2021–2022 SPRING SEMESTER

CS319 OBJECT-ORIENTED SOFTWARE ENGINEERING

TEAM: CHICKEN NUGGETS

DESIGN REPORT FINAL ITERATION

COURSE: CS319 **SECTION:** 1

DATE: 2nd May 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

By using compression and optimizing CSS plugins on our site, the loading time of the system will not exceed 3-4 seconds even when the user density is high. With the script to be written using Laravel, the online appointment page will be updated every 2-3 seconds. In this way, if there is a change in the current appointment times, the user will be able to see it within 2 seconds. With the same logic, changes will be displayed in seconds while users are using the messaging feature. Best practices will be used to ensure that website load time is kept to a minimum and there will be no unnecessary plugins on the service to avoid long load times. To get a faster flow from the database, unnecessary queries will not be sent and SQL queries will be sent.

1.2.2. Usability

Announcements and blood donation requests will 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 will be created. Announcements or notifications with urgency will be colored red to make them stand out on a white background. All the design decisions will be taken with accessibility in mind, and the color scheme will be applied appropriately.

1.2.3. Reliability

All information about healthcare workers displayed as a list must be correct. The confidentiality of the personal information and medical history of the patients in the first registration form will be protected by encrypting all data. By providing authorization between users, access to the registration forms of other patients will be prevented. Only the administrator can access full access. If the website goes down, the session owned by the user will be terminated, preventing data leakage. Applied authorization methods will ensure that the user cannot access any link that does not exist in his account.

1.2.4. Supportability

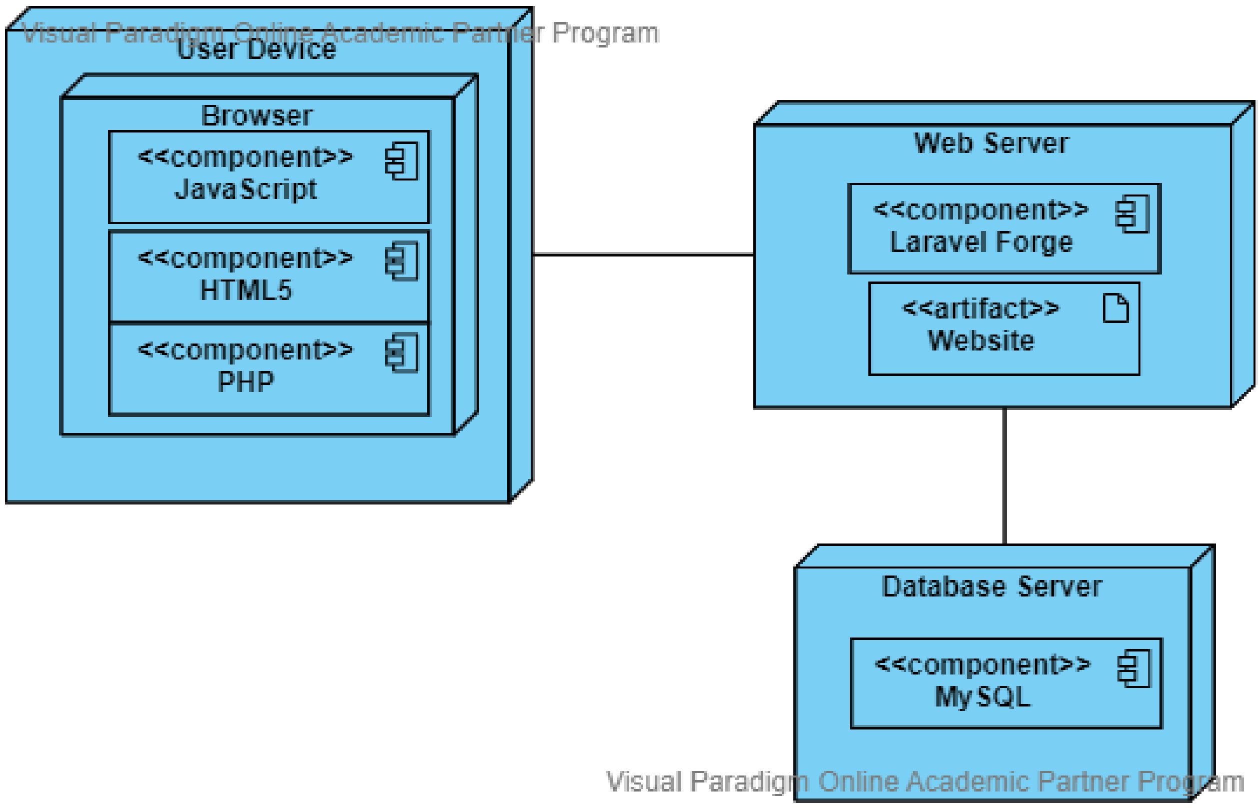
The website will be supported by all browsers. Responsive design will be provided so that the website can be displayed visually on both desktop and mobile interfaces. Design choices will be made with responsive design and accessibility in mind.

1.3. Definitions, Acronyms, and Abbreviations

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

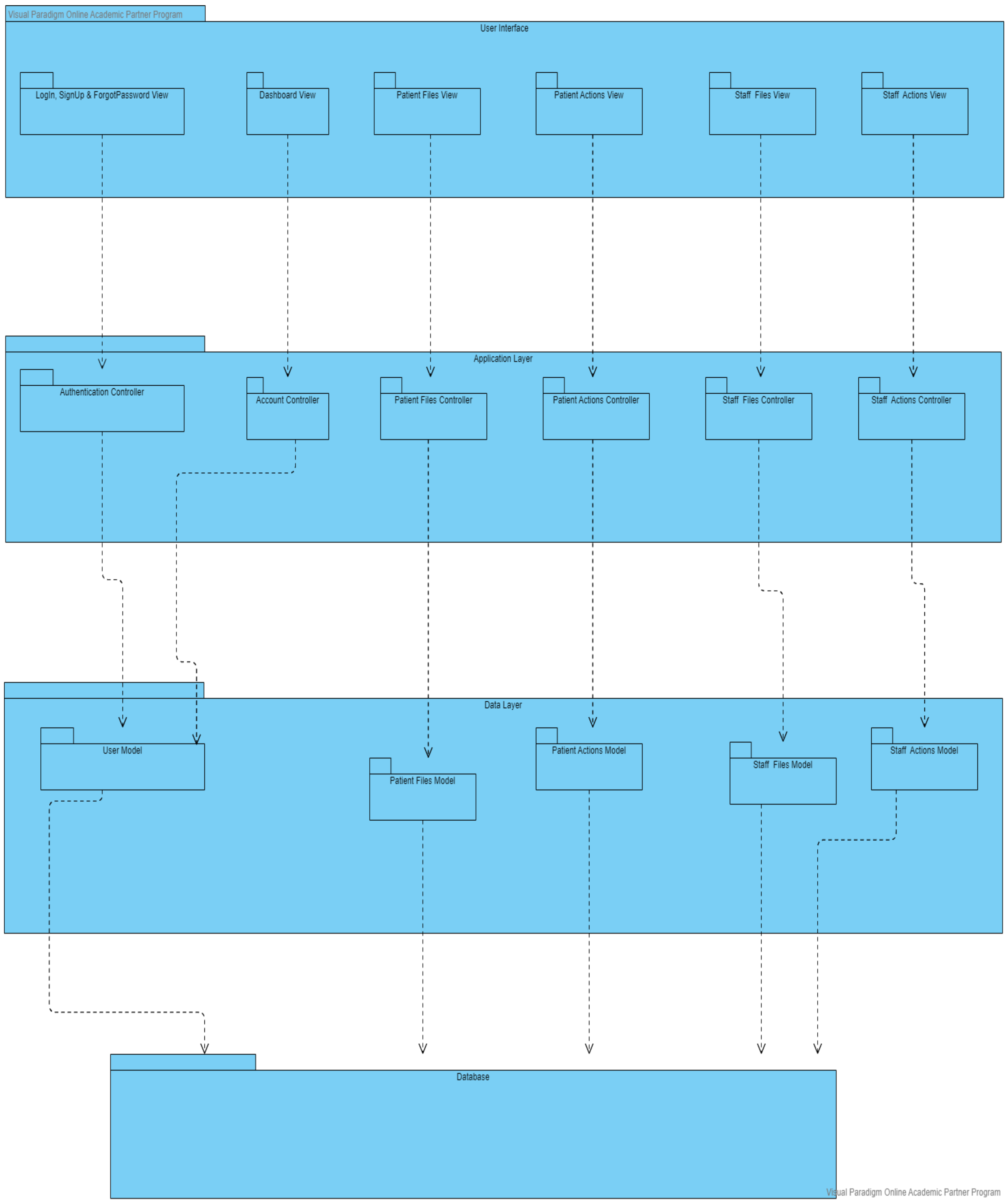
2. SOFTWARE / HIGH-LEVEL ARCHITECTURE

2.1. Deployment Diagram



The website will be hosted on a web server, and the database will be hosted on a database server. The web server can access data from the database server. The user can access the website through any browser on any device. Laravel Forge will be used to publish the website and store the DNS configuration and configure the domain name. The browser must support PHP, HTML5, and JavaScript.

2.2. Subsystem Decomposition



Brief descriptions of each layer can be found under the Subsystem services/Layers heading.

Three layers were used in this project, which are divided into 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, the reliability of the users is ensured.

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.

2.3. Hardware/Software Mapping

This project is a web application. Anyone with a stable internet connection can access the website on any kind of device that supports a modern web browser(Firefox, Chrome, Safari, etc.). A web browser that supports PHP, HTML5, and JavaScript is required. Legacy browsers will not be supported.

2.4. Persistent Data Management

MySQL will be used in the project. The reason for using MySQL is to reach the performance target of the project. Since MySQL is highly scalable, there will be no database-related dropouts during density. This is a good option for a limited number of users. Also, since MySQL is easy to use, it will be easy to apply needed updates in the future. In addition, users' data will be kept in relational tables created in the database. Since the application works bidirectionally interactively, users will be able to dynamically manipulate data using requests such as GET, POST, DELETE and PUT where they are authorized. All user accounts and user data (including documents, forms, tests, reports, etc.) will be stored in the database along with a token and session ID for authorization.

2.5. 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 will 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.

Below is an access control matrix indicating the actions available to each user type:

	Account	Report	Test	Announcement	Diagnosis	Document	Vaccine	Appointment	Visit	PaymentGateway

Patient	viewAccount()
	logIn()
	updateAccount()
		viewReport()	viewTest()	viewAnnouncement()	viewDiagnosis()	viewDocument()
						createDocument()
							viewVaccine()	viewAppointment()
								viewUpcomingAppointments()
								requestAppointment()
									viewPreviousVisits()	viewInvoices()
										payDues()

Receptionist	viewAccount()
	logIn()
	updateAccount()
	ViewAllPatients()
								confirmAppointment()
								viewAllAppointments()
									viewPreviousVisits()	

Nurse	viewAccount()
	logIn()
	updateAccount()
	ViewAllPatients()
		viewReport()
		createReport()
		updateReport()
		deleteReport()
			viewTest()
			createTest()
			updateTest()
			deleteTest()
				viewAnnouncement()	viewDiagnosis()
					createDiagnosis()
					updateDiagnosis()
					deleteDiagnosis()
						viewDocument()
						createDocument()
						updateDocument()
						deleteDocument()
							viewVaccine()
							createVaccine()
							updateVaccine()
							deleteVaccine()
								viewAppointment()
								viewAllAppointments()
									viewPreviousVisits()	

Doctor	viewAccount()
	logIn()
	updateAccount()
	ViewAllPatients()
		viewReport()
		createReport()
		updateReport()
		deleteReport()
			viewTest()
			createTest()
			updateTest()
			deleteTest()
				viewAnnouncement()	viewDiagnosis()
					createDiagnosis()
					updateDiagnosis()
					deleteDiagnosis()
						viewDocument()
						createDocument()
						updateDocument()
						deleteDocument()
							viewVaccine()
							createVaccine()
							updateVaccine()
							deleteVaccine()
								viewAppointment()
								viewUpcomingAppointments()
								viewPreviousVisits()
									viewPreviousVisits()	

Admin	viewAccount()
	logIn()
	createAccount()
	deleteAccount()
	viewAllAccounts()
		viewReport()
		createReport()
		updateReport()
		deleteReport()
			viewTest()
			createTest()
			updateTest()
			deleteTest()
				viewAnnouncement()
				createAnnouncement()
				updateAnnouncement()
				deleteAnnouncement()
					viewDiagnosis()
					createDiagnosis()
					updateDiagnosis()
					deleteDiagnosis()
						viewDocument()
						createDocument()
						updateDocument()
						deleteDocument()
							viewVaccine()
							createVaccine()
							updateVaccine()
							deleteVaccine()
								updateAppointment()
								deleteAppointment()
									viewPreviousVisits()	createInvoice()
										viewInvoice()
										updateInvoice()
										deleteInvoice()
										viewInvoices()

2.6. Boundary Conditions

2.6.1. Initialization

Anyone with a user account can access the website by its domain name. 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.

2.6.2. Termination

Internet connection will not be interrupted while users are making modifications to the data layer. Otherwise, the changes cannot be updated in the data layer. In case an admin decides to terminate the app, he can server the connection between the database layer, and the controller layer rendering the website inaccessible. The data will persist in the database, and a new connection can be made, in case the decision is reversed.

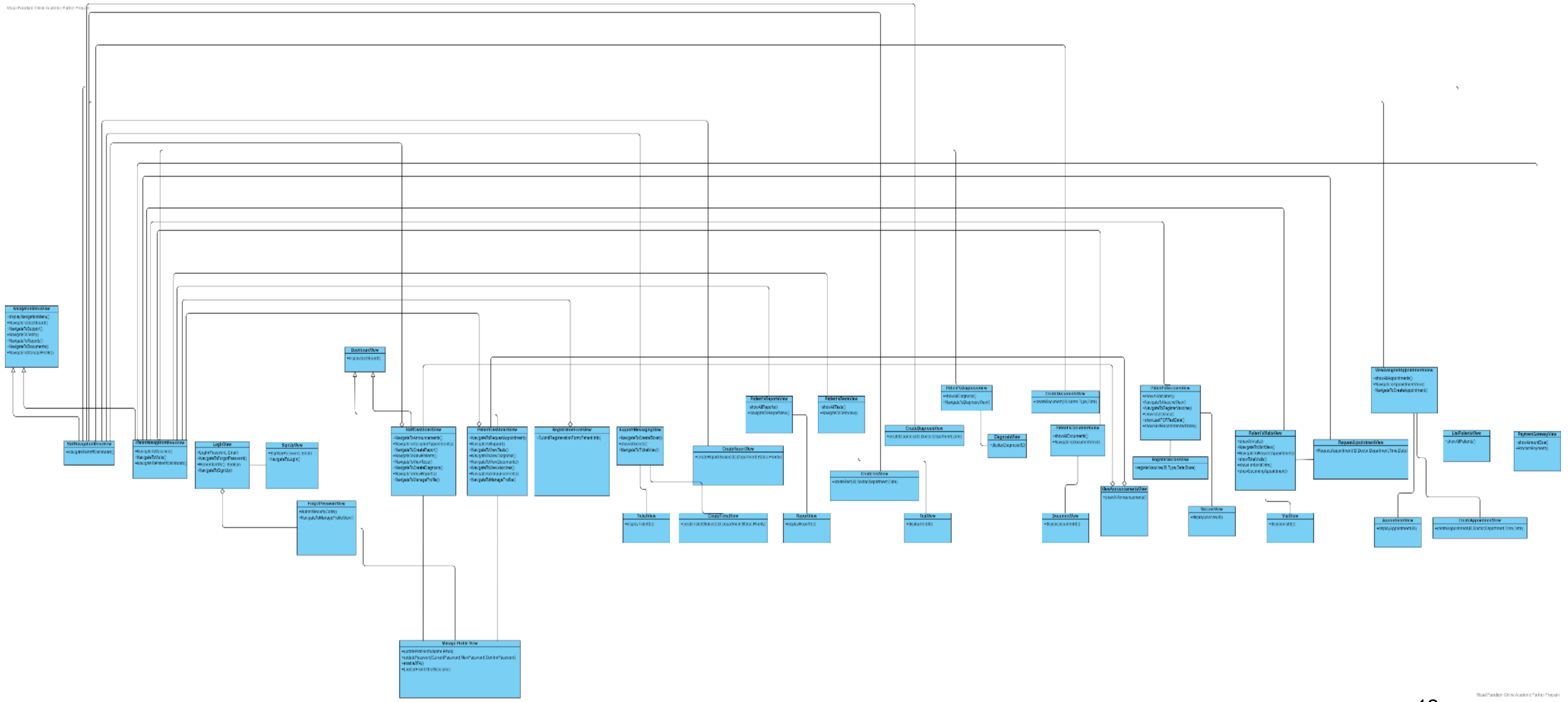
2.6.3. Failure

It will 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. In case of a server failure, the whole system is rendered inaccessible, and a notification is sent to all the developer’s emails. The website will also return the HTTP response status code according to the error.

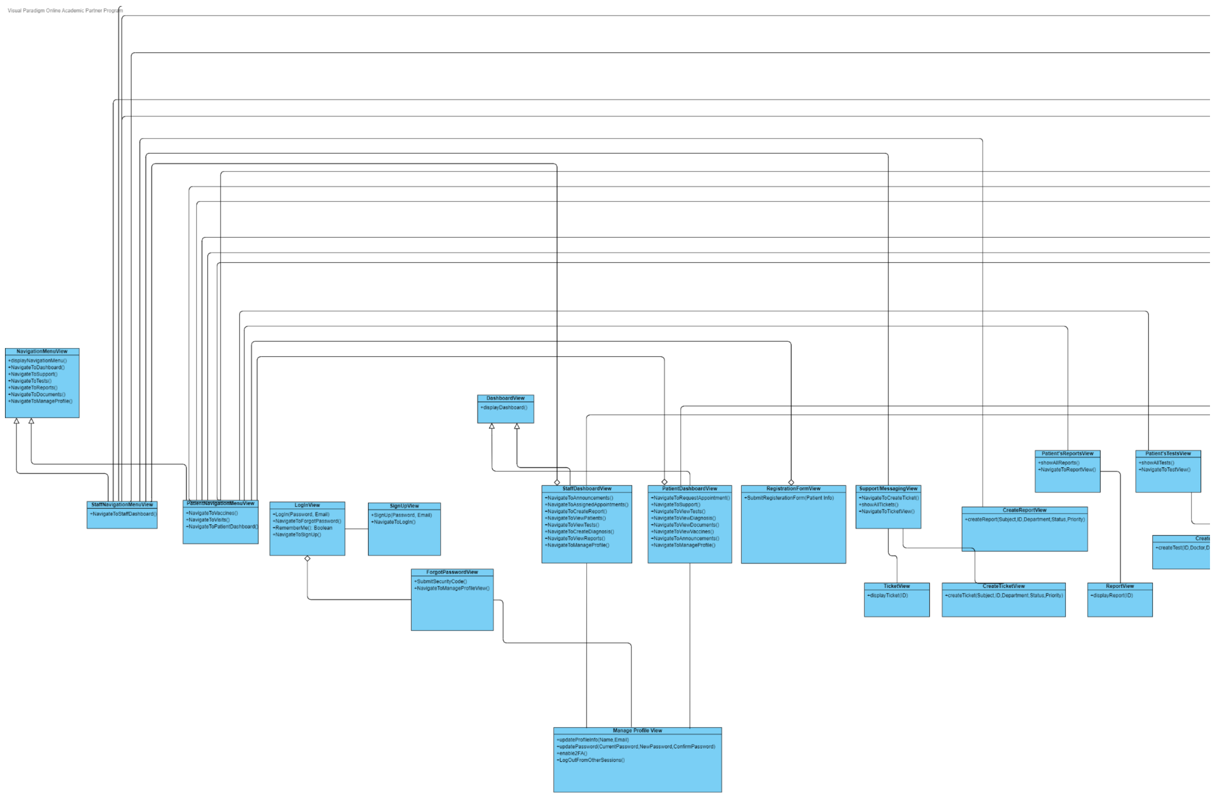
3. SUBSYSTEM SERVICES / LAYERS

3.1. UI Layer

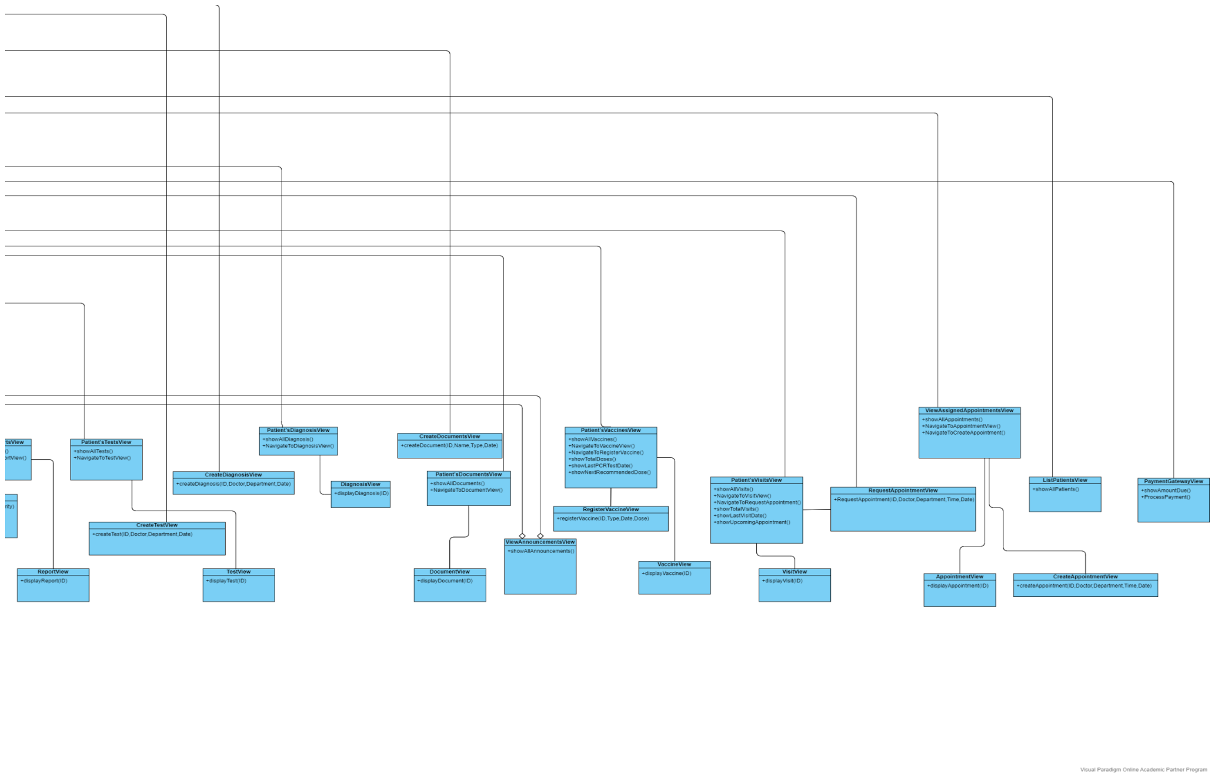
Full Diagram:



Left Side of Diagram:

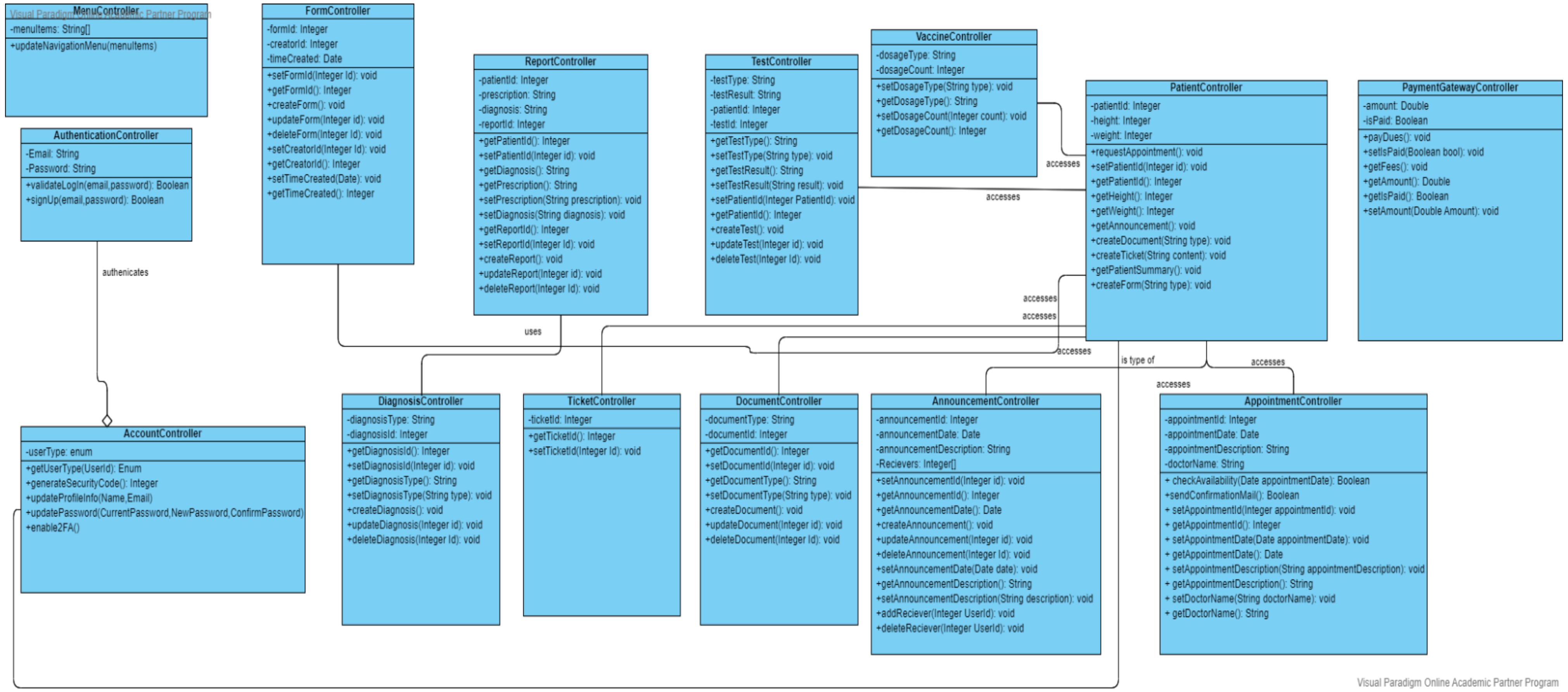


Right Side of Diagram:



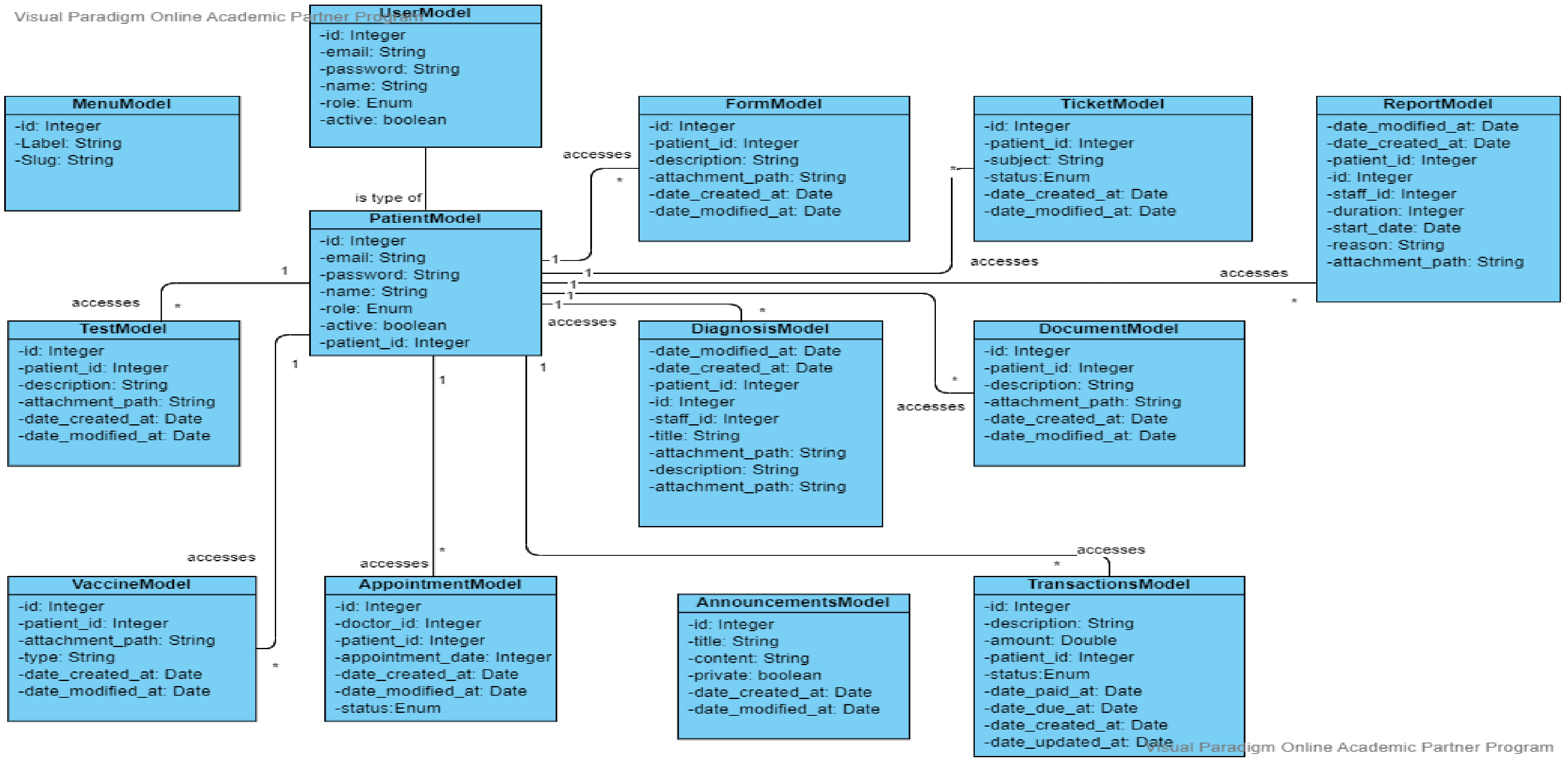
The UI layer is a boundary object between the user and the system and facilitates the interaction between the user and system. All the classes in the layer represent web pages of the website

3.2. Application Layer:


MenuController
-menuItems: String[]
+updateNavigationMenu(menuItems)
AuthenticationController
-Email: String
-Password: String
+validateLogin(email,password): Boolean
+signUp(email,password): Boolean
authenicates
AccountController
-userType: enum
+getUserType(UserId): Enum
+generateSecurityCode(): Integer
+updateProfileInfo(Name,Email)
+updatePassword(CurrentPassword,NewPassword,ConfirmPassword)
+enable2FA()
FormController
-formId: Integer
-creatorId: Integer
-timeCreated: Date
+setFormId(Integer Id): void
+getFormId(): Integer
+createForm(): void
+updateForm(Integer id): void
+deleteForm(Integer Id): void
+setCreatorId(Integer Id): void
+getCreatorId(): Integer
+setTimeCreated(Date): void
+getTimeCreated(): Integer
ReportController
-patientId: Integer
-prescription: String
-diagnosis: String
-reportId: Integer
+getPatientId(): Integer
+setPatientId(Integer id): void
+getDiagnosis(): String
+getPrescription(): String
+setPrescription(String prescription): void
+setDiagnosis(String diagnosis): void
+getReportId(): Integer
+setReportId(Integer Id): void
+createReport(): void
+updateReport(Integer id): void
+deleteReport(Integer Id): void
uses
TestController
-testType: String
-testResult: String
-patientId: Integer
-testId: Integer
+getTestType(): String
+setTestType(String type): void
+getTestResult(): String
+setTestResult(String result): void
+setPatientId(Integer PatientId): void
+getPatientId(): Integer
+createTest(): void
+updateTest(Integer id): void
+deleteTest(Integer Id): void
VaccineController
-dosageType: String
-dosageCount: Integer
+setDosageType(String type): void
+getDosageType(): String
+setDosageCount(Integer count): void
+getDosageCount(): Integer
accesses
accesses
PatientController
-patientId: Integer
-height: Integer
-weight: Integer
+requestAppointment(): void
+setPatientId(Integer id): void
+getPatientId(): Integer
+getHeight(): Integer
+getWeight(): Integer
+getAnnouncement(): void
+createDocument(String type): void
+createTicket(String content): void
+getPatientSummary(): void
+createForm(String type): void
accesses
accesses
accesses
is type of
accesses
accesses
PaymentGatewayController
-amount: Double
-isPaid: Boolean
+payDues(): void
+setIsPaid(Boolean bool): void
+getFees(): void
+getAmount(): Double
+getIsPaid(): Boolean
+setAmount(Double Amount): void
DiagnosisController
-diagnosisType: String
-diagnosisId: Integer
+getDiagnosisId(): Integer
+setDiagnosisId(Integer id): void
+getDiagnosisType(): String
+setDiagnosisType(String type): void
+createDiagnosis(): void
+updateDiagnosis(Integer id): void
+deleteDiagnosis(Integer Id): void
TicketController
-ticketId: Integer
+getTicketId(): Integer
+setTicketId(Integer Id): void
DocumentController
-documentType: String
-documentId: Integer
+getDocumentId(): Integer
+setDocumentId(Integer id): void
+getDocumentType(): String
+setDocumentType(String type): void
+createDocument(): void
+updateDocument(Integer id): void
+deleteDocument(Integer Id): void
AnnouncementController
-announcementId: Integer
-announcementDate: Date
-announcementDescription: String
-Recievers: Integer[]
+setAnnouncementId(Integer id): void
+getAnnouncementId(): Integer
+getAnnouncementDate(): Date
+createAnnouncement(): void
+updateAnnouncement(Integer id): void
+deleteAnnouncement(Integer Id): void
+setAnnouncementDate(Date date): void
+getAnnouncementDescription(): String
+setAnnouncementDescription(String description): void
+addReciever(Integer UserId): void
+deleteReciever(Integer UserId): void
AppointmentController
-appointmentId: Integer
-appointmentDate: Date
-appointmentDescription: String
-doctorName: String
+ checkAvailability(Date appointmentDate): Boolean
+sendConfirmationMail(): Boolean
+ setAppointmentId(Integer appointmentId): void
+ getAppointmentId(): Integer
+ setAppointmentDate(Date appointmentDate): void
+ getAppointmentDate(): Date
+ setAppointmentDescription(String appointmentDescription): void
+ getAppointmentDescription(): String
+ setDoctorName(String doctorName): void
+ getDoctorName(): String

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 passes them on to the Data layer.

3.3. Data Layer:

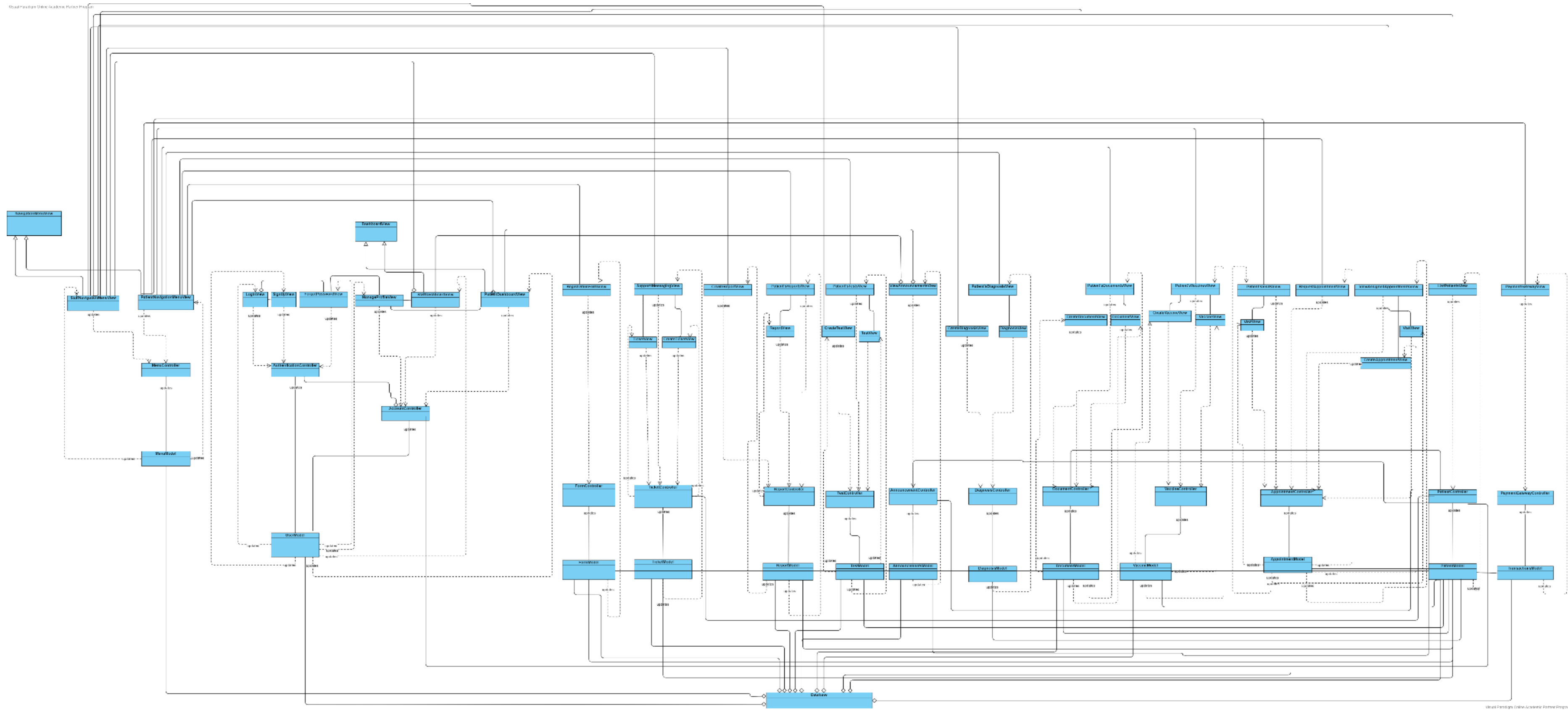


The data Layer contains all the models of the application. It is responsible for managing all the data and storing it in the database. The input for the model is received from the controller.

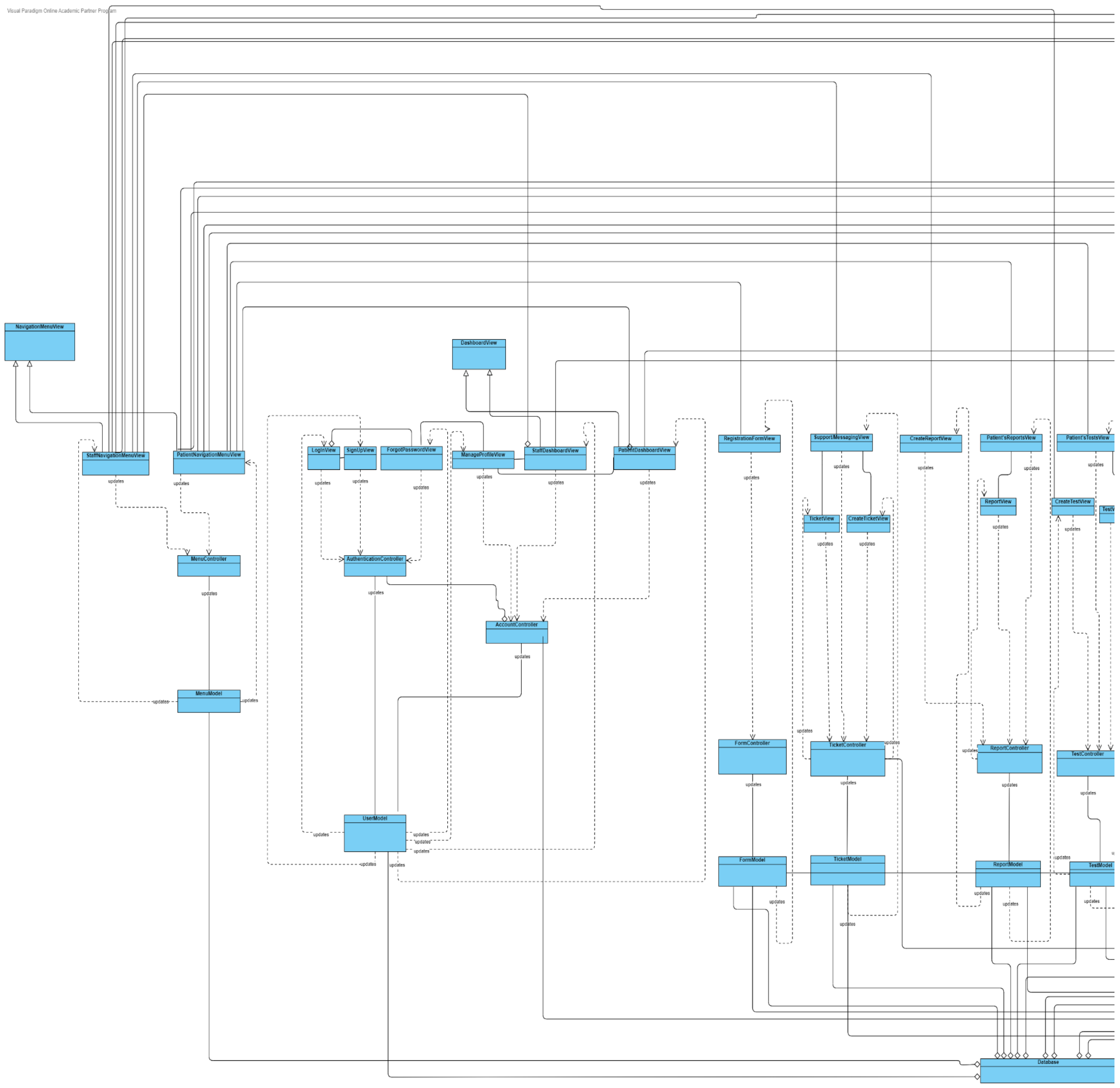
4. OBJECT DESIGN / LOW-LEVEL DESIGN

4.1. Object Design Diagram

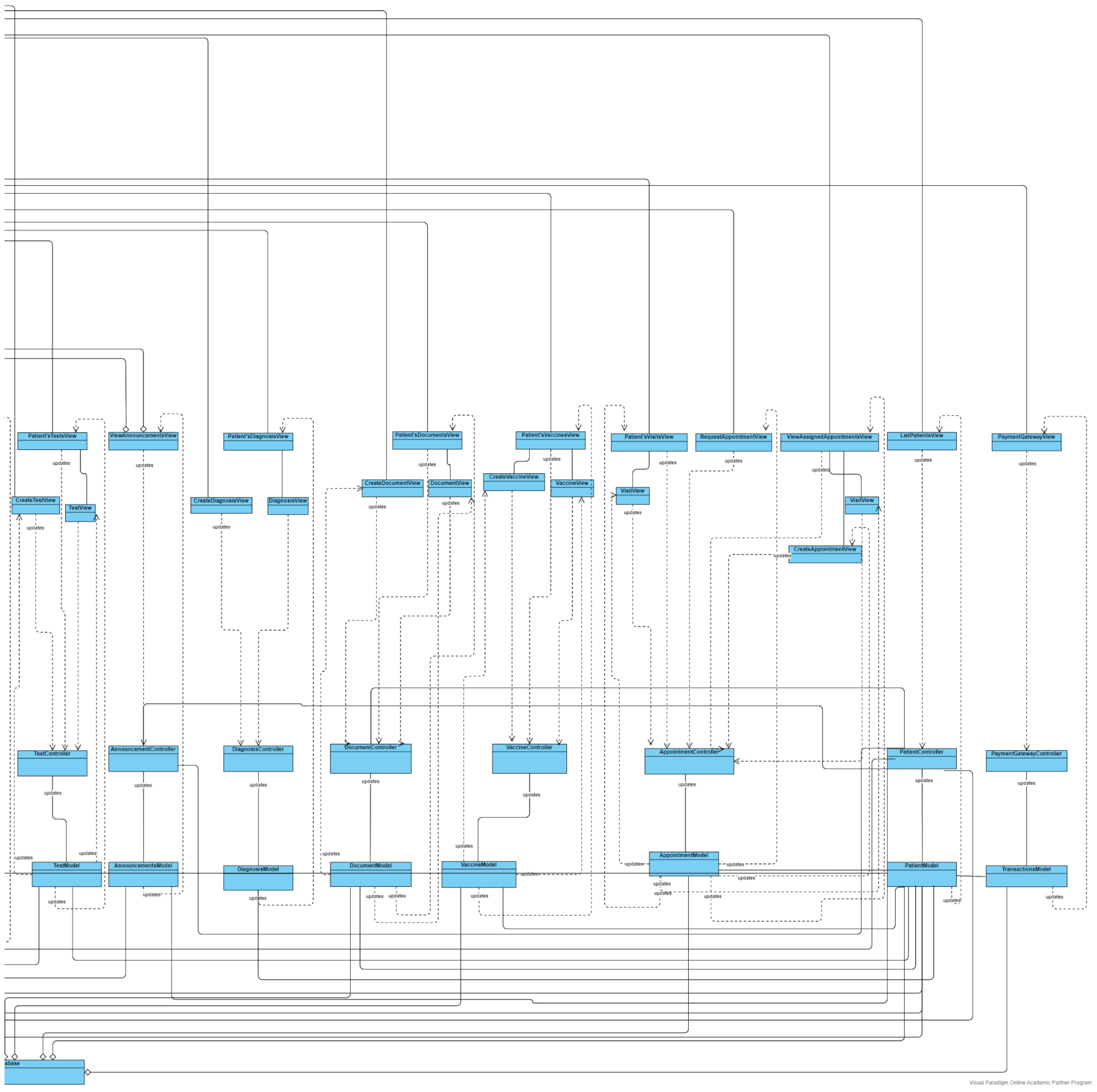
Full Diagram:



Left Side of the Diagram:



Right Side of the Diagram:



4.2. Object Design Trade-Offs

4.2.1. <u>Performance versus Cost:</u>

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 requires a powerful server and creates an extra resource cost for the project.

4.2.2. <u>Functionality versus Maintainability:</u>

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.

4.2.3. <u>Backward Compatibility versus Maintainability:</u>

Our project will be supported by 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.

4.2.4. <u>Portability versus Rapid Development:</u>

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:

+ Boolean validateLogin(): 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:

+ Enum getUserType(UserId): This method returns the user type of given userId.
+ Integer generateSecurityCode(): This method generates security code for accounts.
+ void updateProfileInfo(Name, Email): This method updates profile info.
+ void 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.
+ void enable2FA(): Enables two factor authentication for the user.

6.1.3. MenuController

Attributes:

-String[] menuItems: Stores the menu items

Methods:

+ void 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:

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

6.1.5. PatientController

Attributes:

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

Methods:

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

6.1.6. <u>PatientTestController</u>

Attributes:

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

6.1.7. <u>PatientReportController</u>

Attributes:

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

-Integer reportId: Id numbers that are assigned to reports

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

6.1.8. PatientDocumentController

Attributes:

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

Methods:

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

6.1.9. PatientVaccineController

Attributes:

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

Methods:

+ void setDosageType(String type): Modifies the dosage type.

+ StringgetDosageType(): Returns the dosage type.
+ void setDoseCount(Integer count): Modifies the dose count.
+ Integer getDoseCount(): Returns the dose count.

6.1.10. TicketController

Attributes:

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

Methods:

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

+ void payDues(): Pay the patient's due
+ void setIsPaid(Boolean bool): Set patient's paid condition
+ void displayFees(): Display patient's fees
+ void setAmount(Double amount): 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. MenuModel

- **Integer 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

- **Integer id:** The id of the appointment
- **Integer patient_id:** The patient's id
- **Integer doctor_id:** The doctor's id
- **Date appointment_date:** The date of the appointment
- **Date date_created_at:** The creation date of the appointment
- **Date date_modified_at:** The modification date of the appointment
- **Enum status:** The status of the appointment (new, canceled, past)

6.2.3. AnnouncementModel

- **Integer 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 date_created_at:** The timestamp of the announcement creation
- **Date date_modified_at:** The timestamp of the announcement modification

6.2.4. PatientTestModel

- **Integer id:** The id of the test
- **Integer 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 date_created_at:** The timestamp of the test creation
- **Date date_modified_at:** The timestamp of the test modification

6.2.5. PatientReportModel

- **Integer id:** The id of the health report
- **Integer patient_id:** The id of the patient linked to the health report
- **Integer staff_id:** The id of the staff member who created the health report
- **Integer duration:** The duration that the health report covers
- **date 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 date_created_at:** The timestamp of the report creation
- **Date date_modified_at:** The timestamp of the report modification
-

6.2.6. PatientVaccineModel

- **Integer id:** The id of the vaccine model
- **Integer 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 date_created_at:** The timestamp of the vaccine model creation
- **Date date_modified_at:** The timestamp of the vaccine model modification

6.2.7. PatientDiagnosisModel

- **Integer id:** The id of the diagnosis
- **Integer patient_id:** The id of the patient linked to the diagnosis
- **Integer 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 date_created_at:** The timestamp of the diagnosis creation
- **Date date_modified_at:** The timestamp of the diagnosis modification

6.2.8. PatientDocumentModel

- **Integer id:** The id of the patient’s documents
- **Integer 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 date_created_at:** The timestamp of the document creation
- **Date date_modified_at:** The timestamp of the document modification

6.2.9. PatientFormModel

- **Integer id:** The id of the patient’s forms
- **Integer patient_id:** The id of the patient
- **String description:** The descriptions in the patient’s forms
- **String attachment_path:** The relative path to the attachment
- **Date date_created_at:** The timestamp of the form creation
- **Date date_modified_at:** The timestamp of the form modification

6.2.10. TicketModel

- **Integer id:** The id of the ticket
- **Integer 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 date_created_at:** The timestamp of the ticket creation
- **Date date_modified_at:** The timestamp of the ticket modification

6.2.11. UserModel

- **Integer 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

6.2.12. PatientModel

- **Integer id:** The id of the user
- **Integer patined_id:** The id of the patient
- **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. TransactionsModel

- **Integer id:** The id of the invoice
- **String description:** The description of the invoice
- **Double amount:** The amount which is due against the invoice
- **Integer patient_id:** The id of the patient linked with the invoice
- **Enum status:** The status of the invoice (unpaid, paid, overdue, canceled, refunded)
- **Date date_paid_at:** The timestamp at which the invoice was paid
- **Date date_due_at:** The timestamp at which the invoice is due
- **Date date_created_at:** The timestamp of the invoice creation
- **Date date_updated_at:** The timestamp of the invoice modification

7. Improvement Summary

As recommended by TA and the peer review sibling group, the Design Goals section outlines the actions required to achieve each design goal. Added a detailed model class diagram with relationships and labels and a distribution diagram. In addition, the Subsystem decomposition scheme was corrected, and attention was drawn to the explanations provided later in the document based on the recommendations of TA and the peer review sibling group. Additionally, the application layer has been updated to reflect the relationships between classes, and their relationships have been tagged. A detailed description of what will be stored in the database has been provided, and an access control matrix has been added. The actions and failures that the administrators should take to start/end the application were added to the boundary conditions according to the recommendations from TA and the peer review sibling group. Also, all class diagrams have been updated to show the correct relationships between labeled classes. While making all these updates, references were made to the Professor's slides and past exams. Any note left on its review by TA in the first iteration was considered in the second iteration. The group then discussed all of the peer review sibling group notes, and the report was reviewed to ensure that no further changes were needed. The understanding of diagrams and technical report writing improved considerably during this period, making changes accordingly.

8. 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/