Concordia University

Department of Computer Science

and Software Engineering

Software Process

SOEN 341/4 S --- 2016

Project Design Document

Team: The Force

|  |  |
| --- | --- |
| Team information | |
| Name | SID |
| Stefano Pace | 27454716 |
| Julian Ippolito | 27419112 |
| Adam Arcaro | 27459157 |
| Joey Tedeschi | 27513062 |
| Georges Mathieu | 26863477 |
| Hasan Ahmed | 27546467 |
| Marc-Andre Dragon | 27721579 |
| Jordan Stern | 27400144 |
| George Theophanous | 27400578 |
| Olivier Cameron-Chevrier | 27228805 |

2. Introduction

The following deliverable will present 5 key topics that were essential in the continual development of “The Force” system. They are; Architectural Design, Detailed Design, Dynamic Design Scenario, Estimation and Rapid Prototyping & Risk.

In the Architectural Design section, we provide a high-level description of the system in terms of its modules and their respective purpose as well as exact interfaces. There are 2 subsections to this; Architecture Diagram and Subsystem Interface Specification. The architecture diagram will show the high-level structure of the system and the reasons for this design. The Subsystem Interface Specification will describe the parameters passed in these functions.

In the Detailed Design section, we provide a description of the system design and each subsystem and their purpose. They subsystems are User, Course and Scheduler. Here we also included 2 subsections; Detailed Design Diagram and Unit Descriptions. The detailed design diagram will show the internal working of the subsystems mentioned and describe the reasoning for their design. The unit descriptions list each class from the subsystem mentioned and describe their purpose as well as the attributes, functions and parameters used.

In the Dynamic Design Scenarios, we provide System Sequence Diagrams, Operational Contracts and full Sequence Diagrams of two main uses cases described from previous deliverables. These two uses are Generate Schedule and Add a Course.

In the Estimation section, we revise the estimates for previously identified tasks. We then update the tables describing the cost (man hours), number of days allocated as well as the start date for the tasks. Last, we show an updated schedule (Gantt chart) with the changes incorporated.

The last section is Rapid Prototyping and Risk. In this section we outline the updates made to the prototype of our system “The Force”. Included are screenshots as well as an updated Domain Model, Use Cases and newly found risks. Previous risks are also assessed to see if they are still valid to the scope of the project.

3. Architecture Design



As was discussed in Deliverable 1, The Force’s high-level architecture follows Model-View-Controller (MVC) architecture. Since the MVC architecture allows for a clear separation between domain logic, controllers, and user interfaces into separate entities/components, it is ideal for this project. Any requests for communication will be made through the appropriate controller, and the components designated to be models would not be exclusive to a single controller. Since the same data might be needed over numerous web pages (for example, when displaying a schedule on one page and when displaying student information on another), the controllers would be able to access said data directly. In addition, the separation of concern allows for development without the need for concern over minor interactions leading to unsatisfactory interference between components.

The complete design of The Force will be presented in various forms and contexts using the 4+1 view. This architectural view will break The Force into Logical, Process, Development, and Physical View, as well as include Scenarios for user interaction.

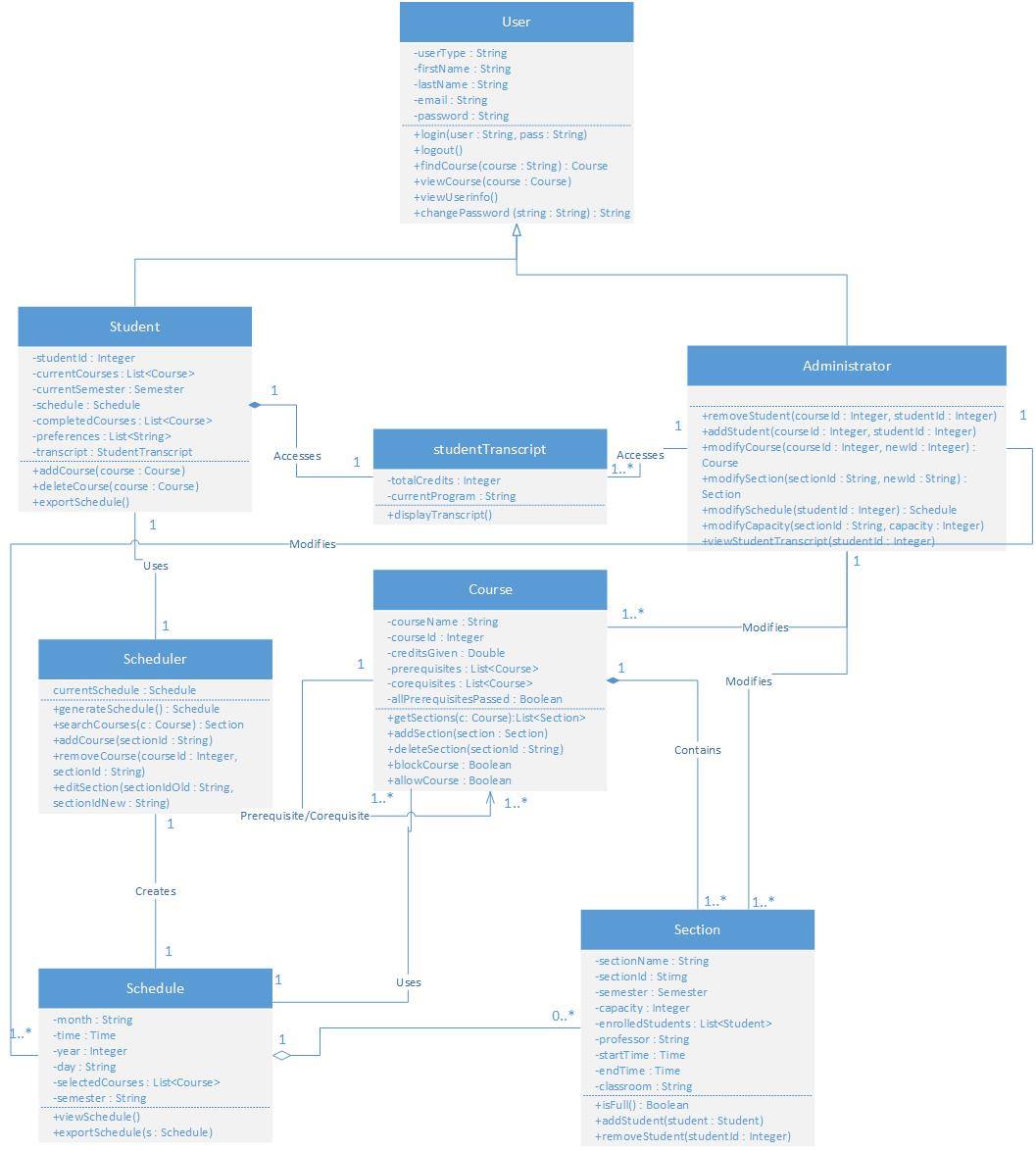
3.1. Architecture Diagram

3.1.1. Logical View

Logical views are concerned with presenting the functionality provided by the system to the users. As such, the logical view presented here is a presentation of the subsystems (and their components) of The Force’s model component (Refer to 3.1.2 for a full component diagram).

The actors in the system are denoted as Student and Administrator, both inheriting general characteristics of a standard “User”, but with their own unique functions as well. Students are given the ability to use the Scheduler to generate a possible combination of Courses he/she has yet to take, as well as view said Schedule once it’s been made. Each Student also possesses a studentTranscript (which cannot exist without said student), which is used to track his or her academic progress. The Scheduler creates Schedules through the use of Courses and Sections, these being available options that the Student still has remaining in their academic careers. The Student can make modifications to their Schedule, assuming the necessary requirements are met (e.g.: Prerequisites for a particular class), and can also export said Schedule for ease of access. The Administrator, though lacking in the ability to use the Scheduler, can modify the contents of all Courses, Sections, and Schedules, as well as view any studentTranscript he or she might want to see.

Through the use of the following class diagram, it can be seen that concerns are well separated through the use of multiple different classes. Unlike in a system where design dictates that a single class holds a large majority of the functionality, The Force ensures to the best of its abilities that concerns are separated appropriately. As such, changes can more easily be made without affecting a majority of the system, thus allowing for faster progress and implementation.

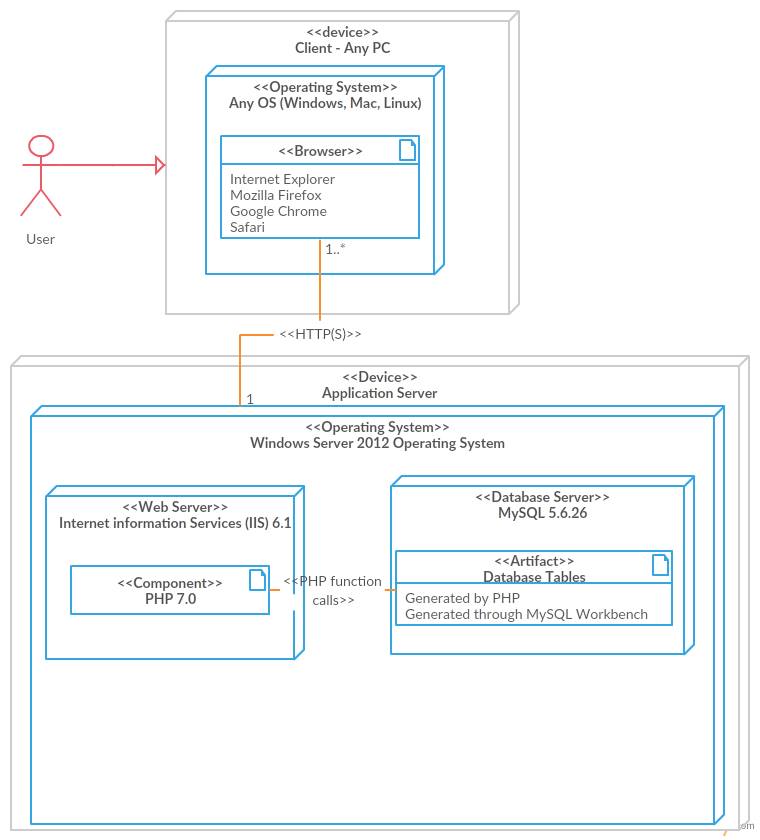
****3.1.2. Development View

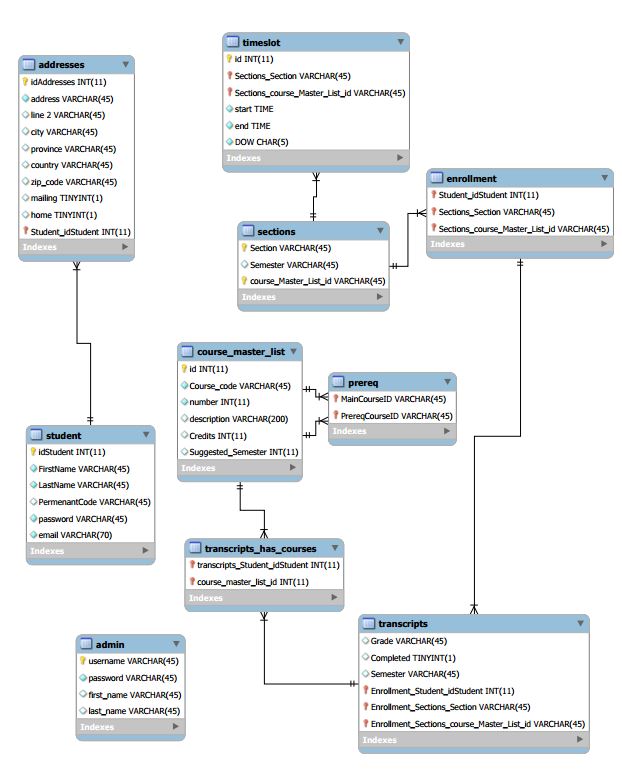
Development views, or implementation views, are concerned with software management. They aim to present a system from a programmer’s perspective.

The Force, as was previously mentioned, is designed using the MVC architecture and as such will be broken down into three components: Model, View, and Controller. All logic, for both the model and controller), is being implemented using PHP and is being handled by a single server. On said server, Windows Server 2012 Operating System is being used as the Operating System, and MySQL 5.6.26 is used for all database needs. The view is being rendered through the user’s browser using HTML, CSS and JavaScript, which the application is generating.

The client side of the application relies strictly on multi-platform browsers such as Google Chrome. As such, The Force will work on all clients but support is mainly given to current versions of most popular browsers currently being used. Through the use of HTTP requests and responses, communication is made possible between the user’s browser and the server.

In addition, a database model is provided in order to present the interactions between tables currently being stored on the server. This database is manipulated through the use of PHP (and MySQLi queries), and MySQL workbench.

****

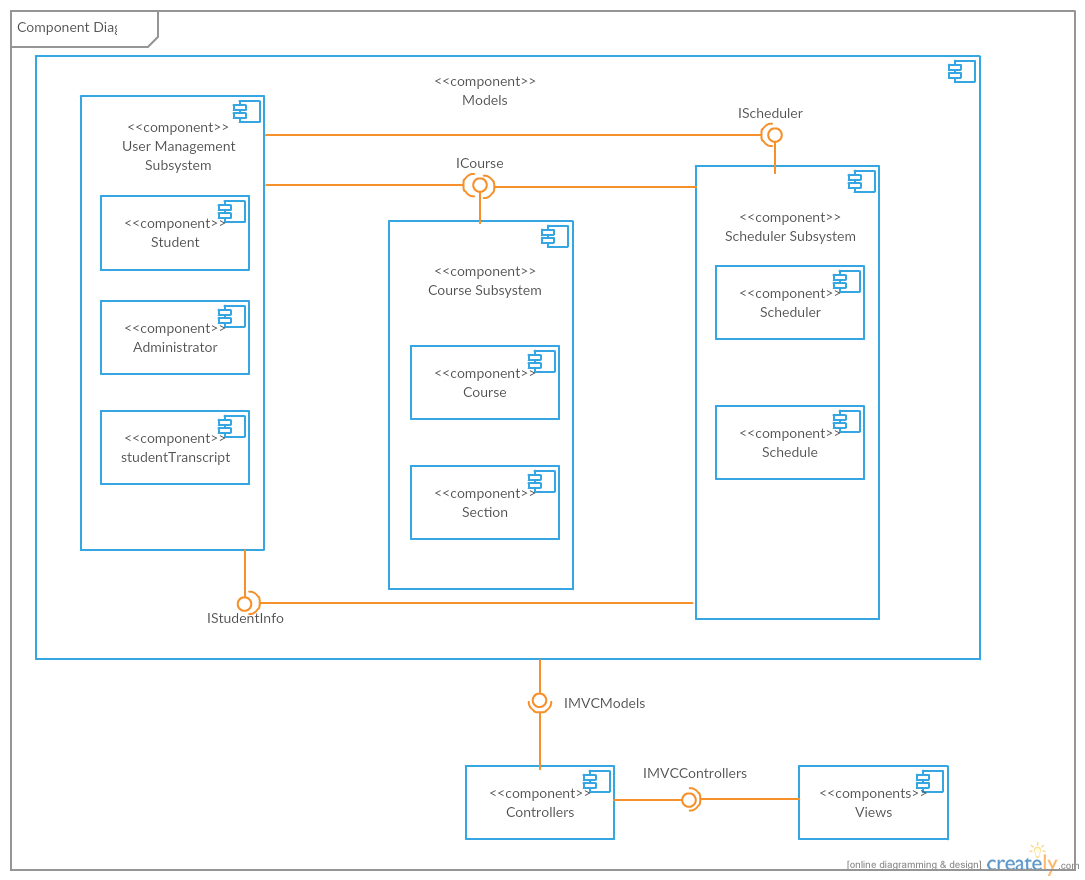
****

3.1.3. Physical View

Physical views are concerned with the software components of a given system, as well as how they communicate between each other. As The Force follows MVC architecture, it is broken down into three major components: Model, View, and Controller.

The Views component is composed of all the user interfaces a user would need to interact with while using The Force. It is used to connect information found within the Models to HTML templates, thus allowing a user to understand and interact with them. Each Views component is paired with a controller from the Controllers component, as it is to manipulate the Models based on user input and changes the Views accordingly.

The largest component is the Models component, which is made up of 3 distinct components representing the major subsystems of the application. The User Management subsystem is comprised of classes used to represent the various types of users who might be using The Force. In addition, they deal with all attributes and services unique to just the User classes, such as viewing student transcripts. Classes within the User Management subsystem not only provide information to other subsystems (namely the Scheduler subsystem, allowing its to compose schedules based on the appropriate student information), but it also requires methods from both subsystems in order to manipulate the schedule by modifying the courses being taken by said Student (or by a Student, in the Administrators case). The Scheduler subsystem, in addition to representing all classes relating to one’s schedule and schedule-making, also requires information from the Course subsystem in order to create proper representations of a Student’s academic semester. The schedules it houses are the used by the User Management subsystem, namely when exporting or viewing schedules. Finally, the Course subsystem encompasses all courses and sections provided by the university. It provides services to both other subsystems in order to generate, display, and modify appropriate configurations of courses.

****

**3.1.4. Process View**

Process views deal with how a system’s processes communicate, as well as focusing on the runtime behavior of the system. One activity diagram (Figure 3.1.4.1) is used in order to denote the main purpose of The Force (the generation of a schedule), and two more activity diagrams are used to go into greater detail on how schedules would be auto generated (Figure 3.1.4.2) or manually generated (Figure 3.1.4.3) during the runtime of the application.

Figure 3.1.4.1 describes the general runtime behavior of the complete schedule generation process. Once a student logins in (as an administrator doesn’t possess this feature), they are greeted with a main page consisting of their current semester’s schedule and courses. From here they may either exit the program, access their account page in order to change their personal information (and be redirected to the main page), or remain on the same page. From the main page, the student’s semester could be changed (allowing them to view past or future semesters) thus updating the displayed schedule and course list, or a schedule could be

generated. Generation could be done in one of two ways, auto generation (Figure 3.1.4.2) or manual generation (Figure 3.1.4.2), both of which will be explained in more detailed shortly. Once a schedule’s been made, the student can either quit the application, or modify their schedule manually (perhaps they are unhappy with an automatically generated course they were given, or they failed to notice an error while manually creating their schedule). Attempting to modify the current schedule will result in a display of the student’s current courses with modification options now present. Removing a course will simply update the schedule, however requesting to add or change a courses section will result in a list of alternatives (found similarly to what is seen in Figure 3.1.4.3), and once a new option is chosen the complete schedule will be displayed. Once modifications are complete, the student can make different modifications, or simply quit the application.

Figure 3.1.4.2 describes the auto generation process of creating a schedule. Once access to a student’s transcript and preferences are achieved, their current total credits within their program are analyzed. Should it be that they already have the maximum possible amount of credits they can achieve, they would have no courses left to complete and thus would be done, generating an empty schedule. Should their credits be lower than this arbitrary total amount, access is given to the student’s current program and course sequence, and the courseId for the next course in their sequence is acquired. Should the number of courses ever reach the max they are allowed per semester (e.g.: Perhaps 5 or 6), the schedule will stop adding courses and update itself. Otherwise, the next course will be selected and examined. If the student is still taking less than the recommended amount of classes, and the classroom still has rom in it, the course will be added to a list. So long as another course exists in the sequence, this cycle will continue until the maximum amount of courses is being taken, or there are no courses left (credits would equal the maximum amount), thus updating the schedule and ending the generation.

Figure 3.1.4.3 describes the process by which a schedule is generated manually by a student. Upon selecting the option, the student is told to enter their unavailability, or preferences (e.g.: No school on Friday, no class after 4:00pm), and they are then able to enter the name of the course they wish to add. If the course doesn’t exist, an error message appears and they can try again, otherwise the course information is displayed. Before searching for possible sections based on the students unavailability, room availability (whether there’s space in the class), and whether the necessary prerequisites are met, the student could decide against the current choice and change their mind. Once all checks have been performed, the student could cancel their selections (exiting the generation), or should a proper timeslot have been found, they may select their preferred section within the course. This will lead to their schedule being updated, and they may either stop choosing courses, or repeat the process again. It should be noted that the process used to check for a course availability is the same process used to determine the list of alternative sections proposed in Figure 3.1.4.1, thus it was omitted there to reduce redundancy.

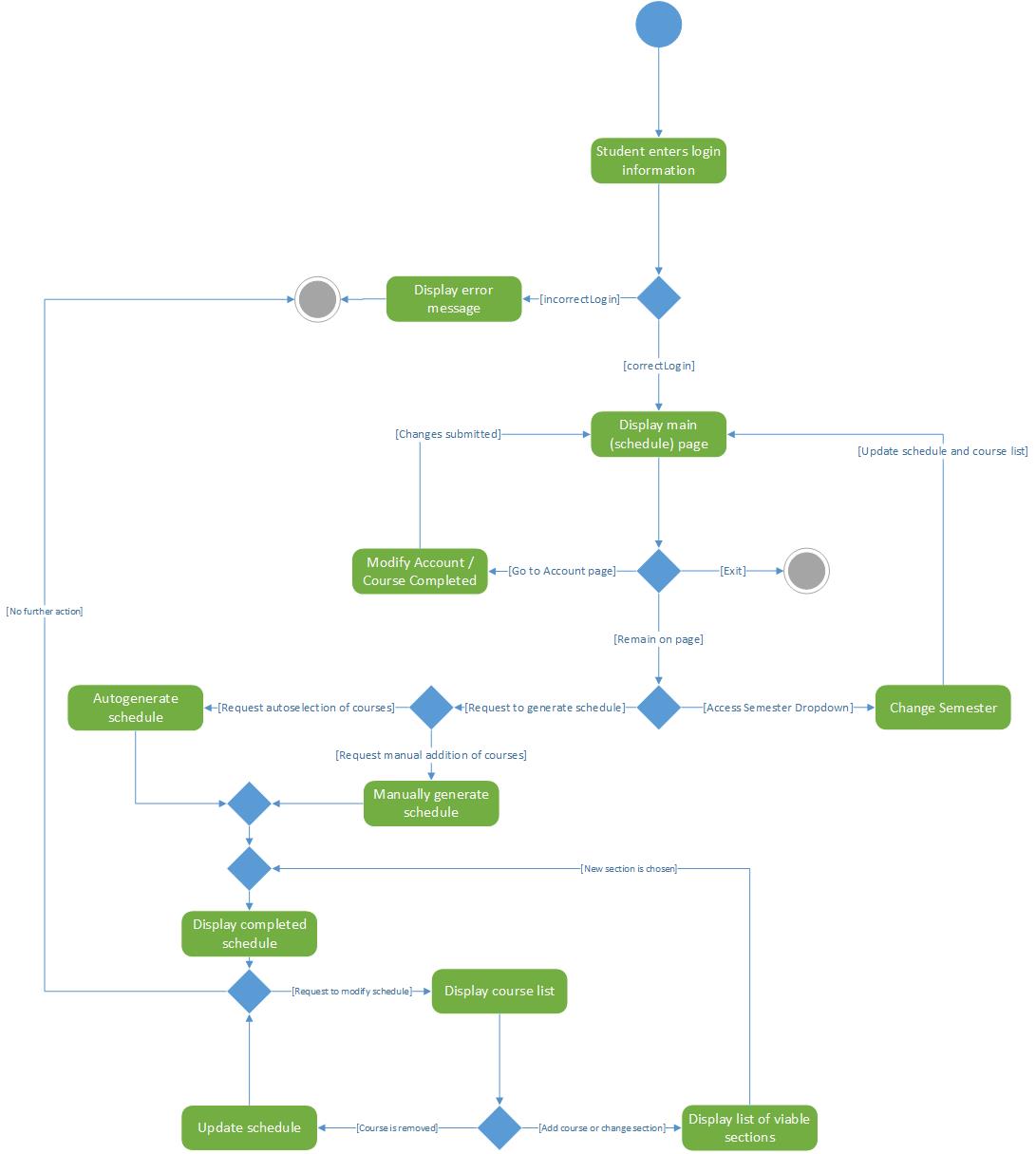
****

Figure 3.1.4.1: Main Schedule Generation

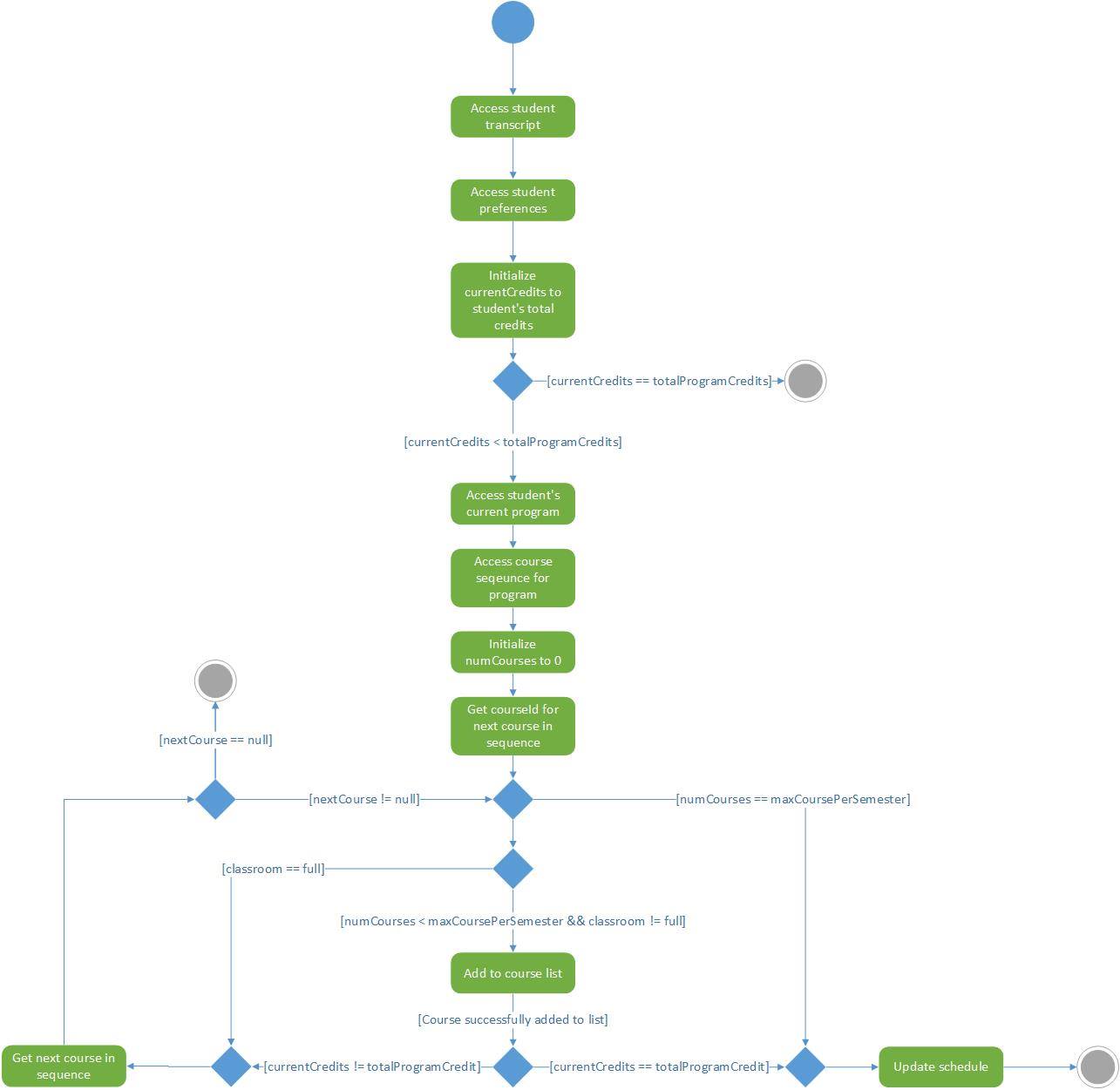
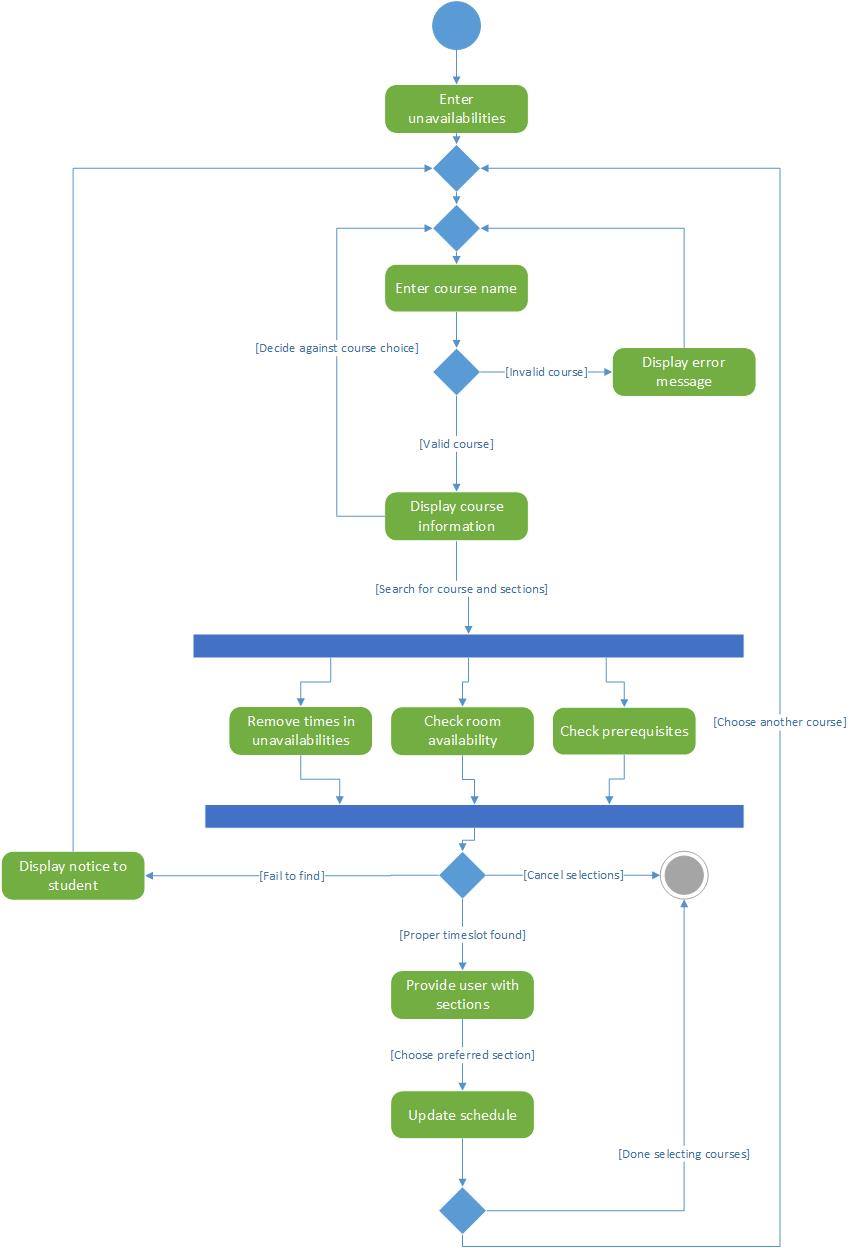


Figure 3.1.4.2: Auto Generated Schedule

Figure .1.4.3: User Generated Schedule

3.1.5. Scenarios

Scenarios are used to interactions between objects and between processes, as such Use Case Diagrams will be used to denote the way the various types of users can interact with The Force.

It should be noted that, by comparison to the Use Case Diagram made in Deliverable 1 (Figure 3.1.5.1), quite a bit has changed. First and foremost, the UCD from Deliverable 1 was strictly for one situation, most notably assuming that the only type of user would be a student. Since then, the addition of the administrator user type not only required a separate diagram (Figure 3.1.5.4), but also allowed for the creation of a general User diagram (Figure 3.1.5.2). This use case diagram denotes interactions which both Students and Administrators have access too, and helped not only reduce redundancy within each unique diagram, but also allowed a clear distinction to be made between the roles each type of user is allowed to play, and the access each type of user is given. As before, a Use Case Diagram is still present for Students (Figure 3.1.5.3), but it now includes new interactions which were added to The Force between Deliverable 1 and 2, and allows for a clearer view of its unique interactions in comparison to the Administrators.

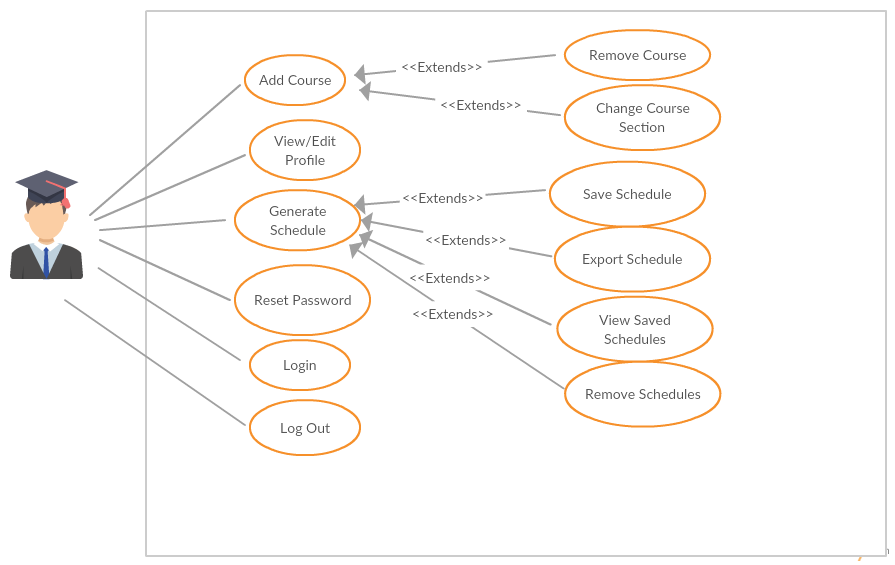


Figure 3.1.5.1. : General UCD from Deliverable 1

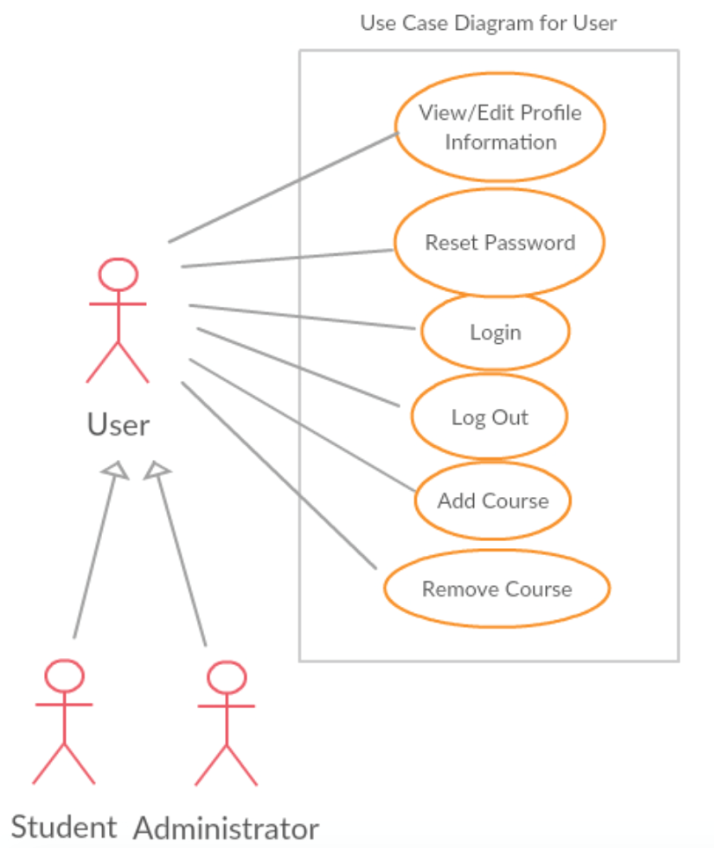


Figure 3.1.5.2. : General User UCD

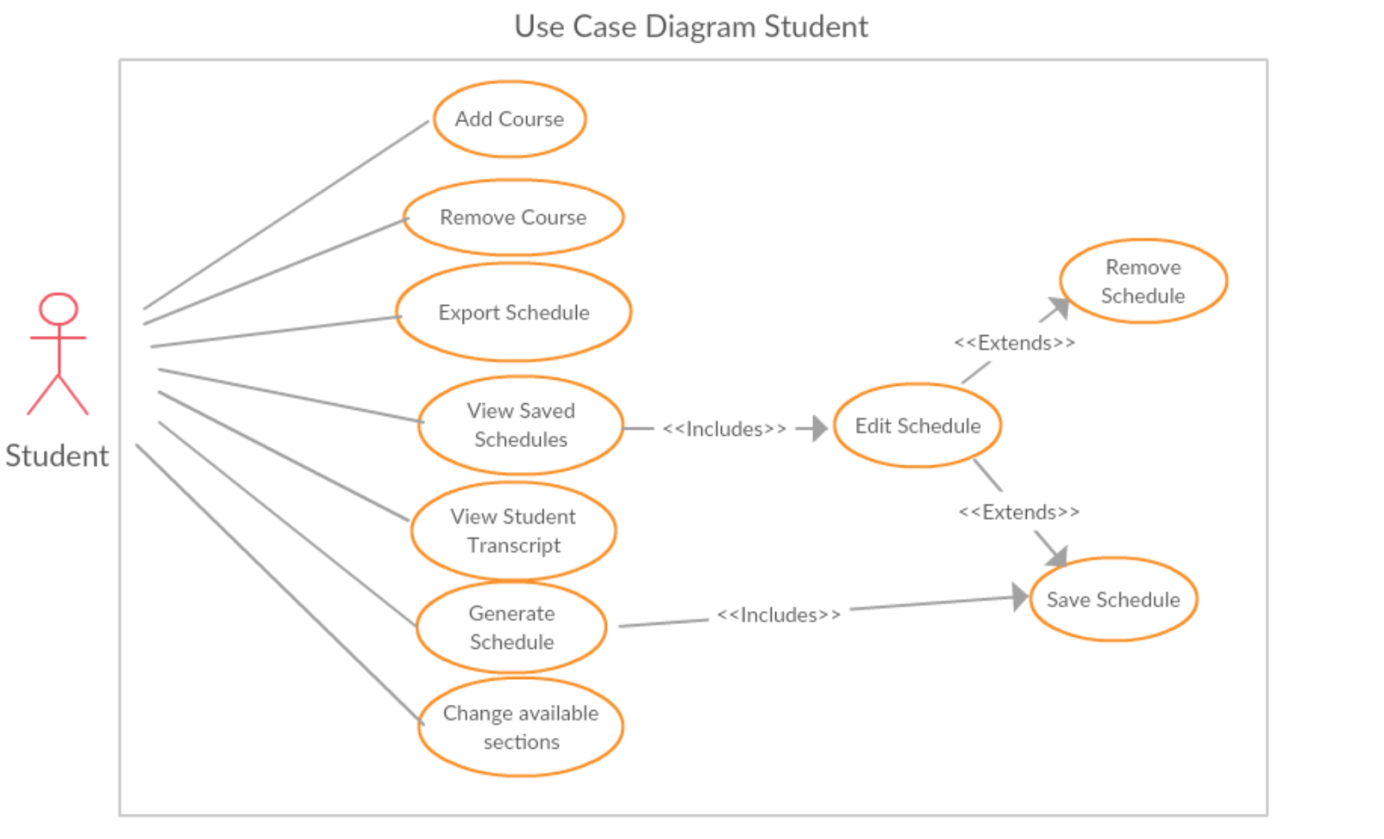


Figure 3.1.5.3. : Student UCD

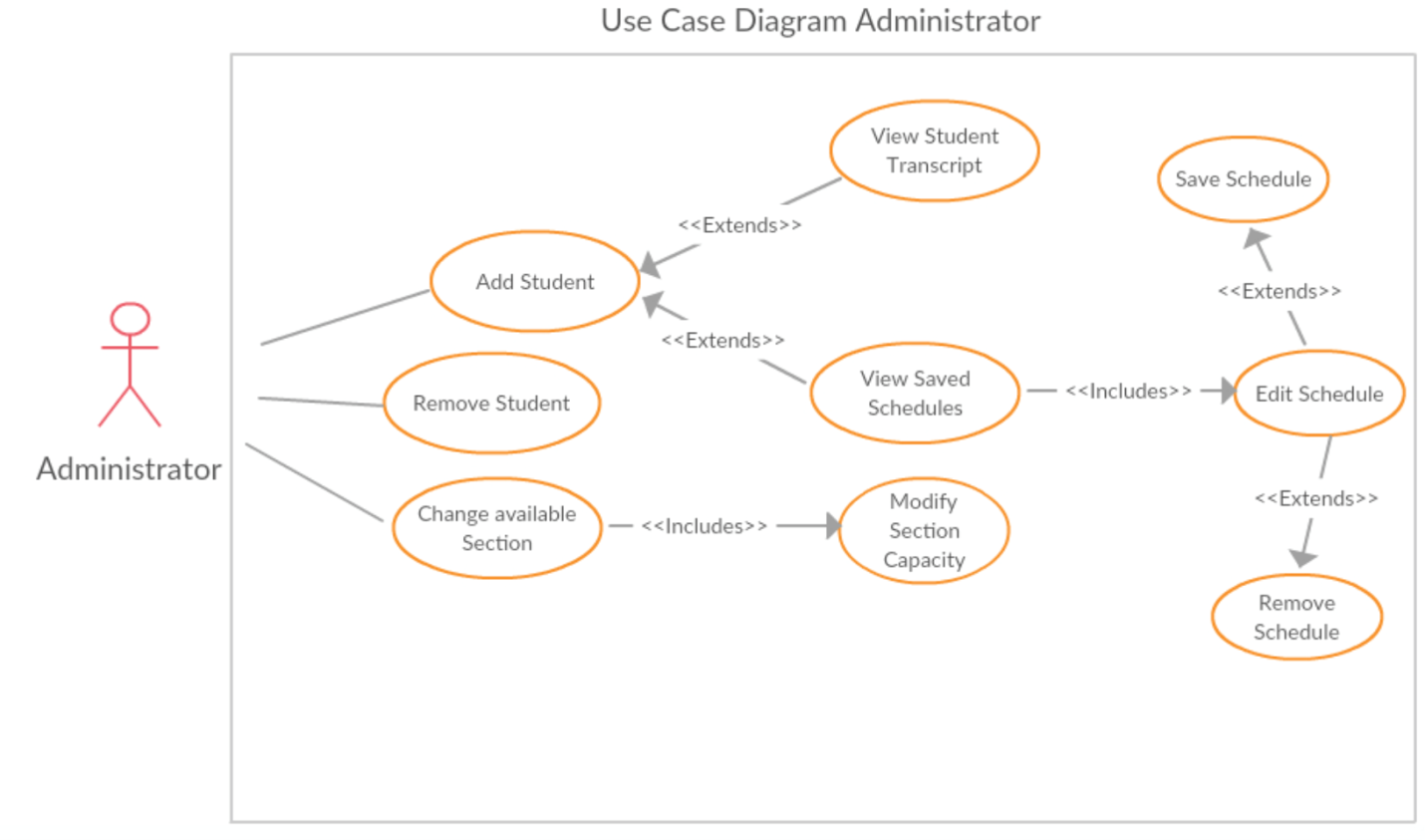
****

Figure 3.1.5.4. : Administrator UCD

3.2. Subsystem Interfaces Specifications

The system will be broken down into three separate subsystems in order to not only facilitate separation of concerns, but also to facilitate public interfacing. All methods referenced below will be those found in the class diagram used to represent Logical View (refer to section 3.1.1.). All constructors, as well as all getter and setter methods, are assumed whenever necessary (as all variables as private), but won’t be included in order to reduce redundancy. Another important note is that some methods, such as the inherited login and logout, are not particularly relevant to one interface or another, nor are they used by subsystems they’re not a part of. These methods will also be removed from the specifications to reduce clutter and increase cohesion.

3.2.1. IStudentInfo

|  |  |
| --- | --- |
| Class | Student |
| Methods: | * addCourse(Course course) * Add a given course to the current student’s course list. * Parameters: * course – The course to be added to the current semester’s course list. * Invalid values: NULL * Returns: void |
|  | * deleteCourse(Course course) * Remove a given course from the current student’s course list. * Parameters: * course – The course to be removed from the course list. * Invalid values: NULL * Returns: void |
|  | * exportSchedule() * Export the current student’s schedule. * Parameters: * None * Returns: void |

|  |  |
| --- | --- |
| Class | studentTranscript |
| Methods: | * displayTranscript() * Display current student’s transcript to be seen by the user. * Parameters: * None * Returns: void |

|  |  |
| --- | --- |
| Class | Administrator |
| Methods: | * removeStudent(Integer courseId, Integer studentId) * Remove the given student from the given course. Student’s course list is updated as well. * Parameters: * courseId – The course the student will be removed from * Invalid values: NULL. Student must be enrolled to be removed. * StudentId – The student to be removed * Invalid values: NULL * Returns: void |
|  | * addStudent(Integer courseId, Integer studentId) * Add the given student to the given course. Student’s course list is updated as well. * Parameters: * courseId – The course the student will be added to * Invalid values: NULL. Student must not be enrolled already. * StudentId – The student to be removed * Invalid values: NULL * Returns: void |
|  | * modifyCourse(Integer courseId, Integer newId) * Modify a given course’s courseId to a new value. * Parameters: * courseId – The course to be modified * Invalid values: NULL * newId – The new courseId * Invalid values: NULL * Returns: Course. Returns the updated Course |
|  | * modifySection(String sectionId, String newId) * Modify a given section’s sectionId to a new value. * Parameters: * sectionId – The section to be modified * Invalid values: NULL * newId – The new sectionId * Invalid values: NULL * Returns: Section. Returns the updated Section |
|  | * modifySchedule(Integer studentId) * Gain access to a given student’s schedule for future modifications. * Parameters: * studentId – The student whose schedule will be accessed. * Invalid values: NULL * Returns: Schedule. Returns the Schedule to be modified at a later time. |
|  | * modifyCapacity(String sectionId, Integer capacity) * Modify a given section’s maximum capacity. * Parameters: * sectionId – The section to be modified * Invalid values: NULL * capacity – The new capacity of the section * Invalid values: NULL, capacity < 0. Capacity must be defined, and can’t be a negative number. * Returns: void |
|  | * viewStudentTranscript(Integer studentId) * View a given student’s official transcript * Parameters: * studentId – The student whose transcript will be accessed. * Invalid values: NULL * Returns: void |

3.2.2. IScheduler

|  |  |
| --- | --- |
| Class | Scheduler |
| Methods: | * generateSchedule() * Generates a schedule for a student. * Parameters: * None * Returns: Schedule |
|  | * searchCourses(Course c) * Retrieves a section within Course c. * Parameters: * c – The course whose section is being identified * Invalid values: NULL * Returns: Section |
|  | * addCourse(String sectionId) * Add course with the given section, and sectionId, to the schedule. Eliminates the need to pass a course as well as a section. * Parameters: * sectionId – The id of the section found within the course to be added. * Invalid values: NULL * Returns: void |
|  | * RemoveCourse(Integer courseId, String sectionId) * Removes the course and section whose ids were passed, from the schedule. * Parameters: * courseId – The id of the course to be removed. * Invalid value: NULL * sectionId – The id of the section to be removed. * Invalid value: NULL * Returns: void |
|  | * editSection(String sectionOld, String sectionNew) * Find the section whose id is sectionOld, replace the id by sectionNew. * Parameters: * sectionOld – The section to be found. * Invalid value: NULL * sectionNew – The new id of the section. * Invalid value: NULL * Returns: void |

|  |  |
| --- | --- |
| Class | Schedule |
| Methods: | * viewSchedule() * Display current schedule for user to see. * Parameters: * None * Returns: void |
|  | * exportSchedule(Schedule s) * Export a given schedule to PDF format. * Parameters: * s – The schedule to be exported * Invalid value: NULL * Returns: void. |

3.2.3. ICourse

|  |  |
| --- | --- |
| Class | Course |
| Methods: | * getSections(Course c) * Retrieves all sections associated with a given course c. * Parameters: * c – The course which contains the sections required. * Returns: List<Schedule> |
|  | * addSection(Section section) * Add a section to the course calling the method. * Parameters: * section – The section to be added. * Invalid values: NULL * Returns: void |
|  | * deleteSection(String sectionId) * Deletes the section associated with the provided id. * Parameters: * sectionId – The id of the section to be removed. * Invalid values: NULL * Returns: void |
|  | * blockCourse() * Blocks the course from being registered to. * Parameters: * N/A * Returns: Boolean. Returns true if course is now blocked, false otherwise. |
|  | * allowCourse() * Allows the course to be registered to. * Parameters: * N/A * Returns: Boolean. Returns true if course is now allowed, false otherwise. |

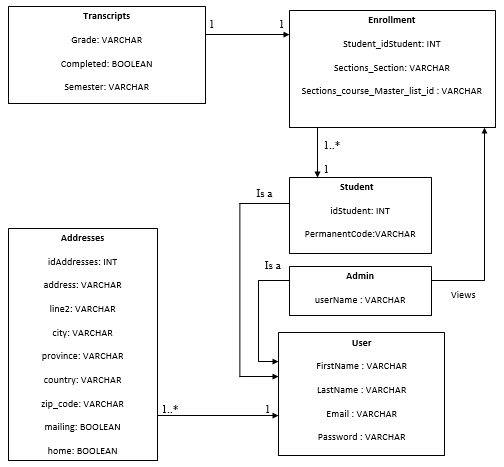
|  |  |
| --- | --- |
| Class | Section |
| Methods: | * isFull() * Checks if the calling section is full. * Parameters: * N/A * Returns: Boolean. Returns true if the section is full, false otherwise. |
|  | * addStudent(Student student) * Add a student to the calling section * Parameters: * student – The student to be added * Invalid values: NULL * Returns: void |
|  | * removeStudent(Integer studentId) * Remove the student with the id being passed from the calling section. * Parameters: * studentId – The id of the student to be removed. * Invalid values: NULL * Returns: void |

# 4. Detailed Design

4.1 Detailed Design Diagram

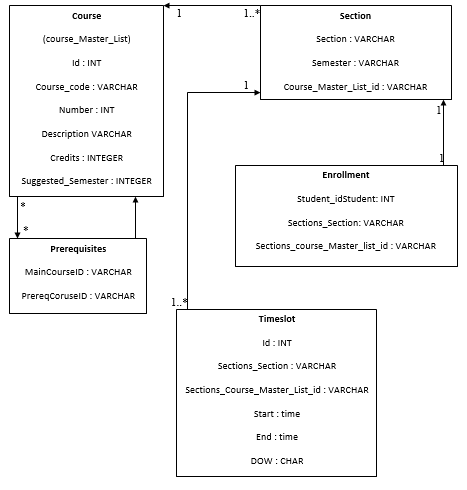
This detailed design will include the logical back-end organization of “The Force” thus far. These diagrams differ from the class diagrams in that they represent the system as it is organized in the database. Non labeled arrows show a “Has-a” relationship.

User management subsystem



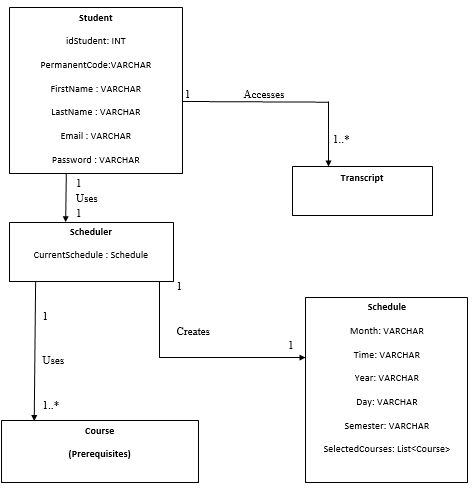
This subsystem includes all student and admin information information. It allows for the creation of an account on “The Force”. The student may first sign up, and then subsequently sign in with their credentials. Any account will be considered a “User”, but depending on restrictions the account will either be of type “student” or “admin”. Furthermore, the subsystem stores relevant information regarding transcripts and student records. This allows the subsystem to communicate clearly with the course registration subsystem. The Student class will be referenced by “Enrollment” in the database, which in turn will be referenced by the “transcripts” class. An instance of the “transcript” class refers to a course the student has taken. Any superfluous information about the student that is not directly needed to enroll in a course is stored in a separate class called “Addresses” which references the “Student” class. The address table is included in the design for two reasons. The first reason is so the user can maintain his/her current address through this portal (stretch goal of the system). The second was to account for this system’s storage to draw information for other systems the school could use. This separation allows for easier manipulation of a user. The “enrollment” class is what will allow the user management subsystem to communicate with the course registration subsystem. Finally, if an admin wishes to view the transcript of a student in a specific course, they may because the enrollment and transcript classes are linked.

Course Subsystem



The course subsystem contains all information regarding course numbers, IDs, and times. Having an entire subsystem where this information can be stored allows for a clean separation of tasks between the two subsystems. Although a student will enroll to a course from their point of view, the system sees it as an enrollment in a section. This is because different sections have different timeslot instances. This allows the system to simultaneously enroll the student while checking for time conflicts. It is important to note that any information regarding the course name and number starts at the course class. From there, you can even find the suggested semester. As you go further away from the course class, there is a “breadcrumb trail” that can always lead you back to it easily. Class Section contains an ID that matches to a class in the master list. Enrollment and timeslot classes have the same course name and number, although we append “Section\_” at the front, to force the path through the section class, without accidentally accessing every single section in the course. On top of this information, they contain the section ID in order to not pass through and get enrolled into the wrong section.

Scheduler Subsystem



The Scheduler Subsystem includes all elements of the database and app that allow a student user to create a comprehensive schedule that they can use. The Student class has been shown with all the information of a user included to show that that information is crucial to creating the correct schedule and assigning it to the correct student. The part of the diagram that details that the student accesses the transcript is simplified. From the database’s point of view, the user accesses their transcript of each course separately. The app, however, collects these into one transcript to make it easier for the student to view. This is essential to know what courses are necessary to enroll in. The Scheduler itself is contained within the app, and it uses the student’s selections of courses to generate a schedule. Once again, this aprt has been slightly simplified. The prerequisites and courses have been grouped in order to have a cleaner diagram. Finally, the scheduler creates a schedule. This schedule has all the information from timeslot classes in it, as well as a list of courses. This subsystem has a purpose of bridging the gap between the user and the database. Effectively, it provides all information to the student clearly, as well as manipulates it. This is probably the most essential of subsystems, since without it, there is no app. Finally, it serves as a bridge between the other subsystems as well, functionally speaking (organizationally, they are already linked, but not functionally without this subsystem).

# 4.2 Unit description

User management subsystem

**Class User**

* FirstName (VARCHAR, String): stores the user’s first name
* LastName (VARCHAR, String): stores the user’s last name
* Email (VARCHAR, String) : stores the user’s email.
* Password (VARCHAR, String) : stores the user’s password for authentification purposes.

**Class Admin**

* userName (VARCHAR, String) : stores the admin’s User Name to display.

**Class Student**

* idStudent (INT): stores the student id number
* PermanentCode (VARCHAR, String) : stores the student’s permanent code

All above information is what is vital to the student’s ability to registering for courses and singing in.

**Class Addresses**

* idAddresses (INT): Because a user may possess many addresses, idAddresses stores an id number which may help refer to the same user without confusing the address instances.
* address (VARCHAR, String) : stores the user’s address
* line 2 (VARCHAR, String) : stores the user’s alternate address
* city (VARCHAR, String) : stores the user’s city
* province (VARCHAR, String) : stores the user’s province
* country (VARCHAR, String) : stores the user’s country
* zip\_code (VARCHAR, String) : stores the user’s zip code (postal code)
* mailing (BOOLEAN) : determines whether or not the given address is the mailing address to send any important documentation.
* home (BOOLEAN) : stores whether or not the address is a home address or not

**Class Enrollment**

* Student\_idStudent (INT) : stores a student’s id number. This allows the instance of “enrollment” to associate with a specific student.
* Sections\_section (VARCHAR, String) : stores the section the student is enrolled in.
* Sections\_course\_Master\_list\_id (VARCHAR, String) : stores the id number of the course the student is enrolled in from the database. This allows for the program to determine which courses have been completed or not.

**Class Transcripts**

* Grade (VARCHAR, String): stores the student’s grade in the given course
* Completed (BOOLEAN) : stores whether or not the student successfully completed the course or not.
* Semester (VARCAHR, String): stores the semester in which the student took the course

Course Subsystem

**Class Enrollment**

* See “User management subsystem”
* This class serves as a “UI” between the two subsystems
* Sections\_Course\_Master\_List\_id (VARCHAR, String) : Refers to the id of a course. This id is first matched to a section along with Sections\_Section (VARCHAR, String) to match to a section the student is enrolled in.

**Class Section**

* Section (VARCHAR, String) : Stores the section number of a course the student is enrolled in.
* Semester (VARCHAR, String) : Stores the semester of the course the student is enrolled in.
* Course\_Master\_list\_id (VARCHAR, String) : Stores the ID of the course the student is enrolled in. This is matched to a specific ID in the database (Course class).

**Class Course**

* Takes the form of Course\_Master\_list in the database. Contains all information regarding courses.
* Id (INT) : Stores the ID number of a course.
* Course\_Code (VARCHAR, String) : Stores the course code in a string (4 letters)
* Number (INT) : Stores the course number (3 digits).
* Description (VARCHAR, String) : Stores the description of a course so that the student may know what they are chosing.
* Credits (INTEGER) : Stores the number of credits the course provides.
* Suggested\_Semester (INTEGER) : Stores the suggested semester of the year for the student (between 1 and 3). This allows to provide the student with a recommended course sequence.

**Class Prerequisites**

* MainCourseID (VARCHAR, String) : Stores the course name and number of the course the student is trying to enroll in.
* PrereqCourseID (VARCHAR, String): Stores the course name(s) and number(s) of the course(s) the student should have previously completed in order to enroll to “MainCourse”.

**Class Timeslot**

* ID (INT) : contains the student number of the student taking a course in that given timeslot.
* Sections\_Section (VARCHAR, String) : Stores the section number of the course of the timeslot the student is enrolled in.
* Sections\_Course\_Master\_List\_id (VARCHAR, String) : Stores the course name and number of the section of the timeslot the student is enrolled in.
* Start (Time) : the starting time of the timeslot.
* End (Time) : The ending time of the timeslot.
* DOW (CHAR): Day of the Week represented by a single character.

User Subsystem

**Class Student**

* See “user management subsystem” to see the separation between “user” and “Student”. In this diagram, those have been shown as one.

**Class Transcript**

* See “user management subsystem” to see the separation between “enrollment” and “transcript” in the database. In the app, however, a transcript contains all relevant information of all courses the student may have taken.

**Class Scheduler**

* CurrentSchedule: Stores the schedule which the scheduler generates.

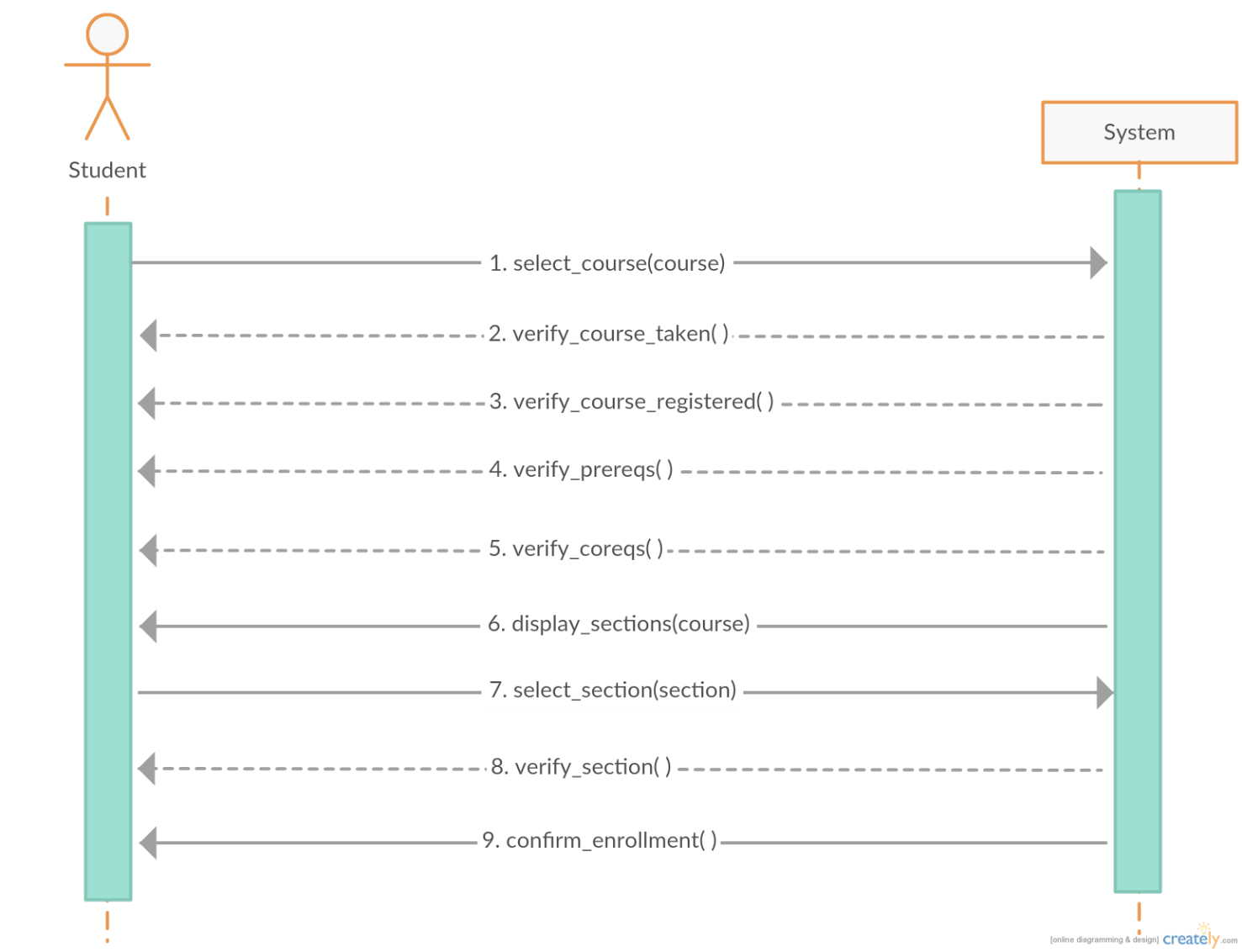
**Class Schedule**

* Month (VARCHAR, String) : Stores the month the schedule is displaying.
* Time (VARCHAR, String) : Stores the time that the schedule is displaying.
* Year (VARCHAR, String) : Stores the year that the schedule is displaying.
* Day (VARCHAR, String) : Stores the day that the schedule is displaying.
* Semester (VARCHAR, String) : Stores the semester that the schedule is displaying.
* SelectedCourses (List<Courses) : List of courses the student has chosen to include in the schedule.

**Class Course (Prerequisites)**

* See “Course subsystem” to see the separation between the “Course” and “Prerequisites” classes in the database. In the app, these are joined together.

# 5. Dynamic Design Scenarios



*Manually Registering a Course – System Sequence Diagram*

This system sequence diagram shown above demonstrates the interaction between the actor (student) and the system (course generator). This simplified model omits any technical functions performed by the system itself, and focused on the student’s perspective.

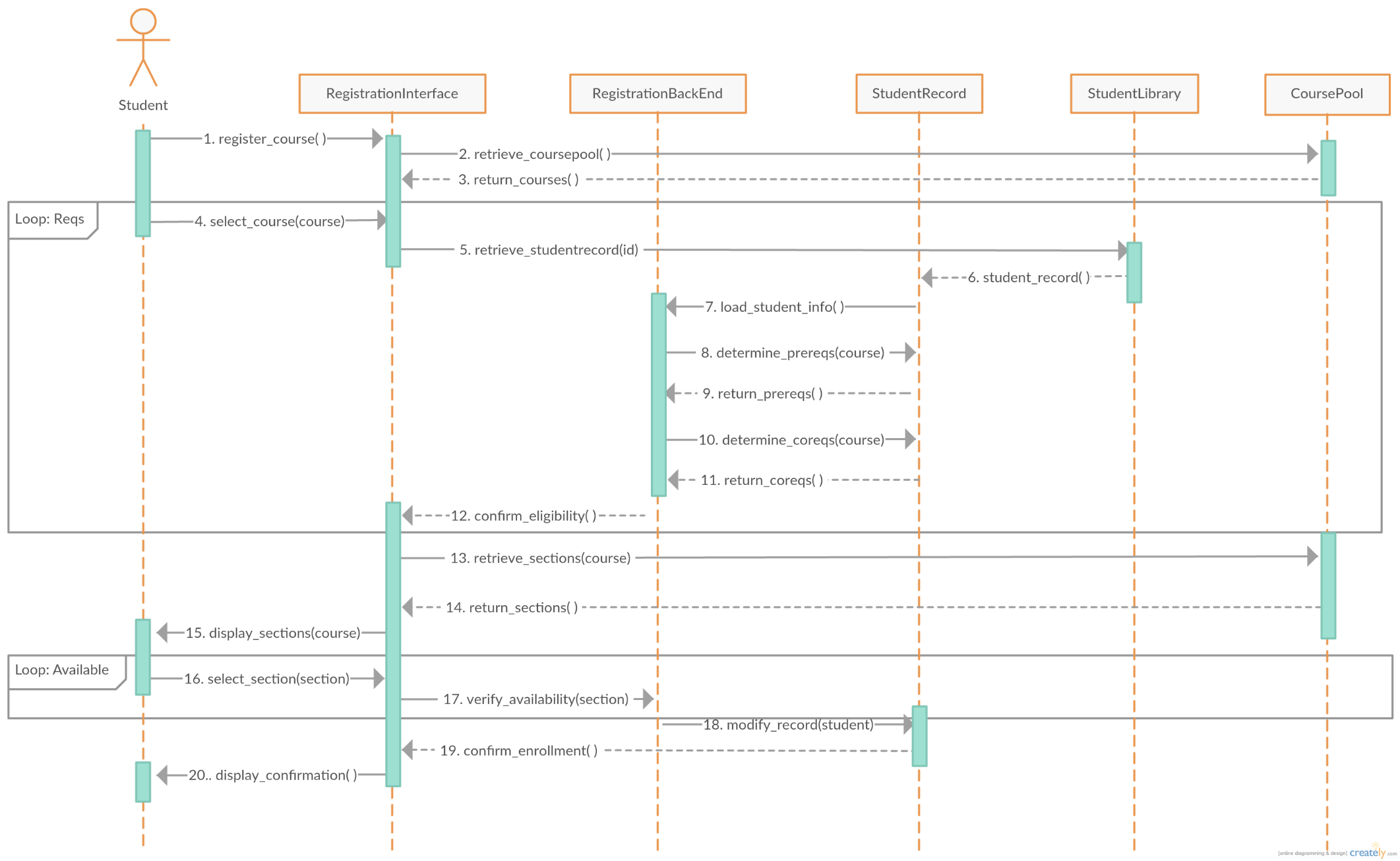
Once the student initially accesses the application, they must select the course they desire to be enrolled in. The system then verifies all the conditions for enrollment: if the course has already been credited to the student, if the student is already registered for the course, if the student has the necessary prerequisites completed, and if the student is registered in the required requisite courses.

Next, the system will return the selected course’s sections, after which the student will select their preferred section. The system will verify if the section is available, and if so, will enroll the student. The system returns a confirmation message to the student when the process is completed.

**Operational Contracts:**

|  |  |
| --- | --- |
| Name: | Manually Adding a Course |
| Operation(s): | select\_course(course) |
| Cross Reference: | UC05 Add Course |
| Preconditions: | * User is logged into the system * At least one search result displayed after course search and filter |
| Postconditions: | * Course is added to student schedule |

|  |  |
| --- | --- |
| Name: | Select Course Section |
| Operation(s): | select\_section(section) |
| Cross Reference: | UC06 Change Course Section |
| Preconditions: | * User is logged into the system * At least one search result displayed after section search and filter |
| Postconditions: | * Section is added to student schedule |



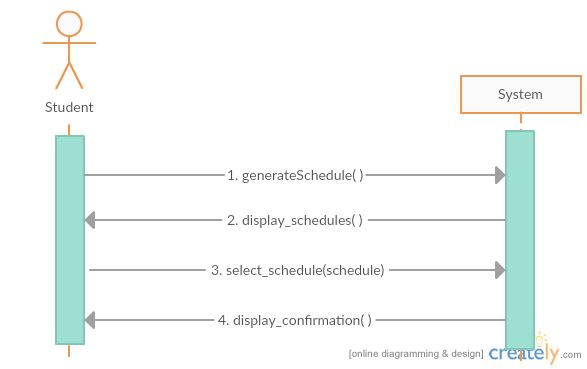
*Manually Registering a Course – Full Sequence Diagram*

This full sequence diagram shown above demonstrates the detailed interaction between every component involved in the system when manually adding a course.

The student first initiates the system, after which the interface retrieves the relevant courses from the course pool. The student then selects the desired course to be enrolled in from this list.

The system will then retrieve the student’s record from the student library and verify the course requisites in the backend. If all requisites are met, a confirmation of eligibility will be returned to the registration interface. If the requisites are not fulfilled, then the student will be prompted to select a class anew, until the conditions are met.

Next, the system retrieves the course’s sections for the student to choose from. The system verifies the availability of the section, and if it is available, the system modifies the student record to add the course. If the section is not available, the system will describe the issue and prompt the user to select a new section. When all is completed, a completion message is returned to the registration interface to be shown to the user.



*Auto-Generating Schedules – System Sequence Diagram*

This system sequence diagram demonstrates the interaction between the student and the system when automatically generating schedules. The user initializes the program, and then the system displays the possible schedules that satisfy the course sequence. The user then picks their preferred schedule, after which the system confirms enrollment in the selected courses.

There are not many steps involved in this process from the user’s perspective, which is why the diagram appears very simple. This is due to the fact that the majority of the functions involved in this process are within the system, which can only be demonstrated through the use of a full sequence diagram, as shown below.

**Operational Contracts:**

|  |  |
| --- | --- |
| Name: | Generate Schedule |
| Operation(s): | generateSchedule() |
| Cross Reference: | UC08 Generate Schedule |
| Preconditions: | * The user must be logged in * The system has access to the course list and schedules * System checks the pre-requisites/co-requisites for all relevant courses |
| Postconditions: | * Schedule for remaining courses is auto generated |

|  |  |
| --- | --- |
| Name: | Display Schedule |
| Operation(s): | select\_schedule() |
| Cross Reference: | UC011 View Saved Schedule |
| Preconditions: | * The user must be logged in * The system has access all created schedules |
| Postconditions: | * Schedule selected is displayed |



*Auto-Generating Schedules – Full Sequence Diagram*

This full sequence diagram shown above demonstrates the detailed interaction between every component in the system when generating schedules automatically.

The student must only initialize the generator, and the remainder of the operations are completed within the system. First, the generator retrieves the student’s record from the student library, containing the student’s program, past courses and credits. Next, the generator retrieves the appropriate course sequence for the student’s program from the course pool.

A possible schedule for the necessary classes are generated in the backend using an algorithm. This schedule is then displayed to the student through the interface. The system will then modify the student’s record to add the specified courses to be enrolled in. Once all is completed, a confirmation of enrollment is returned to the interface, which is then displayed as a message to the user.

# 6. Estimation

## Revised Project Estimates

This is a copy of the original Project Estimates table.

|  |  |  |  |
| --- | --- | --- | --- |
| Task Name | Duration (Number of Days) | Total Cost (Hours) | Starting Date |
| Deliverable 0 | 3 | 10 | 10-Jan |
| Deliverable 1 | 15 | 35 | 22-Jan |
| Deliverable 2 | 15 | 30 | 22-Feb |
| Deliverable 3 | 15 | 30 | 10-Mar |
| Basic Project Structure Discussion | 5 | 6 | 10-Jan |
| Initial Diagrams | 5 | 10 | 11-Jan |
| List of Features and Technologies Required | 10 | 10 | 14-Jan |
| Creation of Proper Programming Environment | 2 | 5 | 10-Jan |
| Implementation Discussion | 10 | 20 | 14-Jan |
| Use Cases | 10 | 15 | 26-Jan |
| User Interface | 10 | 15 | 26-Jan |
| User Testing | 20 | 65 | 28-Feb |
| Design of Script to Obtain Concordia Class Data | 5 | 6 | 02-Feb |
| ER Diagram Creation | 5 | 6 | 06-Feb |
| Architecture Programming, Use Cases | 20 | 90 | 19-Feb |
| Scope of Project | 60 | 20 | 08-Feb |
| Test Cases | 12 | 15 | 28-Feb |
| Performance | 8 | 20 | 08-Mar |
| Design of UI / Aesthetics of Application | 10 | 20 | 12-Mar |
| Debugging | 10 | 20 | 22-Mar |
| Total | 90 | 440 | N/A |

The following is a revised cost estimate for the project. Upon review of what has been completed by the Project team so far as well as challenges/risks encountered, the cost estimates have changed. In some cases, they have risen and in others they have dropped.

\*Numbers in Blue signify costs that have dropped, Numbers in Red signify costs that have risen

\*\*\*Highlighted Dates signify changes in start Dates

|  |  |  |  |
| --- | --- | --- | --- |
| Task Name | Duration (Number of Days) | Total Cost (Hours) | Starting Date |
| Deliverable 0 (Documentation) | 3 | 10 | 10-Jan |
| Deliverable 1 (Documentation) | 15 | 35 | 22-Jan |
| Deliverable 2 (Documentation) | 10 | 25 | 22-Feb |
| Deliverable 3 (Documentation) | 15 | 30 | 10-Mar |
| Basic Project Structure | 5 | 6 | 10-Jan |
| Initial Diagrams | 5 | 10 | 11-Jan |
| List of Features and Technologies Required | 10 | 10 | 14-Jan |
| Proper Programming Environment | 2 | 5 | 10-Jan |
| Implementation | 15 | 35 | 14-Jan |
| Use Cases | 10 | 15 | 26-Jan |
| User Interface | 10 | 15 | 26-Jan |
| User Testing | 15 | 55 | 28-Feb |
| Design of Script to Obtain Concordia Class Data | 1 | 3 | 02-Feb |
| ER Diagram | 1 | 3 | 06-Feb |
| Architecture Programming, Use Cases | 20 | 90 | 19-Feb |
| Scope of Project | 60 | 15 | 08-Feb |
| Test Cases | 12 | 15 | 28-Feb |
| Performance | 8 | 20 | 08-Mar |
| Design of UI / Aesthetics of Application | 10 | 15 | 12-Mar |
| Debugging | 10 | 30 | 22-Mar |
| Deliverable 4 : Final Report (Documentation) | 10 | 30 | 4-Apr |
| Total | 95 | 472 | N/A |

The changes to the Project Estimates table were made for the following reasons:

Deliverable 2 (Reduced from 15 days, 30 hours 🡺 10 days, 25 hours)

* The deliverable was completed in a shorter time period than expected by the team.
* This allows more time allocation to different tasks in the project

Implementation (Increased from 15 days, 20 hours 🡺 15 days, 35 hours)

* The implementation proved to be more complicated than previously expected, so more time was allocated to the implementation to allow the team to properly implement all functionalities.

User Testing (Reduced from 20 days, 65 hours 🡺 15 days, 55 hours)

* It was determined that since the user testing would only be started after the completion of all underlying functionalities (i.e course generator etc.), it leaves the team with less time to complete testing, since implementation will now take longer.
* This was not something the team wanted to do, but as we had no choice, the team will need to be very efficient and incorporate a thorough plan while completing the testing.

Design of Script to Obtain Concordia Class Data (Reduced from 5 days 🡺 1 day)

* The script was completed in a shorter time period than expected by the team.

ER Diagram (Reduced from 5 days 🡺 1 day)

* The script was completed in a shorter time period than expected by the team.

Scope Project (Reduced from 20 hours 🡺 15 hours)

* The team determined a thorough scope of the system fairly early within the project, so as time progressed not many changes were needed. The team would still periodically check over the course of the project.

Design of UI / Aesthetics of Application (Reduced from 20 hours 🡺 15)

* Through the use of bootstrap, the Design of the UI was completed in a shorter time period than expected by the team

Debugging (Increased from 20 hours 🡺 30)

* Because of time saved in the other tasks, more time was allocated to the debugging of the system. This was decided by the team we can make our system as efficient and easy to use as possible for the Actors (Administrator and Student)

Deliverable 4 : Final Report (Not previously Included 🡺 10 days, 30 hours)

* This part of the project was thought to be straight forward and just a combination of all previous Deliverables, but after contemplation it was decided that a lot of editing and re-structuring would need to be done in order to present the final product.

\*\*\*\* Estimates were made based on time required to create, as well as based on each team members’ individual work ethic. Based on the activities assigned to each member (and the artifacts said activities produce), an appropriate amount of time was estimated, taking into consideration the number of people striving to achieve said artifact, and the time they have available for said tasks. For all artifacts not yet created/achieved, re-estimation might occur if it is discovered that the time provided is inefficient for the members responsible to complete their task. In addition, the opposite can be said; re-estimation might occur if the task is easier than expected, thus saving time in that regard and allowing a reallocation of time to another issue. Re-estimation would also be required in the case that additional components not yet listed were designed and implemented. In such as a case, time would need to be allocated to these new tasks and the time provided to our sections might decrease as a result.

## Revised Schedule & Gantt Chart

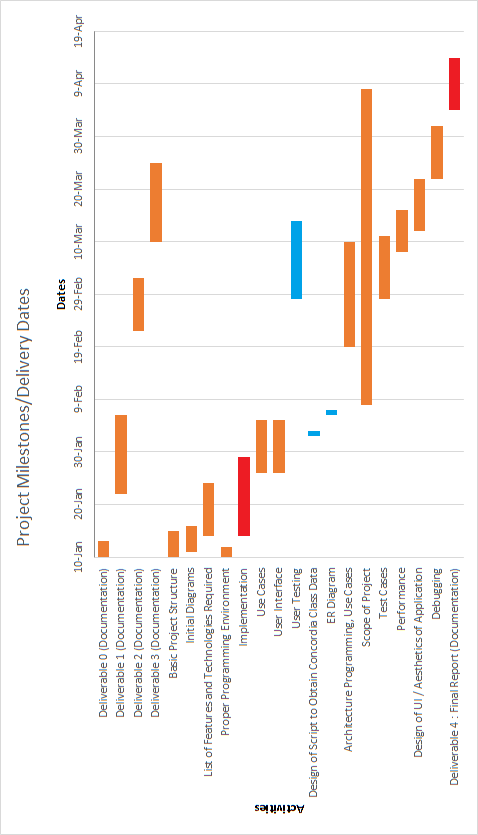
This is a copy of the original Schedule & Gantt Chart.



The initial Overall Man-hours for the project was 440. The revised Overall Man-hours for the project is now 472. This is an increase of roughly 7.2%. As a result of the changes mentioned previously, the team has updated the schedule to the following:

\*Bars in Blue signify time allocation that has decreased

\*\*Bars in Red signify time allocation that has increased



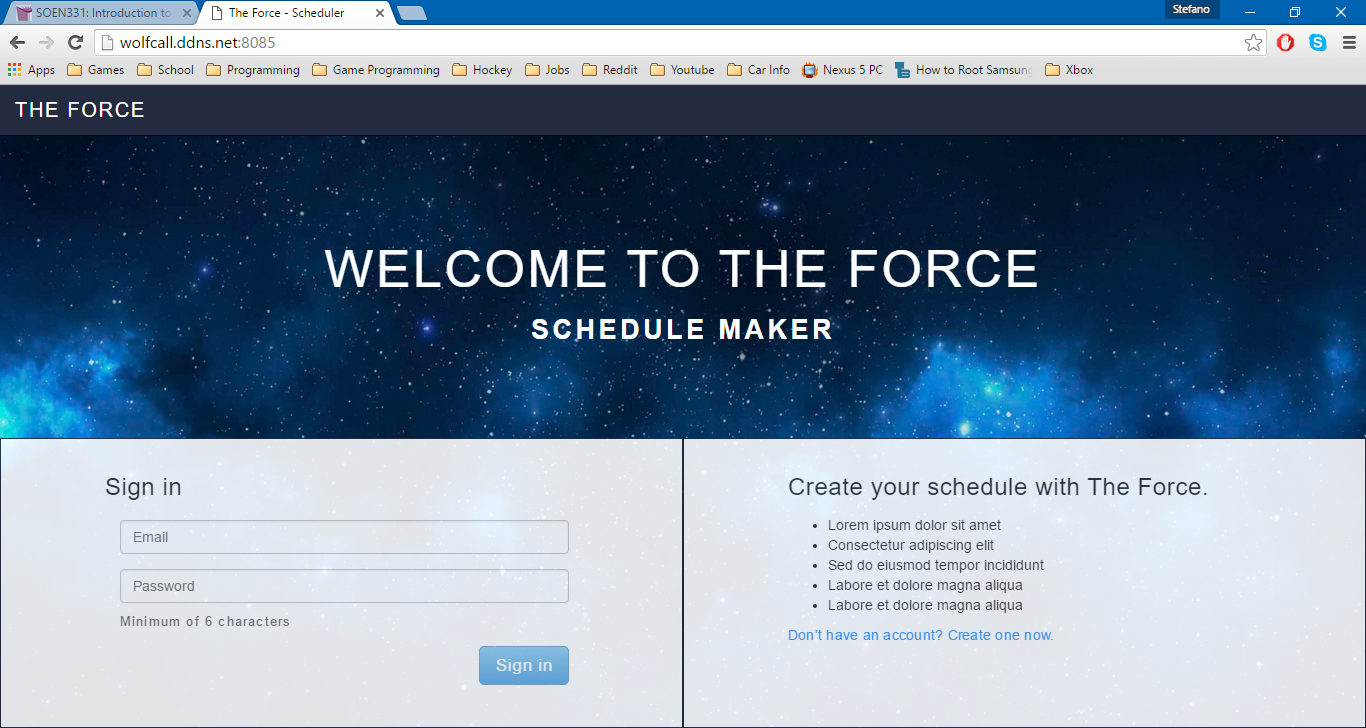
# Rapid Prototyping and Risk

## Prototype

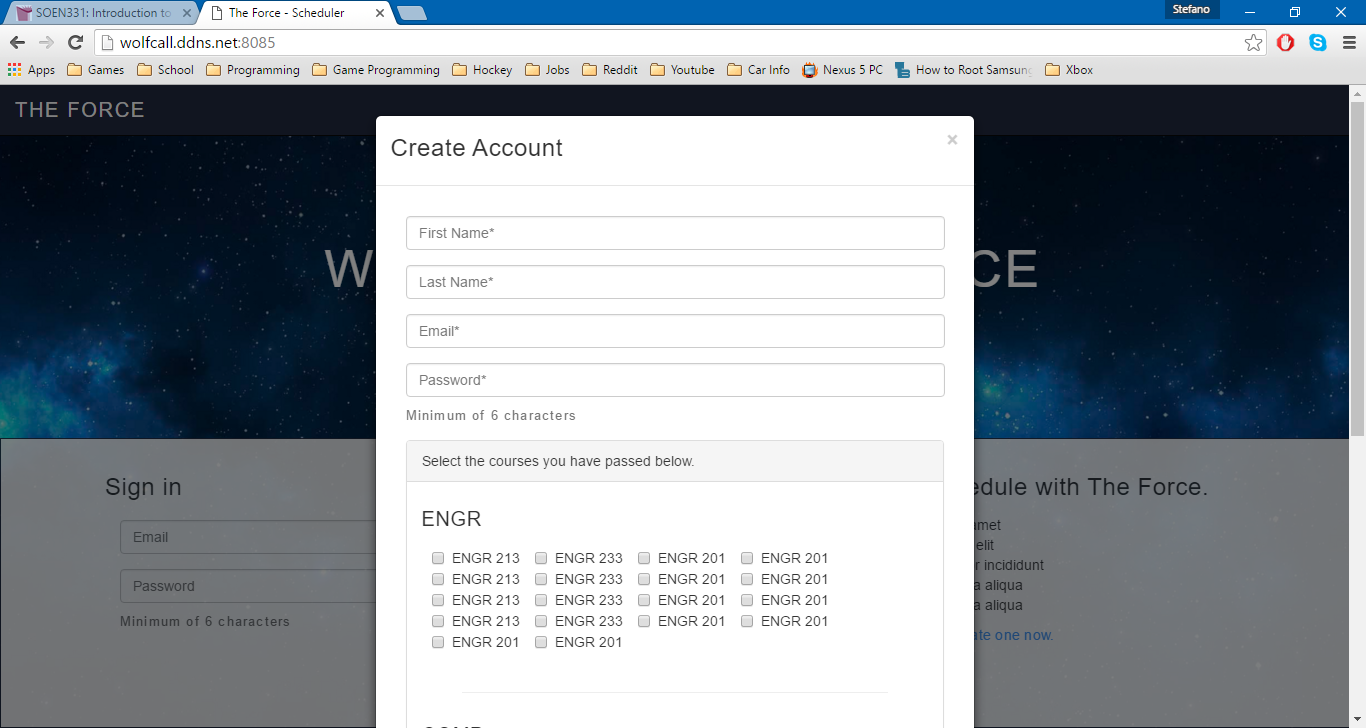
In order to accelerate the development of the project, continuous prototypes will be produced.

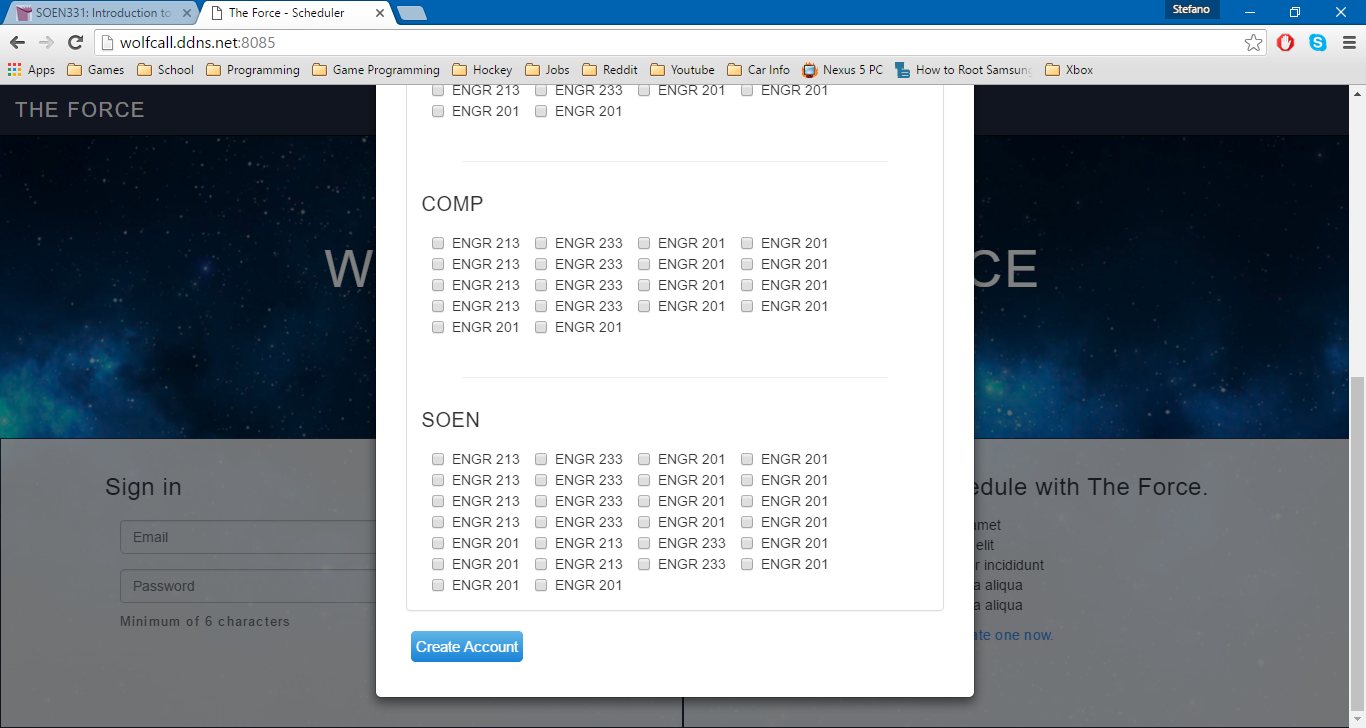
The rapid prototyping method will allow the programmers and designers to adapt and implement new ideas faster, as well as to accelerate the learning curve of the programming language (if any).

The following describes work undertaken in development of our second prototype “The Force”.

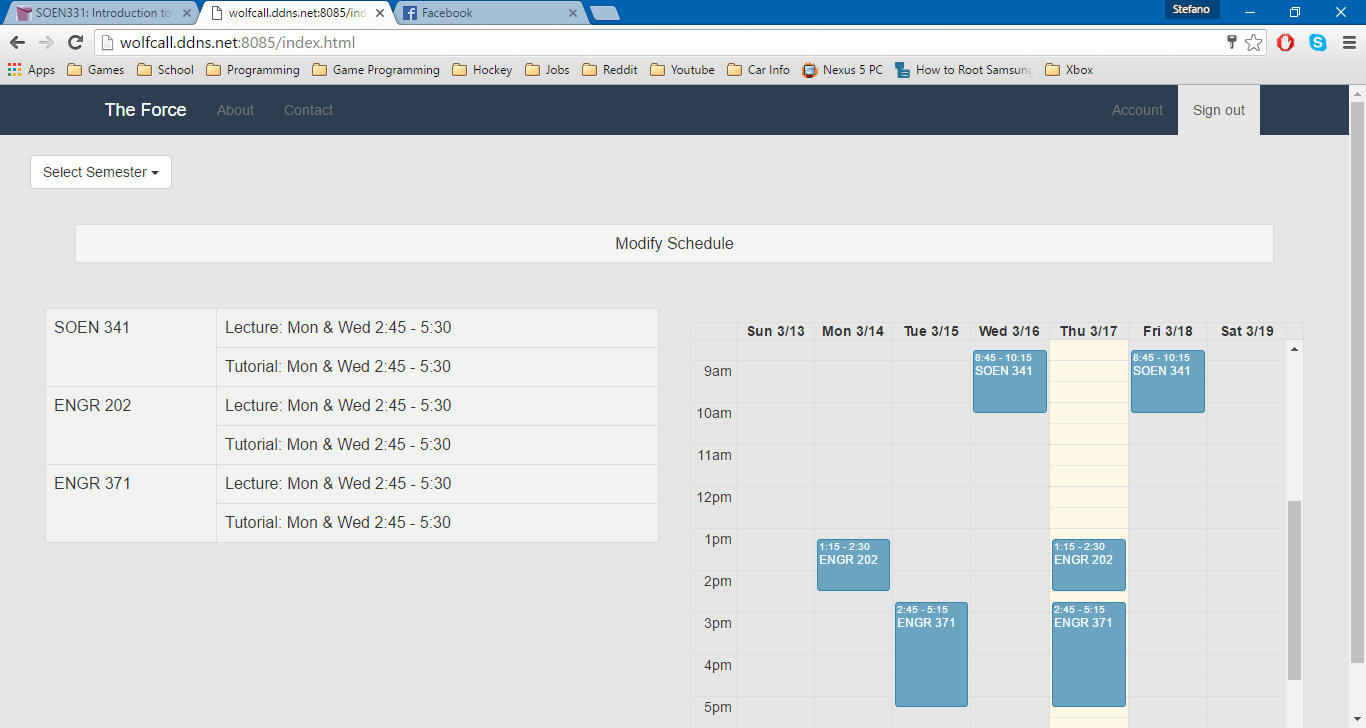


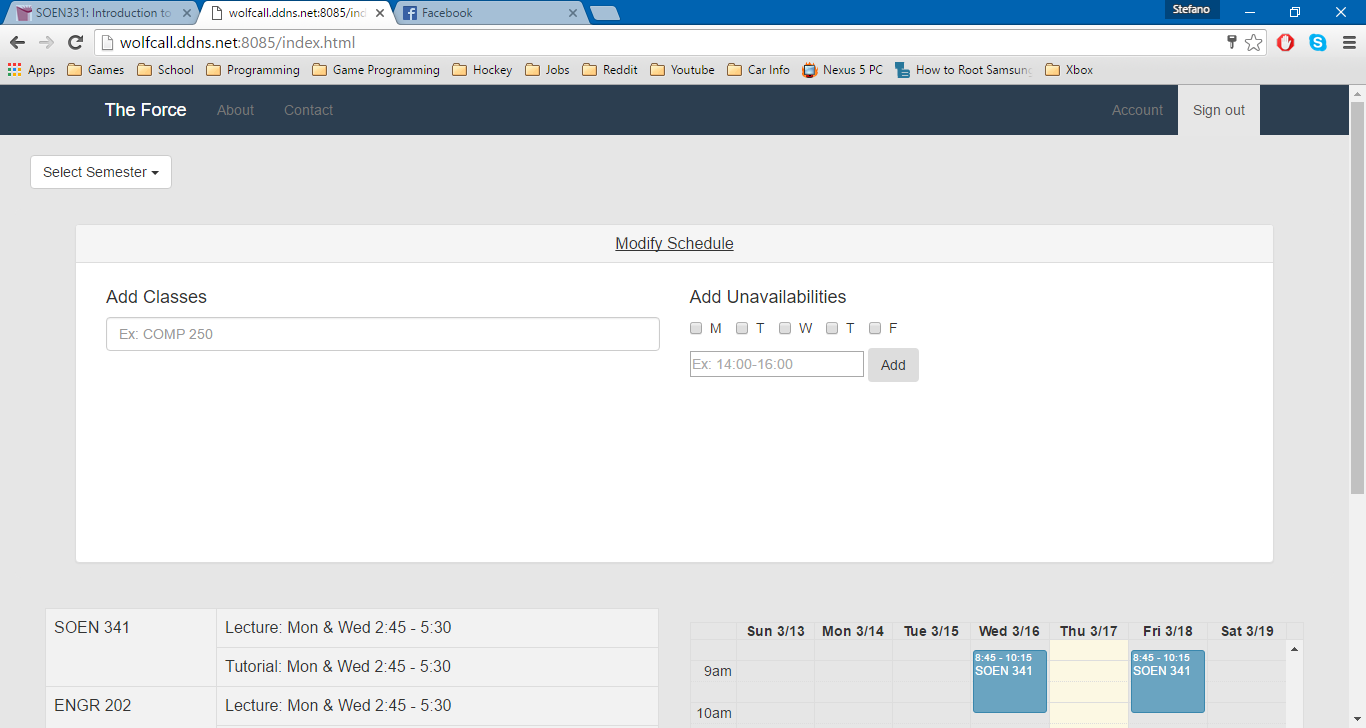
The screenshot above shows the updated home page of the website. Here, you will be able to sign in with the email you used when you created the account. If the information you entered does not match anything in the database, then you will not be able to sign in. It is suggested that you use the email that is associated to your Concordia Student account. If you do not have an account, you can create a new account.





If you select the “create an account” option, a dynamic window will appear. It will prompt the user to enter all pertinent information such as Name, Email, Password and Courses taken. To specify the courses they have taken and passed, they will need to check the boxes corresponding to the course. This way, they can be saved to the database and associated to the current student (see information on ID mentioned below). Once all information is entered, it will be saved to the database and the user will be redirected to the main page of the website.



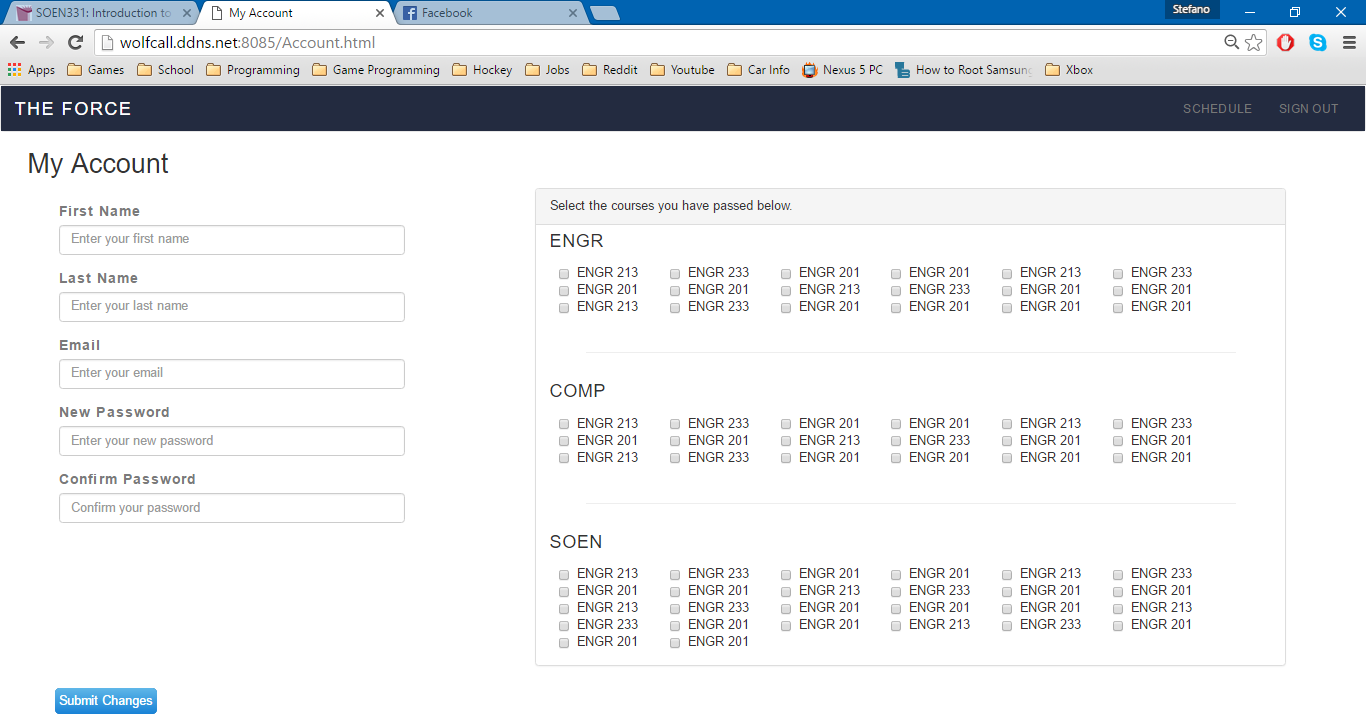


The screenshot above shows the main page of the website. Here, you can search for courses by entering the course code and add them to your schedule. You can specify the time and date constraints to your search so it will provide you with filtered search results. An autocomplete feature will kick in once the user begins to type a course code, allowing for fast filtering and searching.

The courses added by the user are added to database and are associated using an ID. Every user on the website is given a unique ID, so there is no way to display the wrong data.

By clicking on courses already added, you can also remove them from your schedule or browse for other sections available at different times. If the user tries to add a course that a student already has in their schedule, they will be prompted if they want to overwrite the course. If they select yes, the data will overwrite in the database. If they select no, then the data will stay the same.

The Header includes links to the “About Us” & “Contact Us” section of the site, which have not been developed yet. As well, there is a link to the “Account” section, which will be described below and an option to sign out.



The screenshot above shows the “My Account” portion of the website. Here, any student can view and edit any of the information associated to their account (mentioned above). If the password does not match, the user will be informed through a message on the screen. When the user changes any information here, it will be overwritten in the database.

If you would like to re-see a diagram establishing the connections between the database and front-end HTML pages (as described here) please refer to the diagram presented in section 3.1.

## Previously Identified Risks

Four risks were initially identified within the scope of this project.

The first was the Lack of Communication and Time Allocation. The concern was that since most of the members of the team have never worked together before, the overall efficiency would be slow at the beginning of the project while a proper team dynamic was formed. This proved to be a non-issue, as the team gelled quickly and it allowed the completion of the early tasks in shorter times than expected. As a result, the team decided to allocate more time to tasks deemed more challenging than initially expected (i.e. Implementation, Relearning required Languages). Time management remains extremely important to our team and it’s possible that some features will need to be left out due to lack of time.

The second risk was relearning required programming languages. The concern was that members of The Force, whether they had basic or advanced knowledge of web programming, were not practicing recently so they had to take some time to re-familiarize themselves. As mentioned above, since the early tasks were completed quickly, more time was allocated for this and it allowed everyone to regain a good understanding of web programming in the early goings of the project.

The third risk was the security of the server. The concern was that proper security protocols had to be applied. Without proper testing of its limitations, there was a fear it could eventually be revealed that the server did not store all the data required, or that certain requests might not be handled correctly. This would lead to the collapse of data management and as a result, of the entire project. With the work done so far, we have taken extra caution in ensuring that this does not happen, so there is no change with regard to this risk on the scope of the project.

The fourth and final initial risk was the miscommunication between the client and server side. The concern was that if the user would not properly register, then the entire system would stop functioning properly. Therefore all aspects of the system associated to the GUI’s (button’s not working, or information not being passed/retrieved from the database) need to be top priority. With the work done so far, the team has not encountered any problems with the communication between the client and server side, so there is no change with regard to this risk on the scope of the project.

## Newly Identified Risks

As we have continued our work on “The Force System”, the team has discovered three new risks that could potentially affect the final implementation of system.

The first new risk is the reduction of time allocated for User Testing. As mentioned previously, since the time allocated for User Testing was reduced from 20 days and 65 hours, to 15 days and 55 hours, there is a risk that the team might not complete all necessary testing before the final implementation of the system. This is due to the fact that underlying functionalities have taken longer than expected to incorporate into the system. To combat this risk, the team has prepared a thorough plan to ensure efficiency when completing the testing of the system.

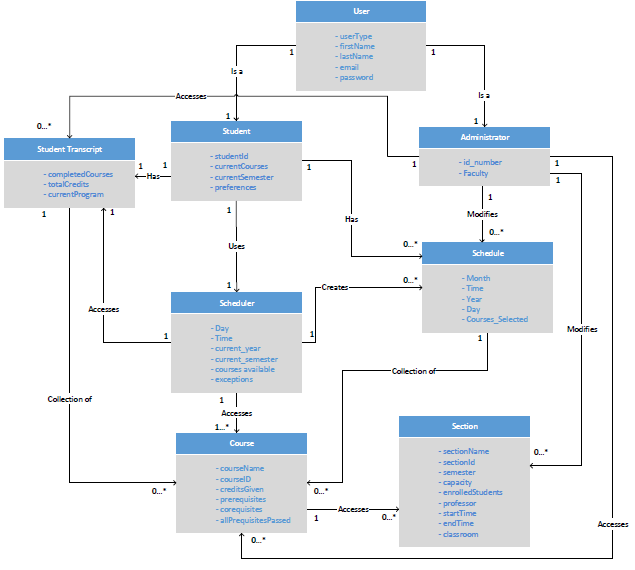
The second new risk is the addition of an Administrator to the system. The Administrator would have the power to search through any registered students schedule and make changes (i.e. course section). They would also have the power to change the details of the Students account (i.e. password, classes taken & pre-requisites).

The risk associated to the creation of this new Administrative user was ensuring that changes are not made to a student’s schedule which lead to time conflicts or errors due to lack of pre-requisites for course. To prevent this from happening, the team decided that they will give the Administrative user access to the see the Students list of all courses they previously took as well as create a function to see if an added course has a time conflict with any course already in the schedule.

The third new risk was the differentiation between the Student account and the Administrative account. We did not want the users to be able to create a new Administrative account from the log in page, so the team decided to that the administrator would be a protected account (By protected, it is meant that only system administrators would be aware of this account and it would be hidden from the students). This means that when the user “Clicks on Create an Account”, the only option to appear would remain Student account. However, although this is true if an Administrative user is signed, they can create more administrative accounts for others to use (if necessary). As a result, this no change with regard to this risk on the scope of the project.

The following diagrams will entail the modifications made to the system to include an administrator.

Here is the updated Domain Model to include Administrators:



User

* The ‘User’ class is an entity representing a user of the system. This class contains 5 attributes. They are userType, firstName, lastName, email and password. The email and password are used for logging into the website. The userType is what allows the server to distinguish between a Student User and an Administrative user. The firstName and lastName are self-explanatory.
* The ‘User’ class is related to one ‘Student’ class and one ‘Administrator’ class to represent that each user can be either a registered student at Concordia University or an Administrator at Concordia University.

Scheduler

* The ‘Scheduler’ class is an entity representing the key concept of the system, the schedule creation. The class contains 6 attributes; Day, Time, current\_year, current\_semester, courses\_available and exceptions. Day, Time, current\_year and current\_semester and variables that will be filled when a student add/drops a class in their schedule. Courses\_available are courses that will be given in a certain semester. Exceptions will be created on the user\_constraints entered by the user.
* The ‘Scheduler’ class is related to 0 or more ‘Schedule’ objects to represent that the scheduler can create one or more schedules for various semesters and years.
* The ‘Scheduler’ class is related to 1 ‘Student Transcript’ object to represent that the scheduler will consult the transcript of the student who is currently use it to see if they have the course pre-requisites/co-requisites.
* The ‘Scheduler’ class is related to 1 or more ‘Course’ objects to represent that the scheduler will access available courses to see if the scheduler can add them to the current students schedule.

Schedule

* The ‘Schedule’ class is an entity representing the schedule made by a user for a certain semester/year. The class contains 5 attributes: Day, Time, Month, Year, Courses\_Selected
* The ‘Schedule’ class is a subset of the scheduler is related to 0 or more ‘Courses’ to represent that a ‘Schedule’ is a collection of courses made by a ‘Student’.

Student

* The ‘Student’ class is a specialized case of the user. The class contains 4 attributes; studentId, currentCourses, currentSemester, preferences (All self-explanatory).
* Each 'Student' in the system is associated one 'Student Transcript'. This is the student’s academic history. Also, each ‘Student’ is related with 0 or more ‘Schedule’ which represent the user’s schedules for any academic year.

Administrator

* The ‘Administrator’ class is a specialized case of the user. The class contains 2 attributes; Id\_number and Faculty (All self-explanatory).
* The ‘Administrator’ can access zero of more ‘Schedules’, ‘Sections’ and ‘Courses’ of a particular 'Student' in the system to change them if necessary.

Student Transcript

* The ‘Student Transcript’ class is an entity representing courses that the ‘Student’ has already taken. The clas contains 3 attributes: completedCourses, totalCredits, currentProgram (All self-explanatory).
* Each ‘Student Transcript’ is related to 0 or more ‘Courses’ to represent the completed courses of a ‘Student’.

Course

* The ‘Course’ class is an entity detailing all the information about courses available to software engineering students during various years/semesters.

Section

* The ‘Section’ class is a subtype of ‘Course’. It contains all the information about availabilities (time slots) of particular courses as well as the professor who will be teacher that section of the course.

Here are the Use Cases that were updated to include Administrator:

|  |  |
| --- | --- |
| ID: | UC01 |
| Name: | Login |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 2/5 |
| Risk Assessment: | Low |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrator access to the site’s home page |
| Summary: | By using this feature, the student and administrator will be able to view their profile information and search through schedules to make changes or create new ones. |
| Preconditions: | The user must be on the login page |
| Basic Flow: | 1. The user provides a valid username and password 2. The system responds by checking the username and password 3. The system redirects the user to another page of the website |
| Post Conditions: | Success: The user will be brought to the home page of the website  Failure: The user will remain on the login page since he/she will be denied access. An error message will be displayed. If the user cannot figure out their password, they must go through the proper procedure to change it. |

|  |  |
| --- | --- |
| ID: | UC02 |
| Name: | Logout |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 2/5 |
| Risk Assessment: | Low |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to log off of their account in the system |
| Summary: | Any user can log out of the system to terminate their session |
| Preconditions: | User must have successfully logged into the system |
| Basic Flow: | 1. User selects “Log Out” 2. The System prompts the user to confirm the log out 3. The user confirms the log out 4. The System redirects the User to the home page |
| Post Conditions: | Success: User is brought back to the main page  Failure: User remains on the same page and an error is displayed |

|  |  |
| --- | --- |
| ID: | UC03 |
| Name: | View/ Edit Profile |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Low |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to view/edit their profile information |
| Summary: | By using this feature, the student will be able to view their profile information, including ID number, first name, last name, password and email and edit it if necessary. |
| Preconditions: | User must have successfully logged into the system |
| Basic Flow: | 1. User selects the “Account” section 2. System redirects user to their profile page 3. System displays all the user’s profile information 4. The user edits one of the displayed fields 5. The user selects the “save” option 6. The system prompts the user to confirm the changes 7. The user confirms the changes 8. The system redirects the user to the home page |
| Post Conditions: | Success: System redirects User to home page, changes are saved  Failure: User remains on current page, changes unsaved |

|  |  |
| --- | --- |
| ID: | UC04 |
| Name: | Reset Password |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | High |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to reset their password if forgotten |
| Summary: | The user can reset his/her password by providing their email address. An email will be automatically sent to the email address with instructions on how to reset the password. |
| Preconditions: | The user must be on the login page |
| Basic Flow: | 1. User selects the “reset password” option 2. System prompts the user for the email address associated to the account 3. User enters the email address 4. System confirms the email address is linked to an account 5. System send email to email address with instructions to change the password |
| Post Conditions: | Success: Email is sent to the user, user remains on current page  Failure: Email is not sent, error message is displayed, user remains on current page |

|  |  |
| --- | --- |
| ID: | UC05 |
| Name: | Add Course |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student, Administrator |
| Goals: | * Allow students and administrator to add specified course to a schedule |
| Summary: | Specified courses can be successfully added by the user and to a schedule. |
| Preconditions: | * The user must be logged in and on the home page * The system has access to the course list & schedules * The system has access to previously created schedules * The user must have the pre-requisites (System will check) |
| Basic Flow: | 1. User selects “add course” option 2. System prompts the user to enter a course 3. User enters the course they would like to add 4. System prompts the user to confirm the course 5. User confirms the course they would like to add 6. System checks the pre-requisites/co-requisites 7. System indicates if the add was successful |
| Post Conditions: | Success: Specified course is added to the schedule  Failure: Process fails, Error is displayed to the user |
| Notes | \*\*\*\*\*Registration constraints include the following:   * Timeslots for courses cannot overlap (minimum 15 minutes between back-to-back courses). * Students cannot register for the same course twice within the same semester. * Students cannot register for a course which they already completed. * Students cannot register for courses for which they have not fulfilled the pre-requisites/co-requisites |

|  |  |
| --- | --- |
| ID: | UC06 |
| Name: | Change Course Section |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to change the section of a course in a schedule |
| Summary: | Specified courses can change their sections in a schedule. |
| Preconditions: | * The user must be logged in and on the home page * The system has access to the course list & schedules * The system has access to previously created schedules * The Course must already be on the schedule |
| Basic Flow: | 1. User selects a specified course on the schedule 2. System displays alternate sections available for the course 3. User selects one of the other available sections of the course 4. System prompts the user to confirm the change 5. User confirms the change 6. System changes the schedule, then displays it |
| Post Conditions: | Success: Specified course section is changed  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC07 |
| Name: | Remove Course |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student, Administrator |
| Goals: | * Allow students and administrators to remove courses from a schedule |
| Summary: | Specified courses can be removed by the user from a schedule. |
| Preconditions: | * The user must be logged in and on the home page * The system has access to the course list & schedules * The system has access to previously created schedules * The Course must already be on the schedule |
| Basic Flow: | 1. User selects specified course 2. System displays the details of the course, as well as a “remove” option 3. User selects “remove” 4. System prompts user to confirm the removal of the course 5. User confirms the removal of the course 6. System removes the course, then displays the updated schedule |
| Post Conditions: | Success: Specified course is removed  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC08 |
| Name: | Generate Schedule |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 5/5 |
| Risk Assessment: | High |
| Actors: | Student |
| Goals: | * To allow students to auto-generate a schedule through the system |
| Summary: | The system will auto-generate a schedule for the user |
| Preconditions: | * The user must have logged in & be on the main page * The system has access to the course list & schedules * System indicates if the add was successful after checking the pre-requisites/co-requisites. |
| Basic Flow: | 1. User clicks on Generate course 2. System responds by creating a schedule |
| Post Conditions: | Success: Schedule is auto-generated  Failure: Process fails, Error is displayed to the user |
| Notes | \*\*\*\*\*Registration constraints include the following:   * Timeslots for courses cannot overlap (minimum 15 minutes between back-to-back courses). * Students cannot register for the same course twice within the same semester. * Students cannot register for a course which they already completed. * Students cannot register for courses for which they have not fulfilled the pre-requisites/co-requisites |

|  |  |
| --- | --- |
| ID: | UC10 |
| Name: | Save Schedule |
| Importance (/5): | 3/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to save a schedule for future access |
| Summary: | The schedules generated by the system and edited by the user can be saved as a preference |
| Preconditions: | * The user must have logged in & be on the main page * The user must have a schedule completed |
| Basic Flow: | 1. User selects “save schedule” 2. System prompts the user for confirmation 3. User confirms the saving of the schedule 4. System then saves the schedule and displays it |
| Post Conditions: | Success: Schedule is saved and added to database of the Student  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC11 |
| Name: | View Saved Schedule |
| Importance (/5): | 2/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student, Administrator |
| Goals: | * To allow students and administrators to view a saved schedules |
| Summary: | The user can load any schedule previously created for viewing purposes |
| Preconditions: | * The user must have logged in & be on the main page * The user must have access to a schedule saved * The system must have access to all saved schedules |
| Basic Flow: | 1. User selects “View Saved Schedules” 2. System provides the list of the schedules saved 3. User selects the desired one 4. System displays the selected schedule |
| Post Conditions: | Success: Schedule is displayed from the saved list  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC12 |
| Name: | Remove Schedule |
| Importance (/5): | 1/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Student |
| Goals: | * To allow students and administrators to remove a saved schedules |
| Summary: | The user can remove any schedule from the list of saved schedules. |
| Preconditions: | * The user must have logged in & be on the main page * The user must have a schedule in the preferred section * The system must have access to all saved schedules |
| Basic Flow: | 1. User selects “View Saved Schedules” 2. System displays the list of the schedules saved 3. User selects a schedule 4. System displays the schedule 5. User selects “remove” 6. System prompts user for confirmation to remove the schedule selected 7. User confirms the selection 8. System removes the schedule, redirects user to home page |
| Post Conditions: | Success: Schedule is removed from the saved list  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC13 |
| Name: | Modify Section Capacity |
| Importance (/5): | 2/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Administrator |
| Goals: | * Allow administrators to change the student capacity of a section |
| Summary: | The administrator can change the capacity of any section specified. |
| Preconditions: | * The Course and specified Section must exist * The system must have access to all courses and sections |
| Basic Flow: | 1. Administrator selects specified course 2. System displays the list of available sections 3. Administrator selects a section 4. System displays the section details 5. Administrator selects “Change Capacity” 6. System prompts Administrator for a new capacity 7. Administrator enters new capacity 8. System prompts Administrator to confirm the new capacity 9. Administrator confirms 10. System redirects Administrator to the home page |
| Post Conditions: | Success: Section Capacity changed  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC14 |
| Name: | Add Student |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Administrator |
| Goals: | * Allow administrators to select student in order to make changes to their schedule |
| Summary: | The administrator can change any specified student’s schedule. |
| Preconditions: | * The Student must have an account. * The Student must have valid schedules completed * The System must have access to the Students schedules * The Student must have the pre-requisites for the courses that the Administrator may add |
| Basic Flow: | 1. Administrator enters student name 2. System displays the list of students with the same name 3. Administrator selects the student 4. System displays the list of schedules made by the student 5. Administrator selects a schedule 6. System displays the selected schedule |
| Post Conditions: | Success: Selected schedule displayed to the Administrator, ready for change  Failure: Process fails, Error is displayed to the user |

|  |  |
| --- | --- |
| ID: | UC15 |
| Name: | Remove Student |
| Importance (/5): | 5/5 |
| Difficulty (/5): | 3/5 |
| Risk Assessment: | Medium |
| Actors: | Administrator |
| Goals: | * Once an administrators has modified a student’s schedule, they remove the student from their queue. |
| Summary: | The administrator removes the selected student from the queue after work was completed. |
| Preconditions: | * The Student must have been selected |
| Basic Flow: | 1. Administrator selects student 2. System displays the student’s details 3. Administrator selects “Remove Student” 4. System prompts the user for confirmation of the removal of the student 5. Administrator confirms the removal of the student 6. System removes the student from the queue and redirects Administrator to the home page |
| Post Conditions: | Success: Student removed from the queue  Failure: Process fails, Error is displayed to the user |

