

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

**«Вебдодаток для забезпечення роботи персоналу
медичного закладу»**

Виконала студентка групи КН-20-1
СТЕЛЬМАХ Аліна Олегівна

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
МАРЧУК Галина Вікторівна

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«25» січня 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **СТЕЛЬМАХ Аліна Олегівна**

Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**

Тема роботи: **«Вебдодаток для забезпечення роботи персоналу медичного закладу»**,
затверджена наказом закладу вищої освіти від **«25» січня 2024 р., №22/с**

Термін здачі закінченої роботи: **«05» червня 2024 р.**

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): об'єктом дослідження є комплекс процесів та технологій, які використовуються для організації та оптимізації роботи персоналу медичного закладу. Предметом дослідження є розробка вебдодатку, який сприяє ефективній роботі медичного персоналу, а також забезпечує збереження та обробку медичних даних.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2024 – 5 березня 2024	Виконано
2	Проектування структури вебдодатку	6 березня 2024 – 25 березня 2024	Виконано
3	Написання програмного коду	26 березня 2024 – 26 квітня 2024	Виконано
4	Тестування вебдодатку	27 квітня 2024 – 14 травня 2024	Виконано
5	Оформлення пояснювальної записки	15 травня 2024 – 1 червня 2024	Виконано
6	Попередній захист	1 червня 2024	Виконано

Здобувач вищої освіти **Аліна СТЕЛЬМАХ**

Керівник **Віталій ЛЕВКІВСЬКИЙ**

РЕФЕРАТ

Кваліфікаційна робота складається з вебдодатку для забезпечення роботи персоналу медичного закладу та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 82 сторінки, 38 ілюстрацій, 8 таблиць, 2 додатки та 20 використаних джерел.

Метою роботи є розробка вебдодатку для автоматизації роботи персоналу медичного закладу з метою підвищення ефективності та забезпечення оптимальної організації робочих процесів. В роботі було визначено основні завдання на розробку вебдодатку, проаналізовано аналоги розробленого програмного продукту, визначено основний функціонал та вимоги. Далі було обрано технології розробки вебдодатку, спроектовано та реалізовано алгоритми роботи, спроектована база даних та розроблена архітектура вебдодатку для медичного закладу. Після аналізу та проектування було реалізовано та протестовано програмний продукт.

Ключові слова: МЕДИЧНИЙ ЗАКЛАД, ВЕБДОДАТОК, ЛІКАР, ПАЦІЄНТ.

					КН.КР.Б – 122 – 24 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата								
Розроб.		А.О. СТЕЛЬМАХ			Вебдодаток для забезпечення роботи персоналу медичного закладу Пояснювальна записка				Літ.	Арк.	Аркушів	
Керівник		В.Л. ЛЕВКІВСЬКИЙ									4	82
Рецензент		Г.В. МАРЧУК							Житомирська політехніка, група КН-20-1			
Н. контр.												
Зав. каф.		М.С. ГРАФ										

ABSTRACT

The qualification work consists of a web application for ensuring the work of the staff of a medical institution and an explanatory note. The explanatory note to the qualification work contains 82 pages, 38 illustrations, 8 tables, 2 appendices and 20 used sources.

The purpose of the work is the development of a web application for automating the work of the staff of a medical institution in order to increase efficiency and ensure optimal organization of work processes. In this work, the main tasks for the development of the web application were defined, the analogues of the developed software product were analyzed, the main functionality and requirements were defined. Next, web application development technologies were selected, work algorithms were designed and implemented, a database was designed, and the architecture of the web application for a medical institution was developed. After analysis and design, the software product was implemented and tested.

Keywords: MEDICAL INSTITUTION, WEB APPLICATION, DOCTOR, PATIENT.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ	7
1.1. Постановка задачі.....	7
1.2. Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи.....	9
1.3. Інструментальні засоби реалізації програмного продукту	11
Висновки до першого розділу	13
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ЗАБЕЗПЕЧЕННЯ РОБОТИ ПЕРСОНАЛУ МЕДИЧНОГО ЗАКЛАДУ	14
2.1. Визначення варіантів використання вебдодатку	14
2.2. Проектування та реалізація алгоритмів роботи вебдодатку	19
2.3. Проектування бази даних програмного продукту	22
2.4. Архітектура та узагальнена структура вебдодатку	28
2.5. Розробка вебдодатку для забезпечення роботи персоналу медичного закладу.....	32
Висновки до другого розділу	39
РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ ВЕБДОДАТКУ	41
3.1. Структура інтерфейсу та опис роботи вебдодатку	41
3.2. Тестування роботи вебдодатку.....	52
Висновки до третього розділу	58
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CORS – Cross-Origin Resource Sharing (механізм безпеки сучасних браузерів)

CRUD – Create, Read, Update, Delete (основні функції управління даними)

DOM – Document Object Mode (програмний інтерфейс для документів HTML і XML)

MERN – MongoDB, Express.js, React.js, Node.js (стек технологій на основі JavaScript)

UML – Unified Modeling Language (уніфікована мова моделювання)

БД – база даних

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Сучасний медичний сектор стикається зі значними викликами, пов'язаними зі зростанням популяції, змінами в клінічних практиках, а також потребами в удосконаленні адміністративних процесів. Відповідно до останніх досліджень, ефективне управління персоналом є ключовим чинником для покращення якості надання медичних послуг і забезпечення позитивного досвіду для пацієнтів. Вебдодатки стають потужним інструментом для впровадження автоматизації та оптимізації процесів у медичних закладах[1].

Розробка спеціалізованого вебдодатку для управління персоналом медичного закладу відповідає потребам сучасної медицини. Цей інструмент дозволить ефективно планувати розклади роботи медичного персоналу, координувати комунікацію між різними відділами і спеціалістами, а також забезпечити високий рівень координації усіх аспектів медичної практики.

Окрім того, вебдодаток надає можливості для ефективного моніторингу використання ресурсів, включаючи обладнання і матеріальні ресурси, що дозволяє оптимізувати витрати та забезпечити ефективне їх використання. Це особливо важливо в умовах зростаючої фінансової обтяженості медичних закладів і необхідності максимізації результативності [2].

Загальні тренди в цифровізації та впровадженні новітніх технологій у галузі охорони здоров'я підкреслюють важливість розвитку інноваційних рішень, таких як вебдодатки для управління персоналом, які сприяють покращенню якості медичних послуг і ефективності управлінських процесів. Таким чином, розробка і впровадження вебдодатку для забезпечення роботи персоналу медичного закладу стає стратегічним кроком у напрямку сучасного управління медичними ресурсами і покращення медичної допомоги в цілому.

Мета роботи – розробка вебдодатку для автоматизації роботи персоналу медичного закладу з метою підвищення ефективності та забезпечення оптимальної організації робочих процесів.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Для досягнення поставленої мети розробки вебдодатку для забезпечення роботи персоналу медичного закладу необхідно вирішити наступні основні завдання:

- визначити основні вимоги різних категорій користувачів та ідентифікувати функціональні можливості, які має забезпечити додаток для оптимізації робочих процесів;
- визначити технологічний стек для клієнтської та серверної частини;
- створити варіанти використання для кожної категорії користувачів;
- спроектувати та розробити клієнтську частину додатку з урахуванням зручності використання та інтуїтивності інтерфейсу для кожної категорії користувачів.

Об’єктом дослідження кваліфікаційної роботи є комплекс процесів та технологій, які використовуються для організації та оптимізації роботи персоналу медичного закладу.

Предметом дослідження є розробка вебдодатку, який сприяє ефективній роботі медичного персоналу, а також забезпечує збереження та обробку медичних даних.

В рамках розробки кваліфікаційної роботи були опубліковані наукові праці:

Стельмах А. О., Левківський В. Л. Алгоритми взаємодії в інформаційних системах. Тези VI Всеукраїнської науково-технічної конференції "Комп’ютерні технології: інновації, проблеми, рішення", 29-30 листопада 2023 р. Житомир: Житомирська політехніка, 2023. С. 141-142.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ

1.1. Постановка задачі

Вебдодаток для забезпечення роботи медичного закладу має надати можливість ефективного управління медичним персоналом, ресурсами та пацієнтами. Користувачі вебдодатку (адміністратори, лікарі та медичний персонал) повинні мати змогу виконувати свої завдання швидко та безпечно, забезпечуючи високу якість медичних послуг. Основні функції включають управління інформацією, створення медичних записів, планування прийомів та створення звітів.

Аналіз вебпроєкту

1. Визначення функціональних потреб медичного закладу.

1.1. Ідентифікація вимог різних категорій користувачів (адміністратори, лікарі, медичний персонал).

1.2. Визначення основних функцій, які має забезпечити вебдодаток для ефективного виконання медичних процесів.

2. Аналіз існуючих рішень для медичних вебдодатків.

2.1. Огляд наявних вебдодатків та систем, які використовуються в медичній сфері.

2.2. Визначення їх функціональних можливостей та визначення сильних і слабких сторін.

3. Вибір технологій для розробки вебдодатку.

3.1. Вибір технологічного стека для клієнтської та серверної розробки.

3.2. Визначення баз даних, які будуть використовуватися для зберігання медичних даних.

Проєктування вебдодатку для медичного закладу

1. Проєктування варіантів використання вебдодатку для різних ролей користувачів.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		7

- 1.1. Створення діаграм варіантів використання (Use Case diagrams) для кожної категорії користувачів.
- 1.2. Опис сценаріїв взаємодії користувачів з вебдодатком.
2. Проектування алгоритмів роботи вебдодатку.
 - 2.1. Проектування алгоритмів роботи вебдодатку для входу в систему.
 - 2.2. Проектування алгоритмів роботи вебдодатку для управління персоналом та пацієнтами.
 - 2.3. Проектування алгоритмів роботи вебдодатку для планування прийомів та створення медичних записів.
3. Проектування архітектури вебдодатку для медичного закладу.
 - 3.1. Розробка діаграми компонентів для клієнтської частини.
 - 3.2. Розробка діаграми компонентів для серверної частини.
 - 3.3. Розробка загальної архітектури вебдодатку з використанням діаграми розгортання.

Реалізація вебсайту

1. Розробка клієнтської частини.
 - 1.1. Ініціалізація та налаштування проєкту.
 - 1.2. Розробка основної структури та компонентів.
 - 1.3. Налаштування управління станом.
2. Розробка серверної частини.
 - 2.1. Ініціалізація та налаштування серверного оточення.
 - 2.2. Розробка основної серверної логіки.
 - 2.3. Налаштування додаткових сервісів та безпеки.

Тестування

1. Тестування форми для входу в систему.
2. Тестування форм для управління інформацією про персонал.
3. Тестування форм для створення медичних записів.
4. Тестування форм для планування прийомів.

Результат реалізації поставленого завдання – функціональний вебдодаток, що надає можливість ефективного управління медичним закладом.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		8

1.2. Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи

Для ефективної розробки вебдодатку для забезпечення роботи медичного закладу потрібно провести аналіз аналогів програмного забезпечення, що вже існують на ринку. Це дозволить отримати важливі відомості про те, які можливості і функціональність вже реалізовані у схожих програмах, а також виявити їхні переваги та недоліки.

Першим розглянутим програмним продуктом є «Clinica Web»[3]. Clinica Web – це медична система, яка автоматизує роботу медичних центрів та клінік. Для того, щоб авторизуватися у системі, потрібно здійснити вхід з головної сторінки сайту (рис. 1.1).

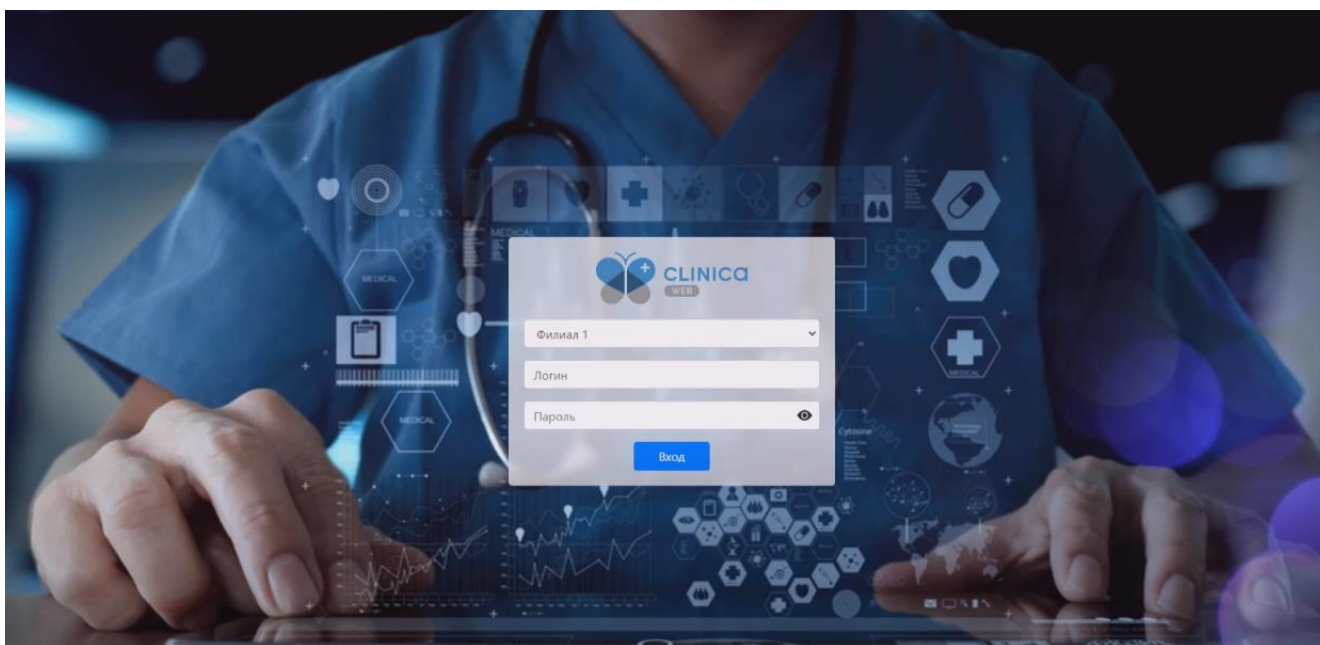


Рис. 1.1. Сторінка входу до системи

Після цього ми опинимося на сторінці медичного працівника, де є можливість додавати та переглядати інформацію про пацієнтів, що показано на рисунку 1.2. Також доступними є секції для управління медичними записами, фінансового обліку, комунікації з пацієнтами та звітності.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		9

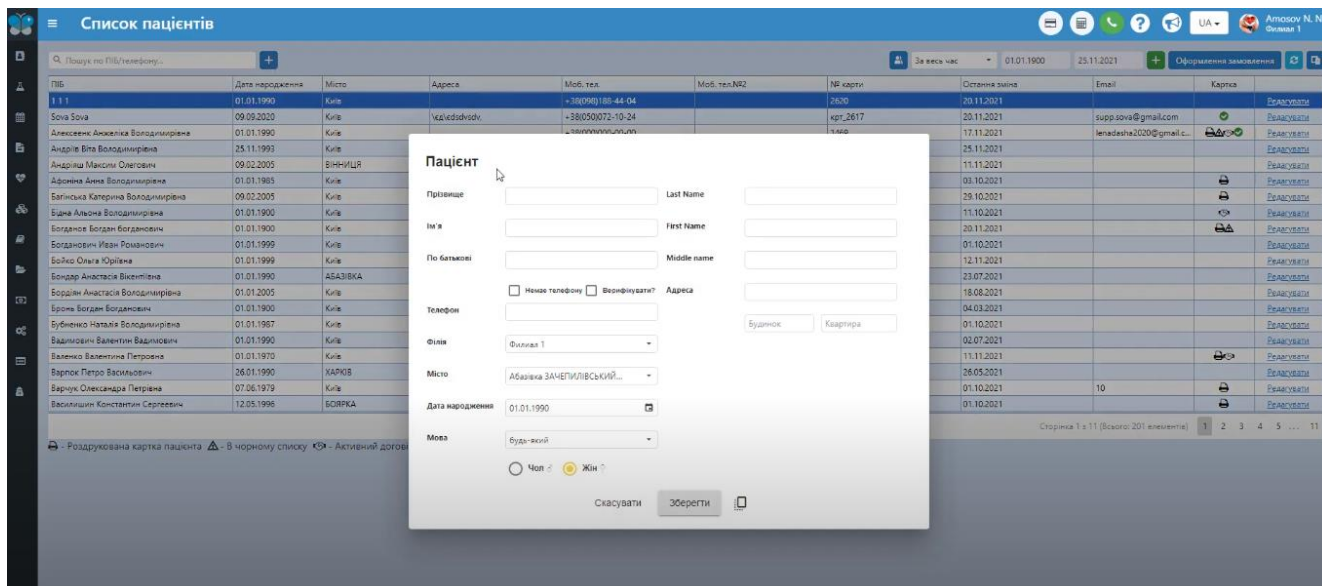


Рис. 1.2. Сторінка перегляду інформації про пацієнтів

Основна перевага даного програмного продукту полягає у автоматизації процесів, що допомагає зменшити ручну працю та уникнути помилок під час ведення медичної документації. Оптимізація управлінських та клінічних процесів підвищує продуктивність медичного закладу, забезпечуючи ефективнішу роботу всього персоналу.

Проте варто враховувати кілька недоліків. Вартість впровадження такої системи може бути значною, особливо для малих медичних закладів, що може стати перешкодою в її впровадженні. Необхідність навчання персоналу для ефективної роботи з новими технологіями також потребує часу та фінансових ресурсів.

Ще одним програмним продуктом, що було розглянуто на етапі порівняння аналогів програмних продуктів, є «SimplexMed»[4]. Це вебзастосунок, спрямований на автоматизацію медичних процесів і управління клінікою. Платформа пропонує розширений набір функціональності, включаючи управління пацієнтами з можливістю запису на прийом, електронну медичну картку та історію лікування. Наприклад, на рисунку 1.3 зображено вигляд сторінки зареєстрованого пацієнта та меню основних функцій.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		
						10

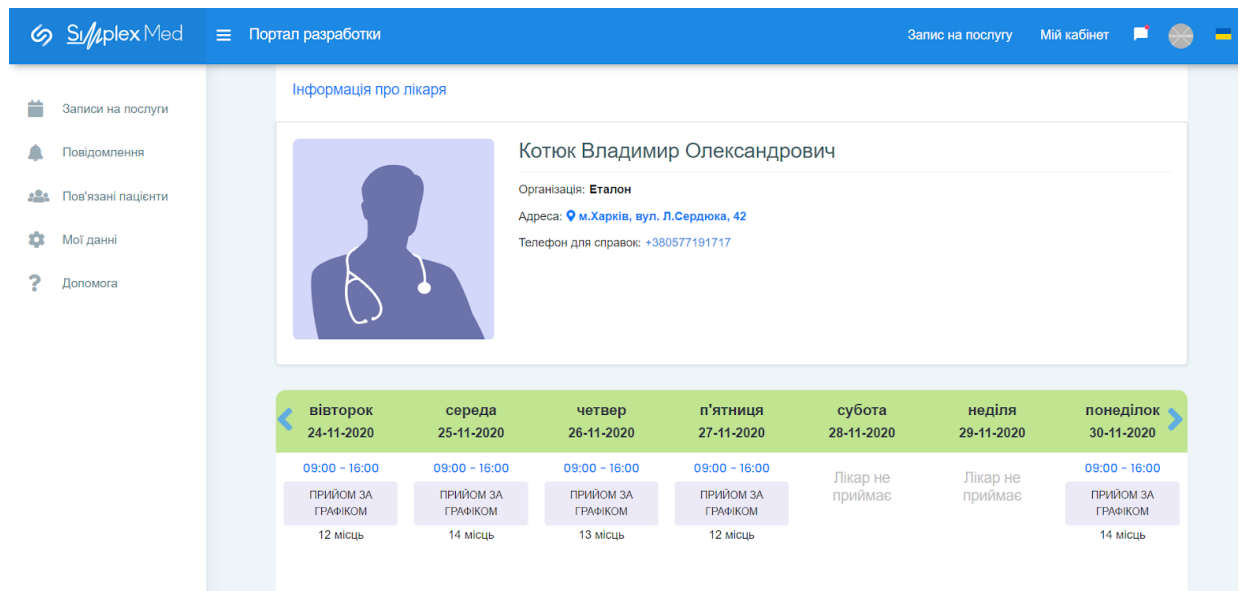


Рис. 1.3. Сторінка зареєстрованого пацієнта

Отже, основні переваги «SimplexMed» полягають у значній ефективності та покращенні якості надання медичних послуг. Це спрощує робочі процеси медичного персоналу, зменшує час на обробку даних і покращує доступність медичної інформації.

Проте, серед недоліків можемо відзначити високі витрати на впровадження та підтримку системи. Також можуть виникати проблеми з інтеграцією та нестача гнучкості в налаштуванні під унікальні потреби конкретних клінік чи лікарів.

1.3. Інструментальні засоби реалізації програмного продукту

Правильний вибір інструментальних засобів реалізації програмного продукту є ключовим фактором для успіху будь-якого проекту. Цей вибір впливає на продуктивність, масштабованість, зручність підтримки та майбутній розвиток програмного забезпечення. Важливо враховувати специфіку проекту, його вимоги, наявні ресурси, а також досвід і знання команди розробників. Обрані інструменти повинні забезпечувати ефективну розробку, тестування, розгортання та підтримку програмного продукту.

Для реалізації вебдодатку, призначеного для забезпечення роботи медичного закладу, було обрано мову програмування JavaScript та стек технологій MERN. JavaScript є однією з найпопулярніших мов програмування у світі, що використовується як на клієнтській, так і на серверній стороні. Вона дає

можливість реалізувати високу продуктивність, гнучкість та інтерактивність вебдодатків. Цей підхід забезпечує всі необхідні компоненти для створення сучасного вебдодатку з високою продуктивністю та масштабованістю [5].

Отже, стек технологій MERN складається з наступних ключових компонентів:

- MongoDB – це документно-орієнтована база даних NoSQL, яка зберігає дані у форматі JSON-подібних документів. Це дозволяє легко масштабувати базу даних горизонтально, забезпечуючи високу доступність та продуктивність. MongoDB надає гнучкість у зберіганні даних, що особливо корисно для медичних додатків, де структура даних може змінюватися з часом. Використання MongoDB також сприяє швидкому прототипуванню та розвитку додатку [6].
- Express.js – веб-фреймворк для Node.js, який спрощує створення серверних додатків і API. Він надає потужний набір функцій для керування HTTP-запитами, маршрутизації, обробки помилок та взаємодії з базами даних. Завдяки своїй мінімалістичній природі та високій продуктивності, Express.js є відмінним вибором для створення надійних та масштабованих серверних додатків. У контексті медичного закладу, де необхідна швидка та надійна обробка запитів, Express.js є ідеальним рішенням [7].
- React.js – це бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона дозволяє створювати інтерактивні, динамічні та високопродуктивні веб-інтерфейси. Використовуючи компоненти та віртуальний DOM, React.js забезпечує ефективне оновлення інтерфейсу користувача, що є критично важливим для додатків з високими вимогами до користувацького досвіду. Для медичного закладу, де інтерфейс має бути інтуїтивно зрозумілим та швидким, React.js є оптимальним вибором [8].

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		12

- Node.js – серверне середовище виконання JavaScript, яке дозволяє створювати масштабовані та високопродуктивні мережеві додатки. Node.js використовує неблокуючу, подієво-орієнтовану модель, що робить його ідеальним для обробки великої кількості одночасних запитів з мінімальними затримками. Для медичного закладу, де важлива швидка і надійна обробка даних та інтеграція з різними системами, Node.js забезпечує необхідну продуктивність та гнучкість [9].

Загалом використання обраних технологій дозволяє досягти високої швидкості розробки, легкої підтримки та надійної роботи додатку.

Висновки до першого розділу

У першому розділі роботи проведено детальний аналіз напрямків використання технологій для реалізації вебдодатку, спрямованого на забезпечення роботи медичного закладу. Встановлено основні вимоги до функціональності вебдодатку, визначено потреби різних категорій користувачів, таких як адміністратори, лікарі та медичний персонал. Було проведено аналіз існуючих рішень на ринку, зокрема «Clinica Web» та «SimplexMed», що дозволило визначити їхні переваги та недоліки.

Порівняльний аналіз цих продуктів допоміг виявити ключові функціональні можливості, які необхідно реалізувати у розроблюваному вебдодатку, а також визначити області, де існує можливість для поліпшення.

Вибір технологій для розробки вебдодатку було здійснено на основі аналізу вимог та особливостей проекту. Обраний технологічний стек MERN забезпечує високу продуктивність, гнучкість, масштабованість та інтерактивність, що є критично важливим для успішної реалізації проекту.

Таким чином, проведений аналіз дозволяє сформулювати чітке уявлення про архітектуру, алгоритми роботи та технології, необхідні для створення ефективного та надійного вебдодатку для персоналу медичного закладу. Ці результати слугуватимуть основою для наступних етапів проектування.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		13

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ЗАБЕЗПЕЧЕННЯ РОБОТИ ПЕРСОНАЛУ МЕДИЧНОГО ЗАКЛАДУ

2.1. Визначення варіантів використання вебдодатку

На етапі визначення варіантів використання вебдодатку потрібно здійснити аналіз різних вимог та сценаріїв, в яких може бути застосований розроблений програмний продукт, зокрема, ідентифікацію можливих користувачів, їх потреб та способів взаємодії з вебдодатком [10].

Бізнес-вимоги:

- проєкт має на меті створення вебдодатку для підтримки ефективної роботи медичного закладу, забезпечуючи зручний доступ до медичної інформації та послуг для персоналу;
- вебдодаток буде реалізовано у вигляді інтерактивного вебсайту, який містить структуровану інформацію та функціональні можливості для керування медичними записами, розкладом візитів, та персоналом;
- вебдодаток буде містити функціонал, характерний для сучасного медичного програмного забезпечення, з можливістю створення та управління профілями користувачів, обміну інформацією та забезпечення високого рівня захисту даних.

Вимоги користувачів

1. Адміністратор:

- здійснити вхід;
- переглянути загальну інформацію на головній сторінці вебсайту;
- додати інформацію про нового лікаря;
- додати інформацію про новий медичний персонал;
- додати інформацію про нового адміністратора;
- додати інформацію про місця та палати;
- додати інформацію про медичний транспорт;
- переглянути інформацію про лікарів;

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		14

- переглянути інформацію про медичний персонал;
- переглянути інформацію про наявні місця;
- здійснити вихід.

2. Лікар:

- здійснити вхід;
- переглянути загальну інформацію на головній сторінці вебсайту;
- переглянути власний профіль;
- редагувати власний профіль;
- переглянути інформацію про наявні місця;
- переглянути інформацію про записи;
- переглянути інформацію про звіти;
- створити звіт;
- здійснити вихід.

3. Медичний персонал:

- здійснити вхід;
- переглянути загальну інформацію на головній сторінці вебсайту;
- переглянути власний профіль;
- редагувати власний профіль;
- додати інформацію про пацієнта;
- створити запис до лікаря;
- переглянути інформацію про наявні місця;
- здійснити вихід.

Функціональні вимоги:

1. Авторизація і аутентифікація – вебдодаток повинен забезпечувати авторизацію користувачів з різними ролями (адміністратор, лікар, медичний персонал) за допомогою унікального ідентифікатора та пароля.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		15

2. Управління користувачами – адміністратор повинен мати можливість додавати, переглядати та редагувати інформацію про користувачів різних ролей (лікарі, медичний персонал, інші адміністратори).
3. Управління профілями – кожен користувач (лікар, медичний персонал) має можливість переглядати і редагувати власний профіль, зокрема особисті дані та контактну інформацію.
4. Управління медичним обладнанням та ресурсами – адміністратор повинен мати можливість додавати та редагувати інформацію про медичне обладнання, медичні засоби, місця та палати в закладі.
5. Безпека і конфіденційність даних – вебдодаток повинна забезпечувати високий рівень захисту особистих медичних даних пацієнтів від несанкціонованого доступу та витоку інформації.

Нефункціональні вимоги:

1. Доступність – вебдодаток повинен бути доступним для користувачів не менше 99.9% часу робочих годин.
2. Швидкодія – вебдодаток повинен забезпечувати швидке завантаження сторінок та оперативну відповідь на запити користувачів, з середнім часом відгуку не більше 5 секунд.
3. Масштабованість – вебдодаток повинна підтримувати масштабування для обробки зростаючої кількості користувачів та обсягу даних без втрати продуктивності.
4. Адаптивність – вебдодаток повинен мати адаптивний дизайн, який забезпечує зручний доступ і використання на різних типах пристроїв (комп'ютери, планшети, смартфони).
5. Сумісність – вебдодаток повинен бути сумісним з різними веб-браузерами (Chrome, Firefox, Edge).

Системні вимоги:

- вебдодаток повинен підтримувати операційну систему Windows (версії 10 і вище);

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		16

- вебдодаток повинен використовувати документно-орієнтовану базу даних MongoDB для зберігання медичних даних та іншої інформації;
- мінімальні системні вимоги для вебдодатка повинні відповідати наступним параметрам: процесор Intel Core i5 або аналогічний, 8 ГБ оперативної пам'яті.

Антивимоги:

- вебдодаток не повинен мати недоліки в системі безпеки, такі як відсутність шифрування даних або слабкість механізмів автентифікації, що може призвести до витоку конфіденційної інформації;
- вебдодаток не повинен використовувати алгоритми обробки даних, які є нечіткими, що може ускладнити перевірку коректності обробки медичної інформації.

Після визначення та аналізу ключових вимог до вебдодатку для забезпечення роботи медичного закладу, можна сформуванати діаграму варіантів використання (Use Case diagram).

Діаграма варіантів використання слугує для моделювання функціональності вебдодатку з точки зору користувачів [11]. Це графічний засіб для візуалізації та опису того, як різні актори (користувачі або зовнішні системи) взаємодіють з вебсайтом, використовуючи різні функції або функціональні можливості.

Діаграма варіантів використання є однією з ключових діаграм в моделюванні вимог у методології UML. Основна мета діаграми варіантів використання полягає в тому, щоб визначити функціональність вебдодатку, уточнити взаємодії між користувачами та системою, полегшити визначення вимог.

На рисунку 2.1 зображено діаграму варіантів використання для вебдодатку для забезпечення роботи персоналу медичного закладу. Ця діаграма візуалізує основні функції та можливості з точки зору акторів, які взаємодіють з вебдодатком.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		17

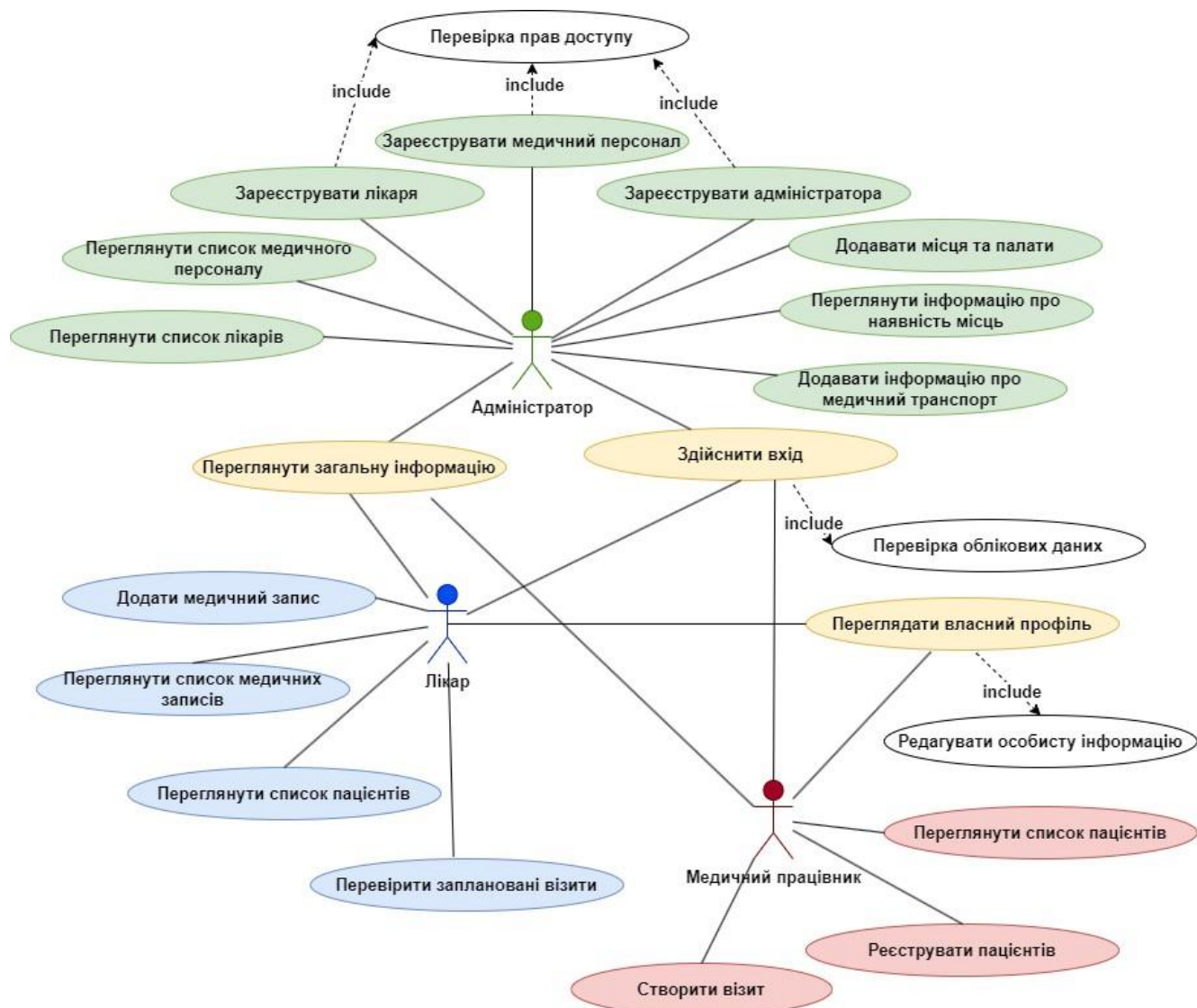


Рис. 2.1. Діаграма варіантів використання вебдодатку для забезпечення роботи персоналу медичного закладу

Кожна роль має свої функціональні можливості, які забезпечують ефективну роботу медичного закладу.

Варіанти дій для ролі Адміністратор:

- адміністратор має можливість додати інформацію про нового лікаря;
- адміністратор має можливість додати інформацію про новий медичний персонал;
- адміністратор має можливість додати інформацію про нового адміністратора;
- адміністратор має можливість додати інформацію про місця та палати;

- адміністратор має можливість додати інформацію про медичний транспорт;
- адміністратор має можливість переглянути інформацію про лікарів;
- адміністратор має можливість переглянути інформацію про медичний персонал;
- адміністратор має можливість переглянути інформацію про наявні місця.

Варіанти дій для ролі Лікар:

- лікар має можливість переглянути власний профіль;
- лікар має можливість редагувати власний профіль;
- лікар має можливість переглянути інформацію про наявні місця;
- лікар має можливість переглянути інформацію про записи;
- лікар має можливість переглянути інформацію про звіти;
- лікар має можливість створити звіт.

Варіанти дій для ролі Медичний персонал:

- медичний персонал має можливість переглянути власний профіль;
- медичний персонал має можливість редагувати власний профіль;
- медичний персонал має можливість додати інформацію про пацієнта;
- медичний персонал має можливість створити запис до лікаря;
- медичний персонал має можливість переглянути інформацію про наявні місця.

Загальні варіанти дій:

- зареєстровані користувачі мають можливість здійснити вхід з використанням унікального ідентифікатора та пароля;
- після входу користувач може бачити загальну інформацію на головній сторінці вебсайту.

2.2. Проектування та реалізація алгоритмів роботи вебдодатку

Одним із ключових етапів у розробці будь-якої програмної продукту є проектування алгоритмів. Для цього будемо використовувати діаграми

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		19

активності. Діаграми активності є потужним інструментом для моделювання динамічних аспектів вебдодатку. Вони дозволяють візуально представити послідовність дій, що виконуються. Даний вид діаграм схожий на блок-схеми, але має додаткові можливості для відображення складніших процесів і паралельних виконань [12].

Основні переваги використання діаграм активності полягають у наочному уявленні про послідовність і логіку дій, що робить процес проєктування зрозумілішим, та спрощенні моделювання паралельних процесів, що важливо для складних систем з багатьма одночасними завданнями.

На рисунку 2.2 зображено діаграму активності, що представляє послідовність дій, які виконує лікар.

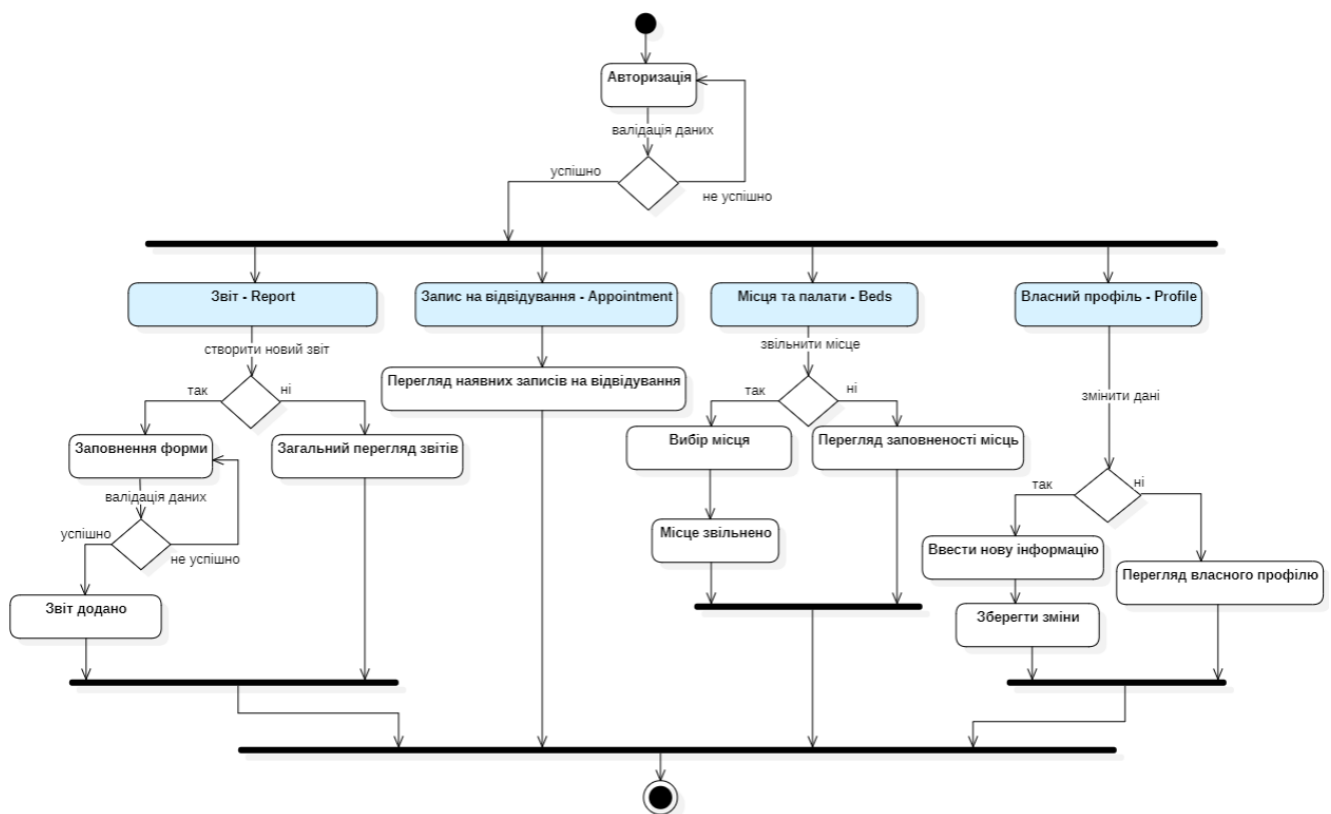


Рис. 2.2. Діаграма активності для ролі Лікар

Після авторизації лікар потрапляє до основного меню, де доступні наступні опції: «Звіт», «Запис на відвідування», «Місця та палати» та «Власний профіль». У розділі «Звіт» лікар може створити новий звіт або переглянути існуючі звіти. Процес створення звіту включає заповнення форми та валідацію даних. У розділі «Запис на відвідування» лікар переглядає наявні записи без додаткових дій.

Розділ «Місця та палати» дозволяє лікарю звільняти місця або переглядати їх заповненість. Власний профіль лікаря надає можливість змінювати особисті дані або просто переглядати їх. У випадку зміни даних лікар вводить нову інформацію та зберігає зміни.

Наступна діаграма активності (рис. 2.3) ілюструє послідовність дій, які виконує медичний працівник. Після авторизації працівник має доступ до таких опцій: «Пацієнт», «Місця та палати», «Запис на відвідування» та «Власний профіль». У розділі "Пацієнт" він може додати нового пацієнта через заповнення форми, яка проходить валідацію. У розділі «Місця та палати» працівник може звільнити місце або переглянути заповненість місць. У розділі «Запис на відвідування» працівник створює новий запис через заповнення форми, що також проходить валідацію. У розділі «Власний профіль» медичний працівник може змінити свої дані або переглянути профіль.

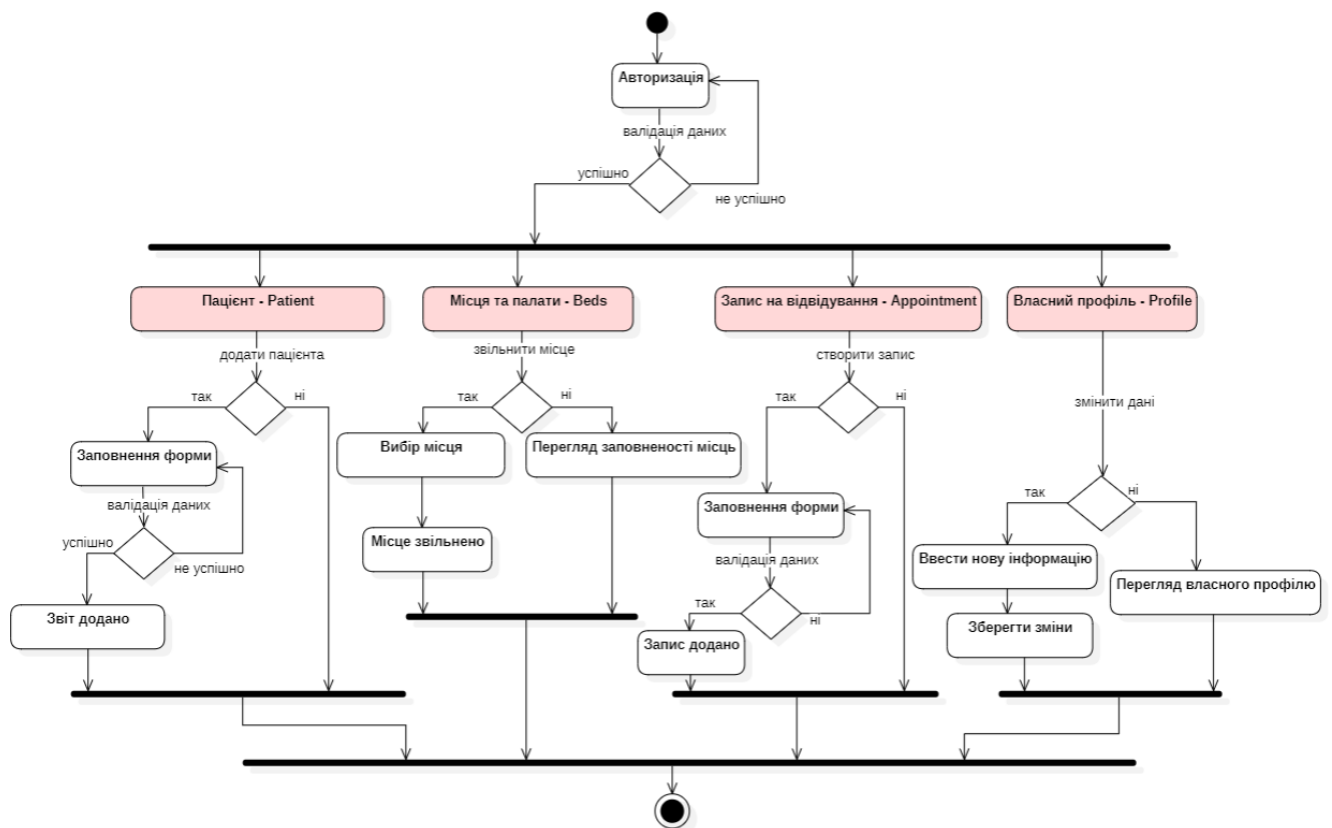


Рис. 2.3. Діаграма активності для ролі Медичний персонал

Діаграма активності на рисунку 2.4 демонструє дії адміністратора. Після авторизації адміністратор має доступ до таких опцій: «Лікар», «Медичний персонал», «Місця та палати», «Медичний транспорт» та «Адміністратор». У

розділі «Лікар» адміністратор може додати або видалити лікаря через заповнення форми та валідацію даних. Аналогічним чином адміністратор може додати або видалити медичного працівника. У розділі «Місця та палати» адміністратор додає або звільняє місця. Також він може додати новий транспорт, а у розділі «Адміністратор» - нового адміністратора після заповнення відповідної форми та валідацію даних.

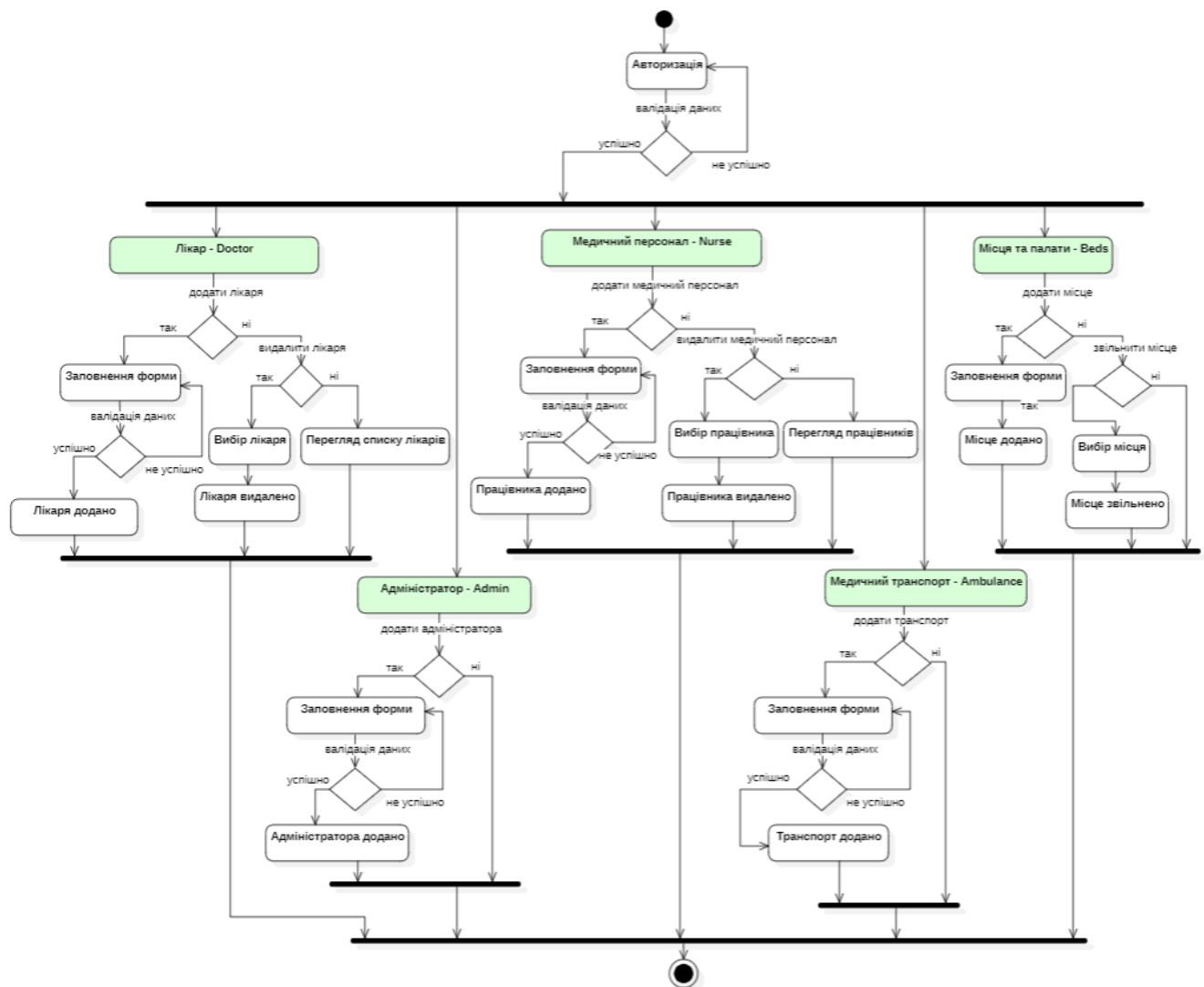


Рис. 2.4. Діаграма активності для ролі Адміністратор

2.3. Проектування бази даних програмного продукту

Від належного планування структури бази даних залежать ефективність, надійність і продуктивність всього програмного забезпечення. Якісне проектування забезпечує зручне та швидке отримання необхідних даних,

мінімізує ризик помилок і дозволяє легко масштабувати систему у разі збільшення обсягу даних.

Отже, обрана раніше MongoDB є нереляційною базою даних, яка використовує документ-орієнтований підхід до зберігання даних. Основні переваги її використання:

1. Гнучкість схеми – дозволяє зберігати дані без фіксованої схеми.
2. Швидкодія – MongoDB відома своєю високою швидкістю завдяки вбудованому індексуванню та підтримці запитів.
3. Масштабованість – ця система баз даних добре масштабується горизонтально, що дозволяє легко розширювати обсяги даних та оброблювати великі обсяги трафіку.

У MongoDB дані зберігаються у вигляді документів, що організовані у колекції. Кожен документ є набором пар «ключ-значення» і може мати довільну структуру [13]. Отже, далі опишемо будову колекції спроектованої БД.

Таблиця 2.1 відображає структуру документа, що слугує для зберігання адміністраторів у базі даних.

Таблиця 2.1

Колекція «admins»

Назва ключа	Тип	Опис поля
userType	string	Тип користувача (роль)
adminID	number	Унікальний ідентифікатор адміністратора
adminName	string	Ім'я адміністратора
email	string	Email адміністратора
password	string	Хеш пароля адміністратора
gender	string	Стать адміністратора
age	number	Вік адміністратора
mobile	string	Номер телефону адміністратора
DOB	date	Дата народження адміністратора

Продовження таблиці 2.1

address	string	Адреса адміністратора
education	string	Освіта адміністратора
image	string	Посилання на зображення адміністратора

Таблиця 2.2 відображає структуру документів, що представляють дані лікарів у колекції.

Таблиця 2.2

Колекція «doctors»

<i>Назва ключа</i>	<i>Тип</i>	<i>Опис поля</i>
userType	string	Тип користувача (роль)
docID	number	Унікальний ідентифікатор лікаря
docName	string	ПІБ лікаря
mobile	string	Номер телефону лікаря
email	string	Email лікаря
password	string	Хеш пароля лікаря
age	number	Вік лікаря
gender	string	Стать лікаря
bloodGroup	string	Група крові лікаря
DOB	date	Дата народження лікаря
address	string	Адреса лікаря
education	string	Освіта лікаря
department	string	Відділення/спеціалізація лікаря
image	string	Посилання на зображення лікаря

Таблиця 2.3 відображає структуру документа для даних медичного персоналу колекції «nurses».

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Колекція «nurses»

<i>Назва ключа</i>	<i>Тип</i>	<i>Опис поля</i>
userType	string	Тип користувача (роль)
nurseID	number	Унікальний ідентифікатор медпрацівника
nurseName	string	ПІБ медпрацівника
mobile	string	Номер телефону медпрацівника
email	string	Email медпрацівника
password	string	Хеш пароля медпрацівника
age	number	Вік медпрацівника
gender	string	Стать медпрацівника
bloodGroup	string	Група крові медпрацівника
DOB	string	Дата народження медпрацівника
address	string	Адреса медпрацівника
education	string	Освіта медпрацівника
image	string	Посилання на зображення медпрацівника

У таблиці 2.4 описано структуру колекції даних пацієнтів медичного закладу.

Таблиця 2.4

Колекція «patients»

<i>Назва ключа</i>	<i>Тип</i>	<i>Опис поля</i>
userType	string	Тип користувача (роль)
userType	string	Тип користувача (роль)
patientID	number	Унікальний ідентифікатор пацієнта
patientName	string	ПІБ пацієнта
mobile	number	Номер телефону пацієнта
email	string	Email пацієнта
age	number	Вік пацієнта

Продовження таблиці 2.4

department	string	Відділення, до якого належить пацієнт
gender	string	Стать пацієнта
DOB	string	Дата народження пацієнта
address	string	Адреса пацієнта
disease	string	Хвороба пацієнта
details	string	Деталі щодо стану пацієнта
admitted	boolean	Прийнятий у лікарню чи ні
date	date/time	Дата прийому в лікарню
docID	string	ID лікаря, призначеного для пацієнта

Таблиця 2.5 відображає структуру документа, що представляє запис на відвідування лікаря в колекції «appointments».

Таблиця 2.5

Колекція «appointments»

Назва ключа	Тип	Опис поля
userType	string	Тип користувача (роль)
patientName	string	ПІБ пацієнта
mobile	string	Номер телефону пацієнта
email	string	Email пацієнта
address	string	Адреса пацієнта
disease	string	Хвороба пацієнта
department	string	Спеціальність лікаря
time	string	Час візиту
date	string	Дата візиту
age	number	Вік пацієнта
gender	string	Стать пацієнта

У таблиці 2.6 представлена структура документа для колекції звіту.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Колекція «reports»

Назва ключа	Тип	Опис поля
docName	string	ПІБ лікаря
docDepartment	string	Відділення лікаря
docMobile	string	Номер телефону лікаря
medicines	array	Масив ліків, які призначені пацієнту
extrainfo	string	Додаткова інформація про звіт
patientName	string	ПІБ пацієнта
patientAge	number	Вік пацієнта
email	string	Email пацієнта
patientGender	string	Стать пацієнта
patientMobile	string	Номер телефону пацієнта
patientBloodGroup	string	Група крові пацієнта
patientDisease	string	Хвороба пацієнта
patientTemperature	number	Температура пацієнта
patientWeight	number	Вага пацієнта
patientBP	number	Артеріальний тиск пацієнта
patientGlucose	number	Рівень глюкози в крові пацієнта
date	string	Дата звіту
time	string	Час звіту

Таблиця 2.7 містить структуру документа, що описує дані, які зберігаються у колекції «ambulances» для даних про медичний транспорт.

Таблиця 2.7

Колекція «ambulances»

Назва ключа	Тип	Опис поля
type	string	Тип медичного транспорту
charges	number	Вартість послуг
ambulanceID	number	Унікальний ідентифікатор транспорту

ambulanceDriver	string	ПІБ водія медичного транспорту
number	number	Номер медичного транспорту

Таблиця 2.8 відображає структуру документа, що представляє місце (ліжко) в MongoDB колекції «beds».

Таблиця 2.8

Колекція «beds»

<i>Назва ключа</i>	<i>Тип</i>	<i>Опис поля</i>
bedNumber	number	Номер ліжка
roomNumber	number	Номер кімнати
occupied	string	Статус зайнятості

2.4. Архітектура та узагальнена структура вебдодатку

Вибір правильної архітектури програмної продукту визначає успішність проекту з кількох ключових причин. Правильна архітектура забезпечує масштабованість системи, її здатність ефективно зростати та пристосовуватися до змін. Вона спрощує підтримку та розвиток вебдодатку, роблячи розуміння, модифікацію та виправлення помилок більш простими і ефективними.

Отже, для розробки вебдодатку, який забезпечує роботу медичного закладу, була обрана клієнт-серверна архітектура з кількох вагомих причин. По-перше, така архітектура забезпечує централізований доступ до даних, що є критичним для медичного середовища, де швидкий та надійний доступ до інформації є важливим. Крім того, клієнт-серверна модель дозволяє ефективно керувати ресурсами, такими як обробка даних та обчислення, розподіляючи їх між сервером і клієнтськими пристроями. Вона спрощує підтримку та управління вебсайтом, адже сервер може централізовано керувати базою даних та логікою додатку, забезпечуючи однаковий інтерфейс для різних клієнтів. Надійність і безпека також підсилюються завдяки централізованому керуванню доступом до даних і заходами захисту. Таким чином, клієнт-серверна архітектура є

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		28

оптимальним вибором для медичних вебдодатків, оскільки вона забезпечує ефективність, надійність і безпеку в обробці та зберіганні чутливої медичної інформації [14].

Клієнтська частина вебдодатку є ключовим елементом, що забезпечує взаємодію користувача з вебдодатком через веб-браузер. Вона складається з кількох важливих компонентів і папок, які сприяють організації та ефективному розгортанню додатку.

На рисунку 2.5 показано узагальнену структуру вебдодатку, яка ілюструє взаємодію різних компонентів та модулів клієнтської частини вебдодатку для медичного закладу. В основі є папка src, яка містить всю основну логіку JavaScript програми. Тут розміщуються компоненти «Components», що становлять логічні частини інтерфейсу, а також сторінки «Pages», що визначають різні відображення та функціональні можливості додатку.

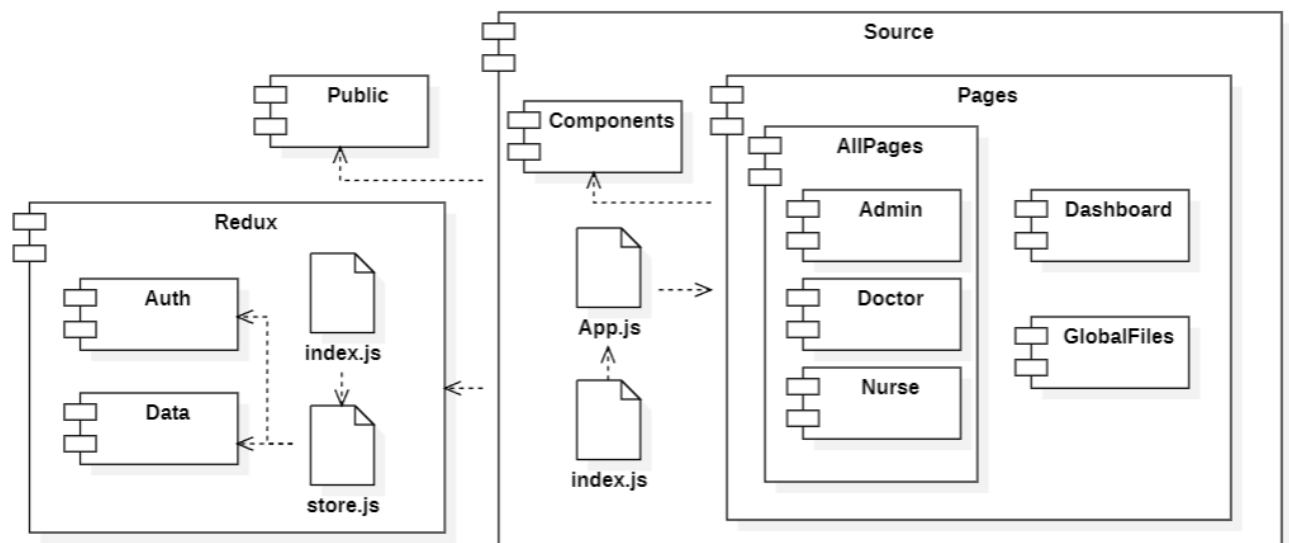


Рис. 2.5. Діаграма компонентів клієнтської частини вебдодатку

Для зручності управління ресурсами і стилізації використовуються папки public для статичних файлів, таких як HTML, CSS та зображення, а також App.css для глобальних CSS-стилів. Компоненти, пов'язані з конкретними функціональними частинами, такими як вхід «Dashboard-Login», головна сторінка дашборду «Main-Dashboard», а також сторінки для різних типів користувачів

«Admin, Doctor, Nurse», відокремлені для зручності управління та розширення функціональності.

Для керування станом додатку і взаємодії між компонентами можливо використовується Redux, що дозволяє ефективно управляти даними та їх станами. Крім того, маршрутизація «Routes» забезпечує навігацію між різними сторінками та компонентами додатку, забезпечуючи користувачеві зручність та логічність у використанні.

На рисунку 2.6 показана типова структура серверної частини вебдодатку, побудованого на Node.js. Основний каталог «Backend» містить ключові компоненти, які забезпечують функціональність та обробку запитів до сервера.

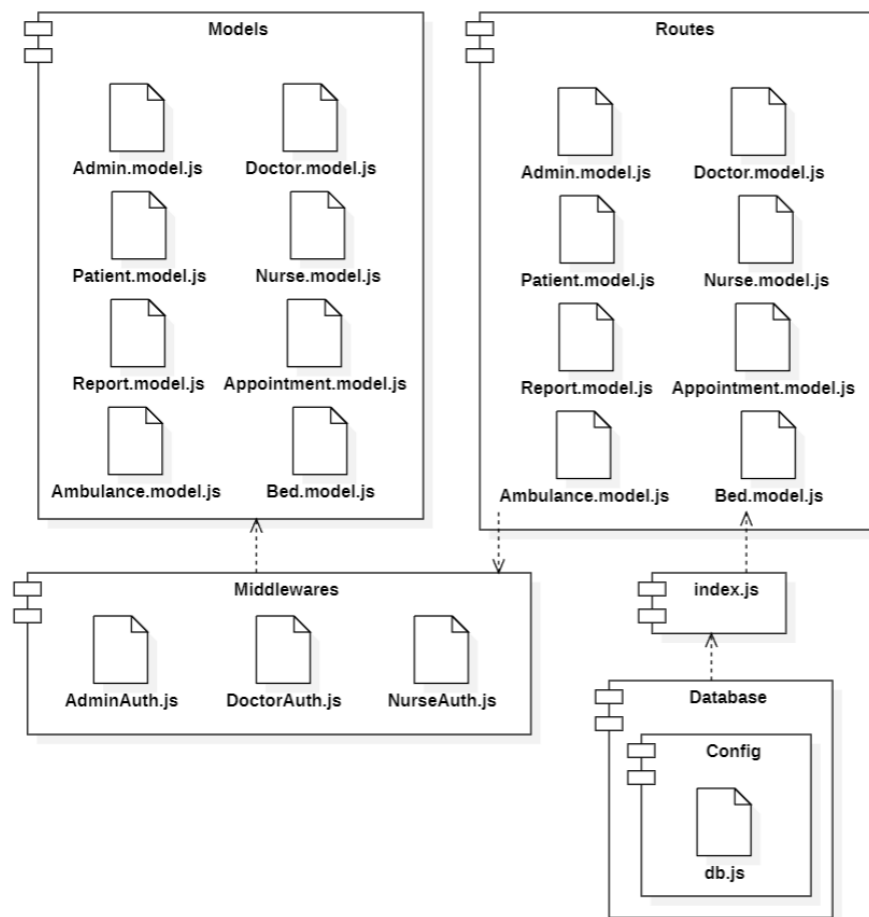


Рис. 2.6. Діаграма компонентів серверної частини вебдодатку

Каталог «configs» містить конфігураційні файли, які визначають параметри сервера, такі як налаштування бази даних, з'єднання з API та середовища розгортання, що сприяє керуваності і підтримці додатку.

Папка «middlewares» містить middleware-функції, які обробляють запити до сервера перед тим, як вони досягнуть кінцевих точок, забезпечуючи обробку аутентифікації, авторизації та інших важливих завдань. Каталог «models» містить визначення моделей даних, які використовуються в додатку, і готує дані для зберігання в базі даних або обміну з клієнтською частиною.

«Routes» є каталогом, що містить маршрутизацію запитів, яка визначає, як запити до сервера повинні бути оброблені і які дії повинні бути виконані для кожного маршруту. Основний файл «index.js» є головним файлом запуску сервера, який ініціалізує сервер та налаштовує його за допомогою вказаних конфігурацій і маршрутів.

Ще одним важливим інструментом у процесі проектування програмних продукту, який ілюструє фізичне розташування компонентів системи на апаратному забезпеченні, є діаграма розгортання. Вона показує, як різні частини програмного забезпечення розгорнуті на фізичних або віртуальних серверах, комп'ютерах, або інших пристроях у мережі [15].

На рисунку 2.7 зображено компоненти клієнт-серверної архітектури вебдодатку, що містять кілька ключових елементів. Клієнт містить логічні частини інтерфейсу, сторінки додатку та стан додатку, керований Redux.

Серверна частина включає код серверної частини, функції обробки запитів, моделі даних, маршрутизацію запитів та конфігураційні файли для сервера.

База даних містить колекції для зберігання медичних даних.

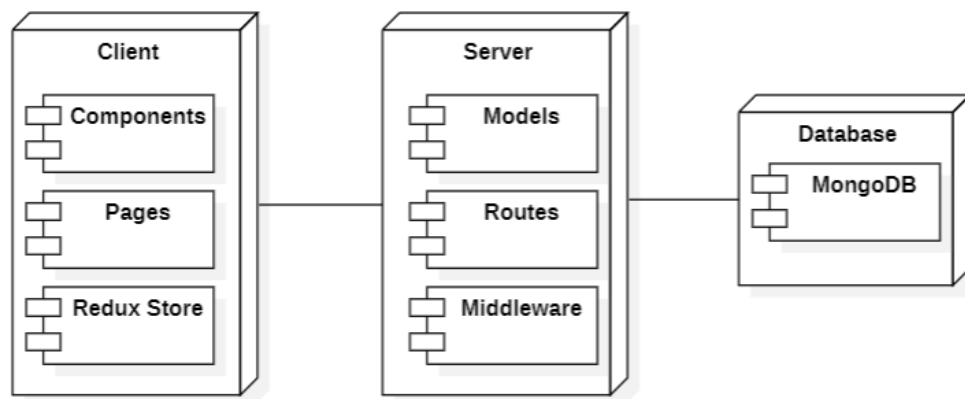


Рис. 2.7. Діаграма розгортання вебдодатку

2.5. Розробка вебдодатку для забезпечення роботи персоналу медичного закладу

Після завершення етапів проектування варіантів використання, розробки алгоритмів роботи та створення узагальненої структури вебдодатку, наступним кроком є написання програмного коду. Враховуючи, що для реалізації проекту було обрано клієнт-серверну архітектуру, компоненти проекту розділимо на дві теки: «Frontend» та «Backend».

Далі для кожної теки окремо створимо файли «package.json», що містять інформацію про проект, список залежностей, скрипти для запуску, білду та тестування коду. Створення файлів «package.json» для кожної теки дозволить розділити фронтенд та бекенд частини проекту та забезпечити незалежність їхнього використання.

Наступним кроком налаштуємо підключення до бази даних. Для цього було використано бібліотеку Mongoose. Вона надає простий і зручний спосіб взаємодії з MongoDB за допомогою схем і моделей. Mongoose дозволяє визначати схеми для даних, валідувати їх, виконувати запити до бази даних та виконувати різні операції з даними [16].

Спочатку було створено файл «.env» для зберігання конфіденційних даних, таких як URL до бази даних та секретні ключі. Файл «db.js» забезпечує підключення до бази даних MongoDB. Спочатку імпортується Mongoose та dotenv, а потім завантажуються змінні оточення з файлу «.env». Використовуючи метод connect, встановлюється з'єднання з MongoDB за URL, збереженим у змінній «process.env.dbURL». З'єднання експортується для використання в інших частинах проекту, що забезпечує безпечне та зручне підключення до бази даних із ізоляцією конфіденційних даних.

```
const mongoose = require("mongoose");
require("dotenv").config();

const connection = mongoose.connect(process.env.dbURL);

module.exports = { connection };
```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Після налаштування підключення до бази даних, наступним кроком є створення моделей для взаємодії з MongoDB. Додаємо окрему теку «models» (рис. 2.8) до проекту та створюємо моделі, які представлятимуть структуру даних у базі даних. Були розроблені такі моделі: «Admin.model.js», «Doctor.model.js», «Nurse.model.js», «Patient.model.js», «Appointments.model.js», «Reports.model.js», «Ambulances.model.js» та «Beds.model.js».

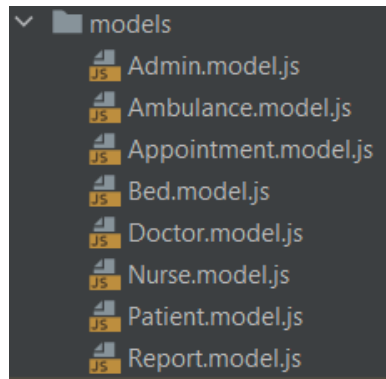


Рис. 2.8. Вміст теки «models»

Наступним кроком є розробка маршрутів для обробки HTTP-запитів. Розглянемо приклад файлу «Doctor.route.js», який відповідає за маршрути для моделі «Doctor».

Спочатку імпортуються необхідні бібліотеки express, dotenv, jsonwebtoken, bcryptjs, а також модель «DoctorModel» для взаємодії з колекцією лікарів у базі даних. Використовується express.Router() для створення об'єкта маршрутизатора. Далі визначаються потрібні маршрути. Маршрут «GET» повертає список лікарів, які відповідають заданому запиту; «POST /register» перевіряє, чи існує лікар з таким email, і якщо ні, створює нового лікаря та зберігає його у базі даних; «POST /login» перевіряє облікові дані лікаря (docID та пароль) і повертає токен JWT при успішній автентифікації; «PATCH /:doctorId» оновлює інформацію про лікаря за заданим doctorId, а маршрут «DELETE /:doctorId» видаляє лікаря за заданим doctorId.

```
const express = require("express");
const { DoctorModel } = require("../models/Doctor.model");
require("dotenv").config();
const jwt = require("jsonwebtoken");
const bcrypt = require('bcryptjs');
const router = express.Router();

router.get("/", async (req, res) => {
```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		33

```

let query = req.query;
try {
  const doctors = await DoctorModel.find(query);
  res.status(200).send(doctors);
} catch (error) {
  console.log(error);
  res.status(400).send({ error: "Something went wrong" });
}
});

router.post("/register", async (req, res) => {
  const { email } = req.body;
  try {
    const doctor = await DoctorModel.findOne({ email });
    if (doctor) {
      return res.send({
        message: "Doctor already exists",
      });
    }
    let value = new DoctorModel(req.body);
    await value.save();
    const data = await DoctorModel.findOne({ email });
    return res.send({ data, message: "Registered" });
  } catch (error) {
    res.send({ message: "error" });
  }
});

router.post("/login", async (req, res) => {
  const { docID, password } = req.body;
  try {
    const doctor = await DoctorModel.findOne({ docID });
    if (doctor) {
      const isMatch = await bcrypt.compare(password, doctor.password);

      if (isMatch) {
        const token = jwt.sign({ foo: "bar" }, process.env.key, {
          expiresIn: '24h',
        });
        res.send({ message: 'Successful', user: doctor, token });
      } else {
        res.status(401).send({ message: 'Wrong credentials' });
      }
    } else {
      res.status(401).send({ message: 'Wrong credentials' });
    }
  } catch (error) {
    console.log({ message: "Error" });
    console.log(error);
  }
});

router.patch("/:doctorId", async (req, res) => {
  const id = req.params.doctorId;
  const payload = req.body;
  try {
    await DoctorModel.findByIdAndUpdate({ _id: id }, payload);
    const doctor = await DoctorModel.findById(id);
    if (!doctor) {
      return res
        .status(404)
        .send({ message: `Doctor with id ${id} not found` });
    }
  }
});

```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		34

```

    res.status(200).send({ message: `Doctor Updated`, user: doctor });
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Update." });
  }
});

router.delete("/:doctorId", async (req, res) => {
  const id = req.params.doctorId;
  try {
    const doctor = await DoctorModel.findByIdAndDelete({ _id: id });
    if (!doctor) {
      res.status(404).send({ msg: `Doctor with id ${id} not found` });
    }
    res.status(200).send(`Doctor with id ${id} deleted`);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Delete." });
  }
});

module.exports = router;

```

Інші маршрути вебдодатку також додано до теки «routes» (рис. 2.9), вони мають схожу структуру і включають основні операції CRUD для відповідної моделі.

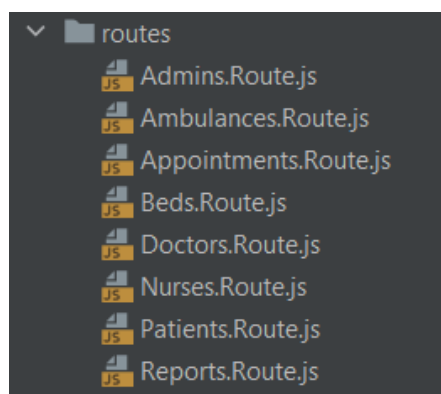


Рис. 2.9. Вміст теки «routes»

Далі створюємо файл «index.js», що є основою вебдодатку, побудованого на Express.js.

Спочатку відбувається імпорт необхідних модулів, налаштування підключення до бази даних MongoDB, завантаження змінних середовища з файлу «.env» та налаштування CORS. Створюється екземпляр додатку Express, налаштовуються обробники для JSON і CORS. Додаток визначає простий маршрут для кореневого URL і підключає маршрути для різних ресурсів, таких як адміністратори, лікарі, пацієнти тощо. Завершується файл викликом `app.listen()` для запуску сервера на вказаному порті. Ця структура забезпечує гнучкий та

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		35

надійний механізм обробки HTTP-запитів і взаємодії з базою даних для вебдодатку.

```
const express = require("express");
const { connection } = require("../configs/db");
require("dotenv").config();
const cors = require("cors");

const adminRouter = require("../routes/Admins.Route");
const ambulanceRouter = require("../routes/Ambulances.Route");
const appointmentRouter = require("../routes/Appointments.Route");
const bedRouter = require("../routes/Beds.Route");
const doctorRouter = require("../routes/Doctors.Route");
const hospitalRouter = require("../routes/Hospitals.Route");
const nurseRouter = require("../routes/Nurses.Route");
const patientRouter = require("../routes/Patients.Route");
const reportRouter = require("../routes/Reports.Route");

const app = express();

app.use(express.json());
app.use(cors());

app.get("/", (req, res) => {
  res.send("Homepage");
});

app.use("/admin", adminRouter);
app.use("/ambulances", ambulanceRouter);
app.use("/appointments", appointmentRouter);
app.use("/beds", bedRouter);
app.use("/doctors", doctorRouter);
app.use("/hospitals", hospitalRouter);
app.use("/nurses", nurseRouter);
app.use("/patients", patientRouter);
app.use("/reports", reportRouter);

app.listen(process.env.port, async () => {
  try {
    await connection;
    console.log("Connected to DB");
  } catch (error) {
    console.log("Unable to connect to DB");
    console.log(error);
  }
  console.log(`Listening at port ${process.env.port}`);
});
```

Далі переходимо до розробки структури сторінок. Тека «pages» (рис. 2.10) організована в кілька рівнів, що сприяє систематизації та зручній розробці програмного коду. Починаючи з теки «Dashboard», вона містить дві підтеки: «Dashboard-Login» і «Main-Dashboard».

У підтеці «Dashboard-Login» знаходяться два основних файли, які відповідають за логіку та відображення сторінки входу. Підтека «Main-Dashboard» розділена на «AllPages» і «GlobalFiles». У «AllPages» додамо окремі

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		36

підтеки для адміністраторів, лікарів і медсестер з відповідними сторінками для кожної групи користувачів.

«GlobalFiles» включає загальні компоненти та стилі, які використовуються на всіх сторінках панелі керування. Тому створимо «CommonCSS.css» для загальних стилів, «FrontPage.jsx» для загального компонента головної сторінки, а також компонент навігації «Sidebar.jsx» для бокової панелі.

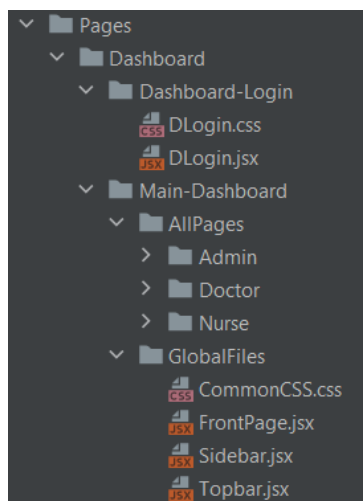


Рис. 2.10. Вміст теки «pages»

Наступним кроком переходимо до реалізації патерну управління станом додатку за допомогою Redux в середовищі React. Redux – це потужний інструмент для управління станом додатку в JavaScript додатках. Він дозволяє централізовано зберігати стан всього додатку в Redux-сторі, що роблює його доступним для всіх компонентів безпосередньо [17].

Створюємо теку «Redux» (рис. 2.11), що міститиме кілька підтек з різними модулями функціональності.

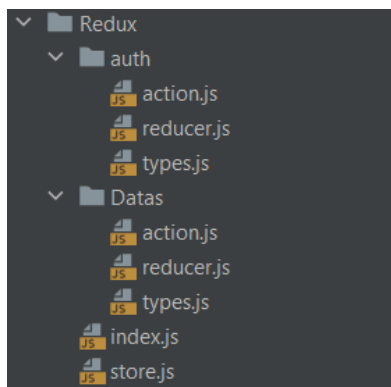


Рис. 2.11. Вміст теки «Redux»

Для реалізації запитів використаємо бібліотеку Axios, що є потужним інструментом для здійснення HTTP-запитів у JavaScript додатках. Вона надає простий і зрозумілий інтерфейс для виконання різних видів запитів (GET, POST, PUT, DELETE тощо) до вебсерверів і обробки відповідей [18].

У теці «auth» знаходяться файли, що стосуються автентифікації користувачів. Файл «action.js» містить функції для створення дій, таких як вхід та вихід. Наведемо приклад фрагменту даного файлу.

```
import * as types from "../types";
import axios from "axios";

export const NurseLogin = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.LOGIN_NURSE_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/nurses/login",
      data
    );
    dispatch({
      type: types.LOGIN_NURSE_SUCCESS,
      payload: {
        message: res.data.message,
        user: res.data.user,
        token: res.data.token,
      },
    });
    return res.data;
  } catch (error) {
    dispatch({
      type: types.LOGIN_NURSE_ERROR,
      payload: {
        message: error,
      },
    });
  }
};
```

У першому рядку імпортується файл «types.js», який містить всі константи типів дій, необхідні для ідентифікації дій у редукторах та діях. Далі, за допомогою бібліотеки Axios, здійснюється HTTP-запит до сервера за URL-адресою «http://localhost:5000/nurses/login». Цей запит передає дані, отримані через параметр data, який містить інформацію, необхідну для автентифікації медичного працівника.

Далі редуктор у файлі «reducer.js» обробляє ці дії, змінюючи стан «auth» в Redux-сторі. Файл «types.js» містить константи типів дій, які використовуються для ідентифікації цих дій у редукторі.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Таку саму структуру має тека «Datas», але вона містить файли для управління даними. Файл «action.js» відповідає за функції для отримання, оновлення та інших операцій з даними з сервера. Редуктор у «reducer.js» обробляє відповідні дії, що стосуються управління станом даних в Redux. Файл «types.js» визначає константи типів дій для цих операцій.

Файл «index.js» використаємо як точку входу для комбінування всіх редукторів у кореневий редуктор і експортування його для створення Redux-сторю.

```
import { combineReducers } from "redux";
import authReducer from "../auth/reducer";
import dataReducer from "../Datas/reducer";
export const rootReducer = combineReducers({
  auth: authReducer,
  data: dataReducer,
});
```

На кінцевому етапі написання програмного коду додаємо файл «index.js», що є ключовим елементом конфігурації React додатка, який забезпечує початкову точку рендерингу для всіх інших компонентів та функцій додатку.

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "../App";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import { store } from "../Redux/store";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <BrowserRouter>
    <Provider store={store}>
      <App />
    </Provider>
  </BrowserRouter>
);
```

Висновки до другого розділу

Другий розділ було присвячено проектуванню та розробці вебдодатку для роботи персоналу медичного закладу. Визначено варіанти використання, бізнес-вимоги та потреби користувачів, зокрема адміністратора, лікаря та медичного персоналу. Функціональні вимоги охоплювали авторизацію, управління користувачами, медичними записами, ресурсами, а також безпекою та конфіденційністю даних. Нефункціональні вимоги включали доступність, швидкодію, масштабованість та сумісність з різними веббраузерами. Системні

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		39

вимоги передбачали підтримку Windows, базу даних MongoDB та мінімальні апаратні параметри.

Після аналізу вимог створено діаграми варіантів використання та активності, що візуалізують основні функції та послідовність дій користувачів. Проектування бази даних на MongoDB забезпечило ефективність і масштабованість. Вибрано клієнт-серверну архітектуру, що забезпечує централізований доступ до даних та високий рівень безпеки. Клієнтська частина вебдодатку забезпечує взаємодію користувачів з вебдодатком, а серверна частина на Node.js обробляє запити.

Таким чином, другий розділ висвітлив ключові аспекти проектування та розробки вебдодатку, створивши основу для подальшої розробки функціонального, надійного та безпечного програмного продукту для ефективної роботи медичного персоналу.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		40

РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ ВЕБДОДАТКУ

3.1. Структура інтерфейсу та опис роботи вебдодатку

Отже, перейдемо до розгляду основних компонентів, з яких складається користувацький інтерфейс, принципи його проектування та взаємодії з користувачем.

Після завантаження вебдодатку спочатку користувач потрапляє на сторінку входу (рис. 3.1). Тут він має обрати одну із трьох доступних ролей – Медичний персонал, Лікар, Адміністратор. Потім потрібно ввести індивідуальний ідентифікатор та пароль до форми. Якщо дані коректні, користувач здійснює вхід до вебдодатку, в іншому разі – буде відображено повідомлення про некоректність даних.

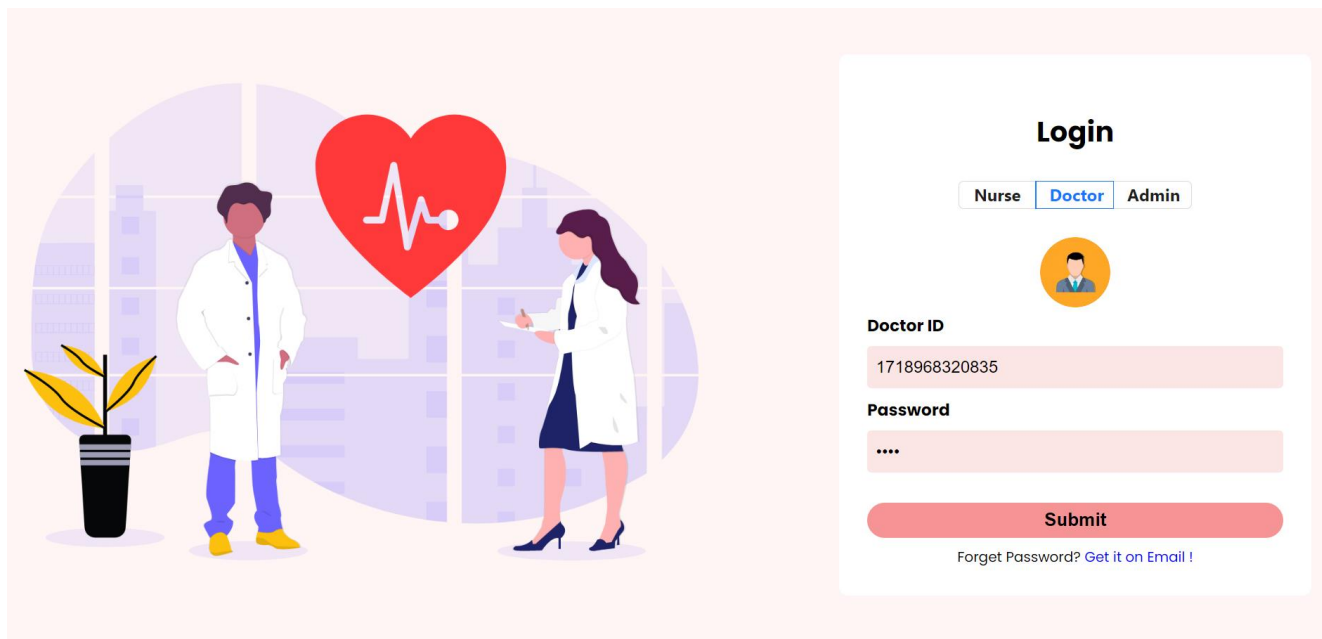


Рис. 3.1. Сторінка входу

Авторизований користувач потрапляє на головну сторінку (рис. 3.2), де відображається загальна інформація про медичний заклад. Також на сторінці відображається бічне меню з функціоналом відповідно до ролі.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		41

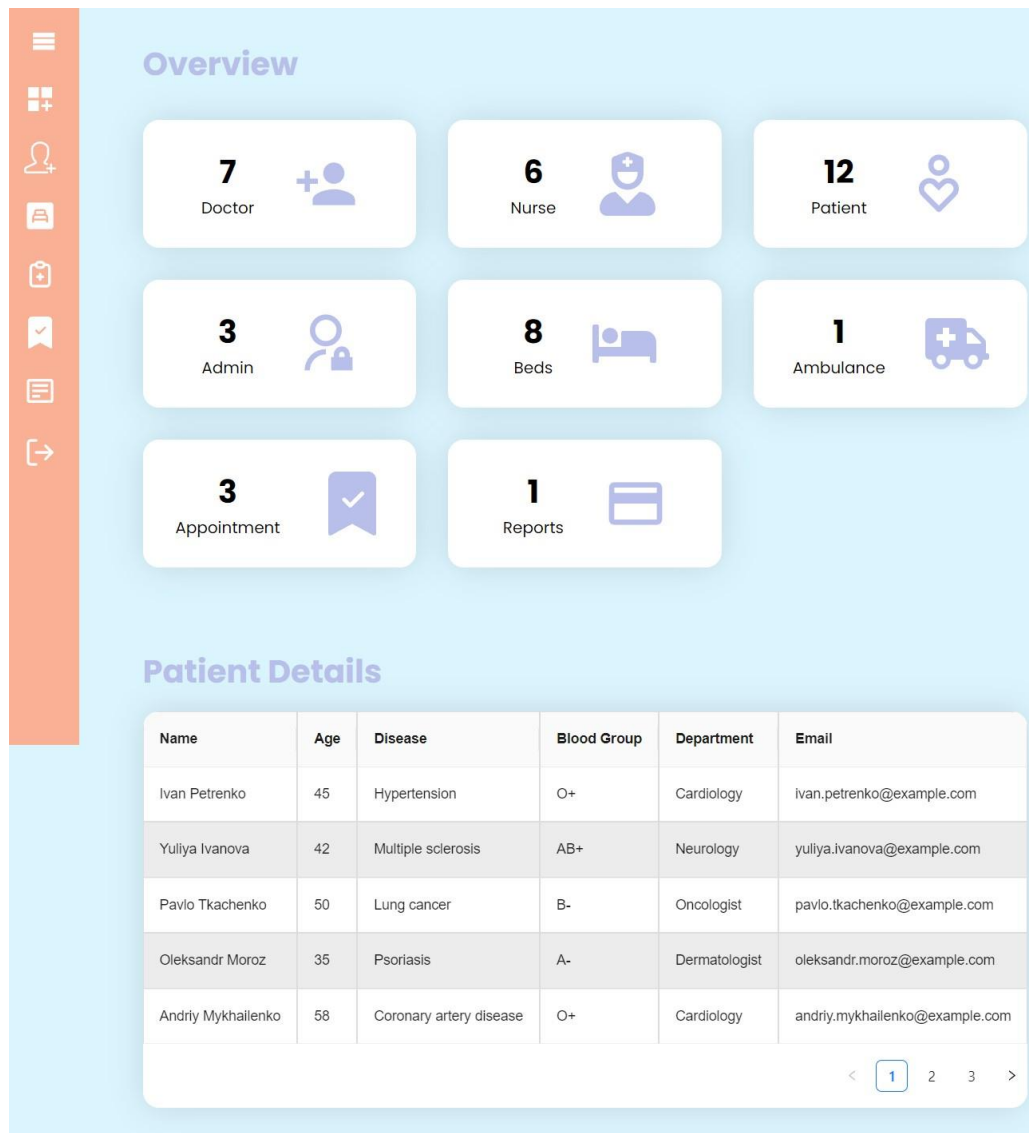


Рис. 3.2. Головна сторінка

Користувач, який має роль адміністратора, може додавати інформацію про лікарів до бази даних. Для цього адміністратор заповнює відповідну форму (рис. 3.3), де вказує потрібні дані.

Рис. 3.3. Форма для додавання лікаря

Якщо валідацію цих даних здійснено успішно, то відображається повідомлення про додавання інформації до бази даних (рис. 3.4). Якщо лікар з такими даними вже існує або якщо дані не коректні, то адміністратор отримає відповідне повідомлення.

Рис. 3.4. Повідомлення про успішне додавання інформації

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Аналогічно адміністратор має можливість додати новий медичний персонал (рис. 3.5) та нового адміністратора (рис. 3.6). Для цього потрібно обрати відповідні пункти меню та заповнити форми.

Add Nurse



Name: Dmytro Bondarenko

Age: 27

Contact Number: 0939876987

Email: dbondarenko@example.com

Gender: Male

Birthdate: 15.08.1997

Address: Zhytomyr, Ukraine

Education: ND, Kyiv Medical College

Blood Group: AB+

Password: dmytro

Other Info: Proficient in wound care and rehabilitation.

Submit

Рис. 3.5. Форма для додавання медичного персоналу

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Add Admin



Name: Oksana Chernenko

Age: 36

Contact Number: 0963214567

Email: ochernenko@example.com

Gender: Female

Birthdate: 16.06.1988

Address: Zhytomyr, Ukraine

Education: BA in Health Administration, Lviv National Universi

Password: oksana

Submit

Рис. 3.6. Форма для додавання адміністратора

Також адміністратор може додавати до бази даних записи про місця у палатах медичного закладу (рис. 3.7). Для цього він вносить до форми відповідні номери.

Add Beds

Bed Number: 1

Room Number: 4

Submit

Рис. 3.7. Форма для додавання місць у палатах медичного закладу

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Наступною можливістю адміністратора є додавання даних про медичний транспорт (рис. 3.8). Після внесення інформації до форми та підтвердження, відображається повідомлення про додавання запису до бази даних.

Add Ambulance

Ambulance Type: Basic Life Support Ambulance

Ambulance Code: AM3421PB

Driver Name: Artem Kovalenko

Driver Contact No: 0935285212

Submit

Рис. 3.8. Форма для додавання даних про медичний транспорт

Користувач, який має роль адміністратора, може переглядати списки наявних лікарів та медичного персоналу у базі даних (рис. 3.9). На даних сторінках доступна можливість фільтрації даних відповідно до обраного параметра випадного списку. Також адміністратор може видаляти записи про користувачів, натиснувши «Delete» навпроти потрібного запису.

All Doctors

Doctor Name

Filter by docName

Doctor Name	Mobile Number	Email	Age	Gender	Date of birth	Department	
Dr. Natalia Petrenko	0987654321	npetrenko@example.com	42	Female	15.03.1982	Cardiology	Delete
Dr. Ivan Mykhailenko	0955678945	imykhalenko@example.com	35	Male	17.07.1989	Neurology	Delete
Dr. Olena Hryhorova	0934567890	ohryhorova@example.com	38	Female	12.10.1986	Oncologist	Delete
Dr. Yulia Sokolova	0963214578	ysokolova@example.com	45	Female	30.04.1979	Pediatrics	Delete
Dr. Volodymyr Kovalenko	0997890123	vkovalenko@example.com	50	Male	15.02.1974	Dermatologist	Delete
Dr. Kateryna Sydorenko	0968765432	ksydorenko@example.com	25	Female	12.10.1995	Dermatologist	Delete
Dr. Olena Ivanenko	0501234567	olena.ivanenko@example.com	45	Female	16.05.1979	Cardiology	Delete

Рис. 3.9. Сторінка з інформацією про лікарів

Також адміністратор має можливість переглянути інформацію про місця та їх заповненість (рис. 3.10). У разі потреби він може звільнити потрібне місце у палаті.

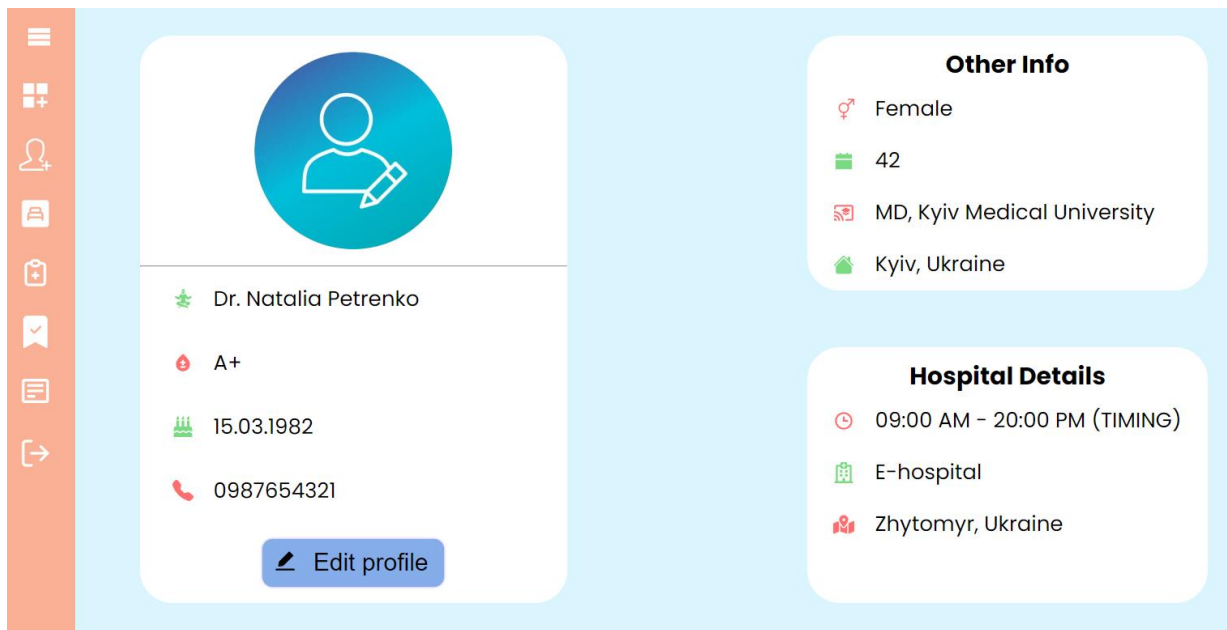


All Beds

Room	Bed	Status	Patient	Disease	Doctor	Discharge
1	1	occupied	Ivan Petrenko	Hypertension	Dr. Natalia Petrenko	Discharge
1	2	available	No Data	No Data	No Data	Discharge
2	1	occupied	Pavlo Tkachenko	Lung cancer	Dr. Olena Hryhorova	Discharge
2	2	occupied	Oleksandr Moroz	Oleksandr Moroz	Dr. Volodymyr Kovalenko	Discharge
3	1	available	No Data	No Data	No Data	Discharge
3	2	available	No Data	No Data	No Data	Discharge
4	1	available	No Data	No Data	No Data	Discharge
4	2	available	No Data	No Data	No Data	Discharge

Рис. 3.10. Сторінка з інформацією про місця та їх заповненість

Користувач, який має роль лікаря, може переглядати та редагувати інформацію власного профілю (рис. 3.11).



Other Info

- Female
- 42
- MD, Kyiv Medical University
- Kyiv, Ukraine

Hospital Details

- 09:00 AM - 20:00 PM (TIMING)
- E-hospital
- Zhytomyr, Ukraine

Dr. Natalia Petrenko

A+

15.03.1982

0987654321

Edit profile

Рис. 3.11. Сторінка профілю

Також лікар може додавати до бази даних звіти про пацієнтів (рис. 3.12). Для цього він вносить до форми відповідні дані та показники пацієнта.

Create Report

Doctor Name

Dr. Natalia Petrenko

Department

Cardiology

Doctor Mobile

0987654321

Patient Name

Ivan Horobets

Patient Age

58

Patient Mobile

0934567890

Email

ihorobets@example.com

Patient Gender

Male

Patient Blood Group

B+

Patient Disease

Hypertension

Patient Temperature

36

Patient Weight

85

Patient BP

150/90

Patient Glucose

5.6

Extra Info

Patient advised to follow a low-sodium diet and reg

Medicines

PCM

Durat

Dosa

Add

Date

04.06.2024

Time

12:00

Generate Report

Рис. 3.12. Форма для додавання звіту про пацієнта

Після цього додані звіти про пацієнтів відображаються на сторінці «Reports» (рис. 3.13). На даній сторінці доступна можливість фільтрації даних відповідно до обраного параметра випадного списку.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		48

All Reports

Patient Na...

Search by patientName

Patient Name	Department	Doctor Name	Patient Mobile	Patient Gender	Patient Age	Blood Group	Disease	Temperature	Weight	BP	Glucose	Date
Ivan Horobets	Cardiology	Dr. Natalia Petrenko	0965656555	Male	58	AB+	Hypertension	37	85	150/90	5.6	2024-06-04

Рис. 3.13. Сторінка з інформацією про створені звіти

Користувач, який має роль лікаря, може переглядати список запланованих візитів пацієнтів (рис. 3.14).

[illegible]

Рис. 3.14. Сторінка з інформацією про заплановані візити

Користувач, який має роль медичного персоналу, аналогічно до лікаря може переглядати та редагувати інформацію власного профілю (рис. 3.15).

Maria Ivanenko

B+

10.05.1994

0987654111

Edit details

Edit details

X

Maria Ivanenko

30

Select gender

B+

ND, Kyiv Medical College

0987654111

1994-05-10

Cancel

Edit

Other Info

Female

30

ND, Kyiv Medical College

Zhytomyr, Ukraine

Hospital Details

09:00 AM - 20:00 PM (TIMING)

E-hospitals

Zhytomyr, Ukraine

Рис. 3.15. Форма для редагування профілю

Медичний персонал має можливість додавати інформацію про пацієнтів до бази даних, заповнивши відповідну форму та вказавши необхідні дані (рис. 3.16).

Add Patient



Patient Name

Yulia Sokolova

Age

63

Email

yulia.sokolova@example.com

Date

10.06.2024

Gender

Female

Birth Date

12.07.1961

Contact Number

0504561234

Details

Taking medication to control heart rhythm.

Disease

Atrial fibrillation

Address

Zhytomyr, Ukraine

Bed Number

2

Room Number

3

Department

Neurology

Doctor

Dr. Olena Hryhorova

Patient Blood Group

O+

Submit

Рис. 3.16. Форма для додавання пацієнтів

Якщо дані пройшли валідацію успішно, з'являється повідомлення про успішне додавання інформації до бази даних (рис. 3.17). У випадку, якщо пацієнт

з такими даними вже існує або якщо дані некоректні, медичний персонал отримує відповідне попередження.

The screenshot shows a web application interface for adding a patient. On the left is a vertical sidebar with icons for home, add, patient, hospital, appointment, and a right arrow. The main content area is titled 'Add Patient' and features a doctor's profile icon. Below the title are input fields for 'Patient Name' (with a placeholder 'Full Name'), 'Age', 'Email' (with placeholder 'abc@abc.com'), 'Date' (with placeholder 'ДД.ММ.РРРР' and a calendar icon), and 'Gender' (with a 'Select Gender' dropdown). In the top right corner, there are two notification boxes: 'Patient Added' and 'Bed updated', both with close buttons.

Рис. 3.17. Повідомлення про успішне додавання інформації

Також медичний персонал може планувати візити пацієнтів (рис. 3.18). Для цього потрібно заповнити відповідну форму, де вказати потрібні дані.

The screenshot shows a web application interface for booking an appointment. On the left is a vertical sidebar with icons for home, add, patient, hospital, appointment, and a right arrow. The main content area is titled 'Book Appointment'. It contains the following fields: 'Patient Name' (filled with 'Oleksandr Ivanenko'), 'Age' (filled with '58'), 'Gender' (dropdown menu with 'Male' selected), 'Contact Number' (filled with '0501234565'), 'Email' (filled with 'oleksandr.ivanenko@example.com'), 'Type of Disease' (dropdown menu with 'Headaches' selected), 'Address' (filled with 'Zhytomyr, Ukraine'), 'Department' (dropdown menu with 'Neurology' selected), and 'Date and Time' (split into a date field with '24.06.2024' and a calendar icon, and a time field with '15:15' and a clock icon). At the bottom of the form is an orange button labeled 'Book Appointment'.

Рис. 3.18. Форма для створення візитів

Аналогічно до адміністратора медичний персонал має можливість переглядати інформацію про місця та їх заповненість (рис. 3.19), а в разі потреби може звільнити потрібне місце у палаті.



Room	Bed	Status	Patient	Disease	Doctor	Discharge
1	1	occupied	Ivan Petrenko	Hypertension	Dr. Natalia Petrenko	Discharge
1	2	available	No Data	No Data	No Data	Discharge
2	1	occupied	Pavlo Tkachenko	Lung cancer	Dr. Olena Hryhorova	Discharge
2	2	occupied	Oleksandr Moroz	Oleksandr Moroz	Dr. Volodymyr Kovalenko	Discharge
3	1	available	No Data	No Data	No Data	Discharge
3	2	available	No Data	No Data	No Data	Discharge
4	1	available	No Data	No Data	No Data	Discharge
4	2	available	No Data	No Data	No Data	Discharge

Рис. 3.19. Сторінка з інформацією про місця та їх заповненість

3.2. Тестування роботи вебдодатку

Після розробки програмного коду переходимо до важливого етапу – тестування вебдодатку. Цей етап є критичним для забезпечення якості та надійності продукту перед його впровадженням у виробниче середовище. Воно дозволяє виявити та виправити помилки, забезпечити відповідність функціоналу вимогам та підвищити якість кінцевого продукту.

Для тестування вебдодатку для забезпечення роботи персоналу медичного закладу було виконано функціональне тестування з метою забезпечення високої якості та надійності продукту перед його впровадженням в реальні умови використання. Основні причини, чому функціональне тестування є критичним етапом у розробці цього вебдодатку:

- функціональне тестування дозволяє переконатися, що розроблений вебдодаток відповідає усім функціональним вимогам, визначеним специфікацією;
- під час функціонального тестування виявляються та документуються будь-які програмні помилки, дефекти чи недоліки у функціональності додатку;
- переконатися, що вебдодаток працює стабільно та надійно в умовах реального використання [19].

Для функціонального тестування вебдодатку було використано Playwright. Playwright – це інструмент для автоматизації тестування вебдодатків, розроблений командою Microsoft. Він дозволяє створювати тести, які емулюють взаємодію з веб-браузером, виконуючи такі дії, як натискання на кнопки, введення тексту та перевірку стану веб-сторінок. Playwright підтримує різні браузери (Chrome, Firefox, Safari) і дозволяє писати тести на різних мовах програмування, таких як JavaScript, TypeScript, Python, і багатьох інших [20].

Розпочнемо з тестування додавання нового лікаря через інтерфейс адміністратора. Перший крок полягає у переході на головну сторінку за вказаною URL. Далі виконується клік по «Admin», щоб увійти як адміністратор. Після цього заповнюються форми для входу та відправляються за допомогою кнопки «Submit». Наступний етап передбачає навігацію до сторінки, де додається новий запис про лікаря або «Add Doctor», і заповнення різних полів форми, таких як ім'я, вік, номер телефону, електронна пошта, стать, група крові, дата народження, адреса, освіта, відділення, пароль та додаткова інформація. Після цього знову натискається кнопка «Submit». На завершення виконується перехід до «DashBoard» за допомогою кліку на відповідне посилання.

```
import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await page.getByText('Admin').click();
  await page.getByRole('spinbutton').click();
  await page.getByRole('spinbutton').fill('1718972626202');
  await page.getByRole('textbox').click();
  await page.getByRole('textbox').fill('yulia');
  await page.getByRole('button', { name: 'Submit' }).click();
  await page.locator('a').filter({ hasText: 'Add Doctor' }).click();
  await page.getByPlaceholder('Full Name').click();
  await page.getByPlaceholder('Full Name').fill('Test Doctor');
  await page.getByPlaceholder('Age').click();
  await page.getByPlaceholder('Age').fill('40');
  await page.getByPlaceholder('Emergency Number').click();
  await page.getByPlaceholder('Emergency Number').fill('0956366363');
  await page.getByPlaceholder('abc@abc.com').click();
  await page.getByPlaceholder('abc@abc.com').fill('testdoctor@doc.com');
  await page.locator('select[name="gender"]').selectOption('Female');
  await page.locator('select[name="bloodGroup"]').selectOption('B+');
```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		53

```

await page.getByPlaceholder('dd-mm-yy').fill('1984-01-12');
await page.getByPlaceholder('Address').click();
await page.getByPlaceholder('Address').fill('Address');
await page.getByPlaceholder('eg.MBBS').click();
await page.getByPlaceholder('eg.MBBS').fill('Education');
await page.locator('select[name="department"]').selectOption('Cardiology');
await page.getByPlaceholder('Password').click();
await page.getByPlaceholder('Password').fill('testdoctor');
await page.getByPlaceholder('Extra Info').click();
await page.getByPlaceholder('Extra Info').fill('No information');
await page.getByRole('button', { name: 'Submit' }).click();
await page.locator('a').filter({ hasText: 'DashBoard' }).click();
});

```

На рисунку 3.20 зображено результат виконання тесту.

```

PS E:\STUDY\qualifi_work> npx playwright test test_doctor.spec.js --project=chromium

Running 1 test using 1 worker
  1 passed (5.3s)

To open last HTML report run:

  npx playwright show-report

```

Рис. 3.20. Результат виконання тесту додавання нового лікаря через інтерфейс адміністратора

Наступний тестовий сценарій розроблений для вебдодатку, який взаємодіє зі стороною лікаря. Спочатку тест виконує перехід на головну сторінку за вказаною URL. Далі виконується клік на «Doctor», щоб увійти до відповідного розділу вебдодатку. Після цього виконується заповнення текстового поля і введення даних. Потім натискається кнопка «Submit» для відправлення заповнених даних. Наступним кроком є перехід до профілю лікаря, натискання на кнопку «Edit profile» для внесення змін. Виконується клік на полі для введення мобільного номеру та введення нового значення. Після цього натискається кнопка «Edit» для збереження внесених змін. Завершення тесту полягає в кліку на посилання «Logout» для виходу з облікового запису лікаря.

```

import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await page.getByText('Doctor').click();

```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		54

```

await page.getByRole('spinbutton').click();
await page.getByRole('spinbutton').fill('1718967959388');
await page.getByRole('textbox').click();
await page.getByRole('textbox').fill('natalia');
await page.getByRole('button', { name: 'Submit' }).click();
await page.locator('a').filter({ hasText: 'Profile' }).click();
await page.getByRole('button', { name: 'Edit profile' }).click();
await page.getByPlaceholder('mobile').click();
await page.getByPlaceholder('mobile').fill('0958635212');
await page.getByRole('button', { name: 'Edit', exact: true }).click();
await page.locator('a').filter({ hasText: 'Logout' }).click();
});

```

На рисунку 3.21 зображено результат виконання тесту.

```

PS E:\STUDY\qualifi_work> npx playwright test test_profile.spec.js --project=chromium

Running 1 test using 1 worker
  1 passed (8.0s)

To open last HTML report run:

npx playwright show-report

```

Рис. 3.21. Результат виконання тесту редагування власного профілю лікаря

Наступний тестовий сценарій призначений для перевірки видалення лікаря. Спочатку тест відкриває головну сторінку додатку за вказаною URL та здійснює вхід як адміністратор. Наступним кроком є перехід до списку усіх лікарів, де адміністратор використовує комбінований список для вибору опції «email». Потім він вводить текст «testdoctor@doc.com» у поле для фільтрації за «email» і натискає клавішу «Enter» для застосування фільтру. Останнім етапом тесту є натискання на кнопку «Delete» для видалення лікаря з введеним «email». Після цього адміністратор виходить з облікового запису, натискаючи на посилання «Logout».

```

import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await page.getByText('Admin').click();
  await page.getByRole('spinbutton').click();
  await page.getByRole('spinbutton').fill('1718973036301');
  await page.getByRole('textbox').click();
  await page.getByRole('textbox').fill('oleksandr');
  await page.getByRole('button', { name: 'Submit' }).click();
  await page.locator('a').filter({ hasText: 'All Doctors' }).click();
});

```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		55

```

await page.getByRole('combobox').selectOption('email');
await page.getByPlaceholder('Filter by email').click();
await page.getByPlaceholder('Filter by email').fill('testdoctor@doc.com');
await page.getByPlaceholder('Filter by email').press('Enter');
await page.getByRole('button', { name: 'Delete' }).click();
await page.locator('a').filter({ hasText: 'Logout' }).click();
});

```

На рисунку 3.22 зображено результат виконання тесту.

```

PS E:\STUDY\qualifi_work> npx playwright test test_alldoctors.spec.js --project=chromium

Running 1 test using 1 worker
  1 passed (4.8s)

To open last HTML report run:

  npx playwright show-report

```

Рис. 3.22. Результат виконання тесту видалення лікаря

Далі переходимо до тестування функціоналу додавання записів на відвідування. Спочатку тест відкриває головну сторінку додатку за вказаною URL. Після цього виконується заповнення форми для входу. Наступним кроком є перехід до розділу «Appointments», де тест вводить дані пацієнта у форму для запису на прийом – введення імені, віку, вибір статі, номера телефону, електронної пошти, хвороби, адреси та вибір відділення. Після введення всіх необхідних даних, встановлюється дата та час прийому, використовуючи відповідні поля, та відбувається підтвердження запису на прийом.

```

import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await page.getByRole('spinbutton').click();
  await page.getByRole('spinbutton').fill('1718969991688');
  await page.getByRole('textbox').click();
  await page.getByRole('textbox').fill('maria');
  await page.getByRole('button', { name: 'Submit' }).click();
  await page.locator('a').filter({ hasText: 'Appointments' }).click();
  await page.getByPlaceholder('First Name').click();
  await page.getByPlaceholder('First Name').fill('Test Patient');
  await page.getByPlaceholder('Age').click();
  await page.getByPlaceholder('Age').fill('45');
  await page.locator('select[name="gender"]').selectOption('Female');
  await page.getByPlaceholder('Number').click();
});

```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		56

```

await page.getByPlaceholder('Number').fill('0963214587');
await page.getByPlaceholder('example@email.com').click();
await page.getByPlaceholder('example@email.com').fill('testpatient@mail.com');
await page.locator('select[name="disease"]').selectOption('Headaches');
await page.getByPlaceholder('Address line').click();
await page.getByPlaceholder('Address line').fill('Test Address');
await page.locator('select[name="department"]').selectOption('Neurology');
await page.getByPlaceholder('Choose Date').fill('2024-06-12');
await page.getByPlaceholder('Choose Time').click();
await page.getByPlaceholder('Choose Time').fill('12:00');
await page.getByRole('button', { name: 'Book Appointment' }).click();
await page.locator('a').filter({ hasText: 'Logout' }).click();
});

```

На рисунку 3.23 зображено результат виконання тесту.

```

PS E:\STUDY\qualifi_work> npx playwright test test_appointment.spec.js --project=chromium

Running 1 test using 1 worker
  1 passed (4.9s)

To open last HTML report run:

  npx playwright show-report

```

Рис. 3.23. Результат виконання тесту додавання запису на відвідування лікаря

Наступний тест перевіряє функціонал додавання звіту для лікаря. Виконується клік на «Doctor» для входу до відповідного розділу вебдодатку. Після цього виконується заповнення форми для входу. Далі відбувається перехід до розділу «Create Report», де тест заповнює форму для створення звіту лікаря. Після введення всіх необхідних даних, встановлюється дата та час створення звіту та відбувається підтвердження створення звіту за допомогою кнопки «Generate Report».

```

import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('http://localhost:3000/');
  await page.getByText('Doctor').click();
  await page.getByRole('spinbutton').click();
  await page.getByRole('spinbutton').fill('1719058258598');
  await page.getByRole('textbox').click();
  await page.getByRole('textbox').fill('olena');
  await page.getByRole('textbox').press('Enter');
  await page.locator('a').filter({ hasText: 'Create Report' }).click();
  await page.getByPlaceholder('Full Name').click();
});

```

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		57

```

await page.getByPlaceholder('Full Name').fill('Olena Ivanenko');
await page.locator('select[name="docDepartment"]').selectOption('Cardiology');
await page.getByPlaceholder('No').click();
await page.getByPlaceholder('No').fill('0963214569');
await page.getByPlaceholder('Name', { exact: true }).click();
await page.getByPlaceholder('Name', { exact: true }).fill('Test Patient');
await page.getByPlaceholder('Age').click();
await page.getByPlaceholder('Age').fill('29');
await page.getByPlaceholder('Mobile').click();
await page.getByPlaceholder('Mobile').fill('0852365252');
await page.getByPlaceholder('abc@abc').click();
await page.getByPlaceholder('abc@abc').fill('testpatient@mail.com');
await page.locator('select[name="patientGender"]').selectOption('Others');
await page.locator('select[name="patientBloodGroup"]').selectOption('A+');
await page.getByPlaceholder('Disease').click();
await page.getByPlaceholder('Disease').fill('headache');
await page.getByPlaceholder('^C').click();
await page.getByPlaceholder('^C').fill('38');
await page.getByPlaceholder('KG').click();
await page.getByPlaceholder('KG').fill('60');
await page.getByPlaceholder('/90 mmHg').click();
await page.getByPlaceholder('/90 mmHg').fill('130/80');
await page.getByPlaceholder('mg/dL').click();
await page.getByPlaceholder('mg/dL').fill('5');
await page.getByPlaceholder('Info').click();
await page.getByPlaceholder('Info').fill('No information');
await page.getByPlaceholder('dd-mm-yyyy').fill('2024-06-13');
await page.locator('input[name="time"]').click();
await page.locator('input[name="time"]').fill('10:50');
await page.getByRole('button', { name: 'Generate Report' }).click();
});

```

На рисунку 3.24 зображено результат виконання тесту.

```

PS E:\STUDY\qualifi_work> npx playwright test test_report.spec.js --project=chromium

Running 1 test using 1 worker
  1 passed (5.2s)

To open last HTML report run:

npx playwright show-report

```

Рис. 3.24. Результат виконання тесту додавання звіту

Висновки до третього розділу

У третьому розділі було розглянуто структуру інтерфейсу вебдодатку та порядок його роботи. Основні компоненти інтерфейсу включають сторінку входу,

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		58

головну сторінку, різні форми для додавання та редагування даних, а також сторінки з інформацією про лікарів, медичний персонал та інші аспекти медичного закладу. Детально описано функціонал для трьох ролей користувачів: адміністратора, лікаря та медичного персоналу.

Адміністратори можуть керувати інформацією про лікарів, медичний персонал, місця в палатах та медичний транспорт, а також переглядати та фільтрувати дані, що дозволяє ефективно організувати роботу медичного закладу. Лікарі можуть додавати звіти про пацієнтів, переглядати заплановані візити та редагувати власний профіль, що сприяє поліпшенню медичних послуг. Медичний персонал може додавати інформацію про пацієнтів та планувати їх візити, що допомагає забезпечити належний догляд за пацієнтами.

У підрозділі тестування було проведено функціональне тестування вебдодатку з використанням інструменту Playwright, який дозволяє автоматизувати перевірку різних сценаріїв роботи системи. Було протестовано основні функції, такі як додавання нового лікаря, редагування профілю лікаря, видалення лікаря, додавання записів на відвідування, а також створення звітів для лікаря. Результати тестування підтвердили, що вебдодаток відповідає функціональним вимогам, працює стабільно та надійно, що забезпечує високу якість кінцевого продукту перед його впровадженням у виробниче середовище.

Таким чином, структурний підхід до розробки та тестування вебдодатку для медичного закладу дозволив створити інтуїтивно зрозумілий, функціональний та надійний інструмент, який сприятиме ефективній роботі медичного персоналу та підвищенню якості медичних послуг.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		59

ВИСНОВКИ

Під час виконання кваліфікаційної роботи на тему «Вебдодаток для забезпечення роботи персоналу медичного закладу» було проведено детальний аналіз вимог та потреб користувачів, встановлено ключові функціональні вимоги, вибрано оптимальний технологічний стек та розроблено ефективну архітектуру вебдодатку. Результати дослідження показали, що обрані рішення відповідають поставленим цілям проекту, забезпечуючи високу якість, надійність та безпеку розробленого вебдодатку.

Основні отримані результати включають розробку інтуїтивно зрозумілого інтерфейсу для трьох категорій користувачів (адміністраторів, лікарів та медичного персоналу), реалізацію всіх основних функціональних вимог, таких як управління користувачами, медичними записами та ресурсами, а також забезпечення високої швидкодії та масштабованості системи.

Практична цінність розробленого вебдодатку полягає у забезпеченні ефективної організації роботи медичного закладу, підвищенні якості медичних послуг та зручності для користувачів. Використання сучасних технологій, таких як стек MERN, сприяє гнучкості та масштабованості вебдодатку, що важливо для його подальшого розвитку та адаптації до зміни потреб медичної галузі.

Можливі шляхи подальшого вдосконалення об'єкту дослідження включають розширення функціональності для підтримки нових потреб медичного персоналу, удосконалення інтерфейсу користувача з урахуванням зворотнього зв'язку від користувачів та інтеграцію додаткових модулів для підвищення зручності та продуктивності роботи.

Підсумовуючи роботу над кваліфікаційною роботою, можна констатувати успішність виконання всіх поставлених завдань. Розроблений вебдодаток відповідає всім ключовим вимогам, що підтверджується результатами функціонального тестування.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		60

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Information Technology in Healthcare: Applications and Role [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://www.fingent.com/blog/the-application-and-impact-of-information-technology-in-healthcare/>.
2. Впровадження інформаційних технологій у медицину [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: <https://medplatforma.com.ua/article/570-qqq-15-m9-03-09-2015-vprovadjennya-nformatsynih-tehnology-u-meditsinu>.
3. Медична інформаційна система Clinica Web [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.clinica-web.ua/>.
4. SIMPLEX MED [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://simplex-med.com/>.
5. MERN Stack Explained [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.mongodb.com/resources/languages/mern-stack>.
6. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage / Kristina Chodorow., 2013. – 430 с.
7. Express - фреймворк для веб-застосунків, побудованих на Node.js [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://expressjs.com/uk/>.
8. Learning React: Functional Web Development with React and Redux / Alex Banks., 2017. – 348 с.
9. Node.js - Run JavaScript Everywhere [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://nodejs.org/>.
10. Classification of Software Requirements – Software Engineering [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements>.

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		61

11. Use Case Diagrams | Unified Modeling Language (UML) [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/use-case-diagram/>.
12. What is Activity Diagram? [Електронний ресурс]. – 2023 – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>.
13. MongoDB Architecture [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mongodb-architecture/>.
14. What is Client-Server Architecture? Explained in Detail [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://www.theknowledgeacademy.com/blog/client-server-architecture/>.
15. Deployment Diagram in Unified Modeling Language(UML) [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/deployment-diagram-unified-modeling-languageuml/>.
16. Вступ до Mongoose для MongoDB та Node.js [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: <https://code.tutsplus.com/uk/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527a>.
17. Redux a JS library for predictable and maintainable global state management [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://redux.js.org/>.
18. Getting Started | Axios Docs [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://axios-http.com/uk/docs/intro>.
19. Функціональне тестування [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/>.
20. Playwright End To End Testing: Complete Guide [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: <https://luxequality.com/blog/playwright-end-to-end-testing/>.

ДОДАТКИ

		А.О. СТЕЛЬМАХ			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Маршрути для керування даними користувачів на основі Node.js та Express

```
const express = require("express");
const { DoctorModel } = require("../models/Doctor.model");
require("dotenv").config();
const jwt = require("jsonwebtoken");
const bcrypt = require('bcryptjs');
const router = express.Router();

router.get("/", async (req, res) => {
  let query = req.query;
  try {
    const doctors = await DoctorModel.find(query);
    res.status(200).send(doctors);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong" });
  }
});

router.post("/register", async (req, res) => {
  const { email } = req.body;
  try {
    const doctor = await DoctorModel.findOne({ email });
    if (doctor) {
      return res.send({
        message: "Doctor already exists",
      });
    }
    let value = new DoctorModel(req.body);
    await value.save();
    const data = await DoctorModel.findOne({ email });
    return res.send({ data, message: "Registered" });
  } catch (error) {
    res.send({ message: "error" });
  }
});

router.post("/login", async (req, res) => {
  const { docID, password } = req.body;
  try {
    const doctor = await DoctorModel.findOne({ docID });
    if (doctor) {
      const isMatch = await bcrypt.compare(password, doctor.password);

      if (isMatch) {
        const token = jwt.sign({ foo: "bar" }, process.env.key, {
          expiresIn: '24h',
        });
        res.send({ message: 'Successful', user: doctor, token });
      } else {
        res.status(401).send({ message: 'Wrong credentials' });
      }
    } else {
      res.status(401).send({ message: 'Wrong credentials' });
    }
  } catch (error) {
    console.log({ message: "Error" });
    console.log(error);
  }
});
```

```

router.patch("/:doctorId", async (req, res) => {
  const id = req.params.doctorId;
  const payload = req.body;
  try {
    await DoctorModel.findByIdAndUpdate({ _id: id }, payload);
    const doctor = await DoctorModel.findById(id);
    if (!doctor) {
      return res
        .status(404)
        .send({ message: `Doctor with id ${id} not found` });
    }
    res.status(200).send({ message: `Doctor Updated`, user: doctor });
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Update." });
  }
});

router.delete("/:doctorId", async (req, res) => {
  const id = req.params.doctorId;
  try {
    const doctor = await DoctorModel.findByIdAndDelete({ _id: id });
    if (!doctor) {
      res.status(404).send({ msg: `Doctor with id ${id} not found` });
    }
    res.status(200).send(`Doctor with id ${id} deleted`);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Delete." });
  }
});

module.exports = router;

```

```

const express = require("express");
const { NurseModel } = require("../models/Nurse.model");
require("dotenv").config();
const jwt = require("jsonwebtoken");
const bcrypt = require('bcryptjs');

const router = express.Router();

router.get("/", async (req, res) => {
  let query = req.query;
  try {
    const doctors = await NurseModel.find(query);
    res.status(200).send(doctors);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong" });
  }
});

router.post("/register", async (req, res) => {
  const { email } = req.body;
  try {
    const nurse = await NurseModel.findOne({ email });
    if (nurse) {
      return res.send({
        message: "Nurse already exists",

```

```

    });
  }
  let value = new NurseModel(req.body);
  await value.save();
  const data = await NurseModel.findOne({ email });
  return res.send({ data, message: "Registered" });
} catch (error) {
  res.send({ message: "error" });
}
});

router.post("/login", async (req, res) => {
  const { nurseID, password } = req.body;
  try {
    const nurse = await NurseModel.findOne({ nurseID });

    if (nurse) {
      const isMatch = await bcrypt.compare(password, nurse.password);

      if (isMatch) {
        const token = jwt.sign({ foo: "bar" }, process.env.key, {
          expiresIn: '24h',
        });
        res.send({ message: 'Successful', user: nurse, token });
      } else {
        res.status(401).send({ message: 'Wrong credentials' });
      }
    } else {
      res.send({ message: "Wrong credentials" });
    }
  } catch (error) {
    console.log({ message: "Error" });
    console.log(error);
  }
});

router.patch("/:nurseId", async (req, res) => {
  const id = req.params.nurseId;
  const payload = req.body;
  try {
    await NurseModel.findByIdAndUpdate({ _id: id }, payload);
    const nurse = await NurseModel.findById(id);
    if (!nurse) {
      return res.status(404).send({ message: `Nurse with id ${id} not found` });
    }
    res.status(200).send({ message: `Nurse Updated`, user: nurse });
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Update." });
  }
});

router.delete("/:nurseId", async (req, res) => {
  const id = req.params.nurseId;
  try {
    const nurse = await NurseModel.findByIdAndDelete({ _id: id });
    if (!nurse) {
      res.status(404).send({ msg: `Nurse with id ${id} not found` });
    }
    res.status(200).send(`Nurse with id ${id} deleted`);
  } catch (error) {
    console.log(error);
  }
});

```

```

    res.status(400).send({ error: "Something went wrong, unable to Delete." });
  }
});

module.exports = router;

```

```

const express = require("express");
const { AdminModel } = require("../models/Admin.model");
require("dotenv").config();
const jwt = require("jsonwebtoken");
const bcrypt = require('bcryptjs');
const nodemailer = require("nodemailer");

const router = express.Router();

router.get("/", async (req, res) => {
  try {
    const admins = await AdminModel.find();
    res.status(200).send(admins);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong" });
  }
});

router.post("/register", async (req, res) => {
  const { email } = req.body;
  try {
    const admin = await AdminModel.findOne({ email });
    if (admin) {
      return res.send({
        message: "Admin already exists",
      });
    }
    let value = new AdminModel(req.body);
    await value.save();
    const data = await AdminModel.findOne({ email });
    return res.send({ data, message: "Registered" });
  } catch (error) {
    res.send({ message: "error" });
  }
});

router.post("/login", async (req, res) => {
  const { adminID, password } = req.body;
  try {
    const admin = await AdminModel.findOne({ adminID });
    if (admin) {
      const isMatch = await bcrypt.compare(password, admin.password);

      if (isMatch) {
        const token = jwt.sign({ foo: "bar" }, process.env.key, {
          expiresIn: '24h',
        });
        res.send({ message: 'Successful', user: admin, token });
      } else {
        res.status(401).send({ message: 'Wrong credentials' });
      }
    } else {
      res.send({ message: "Wrong credentials" });
    }
  } catch (error) {

```

```

    console.log({ message: "Error" });
    console.log(error);
    res.status(500).send({ message: "Server error" });
  }
});

router.patch("/:adminId", async (req, res) => {
  const id = req.params.adminId;
  const payload = req.body;
  try {
    const admin = await AdminModel.findByIdAndUpdate({ _id: id }, payload);
    if (!admin) {
      res.status(404).send({ msg: `Admin with id ${id} not found` });
    }
    res.status(200).send(`Admin with id ${id} updated`);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Update." });
  }
});

router.delete("/:adminId", async (req, res) => {
  const id = req.params.adminId;
  try {
    const admin = await AdminModel.findByIdAndDelete({ _id: id });
    if (!admin) {
      res.status(404).send({ msg: `Admin with id ${id} not found` });
    }
    res.status(200).send(`Admin with id ${id} deleted`);
  } catch (error) {
    console.log(error);
    res.status(400).send({ error: "Something went wrong, unable to Delete." });
  }
});

```

Файли структури «Redux» для керування аутентифікацією користувачів та даними
у вебдодатку

```
import * as types from "../types";
import axios from "axios";

export const NurseLogin = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.LOGIN_NURSE_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/nurses/login",
      data
    );
    dispatch({
      type: types.LOGIN_NURSE_SUCCESS,
      payload: {
        message: res.data.message,
        user: res.data.user,
        token: res.data.token,
      },
    });
    return res.data;
  } catch (error) {
    dispatch({
      type: types.LOGIN_NURSE_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const DoctorLogin = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.LOGIN_DOCTOR_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/doctors/login",
      data
    );
    console.log(res.data);
    dispatch({
      type: types.LOGIN_DOCTOR_SUCCESS,
      payload: {
        message: res.data.message,
        user: res.data.user,
        token: res.data.token,
      },
    });
    return res.data;
  } catch (error) {
    dispatch({
      type: types.LOGIN_DOCTOR_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const AdminLogin = (data) => async (dispatch) => {
```

```

    try {
      dispatch({ type: types.LOGIN_ADMIN_REQUEST });
      const res = await axios.post(
        "http://localhost:5000/admin/login",
        data
      );
      console.log(res.data);
      dispatch({
        type: types.LOGIN_ADMIN_SUCCESS,
        payload: {
          message: res.data.message,
          user: res.data.user,
          token: res.data.token,
        },
      });
      return res.data;
    } catch (error) {
      dispatch({
        type: types.LOGIN_ADMIN_ERROR,
        payload: {
          message: error,
        },
      });
    }
  }
};

export const DoctorRegister = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.REGISTER_DOCTOR_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/doctors/register",
      data
    );

    return res.data;

  } catch (error) {
    dispatch({
      type: types.REGISTER_DOCTOR_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const NurseRegister = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.REGISTER_NURSE_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/nurses/register",
      data
    );

    return res.data;

  } catch (error) {
    dispatch({
      type: types.REGISTER_NURSE_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

```

```

};

export const AdminRegister = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.REGISTER_ADMIN_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/admin/register",
      data
    );

    return res.data;
  } catch (error) {
    dispatch({
      type: types.REGISTER_ADMIN_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const AmbulanceRegister = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.REGISTER_AMBULANCE_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/ambulances/add",
      data
    );
    console.log(res);
  } catch (error) {
    dispatch({
      type: types.REGISTER_AMBULANCE_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const UpdateNurse = (data, id) => async (dispatch) => {
  try {
    dispatch({ type: types.EDIT_NURSE_REQUEST });
    const res = await axios.patch(
      `http://localhost:5000/nurses/${id}`,
      data
    );
    console.log(res);
    dispatch({ type: types.EDIT_NURSE_SUCCESS, payload: res.data.user });
  } catch (error) {
    console.log(error);
  }
};

export const UpdateDoctor = (data, id) => async (dispatch) => {
  try {
    dispatch({ type: types.EDIT_DOCTOR_REQUEST });
    const res = await axios.patch(
      `http://localhost:5000/doctors/${id}`,
      data
    );
  }
};

```

```

    console.log(res);
    dispatch({ type: types.EDIT_DOCTOR_SUCCESS, payload: res.data.user });
  } catch (error) {
    console.log(error);
  }
};

```

```

import * as types from "../types";
const TOKEN = localStorage.getItem("token");
const initialState = {
  userLogin: { loading: false, error: false, message: "" },
  userLogout: { message: "" },
  data: {
    isAuthenticated: !!TOKEN,
    token: TOKEN,
    user: null,
  },
};
export default function authReducer(state = initialState, { type, payload }) {
  switch (type) {
    case types.LOGIN_NURSE_REQUEST ||
      types.LOGIN_ADMIN_REQUEST ||
      types.LOGIN_DOCTOR_REQUEST:
      return {
        ...state,
        userLogin: { loading: true, error: false },
      };
    case types.LOGIN_NURSE_SUCCESS ||
      types.LOGIN_ADMIN_SUCCESS ||
      types.LOGIN_DOCTOR_SUCCESS:
      localStorage.setItem("token", payload.token);
      return {
        ...state,
        userLogin: { loading: false, error: false, message: payload.message },
        data: {
          isAuthenticated: payload.token ? true : false,
          token: payload.token,
          user: payload.user,
        },
      };
    case types.LOGIN_NURSE_ERROR ||
      types.LOGIN_ADMIN_ERROR ||
      types.LOGIN_DOCTOR_ERROR:
      return {
        ...state,
        userLogin: { loading: false, error: true, message: payload.message },
      };
    case types.LOGIN_DOCTOR_REQUEST:
      return {
        ...state,
        userLogin: { loading: true, error: false },
      };
    case types.LOGIN_DOCTOR_SUCCESS:
      localStorage.setItem("token", payload.token);
      return {
        ...state,
        userLogin: { loading: false, error: false, message: payload.message },
        data: {
          isAuthenticated: payload.token ? true : false,
          token: payload.token,
          user: payload.user,
        },
      };
  }
}

```

```

    };
    case types.LOGIN_DOCTOR_ERROR:
    return {
        ...state,
        userLogin: { loading: false, error: true, message: payload.message },
    };
    case types.LOGIN_ADMIN_REQUEST:
    return {
        ...state,
        userLogin: { loading: true, error: false },
    };
    case types.LOGIN_ADMIN_SUCCESS:
    localStorage.setItem("token", payload.token);
    return {
        ...state,
        userLogin: { loading: false, error: false, message: payload.message },
        data: {
            isAuthenticated: payload.token ? true : false,
            token: payload.token,
            user: payload.user,
        },
    };
    case types.EDIT_NURSE_SUCCESS:
    return {
        ...state,
        data: {
            user: payload,
        },
    };
    case types.EDIT_DOCTOR_SUCCESS:
    return {
        ...state,
        data: {
            user: payload,
        },
    };
    case types.LOGIN_ADMIN_ERROR:
    return {
        ...state,
        userLogin: { loading: false, error: true, message: payload.message },
    };

    case "AUTH_LOGIN_RESET":
    return {
        ...state,
        userLogin: { loading: false, error: false, message: "" },
    };
    case types.AUTH_LOGOUT:
    localStorage.removeItem("token");
    return {
        ...state,
        userLogin: { loading: false, error: false, message: "" },
        userLogout: { message: "Logout Successfully" },
        data: {
            isAuthenticated: false,
            token: null,
            user: null,
        },
    };
    default:
    return state;
}
}

```

```

import * as types from "../types";
import axios from "axios";

export const CreateReport = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.CREATE_REPORT_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/reports/create",
      data
    );
    console.log(res);
    return res.data;
  } catch (error) {
    dispatch({
      type: types.CREATE_REPORT_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const GetDoctorDetails = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_DOCTOR_REQUEST });
    const res = await axios.get(
      "http://localhost:5000/doctors"
    );
    console.log(res);
  } catch (error) {
    dispatch({
      type: types.GET_DOCTOR_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const AddPatients = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.ADD_PATIENT_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/patients/register",
      data
    );
    return res.data;
  } catch (error) {
    dispatch({
      type: types.ADD_PATIENT_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const CreateBeds = (data) => async (dispatch) => {
  try {

```

```

    dispatch({ type: types.ADD_BED_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/beds/add",
      data
    );
    return res.data;
  } catch (error) {
    dispatch({
      type: types.ADD_BED_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const GetBeds = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_BED_REQUEST });
    const res = await axios.get("http://localhost:5000/beds");
    console.log(res);
    dispatch({
      type: types.GET_BED_SUCCESS,
      payload: res.data,
    });
  } catch (error) {
    dispatch({
      type: types.GET_BED_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const CreateBooking = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.CREATE_BOOKING_REQUEST });
    const res = await axios.post(
      `http://localhost:5000/appointments/create`,
      data
    );
    return res.data;
  } catch (error) {
    dispatch({
      type: types.CREATE_BOOKING_ERROR,
      payload: {
        message: error,
      },
    });
  }
};

export const AddBed = (data) => async (dispatch) => {
  try {
    dispatch({ type: types.ADD_BEDS_REQUEST });
    const res = await axios.post(
      "http://localhost:5000/beds/add",
      data
    );
    console.log(res);
  }
};

```

```

        return res.data;
    } catch (error) {
        dispatch({
            type: types.ADD_BEDS_ERROR,
            payload: {
                message: error,
            },
        });
    }
};

export const GetSingleBed = (data) => async (dispatch) => {
    try {
        dispatch({ type: types.GET_SINGLE_BEDS_REQUEST });
        const res = await axios.post(
            "http://localhost:5000/beds/single",
            data
        );

        return res.data;

    } catch (error) {

        console.log(error);
    }
};

export const EditSingleBed = (data, id) => async (dispatch) => {
    try {
        dispatch({ type: types.GET_SINGLE_BEDS_REQUEST });
        const res = await axios.patch(
            `http://localhost:5000/beds/${id}`,
            data
        );

        return res.data;

    } catch (error) {

        console.log(error);
    }
};

export const dischargePatient = (data) => async (dispatch) => {
    try {
        dispatch({ type: types.DISCHARGE_PATIENT_REQUEST });
        const res = await axios.put(
            `http://localhost:5000/beds/discharge`,
            data
        );
        console.log(res);

        dispatch({
            type: types.DISCHARGE_PATIENT_SUCCESS,
            payload: {
                bed: res.data.bed,
            },
        });
    } catch (error) {
        console.log(error);
    }
};

```

```
export const GetPatients = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_PATIENT_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/patients`
    );
    console.log(res.data);
    dispatch({
      type: types.GET_PATIENT_SUCCESS,
      payload: res.data,
    });
  } catch (error) {
    console.log(error);
  }
};

export const GetAllData = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_ALLDATA_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/hospitals`
    );
    console.log(res.data);
    dispatch({
      type: types.GET_ALLDATA_SUCCESS,
      payload: res.data,
    });
  } catch (error) {
    console.log(error);
  }
};

export const GetAllAppointment = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_APPOINTMENT_DETAILS_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/appointments`
    );

    dispatch({
      type: types.GET_APPOINTMENT_DETAILS_SUCCESS,
      payload: res.data,
    });
  } catch (error) {
    console.log(error);
  }
};

export const DeleteAppointment = (id) => async (dispatch) => {
  try {
    dispatch({ type: types.DELETE_APPOINTMENT_REQUEST });
    const res = await axios.delete(
      `http://localhost:5000/appointments/${id}`
    );
    console.log(res.data);
    dispatch({
      type: types.DELETE_APPOINTMENT_SUCCESS,
      payload: id,
    });
  } catch (error) {
    console.log(error);
  }
};
```

```
export const GetAllReports = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_REPORTS_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/reports`
    );

    return res.data;

  } catch (error) {
    console.log(error);
  }
};

export const GetAllDoctors = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_DOCTORS_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/doctors`
    );

    return res.data;

  } catch (error) {
    console.log(error);
  }
};

export const DeleteDoctor = (id) => async (dispatch) => {
  try {
    await axios.delete(`http://localhost:5000/doctors/${id}`);
    dispatch({ type: "DELETE_DOCTORS_REQUEST", payload: id });
  } catch (error) {
    console.error("Error deleting doctor:", error);
  }
};

export const GetAllNurses = () => async (dispatch) => {
  try {
    dispatch({ type: types.GET_NURSES_REQUEST });
    const res = await axios.get(
      `http://localhost:5000/nurses`
    );

    return res.data;

  } catch (error) {
    console.log(error);
  }
};

export const DeleteNurse = (id) => async (dispatch) => {
  try {
    await axios.delete(`http://localhost:5000/nurses/${id}`);
    dispatch({ type: "DELETE_NURSES_REQUEST", payload: id });
  } catch (error) {
    console.error("Error deleting nurse:", error);
  }
};
```

```
import * as types from "../types";

const initialState = {
  loading: false,
  error: false,
  reports: [],
  beds: [],
  doctors: [],
  patients: [],
  nurses: [],
  dashboard: [],
  Appointments: [],
};

export default function dataReducer(state = initialState, { type, payload }) {
  switch (type) {
    case types.GET_BED_REQUEST:
      return {
        ...state,
        loading: true,
      };
    case types.GET_BED_SUCCESS:
      return {
        ...state,
        loading: false,
        beds: payload,
      };
    case types.GET_PATIENT_SUCCESS:
      return {
        ...state,
        loading: false,
        patients: payload,
      };
    case types.GET_ALldata_SUCCESS:
      return {
        ...state,
        loading: false,
        dashboard: payload,
      };
    case types.DISCHARGE_PATIENT_SUCCESS:
      let data = state.beds.map((ele) => {
        if (ele._id === payload.bed._id) {
          return payload.bed;
        }
        return ele;
      });
    case types.DELETE_APPOINTMENT_SUCCESS:
      return {
        ...state,
        loading: false,
        Appointments: state.Appointments.filter((ele) => ele._id !== payload),
      };
    case types.DELETE_DOCTORS_SUCCESS:
      return {
        ...state,
        doctors: state.doctors.filter((doctor) => doctor._id !== payload),
      };
    case types.DELETE_NURSES_SUCCESS:
      return {
        ...state,
        nurses: state.nurses.filter((nurse) => nurse._id !== payload),
      };
    case types.GET_APPOINTMENT_DETAILS_SUCCESS:
```

```
    return {
      ...state,
      loading: false,
      Appointments: payload,
    };

    default:
      return state;
  }
}
```