2021–2022 SPRING SEMESTER

CS319 OBJECT-ORIENTED SOFTWARE ENGINEERING

TEAM: CHICKEN NUGGETS

DESIGN REPORT 1ST ITERATION

COURSE: CS319 **SECTION:** 1

DATE: 10th April 2022

INSTRUCTOR: Eray Tüzün

TEACHING ASSISTANTS: Muhammad Umair Ahmed, Elgun Jabrayilzade

TEAM MEMBERS:

Abdullah Riaz (22001296)

Ahmet Faruk Ulutaş (21803717)

Dilay Yiğit (21602059)

Mohammed Sohail (22001513)

Mostafa Khaled A. Mohamed Higazy (22001062)

1. INTRODUCTION

1.1. Purpose of the system

The project is a web application for Bilkent University's health center. The project also aims to solve the issues of slowness and complexity existing in the current system of the health center. The features provided by the project will include accessing patients' medical history, managing analysis and examination reports, writing and replying to messages, managing announcements, scheduling appointments, and an online payment gateway.

1.2. Design Goals

The design goals for our project will be discussed in this section. The design goals are based on the nonfunctional requirements from the requirement analysis report. The design goals will be addressed according to the aims of the project, which are to speed up the system and reduce the workload required by both patients and health workers.

1.2.1. Performance

Even if the user density is high, the system should not load longer than 3-4 seconds. The online appointment page should be updated every 2-3 seconds. If there is a change in the available appointment hours, the user should be able to see it within 2 seconds. Changes should display within seconds when users are using the messaging feature.

1.2.2. Usability

Announcements and blood donation requests should be under different labels to avoid confusion for users. Instead of a complex theme that is tiring on the eyes, a simple and white background should be created. Announcements or notifications with urgency should be colored red to make them stand out on a white background.

1.2.3. Reliability

All information about health workers displayed as a list must be correct. The confidentiality of the patients' personal information and medical history in the initial registration form should be protected. Another patient should not be able to access the registration forms. Authorizations must be done correctly. Complete access should be accessible by only the admin. If the website goes down, there should not be a data leak.

1.2.4. Supportability

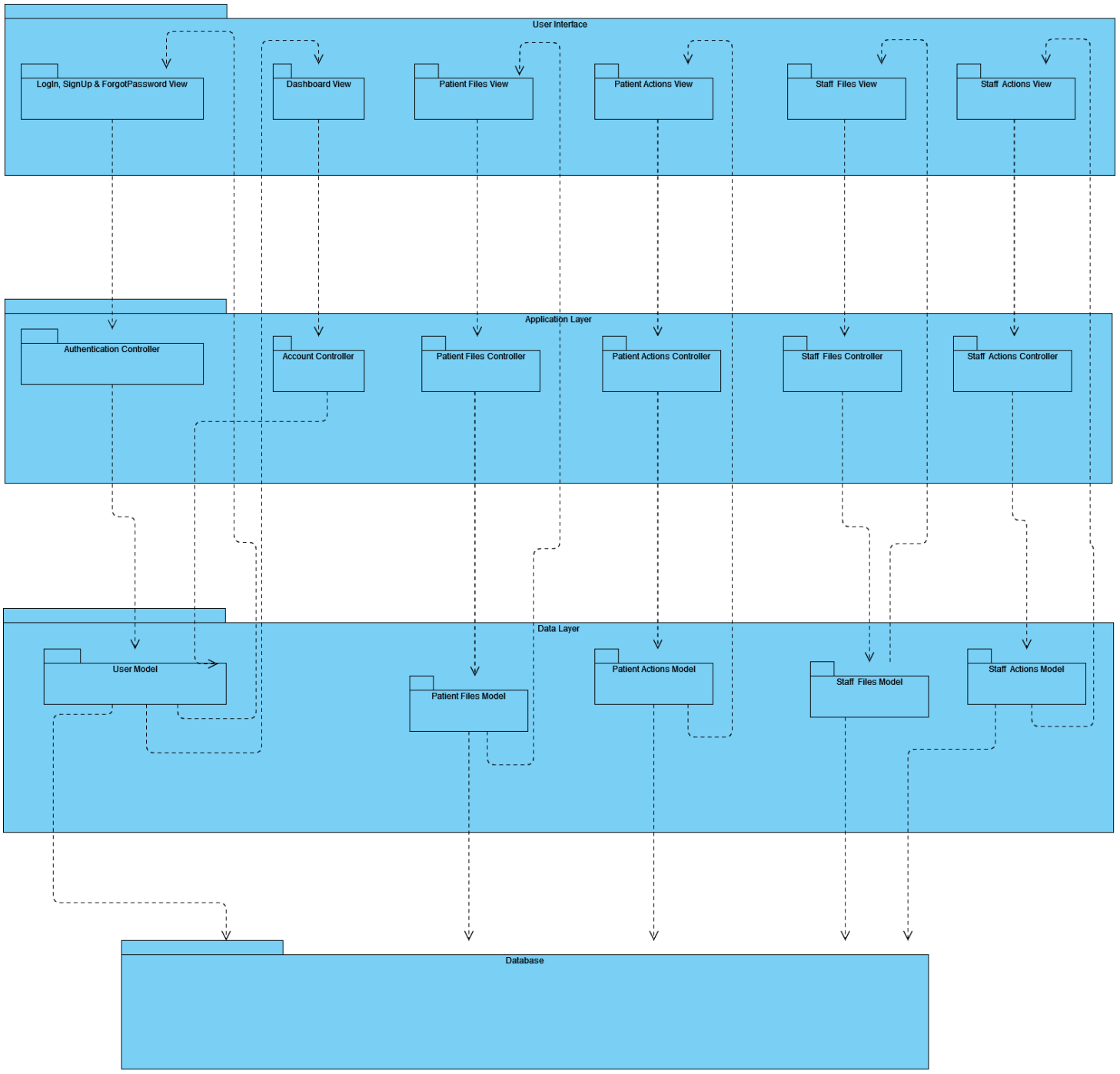
The website should be supported by all operating systems and all browsers. Responsive design will be ensured so that the website can be viewed without any visual problems, both on desktop and mobile interfaces.

1.3. Definitions, Acronyms, and Abbreviations

MVC: Model View Controller (Software design pattern)
SQL: Structured query language

2. SOFTWARE / HIGH-LEVEL ARCHITECTURE

2.1. Subsystem Decomposition



In this project, we used three layers that we divided as user interface, application layer, and data layer.

First, the user interface layer is the page that users interact with by viewing. For this reason, this layer has an easily understandable and easy-to-use interface. Every operation done on this page interacts with the application layer. If there is an entity in the data layer that needs to be updated, it is updated. We're looking at controlling authorizations when interacting from the user layer. Patients can only see their profile and doctors can only see their interface. Thus, we ensure the reliability of our users.

Secondly, there is the application layer, which is not in direct interaction with the user but connects the data layer and the user. This layer provides the necessary data and operations to be transferred from the data layer to the user interface. On the contrary, it enables the data to be saved by being processed and transferred to the data layer from the user interface.

Finally, the data layer is the layer where the data that users add or modify is saved, updated, and transferred back to the user.

2.2. Persistent Data Management

In our project, we are using MySQL. The reason we use MySQL is to meet our Performance target. Since MySQL has high scalability, no database-related drop will occur during the density. This is a good option for our limited number of users. In addition, since MySQL is easy to use, it will be easy to implement the updates needed in the future. In addition, we will keep our users' data in relational tables that we create in our database. Since our application works bidirectionally interactively, users will be able to dynamically process data using requests such as GET, POST, DELETE and PUT where they have authorization.

2.3. Access Control and Security

Since our project will include the personal information of users such as patients and doctors working in hospitals, privacy and the security of the application are very important to ensure this confidentiality.

First of all, unauthorized people should not be able to use the application in any way. To achieve this, when users log in, a cookie and the token-bound session are created, in which their authorization is provided. Until this session expires, the logged-in user can use the application with their rights. The whole application works in integration with authorizations. Data that is not brought to the application layer cannot be pulled from the database.

All information kept in the database will be fully encrypted with asymmetric encryption methods. Thus, in case of a possible leak, the information that can be obtained from the database will be irreversible hashes.

2.4. Boundary Conditions

In order to log in to our website, the user must first be created by the admins. If there is no specific user in the database, the user cannot log in to the site and receives an error warning on the login page.

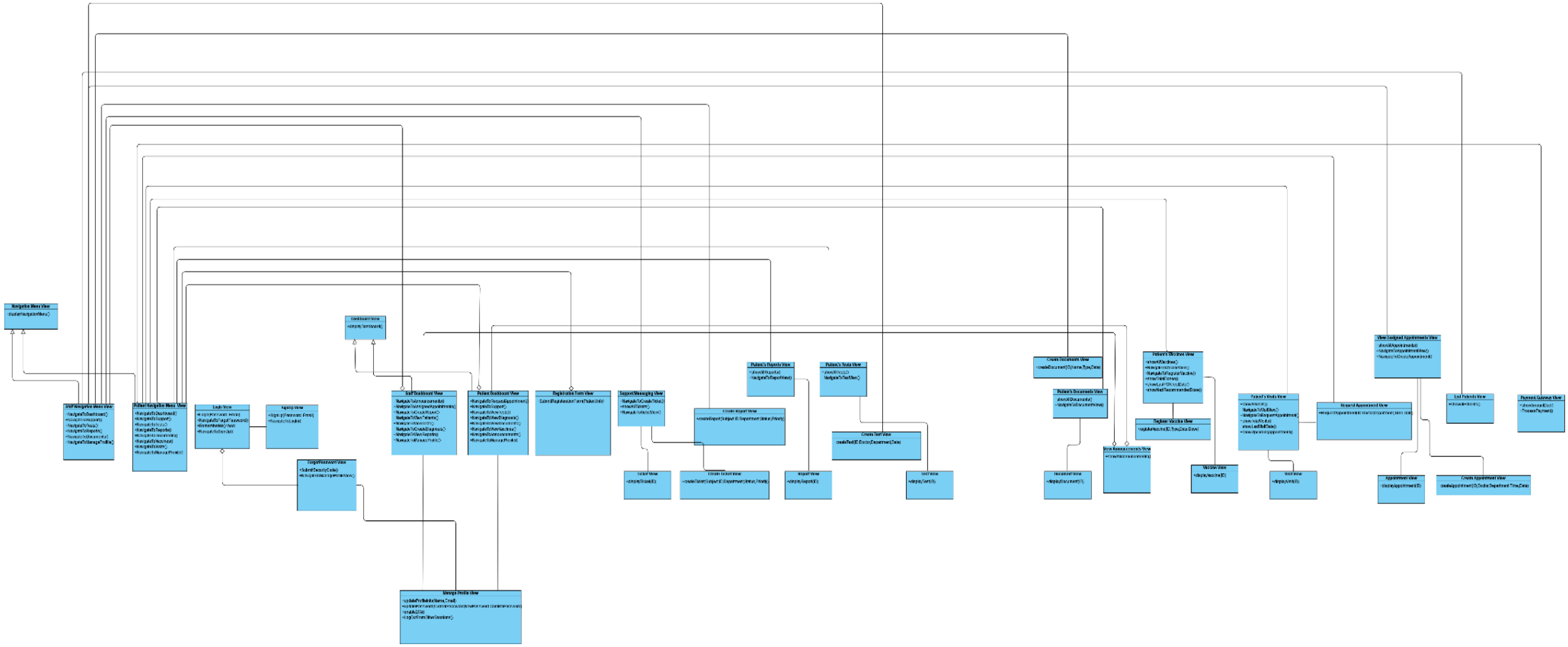
Internet connection should not be interrupted while users are making modifications to the data layer. Otherwise, the changes cannot be updated in the data layer.

It should be checked whether the necessary information is filled with valid entries in the form filled by the users in the patient role after logging in for the first time.

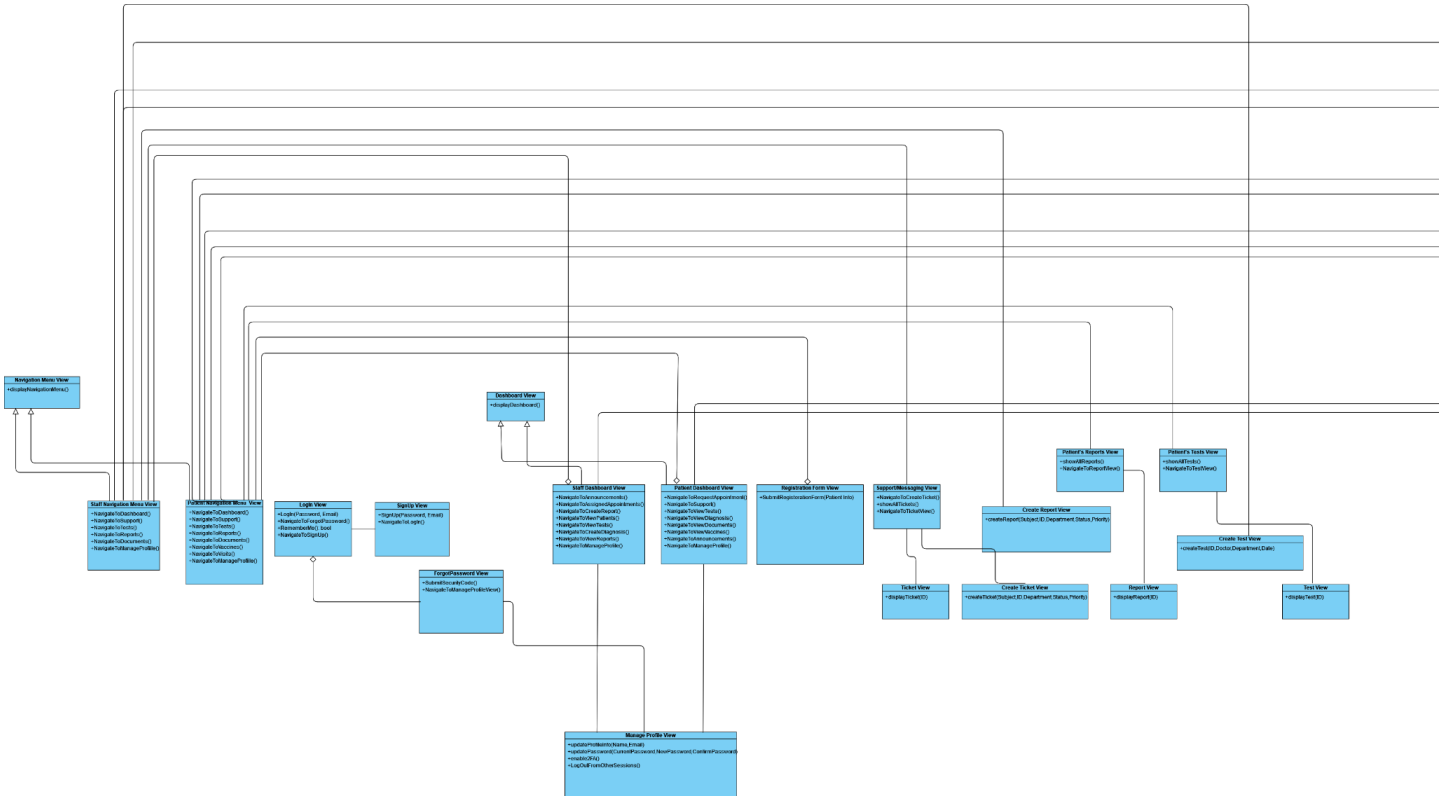
3. SUBSYSTEM SERVICES / LAYERS

3.1. UI Layer

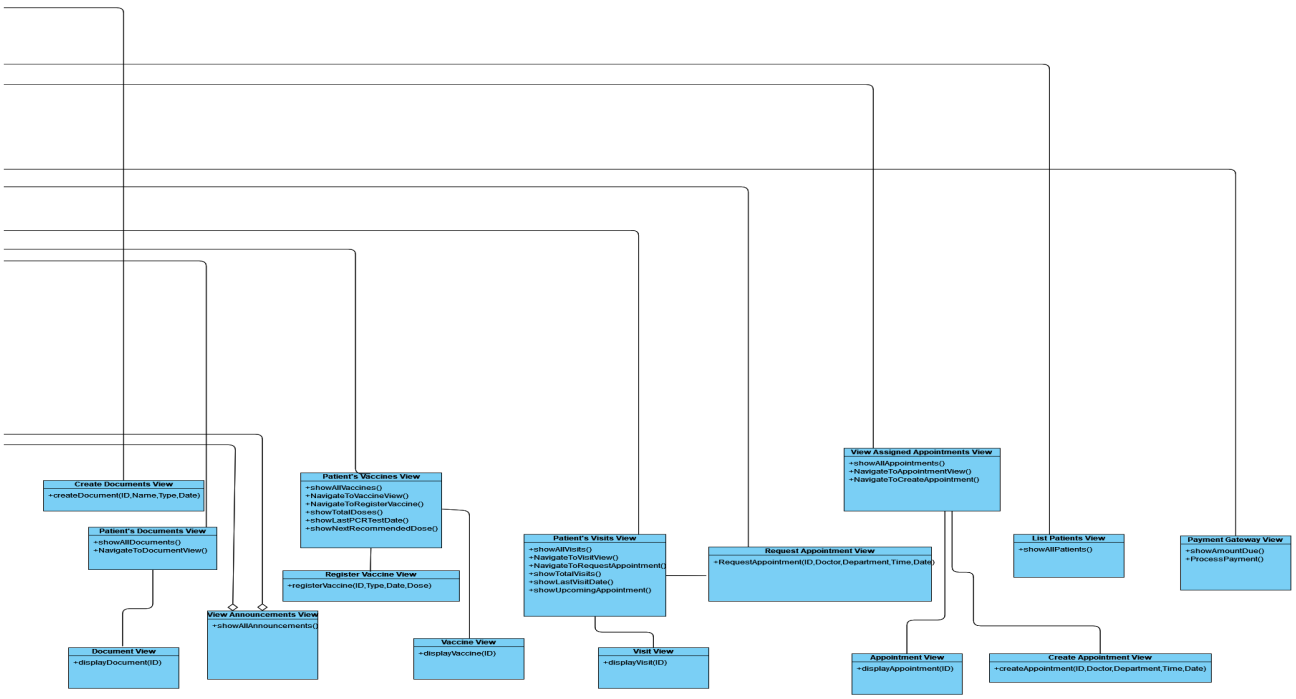
Full Diagram:



Left Side of Diagram:

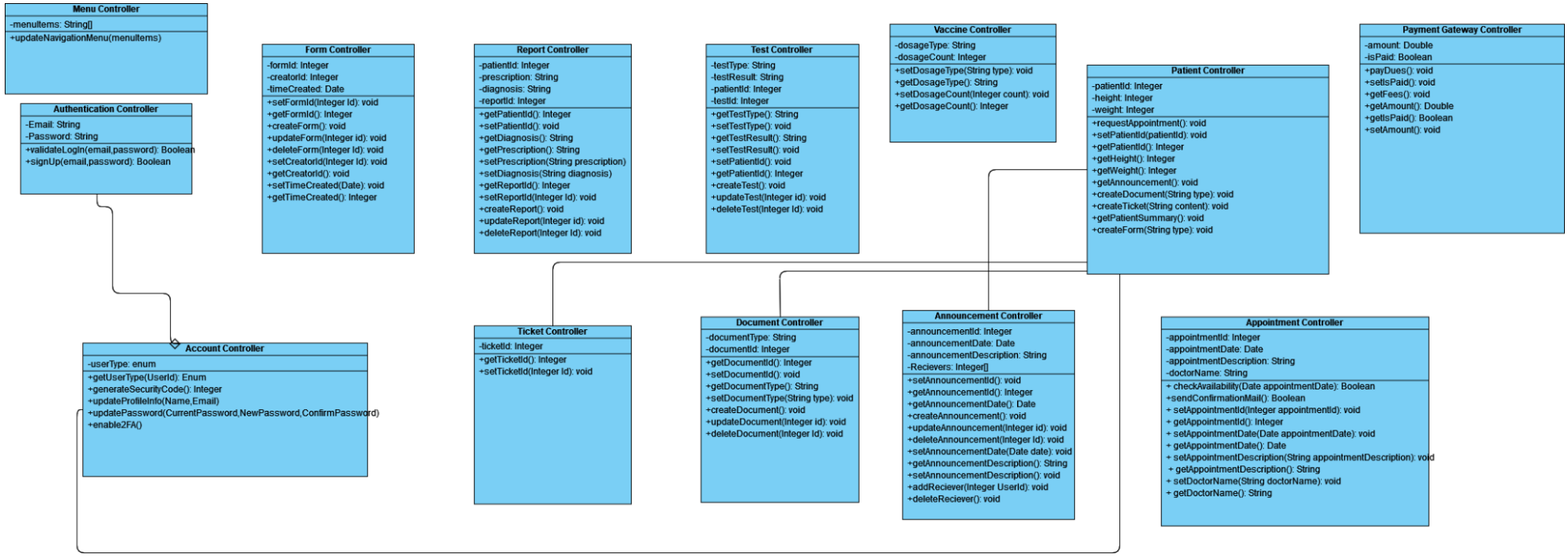


Right Side of Diagram:



The UI layer is a boundary object that is between the user and the system and facilitates the interaction between the user and system. All the classes in the layer represent webpages of the website

3.2. Application Layer:

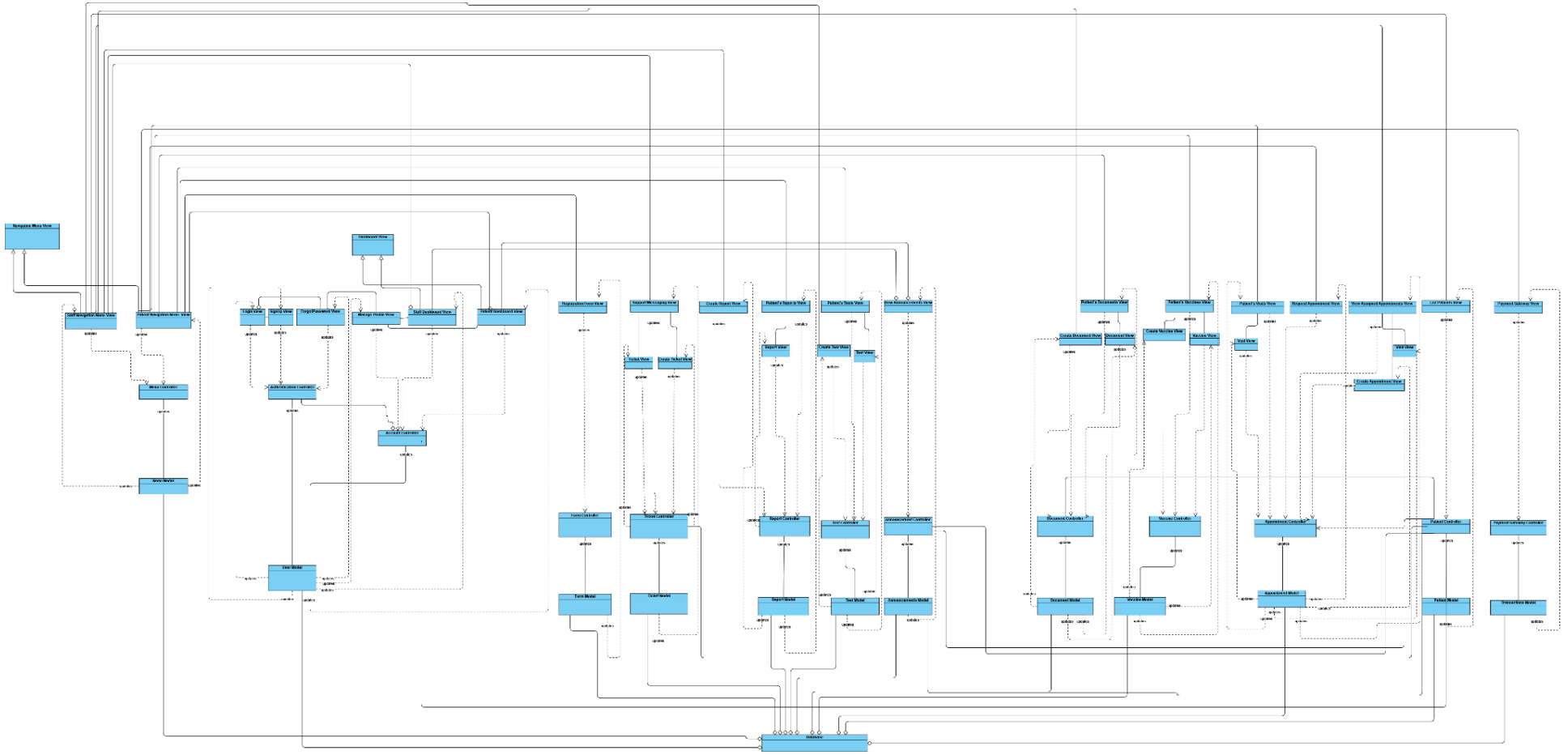


The Application layer is a control object and the classes inside the diagram are the control tasks that are performed by our website. It handles the user actions from the UI layer and passed them on to the Data layer.

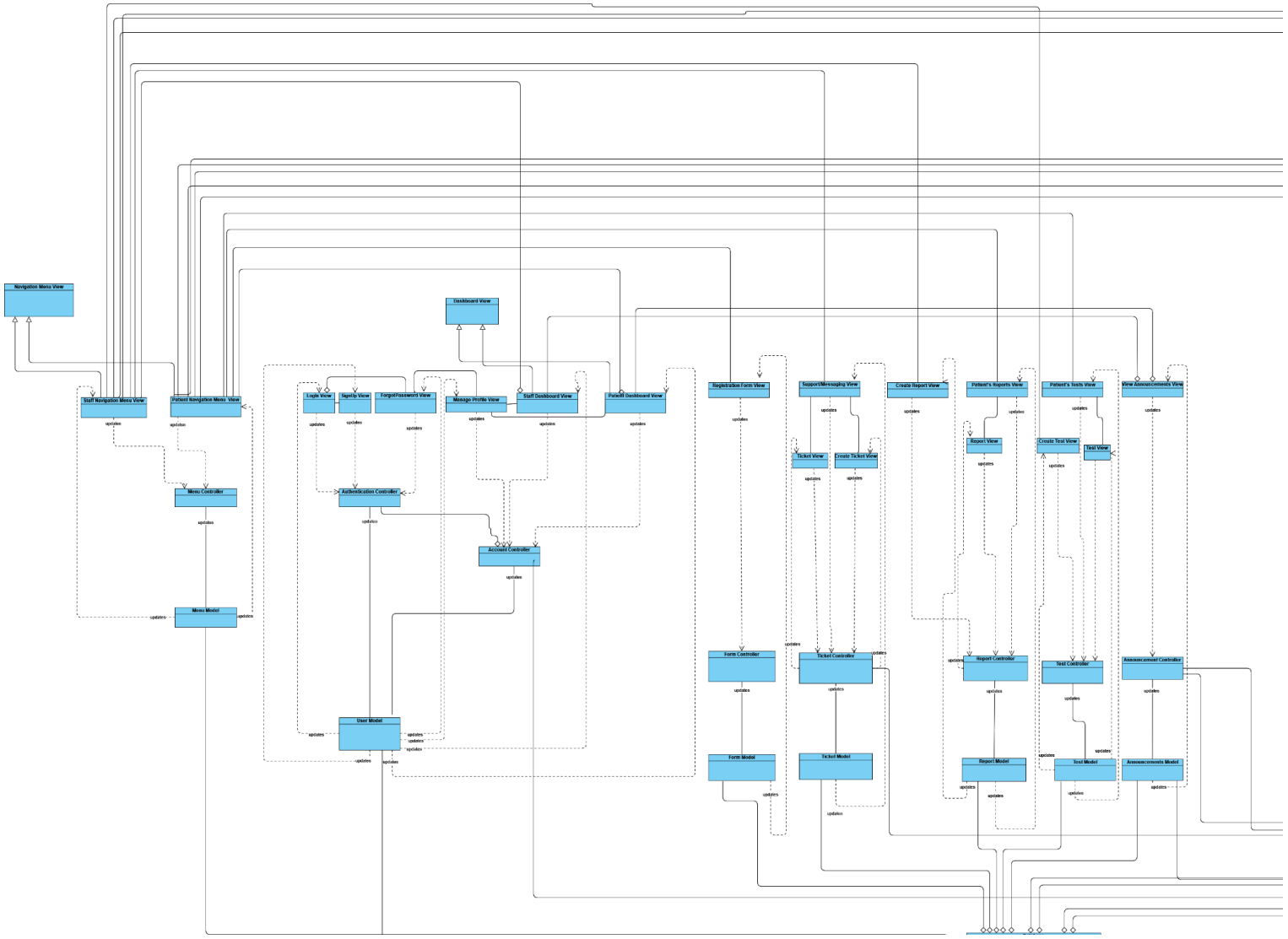
4. OBJECT DESIGN / LOW-LEVEL DESIGN

4.1. Object Design Diagram

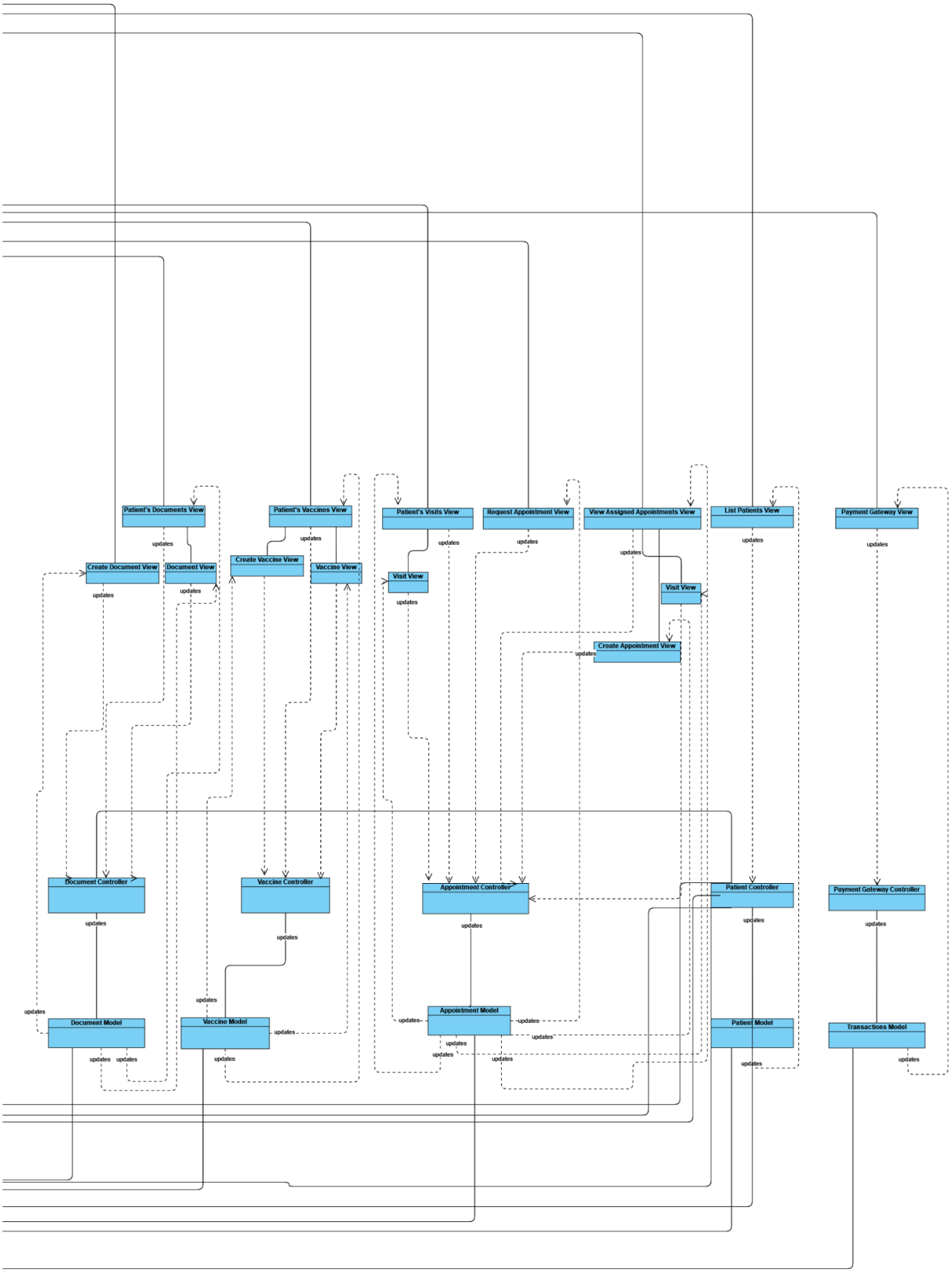
Full Diagram:



Left Side of the Diagram:



Right Side of the Diagram:



4.2. Object Design Trade-Offs

Performance versus Cost :
Our project will have a fast response time. Main services such as the appointment system and doctor availability hours have to get updated quite often to avoid overlap between bookings. This creates an extra resource cost for the project.

Functionality versus Maintainability:
The project has separate systems for staff and patients. Furthermore, both have their own unique and shared features, which means a lot of functionality is required. This leads to a more considerable effort needed for the maintainability of the project.

Backward Compatibility versus Maintainability:
Our project will be supported by all operating systems and all browsers. Since a lot of users are also on old browsers such as Internet Explorer, extra effort will be needed to maintain the website functionality on old browsers and mobile devices.

Portability versus Rapid Development:
The project is a website built using PHP. By making it in the form of a website, it can run on any desktop and mobile operating system. However, more development time will be required to make sure the web application works fine on various devices.

5. Packages

5.1. Internal Packages:

5.1.1. Authentication package:

This package contains all the classes related to authentication. This involves login validation, email verification, and two-factor authentication.

5.1.2. Announcement package:

This package involves all classes related to the management of announcements, notifications sent out by email, and the classes that handle the views of the announcement module.

5.1.3. Patient package:

This package involves all of a patient's sub-modules classes, such as patient’s visits, reports, vaccines, documents, tests, and diagnosis.

5.1.4. Staff package:

This package consists of the classes related to the staff role of the application, such as patient summary, availability of doctors, and managing appointments.

5.1.5. Form package:

This package consists of all the form classes that are used throughout the website.

5.1.6. Report package:

This package consists of all the report classes that include the reports generated by the website.

5.1.7. Payment package:

This package has all the classes that implement the functionality of the payment processing feature.

5.2. External Packages:

These packages are not developed or maintained by the team. They are obtained from external sources, and the application will have them as dependencies.

5.2.1. filament/forms:

This package is used to create forms from the model layer by providing the appropriate field names and field types.

5.2.2. filament/tables:

This package is used to create tables from the model layer by providing the appropriate field names and field types.

5.2.3. guzzlehttp/guzzle:

An HTTP client for managing HTTP requests. It is used to access APIs and handle their responses.

5.2.4. laravel/jetstream:

Jetstream manages user's browser sessions, password resets, base authentication (2FA, QR codes) for users

5.2.5. laravel/sanctum:

This package provides security essentials for route and API authentication

5.2.6. laravel/tinker:

This package allows us to run code separately from the controller layer for testing and enhancement purposes

5.2.7. livewire/livewire:

This package allows us to create dynamic interfaces which allow the views to connect to the controller without sophisticated API structures or AJAX calls

5.2.8. laravel/nova:

This package provides UI and resource management for the administration panel

5.2.9. spatie/laravel-permission

This package is used to restrict and allow permissions to the different user roles in our application

5.3. Development Packages:

These packages will be solely used by developers when developing/testing the functionality of the website. They will not be available in a production environment.

5.3.1. barryvdh/laravel-debugbar:

This package provides a debug bar at the bottom of the browser view when the environment of the application is set to development. It provides the developer with helpful information for debugging, such as SQL queries, memory usage, session data, connected models, and views of a specific page.

5.3.2. fakerphp/faker:

This library allows developers to generate fake/dummy data that can be sent into the model layer. This fake data is used for testing and enhancement of the model and controller layer.

5.3.3. phpunit/phpunit:

This package is used to write unit tests for testing the functionality of the website. These tests can then be run in a CI/CD pipeline to ensure all tests pass before updates are deployed on production.

6. Class Interfaces

6.1. Controller Classes:

6.1.1. AuthenticationController

Attributes:

- string email: Stores the email of the user attempting to log in
- string password: Stores the password of the user attempting to log in

Methods:

+ validateLogin(): boolean: Validates the login credentials against the database and returns true if the credentials are valid.
+ signUp(email, password): boolean: It receives email and password information and verifies whether there is a user in the database of this information.

6.1.2. AccountController

Attributes:

- enum userType: Indicates the type of user (patient, staff, doctor, etc.)

Methods:

+ getUserType(UserId): Enum: This method returns the user type of given userId.
+ int generateSecurityCode(): This method generates security code for account.
+ updateProfileInfo(Name, Email): This method updates profile info.
+ updatePassword (CurrentPassword, NewPassword, ConfirmPassword): If the user has entered his/her current password correctly, if the new password and the confirmation password match, he/she will update his/her password with the new password.
+ enable2FA(): Enables two factor authentication for the user.

6.1.3. MenuController

Attributes:

-menuItems: String[]: Stores the menu items

Methods:

+ updateNavigationMenu(Menu menuItems): Updates the navigation menu according to the menu items.

6.1.4. AppointmentController

Attributes:

- integer appointmentId: Id numbers that assign to appointments
- date appointmentDate: The date the appointment will take place
- string appointmentDescription: Description of what the appointment is about
- string doctorName: Name of the doctor assigned to the appointment

Methods:
+ **checkAvailability(date appointmentDate): Boolean:** It checks whether there is a suitable appointment on the entered date.
+ **sendConfirmationMail(): Boolean:** Checks whether a confirmation email is sent to the user while making an appointment.
+ **setAppointmentId(Integer appointmentId)::** Modifies the id number assigned to the appointments.
+ **getAppointmentId(): Integer:** Returns the id number assigned to the appointments.
+ **setAppointmentDate(date appointmentDate)::**Modifies the appointment date.
+ **getAppointmentDate(): Date:** Return the appointment date.
+ **setAppointmentDescription(String appointmentDescription)::** Modifies the appointment description.
+ **getAppointmentDescription(): String:**Returns the appointment description.
+ **setDoctorName(String doctorName)::** Modifies the doctor's name for a specific appointment.
+ **getDoctorName(): String:** Returns the doctor's name in a specific appointment.

6.1.5. PatientController

Attributes:

- int patientId: Id numbers that are assigned to the patients.
- int height: Patient's heights
- int weight: Patient's weight

Methods:

+ requestAppointment(): It allows patients to create an appointment request.
+ createForm(String type): It creates the medical history forms assigned to the patients themselves.
+ getPatientSummary(): It provides the summary of patient information to be displayed.
+ createTicket(String content): It allows the creation of tickets so that patients can ask questions to doctors or make announcements.
+ createDocument(String type): It creates documents for patients.
+ setPatientId(): Modifies patient id.
+ int getPatientId(): Integer: Returns the patient id.
+ getHeight(): Integer: Returns the patient's height.
+ getWeight(): Integer: Returns the patient's weight.
+ getAnnouncements(): It allows patients to retrieve the announcements.

6.1.6. PatientTestController

Attributes:

- int patientId: Id numbers that are assigned to the patients.
- String testType: Patient's test type.
- String testResult: Patient's test result.
- Integer testId: Id number of test.

Methods:

+ String getTestType(): String: Returns patient's test result.
+ setTestType(): Set patient's test type.
+ String getTestResult():String: Get patient's test result.
+ setTestResult(): Set patient's test result.
+ int getPatientId(): Integer: Returns the patient id.
+ setPatientId(): Modifies patient id.
+ createTest(): Creates a test
+ updateTest(Integer id): updates test contents

6.1.7. PatientReportController

Attributes:

- int patientId: Id numbers that are assigned to the patients.
-String prescription: prescription of medicine to the patient.
-String diagnosis: diagnosis of the patient
-int reportId: Id numbers that are assigned to reports

Methods:

+ getPatiendId(): Integer: Returns the patient id.
+ setPatientId(): Modifies the patient id.
+getDiagnosis(): String: Returns patient's diagnosis.
+getPrescription(): String: Returns patient's prescriptions.
+setPrescription(String prescription): Sets patient's prescription.
+setDiagnosis(String diagnosis): Sets patient's diagnosis.
+getReportId(): Integer: Returns id of report.
+setReportId(Integer Id): Setes id of report.
+createReport(): Creates report.
+updateReport(Integer id): Updates Report.
+deleteReport(Integer Id): Deletes Report.

6.1.8. PatientDocumentController

Attributes:

- int documentId: Id numbers that are assigned to the patient's documents.
-documentType: String: The type of the document(pdf,Image, etc.)

Methods:

+ int getDocumentId(): Returns the document id.
+ setDocumentId(): Modifies the document id.
+ getDocumentType(): String: Returns type of the document
+ setDocumentType(String type): Sets the type of the document
+ createDocument(): Creates Document
+ updateDocument(Integer id): Updates Document
+ deleteDocument(Integer Id): Deletes Document

6.1.9. PatientVaccineController

Attributes:

- String dosageType: Type of vaccine that the patient has done.
- int dosageCount: Number of vaccine dosages that patient has done.

Methods:

+ setDosageType(): Modifies the dosage type.
+ getDosageType(): Returns the dosage type.
+ setDoseCount(): Modifies the dose count.
+ getDoseCount(): Returns the dose count.

6.1.10. TicketController

Attributes:

- int ticketId: Id numbers that are assigned to the tickets.

Methods:

+ getTicketId(): Returns the ticket id number.
+ setTicketId(): Modifies the ticket id number.

6.1.11. PaymentGatewayController

Attributes:

- double amount: Patient's payment amount
- boolean isPaid: Store condition of whether patient's payment is done

Methods:

+ payDues(): Pay the patient's due
+ setIsPaid(): Set patient's paid condition
+ displayFees(): Display patient's fees
+ setAmount(): Set patient's amount
+ double getAmount(): Get patient's amount
+ boolean getIsPaid(): Get patient's paid condition

6.2. Model Classes:

The model classes extend a base model class provided by the Laravel framework that includes methods to create, find, update and delete entries inside each model. The attributes for each class are listed below. For simplicity, the methods are not listed here and a list of them is available over here: https://laravel.com/docs/9.x/eloquent

6.2.1. NavigationModel

- **int id:** The id of the navigation item
- **string label:** The label that appears on the user-end
- **string slug:** The slug which is used to route to the page

6.2.2. AppointmentModel

- **int id:** The id of the appointment
- **int patient_id:** The patient's id
- **int doctor_id:** The doctor's id
- **date appointment_date:** The date of the appointment
- **date created_at:** The creation date of the appointment
- **date modified_at:** The modification date of the appointment
- **enum status:** The status of the appointment (new, canceled, past)

6.2.3. AnnouncementModel

- **int id:** The id of the announcement
- **string title:** The title of the announcement
- **string content:** The text content of the announcement
- **boolean private:** If true, the announcement will be hidden from users
- **date created_at:** The timestamp of the announcement creation
- **date modified_at:** The timestamp of the announcement modification

6.2.4. PatientTestModel

- **int id:** The id of the test
- **int patient_id:** The id of the patient linked to the test
- **string description:** The short text description of the test
- **string attachment_path:** The relative path to the attachment
- **date created_at:** The timestamp of the test creation
- **date modified_at:** The timestamp of the test modification

6.2.5. HealthReportModel

- **int id:** The id of the health report
- **int patient_id:** The id of the patient linked to the health report
- **int staff_id:** The id of the staff member who created the health report
- **int duration:** The duration that the health report covers
- **int start_date:** The start date of the health report
- **string reason:** The reason for the health report
- **string attachment_path:** The relative path to the attachment
- **date created_at:** The timestamp of the report creation
- **date modified_at:** The timestamp of the report modification

6.2.6. PatientVaccineModel

- **int id:** The id of the vaccine model
- **int patient_id:** The id of the patient linked to the vaccine
- **string attachment_path:** The relative path to the attachment
- **string type:** The type of vaccine
- **date created_at:** The timestamp of the vaccine model creation
- **date modified_at:** The timestamp of the vaccine model modification

6.2.7. PatientDiagnosisModel

- **int id:** The id of the diagnosis
- **int patient_id:** The id of the patient linked to the diagnosis
- **int staff_id:** The id of the staff member who created the diagnosis
- **string title:** The title of the diagnosis
- **string description:** The text description of the diagnosis
- **string attachment_path:** The relative path to the attachment
- **date created_at:** The timestamp of the diagnosis creation
- **date modified_at:** The timestamp of the diagnosis modification

6.2.8. PatientVisitModel

- **int id:** The id of the visit
- **int patient_id:** The id of the patient linked to the visit
- **date visit_date:** The timestamp of the visit
- **date created_at:** The timestamp of the visit creation
- **date modified_at:** The timestamp of the visit modification

6.2.9. PatientDocumentModel

- **int id:** The id of the patient's documents
- **int patient_id:** The id of the patient
- **string description:** The descriptions in the patient's documents
- **string attachment_path:** The relative path to the attachment
- **date created_at:** The timestamp of the document creation
- **date modified_at:** The timestamp of the document modification

6.2.10. TicketModel

- **int id:** The id of the ticket
- **int patient_id:** The id of the patient linked to the ticket
- **string subject:** The subject of the ticket
- **enum status:** The status of the ticket (new, pending, answered, closed, on-hold)
- **date created_at:** The timestamp of the ticket creation
- **date modified_at:** The timestamp of the ticket modification

6.2.11. MessageModel

- **int id:** The id of the message
- **int ticket_id:** The id of the ticket to which the message is linked to
- **int user_id:** The id of the user who created the message
- **string message:** The text content of the message
- **date created_at:** The timestamp of the message creation
- **date modified_at:** The timestamp of the message modification

6.2.12. UserModel

- **int id:** The id of the user
- **string email:** The email address that the user will use to log in
- **string password:** Password that the user will use to log in
- **string name:** Name of the user
- **enum role:** Indicates which user role the user is in (staff, patient, doctor… etc.)
- **boolean active:** If false, the user is not allowed to access the application

6.2.13. InvoiceModel

- **int id:** The id of the invoice
- **string description:** The description of the invoice
- **double amount:** The amount which is due against the invoice
- **int patient_id:** The id of the patient linked with the invoice
- **enum status:** The status of the invoice (unpaid, paid, overdue, canceled, refunded)
- **date paid_at:** The timestamp at which the invoice was paid
- **date due_at:** The timestamp at which the invoice is due
- **date created_at:** The timestamp of the invoice creation
- **date updated_at:** The timestamp of the invoice modification

7. Glossary & References

https://filamentphp.com/docs/2.x/forms/

https://github.com/laravel-filament/forms/

https://laracasts.com/series/laravel-8-from-scratch

https://laravel-livewire.com

https://laravel.com/docs/9.x

https://filamentphp.com/docs/2.x/tables/

https://docs.guzzlephp.org/en/stable/

https://jetstream.laravel.com/2.x/introduction.html

https://github.com/laravel/sanctum

https://laravel.com/docs/9.x/artisan

https://nova.laravel.com/

https://spatie.be/docs/laravel-permission/v5/introduction

https://github.com/barryvdh/laravel-debugbar

https://fakerphp.github.io/

https://phpunit.de/