

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Платформа автоматизації роботи медичних
закладів»**

Виконав студент групи ПЗК-20-1
СОБОЛЕВСЬКИЙ Дмитро Анатолійович

Керівник роботи:
КОНЦЕДАЙЛО Валерій Валерійович

Консультант:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ЛЕВКІВСЬКИЙ Віталій Леонідович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СОБОЛЕВСЬКИЙ Дмитро Анатолійович**

Керівник роботи: **КОНЦЕДАЙЛО Валерій Валерійович**

Тема роботи: **«Платформа автоматизації роботи медичних закладів»,**
затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: Об'єктом дослідження є процес створення веб додатку автоматизації роботи медичних закладів, інтерфейсної частини, серверної частини та їх взаємодія

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник	Завдання видав	Завдання прийняв
1	Концедайло В.В. Чижмотря О.В.	16.02.2023р.	16.02.2023р.
2	Концедайло В.В. Чижмотря О.В.	16.02.2023р.	16.02.2023р.
3	Концедайло В.В. Чижмотря О.В.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	16 лютого 2023 – 18 лютого 2023	виконано
2	Аналіз аналогів програмного продукту	18 лютого 2023 – 23 лютого 2023	виконано
3	Визначення вимог до додатку	23 лютого 2023 – 3 березня 2023	виконано
4	Вибір інструментальних засобів	3 березня 2023 – 4 березня 2023	виконано
5	Визначення варіантів використання	4 березня 2023 – 10 березня 2023	виконано
6	Проектування та реалізація алгоритмів системи	10 березня 2023 – 23 березня 2023	виконано
7	Написання програмного коду	23 березня 2023 – 7 травня 2023	виконано
8	Тестування роботи програмного продукту	8 травня 2023 – 12 травня 2023	виконано
9	Оформлення пояснювальної записки	12 травня 2023 – 30 травня 2023	виконано
10	Попередній захист	31 травня	виконано

РЕФЕРАТ

Випускна робота бакалавра складається з розробки вебдодатку, серверної частини до нього та пояснювальної записки. Пояснювальна записка до випускної роботи містить 91 сторінку, 54 ілюстрації, 7 таблиць, 15 додатків, 15 літературних джерел.

Метою роботи є розробка вебдодатку платформи автоматизації роботи медичних закладів.

В роботі визначено основні завдання на розробку системи, проаналізовано аналоги розробленої системи. Наведено сценарії додатку, загальна структура додатку, опис алгоритмів взаємодії окремих модулів системи, опис об'єктної структури системи (діаграма класів). Також приводяться результати тестування системи.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗОВАНА СИСТЕМА, ВЕБДОДАТОК, ВЕБ-ПРОЄКТ, FRONT-END, BACK-END, UI

					ІПЗ.КР.Б – 121 – 23 - ПЗ			
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>	Пояснювальна записка Платформа автоматизації роботи медичних закладів	<i>Літ.</i>	<i>Арк.</i>	<i>Аркушів</i>
<i>Розроб.</i>		<i>Д.А. Соболевський</i>						
<i>Керівник.</i>		<i>В.В. Концедайло</i>					4	91
<i>Консульт.</i>		<i>О.В. Чижморя</i>				Житомирська політехніка, Група ІПЗк 20-1		
<i>Рецензент.</i>		<i>В.Л. Левківський</i>						
<i>Зав. каф.</i>		<i>А.В. Морозов</i>						

ABSTRACT

The bachelor's thesis consists of the development of a web automatization platform for medical mortgage, back-end part for them and explanatory note. The explanatory note to the final thesis contains 91 pages, 54 illustrations, 7 tables, 15 appendices, 15 sources used.

The purpose of the work is to develop a web automatization platform for medical mortgage.

The paper defines the main tasks for the development of the system, and analyzes the analogues of the developed system. Scenarios of the software complex, the general structure of the software complex, a description of the interaction algorithms of individual system modules, a description of the object structure of the system (class diagram) are given. The results of system testing are also provided.

KEY WORDS: FRONT-END, BACK-END, UI, AUTOMATION SYSTEM, WEB APPLICATION, WEB PROJECT.

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		5

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	10
1.3. Вибір архітектури	13
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ	22
2.1. Визначення варіантів використання та структури системи.....	22
2.2. Проєктування та реалізація алгоритмів роботи системи	33
2.3. Реалізація функціонала додатку	37
РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ СИСТЕМИ.....	41
3.1. Архітектура додатку та його розгортання.	41
3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з додатком.....	42
3.3. Тестування роботи додатку	48
ВИСНОВКИ	52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	53
ДОДАТКИ	54

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FE – Front-end

BE – Back-end

UI – User interface

MUI – Material User Interface

GQL – GraphQL

TS – Typescript

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Швидкість медичної допомоги та її якість є запорукою забезпечення здоров'я пацієнтів, зростання рівня якості медицини та її доступності. Майже не існує людини яка хоча б раз не була залучена в медичну сферу, а ринок медичних послуг та суміжні з ним займають велику частку економіки та залучає велику частку висококваліфікованих кадрів. Постає питання поліпшення якості взаємодії між сферою медичних послуг та залученими у неї людьми. Наразі медичні заклади мають дуже великі проблеми з актуалізацією своїх бізнес-процесів, сюди входять як великі архіви в яких часто може бути втрачена чутлива інформація так і проблеми з навіть потрапити на сеанс до лікаря.

Метою даної випускної роботи є Розробка веб додатку платформи автоматизації роботи медичних закладів.

Встановлена мета обумовлює наступні завдання:

- Розробка інтерфейсу користувача веб додатку;
- Розробка серверної частини додатку;
- Реалізація бізнес-логіки користувачів та закладів таких як: реєстрація, авторизація, перегляд, додавання звітів, створення медичних сеансів тощо;

Об'єктом дослідження є процес створення веб додатку, інтерфейсу додатку, серверної частини та їх взаємодія

Предметом дослідження є використання бібліотеки React для створення графічних інтерфейсів та фреймоврк NestJS та мови запитів GraphQL для створення серверної частини.

Методи дослідження. В роботі було використано методи об'єктно-орієнтованого проектування, вебпрограмування та дизайну інтерфейсів.

Розроблена система може бути використана для роботи як одного медичного закладу, так і цілої мережі.

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

Основною метою розробки веб додатку платформи автоматизації роботи медичних закладів є створення зручного інтерфейсу та стійкої серверної частини. Користувач не повинен відчувати незручностей при використанні додатку а кількість помилок повинна бути мінімальна.

У випускній роботі необхідно розробити додаток веб додатку платформи автоматизації роботи медичних закладів, використати найбільш вдалі та сучасні підходи для створення максимально якісного продукту. Для реалізації поставленого завдання необхідно дотримуватися певного спектра вимог.

Для реалізації поставленого завдання, основними етапами є:

1. Розробити дизайн користувацького інтерфейсу.

1.1. Створити загальну концепцію дизайну користувацького інтерфейсу.

1.2. Створити та застосувати елементи інтерфейсу за концептом компонентів:

- Панель авторизації
- Панель реєстрації
- Головна сторінка
- Функціональна дошка
- Особистий кабінет

2. Проектування та розробка основних модулів серверної частини

- Модуль користувачів
- Модуль Лікарень
- Модуль Файлів
- Модуль для роботи з поштою
- Модуль медичних напрямків
- Модуль медичних сервісів

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

- Модуль візитів
- Модуль Оплати

1.2. Аналіз аналогів програмного продукту

На сьогоднішній день медична допомога це загальнодоступна послуга, але під час її використання існує великий спектр проблем, неможливість потрапити до лікаря, втрачені документи, проблеми з лікарським графіком.

Основні фактори при взаємодії з медичними закладами:

1. Відсутня прозорість робочого графіку: Багато лікарів мають певний специфічний графік роботи який не завжди є контрольованим
2. Складний бюрократичний апарат: До цієї проблеми входить як і необхідність великої кількості документів та сертифікатів
3. Зручність взаємодії: Більшість медичних закладів потребують фізичної присутності для взаємодії;

Наразі на ринку не так багато продуктів що можуть задовольнити потреби медичної сфери, до них входять такі представники як Clinic box, Veeva, HealthCare CRM

Таблиця 1.1.

Порівняльна характеристика

Назва	Зручність GUI	Графічне оформлення	Демонстраційна версія
Salesforce Health Cloud	Добре	Добре	Ні
Veeva	Нормально	Добре	Ні
HealthCare CRM	Добре	Добре	Ні

Розглянемо більш детально табл. 1.1.

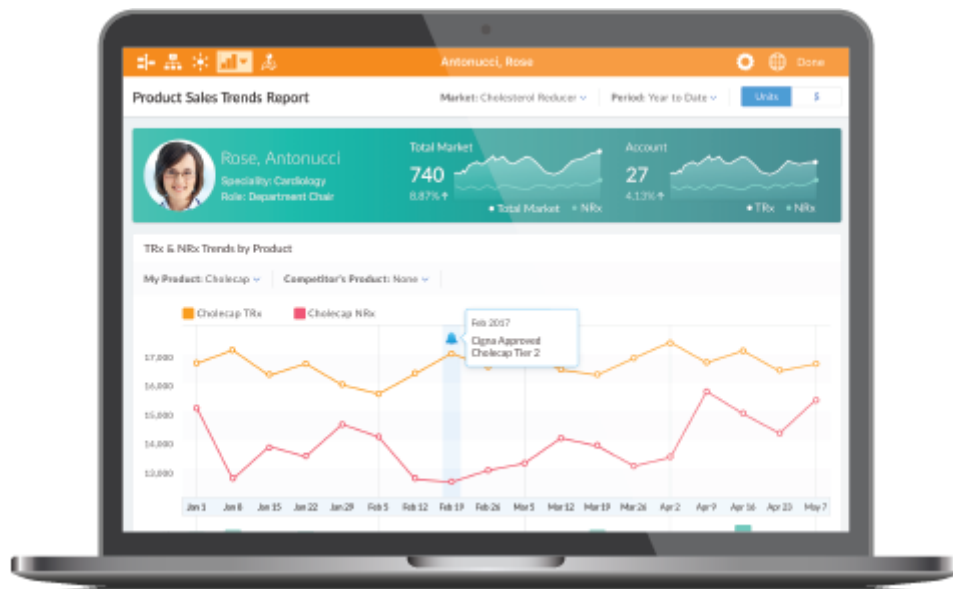


Рис. 1.1 – Veeva CRM

Veeva CRM є відомою системою управління відносинами з клієнтами, яка спеціалізується на медичній і фармацевтичній промисловості. Вона розроблена для покращення ефективності та продуктивності маркетингових та продажних процесів в медичних закладах. Ось деякі ключові аспекти використання Veeva CRM у медичних закладах:

- Керування взаємодією з лікарями: Veeva CRM дозволяє вести ефективну комунікацію з лікарями, записувати дані про зустрічі, візити та спілкування. Він надає можливість створювати та відстежувати маршрути візитів до лікарів, розподіляти ресурси та змінювати плани в залежності від потреб.

- Маркетингові активності: Veeva CRM допомагає виконувати маркетингові кампанії, такі як розсилка електронної пошти, надсилання інформації про нові продукти або ліки, організація навчальних заходів для медичних фахівців та інше. Ви можете відстежити ефективність кампаній та аналізувати інформацію для покращення стратегій маркетингу.

- Ділові аналітики: Veeva CRM забезпечує детальний аналіз даних про клієнтів, сприяючи прийняттю обґрунтованих рішень. Ви можете відстежувати показники продажів, результати маркетингових кампаній, звіти про відвідування лікарів, аналізувати ринкові тенденції та інші ключові метрики.

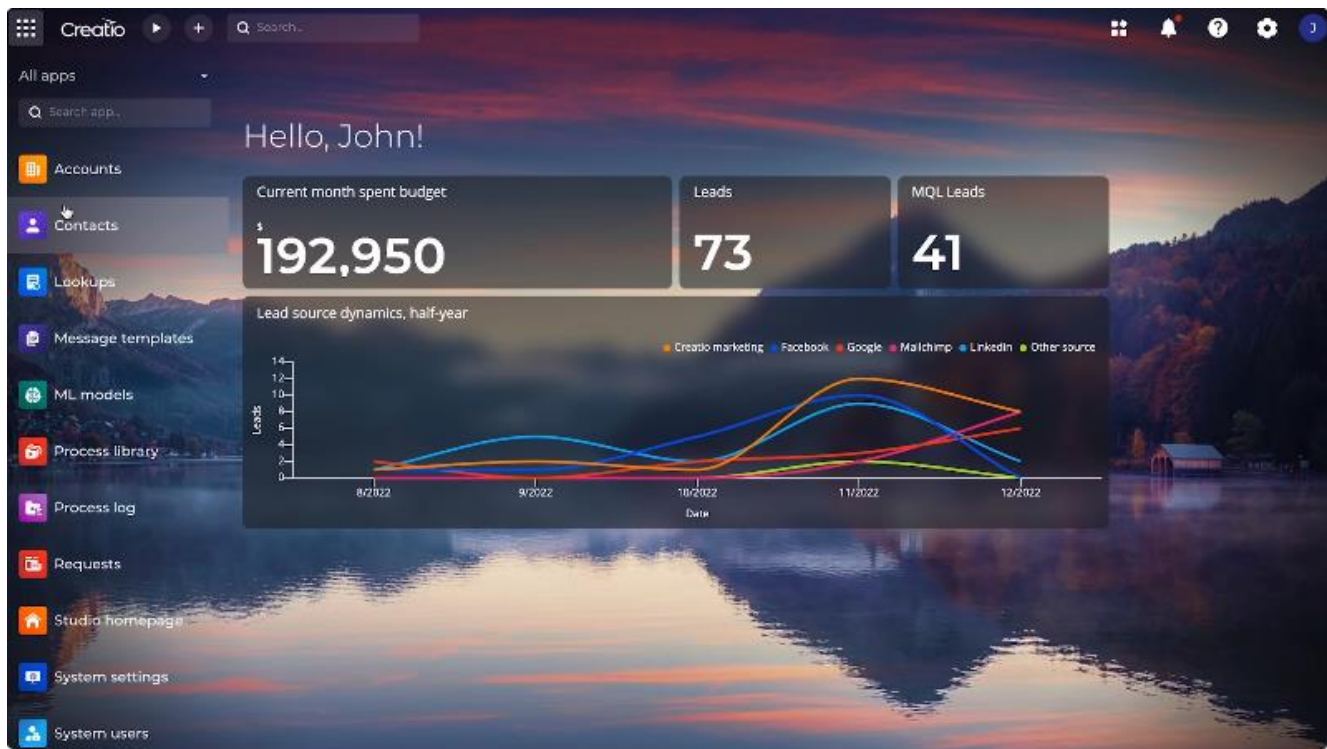


Рис. 1.2 – HealthCare CRM.

HealthCare CRM (Customer Relationship Management) є програмним рішенням, спеціально розробленим для медичних закладів і організацій у сфері охорони здоров'я. Воно допомагає покращити ефективність та ефективність управління відносинами з пацієнтами, співпрацю зі страховими компаніями та іншими зацікавленими сторонами в галузі охорони здоров'я.

Основні функції HealthCare CRM включають наступне:

- Управління взаємодією з пацієнтами: HealthCare CRM дозволяє медичним закладам ефективно керувати взаємодією з пацієнтами, включаючи записи на прийом, нагадування про медичні процедури або візити, сповіщення про результати досліджень, надання звітності про медичні послуги та зберігання історії лікування.

- Керування взаємодією зі страховими компаніями: HealthCare CRM допомагає встановлювати та підтримувати зв'язок зі страховими компаніями. Він надає можливість обробляти та відстежувати клієнтські політики страхування, рахунки, заявки на оплату послуг, автоматично відправляти рахунки страховикам та вести облік фінансових взаєморозрахунків.

		Соболевський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

– Управління маркетинговими стратегіями: HealthCare CRM дозволяє виконувати маркетингові кампанії в медичній галузі. Ви можете відстежувати інтереси та потреби пацієнтів, надсилати персоналізовані повідомлення, проводити опитування та відгуки, організовувати програми лояльності та акції для залучення нових пацієнтів

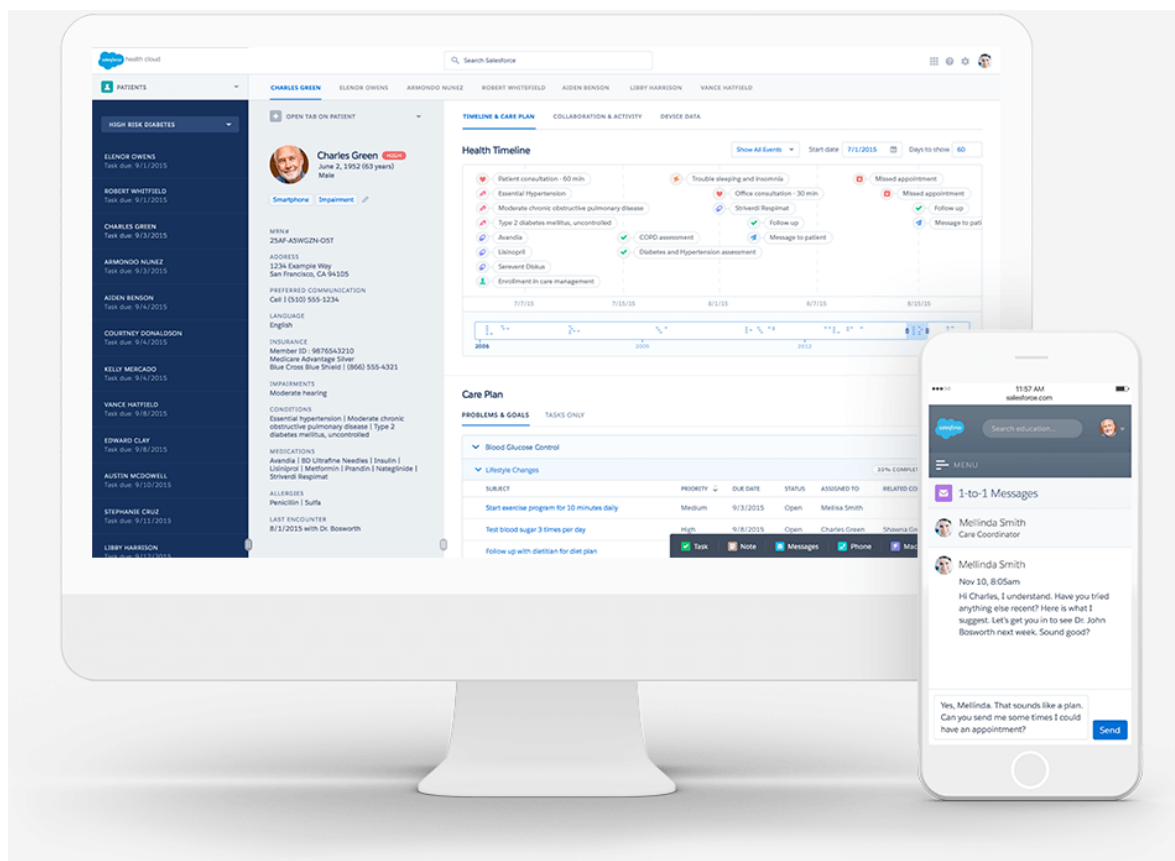


Рис. 1.3 – Salesforce Health Cloud: Salesforce Health Cloud

Є однією з найпопулярніших CRM-систем для медичної сфери. Вона надає можливості для керування відносинами з пацієнтами, координації опіки, управління заявками на медичні послуги та забезпечення комунікації між медичним персоналом.

1.3. Вибір архітектури

Додаток буде використовувати клієнт серверну архітектуру.

		Соболєвський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Клієнт-серверна архітектура це модель розподіленого обчислення, де функціональність системи розподілена між двома основними компонентами: клієнтом і сервером [9].

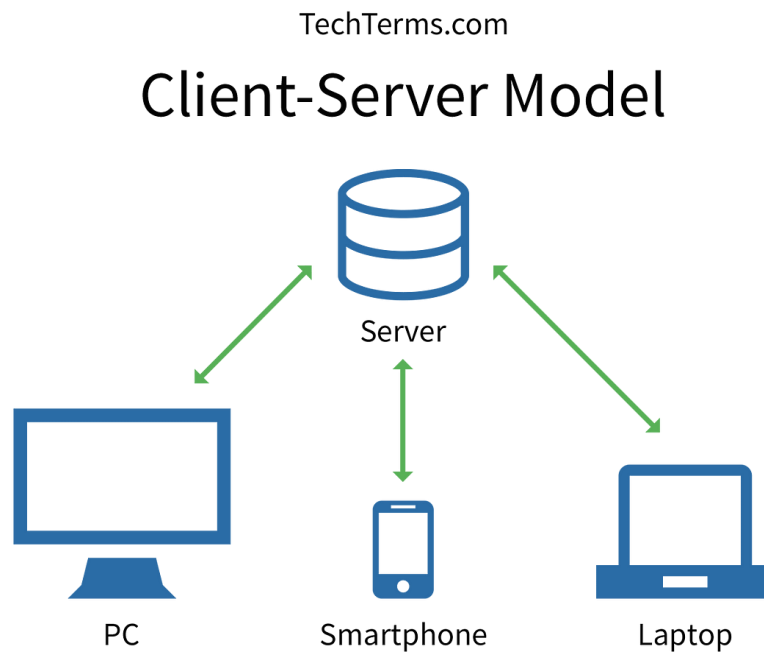


Рис 1.4. Клієнт-серверна архітектура

Для розробки вищезазначеної системи є можливість використати наступні шаблони:

- MVC
- MVVM
- DI

NestJS змушує нас використовувати шаблон MVC в той час, як React реалізує шаблон MVVM

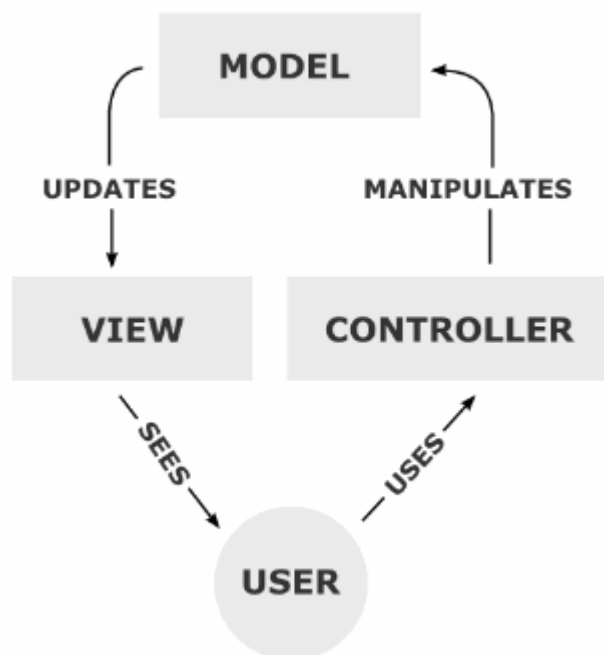


Рис. 1.5 – Model-View-Controller

MVC (Model-View-Controller) - це шаблон проєктування програмного забезпечення, який допомагає організувати структуру інтерфейсу користувача та бізнес-логіки додатку. Він розділяє програму на три основні компоненти: Model (модель), View (вид) і Controller (контролер), кожен з яких відповідає за певні аспекти додатку [13].

– Модель (Model): Модель представляє бізнес-дані та бізнес-логіку додатку. Вона відповідає за збереження та обробку даних, виконання операцій над ними, а також взаємодію з базою даних або інших джерелами даних. Модель не залежить від інтерфейсу користувача або відображення даних.

– Вид (View): Вид відповідає за представлення даних і управління інтерфейсом. Він відображає дані з моделі та дозволяє користувачу взаємодіяти з додатком. Вид може бути графічним інтерфейсом, вебсторінкою, текстовим виводом або будь-яким іншим способом відображення даних.

– Контролер (Controller): Контролер обробляє вхідні дані від користувача та керує потоком додатку. Він приймає дії користувача з вигляду, взаємодіє з моделлю для отримання або оновлення даних і відповідає за оновлення

відображення відповідно до змін у моделі. Контролер також може включати логіку обробки помилок та маршрутизації.

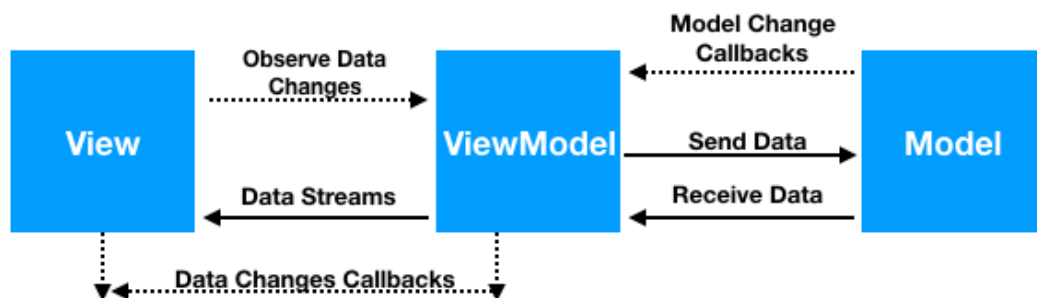


Рис. 1.6 – Mode-ViewModel-Model

MVVM (Model-View-ViewModel) - це шаблон проєктування програмного забезпечення, який використовується для структурування та розробки користувацького інтерфейсу. Він є розширенням шаблону MVC і надає більшу розділеність між логікою бізнесу та представленням даних [11].

– Модель (Model): Модель в MVVM відповідає за управління даними та бізнес-логікою додатку. Вона зберігає дані, виконує операції з цими даними та надає методи для отримання або оновлення даних. Модель не залежить від інтерфейсу користувача або представлення даних.

– Вид (View): Вид в MVVM відображає дані та інтерфейс користувача. Це може бути графічний інтерфейс, вебсторінка або будь-яке інше представлення даних. Він відповідає за відображення даних, обробку подій користувача та взаємодію з користувачем.

– ViewModel: ViewModel є посередником між Моделлю та Видом. Вона надає дані та операції для відображення в Виді і взаємодії з Моделлю. ViewModel містить логіку, яка відповідає за обробку введення від користувача, маніпулювання даними, валідацію та управління станом додатку. Вона також надає прив'язку даних між Видом і Моделлю, що дозволяє автоматично оновлювати відображення даних після їх змін.

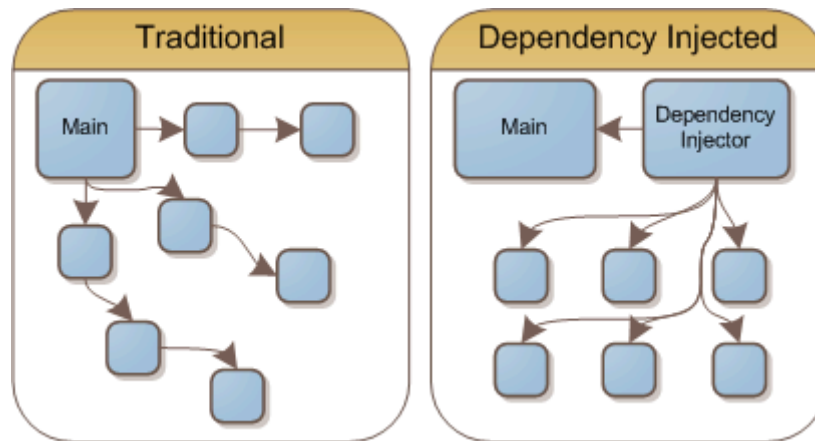


Рис. 1.7 – Dependency Injection

Dependency Injection є шаблоном в об'єктно-орієнтованому програмуванні, який дозволяє керувати залежностями між об'єктами в програмі. Замість того, щоб об'єкти створювали свої залежності напряду, вони отримують їх зовні, додаючи їх у свої конструктори, методи або властивості. Це дозволяє полегшити створення об'єктів залежностей, спрощує тестування та забезпечує більшу гнучкість і перевикористання коду. Основна ідея ДІ полягає в тому, що клас не повинен самостійно створювати або залежати від своїх залежностей. Замість цього, залежності надаються зовні, часто через інтерфейси або абстрактні класи. Це робить код більш модульним, оскільки залежності можна замінювати без необхідності зміни самого класу [10].

В якості архітектури клієнт-серверної взаємодії використаємо GraphQL

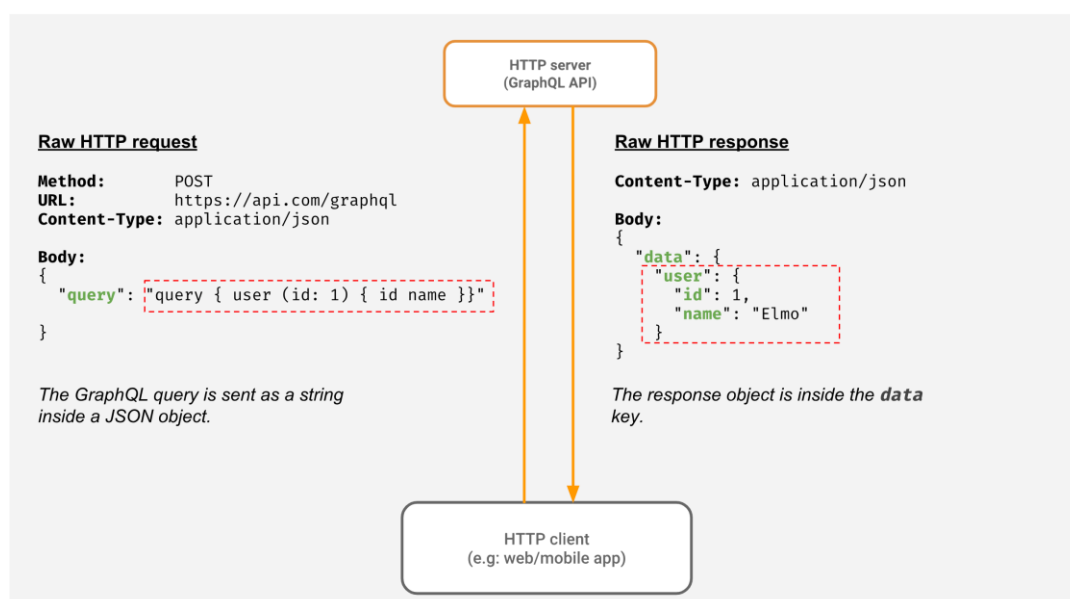


Рис. 1.8 – GraphQL

GraphQL - це мова запитів та інструмент розробки, призначений для ефективної взаємодії з API. GraphQL був розроблений компанією Facebook і широко використовується веброзробниками для побудови API з високою продуктивністю та гнучкими можливостями [7].

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.

Бібліотека для створення графічних інтерфейсів React

React - це відкрита JavaScript-бібліотека для розробки користувацьких інтерфейсів (UI). Він створений компанією Facebook і широко використовується для побудови ефективних та масштабованих вебдодатків. Це найбільш використовувана технологія на ринку графічних інтерфейсів, що дає нам велику кількість сторонніх модулів та велику підтримку як від самих розробників, так і від спільноти програмістів. React додає в розробку вебінтерфейсів такий підхід як компонентна архітектура, що означає, що весь інтерфейс розбивається на невеликі, самодостатні компоненти. Це спрощує розробку, підтримку та повторне використання коду, використовує віртуальний DOM, що дозволяє ефективно оновлювати та відображати зміни в інтерфейсі. Він автоматично визначає оптимальні способи оновлення компонентів, що призводить до швидкої та плавної роботи додатків, сприяє використанню імутабельних даних, що спрощує відстеження змін та уникнення побічних ефектів. Також, дані рухаються в одному напрямку (від батьківського компонента до дочірніх), що полегшує управління станом додатку, дозволяє підключати різні бібліотеки та розширення для покращення функціональності додатків. Також, React має велику спільноту розробників, яка активно вносить нові інструменти та рішення [3].

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

Фреймворк для написання серверної частини додатку NestJS

NestJS - це прогресивний фреймворк для розробки серверних додатків на мові JavaScript/TypeScript. Він базується на архітектурному шаблоні "Dependency Injection" та "MVC" та багато інших популярних концепцій [4].

- Модульність: NestJS пропонує модульну структуру розробки, яка дозволяє розділити функціональність додатку на невеликі, самостійні модулі. Це спрощує організацію та розширення проєкту.

- Dependency Injection: За допомогою механізму Dependency Injection (DI), NestJS дозволяє ефективно управляти залежностями між компонентами додатку. Це полегшує тестування, розширення та підтримку коду.

- TypeScript: NestJS повністю підтримує TypeScript, що дозволяє використовувати типізацію, переваги сучасного синтаксису та засоби перевірки помилок на етапі розробки.

- Розширені можливості: NestJS надає багато вбудованих функцій, таких як Middleware, Guards, Interceptors та Pipes, що допомагають забезпечити безпеку, автентифікацію, авторизацію та інші аспекти розробки серверних додатків.

- Підтримка різних протоколів: NestJS може працювати з різними протоколами, такими як HTTP, WebSocket, gRPC тощо, що дає можливість створювати різноманітні типи додатків.

- Жива спільнота: NestJS має активну спільноту розробників, яка активно розвиває додаткові пакети та надає допомогу через форуми та соціальні мережі.

Мова розробки Javascript:

JavaScript (JS) є мовою програмування, яка використовується для розробки вебдодатків та динамічної взаємодії з користувачем на стороні клієнта (в браузері) або на стороні сервера (за допомогою платформи Node.js). Основні риси та переваги JavaScript [15]:

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		19

– Веб-розробка: JavaScript є основною мовою для розробки вебдодатків. Він дозволяє створювати інтерактивні елементи, маніпулювати DOM (Document Object Model) та обробляти події на вебсторінці. JavaScript також підтримує багато фреймворків та бібліотек, таких як React, Angular та Vue.js, що полегшують розробку вебінтерфейсів.

– Широке застосування: JavaScript використовується не тільки для веброзробки, але й для розробки мобільних додатків (за допомогою фреймворків, таких як React Native та Ionic), настільних додатків (за допомогою Electron), розширень для браузерів та інших платформ.

– Динамічна мова: JavaScript є динамічною мовою, що означає, що типи даних не обов'язково вказувати при оголошенні змінних. Це дозволяє швидко писати та впроваджувати код. Однак, з'явилася підтримка статичного типу даних у вигляді TypeScript, що дозволяє використовувати типізацію та забезпечує перевірку помилок на етапі розробки.

Мова програмування та розширення мови Javascript – Typescript
TypeScript є мовою програмування, яка є розширенням JavaScript, розроблена компанією Microsoft. Вона надає можливість додавати статичну типізацію до JavaScript і допомагає в розробці більших та складніших проєктів. Ключові особливості Typescript це статична типізація, розширені функції JavaScript, підтримка сучасних стандартів, інтеграція з існуючим JavaScript кодом [5].

Мова запитів GraphQL

GraphQL – це мова запитів та середовище виконання запитів, яке дозволяє ефективно отримувати та маніпулювати даними з сервера. Він був розроблений компанією Facebook та зарекомендував себе як потужний інструмент для розробки API [2].

Основні переваги GraphQL:

– Гнучкість та ефективність: GraphQL дозволяє клієнтам точно вказувати, які дані вони потребують. Вони можуть запитувати лише необхідні поля, уникаючи надлишкових даних. Це забезпечує ефективнішу передачу даних і зменшує навантаження на мережу.

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

– Один запит, багато даних: За допомогою GraphQL клієнти можуть отримувати багато різних типів даних в одному запиті. Замість виконання кількох HTTP-запитів, вони можуть зробити лише один запит до сервера та отримати всю необхідну інформацію.

– Гнучка структура запиту: GraphQL використовує типізовану схему, що дозволяє визначити доступні типи даних та операції. Клієнти можуть динамічно конструювати запити, змінюючи поля, фільтри та сортування, що дає їм контроль над отриманими результатами.

– Автодокументація: GraphQL надає автоматичну документацію, що визначає всі доступні типи даних, поля та операції. Це спрощує розуміння API та дозволяє розробникам швидко зорієнтуватись у структурі даних [8].

– Загальна схема та розширення: GraphQL може використовуватись як шар над існуючими джерелами даних, такими як бази даних або вже створений API. Він дозволяє об'єднувати різні джерела даних в одну загальну схему та розширення.

ORM для роботи з базою даних Prisma

Prisma JS - це сучасний ORM (Object-Relational Mapping) та база даних, яка надає зручні засоби для роботи з базами даних у програмах JavaScript і TypeScript. Використання Prisma JS дозволяє легко і зручно взаємодіяти з реляційними базами даних, такими як MySQL, PostgreSQL та SQLite [6].

Висновки до розділу 1

Проведений аналіз предметної області дозволив визначити основні аспекти розробки додатку автоматизації роботи медичних закладів. Оглянуто схожі за призначення продукти, виявлено їх переваги та недоліки, описано основні вимоги до додатку.

Визначено архітектуру додатку, технологічний стек для розробки.

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ

2.1. Визначення варіантів використання та структури системи.

Даний підрозділ описує вимоги, якими повинна володіти «Платформа автоматизації роботи медичних закладів», щоб мати можливість безперервно виконувати свої функції, задовольнивши всі зазначені стандарти, специфікації та інші формальні документи.

Бізнес-вимоги:

1. Основна мета: створення платформи для автоматизації процесів медичних закладів таких як створення сеансів, оплата, відправлення звітів, комунікація з лікарнею та лікарем.
2. Представлення: проєкт має бути реалізовано як веб додаток

Функціональні вимоги:

1. Авторизація та реєстрація користувача в системі.
2. Перегляд інформації про медичні заклади та лікарів
3. Можливість надсилати користувачу звіти
4. Можливість перегляду статистики
5. Можливість створення лікарського сеансу
6. Можливість оплати послуг

Нефункціональні вимоги:

1. Сприйняття:
 - Інтерфейс повинен бути зручним та інтуїтивно зрозумілим для користувача.
 - Час відповіді системи повинен перевищувати 7 секунд
2. Надійність:
 - Додаток повинен бути стійким до великих навантажень та бути захищеним від атак.

Бізнес правила:

1. Інтерфейс має коректно відображатися на всіх популярних браузерах

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

Антивимоги:

1. Відсутність доступу до інформації сторонніх осіб



Рис. 2.1 – Варіанти використання для користувача лікаря

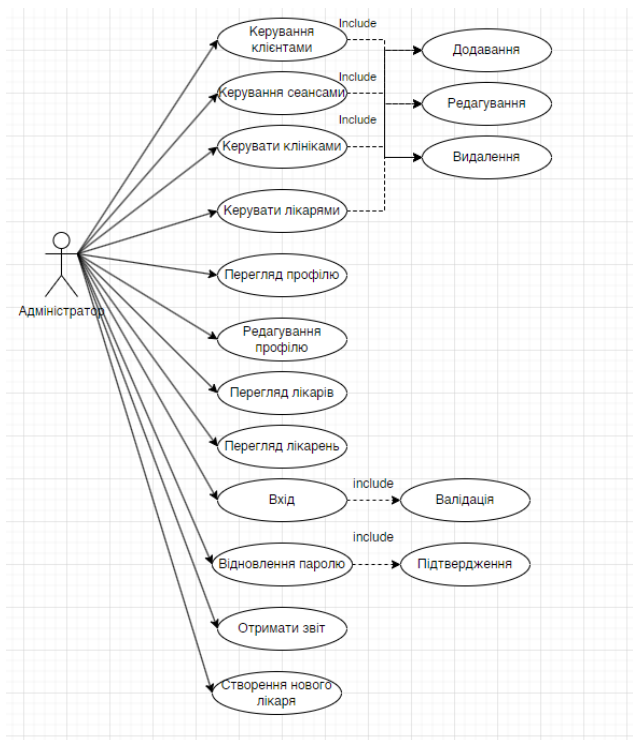


Рис. 2.2 – Варіанти використання для користувача клієнт

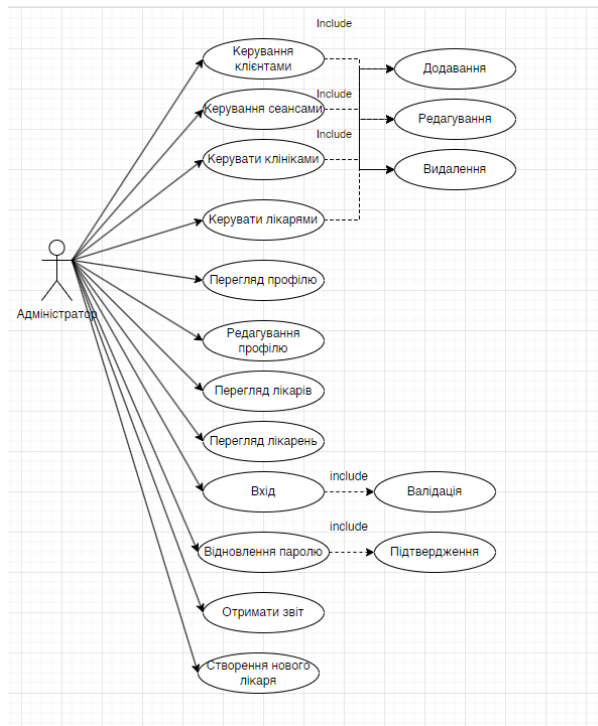


Рис. 2.3 – Варіанти використання для користувача адміністратор

Аналіз функціональних вимог дозволив виділити певні незалежні компоненти системи, що забезпечать реалізацію програмного комплексу системи. Було реалізовано модель бази даних

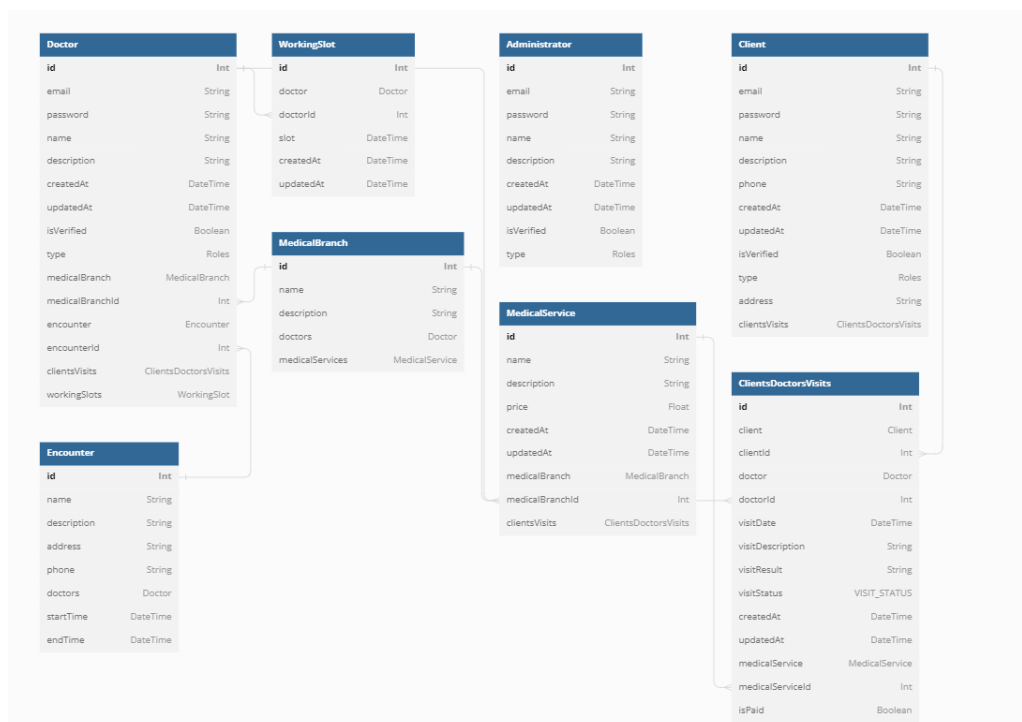


Рис. 2.4 – Модель бази даних

На рис. 2.5 наведено діаграму класів, які використовуються в серверній частині проєкту.

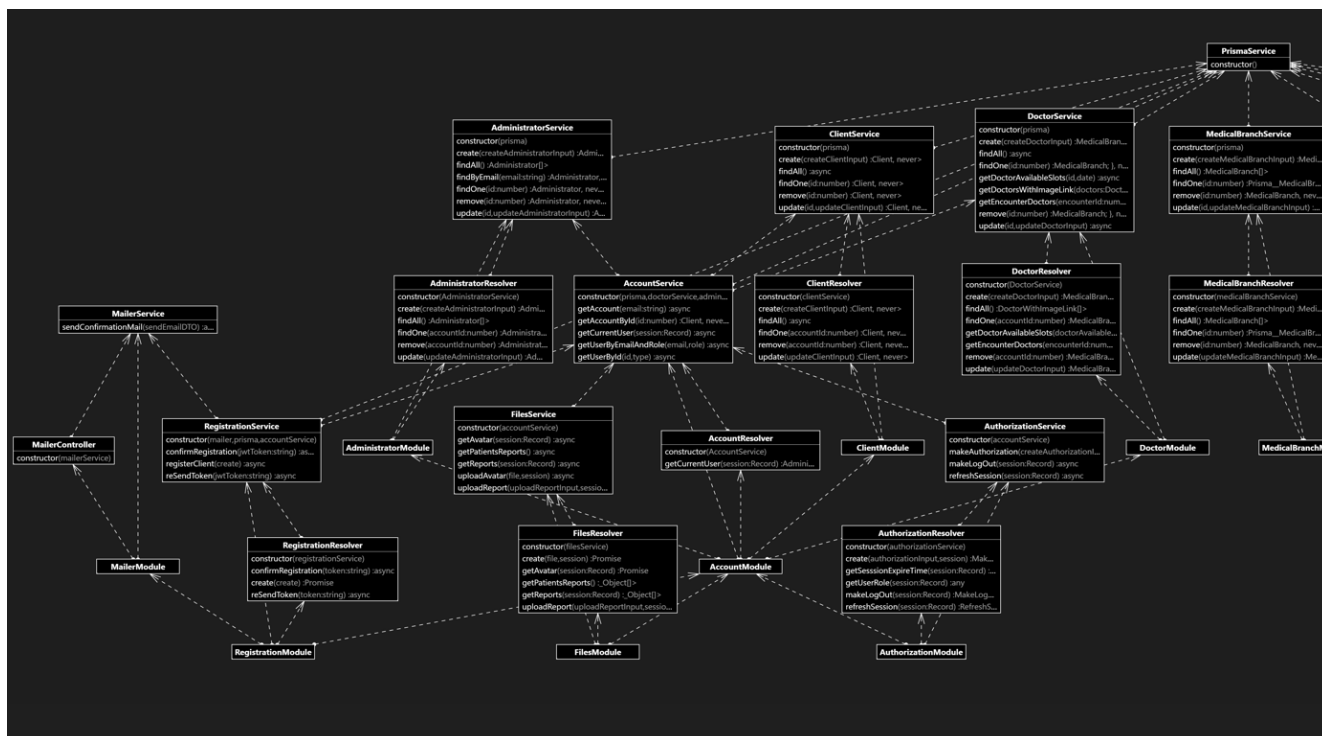


Рис. 2.5 – Діаграма класів Back-end частини проєкту

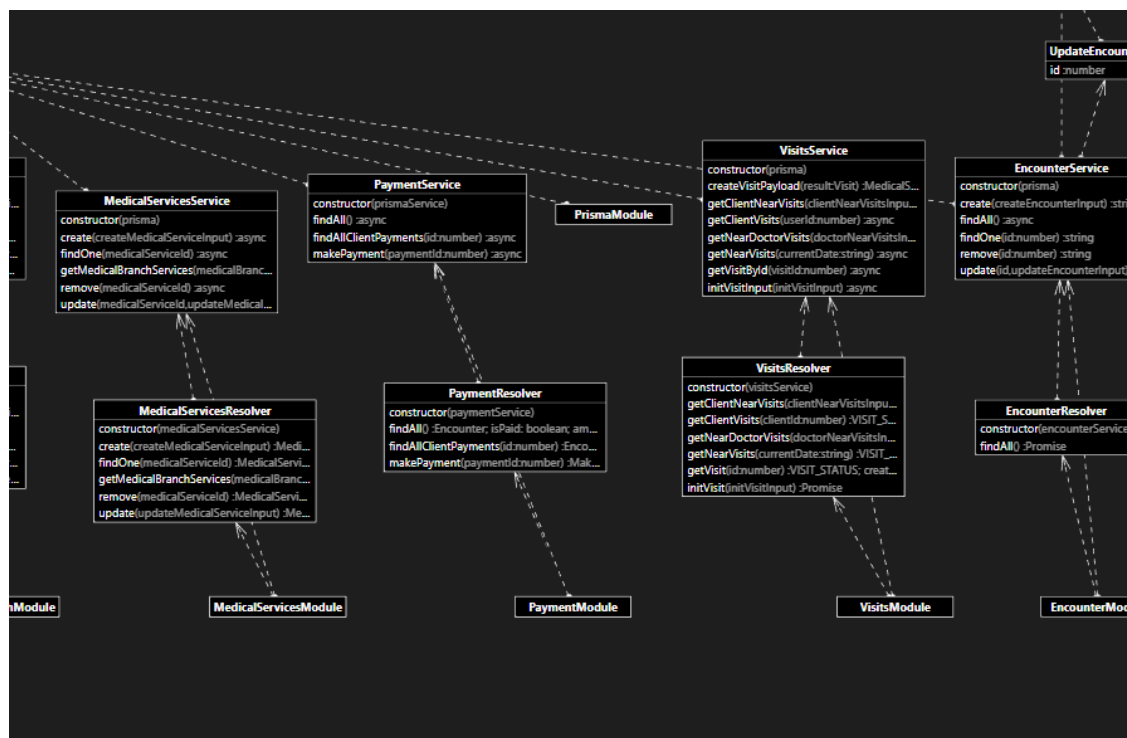


Рис. 2.6 – Діаграма класів Back-end частини проєкту

У внутрішньому механізмі Nestjs використовується шаблон па “Одинак” та “Впровадження залежностей”

Основна мета шаблону Singleton - забезпечити, щоб клас мав лише один екземпляр, з якого можна отримати доступ з будь-якого місця програми. Це часто використовується, коли потрібна глобальна точка доступу до певних ресурсів або коли об'єкт потрібен для координації дій в системі [1].

Основна ідея шаблону Впровадження залежностей полягає в тому, що залежності, необхідні для роботи компонента, надходять зовні, замість того, щоб бути створеними або знайденими в самому компоненті. Це забезпечує більшу гнучкість, можливості тестування та легшу зміну поведінки компонента, оскільки залежності можуть бути легко замінені без необхідності вносити зміни до самого компонента.

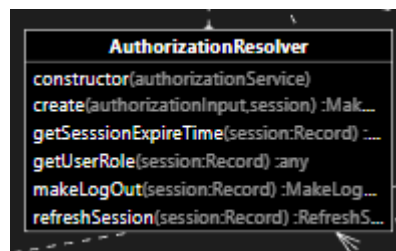


Рис. 2.7 – Діаграма класу AuthorizationResolver

Клас AuthorizationResolver процес авторизації :

- getUserRole () – отримання ролі користувача;
- getSessionExpireTime () – отримання часу завершення сесії;
- create () – запускає процес авторизації;
- makeLogOut () – викликає вихід з облікового запису.
- refreshSession () – перезапуск сесії активного користувача;

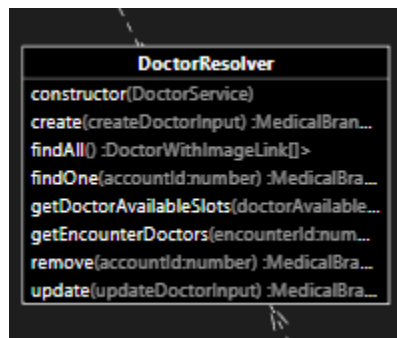


Рис. 2.8 – Діаграма класу DoctorResolver

Клас DoctorResolver відповідає за процес взаємодії з даними лікарів:

- create () – викликається при ініціалізації лікаря;
- findAll () – викликається для отримання списку лікарів;
- findOne () – викликається для лікаря по ідентифікатору;
- update () – викликається при зміні даних про лікаря
- remove () – викликається у процесі видалення лікаря.
- getEncounterDoctors () – отримання списку лікарів певного медичного закладу.
- getDoctorAvailableSlots () – отримання доступних годин прийому лікаря

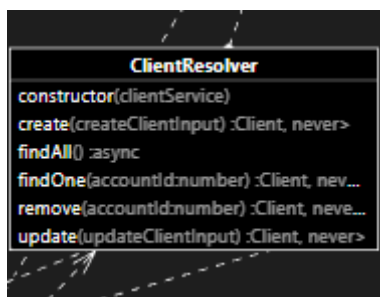


Рис. 2.9 – Діаграма класу ClientResolver

Клас ClientResolver відповідає за взаємодію з даними клієнта:

- create () – створює нового користувача;
- findAll () – отримання списку клієнтів;
- findOne () – викликається для отримання даних клієнта по ідентифікатору;
- update () – викликається для зміни даних про клієнта.
- remove () – викликається при видаленні клієнта

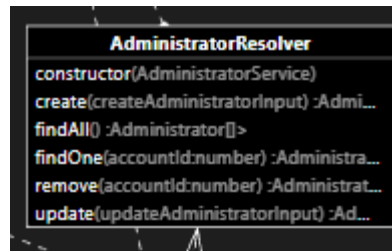


Рис. 2.10 – Діаграма класу AdministratorResolver

Клас AdministratorResolver відповідає за взаємодію з даними адміністратора:

- create () – створює нового адміністратора;
- findAll () – отримання списку адміністраторів;
- findOne () – викликається для отримання даних адміністратора по ідентифікатору;
- update () – викликається для зміни даних про адміністратора.
- remove () – викликається при видаленні адміністратора.

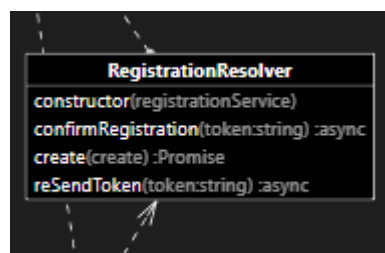


Рис. 2.11 – Діаграма класу RegistrationResolver

Клас RegistrationResolver відповідає за реєстрацію клієнта:

- create () – реєстрація нового клієнта;
- confirmRegistration () – підтвердження реєстрації;
- reSendToken () – створення повторного підтвердження реєстрації для користувача;

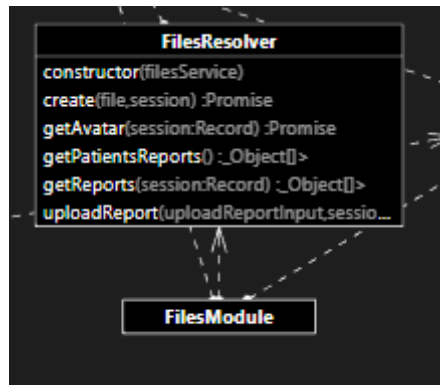


Рис. 2.12 – Діаграма класу FilesResolver

Клас FilesResolver відповідає за роботу з файлами:

- uploadAvatar () – запит на завантаження зображення користувача;
- getAvatar () – запит на отримання зображення користувача
- uploadReport () – запит на завантаження звіту для користувача
- getReports () – запит на отримання звітів користувача
- getPatientsReports () – запит на отримання звітів

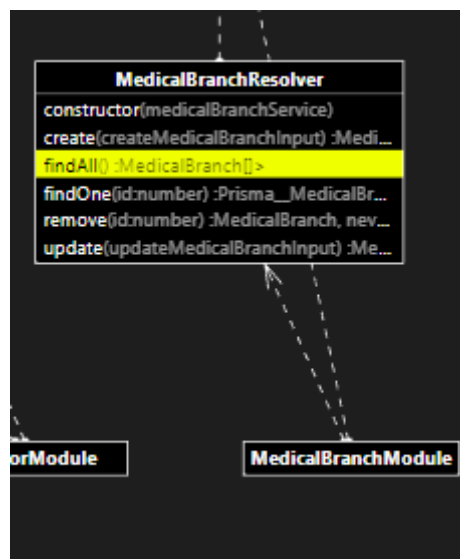


Рис. 2.13 – Діаграма класу MedicalBranchResolver

Клас MedicalBranchResolver роботу з медичними напрямками :

- create () – створення медичної гілки;
- findAll () – отримання медичних гілок;
- findOne () – отримання медичної гілки по ідентифікатору;
- update () – викликається для зміни даних про медичну гілку.
- Remove () – видалення медичної гілки

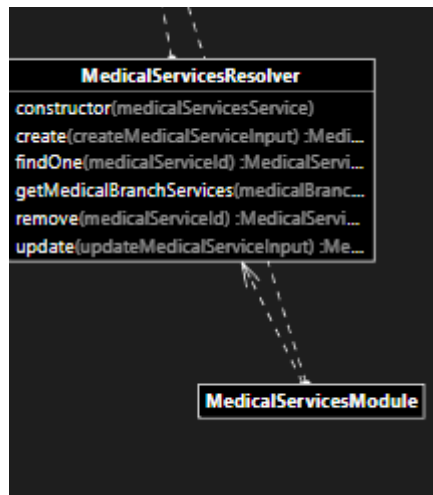


Рис. 2.14 – Діаграма класу MedicalServices

Клас MedicalServices відповідає за контроль поведінки ворожого персонажа в грі:

- getMedicalBranchServices () – отримання послуг медичних гілок
- findOne () – отримання послуги медичної гілки по ідентифікатору;
- create () – створення медичної послуги;
- update () – зміна інформації про медичну послугу
- remove () – видалення медичної послуги

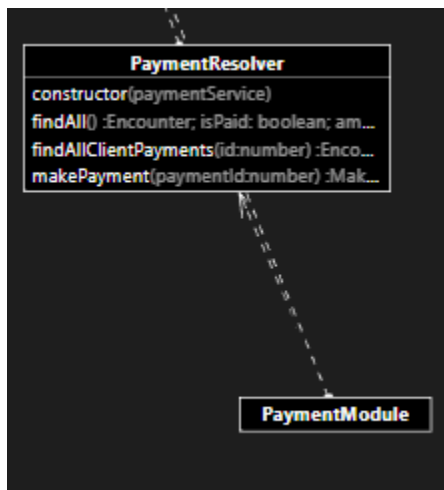


Рис. 2.15 – Діаграма класу PaymentResolver

Клас PaymentResolver роботу з оплатою послуг

- `findAll ()` – отримання всіх чеків;
- `findAllClientPayments ()` – отримання чеків клієнта по ідентифікатору
- `makePayment ()` – здійснити оплату



Рис. 2.16 – Діаграма класу VisitsResolver

Клас VisitsResolver відповідає за роботу з візитами:

- `initVisit ()` – викликається при старті гри або при активації об'єкта.
- `getClientVisits ()` – отримання візитів клієнта
- `getClientNearVisits ()` – отримання майбутніх візитів клієнта
- `getNearVisits ()` – отримання майбутніх візитів

- getNearDoctorVisits () – отримання майбутніх візитів лікаря
- getVisit () – отримання візитів

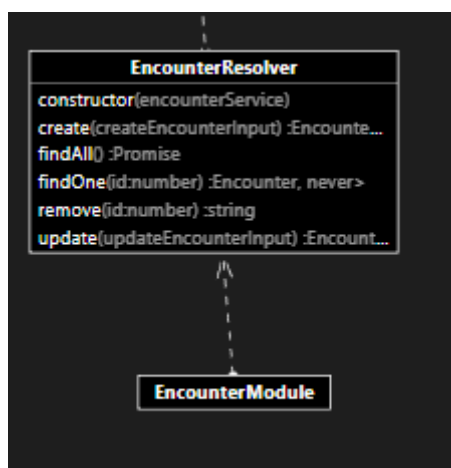


Рис. 2.17 – Діаграма класу EncounterResolver

Клас EncounterResolver відповідає за контроль головного меню гри:

- findAll () – викликається для отримання списку медичного закладу;
- create () – викликається при створенні медичного закладу
- findOne () – знайти медичний заклад по ідентифікатору
- update () – викликається при зміні даних про медичний заклад
- remove () – видалення медичного закладу

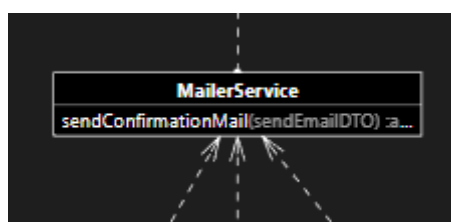


Рис. 2.18 – Діаграма класу MailService

Клас MailService відповідає за роботу з поштою:

- sendConfirmationMail () – відправляє лист підтвердження реєстрації;

2.2. Проєктування та реалізація алгоритмів роботи системи

Додаток поділено на дві частини, користувацький інтерфейс та серверна частина

Основним модулем системи є серверна частина яка відповідає за бізнес-логіку додатку:

- реєстрація
- авторизація
- робота з акаунтами
- робота з медичними закладами
- робота з медичними направленнями
- робота з візитами
- робота з оплатою

При реалізації системи було найбільш точно реалізовано процес керування гравцем. Аналогічні дії виконуються і для реальних об'єктів.

Для візуалізації процесів було створено діаграми активностей.

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33

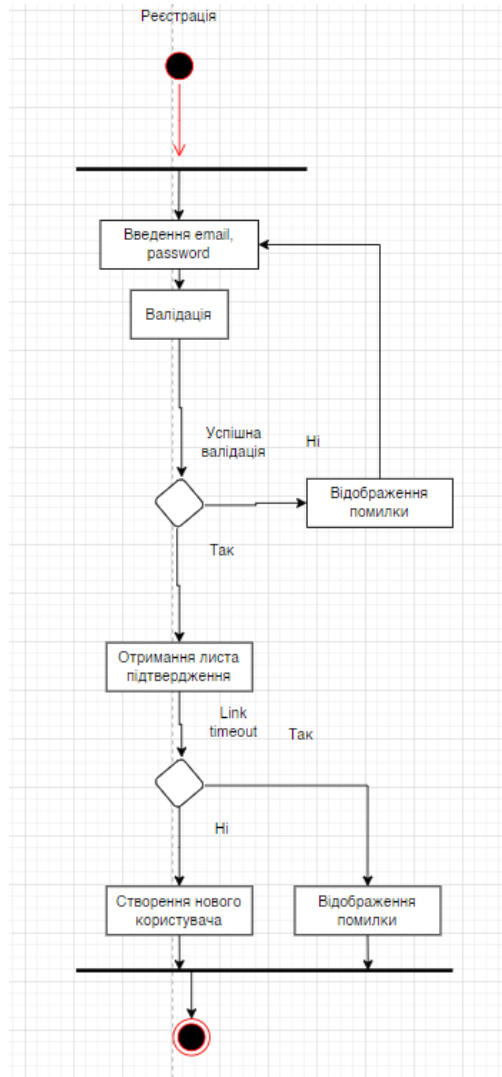


Рис. 2.19 – Діаграма активності реєстрації

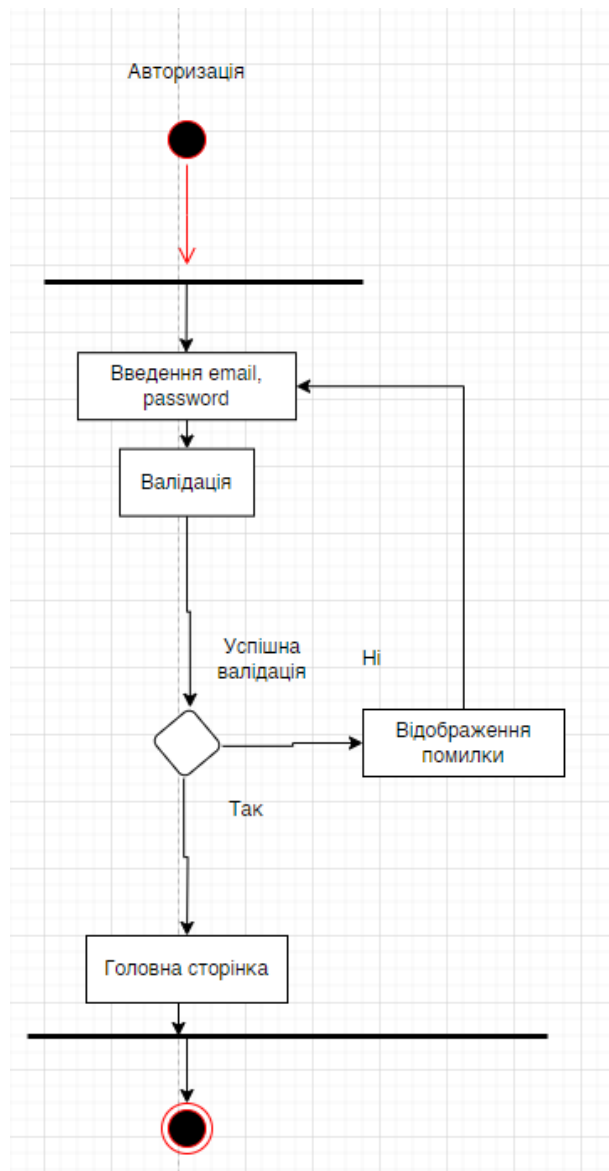


Рис. 2.20 – Діаграма активності авторизація

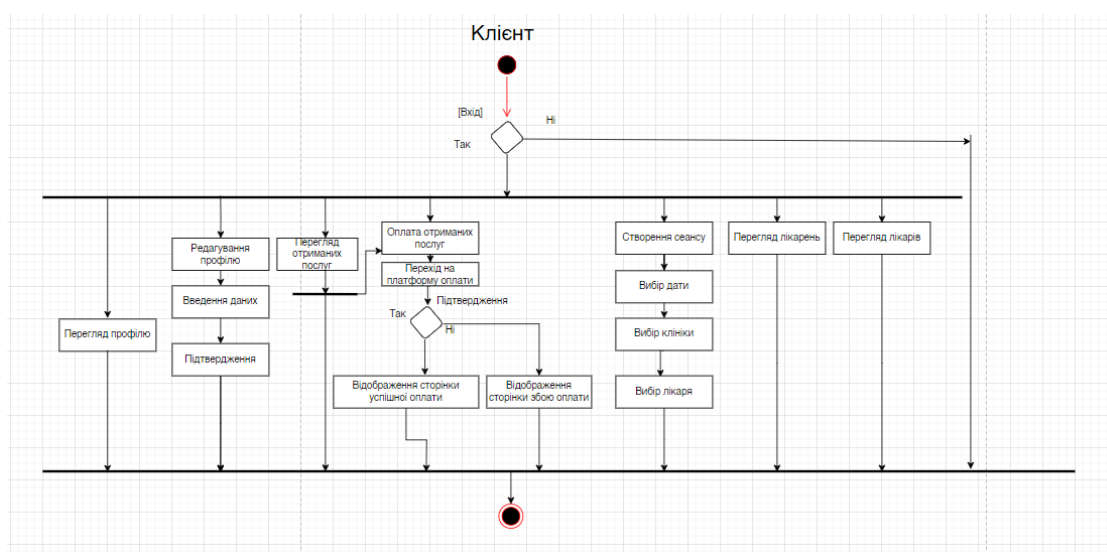


Рис. 2.21 – Діаграма активності клієнтів

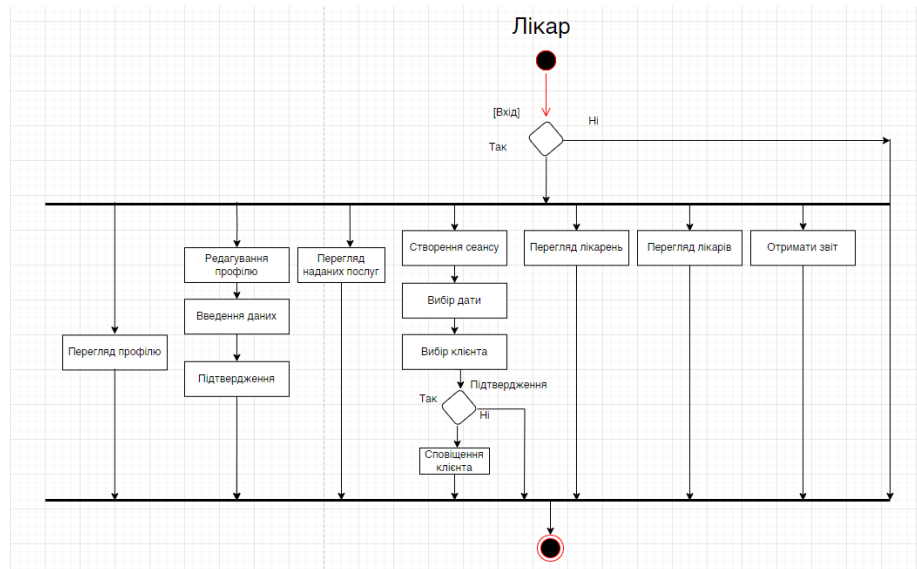


Рис. 2.22 – Діаграма активності лікарів

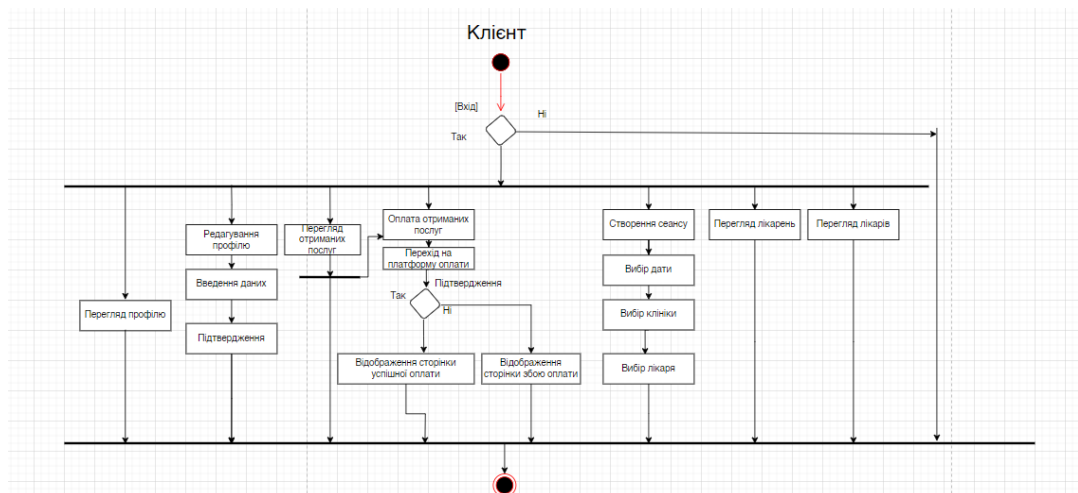


Рис. 2.23 – Діаграма активності клієнта

2.3. Реалізація функціонала додатку

Додаток автоматизації роботи медичних закладів повинен мати модель поведінки для взаємодії клієнта та лікаря. Розглянемо функціонал для створення лікарського сеансу.

Розглянемо компонент створення лікарського сеансу:

```
export function DoctorRequestPage() {
  const currentUserQuery = useQuery<{ currentUser: CurrentUserType }>(CURRENT_USER);
  const requestCard = useRef<HTMLDivElement>(null);
  const [selectedDate, setSelectedDate] = useState<moment.Moment>();
  const [selecteSlot, setSelectedSlot] = useState<string>();
  const [activeDoctor, setActiveDoctor] = useState<DoctorWithImageLink>();
  const [activeEncounter, setActiveEncounter] = useState<EncounterWithImage>();
  const dateCalendarRef = useRef<HTMLDivElement>(null);
  const [activeMedicalService, setActiveMedicalService] = useState<MedicalService>();

  const handleDateChange = (newValue: moment.Moment | null) => newValue && setSelectedDate(newValue);

  const selectSlot = (slot: string) => () => setSelectedSlot(slot);

  const validRequest = () => !!selectedDate && !!selecteSlot && !!activeDoctor && !!activeEncounter;

  const availableDoctorsSlots = useQuery<{ GetDoctorAvailableSlots: string[] }>(DOCTOR_AVAILABLE_SLOTS, { ...
  });

  const [initVisit] = useMutation<{ InitVisit: InitVisitResponse }>(INIT_VISITS);

  const createRequest = async () => { ...
  }

  return ( ...
  )
}
```

Рис. 2.24 – Компонент створення візиту

Процес створення візиту на стороні клієнта включає відображення сторінки створення візиту, введення даних та підтвердження, запиту на сторону сервера, сервер додає новий візит і відправляє результат, клієнт потрапляє на головну сторінку де відображається список найближчих візитів з новим візитом

Для відправлення запиту використовується GraphQL мутацію

```
import { gql } from "@apollo/client";

export default gql`
  mutation InitVisit($initVisitInput: InitVisitInput!) {
    InitVisit(initVisitInput: $initVisitInput)
  }
`;
```

Рис. 2.25 – Мутація на створення візиту

Після запиту виконується роут в якому створюється новий візит

```
@Resolver('Visit')
export class VisitsResolver {
  constructor(private readonly visitsService: VisitsService) {}

  @Mutation('InitVisit')
  initVisit(@Args('initVisitInput') initVisitInput: InitVisitInput) {
    return this.visitsService.initVisitInput(initVisitInput);
  }
}
```

Рис. 2.26 – Роут створення візиту

Викликається сервісна функція створення користувача

```
constructor(private readonly prisma: PrismaService) {}

async initVisitInput(initVisitInput: InitVisitInput) {
  try {
    await this.prisma.clientsDoctorsVisits.create({
      data: {
        medicalServiceId: initVisitInput.medicalServiceId,
        doctorId: initVisitInput.doctorId,
        clientId: initVisitInput.clientId,
        visitDate: new Date(`${initVisitInput.visitDate} ${initVisitInput.visitTime}`),
        visitStatus: VISIT_STATUS['PENDING'],
        createdAt: new Date(),
        updatedAt: new Date(),
      },
    });

    return InitVisitResponse['SUCCESS'];
  } catch (err) {
    return err;
  }
}
```

Рис. 2.27 – Сервісна функція створення візиту

```

type Visit {
  id: Int!
  client: Client!
  doctor: Doctor!
  encounter: Encounter!
  medicalService: MedicalService
  visitDate: String!
  visitDescription: String
  visitResult: String
  visitStatus: VISIT_STATUS!
  createdAt: String!
  updatedAt: String!
  isPaid: Boolean!
}

```

Рис. 2.28 – Модель GraphQL візиту

```

st > src > modules > visits > visits.graphql
}

type Query {
  visits: [Visit]
  visit(id: Int!): Visit
}

input InitVisitInput {
  clientId: Int!
  doctorId: Int!
  visitDate: String!
  visitTime: String!
  medicalServiceId: Int!
}

enum InitVisitResponse {
  SUCCESS
}

input ClientNearVisitsInput {
  clientId: Int!
  currentDate: String!
}

input DoctorNearVisitsInput {
  doctorId: Int!
  currentDate: String!
}

type Query {
  Visit(id: Int!): Visit
  ClientVisits(clientId: Int!): [Visit]
  ClientNearVisits(clientNearVisitsInput: ClientNearVisitsInput!): [Visit]
  NearVisits(currentDate: String!): [Visit]
  NearDoctorVisits(doctorNearVisitsInput: DoctorNearVisitsInput!): [Visit]
}

type Mutation {
  InitVisit(initVisitInput: InitVisitInput!): InitVisitResponse
}

```

Рис. 2.29 – GraphQL Запити та мутації візитів

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

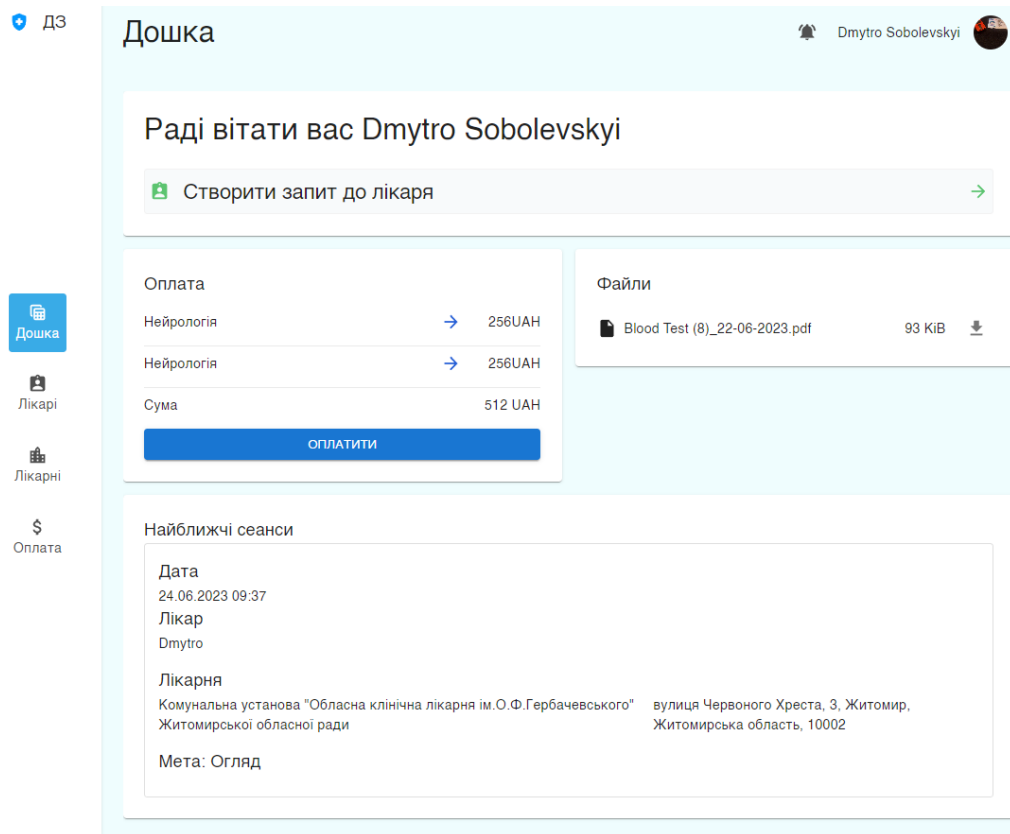


Рис. 2.30 – Відображення результату

Користувач отримує відображення нового візиту на головній сторінці

Висновки до розділу 2

Додаток реалізує процес взаємодії між клієнтом, лікарем та лікарнею, взаємодія ця створюється за допомогою клієнт-серверної архітектури додатку.

Було розглянуто алгоритми основних процесів системи та проаналізовано взаємодію класів системи, діаграми активностей.

В останньому пункті наводиться процес створення візиту.

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		40

РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Архітектура додатку та його розгортання.

Для відображення архітектури додатку було реалізовано діаграму компонентів та діаграму розгортання.

Діаграма компонентів (Component Diagram) є одним з типів структурних діаграм UML і використовується для візуалізації структури системи, її компонентів та зв'язків між ними. Діаграма компонентів дозволяє зосередитись на фізичній архітектурі системи, а також показати взаємодію компонентів [12].

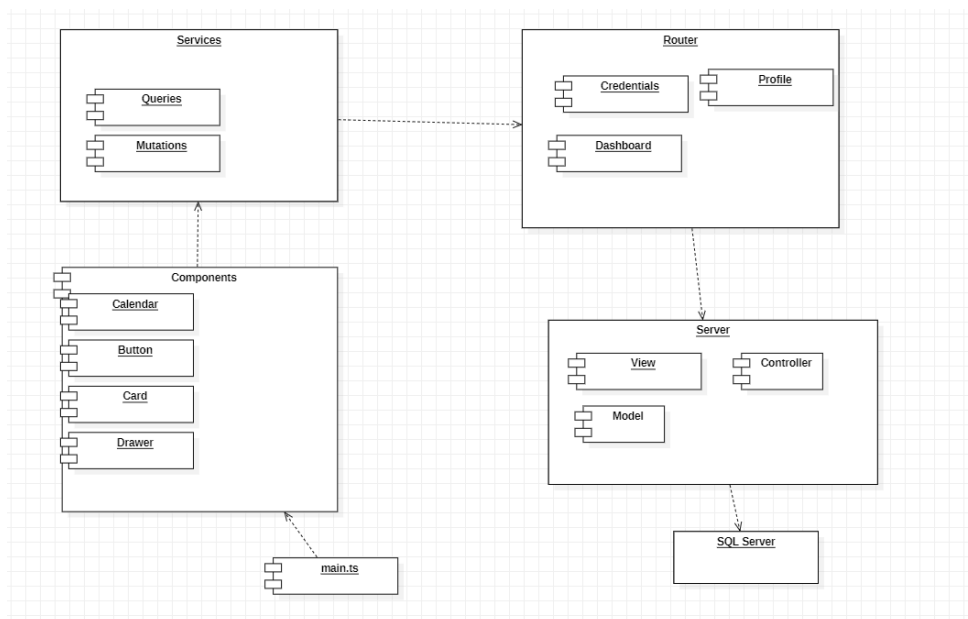


Рис. 3.1 – Діаграма компонентів

Діаграма розгортання (Deployment Diagram) є одним з типів структурних діаграм UML і використовується для візуалізації фізичного розміщення компонентів системи, архітектурного розгортання програмного забезпечення та залежностей між ними. Діаграма розгортання дозволяє показати, як компоненти програмного забезпечення розташовані на апаратному обладнанні або середовищі виконання [12].

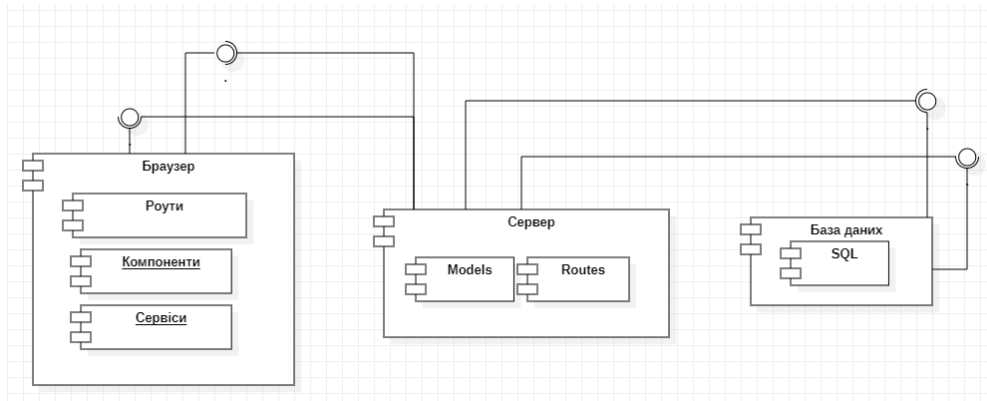


Рис. 3.2 – Діаграма Розгортання

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з додатком.

Під час першого візиту користувач потрапляє на сторінку авторизації (рис. 3.1).

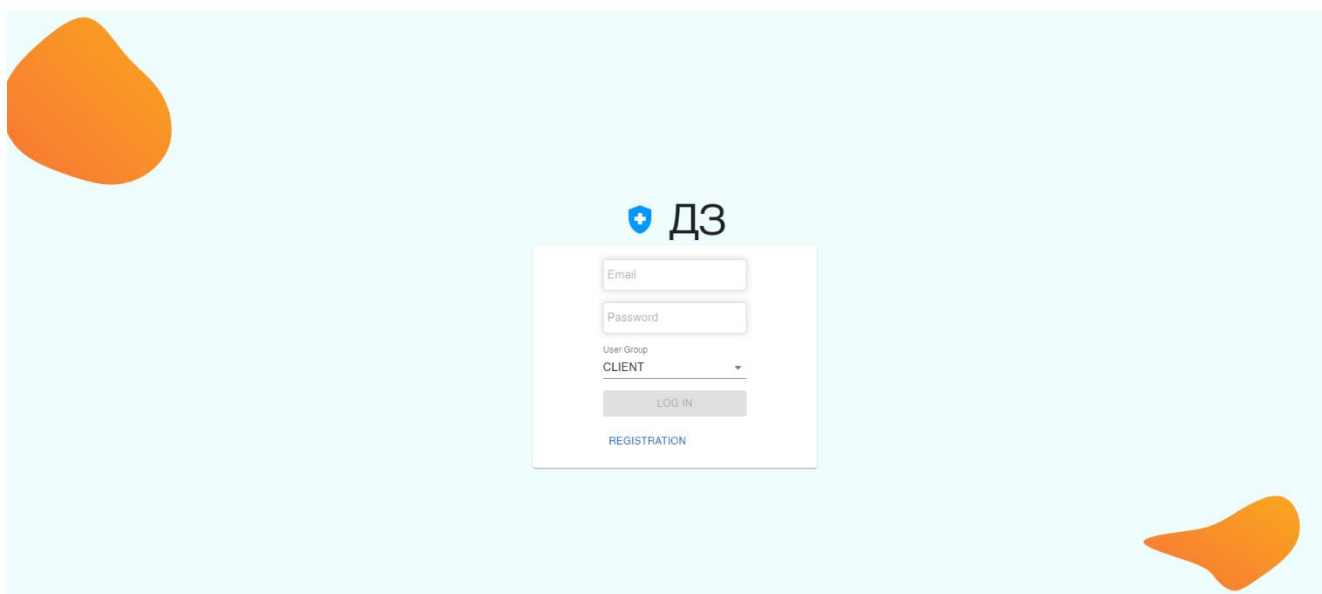


Рис. 3.3 – Форма авторизації

Тут користувач має можливість вибрати групу до якої він належить, ввести дані та виконати авторизацію. Також користувач може перейти на сторінку реєстрації

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

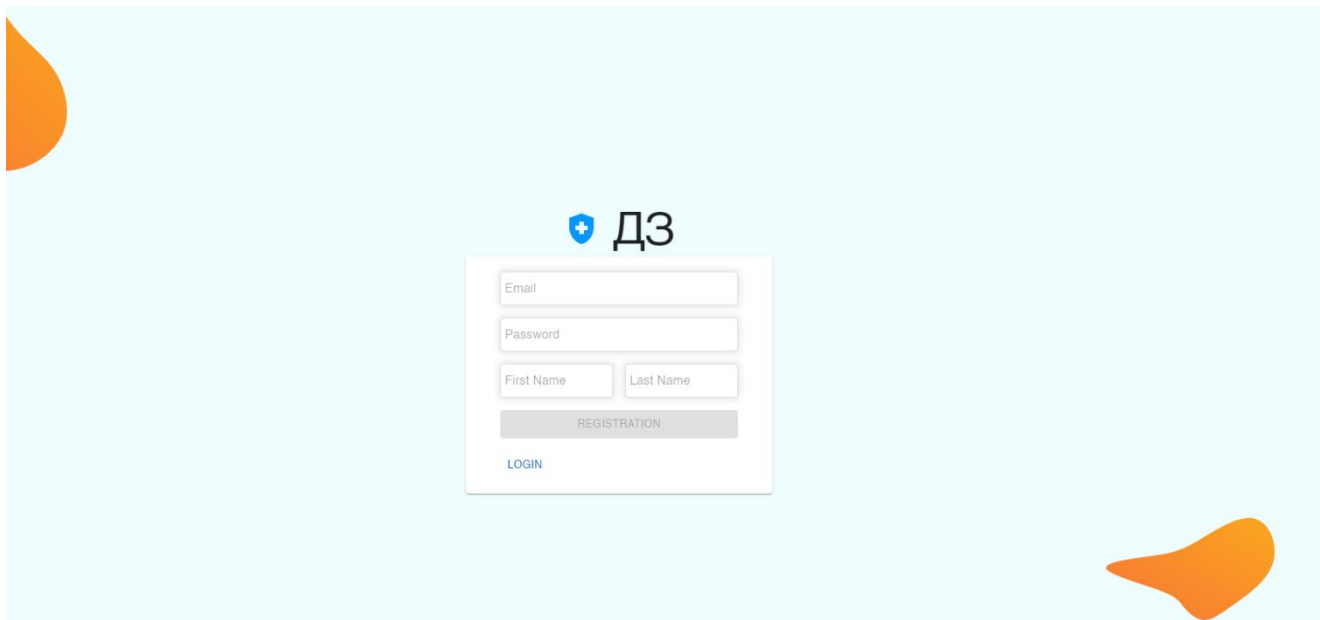


Рис. 3.4 – Сторінка реєстрації

Після авторизації користувач потрапляє на головну сторінку, де може створити візит, переглянути лікарів та лікарні, свої візити, перейти на сторінку особистого кабінету користувача

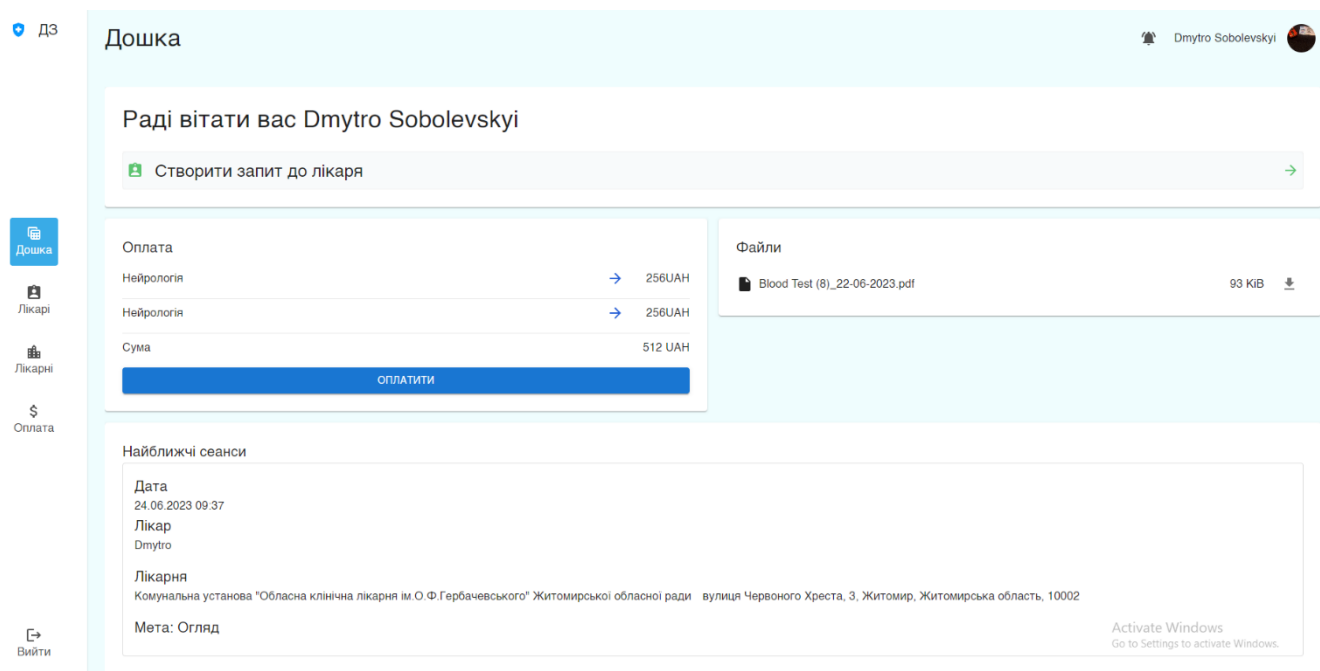


Рис. 3.5 – Головна сторінка

Після переходу на сторінку особистого кабінету користувач може дізнатися більш детальну інформацію про свій обліковий запис

		Соболєвський Д.А			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

←



ЗАВАНТАЖИТИ

Ім'я

Dmytro

Прізвище

Sobolevskyi

Пошта

dimulka62@gmail.com

Юзер Група

CLIENT

Створено

2023-06-23 06:06

Відредаговано

2023-06-23 06:06

Рис. 3.6 – Сторінка профілю

Сторінка створення візиту дозволяє користувачу вибрати лікарню, лікаря, медичну послугу та час на який він хоче отримати цю послугу.

ДЗ

Дошка

Dmytro Sobolevskyi

Відвідати лікаря

Лікарня

Комунальна установа

"Обласна клінічна лікарня

ім. О.Ф.Гербачевського"

Житомирської обласної

ради

Лікар

Dmytro

Нейрологія

Огляд

червень 2023

<

>

П

В

С

Ч

П

С

Н

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

09:37

СТВОРИТИ ЗАПИТ

Рис. 3.7 – Сторінка створення візиту

Після створення сеансу клієнт також може перейти на сторінку сеансу для перегляду інформації

		Соболєвський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		44

Дошка

Dmytro Sobolevskyi

Зустріч

Лікар
Dmytro
Нейрологія

Пацієнт

Комунальна установа "Обласна клінічна лікарня ім.О.Ф.Гербачевського" Житомирської обласної ради
Огляд

Рис. 3.8 – Сторінка візиту

Користувач може перейти на сторінку лікарів для перегляду інформації про них

ДЗ

Дошка

Dmytro Sobolevskyi

Лікарі

Dmytro
dmytrosobolevskyi@gmail.com

Нейрологія

Комунальна установа "Обласна клінічна лікарня ім.О.Ф.Гербачевського" Житомирської обласної ради

Дошка

Лікарі

Лікарні

Оплата

Рис. 3.9 – Сторінка перегляду лікарів

Користувач може перейти на сторінку лікарень для перегляду інформації про них

ДЗ

Дошка

Dmytro Sobolevskyi

Лікарні

Комунальна установа "Обласна клінічна лікарня ім.О.Ф.Гербачевського" Житомирської обласної ради

вулиця Червоного Хреста, 3, Житомир, Житомирська область, 10002

Дошка

Лікарі

Лікарні

Оплата

Рис. 3.10 – Сторінка перегляду лікарень

		Соболевський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		45

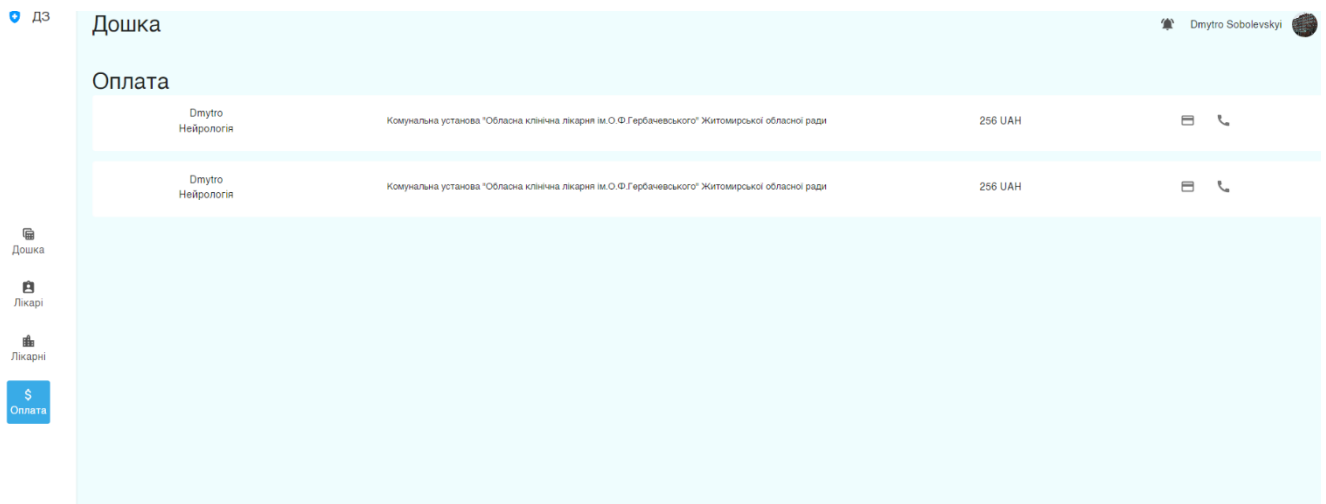


Рис. 3.11 – Сторінка оплат за послуги

Лікар має можливість переглядати список пацієнтів

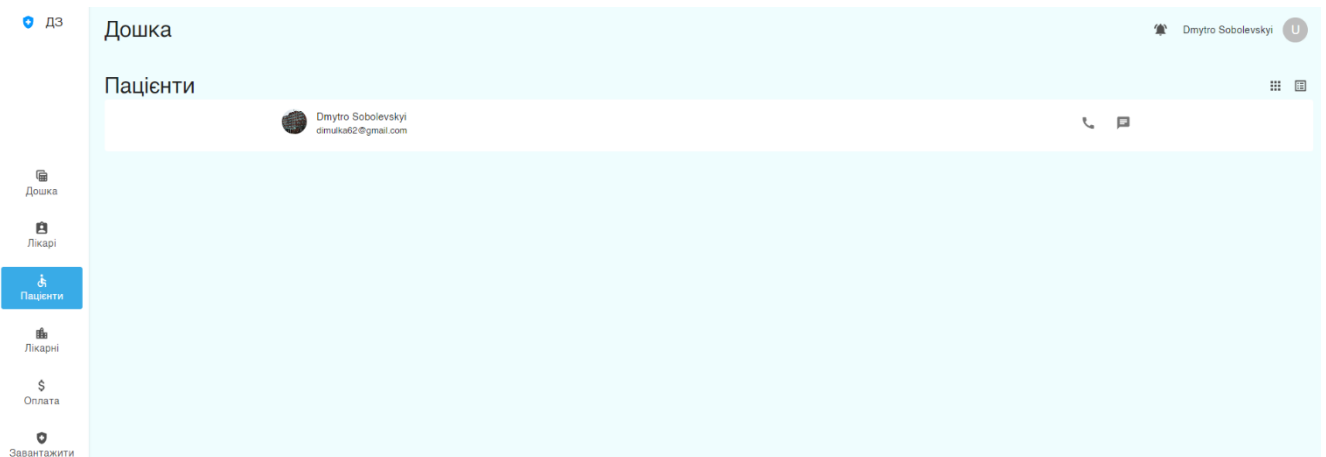


Рис. 3.12 – Сторінка пацієнтів

Лікар може здійснювати відправлення файлів користувачам.

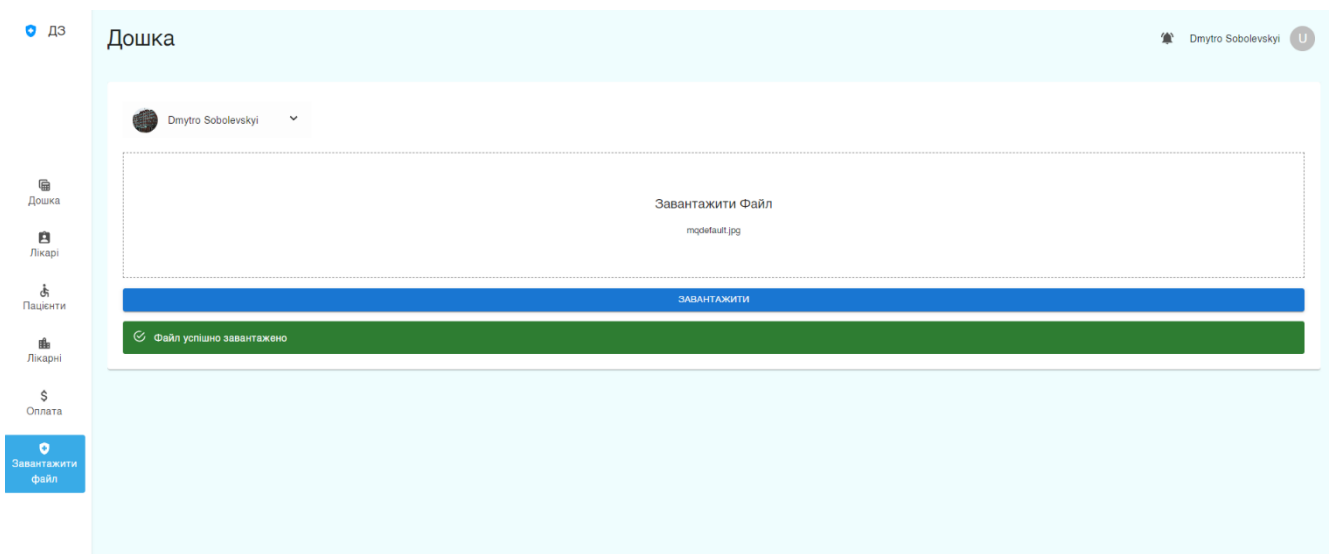


Рис. 3.13 – Успішне відправлення файлу

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

ДЗ

Дошка

Дмитро Соболевський

Раді вітати вас Dmytro Sobolevskyi

Створити запит до лікаря

Дошка

Лікарі

Лікарні

Оплата

Оплата

Нейрологія → 256UAH

Нейрологія → 256UAH

Сума 512 UAH

ОПЛАТИТИ

Файли

Blood Test (8)_22-06-2023.pdf 93 KiB

mqdefault_23-06-2023.jpg 3.2 KiB

Найближчі сеанси

Дата

24.06.2023 09:37

Лікар

Dmytro

Лікарня

Комунальна установа "Обласна клінічна лікарня ім. О.Ф.Гербачевського" Житомирської обласної ради вулиця Червоного Хреста, 3, Житомир, Житомирська область, 10002

Мета: Огляд

Activate Windows

Рис. 3.14 – Отриманий файл

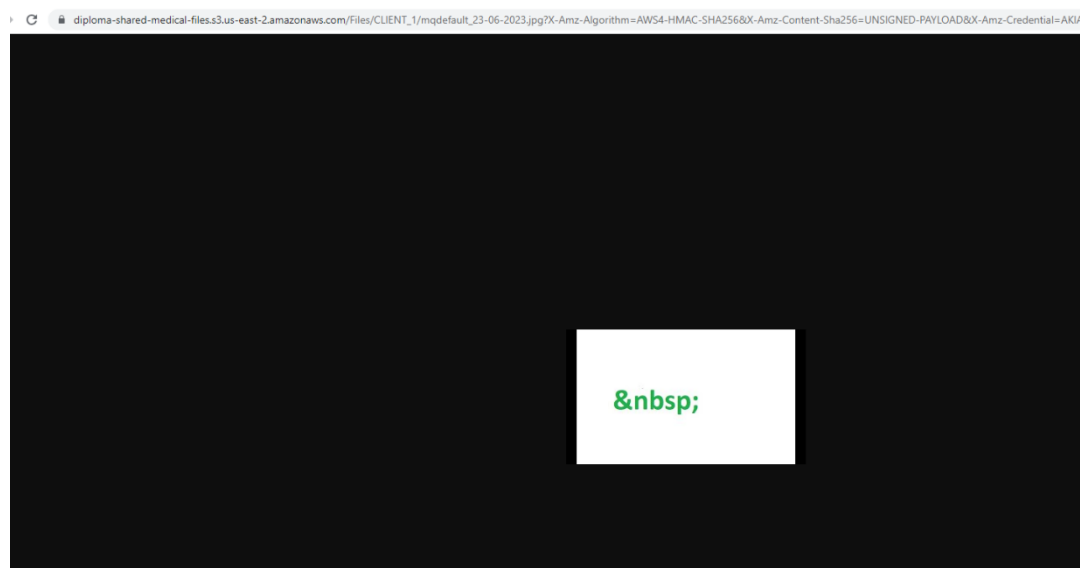


Рис. 3.15 – Результат перегляду

Користувач може виконувати оплату

		Соболевський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

QR-code for payment



Use Privat24

Test mode

To pay: 256.00 UAH

Doctor: Dmytro, Комунальна установа "Обласна клінічна лікарня ім.О.Ф.Гербачевського" Житомирської обласної ради

Pay with Privat24

24 Pay

Pay

VISA **** 6162

Card

Another method

Card number

**** * * * *

Expiration CVV2

MM/YY * * *

By clicking the «Pay» button, you confirm that you are familiar with the list of information about the service and accept the terms of the [Terms and conditions](#)

Pay

 [Help online](#)
 [3700 free from cell phones](#)

Рис. 3.16 – Результат перегляду

3.3. Тестування роботи додатку

Під час розробки додатку виконувалися тестування як окремої функціональності та і взаємодія між модулями.

При тестуванні використано такий підхід як мануальне тестування.

Мануальне тестування є процесом, в якому людина вручну перевіряє функціональність, якість та відповідність програмного продукту вимогам та очікуванням [15].

		Соболєвський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		48

Таблиця 3.1

Тестування функціонала форми авторизації

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Форма не відправляється з пустими полями	Форма не відправляється з пустими полями	Пройдено
2	Форма не відправляється доки формат введення не буде правильним	Форма не відправляється доки формат введення не буде правильним	Пройдено
3	Якщо пошта чи пароль введені не правильно виводиться повідомлення про це	Якщо пошта чи пароль введені не правильно виводиться повідомлення про це	Пройдено

Таблиця 3.2

Тестування функціонала форми авторизації

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Форма не відправляється з пустими полями	Форма не відправляється з пустими полями	Пройдено
2	Форма не відправляється доки формат введення не буде правильним	Форма не відправляється доки формат введення не буде правильним	Пройдено
3	Якщо обліковий запис вже існує виводиться помилка про це	Якщо обліковий запис вже існує виводиться помилка про це	Пройдено

Таблиця 3.3

Тестування функціонала створення сеансу

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Неможливо обрати лікаря поки не обереш лікарню	Неможливо обрати лікаря поки не обереш лікарню	Пройдено
2	Неможливо обрати послугу поки не обереш лікаря	Неможливо обрати послугу поки не обереш лікаря	Пройдено
3	Неможливо обрати дату візиту поки не обереш лікарню, лікаря та послугу	Неможливо обрати дату візиту поки не обереш лікарню, лікаря та послугу	Пройдено
4	Після створення візиту відбувається редирект на головну сторінку	Після створення візиту відбувається редирект на головну сторінку	Пройдено

Таблиця 3.4

Тестування роботи з доступними файлами

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Можливо завантажити файл при натисканні кнопки завантажити	Можливо завантажити файл при натисканні кнопки завантажити	Пройдено

Таблиця 3.5

Тестування форми завантаження файлу

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Неможливо відправити файл поки	Неможливо відправити файл поки	Пройдено

	не додано файл та не вибрано користувача	не додано файл та не вибрано користувача	
--	---	---	--

Таблиця 3.6

Тестування захищеності шляхів та функціоналу сайту

Тест №	Очікуваний результат	Фактичний результат	Пройдено/не пройдено
1	Зайти на головну сторінку якщо не авторизований	Зайти на головну сторінку якщо не авторизований	Пройдено
2	Клієнти не мають доступу до сторінки пацієнтів	Клієнти не мають доступу до сторінки пацієнтів	Пройдено
3	Клієнти не мають доступу до сторінки завантаження файлу	Клієнти не мають доступу до сторінки завантаження файлу	Пройдено

Висновки до розділу 3

В даному розділі було розроблена компонентна структура та розгортання додатку.

Реалізована система симуляції містить функції авторизації, реєстрації, створення візитів оплати, перегляд даних

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано систему автоматизації роботи медичних закладів:

1. Оглянувши аналогічні продукти в заданій предметній області виявив суперечності між можливостями переглянутих продуктів та необхідним функціоналом, визначив що головним функціоналом додатку має бути: робота з візитами, платежами, наявність зручного UI, перегляд інформації про візити, оплати, лікарів, лікарень, комунікація між клієнтами, лікарнями, лікарями. Вибраний стек технологій визначивши архітектуру додатку. В якості архітектури додатку було обрано клієнт-серверну архітектуру та MVC та MVVM шаблони проєктування, обрано необхідний та достатній набір програмних продуктів для реалізації поставлених задач: React, NestJS, GraphQL, Typescript, PostgreSQL, Prisma.

2. Було проаналізовано функціональні та нефункціональні вимоги додатку, для чого побудовано діаграму варіантів використання, розроблено діаграму класів, компонентів та розгортання.

3. Розглянуто алгоритми процесів системи та проаналізовано взаємодію класів системи та їх взаємодії.

4. Було наведено детальний розгляд основних класів додатку.

5. Було проведено мануальне тестування додатку. Виправлено помилки.

Система працює коректно

Створений програмний комплекс готовий до використання.

Надалі дана система може бути розширена шляхом впровадження нових функціональностей особистого профілю, живої інтеракції з лікарями, техпідтримки. Завдяки обраній архітектурі розширення системи відбуватиметься значно швидше.

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Чистий код: Посібник з майстерності гнучкого програмного забезпечення", Роберт К. Мартін, Prentice Hall (2008) – 441 с.
2. Learning GraphQL: Declarative Data Fetching for Modern Web Apps, Porcello, Eve; Banks, Alex – 200 с.
3. Документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn>
4. Документація NestJS [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/first-steps>
5. Документація Typescript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
6. Документація Prisma [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io/docs>
7. Документація GraphQL [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/learn/>
8. Документація Apollo [Електронний ресурс] – Режим доступу до ресурсу : <https://www.apollographql.com/docs/>
9. Designing Web APIs: Building APIs That Developers Love 1st Edition, Amir Shevat, Brenda Jin, Saurabh Sahni – 232 с.
10. "Чиста архітектура", Роберт К. Мартін – 254 с.
11. Фрімен Е. Патерни проєктування / Е. Фрімен, Е. Фрімен. – 2011. – 656 с.
12. "Learning UML", Sinan Si Alhir – 304 с.
13. "Design Patterns in ABAP Objects", Kerem Koseoglu – 387 с.
14. "The Art of Software Testing", Glenford J. Myers, Corey Sandler, Tom Badgett – 256 с.
15. Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites 1st Edition, Micah Godbolt – 198 с.

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53

ДОДАТКИ

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		54

Клас AppModule

```

import { AppController } from './app.controller';
import { AppService } from './app.service';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloDriver } from '@nestjs/apollo';
import { Module } from '@nestjs/common';
import { join } from 'path';
import { ApolloServerPluginLandingPageLocalDefault } from
'@apollo/server/plugin/landingPage/default';
import { PrismaModule } from './modules/prisma/prisma.module';
import { PrismaService } from './modules/prisma/prisma.service';
import { MedicalBranchModule } from './modules/medical-branch/medical-branch.module';
import { ClientModule } from './modules/client/client.module';
import { DoctorModule } from './modules/doctor/doctor.module';
import { AdministratorModule } from './modules/administrator/administrator.module';
import { EncounterModule } from './modules/encounter/encounter.module';
import { VisitsModule } from './modules/visits/visits.module';
import { RegistrationModule } from './modules/registration/registration.module';
import { SessionModule } from 'nestjs-session';
import * as expressSession from 'express-session';
import * as expressPostgresSession from 'connect-pg-simple';
import { Pool } from 'pg';
import { AuthorizationModule } from './modules/authorization/authorization.module';
import { FilesModule } from './modules/files/files.module';
import { PaymentModule } from './modules/payment/payment.module';
import { MedicalServicesModule } from './modules/medical-services/medical-services.module';

const PostgresStore = expressPostgresSession(expressSession);

@Module({
  imports: [
    SessionModule.forRoot({

```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55

```

session: {
  secret: process.env.SESSION_SECRET,
  store: new PostgresStore({
    pool: new Pool({
      user: process.env.DATABASE_USER,
      host: process.env.DATABASE_HOST,
      database: process.env.DATABASE_NAME,
      password: process.env.DATABASE_PASSWORD,
      port: parseInt(process.env.DATABASE_PORT),
    }),
    tableName: 'USER_SESSIONS',
    createTableIfMissing: true,
  }),
  resave: false,
  saveUninitialized: false,
  cookie: {
    maxAge: 1000 * 60 * 60 * 1,
    httpOnly: process.env.NODE_ENV === 'production' ? false : true,
    secure: process.env.NODE_ENV === 'production' ? true : false,
    sameSite: process.env.NODE_ENV === 'production' ? 'none' : 'strict',
  },
  unset: 'destroy',
}
}),
GraphQLModule.forRoot({
  context: ({ req, res }) => ({ req, res, cors: true }),
  cors: {
    origin: process.env.CORSR_URL,
    credentials: true,
    methods: '*',
    allowedHeaders: "*"
  },
  driver: ApolloDriver,
  playground: false,
  plugins: [ApolloServerPluginLandingPageLocalDefault()],

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

```

typePaths: ['./**/*.graphql'],
definitions: {
  path: join(process.cwd(), 'src/types/graphql.ts'),
  outputAs: 'class',
},

}),
PrismaModule,
MedicalBranchModule,
ClientModule,
DoctorModule,
AdministratorModule,
EncounterModule,
VisitsModule,
RegistrationModule,
AuthorizationModule,
FilesModule,
PaymentModule,
MedicalServicesModule,
],
controllers: [AppController],
providers: [AppService, PrismaService],
})
export class AppModule {}

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

Main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as cors from 'cors';
import { graphqlUploadExpress } from 'graphql-upload';
import * as cookieParser from 'cookie-parser';

(async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.use(cookieParser());
  app.use(cors({
    origin: process.env.CORSR_URL,
    credentials: true,
  }));

  app.use(cookieParser());
  app.use(graphqlUploadExpress());

  await app.listen(process.env.PORT);
})();
```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		58

Клас VisitsResolver

```

import { Resolver, Query, Mutation, Args } from '@nestjs/graphql';
import { VisitsService } from './visits.service';
import { ClientNearVisitsInput, DoctorNearVisitsInput, InitVisitInput, USER_ROLES } from
'src/types/graphql';
import { UseGuards } from '@nestjs/common';
import { GqlSessionMiddleware } from 'src/middleware/gql-session.middleware';
import { GqlSessionRolesMiddleware } from 'src/middleware/session-role.gql.middleware';

@Resolver('Visit')
export class VisitsResolver {
  constructor(private readonly visitsService: VisitsService) {}

  @Mutation('InitVisit')
  initVisit(@Args('initVisitInput') initVisitInput: InitVisitInput) {
    return this.visitsService.initVisitInput(initVisitInput);
  }

  @Query('ClientVisits')
  getClientVisits(@Args('clientId') clientId: number) {
    return this.visitsService.getClientVisits(clientId);
  }

  @Query('ClientNearVisits')
  getClientNearVisits(@Args('clientNearVisitsInput') clientNearVisitsInput: ClientNearVisitsInput) {
    console.log("NEAR TEST");
    return this.visitsService.getClientNearVisits(clientNearVisitsInput);
  }

  @Query('NearVisits')
  @UseGuards(GqlSessionMiddleware)
  @UseGuards(new GqlSessionRolesMiddleware([USER_ROLES['DOCTOR'],
USER_ROLES['ADMIN']]))
  getNearVisits(@Args('currentDate') currentDate: string) {
    return this.visitsService.getNearVisits(currentDate);
  }

  @Query('NearDoctorVisits')
  @UseGuards(GqlSessionMiddleware)
  @UseGuards(new GqlSessionRolesMiddleware([USER_ROLES['DOCTOR'],
USER_ROLES['ADMIN']]))
  getNearDoctorVisits(@Args('doctorNearVisitsInput') doctorNearVisitsInput: DoctorNearVisitsInput)
  {
    return this.visitsService.getNearDoctorVisits(doctorNearVisitsInput);
  }

  @Query('Visit')
  getVisit(@Args('id') id: number) {
    return this.visitsService.getVisitById(id);
  }

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

```
}  
}
```

Додаток Г

Клас VisitsService

```
import { Injectable } from '@nestjs/common';  
import { ClientNearVisitsInput, DoctorNearVisitsInput, InitVisitInput, InitVisitResponse,  
VISIT_STATUS, Visit } from 'src/types/graphql';  
import { PrismaService } from '../prisma/prisma.service';  
import * as moment from 'moment';  
import { ClientsDoctorsVisits, Prisma } from '@prisma/client';
```

```
@Injectable()
```

```
export class VisitsService {
```

```
  constructor(private readonly prisma: PrismaService) {}
```

```
  async initVisitInput(initVisitInput: InitVisitInput) {
```

```
    try {
```

```
      await this.prisma.clientsDoctorsVisits.create({
```

```
        data: {
```

```
          medicalServiceId: initVisitInput.medicalServiceId,
```

```
          doctorId: initVisitInput.doctorId,
```

```
          clientId: initVisitInput.clientId,
```

```
          visitDate: new Date(`${initVisitInput.visitDate} ${initVisitInput.visitTime}`),
```

```
          visitStatus: VISIT_STATUS['PENDING'],
```

```
          createdAt: new Date(),
```

```
          updatedAt: new Date(),
```

```
        },
```

```
      });
```

```
      return InitVisitResponse['SUCCESS'];
```

```
    } catch (err) {
```

```
      return err;
```

```
    }
```

```
  }
```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		60

```

async getClientVisits(userId: number) {
  const result = await this.prisma.clientsDoctorsVisits.findMany({
    where: { clientId: userId },
    include: {
      medicalService: true,
      client: true,
      doctor: { include: { medicalBranch: true, encounter: true } },
    },
  });

  return result.map((item) => this.createVisitPayload(item as unknown as Visit));
}

```

```

async getClientNearVisits(clientNearVisitsInput: ClientNearVisitsInput) {
  const result = await this.prisma.clientsDoctorsVisits.findMany({
    where: {
      clientId: clientNearVisitsInput.clientId,
      visitDate: {
        gte: new Date(clientNearVisitsInput.currentDate),
      }
    },
    include: {
      medicalService: true,
      client: true,
      doctor: { include: { medicalBranch: true, encounter: true } },
    },
  });

  return result.map((item) => this.createVisitPayload(item as unknown as Visit));
}

```

```

async getNearVisits(currentDate: string) {
  const result = await this.prisma.clientsDoctorsVisits.findMany({
    where: {
      visitDate: { gte: new Date(currentDate) }
    }
  });
}

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		61

```

    },
    include: {
      medicalService: true,
      client: true,
      doctor: { include: { medicalBranch: true, encounter: true } },
    },
  });

  return result.map((item) => this.createVisitPayload(item as unknown as Visit));
}

async getNearDoctorVisits(doctorNearVisitsInput: DoctorNearVisitsInput) {
  const result = await this.prisma.clientsDoctorsVisits.findMany({
    where: {
      doctorId: doctorNearVisitsInput.doctorId,
      visitDate: {
        gte: new Date(doctorNearVisitsInput.currentDate),
      }
    },
    include: {
      medicalService: true,
      client: true,
      doctor: { include: { medicalBranch: true, encounter: true } },
    },
  });

  return result.map((item) => this.createVisitPayload(item as unknown as Visit));
}

async getVisitById(visitId: number) {
  const result = await this.prisma.clientsDoctorsVisits.findUnique({
    where: { id: visitId },
    include: {
      medicalService: true,
      client: true,
    },
  });

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    doctor: { include: { medicalBranch: true, encounter: true } },
  },
});

```

```

return this.createVisitPayload(result as unknown as Visit);
}

```

```

createVisitPayload(result: Visit) {
  return {
    ...result,
    id: result.id,
    client: result.client,
    doctor: result.doctor,
    encounter: result.doctor.encounter,
    isPaid: result.isPaid,
    amount: result.medicalService.price,
    medicalService: result.medicalService,
    visitDate: moment(result.visitDate).format('YYYY-MM-DD HH:mm'),
  }
}
}

```

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		63

Клас RegistrationResolver

```

import { Resolver, Mutation, Args } from '@nestjs/graphql';
import { RegistrationService } from './registration.service';
import { ConfirmRegistrationResponses, ReSendTokenResponse, RegisterClientInput,
RegisterClientResponses } from 'src/types/graphql';

@Resolver('Registration')
export class RegistrationResolver {
  constructor(private readonly registrationService: RegistrationService) {}

  @Mutation('RegisterClient')
  create(@Args('create') create: RegisterClientInput): Promise<RegisterClientResponses> {
    return this.registrationService.registerClient(create);
  }

  @Mutation('ConfirmRegistration')
  async confirmRegistration(@Args('token') token: string): Promise<ConfirmRegistrationResponses> {
    return await this.registrationService.confirmRegistration(token);
  }

  @Mutation('ReSendToken')
  async reSendToken(@Args('token') token: string): Promise<ReSendTokenResponse> {
    return await this.registrationService.reSendToken(token);
  }
}

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас RegistrationService

```

import { Injectable } from '@nestjs/common';
import { MailerService } from '../mailer/mailer.service';
import { PrismaService } from '../prisma/prisma.service';
import { ConfirmRegistrationResponses, ReSendTokenResponse, RegisterClientInput,
RegisterClientResponses } from 'src/types/graphql';
import * as jwt from 'jsonwebtoken';
import * as moment from 'moment';
import * as bcrypt from 'bcrypt';
import { AccountService } from '../account/account.service';

interface JwtPayload {
  email: string;
  password: string;
  name: string;
  expiredAt: string;
}

@Injectable()
export class RegistrationService {
  constructor(
    private readonly mailer: MailerService,
    private readonly prisma: PrismaService,
    private readonly accountService: AccountService,
  ) {}

  async registerClient(create: RegisterClientInput) {
    const account = await this.accountService.getAccount(create.email);

    if(account && account.isVerified) {
      return RegisterClientResponses['ALREADY_REGISTERED'];
    }

    const confirmationJWT = jwt.sign(

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		65

```

{
  email: create.email,
  password: await bcrypt.hash(create.password, 10),
  name: create.firstName + ' ' + create.lastName,
  expiredAt: moment(new Date().getTime()).add('30', 'minutes').toISOString(),
},
process.env.JWT_SECRET,
);

await this.mailer.sendConfirmationMail({
  email: create.email,
  jwtToken: confirmationJWT,
  firstName: create.firstName,
  lastName: create.lastName,
});

if(account?.email && !account?.isVerified) {
  return RegisterClientResponses['REPEAT_CONFIRMATION'];
}

await this.prisma.client.create({
  data: {
    email: create.email,
    password: await bcrypt.hash(create.password, 10),
    name: create.firstName + ' ' + create.lastName,
    createdAt: new Date().toISOString(),
    updatedAt: new Date().toISOString(),
    isVerified: false,
  }
});

return RegisterClientResponses['SEND_CONFIRMATION_LETTER'];
}

```

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

async confirmRegistration(jwtToken: string) {
  const decoded = jwt.verify(jwtToken, process.env.JWT_SECRET) as JwtPayload;

  if(!decoded || !decoded?.email || !decoded?.password || !decoded?.name || !decoded?.expiredAt) {
    return ConfirmRegistrationResponses['INVALID_TOKEN'];
  }

  if(moment(decoded.expiredAt).isBefore(moment(new Date().getTime())) {
    return ConfirmRegistrationResponses['TOKEN_EXPIRED'];
  }

  const account = await this.prisma.client.findUnique({
    where: { email: decoded.email }
  });

  if(!account?.email && account?.isVerified) {
    return ConfirmRegistrationResponses['EMAIL_ALREADY_REGISTERED'];
  }

  if(!account) {
    return ConfirmRegistrationResponses['EMAIL_NOT_REGISTERED'];
  }

  await this.prisma.client.update({
    where: {
      email: decoded.email,
    },
    data: {
      updatedAt: new Date().toISOString(),
      isVerified: true,
      type: 'CLIENT',
    }
  });

  return ConfirmRegistrationResponses['REGISTRATION_CONFIRMED'];
}

```

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}

async reSendToken(jwtToken: string) {
  const decoded = jwt.verify(jwtToken, process.env.JWT_SECRET) as JwtPayload;

  if(!decoded || !decoded?.email || !decoded?.password || !decoded?.name || !decoded?.expiredAt) {
    return ReSendTokenResponse['INVALID_TOKEN'];
  }

  if(!moment(decoded.expiredAt).isBefore(moment(new Date().getTime())) {
    return ReSendTokenResponse['TOKEN_NOT_EXPIRED'];
  }

  const confirmationJWT = jwt.sign(
    {
      email: decoded.email,
      password: decoded.password,
      name: decoded.name,
      expiredAt: moment(new Date().getTime()).add('30', 'minutes').toISOString(),
    },
    process.env.JWT_SECRET,
  );

  await this.mailer.sendConfirmationMail({
    email: decoded.email,
    jwtToken: confirmationJWT,
    firstName: decoded.name.split(' ')[0],
    lastName: decoded.name.split(' ')[1],
  });

  return ReSendTokenResponse['REPEAT_CONFIRMATION']
}
}

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

Клас MedicalBranchResolver

```

import { Resolver, Query, Mutation, Args } from '@nestjs/graphql';
import { MedicalBranchService } from './medical-branch.service';
import { CreateMedicalBranchInput, UpdateMedicalBranchInput } from 'src/types/graphql';

@Resolver('MedicalBranch')
export class MedicalBranchResolver {
  constructor(private readonly medicalBranchService: MedicalBranchService) {}

  @Mutation('CreateMedicalBranch')
  create(@Args('createMedicalBranchInput') createMedicalBranchInput: CreateMedicalBranchInput) {
    return this.medicalBranchService.create(createMedicalBranchInput);
  }

  @Query('medicalBranches')
  findAll() {
    return this.medicalBranchService.findAll();
  }

  @Query('medicalBranch')
  findOne(@Args('id') id: number) {
    return this.medicalBranchService.findOne(id);
  }

  @Mutation('UpdateMedicalBranch')
  update(@Args('updateMedicalBranchInput') updateMedicalBranchInput:
UpdateMedicalBranchInput) {
    return this.medicalBranchService.update(updateMedicalBranchInput.id,
updateMedicalBranchInput);
  }

  @Mutation('RemoveMedicalBranch')
  remove(@Args('id') id: number) {
    return this.medicalBranchService.remove(id);
  }

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

}

}

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас MedicalBranchServices

```

import { Injectable } from '@nestjs/common';
import { PrismaService } from 'src/modules/prisma/prisma.service';
import { CreateMedicalBranchInput, UpdateMedicalBranchInput } from 'src/types/graphql';

@Injectable()
export class MedicalBranchService {
  constructor(private prisma: PrismaService) {}

  create(createMedicalBranchInput: CreateMedicalBranchInput) {
    return this.prisma.medicalBranch.create({
      data: {
        name: createMedicalBranchInput.name,
        description: createMedicalBranchInput.description || null,
      },
    });
  }

  findAll() {
    return this.prisma.medicalBranch.findMany();
  }

  findOne(id: number) {
    return this.prisma.medicalBranch.findUnique({
      where: { id },
      select: {
        id: true,
        name: true,
        description: true,
      },
    });
  }

  update(id: number, updateMedicalBranchInput: UpdateMedicalBranchInput) {

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return this.prisma.medicalBranch.update({
  where: { id },
  data: updateMedicalBranchInput,
})
}

```

```

remove(id: number) {
  return this.prisma.medicalBranch.delete({
    where: { id },
  });
}
}

```

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас EncounterResolver

```

import { Resolver, Query, Mutation, Args } from '@nestjs/graphql';
import { EncounterService } from './encounter.service';
import { CreateEncounterInput, UpdateEncounterInput } from 'src/types/graphql';

@Resolver('Encounter')
export class EncounterResolver {
  constructor(private readonly encounterService: EncounterService) {}

  @Query('Encounters')
  findAll() {
    return this.encounterService.findAll();
  }

  @Mutation('CreateEncounter')
  create(@Args('createEncounterInput') createEncounterInput: CreateEncounterInput) {
    return this.encounterService.create(createEncounterInput);
  }

  @Query('Encounter')
  findOne(@Args('id') id: number) {
    return this.encounterService.findOne(id);
  }

  @Mutation('UpdateEncounter')
  update(@Args('updateEncounterInput') updateEncounterInput: UpdateEncounterInput) {
    return this.encounterService.update(updateEncounterInput);
  }

  @Mutation('RemoveEncounter')
  remove(@Args('id') id: number) {
    return this.encounterService.remove(id);
  }
}

```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

}

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас EncounterService

```

import { Injectable } from '@nestjs/common';
import { PrismaService } from '../prisma/prisma.service';
import { ListObjectsV2Command, GetObjectCommand, S3Client } from '@aws-sdk/client-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';
import { CreateEncounterInput, UpdateEncounterInput } from 'src/types/graphql';

@Injectable()
export class EncounterService {
  constructor(private prisma: PrismaService) {}

  create(createEncounterInput: CreateEncounterInput) {
    return this.prisma.encounter.create({
      data: {
        ...createEncounterInput,
      }
    })
  }

  async findAll() {
    const result = await this.prisma.encounter.findMany({});
    const client = new S3Client({
      region: process.env.S3_REGION,
      credentials: {
        accessKeyId: process.env.AWS_ACCESS_KEY_ID,
        secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
      }
    });

    const response = [];

    for(let index = 0 ; index < result.length; index++) {

      const command = new ListObjectsV2Command({
        Bucket: process.env.S3_BUCKET_NAME,

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

    Prefix: `Hospitals/${result[index].id}/`,
  });

const files = (await client.send(command)).Contents.filter((file) => !file.Key.endsWith('/'));
const signedUrl = await getSignedUrl(
  client,
  new GetObjectCommand({
    Bucket: 'diploma-shared-medical-files',
    Key: files[0].Key,
  }));

response.push({
  encounter: result[index],
  img: signedUrl,
})
}

return response;
}

findOne(id: number) {
  return this.prisma.encounter.findUnique({where: { id } })
}

update(updateEncounterInput: UpdateEncounterInput) {
  return this.prisma.encounter.findUnique({ where: { id: updateEncounterInput.encounterId } });
}

remove(id: number) {
  return `This action removes a #${id} encounter`;
}
}

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

Prisma схема

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url = env("DATABASE_URL")
}

model Doctor {
  id Int @id @default(autoincrement())
  email String @unique
  password String
  name String?
  description String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isVerified Boolean @default(false)
  type Roles @default(DOCTOR)
  medicalBranch MedicalBranch @relation(fields: [medicalBranchId], references: [id], onDelete: Cascade)
  medicalBranchId Int
  encounter Encounter @relation(fields: [encounterId], references: [id])
  encounterId Int
  clientsVisits ClientsDoctorsVisits[]
  workingSlots WorkingSlot[]
}

model WorkingSlot {
  id Int @id @default(autoincrement())
```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		77

```

doctor Doctor @relation(fields: [doctorId], references: [id])
doctorId Int
slot DateTime
createdAt DateTime @default(now())
updatedAt DateTime @updatedAt
}

```

```

model Administrator {
  id Int @id @default(autoincrement())
  email String @unique
  password String
  name String?
  description String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isVerified Boolean @default(false)
  type Roles @default(ADMIN)
}

```

```

model Client {
  id Int @id @default(autoincrement())
  email String @unique
  password String
  name String?
  description String?
  phone String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isVerified Boolean @default(false)
  type Roles @default(CLIENT)
  address String?
  clientsVisits ClientsDoctorsVisits[]
}

```

```

model Encounter {

```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		78

```

id Int @id @default(autoincrement())
name String
description String?
address String
phone String
doctors Doctor[]
startTime DateTime
endTime DateTime
}

```

```

model MedicalBranch {
  id Int @id @default(autoincrement())
  name String
  description String?
  doctors Doctor[]
  medicalServices MedicalService[]
}

```

```

model MedicalService {
  id Int @id @default(autoincrement())
  name String
  description String?
  price Float
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  medicalBranch MedicalBranch @relation(fields: [medicalBranchId], references: [id], onDelete: Cascade)
  medicalBranchId Int
  clientsVisits ClientsDoctorsVisits[]
}

```

```

model ClientsDoctorsVisits {
  id Int @id @default(autoincrement())
  client Client @relation(fields: [clientId], references: [id])
  clientId Int
}

```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

```

doctor Doctor @relation(fields: [doctorId], references: [id])
doctorId Int
visitDate DateTime
visitDescription String?
visitResult String?
visitStatus VISIT_STATUS @default(PENDING)
createdAt DateTime @default(now())
updatedAt DateTime @updatedAt
medicalService MedicalService @relation(fields: [medicalServiceId], references: [id])
medicalServiceId Int
isPaid Boolean @default(false)
}

```

```

enum Days {
  monday
  tuesday
  wednesday
  thursday
  friday
  saturday
  sunday
}

```

```

enum Roles {
  ADMIN
  CLIENT
  DOCTOR
}

```

```

enum VISIT_STATUS {
  PENDING
  ACCEPTED
  REJECTED
  CANCELED
}

```

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		80

Dashboard page

```

import { Grid, Paper, Stack, Typography } from "@mui/material";
import { DashboardDrawer } from "../../components/DashboardDrawer";
import AssignmentIndIcon from '@mui/icons-material/AssignmentInd';
import { ScrollStack } from "../../components/ScrollStack";
import FilesCard from "../../components/FilesCard";
import { useQuery } from "@apollo/client";
import CURRENT_USER from "../../graphql/queries/CURRENT_USER";
import { CurrentUserType } from "../../graphql/generated";
import PaymentsCard from "../../components/PaymentsCard";
import AppointmentsCard from "../../components/AppointmentCard";
import DashboardHeader from "../../components/DashboardHeader";
import { Outlet, useLocation, useNavigate } from "react-router-dom";
import { DashboardWrapper } from "../../components/DashboardWrapper";
import DashboardActionItem from "../../components/DashboardActionItem";

```

```

export default function DashboardPage() {
  const location = useLocation();

  const isDashboardOnlyLocation = location.pathname === '/dashboard';

  return (
    <DashboardWrapper>
      <DashboardDrawer />
      <Stack spacing={2} flex='1 0' padding='0 25px'>
        <DashboardHeader />
        <ScrollStack sx={{ flex: '1 0' }}>
          { isDashboardOnlyLocation && <DashboardContent /> }
        <Outlet />
      </ScrollStack>
    </Stack>
  </DashboardWrapper>
)

```

		Соболевський Д.А.			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		81

```
}
```

```
function DashboardContent() {
  const navigate = useNavigate();
  const currentUserQuery = useQuery<{ CurrentUser: CurrentUserType }>(CURRENT_USER);

  return (
    <Stack spacing={2}>
      <Paper sx={{ padding: '25px' }}>
        <Stack spacing={3}>
          <Typography variant='h4'>Раді вітати вас { currentUserQuery.data?.CurrentUser.name
        }</Typography>
          <Stack>
            <DashboardActionItem
              onClick={() => navigate('/dashboard/doctor-requests')}
              title='Створити запит до лікаря'
              icon={<AssignmentIndIcon sx={{ color: '#5CC470'}}/>}
            />
          </Stack>
        </Stack>
      </Paper>
      <div>
        <Grid container spacing={2}>
          <Grid item xs={6}>
            <PaymentsCard />
          </Grid>
          <Grid item xs={6}>
            <FilesCard />
          </Grid>
          <Grid item xs={12}>
            <AppointmentsCard />
          </Grid>
        </Grid>
      </div>
    </Stack>
  );
}
```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		82

)
}

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

Router

```

import { Navigate, createBrowserRouter } from "react-router-dom";
import DashboardPage from "../pages/Dashboard";
import LoginPage from '../pages/Credentials/LoginPage';
import RegistrationPage from "../pages/Credentials/RegistrationPage";
import PatientsPage from "../pages/Dashboard/Patients";
import DoctorsPage from "../pages/Dashboard/Doctors";
import EncountersPage from "../pages/Dashboard/Encounters";
import CredentialsPage from "../pages/Credentials";
import SettingsPage from "../pages/Dashboard/Settings";
import BillingsPage from "../pages/Dashboard/Billing";
import ProfilePage from "../pages/Profile";
import { ReportsPage } from "../pages/Dashboard/Reports";
import { DoctorRequestPage } from "../pages/Dashboard/DoctorRequest";
import { UploadFilePage } from "../pages/Dashboard/UploadFile";
import { AppointmentPage } from "../pages/Dashboard/Appointment";
import { ConfirmationPage } from "../pages/Credentials/ConfirmationPage";
import { UserOnlyRoute } from "../Protected/UserOnlyRoute";
import PatientsReportsPage from "../pages/Dashboard/PatientsReports";
import { UserGroupOnlyRoute } from "../Protected/UserGroupOnlyRoute";
import { User_Roles } from "../graphql/generated";

export const router = createBrowserRouter([
  {
    path: "/",
    Component: () => <Navigate to="/dashboard" replace={true} />
  },
  {
    path: 'credentials',
    Component: CredentialsPage,
    children: [
      {
        path: "login",

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

    Component: LoginPage,
  },
  {
    path: "registration",
    Component: RegistrationPage,
  },
  {
    path: "confirmation",
    Component: ConfirmationPage
  }
],
},
{
  path: "/dashboard",
  Component: () => <UserOnlyRoute children={<DashboardPage />} />,
  children: [
    {
      path: "patients",
      Component: () => <UserGroupOnlyRoute allowedRoles={[User_Roles['Doctor'],
User_Roles['Admin']]} children={<UserOnlyRoute children={<PatientsPage />} />} />,
    },
    {
      path: 'doctors',
      Component: () => <UserOnlyRoute children={<DoctorsPage />} />,
    },
    {
      path: 'encounters',
      Component: () => <UserOnlyRoute children={<EncountersPage />} />,
    },
    {
      path: 'settings',
      Component: () => <UserOnlyRoute children={<SettingsPage />} />,
    },
    {
      path: 'billings',

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		85

```

        Component: () => <UserOnlyRoute children={<BillingsPage />} />,
    },
    {
        path: 'reports',
        Component: () => <UserGroupOnlyRoute allowedRoles=[[User_Roles['Doctor'],
User_Roles['Admin']]] children={<UserOnlyRoute children={<ReportsPage />} />} />
    },
    {
        path: 'doctor-requests',
        Component: () => <UserOnlyRoute children={<DoctorRequestPage />} />,
    },
    {
        path: 'upload-file',
        Component: () => <UserGroupOnlyRoute allowedRoles=[[User_Roles['Doctor'],
User_Roles['Admin']]] children={<UserOnlyRoute children={<UploadFilePage />} />} />
    },
    {
        path: 'appointment/:id',
        Component: () => <UserOnlyRoute children={<AppointmentPage />} />,
    },
    {
        path: 'patients-reports',
        Component: () => <UserGroupOnlyRoute allowedRoles=[[User_Roles['Doctor'],
User_Roles['Admin']]] children={<UserOnlyRoute children={<PatientsReportsPage />} />} />
    }
]
},
{
    path: '/profile',
    Component: () => <UserOnlyRoute children={<ProfilePage />} />,
},
]);

```

		Соболевський Д.А			ІІЗ.КР.Б – 121 – 23 - ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		86

App.tsx

```
import './App.css'
import { RouterProvider } from "react-router-dom";
import { router } from './router';
import CssBaseline from '@mui/material/CssBaseline';
import { ApolloClient, InMemoryCache, ApolloProvider, createHttpLink, from } from
'@apollo/client';
import { createUploadLink } from 'apollo-upload-client';
import { onError } from "@apollo/client/link/error";
```

```
const errorLink = onError(({ graphQLErrors, networkError }) => {
  if (graphQLErrors) {
    graphQLErrors.forEach(({ message, locations, path }) => {
      if(message === 'Unauthorized') {
        window.location.pathname = '/credentials/login';
      }
    });
  }
});
```

```
function App() {
  const client = new ApolloClient({
    link: from([
      errorLink,
      createUploadLink({
        uri: import.meta.env.GRAPHQL_URL,
        credentials: 'include',
        headers: {
          'x-apollo-operation-name': "UploadFile",
        }
      })
    ]),
```

		Соболевський Д.А			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

    createHttpLink({
      uri: import.meta.env.GRAPHQL_URL,
      credentials: 'include',
    }),
  ],
  cache: new InMemoryCache(),
});
return (
  <ApolloProvider client={client}>
    <CssBaseline>
      <RouterProvider router={router} />
    </CssBaseline>
  </ApolloProvider>
)
}

export default App

```

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

Клас UploadFile page

```

import { Stack, Paper, Typography, Button, Alert } from '@mui/material';
import { useState } from 'react';
import { useDropzone } from 'react-dropzone';
import { ClientWithImage, UploadReportResponse } from '../../graphql/generated';
import DropZone from './components/DropZone';
import { UserSelect } from './components/UserSelect';
import { useMutation } from '@apollo/client';
import UPLOAD_REPORT from '../../graphql/mutations/UPLOAD_REPORT';

export function UploadFilePage() {
  const [file, setFile] = useState<File | null>(null);
  const [assignedUser, setAssignedUser] = useState<ClientWithImage>();
  const { getRootProps } = useDropzone({ onDrop: files => setFile(files[0]) });
  const [uploadReportMutation] = useMutation<{ UploadReport: UploadReportResponse }>(UPLOAD_REPORT);
  const [status, setStatus] = useState<UploadReportResponse>();

  const handleUpload = async () => {
    if (!file || !assignedUser) {
      return;
    }

    const result = await uploadReportMutation({
      variables: {
        uploadReportInput: {
          file,
          account: {
            id: assignedUser.client.id,
            type: assignedUser.client.type,
          },
        },
      },
    });
  }
}

```

		Соболевський Д.А.			ПЗ.КР.Б – 121 – 23 - ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

});

setStatus(result.data?.UploadReport);

setTimeout(() => {
  setStatus(undefined);
  setFile(null);
  setAssignedUser(undefined);
}, 1400)
};

return (
  <Paper sx={{ padding: '25px' }}>
    <Stack spacing={2}>
      <UserSelect onChange={(user: ClientWithImage) => setAssignedUser(user)} />
      <DropZone className='drop-zone' {...getRootProps()}>
        <Stack spacing={2} textAlign='center'>
          <Typography variant='h6'>Завантажити Файл</Typography>
          { file && <Typography variant='body2'>{file?.name || ""}</Typography> }
        </Stack>
      </DropZone>
      <Button
        disabled={!file || !assignedUser}
        variant='contained'
        onClick={handleUpload}
      >
        Завантажити
      </Button>
      {
        status === UploadReportResponse['Success'] &&
        <Alert variant="filled" severity="success">
          <Typography variant='body1'>Файл успішно завантажено</Typography>
        </Alert>
      }
    </Stack>
  </Paper>
);

```

		Соболевський Д.А.			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
  status === UploadReportResponse['Failed'] &&
  <Alert variant="filled" severity="error">
    <Typography variant='body1'>Під час завантаження виникла помилка</Typography>
  </Alert>
}
</Stack>
</Paper>
)
}

```

		Соболевський Д.А			ІПЗ.КР.Б – 121 – 23 - ІПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		