CS319 Object-Oriented Software Engineering

Project Analysis Report

Section 2

GROUP 2H

<u>BILDEMIC</u>

Ahmet Salih Cezayir (21802918)

İsmail Sergen Göçmen (21903707)

Egemen Öztürk (21801601)

Ezgi Lena Sönmez (21703799)

Muhammed İkbal Doğan (21702990)

Contents

1. Introduction

In our project, we created a pandemic manager website which is called Bildemic for Bilkent University. Students, Bilkent University staff, and instructors can use an easy-to-access website to make campus life easier during the pandemic. The primary goal of this website is to create a lower-level covid risk environment at the campus, where students, faculty, and staff may easily avoid high-level covid Risk activities.

Our project aims to ease several parts of the university life in a pandemic. In this manner, lectures, Bilkent sports centers, health centers, Bilkent cafeteria, and diagnovir test center will become more easy to manage and use in a pandemic. With our project, there is a system to control the Covid risk status of a classroom. Instructors can check the whole classroom Covid risk status and take attendance easier concerning Covid restrictions. Students can also order meals each day from the Bilkent cafeteria while they are in quarantine or when they do not want to be in crowded places like the school cafeteria. Moreover, they can book a reservation from Bilkent sports center using our application instead of face-to-face reservations, so staff and students do not have to interact often.

Our project has the following features and pages:

- Login Page
- Lecture Attendance Taker, Seating Plan, and Risk Status Monitor
- Profile
- Diagnovir Center
- Health Center
- Sports Center
- Bilkent Cafeteria

· Settings

2. Requiring Application Domain Knowledge

Instructor: We interviewed with Instructor Eray Tüzün. He mentioned that before starting the lesson, he and his colleagues may forget to check the HES status of the students. He also states that it is a difficult and time-consuming task for instructors to check whether students sit in the same place or not every time.

Our program aims to solve these two problems as follows:

1. Students enter a randomly created lecture code given by the instructor before starting the lecture, and students who are not allowed to enter the class appear directly in red on the instructor's and their classmates' accounts after entering this code. In this way, both the instructor and the classmates can warn these students.

2. There can be an exchange of random codes created on behalf of a particular course with the students they are neighbours to each other. This means that students who enter the random lecture code created by the instructor for the lesson and appear green can enter the random seat codes of their friends on their right and left into their own accounts. If these people match the information they have registered for the first time in the seat they sit in, they and the instructor understand that they are sitting in the correct place. This information is sent to the instructor by the system.

Student: We talked to a few of our friends. They stated that the problem they have in common during the pandemic period is that it is difficult to book a place in sports centers in the university. In other words, in the pandemic period, the number of users in the same center was restricted to ten people. Therefore, there may be no space left in the centers whenever they go and they add that they waste their time there while trying to make a reservation. They also stated that making a reservation on the phone may cause the wrong day and time to be set due to a communication error.

Our program aims to solve this problem by:
To register to the daily schedule through our program without need for phone calls or going to the reception.

Sport Center Staff: Meeting with central sports center business manager Orhan Saffetoğlu. Mr. Orhan mentioned that most of the time, users who want to use the sports centers do not come even though they have reserved a place, and that they prevent other people who want to use the centers during the pandemic process from benefiting from sports activities. He added that they could not see who these users were because they made the reservations on paper and filed at the reception. Since they do not have an electronic system, they cannot follow their attitude.
What he wants from us is to add a feature to our program where these users can be warned if their behavior continues or can be removed from using the activities by prohibiting them from making reservations for a while. With these features, it is aimed to ensure that these facilities, which have a limited number of people in the pandemic period, are used more efficiently.

Cafeteria Staff: They talked about the difficulty of sending food to quarantine dormitories. The students share the types of the meals they want and quantities of the meals to the nurse, from the nurse to the dietitian/cafeteria, and from the cafeteria to the cafeteria staff.
This chain is also followed when the cafeteria staff want to ask any questions to the student during the preparation of the meal or want to inform the student. In such a situation, they stated that this process also makes them wait a lot and makes them slow down in the preparation process.

In order to prevent this problem in our program, we aimed to develop a feature as follows:
Students/instructors order food from the program and cafeteria staff can see these orders first hand. We also ensure that some of the students' information is shared with the staff so that the cafeteria staff could contact the students if they need it.
This feature will both reduce this long way of communication and prevent the crowd in the cafeteria during the general pandemic period, by distributing students to the regions where they order their meals.

Health Center Staff: In our conversation with one of the university doctors, the doctor mentioned that it would be good for nurses to make their communication with patients more efficient. For example, the doctor said that the patient often had to be called repeatedly in order to follow the disease process and patient’s information. In addition, he also stated that people whose symptoms are insufficient for covid test/diagnosis also come to the health center by wasting time.
We aimed to solve these problems in our program as follows:
We opened a form indicating whether people should go to the health center or not. The person will not have to go to the health center unless they show certain symptoms. If the patient's symptoms are at a level that requires him/her to go to the health center, the form filled by the patient is sent to the health care workers. After the health center workers read the form and collect the patient's information, they start a chat. The patient and the healthcare worker talk over the chat, and the healthcare worker sets up an appointment for the patient. As long as the healthcare worker needs information and does not delete the form/chat, he can follow the patient's progress from the list of forms and access the patient's information whenever he/she wants.

3. Proposed System

3.1 Functional requirements

The system has six kinds of users:

-Instructor

-Diagnovir Tester

-Sport Staff

-Cafeteria Staff

-Health Center Worker

-Student

3.1.1 Login and Register Related Functionalities

When people enter our application, they encounter a login screen asking for an e-mail and a password. For those who do not have an account, there is also a register option. In the register, section users have to fill in the required information about themselves, such as their role at the campus, HES code, full name, ID if they are either a student or an instructor, email, password, and contact information.

3.1.2 Courses Related Functionalities

In the “Courses” tab, the instructors can create courses. When they decide to create the course, they are presented with various preset options about the course such as course name, section, and lecture place. The instructors can also edit their course options later from this tab. Instructors can take attendance of present students by our application and take action if there are any illegal situations with respect to Covid restrictions. They can also see and manage classroom seating plan.

Using the “Courses” tab, the students can enroll in a course using the instructor’s unique course code. After clicking on the course, students can enter where they are sitting in the classroom. If the student has already entered his/her seat to the application, they cannot change it and they can only see where they sit. In each class, instructors share a lecture code that is randomly generated and students are expected to enter this code into their application. After entering, students are counted as present in the class and their seating plan section becomes active. In the active state of the seating plan, each student is given a randomized integer seating code. When students click on the seating plan, they can see their seating code. On this screen, there are also places to enter the neighbor student’s seating code. In each lecture, students are expected to enter these codes into their application to ensure that they are sitting in the same seat as usual. After entering the correct neighbor codes, on the instructors’ application, the related seat becomes green. In this way, instructors can easily see who is sitting in the right place. If some students do not have access to the application, instructors can also confirm such students’ seating status from their application.

3.1.3 Profile Related Functionalities

Every user has a profile page to see and update their account info. There is an ID part for students and teachers but the staff. Users identify themselves and provide information about themselves on their profile pages. Users give their contact information such as their email address, student ID, and phone numbers on their profiles. This information is used throughout the application

3.1.4 Diagnovir Center Related Functionalities

In this tab, users can make reservations for the covid test at the Diagnovir Test Center. They can see the reservation-related info such as the date, time, and place of the test reservation. After taking the test, users can see their test results immediately. They also can access their previous test results. To lower the population at the Test Center, users can also cancel their reservations with one click.

Diagnovir Test Center Staff can observe upcoming reservations with brief information about the patient. After doing the test, they can also enter the test result th application. To lower the population at Diagnovir Test Center, staff can also make an additional reservation if any patient is waiting for their next test.

3.1.5 Health Center Related Functionalities

In this tab, users can call for an emergency by clicking to call an ambulance button. When this button is clicked, the patient’s address and contact number information is given to the health center staff to make the communication process with related health center staff faster. After receiving the emergency notification, the health center staff can call the person and learn about their situation. If necessary, an ambulance is sent to the patient. In addition to the emergency button, there is also a “Communicate with Medical Staff” button. After clicking, the user is given a form consisting of several questions about Covid. If the patient’s answers indicate that they might have covid, a chat between the patient and a health center worker starts. Through this chat, health center staff can create a reservation for the patient. In addition to this, health center staff can give some tips to the patient for them to feel better. Also, if necessary, health center staff can send an ambulance to the patient.

3.1.6 Bilkent Sports Center Related Functionalities

In this tab, users can make a sports reservation. Users can make a reservation at one of the three sports centers, by choosing their sport activity, day, and time. Students can make multiple reservations in different activities, however, they cannot make multiple reservations in the same activity. If the user misses more than three reservations, i.e if they make a reservation and do not come to the sports center more than three times, they become penalized. They cannot make a reservation at the sports center for at least two weeks.

Sports center staff can see the students and their reservations with the necessary information to manage the sports center quickly and easily and follow the covid regulations.

3.1.7 Bilkent Cafeteria Related Functionalities

In this tab, both instructors and students can see the daily menu of the Bilkent cafeteria. Users can also choose from three types of food (vegan, vegetarian or regular) and make a reservation. These meals are delivered to the user-determined places at the predefined times by the cafeteria staff. In this way, the crowdedness of the cafeteria can be reduced and the risk of spreading covid lessens. Students who are in quarantine dormitories can also easily communicate with the cafeteria about what type of menu they want to eat without any human interaction.

The cafeteria staff can also see several data about food reservations. They can see the general menu distribution, regional menu distribution, and individual menu distribution. By using this info, cafeteria workers can determine how much food they need for each menu type. Also, in the current situation, students in quarantine dormitories sometimes have difficulties getting the correct menu type and this feature aims to help the cafeteria staff to prevent such situations.

3.2 Non-Functional Requirements

3.2.1 User-Friendly Interface

The user interface is one of the most important aspects of a program since it creates a positive impression and makes it easier to use. As a result, we designed our designs to be as minimal as possible while yet being functional. To better user experience, these early designs focus on the user's necessities. As a result, we intend to draw the web app's materials as nicely and cleanly as feasible to make it look straightforward. Each part shall be easy to reach to make sure that the user has a pleasant experience.

3.2.2 Safety

All students will be assigned a random course code by the instructor when the course is created, preventing users who enter the system and generating undesirable circumstances. Additionally, if a person doesn't have a close-by seat plan with other students, they will not see another student's account, preventing contacts that are not required for the course.

3.2.3 Maintainability

Due to using the Object Oriented Program to build our application, adding new features will be easier than usual. Maintainability will be simple, especially for bug fixes. Adding new features or bug fixes will not affect the main code, which will be helpful to not mix the app up during optimization.

3.3 Pseudo Requirements

- The programming language must support Object Oriented Programming(OOP).
- The project must be a web based application.

3.4 System Models

3.4.1 Use Case Model

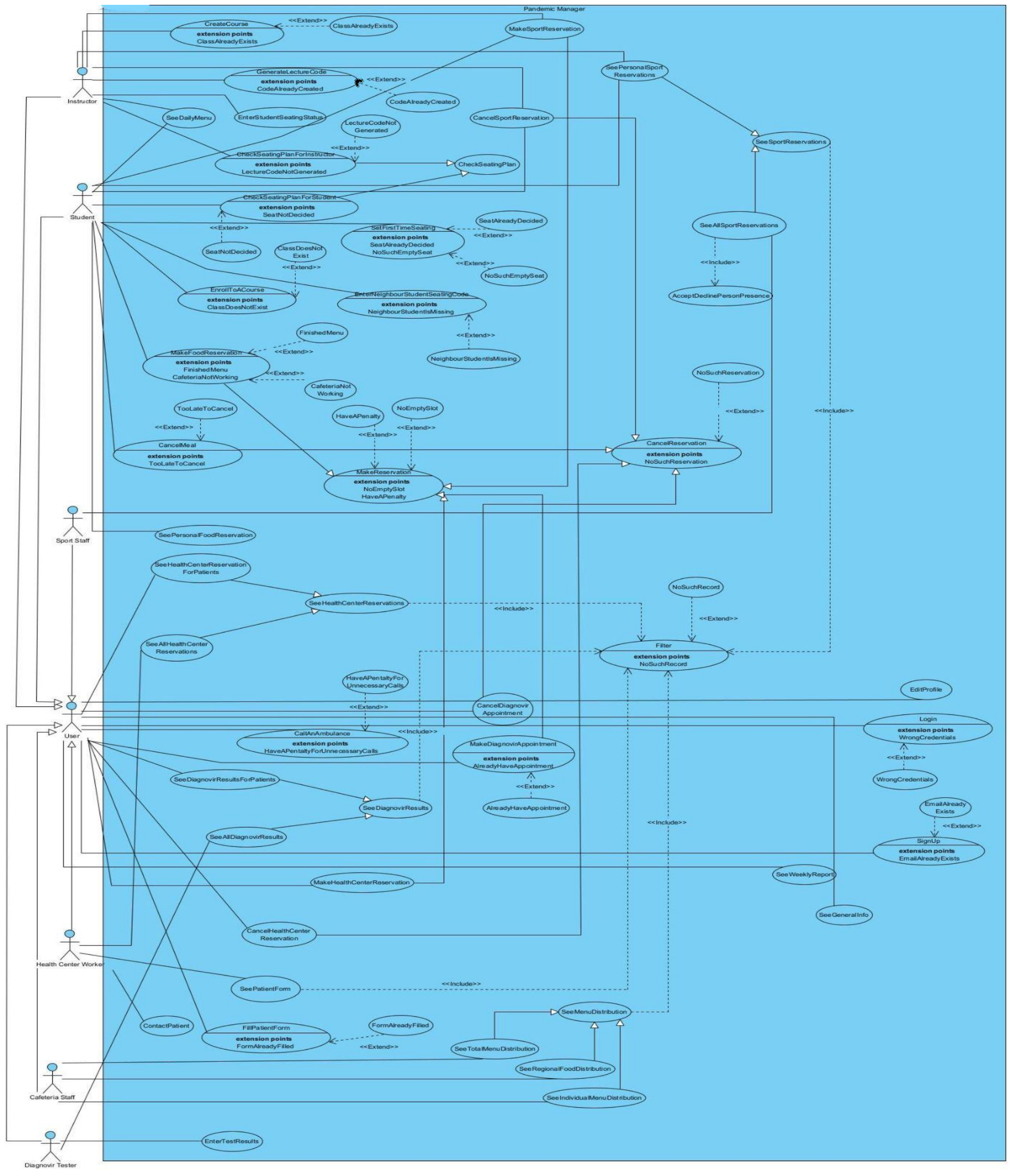


Fig. 1: Use Case Diagram for the Pandemic Management System

Use case name: CreateCourse
Participating Actors: Initiated by Instructor
Flow of Events:
1. The Instructor clicks GenerateButton and activates the “Create Course” function of his/her terminal.
   2. Pandemic Manager responds by presenting a form to the Instructor.
3. The Instructor fills out the form by writing the course name, section, class, and time and selecting the building. Once the form is completed, the Instructor submits the form and clicks the CreateClassButton.
4. Otherwise, if the Instructor changes their mind about creating a new course class, they click on CancelButton to exit the form.
   5. The Pandemic Manager System receives the form and the system creates a half random course code to make students enroll.
   6. Otherwise, the system does not create any course if the instructor clicks CancelButton and closes the form screen.

Entry Conditions:
- The instructor is logged into Pandemic Manager

Exit Conditions:
- The Instructor receives an acknowledgment and get a course code with the specific section and course code, OR
- The Instructor changes their mind about creating and clicks on CancelButton, and the system closes the form.

Use case name: ClassAlreadyExists
Participating Actors: Communicates with Instructor
Flow of Events:
1. The instructor fills the form for creating a new course class and clicks CreateClassButton.
   2. If the class already exists, the system sends a warning message about “This class already exists.”

Entry Conditions:
- This use case **extends** the Create Course use case. It is initiated by the system whenever the class trying to be created already exists.

Use case name: CheckSeatingPlanForInstructor
Participating Actors: Inherited from CheckSeatingPlan use case

Flow of Events:
1. The Instructor clicks the Course Seating Plan and sees whether the students are allowed to enter the class and if they sit in the right places or not.
Entry Conditions: Inherited from CheckSeatingPlan use case.
Exit Conditions: Inherited from CheckSeatingPlan use case.

Use case name: LectureCodeNotGenerated
Participating Actors: Communicates with Instructor
Flow of Events:
1. The instructor tries to check the correctness of the seating plan of the students.
    2. If the Lecture Code is not generated yet and since the students cannot enter the class, their seating cannot be arranged so the system sends a warning message to the instructor saying that the instructor should click to create a random lecture code.
Entry Conditions:
- This use case **extends** the CheckSeatingPlanForInstructor use case. It is initiated by the system whenever the instructor tries to check the seating plan of the wanted course.

Use case name: CheckSeatingPlanForStudent
Participating Actors: Inherited from CheckSeatingPlan use case
Flow of Events:
1. The student clicks on the course they want to see where they sit.
    2. The system displays the seating plan of the clicked lesson.
Entry Conditions: Inherited from CheckSeatingPlan use case.
Exit Conditions: Inherited from CheckSeatingPlan use case.

Use case name: SeatNotDecided
Participating Actors: Communicates with Student
Flow of Events:
1. The student clicks to CheckSeatingPlan to see their seat.
    2. The system sends a warning message that says: “Not entered any seating yet.”
Entry Conditions:

- This use case **extends** the CheckSeatingPlanForStudent use case. It is initiated by the system whenever the student tries to see their seat, which was not created before.

Use case name: EnterStudentSeatingStatus
Participating Actors: Initiated by Instructor
Flow of Events:
1. If the teacher would like to confirm that the student is in the classroom, he/she presses the seat button where the student is sitting on the CheckSeatingPlan screen and presses the confirm button on the pop-up that appears.
    2. The system converts the statue of the student from red to green.
3. If the status of the wrong student is updated, the button of the seat of this student is pressed, and the delete option is selected in the pop-up that appears.
    4. The system converts the statue of the student from green to red again.
5. If the instructor changes their mind, they press the CancelButton.
    6. The system closes the pop-up.

Entry Conditions:
- The instructor clicks CheckSeatingPlan.

Exit Conditions:
- The instructor changes the status of a student, and the system updates it, OR
- The instructor changes their mind and clicks on CancelButton, and the system closes the pop-up.

Quality Requirements:
- The students should primarily make these changes made by the instructor. If students ask the instructor to do this, the instructor changes the students' status.

Use case name: GenerateLectureCode
Participating Actors: Initiated by Instructor
Flow of Events:
1. The instructor clicks on the lesson they created with the CreateClass button before.
    2. The system shows the page with the content of the wanted course.
3. On this page, the instructor clicks the create random lecture code button that allows students to enter the course before each lesson starts.
    4. The system randomly creates code and shows the code for once for every lesson with the instructor.

Entry Conditions:

- The instructor clicks the course/lesson in which the code is desired to be created.

Exit Conditions:

- The instructor creates a random lecture code, and the system closes the screen.
- Otherwise, the instructor changes their mind about creating a code, and the system closes the screen.

Quality Requirements:

- With this generated code, students can log into the lecture and their HES statue info is sent to the instructor.

Use case name: CodeAlreadyCreated

Participating Actors: Communicates with Instructor

Flow of Events:

1. The instructor clicks to create random lecture code again.

 2. If the course already has a course code, the system sends a warning message to the instructor and says: "Code is already created."

Entry Conditions:

- This use case **extends** the GenerateLectureCode use case. It is initiated by the system whenever the instructor tries to generate a new code.

Use case name: SetFirstTimeSeating

Participating Actors: Initiated by Student

Flow of Events:

1. The student clicks on the seating plan page to save where they sit in the class/lesson.

 2. The system shows a pop-up of the seating plan.

3. The student chooses where they want to sit during the school term and saves it.

 4. The system registers the student and shows where they are sitting.

Entry Conditions:

- The student first selects the lesson in which they want to save the sitting plan for that lesson/course/class.

Exit Conditions:

- The student saves their seat, and the system closes the page
- Otherwise, the student presses the cancelButton to save the seat later, and the system shuts the page.

Quality Requirements:

- The student chooses where to sit and cannot change it again during the school term because of the Pandemic.

Use case name: NoSuchEmptySeat
Participating Actors: Communicates with Student
Flow of Events:
1. The student tries to pick a seat.
   2. The system sends a warning message as the place is already taken.

Entry Conditions:

- This use case **extends** the SetFirstTimeSeating use case. It is initiated by the system whenever the student tries to pick their seat, which is taken already.

Use case name: SeatAlreadyDecided
Participating Actors: Communicates with Student
Flow of Events:
1. The student tries to pick another seat after saving for the first time.
   2. The system sends a warning message as the place is already decided and saved.

Entry Conditions:

- This use case **extends** the SetFirstTimeSeating use case. It is initiated by the system whenever the student tries to pick their seat, which was saved already before.

Use case name: EnterNeighbourStudentSeatingCode
Participating Actors: Initiated by Student
Flow of Events:
1. The student clicks their seat.
   2. The system shows a pop-up for the seating plan.
3. The student enters the students' code sitting on their right and left in this pop-up and saves it.
   4. The system verifies whether "this" student is sitting in the correct place or not.

Entry Conditions:

- The student enters the class which they have a lesson at this time on the system, AND
- They enter the course code, which is already shared by the instructor before the lesson is started, AND
- If they are allowed in the class, then they enter when the course is created, all can take their neighbors' codes to verify their places.

Exit Conditions:

- The student enters and saves the code of the students on the right and left, and the system closes the page.

Quality Requirements:

- With this feature, it is checked whether the students have moved from their seats or not during the pandemic period, and this change appears in the system of the course instructor.

Use case name: NeighbourStudentIsMissing
Participating Actors: Communicates with Student
Flow of Events:

1. The student clicks on the seating chart/plan to enter the other students' code sitting on their right and left.

    2. The system shows a pop-up for the seating plan.

3. The student wants to enter the code of the others who have not entered the course code to be checked whether they are allowed.

    4. The system sends a warning message as a neighboring student is missing.

Entry Conditions:

- This use case **extends** the EnterNeighbourStudentSeatingCode use case. It is initiated by the system whenever the student tries to enter the code of an unapproved person in the class.

Use case name: EnrollToACourse
Participating Actors: Initiated by Student
Flow of Events:

1. The student clicks the EnrollCourse button for saving a course in their system.

    2. The system shows a pop-up to make the student enter a course code-shared by their instructor.

3. The student enters the code and submits it.

4. Otherwise, the student changes their mind and closes the submission page before entering any code.

5. The system adds the matching course to the student's page if the code matches any course in the database.

6. Otherwise, the system closes the page without upgrading any changes.

Entry Conditions:

- The student logs into their account and clicks on the "my courses" page.

Exit Conditions:

- The student submits a code and adds the course to their page, and the system closes the pop-up, OR
- The student changes their mind about enrolling in a course,, and the system closes the pop-up.

Use case name: ClassDoesNotExist
Participating Actors: Communicates with Student
Flow of Events:
1. The student tries to enter a class code by clicking the EnrollCourse button.

2. The system sends a warning message as class/code does not exist.

Entry Conditions:

- This use case **extends** the EnrollToACourse use case. It is initiated by the system whenever the student tries to enter any course code which does not exist.

Use case name: MakeDiagnovirTestAppointment
Participating Actors: Inherited from MakeReservation use case.
Flow of Events:
1. The user clicks the MakeAnAppointment button.

2. The system opens by stretching the page down to make the user fill the form.

3. The user fills the given form with their availability.
4. Otherwise, if the user changes their mind, they can cancel the appointment.

5. The system saves the user's appointment on the system.

6. Otherwise, the system closes by stretching the page up.

Entry Conditions: Inherited from MakeReservation use case.
Exit Conditions: Inherited from MakeReservation use case.
Quality Requirements:

- This feature allows users to make their appointments in a shorter time during the pandemic process.

Use case name: AlreadyHaveAppointment
Participating Actors: Communicates with the User
Flow of Events:
1. The user enters the Diagnovir page upon making an appointment.
	2. The system opens the page.
	3. If the user already has an appointment but wants to make another one, the system sends a warning message as there is already an appointment.

Entry Conditions:
- This use case **extends** the MakeDiagnovirTestAppointment use case. It is initiated by the system whenever the user tries to make another reservation on the system.

Use case name: NoEmptySlot
Participating Actors: Communicates with the User
Flow of Events:
1. The user wants to make an appointment and clicks the button for it.
	2. The system opens the page.
3. The user starts to enter their information into the form.
	4. If the Diagnovir Center, Health Center or Sports Center have no empty slot for any other appointments, the system sends a message to the user as no empty slot.

Entry Conditions:
- This use case **extends** the MakeReservation use case. It is initiated by the system whenever the user tries to make a reservation, even if there is no empty slot in the schedule.

Use case name: SeeDiagnovirResultsForPatients
Participating Actors: Inherited from SeeDiagnovirResults
Flow of Events:
1. The user wants to see their Diagnovir test results, they click the Test Results button.
	2. The system shows results.

Entry Conditions: Inherited from SeeDiagnovirResults
Exit Conditions: Inherited from SeeDiagnovirResults

Use case name: SeeAllDiagnovirResults
Participating Actors: Inherited from SeeDiagnovirResults
Flow of Events:
1. The Diagnovir Tester wants to see their patient's test result; they click the Test Results button.
    2. The system shows a results list.
Entry Conditions: Inherited from SeeDiagnovirResults
Exit Conditions: Inherited from SeeDiagnovirResults

Use case name: CancelDiagnovirAppointment
Participating Actors: Inherited from CancelReservation
Flow of Events:
1. The user changes their mind about providing a Diagnovir test; they click on their previous appointment.
    2. The system shows their appointment.
3. The user clicks the Delete Appointment button.
    4. The system deletes their appointments before the reservation time.
Entry Conditions: Inherited from CancelReservation
Exit Conditions: Inherited from CancelReservation

Use case name: NoSuchReservation
Participating Actors: Communicates with the User
Flow of Events:
1. The user logs into the system and clicks on the title he/she booked to cancel their appointment.
    2. The system opens the page and waits for the user's next step.
3. The user writes the date they want to cancel on the search button.
    4. If there is no such appointment on that date and time, the system sends a warning message as no such appointment.
Entry Conditions:

- This use case **extends** the CancelReservation use case. It is initiated by the system whenever the user tries to cancel a non-exist appointment.

Use case name: SeeDiagnovirResults
Participating Actors: Initiated by the User
Flow of Events:
1. The user enters the Diagnovir Test page.
	2. The system shows the list.
3. If the user is patient, can scroll down the page and find the test results.
4. If the user is a Diagnovir tester, then the tester can scroll down the page and find the patients' test results.

Entry Conditions:
- If the person entering is a patient, they can see only their test results in this list.
- If the person entering is a tester, they can see the test results of all patients in the searched time and date.

Exit Conditions:
- The user sees the test results, presses the back button, and the system closes the page.

Quality Requirements:
- At any point during the flow of events, this use case can **include** the Filter use case. The Filter use case is initiated when the user wants to filter the all test list for specific usage/searching. When filtered within this use case, the system shows the specific test results according to the filter. The filter can contain the date and time to be searched.

Use case name: NoSuchRecord
Participating Actors: Communicates with the User
Flow of Events:
1. The user wants to see the test result by entering the date in the search button on the test results page.
	2. If there are no test results on the searched date, the system sends a warning message as no such test results.

Entry Conditions:
- This use case **extends** the Filter use case. It is initiated by the system whenever the student tries to enter any course code which does not exist.

Use case name: EnterTestResults
Participating Actors: Initiated by Diagnovir Tester
Flow of Events:
1. The Diagnovir Tester clicks on the Diagnovir page to find the patient for entering the test result.
   2. The system opens the page.
3. The Diagnovir Tester enters the date and time of the test, which is taken from the patient, for finding the information of the patient.
   4. The system lists the appointments and photos of the people who had the test at the entered interval of time.
5. The Diagnovir Tester finds the appropriate patient and clicks on its information box.
   6. The system opens the patient's records.
7. The Diagnovir Tester enters and saves the patient's status.
   8. The system updates the information entered by the user.
9. If the Tester clicks on the wrong patient, they can close the page.
   10. The system closes the page without making any changes.

Entry Conditions:
- The Diagnovir Tester logs into the system and clicks on the patient for whom he/she wants to enter the test results.

Exit Conditions:
- The Diagnovir Tester enters the patient's test results into the system and the system closes the page
- Otherwise, the Diagnovir Tester clicks the wrong patient's information page and clicks the Cancel button and the system closes the page.

Use case name: SeeDailyMenu
Participating Actors: Initiated by Student/Instructor
Flow of Events:
1. The student enters the meal page and clicks the link to the daily menu of Catering.

Entry Conditions:
- The student logs into their account.

Exit Conditions:
- The student clicks the cancel Button on the right top, and the system closes the page.

Use case name: MakeFoodReservation
Participating Actors: Inherited from MakeReservation use case
Flow of Events:
1. The Student clicks on the Meal Page.
	2. The system displays a form.
3. The student fills and saves the form and orders a meal before the time is up.
	4. The system sends the information/order to the Cafeteria Employee according to the type and place the student wants to eat.
Entry Conditions: Inherited from MakeReservation use case
Exit Conditions: Inherited from MakeReservation use case
Quality Requirements:

- This feature ensures that the crowd in front of the cafeteria is divided,sports and density in that area is decreased during the pandemic period.

Use case name: CafeteriaNotWorking
Participating Actors: Communicates with Student/Instructor
Flow of Events:
1. The Student clicks on the Meal Page.
	2. If the student wants to order food on public holidays, the system displays a warning message to inform the student that they cannot order a meal.
Entry Conditions:

- This use case **extends** the MakeFoodReservation use case. It is initiated by the system whenever the student tries to order a meal on public holidays.

Use case name: FinishedMenu
Participating Actors: Communicates with Student/Instructor
Flow of Events:
1. The student enters the form page.
	2. If the meal is finished at the cafeteria, the system sends the information to the student that That menu is finished; please choose another.
Entry Conditions:

- This use case **extends** the MakeFoodReservation use case. It is initiated by the system whenever the student tries to order a finished dish.

Use case name: HaveAPenalty
Participating Actors: Communicates with the User
Flow of Events:
1. The user fills and saves the form on the reservation page for a meal, diagnovir test, health center appointment or sports center reservation.

2. If the user did not take/go their order/booked place even though they have ordered/booked three times before, the system sends a warning message as “not permitted for ordering meal or booking” and prohibits ordering/booking for the next week but for diagnovir test. For the diagnovir test, the system will send only a warning message.

Entry Conditions:

- This use case **extends** the MakeReservation use case. It is initiated by the system whenever the user tries to make a reservation after the user had previously booked three times and never showed up for any of them in a row

Quality Requirements:

- 1. If the user has made three reservations in a row and does not come to any of them and wants to make a “new” reservation, the system sends a warning message as they are not permitted to book a new reservation until the following week.
- In the pandemic period, especially in the sports center, the number of people whı can use the hall at the same time has decreased significantly. With the giving a penalty way, since they are penalized who do not show up cannot prevent other people from making reservations.

Use case name: Cancel Meal
Participating Actors: Inherited from CancelReservation use case
Flow of Events:
1. The student wants to cancel their own order, he/she clicks the ordering page.
2. If the student wants to cancel before the meal arrives, he/she can press the cancel button.

3. The system cancels their order and sends the information to the Cafeteria staff, and one is deducted from the number of dishes ordered.

4. If the student changes their mind about canceling the meal, they turn back to the previous page.

5. The system closes the page.

Entry Conditions: Inherited from CancelReservation use case
Exit Conditions: Inherited from CancelReservation use case

Quality Requirements:

- Until the last 30 minutes, the student can cancel their meal. Since the meals have to be on the way for the previous 30 minutes, the students must take the meal when it arrives. If they order but do not go to take that ordered meal, 1 point is added to the penalty box.

Use case name: TooLateToCancel
Participating Actors: Communicates with Student/Instructor
Flow of Events:

1. If the student wants to cancel the meal in the last 30 minutes, the system sends a warning message and does not allow the cancellation.

Entry Conditions:

- This use case **extends** the CancelMeal use case. It is initiated by the system whenever the student tries to cancel their ordered meal in the last 30 minutes.

Use case name: SeeTotalMenuDistribution
Participating Actors: Inherited from SeeMenuDistribution use case
Flow of Events:

1. The system collects the amount of all the meals and sends the information to the Cafeteria Staff on how many of each of the three types, which are regular, vegetarian, and vegan, of the meal is requested.

Entry Conditions: Inherited from SeeMenuDistribution use case
Exit Conditions: Inherited from SeeMenuDistribution use case

Use case name: SeeRegionalMenuDistribution
Participating Actors: Inherited from SeeMenuDistribution use case
Flow of Events:

1. The system sends information to the Cafeteria Staff about how many meals this Staff should send from which meal to which region.

Entry Conditions: Inherited from SeeMenuDistribution use case
Exit Conditions: Inherited from SeeMenuDistribution use case

Use case name: SeeIndividualMenuDistribution
Participating Actors: Inherited from SeeMenuDistribution use case
Flow of Events:
    1. The system sends information to the Cafeteria Staff which student wants which meal.
Entry Conditions: Inherited from SeeMenuDistribution use case
Exit Conditions: Inherited from SeeMenuDistribution use case

Use case name: SeeMenuDistribution
Participating Actors: Initiated by Cafeteria Staff
Flow of Events:
1. This use case can **include** the Filter use case. The Filter use case is used to search for the students who are controlled by the Cafeteria Staff as to whether they order a meal or not from the daily ordered meal list.
Entry Conditions:
Exit Conditions:
Quality Requirements:
- This feature ensures that the meal ordered by the student reaches their meal safely.

Use case name: MakeSportReservation
Participating Actors: Inherited from MakeReservation use case
Flow of Events:
1. The user enters the sports page to make a reservation
    2. The system opens a form to the user
3. The user chooses the time, date, and place according to the sport they want to do in the center and books
    4. The system adds the user's reservation to the schedule.
Entry Conditions: Inherited from MakeReservation use case
Exit Conditions: Inherited from MakeReservation use case
Quality Requirements:
- During the pandemic period, to prevent students and instructors from going to sport center to make a reservation by waiting in line, AND
- To reduce the phone traffic to the sports staff.

- Selection of a sports center place can be dormitories' sports hall, main sports hall or east sports hall.
- Selection activities can be archery polygon, ball games, fitness, squash courts, swimming pool, and so on.

Use case name: SeeAllSportReservations
Participating Actors: Inherited from SeeSportReservation use case
Flow of Events:
1. The sport staff can see all the reservations of the instructors and students for accepting or declining or only for checking.
    2. The system shows a booking system list of the booked days and times and the booker.
Entry Conditions: Inherited from SeeSportReservation use case
Exit Conditions: Inherited from SeeSportReservation use case
Quality Requirements:
- At any point during the flow of events, this use case can **include** the AcceptDeclineUserPresence use case. The AcceptDeclineUserPresence use case is initiated when the sports staff wants to add to the system whether the student or instructor is present at the appointment time or not.

Use case name: SeePersonalSportReservation
Participating Actors: Inherited from SeeSportReservation use case
Flow of Events:
1. The instructor and student can see all the reservations for checking the time, date, place, and which sport activity is booked by themselves.
    2. The system shows their reservation list.
Entry Conditions: Inherited from SeeSportReservation use case
Exit Conditions:Inherited from SeeSportReservation use case

Use case name: SeeSportReservation
Participating Actors: Initiated by Instructor, Student or Sport Staff
Flow of Events:

1. If the user is sports staff, he/she can check the reservations, such as checking whether a person has a booked time or not.
2. If the user is a student/instructor, he/she can check his/her own reservations from their own account.

Entry Conditions:

- If the user is a sports staff member, he/she enters the book list of all users.
- If the user is a student/instructor, he/she enters their sports page and goes to their reservations list page.

Exit Conditions:

- The sports staff/student/instructor closes the page.

Quality Requirements:

- At any point during the flow of events, this use case can **include** the Filter use case. The use case is initiated when the user wants to filter the reservation list for specific usage/searching. When filtered within this use case, the system shows the specific reservations according to the filter. The filter can contain the date, time, activity, place, and the name of the person who makes the reservation to be searched. This filter does this action by accessing the specific person.
- The sports staff can also use this use case for checking the booking system if a person claiming to have a date has a date or not

---

Use case name: CancelSportReservation
Participating Actors: Inherited from CancelReservation use case
Flow of Events:
1. The student/instructor wants to cancel their own reservation, he/she clicks the Sports Page and continues with his/her suitable reservation.
2. If the student/instructor wants to cancel before the booking time, he/she can press the cancel button.
    3. The system cancels their reservation and sends the information to the Sport staff, and the slot is rescheduled.
4. If the student/instructor changes their mind about canceling the booking, he/she turns back to the previous page.
    5. The system closes the page.

Entry Conditions: Inherited from CancelReservation use case
Exit Conditions: Inherited from CancelReservation use case
Quality Requirements:

- Until the last 30 minutes, the student/instructor can cancel their reservation. The student/instructor must go to the hall as it is unlikely that a new student/instructor will book after that time. If he/she does not go to the reserved center at that time, 1 point is added to his/her penalty box.

Use case name: CancelHealthCenterReservation
Participating Actors: Inherited from CancelReservation use case
Flow of Events:
1. The user wants to cancel their own reservation, he/she clicks the Health Center page and continues with his/her reservation.
2. If the user wants to cancel before the reserved time, he/she can press the cancel button.
    3. The system cancels their reservation and sends the information to the Health Center Workers, and the slot is rescheduled.
4. If the user changes their mind about canceling the booking, he/she turns back to the previous page.
    5. The system closes the page.

Entry Conditions: Inherited from CancelReservation use case
Exit Conditions: Inherited from CancelReservation use case
Quality Requirements:
- Until the last 30 minutes, the user can cancel their reservation. The user must go to the center because he/she has no right to distract workers. Even if he/she does not come, he/she has to call and show excuses. If he/she does not communicate and does not go to the health center at that time, 1 point is added to his/her penalty box.

Use case name: MakeHealthCenterReservation
Participating Actors: Inherited from MakeReservation use case
Flow of Events:
1. The user enters the health center page to make a reservation
    2. The system opens a form to the user and this form is a kind of test that after the user fills this form, the system decides whether that user needs an appointment with a doctor about Covid-19 or not.
    3. If the user has to see with health center staff according to their filled form, the system sends the information of the user and his/her form to the Health Center Staff.
4. The Health center staff opens a chat with the patient and if they decide to set an appointment, health center staff does this for the patient and sends the information of the appointment to the patient's account.

Entry Conditions: Inherited from MakeReservation use case
Exit Conditions: Inherited from MakeReservation use case
Quality Requirements:

- Since the basis of the application is the pandemic period, people who want to make an appointment at the health center only fill out a form for covid-19.

Use case name: SeeHealthCenterReservation
Participating Actors: Initiated by the User
Flow of Events:
1. The Health Center Worker can check the reservations, such as checking a person whether they have a booked time or not.
2. If the user is a student/instructor/diagnovir tester/sports or cafeteria staff member, he/she can check his/her own reservations from their own account.

Entry Conditions:

- If the user is a Health Center Worker, he/she enters the book list of all users.
- If the user other than a healthcare worker, he/she enters their sports page and goes to their reservations list page.

Exit Conditions:

- All users close the page

Quality Requirements:

- At any point during the flow of events, this use case can **include** the Filter use case. The use case is initiated when the user wants to filter the reservation list for specific usage/searching. When filtered within this use case, the system shows the specific reservations according to the filter. The filter can contain the date, time, and the name of the person who makes the reservation to be searched. This filter does this action by accessing the specific person.

Use case name: SeeHealthCenterReservationsForPatients
Participating Actors: Inherited from SeeHealthCenterReservation use case
Flow of Events:
1. The user enters the Health Center page and clicks where his/her reservations are.
    2. The system opens the list of reservations.
3. The user searches the list for the reservation he/she wants to check.

Entry Conditions: Inherited from SeeHealthCenterReservation use case
Exit Conditions: Inherited from SeeHealthCenterReservation use case

Quality Requirements:

- In this way, if the user does not remember the time or day of his/her reservation, he/she can check it here.

Use case name: SeeAllHealthCenterReservations
Participating Actors: Inherited from SeeHealthCenterReservation use case
Flow of Events:

1. The Health Center Worker clicks on the list of patients' reservations and the system shows which patient comes on which day and time.

Entry Conditions: Inherited from SeeHealthCenterReservation use case
Exit Conditions: Inherited from SeeHealthCenterReservation use case
Quality Requirements:

- With this feature, he/she checks the reservation of the patient he/she wants to see when the patient comes, and there is no need to keep the patient busy for such a reason as asking him/herself.

Use case name: SeePatientForms
Participating Actors: Initiated by Health Center Workers
Flow of Events:

1. When the patient fills out a form, the system sends a notification to the healthcare worker with the least number of forms on his/her page according to the number of disease symptoms that the patient fills in the form.

Entry Conditions:

- The patient fills out a form and makes an appointment according to the sensitivity of the form.

Exit Conditions:

- The Health Center Worker can delete the form

Quality Requirements:

- With this feature, patients who are sick get an appointment when they fill out the form and the form results are seen as serious.
- Users who do not have symptoms of illness enough do not need to go to the health center.
- The system aims to distribute the load equally among the employees by sending form notification to have the least number of forms on his/her page.

Use case name: SeePersonalFoodReservation
Participating Actors: Initiated by Student/Instructor
Flow of Events:

1. The student/instructor wants to check what and to where he/she ordered his/her meal, the system shows his/her last order.

Entry Conditions:

- The student/instructor clicks to their meal page and can see their last order at the bottom.

Exit Conditions:

- The student/instructor can close the page.

Use case name: CallAnAmbulance
Participating Actors: Initiated by all users but Health Center Workers
Flow of Events:

1. If the user thinks he/she has covid-19 disease and wants to be taken to the health center or hospital, he/she can click the CallAnAmbulance button.
2. The system notifies the health center workers.

Entry Conditions:

- The user logs in from his/her own account and clicks this button from the health center page.

Exit Conditions:

- The user can exit the page before pressing this button.
- Otherwise, a phone call is made and the system automatically closes the page.

Quality Requirements:

- Since this feature notifies the health center worker by notification, it notifies the health center worker in seconds in advance of what the call is about.

Use case name: HaveAPenaltyForUnnecessaryCalls
Participating Actors: Communicates with the User but Health Center Workers
Flow of Events:

1. If users are found to make unnecessary calls more than two times by health center workers, the system does not turn off the call feature, but adds 1 penalty point to the person’s profile.
2. The system is notified by health center workers to make the system put a penalty point on the wanted user.

3. The system sends a warning message after the second unnecessary call that they have a penalty point for this call.

Entry Conditions:

- This use case **extends** the CallAnAmbulance use case. It is initiated by the system whenever the second call is made by the user unnecessarily.

Use case name: ContactPatient

Participating Actors: Initiated by Health Center Worker
Communicates with the User others than Health Center Worker

Flow of Events:

1. The Health Center Worker can open a chat with the patient if he/she thinks it is necessary to communicate through the form he/she reads.

Entry Conditions:

- The patient fills out a form and makes an appointment according to the sensitivity of the form.

Exit Conditions:

- The Health Center Worker can delete the form, OR
- The Health Center Worker can open a chat window.

Quality Requirements:

- With this feature, patients who are sick get an appointment when they fill out the form and the form results are seen as serious.
- Users who do not have symptoms of illness enough do not need to go to the health center.

Entry Conditions:

- The system sends the form filled by the user to the healthcare worker.

Exit Conditions:

- Health center workers can close the page any time.

Quality Requirements:

- The system does not send every form filled by users to the healthcare worker. Which form the system will send will be determined as a result of discussions with the university doctor.
- If the covid-19 symptoms in the patient are in the majority who need to contact the health center, the system sends the form to the health worker.

Use case name: FillPatientForm
Participating Actors: Initiated by the User but Health Center Workers
Flow of Events:
1.The user who feels sick opens the health center page and presses the "contact" button.
2. The system opens a form to the user.
3. The user fills and submits this test.
4. The system shows the test/form result to the user.
5. If as a result of the test that the user may need to contact the health center, the system shows a pop up saying that: "Would you like to send the form to the health center?".
6. The user can send the form to the health center worker or close the page.
Entry Conditions:

- The user opens the health center page and click contact/test button

Exit Conditions:

- The user can choose to send the form if a pop up appears.
- Otherwise, the user can close the page after he/she sees his/her own form/test result.

Quality Requirements:

- This use case also allows the user to test from time to time when he/she doubts himself/herself, without the need to contact anyone.

Use case name: FormAlreadyFilled
Participating Actors: Communicates with the User but Health Center Workers
Flow of Events:
1. If an appointment is set, the system prevents filling out a second form until that reservation date has passed or the health center/patient cancels the appointment.
Entry Conditions:

- This use case **extends** the FillPatientForm use case. It is initiated by the system whenever the user has an appointment with health center already.

Use case name: EditProfile
Participating Actors: Initiated by the User
Flow of Events:
1. The user enters his/her account and clicks the edit profile page

2. The system opens that page and waits for the updates.

3. The user can add a new profile photograph, can change his/her address and so on, then he/she clicks the update button.

4. The system loads the last updates and closes the page.

Entry Conditions:

- The user clicks on the setting from the menu “tab” and edits his/her profile.

Exit Conditions:

- The user can submit his/her changes and the system updates and closes the page.
- The user can change his/her mind and click another page to close this edit profile page.

Quality Requirements:

- This feature is where the user edits how he/she appears to employees such as Health Center Workers, Sport Staff, Cafeteria Staff, and the Diagnovir tester.

Use case name: Login

Participating Actors: Initiated by the User

Flow of Events:

1. The user must verify his/her email and password on this page in order to enter his/her own account on the website.

2. If the system verifies the person who tries to enter, it sets his/her account and opens it.

Exit Conditions:

- The user leaves the website without logs into his/her own account.
- Otherwise, the user can click to logout button.

Use case name: WrongCredentials

Participating Actors: Communicates with the User

Flow of Events:

1. If the user enters his/her mail or password incorrectly, the system sends a message.

Entry Conditions:

- This use case **extends** the Login use case. It is initiated by the system whenever the user writes his/her account info incorrectly.

Use case name: SignUp
Participating Actors: Initiated by the User
Flow of Events:
1. The user comes to the sign-up screen of the website and fills in the requested information and saves.

2. The system adds the user to the site and the person can access the content at any time by entering the site and opening their own account.

Quality Requirements:
- Anyone who wants to log in to the website has an account. In this way, a user interface comes to their page according to the roles they specify. This interface varies from user to user.

Use case name: EmailAlreadyExists
Participating Actors: Communicates with the User
Flow of Events:

1. If the user writes the e-mail address he/she wrote before while registering on the site, the system will send a warning message that this account already exists.

Entry Conditions:
- This use case **extends** the SignUp use case. It is initiated by the system whenever the user enters the same mail address while he/she is registering.

Use case name: SeeWeeklyReport
Participating Actors: Initiated by the User
Flow of Events:
1. Weekly report appears on the main screen when the user logs in to their account and is updated every week.

Entry Conditions:
- The user logs into his/her account and checks the main page.

Quality Requirements:
- All users follow the events in the school within 1 week and see the report related to covid.
- This feature keeps the user informed of the latest happenings at the school.

Use case name: SeeGeneralInfo
Participating Actors: Initiated by the User
Flow of Events:
1. If the user wants to see the contact information of the university, he/she presses the tab button in the upper left corner and then continues with the general info button.
	2. The system opens the information page.

Entry Conditions:
- The user presses the tab button that will always be on the left side of the screens and clicks on the general info button above the settings.

Exit Conditions:
- The user can closes the page by pressing any other button.

Quality Requirements:
- The general info page contains all contact information that will help and facilitate the user during the pandemic period, especially the contact information of the health center.

3.4.2 Class Diagram

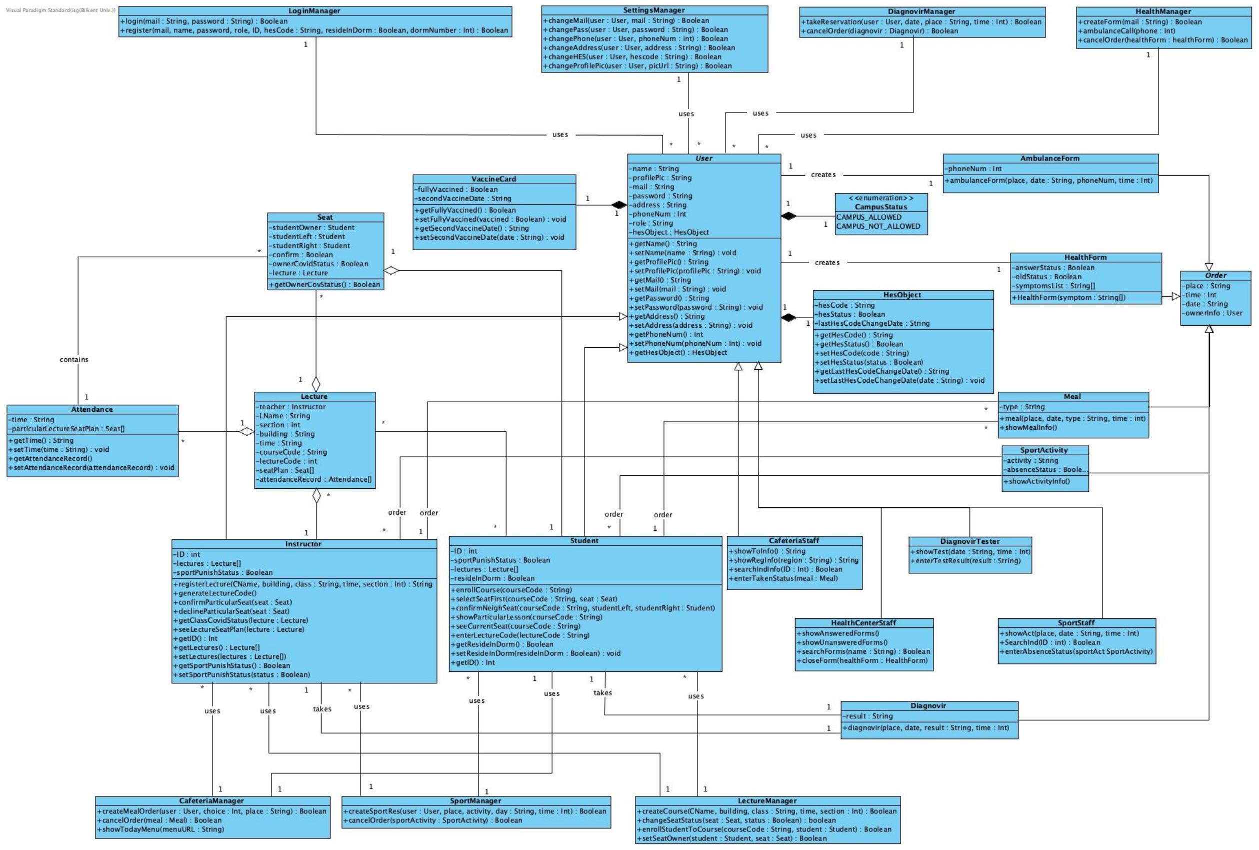


Fig. 2: Class Diagram for the Pandemic Management System

For better understanding, some of the getter/setter methods are removed from the diagram.

User: User is an abstract class. This class contains common information of all users such as their names, mail addresses, profile pictures, passwords, phone numbers, roles(instructor, student and so on), and HES code. Different users are created with this abstract class which are student, instructor, cafeteria staff, health center staff, diagnovir tester, and sport staff. Users can benefit from the features of the program which is updating their personal information from the settings, taking a diagnovir test, and filling out a health form, specifically.

Instructor*:* Instructor is a child class of Abstract User Class. The instructor has an ID unlike the user class. This class members can create a new lecture and generate a specific code according to the lesson and the section. This user can check whether a student is allowed in the class or not according to their HES status. The user can take advantage of the features of the program in order to order a meal from the university cafeteria for lodging where he/she lives in the campus area and to use the sports activities in the sports centers on the campus.

***Student**:* Student is a child class of Abstract User Class. The student has an id unlike the user class. The student can enroll in a specific course section with a generated code by the instructor. This user has to determine the seat for each lesson throughout the semester because of the pandemic period. In addition, the student should check that his/her seat is the same every time he/she attends the class. The Student shares a one-time random code with other students sitting on his/her right and left and confirms his/her place during the lesson. He/she can do these with the selectSeatFirst and confirmNeighSeat methods. This user can benefit from the features of the program such as ordering a meal from the university cafeteria, and making a reservation from a sports center to do sport activities, specifically.

***CafeteriaStaff**:* CafeteriaStaff is a child class of Abstract User Class. This user has access to three different groups and amounts of meals. With the showTotalInfo method, the staff can see the amount of the meals as their types which are normal, vegetarian and vegan. In this way, they see how many dishes of each variety they will prepare. With the showRegInfo method, the staff can see how many of which type of meal wil be sent to which part of the campus and make the packaging according to these conditions. With the searchIndividualInfo method, this user can see the individual orders of the users who placed the order. In this way, if the user who ordered the meal did not come to pick up his/her food, staff can identify this person and change order taken status with the enterTakenStatus method.

***HealthCenterStaff**:* HealthCenterStaff is a child class of Abstract User Class. The health center staff member can see and read the patients’ forms sent to him/her by the system then open a chat window with the patient. The user can also make an appointment for the patient according to their situation. The member can access or delete old forms and chats. In addition, a patient contacts this user every time he/she calls for an ambulance.

***DiagnovirTester**:* DiagnovirTester is a child class of Abstract User Class. The user enters the test results of the patients who have taken the test into the system. He/she can also view past test results of all patients and the time they took the test.

***SportStaff**:* SportStaff is a child class of Abstract User Class. The user can see the general weekly schedule of the reservations. This user reports to the system whether the users who have made a reservation show up or not. This allows the system to count the number of consecutively booked and absent users.

***Order**:* Order is an abstract class. This class contains common needs to create an order such as a place to make an order to there, time, date, users needed information such as which user makes an order and where this order go, and taken status which means a user makes an order and whether he/she takes/goes to the order. There are five different usage of this abstract class which are AmbulanceForm, Diagnovir, HealthForm, Meal, and SportActivity.

***AmbulanceForm**:* AmbulanceForm is a child class of Abstract Order Class. The patient sends information to the health center staff for the ambulance. He/she press callAmbulance button from the HealthManager page and this notification sends the health center worker the information which is in the AmbulanceForm. This form, which is created by the system, includes the patient's phone number, name, surname, where he/she stays in the campus and the date he/she sent this notification. When the health center workers see this notification, they call and contact the patient/user.

***Diagnovir**:* Diagnovir is a child class of Abstract Order Class. The patient and the diagnovir tester see the appointment date, place, and time of the patient when the patient makes a reservation under the title of abstract order class. They can also see the test result of the patient with this class object.

***HealthForm**:* HealthForm is a child class of Abstract Order Class. Health form contains different types of questions to determine whether the person who filled the form may have the covid or not. After the patient fills this form, if they may have the disease, the system sends this form to the health center worker and he/she makes an appointment for the patient while they are chatting.

***Meal**:* Meal is a child class of Abstract Order Class. The student or the instructor can order a meal. They can choose the types of meals which are normal, vegetarian or vegan. The order is sent to the cafeteria staff with the information as place where the meal will be sent, choice of the meal, and time by the system.

***SportActivity**:* SportActivity is a child class of Abstract Order Class. The student or the instructor can make a reservation with the type of the activity. They can choose the types of activities which are polygon, swimming, table tennis, fitness, basketball, tennis, football and so on. The reservation is sent to the sports staff with the information as place where the activity will be performed, user information, and time when the activity will be performed by the system.

***HesObject**:* With this class, users enter their hescode to the system and according to their hes status, they can benefit from the features in this program. The user can add only one hescode to the system and change their code if it is necessary.

***Lecture**:* Lecture class contains the instructor object because one lecture with the section has only one instructor and one instructor creates a unique lecture. Lecture has some features such as name, section, building to set a seating plan, weekly time in the schedule, course code to let students enroll this lecture, lecture code which is randomly generated by the system in order to check the HES status of the students every lesson.

***Seat**:* Seat class object is created when a lecture is generated by the instructor. Seat class keeps the student objects because every seat has to be taken by each student separately and one time since they will sit there until the end of the semester because of the pandemic period. When a lecture is started by the instructor by forming a lectureCode, students enter this code and according to their hes status, their seat status will be changed. If students have no disease and come to class, they will check whether they sit on their fixed seat by sending their randomly created seatCode to neighbours. That means, a student enters the neighbours' seatCode and if the neighbours' information match with the first time seating information, that means they always sit in the same place.

***LoginManager**:* Every user sees the same screen before logging into their account. Users write their mail addresses and passwords to this screen. If they write correctly, they open their own account.

***SettingsManager**:* Users can change some information such as mail address, password, phone number, their location address, HES code, and their profile picture. However, they cannot change their id numbers if it exists, their roles such as student to instructor or cafeteria staff. If they enter this information incorrectly at the beginning, they should contact the admins.

***DiagnovirManager**:* With this class users can see the MakeAnAppointment, TestResults, and CancelAppointment features if they have this DiagnovirManager page in their main screen.

***HealthManager**:* With this class users can see the ContactHealthCenter, CallAmbulance, and CancelAppointment features if they have this HealthManager page in their main screen.

***CafeteriaManager**:* With this class users can see the DailyMealNotification, MakeAMealOrder, and CancelMeal features if they have this CafeteriaManager page in their main screen.

***SportManager**:* With this class users can see the MakeSportsReservation and CancelSportsReservation features if they have this SportManager page in their main screen.

***LectureManager**:* With this class users can see the createCourse, changeSeatStatus, and enrollStudentToCourse regarding their user types.

***VaccineCard**:* VaccineCard class object is created for every single user. VaccineCard class keeps the user's vaccination information because every individual has to be under Covid regulations. This class offers 2 getter and 2 setter methods such as getFullVaccinated, setFullVaccinated, and if they have not taken the second shot getSecondVaccineDate, setSecondVaccineDate.

***Attendance**:* Attendance class object is created for every single user. Attendance class keeps the lectures' time and seating plan information because every student has to be seating at the same spot. This class offers 2 getter and 2 setter methods such as getTime, setTime, and for taking attendance getAttendanceRecord, and setAttendanceRecord.

3.4.3 State Diagram

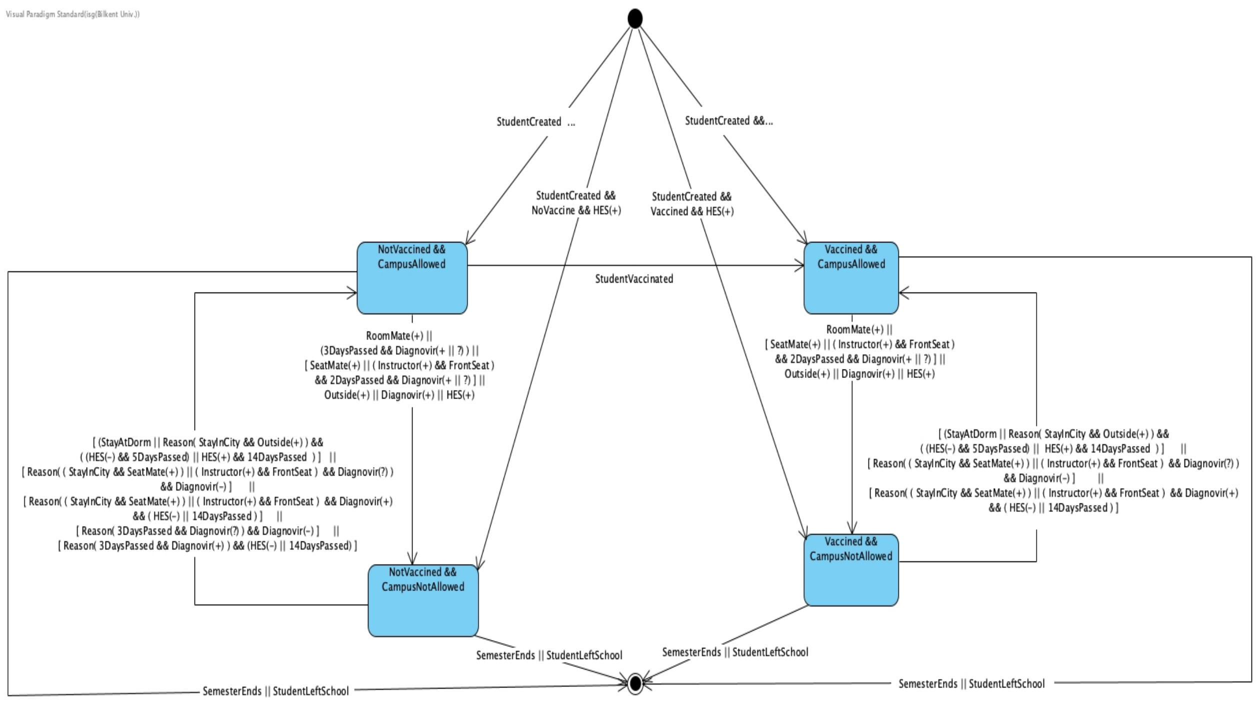

Visual Paradigm Standard(isg(Bilkent Univ.))
StudentCreated ...
StudentCreated &&...
StudentCreated && NoVaccine && HES(+)
StudentCreated && Vaccined && HES(+)
NotVaccined && CampusAllowed
StudentVaccinated
Vaccined && CampusAllowed
RoomMate(+) \|\| (3DaysPassed && Diagnovir(+ \|\| ?)) \|\| [SeatMate(+) \|\| (Instructor(+) && FrontSeat) && 2DaysPassed && Diagnovir(+ \|\| ?)] \|\| Outside(+) \|\| Diagnovir(+) \|\| HES(+)
RoomMate(+) \|\| [SeatMate(+) \|\| (Instructor(+) && FrontSeat) && 2DaysPassed && Diagnovir(+ \|\| ?)] \|\| Outside(+) \|\| Diagnovir(+) \|\| HES(+)
[(StayAtDorm \|\| Reason(StayInCity && Outside(+)) && ((HES(-) && 5DaysPassed) \|\| HES(+) && 14DaysPassed)] \|\| [Reason((StayInCity && SeatMate(+)) \|\| (Instructor(+) && FrontSeat) && Diagnovir(?)) && Diagnovir(-)] \|\| [Reason((StayInCity && SeatMate(+)) \|\| (Instructor(+) && FrontSeat) && Diagnovir(+) && (HES(-) \|\| 14DaysPassed)] \|\| [Reason(3DaysPassed && Diagnovir(?)) && Diagnovir(-)] \|\| [Reason(3DaysPassed && Diagnovir(+)) && (HES(-) \|\| 14DaysPassed)]
[(StayAtDorm \|\| Reason(StayInCity && Outside(+)) && ((HES(-) && 5DaysPassed) \|\| HES(+) && 14DaysPassed)] \|\| [Reason((StayInCity && SeatMate(+)) \|\| (Instructor(+) && FrontSeat) && Diagnovir(?)) && Diagnovir(-)] \|\| [Reason((StayInCity && SeatMate(+)) \|\| (Instructor(+) && FrontSeat) && Diagnovir(+) && (HES(-) \|\| 14DaysPassed)]
NotVaccined && CampusNotAllowed
Vaccined && CampusNotAllowed
SemesterEnds \|\| StudentLeftSchool
SemesterEnds \|\| StudentLeftSchool
SemesterEnds \|\| StudentLeftSchool
SemesterEnds \|\| StudentLeftSchool

Fig. 3: Students Campus Allowance State Diagram for the Pandemic Management System

This state diagram shows the main state transition of students' campus allowance status. There are two different procedures for students depending on their vaccine status. Their vaccines and HES status will determine students' initial status. In the first part, the vaccinated student's procedure will be elucidated. Suppose a student gets contacted and resides in dormitories. In that case, he/she will be quarantined in quarantine dormitories for at least 5 days regardless of the reason. Depending on the student's diagnovir test result, he/she can stay for 9 more days in the quarantine dorm or can go to his/her original dorm. For the students that do not reside in dormitories, if a student's seatmate gets sick or the course instructor gets sick and student seats in the first row, he/she has 48 hours to get a diagnovir test. Depending on the result, students can be sent to take the PCR test, and if the PCR test result is positive, students' allowance status will switch to not allowed status for 14 days.
In the second part, unvaccinated students' procedure will be elucidated. These students need to get a diagnovir test once every 3 days to remain in campus allowed status. Except for this control mechanism, if their seatmate gets sick or the course instructor gets sick, and the student sits in the first row, he/she has 48 hours to get a diagnovir

test. Depending on the result, students can be sent to take the PCR test, and if the PCR test result is positive, students' allowance status will switch to not allowed status for 14 days. These students can get vaccinated while not sick and become subject to vaccinated students' procedure.

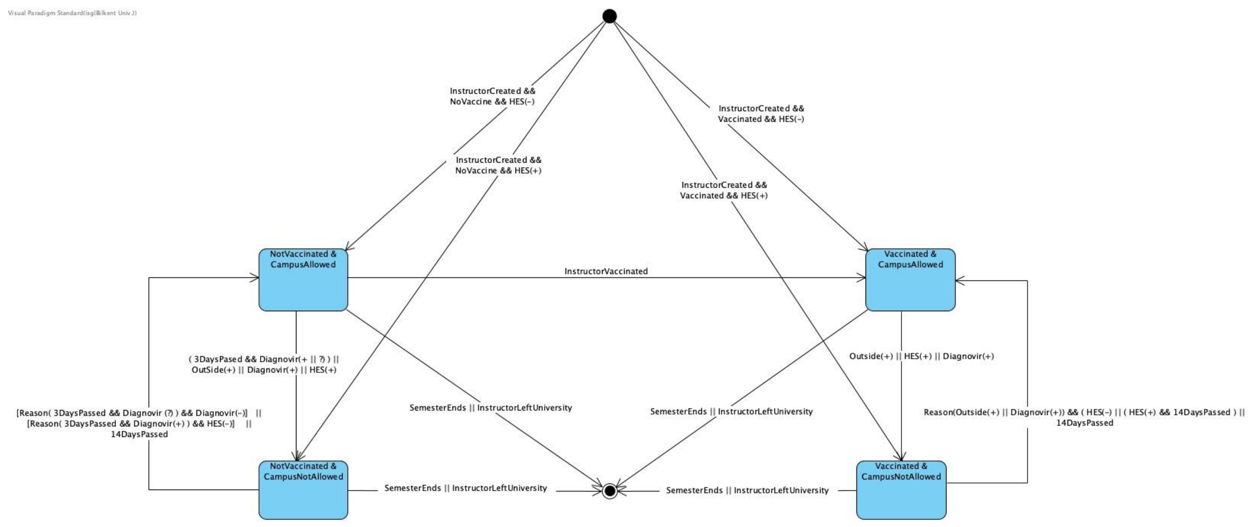

Visual Paradigm Standard(isg(Bilkent Univ.))
InstructorCreated && NoVaccine && HES(-)
InstructorCreated && NoVaccine && HES(+)
InstructorCreated && Vaccinated && HES(-)
InstructorCreated && Vaccinated && HES(+)
NotVaccinated & CampusAllowed
InstructorVaccinated
Vaccinated & CampusAllowed
(3DaysPased && Diagnovir(+ || ?)) || OutSide(+) || Diagnovir(+) || HES(+)
Outside(+) || HES(+) || Diagnovir(+)
[Reason(3DaysPassed && Diagnovir (?)) && Diagnovir(-)] || [Reason(3DaysPassed && Diagnovir(+)) && HES(-)] || 14DaysPassed
SemesterEnds || InstructorLeftUniversity
SemesterEnds || InstructorLeftUniversity
Reason(Outside(+) || Diagnovir(+)) && (HES(-) || (HES(+) && 14DaysPassed) || 14DaysPassed
NotVaccinated & CampusNotAllowed
SemesterEnds || InstructorLeftUniversity
SemesterEnds || InstructorLeftUniversity
Vaccinated & CampusNotAllowed

Fig. 4: Instructor Campus Allowance State Diagram for the Pandemic Management System

Instructors have less risk factors. They do not have roommates or seatmates as risk factors. They also have the same campus allowance states with students. Considering an instructor is vaccinated their campus allowance status can be changed by giving positive PCR or diagnovir test. Also, their status can be changed if people around them, such as family members or recently met people, are infected with covid. The infected instructor will be allowed on campus after 14 days.
Considering an instructor is not vaccinated, the instructor needs to get diagnovir test once every 3 days to remain in campus allowed status. Except for this control mechanism, they have the same risk factors as vaccinated instructors. They also need 14 days to be campus allowed if they become infected. Lastly, unvaccinated instructors can get vaccinated while not sick and become subject to vaccinated students' procedure.

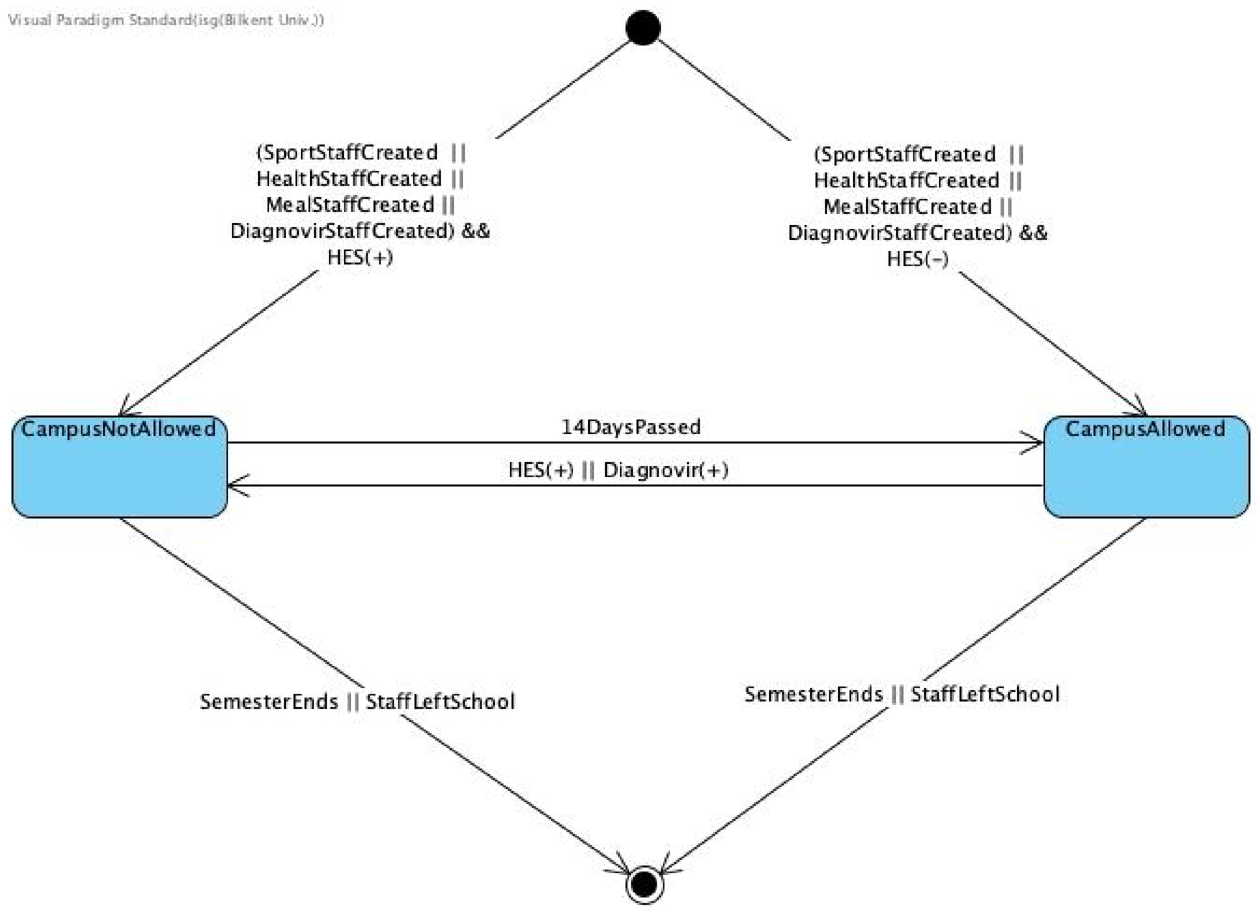

Visual Paradigm Standard(isg(Bilkent Univ.))
(SportStaffCreated || HealthStaffCreated || MealStaffCreated || DiagnovirStaffCreated) && HES(+)
(SportStaffCreated || HealthStaffCreated || MealStaffCreated || DiagnovirStaffCreated) && HES(-)
CampusNotAllowed
CampusAllowed
14DaysPassed
HES(+) || Diagnovir(+)
SemesterEnds || StaffLeftSchool
SemesterEnds || StaffLeftSchool

Fig. 5: Other Staff Campus Allowance State Diagram for the Pandemic Management System

There is an assumption made for this state diagram. Bilkent does not let their employees (except for instructors and managers) remain unvaccinated. This is why there are only 2 states in this diagram. If they take a PCR test or diagnovir test and get a positive result, they will not be allowed on campus for 14 days.

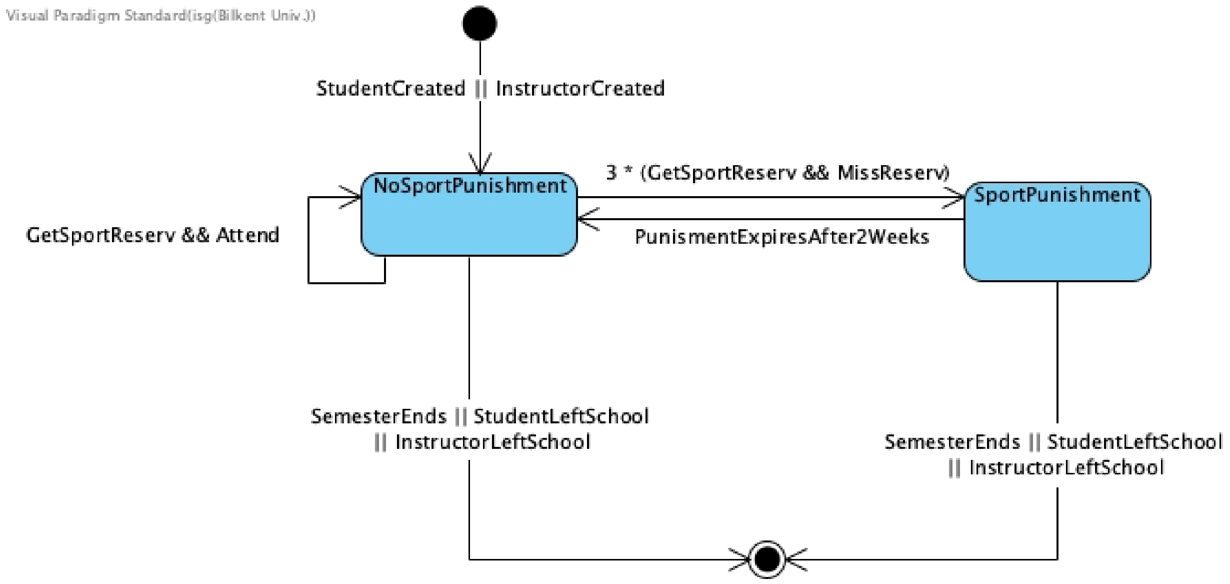

Visual Paradigm Standard(isg(Bilkent Univ.))
StudentCreated || InstructorCreated
NoSportPunishment
3 * (GetSportReserv && MissReserv)
SportPunishment
PunismentExpiresAfter2Weeks
GetSportReserv && Attend
SemesterEnds || StudentLeftSchool || InstructorLeftSchool
SemesterEnds || StudentLeftSchool || InstructorLeftSchool

Fig. 6: Student State Diagram for the Pandemic Management System

This state diagram illustrates the management of sport activities in the pandemic management system. If a student or an instructor does not attend their reservation three times without canceling, they will be punished by not getting a sport reservation for 2 weeks.

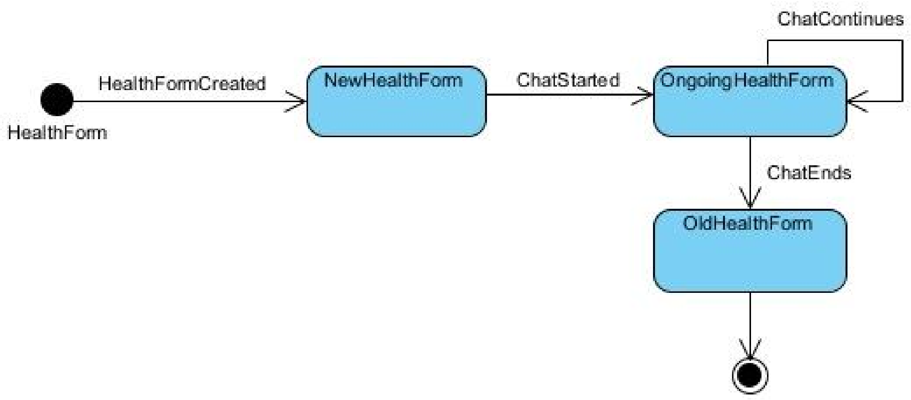


Fig. 7: Health Form State Diagram for the Pandemic Management System

This state diagram sketches the main states of health forms in the pandemic management system. After it is created, the health form will be classified as a new health form. The form's status will be changed to an ongoing health form after the health staff starts the chat with the applicant. In this state, a doctor appointment can be arranged if necessary. Health staff end chat when there is no need to chat with applicant anymore (either applicant is healed or health staff concludes that there is no need for doctor appointment).

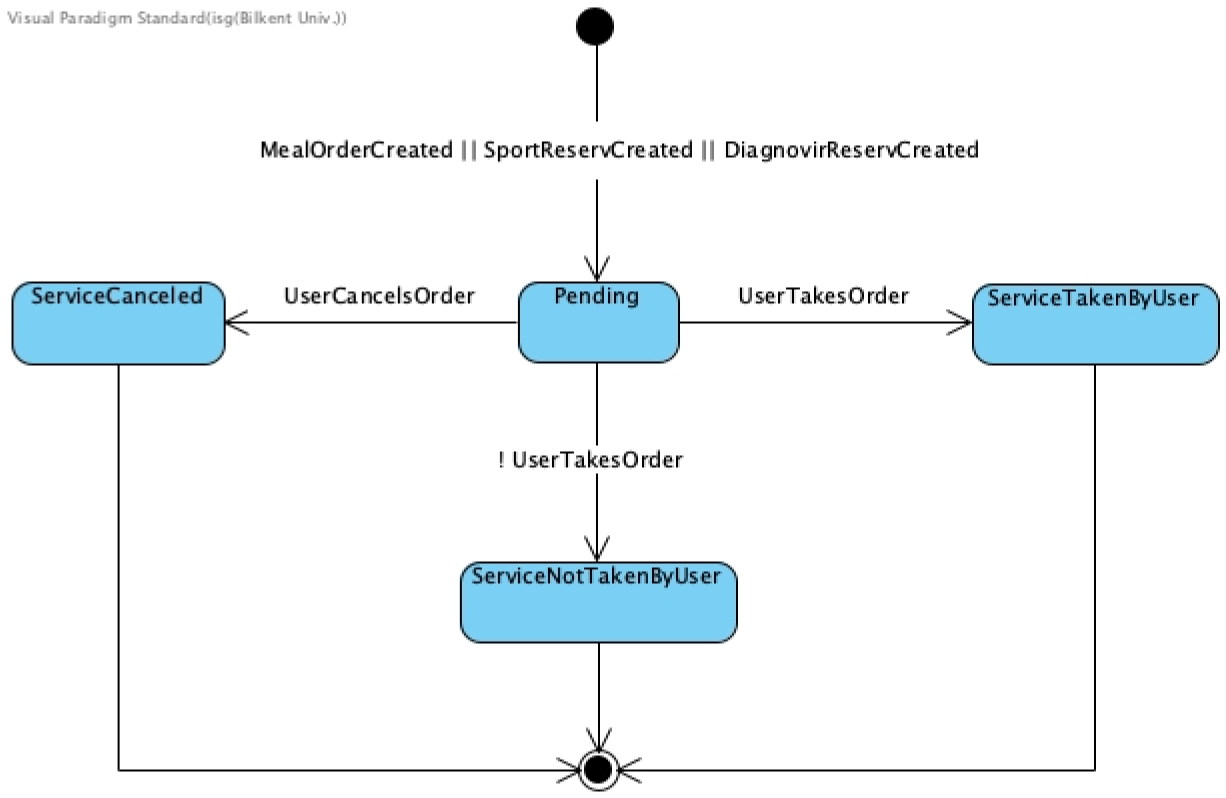

Visual Paradigm Standard(isg(Bilkent Univ.))
MealOrderCreated || SportReservCreated || DiagnovirReservCreated
ServiceCanceled
UserCancelsOrder
Pending
UserTakesOrder
ServiceTakenByUser
! UserTakesOrder
ServiceNotTakenByUser

Fig. 8: Orders State Diagram for the Pandemic Management System

This state diagram represents the mechanism behind meal orders, sports reservations and Diagnovir reservations. The orders will be in a pending state right after creation. Users can cancel their meal order, sports reservation or Diagnovir reservation which puts the order in the ServiceCanceled state. The orders will pass to the ServiceTakenByUser state if the users get their order or attend their reservation in time. Contrarily, if users did not cancel their orders or reservations and did not get their orders or attend their reservations, the orders will pass to the ServiceNotTakenbyUser state.

3.4.4 Activity Diagram

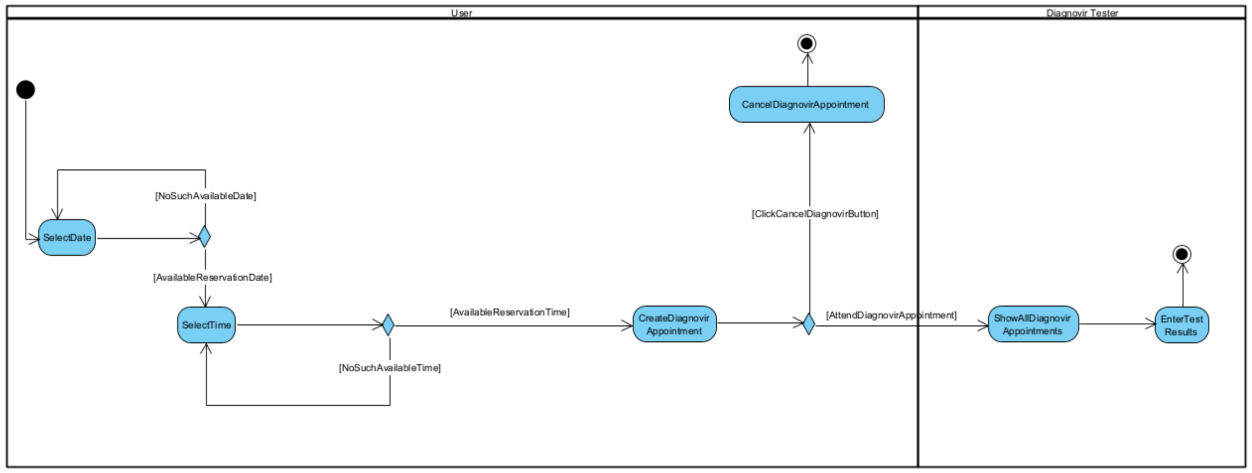

User
Diagnovir Tester
SelectDate
[NoSuchAvailableDate]
[AvailableReservationDate]
SelectTime
[AvailableReservationTime]
[NoSuchAvailableTime]
CreateDiagnovir Appointment
[ClickCancelDiagnovirButton]
CancelDiagnovirAppointment
[AttendDiagnovirAppointment]
ShowAllDiagnovir Appointments
EnterTest Results

Fig. 9: Diagnovir Test Appointment Activity Diagram for the Pandemic Management System

This activity diagram shows the activities related to Diagnovir test appointments. To create an appointment, the user first needs to select an appropriate date and time. If the chosen date or time does not have an empty slot, users have to select another one. After the selection of available time slots, the Diagnovir test appointment is created. After the creation of the appointment, users can cancel them. In addition to these, Diagnovir testers can see all test appointments and enter their results.

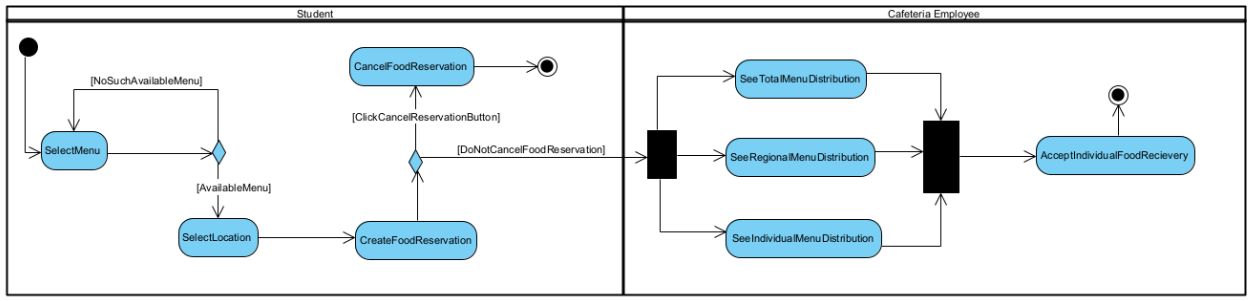


Fig. 10: Cafeteria Food Reservation Activity Diagram for the Pandemic Management System

This activity diagram shows the activities related to cafeteria food reservations. To create a reservation, the user needs to select a menu. If the selected menu is not available, they have to reselect another menu. After choosing the menu, they have to choose a location where they want to get their food to complete the reservation process. After the creation of the appointment, users can cancel them. However, users cannot cancel their food reservations if there are less than thirty minutes for the food delivery. The cafeteria staff can see different types of data related to these reservations. They can see individual, menu-based, and regional food distributions to help them in different parts of their work. In addition to these, they also can record whether people who made a reservation came to get their food. This aims to prevent people from making reservations and not taking their food.

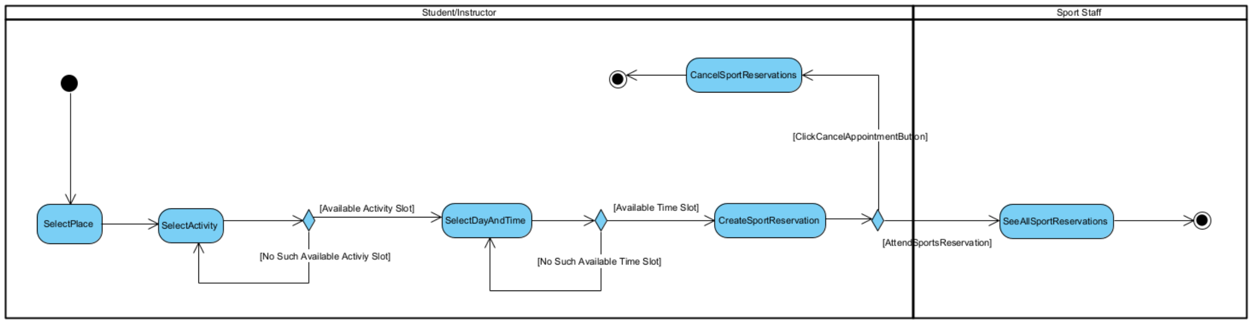


Fig. 11: Sports Appointment Activity Diagram for the Pandemic Management System

This activity diagram shows the activities related to sports appointments. To create an appointment, the user needs to select a place, activity, and time slot. If the selected time slot is not available, they have to reselect it. After the creation of the appointment, users can cancel them. In addition, sports staff can see all the reservations and record whether people who made an appointment came to their appointment. This aims to prevent people from making appointments and not coming to them.

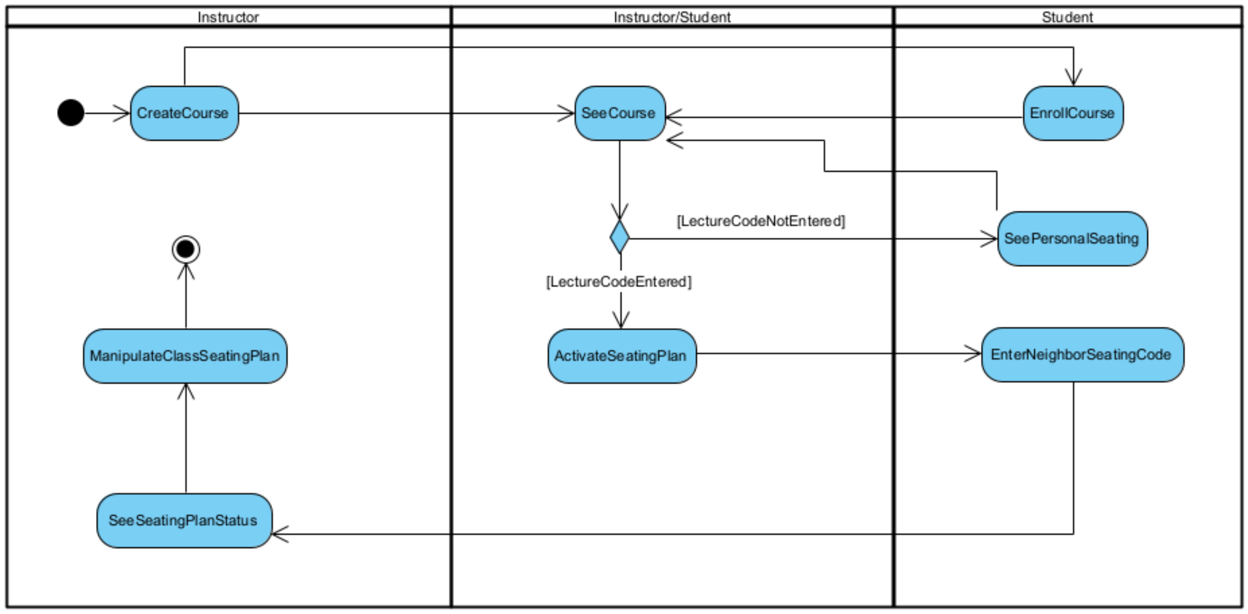


Fig. 12: Class Seating Plan Activity Diagram for the Pandemic Management System

This activity diagram shows the activities starting from the creation of a class until a seating plan is decided. Firstly, classes are created by instructors, and after that, students can enroll in them. After these, both instructors and students can see their classes. In each class, if the randomized lecture code is not generated, students can see their seats. After the lecture code is generated, the seating plan becomes activated, and students can enter their neighbor-student's seating code. After students enter these, instructors can see who is sitting in the correct place and who is not. In this state, if a student does not have access to the application, instructors can check their application and add that student to the seating plan.

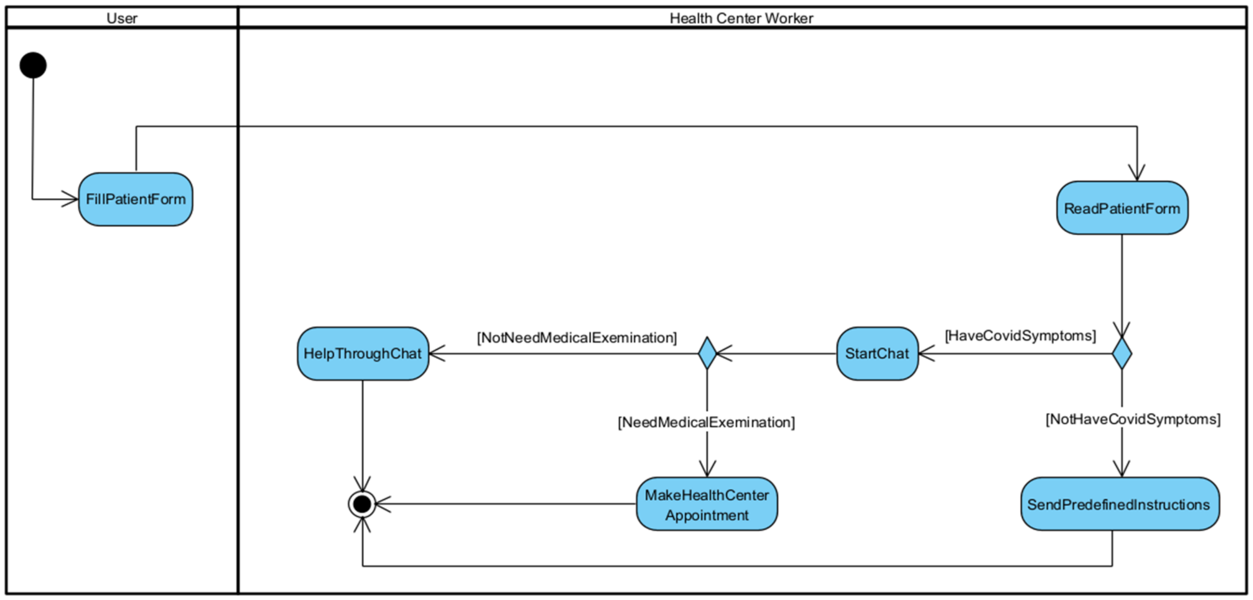


Fig. 13: Contacting Health Center Activity Diagram for the Pandemic Management System

This activity diagram shows the activities related to health center-user communication. Firstly, the patient fills the general symptom form. Then, the health center worker reads it. If the patient has covid symptoms, the health center worker starts a chat with the patient. If the patient does not have covid symptoms, some predefined instructions about what they can do to feel better are sent to them. If the patient has the covid symptoms, make an appointment, or help through the chat based on the severity.

3.4.5 Sequence Diagram

Scenario 1: Student Selects the Seats, Confirm Neighbor Seats and Instructor Creates Random Lecture Code

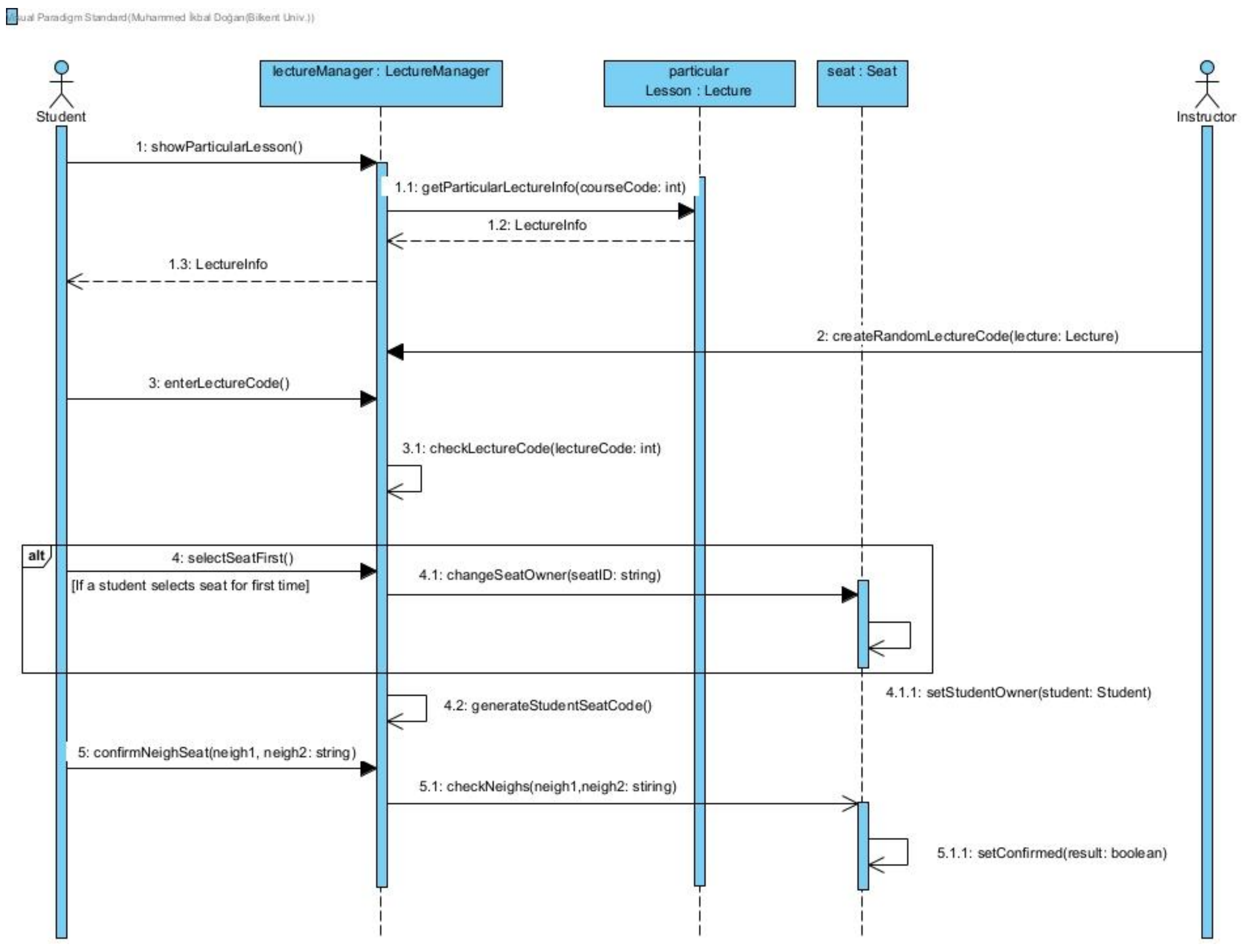

Visual Paradigm Standard(Muhammed İkbal Doğan(Bilkent Univ.))
Student
lectureManager : LectureManager
particular Lesson : Lecture
seat : Seat
Instructor
1: showParticularLesson()
1.1: getParticularLectureInfo(courseCode: int)
1.2: LectureInfo
1.3: LectureInfo
2: createRandomLectureCode(lecture: Lecture)
3: enterLectureCode()
3.1: checkLectureCode(lectureCode: int)
alt
4: selectSeatFirst()
[If a student selects seat for first time]
4.1: changeSeatOwner(seatID: string)
4.1.1: setStudentOwner(student: Student)
4.2: generateStudentSeatCode()
5: confirmNeighSeat(neigh1, neigh2: string)
5.1: checkNeighs(neigh1,neigh2: stiring)
5.1.1: setConfirmed(result: boolean)

Fig.14: Student Selects Seat, Confirm Neighbor Seats and Instructor Creates Random Lecture Code Sequence Diagram

Students can show their particular lesson from their Lecture Page. They can click a particular lesson for getting its information. If Instructors create a random lecture code, students can see the seating plan information in the particular lesson. If the students enter the given lecture code for the first time, they can select their seats at the beginning of the semester. After the first time, they cannot change their seats. Then students enter their neighbors' seat code for checking whether they seat in the right place or not.

Scenario 2: Users Take a Diagnovir Test Reservation

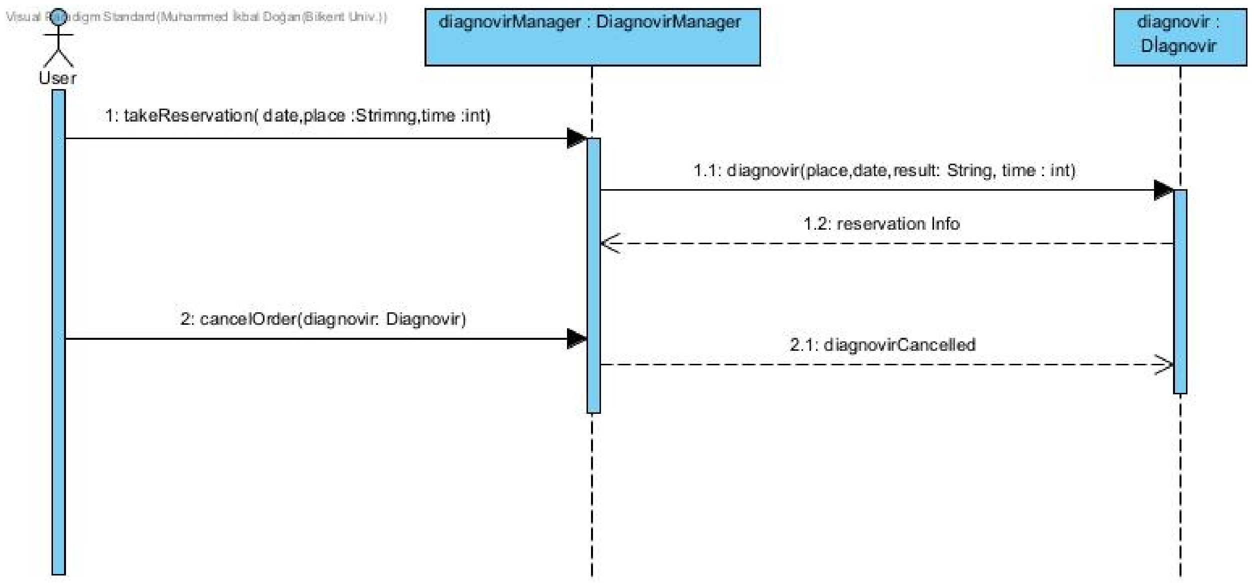

Visual Paradigm Standard(Muhammed İkbal Doğan(Bilkent Univ.))
User
diagnovirManager : DiagnovirManager
diagnovir : Diagnovir
1: takeReservation(date,place :Strimng,time :int)
1.1: diagnovir(place,date,result: String, time : int)
1.2: reservation Info
2: cancelOrder(diagnovir: Diagnovir)
2.1: diagnovirCancelled

Fig.15: Diagnovir Test Reservations of the Users Sequence Diagram

Users can take diagnovir test reservation and see their reservation. Also they can cancel their reservation.

Scenario 3: Users Take a Health Form

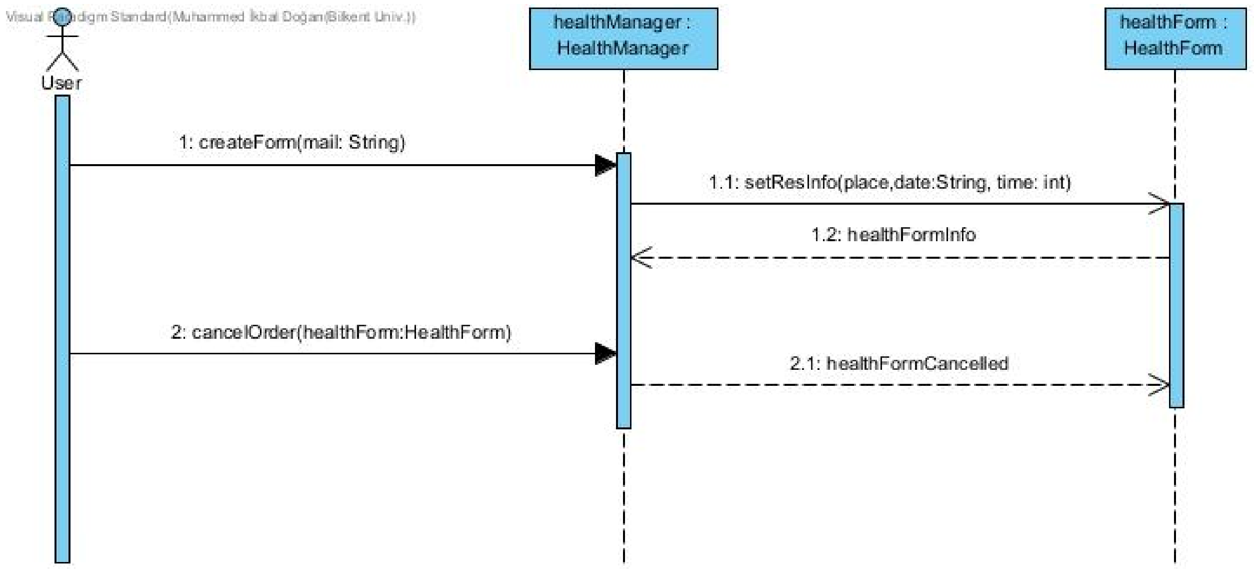

Visual Paradigm Standard(Muhammed İkbal Doğan(Bilkent Univ.))
User
healthManager : HealthManager
healthForm : HealthForm
1: createForm(mail: String)
1.1: setResInfo(place,date:String, time: int)
1.2: healthFormInfo
2: cancelOrder(healthForm:HealthForm)
2.1: healthFormCancelled

Fig.16: Users Take Health Form Sequence Diagram

Users can create a health form, show their form information and before the reservation time, they can cancel their health form reservation.

Scenario 4: Student Meal Selection and Cancellation

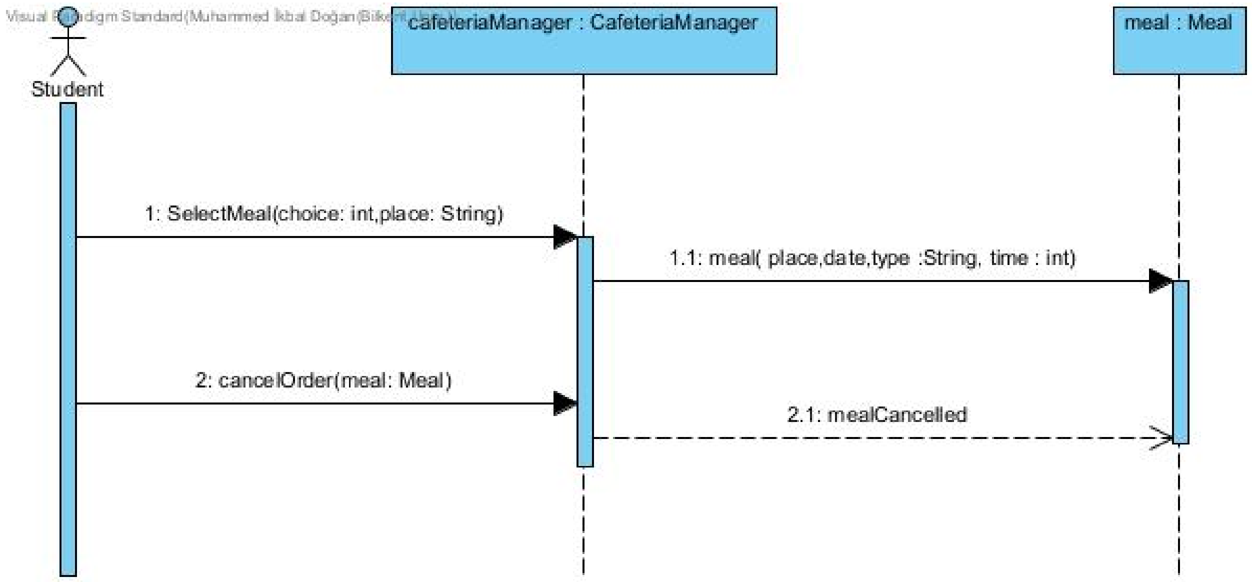
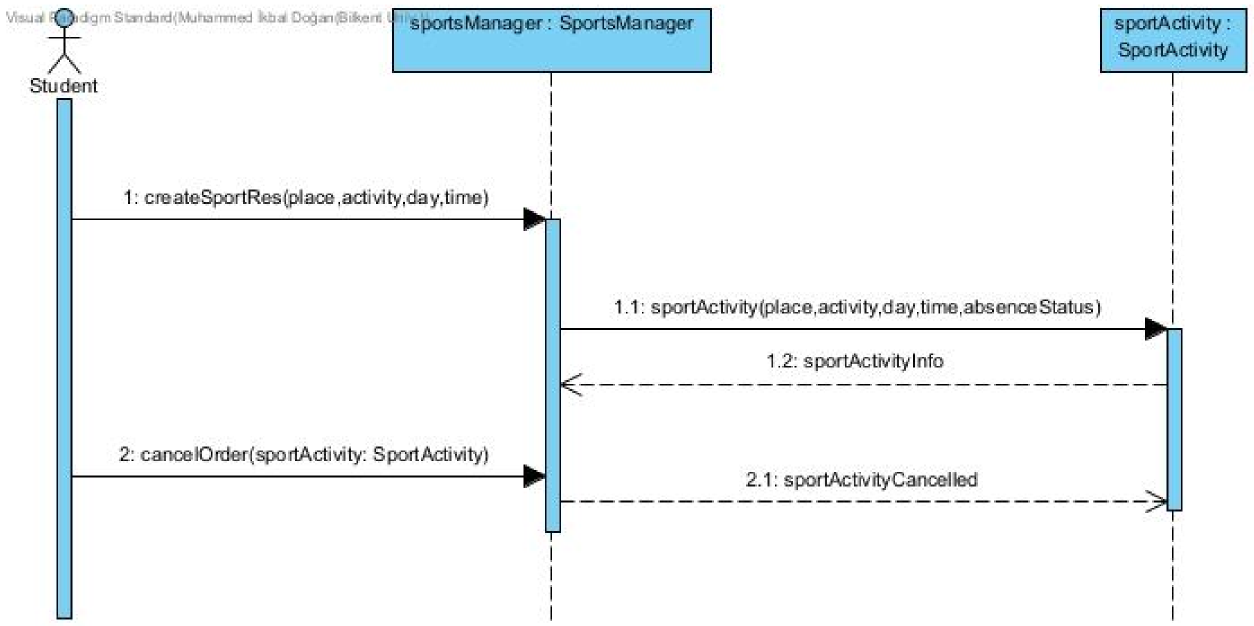

Visual Paradigm Standard(Muhammed İkbal Doğan(Bilkent Univ.))
Student
cafeteriaManager : CafeteriaManager
meal : Meal
1: SelectMeal(choice: int,place: String)
1.1: meal(place,date,type :String, time : int)
2: cancelOrder(meal: Meal)
2.1: mealCancelled

Fig.17: Students Meal Selection and Cancellation Sequence Diagram

Students can select their meals by selecting choice and place therefore a meal order would be created according to meal time,place and selection.On the other hand, if a student wants to cancel the order, by selecting meal, order can be canceled.

Scenario 5: Student Sports Activity Selection and Cancellation



Visual Paradigm Standard(Muhammed İkbal Doğan(Bilkent Univ.))
Student
sportsManager : SportsManager
sportActivity : SportActivity
1: createSportRes(place,activity,day,time)
1.1: sportActivity(place,activity,day,time,absenceStatus)
1.2: sportActivityInfo
2: cancelOrder(sportActivity: SportActivity)
2.1: sportActivityCancelled

Fig.18:Student SportActivity Create and Cancellation Sequence Diagram

Students can create sports reservations by selecting place, activity, day and time. Then firstly their absence status would be true. If a student attends an activity, absence would become false. Before the deadline, students can cancel their reservation, so sport activity would be deleted in that case.

3.4.6 User Interface

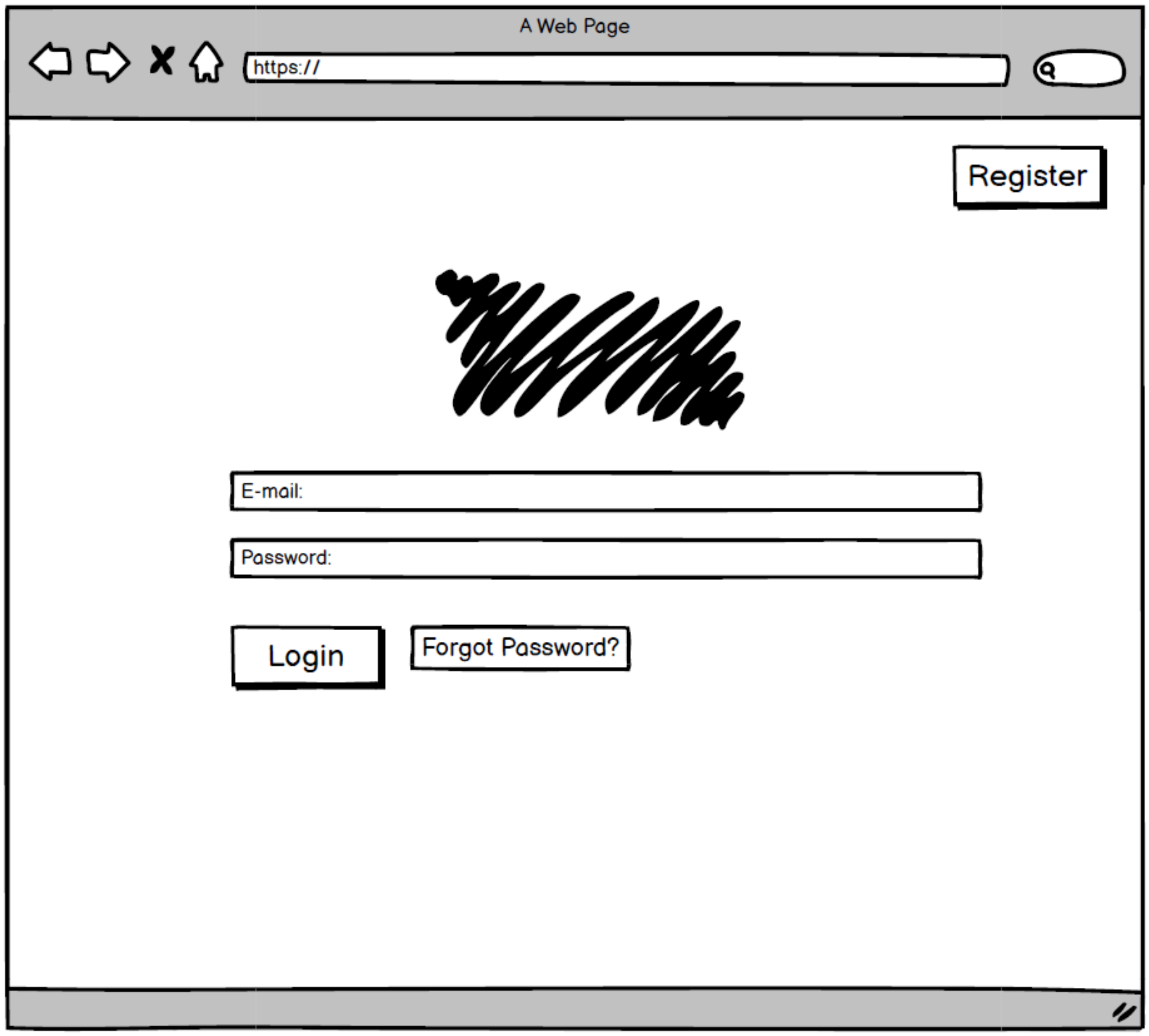


Fig.26: Login Screen

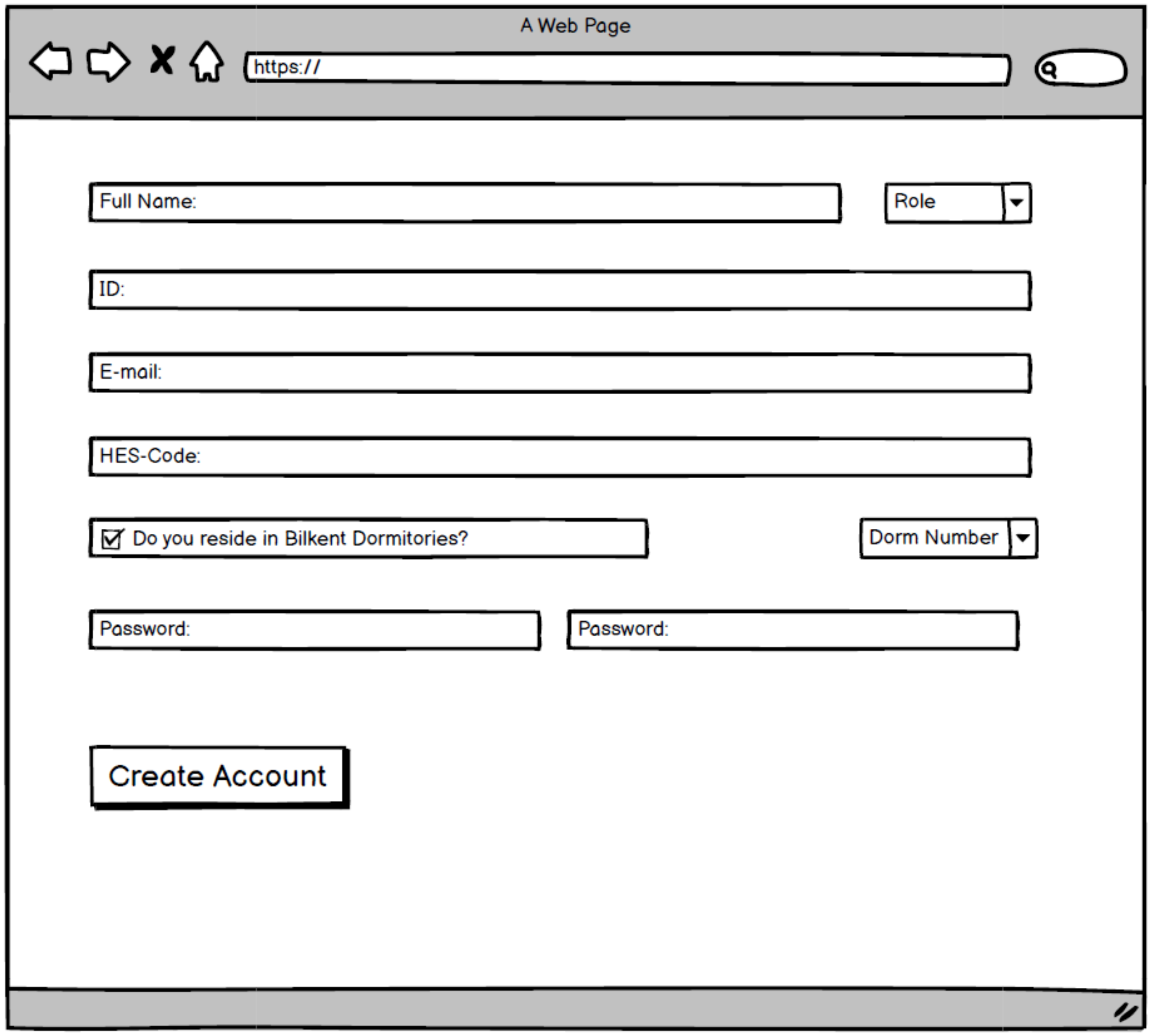


Fig.27: Register Screen

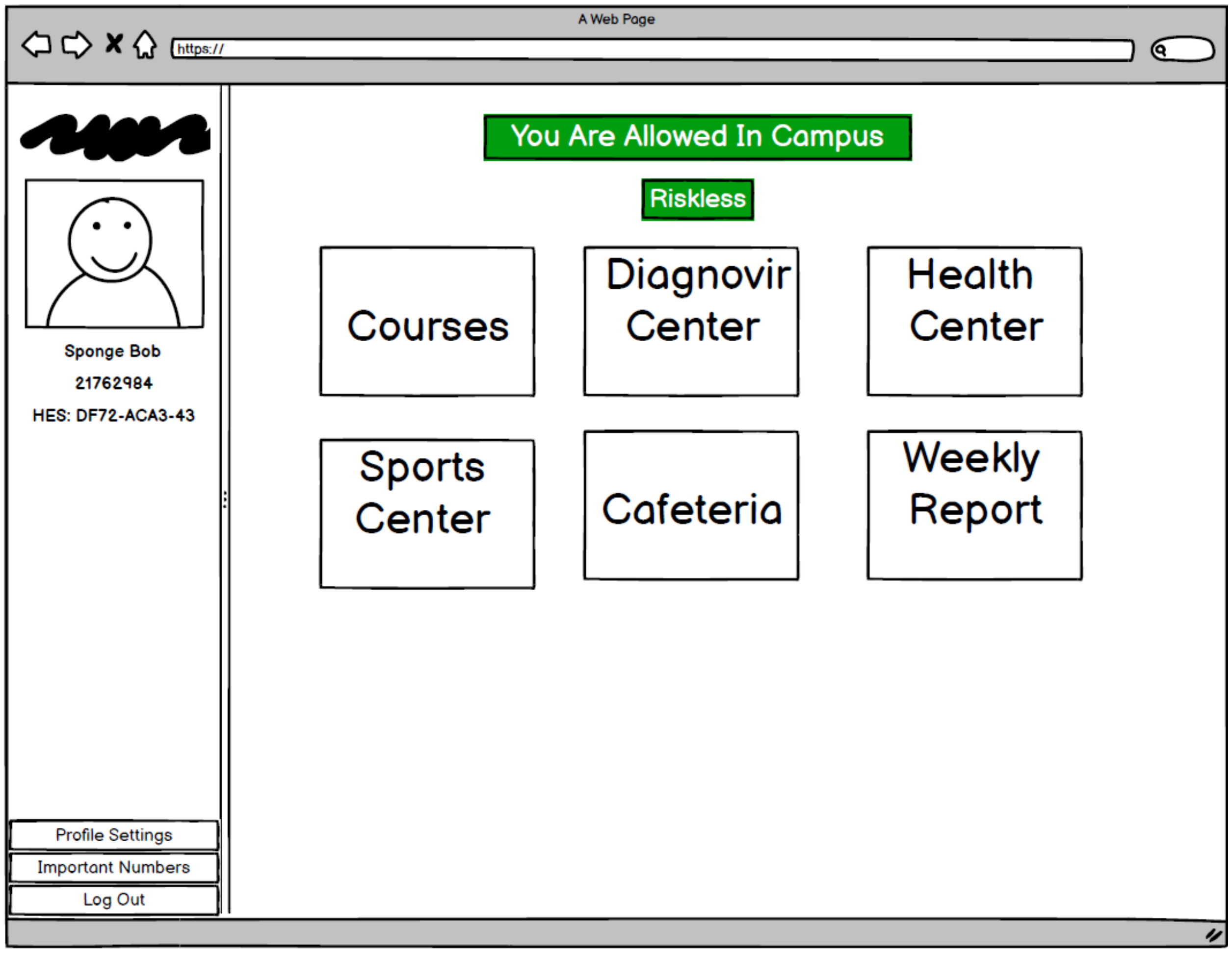


Fig.28: Main Page Campus Allowed

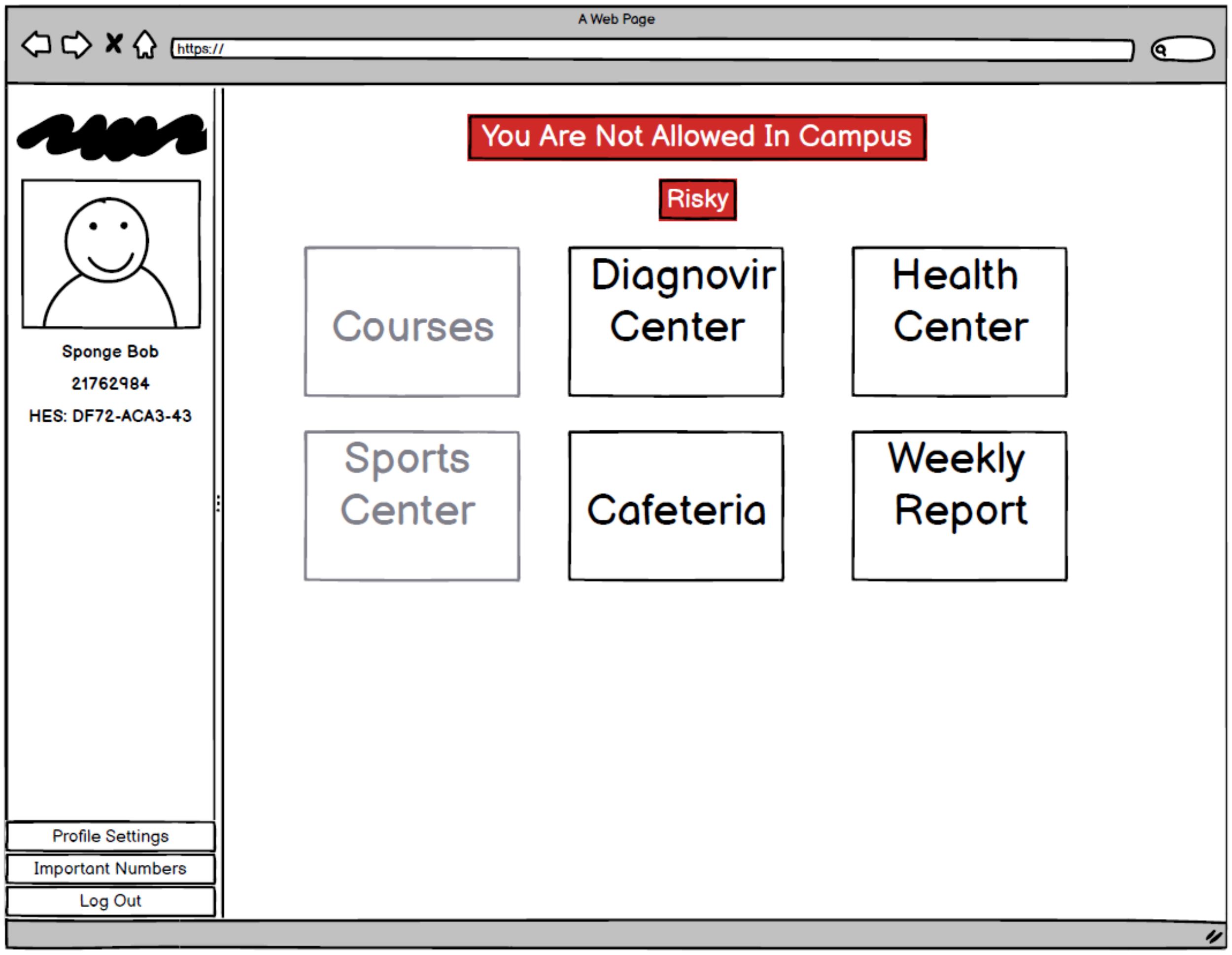


Fig.29: Main Page Campus Not Allowed

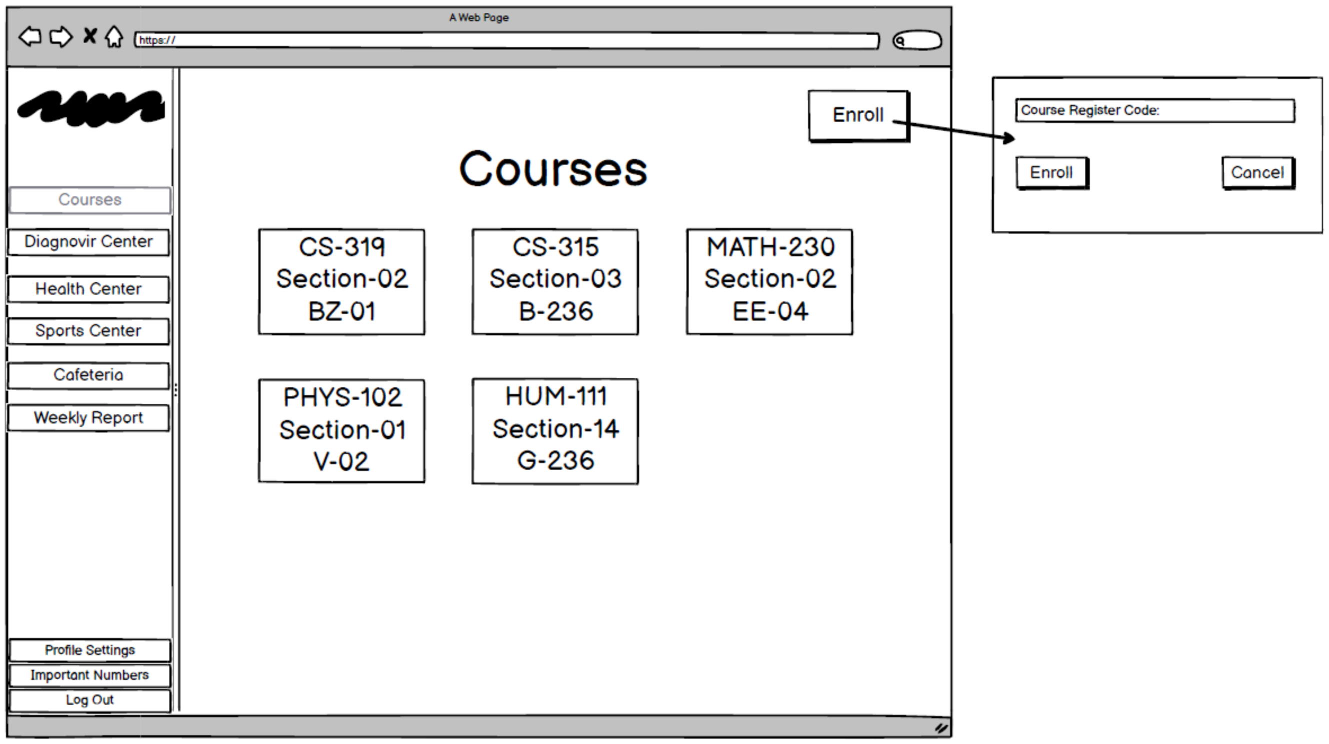


Fig.30: Courses Page for Students

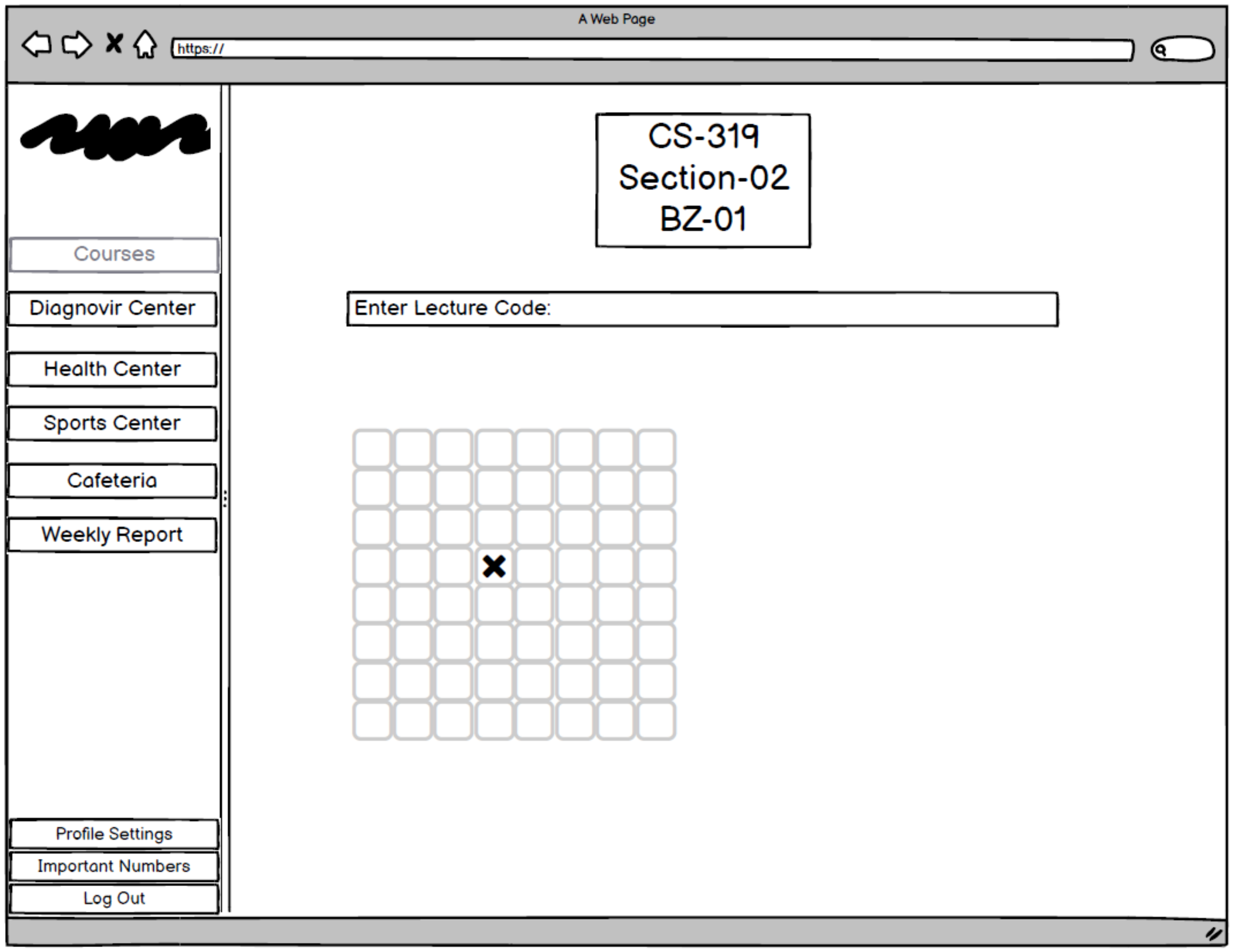


Fig.31: Inactive Seating Plan for Student

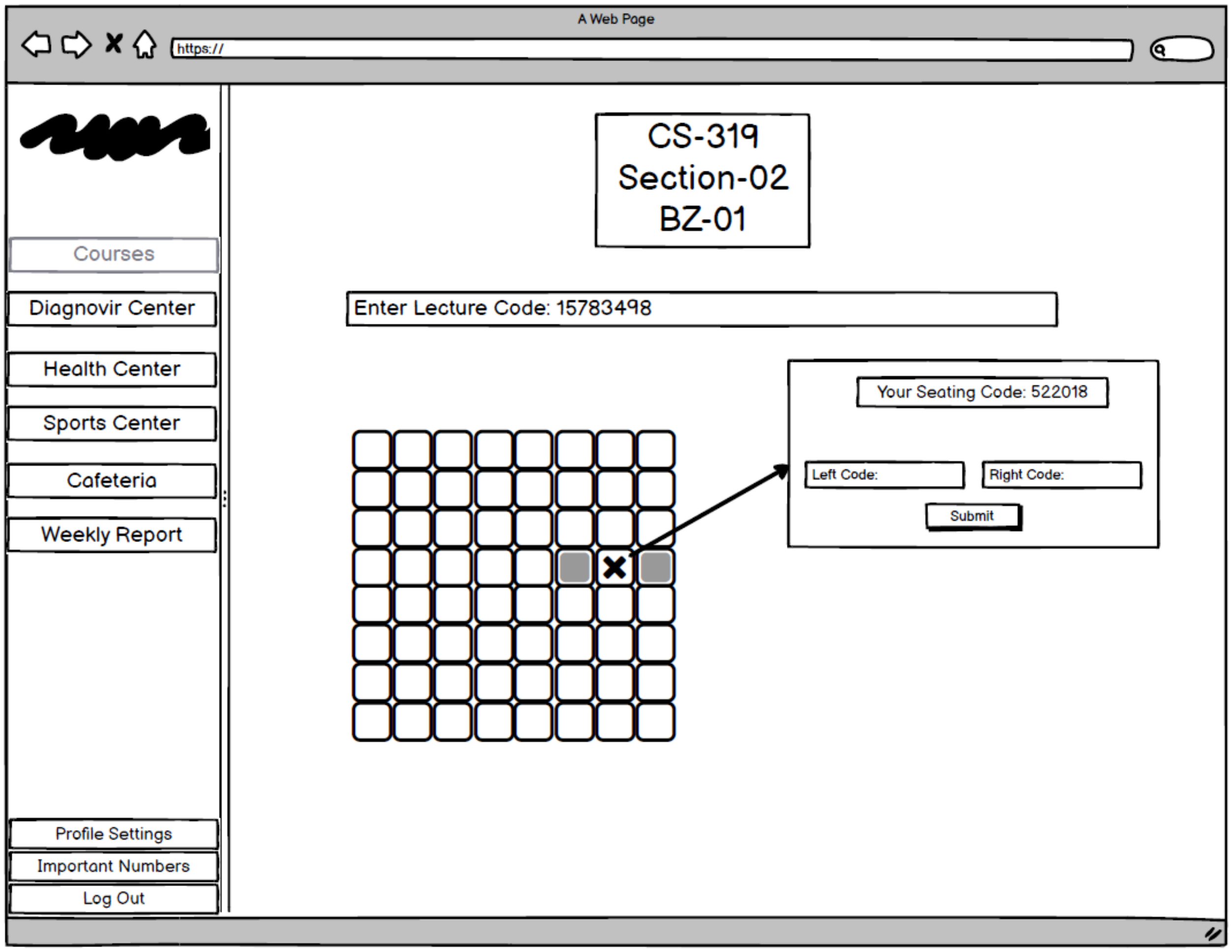
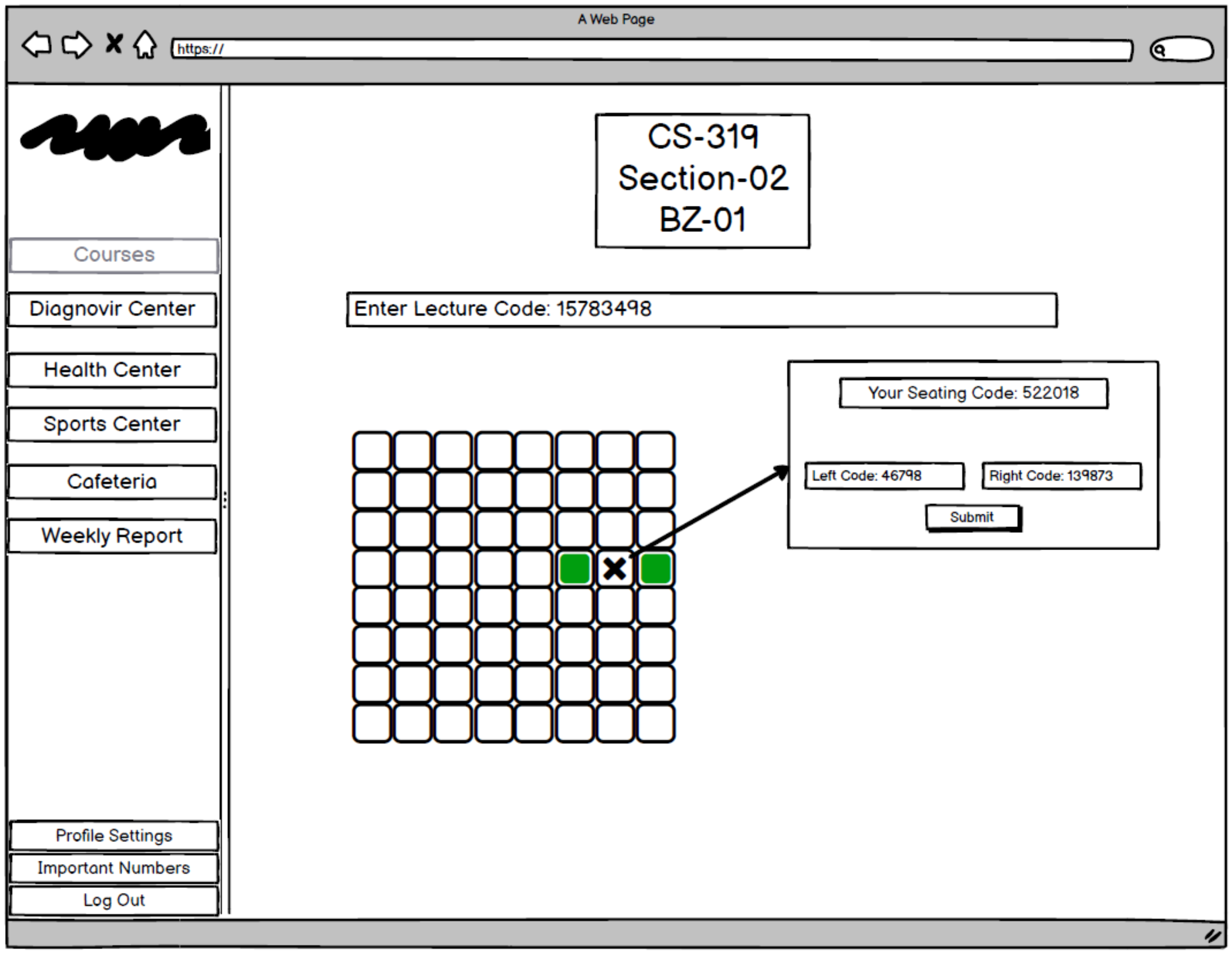


Fig.32: Active Seating Plan for Students (2 Neighbors / waiting)



CS-319
Section-02
BZ-01

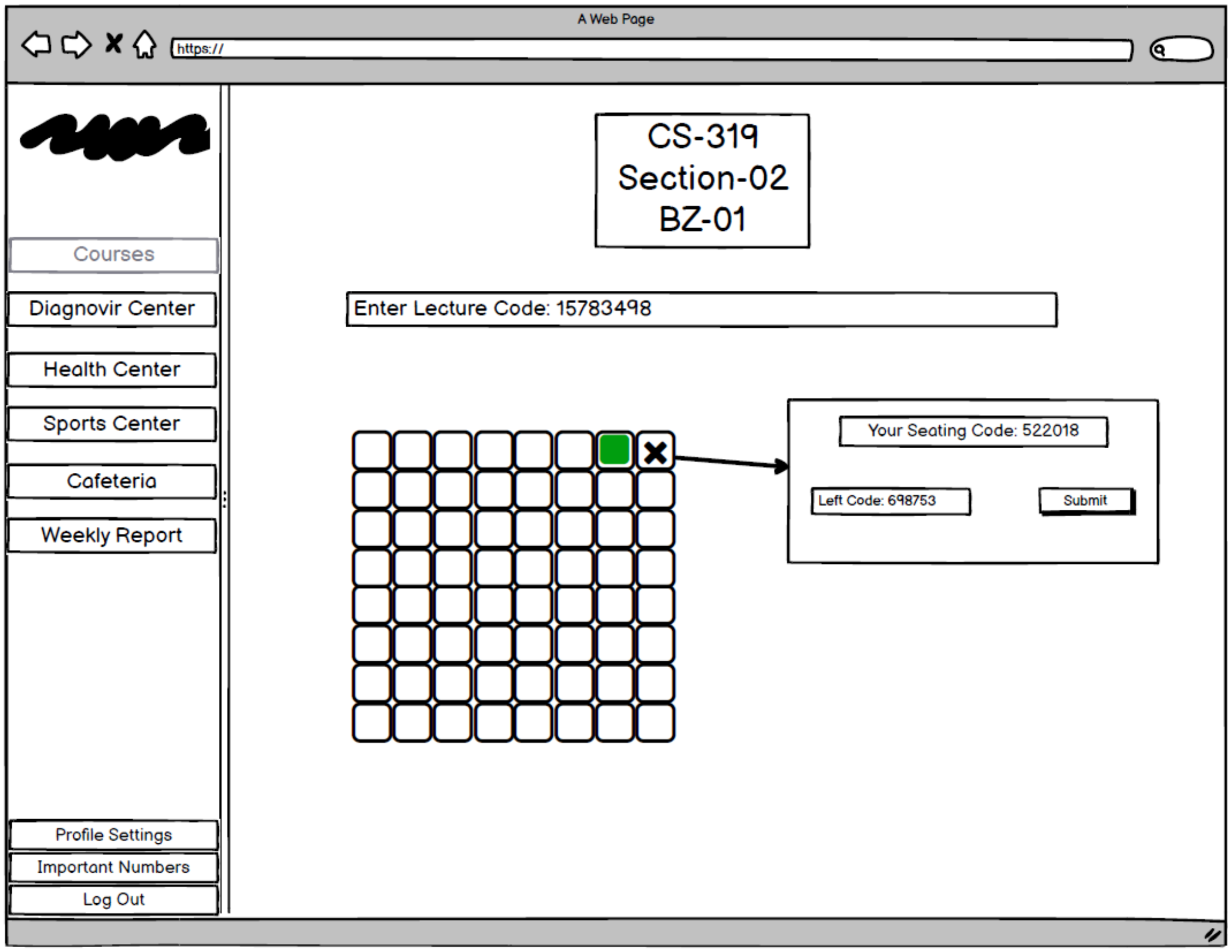
Enter Lecture Code: 15783498

Your Seating Code: 522018

Left Code: 46798

Right Code: 139873

Fig.33: Active Seating Plan for Students (2 Neighbors / done)



CS-319
Section-02
BZ-01

Enter Lecture Code: 15783498

Your Seating Code: 522018

Left Code: 698753

Fig.34: Active Seating Plan for Students (1 Neighbor / done)

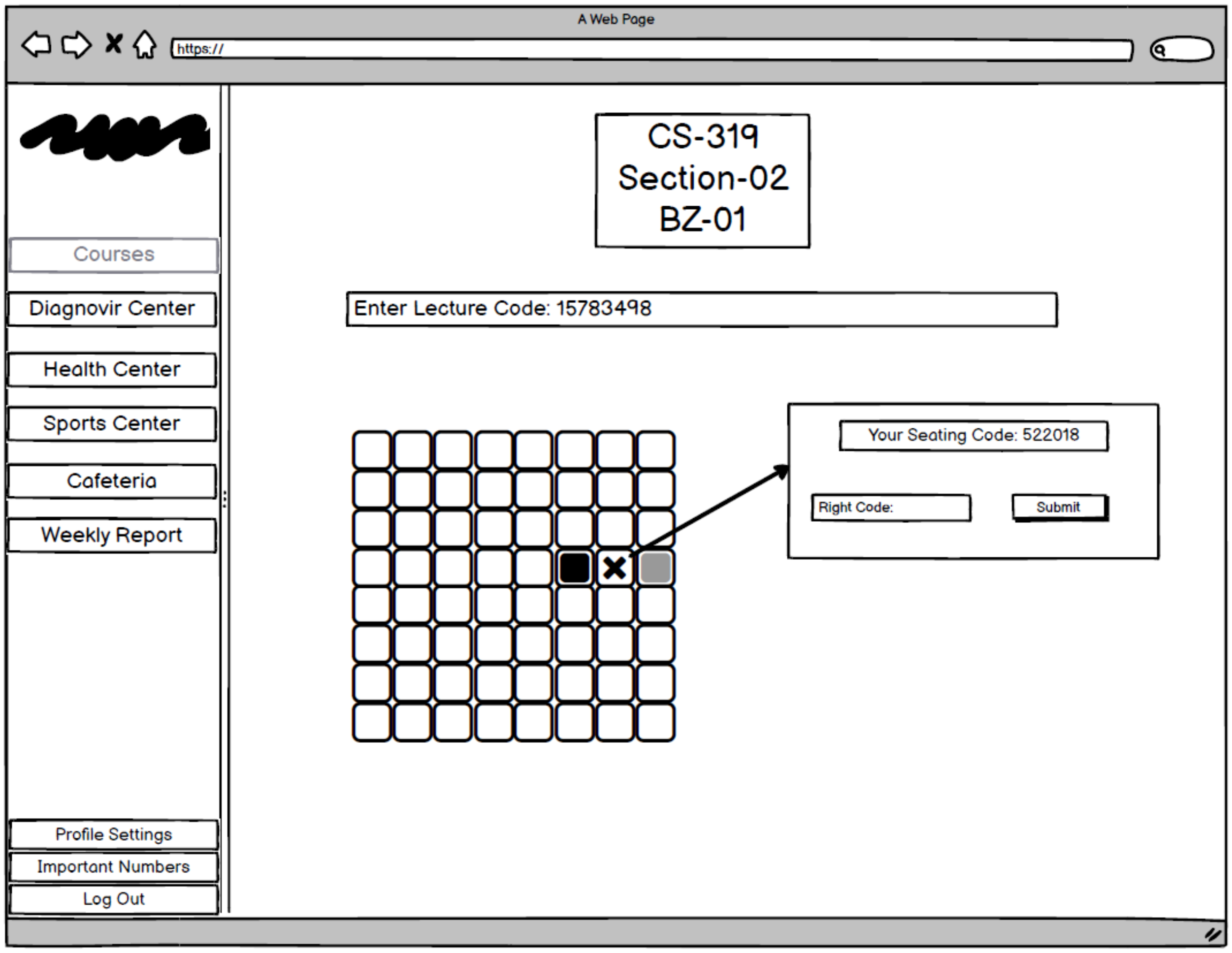


Fig.35: Active Seating Plan for Students (1 Neighbor and 1 Missing / waiting)

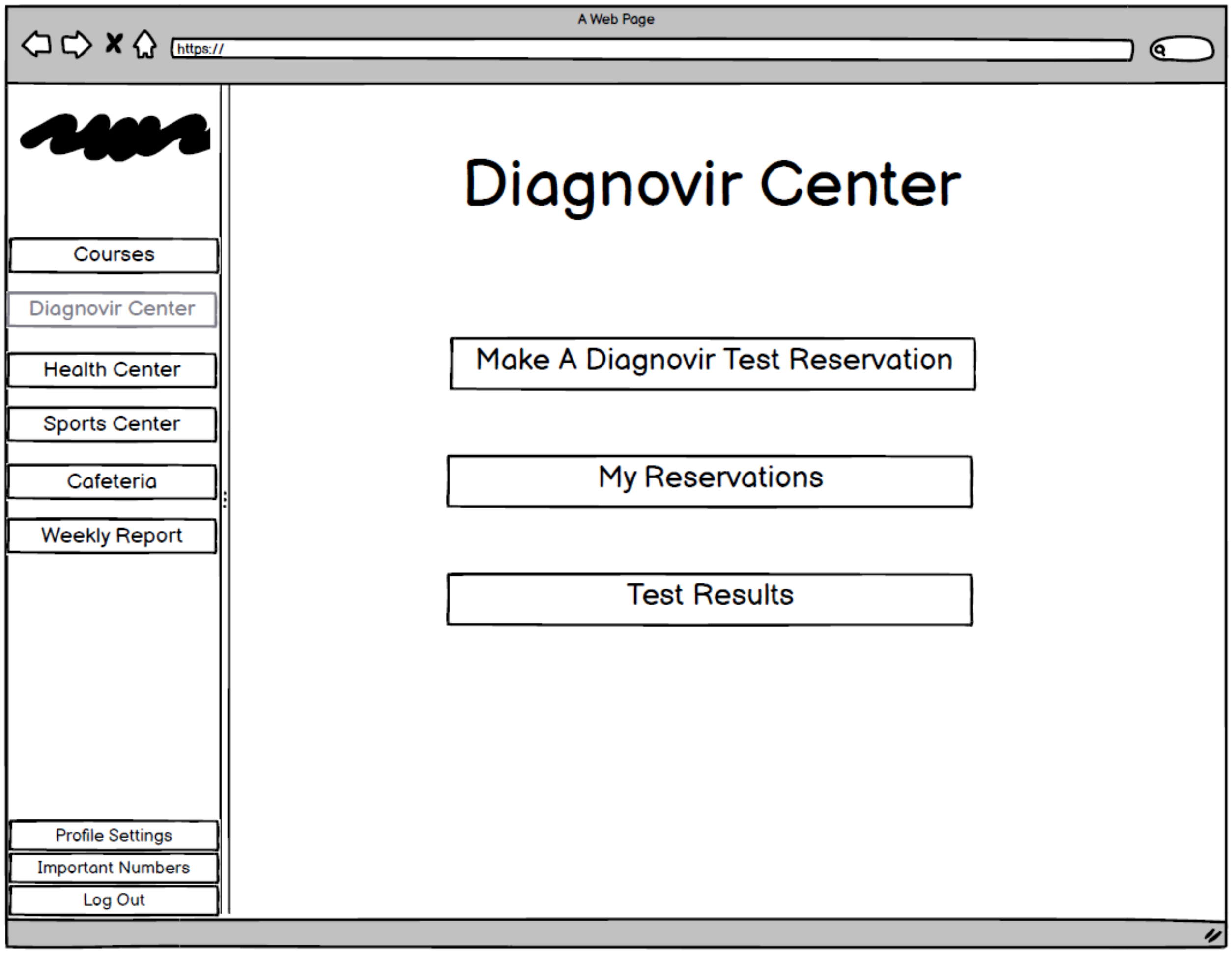


Fig.36: Diagnovir Center Page for Patients

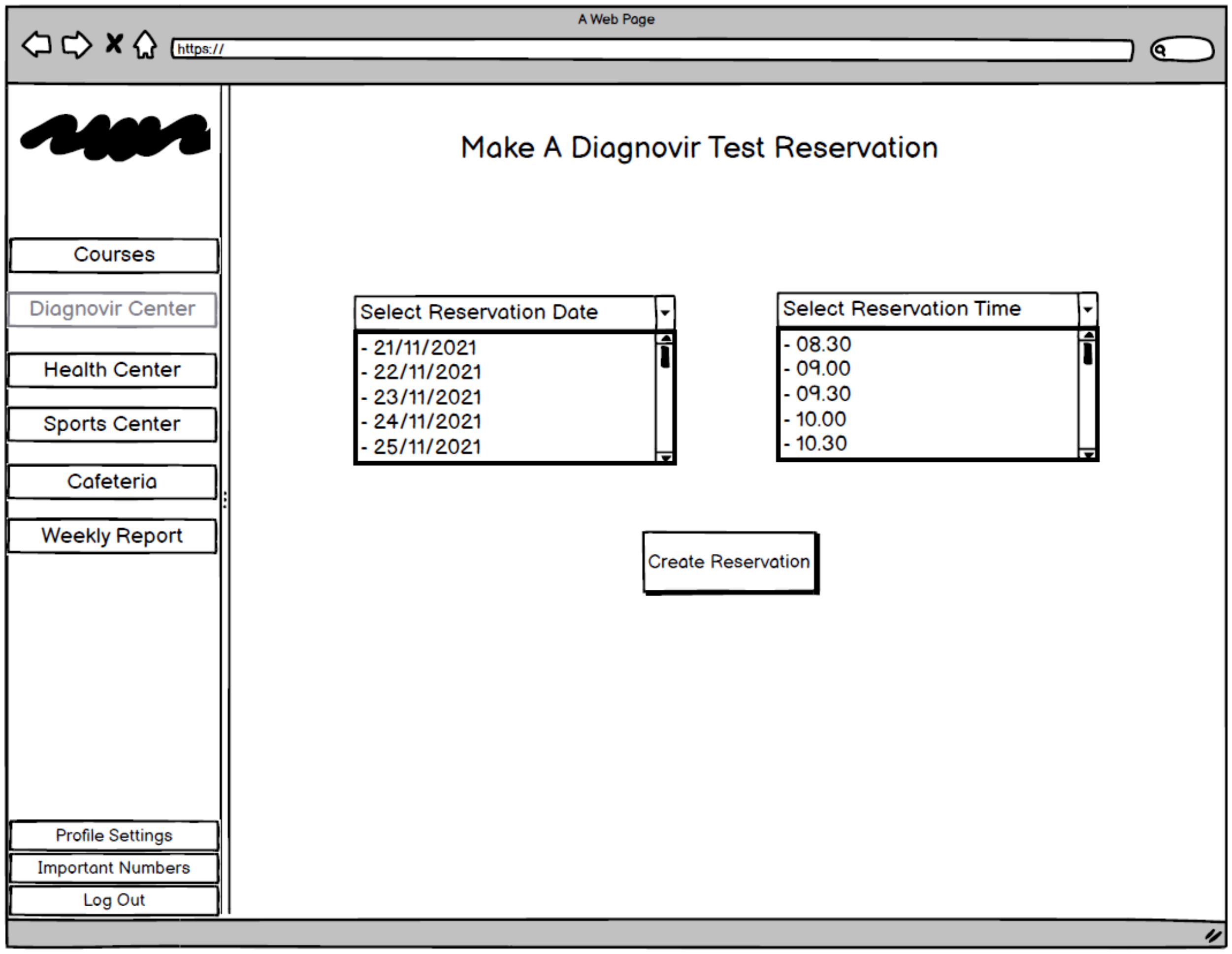
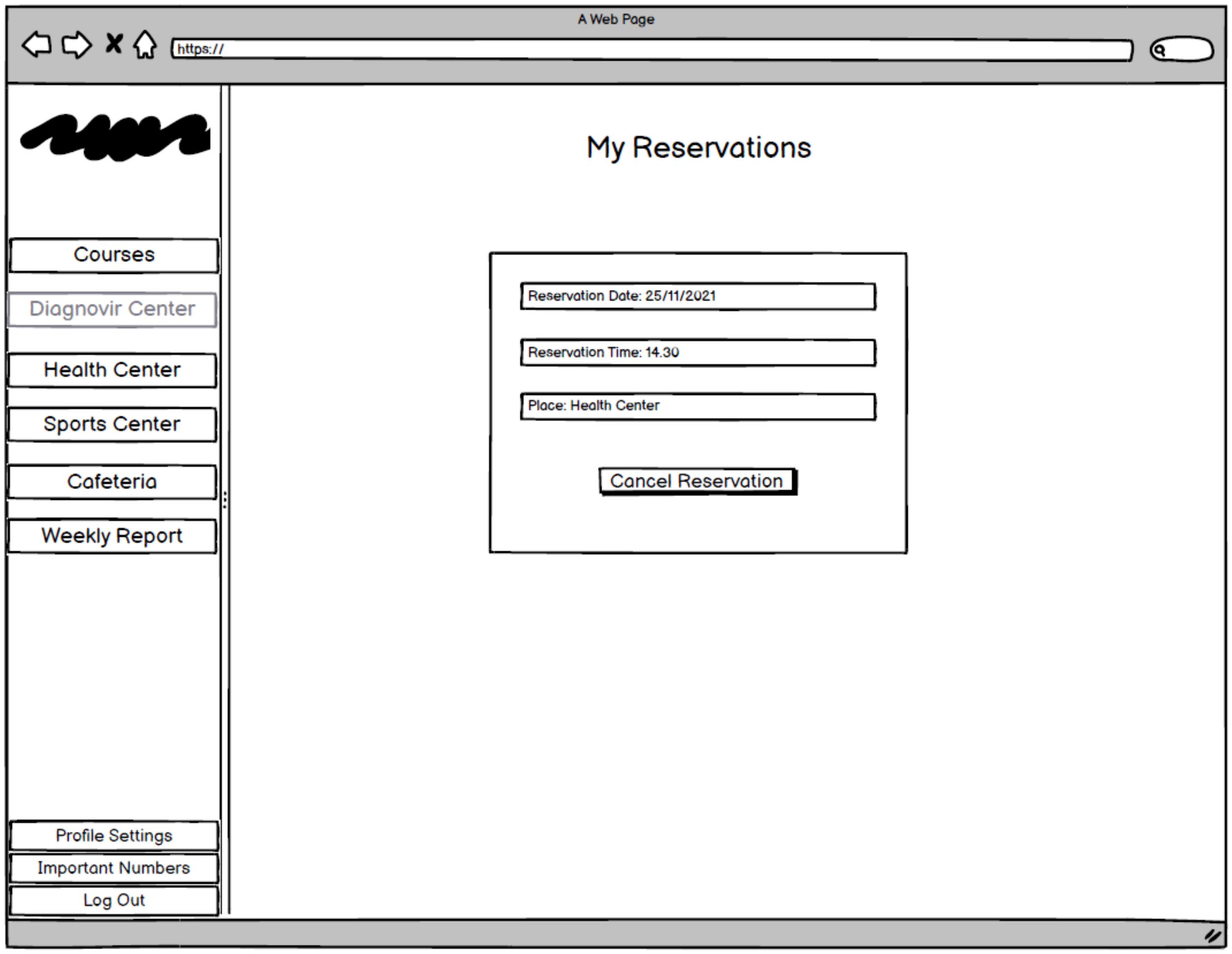


Fig.37: Make Diagnovir Appointment Page



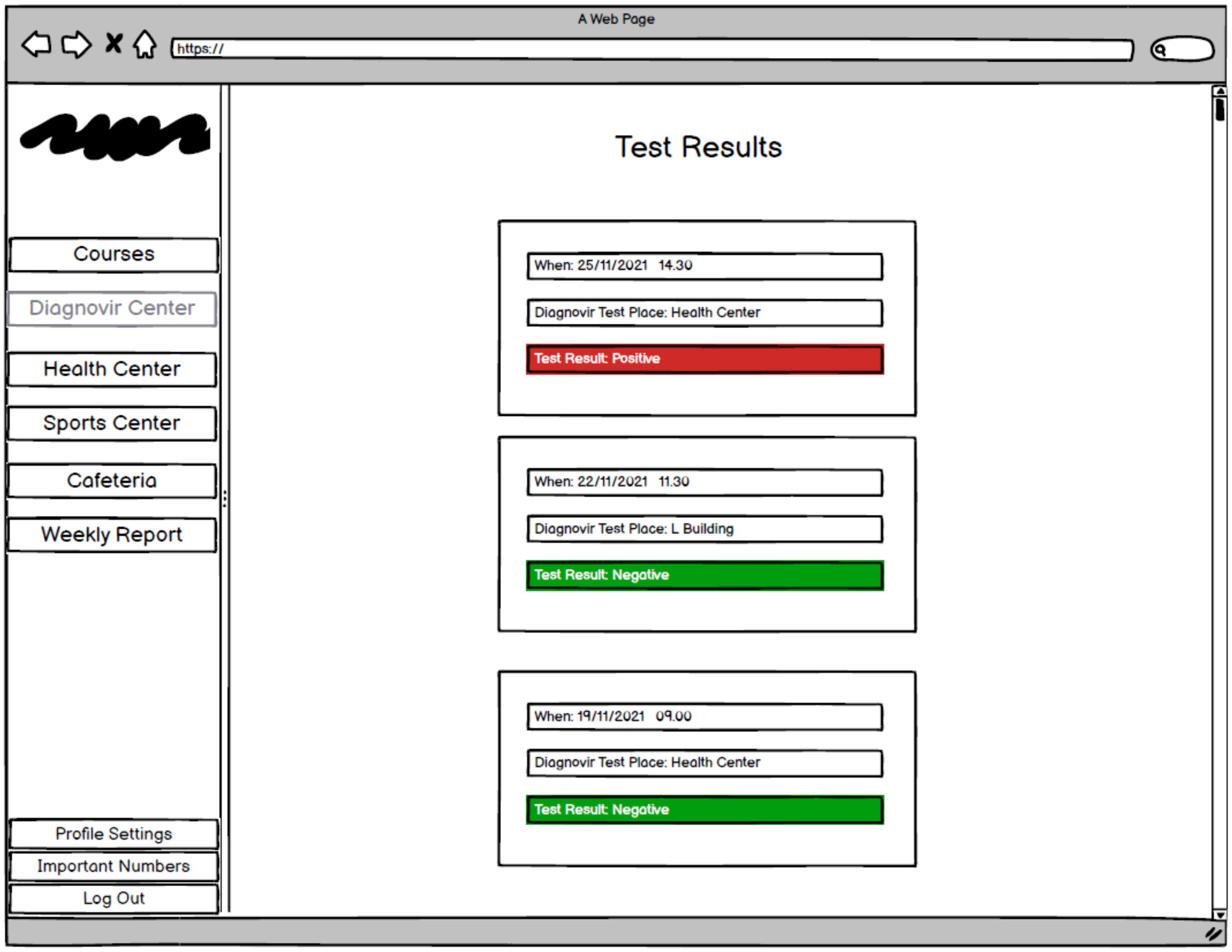
My Reservations

Reservation Date: 25/11/2021

Reservation Time: 14.30

Place: Health Center

Fig.38: My Appointments Page



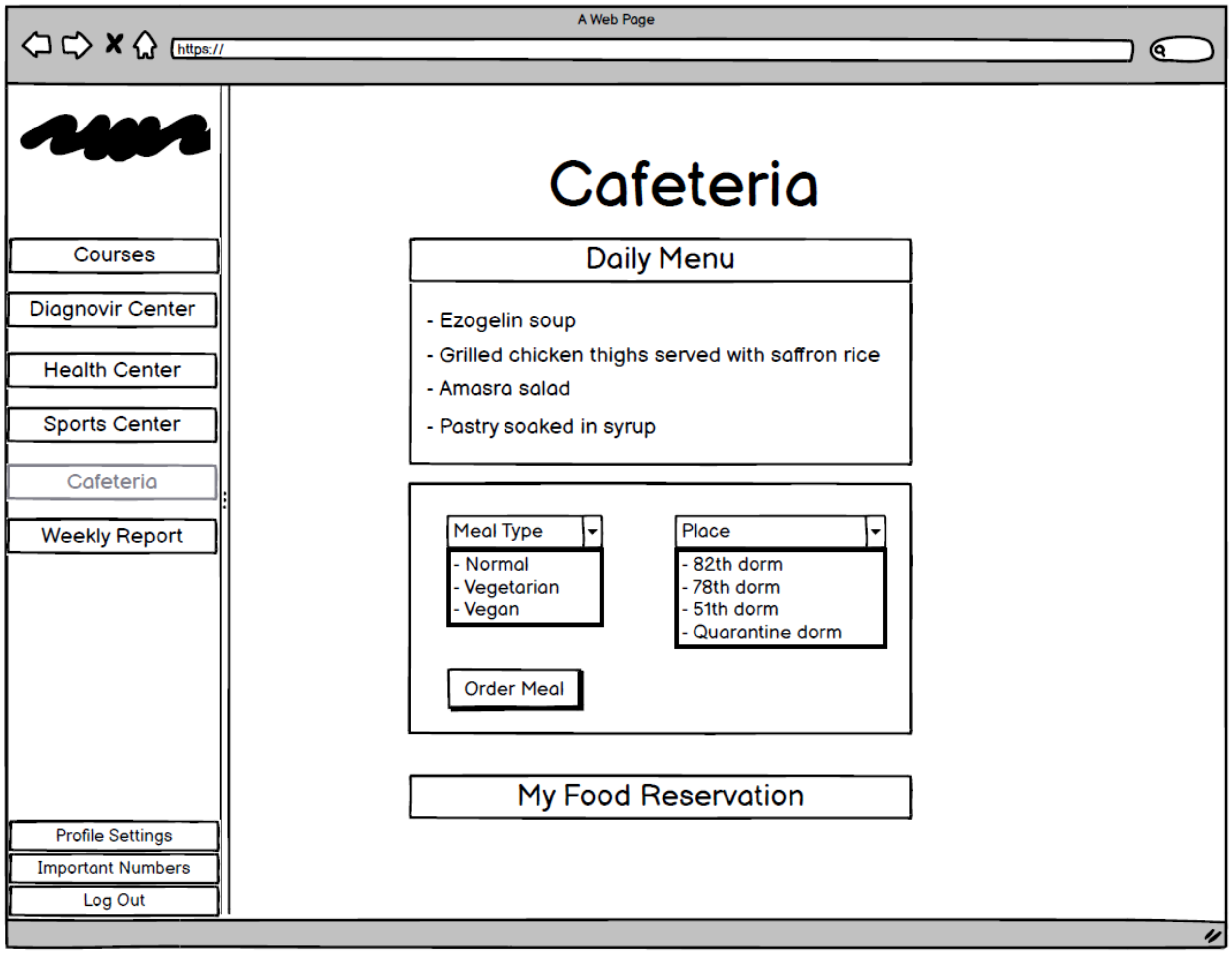
Test Results

When: 25/11/2021 14.30
Diagnovir Test Place: Health Center
Test Result: Positive

When: 22/11/2021 11.30
Diagnovir Test Place: L Building
Test Result: Negative

When: 19/11/2021 09.00
Diagnovir Test Place: Health Center
Test Result: Negative

Fig.39: Diagnovir Test Results Page



Cafeteria

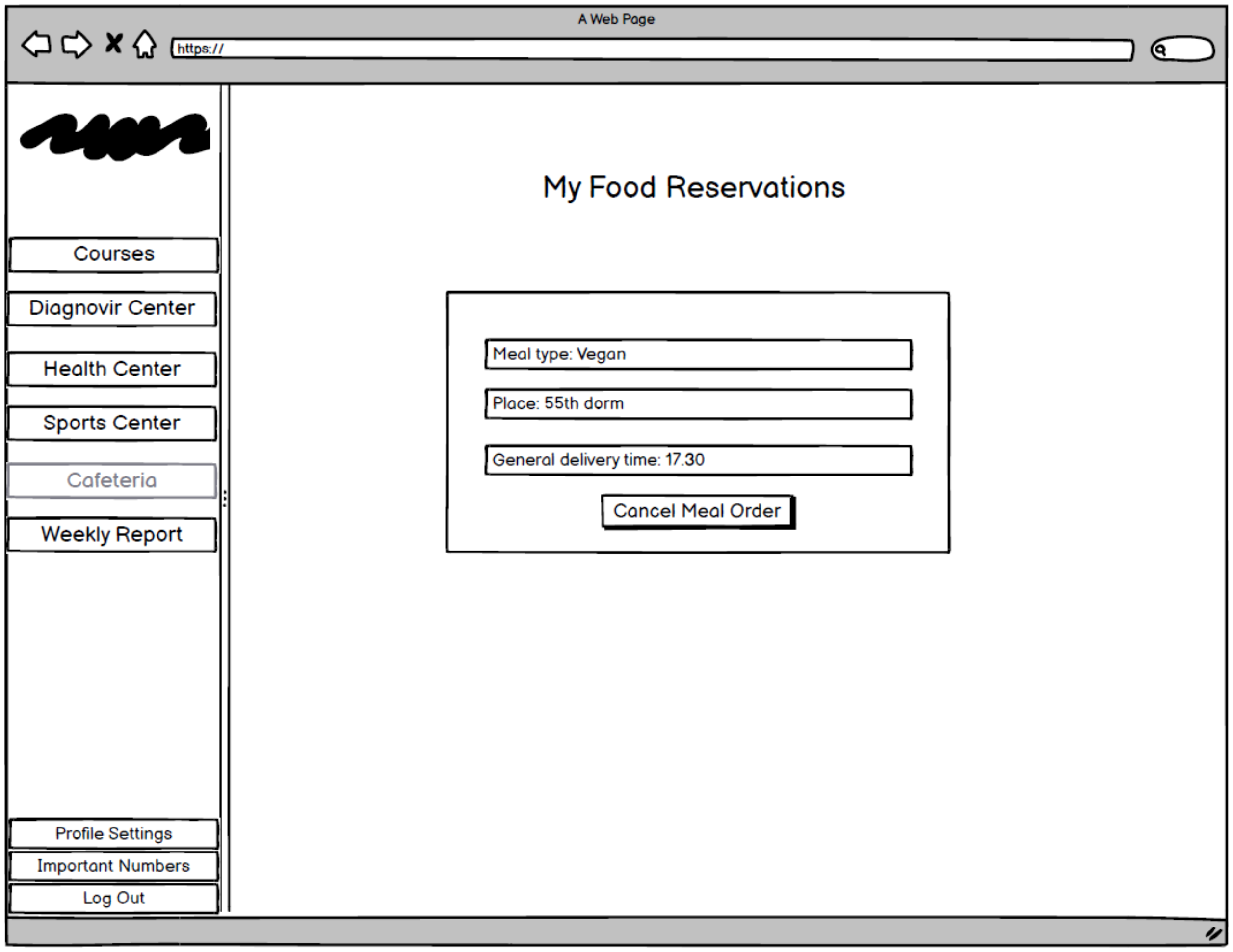
Daily Menu

- Ezogelin soup
- Grilled chicken thighs served with saffron rice
- Amasra salad
- Pastry soaked in syrup

Meal Type
- Normal
- Vegetarian
- Vegan

Place
- 82th dorm
- 78th dorm
- 51th dorm
- Quarantine dorm

Fig.40: Cafeteria Page



My Food Reservations

Meal type: Vegan

Place: 55th dorm

General delivery time: 17.30

Fig.41: Make Food Reservation Page

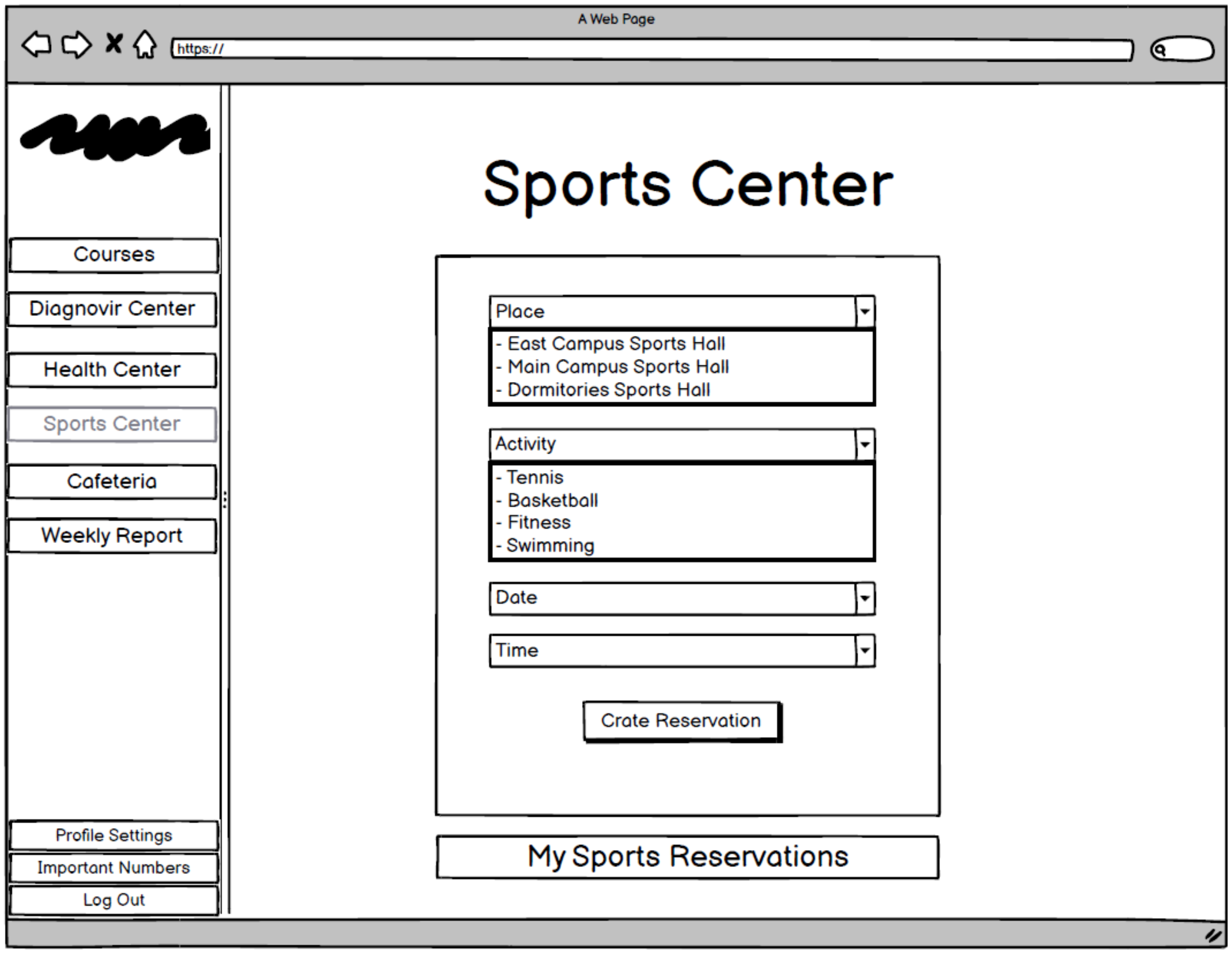
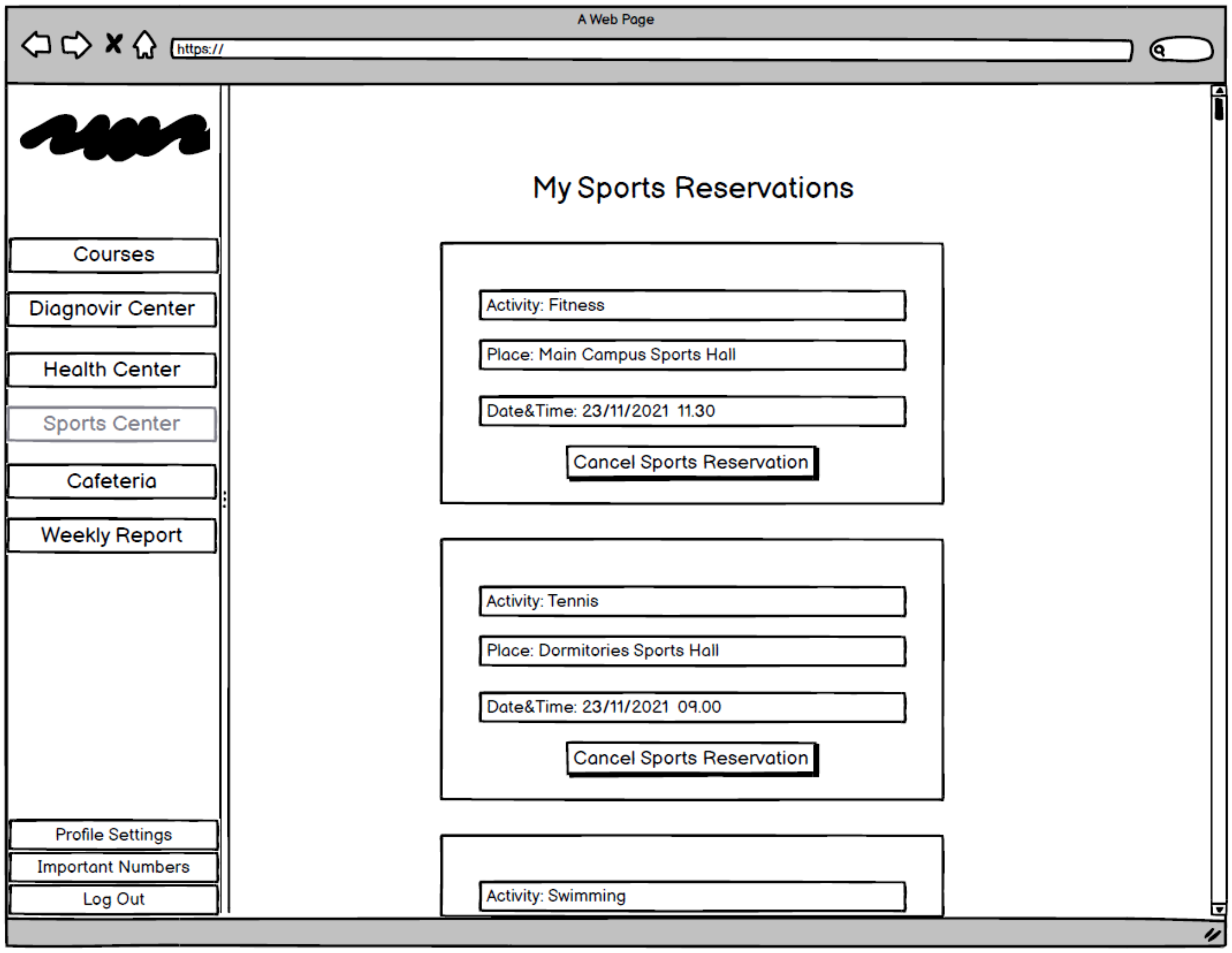


Fig.42: Sports Center Page



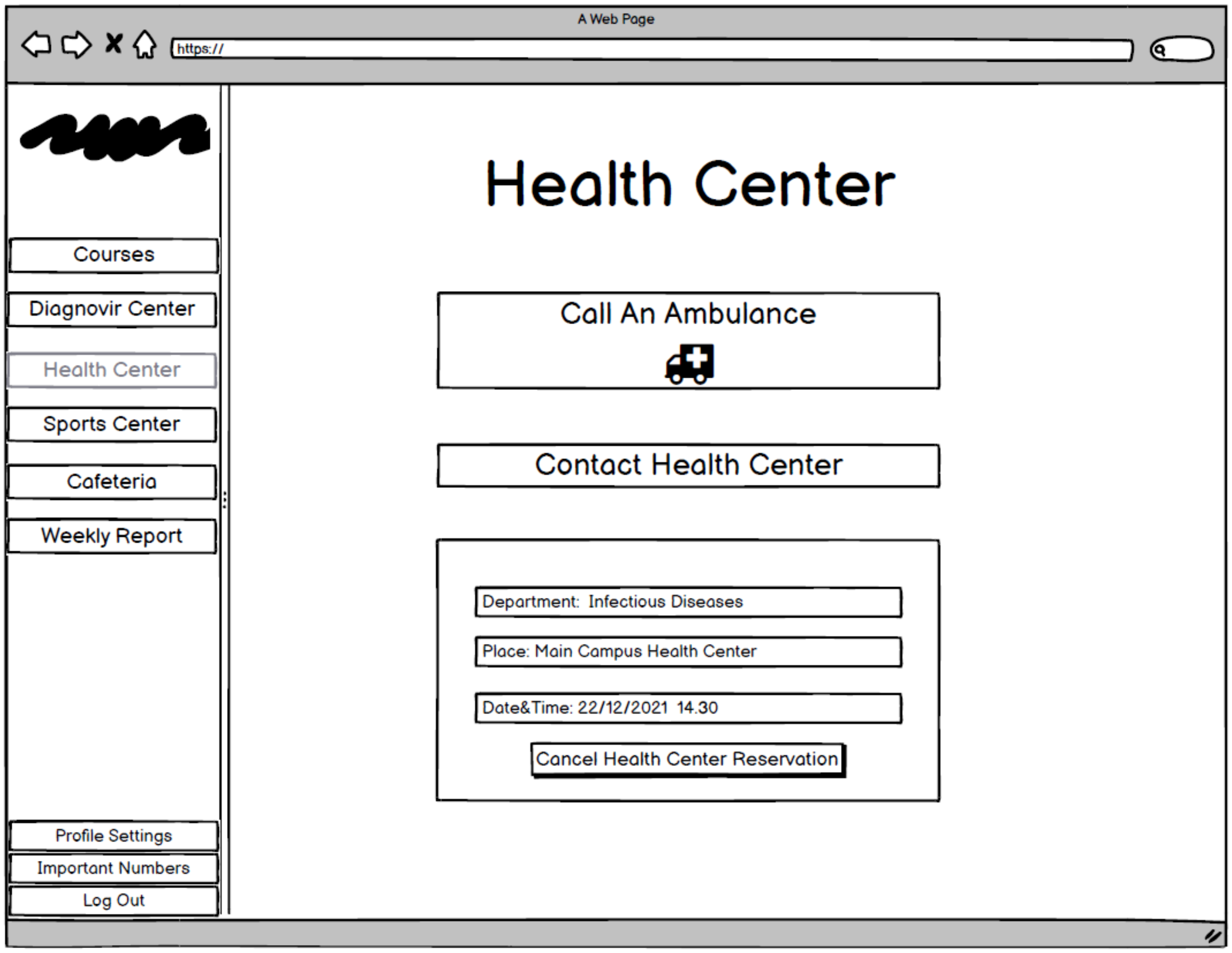
My Sports Reservations

Activity: Fitness
Place: Main Campus Sports Hall
Date&Time: 23/11/2021 11.30

Activity: Tennis
Place: Dormitories Sports Hall
Date&Time: 23/11/2021 09.00

Activity: Swimming

Fig.43: My Sports Reservations Page

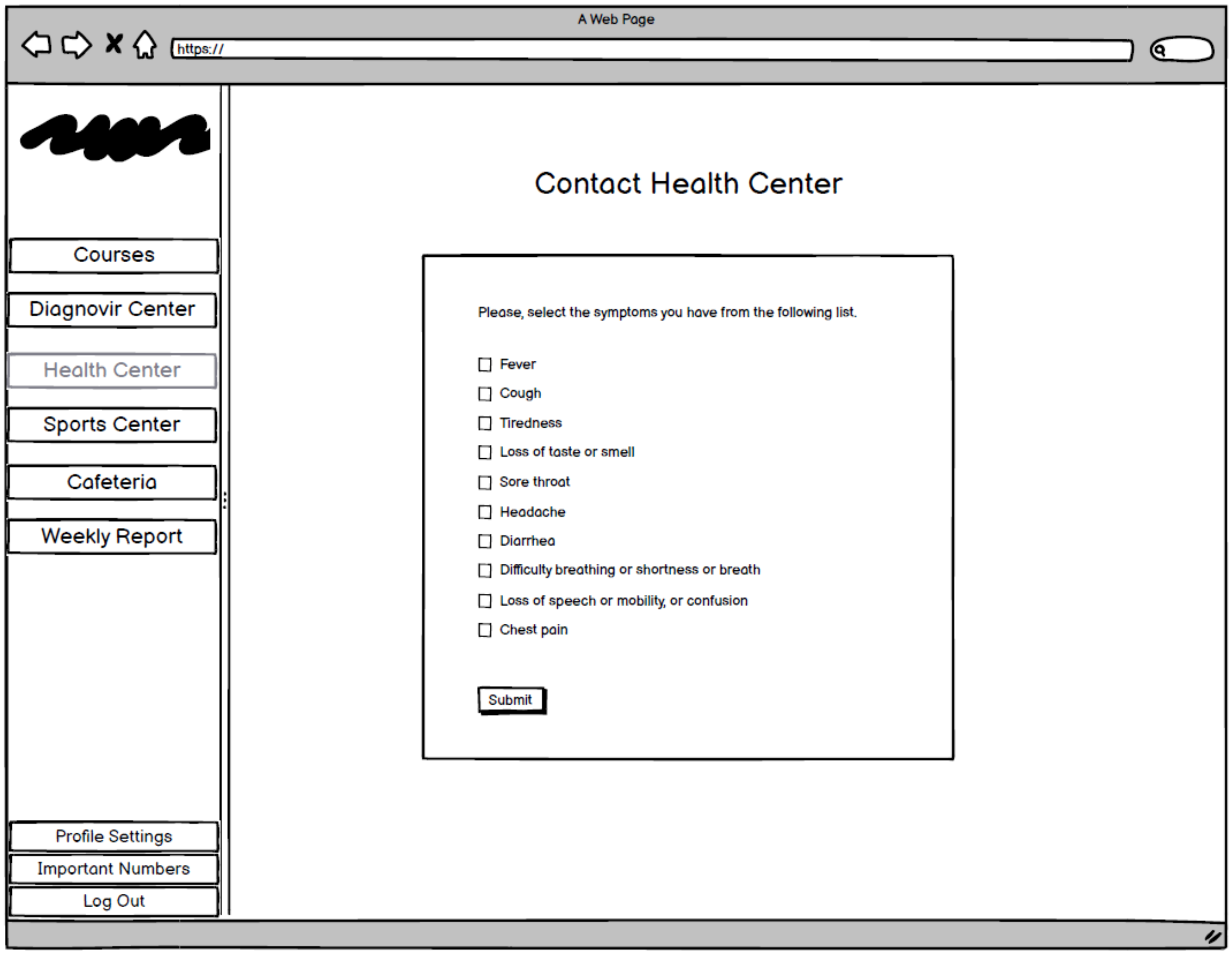


Department: Infectious Diseases

Place: Main Campus Health Center

Date&Time: 22/12/2021 14.30

Fig.44: Health Center Page

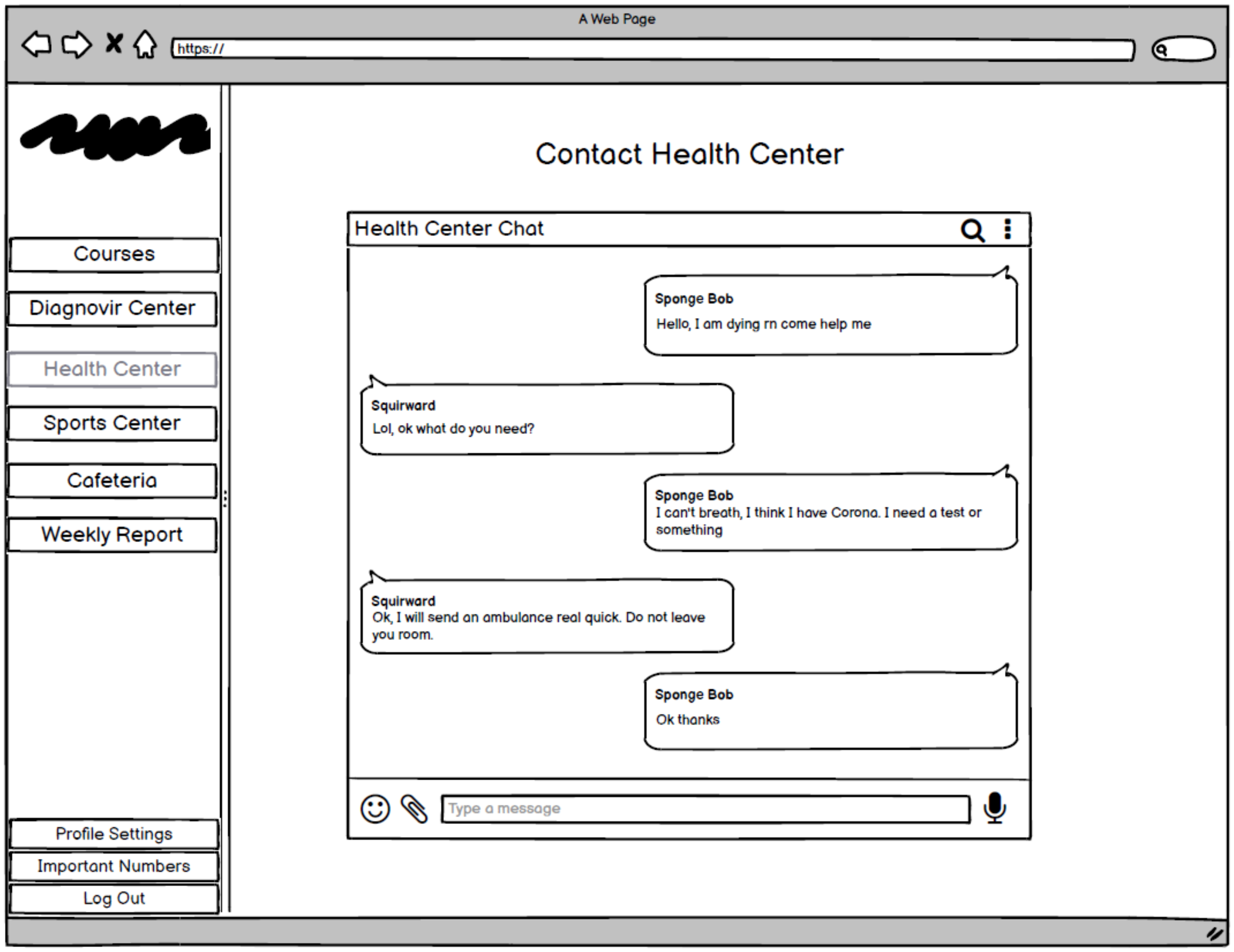


Contact Health Center

Please, select the symptoms you have from the following list.

- ☐ Fever
- ☐ Cough
- ☐ Tiredness
- ☐ Loss of taste or smell
- ☐ Sore throat
- ☐ Headache
- ☐ Diarrhea
- ☐ Difficulty breathing or shortness or breath
- ☐ Loss of speech or mobility, or confusion
- ☐ Chest pain

Fig.45: Health Center Form Page



Sponge Bob
Hello, I am dying rn come help me

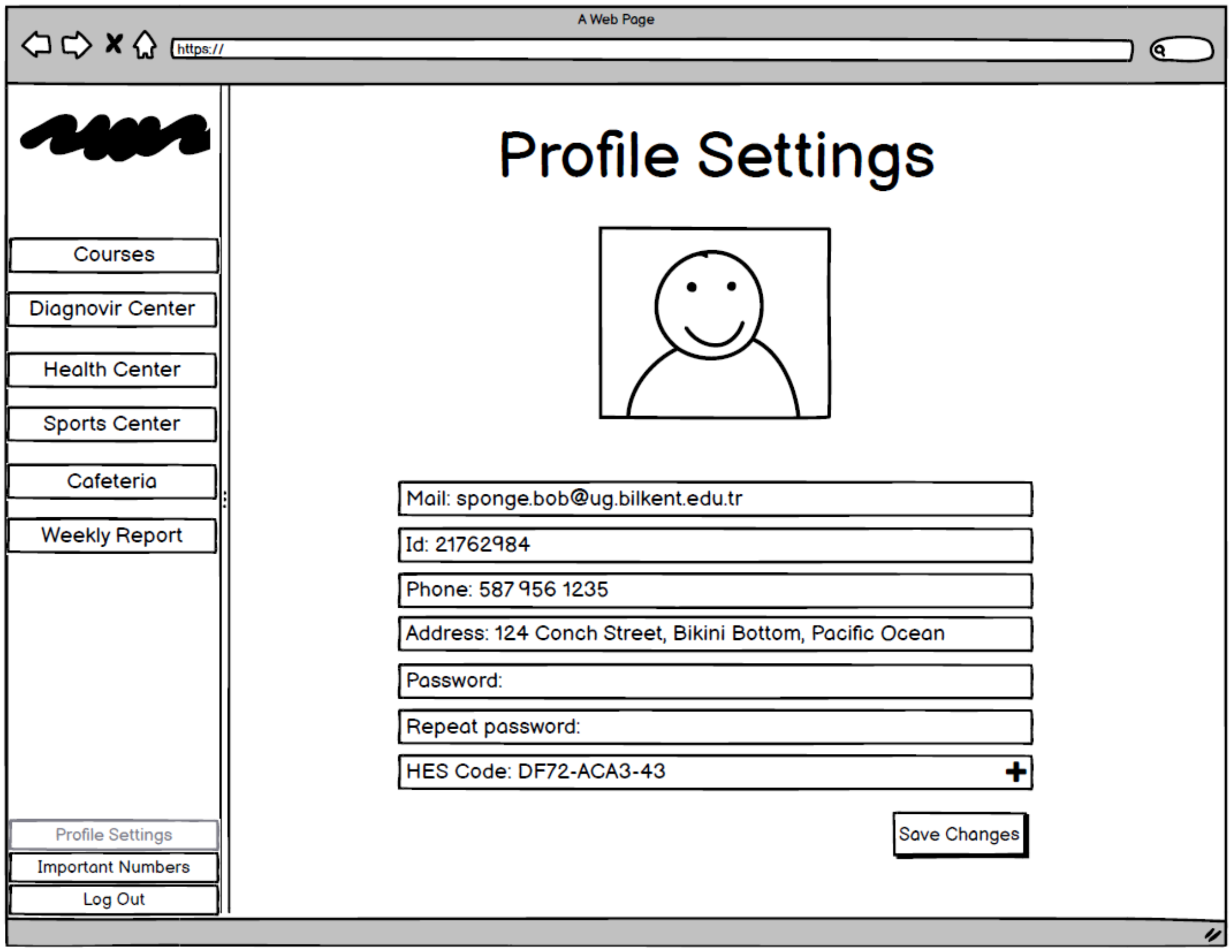
Squirward
Lol, ok what do you need?

Sponge Bob
I can't breath, I think I have Corona. I need a test or something

Squirward
Ok, I will send an ambulance real quick. Do not leave you room.

Sponge Bob
Ok thanks

Fig.46: Contact Health Center Chat Page



Mail: sponge.bob@ug.bilkent.edu.tr

Id: 21762984

Phone: 587 956 1235

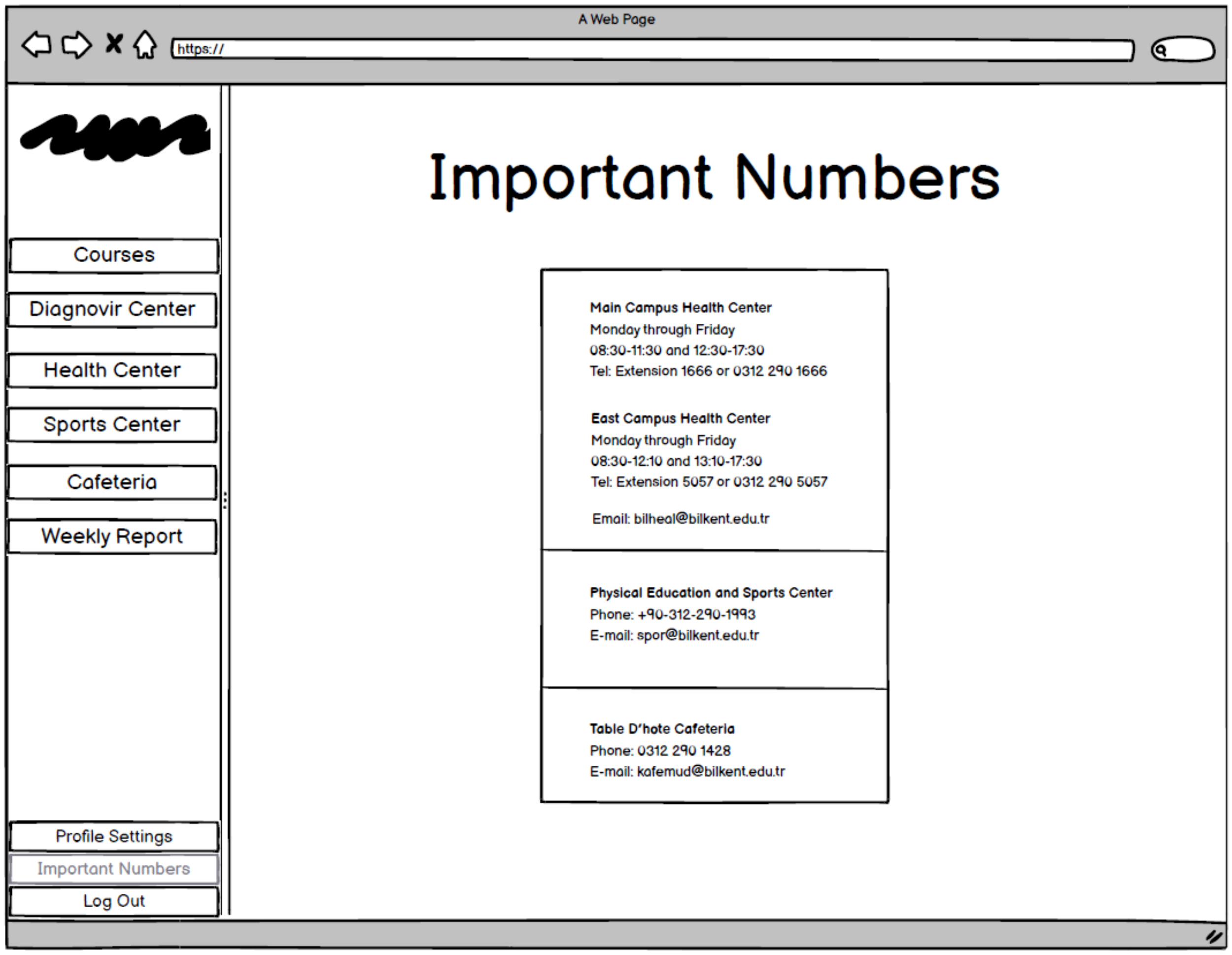
Address: 124 Conch Street, Bikini Bottom, Pacific Ocean

Password:

Repeat password:

HES Code: DF72-ACA3-43

Fig.47: Profile Settings Page



Important Numbers

Main Campus Health Center
Monday through Friday
08:30-11:30 and 12:30-17:30
Tel: Extension 1666 or 0312 290 1666

East Campus Health Center
Monday through Friday
08:30-12:10 and 13:10-17:30
Tel: Extension 5057 or 0312 290 5057

Email: bilheal@bilkent.edu.tr

Physical Education and Sports Center
Phone: +90-312-290-1993
E-mail: spor@bilkent.edu.tr

Table D'hote Cafeteria
Phone: 0312 290 1428
E-mail: kafemud@bilkent.edu.tr

Fig.48: Important Numbers Page

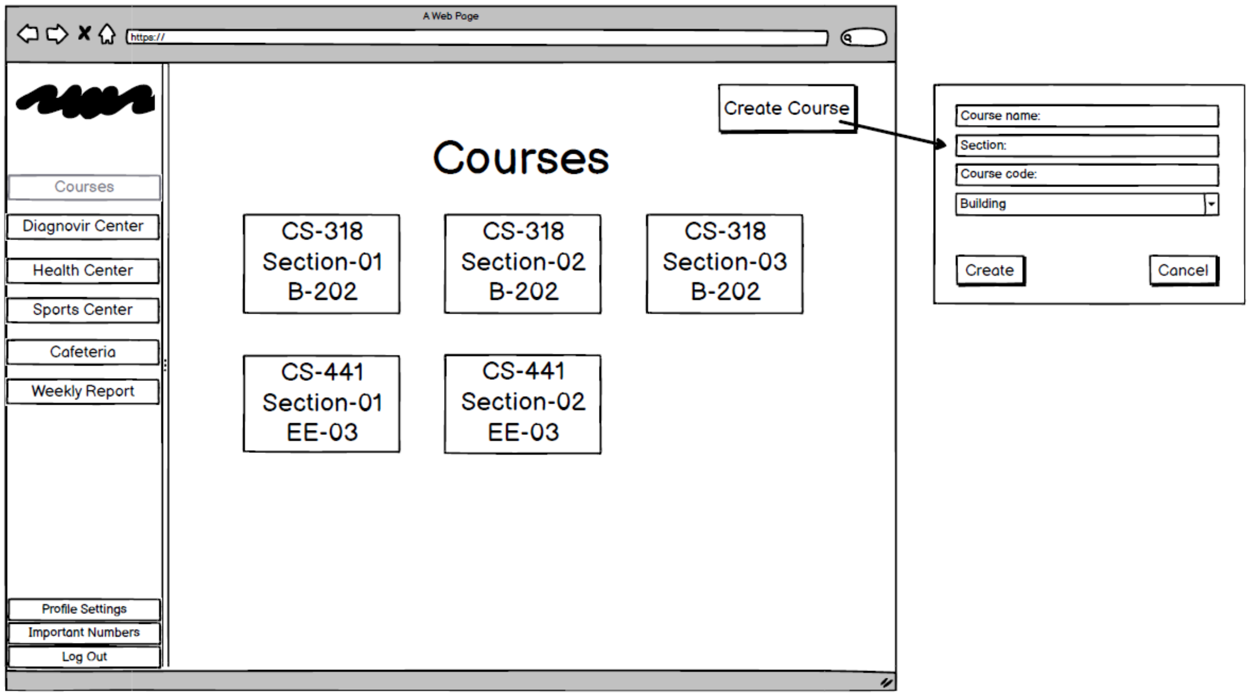


Fig.49: Instructor Courses Page

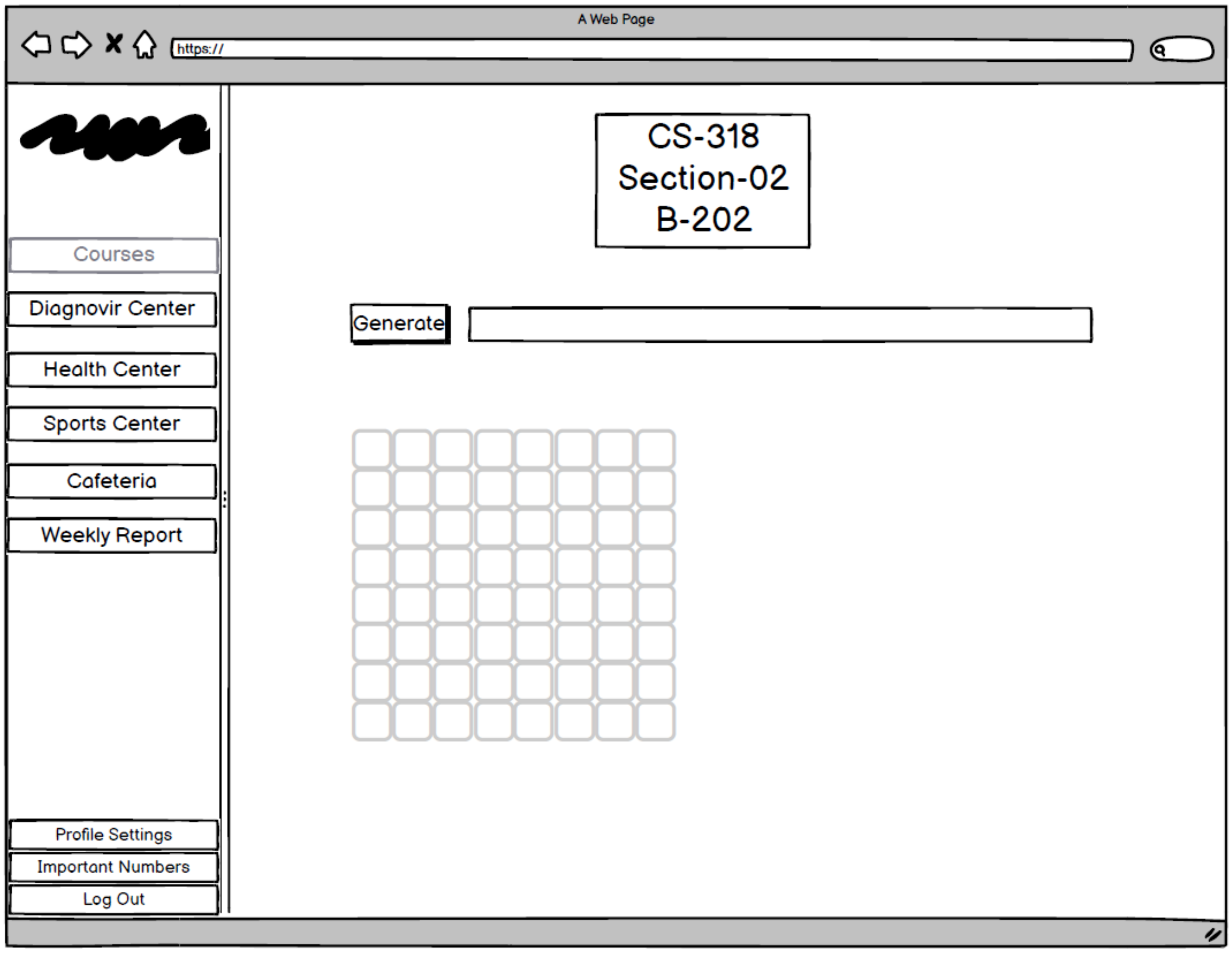


Fig.50: Inactive Instructor Seating Plan

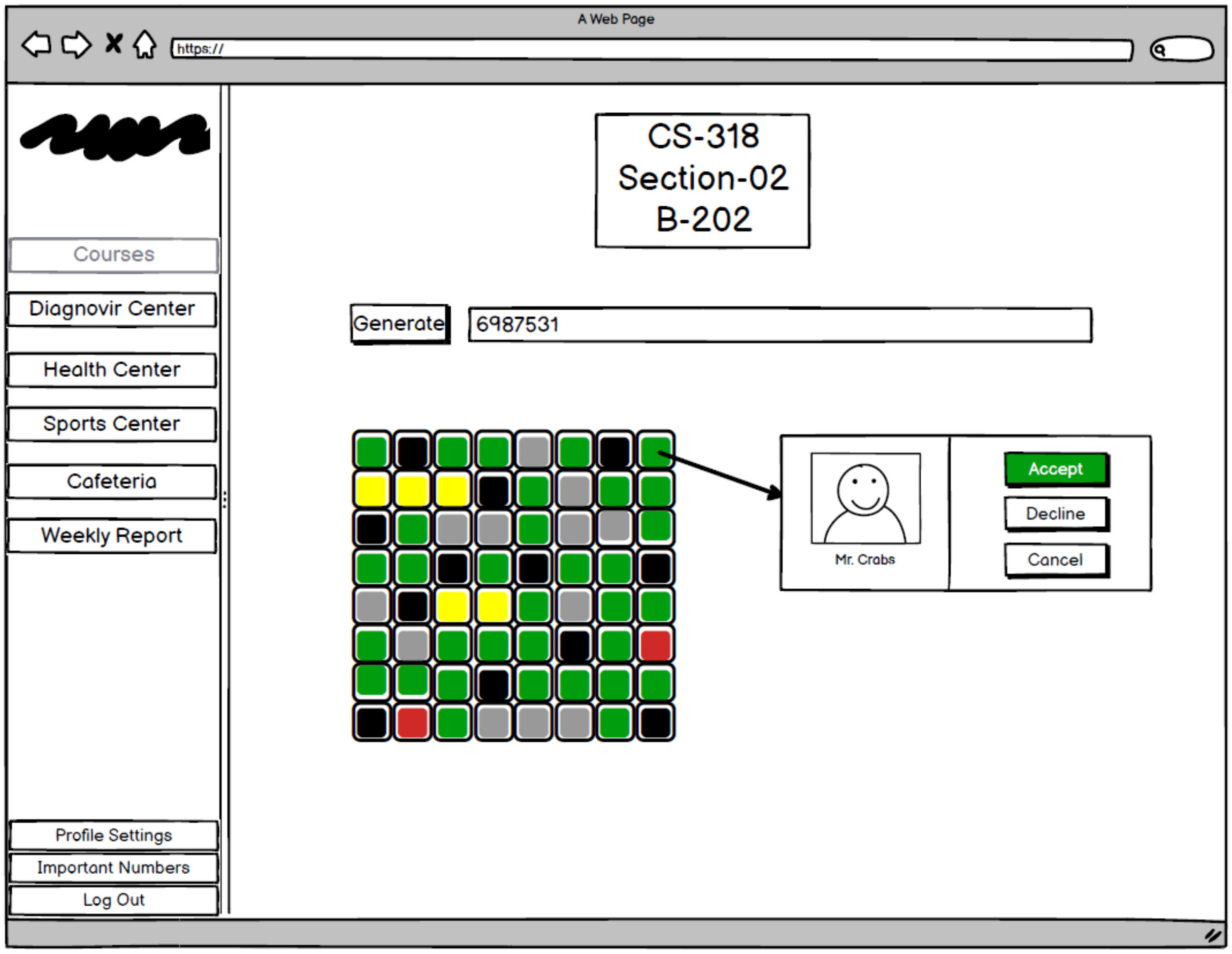


Fig.51: Active Instructor Seating Plan

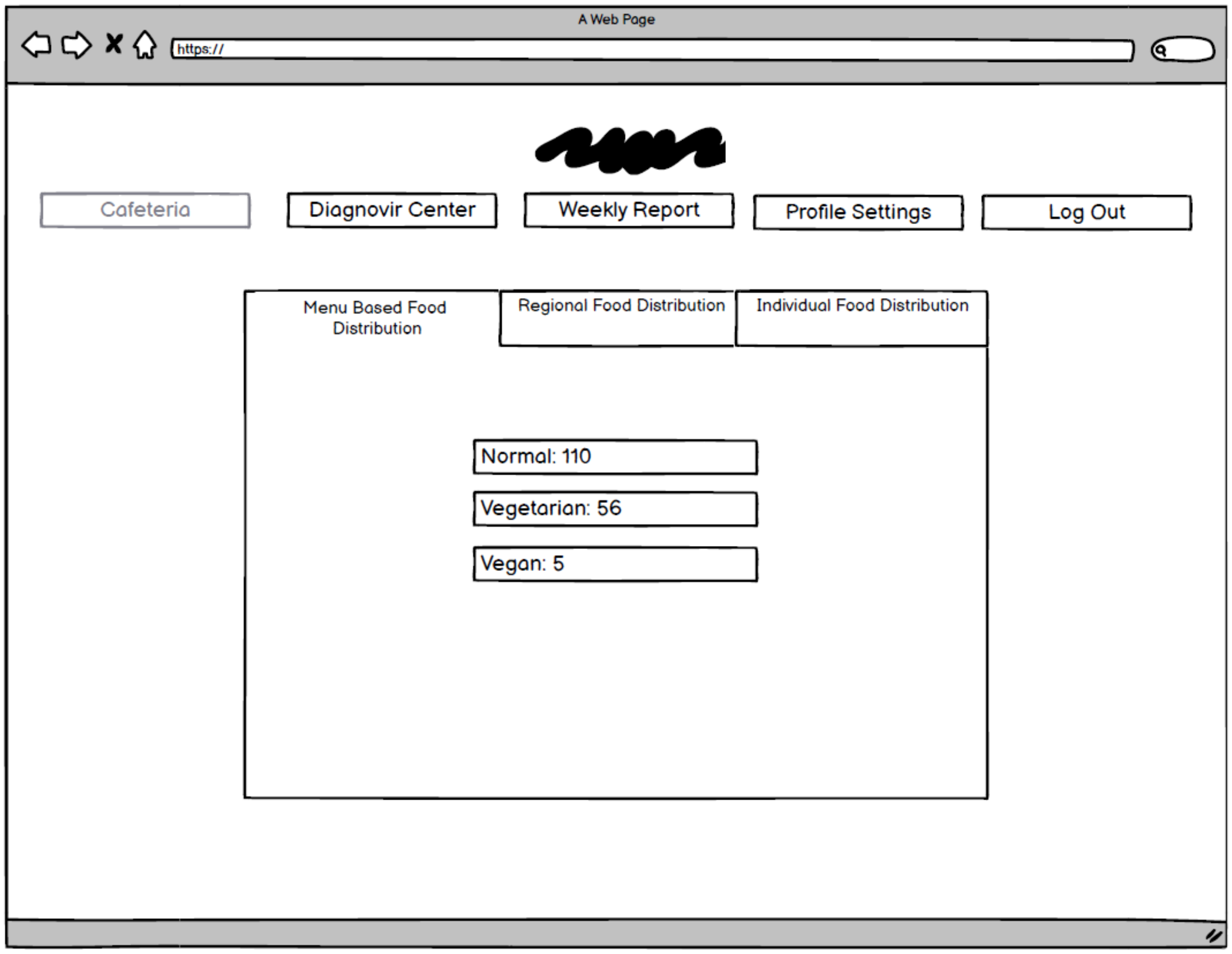
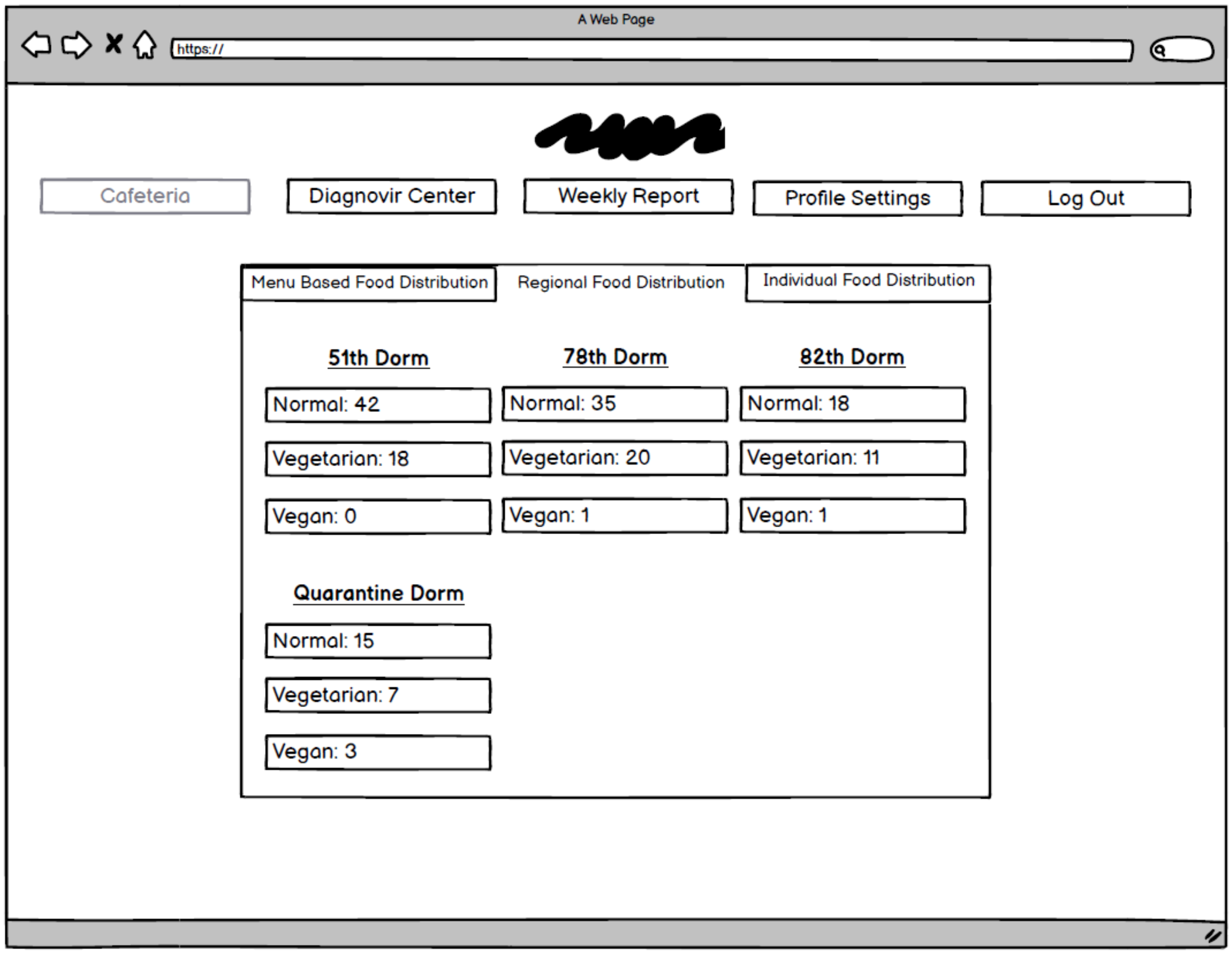


Fig.52: Cafeteria Staff Page (Menu Based Food Distribution)


A Web Page
https://
Cafeteria
Diagnovir Center
Weekly Report
Profile Settings
Log Out
Menu Based Food Distribution
Regional Food Distribution
Individual Food Distribution

51th Dorm

Normal: 42

Vegetarian: 18

Vegan: 0

78th Dorm

Normal: 35

Vegetarian: 20

Vegan: 1

82th Dorm

Normal: 18

Vegetarian: 11

Vegan: 1

Quarantine Dorm

Normal: 15

Vegetarian: 7

Vegan: 3

Fig.53: Cafeteria Staff Page (Regional Food Distribution)

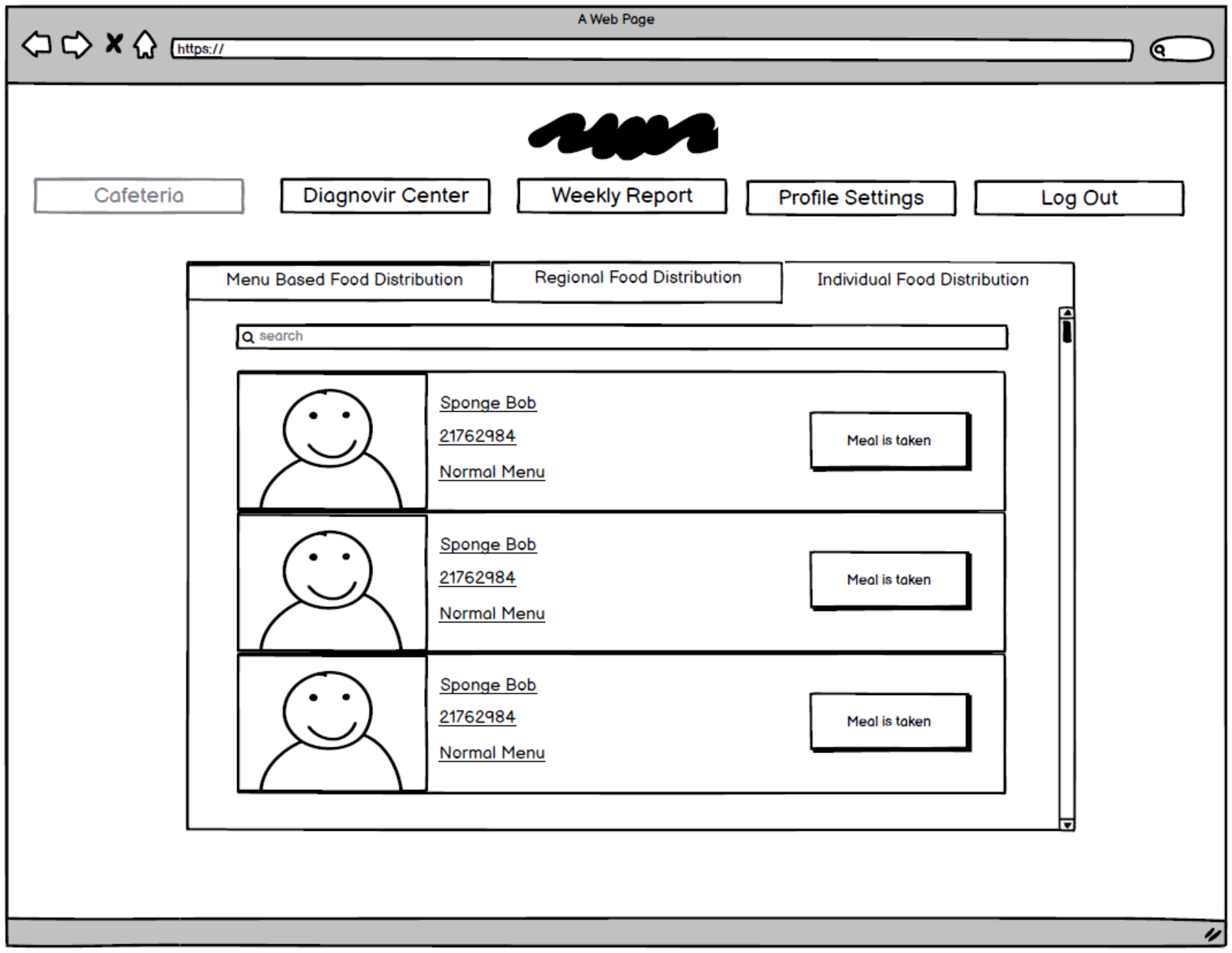


Fig.54: Cafeteria Staff Page (Individual Food Distribution)

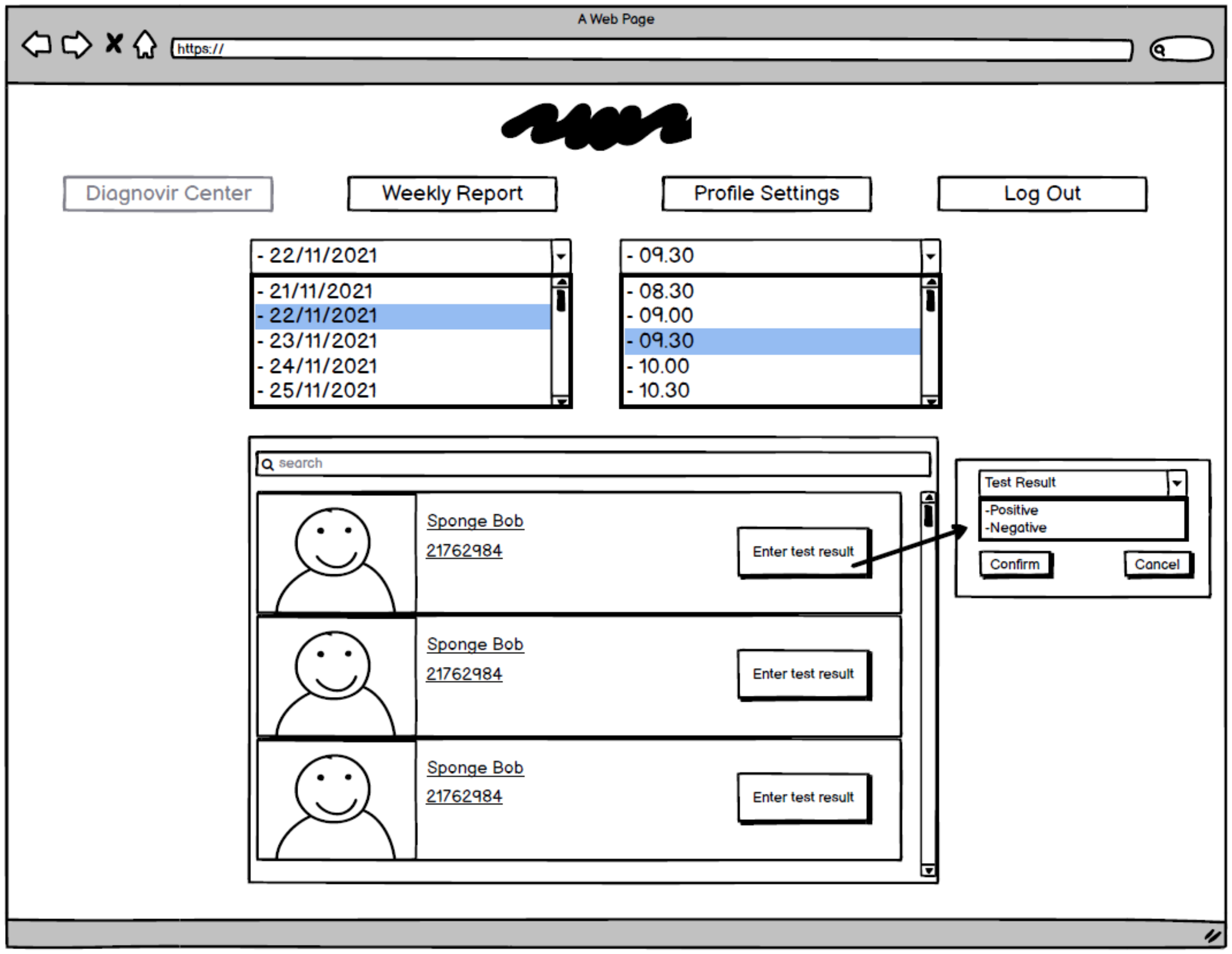


Fig.55: Diagnovir Staff Page

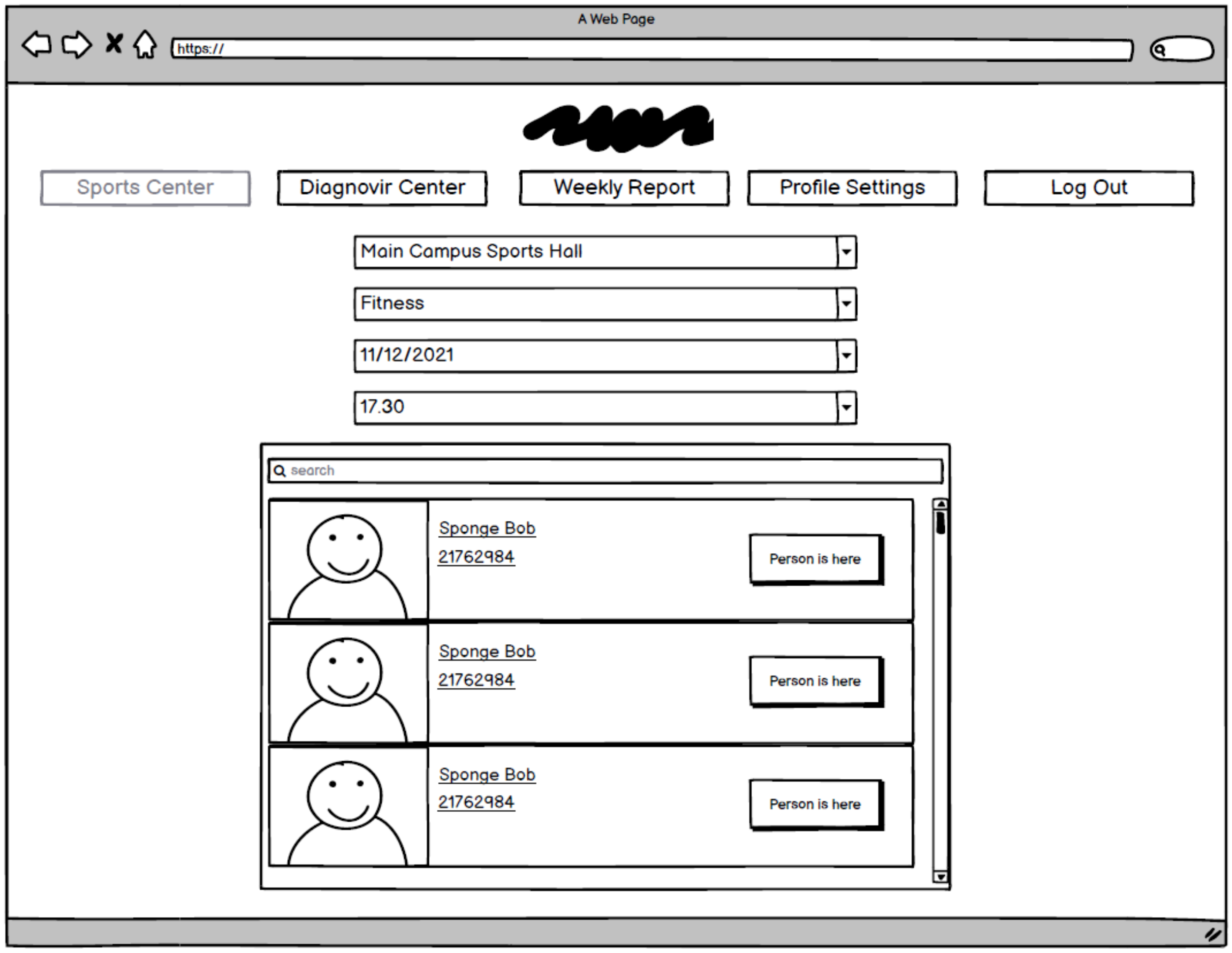


Fig.56:Sports Center Staff Page

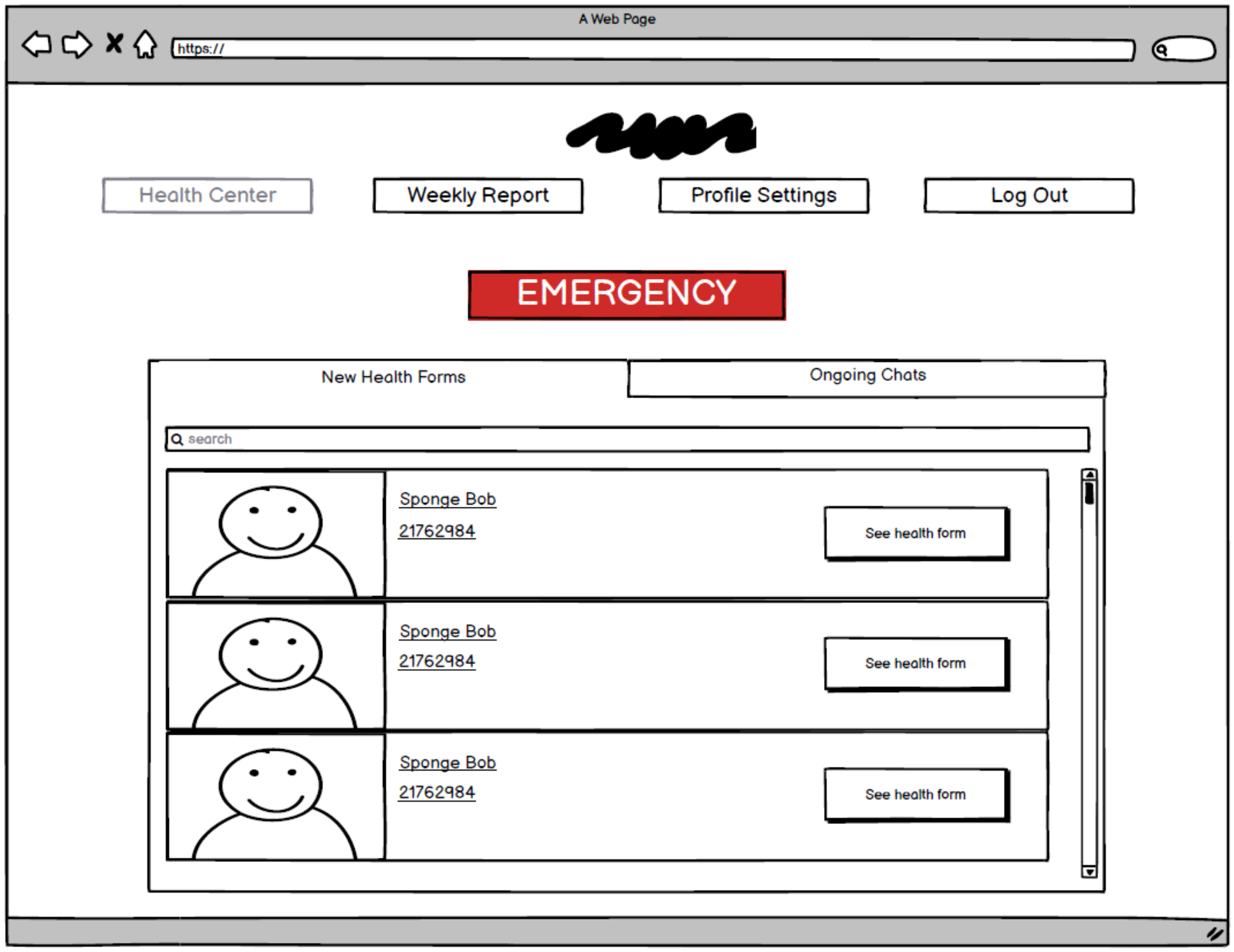


Fig.57: Health Center Staff Page (New Health Forms)

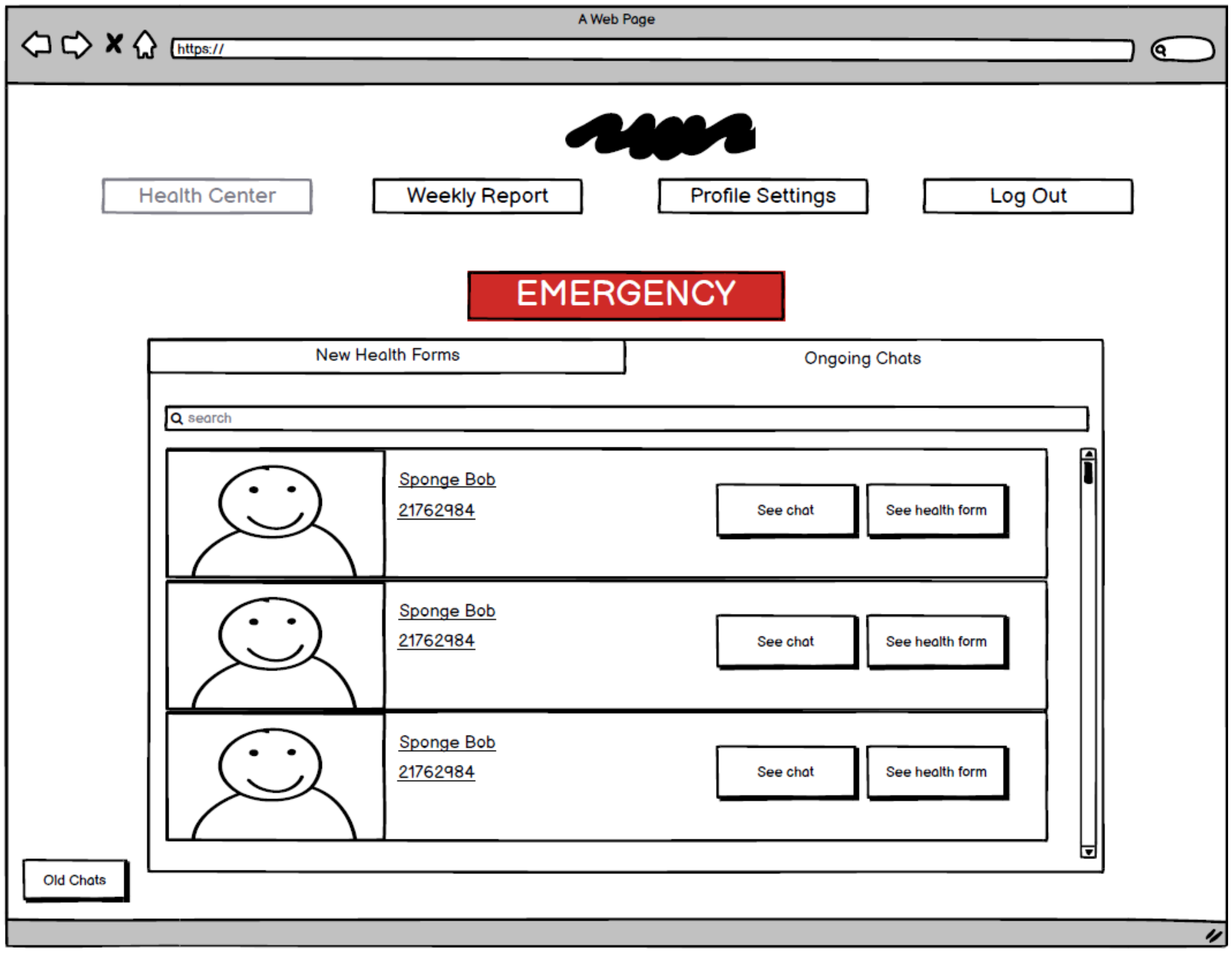
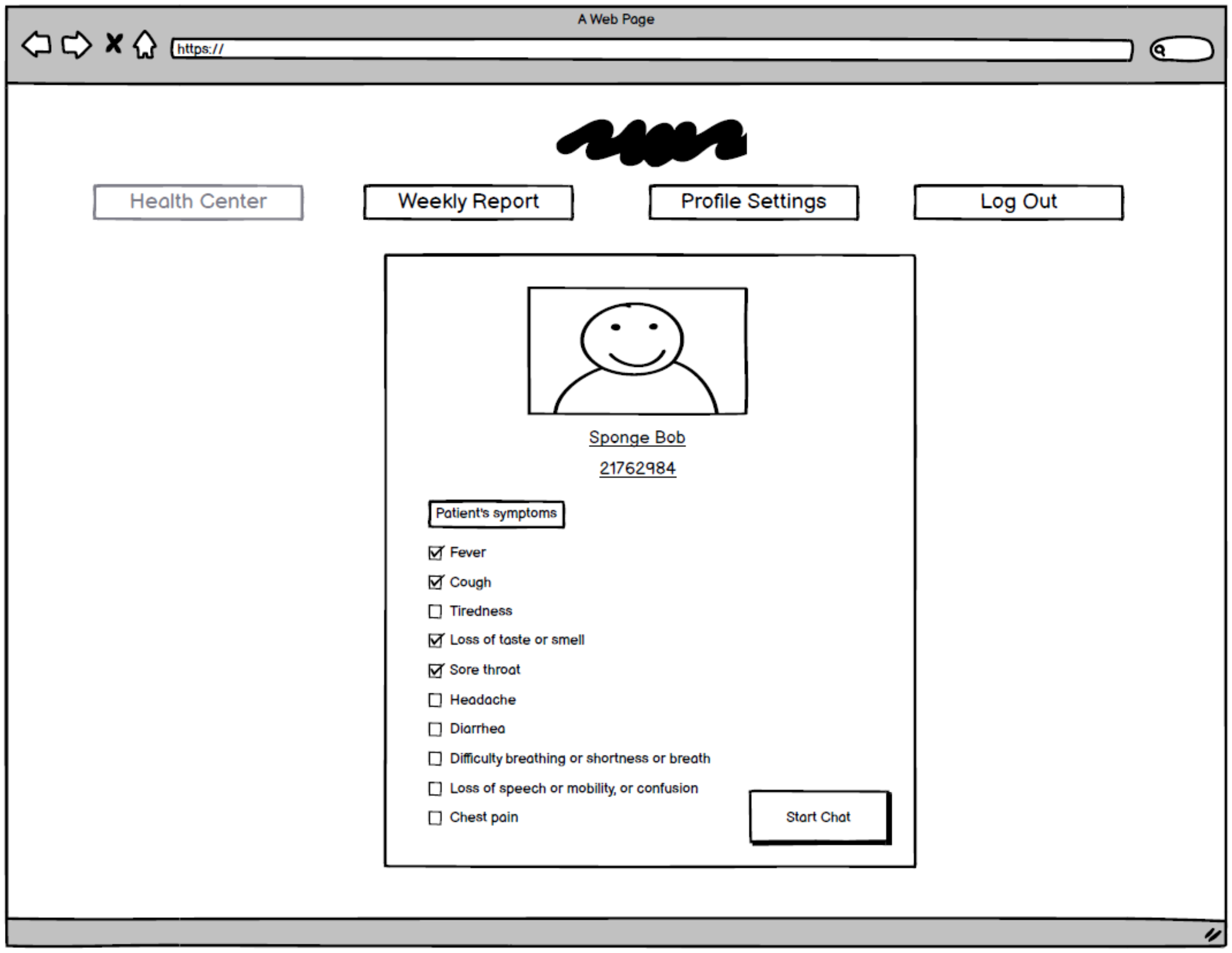


Fig.58: Health Center Staff Page (Ongoing Chats)



Sponge Bob

21762984

Patient's symptoms

- ✓ Fever
- ✓ Cough
- ☐ Tiredness
- ✓ Loss of taste or smell
- ✓ Sore throat
- ☐ Headache
- ☐ Diarrhea
- ☐ Difficulty breathing or shortness or breath
- ☐ Loss of speech or mobility, or confusion
- ☐ Chest pain

Fig.59: Health Center Staff Patient Form Page

4. Improvement Summary

4.1 Use Case Diagram

- For the use case diagram, we added some of the forgotten use case descriptions.
- Changed some of the arrows.

4.2 State Diagram

- Draw many of the state diagrams from scratch.
- Removed decision nodes from the state diagrams.
- Added newly learned diagnovir/campus-access conditions to the campus allowance state diagram.

4.3 Activity Diagram

- Some missing decision nodes are added.
- Removed some of the unnecessary activities.

4.4 Class Diagram

- Removed some of the unnecessary classes.
- Added new wanted classes like VaccineCard, Attendance, etc.
- Added new types like Enumeration.
- Added association types.
- Changed some of the multiplicities and relation types.
- Provided all class' descriptions
- Removed some confusing lines into separate ones

4.5 Sequence Diagram

- Removed some of the diagrams with wrong notation.
- Changed the diagrams so that the methods that are used in the sequence diagram matches the ones in the class diagram.
- Removed some of the really similar sequence diagrams.

4.6 UI Mockups

- Did the whole ui's from scratch using balsamiq.
- Added many of the ui mockups that were missing in the first iteration.