CAPSTONE PROJECT
INTERMEDIATE REPORT II

Submitted to
Department of Computer Science
College of Computing Sciences and Engineering
Kuwait University

Advisor

Dr. Maha Alabduljalil

Group Members

AMNA ALMUTAIRI [2131110038]

ASMAA SULTAN [2151117051]

9 SEPTEMBER 2020

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

The vital presence of universities in our modern life dictates that we must make their experience as smooth as can be for our students and future workforce. From that belief, we picked up the initiative to create a student course advisory website specific to the department of Computer Science at Kuwait University.

The project includes developing a Django web-app named Scheduly. Scheduly is meant to deal with Computer Science students, instructors, and staff. What it does is provide those parties with a smart way to plan next semester course schedule. That is, to digitally communicate information between the faculty and the student such as list of courses students wish to take while checking required conditions are met such as eligibility of a student to take a course.

In this document, the progress is stated and a discussion about the decisions made regarding the project are presented. We will call the major sheet progression progress and the advised plan that will be produced by the system the plan.

The previous phase of the project included the criteria used to build this web-app. That included studying the features that best fit our department's way of setting courses and deciding the optimal way to plan following semester for every student out of the many possible ways.

In this phase, the progress made so far included implementing the functions handling the different-users of Scheduly such as the students, the instructors, and the admin to interact with the system and feed it with input and output, while sticking to their respective roles and authorizations. In addition, developing the python algorithm, which is important to the purpose of the project, the AI algorithm (Artificial Intelligence: A Modern Approach, 1994) to produce the academic plan for a specific student based on his or her previous academic progress. That is, student inputs the computer science courses he or she passed as input to be advised the courses that are recommended to be taken in future. Also, implementing the features for both the student and instructor to add courses to their wish list. And producing statistics based on the students plans and wish list inputs. The statistics which will be displayed after every modification will be visible for both instructor and admin. While wish lists of instructors is shown only to the admin. Further progress had been made including the change of the display of the website therefore a huge number of CSS modifications were made. This phase also included populating the database that interacts with the website to imitate real and accurate use of the website and its intended features. Finally, steps towards deploying the website to be open for public use through AWS (AWS Documentation, 2020) has been made but not finalized.

Next and final phase of the project includes but not strict to finishing up deployment of the web-app on AWS and improving the artificial intelligent python algorithm to help offer guidance even outside of the department courses.

2. DETAILED SYSTEM ARCHITECTURE

The system's external architecture is as seen in Figure 1

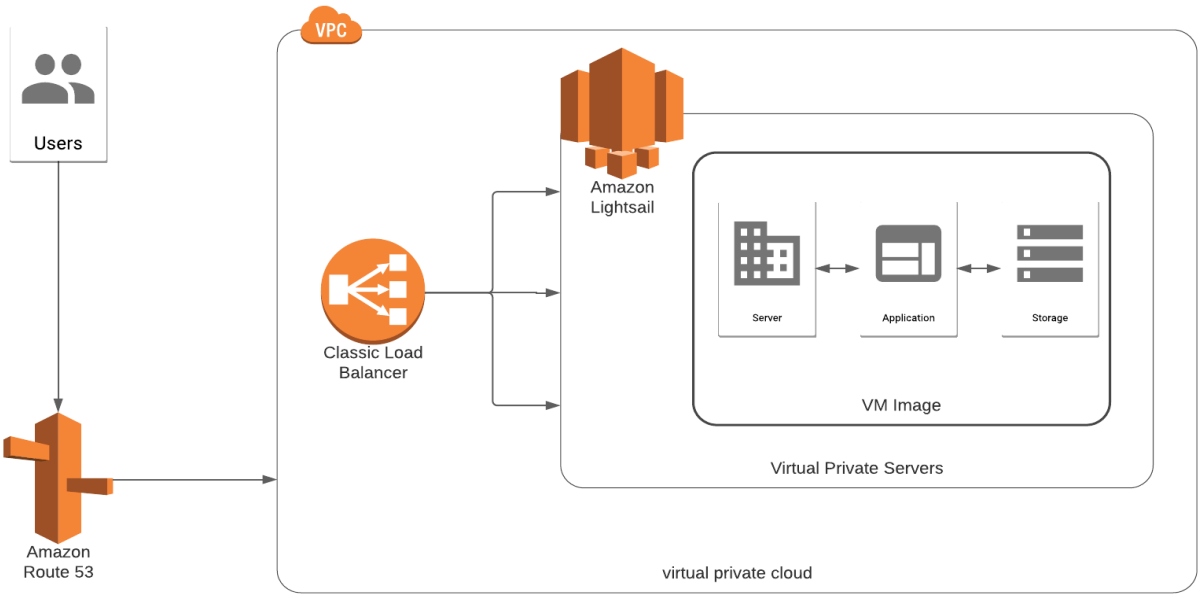

FIGURE 1: HIGH LEVEL SYSTEM ARCHITECTURE

The team used the cloud computing service Lightsail (Lightsail Documentation, 2020) which is provided by the Amazon Web Services (AWS Documentation, 2020). Which is a Virtual Private Cloud services platform. This services platform provided the team with a bundle of existing AWS products. Specifically:

3. Amazon Route 53: the highly available and scalable cloud Domain Name System web services which effectively route end users to the infrastructures running in AWS. And allowed the team to register the domain name (http://www.thescheduly.com). As well as the DNS service to route our domain name to the fixed and public static IP of our infrastructures. And finally, health checking where the service sends automated requests over the internet to our infrastructure to verify it is reachable, available, and functional.
4. Load Balancer: the service to distribute incoming web traffic among multiple instances helped the team increase availability and fault tolerance of the application such that if an instance is corrupted users are directed to other instances.
5. Lightsail instances: or the virtual private server, is the cloud virtual machines with preconfigured Linux enviorment. Which had preinstalled Bitnami Django stack from the Bitnami software library. Which provided the framework Django, Python, and the Web Server Apache. All needed for running the web-app inside the instances.

The client-server software architecture or the architecture within the virtual image of Lightsail instance is as seen from Figure 2. The web-app or the system handles user requests through the web browser. The web-browser requests from an Apache HTTP server through the user's browser and directs them to a worker process running the Django application. Database used was SQLite, the files that built the app's internal architecture written in Python.

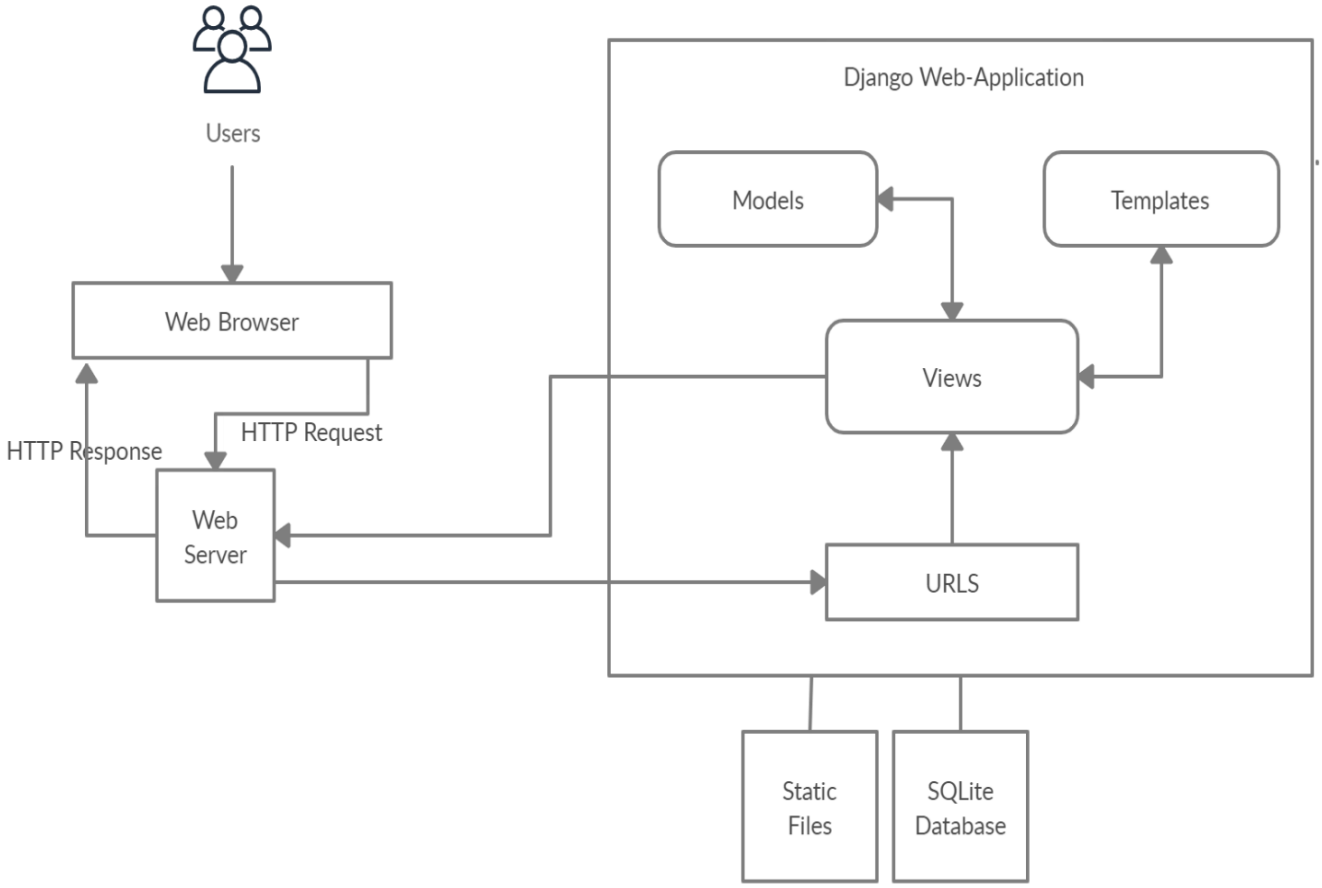

FIGURE 2: SYSTEM CLIENT-SERVER ARCHITECTURE

And finally, Figure 3 shows the component that classifies our group of classes for interchangeability and modularity of codes.

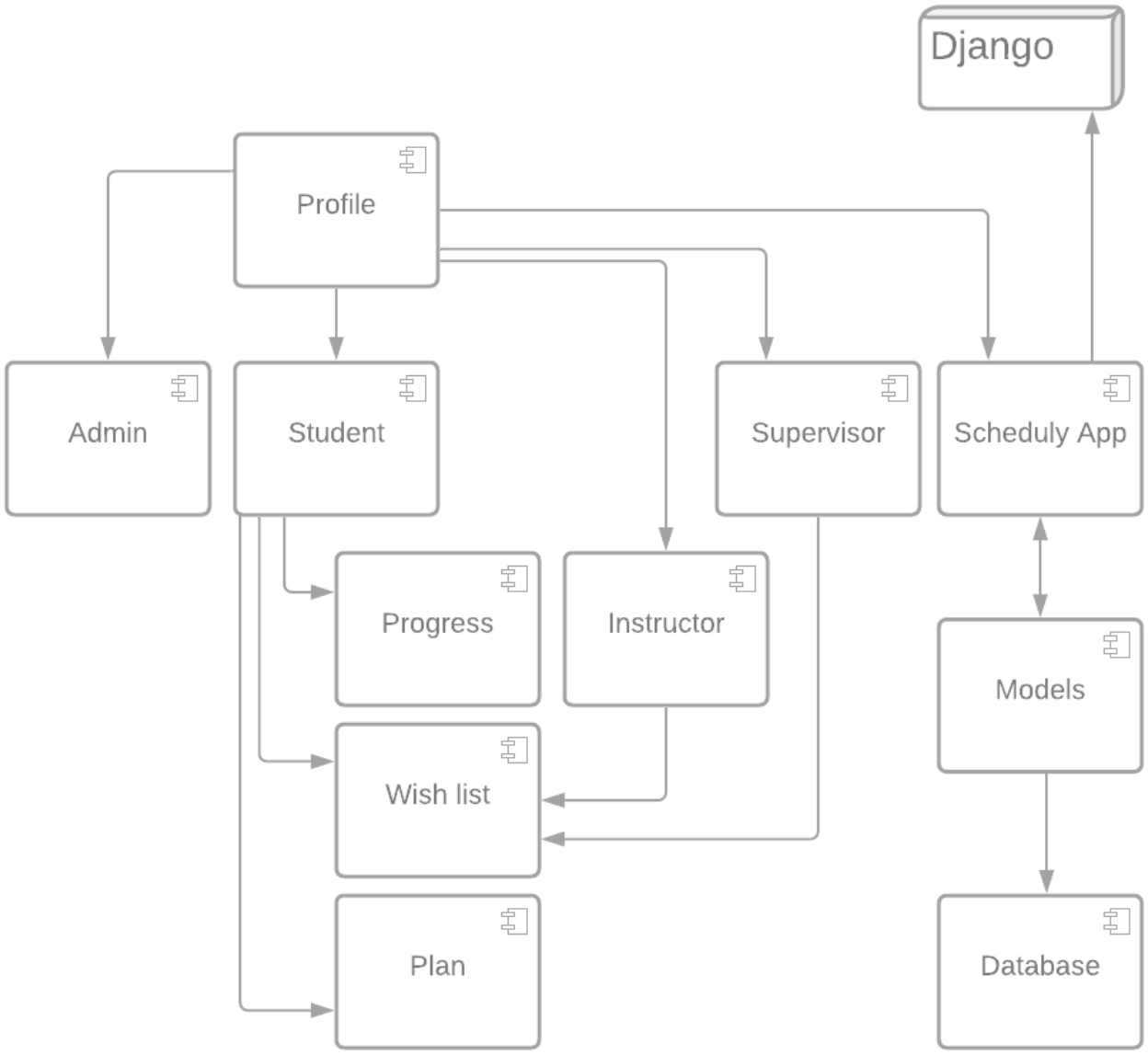

FIGURE 3: SYSTEM HIGH-LEVEL SOFTWARE ARCHITECURE

3. SYSTEM DESIGN

This section discusses the system design and to highlight the details and the component of the system. The internal architecture heavily relied on the Django architecture of Models-Templates-Views. With models, we manipulated most data relative to our users so we could process the many requests of our system such as students and their information, instructors and their information and courses and their information, the plan, the academic progress, the wish lists, and the relationship between them. Based on the entities defined with models the relationship between them and the web-app is reduced to HTTP requests and Django's framework.

The system used a web-app to interact between a student and their progress, plan, wish list. The instructor to interact with a wish list and statistics. The supervisor which is the special case of instructor who can view other instructor choices, to do so on the website along with the same instructor jobs. And finally, the admin to manage the users, courses, and sending emails to notify students when to add their choices to the system.

3.1 DEVELOPED ARTIFACTS

This section discusses the system design and to highlight the details and the component of the system. The internal architecture heavily relied on the Django architecture of Models-Templates-Views. With models, we manipulated most data relative to our users so we could process the many requests of our system such as students and their information, instructors and their information and courses and their information, the plan, the academic progress, the wish lists, and the relationship between them. Based on the entities defined with models the relationship between them and the web-app is reduced to HTTP requests and Django's framework.

The system used a web-app to interact between a student and their progress, plan, wish list. The instructor to interact with a wish list and statistics. The supervisor which is the special case of instructor who can view other instructor choices, to do so on the website along with the same instructor jobs. And finally, the admin to manage the users, courses, and sending emails to notify students when to add their choices to the system.

3.2 DESIGN DECISIONS

For a student, the login page is the first step of interacting with our system. The login needed from the student is their email and password. Following that, the student is presented the following actions: 1) Edit their profile information 2) view or update their academic progress 3) view their suggested plan 4) view or edit their elective courses wish list. The information the student can edit is restricted to the email and password since other information such as ID number or year cannot be decided by the student. The student's academic progress is left with the student as a form that contains all Computer Science department compulsory courses and math courses. That is because there are many courses outside this department that do not have dependencies that could affect the student's long-term progress such as the math and compulsory courses. Computer Science elective courses were kept off the student's progress as well for the same reason. For the student's plan, the student is presented with a table that suggests the courses from the next semester of the student, so their future academic progress is optimal. For the student's wish list, the student is presented with the page that displays all Computer Science department elective courses and then they add it to their own wish list.

For an instructor, the login process as well as editing profile is similar to that of the student. But the profile of the instructor included their respective un-preferred times of teaching which they can edit anytime. The instructor also has a wish-list page that shows both compulsory and elective Computer Science courses so the instructor can view or edit it. And a statistics page that shows the courses and statistics regarding number of students who have each course added to the students' wish-lists.

For the supervisor, the same as the instructor with an extra page of viewing other instructors statistics such as their un-preferred times of teaching, the courses and the number of instructors who have the course added to their wish-list.

For an admin, the login process is similar to the previous users. The admin has pages to manage the system such as adding the users or deleting them. The admin also can add courses or edit them. And a page that allows the admin to change the wish list allowed time which also automatically sends out emails for the students and instructors to inform them of this change. The admin is also tasked with assigning the supervisor role to an instructor or removing it.

For web-app interface is designed simply to provide formality to the academic purpose it served. Also, with ease of use in mind considered, as all the actions and pages are easily accessed from a navbar.

3.2 ENTITY-RELATIONSHIP MODEL AND DIAGRAM

For our data and database, the entity-relationship model was used to implement the information needed for and from the users as single model or entity and the relations between them as one to one relationship, one-to-many relationship, and many-to-many relationship, or another entity in some cases.

The following Entity Relationship Diagram shows how the data was manipulated by the web-app. The entities Student and Instructor were linked with the build-in Django user profile. While the instructor types of normal instructor, supervisor, or admin, were merged in the model Instructor. The course entity has three relationships vital to the algorithms such as 1) Dependency, which is a many-to-many relationship to implement prerequisites of the course as another entity. 2) Progres-Course, which is the many-to-many relationship to prove the student has passed a course or not, while having the attribute credit that helped our algorithm suggest the credits the student needs to take outside the Computer Science compulsory courses. 3) WishList-Course, which is the many-to-many relationship to link each course and the wish-list it was added to by either instructor or student, The wish-list entity has the role of linking the owner of that wish-list and the owner either student or instructor. While the subject entity is used for algorithms as another table to get results from without overlapping with the courses entity.

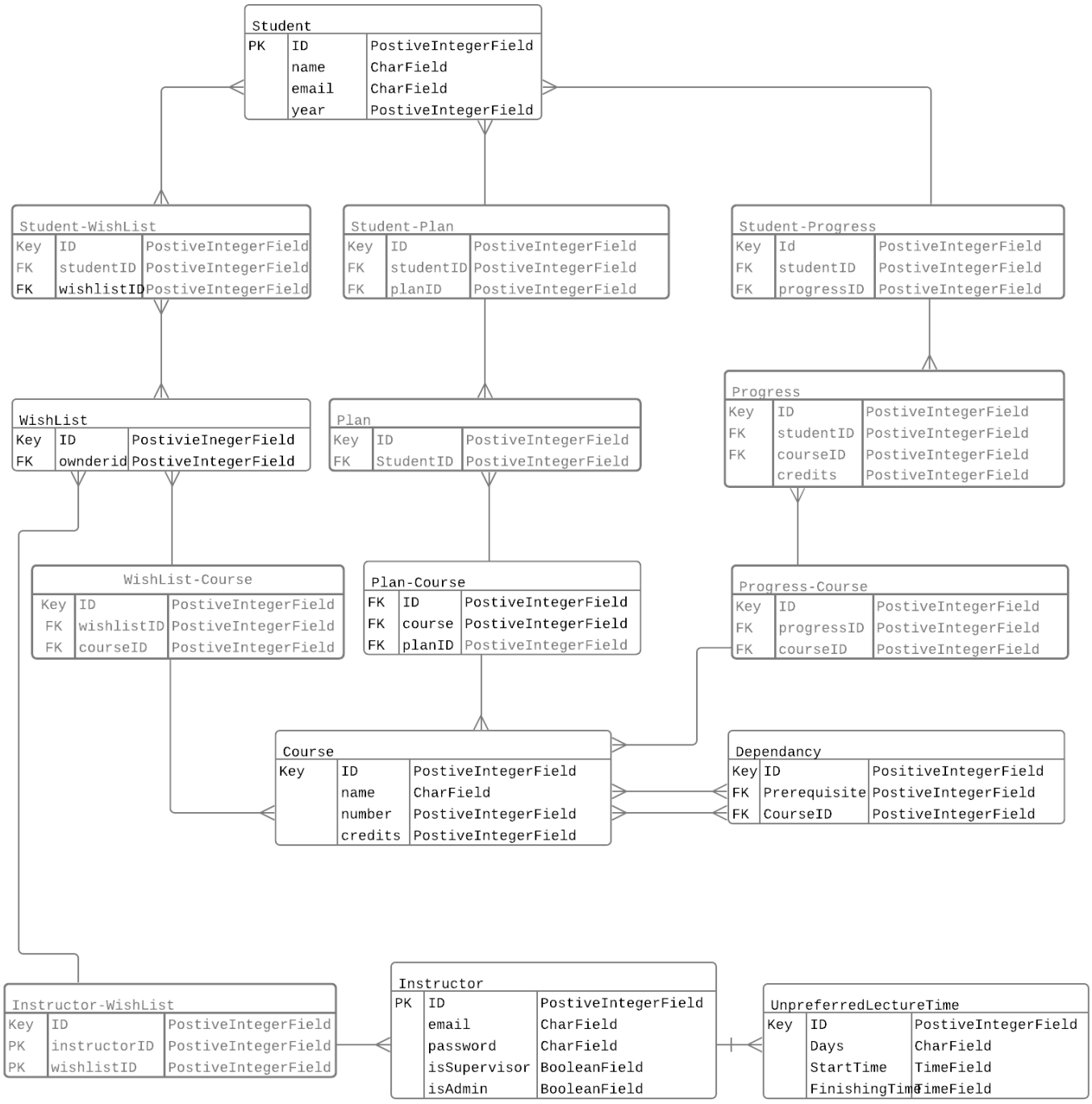

FIGURE 4: ER DIAGRAM

3.3 CLASS DIAGRAM

The following conceptual class diagram shows the blueprint for the objects that the system uses.

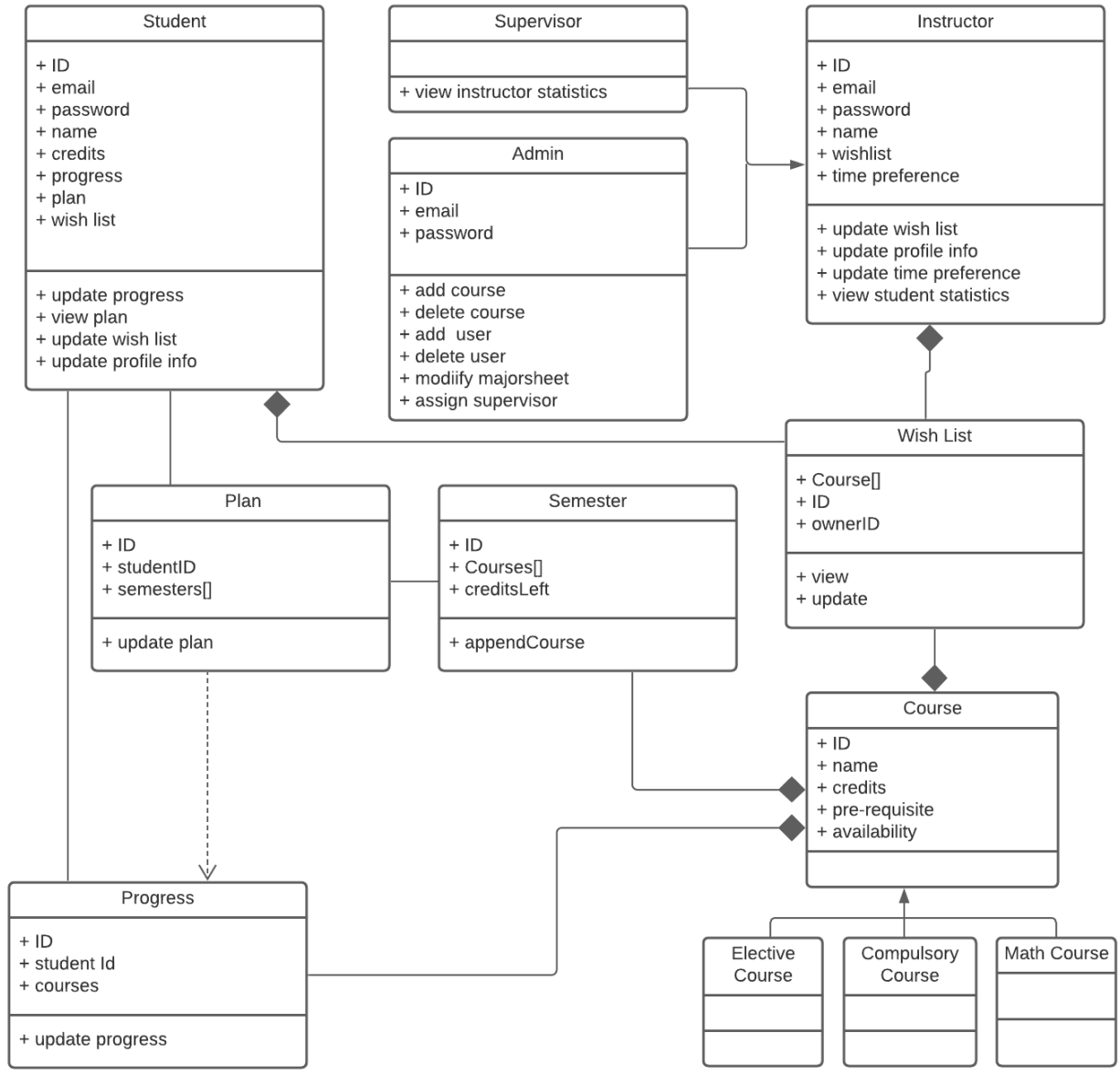

FIGURE 5: CLASS DIAGRAM

For the different type of users, we have four types of classes, the student, the instructor, and the two extensions of instructor: the supervisor and the admin. All mentioned users have a wish list that is composed of courses. While student has classes plan and progress with the plan having dependency with the progress. The course has three different type of courses that inherit from it: Compulsory Course for Compulsory courses in the Computer Science department. Elective Course for electives from the same department, and finally compulsory math courses from the math department.

3.4 USER INTERFACES

In this section we show pages the team developed for the web-app interfaces. Design decisions of the interface varied such as simplicity fit for the academic usage and to include ease of access. The interface design followed the eight golden rules of interface design (Shneiderman, 2010) including:

- Consistency: the design is consistent through every page in the web-app
- Seek universal usability: the diverse users from new to returning can easily find their way around the web-app.
- Offer informative feedback: all frequent and minor actions in the system have an appropriate response.
- Design dialogs that yield closure: for sequences of actions a notification of completion is offered.
- Permit easy reversal of actions: errors can be undone to allow all users to have efficient use of the system and encourage exploration of unfamiliar options.
- Keep users in control: the interface responds to user actions and stay consistent with familiar behavior.
- Reducing short-memory load: each functionality is given its own page and can be reached from the navbar so it can be easily accessed

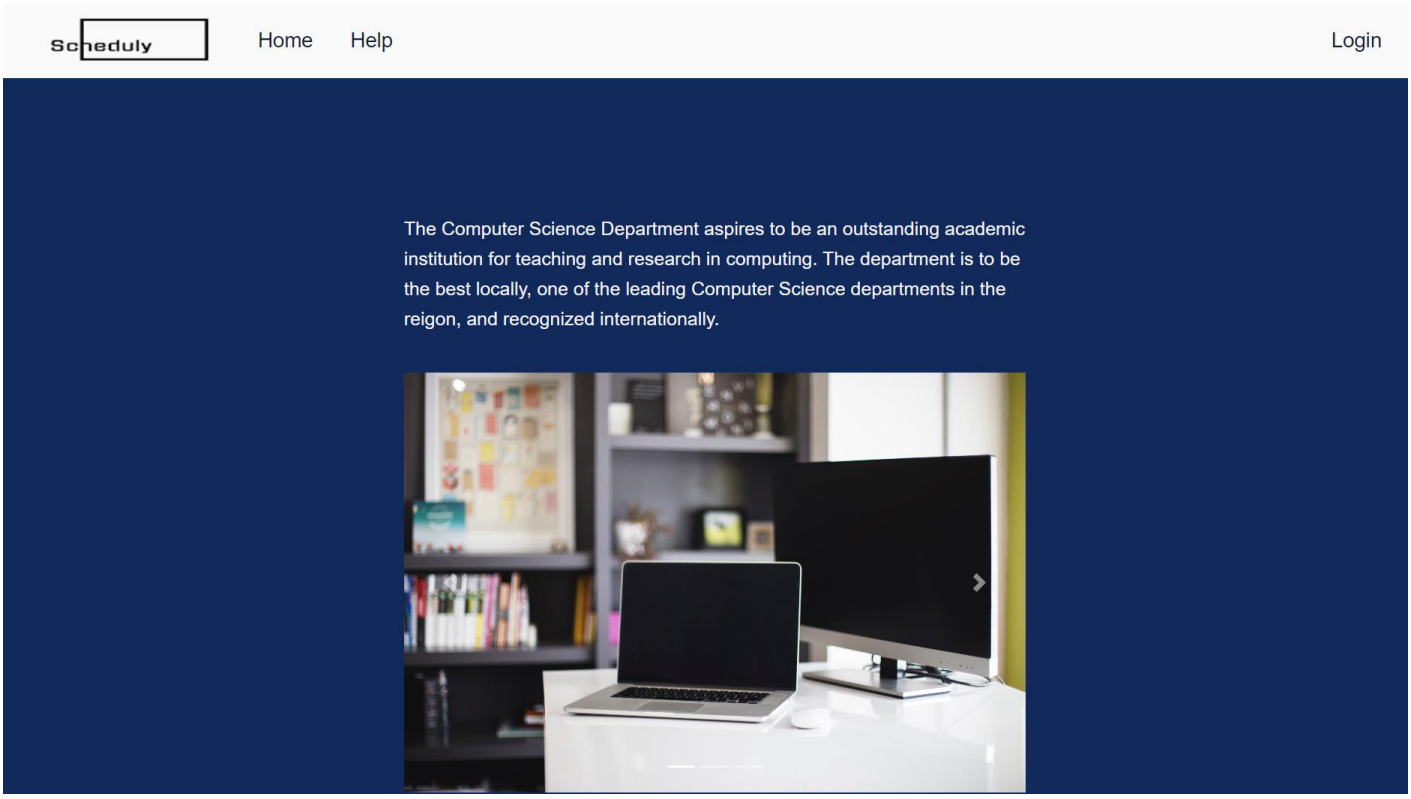

FIGURE 6: HOMEPAGE FOR VISITORS

Figure 5 shows the homepage from a new user or a logged-out user's point of view.

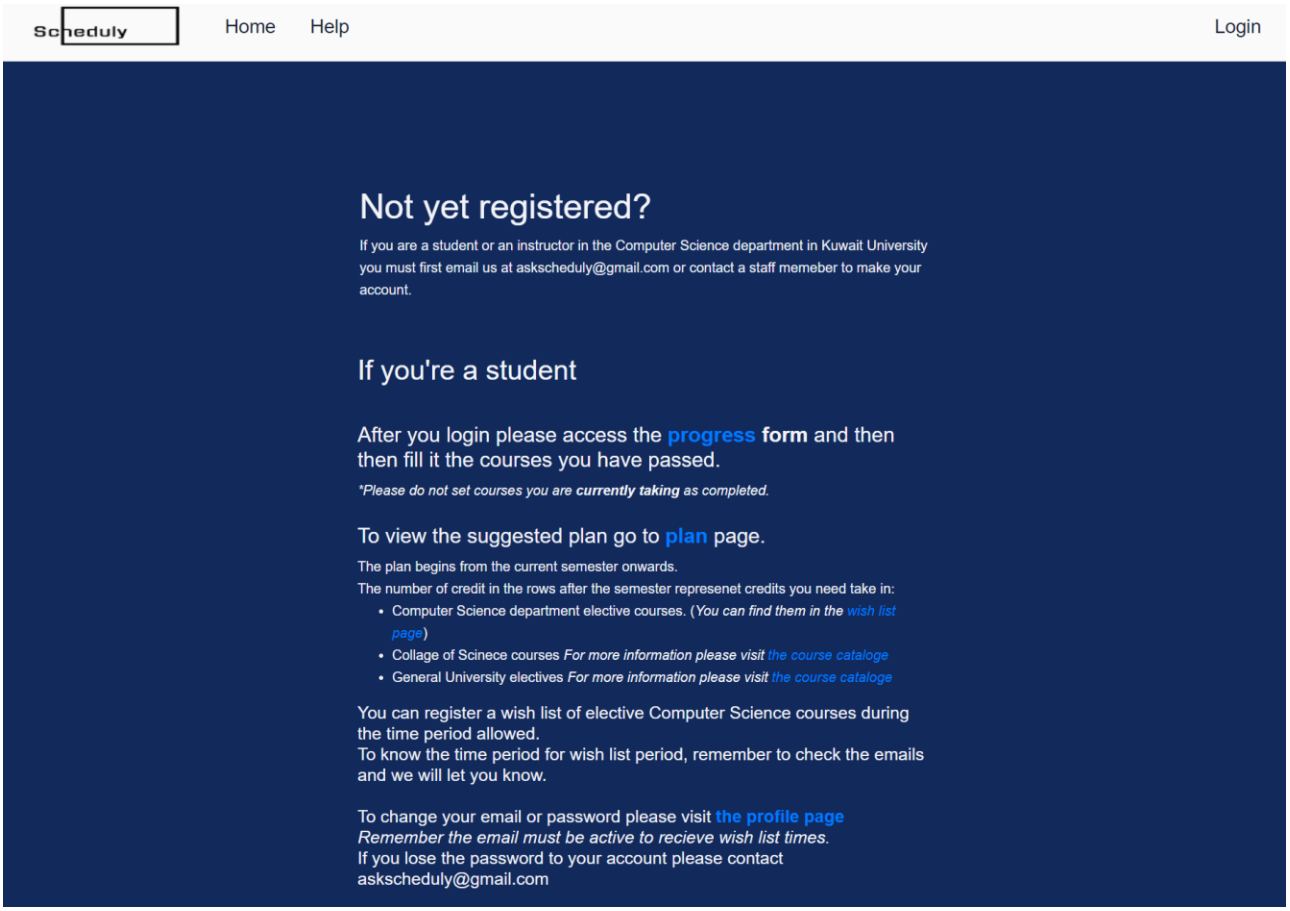

FIGURE 7: HELP PAGE

Visitors and students can access a help page from the homepage or any other menu bar that directs them to system features and offer guidance regarding how to use and what to expect when accessing the features.

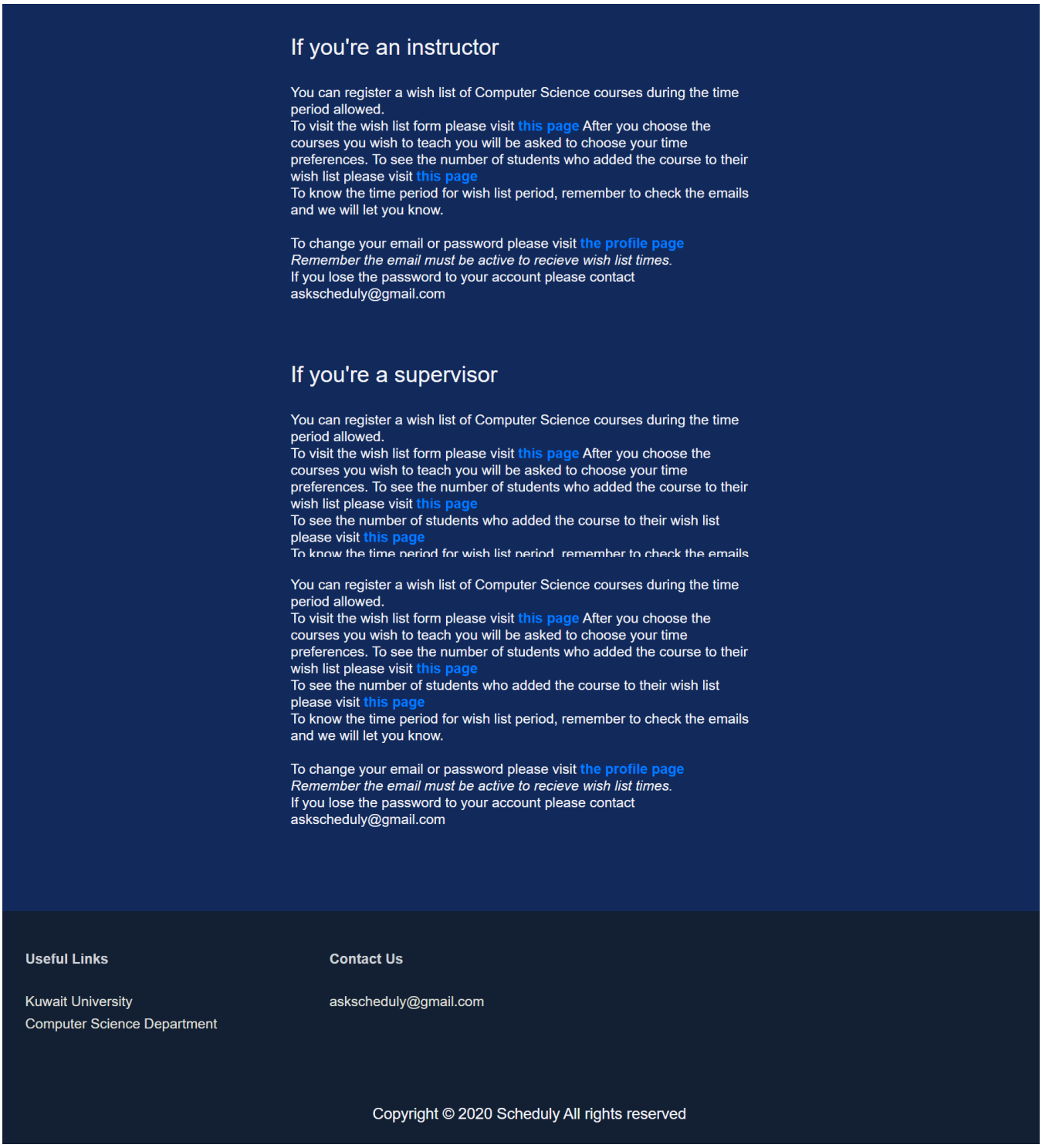

FIGURE 8: HELP PAGE

Instructors and supervisor can access a help page from the homepage or any other menu bar that directs them to system features and offer guidance regarding how to use and what to expect when accessing the features.

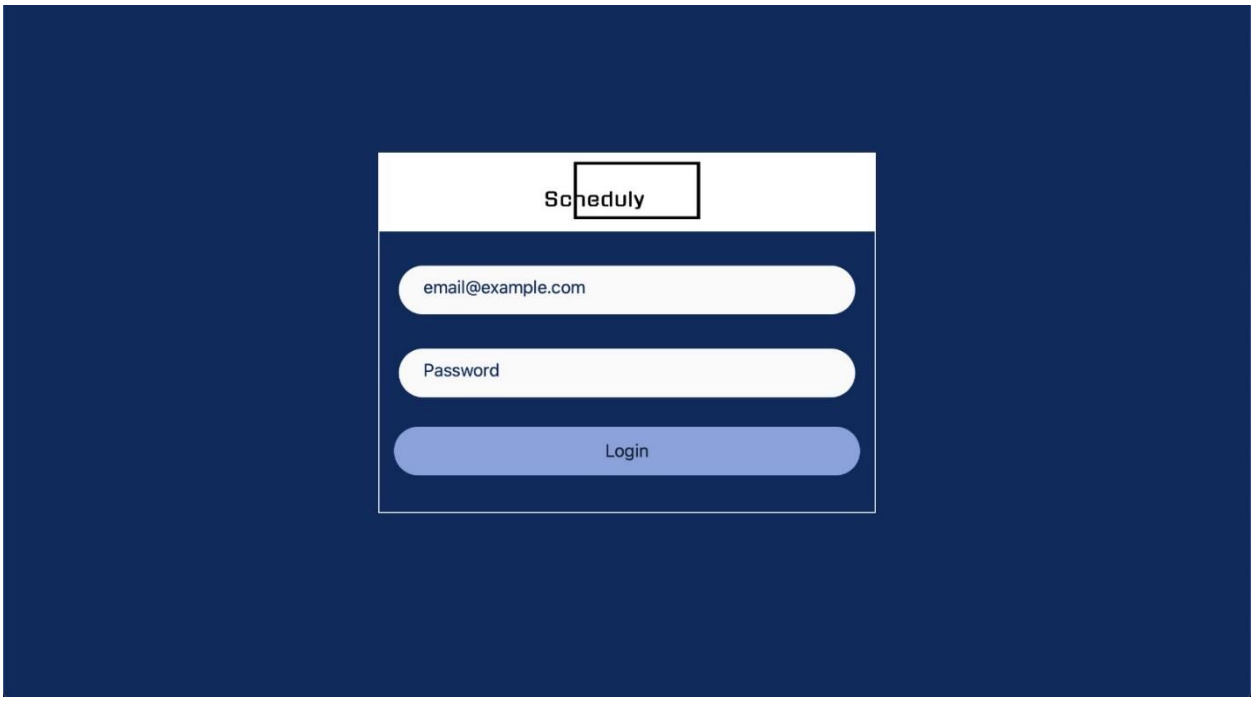

FIGURE 9: LOGIN PAGE

The login information needed to access the system is the email (university or personal email) and password that users get from admin or staff.

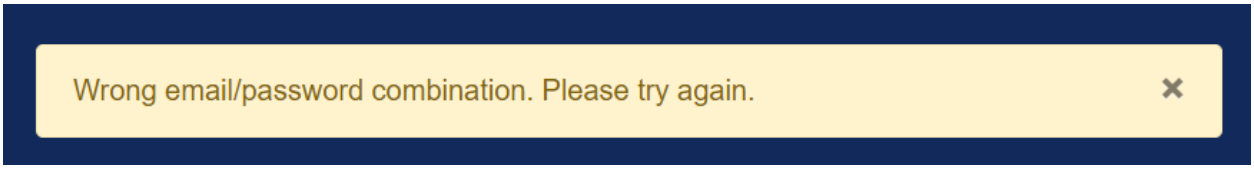

FIGURE 10: INFORMATIVE FEEDBACK #1

FIGURE 11: INFORMATIVE FEEDBACK #2

Login errors are handled and communicated to the users of the system as well as system status and action closures.

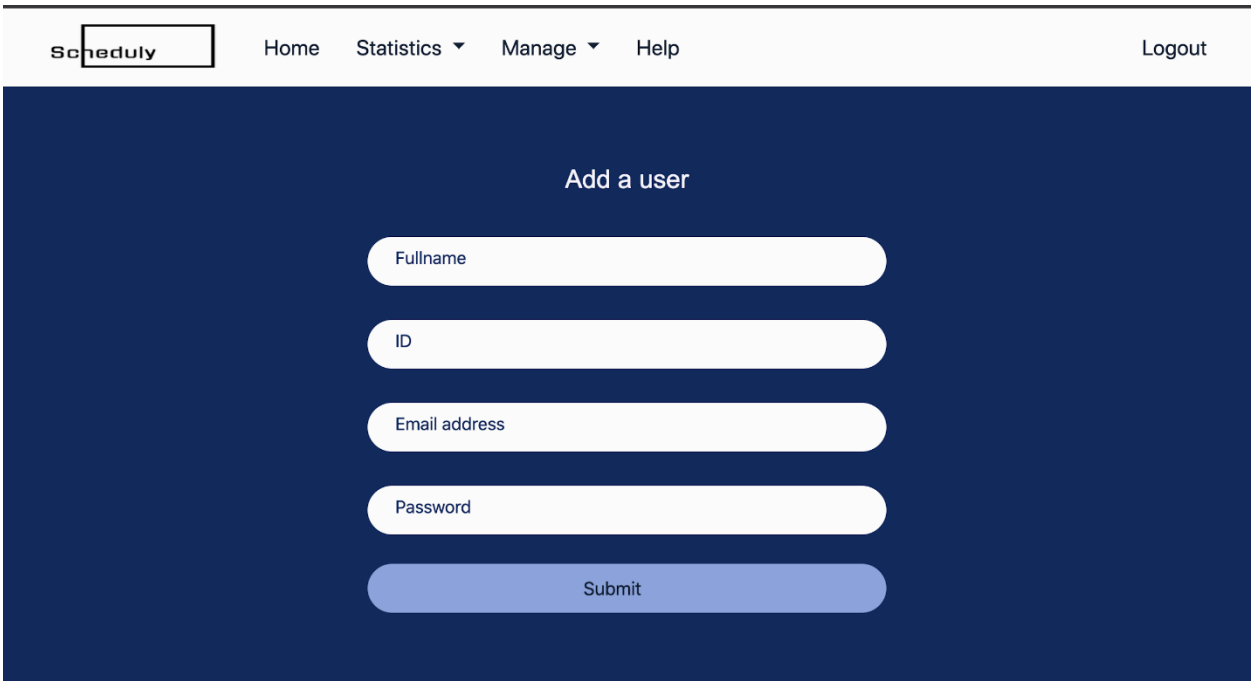

FIGURE 12: ADMIN PAGE TO ADD A USER

For the admin to add a user the following information is needed: their name, their unique university ID, and their unique email address. Following that, the admin can assign a password that the users can change in their respective interfaces.

FIGURE 13: INFORMATIVE FEEDBACK #4

FIGURE 14: INFORMATIVE FEEDBACK #4

Usage errors are handled and communicated to the admin of the system as well as system status and action closures.

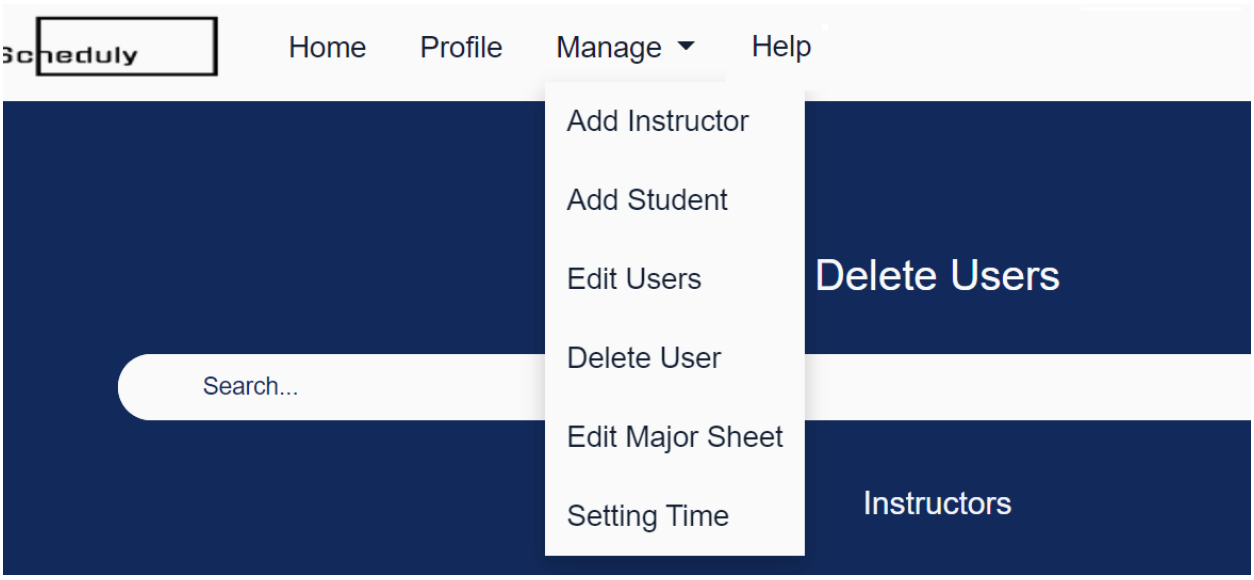

FIGURE 15: ADMIN MANEGEMENT MENU

The menu with the pages that the admin uses. The editing users, deleting users, and editing major sheet pages all include a search options due to the large number of options there is.

FIGURE 16: ADMIN PAGE TO EDIT REGISTERED USERS

The admin can view the list of registered users (both instructors and students) where the admin can find the selected user profile to be edited.

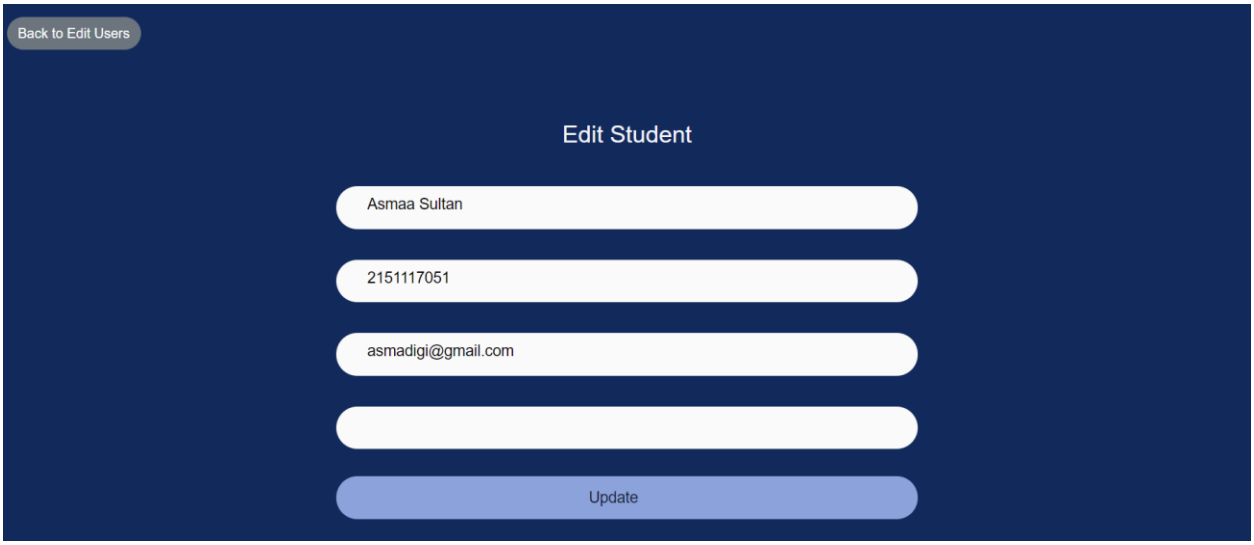

FIGURE 17: PAGE TO EDIT A STUDENT

The page where the admin can edit a student profile has the option reverse the selected action.

FIGURE 18: PAGE TO VIEW LIST OF USERS TO DELETE

The admin can view the list of registered users (both instructors and students) where the admin can find the selected user profile to be deleted.

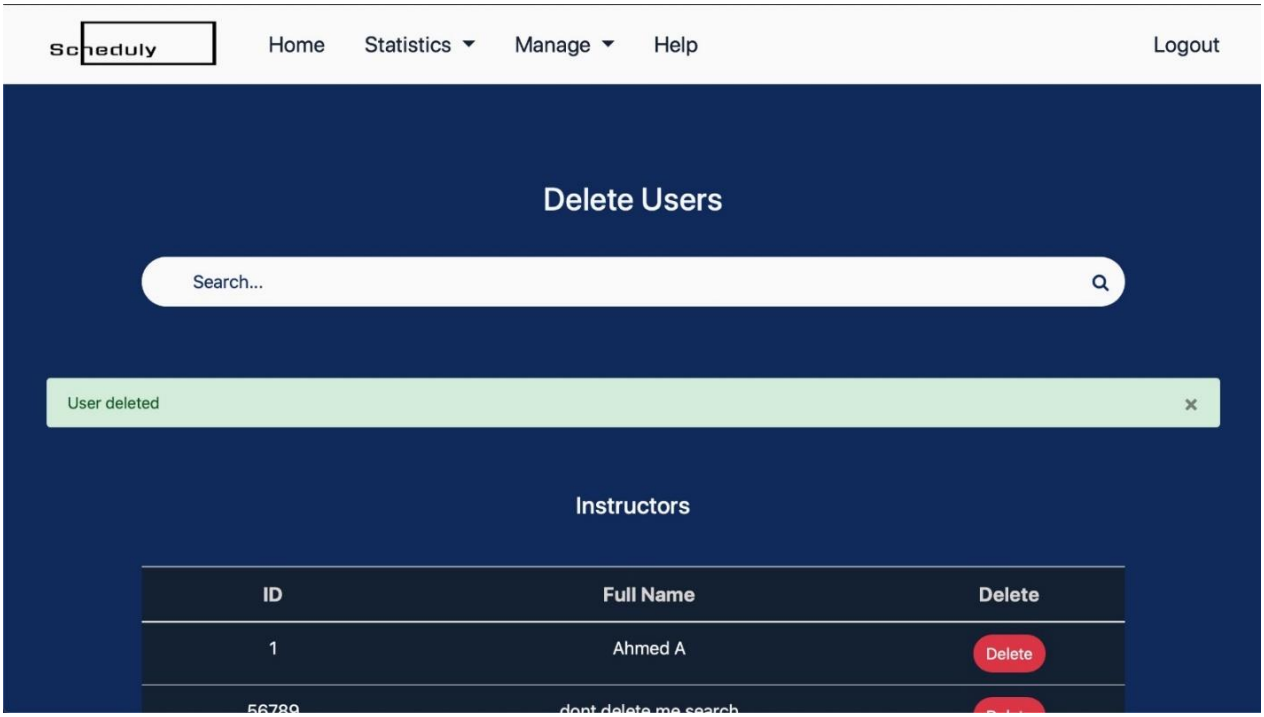

FIGURE 19: INFORMATIVE FEEDBACK #5

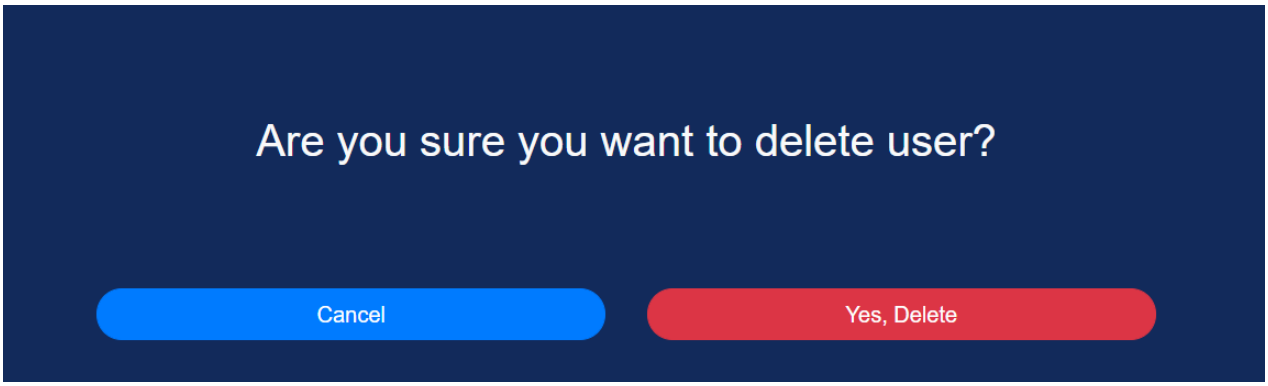

FIGURE 20: REVERSAL OF ACTIONS #2

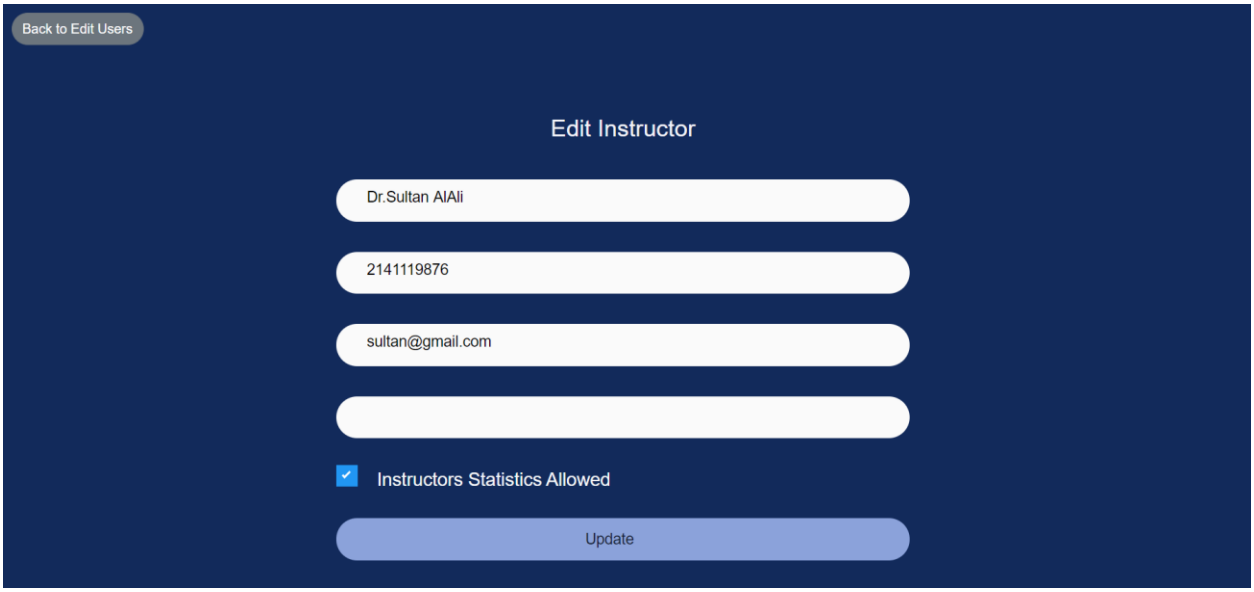

FIGURE 21: ADDING INSTRUCTOR ADMIN PAGE

Editing or adding an instructor includes the option for the admin to assign the supervisor.

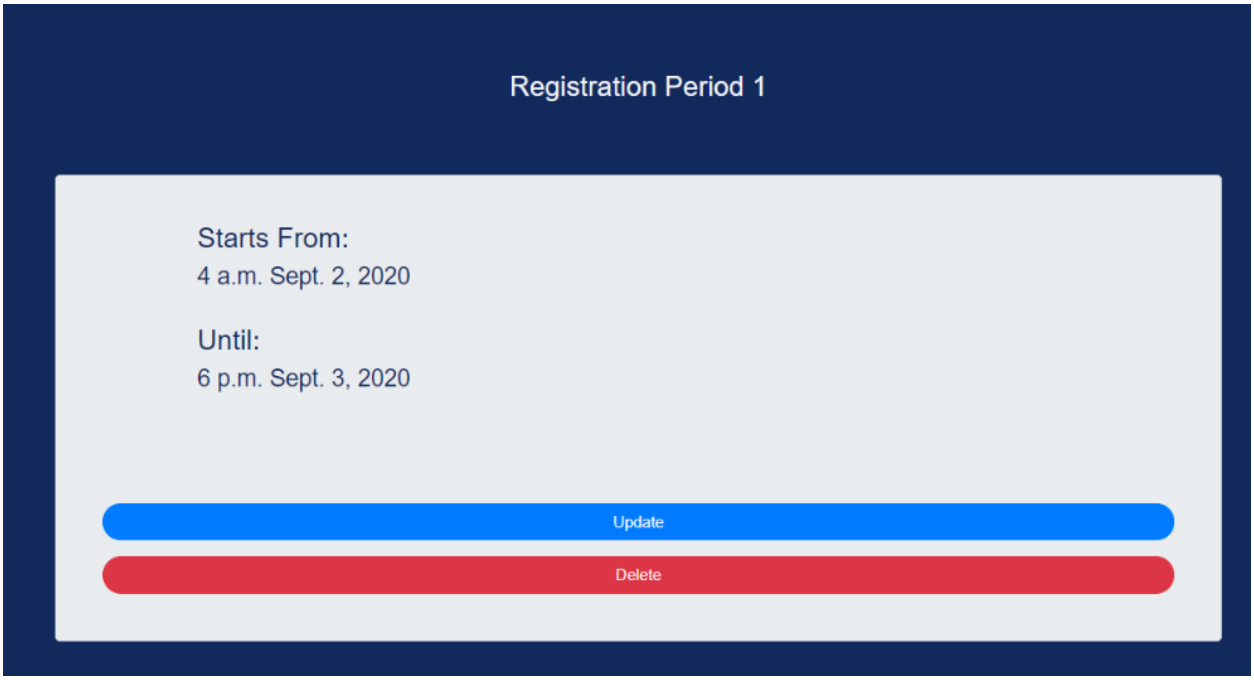

FIGURE 22: SETTING WISH-LIST TIME PAGE FOR ADMIN

After updating the wish-list registration time, the system automatically sends emails to notify both students and instructors.

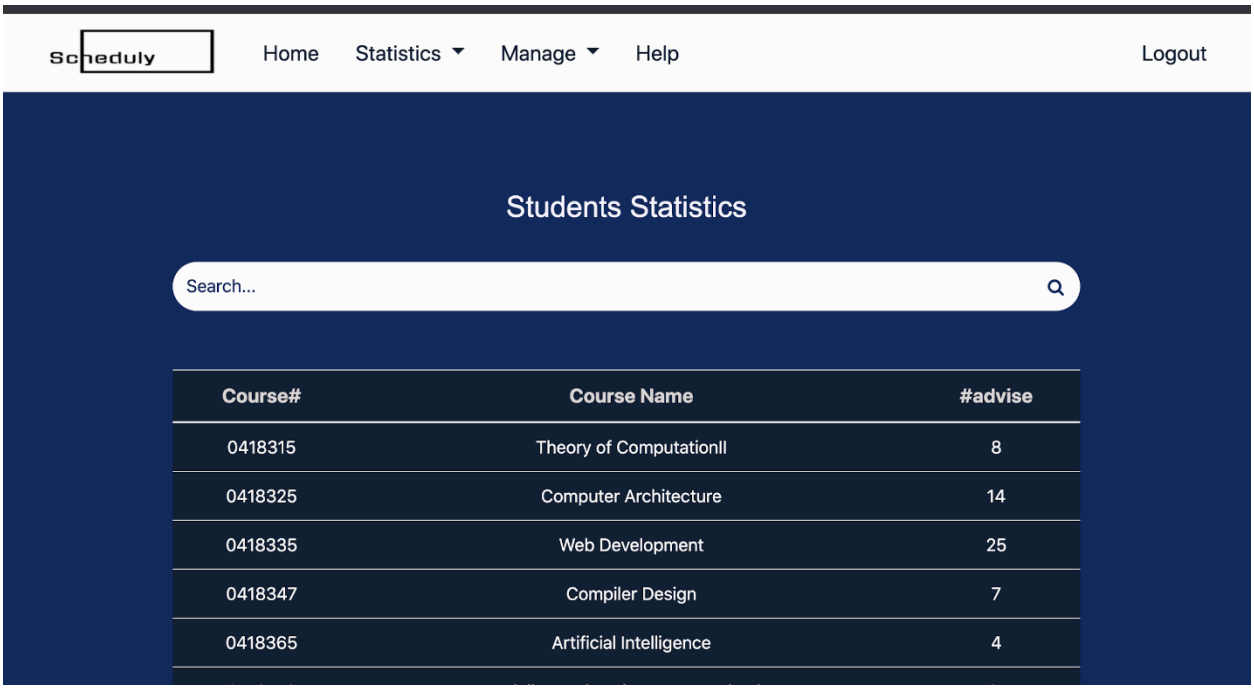

FIGURE 23: INSTRUCTOR PAGE TO VIEW STATISTICS FOR ELECTIVE CS COURSES

This informational page is to communicate to the instructors the courses that are in demand by students.

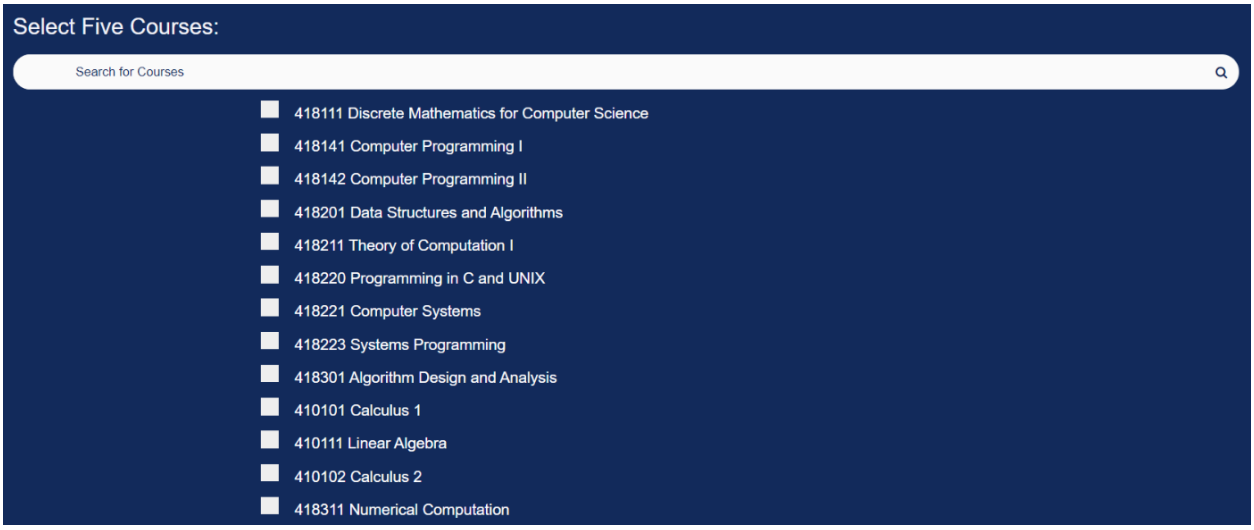

FIGURE 24: INSTRUCTOR PAGE TO VIEW OR UPDATE WISH LIST

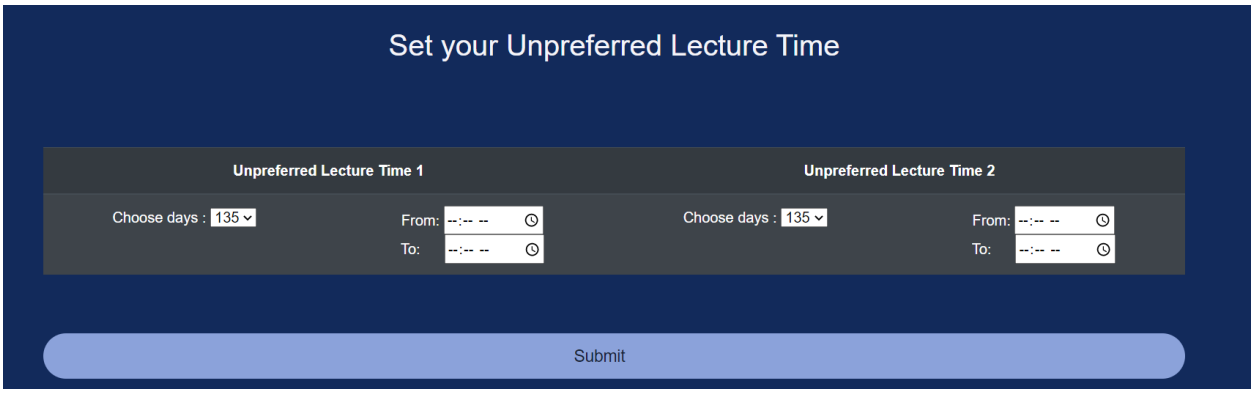

FIGURE 25: FURTHER INTRUCTOR WISH LIST CHOICE

FIGURE 26: INFORMATIVE FEEDBACK #6

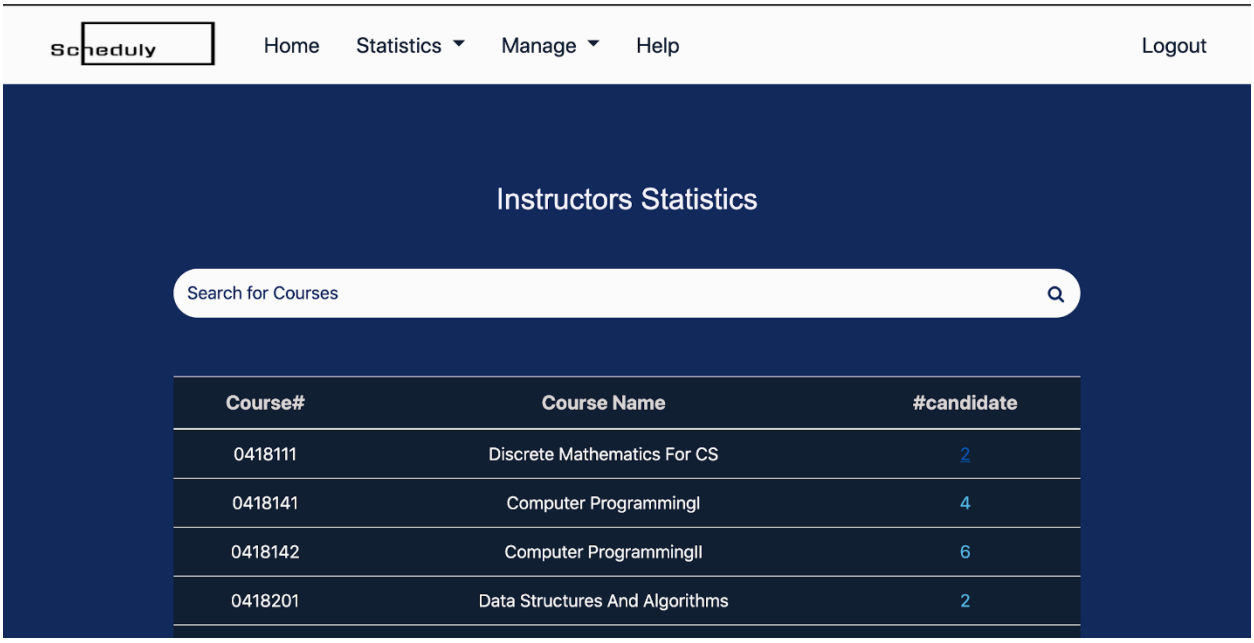

FIGURE 27: SUPERVISOR PAGE TO VIEW STATISTICS FOR COMULSORY COURSES

This informational page is to communicate to the schedule supervisor the number of instructors offering to teach a certain course.

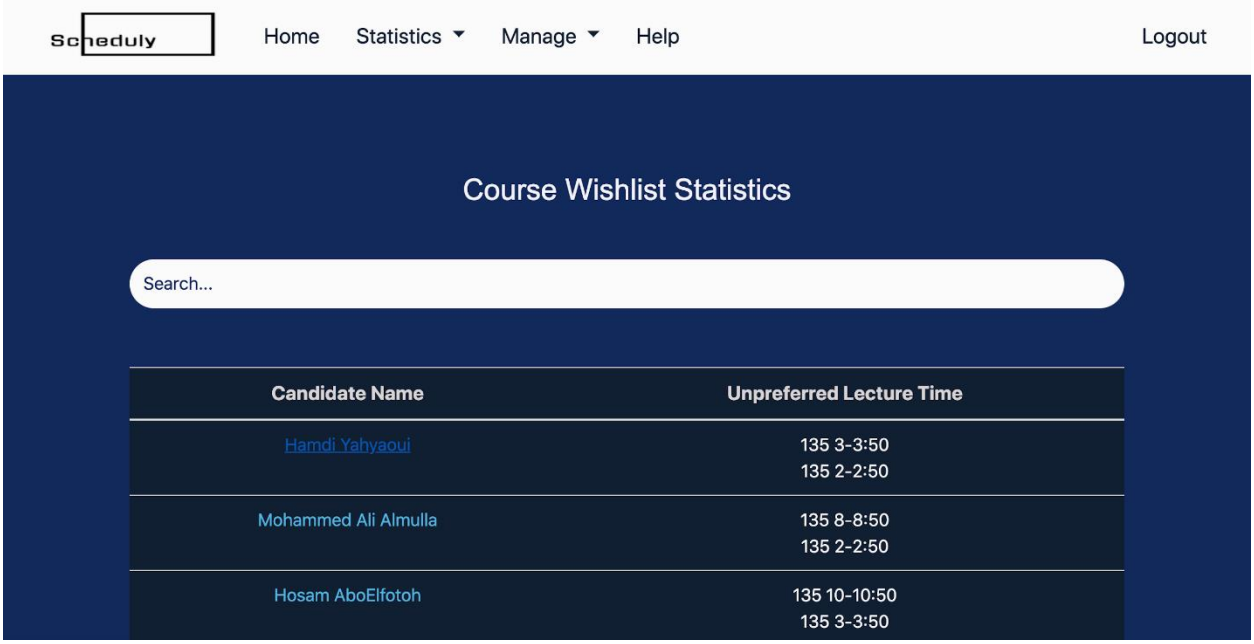

FIGURE 28: SUPERVISOR PAGE TO VIEW STATISTICS FOR INSTRUCTOR'S TIME PREFERENCES

This informational page is to communicate to the schedule supervisor the time preferences of instructors offering to teach a certain course.

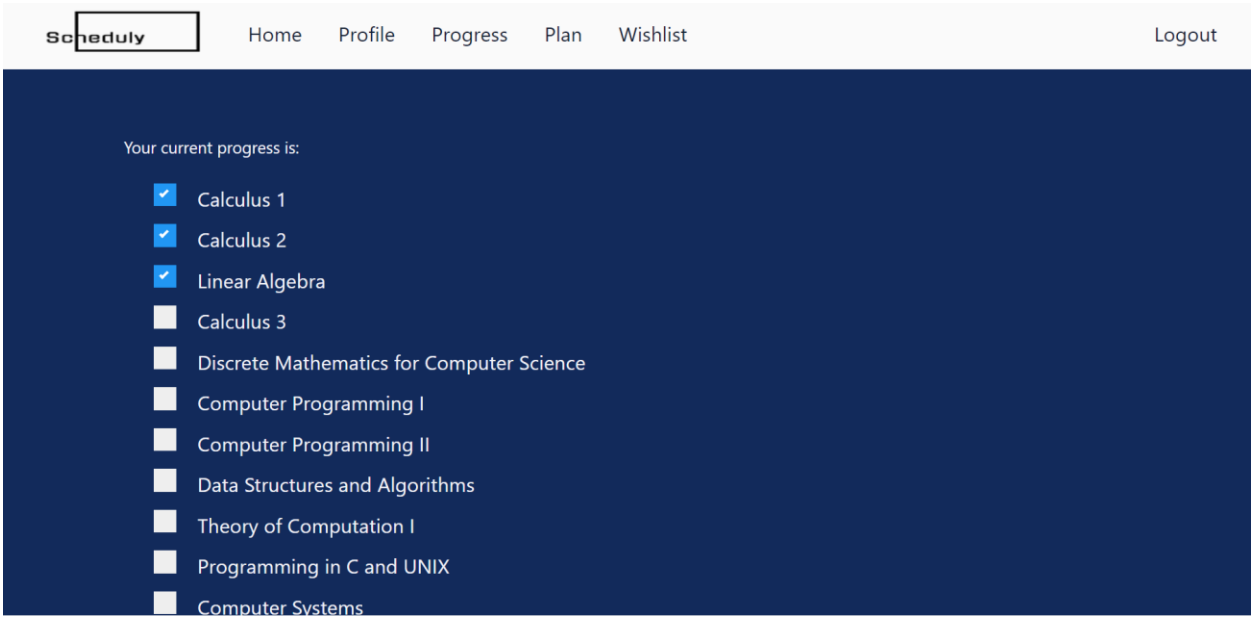

FIGURE 29: STUDENT PAGE FOR UPDATING OR VIEWING MAJOR SHEET PROGRESS

This page is to offer the student the option to update or view their progress.

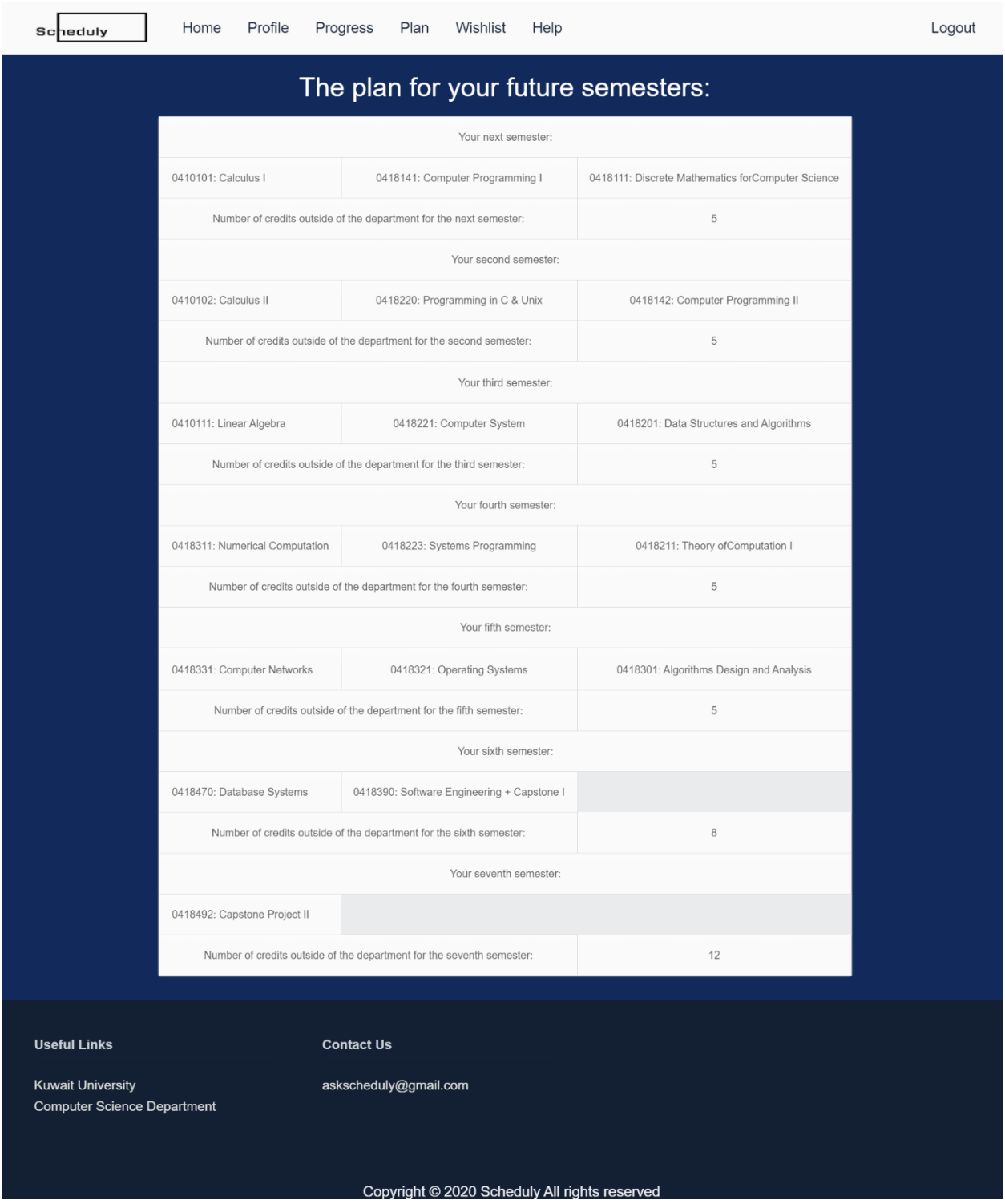

FIGURE 30: STUDENT PLAN EXAMPLE CASE

3.5 ALGORITHMS

To implement features of the system such as formulating the plan, validating the progress, and retrieving statistics. The team developed algorithms using python, which allowed the data to be treated as objects and could be easily iterated through and manipulated.

All statistics are handled by many-to-many relations with python inbuilt count function

```
If the post request is of a course to be added to a student's or instructor's wish list
        Add a new relation
If the get request is to view statistics
        Count number of relations
```

Python handles the relations as objects which makes it easier for information to be counter or filtered.

The course class had the following methods that returns a value to assures dependencies:

```
function is_available():
        return if all is true
                        from Progress filter (course, student id)
                        for course in self.dependancy.all()
function is_available_after():
          return true if all is true
             course in courses_completed
             for course in self.number.prerequisite.all()
```

The progress object existence proves the student finished the course. These methods are called often in the progress algorithm and in the plan algorithm.

For the optimal plan

```
    To fetch list of courses completed = [progress.course for progress in
Progress.objects.filter(studentID=this.user)
    Fetch the number of credits of the student
    If the student is a freshmen
        Set the limit of the courses in a semester to 3
    Else
        Set the limit of the courses in a semester to 2
    Couses_left = list of courses – list of courses completed
    while courses_left:
         courses_available = [
            course for course in courses_left
            if course.is_available_after(courses_completed) ]
         if not courses_available:
            courses_completed.extend(semester)
            semesters.append(semester)
         while courses_available:
```

```
        course = courses_available.pop()
        courses_left.remove(course)
        if number of courses in this semester is less than limit of courses in a semester:
            add new course to the semester
            credits for the semester -= number of the credits of added course
append to plan the semester
```

3.6 DESCRIPTIVE UML DIAGRAMS

3.6.1 ACTIVITY DIAGRAM

These activity diagrams help show the actions allowed and how our system deals with them for each different type of user.

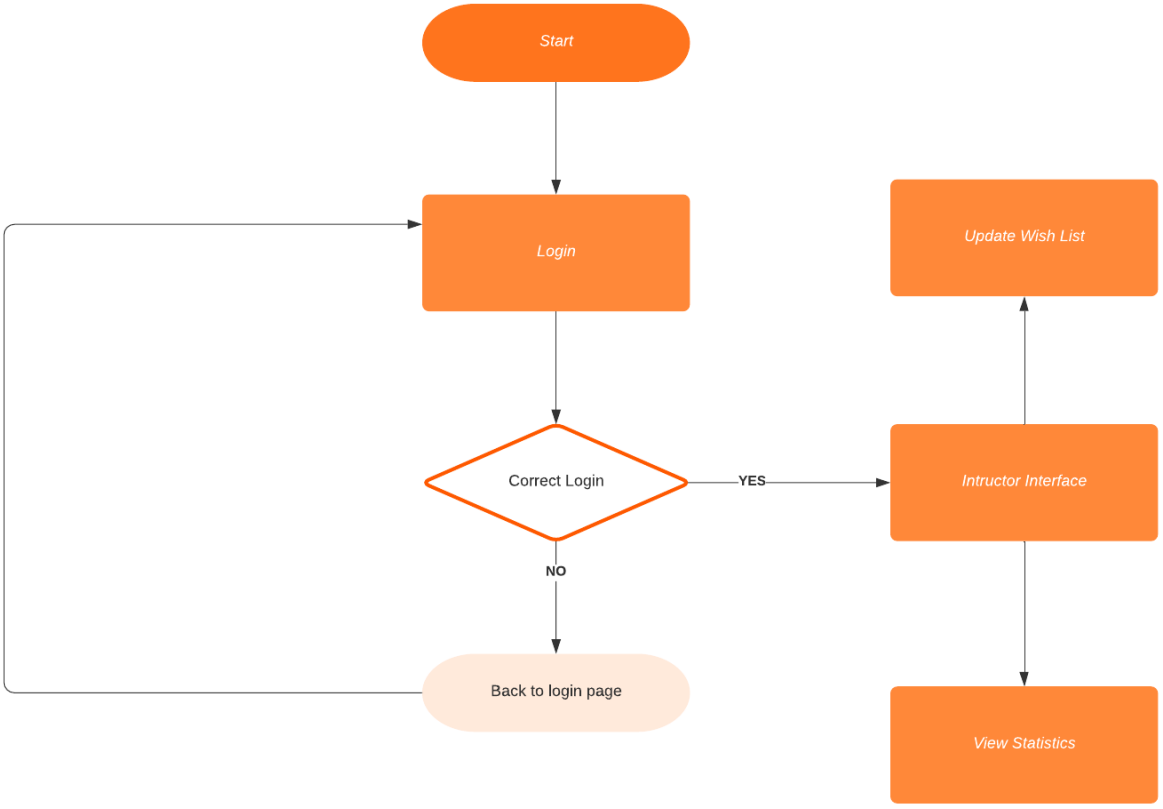

FIGURE 31: ACTIVITY DIAGRAM FOR AN INSTRUCTOR

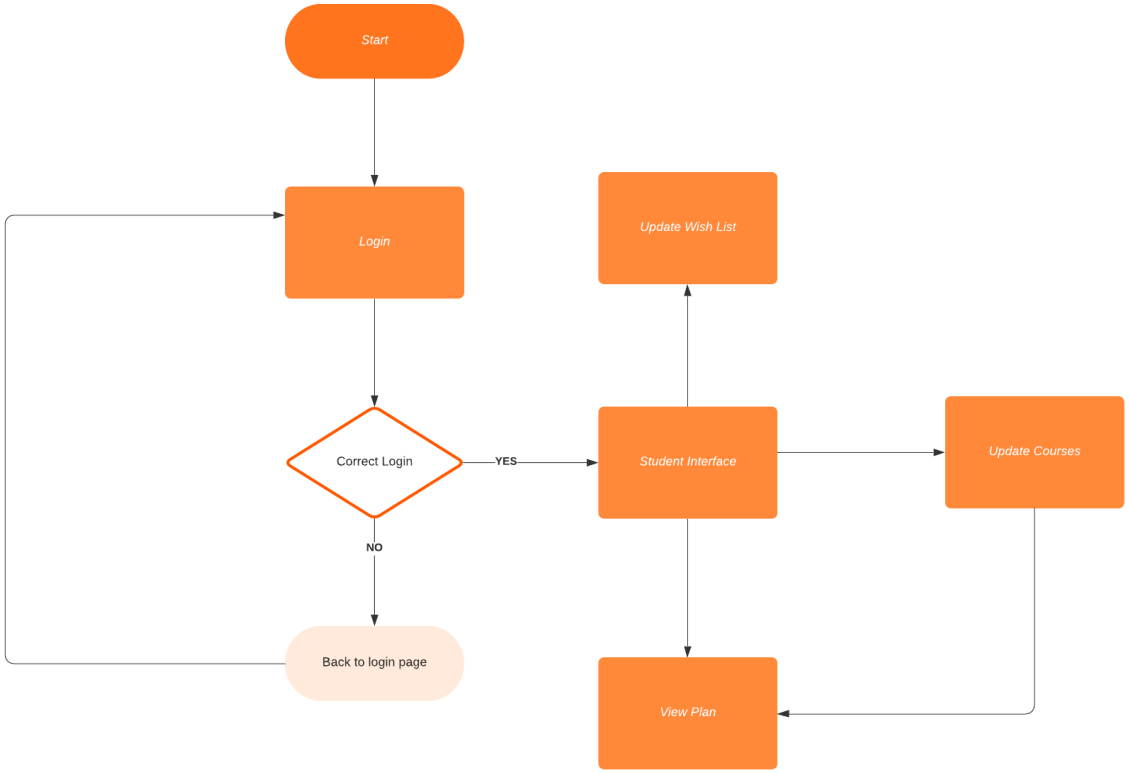

FIGURE 32: ACTIVITY DIAGRAM FOR A STUDENT

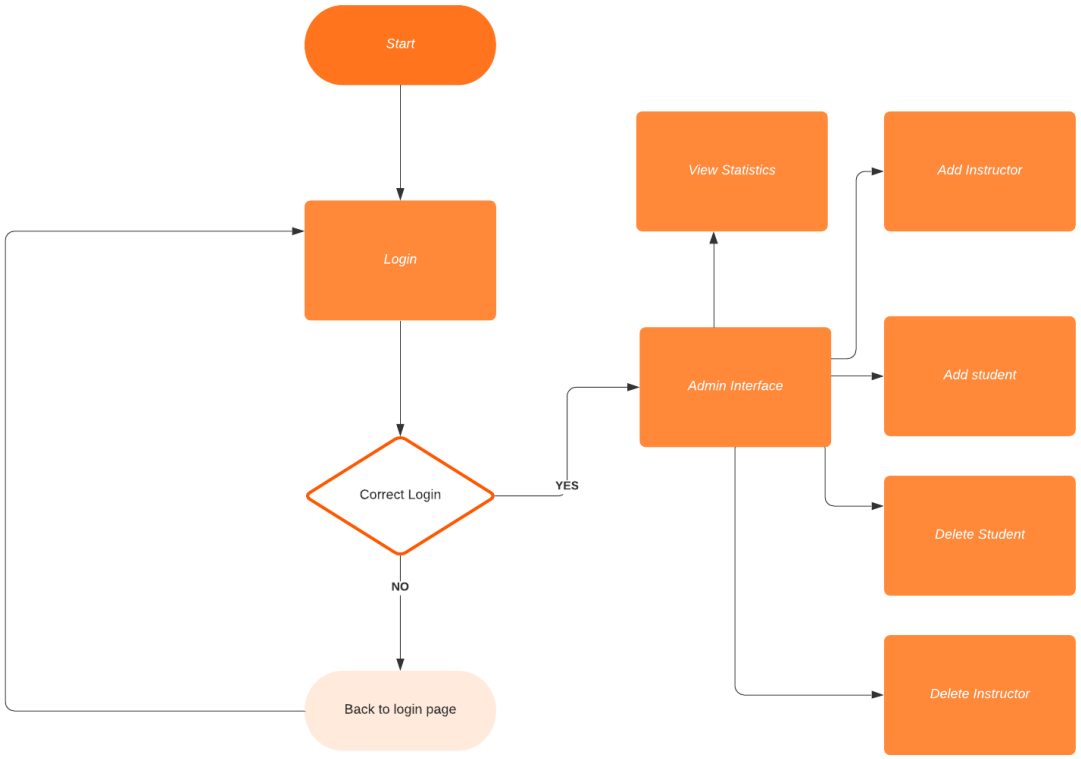

FIGURE 33: ACTIVITY DIAGRAM FOR THE ADMIN

The following sequence diagrams explain how the requests are handled by the system.

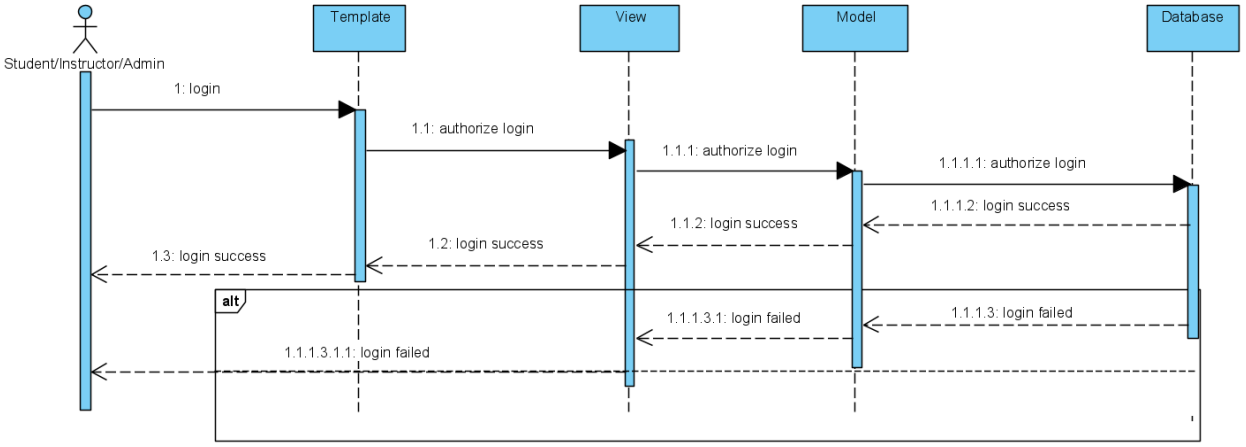

FIGURE 34: SEQUENCE DIAGRAM FOR USERS LOGIN

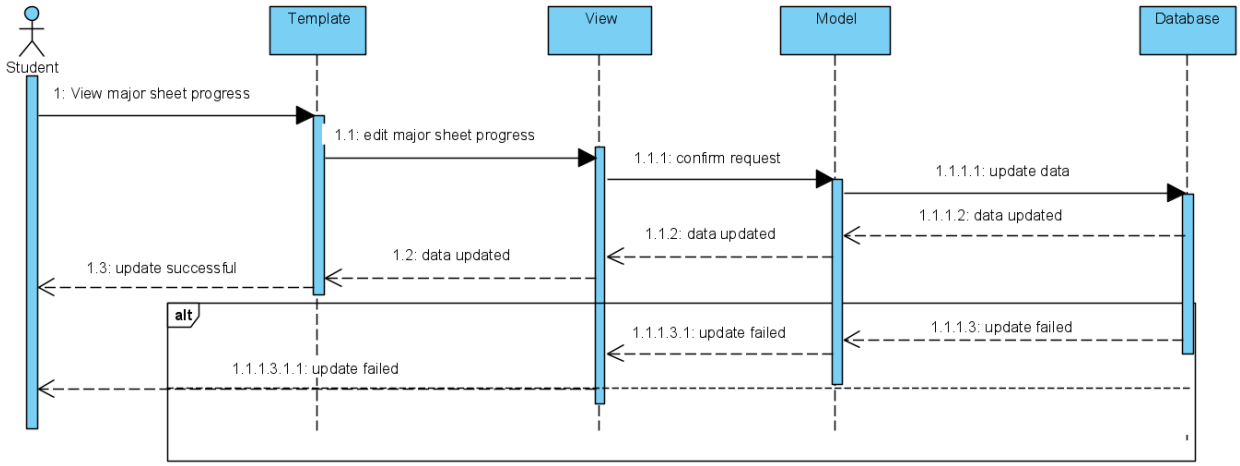

FIGURE 35: SEQUENCE DIAGRAM FOR A STUDENT TO VIEW OR UPDATE MAJOR SHEET PROGRESS

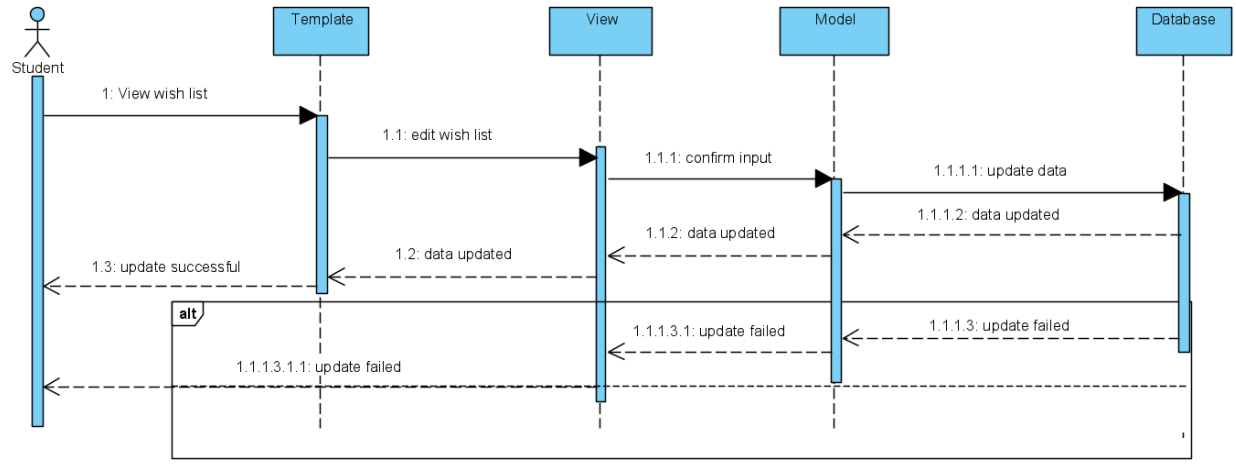

FIGURE 36: SEQUENCE DIAGRAM FOR A STUDENT TO VIEW OR EDIT A WISH LIST

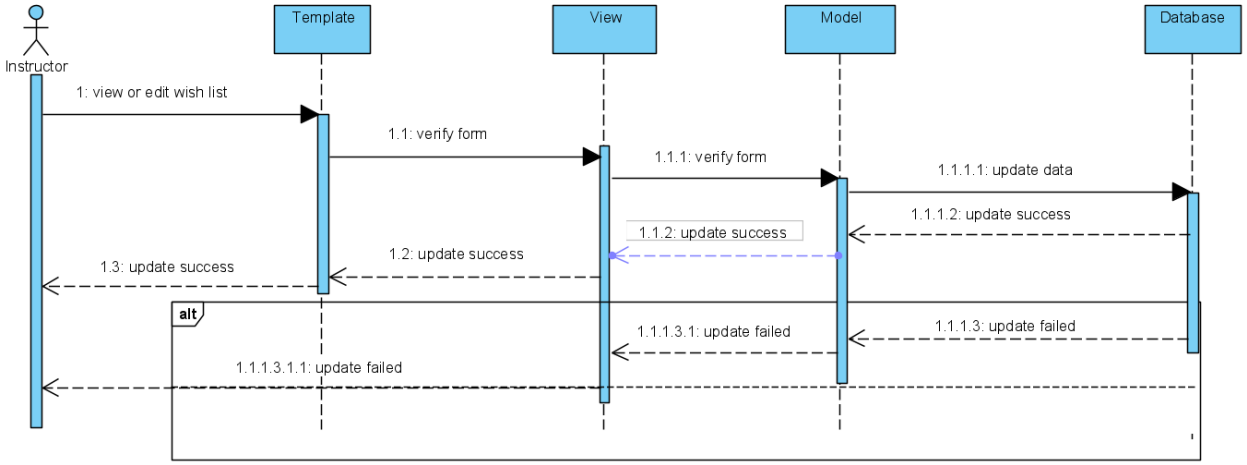

FIGURE 37: SEQUENCE DIAGRAM FOR AN INSTRUCTOR TO VIEW OR UPDATE WISH LIST

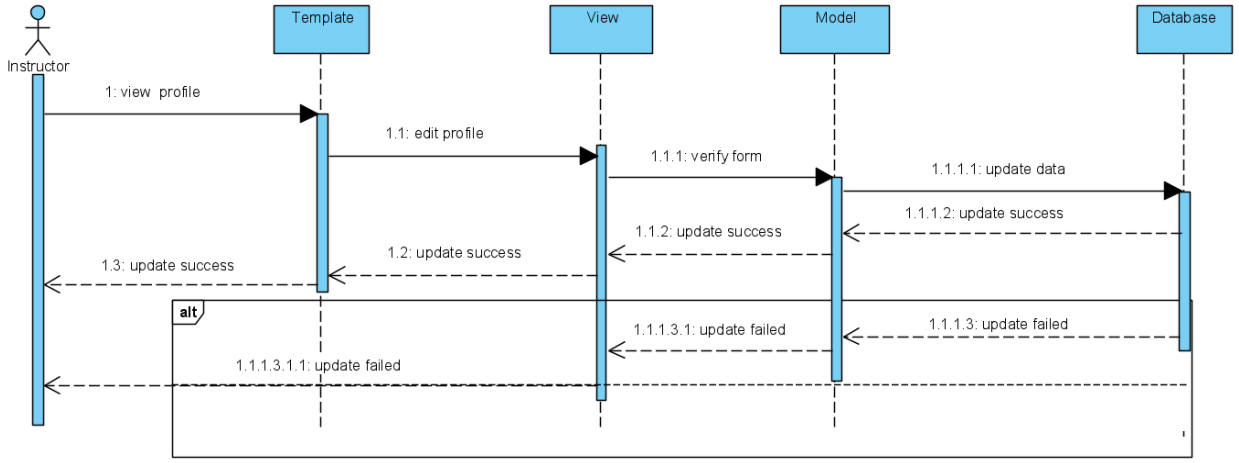

FIGURE 38: SEQEUNCE DIAGRAM FOR AN INSTRUCTOR TO VIEW OR EDIT PROGRESS

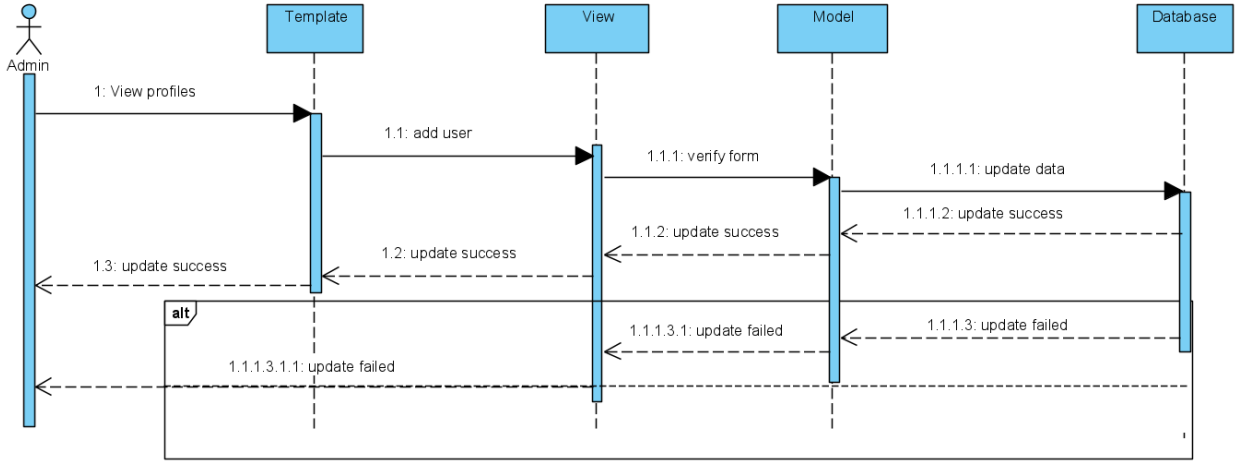

FIGURE 39: SEQUENCE DIAGRAM FOR ADMIN TO ADD A USER

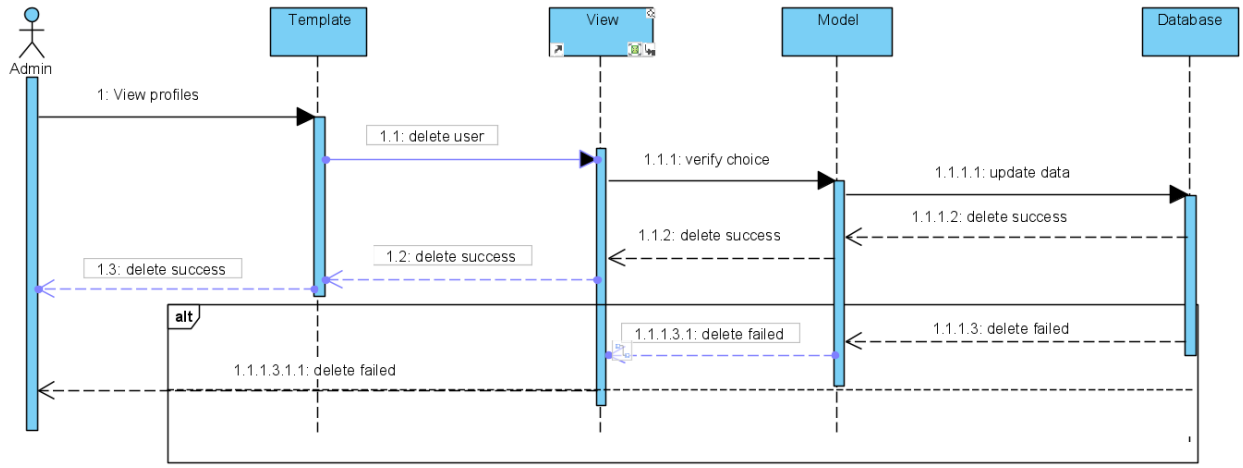

FIGURE 40: SEQUENCE DIAGRAM FOR ADMIN TO DELETE A USER

4. NEWLY DISCOVERED REQUIREMENTS

- Modified wish lists choices
    - The choice of wish list courses was changed to only include elective courses of the computer science department rather than both compulsory and elective courses.
- Statistics for instructors
    - The statistics shown to instructors now only the number of students with the course added to their wish-list directly.
- Courses outside of computer science department
    - The non-CS courses are included in the plan but only as credits without specifying the courses due to large options of courses available.

5. IMPLEMENTATION DETAILS

5.1 FRAMEWORKS AND PLATFORMS

The services offered by the system are targeted for web browsers on desktop and mobile platforms.

- Django Framework: the python-operated framework was used to develop and construct the major component critical to the system
- Bootstrap: the open-source CSS framework helped build responsive and mobile-friendly front-end interactions with users.

5.2 COMPONENT AND CODE REUSE

Components of the system were based on built-in Django library functions such as:

- The Django model layer which is the abstraction for structuring and manipulating the data of the web application
- The Django view layer to encapsulate logic of request and response
- The Django template layer for rendering information into HTML files
- The Django templates built-in filer to adjust variables on each individual page
- Django QuerySet to retrieve data the database
- Django Database SQLite to store the data of the system
- Django built-in Forms was a component heavily used to create and manipulate form data
- Django built-in admin library was expanded to fit the admin intended of this system

5.3 CASE TOOLS

The tools used to help develop the website by documentation the progress, to code, or to communicate between team members included:

- Atom text editor Atom: the open-source text editor was used to write the python codes that built the system and create the local web server for testing and deployment.
- Lucid Charts: to create diagrams to serve the purpose of visualization and understanding the components of the system, activities, and user behavior.
- Visual Paradigm: to create diagrams to serve the purpose of visualization and understanding the components of the system, activities, and user behavior.
- Git and GitHub: to host code without fear of losing it.
- Microsoft Word: to document progress of the project
- Microsoft Teams and Zoom: to run meetings between team members.

CURRENT ISSUES

6.1 DESIGN CHALLENGES

- Dealing with new platforms, technologies, and programming languages.
- Planning the work in a systematic manner and following the proposed plan.
- Synchronizing the work between the team members so there would not be any incompatibilities between the two.
- Sending emails for users to notify them about the time for wish list and in case we change the time.
- Design UI as it is not easy to create design that follows all opinions, formal and distinctive
- If users forget his/her email or password, he or she must contact the admin side

6.2 RISK MANAGEMENT

- Some courses such as Capstone and Software Engineering had to be taken at the same time and courses of math department had restrictions of their own, so some courses were merged as one course.

6.3 NEWLY DISCOVERED RISKS

- Learning new technology such as deployment to AWS could take most of the team's time.
- Tools and APIs used in this project could no longer be free for the team to use.

7. FINAL PROJECT PLAN

Based on the exceptional events that occurred around the world, we stood in a bewildering and did not know the fate of the project, so we tried to follow the previous plan but with changes and delay. However future work includes the finalization of deployment of the web-app online and finding a more visual way to represent the plan that is found by the algorithm.

8. SUMMARY

To summarize what was achieved in this phase, a major part of the project which had been purposed earlier was finalized. That is the algorithm intended to automate the planning for the student's has been developed with python. Furthermore, implementation of the different user features was finished to assure fit usage of the system. User interface along with user experience has been established to achieve criteria fit for a project of this institution and the academic environment.

In conclusion, our system strives to solve a problem in today's department of Computer Science in Kuwait University, which we mentioned in the introduction. By the end of this project the team hopes to help the students and instructors of the Computer Science department having more fulfilled and smooth university experience. The team will keep building the system and testing all cases of usage to fulfil this purpose.

REFERENCES

Artificial Intelligence: A Modern Approach. (1994).

AWS Documentation. (2020). Retrieved from Amazon: https://docs.aws.amazon.com/index.html?nc2=h_ql_doc_do

Makai, M. (2012). Retrieved from https://www.fullstackpython.com/

Norvig, S. R. (2009). *Artificial Intelligence: A Modern Approach.*

Vincent, W. S. (2018). *Django for APIs: Build web APIs with Python and Django.*