the system will log such information in the database, and the course information for the newly created course will be displayed on screen, along with the course's newly assigned ID.

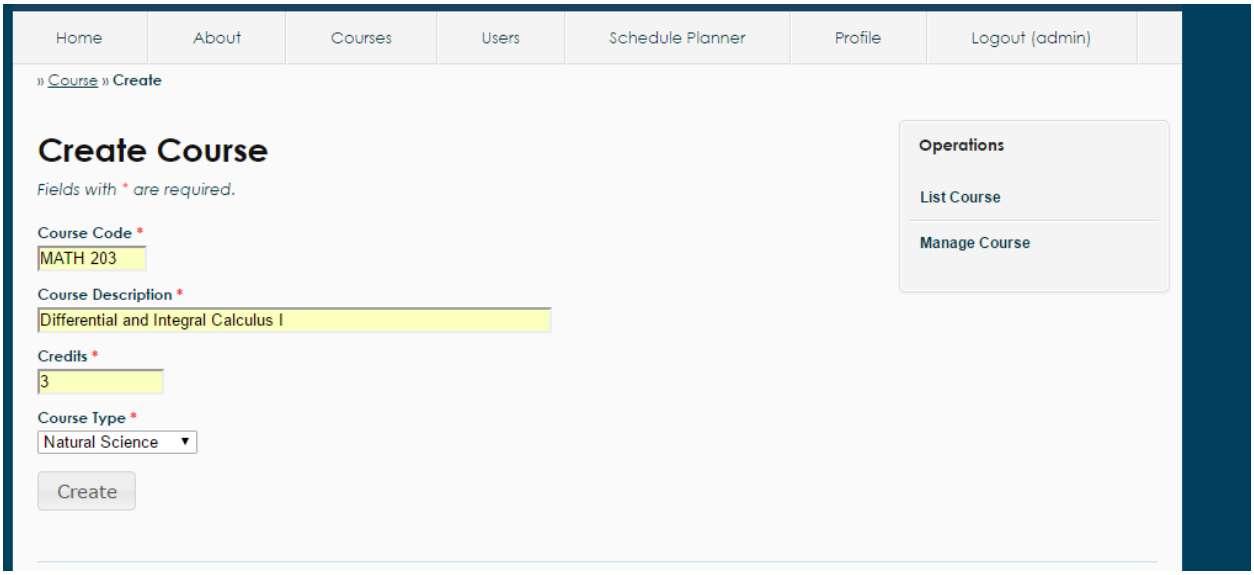


Figure 17.3.35. Sample input for Create Course functionality

17.2.2.9 Browse Courses

While logged in, the administrator may wish to view the database of courses in the system. To accomplish this, the administrator may simply click on the Courses tab; this will cause the list of all courses to be displayed on screen (see Figure 17.2.33). Courses are grouped by page, 10 to a page. If the administrator wishes to view more courses, he may scroll down to the bottom of the page and click on the desired page number.

While on the Courses tab, the administrator may navigate away by clicking the two links located in the 'Operations' box to the right of the page: Create Course and Manage Course. Clicking these links will bring the administrator to those respective pages where other activities and actions may be carried out.

If the administrator wishes to view the prerequisite information of a course, he may simply click on the name of the prerequisite to that course (see Figure 17.2.36). Clicking on the name of the prerequisite will display the course information of the prerequisite.

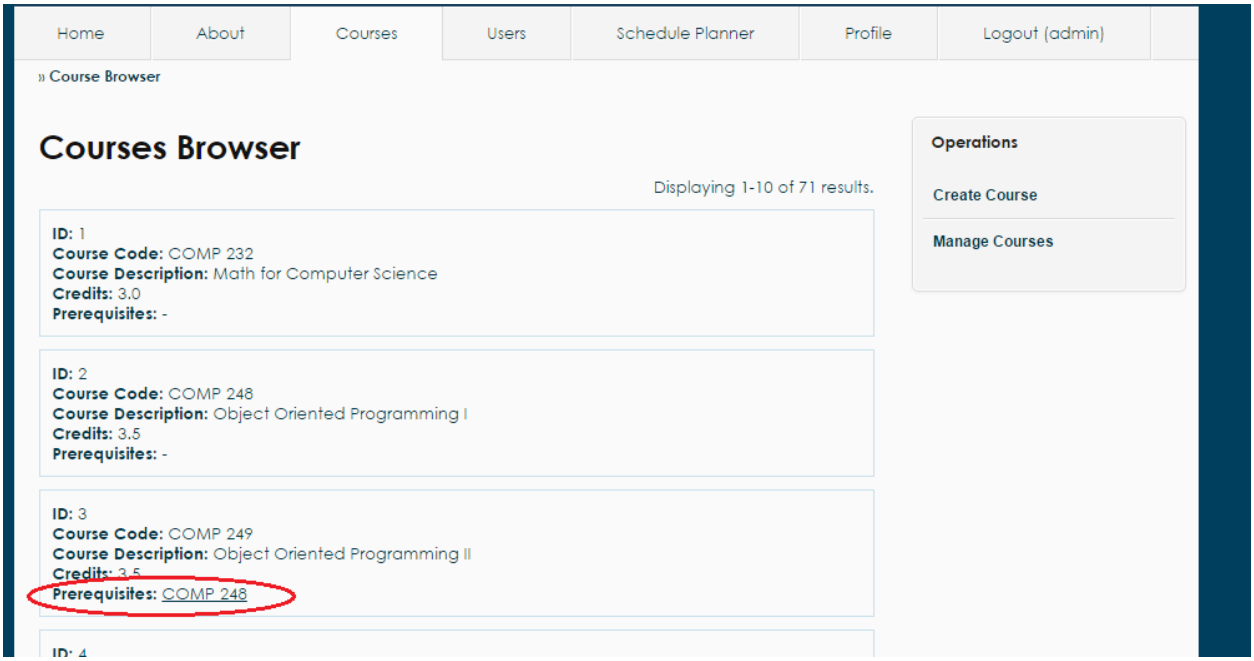


Figure 17.2.36. Prerequisites for each Course located in the course information box

17.2.2.10 Add User

Once logged in, the administrator can add a new user into the system by first clicking on the 'Users' tab.

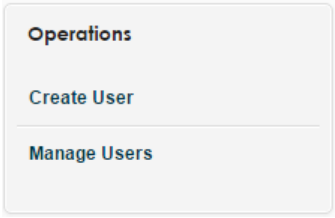


Figure 17.2.37. Create user operation

A new page will be loaded, displaying a list of users. On the right side menu, clicking on the 'Create User' link (see Figure 17.2.37) will redirect the administrator to another page where information about the new user can be entered. (see Figure 17.2.38).

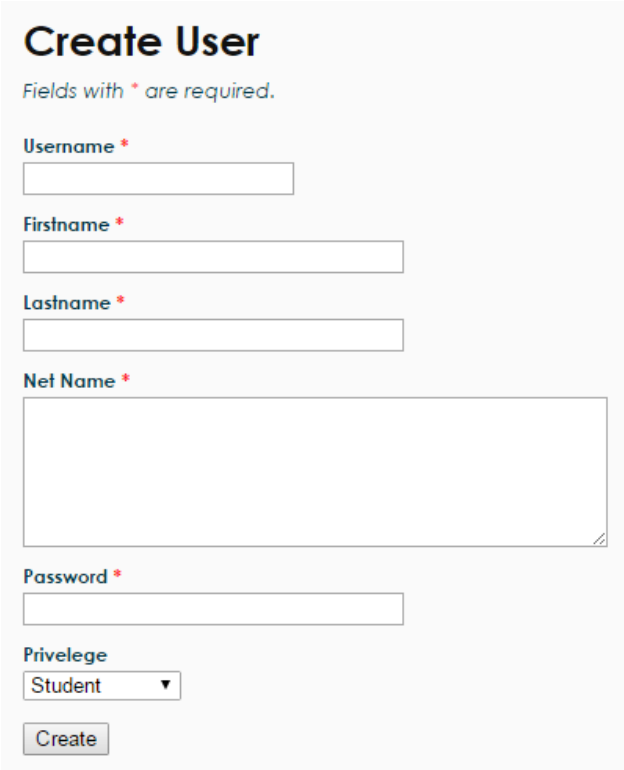


Figure 17.2.38. Create user form

All fields of the form must be completed to be valid. The password must be at least 6 characters long and this will be the password used to log into the system for the user, along with the username. The privilege options allow the administrator to select whether the new user is a student or an administrator.

Upon completing all fields of the form, the administrator may click on the 'Create' button at the bottom to submit it. If the user was successfully created, the newly created user's information will be displayed on another page to the administrator, along with an ID number (see Figure 17.2.39).

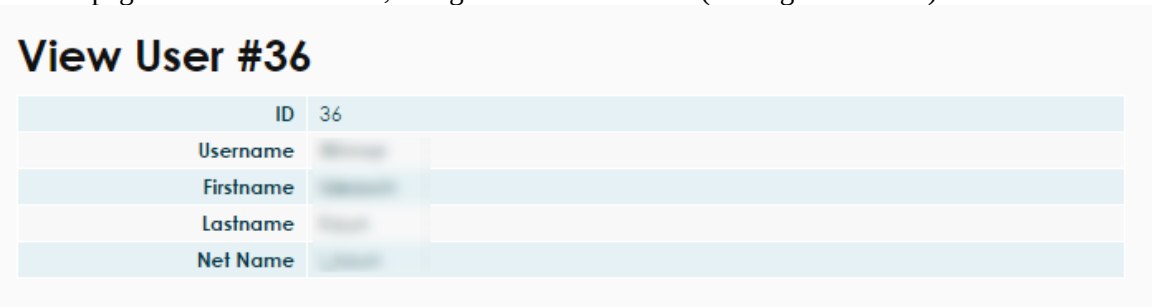


Figure 17.2.39. View the newly created user's information

17.2.2.11 Manage Users

When logged in as an administrator, users of the system can be managed. More precisely, search can be conducted on the system in order to find one or more specific users; information about those users can be viewed and updated, and users can be deleted through this managing section.

In order to manage users, the administrator must first go to the 'Users' tab from the home page (see Figure 17.2.40).

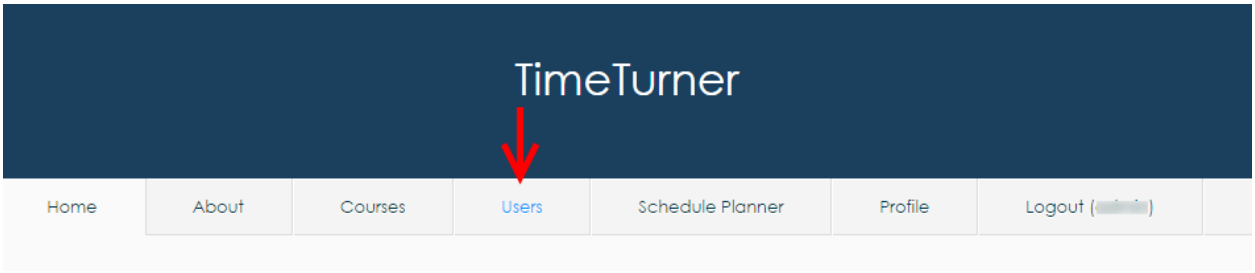


Figure 4.2.40. 'Users' tab from the main menu when logged in as an Administrator

The administrator will then see a list of users on the system, along with a list of operations that the administrator is allowed to perform. To access the user management page, the administrator must click on the 'Manage Users' operation (see Figure 17.2.41).

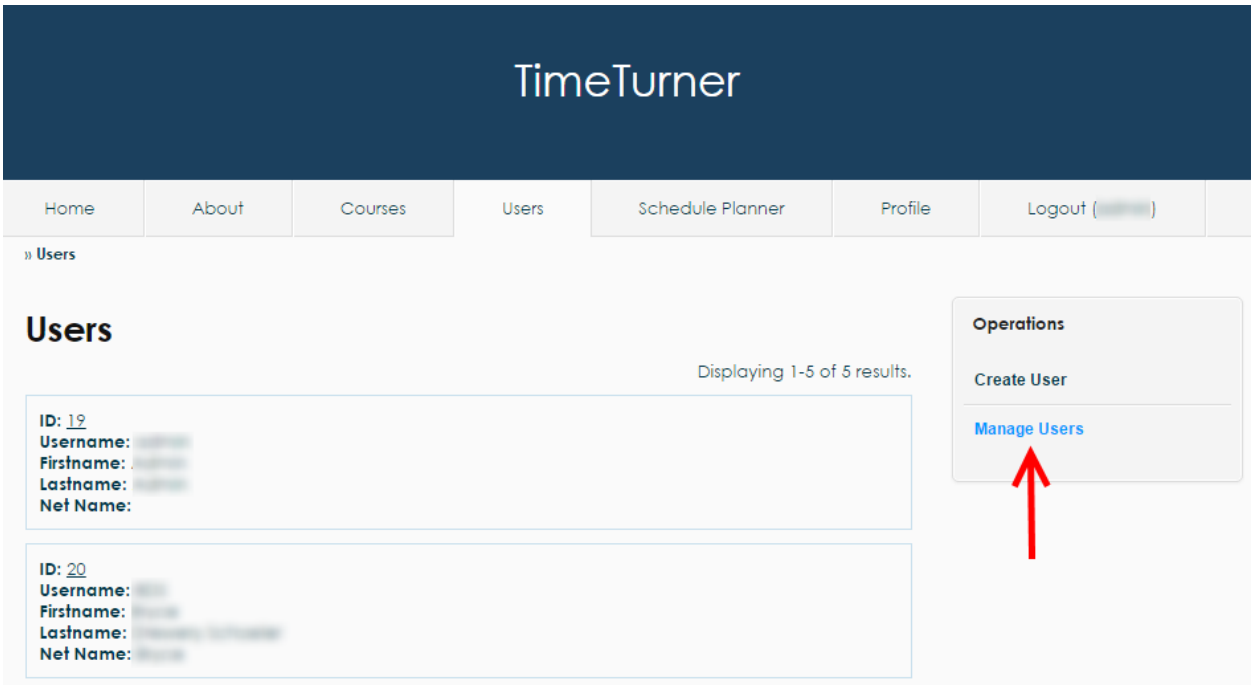


Figure 17.2.41. 'Manage Users' operation found on the 'Users' tab

The administrator should then land on the user management page where multiple search operations are presented with a default search result containing a list of editable users (see Figure 17.2.42).

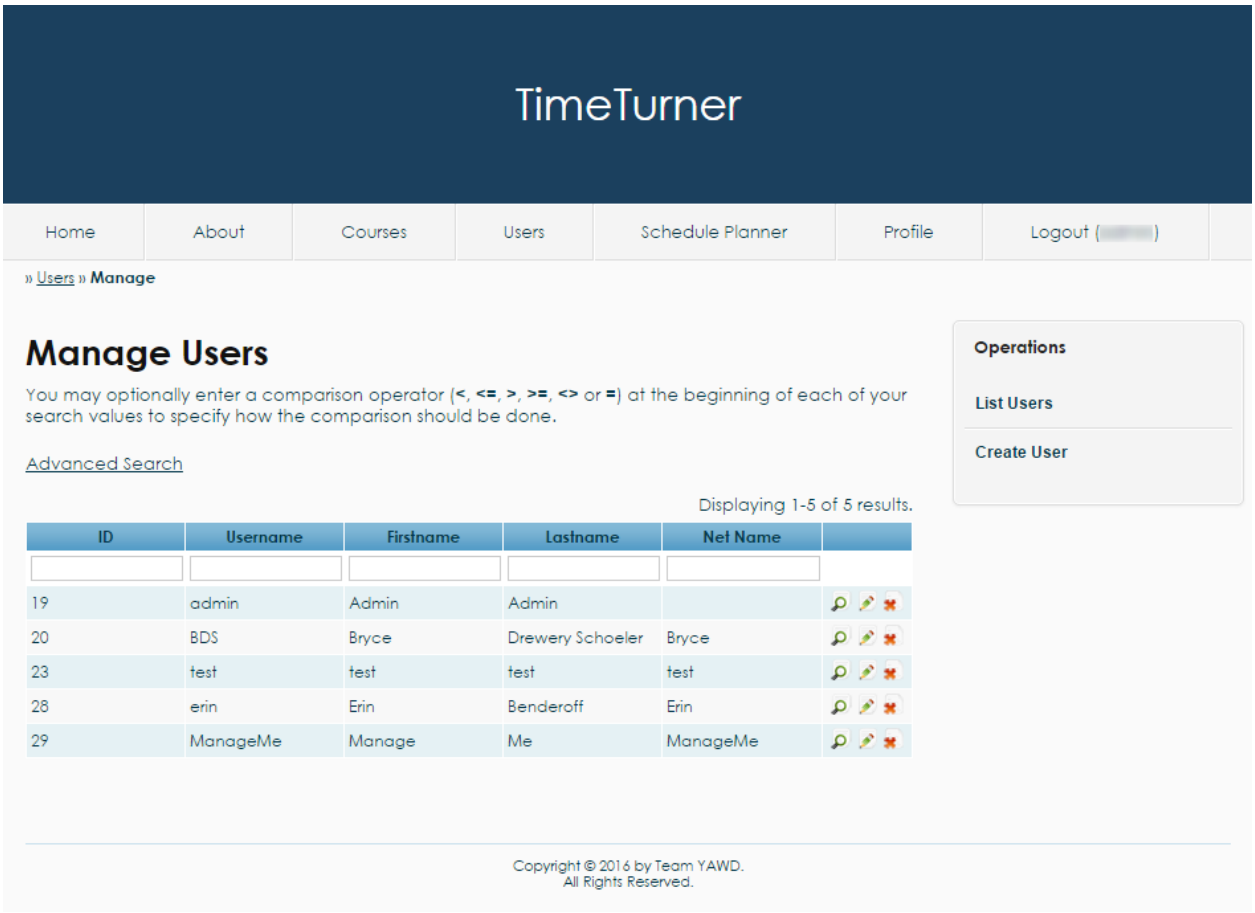


Figure 17.2.42. User management page with a default table of users

The following next operations will be performed from the user management page.

17.2.2.12 Search for Users

From the user management page, users can be found using the textboxes located on the second row of the table of users. For instance, if an administrator is looking for the user whose ID is 29, the ID number would be typed on the textbox in the 'ID' column (see Figure 17.2.43).

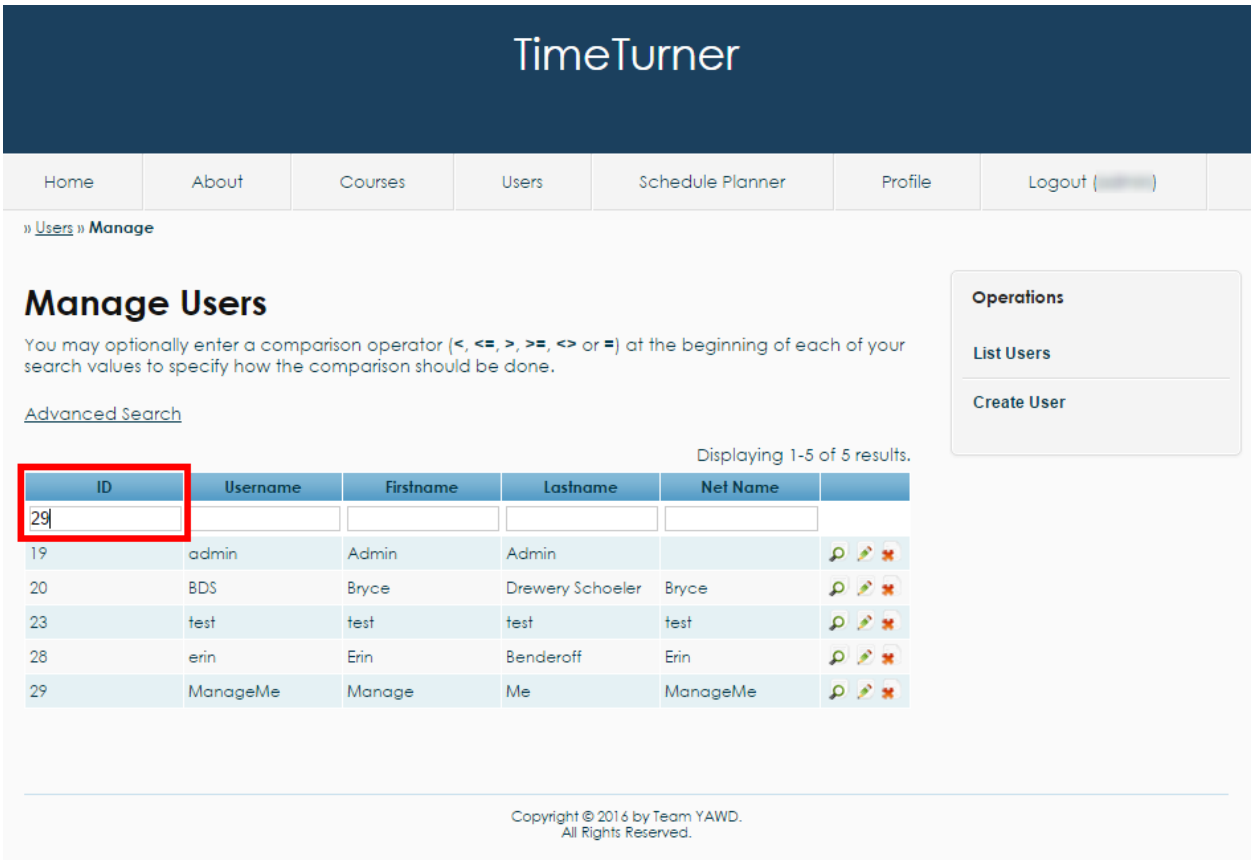


Figure 17.2.43 Entering the desired search criteria in the corresponding textbox

Once the field is completed, the administrator can press 'enter' or simply click outside of the textbox to refresh the search table. As a result, the user with the matching ID will be displayed on the table (see Figure 17.2.44). If no users have been found to match the search criteria, the table of users will be replaced with a message indicating that no results have been found.

Note that many criteria can be entered in the various textboxes. In such case, the table will reload every time a textbox is out of focus or the key 'enter' is pressed. Also, comparison operators can be used in those textboxes, for example to filter all users having an ID less than 30. Another feature is that all search fields except for ID based on a 'contains' relationship, so for example, if we are looking for all usernames containing the letter 't', 't' could be written in the username search field. The search is also case insensitive.

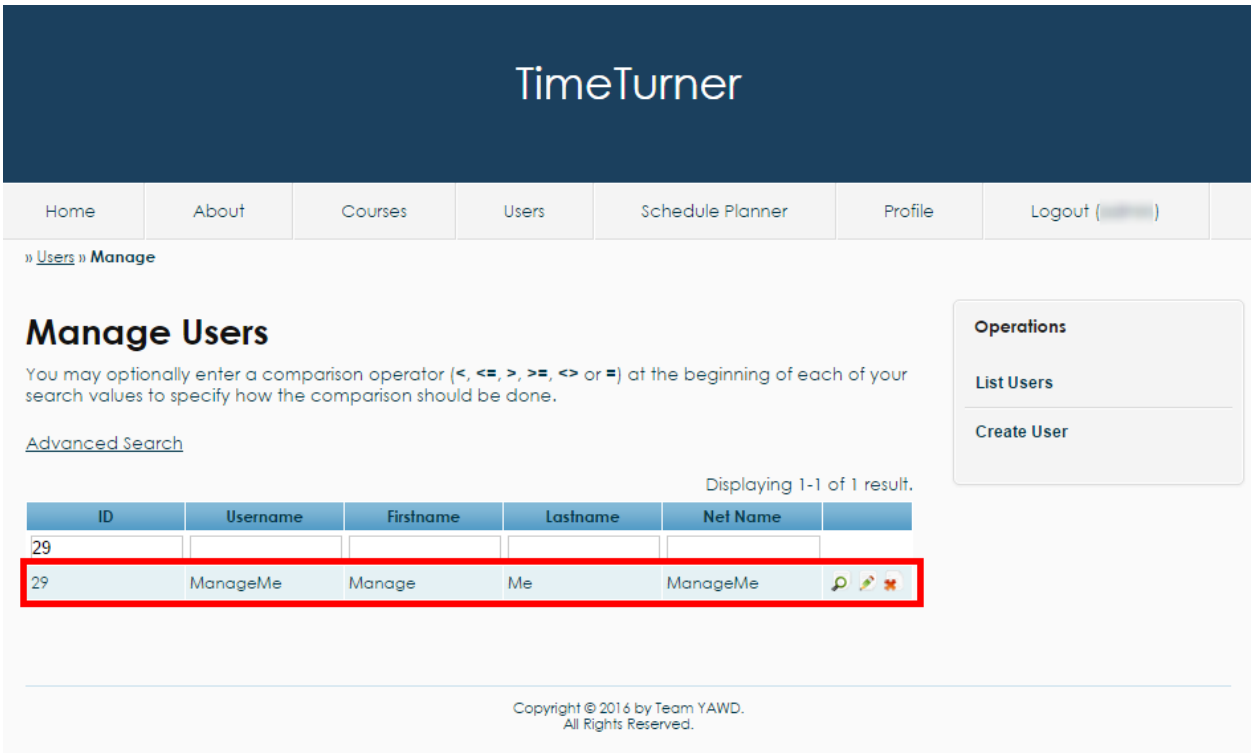


Figure 17.2.44 Result of the simple search by ID

17.2.2.13 Advanced Search for Users

An administrator may decide to perform an advanced search if many criteria are to be used in the search, and the administrator does not wish the table to constantly refresh. To do so, the administrator may click on the 'Advanced Search' link (see Figure 17.2.45).

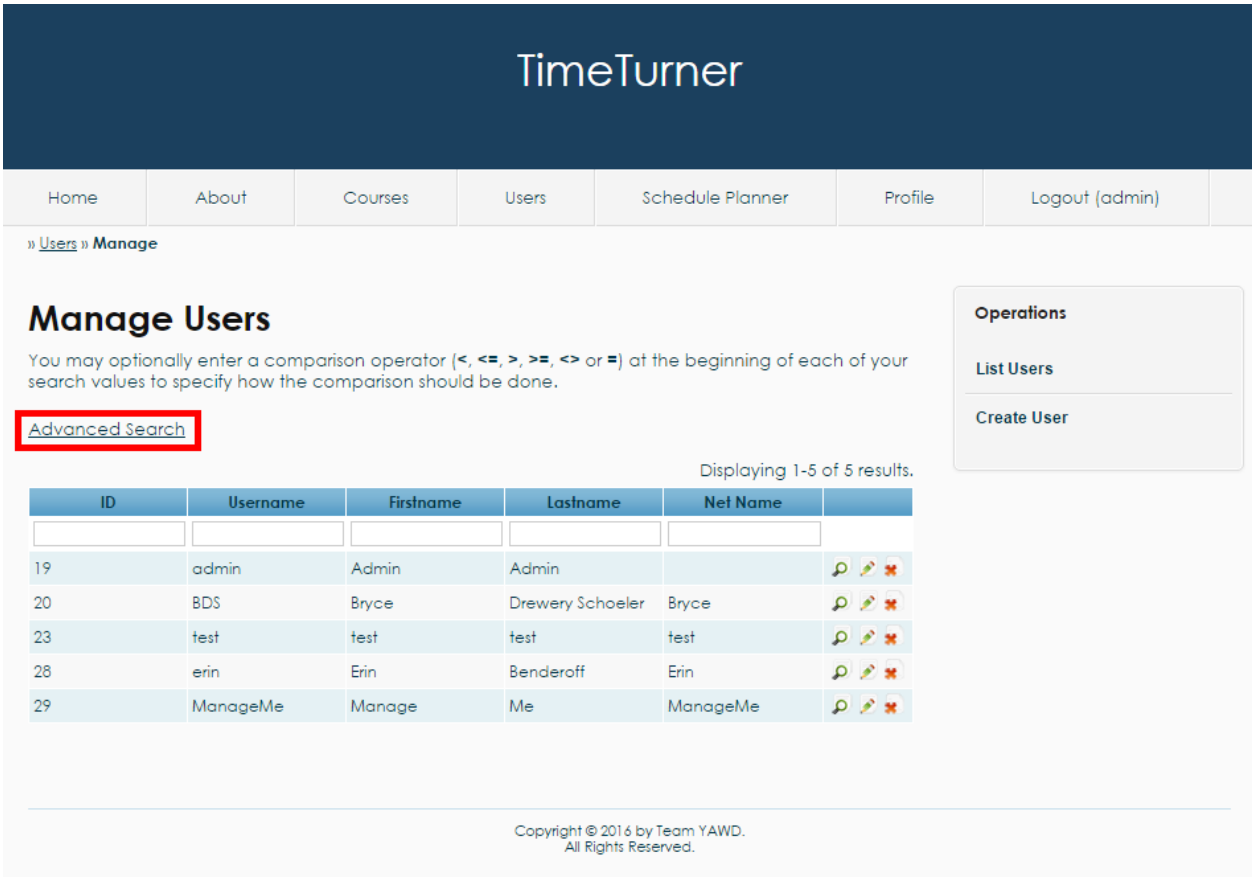


Figure 17.2.45. The 'Advanced Search' link

An advanced search form should appear once the link has been clicked. The form allows the administrator to look for a user using a combination of multiple criteria by filling up the fields that the administrator wishes to use (see Figure 17.2.46).

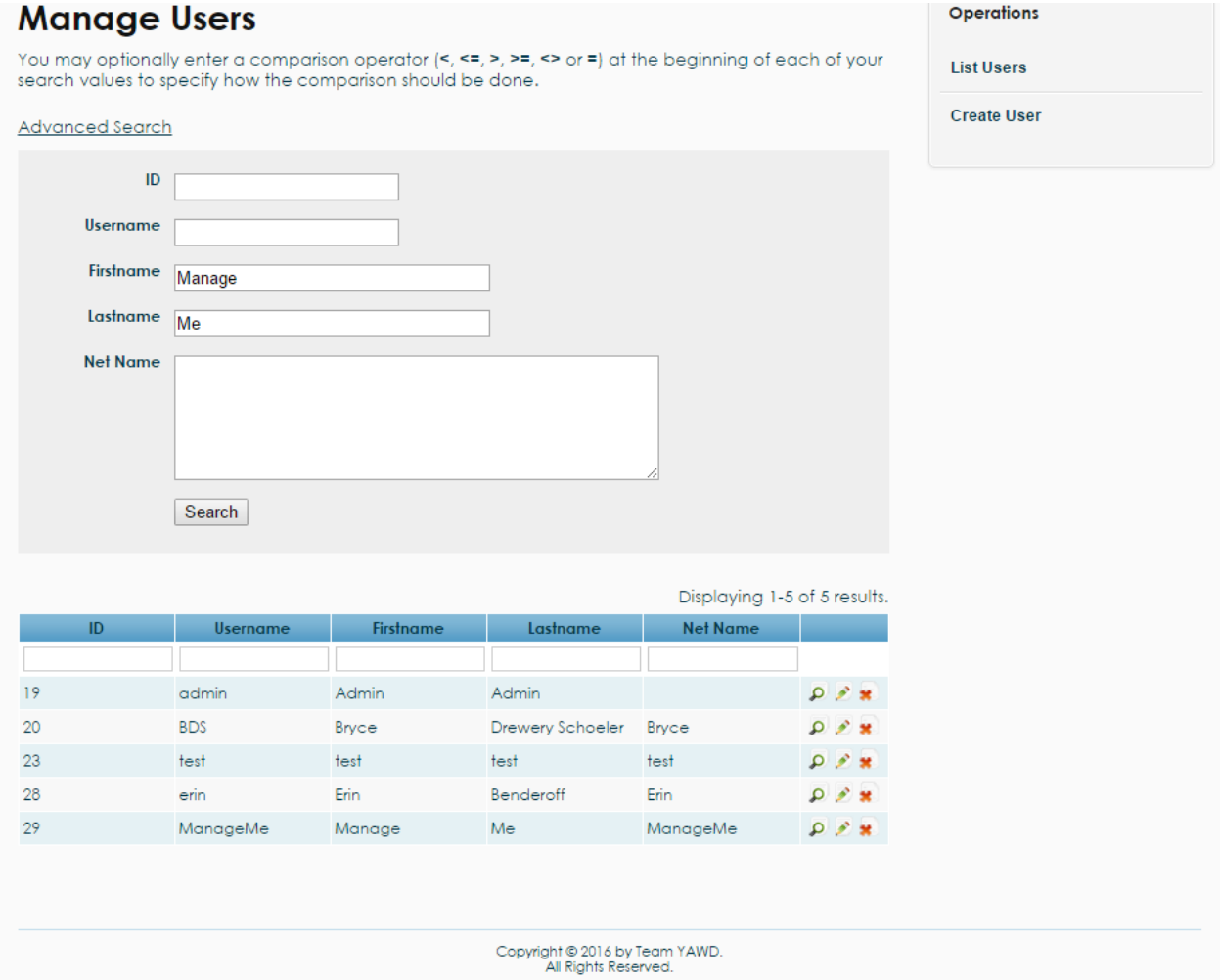


Figure 17.2.46 Usage of the advanced search fields

Once all the criteria have been entered, the administrator can hit the 'search' button at the bottom of the advanced search form. A table should be loaded with the user(s) that match the criteria (see Figure 17.2.47). Notice also that the search criteria have been copied into the textboxes for the simple search for clarity.

Like the simple search, the advanced search textboxes support comparison operators and will also display a message that no results have been found if the system fails to find a user based on the specified criteria. The searches are also based on a 'contains' relationship, meaning that all users which search field(s) contain the search criteria will be displayed. The text entered in the search fields are case insensitive.

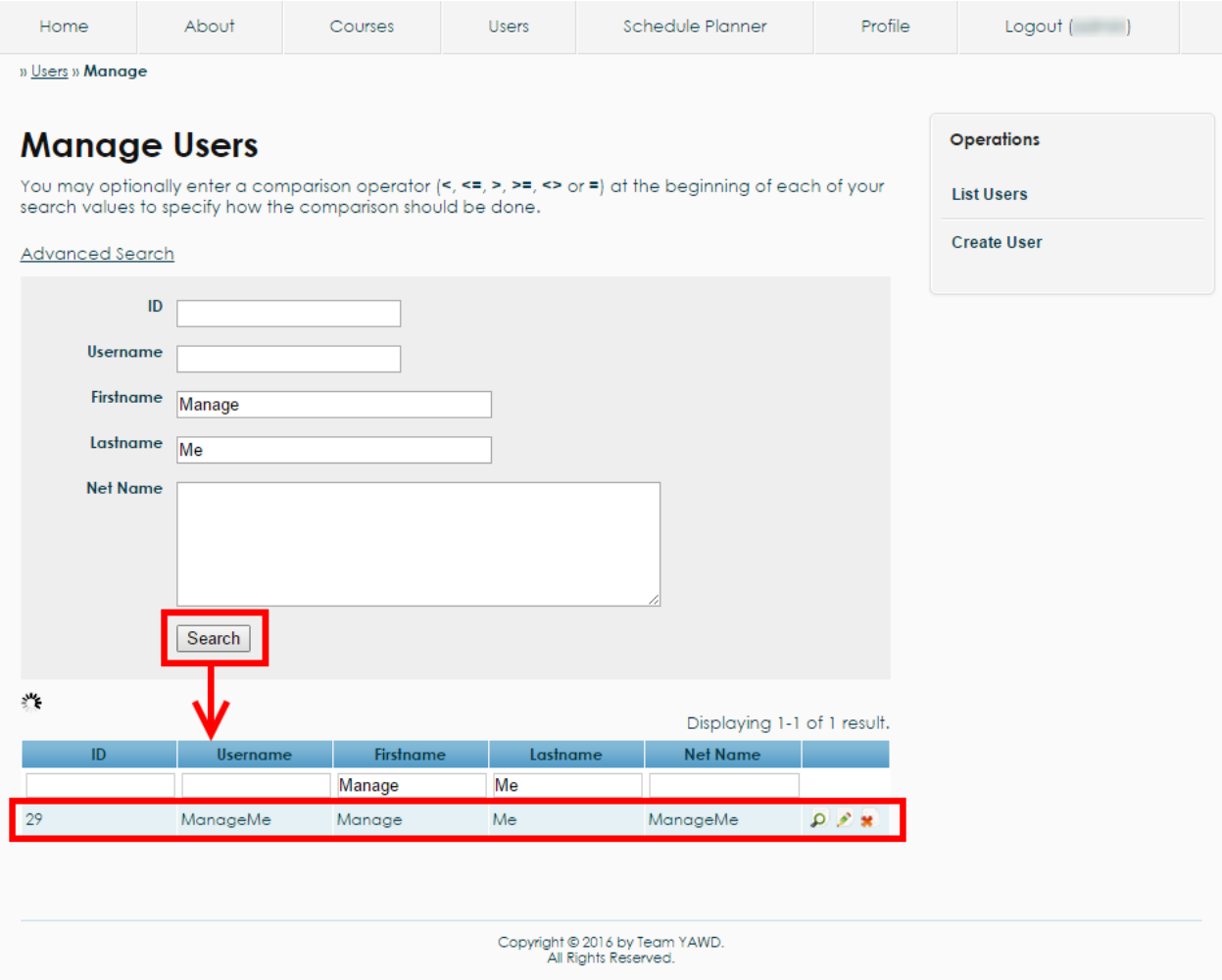


Figure 17.2.47. Post-advanced search results

17.2.2.14 View User

Whenever a table of users is displayed, the administrator may decide to view the profile of a user. To consult the profile of a user, the administrator may click on the magnifying glass found on the right of row corresponding to the user (see Figure 17.2.48).

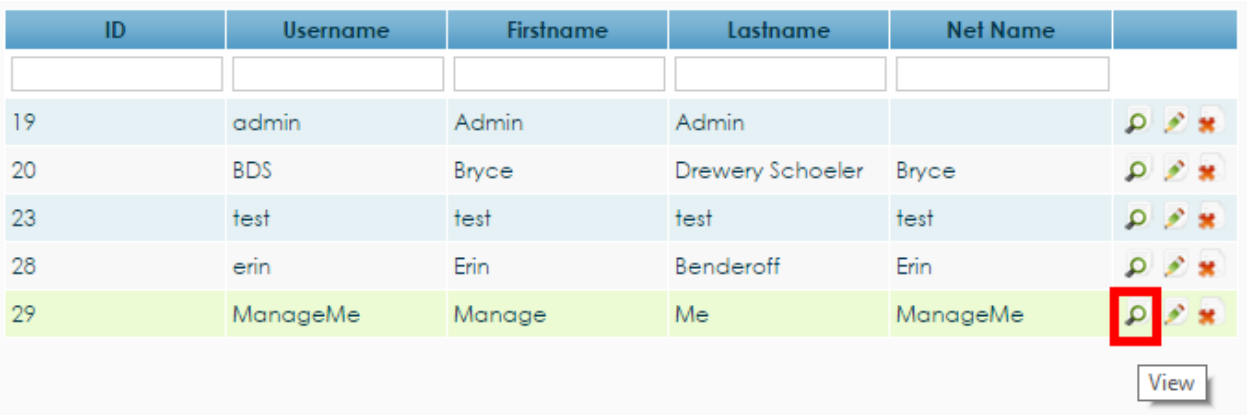


Figure 17.2.48. The view magnifying glass corresponding to the user 29 allowing administrators to consult the profile of this user

Once the magnifying glass has been clicked, the profile of the user should be displayed among with a list of available operations that will be discussed in the following sections (see Figure 17.2.49).

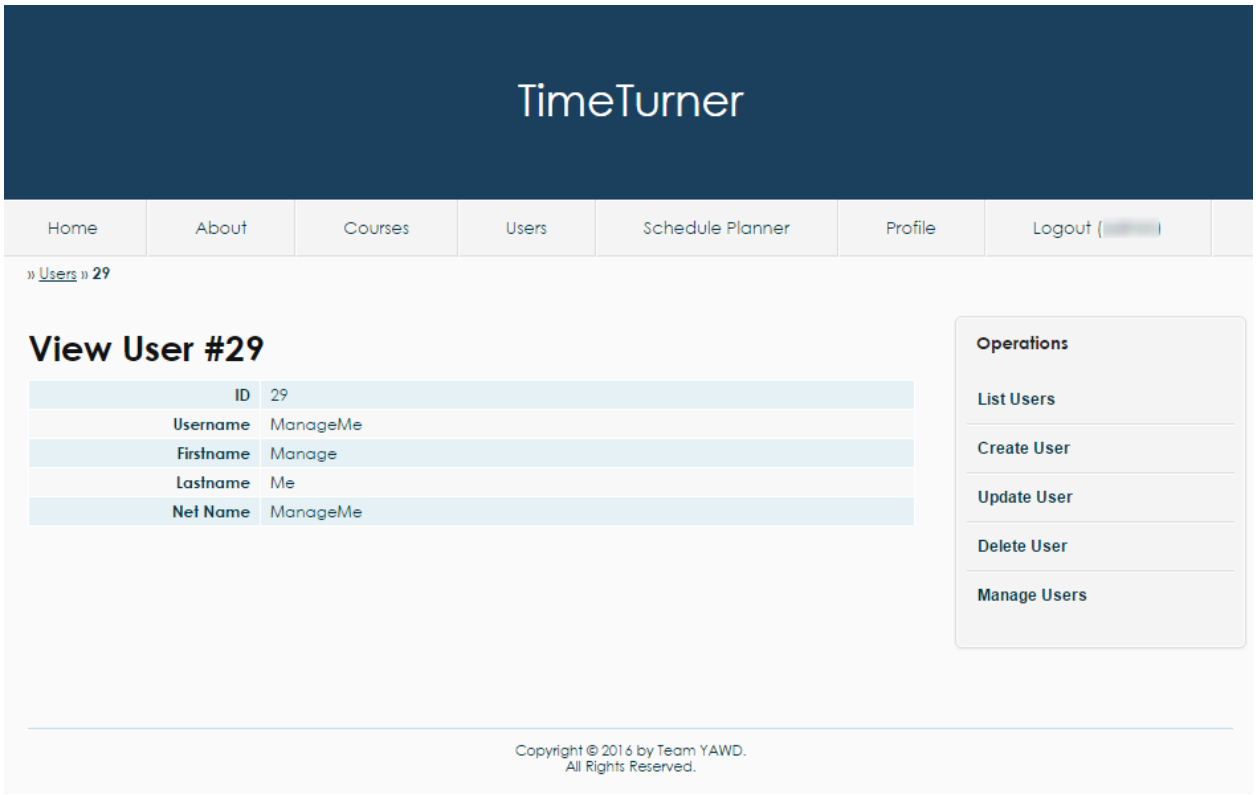


Figure 17.2.49 The profile of a user with a set of operations on the right

17.2.2.15 Update User

Users' information can be updated through the user management page. More precisely, updating users' information can be either be done through the table of users, or through consulting the profile of a user.

To update a user's information through the table of users, the pencil button found on the right of the row corresponding to the user to be updated should be clicked (see Figure 17.2.50).

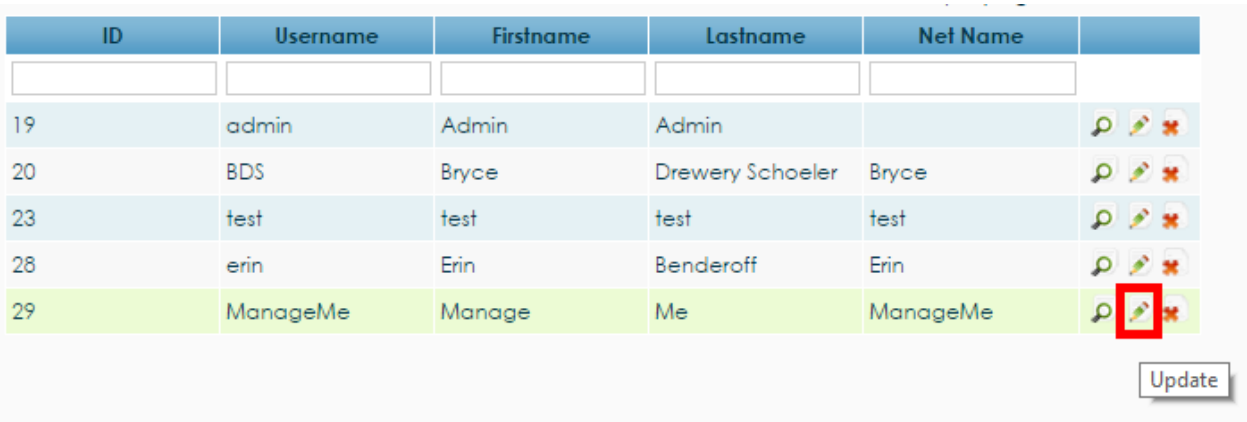


Figure 17.2.50 Pencil button to update the user 29

To update a user's information through the user's profile view, the administrator needs to choose the 'Update User' operation from the list of operations found on the left of the page (see Figure 4.2.51).

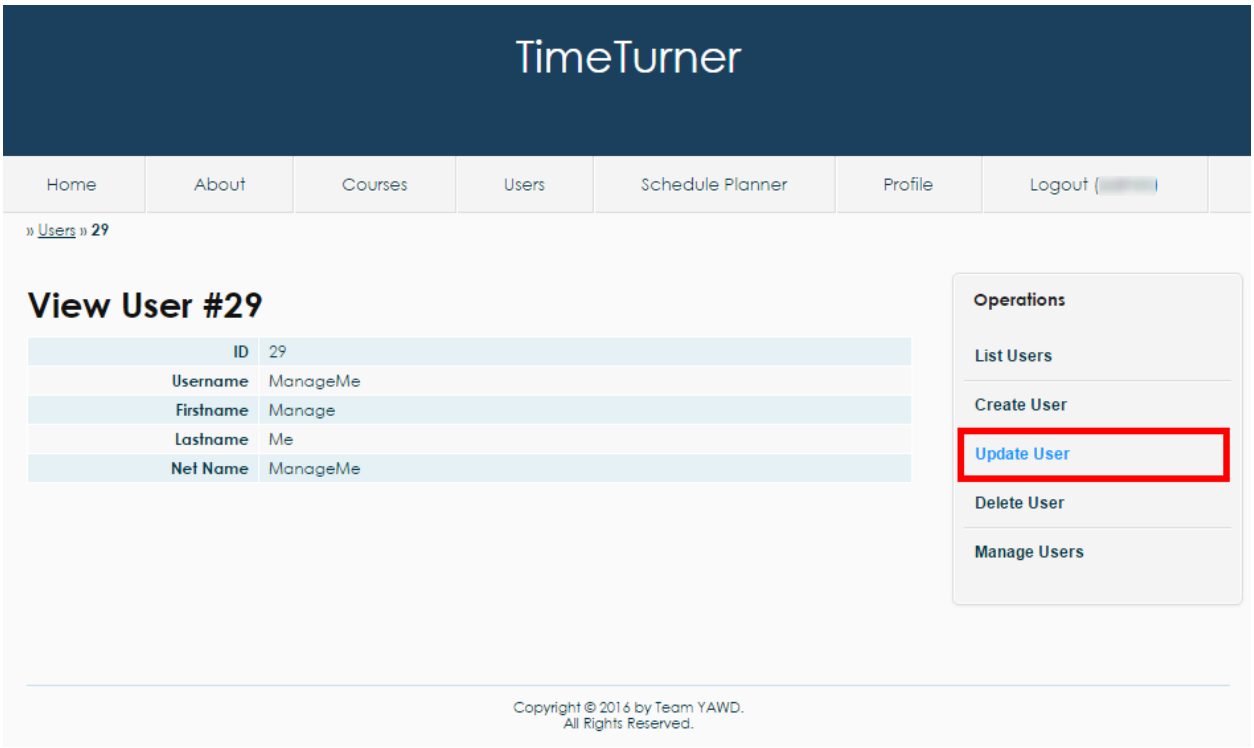


Figure 17.2.51 The operation 'Update User' found on the user's profile page view by an administrator

From either method, the 'Update User' page should now be displayed. The page contains a form which fields are completed with the user's current information. The administrator can decide to change any of those field and save the changes by clicking the 'Save' button (see Figure 17.2.52).

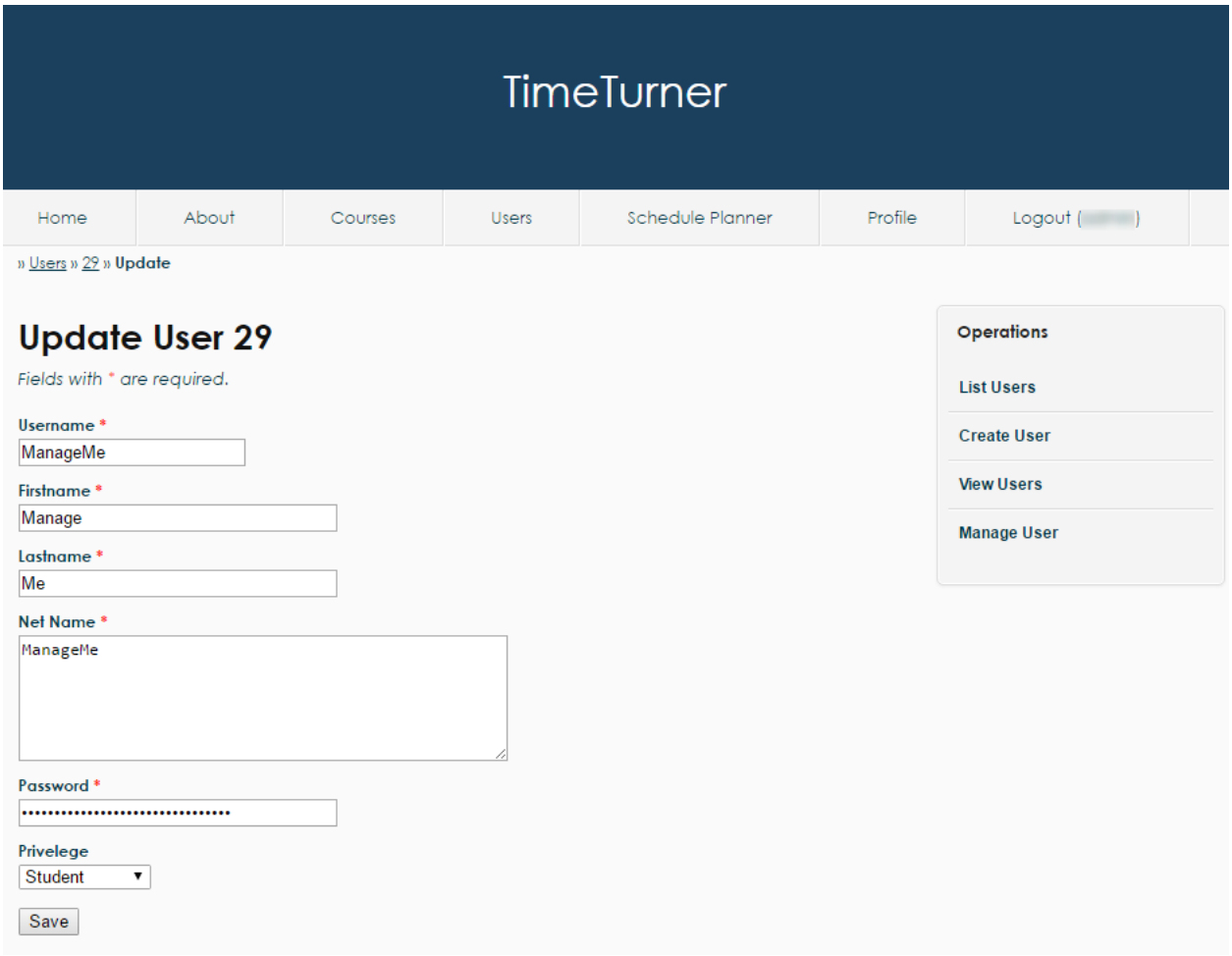


Figure 17.2.52. The 'Update User' page having fields that contain the user's initial information and that can be modified by the administrator once the 'Save' button has been clicked

Once the save button has been clicked, the updated profile of the user should be displayed. Otherwise, some conditions about the fields might have been violated and the administrator needs to correct those violation before updating the user.

17.2.2.16 Delete User

The administrator also has the power to delete a user through the user management page. Once again, deleting a user can be done either through the table of users or through the profile of the user.

To delete a user using the table of users, the administrator should click the 'X' button found on the same row as the user to be deleted (see Figure 17.2.53).

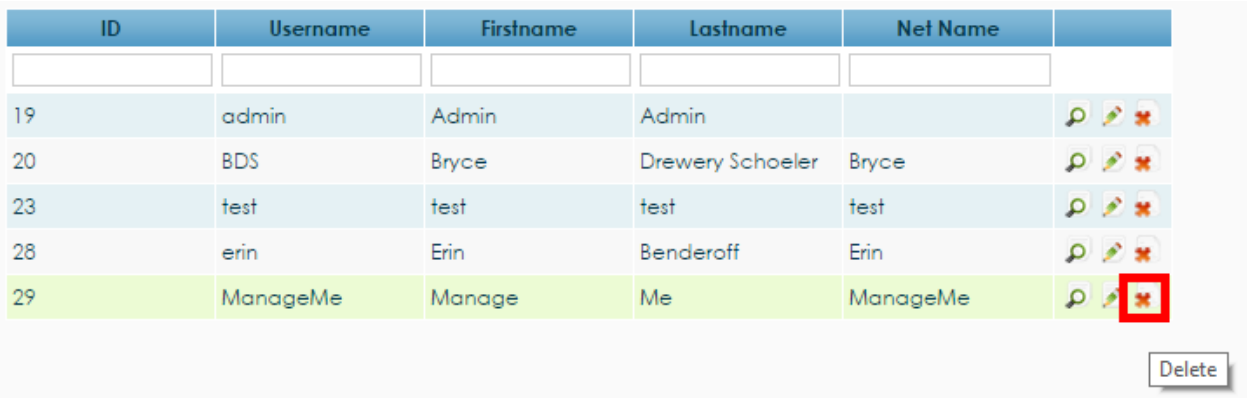


Figure 17.2.53. The 'X' button to delete user 29

To delete a user through the user's profile page, the operation 'Delete User' should be clicked from the list of operations found on the right (see Figure 17.2.54).

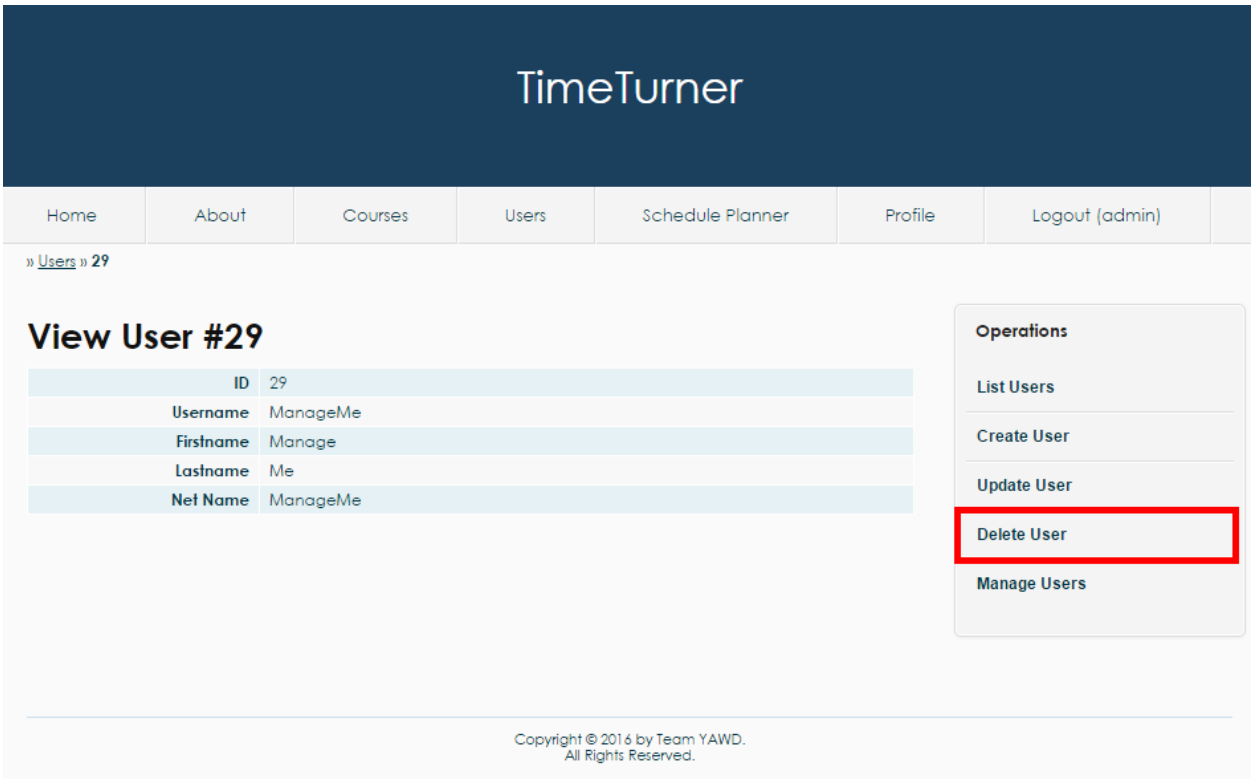


Figure 17.2.54. The operation 'Delete User' on the user's profile page viewed by an administrator

When either the 'X' button or the 'Delete User' operation is clicked, a message should pop up to confirm the administrator's decision (see Figure 17.2.55).

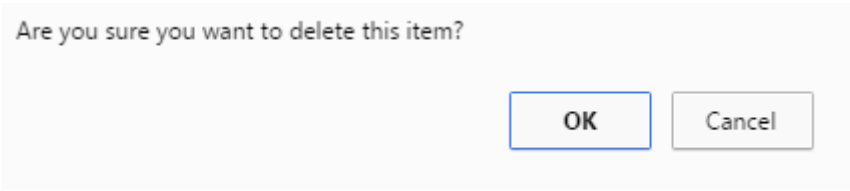


Figure 17.2.55. Confirmation by the system to delete a user

The administrator should press the 'OK' button on the confirmation box to delete the user. Once deleted, the administrator will be brought back to the user management page with an updated table of users.

17.2.2.17 Viewing Users

An administrator can view all users currently in the database. To do so, he/she can select the user's tab from the home page, once they have logged in (shown in Figure 17.2.56)

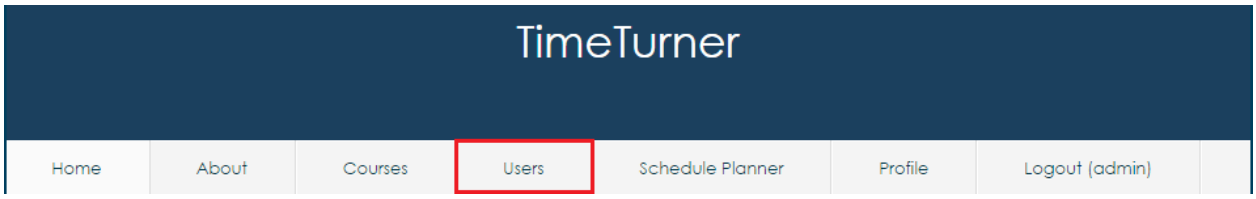


Figure 17.2.56: Accessibility to the 'User' tab from the home screen

Once the tab has been selected, all the users will be displayed on screen. The users are sorted by ID and displayed are each user's specific credentials, such as, first and last name and their net names (shown in figure 17.2.57)

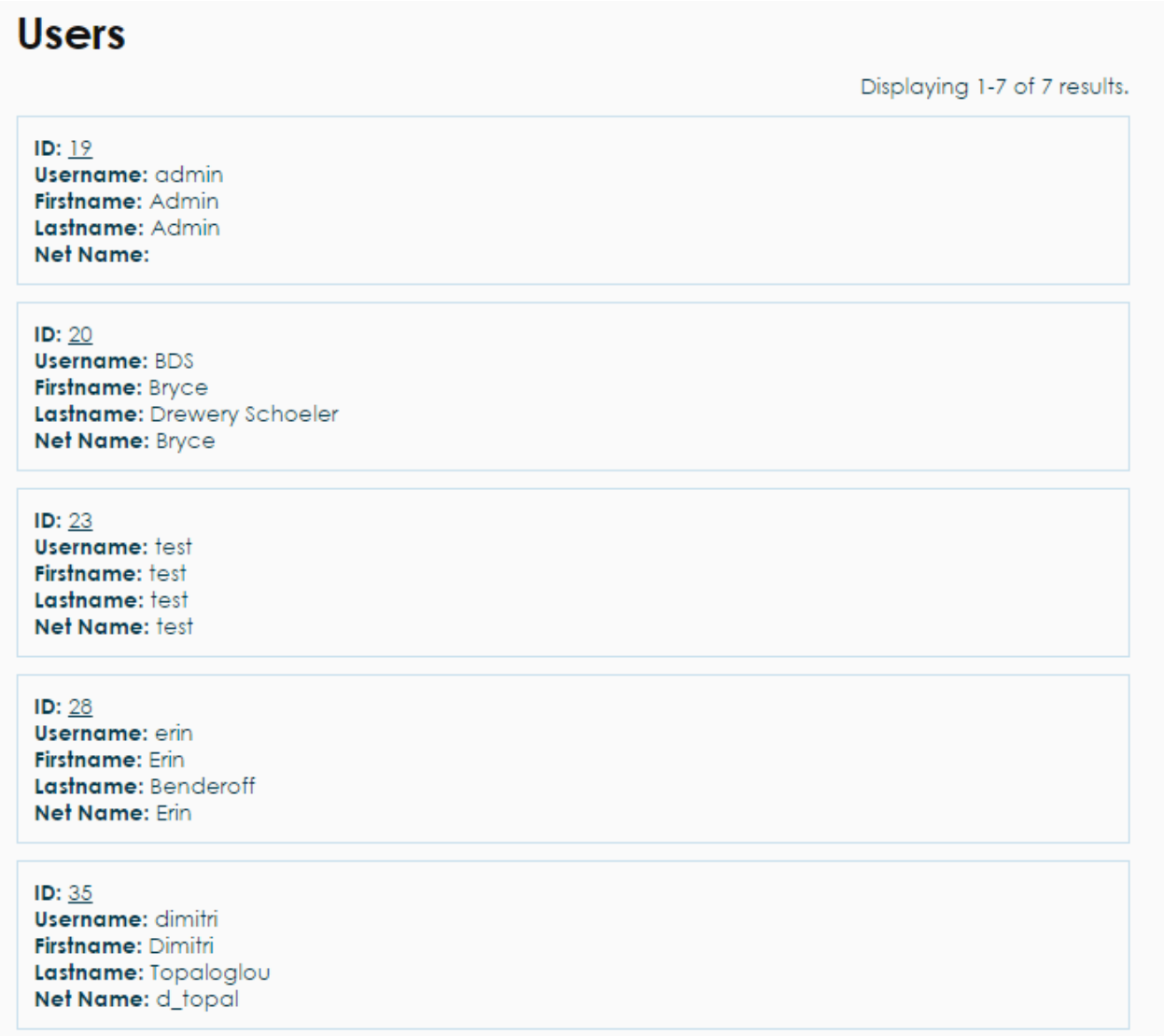


Figure 17.3.57. All users displayed on the 'User' tab

18. Final Cost Estimation

18.1 Final Estimations

18.1.1 Deliverable 0 Estimation

Task Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (Hours)
Domain Model	6	6	6
Identification of Technologies	12	12	12
Team Members	0.5	0.5	0.5
Implementation Discussion	5	5	5
DMO Descriptions	6	6	6
Total	**29.5**	**29.5**	**29.5**

18.1.2 Deliverable 1 Estimation

Task Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (Hours)
Requirements	10	10	36
Use Case Diagrams/Models	8	8	20
Resources	5	5	8
Scope	5	5	8
Architecture	7	7	15
Learning Technologies	----	----	137.85
Total	**35**	**35**	**224.85**

18.1.3 Deliverable 2 Estimation

Task Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (Hours)
Programming Architecture	100	116.4	125.5
Subsystem Specification	50	33.95	43.95
Dynamic Design Scenarios	25	29.1	30
Estimation	15	9.7	8.44
Prototyping and Risk	----	4.85	**5**
Total	**190**	**194**	**212.89**

18.1.4 Deliverable 3 Estimation

Task Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (Hours)
Execute Test Cases	30	29.1	29.1
Debugging/Optimizing code based on test cases	15	29.1	30
User Performance Test	20	33.95	35
Security and Stress Testing	10	4.85	4.47
Total	**75**	**97**	**98.57**

18.1.5 Deliverable 4 Estimation

Task Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (Hours)
Review and Edit past work	20	29.1	29.1
Compiling data into one file	30	9.7	9.7
Final Test on System	10	9.7	9.7
Total	**60**	**48.5**	**48.5**

18.1.6 Final Estimations

Deliverable Names	Cost (Hours)	Revised Cost (Hours)	Final Cost (hours)
Deliverable 0	29.5	29.5	29.5
Deliverable 1	35	35	224.85
Deliverable 2	190	194	212.89
Deliverable 3	75	97	98.57
Deliverable 4	60	48.5	48.5
Total	**389.5**	**404**	**614.31**

	Claudia	Dimitri	Daniel	Lori	Erin	Bryce	Aline	Philip	Ideawin	Ryan	Marc	Kevin
Deliverable 1	19.3	27	20	22	19	20	16.4	16.15	14	16	20	15
Deliverable 2	23.06	34	22	11.6	11.5	15	19.6	12.13	15	20	15	14
Deliverable 3	9.7	8	12	11.3	14.67	10	5.4	8.3	7	5	10	5.5

18.2 Updated Gantt Chart

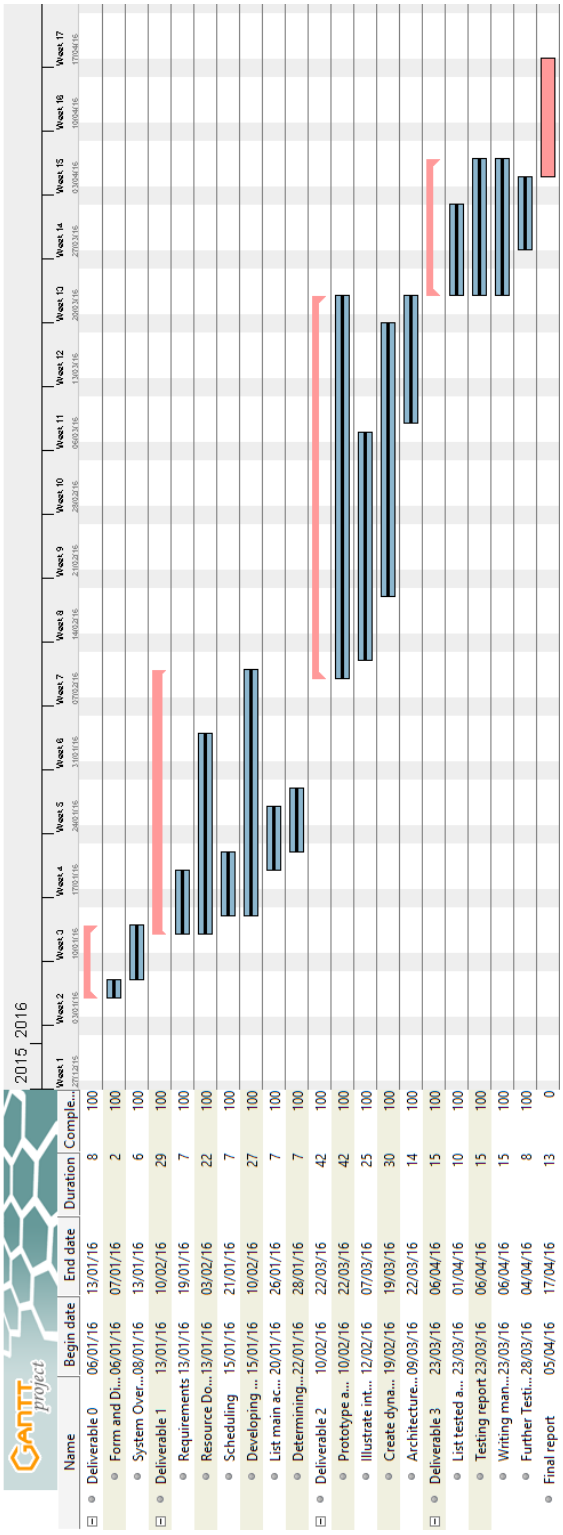


Figure 18.2.1 Updated Gantt chart

18.3 Final Gantt Chart

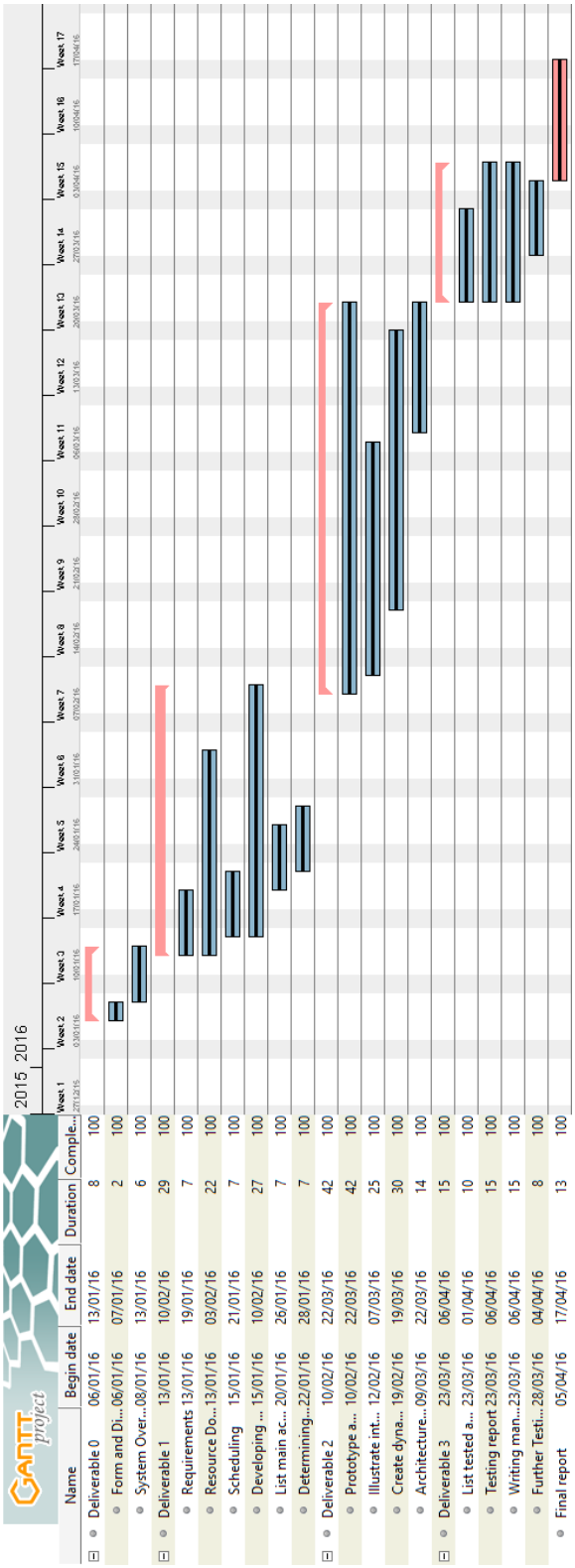


Figure 18.3.1 Final Gantt Chart