CS 319 Term Project

PHS: Health Center Management App

Design Report

Alper Kandemir 21703062

Bora Fenari Köstem 21801823

Oğulcan Karakollukçu 21802642

Instructor: Eray Tüzün

Teaching Assistant(s): Elgun Jabrayilzade and Muhammad Umair Ahmed

May 2, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Object-Oriented Software Engineering course CS319.

1 Introduction

1.1 Purpose of the system

The app's purpose is to provide an easy and useful appointment system, a secure and reliable place to hold patients' health history for Bilkent Health Center. Our application is a web-based hospital management system that we designed for Bilkenters. Our application has the features of making appointments, cancelling and postponing the appointment to be made by the secretary. It has the necessary functions for MR, Lab, and blood samples requested by the doctor to appear on the doctor's home page so that the preliminary examination results are immediately displayed in front of the doctor. In this way, we aim to ensure a quick and easy examination. In addition to these, it is possible to see the number of covid cases and health news within the school in our application.

1.2 Design Goals

1.2.1 Maintainability

Application subsystems will have a hierarchical structure so that new features will be added easily without huge changes in the upper systems. Additionally, code will be written in a way that can be easily understood by others. Since we use the PHP Laravel framework in the back-end part, the reliability of our application will increase as the framework improves itself.

1.2.2 User – Friendliness

Thanks to our user-friendly interface, users can easily switch between menus and our easy to understand icons make our site easier to navigate. Using minimalist

design on our site, we avoided things that might confuse the user. We gave our buttons names that are understandable and contain a summary of what the button does so that the user can understand it more easily.

1.2.3 Performance

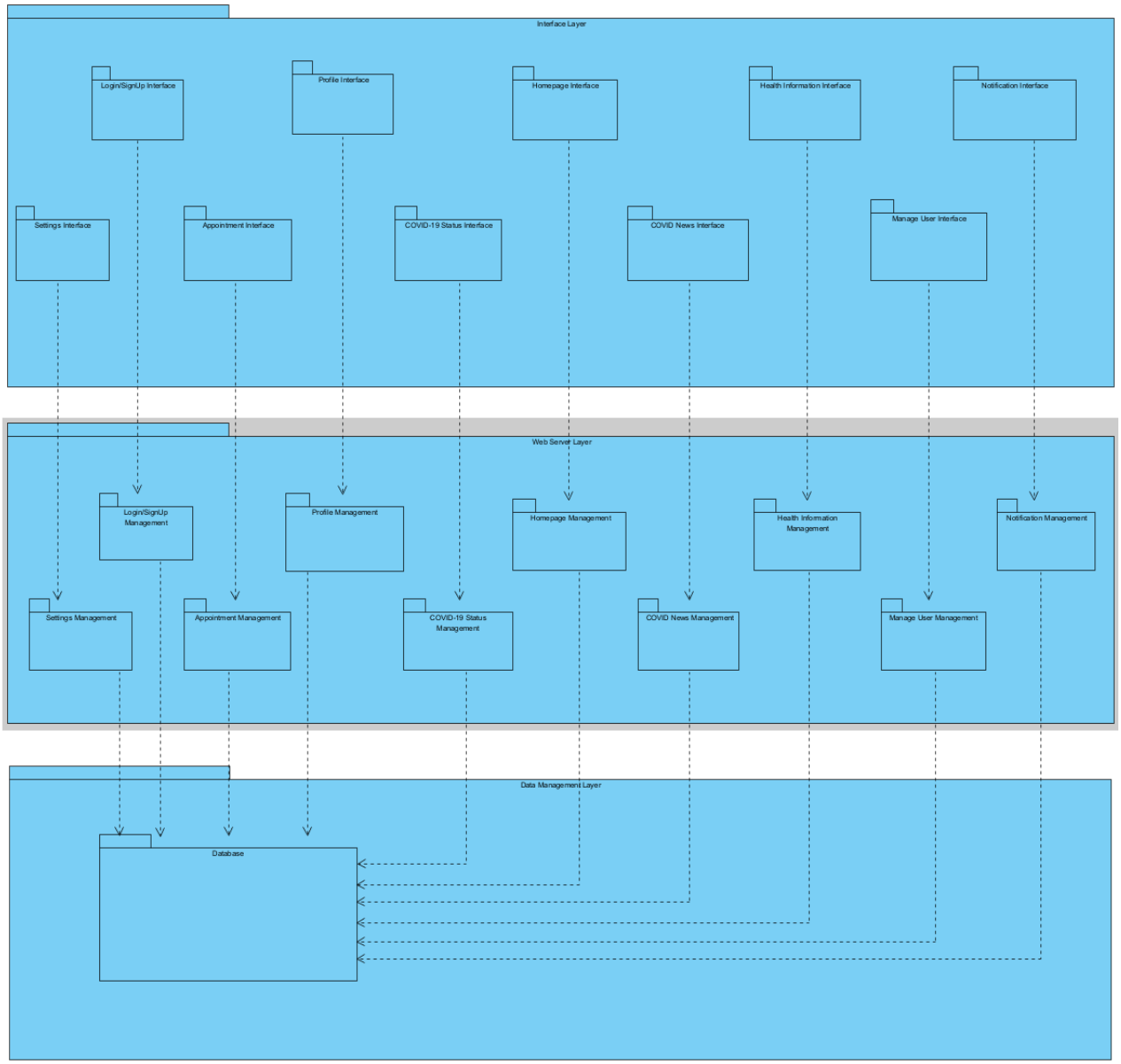
Since localhost is used in the development phase, there will be no problem with multiple user usings simultaneously, but later on, systems such as AWS can be rented, and performance problems can be minimised. Also, our application will be web-based, and it will be able to run on almost any computer and phone. For our application, any browser that supports HTML5 will be able to access our program.

1.2.4 Security

Personal information such as student numbers, phone numbers, and passwords will be encrypted with the sha256 hashing algorithm in our application. Files and folders with health history will be encrypted with the encrypt functions of windows. Our goal is to create our system with full user privacy.

2 Software Architecture

2.1 Subsystem Decomposition



2.1.1 Interface Layer

Settings Interface: The settings interface is a subsystem that contains all the UI elements and user interactions required by the settings page.

Login/SignUp Interface: The login/SignUp interface is a subsystem that contains all the UI elements and all the user interactions required by the login/sign up page.

Appointment Interface: The appointment interface is a subsystem that contains all the UI elements and all the user interactions required by the appointment page.

Profile Interface: The profile interface is a subsystem that contains all the UI elements and all the user interactions required by the profile page.

COVID-19 Status Interface: COVID-19 STATUS interface is a subsystem that contains all the UI elements and all the user interactions required by the covid-19 status page.

Homepage Interface: The homepage interface is a subsystem that contains all the UI elements and all the user interactions required by the homepage page.

COVID News Interface: COVID news interface is a subsystem that contains all the UI elements and all the user interactions required by the covid news page.

Health Information Interface: Health information interface is a subsystem that contains all the UI elements and user interactions required by the health information page.

Manage User Interface: The manage User Interface interface is a subsystem that contains all the UI elements and user interactions required by the manage user page.

Notification Interface: The notification Interface interface is a subsystem that contains all the UI elements and all the user interactions required by the notification page.

2.1.2 Web Server Layer

Settings Management: Settings management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the settings page's front-end components.

Login/SignUp Management: Login/SignUp management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the login/sign up page's front-end components.

Appointment Management: Appointment management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the appointment page's front-end components.

Profile Management: Profile management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the profile page's front-end components.

COVID-19 STATUS Management: COVID-19 STATUS management is a layer between our UI elements and the database. It enables us to fetch, store, update and delete the necessary data from our database for the covid-19 status page's front-end components.

Homepage Management: Homepage management is a layer between our UI elements and the database. It enables us to fetch, store, update and delete the necessary data from our database for the homepage page's front-end components.

COVID News Management: COVID news management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the covid news page's front-end components.

Health Information Management: Health information management is a layer between our UI elements and the database. It enables us to fetch, store, update and delete the necessary data from our database for the health information page's front-end components.

Manage User Management: Manage User Interface management is a layer between our UI elements and the database. It enables us to fetch, store, update

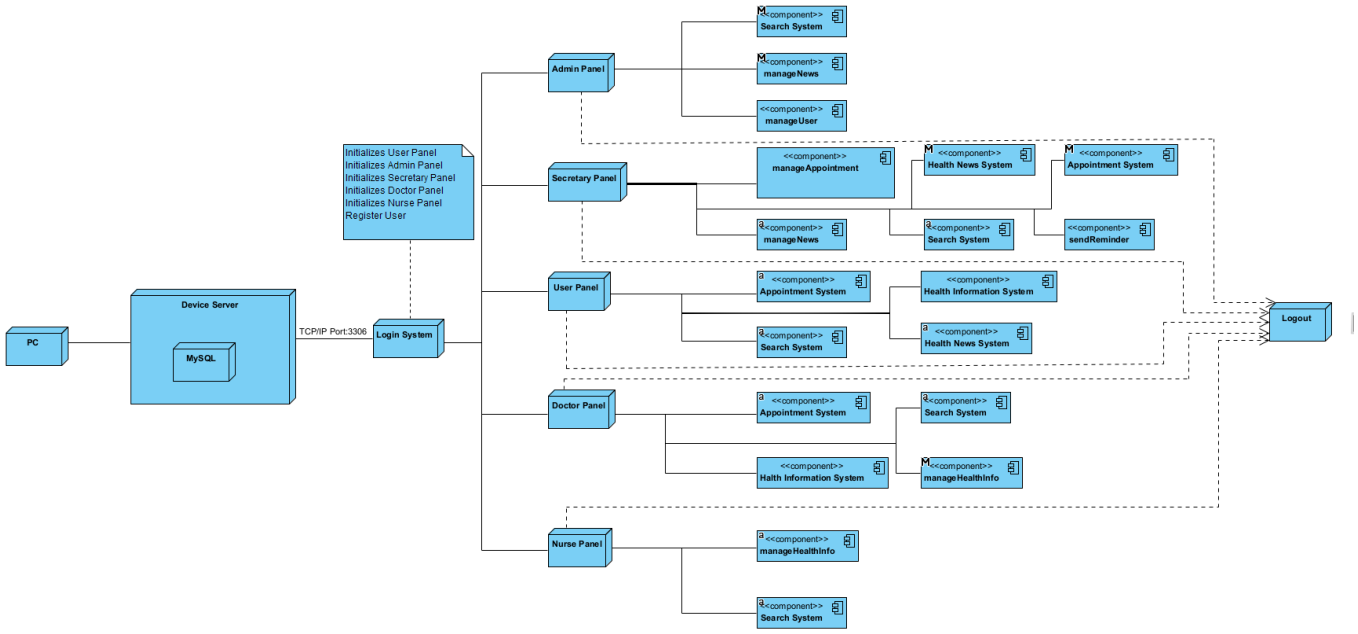
and delete the necessary data from our database to manage the page's front-end components.

Notification Management: Notification Interface management is a layer between our UI elements and the database. It lets us fetch, store, update and delete the necessary data from our database for the notification page's front-end components.

2.1.3 Data Management Layer

Database: The database is a part of the Data Management Layer. The layers in the Web Server layer can access the database through this layer. The database management layer is where our data will be stored. Our program consists of 3 layers. The user interface, an application that includes an appointment system, user management etc. and lastly database. When a user logs in to our program, if the login credentials entered by the user match those in the database, the user information in the database is retrieved and assigned to a PHP session. After that, we aimed to reduce the density in the database as much as possible By pulling information from the PHP session instead of the database connection unless the user requires additional information or changes. Our database keeps the location of the files with user health histories and user information as data. Information exchange operations are provided by the connection between the backend of our application and the database.

2.1.4 Deployment Diagram



The computer will be connected to the Device Server containing the DataBase.
We will use MySQL 8.0 as a database. We will collect and distribute our information through the Device Server. The device Server will be connected to the Login System. Then, Login System will be divided into five panels as Admin, Doctor, Nurse, User and Secretary. Each panel will have its own components, as well as components that they have in common. Moreover, each panel will be connected to the Logout.

2.2 Hardware / Software Mapping

Since our application is not heavy, a production server powerful enough to run internet browsers is sufficient. However, to prevent performance problems that may occur, we follow the recommended requirements. The recommended requirements are as follows[1]:

- 3.3 gigahertz (GHz) or faster 64-bit dual-core processor
- 4 GB RAM
- Super VGA with a resolution of 1024 x 768
- Bandwidth greater than 50 KBps (400 kbps)
- Latency under 150 ms
- Database with 20GB of storage

We think that a storage area of 20 GB will be sufficient because we will store only PDF and PNG files, which can be considered large, and the rest will not require much space. Since our application is a web application, there is no specific hardware and software requirement, but it is sufficient to have an internet connection, and an HTML5 supported browser. Because of the flexibility of web applications, our app can run on smartphones, tablets, and personal computers. For this reason, our application is designed to be suitable for any screen size.

2.3. Persistent Data Management

We use the server's local storage and database to store the necessary data. At the same time, if the user wishes, it stores the user's login information on the user's local computer by using browser cookies to facilitate the next login. While information such as the user's name, surname, password, Bilkent ID, e-mail, phone number, HES code, date of birth, and gender are kept in the database, the files containing the profile photo and

2.4 Access Control and Security

Different types of users have different access to the application. We demonstrated this by mapping appropriate use cases to each actor in our Use-Case diagram and mapping some functions to specific users in our Class diagram.

Classes / Actors	User	Secretary	Doctor	Nurse	Admin
Login	submitLoginInfo() forgotPassword() displayCovidCases() goToAboutCovid()	submitLoginInfo() forgotPassword() displayCovidCases() goToAboutCovid()	submitLoginInfo() forgotPassword() displayCovidCases() goToAboutCovid()	submitLoginInfo() forgotPassword() displayCovidCases() goToAboutCovid()	submitLoginInfo() forgotPassword() displayCovidCases() goToAboutCovid()
MainPageDashboard	accessProfile() accessSettings() accessAppointmentsPage() accessHealthInfo() displayNews()	accessProfile() accessSettings() accessHealthInfo() displayNews()	accessProfile() accessSettings() accessAppointmentsPage() accessHealthInfo() displayNews()	accessProfile() accessSettings() accessAppointmentsPage() accessHealthInfo() displayNews()	accessProfile() accessSettings() accessAppointmentsPage() accessHealthInfo() displayNews()
Profile	displayUserDetails() displayCovidStatus() displayHealthRecords() setHESCode()	displayUserDetails() displayCovidStatus() displayHealthRecords() setHESCode()	displayUserDetails() displayCovidStatus() displayHealthRecords() setHESCode()	displayUserDetails() displayCovidStatus() displayHealthRecords() setHESCode()	displayUserDetails() displayCovidStatus() displayHealthRecords() setHESCode()
AppointmentsPage	displayAppointmentHistory() showDetails() makeAppointment() cancelAppointment() changeAppointment()	displayAppointmentHistory() showDetails() makeAppointment() cancelAppointment() changeAppointment()	displayAppointmentHistory() showDetails()	displayAppointmentHistory() showDetails()	displayAppointmentHistory() showDetails() makeAppointment() cancelAppointment() changeAppointment()
HealthInfo	displayHealthInfo() downloadFile()	downloadFile()	displayHealthInfo() downloadFile()		displayHealthInfo() downloadFile()
Settings	changePassword()	changePassword()	changePassword()	changePassword()	changePassword()
AppointmentList	viewAppointmentDetails()				
DoctorList		viewDoctorDetails()	viewDoctorDetails()		
UserList				viewUserList()	viewUserList()

2.5 Boundary Conditions

2.5.1. Initialization

Bilkent Health Center needs to download the XAMPP web server and start Apache and MySQL modules to start the website. Our application is a website; the front-end uses HTML, CSS and JavaScript. The first page of this website is the login page, accessible by anybody with an Internet connection. This page also displays the latest health news at Bilkent. With correct identifiers, one can access the rest of the website, and these identifiers are provided directly by the university. Therefore, there is no registration option.

2.5.2 Termination

To stop the website, Bilkent Health Center needs to stop Apache and MySQL modules from XAMPP. Admins of the website can proceed to maintenance which will shut down the website for a specific time. However, all data is saved constantly to avoid all possible losses.

2.5.3. Failure

Data that is not successfully saved is lost. The system stops processing by giving a connection error when the user's connection is lost. If the admin tries to add the existing user again, an error is given. In case of a system crash, it continues with the last backed up data. When a doctor, nurse or a student leaves the health center, the information of the people who leave is deleted from the system by the admin, and the system does not allow the creation of a user with the id of the deleted person.

3 Low-level Design

3.1 Object Design Trade-offs

3.1.1. Maintainability versus Performance

We try to write our code as cleanly as possible, and we explain what we are doing in the comments lines in our code. Thus, we can easily do it when we need to make changes or add new features later. In this way, we ensure that our application is sustainable by writing high-quality code. The bad benefit of this for us is that we do not add very complex codes to increase performance and that we try to solve the problems we encounter in the simplest way, making our code less bug-prone.

3.1.2. Rapid development versus Functionality

Due to time constraints, we did not add many functions other than the primitive functions to provide a working application. For this reason, our application does not offer many options to the user. Instead, just the necessary options are offered. This is not a big problem since our code is maintainable because we can easily add additional functions to our code in the future.

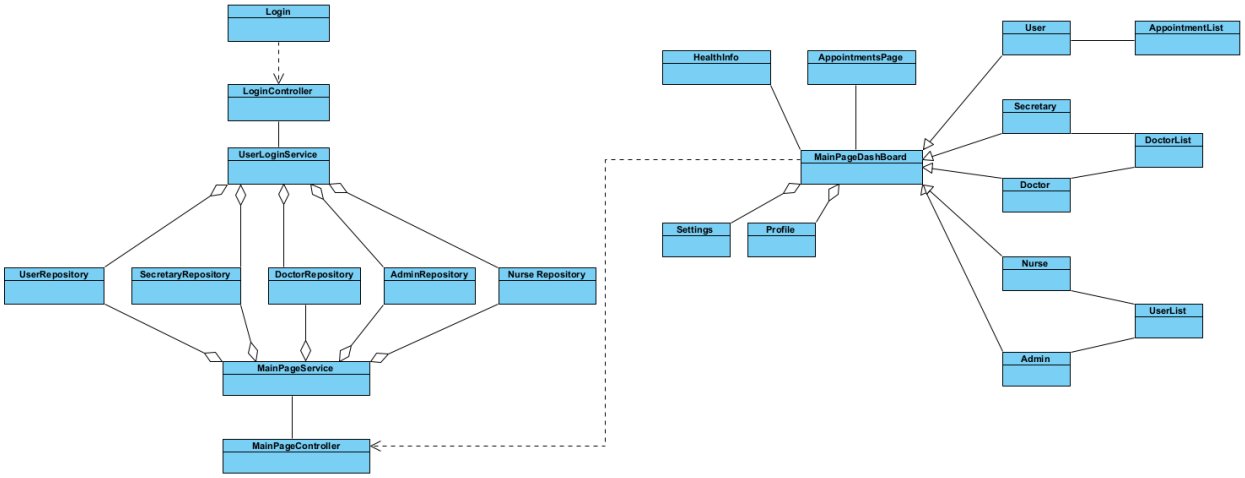
3.1.3. Memory versus Performance

We kept the number of classes small and crammed the necessary functions into these classes, so we tried to use the memory as best as possible in our application. This is because we want to make our project scalable for further growth. The fact that we do not have functions that will reduce performance other than the primitive functions and that the memory is used well will ensure that our application works fast and fluently.

3.1.4. Cost versus Robustness

We used free tools in the writing of our application, we designed the site ourselves, and we did not receive professional help by paying money, so the cost of our application is low, but our robustness is just as low.

3.2 Final Object Design



3.3 Packages

3.3.1 Internal Packages

3.3.1.1 Appointment Manages

This package contains the classes for the Appointment feature of the application which includes creating, deleting and modifying Appointments. This package also includes doctor's appointment operations like requests for additional tests and handling diagnosis reports.

3.3.1.2 Health History

Classes in this package contain the functions for adding new files to the database, searching the database for necessary files by name, type and date.

3.3.1.3 Utilities

This package includes the classes for small features like covid news and announcements.

3.3.1.4 Login

Includes classes for login and user PHP session management.

3.3.1.5 Admin Settings

This package includes specific admin functions like seeing actions logs and staff management.

3.3.1.6 Encryption

Contains the necessary classes to encrypt private information.

3.3.1.7 User Management

The User Management package contains the classes for user creation, deletion and management.

3.3.1.8 Model

This package contains the entities that are shown in our class diagram. They have instance variables, getters, setters, and some additional methods.

3.3.1.9 Database

This package includes the classes for the data management between the backend and the database.

3.3.2 External Library Packages

3.3.2.1 Laravel Middleware

This package is for the Laravels security layer that controls user actions and login.

3.3.2.2 Laravel Routes

This package is for UI redirection and management.

3.3.2.3 Laravel Controller

This package contains the necessary controller classes of Laravel that act as a bridge between routes, middleware and our packages.

3.4 Design Patterns

3.4.1 Facade Design Pattern

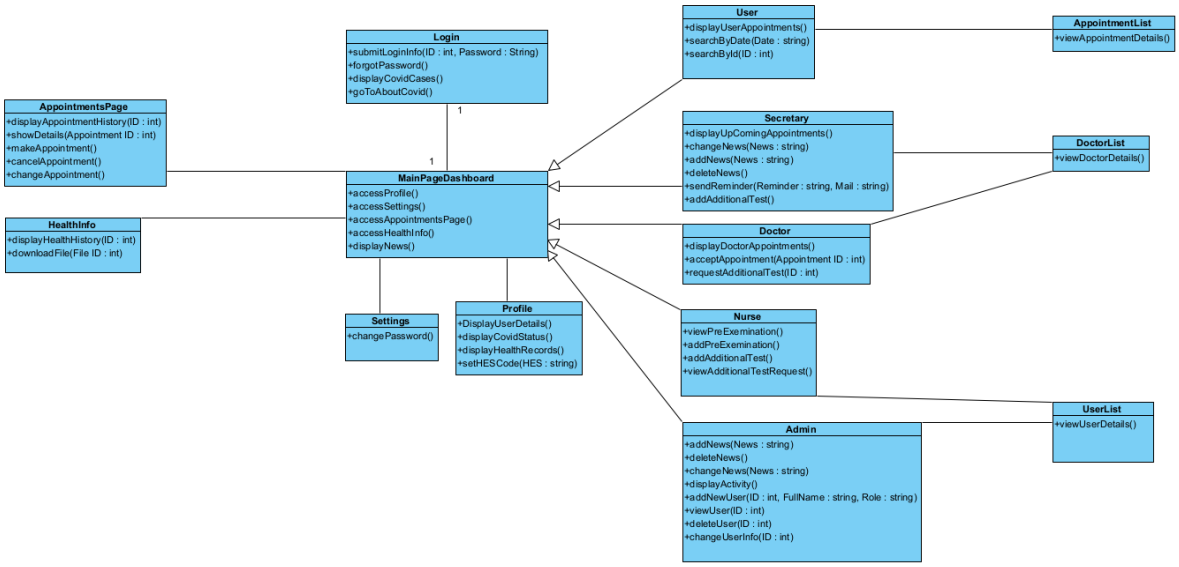
Accessing the subsystems would be very difficult without a proper interface since many different classes would have to be accessed. The user needs to perform operations with a simple interface without dealing with subsystems. Also, changes in one class should not affect the rest of the program's implementation. So the user needs to interact only with the necessary parts. The Facade Design pattern provides a simple interface to a complex subsystem with many moving parts.

3.4.2 State Design Pattern

Since each appointment consists of different states, we used the state design pattern to control the whole appointment system for these different appointment states. These states are “created”, “the patient is here”, “the patient did not come”, “the additional tests are awaited”, and “completed.”

3.5 Class Interfaces

3.5.1 User Interface Management Layer Class Interfaces



The user interface management layer establishes the connection between the user and the system. The format of the files for the user management layer is a combination of HTML and CSS files with Tailwind CSS. The UI management layer has dynamic objects of HTML. It connects with the backend with the different IDs of each object and POST methods. UI management is provided by middleware and routes in PHP's Laravel framework. All connections are mostly middleware controlled, as a user must log in before accessing their information. Since the user will be directed to the User Dashboard first when they log in, all classes are in aggregation with the User Dashboard.

3.5.1.1 Appointments Page

The appointment page allows users and doctors to see their past and incoming appointments. Also, the secretary can create and modify appointments on this page.

Operations:

public displayAppointmentHistory(ID: int): This method will show the appointments that can be seen by the user according to the ID. It is taken as a parameter.

public showDetails(Appointment ID : int): This method will display the appointment details according to the appointment ID that is received.

public makeAppointment(): This method will create a new appointment.

public cancelAppointment(): This method will cancel an existing appointment.

public changeAppointment(): This method will modify an existing appointment.

3.5.1.2 Health Info

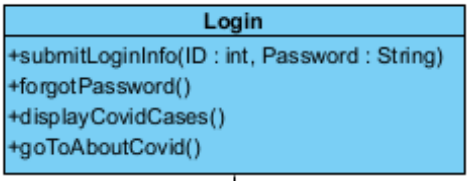
Health info page allows users to download some health files that provide useful information. This page also displays the last appointments that the user had.

Operations:

public displayHealthHistory(ID : int): This method will show the health history of the owner of the user ID it receives.

public downloadFile(File ID : int): This method will download the file associated with the given id.

3.5.1.3 Login



This page displays some Bilkent health news. Moreover, users can type their ID and password to access the rest of the website. Finally, users can launch a process of creating a new password if they forgot theirs.

Operations:

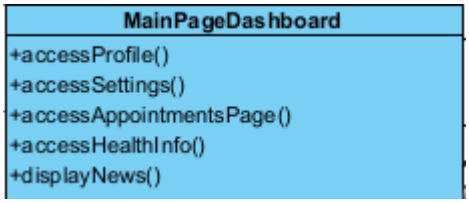
public submitLoginInfo(ID : int, Password : String): It takes the login credentials from the user and compares them with the database. If it's a match, it logs in the user.

public forgotPassword(): When it is used, it sends the email to users to change their password.

public displayCovidCases(): It displays the new covid case news entered by the Secretary or Admin

public goToAboutCovid(): Redirects user to governments covid page.

3.5.1.4 Main Page Dashboard



On the main page dashboard, clickable buttons redirect users to many different pages (Profile, Settings, Appointments, Health Information). As the login page, health news is readable on the main page.

Operations:

public accessProfile(): From the main page, all users can access the profile page via a button on the header.

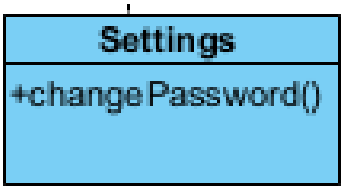
public accessSettings(): From the main page, all users can access the settings page via a button on the header.

public accessAppointmentsPage(): From the main page, all users can access the appointments page via a button on the header.

public accessHealthInfo(): From the main page, all users can access the health information page via a button on the header.

public displayNews(): All users can see the Bilkent health news on this page.

3.5.1.5 Settings

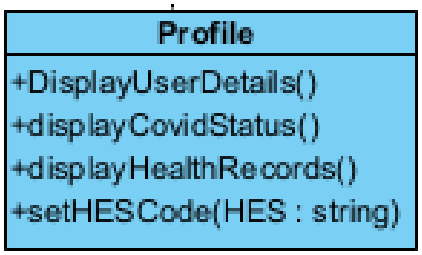


Settings page is for changing passwords.

Operations:

public changePassword(): The user can change their password by typing the current password and the new one they want.

3.5.1.6 Profile



Profile page displays all user health information (Name, ID, age, gender, phone number, weight, height, blood type, COVID status). Users can also send their HES code on this page.

Operations:

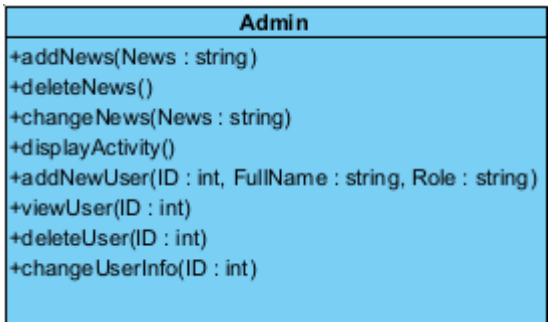
public displayUserDetails(): This method will pull the user's information from the session and display it.

public displayCovidStatus(): This method will show the person's covid-19 status.

public displayHealthRecords(): This method enables users to see their latest past appointments.

public setHESCode(HES : string): With this method, users can enter their HES code into the system.

3.5.1.7 Administrator



Administrators have a wide range of different rights to manage the system's users.

Operations:

public addNews(News : string): Admins can write news that will be displayed on login and main page.

public deleteNews(): Admins can remove news currently displayed on login and main pages.

public changeNews(News : string): Admins can edit news currently displayed on login and main pages.

public displayActivity(): It displays the role, date, hour, ID and activities of logged-in users.

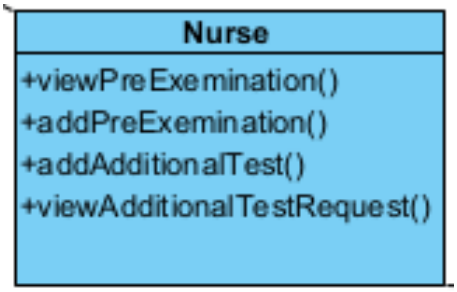
public addNewUser(ID : int, FullName : string, Role : string): Admins can create and add a new user into the system.

public viewUser(ID : int): Admins can access all the info about every application user.

public deleteUser(ID : int): Admins can remove a user from the system.

public changeUserInfo(ID : int): Admins can edit every user's information.

3.5.1.8 Nurse



Operations:

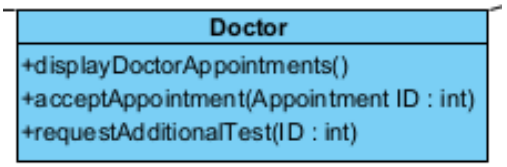
public viewPreExemination(): This shows a form to fill with the patient's pre-examination results.

public addPreExemination(): This method adds a new pre-examination to the database.

public addAdditionalTest(): After an appointment, if the doctor wants the patient to have more examinations/tests, a nurse can add them to the system.

public viewAdditionalTestRequest(): Gives the list of additional test requests about the doctor's appointment.

3.5.1.9 Doctor



Doctors have some rights that are different from the other users of the application.

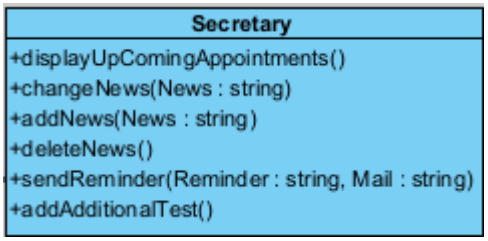
Operations:

public displayDoctorAppointments(): This method will display the doctor's appointments for that day.

public acceptAppointment(Appointment ID : int): This method will accept the appointment with the entered appointment ID.

public requestAdditionalTest(ID: int): The method creates a request to the nurse for the patient with the entered ID.

3.5.1.10 Secretary



Secretary is the only user that can manage appointments. Secretary, like the admins, can manage health news.

Operations:

public displayUpComingAppointments(): Secretary can see all the upcoming appointments on the agenda.

public changeNews(News : string): Admins can edit news currently displayed on login and main pages.

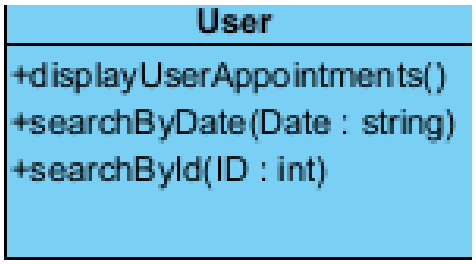
public addNews(News : string): Secretary can write news that will be displayed on the login and main page.

public deleteNews(): Secretary can remove news currently displayed on login and main pages.

public sendReminder(Reminder : string, Mail : string): Secretary can send a notification to the patient that they will have an appointment soon.

public addAdditionalTest(): After an appointment, if the doctor wants the patient to have more examinations/tests, the secretary can add them to the system.

3.5.1.11 User



User has the option to see his future appointments.

Operations:

public displayUserAppointments(): Users can see all their appointments (incoming and past).

public searchByDate(Date : string): Users can search specific appointments by their date.

public searchById(ID : int): Users can search specific appointments by their ID.

3.5.1.12 Appointment List

Operations:

public viewAppointmentDetails(): Gives a list of the upcoming and past appointments with details.

3.5.1.13 Doctor List

Operations:

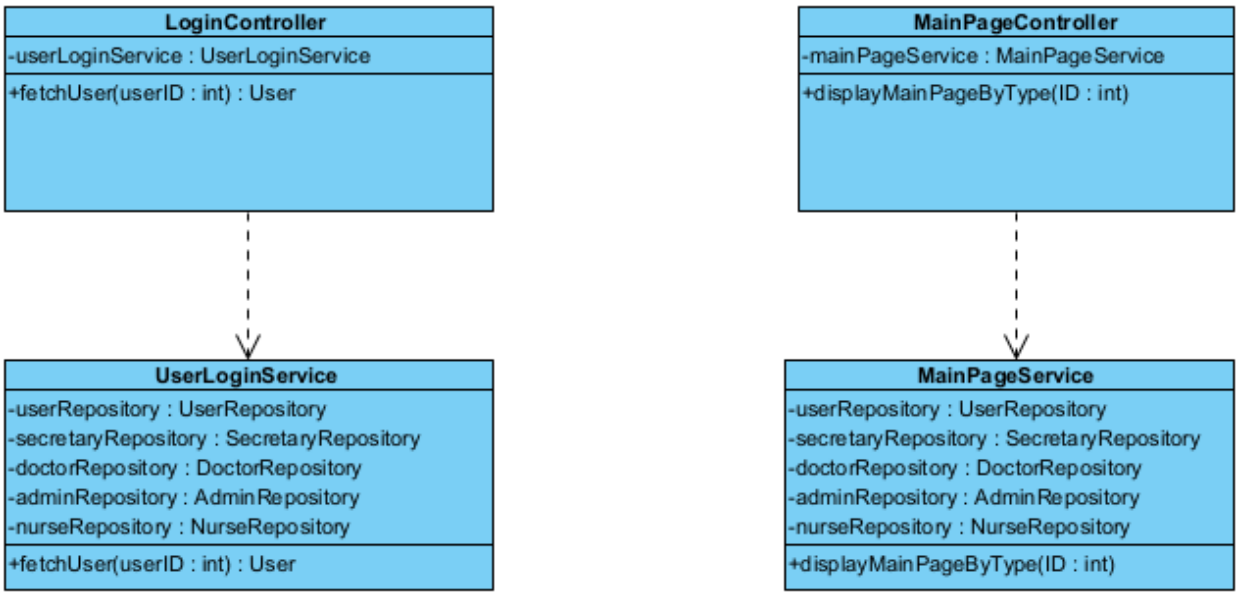
public viewDoctorDetails(): Returns a list of the doctors with their details.

3.5.1.14 User List

Operations:

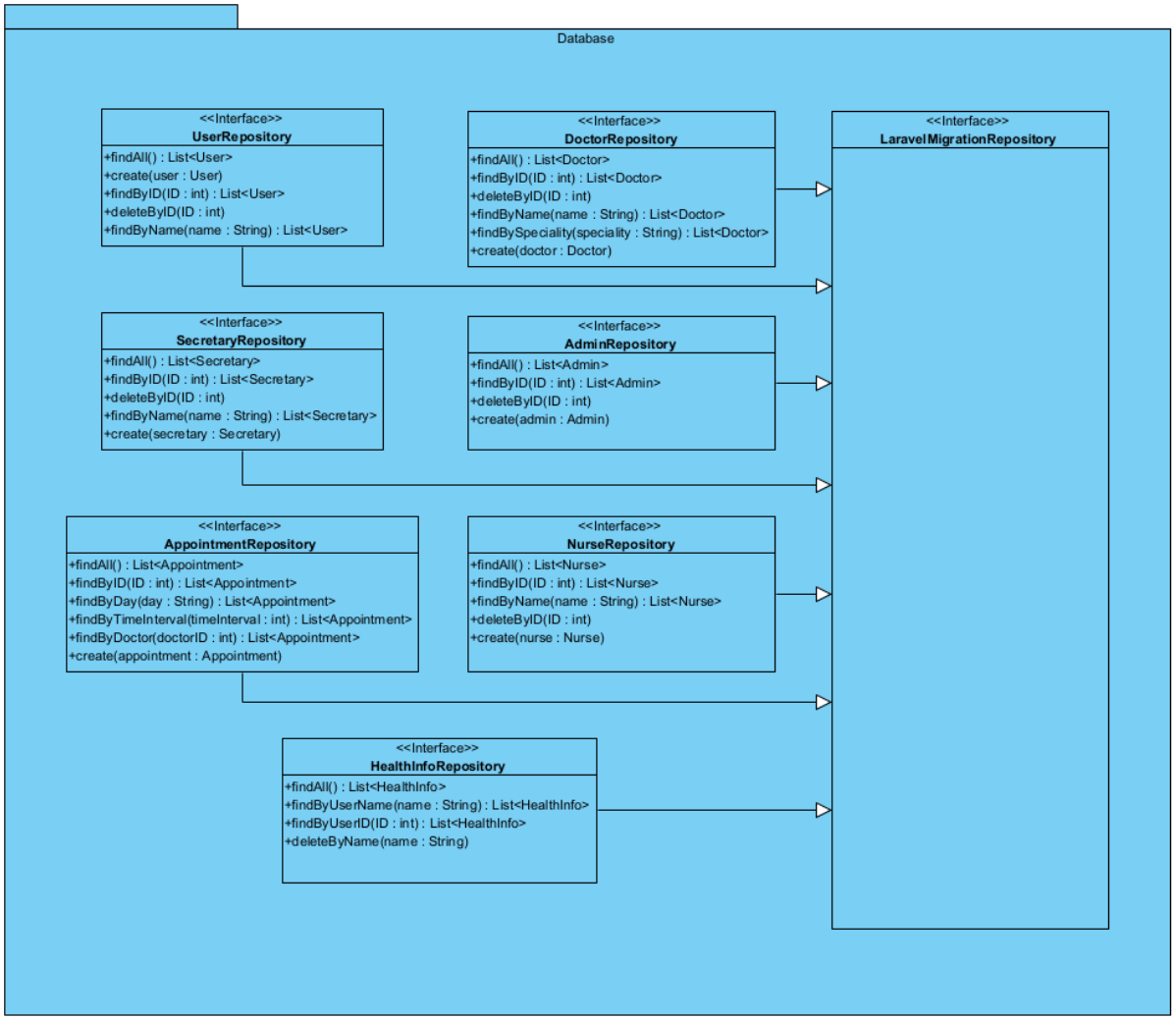
public viewUserDetails(): Gives all users' details.

3.5.2 Web Server Layer Class Interface



This layer is for the server's connection with the database. This layer allows the server to interact, change and see the data in the database. Thanks to this, user data can be kept in the database and viewed when necessary.

3.5.3 Database Layer



3.5.4 Database Layer Explanation

- **UserRepository Interface**

  This class is a Laravel repository used to implement various operations related to the User objects' storage in the database.

  function findAll(): This function returns a list of all the User objects available in the database. If there is no match, it will return an empty list.

  function create(user : User): This function saves a new User object to the database.

function findByID(ID : int): This function returns a list of all the User objects with the given ID attribute as an input. If there is no match, it will return an empty list.

function deleteByID(ID : int): This function returns a list of all the User objects with the given ID attribute as input and deletes the matching ones from the database.

function findByName(name : String): This function returns a list of all the User objects with the given name attribute as an input. If there is no match, it will return an empty list.

- **DoctorRepository Interface**

  This class is a Laravel repository used to implement various operations related to the Doctor objects' storage in the database.

  function findAll(): This function returns a list of all the Doctor objects available in the database. If there is no match, it will return an empty list.

  function findByID(ID : int): This function returns a list of all the Doctor objects with the given ID attribute as an input. If there is no match, it will return an empty list.

  function deleteByID(ID : int): This function returns a list of all the Doctor objects with the given ID attribute as input and deletes the matching ones from the database.

  function findByName(name : String): This function returns a list of all the Doctor objects with the given name attribute as an input. If there is no match, it will return an empty list.

  function findBySpeciality(speciality : String): This function returns a list of all the Doctor objects with the given speciality attribute as an input. If there is no match, it will return an empty list.

  function create(doctor : Doctor): This function saves a new Doctor object to the database.

- **SecretaryRepository Interface**

  This class is a Laravel repository used to implement various operations related to the Secretary objects' storage in the database.

function findAll(): This function returns a list of all the Secretary objects available in the database. If there is no match, it will return an empty list.
function findByID(ID : int): This function returns a list of all the Secretary objects with the given ID attribute as an input. If there is no match, it will return an empty list.
function deleteByID(ID : int): This function returns a list of all the Secretary objects with the given ID attribute as input and deletes the matching ones from the database.
function findByName(name : String): This function returns a list of all the Secretary objects with the given name attribute as an input. If there is no match, it will return an empty list.
function create(Secretary : Secretary): This function saves a new Secretary object to the database.

- **AdminRepository Interface**
  This class is a Laravel repository used to implement various operations related to the User objects' storage in the database.
  function findAll(): This function returns a list of all the Admin objects available in the database. If there is no match, it will return an empty list.
  function create(user : User): This function saves a new Admin object to the database.
  function findByID(ID : int): This function returns a list of all the Admin objects with the given ID attribute as an input. If there is no match, it will return an empty list.
  function deleteByID(ID : int): This function returns a list of all the Admin objects with the given ID attribute as input and deletes the matching ones from the database.

- **AppointmentRepository Interface**
  This class is a Laravel repository used to implement various operations related to the Appointment objects' storage in the database.
  function findAll(): This function returns a list of all the User objects available in the database. If there is no match, it will return an empty list.

function create(user : User): This function saves a new User object to the database.

function findByID(ID : int): This function returns a list of all the User objects with the given ID attribute as an input. If there is no match, it will return an empty list.

function findByDay(day : String): This function returns a list of all the Appointment objects with the given day attribute as an input. If there is no match, it will return an empty list.

function findByTimeInterval(timeInterval : String): This function returns a list of all the Appointment objects with the given timeInterval attribute as an input. If there is no match, it will return an empty list.

function findByDoctor(doctorID : int): This function returns a list of all the Appointment objects with the given doctorID attribute as an input. If there is no match, it will return an empty list.

- **NurseRepository Interface**

  This class is a Laravel repository used to implement various operations related to the Nurse objects' storage in the database.

  function findAll(): This function returns a list of all the Nurse objects available in the database. If there is no match, it will return an empty list.

  function findByID(ID : int): This function returns a list of all the Nurse objects with the given ID attribute as an input. If there is no match, it will return an empty list.

  function findByName(name : String): This function returns a list of all the Nurse objects with the given name attribute as an input. If there is no match, it will return an empty list.

  function deleteByID(ID : int): This function returns a list of all the Nurse objects with the given ID attribute as input and deletes the matching ones from the database.

  function create(nurse : Nurse): This function saves a new Nurse object to the database.

- **HealthInfoRepository Interface**

  This class is a Laravel repository used to implement various operations related to the HealthInfo objects' storage in the database.

  function findAll(): This function returns a list of all the HealthInfo objects available in the database. If there is no match, it will return an empty list.

  function findByUserID(ID : int): This function returns a list of all the HealthInfo objects with the given ID attribute as an input. If there is no match, it will return an empty list.

  function findByUserName(name : String): This function returns a list of all the HealthInfo objects with the given name attribute as an input. If there is no match, it will return an empty list.

  function deleteByName(name : String): This function returns a list of all the HealthInfo objects with the given name attribute as input and deletes the matching ones from the database.

4 Improvement Summary

-In the Subsystem Decomposition section, we have added a description for each subsystem.

-For the Hardware / Software Mapping part, we have added the requirements of the production server that will host our site.

-We corrected the structure of the access control matrix that we misunderstood and made the necessary additions.

-In the Boundary conditions section, we added the actions that the admin should take to start and stop the site in the Initialization and Termination sections.

-We added a design pattern part to the design report.

-We added web server layer and database layer diagrams and explanations of these diagrams to the Class interfaces section.

5 Glossary & references

[1]"Web application requirements," Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/power-platform/admin/web-application-requirements. [Accessed: Apr. 28, 2022].