CS 319 - 3
GROUP 3E: COVOID-21

FINAL REPORT
PANDEMIC MANAGER

Kutay Demiray - 21901815

Gökberk Keskinkılıç - 21801666

Berke Uçar - 21902238

Yağız Yaşar - 21902951

Ömer Burak Yıldıran - 21901425

Table of Contents

1. Introduction

As a final result, it cannot be claimed that there is a working application. Although backend and frontend are functional by themselves, they could not be connected to each other so that they create a meaningful website. Frontend itself stayed as a static React code and ended up not reflecting the properties of logged-in users since user authentication could not have been implemented.

2. Project Experience

While analyzing, designing, and implementing this project we have learned different aspects of the software design and engineering processes. We want to inspect and analyze our project by separating these cascades into three as mentioned above: Analyze, design, and implementation.

a. Analysis Step

In the analysis step, we were required to draw five UML diagrams: Use-case, class, sequence, activity, and state diagrams. Hence, we have learned these five diagram styles in order to include them within the report. Furthermore, we have learned to detect general and specific features; define pseudo, non-functional and functional requirements; and revise all the previous contents. While drawing the use-case diagram, we have learned to determine both internal and external actors, use cases, and system boundaries. To draw class diagrams, we have learned the patterns like MVC, entity-boundary-control, etc., and applied these patterns to our diagram from the analysis stage. For the activity diagram, we have learned how to analyze the flow of the use cases and address them in a good manner. For the state diagram, we have identified the objects and their states which can be shown and drawn.

b. Design Step

In this step, we have learned details about lower levels of design in a project. We have first obtained knowledge about subsystem decomposition. After learning the strategies for subsystem decomposition, we have gone over our initial five UML diagrams to make them proper to use in subsystem decomposition. We have especially worked on the class diagram revise since this step also contains the final version of the classes and class diagrams. After changing the class diagrams with respect to feedback, we have drawn our subsystem decomposition diagram. We have then designed each package and its contents. Following that, we have investigated technologies about our goals and reported them. Furthermore, we have designed all the classes and their components and provided deployment diagrams in order to explain the communication system of our project. Moreover, we have learned about both hardware and software mapping of our goals. We have examined the sources and learned about our boundary conditions as well.

c. Implementation Step

In the implementation step, we have first learned about the Spring Boot framework and our decided IDE in order to start writing about the backend. Similarly, the people who were responsible with frontend have learned React.js, JavaScript, HTML, and CSS. After that, we have tried to learn about additional plug-ins, AWS servers, and some other technologies. Overall, we have managed to deploy our server to the AWS cloud and access the database of the remote server. We have learned enough to implement some of the backend logic and frontend appearance of our project.

However, overall there happens to be missing parts that we could not implement and insufficient functionalities we have wanted to implement.

Overall, we have learned a lot of things about software engineering with the background tasks that are actually the real heroes for a project. Moreover, we have seen that software engineering is not just coding but making efficient decisions and planning. On the other hand, we could not finish our project because of our time limitation and the negative impacts of bugs. We have learned our lesson to be more precise about coding and not hang on to just one problem.

3. User's Guide

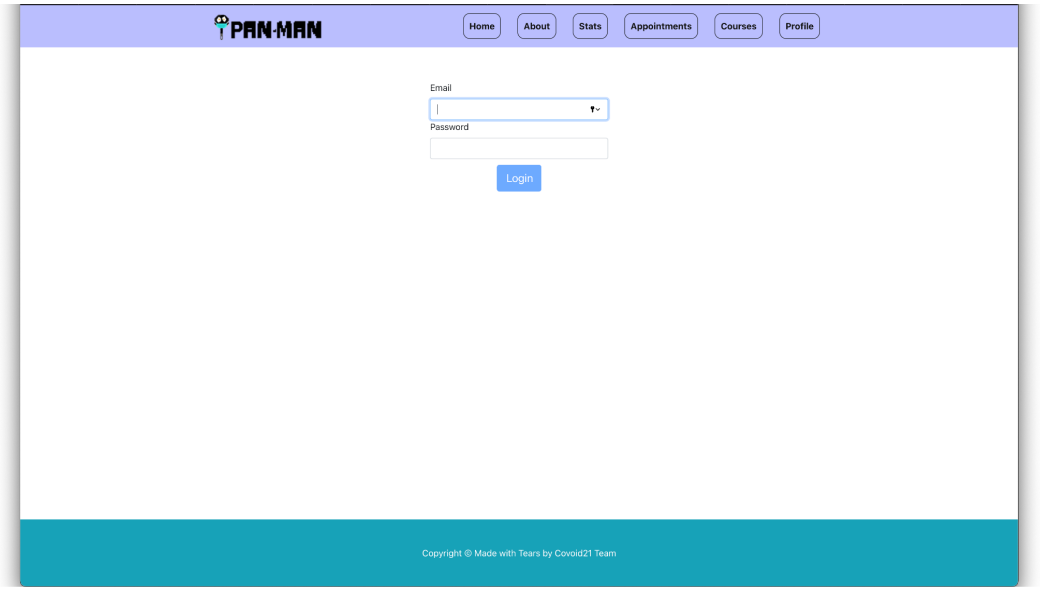

Figure 1: Login Page

Users can log in to the system through this page. Form components check whether the user provided a valid email address and password or not. If login is successful, the user is directed to the main page. This page is only accessible through the link http://localhost:3000/login.

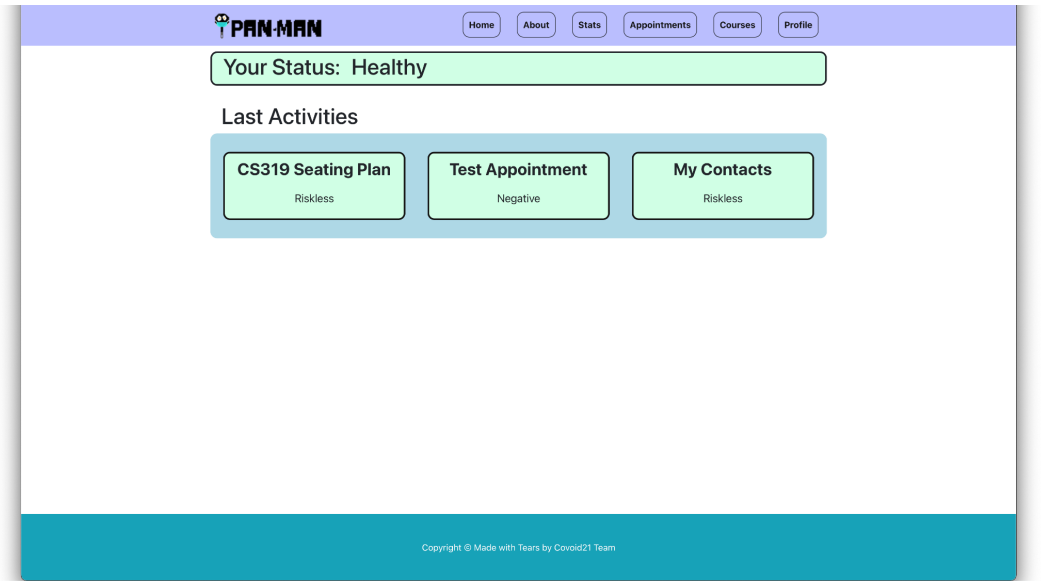

Figure 2: Main Page

On this screen, users can see their last activities and their risk status. This page is the general navigation page of the users. Clicking on the Pan-Man logo is the same as clicking on Home in the navigation bar.

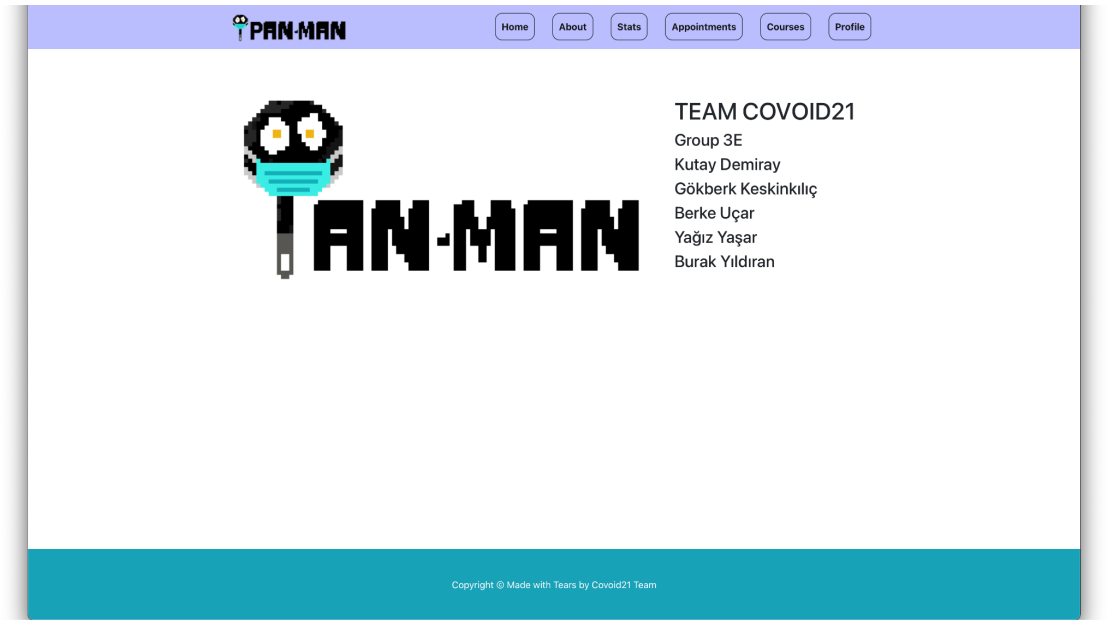

Figure 2: About Page

This page is just for credits, providing information about the development team and logo.

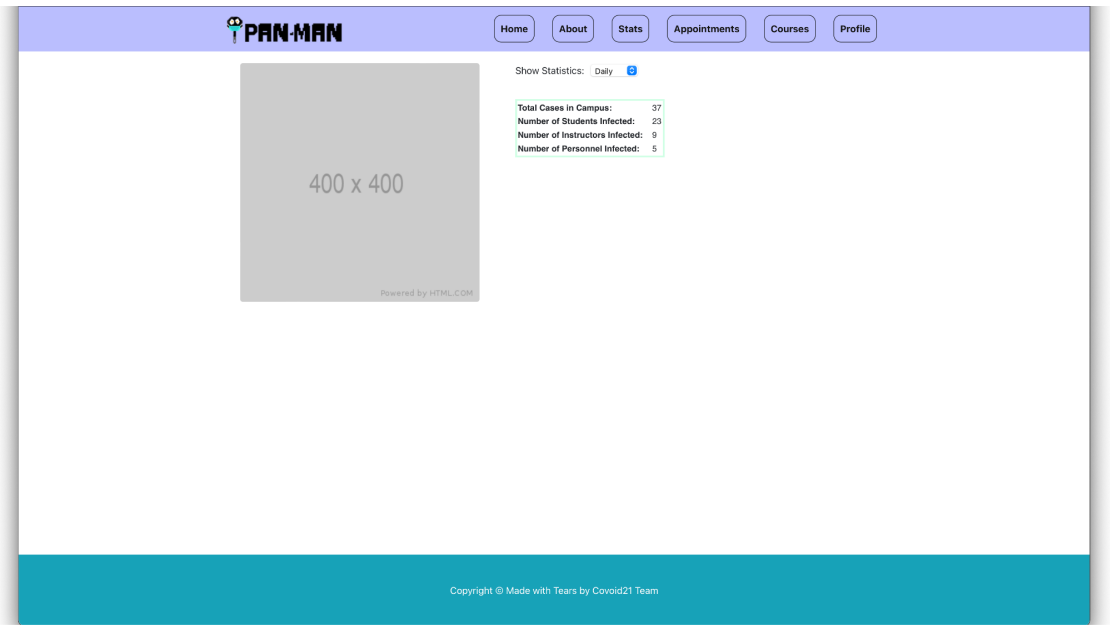

Figure 3: Stats Page

On the Stats page, users can visualize the current status. Note that this screen's functionalities are implemented in the backend, as it will be stated with its reasons in the "What We Did, Did Not, and Why" part.

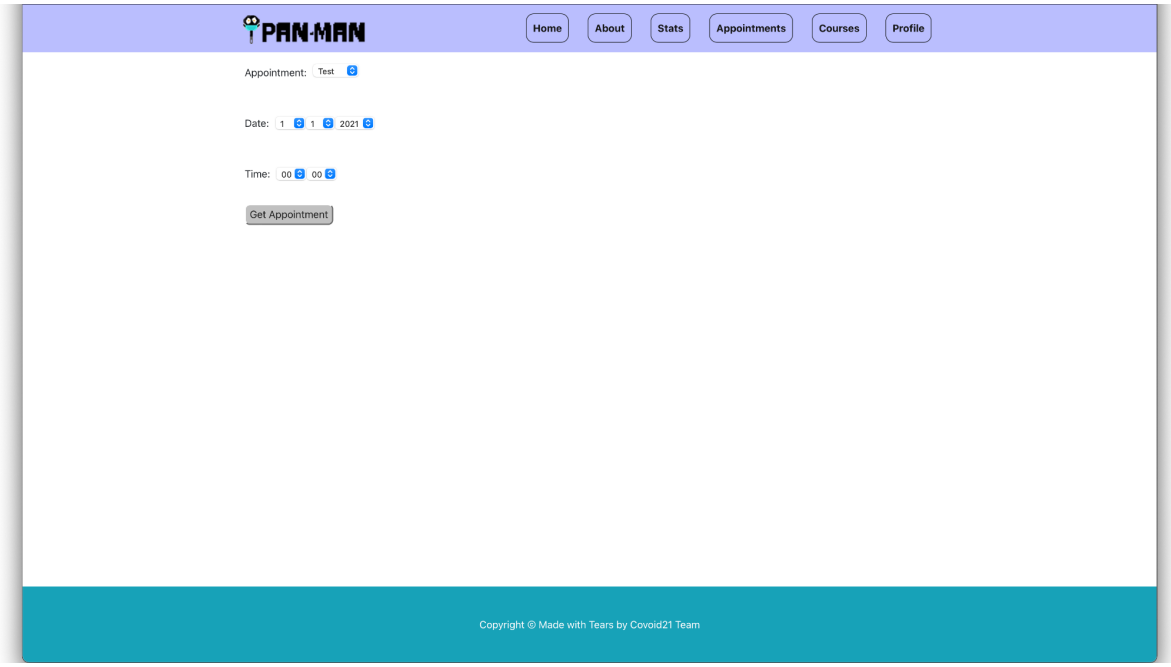

Figure 4: Appointments Page

On the Appointments page, users can book an appointment for their desired choice. Options are tests, doctor's, and sports facility appointments. After choosing the activity type suitable dates will be adjusted properly from the database side so that the user can only see

available dates. The same applies to time preference. Once the date is chosen only suitable hours will be shown to the user so that the user will not be confused and will not cause any unexpected behavior to the system.

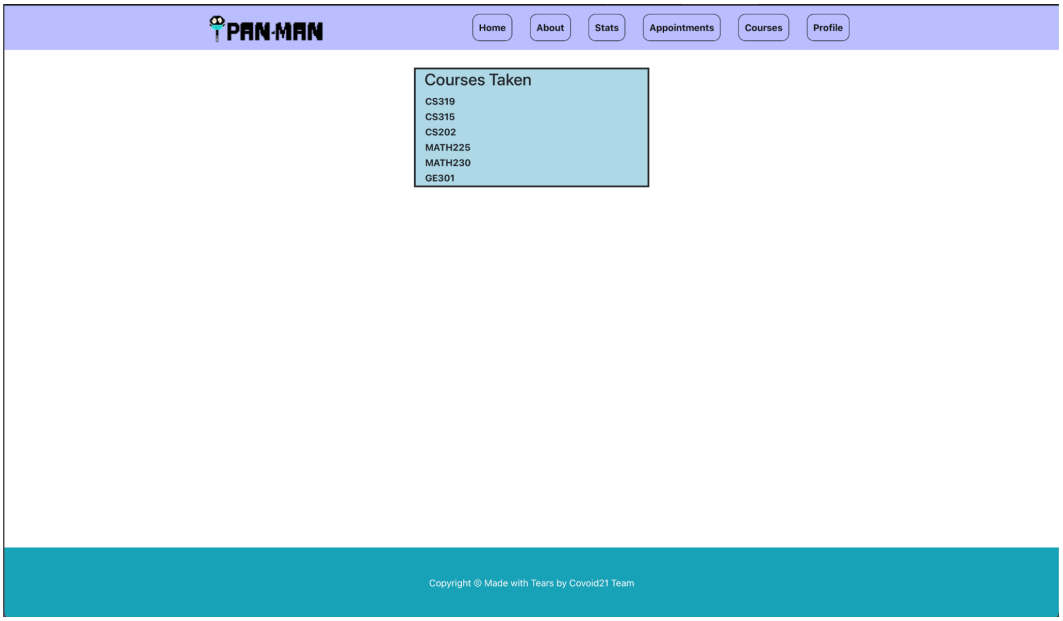

Figure 5: Courses Page

On this page, users can display courses taken. The courses are retrieved from the database by looking at the current user's id. Clicking on any of the courses, the user would proceed to the page where the user's close seats can be seen. On the forwarded page, the user would be able to update its close seat contacts, but it is not written.

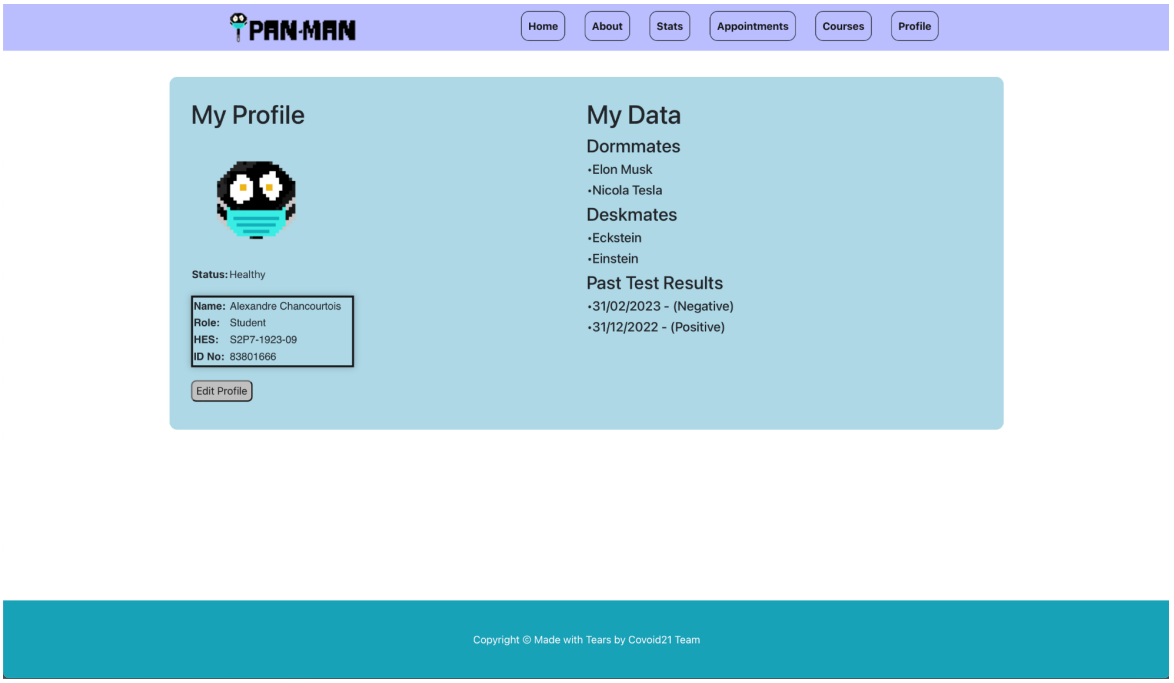

Figure 6: Profile Page

On this page, users can visualize their current status. The status is either healthy, contacted, or risky. Also, other additional data about their close contacts and recent test results are displayed.

4. Build Instructions

4.1. Backend Instructions

Fork/Clone this repository (https://github.com/berkeucar/CS319-Covoid21/) into your local machine. Make sure you are on the “finalbranch” branch. Open the project folder using IntelliJ Ultimate Edition 2021 (which should be free to obtain with an educational e-mail). IntelliJ Community Edition 2021 will probably work as well if you cannot obtain the Ultimate edition for some reason, but please note that we have not tested it.

Check the project SDK from File -> Project Structure -> SDK and make sure it is “Amazon Corretto version 11.0.13”. If it is something else, you can download it from Project Structure -> Platform Settings -> SDKs -> Add new ("+" button) -> Download JDK -> Version 11 and Vendor “Amazon Corretto 11.0.13”, then set the project SDK to that.

If there's no build configuration, select Run -> Edit Configurations -> Add new ("+" button) -> Spring Boot. Under "Build and run," select the “Amazon Corretto 11.0.13” SDK, select “com.covoid21.panman.PanmanApplication” for the main class, and select "@argfile (Java 9+)" for the "Shorten command line" option. The application should now be buildable and runnable.

If you would like, you may open com.covoid21.panman.PanmanApplication and experiment with various services' operations inside the "commandLineRunner" method. Some (mostly commented out) example code is available as examples on how to use the Service instances to perform some operations. If you want to use another Service instance, make sure to add it as a parameter of the "commandLineRunner" method.

The project is set to use our database on AWS by default. If you would like to use another database, you will need to change the following lines in the application.properties file under src/main/resources:

- `spring.datasource.url: Provide your own link for the database, note that format may change depending on your database setup.`
- `spring.sql.init.platform: We recommend you to still use postgres, however if you want to try another platform such as mysql, you will need to change this line.`
- `spring.datasource.username`
- `spring.datasource.password`
- `spring.datasource.driver-class-name: If you’re not using PostgreSQL, provide your platform’s driver class here.`
- `spring.jpa.properties.hibernate.dialect: If you’re not using PostgreSQL, provide your platform’s dialect here.`

You can observe the tables in the database using pgAdmin 4. To do that, use “postgres.czoavhjmmgdt.eu-central-1.rds.amazonaws.com” as host name/address and “5432” as the port. As previously mentioned, the login credentials for the database are available in the “application.properties” file of our folder. However, we have observed that although the database is supposed to be accessible everywhere,

sometimes AWS does not let you connect. Unfortunately, we could not find a permanent solution for this problem, but it can be fixed easily by adding you manually to the whitelist. If you run into such problems, please contact Kutay Demiray from Slack or at kutay.demiray@ug.bilkent.edu.tr.

In order to send manual HTTP packets to test the controllers, Postman can be used. However, as we, unfortunately, could not complete authentication features in time, you will have to comment out spring-boot-starter-security and spring-security-test dependencies in the pom.xml file and comment out the Spring Security parts in our code (IntelliJ should complain about these when you try to build).

4.2. Frontend Instructions

Running the frontend part requires some packages. Initially install node.js and npm and make sure you can use `npm` command.

`npm install bootstrap` is required to use some functionalities from the bootstrap library.

`npm install node-sass` is required to compile scss files to css files.

After following the above installment processes, now the project is ready to launch in IDE. Open the terminal of the IDE and change the directory by simply writing "`cd frontend`".

Once the current directory is the frontend folder, now one can run the react app by writing "`npm start`".

5. Work Allocation

We tried to allocate the work needed in our reports and the source code itself fairly and with respect to each project member's interests and knowledge.

Kutay Demiray

Requirement Analysis Report:

- Some of the functional and non-functional requirements
- Use Case Diagram
- Class Diagram
- Explanations of some classes
- Some of the Sequence Diagrams

Design Report:

- Subsystem Decomposition
- Final Object Design
- Design Patterns
- Class Interfaces

Implementation:

- Backend
    - Some AWS work
    - Entity classes
    - Repository classes
    - Service classes
    - Testing

Final Report:

- Build Instructions - Backend
- Some Backend paragraphs in What We Did, Did Not, and Why

Gökberk Keskinkılıç

Requirement Analysis Report:

- Non-functional Requirements
- Activity Diagrams

- Ensuring anonymity of the report

Design Report:

- Deployment Diagram
- Object Design Tradeoffs
- Boundary Conditions
    - Initialization
    - Termination
    - Failure

Implementation:

- Backend
    - React
    - CSS

Final Report:

- Frontend build instructions

Berke Uçar

Requirement Analysis Report:

- One of the Non-functional Requirements
- Class diagram
- Sequence Diagrams and their explanations
- Mock-up Screens and some of their explanations

Design Report:

- One of the Design Goals
- Subsystem decomposition
- Final object design
- Design patterns
- Class interfaces

Implementation:

- Frontend
    - Some design changes and planning
- Backend
    - Entity classes

    - Authentication
    - Registration
    - E-Mail system
    - Some of the Controller classes

Final Report:

- Project experience

Yağız Yaşar

Requirement Analysis Report:

- Some of the Overview section
- Some of the non-functional requirements
- Use-case diagram explanations
- Some of the functional requirements
- Some of the mock-up screens' explanations
- General organization of the report

Design Report:

- Introduction
- Purpose of the System
- Design Goals
- Access Control Matrix
- Internal and External Packages
- General organization of the report

Implementation:

- Backend
    - Authentication
    - Registration
    - Controller classes
    - Some of the Repository classes
    - Some of the Service classes
    - Postman Rest API Testing

Final Report:

- What We Did, Did Not and Why
- General organization of the report

Ömer Burak Yıldıran

Requirement Analysis Report:

- Some of the functional and non-functional requirements
- State Diagrams
- State Diagram Explanations

Design Report:

- Hardware / Software Mapping
- Persistent Data Management
- Access Control and Security
- Boundary Conditions
    - Initialization
    - Termination
    - Failure

Implementation:

- Backend
    - Some AWS work
    - Database configuration
    - Some Controller classes
    - Some Repository classes
    - Postman RestAPI testing

6. What We Did, Did Not, and Why

We stated 9 functional requirements in our 2nd Iteration Requirement Analysis Report: Viewing Pandemics Statistics, Tracking Infection, Vaccination, and Testing Status, Notifications, Making Test Appointments, Facility Appointments including Health Appointments, Indicating Seating Plans for Courses, Viewing Policies inside the University, Editing and User Information. We did not implement the methods required for Viewing Pandemics Statistics.

We did not connect the Frontend and Backend of the program, so implemented requirements are done through the Backend, and methods can be requested using Postman

API. These requests would create Database instances for corresponding objects using JSON data type.

We excluded Google Maps, STARS, and HES API out of our external packages. We realized that we would not get the authorization to use STARS and HES API in time, thus we excluded them, instead of using these APIs our purpose was for the user to enter their data manually. We did not implement Google Maps API because statistics was not one of our main implementation priorities. We left it to implement it to last but we did not have enough time for it.

We stated that we were going to use MySQL in our 2nd Iteration Design Report but we decided to go with PostgreSQL again. Our first intention to change to MySQL was that it was paid service to use PostgreSQL, but we managed to get PostgreSQL service free via AWS, thus we used PostgreSQL and AWS for the database.

We stated 4 non-functional requirements in our 2nd Iteration Design Report: Information Security & Privacy, Functionality, Usability, and Maintainability. As it is written before we excluded some of the external packages, thus Information Security & Privacy came a bit less of a concerning topic for us other than password encryption. We used BCryptPasswordEncoder to store the password of someone, it is not allowed to see from anywhere in the database unless the user is not registered by Registration Request. For Functionality, the same thing goes for here. We excluded one of our functional requirements and APIs, which caused our program to be less functional, but in the case of Usability, the program became more usable because of the usage of fewer APIs and it increased. In the case of Maintainability, we wrote Generic classes that provided a better inheritance structure to our project to be more maintainable and changeable.

We implemented the Entity classes from our design class diagram with very few problems. We needed to add some Spring-specific annotations and make some small changes (such as adding/changing the type of ID and some other few parameters) to make them work. The biggest change is the removal of the SeatingPlan class, which is replaced with StudentCloseSeats in the final code. That had to be done because Spring does not support multidimensional collections (such as the adjacency matrix in the original SeatingPlan) very well. StudentCloseSeats instead provides the neighboring students of a single student, and a Course contains a collection of several StudentCloseSeats, which works better with database operations.

Unfortunately, the Manager classes we had could not be implemented as successfully. As none of us had previously worked with Spring (or another similar web development library), we did not know very well how these classes could be implemented while writing the previous reports. Due to the Controller (RestAPI)-Service-Repository layer system of Spring, we could not work much with actual Manager classes until we could work out what can be done with Spring's layer system and what can be done with our higher-level classes, so they remain mostly unimplemented. We were able to write DatabaseManager, which should be able to get/set various entities using the Service class instances it contains (we could not adapt DatabaseManager into our code in time, however, the Service instances it delegates its functionality to should work), and UserManager was partially written. The other Manager functionality we were able to implement was done so in Service classes such as CourseService which supports adding new students and neighborship relations.

In the frontend, we couldn't set up a register page since it was not handled in the backend. Even though there is an implemented login page it is not functional since we could not connect it to the backend. Some pages are missing some components in it which is again caused by the same issue, having a disconnected frontend and backend. Different views for different user types are not available. Instead of Heatmap, there is a sample image to state its position in the Stats page and there is no graph to demonstrate cases over time. Notification pop-up display is not present because there is no possible notification user can receive at this stage of the development process. Edit my profile section in profile page has been removed to another place where users can access with a button. However, it is a nonfunctional button at this stage that redirects the user nowhere.