Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-додатку для контролю за веденням документації в медичних закладах з використанням JavaScript технологій

Виконав: студент 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Вінтонів С. О.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Никитюк В. В.
(підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)

« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Вінтоніву Святославу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-додатку для контролю за веденням документації в медичних закладах з використанням JavaScript технологій

Керівник роботи Петрик Михайло Романович, д.ф.-м.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи вимоги до функціоналу системи, використання JavaScript технологій

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз предметної області та існуючих рішень, проєктування архітектури застосунку, розробка веб-додатку для контролю за веденням документації в медичних закладах, використання мови JavaScript та технологій пов'язаних з нею під час розробки системи, тестування розробленого програмного забезпечення, розділ безпеки життєдіяльності та охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма варіантів використання, діаграма класів серверної частини, діаграма бази даних, зображення графічного інтерфейсу веб-додатку, слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності			
Основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі		
2	Огляд існуючих рішень		
3	Проєктування та розробка веб-додатку		
4	Тестування веб-додатку		
5	Підготовка пояснювальної записки		
6	Розділ безпеки життєдіяльності та основ охорони праці		
7	Підготовка презентації та доповіді		
8	Попередній захист		
9	Нормоконтроль, рецензування		
10	Занесення диплому в електронний архів		
11	Допуск до захисту у зав. кафедри		

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41, 2023 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: сторінок – 76, рисунків – 45, таблиць – 1, частин – 4, додатків – 2, посилань – 21.

Тема: Розробка веб-додатку для контролю за веденням документації в медичних закладах з використанням JavaScript технологій

Метою кваліфікаційної роботи є проєктування та розробка веб-додатку який дозволить оптимізувати процес ведення документації надаючи лікарям можливість планувати візити для пацієнтів та швидко їх документувати використовуючи зручний та інтуїтивно зрозумілий графічний інтерфейс.

Результатом виконаної роботи є реалізований веб-додаток який можна використовувати в медичних заклад для пришвидшення процесу ведення документації. Розроблена система надає можливості адміністрування, зберігання та оновлення даних пацієнтів, планування візитів, створення документів як вручну так із використанням шаблонів а також поділу їх на стани для пришвидшення пошуку. Також роздроблений веб-додаток можна легко вдосконалювати та додавати нові функції за рахунок його модульності.

Ключові слова: ЛІКАР, ПАЦІЄНТ, ВІЗИТ, ШАБЛОН ДОКУМЕНТУ, ПЛАН ЛІКУВАННЯ, КОМПОНЕНТ, СЕРВЕР, JAVASCRIPT, REACT, GRAPHQL, MICROSOFT SQL SERVER.

ABSTRACT

Qualification work for obtaining a bachelor's degree in specialty 121 - Software engineering. Ternopil National Technical University named after Ivan Pulyu, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41, 2023. The explanatory note to the qualification work for obtaining the bachelor's degree contains: pages – 76, figures – 45, tables – 1, parts – 4, appendices – 2, references – 21.

Topic: Development of a web application for monitoring documentation in medical institutions using JavaScript technologies.

The goal of the qualification work is to design and develop a web application that will optimize the documentation process, giving doctors the opportunity to plan visits for patients and quickly document them using a convenient and intuitive graphical interface.

The result of the completed work is an implemented web application that can be used in a medical institution to speed up the documentation process. The developed system provides the ability to administer, store and update patient data, plan visits, create documents both manually and using templates, and also divide them into states to speed up the search. Also, a fragmented web application can be easily improved and new features added due to its modularity.

Keywords: DOCTOR, PATIENT, VISIT, DOCUMENT TEMPLATE, TREATMENT PLAN, COMPONENT, SERVER, JAVASCRIPT, REACT, GRAPHQL, MICROSOFT SQL SERVER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 11
1.1 Аналіз предметної області 11
1.2 Аналіз існуючих рішень 14
1.3 Постановка задачі 19
1.4 Визначення акторів системи та варіантів використання 20
1.4.1 Детальний опис ключових варіантів використання 23
1.5 Вибір методології проектування 25
1.6 Проєктування архітектури інформаційної системи 28
1.7 Вибір технологій розробки 29
1.8 Проєктування архітектури бази даних 33
2 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 45
2.1 Реалізація основних класів серверної частини 45
2.2 Реалізація клієнтської частини веб-додатку 51
3 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 56
3.1 Перевірка роботи базових функцій серверного API 56
3.2 Тестування графічного інтерфейсу веб-додатку 59
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 69
4.1 Долікарська допомога при ураженні електричним струмом 69
4.2 Вимоги ергономіки до організації робочого місця оператора ПК 72
ВИСНОВКИ 74

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 75

ДОДАТКИ 77

Додаток А – Лістинг коду додатку для клієнтської частини 78

Додаток Б – Диск із кваліфікаційною роботою 88

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

СУБД – система управління базами даних

CSS – Cascading Style Sheets

SOAP – Subjective Objective Assessment Plan

ВВ – Варіанти використання

ПЗ – Програмне забезпечення

CRM – Customer Relationship Management

HTML – Hyper Text Markup Language

EHR – Electronic Health Record

RAD – Rapid Application Development

RUP – Rational Unified Process

JS – Java Script

NPM – Node Package Manager

DOM – Document Object Model

SPA – Single Page Application

API – Application Programming Interface

REST – Representational State Transfer

СЛР – Серцево-Легенева Реанімація

КПО – Коефіцієнт природньої освітленості

ВСТУП

Сучасна медична система потребує точності та швидкості у веденні даних про пацієнтів, щоб забезпечити ефективність та безпеку медичного обслуговування. Також очевидним є те, що у сучасному світі медицина стає все більш комплексною і вимагає від працівників великої уваги до деталей і точності. Правильне ведення медичної документації є одним з найважливіших елементів успішної практики в будь-якому медичному закладі.

Останні події такі як пандемія COVID-19 продемонстрували, що медична система всього світу стала під загрозою колапсу через велику кількість хворих, які потребували невідкладної та якісної допомоги а колапс в такій ситуації може призвести до трагічних наслідків, тому очевидним є те, що потрібно максимально оптимізовувати процеси не по’вязані безпосередньо з взаємодією з пацієнтами для того, щоб сторонні процеси не забирали у медичного персоналу занадто багато часу.

Великим викликом для медичних закладів було ведення документації про пацієнтів – це важлива частина роботи медичних працівників, оскільки правильне ведення документації допомагає забезпечити безпеку пацієнтів та зберегти важливі медичні дані. Однак ведення документація може бути часом важкою та трудомісткою процедурою а з великою кількістю хворих, важко було підтримувати точну та актуальну інформацію про кожного пацієнта, особливо коли використовуються та застарілі системи.

На даний час є велика кількість програмних систем які допомагали медикам у веденні документації, проте більшість з цих систем є великими за розміром, через це їх важко інтегрувати в процес роботи медичного персоналу. Деякі з існуючих систем є важкими для розуміння зі складним користувацьким інтерфейсом, що займає більше часу для навчання персоналу. У зв'язку з цим стає очевидною необхідність розробки простого, та зручного додатку який міг би допомогти медичним працівникам працювати швидше та ефективніше. Такий додаток міг би

допомогти у запобіганні непотрібному дублюванню інформації, забезпечивши швидкий доступ до необхідних даних. Більш того, він може стати чудовим інструментом для підтримки комунікації між медичним персоналом, що допоможе надавати швидку та кваліфіковану медичну допомогу пацієнтам.

Метою даної роботи є розробка веб-додатку для контролю за веденням документації в медичних закладах. Він буде забезпечувати медичним працівникам можливість швидкого внесення, збереження та оновлення інформації про пацієнтів, зменшуючи тим самим навантаження на медичний персонал і забезпечуючи кращу організацію роботи медичного закладу. Використання JavaScript технологій у розробці повинно дозволить покращити взаємодію з користувачем та забезпечити зручний та інтуїтивно зрозумілий інтерфейс додатку.

Для того, щоб виконати мету роботи, та ефективно реалізувати програмний продукту, потрібно виконати наступні пункти:

- провести детальний аналіз предметної області, визначивши всі необхідні функції які повинен виконувати веб-додаток;
- провести аналіз існуючих рішень та оцінити сильні та слабкі сторони кожного з таких;
- спроектувати всі необхідні частини інформаційної системи;
- реалізувати та протестувати готову систему на відповідність усім поставленим до неї вимогам.

Після того як буде виконано всі з вище перелічених пунктів, буде реалізовано програмний продукт, який буде протестований та готовий до впровадження в медичні заклади для спрощення та покращення процесу ведення документації з метою збільшення ефективності та продуктивності роботи медичних працівників.

1 АНАЛІЗ ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз предметної області

У сучасній галузі охорони здоров’я ефективний документообіг і контроль є надзвичайно важливими, оскільки вони безпосередньо впливають на якість обслуговування пацієнтів. Медичні заклади відповідають за керування великою кількістю даних і документів, пов’язаних з доглядом за пацієнтами, включаючи електронні медичні записи, діагностичні звіти, рецепти та інші важливі документи. Щоб забезпечити належний догляд за пацієнтами, ці документи потрібно безпечно та ефективно зберігати, систематизувати, відстежувати та обмінюватись ними.

Контроль документообігу передбачає управління та відстеження різноманітних документів, які використовуються в медичному закладі. Він охоплює процеси створення, розповсюдження, зберігання, пошуку та архівування документів. Ці процеси зазвичай виконуються вручну, на паперових носіях і займають багато часу, що призводить до затримок, помилок і неефективності.

Серед основних причин через які необхідно автоматизувати процес документообігу в медичних закладах можна виділити наступні:

- Адміністративні витрати можуть досягати від 25% до 33% загальних витрат на охорону здоров’я.
- Працівники витрачають у середньому 50% свого часу на створення, підготовку та керування документами вручну.
- Близько 83% співробітників відтворюють існуючі документи з нуля, тому що вони не можуть знайти їх у мережі своєї компанії.

Медичні установи, такі як лікарні та клініки є складними організаціями, які надають пацієнтам широкий спектр медичних послуг. Ці заклади покладаються на використання різноманітних документів для управління доглядом за пацієнтами. Документи, які використовуються в медичних закладах, часто є конфіденційними та потребують безпечного зберігання та обробки, що забирає багато часу та ресурсів у медичного персоналу. Згідно з дослідженнями які проводились у 2021

році повідомлялося, що лікарі витрачають в середньому 16 годин на тиждень на адміністративні завдання, такі як паперова робота. Щороку організації охорони здоров’я виділяють приблизно 2,1 мільярда доларів США на керування даними, схильними до помилок і неефективними. Це пов’язано з високим відсотком неточностей, виявлених у медичних записах, при цьому 35% інформації відсутні або містять помилки, 28% записів є дублікатами, а 18% містять неправильну або відсутню інформацію про пацієнта [1].

Більш детальний відсотковий розподіл помилок які зустрічаються в медичній документації зображено в таблиці нижче (табл.1.1).

Таблиця 1.1 – Відсотковий розподіл помилок в медичній документації

Типові помилки в медичній документації	Відсоток
Документи, що містять помилки або відсутню інформацію	35%
Документи які дублюються	28%
Документи з помилковими даними про пацієнта	18%
Неправильні або відсутні номери мобільних телефонів пацієнтів	12%
Неправильні або відсутні адреси	11%

Великі витрати ресурсів на ведення документації частково пов’язані з тим, що ці документи використовуються для того, щоб отримати відшкодування від страхових компаній, і через це вони мають бути добре, та детально описані, що забирає багато часу в медичного персоналу. Якщо підчас створення документу про візит пацієнта в паперовому вигляді допускається помилка то досить часто його необхідно переписувати спочатку. Через це Лікарі витрачають більше часу, на документацію та адміністративні завдання, що призводить до незадоволеності та професійного вигорання, все більше лікарів бачать розрив між цінностями, які привели їх до медицини, та повсякденною реальністю з якою їм необхідно стикатись.

Одним із найважливіших етапів контролю документообігу є архівування медичної документації на довгострокове зберігання. Ці записи необхідні для надання якісної медичної допомоги, дотримання законодавства та дослідницьких цілей. Однак паперові архівні системи мають ряд обмежень, які можуть вплинути на ефективність процесу контролю документообігу. З часом кількість документів збільшується, що ускладнює їх ефективне зберігання та пошук. Це може призвести до дублювання записів для пацієнтів, які вже знаходяться в архіві, що призведе до непотрібного навантаження на медичний персонал і марної витрати ресурсів. Крім того, важливі записи можуть бути втрачені або загублені, через що їх важко або неможливо відновити за потреби.

Отже контроль документообігу в медичному закладі – це складний процес, що складається з декількох етапів, починаючи від створення документа і закінчуючи його остаточним архівуванням, підчас всього життєвого циклу документів на їх супровід витрачається велика кількість ресурсів, які можна і необхідно зекономити. Тому розробка програмної системи для контролю документообігу в медичному закладі має вирішальне значення для оптимізації зберігання та пошуку медичної документації. Очевидним є те, що найпростішим вирішення цієї проблеми було б створення веб-додатку, оскільки для його використання не потрібно встановлювати додаткове програмне забезпечення та витрачати час на його налаштування, користувач може просто увійти в браузер використовуючи будь-яку платформу, авторизуватись та отримати доступ до всієї необхідної інформації. Веб-додаток міг би забезпечити надійне та ефективне довгострокове зберігання записів із можливістю легкого доступу та редагування у разі потреби.

1.2 Аналіз існуючих рішень

На даний момент на ринку існує декілька ефективних рішень для управління документацією в медичних установах. Таким чином, щоб розробити кращу систему вкрай важливо провести комплексний і глибокий аналіз кожного рішення, щоб визначити його відповідні переваги та недоліки. Аналіз існуючих рішень дасть цінну інформацію про різноманітні характеристики та функціональні можливості кожної системи, що дозволить розробити більш ефективну, безпечну та результативну систему, яка відповідає конкретним вимогам медичних установ.

Однією з найпопулярніших інформаційних систем для управління документацією в медичних установах є Pabau.

Pabau CRM (рис. 1.1) [2] – це хмарне програмне забезпечення для управління взаємовідносинами з клієнтами, розроблене для постачальників медичних послуг.

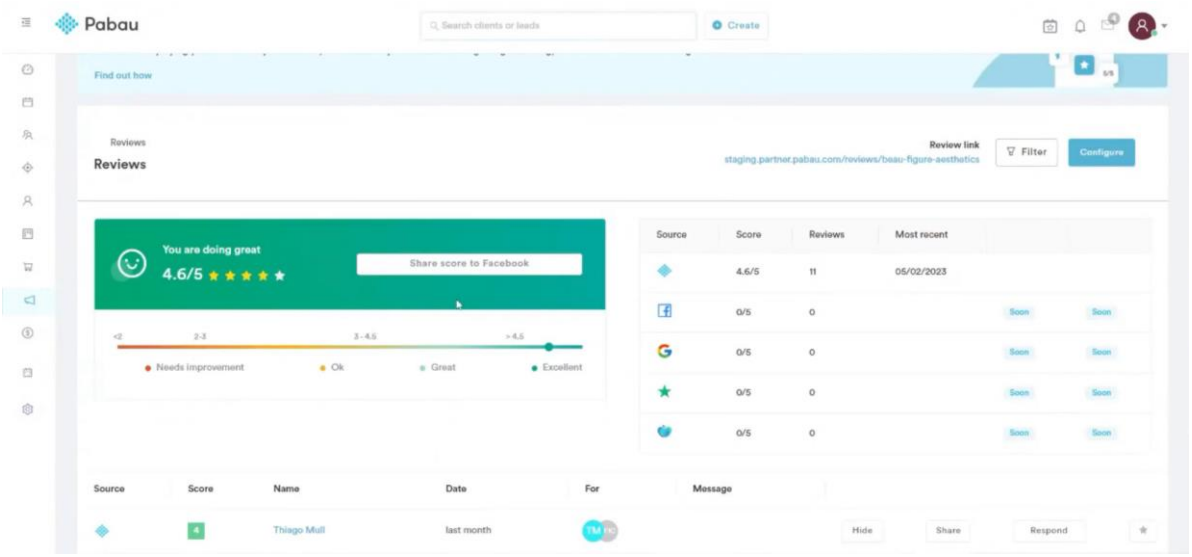

Рисунок 1.1 – Pabau CRM

Переваги даної інформаційної системи:

- Дозволяє користувачам безпечно зберігати та керувати електронними медичними записами, усуваючи потребу у фізичному зберіганні документів.

- Система забезпечує легкий доступ і керування даними пацієнтів та історіями хвороби, що може допомогти покращити результати лікування пацієнтів і зменшити кількість помилок.
- Користувачі можуть установлювати дозволи для контролю доступу до конфіденційної медичної інформації, забезпечуючи конфіденційність пацієнтів і відповідність нормам HIPAA.
- Система пропонує настроювані шаблони для створення та керування електронними формами, такими як форми згоди та форми прийому пацієнтів.
- Система пропонує автоматичні нагадування та сповіщення, щоб зменшити ризик пропущених зустрічей і подальших процедур.

Недоліки даної інформаційної системи:

- Система може бути складною і може потребувати технічних знань для налаштування та обслуговування.
- Деякі користувачі повідомили про проблеми з повільним часом завантаження та періодичними збоями.
- Система може не підійти для невеликих медичних установ із обмеженими ресурсами чи технічними знаннями.
- Під час використання різних функцій системи користувачі можуть відчувати труднощі з навчанням.

Загалом Pabau CRM пропонує надійну систему контролю документів для постачальників медичних послуг. Його функції для зберігання та керування електронними медичними документами та формами разом із настроюваними шаблонами та автоматичними нагадуваннями можуть допомогти підвищити ефективність і зменшити кількість помилок у медичних налаштуваннях. Однак складність системи та потенційні технічні проблеми можуть зробити її менш придатною для невеликих медичних установ.

Доволі поширеною системою в галузі менеджменту медичних документів є Axxess Home Health.

Axxess Home Health (рис. 1.2) [3] – це комплексна система управління домашнім здоров’ям, розроблена для забезпечення незалежного від платформи

рішення для клінічної документації та забезпечення якості. Система пропонує настроювані робочі процеси та плани догляду для задоволення конкретних потреб пацієнтів, а також широкі можливості звітності, щоб допомогти агентствам визначити сфери, які потребують покращення.

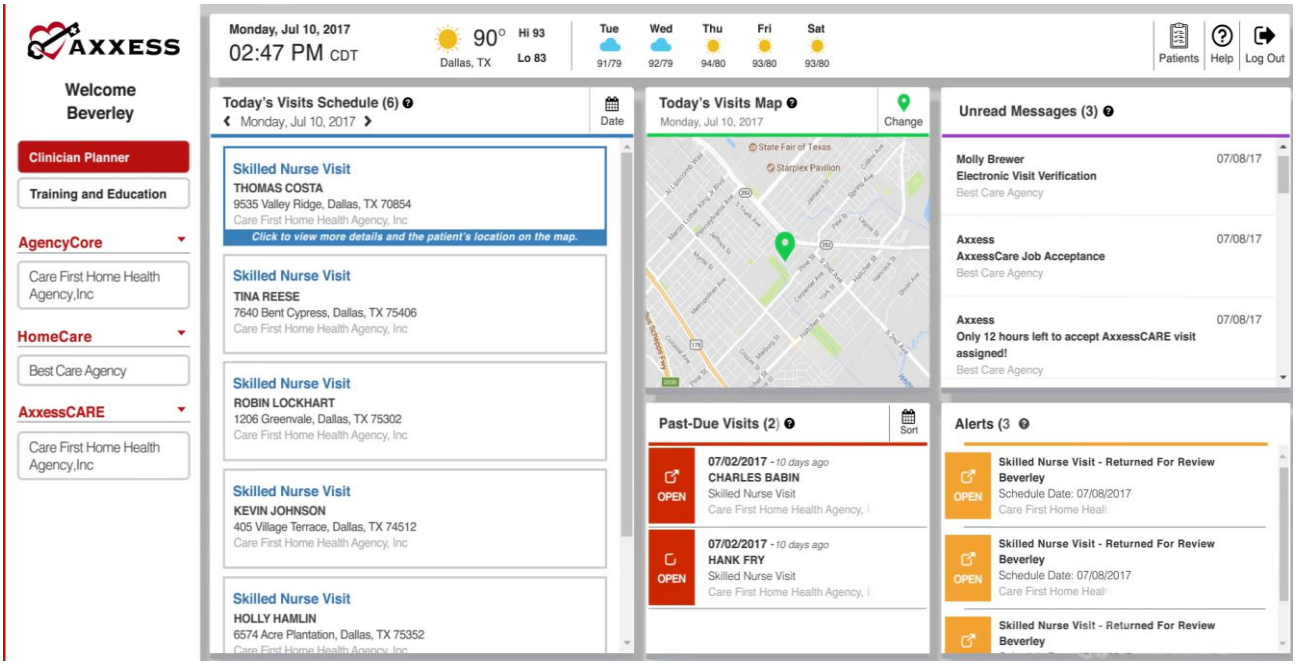

Рисунок 1.2 – Axxess Home Health

Переваги даної інформаційної системи:

- Комплексна система управління здоров'ям удома з акцентом на клінічній документації.
- Настроювані робочі процеси та плани догляду, які можна пристосувати до конкретних пацієнтів або умов.
- Широкі можливості звітності, включаючи інструменти порівняльного аналізу та аналітики, які допомагають агентствам визначити сфери, які потрібно вдосконалити.
- Зручний інтерфейс, простий у навігації та використанні навіть для співробітників із обмеженим технічним досвідом.

- Мобільний додаток, доступний як для пристроїв iOS, так і для Android, дозволяє персоналу отримувати доступ до даних пацієнтів і документації в дорозі.

Недоліки даної інформаційної системи:

- Ціни можуть бути непомірно високими для деяких невеликих або незалежних медичних агентств.
- Обмежена стороння інтеграція з іншими програмними системами, що може обмежити універсальність системи та сумісність з іншими інструментами.
- Хоча систему можна налаштувати, деякі користувачі повідомляють, що може бути важко внести зміни або модифікації без допомоги служби підтримки постачальника.

Загалом, Axxess Home Health — це потужне та комплексне рішення для медичних установ, які прагнуть оптимізувати свою клінічну документацію та процеси забезпечення якості. Проте ціна системи та обмежена інтеграція зі стороннім програмним забезпеченням можуть бути недоліком для більшості користувачів.

Однією з систем яка слугує для допомоги медичним працівникам і привертає увагу своєю популярністю є RXNT EHR.

RXNT EHR (рис. 1.3) [4] – це хмарна система електронних записів про стан здоров’я (EHR), розроблена для малих та середніх медичних установ. Він пропонує комплексний набір інструментів, які допоможуть медичним працівникам оптимізувати свої клінічні робочі процеси та покращити результати лікування пацієнтів.

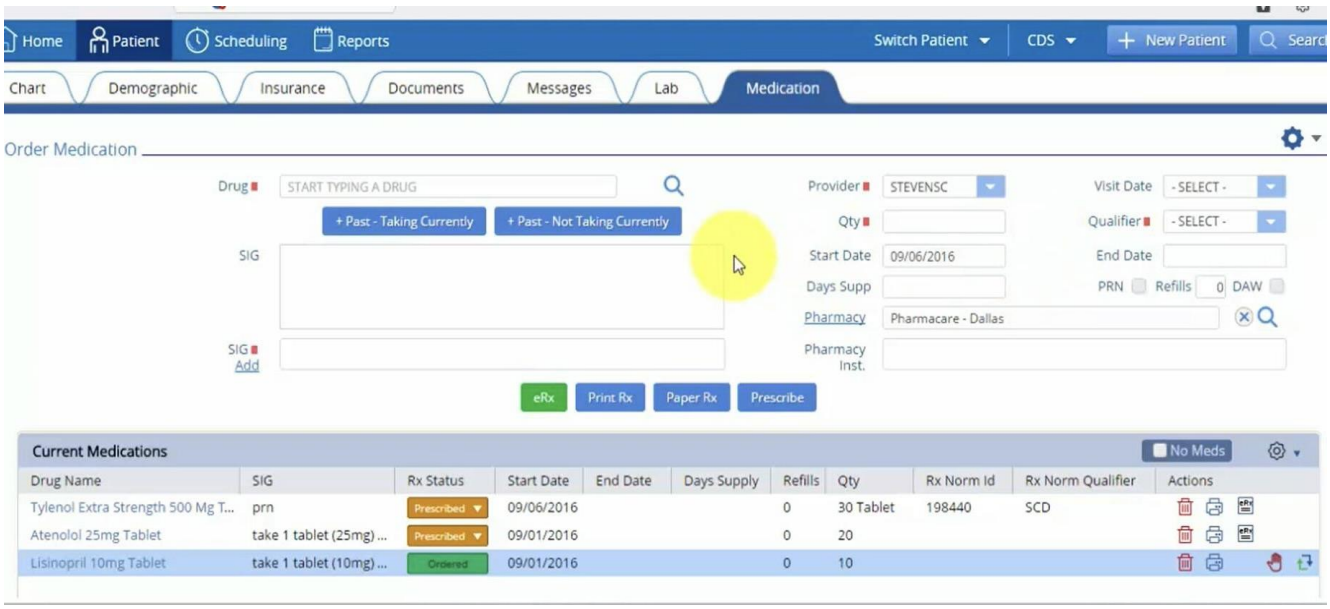

Рисунок 1.3 – RXNT EHR

Переваги даної інформаційної системи:

- Система має інтуїтивно зрозумілий і зручний користувацький інтерфейс, який полегшує медичним працівникам навігацію системою та виконання своїх щоденних завдань.
- Система пропонує настроювані шаблони, які дозволяють провайдерам створювати документацію, яка відповідає потребам їх практики.
- Система пропонує надійні можливості звітування, що дозволяє постачальникам медичних послуг відстежувати ключові показники, такі як результати пацієнтів, планування призначень і клінічні результати.

Недоліки даної інформаційної системи:

- Система не підходить для всіх медичних спеціальностей, оскільки в ній відсутні особливі функції, необхідні деяким спеціальностям, таким як педіатрія чи поведінкове здоров’я.
- Інформаційна система не має інструментів залучення пацієнтів, таких як портал для пацієнтів, який може мати вирішальне значення для покращення результатів лікування.

- Обмежене навчання та підтримка, деякі користувачі повідомили про відсутність навчання та підтримки від RXNT, через що їм важко повністю використовувати функції та можливості системи.

Підсумовуючи, RXNT EHR є хорошим рішення для малих і середніх медичних закладів, яким потрібна надійна та зручна система для керування своїми клінічними робочими процесами. Однак система може не підходити для всіх медичних спеціальностей і не має інструментів залучення пацієнтів. Крім того, деякі користувачі повідомляли про відсутність навчання та підтримки від адміністрації системи.

1.3 Постановка задачі

Після того як було проведено детальний аналіз предметної області та проаналізовано існуючі на ринку програмні рішення можна виділити вимоги які повинен забезпечувати веб-додаток для контролю за веденням документації в медичних закладах, а саме:

- Здійснення реєстрації та авторизації в системі.
- Можливість зміни ролі користувача.
- Можливість деактивації акаунта для користувачів.
- Перегляд статистики медичного закладу.
- Перегляд статистики для конкретного медика.
- Можливість редагування профілю користувача.
- Можливість зберігання, редагування та видалення даних про пацієнта в системі.
- Можливість запланувати візит для пацієнта.
- Можливість створювати нотатки під час візиту пацієнту.
- Можливість переносити візит та змінювати причину візити.
- Можливість видаляти запланований візит.

- Можливість для кожного задокументованого візиту створити суб’єктивний аналіз та оцінки стану здоров’я пацієнта, визначити мету та план лікування.
- Можливість створювати шаблони для документів, а також редагувати та видаляти шаблони для документів.
- Можливість заповнювати документ використовуючи шаблони.
- Можливість створити шаблон використовуючи існуючий документ.
- Пошук пацієнтів в системі.
- Розподіл документів в залежності від стану в якому він перебуває.
- Можливість фільтрувати документи для конкретних пацієнтів.
- Можливість створювати унікальні шаблони документів для кожного конкретного пацієнта.
- Пошук документів в системі.
- Сповіщення для задокументованих візитів які потребують оцінки лікаря і про які він міг забути.
- Можливість архівувати документи та дані про пацієнтів.

1.4 Визначення акторів системи та варіантів використання

Правильне визначення та розуміння ролей і обов’язків акторів системи має важливе значення для розробки ефективної інформаційної системи для контролю документообігу в медичному закладі. Цей процес впливає на всі наступні етапи проєктування та розробки інформаційної системи. Після завершення аналізу предметної області та визначення основних вимог до інформаційної системи можна визначити ключових учасників, залучених до системи [5]. У даній системі можна розрізнити трьох основних дійових осіб, а саме: гостя, адміністратора та лікаря. Ці учасники відіграють різні ролі та мають конкретні обов’язки в системі.

Гість є зовнішнім користувачем, тому він не повинен мати доступу до основних функцій системи. Даний актор буде мати можливість зареєструватися в

системі, та зробити запит до адміністратора, щоб отримати роль адміністратора або лікаря. Гість не зможе виконувати жодних адміністративних або медичних завдань.

Лікар є основним користувачем системи і має доступ до всіх основних функцій системи. Лікар може вносити дані про пацієнтів в систему та працювати з ними, планувати, редагувати та видаляти візити, вводити клінічні дані, керувати планами лікування. Також лікар буде мати можливість створювати та редагувати шаблони документів для того, щоб в подальшому пришвидшити та полегшити процес ведення документації. Лікар також може використовувати інші функції, необхідні для покращення процесу документування. Лікар несе відповідальність за ведення точної та актуальної медичної документації, своєчасному оновленні даних про пацієнтів та забезпечення ефективного виконання планів лікування.

Адміністратор буде керівною роллю в даній системі. Основний обов’язок адміністратора полягає в управлінні обліковими записами користувачів, включаючи їх створення, редагування та деактивацію за необхідності. Крім того, адміністратор повинен мати повноваження призначати ролі користувачам на основі їхніх обов’язків і зв’язків у системі. Адміністратор має можливість аналізувати повну статистику, що стосується всього медичного закладу, а також окремих лікарів. Ця аналітична оцінка дає адміністратору можливість оцінити ефективність зусиль медичних працівників і визначити загальну продуктивність закладу. Адміністратор несе відповідальність за забезпечення безпеки та конфіденційності медичної документації, що зберігається в системі, також контроль за діями інших користувачів системи.

Детальніше та структурованіше кожний з варіантів використання зображено на рисунку 1.4.

Рисунок 1.4 – Діаграма варіантів використання

На діаграмі ВВ зображено актантів та основні варіанти використання системи. Такі ВВ як «Створення документу» і «Додавання шаблону» є узагальнення двох прецедентів, що дозволяють використати різні можливості заповнення даних.

1.4.1 Детальний опис ключових варіантів використання

В інформаційній системі як призначення для контролю за веденням документації в медичному закладі можна виділити декілька ключових варіантів використання для більш детального їх опису:

- «Зберігання даних про пацієнта в системі». Цей варіант використання потрібен для того, щоб лікар міг ввести дані про свого пацієнта в систему, для подальшого планування візитів, та лікування. Коли виконується даний варіант використання лікар має заповнити форму та ввести всі дані про пацієнта, такі як: прізвище та ім’я, номер, дату народження та стать, якщо всі дані введено і збережено, то в системі створиться запис про пацієнта, якщо форма заповнена не повністю, то лікар отримає про це повідомлення.
- «Планування візиту». Даний варіант використання слугує для того, щоб лікар міг запланувати візит для одного з існуючих в системі пацієнтів. Під час створення запису про візит, лікар повинен заповнити форму, яка має складатись з наступних полів: дата та час на яку заплановано візит, причина візиту, та основна скарга пацієнта. Якщо форма заповнена, то дані будуть збережені в системі, якщо відбулась якась помилка, то лікар отримає повідомлення про це.
- «Створення документу». Підчас або після того як відбувся візит, лікар має його задокументувати, а саме в документі має бути вказано суб’єктивну думку лікаря про стан пацієнта, оцінку його здоров’я, мету та план лікування, ці дані лікар може ввести самостійно, або використати один із свої існуючих шаблонів. Після того як форму для створення документу було повністю заповнено і лікар її підтвердив, документ переходить на наступний етап, а саме лікування.
- «Створення шаблону». Шаблони використовуються для того, щоб лікарі могли швидше заповнювати документи, шаблони поділені на загальні та для конкретного пацієнта. Для того, щоб створити шаблон лікар може

використати один із готових документів і на його основі буде створено шаблон, або лікар може власноруч заповнити форму та створити шаблон. Після того, як шаблон буде створено, він буде зберігатися в сховищі шаблонів і під час створення нових документів лікар може їх використовувати.

- «Авторизація». Даний варіант використання слугує для того, щоб користувачі могли увійти в систему. Для того, щоб авторизуватись в системі користувач має ввести свою електрону пошту та пароль, після чого відбудеться перевірка на те, чи користувач існує в системі і чи його акаунт не заблоковано. Якщо користувача який проходив авторизацію немає в системі, йому буде запропоновано провести реєстрацію, у випадку якщо акаунт користувача було деактивовано, він отримає повідомлення про це з пропозицією звернутись до адміністрації. У випадку, якщо авторизація пройшла успішно, користувач увійде в систему і отримає доступ до функцій які передбачені його роллю.
- «Деактивація акаунтів». Дана функція доступна адміністраторам системи, її використовують для того, щоб позбавити деяких користувачів права доступу до даної інформаційної системи. Після того як адміністратор деактивував профіль користувача, він отримає повідомлення про це та більше не зможе увійти в систему та отримати доступ до будь-яких функцій. Також адміністратор може використати функцію відновлення акаунту. Після використання цієї функції для конкретного заблокованого користувача, він знову буде мати змогу увійти в систему та використовувати функціонал який доступні для його ролі.
- «Редагування профілю». Цей варіант використання слугу для того, щоб користувачі мали змогу змінювати свої особисті дані в інформаційній системі. Під час виконання цього варіанту використання користувач може змінити свої дані використовуючи форму для редагування. Якщо користувач заповнив всі необхідні поля, то дані його профілю будуть оновлені, якщо під час виконання виникла помилка, то користувач отримає повідомлення про це, та зможе повторити процес.

1.5 Вибір методології проектування

У сучасному динамічному та конкурентному середовищі вибір правильної методології розробки програмного забезпечення має вирішальне значення для успіху будь-якого проекту. Існує різноманітна кількість методологій, зокрема Waterfall, Agile, RAD і RUP.

Правильна методологія розробки може допомогти забезпечити завершення проекту вчасно, у межах бюджету та відповідно до необхідних стандартів якості. Наприклад, гнучкі методології, такі як Scrum і Kanban, добре підходять для проектів, які потребують гнучкості та адаптивності, де вимоги можуть часто змінюватися. Навпаки, методології Waterfall краще підходять для проектів із чітко визначеними вимогами та фіксованим обсягом.

Вибір неправильної методології може призвести до затримок, перевищення витрат і неможливості отримати бажані результати.

Для того, щоб вибрати методологію для розробки даної системи було розглянуто дві найбільш популярні, це RAD і RUP.

RUP — це структурований ітеративний підхід до розробки ПЗ, який підкреслює важливість аналізу вимог, архітектури та дизайну. Це гнучка методологія, яку можна налаштувати відповідно до конкретних потреб проекту. RUP поділяє процес розробки такі фази: початок, розробка, будівництво та реліз, і кожна фаза має свій власний набір цілей, етапів і результатів [6].

З іншого боку, RAD — це підхід швидкого створення прототипів, мета якого полягає в швидкій розробка робочого прототипу програми. Він зосереджений на залученні користувачів до процесу розробки, щоб гарантувати, що програма відповідає їхнім потребам. RAD наголошує на швидкості та ефективності, і він підходить для проектів з обмеженим бюджетом і стислими термінами [7].

Оскільки веб-додаток для контролю за ведення документації в медичних закладах потребує швидкої та ефективної розробки зі стислими термінами, було прийнято рішення обрати саме методологію RAD.

Методологія швидкої розробки додатків (RAD) — це ітеративний підхід до розробки ПЗ, який наголошує на швидкому створенні прототипів і відгуках користувачів. RAD передбачає підхід спільної команди, яка включає розробників, дизайнерів і кінцевих користувачів, які працюють разом, щоб створити робочий прототип програмного додатку [5].

Схематично базовий принцип роботи RAD можна побачити на рисунку 1.5.

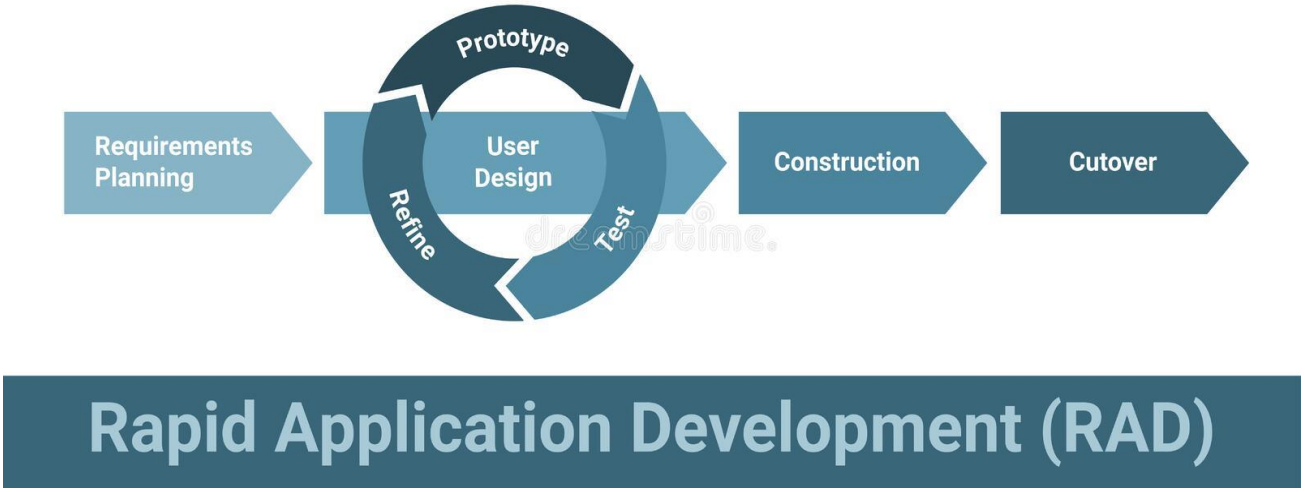

Рисунок 1.5 – Принцип роботи методології RAD

RAD методологія зазвичай складається з чотирьох етапів: планування вимог, дизайну інтерфейсу користувача, конструювання та впровадження.

На етапі планування визначаються вимоги до системи та обсяг проєкту і встановлюються цілі та завдання проєкту. Також створюється план проєкту, визначаються ризики та проблеми.

На етапі дизайну користувача основна увага приділяється створенню інтерфейсу користувача, який відповідає потребам кінцевих користувачів. Розробляється та створюється прототип інтерфейсу користувача, а кінцеві користувачі надають відгуки про дизайн.

На етапі конструювання основна увага приділяється створенню програмного додатку. Використовуються відгуки кінцевих користувачів, щоб створити ітеративний цикл створення та тестування програмного додатку.

На етапі впровадження основна увага приділяється переходу програмного додатку від розробки до виробництва. Виконується остаточне тестування, створюється документація для користувача та відбувається процес навчання кінцевих користувачів роботі з новою програмою.

Основні причини чому методологія RAD є найкращим рішення для розробки даної інформаційної системи:

- Швидкість: як випливає з назви, RAD зосереджується на швидкій розробці та швидкому постачанні робочого програмного забезпечення. Він використовує ітераційний підхід із короткими циклами розробки для швидкого створення та вдосконалення програми. Це може бути ідеальним для проекту з жорсткими часовими рамками, дозволяючи швидко запустити робочу версію програми.
- Гнучкість: RAD дуже адаптивний і гнучкий, що робить його добре придатним для невеликих проектів, які можуть вимагати частих змін або оновлень. Він дозволяє швидко створювати прототипи та відгуки користувачів для прийняття інформативних дизайнерських рішень.
- Економічна ефективність: Оскільки RAD наголошує на швидкості та гнучкості, це може бути економічно ефективнішою методологією розробки для невеликих проектів, яким і є веб-додаток для контролю за веденням документації в медичних закладах. Зосереджуючись на швидкому створенні робочої версії програми, RAD може зменшити ризик дорогих змін або затримок у подальшому процесі розробки.

Загалом методологія RAD наголошує на швидкості та гнучкості розробки програмного забезпечення, і її часто використовують для проектів малого та середнього розміру зі стислими термінами. Саме тому цю методологію було обрано для розробки інформаційної системи для контролю за веденням документації в медичних закладах.

1.6 Проєктування архітектури інформаційної системи

Клієнт-серверна архітектура є популярним підходом для розробки веб-додатків, оскільки вона надає багато переваг. Трирівнева архітектура, зокрема, широко використовується завдяки її перевагам над іншими. Ця архітектура розділяє проблеми інтерфейсу користувача, логіки програми та зберігання даних. Це розділення дозволяє спростити розробку, тестування та підтримку веб-додатку. Компоненти клієнта, сервера та бази даних можна розробляти та тестувати незалежно, що зменшує ризик помилок у кінцевому продукті [8].

Клієнт-серверна архітектура забезпечує масштабованість і гнучкість у розгортанні. Сервер можна збільшити або зменшити залежно від попиту, а клієнт можна розробити для різних платформ, таких як настільні комп'ютери або мобільні пристрої, не впливаючи на роботу сервера. Ця архітектура також передбачає можливість балансування навантаження, яка розподіляє навантаження обробки між декількома серверами, забезпечуючи високу доступність і надійність програми.

Важливою перевагою клієнт-серверної архітектури є те, що вона забезпечує покращену безпеку порівняно з іншими архітектурами. Серверний компонент виконує керування процесами та реалізує бізнес-логіку та правила, гарантуючи, що конфіденційна логіка програми та дані не піддаються клієнту. Такий підхід знижує ризик несанкціонованого доступу до програми та даних.

Можливість наявності великої кількості незалежних користувачів у системі одночасно є однією з головних переваг архітектури клієнт-сервер, особливо для веб-додатків. Архітектура клієнт-сервер дозволяє кільком користувачам одночасно отримувати доступ до одних і тих же самих даних або ресурсів на сервері, не втручаючись у роботу один одного. Це означає, що сервер може керувати кількома запитами від різних клієнтів і координувати їх у режимі реального часу, гарантуючи, що кожен клієнт отримає відповідну відповідь і жодні дані не будуть втрачені чи пошкоджені. Крім того, сервер може бути сконструйований для обробки зростаючої кількості клієнтів і запитів. Це також дозволяє легко

обслуговувати дану систему, оскільки апаратне та програмне забезпечення сервера можна оновлювати, не впливаючи на пристрої клієнтів.

Підсумовуючи, трирівнева клієнт-серверна архітектура є надійним підходом до розробки веб-додатків, основні переваги включають поділ проблем, масштабованість, гнучкість і покращену безпеку. Ці та інші фактори роблять дану архітектуру чудовим вибором для розробки будь-яких веб-орієнтованих додатків.

1.7 Вибір технологій розробки

Вибір технологій розробки відіграє вирішальну роль в успішній реалізації програмного проекту. У сучасному середовищі, що швидко розвивається, вибір правильних технологій для веб-додатку має вирішальне значення для забезпечення ефективної розробки, оптимальної продуктивності та масштабованості. Для побудови як серверної так і клієнтської частини було обрано технології які базуються на основі JavaScript, а саме для серверної частини буде використовуватись технологія Node.js який слугує для можливості виконання коду на JavaScript за межами браузера.

JavaScript – це широко поширена мова програмування, яка в основному використовується для створення динамічних веб-сайтів і веб-додатків. Вона у основному використовується для виконання сценаріїв на клієнтській стороні, що означає, що вона працює у веб-браузері та використовується для додавання інтерактивності, динамічного вмісту та інших інтерактивних функцій [9]. JavaScript підтримує широкий спектр парадигм програмування, включаючи процедурний, об’єктно-орієнтований та функціональний стилі програмування. Він має велику екосистему бібліотек і фреймворків, які надають додаткові функції та оптимізують процеси розробки, що робить його потужним інструментом для створення веб-додатків різної складності.

Останнім часом JavaScript також набув популярності як мова для розробки серверних частин завдяки появі таких технологій, як Node.js, які дозволяють використовувати його на стороні сервера для створення серверних програм і API. Це дозволило розробникам писати код як для зовнішніх, так і для внутрішніх компонентів веб-додатків, використовуючи одну мову.

Node.js – це асинхронне подієве середовище виконання JavaScript на стороні сервера з відкритим кодом, яке дозволяє розробникам запускати код JavaScript поза веб-браузером. Воно побудоване на механізмі JavaScript V8, цей рушій був розроблений компанією Google, і забезпечує керовану подіями неблокуючу модель вводу-виводу, що робить його високо масштабованим і ефективним для створення програм на стороні сервера [10].

Однією з ключових особливостей Node.js є його здатність обробляти велику кількість одночасних з’єднань без блокування циклу подій, що робить його добре придатним для створення додатків, які вимагають зв’язку з сервером в режимі реального часу, таких як додатки для чату, онлайн-ігри або інші інтерактивні веб-застосунки.

Окрім масштабованості та продуктивності, Node.js має велику екосистему модулів і пакетів, доступних через Node Package Manager (NPM), який є найбільшим реєстром пакетів для будь-якої мови програмування. Це дозволяє розробникам легко інтегрувати бібліотеки та модулі сторонніх розробників у свої програми, прискорюючи розробку та підвищуючи продуктивність.

Для створення інтерфейсу користувача буде використано таку JavaScript бібліотеку як React.

React – це бібліотека JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача. Розроблена ця бібліотека була компанією Meta, та отримала широке визнання в спільноті веб-розробників за її ефективний і декларативний підхід до створення компонентів інтерфейсу користувача. React базується на архітектурі, керованій компонентами, де компоненти інтерфейсу користувача створюються як будівельні блоки, які можна комбінувати разом для створення складних інтерфейсів користувача. Кожен компонент React інкапсулює

власний стан і поведінку, що робить його модульним і простим у обслуговуванні. React дотримується моделі односпрямованого потоку даних, де зміни стану компонентів викликають оновлення інтерфейсу користувача, що забезпечує передбачуваний і ефективний процес візуалізації [11].

Однією з ключових особливостей React є його віртуальна реалізація Document Object Model, яка є представленням фактичного DOM. React використовує цей віртуальний DOM для ефективного оновлення лише тих частин інтерфейсу користувача, які потрібно змінити, замість повторного відтворення всього інтерфейсу. Це призводить до кращої продуктивності та покращення взаємодії з користувачем, особливо в програмах із частими оновленнями інтерфейсу користувача.

React має велику та активну спільноту, яка надає обширну документацію, навчальні посібники та багату екосистему сторонніх бібліотек та інструментів завдяки реєстру npm. Це робить його популярним вибором для веб-розробників.

Для виконання запитів від клієнта до сервера буде використано GraphQL API а також бібліотеки для роботи з цією мовою запитів під назвою: Apollo Client та Apollo Server.

GraphQL – це мова запитів для API та середовище виконання для виконання цих запитів. Вона була розроблена компанією Meta і надає більш ефективну, потужну та гнучку альтернативу традиційним REST API. За допомогою GraphQL клієнти можуть запитувати лише ті дані, які їм потрібні, а відповіді сервера адаптуються до конкретних потреб клієнта. GraphQL також надає потужну систему типів, можливості самоаналізу та вбудовану підтримку для підписок у реальному часі, що робить його популярним вибором для сучасної розробки API.

Apollo Client – це потужна та гнучка бібліотека JavaScript, яка надає комплексне рішення для керування і обробки отриманих даних. Вона спеціально розроблена для роботи з API GraphQL і дозволяє розробникам легко взаємодіяти з цією технологією зі сторони клієнта. Apollo Client надає такі функції, як кешування, локальне керування станом, виконання запитів і мутацій, обробка помилок і

підтримка підписки, що робить його популярним вибором для впровадження функціональних можливостей клієнта GraphQL у веб-додатках [12].

Apollo Server – це реалізація сервера GraphQL, яка дозволяє розробникам визначати API GraphQL для своїх програм. Він забезпечує надійне і розширюване середовище виконання на стороні сервера для використання запитів і мутацій GraphQL, обробки підписок і обробки даних з різних джерел даних, таких як бази даних, API та інші служби. Сервер Apollo підтримує широкий спектр джерел даних і дозволяє розробникам впроваджувати користувацькі резолвери для обробки пошуку та мутацій даних, що робить його гнучким і потужним інструментом для створення GraphQL API [12].

Для створення адаптивного та візуально привабливого інтерфейсу користувача будуть використані такі технології як Bootstrap, а саме бібліотека React-Bootstrap, яка використовує компонентний підхід для легкої адаптації з бібліотекою React, а також буде використано scss для створення унікальних стилів для веб-додатку.

Bootstrap – це популярна платформа з відкритим кодом, яка надає набір компонентів CSS для створення розробки веб-додатків, орієнтованих на різні пристрої. Вона містить попередньо розроблені компоненти інтерфейсу користувача, такі як кнопки, форми, панелі навігації тощо, які можна легко налаштувати та інтегрувати у веб-проекти, що допомагає швидко й ефективно створювати сучасні та візуально привабливі користувацькі інтерфейси [13].

React-Bootstrap – це бібліотека, яка поєднує потужність Bootstrap із гнучкістю React для створення інтерфейсів користувача. React-Bootstrap дотримується тих самих принципів і стилів дизайну, що й Bootstrap, але використовуючи React-компоненти і підхід віртуального відтворення DOM. Це означає, що розробники можуть використовувати компоненти React-Bootstrap як звичайні React-компоненти, маючи можливість передавати властивості, керувати станом і обробляти події, що полегшує розробку.

SCSS – це популярний препроцесор CSS, який розширює можливості стандартного CSS, додаючи такі функції, як змінні, вкладення, підтримку

математичних операцій тощо, що робить його потужним і гнучким інструментом для написання підтримуваного та масштабованого коду CSS.

Серед головних переваг SCSS над CSS можна виділити наступні:

- Змінні, які можуть зберігати такі значення, як кольори, розмір шрифту, поля тощо. Ці змінні можна використовувати у всіх файлах SCSS, що полегшує централізоване керування та оновлення стилів.
- Вкладеність яка дозволяє вкладати селектори один в інший, що може допомогти зменшити кількість повторів і зробити код CSS більш упорядкованим і читабельним.
- Імпорт SCSS який дозволяє розділити код на менші файли, що називаються частинами, які можна імпортувати в інші файли.
- Підтримка математичних операцій і керуючих директив, таких як цикли та умови, які можуть допомогти створити динамічні та гнучкі стилі.

Для створення бази даних буде використовуватись Microsoft SQL Server.

Microsoft SQL Server — це СУБД, розроблена компанією Microsoft. Це програмний продукт, який дозволяє створювати, керувати та надсилати запити до баз даних у структурований та організований спосіб. Дана СУБД також забезпечує розширені функції для зберігання даних, пошуку, безпеки, масштабованості, продуктивності, бізнес-аналітики та інтеграції з іншими технологіями Microsoft. Вона широко використовується в додатках різних рівнів для ефективного та безпечного керування та обробки великих обсягів даних [14].

1.8 Проєктування архітектури бази даних

Веб-додаток для контролю за веденням документації в медичному закладі буде використовувати потужну та широко поширеної реляційну систему управління базами даних Microsoft SQL Server. Використовуючи цю технологію, програма отримує можливість створювати та ефективно керувати реляційними

базами даних, які призначені для організації та структурування даних у табличному форматі, що складається з рядків і стовпців, а зв’язки між таблицями встановлюються за допомогою унікальних ключів [15].

Серед основних переваг даної СУБД можна виділити наступні:

- Забезпечення ефективними та масштабованими механізми для зберігання, отримання та обробки даних, включаючи підтримку різних типів даних, індексування та оптимізацію запитів.
- Microsoft SQL Server включає надійні функції безпеки для захисту конфіденційних даних, такі як автентифікація, авторизація, шифрування та аудит, гарантуючи безпечне зберігання та доступ до даних.
- Масштабованість і продуктивність, дана СУБД розроблена для обробки великих обсягів даних і високого рівня одночасного доступу користувачів, що робить її придатним для програм корпоративного рівня з високими вимогами до продуктивності. Вона також включає такі функції, як обробка запитів в пам’яті, розділення та кешування для оптимізації продуктивності.
- Можливість інтеграції з іншими технологіями компанії Microsoft такими як хмарні служби Azure, .NET Framework, Visual Studio та Power BI, що забезпечуючи комплексну та уніфіковану платформу для розробки та розгортання програм.
- Висока доступність і можливість аварійного відновлення. Microsoft SQL Server включає такі функції, як кластеризація після відмови, дзеркальне відображення бази даних і групи доступності, які надають параметри для високої доступності та аварійного відновлення, гарантуючи, що бази даних буде можливо відновити у разі апаратних збоїв або інших аварій.

Після того, як було проведено детальні дослідження предметної області, було визначено головні сутності які повинні бути представлені в БД, для правильного функціонування системи. БД буде складатись з наступних таблиць: Users, Patients, Visits, SOAP, Note_Templates,

Розглянемо детальніше таблицю для зберігання даних про користувача, вона складається з полів в яких має зберігатись наступна інформація: прізвище та ім’я

користувача, закодований пароль для входу в акаунт, адреса електронної пошти, роль користувача, дата створення акаунту та дата останнього входу в систему, а також дані про те, чи акаунт було заблоковано.

Схему таблиці в базі даних для сутності користувача можна побачити на рисунку 1.6.

User	
id	uniqueidentifier
givenName	nvarchar(100)
surname	nvarchar(100)
displayName	nvarchar(200)
hashPassword	nvarchar(MAX)
email	nvarchar(100)
role	nvarchar(8)
createdAt	datetime
lastLoginDate	datetime
deactivated	bool

Рисунок 1.6 – Схема таблиці користувача в базі даних

Сутність користувача містить такі поля:

- id – унікальний ключ користувача;
- givenName – ім’я користувача;
- surname – прізвище користувача;
- displayName – назва користувача, за замовчуванням це буде поєднання імені та прізвища але користувач буде мати змого його змінити.
- hashPassword – хешований пароль;
- email – адреса електронної пошти;
- role – роль користувача в системі;

- createdAt – дата та час створення акаунта;
- lastLoginDate – дата та час останнього входження в систему;
- deactivated – змінна яка позначає те, чи акаунт було деактивовано.

Також однією з основних сутностей системи є пацієнт. Дана сутність призначена для зберігання даних про пацієнта в системі, вона має зберігати наступну інформацію: прізвище та ім’я пацієнт, унікальний номер пацієнта в медичному закладі, стать та дату народження, інформацію про те до якого лікаря відноситься даний пацієнт, також дані про дату та час створення запису про пацієнта в системі і дату оновлення даних якщо такі були. Якщо запис про пацієнта було за архівовано, то в даній сутності мають зберегтись дані про те коли та ким це було зроблено.

Схему для сутності пацієнта зображено на рисунку 1.7.

Patient	
id	uniqueidentifier
providerId	uniqueidentifier
firstName	nvarchar(50)
lastName	nvarchar(50)
patientNumber	nvarchar(50)
gender	nvarchar(8)
dob	date
archived	bool
archivedAt	datetime
archivedBy	uniqueidentifier
createdBy	uniqueidentifier
createdAt	datetime
updatedAt	datetime

Рисунок 1.7 – Схема таблиці пацієнта в базі даних

Сутність пацієнта містить наступні поля:

- id – унікальний ключ пацієнта;
- providerId – унікальний ідентифікатор лікаря, до якого відноситься пацієнт;
- firstName – ім’я пацієнта;
- lastName – прізвище пацієнта;
- patientNumber – унікальний номер пацієнта в медичному закладі;
- gender – стать пацієнта;
- dob – дата народження;
- archived – змінна яка позначає те, чи дані про пацієнта було архівовано;
- archivedAt – дата та час коли дані про пацієнта було архівовано;
- archivedBy – унікальний ідентифікатор користувача який за архівував дані про пацієнта;
- createdBy – унікальний ідентифікатор користувача який створив запис про пацієнта в базі даних;
- createdAt – дата та час коли запис було створено;
- updatedAt – дата та час коли запис було оновлено останній раз.

Наступною сутністю яку необхідно розглянути буде візит. Сутність візиту призначена для зберігання даних про візит пацієнта до лікаря. Дана сутність повинна зберігати таку інформацію як: дату візиту, причину та скаргу пацієнта через яку необхідно провести візит, статус в якому перебуває візит, а також нотатки в які лікар може записувати свої спостереження.

Схему для сутності візиту зображено на рисунку 1.8.

Visit	
id	uniqueidentifier
providerId	uniqueidentifier
patientId	uniqueidentifier
status	nvarchar(18)
statusUpdateDate	datetime
reason	nvarchar(18)
complaint	nvarchar(MAX)
visitDate	datetime
patientArchived	bool
archived	bool
archivedAt	datetime
archivedBy	uniqueidentifier
createdBy	uniqueidentifier
createdAt	datetime
updatedAt	datetime
note	nvarchar(MAX)

Рисунок 1.8 – Схема таблиці візиту в базі даних

Сутність візиту містить наступні поля:

- id – унікальний ідентифікатор візиту;
- providerId – унікальний ідентифікатор лікаря, до якого відноситься візит;
- patientId – унікальний ідентифікатор пацієнта, до якого відноситься візит;
- status – статус в якому перебуває візит;
- statusUpdateDate – дана та час останнього оновлення статусу візиту;
- reson – причина візиту;
- complaint – скарга пацієнта через яку потрібно провести візит;
- visitDate – дата проведення візиту;
- patientArchived – змінна яка позначає те, чи дані про пацієнта який відноситься до даного візиту було архівовано;
- archived – змінна яка позначає те, чи дані про візит було архівовано;

- archivedAt – дата та час коли дані про візит було архівовано;
- archivedBy – унікальний ідентифікатор користувача який за архівував дані про візит;
- createdBy – унікальний ідентифікатор користувача який створив даний візит в базі даних;
- createdAt – дата та час коли запис було створено;
- updatedAt – дата та час коли запис було оновлено останній раз;
- note – нотатки які лікар може заповнювати під час візиту.

Важливою сутністю в створюваній системі є SOAP, вона призначення для того, щоб медичний працівник мав змогу зберегти в базі даних свій суб'єктивний аналіз та оцінку стану здоров'я пацієнта, а також мету та план лікування.

Схему для сутності SOAP зображено на рисунку 1.9.

SOAP	
id	uniqueidentifier
providerId	uniqueidentifier
patientId	uniqueidentifier
visitId	uniqueidentifier
subjective	nvarchar(MAX)
objective	nvarchar(MAX)
assessment	nvarchar(MAX)
plan	nvarchar(MAX)
createdBy	uniqueidentifier
createdAt	datetime

Рисунок 1.9 – Схема таблиці для сутності SOAP в базі даних

Сутність SOAP містить наступні поля:

- id – унікальний ідентифікатор сутності;

- providerId – унікальний ідентифікатор лікаря до якого відноситься SOAP;
- patientId – унікальний ідентифікатор пацієнта для якого було створено SOAP;
- visitId – унікальний ідентифікатор візиту до якого відноситься SOAP;
- subjective – суб’єктивний аналіз стану здоров’я пацієнта;
- objective – мета лікування;
- assessment – оцінка стану здоров’я пацієнта;
- plan – план лікування;
- createdBy – унікальний ідентифікатор користувача який створив даний SOAP в базі даних;
- createdAt – дата та час коли запис було створено.

Наступною сутністю яку необхідно розглянути буде шаблон для створення SOAP. Дана сутність призначена для того, щоб зберігати назву шаблону а також дані які будуть необхідні для швидкого створення SOAP. Використання даної сутності дозволить зменшити час який необхідний лікарю для документації нових візитів, оскільки шаблон можна буде створювати на основі уже існуючих документів та редагувати в відповідності до потреб лікаря і використовувати при створенні нових документів.

Схему для сутності шаблону зображено на рисунку 1.10.

Note_Template	
id	uniqueidentifier
name	nvarchar(255)
subjective	nvarchar(MAX)
objective	nvarchar(MAX)
assessment	nvarchar(MAX)
plan	nvarchar(MAX)
createdAt	datetime
updatedAt	datetime
providerId	uniqueidentifier
sourceSoapId	uniqueidentifier
archived	bool
archivedBy	uniqueidentifier
archivedAt	datetime

Рисунок 1.10 – Схема таблиці для сутності шаблону в базі даних

Сутність шаблону містить наступні поля:

- id – унікальний ідентифікатор шаблону;
- name – назву шаблону;
- subjective – шаблон для суб’єктивного аналізу стану здоров’я пацієнта;
- objective – шаблон мети лікування;
- assessment – шаблон оцінки стану здоров’я пацієнта;
- plan – шаблон плану лікування;
- createdAt – дата та час коли запис було створено;
- updatedAt – дата та час останнього оновлення запису;
- providerId – унікальний ідентифікатор лікаря який створив шаблон;
- sourceSoapId – унікальний ідентифікатор сутності SOAP з якої було створено даний шаблон;
- archived – змінна яка позначає те, чи шаблон було архівовано;

- archivedBy – унікальний ідентифікатор користувача який архівував шаблон;
- archivedAt - дата та час коли шаблон було архівовано.

Після того, як основні сутності системи були визначені та описані, була розроблена діаграма бази даних (рис. 1.11).

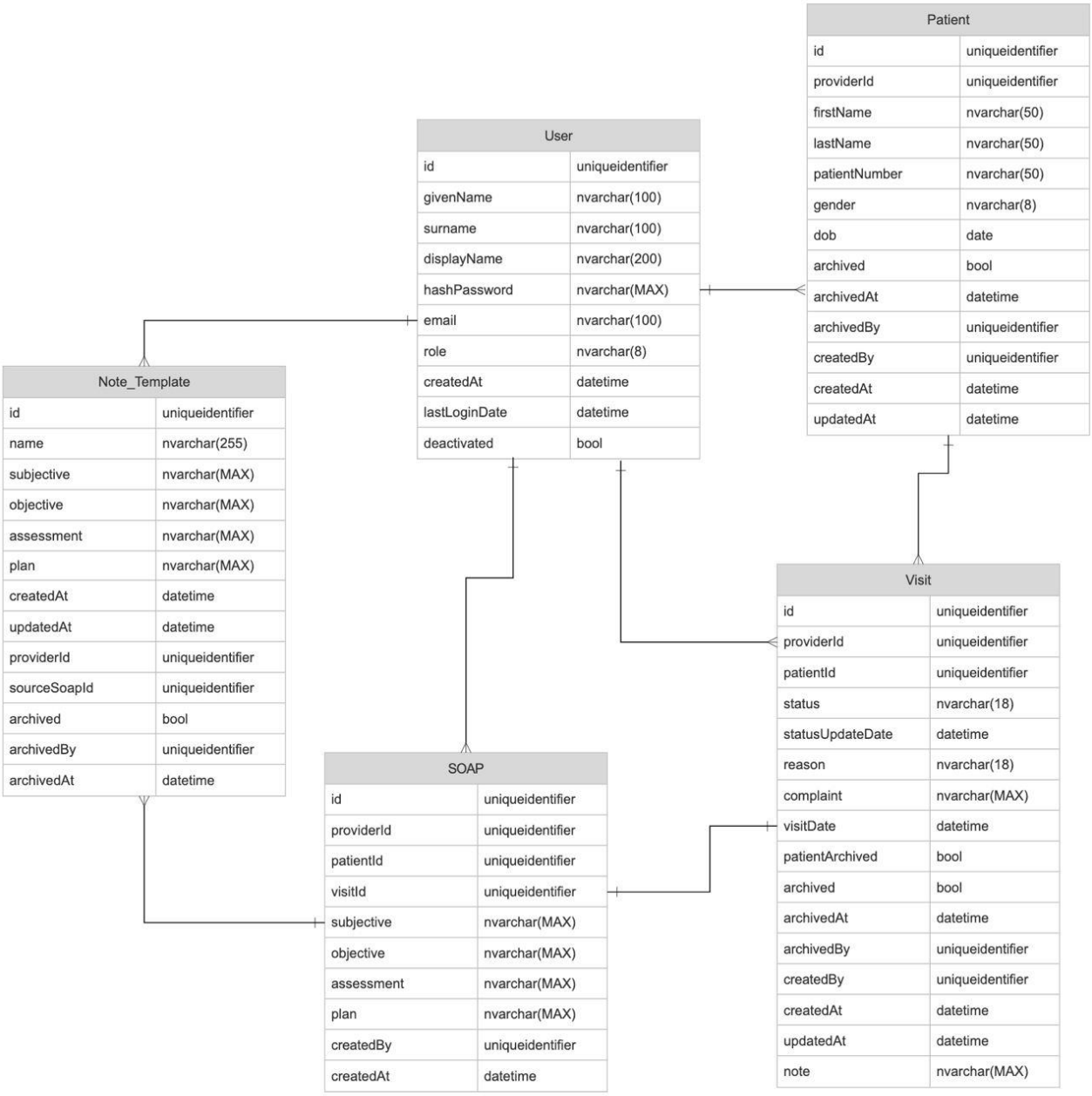

Рисунок 1.11 – Діаграма бази даних

На діаграмі бази даних відображено п’ять сутностей та зв’язки між ними, які допомагають краще зрозуміти структуру бази даних.

На діаграмі бази даних зображені такі відношення:

- Сутність користувача має відношення один до багатьох з сутністю пацієнта, оскільки користувач, в даному випадку лікар може створювати безліч записів про своїх пацієнтів в системі.
- Сутність користувача має відношення один до багатьох з сутністю візиту, оскільки лікар може створювати велику кількість записів про візити.
- Сутність користувача має відношення один до багатьох з сутністю SOAP, оскільки лікар може створювати для кожного візиту пацієнта запис з даними про стан здоров’я, мету та план лікування.
- Сутність користувача має відношення один до багатьох з сутністю шаблону, оскільки користувач може створювати безліч шаблонів для полегшення процесу документації.
- Сутність пацієнту має відношення один до багатьох з сутністю візиту, оскільки для кожного пацієнта можна запланувати безліч візитів.
- Сутність візиту має відношення один до одного з сутністю SOAP, оскільки для кожного візиту можна створити тільки один запис з даними про стан здоров’я, мету та план лікування пацієнта.
- Сутність SOAP має відношення один до багатьох з сутністю шалону, оскільки для кожного запису про план лікування пацієнта можна створити велику кількість шаблонів, щоб в подальшому полегшити процес документації.

1.9 Проєктування класів серверної частини

Для візуалізації архітектури класів серверної частини буде побудовано діаграму. Для забезпечення модульності системи класи загалом будуть поділені на класи-моделі – це класи які призначені для роботи з базою даних та класи-сервіси, які призначені для виконання бізнес логіки. Оскільки класи-сервіси використовуються в резолверах які і є серверним API, то діаграма буде лише

абстрактним відображенням системи і призначена вона для загально розуміння архітектури класів.

Діаграма класів серверної частини зображена на рисунку 1.12.

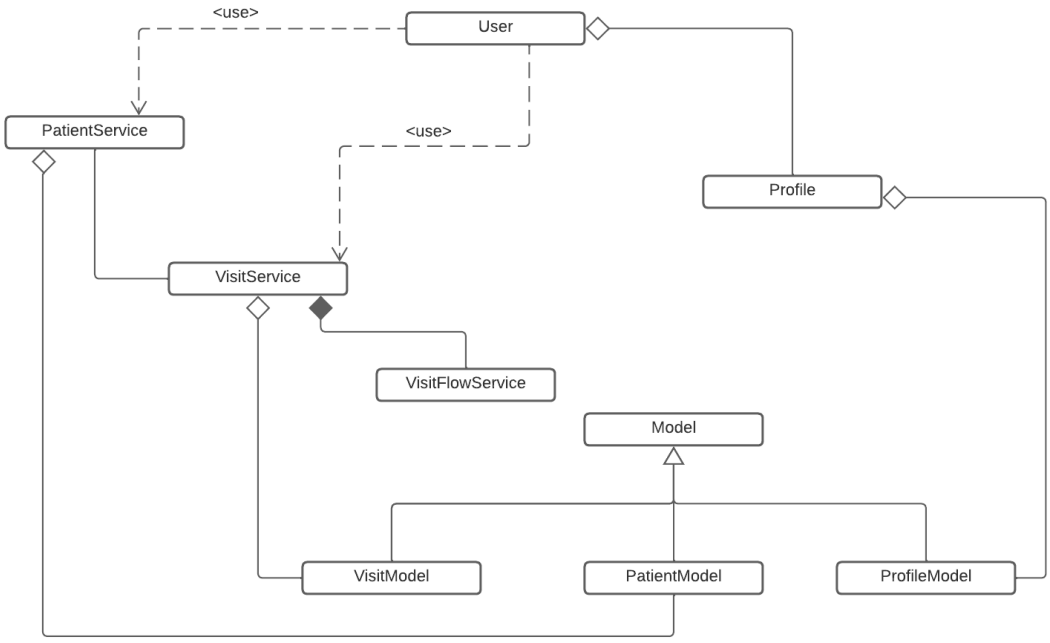

Рисунок 1.12 – Діаграма класів серверної частини

На рисунку вище зображено класи які відповідають за функціонування серверної частини веб-додатку. Основним класом є User, він використовує PatientService та VisitService для роботи з пацієнтами та візитами відповідно, також частиною цього класу є Profile який створений для роботи з даними профілю користувача. Класи моделі такі як VisitModel, PatientModel та ProfileModel, успадковуються від базового Model який містить основні методи для роботи з базою даних. PatientService пов'язаний зв’язком асоціації з класом VisitService, який містить в собі VisitFlowService – цей клас відповідає за правильність переходу документів між станами. Більш детально про основні класи серверної частини, їх методи та атрибути написано в наступній частині кваліфікаційної роботи.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Реалізація основних класів серверної частини

Після проведення ретельного аналізу предметної області та розробки архітектури бази даних і програмної системи розпочато процес реалізації ключових класів серверної частини. Цей розділ має на меті представити основні методи та класи, які використовуються в серверній частині. Варто зазначити, що повний код реалізації серверної частини можна знайти в додатках.

Першим класом який буде розглянуто, це клас під назвою UserService – він призначений для реалізації бізнес логіки по роботі з сутністю користувача, та містить методи які дозволяють створювати, змінювати, активувати та деактивувати облікові записи всі користувачів а також метод для присвоєння ролей. Клас UserService також має методи для отримання даних про всіх користувачів в системі та конкретного користувача за унікальним ідентифікатором або адресою електронної пошти.

Одним з основних методів класу UserService є метод для створення акаунту користувача, він буде виконуватись коли користувач буде реєструватись в системі, код даного методу наведено на рисунку 2.1.

```
async createUser(user) {
  const doc = this.normalizeUserObject(user);
  Logger.info(`[UserService] Creating new user: ${JSON.stringify(user, null, 2)}`);
  const { count } = await UsersModel.findUsers({ onlyCount: true });
  const role = count > 0 ? roles.GUEST : roles.ADMIN;

  const userDoc = await UsersModel.insert({ ...doc, role }, UsersModel.fields);

  Logger.info(`[UserService] User created: ${doc.id}`);
  AuthEvents.signUp(userDoc);

  if (role === roles.DOCTOR) {
    await ProviderTablesInitializer.initProviderTables(userDoc.id);
  }

  return userDoc;
}
```

Рисунок 2.1 – Програмний код методу для створення користувача

Даний метод отримує об’єкт користувача, приводить його но форми яка має бути збережена в базі даних використовуючи інший метод під назвою normalizeUserObject, якщо це перший користувач який створюється в системі, то йому буде присвоєна роль адміністратора. Якщо даний користувач є лікарем, то для нього буде створено необхідні таблиці в базі даних.

Клас PatientsService є важливим класом серверної частини оскільки він відповідає за роботу з сутністю пацієнта, та містить методи для створення, оновлення та видалення пацієнта з системи, а також методи для отримання пацієнта по унікальному ідентифікаторі, та отримання всі пацієнтів для заданого лікаря.

Головним методом класу PatientsService є метод для створення пацієнтів код даного методу наведено на рисунку 2.2.

```
create = async (params, ctx) => {
  const { user } = ctx;

  const { firstName, lastName, patientNumber, gender, dob } = params;
  const providerId = user.id;

  const data = { providerId, firstName, lastName, patientNumber, gender, dob, createdBy: user.id };
  Logger.info(`[PatientsService] Creating patient: ${JSON.stringify(data, null, 2)}`);
  const id = await PatientsModel.insert({ providerId, data });
  Logger.info(`[PatientsService] Created patient: ${id}`);
  return id;
};
```

Рисунок 2.2 – Програмний код методу для створення пацієнта

Даний метод отримує два об’єкти в першому знаходяться дані про пацієнта який необхідно створити, а другий це контекст, в якому знаходяться дані про користувача який намагається створити пацієнта. Після того як дані для збереження в базу даних сформовано, вони зберігаються, а метод повертає унікальний ідентифікатор створеного об’єкта.

Клас VisitsService призначений для виконання бізнес логіки над сутністю візиту, та містить методи для створення, оновлення та видалення візиту з системи, а також методи для отримання візиту по унікальному ідентифікаторі, та отримання всіх візитів для лікаря або пацієнта.

Основним методом класу VisitsService є метод для створення візитів, код даного методу наведено на рисунку 2.3.

```
create = async (params, ctx) => {
  const { user } = ctx;

  const providerId=user.id;

  const { visitDate, reason, complaint, patientId } = params;

  const data = {
    providerId,
    patientId,
    status: this.initialStatus,
    reason,
    complaint,
    visitDate,
    createdBy: user.id,
  };
  Logger.info(`[VisitsService] Creating visit: ${JSON.stringify(data, null, 2)}`);
  const visit = await VisitsModel.insert({ providerId, data, returning: VisitsModel.fields });

  VisitFlowService._transition({ providerId, visitId: visit.id, prevStatus: null, status: this.initialStatus, userId: user.id });
  Logger.info(`[VisitsService] Created visit: ${visit.id}`);
  return visit;
};
```

Рисунок 2.3 – Програмний код методу для створення візиту

Даний метод отримує два об’єкти в першому знаходяться дані про візит який необхідно створити, а другий це контекст, в якому знаходяться дані про користувача який намагається створити візит. Після того, як дані для візиту сформовано та збережено в базу даних, викликається метод для того, щоб присвоїти візиту певний статус та контролювати правильний перехід між статусами візиту, вкінці повертається створений об’єкт.

Клас SOAPService призначений для роботи з сутністю SOAP – яка існує для опису стану здоров’я пацієнта, мети та плану лікування. Даний клас складається з методів для створення та підтвердження SOAP, а також методів для отримання SOAP по унікальному ідентифікаторі або пацієнті.

Основним методом класу SOAPService є метод для створення SOAP, код даного методу наведено на рисунку 2.4.

```
createSOAP = async (params) => {
  const { subjective, objective, assessment, plan, visitId, providerId, patientId, user } = params;
  Logger.info(`[SOAPService] Creating SOAP: ${JSON.stringify({ subjective, objective, assessment, plan }, null, 2)}`);
  const id = await SOAPModel.insert({
    providerId,
    data: {
      visitId,
      providerId,
      patientId,
      subjective,
      objective,
      assessment,
      plan,
      createdBy: user.id,
    },
  });
  Logger.info(`[SOAPService] Created SOAP: ${id}`);
  return id;
};
```

Рисунок 2.4 – Програмний код методу для створення SOAP

Даний метод отримує об’єкт в якому містяться дані необхідні для створення SOAP після чого він зберігається в базі даних, та повертається унікальний ідентифікатор збереженого об’єкта.

Клас NoteTemplatesService слугує для роботи з сутністю шаблону – яка необхідна для полегшення процесу створення документів. Даний клас містить методи для створення, оновлення та видалення шаблонів, а також методи для отримання шаблонів для конкретного лікаря, та отримання шаблону по унікальному ідентифікаторі.

Основним методом даного класу NoteTemplatesService метод для створення шаблону, код даного методу наведено на рисунку 2.5.

```
createNoteTemplate = async ({ data, user }) => {
  const { name, subjective, objective, assessment, plan } = data;
  Logger.info('[NoteTemplatesService] Creating Note Template');
  const noteTemplate = await NoteTemplatesModel.insert(
    {
      name,
      subjective,
      objective,
      assessment,
      plan,
      createBy: user._id,
    },
    NoteTemplatesModel.fields,
  );
  Logger.info(`[NoteTemplatesService] Created Note Template: ${noteTemplate.id}`);
  return noteTemplate;
};
```

Рисунок 2.5 – Програмний код методу для створення шаблону

Даний метод отримує об’єкт в якому містяться дані необхідні для створення шаблону, а також дані про користувача який намагається його створити. Цей метод робить форматування та збереження шаблону в базі даних після чого повертає створений об’єкт.

Коли було завершено створення головних класів системи було розроблено допоміжні класи та функції-утиліти які необхідні для правильного функціонування системи. Важливими класами також є класи-моделі які призначені для роботи з базою даних, та використовуються класами-сервісами.

Важливою частиною сервера є інтерфейс призначений для взаємодії з клієнтом, в даній інформаційні системі для побудови цього інтерфейсу буде використовуватись graphql оскільки він надає більш потужну та гнучку альтернативу традиційним REST API. Також graphql можна розділити на окремі компоненти для кожної з сутностей та описати їх типи, а вбудована система валідації даних буде відбуватись автоматично згідно з описаними типами, що значно полегшує та пришвидшує процес розробки.

Для створення API використовуючи технологію graphql, необхідно спочатку описати схему з типами, а потім створити об’єкт який буде містити резолвери, які і є реалізацією програмного інтерфейсу. Розглянемо graphql-схему яка є частиною API і використовується для роботи з об’єктом шаблону (рис. 2.6).

```
type NoteTemplate {
  id: ID!
  name: String!
  subjective: String!
  objective: String!
  assessment: String!
  plan: String!
  createdAt: DateTime!
  updatedAt: DateTime!
}
type PaginatedNoteTemplate {
  count: Int!
  data: [NoteTemplate]!
}
input CreateNoteTemplateInput {
  name: String!
  subjective: String!
  objective: String!
  assessment: String!
  plan: String!
}
input UpdateNoteTemplateInput {
  name: String
  subjective: String
  objective: String
  assessment: String
  plan: String
}

extend type Query {
  noteTemplate(id: ID!): NoteTemplate! @auth(requires: ${roles.DOCTOR})
  noteTemplates(limit: Int!, skip: Int!): PaginatedNoteTemplate! @auth(requires: ${roles.DOCTOR})
}
extend type Mutation {
  createNoteTemplate(data: CreateNoteTemplateInput!): NoteTemplate! @auth(requires: ${roles.DOCTOR})
  updateNoteTemplate(data: UpdateNoteTemplateInput!, id: ID!): NoteTemplate! @auth(requires: ${roles.DOCTOR})
  deleteNoteTemplate(id: ID!): Boolean @auth(requires: ${roles.DOCTOR})
}
```

Рисунок 2.6 – Graphql-схема для роботи з об’єктом шаблону

Дана схема описує частину API яка призначена для роботи з шаблоном, в ній описано тип наступні типи:

- NoteTemplate – який є типом шаблона.
- PaginatedNoteTemplate – цей тип містить в собі масив шаблонів та їх кількість і призначений для використання в таблицях.

Також в схемі описані типи які необхідні для створення та оновлення даних про шаблон. Окремими типами схеми є запити, для отримання даних, та мутації для їх оновлення.

Об’єкт з резолверами який є реалізацією частини API для шаблонів зображено на рисунку 2.7.

```
export const resolvers = {
  Query: {
    noteTemplate: async (root, { id }, ctx) => {
      const { userId } = ctx;
      const noteTemplate = await NoteTemplateService.findById({ userId, id });
      return noteTemplate;
    },
    noteTemplates: async (root, params, ctx) => {
      const { userId } = ctx;
      const noteTemplates = NoteTemplateService.findNoteTemplates({ params, userId });
      return noteTemplates;
    },
  },
  Mutation: {
    createNoteTemplate: async (root, { data }, ctx) => {
      const { userId } = ctx;
      const noteTemplate = await NoteTemplateService.createNoteTemplate({ data, userId });
      return noteTemplate;
    },
    updateNoteTemplate: async (root, params, ctx) => {
      const { userId } = ctx;
      return NoteTemplateService.updateVisitById({ params, userId });
    },
    deleteNoteTemplate: async (root, { id }, ctx) => {
      return NoteTemplateService.removeNoteTemplateById({ id, ctx });
    },
  },
};
```

Рисунок 2.7 – Реалізація резолверів для об’єкта шаблону

В даному об’єкті знаходяться резолвери з запитами для отримання даних та мутації для оновлення, видалення та створення шаблонів.

Вибір технологій для реалізації серверної частини таких як: GraphQL, Apollo Server та Microsoft SQL Server надають значну кількість переваг, зокрема покращену ефективність пошуку даних, спрощене керування даними, зв’язок у реальному часі та велику екосистему інструментів і ресурсів. Ці переваги впливають на покращення продуктивності, масштабованості та швидкості і якості розробки, що робить їх потужною комбінацією для розробки серверної частини даного веб-додатку.

2.2 Реалізація клієнтської частини веб-додатку

Для реалізації клієнтської частини даного веб-додатку буде використано бібліотеку React яка буде використовуватись для побудови графічного інтерфейсу користувача, а також бібліотеку Apollo Client яка буде використана для виконання запитів на сервер, та контролю стану веб-застосунку. Для створення стилів буде

використовуватись бібліотека React-Bootstrap, а також препроцесор для css під назвою scss.

React.js використовує одно-сторінковий підхід (SPA) до створення веб-додатків. На відміну від традиційних веб-додатків сторінки яких генеруються на стороні сервера, SPA, створені за допомогою React.js, створюють і оновлюють вміст на стороні клієнта, не вимагаючи повного завантаження сторінки [16]. Цей підхід має декілька переваг перед генерацією HTML-сторінок на стороні сервера, основні з цих переваг:

- покращена продуктивність, оскільки SPA завантажують необхідні компоненти та дані лише тоді, коли вони потрібні, вони можуть забезпечити швидшу та оперативнішу взаємодію з користувачем.
- Краща взаємодія з користувачем, оскільки SPA забезпечують більш плавні переходи між сторінками, усуваючи потребу в повному оновленні сторінки та забезпечуючи більш плавну роботу веб-застосунку. Крім того, оскільки дані завантажуються динамічно, користувачі можуть взаємодіяти з програмою без перерв.
- Покращена масштабованість, оскільки SPA завантажують лише необхідні компоненти та дані, вони можуть бути більш масштабованими, ніж програми, створені на стороні сервера, які вимагають завантаження всієї сторінки кожного разу, коли користувач взаємодіє з програмою.

Такі переваги роблять SPA потужним підходом для створення сучасних та добре масштабованих веб-додатків.

Базовий компонент App в якому відображаються всі інші компоненти системи, підключається Apollo Client, та провайдер для авторизації в додатку зображено на рисунку 2.8.

```
const App = () => (
  <MsalProvider instance={msalInstance}>
    <ApolloProvider client={apolloClient}>
      <RootRouter />
      <ConfirmAlertContainer />
      <ToastContainer />
    </ApolloProvider>
  </MsalProvider>
);
```

Рисунок 2.8 – Програмний код компонента App

Для навігації між сторінками в даному веб-додатку буде використано бібліотеку під назвою react router dom. React Router забезпечує декларативний спосіб визначення маршрутів і зв’язує їх із конкретними компонентами в додатку, це надає можливість проводити навігацію між сторінками без необхідності повного оновлення сторінки, це дозволяє використовувати SPA підхід, який буде себе вести як багатосторінкові застосунки.

Компонент який відповідає за навігацію в даному веб-додатку було названо RootRouter (рис. 2.9).

```
const RootRouter = () => {
  const [currentUser, loading] = useMeQuery();

  if (loading) {
    return <FullscreenLoading />;
  }
  return (
    <Router>
      <Switch>
        <Route path="/secure">
          <MsalAuthWrapper>
            {currentUser?.role === roles.DOCTOR ? (
              <CurrentProviderContextProvider>
                <DoctorRouter />
              </CurrentProviderContextProvider>
            ) : null}
            {currentUser?.role === roles.ADMIN ? <AdminRouter /> : null}
            {!currentUser ? <Redirect to={paths.logout} /> : null}
          </MsalAuthWrapper>
        </Route>
        <Route>
          <AuthLayout>
            <Switch>
              <Route path={paths.authenticator} exact component={Authenticator} />
              <Route path={paths.logout} exact component={Logout} />
              <Redirect from={paths.home} to={paths.authenticator} exact />
            </Switch>
          </AuthLayout>
        </Route>
      </Switch>
    </Router>
  );
};
```

Рисунок 2.9 – Програмний код компонента RootRouter

В даному компоненті знаходяться поділ на сторінки які доступні авторизованому та не авторизованому користувачу. Для користувача який не увійшов в систему будуть доступні тільки сторінки для реєстрації та авторизації. Для авторизованих користувачів є поділ доступних сторінок в залежності від їх ролі, так адміністратор буде мати доступ до сторінок зі списком користувачів, статистикою, та сторінки з профілем кожного з користувачів системи.

Для лікаря будуть доступні наступні сторінки:

- Сторінка з статистикою лікаря.
- Сторінка зі списком шаблонів для документів.
- Сторінка зі списком документів.
- Сторінка зі списком всі пацієнтів які відносяться до лікаря.
- Сторінка з даними про конкретного пацієнта.
- Сторінка для документації конкретного візиту.

Однією з найважливіших сторінок в веб-додатку є сторінка з даними про пацієнта, програмний код якої зображено на рисунку 2.10.

```
const PatientOverview = () => {
  const { providerId, patientId } = useParams();

  const { patient, loading, error: fetchError } = usePatientById({ providerId, patientId });

  const links = [
    {
      path: generatePath(paths.patients, { providerId }),
      title: 'Patients',
    },
  ];

  const notFoundError = !loading && !patient ? { message: 'Patient not found' } : null;
  const error = fetchError || notFoundError;
  return (
    <>
      <div className="mb-4">
        <Breadcrumb links={links} activeLinkTitle={patient?.fullName} />
      </div>
      <Row className="mb-4">
        <Col lg="4" sm="12">
          {error ? <ErrorAlert error={error} /> : null}
          <PatientOverviewCard {...{ patient, patientId, providerId, loading }} />
        </Col>
        <Col className="ml-5 mt-4 mt-md-0" lg="8" sm="12">
          <PatientVisitsCard {...{ patientId, providerId }} />
        </Col>
      </Row>
    </>
  );
};
```

Рисунок 2.10 – Програмний код сторінки з даними про пацієнта

На даній сторінці знаходиться посилання на сторінку зі списком усіх пацієнтів, та два компоненти PatientOverviewCard – в цьому компоненті розміщується вся інформація про пацієнта, PatientVisitsCard – це компонент який слугує для розміщення таблиці зі задокументованими або запланованими візитами пацієнта.

Стилі які використовуються в компоненті для відображення даних пацієнта в форматі scss зображено на рисунку 2.11.

Рисунок 2.11 – Стилі для сторінки з даними про пацієнта

Використання React та Apollo Client забезпечує потужний і ефективний спосіб створення клієнтської частини веб-додатку. React надає віртуальний DOM, який дозволяє ефективно оновлювати інтерфейс користувача, а Apollo Client надає систему кешування, використання якої зменшує кількість мережевих запитів необхідних для оновлення інтерфейсу користувача. Це дозволяє легко створювати багаторазові компоненти, з яких будується графічний інтерфейс користувача, їх можна використовувати у всій програмі. За допомогою системи кешування та хуків які надає Apollo Client такий процес як отримання даних з сервера легко інтегрується в компоненти React, роблячи їх ще більш гнучкими та зручними для компонування, що дозволяє легко будувати добре масштабовані веб-застосунки.

3 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Перевірка роботи базових функцій серверного API

Для побудови серверного API було використано GraphQL, через значну кількість переваг перед звичайним REST API, основні переваги:

- Зниження мережевого трафіку, за допомогою GraphQL клієнтська частині може запитувати тільки потрібні дані, що може зменшити обсяг мережевого трафіку, необхідного для кожного запиту. Це призводить до швидшого часу завантаження та покращення взаємодії з користувачем.
- Більша гнучкість, за допомогою GraphQL клієнт визначає потрібні йому дані, що забезпечує більшу гнучкість і адаптивність до мінливих вимог.
- Спрощена підтримка API, REST API зазвичай мають кілька кінцевих точок для різних типів даних. Це може призвести до складної кодової бази та ускладнити підтримку та розвиток API з часом. У GraphQL існує єдина кінцева точка для всіх даних, що може спростити кодову базу та полегшити розвиток і підтримку API з часом.

Для тестування серверного API буде використано GraphQL Playground – це графічна IDE для GraphQL яку можна запустити в будь-якому браузері, для того, щоб вона відкрилась потрібно просто перейти на домен на якому запущено сервер, та дописати в адресну стрічку /graphql.

Перший запит який буде перевірено на правильність роботи буде запит на створення пацієнта в системі (рис. 3.1).

Рисунок 3.1 – Виконання запиту на створення пацієнта в системі

Після виконання запиту сервер повернув дані створеного пацієнта, щоб впевнитись в тому, що запис про пацієнта було створено необхідно зробити запит на отримання всіх пацієнтів (рис.3.2).

Рисунок 3.2 – Результат виконання запиту для отримання всіх пацієнтів

В списку який повернувся з сервера знаходиться створений пацієнт, а отже запит на створення пацієнта був виконаний успішно.

Наступним запитом який необхідно перевірити, це запит для створення візиту пацієнта (рис. 3.3).

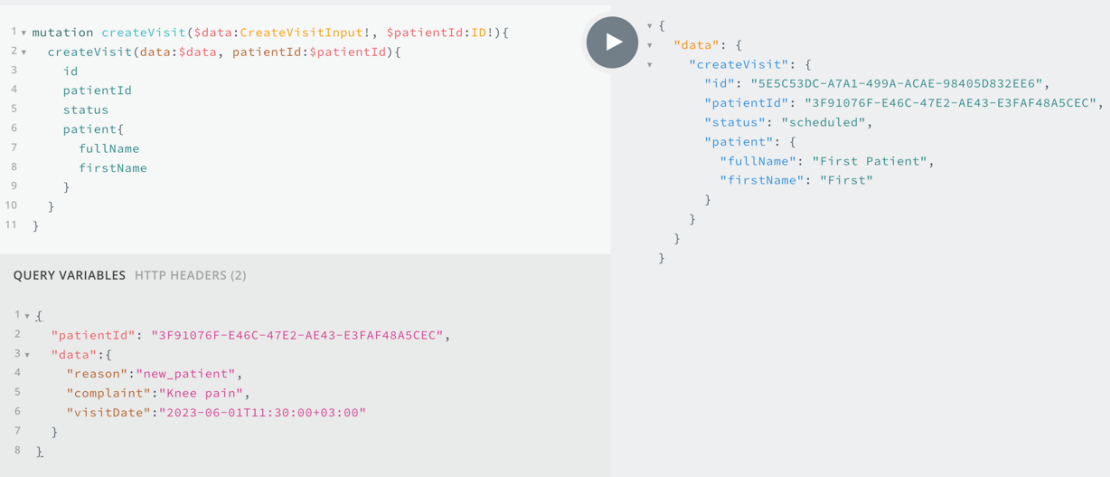

Рисунок 3.3 – Виконання запиту на створення візиту в системі

Після того як виконався запит на створення візиту, сервер повернув його дані, щоб впевнитись в тому, що візит добавлено в список всіх візитів потрібно виконати запит на їх отримання (рис. 3.4).

Рисунок 3.4 – Результат виконання запиту для отримання всіх візитів

В списку повернутих візитів на першому місці знаходиться новостворений візит, а отже запит який виконувався для того, щоб його створити був успішним.

3.2 Тестування графічного інтерфейсу веб-додатку

Коли користувач вперше заходить в систему йому необхідно зареєструватись, для того, щоб це зробити потрібно заповнити форму та натиснути на кнопку створити. Сторінку з заповненою формою реєстрації зображено на рисунку 3.5.

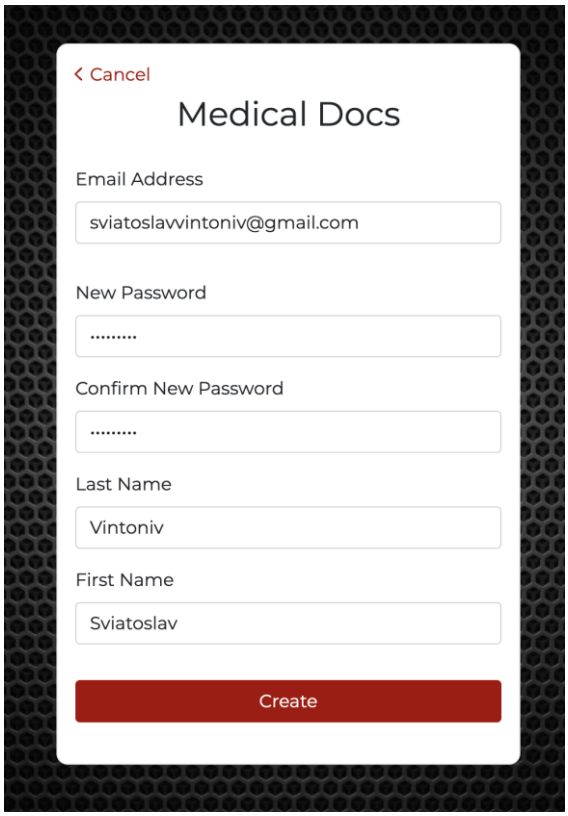

Рисунок 3.5 – Форма для реєстрації в системі

Після того, як користувач зареєструвався системі він може авторизуватись, для цього необхідно заповнити форму логіну (рис. 3.6).

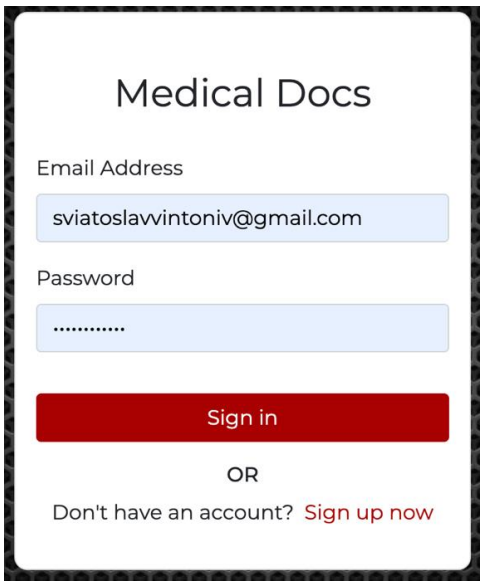

Рисунок 3.6 – Форма для авторизації

Після того, як користувач авторизується в системі в нього буде роль гостя, і він побачить повідомлення про те, що в нього немає доступу до додатку, для того, щоб отримати роль яка має доступ до системи необхідно зв’язатись за адміністрацією (рис. 3.7).

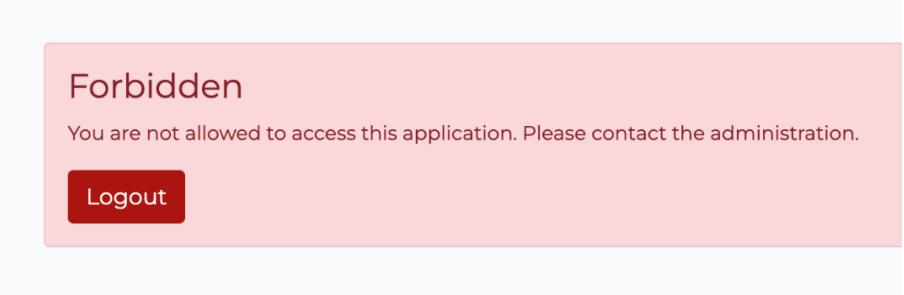

Рисунок 3.7 – Повідомлення про те, що в користувача немає доступу до додатку

Для того, щоб зареєстрований користувач отримав свою роль, в системі повинен авторизувати адміністратор і він має натиснути на кнопку присвоєння ролі для вибраного користувача (рис. 3.8).

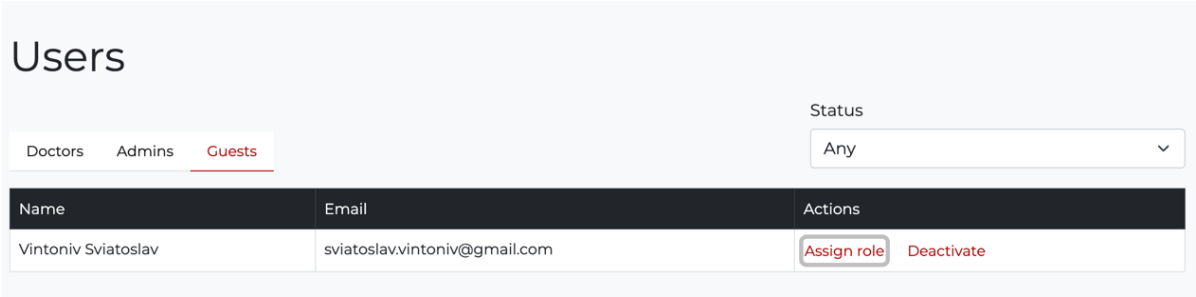

Рисунок 3.8 – Інтерфейс меню управління користувачами

Після того, як адміністратор скористається функцією присвоєння ролі, відкриється вікно в якому можна буде вибрати одну з двох ролей яку має отримати користувач а саме роль адміністратора або лікаря (рис. 3.9).

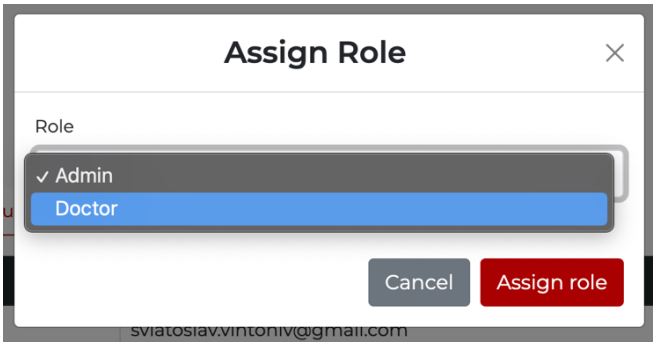

Рисунок 3.9 – Вікно для присвоєння ролі

Після того як користувачу було присвоєно роль лікаря, він має з’явитись в таблиці користувачів в яких роль лікаря (рис. 3.10).

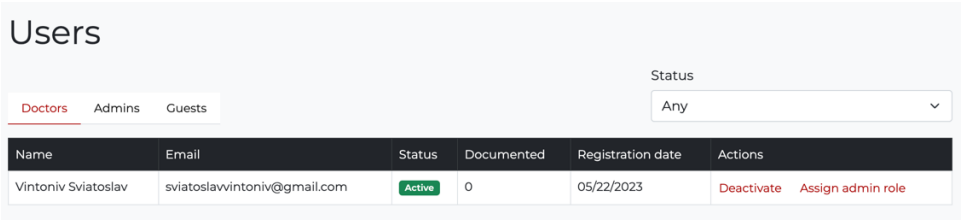

Рисунок 3.10 – Таблиця користувачів з роллю лікаря

Коли користувач отримав роль лікаря він може авторизуватись в системі і отримати доступ до функцій які доступні його ролі. Головною сторінкою яка

відкривається підчас авторизації в системі для лікаря є сторінка зі списком усіх доступних документів (рис. 3.11).

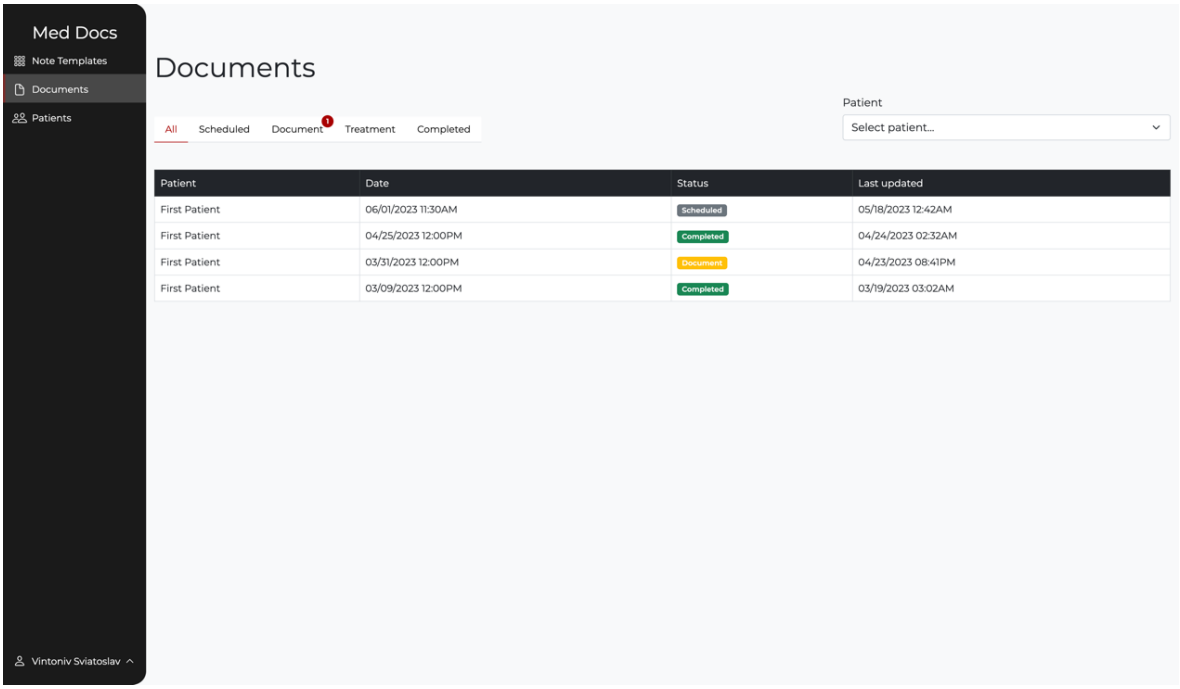

Рисунок 3.11 – Сторінка зі списком усіх документів доступних лікарю

Сторінка зі списком документів розділена на вкладки, кожна з яких відповідає стану в якому зараз перебуває документ, і користувач має змогу легко перемикатись між ними (рис. 3.12).

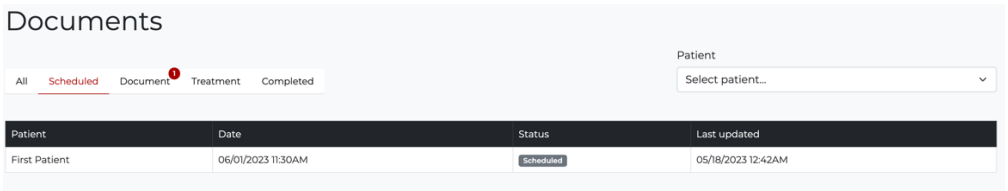

Рисунок 3.12 – Сторінка зі списком документів з увімкнутим фільтром для запланованих візитів

Також на сторінці розміщено фільтр за допомогою якого можна вибирати документи які відповідають конкретному пацієнту.

Для лікаря також доступна сторінка зі списком усіх його пацієнтів, на якій він може додавати нових пацієнтів, редагувати дані про уже існуючі, або видаляти їх, а також є функція пошуку (рис. 3.13).

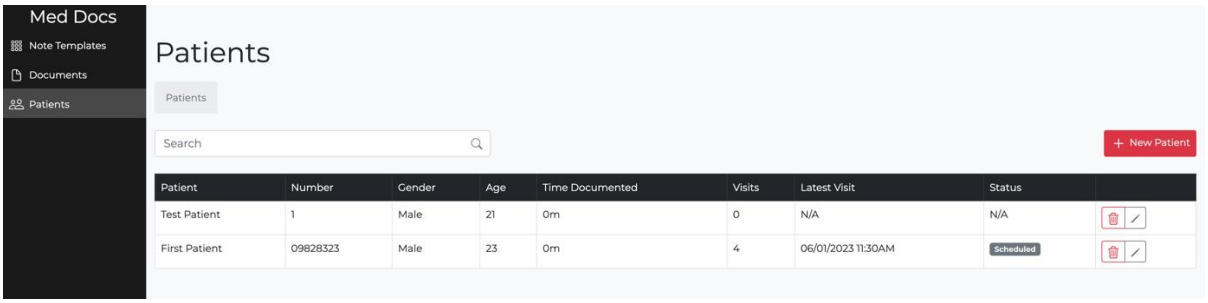

Рисунок 3.13 – Сторінка зі списком пацієнтів

Після натискання на кнопку додати пацієнта повинне відкритись вікно з формою для введення його даних (рис. 3.14).

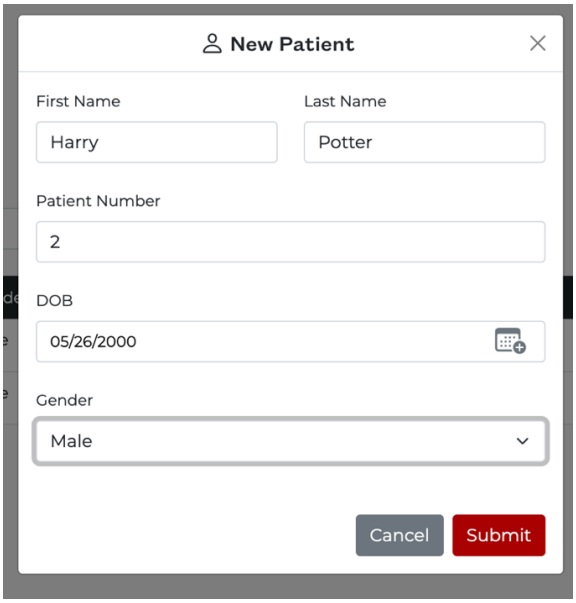

Рисунок 3.14 – Форма для додавання пацієнта

Після заповнення та збереження форми, нові дані про пацієнта з’явились в таблиці. З таблиці можна перейти на сторінку пацієнта, для цього потрібно натиснути на нього. На сторінці пацієнта знаходяться його дані які можна редагувати, також на ній розміщена таблиця зі списком всі візитів пацієнта та кнопка для планування нових (рис. 3.15).

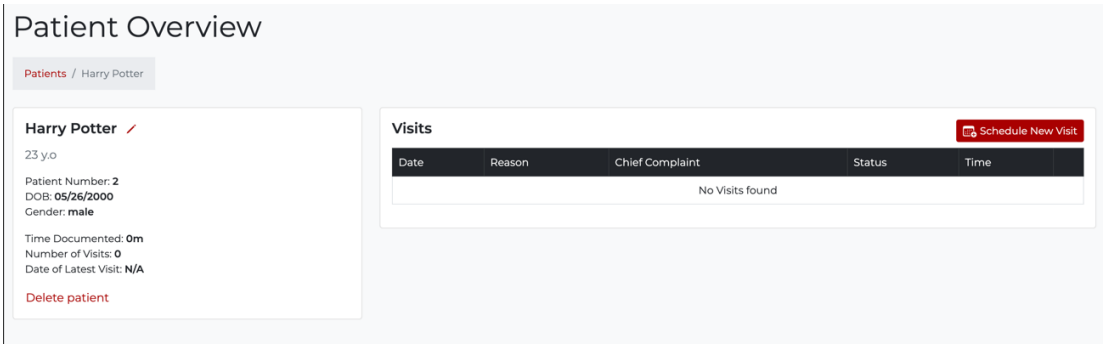

Рисунок 3.15 – Сторінка пацієнта

Для того, щоб створити візит необхідно заповнити форму в якій потрібно вказати дату та причину візиту, а також скаргу пацієнта (рис. 3.16).

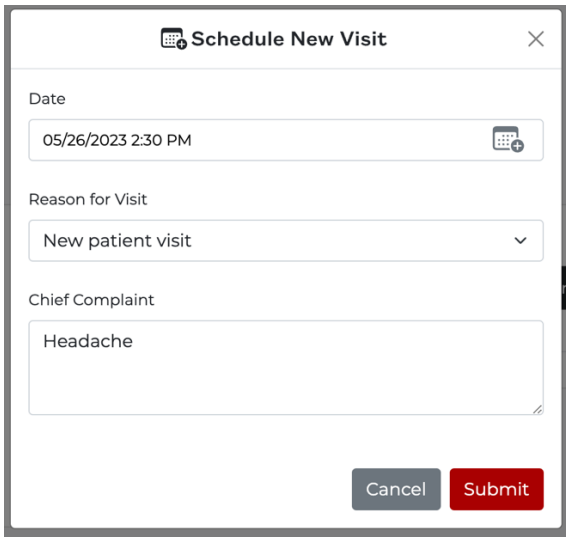

Рисунок 3.16 – Форма створення візиту

Після збереження форми дані про візит було додано в таблицю візитів. Натиснувши на вибраний візит в таблиці відкривається сторінка візиту, на якій знаходяться дані про візит та пацієнта (рис. 3.17).

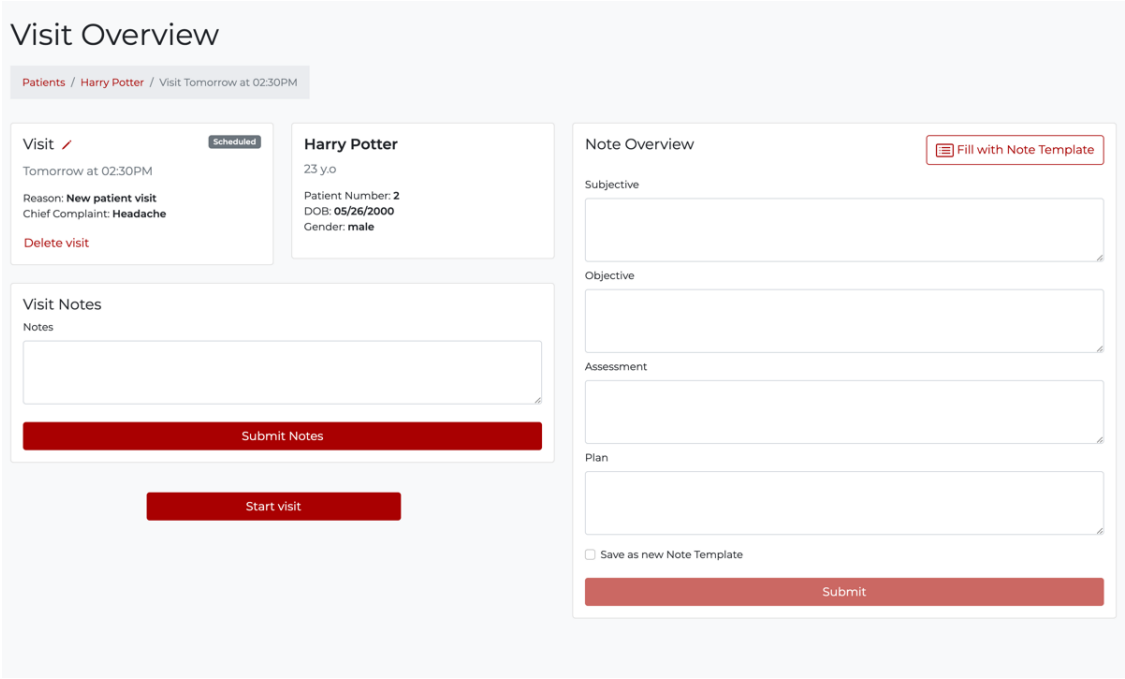

Рисунок 3.17 – Сторінка з даними про візит

Лікар може натиснути на кнопку розпочати візит, після чого візит перейде зі статусу запланованого в статус документування. Після переходу користувач зможе ввести дані необхідні для лікування пацієнта, а саме суб'єктивний аналіз та оцінки стану здоров'я пацієнта, а також мету та план подальшого лікування. Також лікарю доступне поле для ведення нотаток, яке він може постійно редагувати для збереження необхідної та актуальної інформації про процес лікування.

Лікар може заповнити форму для документації візиту вручну або використати один із свої заготованих шаблонів, для цього йому необхідно натиснути кнопку для заповнення форми зі шаблонів. Шаблони розділені на дві групи: усі шаблони і шаблони для даного пацієнта. Після натискання у користувача з'явиться модальне вікно в якому можна вибрати шаблон для заповнення форми (рис. 3.18).

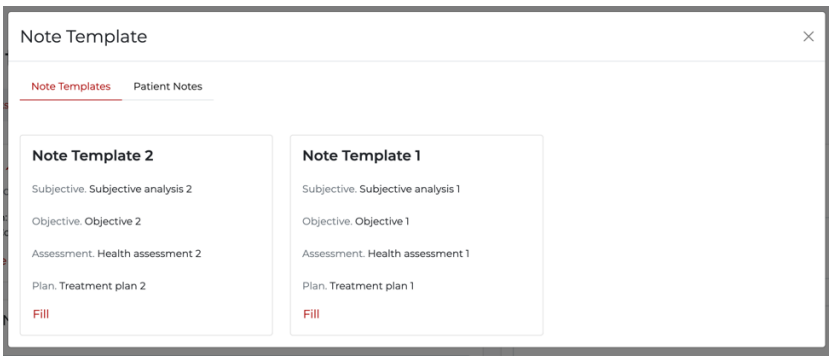

Рисунок 3.18 – Модальне вікно вибору шаблону

Після того як користувач натисне на кнопку для заповнення, форма заповниться даними з шаблону (рис. 3.19).

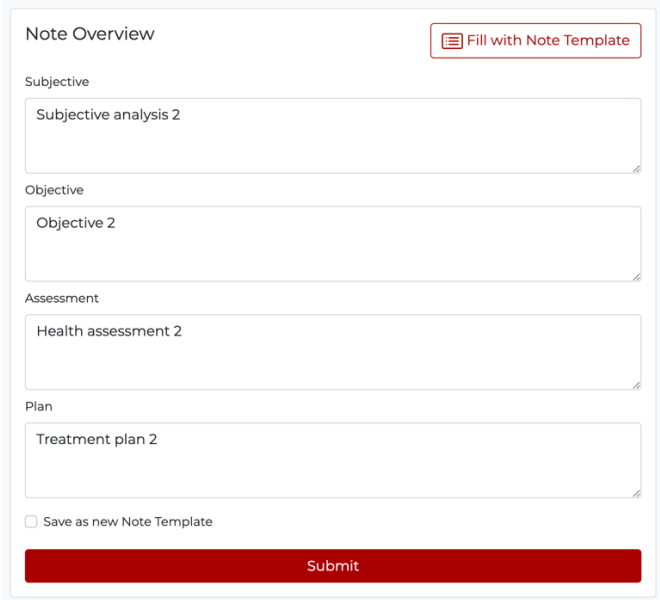

Рисунок 3.19 – Форма заповнена даними з шаблону

Коли лікар заповнив усі необхідні дані, він може перенести візит в стан лікування, після того як це буде зроблено, користувач більше не зможе редагувати форму, змінювати він зможе тільки особисті нотатки. Після закінчення лікування документ можна перенести в стан виконаних (рис. 3. 20).

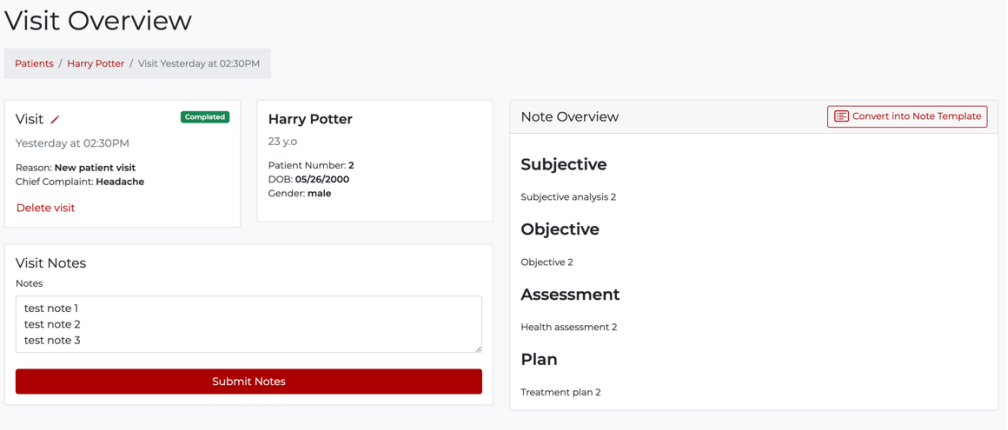

Рисунок 3.20 – Документ в завершеному стані

Якщо лікарю необхідно зберегти дані з документу в шаблон для того, щоб в подальшому їх можна було використати при створенні нових документів, він може скористатися функцією для створення шаблону. Коли лікар натисне на кнопку для створення шаблону, відкриється модальне вікно, де йому необхідно буде ввести назву шаблону, всі інші дані будуть взяті з вибраного документу і їх можна буде редагувати в даній формі (рис. 3.21).

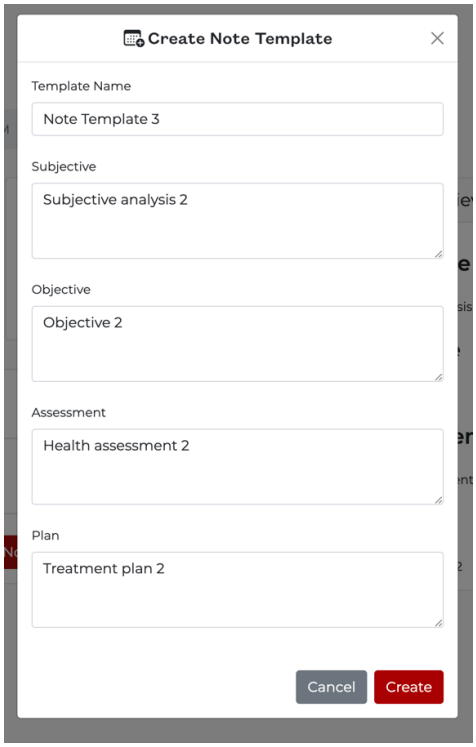

Рисунок 3.21 – Модальне вікно створення шаблону на основі документу

Після створення шаблонну він буде доданий до списку всіх шаблонів, і його можна буде використовувати для створення майбутніх документів. Доступ до всіх своїх шаблонів лікар може отримати на сторінці шаблонів, також на цій сторінці будуть доступні функції редагування, та видалення уже існуючих, та додавання нових шаблонів (рис. 3.22).

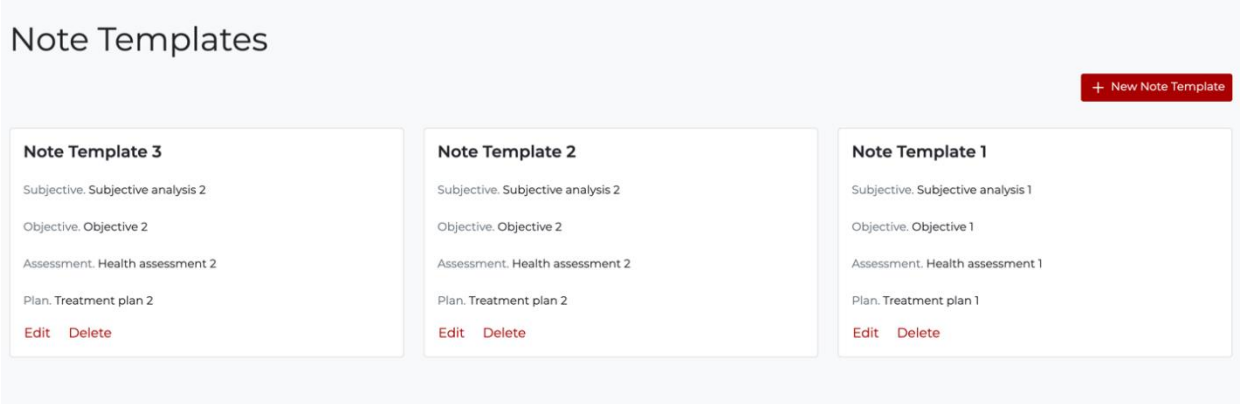

Рисунок 3.22 – Сторінка шаблонів

Існуючі шаблони можна редагувати натиснувши на кнопку для редагування, після чого відкривається модальне вікно для редагування в якому можна змінювати дані, підтвердивши зміни шаблон оновиться і буде відображати нову інформацію. Після натискання на кнопку видалення вибраний шаблон буде вилучено зі списку.

Після тестування графічного інтерфейсу можна стверджувати, що веб-додаток успішно виконує очікувані функції і відповідає базовим вимоги, які були встановлені в процесі розробки. Графічний інтерфейс продемонстрував свою здатність відповідати поставленим задачам, та бути зручним і інтуїтивно зрозумілим для користувача.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ураженні електричним струмом

Долікарська допомога – це комплекс медичних заходів, спрямованих на надання медичної допомоги при невідкладних станах, які стаються на виробництві, у побуті, під час дорожньо-транспортних пригод, катастроф, техногенних аварій та при гострих неврологічних, терапевтичних, хірургічних та термінальних станах. Ненадання долікарської допомоги при нещасних випадках, раптових гострих захворюваннях людини призводить до тяжких наслідків, аж до летальних. Своєчасна допомога відіграє важливу роль у подальшому лікуванні потерпілих і хворих, сприяє скороченню термінів їх медичної та трудової реабілітації [17].

Під час розробки інформаційної системи програмний інженер постійно перебуває за персональним комп’ютером. Також при впровадженні такої системи на серверному обладнанні необхідно працювати з приладами які перебувають під напругою, це збільшує ймовірність ураження електричним струмом. Важливо в такі моменти знати алгоритм та послідовність кроків які необхідно виконати для надання першої медичної допомоги.

Ураження електричним струмом можливе при безпосередньому контакті з провідником електричного струму, електричною дугою, напругою від лежачого на землі дроту, частіше високої напруги. Загалом електротравма становить 1-2 % усіх видів травм. Близько 80 % ураження електричним струмом виникає при контакті зі струмом низької напруги (до 1000 В), до 20 % - струмом високої напруги. У першому випадку летальність становить 3 %, при контакті зі струмом високої напруги – до 30 %. Побутовий струм у 50 Гц особливо небезпечний для нормального функціонування міокарду [18].

Ступінь тяжкості функціональних розладів та ушкоджень залежить також від фізичних характеристик струму, умов контакту та стану постраждалого до моменту отримання електротравми. Також варто враховувати шлях проходження струму в організмі, електропровідність шкіри, загальний стан організму, навколишнє

середовище (сухе, вологе). Так, суха, товста шкіра має опір в 10-200 кОм, тоді, як волога, тонка шкіра створює опір у 100-200 разів менше [18].

Розрізняють легку, середню й тяжку форми ураження електричним струмом. У разі легких уражень постраждалий непритомніє. Електротравми середнього ступеня викликають загальні судоми м'язів, непритомність, розлади дихання й діяльності серця. У разі тяжких уражень дихання та серцева діяльність постраждалого настільки пригнічені, що звичайними методами їх не вдасться відновити (він перебуває в стані клінічної смерті).

Ознаки ураження електричним струмом. Втрату свідомості спостерігають у 70 % випадків, вона може тривати від кількох хвилин до години. Водночас в окремих випадках свідомість може не відновлюватися понад добу. Ознаками ураження головного мозку є клонічні і тонічні судоми, головний біль, сонливість, ретроградна амнезія. Рідше виникають вогнищеві ушкодження головного і спинного мозку. Всі ці явища часто спостерігають у випадках, коли місцем входу струму є голова або верхні кінцівки. Якщо на шляху проходження струму знаходиться головний мозок, серце, то наслідки ураження можуть бути фатальними – розвивається стан клінічної смерті, де у 80 % випадків спостерігають фібриляцію шлуночків, а у 20 % - асистолію [19].

Проходячи крізь живий організм, електричний струм чинить дію:

- термічну - з опіками відповідних ділянок, нагріванням кровоносних судин, нервів;
- електричну - з розкладанням крові та інших органічних речовин;
- біологічну - з подразненням та збудливістю живих тканин організму, що супроводжується судомним скороченням м'язів, у тому числі м'язів серця і легень. Як наслідок виникають різні порушення в організмі і навіть повна зупинка роботи серця та легень.

Електричний струм викликає місцеві й загальні порушення в організмі. Місцеві зміни проявляються опіками тканини в місцях виходу й входу електричного струму. Залежно від стану ураженого (волога шкіра, стомлення, виснаження тощо), сили й напруги струму можливі різні місцеві прояви - від утрати

чутливості до глибоких опіків.

За впливу змінного струму силою 15 мА у постраждалого виникають судоми (так званий струм, що не відпускає). У випадку враження струмом силою 25-50 мА настає зупинка дихання. Через спазм голосових зв'язок постраждалий не може крикнути й покликати на допомогу. Якщо дія струму не припиняється, через кілька хвилин відбувається зупинка серця в результаті гіпоксії й настає смерть постраждалого [20].

Порядок надання першої медичної допомоги при ураженні електричним струмом:

- Першим кроком є забезпечення негайної безпеки рятувальника та потерпілого, від’єднавши джерело живлення або використовуючи ізольовані інструменти, щоб усунути людину від електричного контакту.
- Після того, як потерпілий перебуває в безпечному середовищі, слід провести первинну оцінку, щоб оцінити його життєві функції, чи він перебуває в свідомості та наявність будь-яких небезпечних для життя травм. Якщо потерпілий не реагує, не дихає або у нього відсутній пульс, слід негайно розпочати серцево-легеневу реанімацію (СЛР).
- Етапи, пов’язані з СЛР, включають забезпечення відкритих дихальних шляхів, для цього потрібно нахилити голови назад і підняття підборіддя, виконувати процес штучного дихання та компресії грудної клітки.
- Під час проведення серцево-легеневої реанімації важливо негайно викликати екстринну медичну допомогу або доручити це зробити комусь поблизу. Прибуття кваліфікованих медичних працівників є життєво важливим для подальшого обстеження та лікування потерпілого.
- Очікуючи прибуття екстреної медичної допомоги, важливо постійно контролювати життєво важливі функції потерпілого та заспокоювати його. Якщо жертва починає реагувати, переконайтеся, що вона зайняла зручне положення та запропонуйте їй емоційну підтримку.

При розробці веб-додатку для контролю за веденням документації в медичних закладах з використанням JavaScript технологій не можна не зважати на

ризик ураження електричним струмом. Розуміння відповідних кроків для надання першої медичної допомоги у разі ураження електричним струмом має важливе значення для забезпечення безпеки та благополуччя осіб, які беруть участь у розгортанні та обслуговуванні системи.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Проєктування та розробка веб-додатку передбачає безперервну роботу з персональним комп'ютером, тому вкрай важливо правильно організувати робоче місце оператора ПК. Ергономічно оптимізована робоча зона відіграє важливу роль у сприянні здоров'ю, комфорту та продуктивності людей, які проводять тривалий час за комп'ютером.

Робочі місця працівників, які обладнані комп'ютерами пристроями, повинні відповідати вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи» [20] та ДСанПІН 3.3.2.007-98 «Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [21].

Основні положення ДСТУ 8604:2015:

- Конструкція робочого місця та взаємне розташування всіх його елементів (сидіння, органів керування, засобів відображення інформації тощо) повинні відповідати антропометричним, фізіологічним і психологічним вимогам, а також характеру виконуваної роботи.
- Конструкцією робочого місця повинно бути забезпечено виконання трудових операцій у межах зони досяжності моторного поля.
- Висоти сидіння та підставки для ніг (у разі нерегульованої висоти робочої поверхні). У цьому разі висоту робочої поверхні встановлюють за номограмою для працюючих зростом 1800 мм. Оптимальна робоча поза для менших за зростом працюючих досягається збільшенням висоти робочого

сидіння й підставки для ніг на величину, що дорівнює різниці між висотою робочої поверхні для працюючого зростом 1800 мм і висотою робочої поверхні, оптимальної для зросту даного працюючого.

- Мінімальна товщина стільниці залежатиме від характеристик міцності використовуваного матеріалу та інших технічних вимог і має бути не більше ніж 30 мм.
- Підставка для ніг повинна бути регульованою по висоті. Її ширина повинна бути не менше ніж 300 мм, довжина - не менше ніж 400 мм. Поверхня підставки повинна бути рифленою. По передньому краю доцільно передбачати бортик висотою 10 мм.

Основні положення ДСанПІН 3.3.2.007-98:

- Площа на одне робоче місце має становити не менше ніж 6,0 м2, а об'єм не менше ніж 20,0 м3.
- Приміщення для роботи повинні мати природне та штучне освітлення.
- Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і зебезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%.
- Для внутрішнього оздоблення приміщень слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7-0,8, для стін 0,5-0,6.
- Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 до 90 градусів з вертикаллю в повздовжній та поперечній площинах має становити не більше ніж 200 кд/м2, захисний кут світильників - не менше ніж 40 градусів
- Показник засліплення у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20.
- Необхідно обмежувати нерівномірність розподілу яскравості в полі зору працюючих з екранами. При цьому співвідношення яскравості робочих поверхонь має бути не більшим ніж 3:1, а співвідношення яскравості робочих поверхонь та поверхонь стін, обладнання тощо - 5:1.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було проведено наліз предметної області для кращої побудови вимог до інформаційної системи. Було проведено аналіз існуючих рішень, та визначено основні переваги та недоліки існуючих систем. На основі проведених досліджень було визначено основні вимоги до веб-додатку та акторів системи, а також сформовано список варіантів використання. Також було вибрано методологію проєктування, нею стала RAD, оскільки вона використовується при швидкій розробці додатків.

Провівши аналіз предметної області та існуючих рішень було спроєктовано архітектуру веб-додатку для контролю за веденням документації в медичних закладах. На основі архітектури системи та попередніх дослідженнях було визначено стек основних технологій для розробки даної платформи, основні з них це: Node.js, React, GraphQL та Microsoft SQL Server.

На основі проведених досліджень та строєктованій архітектурі було розроблено веб-застосунок. Розроблена система дозволяє оптимізувати процес ведення документації надаючи лікарям можливість планувати візити для пацієнтів та швидко їх документувати використовуючи зручний та інтуїтивно зрозумілий графічний інтерфейс. Для полегшення створення документів користувачі можуть використовувати шаблони, це дозволяє швидше обробляти скарги пацієнтів та приймати рішення щодо їх лікування на основі уже існуючих задокументованих записах. Веб-додаток також підтримує систему розподілення документів на стани, що полегшує пошук потрібних записів. В системі також передбачена роль адміністраторів для контролю за безпекою та конфіденційністю медичних записів, та виявленням будь-яких порушників і бокування їх.

В подальшому за рахунок модульності системи можливе її легке вдосконалення та додавання нового функціоналу, як наприклад більш детальна статистика медичних закладів та їх працівників а також система для контролю часу роботи медичного персоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Перспективи автоматизації процесу ведення медичної документації [Електронний ресурс] – Режим доступу до ресурсу: https://research.aimultiple.com/medical-record-automation/
2. Сайт застосунку Pabau CRM [Електронний ресурс] – Режим доступу до ресурсу: https://www.pabau.com/home-2/
3. Сайт застосунку Axxess Home Health [Електронний ресурс] – Режим доступу до ресурсу: https://www.axxess.com/
4. Сайт застосунку RXNT EHR [Електронний ресурс] – Режим доступу до ресурсу: https://www.rxnt.com/
5. Діаграма варіантів використання – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
6. RUP методологія [Електронний ресурс] – Режим доступу до ресурсу: https://www.toolshero.com/information-technology/rational-unified-process-rup/
7. RAD методологія [Електронний ресурс] – Режим доступу до ресурсу: https://kissflow.com/application-development/rad/rapid-application-development-methodology-essentials/
8. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/what-is-client-server-architecture-article
9. Мова програмування JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps
10. Документація по Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/docs
11. Документація по бібліотеці React [Електронний ресурс] – Режим доступу до ресурсу: https://uk.legacy.reactjs.org/docs/getting-started.html
12. Документація по Apollo Client/Server [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/docs/

13. Фреймворк Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.3/getting-started/introduction/
14. СУБД Microsoft SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/sql/sql-server/?view=sql-server-ver16
15. Реляційні бази даних [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/relational-databases
16. SPA підхід до розробки веб-додатків [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/hiteshtech/a-beginners-guide-to-create-spa-with-react-js-491c
17. Перша долікарська допомога [Електронний ресурс] – Режим доступу до ресурсу: https://www.pharmencyclopedia.com.ua/article/790/persha-dolikarska-dopomoga
18. Діагностика та лікування уражень електричним струмом [Електронний ресурс] – Режим доступу до ресурсу: https://pidru4niki.com/76919/meditsina/diagnostika_likuvannya_urazhennya_elektrichnim_strumom
19. Жидецький В. Ц., Джигирей В. С., Мельников О. В. Основи охорони праці. 2-ге вид. Львів: Афіша, 2000. 348 с.
20. ДСТУ 8604:2015 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text
21. ДСанПІН 3.3.2.007-98 [Електронний ресурс]. – 1998. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

ДОДАТКИ

Додаток А – Лістинг коду додатку для клієнтської частини

Лістинг коду 1 – Програмний код сторінки зі списком документів

```
import { useState } from 'react';
import { Col, Row } from 'react-bootstrap';
import { format } from 'date-fns';
import { useParams, useHistory, generatePath } from 'react-router-dom';
import { find, orderBy } from 'lodash';
import usePaginatedQuery from '../../hooks/common/usePaginatedQuery';
import visitsTableQuery from '../../graphql/queries/visitsTable';
import useDebouncedValue from '../../hooks/common/useDebouncedValue';
import usePatientSelectOption from
'../../hooks/common/usePatientSelectOption';
import useProviderSelectOption from
'../../hooks/common/useProviderSelectOption';

import Page from '../../components/common/Page';
import Tabs from '../../components/common/Tabs';
import Select from '../../components/common/Select';
import PageLoading from '../../components/common/PageLoading';
import ErrorAlert from '../../components/common/ErrorAlert';
import Table from '../../components/common/Table';
import TablePagination from '../../components/common/Pagination';
import VisitStatus from '../../components/visits/VisitStatus';
import visitStatuses from '../../constants/visitStatuses';
import useActionItemCounters from
'../../hooks/documents/useActionItemCounters';
import { DATE_TIME_FORMAT } from '../../constants/date';

import paths from '../../router/paths';
import { customDateFormat } from '../../utils/date';

const initialTabs = [
  {
    key: 'all',
    label: 'All',
    statuses: undefined,
  },
  {
    key: 'recording',
    label: 'Scheduled',
    statuses: [visitStatuses.RECORDING, visitStatuses.SCHEDULED],
  },
  {
    key: 'documenting',
    label: 'Document',
    statuses: [visitStatuses.DOCUMENTING],
    counter: 0,
  },
  {
    key: 'treatment',
    label: 'Treatment',
    statuses: [visitStatuses.TREATMENT],
  },
  {
```

```
    key: 'completed',
    label: 'Completed',
    statuses: [visitStatuses.COMPLETED],
  },
];

const getOptions = ({ data, hasProviderId }) => {
  if (!data) return [];

  const opt = orderBy(
    data.map((el) => ({ value: el.id, label: hasProviderId ? el.fullName :
el.displayName })),
    ['label'],
    // eslint-disable-next-line comma-dangle
    ['asc']
  );

  const uniqueOptions = opt.reduce((unique, o) => {
    if (!unique.some((obj) => obj.label === o.label && obj.value ===
o.value)) {
      unique.push(o);
    }
    return unique;
  }, []);

  return uniqueOptions;
};

const columns = [
  { id: 'patient', title: 'Patient', cell: (row) => row.patient.fullName },
  { id: 'date', title: 'Date', cell: (row) => format(new
Date(row.visitDate), DATE_TIME_FORMAT) },
  { id: 'status', title: 'Status', cell: ({ status }) => <VisitStatus
status={status} /> },
  { id: 'lastUpdated', title: 'Last updated', cell: (row) =>
customDateFormat(new Date(row.updatedAt)) },
];

const keyExtractor = (row) => row.id;

const Documents = () => {
  const [filterByProviderId, setFilterByProviderId] = useState('');
  const [filterByPatientId, setFilterByPatientId] = useState('');
  const [activeTab, setActiveTab] = useState('all');

  const { providerId } = useParams();
  const history = useHistory();
  const [tabs, setTabs] = useState(initialTabs);

  const filteredColumns = () => {
    if (providerId) {
      return columns.filter((el) => el.id !== 'provider');
    }

    return columns;
  };

  const [searchQuery] = useDebouncedValue();
```

```
  const { patientOptions } = usePatientSelectOption({ searchQuery,
providerId });

  const { providerOptions } = useProviderSelectOption({ searchQuery });

  const { data, loading, error, totalPages, page, goToNPage } =
usePaginatedQuery(visitsTableQuery, {
    itemsPerPage: 10,
    fetchPolicy: 'cache-and-network',
    variables: {
      searchQuery,
      providerId: providerId || filterByProviderId,
      patientId: filterByPatientId !== '' ? filterByPatientId : null,
      statuses: find(tabs, { key: activeTab })?.statuses || undefined,
    },
  });

  useActionItemCounters({
    fetchPolicy: 'cache-and-network',
    variables: {
      providerId: providerId || filterByProviderId,
      patientId: filterByPatientId !== '' ? filterByPatientId : null,
    },
    onCompleted: ({ actionItemCounters }) => {
      const newTabs = tabs.map((tab) => {
        if (tab.key === visitStatuses.DOCUMENTING) {
          return { ...tab, counter: actionItemCounters.documenting };
        }
        if (tab.key === visitStatuses.NOTE_APPROVED) {
          return { ...tab, counter: actionItemCounters.note_approved };
        }
        if (tab.key === visitStatuses.NOTE_REJECTED) {
          return { ...tab, counter: actionItemCounters.note_rejected };
        }
        return tab;
      });
      setTabs(newTabs);
    },
  });

  return (
    <Page title="Documents">
      <Row className="mb-4">
        <Col>
          <Tabs tabs={tabs} activeTab={activeTab}
onActiveTabChange={setActiveTab} />
        </Col>
        <Col md={4}>
          <h6>{providerId ? 'Patient' : 'Provider'}</h6>
          <Select
            clearable
            emptyOptionLabel={`Select ${providerId ? 'patient' :
'provider'}...`}
            options={getOptions({ data: providerId ? patientOptions :
providerOptions, hasProviderId: !!providerId })}
            value={providerId ? filterByPatientId : filterByProviderId}
```

```
            onChange={providerId ? setFilterByPatientId :
setFilterByProviderId}
          />
        </Col>
      </Row>
      {error ? <ErrorAlert error={error} /> : null}
      {loading && !Array.isArray(data) ? <PageLoading /> : null}
      {Array.isArray(data) ? (
        <>
          <Table
            loading={loading}
            columns={filteredColumns()}
            keyExtractor={keyExtractor}
            data={data}
            onClickRow={(row) => {
              history.push(generatePath(paths.visitOverview, { providerId:
row.providerId, visitId: row.id }));
            }}
            noData={providerId ? 'No patients found' : 'No providers
found'}
          />
          <TablePagination {...{ page, totalPages, goToNPage }} />
        </>
      ) : null}
    </Page>
  );
};

export default Documents;
```

Лістинг коду 2 – Програмний код сторінки пацієнта

```
import { Button, Col, Row } from 'react-bootstrap';
import { generatePath, useHistory, useParams } from 'react-router-dom';
import { startCase } from 'lodash';
import Page from '../../components/common/Page';
import { ReactComponent as AddIcon } from '../../assets/icons/add.svg';
import SearchInput from '../../components/common/SearchInput';
import Table from '../../components/common/Table';
import paths from '../../router/paths';
import { formatDurationSeconds } from '../../utils/helpers';
import Breadcrumb from '../../components/common/Breadcrumb';
import VisitStatus from '../../components/visits/VisitStatus';
import PageLoading from '../../components/common/PageLoading';
import ErrorAlert from '../../components/common/ErrorAlert';
import useDebouncedValue from '../../hooks/common/useDebouncedValue';
import TablePagination from '../../components/common/Pagination';
import usePaginatedQuery from '../../hooks/common/usePaginatedQuery';
import patientsTableQuery from '../../graphql/queries/patientsTable';
import { showNewPatientModal } from
'../../components/patients/NewPatientModal';
import PatientTableActionButtons from
'../../components/patients/PatientTableActionButtons';
import { customDateFormat } from '../../utils/date';

const columns = [
  {
```

```
    id: 'fullName',
    title: 'Patient',
    cell: ({ fullName }) => fullName,
  },
  { id: 'patientNumber', title: 'Number', cell: (row) => row.patientNumber
},
  { id: 'gender', title: 'Gender', cell: (row) => startCase(row.gender) },
  { id: 'age', title: 'Age', cell: (row) => row.age },
  {
    id: 'totalTimeDocumentedSeconds',
    title: 'Time Documented',
    cell: (row) => formatDurationSeconds(row.totalTimeDocumentedSeconds),
  },
  { id: 'totalVisits', title: 'Visits', cell: (row) => row.totalVisits },
  {
    id: 'visitDate',
    title: 'Latest Visit',

    cell: (row) => (row.latestVisit?.visitDate ? customDateFormat(new
Date(row.latestVisit.visitDate)) : 'N/A'),
  },
  {
    id: 'status',
    title: 'Status',
    cell: ({ latestVisit }) => (latestVisit?.status ? <VisitStatus
status={latestVisit?.status} /> : 'N/A'),
  },
  {
    id: 'buttons',
    title: '',
    cell: ({ id, providerId }) => (
      <PatientTableActionButtons
        onClick={(e) => {
          e.stopPropagation();
        }}
        patientId={id}
        providerId={providerId}
      />
    ),
  },
];

const keyExtractor = (row) => row.id;
const PatientList = () => {
  const history = useHistory();
  const { providerId } = useParams();

  const [searchQuery, searchInputValue, setSearchInputValue] =
useDebouncedValue();
  const { data, error, loading, totalPages, page, goToNPage, refetch } =
usePaginatedQuery(patientsTableQuery, {
    itemsPerPage: 10,
    fetchPolicy: 'cache-and-network',
    variables: { searchQuery, providerId },
  });

  const links = [];
```

```
  return (
    <Page title="Patients">
      <Row className="mb-4">
        <Col lg="4" sm="8" md="6">
          <Breadcrumb links={links} activeLinkTitle="Patients" />
        </Col>
      </Row>
      <Row className="mb-4">
        <Col md={4} sm={12}>
          <SearchInput
            value={searchInputValue}
            onChange={(v) => {
              setSearchInputValue(v);
            }}
          />
        </Col>
        <Col className="d-flex justify-content-end mt-4 mt-md-0">
          <Button
            size="sm"
            onClick={() => {
              showNewPatientModal({ providerId, onCompleted: () =>
refetch() });
            }}
            variant="danger"
          >
            <AddIcon /> New Patient
          </Button>
        </Col>
      </Row>
      {loading && !data ? <PageLoading /> : null}
      {error ? <ErrorAlert error={error} /> : null}
      {Array.isArray(data) ? (
        <Table
          loading={loading}
          columns={columns}
          data={data}
          keyExtractor={keyExtractor}
          onClickRow={({ id }) => {
            history.push(generatePath(paths.patientOverview, { providerId,
patientId: id }));
          }}
          noData="No patients found"
        />
      ) : null}
      <TablePagination {...{ totalPages, page, goToNPage }} />
    </Page>
  );
};

export default PatientList;
```

Лістинг коду 3 – Програмний код форми для створення пацієнта

```
import { Formik, Form as FormikForm } from 'formik';
import * as yup from 'yup';
import { Row, Col } from 'react-bootstrap';
import { useRef } from 'react';
import FormikGroup from '../common/formik/FromikGroup';
```

```
import FormikFormControl from '../common/formik/FormikFormControl';
import FormikFormSelect from '../common/formik/FormikFormSelect';
import FormikFormDate from '../common/formik/FormikFormDate';
import prepareInitialValues from '../../utils/form';
import Button from '../common/Button';
import { dobToDate } from '../../utils/date';

const gender = ['male', 'female'];

const schema = yup.object().shape({
  firstName: yup
    .string()
    .label('First Name')
    .matches(/^[a-zA-Z]+$/, 'The first name should include only Latin
letters')
    .required()
    .max(40),
  lastName: yup
    .string()
    .label('Last Name')
    .matches(/^[a-zA-Z]+$/, 'The last name should include only Latin
letters')
    .required()
    .max(40),
  patientNumber: yup.string().label('Patient Number').required(),
  dob: yup.date().label('DOB').max(new Date(), 'date should not be in the
future').required(),
  gender: yup.mixed().oneOf(gender).required(),
});

const defaultValues = {
  firstName: '',
  lastName: '',
  patientNumber: '',
  dob: '',
  gender: '',
};

const PatientForm = ({ onSubmit, onClose, initialValues }) => {
  const initialValuesRef = useRef();
  if (!initialValuesRef.current) {
    initialValuesRef.current = prepareInitialValues(
      initialValues ? { ...initialValues, dob: dobToDate(initialValues.dob)
} : {},
      // eslint-disable-next-line comma-dangle
      defaultValues
    );
  }
  return (
    <Formik validationSchema={schema}
initialValues={initialValuesRef.current} onSubmit={onSubmit}>
      {({ isSubmitting }) => (
        <FormikForm>
          <Row>
            <Col md="6">
              <FormikGroup label="First Name">
                <FormikFormControl type="text" required name="firstName" />
              </FormikGroup>
```

```
            </Col>
            <Col className="mt-4 mt-md-0" md="6">
              <FormikGroup label="Last Name">
                <FormikFormControl type="text" required name="lastName" />
              </FormikGroup>
            </Col>
          </Row>
          <FormikGroup className="mt-4" label="Patient Number">
            <FormikFormControl type="text" required name="patientNumber" />
          </FormikGroup>
          <FormikGroup className="mt-4" label="DOB">
            <FormikFormDate type="date" required name="dob" />
          </FormikGroup>
          <FormikGroup className="mt-4" label="Gender">
            <FormikFormSelect required name="gender">
              <option> </option>
              <option value="male">Male</option>
              <option value="female">Female</option>
            </FormikFormSelect>
          </FormikGroup>
          <div className="mt-5">
            <Row>
              <Col className="d-flex justify-content-end">
                <Button className="me-2" onClick={onClose}
variant="secondary">
                  Cancel
                </Button>
                <Button loading={isSubmitting} type="submit"
variant="primary">
                  Submit
                </Button>
              </Col>
            </Row>
          </div>
        </FormikForm>
      )}
    </Formik>
  );
};

export default PatientForm;
```

Лістинг коду 4 – Програмний код сторінки візиту

```
/* eslint-disable react/jsx-wrap-multilines */
import { Col, Row } from 'react-bootstrap';
import { generatePath, useParams } from 'react-router-dom';
import Page from '../../components/common/Page';
import paths from '../../router/paths';
import usePatientById from '../../hooks/common/usePatientById';
import Breadcrumb from '../../components/common/Breadcrumb';
import PatientOverviewCard from
'../../components/patients/PatientOverviewCard';
import PatientVisitsCard from
'../../components/patients/PatientVisitsCard';
import ErrorAlert from '../../components/common/ErrorAlert';
```

```
const PatientOverview = () => {
  const { providerId, patientId } = useParams();

  const { patient, loading, error: fetchError } = usePatientById({
providerId, patientId });

  const links = [
    {
      path: generatePath(paths.patients, { providerId }),
      title: 'Patients',
    },
  ];

  const notFoundError = !loading && !patient ? { message: 'Patient not
found' } : null;
  const error = fetchError || notFoundError;
  return (
    <>
      <div className="mb-4">
        <Breadcrumb links={links} activeLinkTitle={patient?.fullName} />
      </div>
      <Row className="mb-4">
        <Col lg="4" sm="12">
          {error ? <ErrorAlert error={error} /> : null}
          <PatientOverviewCard {...{ patient, patientId, providerId,
loading }} />
        </Col>
        <Col className="ml-5 mt-4 mt-md-0" lg="8" sm="12">
          <PatientVisitsCard {...{ patientId, providerId }} />
        </Col>
      </Row>
    </>
  );
};

export default function PatientOverviewPage() {
  return (
    <Page title="Patient Overview">
      <PatientOverview />
    </Page>
  );
}
```

Лістинг коду 5 – Програмний код форми для створення візиту

```
/* eslint-disable react/jsx-wrap-multilines */

import { Col, Row } from 'react-bootstrap';
import { generatePath, useParams } from 'react-router-dom';
import Page from '../../components/common/Page';
import paths from '../../router/paths';
import usePatientById from '../../hooks/common/usePatientById';
import Breadcrumb from '../../components/common/Breadcrumb';
import PatientOverviewCard from
'../../components/patients/PatientOverviewCard';
import PatientVisitsCard from
'../../components/patients/PatientVisitsCard';
import ErrorAlert from '../../components/common/ErrorAlert';
```

```
const PatientOverview = () => {
  const { providerId, patientId } = useParams();

  const { patient, loading, error: fetchError } = usePatientById({
providerId, patientId });

  const links = [
    {
      path: generatePath(paths.patients, { providerId }),
      title: 'Patients',
    },
  ];

  const notFoundError = !loading && !patient ? { message: 'Patient not
found' } : null;
  const error = fetchError || notFoundError;
  return (
    <>
      <div className="mb-4">
        <Breadcrumb links={links} activeLinkTitle={patient?.fullName} />
      </div>
      <Row className="mb-4">
        <Col lg="4" sm="12">
          {error ? <ErrorAlert error={error} /> : null}
          <PatientOverviewCard {...{ patient, patientId, providerId,
loading }} />
        </Col>
        <Col className="ml-5 mt-4 mt-md-0" lg="8" sm="12">
          <PatientVisitsCard {...{ patientId, providerId }} />
        </Col>
      </Row>
    </>
  );
};

export default function PatientOverviewPage() {
  return (
    <Page title="Patient Overview">
      <PatientOverview />
    </Page>
  );
}
```

Додаток Б – Диск із кваліфікаційною роботою