

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

Факультет комп'ютерно-інформаційних систем та програмної
інженерії

(повна назва факультету)

Кафедра програмної інженерії

(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр

(назва освітнього ступеня)

на тему: Розробка веб-сервісу медичного закладу з використанням
технології React JS

Виконав: студент _____ 4 _____ курсу, групи СП-41
спеціальності _____ 121 «Інженерія програмного
забезпечення»

(шифр і назва спеціальності)

(підпис) Москалик В.І.
(прізвище та ініціали)

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

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

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

Рецензент _____
(підпис) Жаровський Р.О.
(прізвище та ініціали)

РЕФЕРАТ

Дана кваліфікаційна робота призначена для здобуття ступеня бакалавра студента кафедри програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.

Тема: Розробка веб-сервісу медичного закладу з використанням технології React JS.

Робота містить: 43 рисунка та 4 таблиць.

Головною метою цієї кваліфікаційної роботи є розробка веб-сервісу для медичного закладу, де отримання медичної інформації буде швидке та зручне. Цей додаток полегшить роботу медичного персоналу та покращить обслуговування пацієнтів. Також, дасть можливість відвідувачам лікарні в будь-який момент доби отримати швидкий доступ до результатів досліджень та медичної інформації. Така функція дасть можливість покращити якість комунікації між лікарями та пацієнтами.

Розроблено клієнтську та серверну частину використовуючи стек технологій MERN.

Перелік ключових слів: лікар, сервер, клієнт, пацієнт, база даних, припис, діагноз діаграма класів, діаграма послідовності.

ANNOTATION

This qualification work is intended for obtaining a bachelor's degree of a student of the Software Engineering of Ternopil Ivan Puluj National Technical University.

The topic: Development of a web service for a medical institution using React JS technology.

The work contains: 43 pictures and 4 tables.

The main purpose of this qualification work is to develop a web service for a medical organization, where obtaining medical information will be quick and convenient. This application will facilitate the work of medical staff and improve patient care. It will also allow hospital visitors to get quick access to research results and medical information at any time of the day. This feature will improve the quality of communication between doctors and patients.

The client and server parts were developed using the MERN technology stack.

Keywords: doctor, server, client, patient, database, prescription, diagnosis, class diagram, sequence diagram.

ЗМІСТ

РЕФЕРАТ.....	4
ANNOTATION	5
ВСТУП.....	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Актуальність проблеми	10
1.2 Опис предметної області	11
1.3 Аналіз вимог до системи	12
1.4 Аналіз конкурентів	13
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	17
2.1 Проектування відношень між акторами та прецедентами	17
2.2 Опис взаємодії системи	20
3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	28
3.1 Вибір основних технологій	28
3.2 Розробка серверної частини додатку	32
3.3 Розробка клієнтської частини	36
3.3.1 Архітектура клієнта	36
3.3.2 Компоненти.....	38
3.3.3 Маршрутизація	39
3.4 Створення бази даних.....	40
4 ТЕСТУВАННЯ ТА ІНТЕРФЕЙС	43
4.1 Перевірка роботи системи.....	43
5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОХОРОНА ПРАЦІ	51

5.1	Долікарська допомога при ураженні електричним струмом.....	51
5.2	Вимоги безпеки до робочих місць користувачів ПК у медичних закладах..	53
ВИСНОВКИ		55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ		56
ДОДАТКИ		58
ДОДАТОК А		59

ВСТУП

В наш час дуже важко уявити життя людини без використання інформаційних технологій та інтернету – це свідчить про те, що бізнес розширює межі кожного дня і збільшує кількість своїх користувачів. Всі люди користуються інтернетом за допомогою різноманітних електронних пристроїв задовольняючи свої потреби застосовуючи веб додатки.

Розробка веб-сервісу для медичного закладу є дуже важливим та перспективним завданням враховуючи умови розвитку медичної сфери та інформаційних технологій. В даний час медичні заклади мають мати точний, кругло добовий та швидкий доступ до персональної інформації хворих для поліпшення їх лікування і діагностики. Данна розробка є перспективною та дозволить вирішувати ці завдання та оптимізувати їх роботу.

Метою даної дипломної роботи є розробка веб-сервісу для медичного закладу, де отримання медичної інформації буде швидке та зручне. Цей додаток полегшить роботу медичного персоналу та покращить обслуговування пацієнтів. Також, дасть можливість відвідувачам лікарні в любий момент доби отримати швидкий доступ до результатів досліджень та медичної інформації. Така функція дасть можливість покращити якість комунікації між лікарями та пацієнтами.

В сучасному світі забезпечення конфіденційності та безпеки даних користувачів є дуже важливим завданням. В галузі медицини, де зберігається велика кількість особистих та медичних даних пацієнтів – безпека даних відіграє важливу роль. Розробка веб-сервісу забезпечить безпечне та надійне збереження та оброблення даних.

Для досягнення даної мети потрібно виконати такі задачі:

- Провести аналіз медичних закладів та існуючих веб-сервісів і виявити недоліки та переваги;
- Розробити проєкт архітектури веб-сервісу для медичного закладу;

- Розробити ПЗ для веб-сервера, який забезпечить конфіденційність та безпеку даних пацієнтів;
- Провести тестування розробленого веб-сервісу;
- Оцінити ефективність додатку на зручність, продуктивність та якість надання послуг.

Дана робота може бути корисною для приватних та державних лікарень, кабінетів стоматології та всіх медичних закладів, які бажають покращити послуги та комунікацію між медичним персоналом та їх клієнтами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Розробка веб-сервісу для медичних закладів є актуальним сьогодні.

По-перше, цей онлайн сервіс дасть можливість швидко та зручно отримувати доступ до медичної інформації пацієнтів. Пацієнти зможуть легко відшукати інформацію про лікарів, послуги, розклад прийому та іншу корисну інформацію про медичний заклад.

По-друге, створення онлайн сервісу для лікарні оптимізує його роботу та покращить якість медичних послуг. Це веб-сервіс дасть можливість зменшити час на запис на прийом. Це дозволить працівникам медичних закладів більше часу приділяти прямій роботі з хворими.

По-третє, створення онлайн сервісу є вигідним з фінансової точки зору. Веб-сервіс дозволить зменшити фінансові витрати на обробку та зберігання паперової документації та зменшити навантаження на адміністративний персонал закладу.

По-четверте, створення онлайн сервісу для медичного закладу є важливим кроком у розвитку онлайн медицини. Він дозволить зменшити кількість помилок, які пов'язані з ручним введенням інформації, покращить моніторинг здоров'я пацієнтів та дасть можливість виконувати швидшу та точнішу діагностику.

Отже, створення веб-сервісу для медичного закладу є актуальним та потрібним кроком для розвитку медичної сфери в Україні. Цей веб-сервіс поліпшить роботу персоналу медичного закладу оптимізує їхню роботу, а також дасть можливість пацієнтам отримувати інформацію про стан здоров'я та інформацію щодо лікування.

1.2 Опис предметної області

Медичні заклади кожного дня обслуговують велику кількість людей. Вони зберігають та обробляють великі обсяги персональної інформації пацієнтів, яка стосується стану їхнього здоров'я, видають різні довідки та лікарняні. Приклади інформації, яка може зберігатися в базах даних медичних установ:

- Інформація про лікарів;
- Інформація про запис на прийом;
- Інформація про пацієнтів (медична карточка);
- Розклад роботи лікарів.

Може містити інформації про прийоми до лікарів:

- Ім'я лікаря;
- Дата та час;
- Пацієнт;
- Причина звернення;
- Діагноз;
- Лікування.

Також зберігається інформація про пацієнтів в медичних карточках. Вони мають індивідуальні номери та в них зберігається та інформація:

- Прізвище;
- Ім'я;
- По батькові;
- Вік;
- Стать;
- Домашня адреса;
- Страховий поліс.

1.3 Аналіз вимог до системи

Головним завданням розробленого веб-сервісу буде обробка інформації про пацієнтів та працівників медичних закладів. Буде реалізовано два варіанти використання системи. Перший – це використання зі сторони працівника медичного закладу та другий – зі сторони пацієнта.

Функціональні вимоги, які буде реалізовано зі сторони працівника лікарні:

- Реєстрації нового користувача системи;
- Вхід в систему використовуючи логін та пароль;
- Додавання нового пацієнта;
- Видалення пацієнта з бази даних;
- Призначення діагнозу пацієнта;
- Перегляд інформації про пацієнта;
- Пошуку пацієнта за фамілією;
- Виведення інформації про всіх пацієнтів.

Функціональні вимоги, які буде реалізовано зі сторони пацієнта:

- Реєстрація;
- Запис на прийом до лікаря;
- Перегляду інформації про всіх лікарів;
- Перегляду медичної картки;

Не функціональні вимоги:

- Веб-сервіс повинен володіти зручним, доступним для розуміння будь-якому колу користувачів інтерфейсом;
- Система повинна підтримувати велику кількість одночасно працюючих користувачів;
- Час відгуку повинен не перевищувати 1 секунду;
- Персональна інформація має бути захищеною;

- Збереження персональних даних має відбуватися у зашифрованому вигляді.

Вся інформація має зберігатися в базі даних MongoDB. БД буде використовуватися, як основне джерело інформації для коректної роботи системи. Інформація вноситься модифікується та видаляється користувачами системи.

1.4 Аналіз конкурентів

Головним конкурентом для веб-сервісу, який буде розроблено у даній дипломній роботі є Helsi.me. Цей сервіс покращує якість медичного обслуговування в Україні. Надає можливість записувати на прийом до лікаря та замовляти онлайн послуги. Helsi.me допомагає користувачам швидко знайти лікаря за географічним розташуванням, рейтингом або спеціалізацією [1]. На рисунку 1.1 зображено логотип компанії, яка є головним конкурентом на українському ринку.



Рисунок 1.4.1 – Логотип Helsi.me

Також цей веб-сервіс дає можливість з допомогою відео зв'язку або чату отримувати медичні поради або записувати на онлайн консультації.

Клієнти Helsi зберігають свої медичні дані та результати аналізи в цифровому вигляду в базах даних компанії.

Головний принцип Helsi.me забезпечувати максимальну конфіденційність особистої інформації клієнтів. Для цього використовується сучасні способи захисту даних та технологій для шифрування.

Даний веб-сервіс є корисним і популярним, але має недоліки:

1. Кількість медичних закладів та лікарів, які сьогодні співпрацюють з Helsi є обмеженою. Даний недолік обмежує вибір найкращого спеціаліста для потенційних клієнтів.
2. На даний момент інтерфейс сервісу тільки на українській мові, а це означає, що мовний бар'єр може стати перешкодою для іноземних користувачів.
3. Не доступна опція оплати послуг онлайн. Оплачувати послуги потрібно під час відвідування медичного закладу. Це знижує швидкість та зручність.
4. Електронна реєстрація є недоступною. Потенційні пацієнти реєструються в медичному закладі вручну. Це може привести до помилок у реєстрації через людський фактор.
5. Конфіденційність особистої інформації може бути втраченою через несанкціонованого доступу та порушення прав на приватність та безпеку особистих даних користувачів Helsi.

На рисунку 1.2 зображено вигляд запланованого візиту до лікаря.

The screenshot displays the 'Ваші заплановані візити' (Your scheduled visits) section. At the top, there are filters for 'Від: 01.12.2016' (From) and 'До: 05.10.2017' (To), along with a dropdown menu set to 'Всі візити' (All visits). Below this, a card for a specific appointment is shown. It features a profile picture of a woman, her name 'Савіна Жанна Григорівна' (Savina Zhanna Grigoriivna), and her title 'Сімейний лікар' (Family doctor). The appointment details include the date 'вівторок, 26-го вересня 2017' (Tuesday, September 26, 2017), the time '09:00 - 09:15', the address 'місто Київ вул. Північна 4 А' (Kyiv city, Pivnichna 4 A street), and the room 'Кабінет: 1' (Cabinet: 1). A red button labeled 'СКАСУВАТИ ПРИЙОМ' (Cancel reception) is located on the right side of the card.

Рисунок 1.4.2 – Запланований візит в Helsi

Також надає такі послуги:

- Пошук лікаря для укладення декларації;
- Пошук та бронювання ліків;
- Пошук медичного закладу поблизу;
- Запис на донорство крові;
- Пошук пункту вакцинації;
- Отримання е-рецепту.

Отже, Helsi є найбільшим конкурентом в розвитку онлайн медицини в Україні. Цей сервіс надає якісні медичні послуги та економить час та кошти лікарів та пацієнтів використовуючи сучасні технології.

Другим найбільш популярним сервісом в Україні є Doc.ua. На рисунку 1.3 зображено логотип даного сервісу. Окрім пошуку медичних закладів, лікарів та лабораторій для здавання аналізів веб-сервіс пропонує онлайн-консультації з лікарями [2].



Рисунок 1.4.3 – Логотип веб-сервісу

Даний веб-сервіс має такі переваги:

1. Доступність та зручність. Надає можливість знайти лікаря, ознайомитися з розкладом його роботи, записатися на прийом в будь-який час без необхідності чекати в черзі або телефонувати.
2. Великий вибір спеціалістів та лікарів, більше 20 000 працівників різних спеціальностей – це дозволяє знайти найкращого фахівця.

3. Допомогає знайти багато корисної інформації про медицину та здоров'я. Наприклад поради лікарів, статті, які допомагають покращити здоров'я.
4. Мобільність. Додаток можна завантажити на мобільний телефон. Це дає можливість пацієнтам зручно та швидко записатися на прийом з будь-якого місця та в будь-який час.
5. Гарантія захисту персональної інформації клієнтів та забезпечення повної конфіденційності.
6. Швидкість. Зв'язатися з лікарем можна за 15 хвилин.
7. Вартість послуг залежить від кваліфікації спеціаліста. Можна отримати кваліфіковану допомогу від 200 гривень.

З недоліків можна зазначити відсутність можливості онлайн оплати.

Дані переваги приваблюють тих, хто шукає зручний веб-сервіс для отримання медичних послуг.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проектування відношень між акторами та прецедентами

З даним веб-сервісом будуть взаємодіяти два актори це медичний працівник та пацієнт. Для створення UML-діаграм використано сучасний інструмент для проектування програмних продуктів Rational Software Architect. На рисунку 2.2.1 зображено цих акторів.



Рисунок 2.2.1 – Актори веб-сервісу

Для медичного працівника можна виділити варіанти використання, які наведені в таблиці 2.2.1

Таблиця 2.2.1 – Варіанти використання мед. працівника

Актор	Назва дії	Опис дії
Мед. працівник	Реєстрація в системі	Для реєстрації в системі потрібно перейти в розділ створити користувача та вписати свій логін та пароль.
Мед. працівник	Вхід в систему	Для входу в систему потрібно ввести свій особистий логін та пароль.

Продовження таблиці 2.2.1 – Варіанти використання мед. працівника

Мед. працівник	Запис нового пацієнта	Для додавання нового пацієнта потрібно перейти в розділ «Додати пацієнта» та вписати особисті дані.
Мед. працівник	Видалення пацієнта	Для видалення пацієнта з системи потрібно натиснути відповідну кнопку «Видалити»
Мед. працівник	Пошук	Для пошуку інформації про певного пацієнта потрібно ввести його Фамілію та натиснути кнопку пошуку.
Мед. працівник	Всі пацієнти	Для виведення списку всіх пацієнтів, які перебувають на лікуванні потрібно натиснути кнопку «Весь список».
Мед. працівник	Додавання діагнозу	Для додавання до медичної картки пацієнта діагнозу потрібно перейти в розділ «Мед. Картка» та ввести Фамілію пацієнта та його діагноз та натиснути кнопку «Ок».
Мед. працівник	Перегляд інформації про клієнта	Для перегляду інформації про стан здоров'я клієнта потрібно перейти в медичну карту, а для цього в списку пацієнтів потрібно натиснути на бажаного пацієнта.
Мед. працівник	Перегляд записів	Для перегляду записів пацієнтів на прийом лікарю потрібно натиснути кнопку «Записи».

Для клієнта можна виділити варіанти використання, які наведені в таблиця 2.2.2.

Таблиця 2.2.2 – Варіанти використання для пацієнта

Пацієнт	Реєстрація	Для реєстрації клієнту потрібно на сайті перейти в розділ «Реєстрація» і ввести фамілію ім'я та вік.
Пацієнт	Запис на прийом до лікаря	Для того, щоб пацієнт записався на прийом до лікаря потрібно вибрати фахівця і натиснути кнопку «записатися на прийом».
Пацієнт	Список лікарів	Для того, щоб отримати весь список лікарів, потрібно перейти в розділ «запис на прийом» і натиснути «Список лікарів».
Пацієнт	Перегляд мед. картки	Для перегляду медичної картки потрібно на головному екрані натиснути кнопку «Мед картка».

Використовуючи дані з таблиць варіантів використання, для кращого сприйняття та розуміння головних функцій, які буде виконувати система, побудовано use case діаграми. На рисунку 2.2.1 зображено варіант використання для пацієнта.

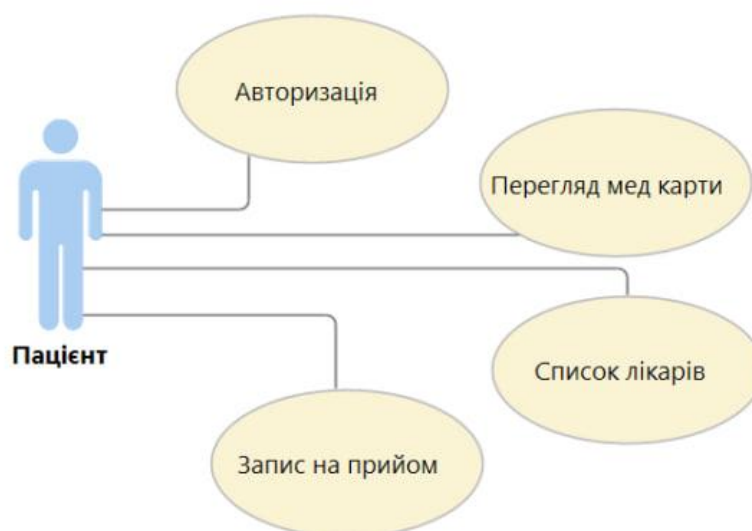


Рисунок 2.2.1 – Варіанти використання для пацієнта

Також побудовано use case діаграму для працівника медичного закладу її зображено на рисунку 2.2.2.



Рисунок 2.2.2 – Варіанти використання для мед. працівника

Створені таблиці та діаграми допоможуть відобразити послідовність дій між акторами та системою.

2.2 Опис взаємодії системи

Для проектування веб-сервісу вибрано архітектуру клієнт-сервер. При використанні цього підходу ПЗ розділено на дві частини: клієнт та сервер. На рисунку 2.3.1 зображено дану модель.

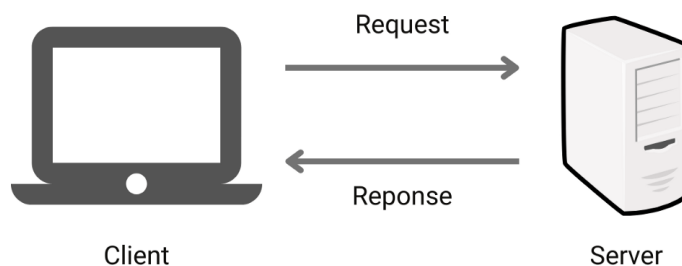


Рисунок 2.3.1 – Модель клієнт-сервер

Клієнт – на цій частині реалізовується взаємодія з користувачем та відображаються дані.

Сервер – на цій частині реалізовується обробка даних та взаємодія з базою даних.

Дана модель розділяє функції програми на дві частини, які працюють паралельно. Клієнт може знаходитися на декількох пристроях та виконувати запити до сервера одночасно. Це дає можливість масштабувати та ефективно працювати з даними [3].

Архітектура клієнт-сервер має такі переваги:

- Розділення функцій на дві паралельні частини, що забезпечує збільшення ефективності;
- Витрати на обладнання зменшуються через те, що клієнтська частина може бути реалізована на дешевих пристроях;
- Збільшення безпеки та конфіденційності особистих даних користувачів, бо сервер може виконувати авторизацію та контроль доступу до даних.

Серед недоліків можна виділити:

- Постійне підключення до сервера;
- Навантаження на сервер збільшується разом із збільшенням кількості клієнтів;
- Не можливо редагувати дані в режимі офлайн.

Для реалізації сервера створено класи, які описують головні сутності майбутньої системи. До них відносять користувач, медична картка, консультація. На рисунку 2.3.2 зображено розроблену діаграму класів.

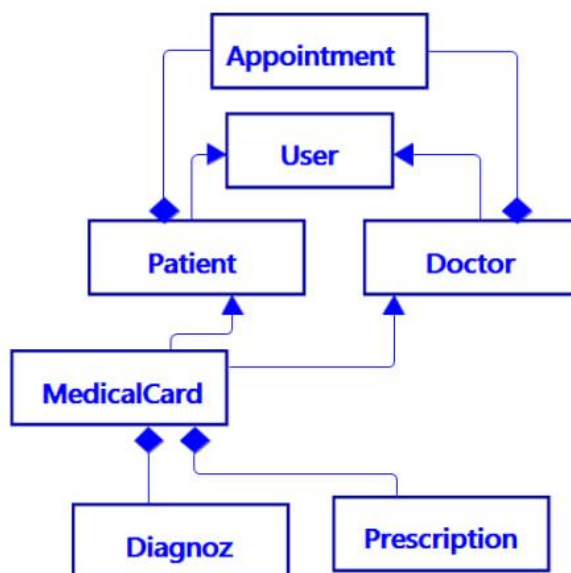


Рисунок 2.3.2 – Діаграма класів

Дана діаграма складається з семи класів, які описують конкретну сутність. Можна замітити, що присутнє наслідування та композиція. Клас User описує користувача системи. Містить такі атрибути: password (пароль), typeUser (тип користувача) та username (логін).

Він зображений на рисунку 2.3.3.

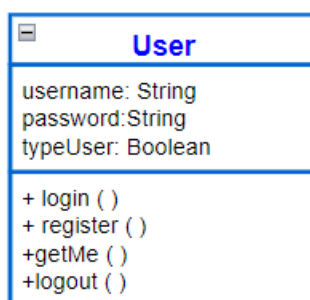


Рисунок 2.3.3 – Клас User

В таблиці 2.3.1 описано призначення методів даного класу.

Таблиця 2.3.1 – Методи класу User

Метод	Роль методу
login	Даний метод відповідає за авторизацію користувача. Він шукає збіг в базі даних Users з даними, які надійшли з клієнта. Також визначає роль користувача, якщо це лікар, то надається доступ до інтерфейсу працівника медичного закладу, якщо пацієнт, то інтерфейс буде іншим. А також генерує jwt-токен.
register	Даний метод створює нового користувача в тому випадку, якщо в базі даних не існує дублікату. Результатом відправляється відповідне повідомлення на клієнт.
getMe	Даний метод надає дані користувачу, який успішно авторизувався.
logout	Деактивує згенерований jsonwebtoken при авторизації.

Клас Patient описує сутність хворого. На рисунку 2.3.4 зображено цей клас.

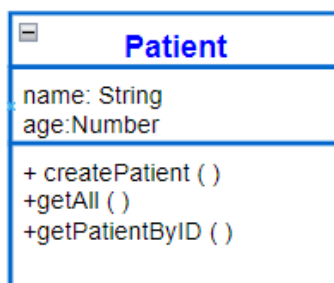


Рисунок 2.3.4 – Клас Patient

Його атрибути: name (ім'я хворого) та age (вік). У таблиці 2.3.2 наведено методи даного класу та описано їх роботу.

Таблиця 2.3.2 – Методи класу Patient

Метод	Роль методу
createPatient	Даний метод створює в базі даних нового пацієнта та заповнює такі поля як: ім'я пацієнта, ім'я доктора та вік. А також генерується дані для авторизації пацієнта в особистому кабінеті.

Продовження Таблиці 2.3.2 – Методи класу Patient

getAll	Даний метод повертає всіх пацієнтів, які були збережені в базі даних
getPatientById	Даний метод відповідає за пошук конкретного пацієнта за ключом, який передається з клієнта та повертає інформацію про даного пацієнта.

Клас MedicalCard описує головний об'єкт системи, який відповідає за роботу з медичною карткою пацієнта. На рисунку 2.3.5 зображено даний клас.

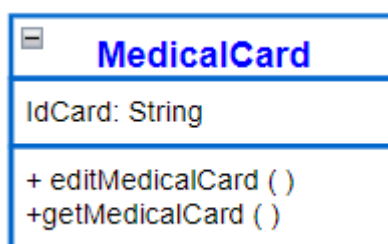


Рисунок 2.3.5 – Клас MedicalCard

Атрибутом цього класу є унікальний номер карти хворого. Методи editMedicalCard та getMedicalCard дозволяють редагувати та отримувати інформації з персональної картки пацієнта.

Класи Diagnose та Prescription є композиціями попереднього класу, тобто вони не можуть існувати один без одного. На рисунку 2.3.6 наведено дані класи.

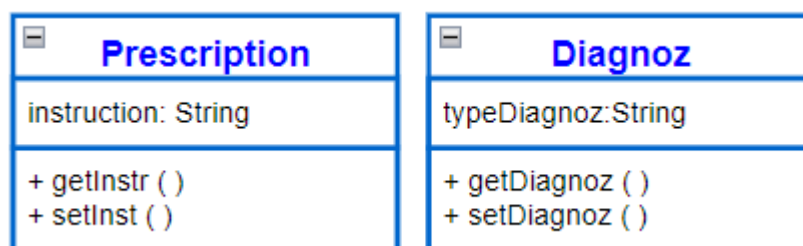


Рисунок 2.3.6 – Класи Diagnose і Prescription

Дані методи в своїх атрибутах зберігають значення діагнозу та інструкції лікування. Методи відповідають за збереження та отримання даних значень.

Для ясності розуміння взаємодії системи використано діаграми послідовностей. Вони показують послідовність взаємодії між об'єктами під час операції, повідомлення які відправляють один одному та в якому порядку. Дані діаграми зображуються у вигляді горизонтальних ліній – відображають об'єкти та вертикальні стрілки – відображають послідовність взаємодії між ними [4].

Процес здійснення авторизації в системі для пацієнта та працівника медичного закладу. Спочатку користувач вводить логін та пароль. Після цього введені дані порівнюються з даним в БД. Після цього користувач входить в систему. Діаграма послідовності для цього процесу зображена на рисунку 2.3.7.

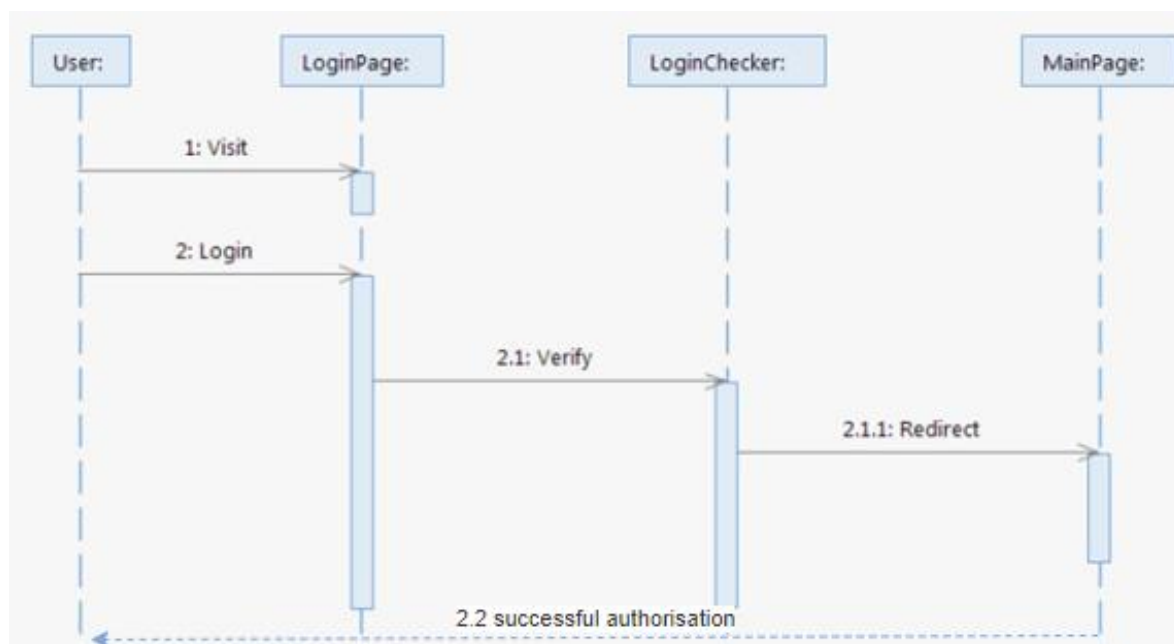


Рисунок 2.3.7 – Діаграма послідовності для авторизації

Процес додавання нового пацієнта. Спочатку медичний працівник вводить данні про хворого такі, як: фамілія, ім'я, та вік. Тоді система зберігає інформацію в екземплярі відповідної моделі. Діаграма послідовності для додавання нового пацієнта зображена на рисунку 2.3.8.

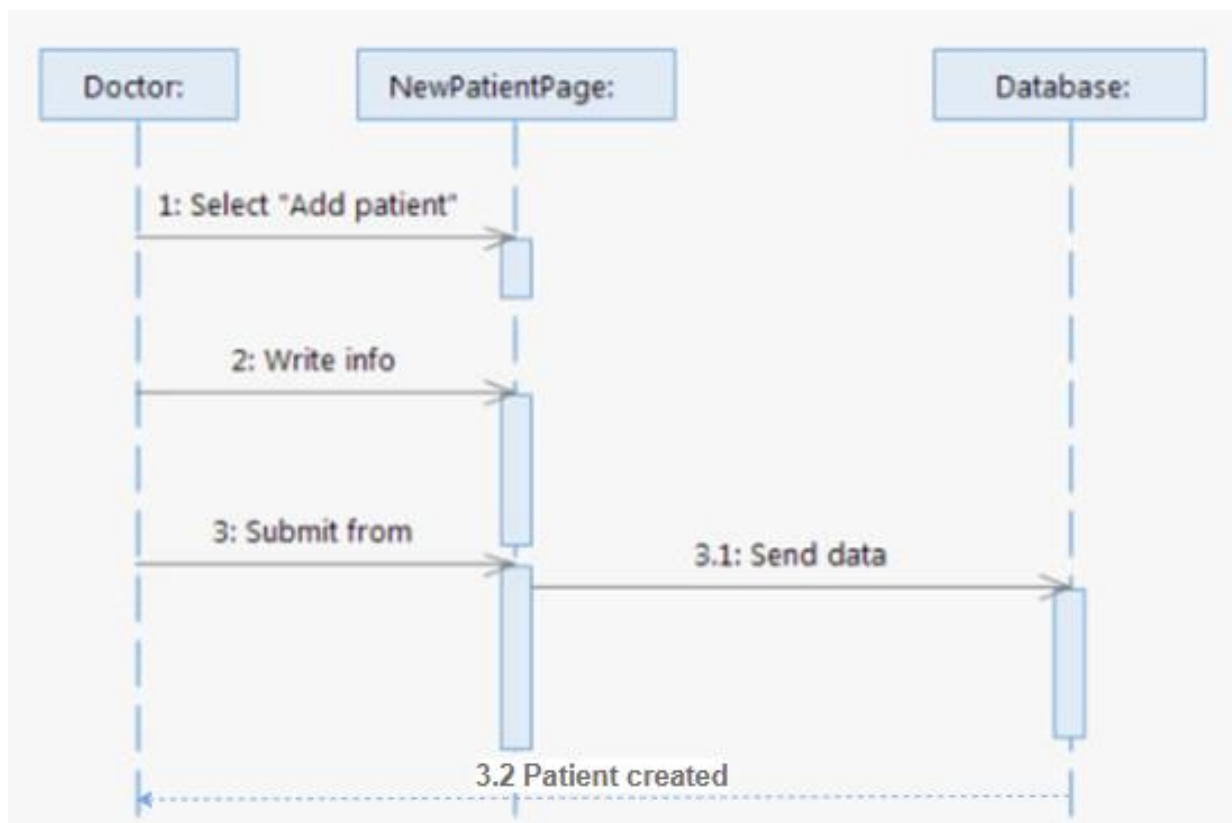


Рисунок 2.3.8 – Додавання нового пацієнта

Процес редагування медичної картки пацієнта. Спочатку медичний працівник вводить прізвище пацієнта, тоді натискає кнопку «Пошук». Після цього визначає діагноз та призначає лікування. В кінці система зберігає зміни. Діаграма послідовності для редагування медичної картки зображена на рисунку 2.3.9.

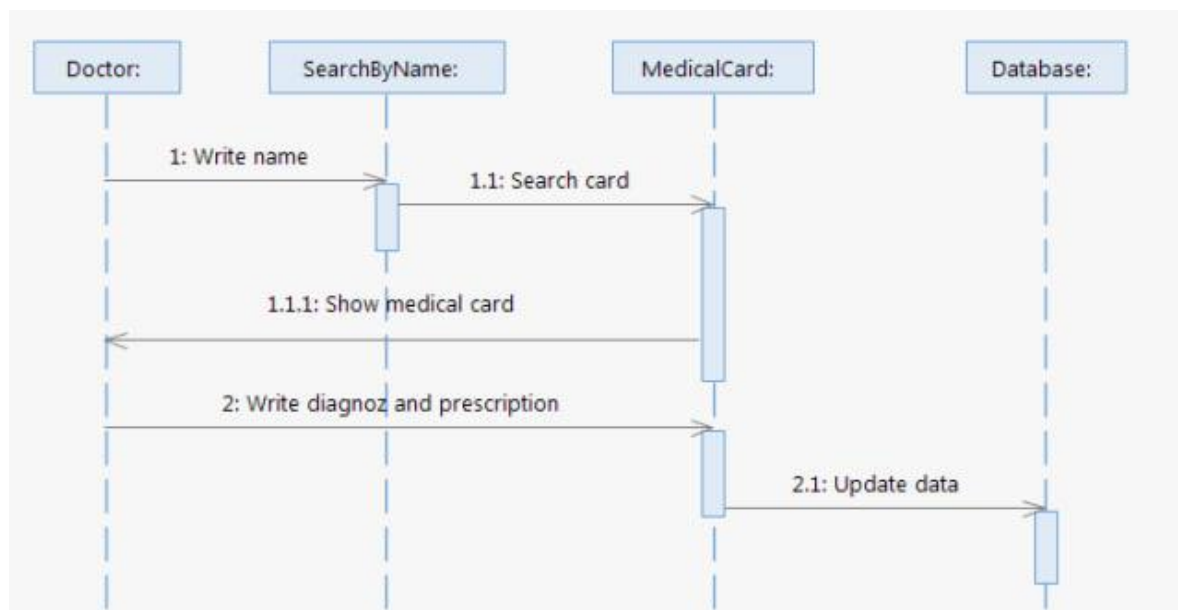


Рисунок 2.3.9 – Редагування медичної картки

Для запису на консультацію до спеціаліста хворий відкриває відповідну форму та вводить час, дату та вибирає лікаря. Після цього перевіряється чи вибрана дата не зайнята і створюється консультація. На рисунку 2.3.10 зображено діаграму послідовності для запису хворого на лікаря.

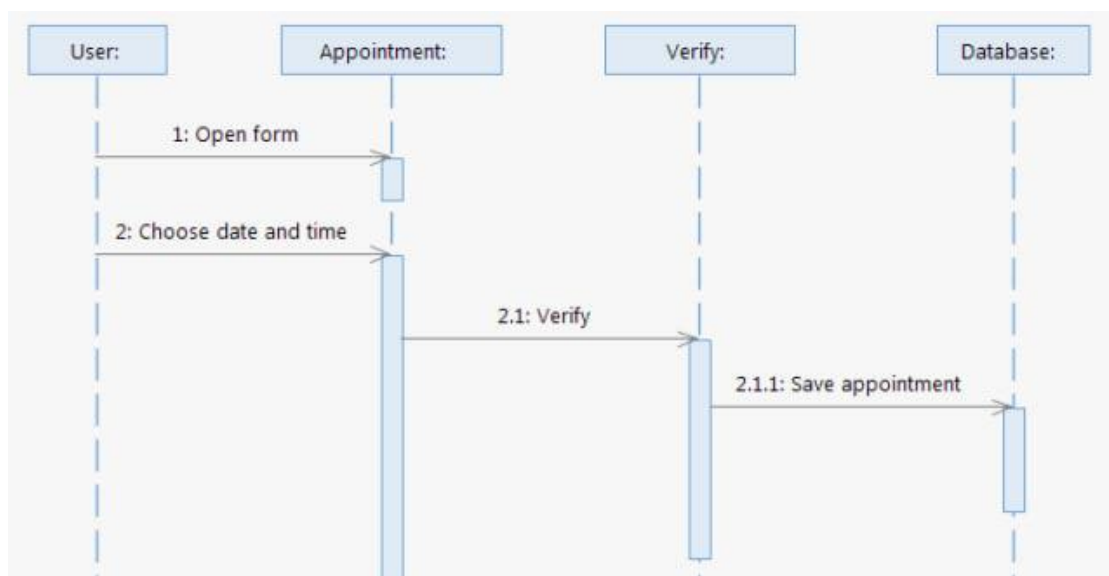


Рисунок 2.3.10 – Створення візиту до лікаря

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір основних технологій

Вибір технологій для розробки продукту є дуже важливим етапом, бо від нього залежать характеристики майбутнього додатку такі як: масштабованість, ефективність та продуктивність.

Питання вибору стеку технологій одночасно є важким і легким завданням. З одної сторони в даний час створено багато інструментів. Але в кожній технології є свої плюси та мінуси. Тому правильно оцінити переваги стеку технологій є важким та відповідальним етапом.

Для серверної частини програмісти найчастіше вибирають мови програмування, такі як Java, PHP, Python, JavaScript або C# та різні фреймворки, які покращують роботу.

Для клієнтської частини вибір мови програмування легший, бо найпопулярнішим варіантом є JavaScript [5]. Для неї створено багато різних бібліотек та фреймворків, які зроблять процес розробки простішим та приємнішим. А також базовими інструментами в розробці будь чого, що пов'язане з браузером є HTML та CSS.

Отже, при виборі стеку потрібно опиратися на вимоги та очікування від розробленого в майбутньому веб-сервісу чи будь якого іншого додатку.

Для реалізації за клієнт-серверною архітектурою веб-сервісу для зберігання та обробки даних пацієнтів у медичному закладі було вибрано такі технології.

JavaScript – мова програмування, яка широко використовується для розробки веб-додатків. Дана мова є одна з найпопулярніших у світі, бо забезпечує взаємодію користувача з веб-сторінкою [6].

Має такі переваги:

- Універсальність. Можна використати при розробці сервера і при розробці клієнтської частини.

- Інтерактивність. Дозволяє динамічно керувати елементами, які розташовані на веб-сторінці, додавати різні анімації.
- Легка у вивченні. На даний момент існує багато документації та відкритих прикладів коду. Це в поєднанні із зрозумілим синтаксисом дозволяє швидко вивчити її.
- Наявність величезної кількості бібліотек та фреймворків, які полегшують процес розробки.
- Швидкий розвиток.

Express.js – використовується в парі з попереднім фреймворком. Дане поєднання дає можливість просто та елегантно побудувати веб-додаток. Він доповнює можливості Node.js для обробки маршрутизації та HTTP-запитів [7].

Перевагами є:

- Мініمالізм. Дає можливість швидко і легко запуснути сервер.
- Маршрутизація. Дозволяє задати маршрути на яких будуть оброблятися різні HTTP – запити до певних URL-шляхів.

MongoDB – дуже популярний інструмент, який відноситься до нереляційних баз даних (NoSQL) і дозволяє ефективно зберігати та обробляти великі об'єми даних. Використовує JSON-подібні документи [8].

Переваги даної БД:

- Гнучка схема даних. Дана технологія зберігає інформацію в гнучкій схемі. Це означає, що документи можуть мати індивідуальні структури.
- Масштабованість. Можна розподілити дані на різні сервери. Це забезпечить обробку великого обсягу даних, витримування великого навантаження та швидкі відповіді на запити.
- Відновлення даних після збоїв.

React – фреймворк для створення клієнтської частини для мобільних чи веб-додатків [9]. Основні переваги:

- Компонентна архітектура. Дана технологія базується на розбитті інтерфейсу на незалежні компоненти. Вони можуть повторно

використовуватися і розширюватися. Це спрощує та пришвидшує розробку та зменшує копіювання та переписування одного і того ж коду.

- Віртуальний DOM. Дозволяє оновлювати тільки зміненні частини інтерфейсу. Дана можливість забезпечує високу продуктивність і полегшує роботу з станами компонентів.
- Широкі можливості. Існує багато бібліотек та інструментів для розробки, які утворюють екосистему.
- Спільнота. React має велике коло розробників, що дозволяє обговорити та знайти вирішення для більшості проблем, які виникають в програмістів, які тільки починають опановувати дану технологію.

JWT – використовується для безпечної передачі інформації у форматі JSON. Він створює спеціальні індивідуальні токени, які потрібно перед використанням правильно декодувати. Дуже часто використовують для авторизації користувачів. Коли користувач системи авторизується, то створиться спеціальний токен, який містить його інформацію і для отримання даної інформації потрібно використати спеціальний ключ [10].

Переваги:

- Безпека. Надає впевненість в тому, що інформація в токені не була змінена під час передачі.
- Розширюваність. Можна помістити в токен додаткову інформацію, наприклад, роль користувача, термін дії, а також додаткові властивості.
- Переносимість. Зручно передавати, бо зберігається у вигляді рядка у форматі JSON.

Redux – бібліотека для керування станом для додатків, які розроблені на React. Забезпечує односторонній потік даних, який називається Flux-архітектура [11].

Переваги використання:

- Простота. Використовуються чіткі та прості правила для управління станом. Вся інформація про стан додатку зберігається в одному об'єкті, це робить код передбачуваним та легким у читанні.
- Розширюваність. Можна швидко і просто додати новий функціонал. Для цього потрібно просто створити нові action, reducers та middleware. Головне те, що не потрібно змінювати код, який був написаний до цього.
- Зручне тестування. Існують потужні інструменти для відслідковування поведінки даних в програмі.

Tailwind – бібліотека стилів, яка надає доступ до готових шаблонів для швидкого створення і стилізації власних сайтів. Даний інструмент відрізняється від своїх аналогів тим, що вона не надає зразу готові компоненти, а надає набір класів, які розробник сам використовує та комбінує на власний розсуд і смак [12].

Основні переваги:

- Економія часу. Дає можливість швидко стилізувати елементи та компоненти без написання власного CSS-коду. Це дуже сильно економить час розробки додатку.
- Простота та зрозумілість. Класи мають дуже зручний синтаксис та інтуїтивні назви, що дозволяє розробнику, який має навіть мінімальні навички в стилізації, створювати красиві користувацькі інтерфейси.

Axios – дана бібліотека дає можливість виконувати HTTP-запити з сервера або з сервера. Має зручний інтерфейс для отримання асинхронних даних [13].

Основні переваги:

Простота використання. За допомогою простого і зрозумілого використання дає можливість швидко і легко виконувати запити різних видів, наприклад get, post, delete або put.

Автоматичне перехоплення та обробка помилок, які можуть виникнути під час виклику запиту. Це сильно спрощує обробку помилок та збільшує швидкість виявлення та усунення помилки в разі її виникнення.

Відповіді, які надходять автоматично перетворюються у формат JSON.

3.2 Розробка серверної частини додатку

У даному підрозділі буде описано реалізація серверної частини для веб-додатку, який реалізовується за архітектурою клієнт-сервер. Часто цю частину також називають backend.

Структуру серверної частини веб-додатку зображено на рисунку 3.2.1.

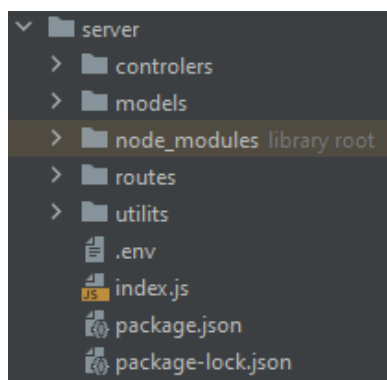


Рисунок 3.2.1 – Елементи сервера

З рисунка зрозуміло, що структура розбита на окремі теки в яких зберігаються певні файли, які відповідають за успішну роботу сервера. Дана структура містить такі елементи:

- Папка `controllers` – зберігає контролери, які в свою чергу відповідають за правильну обробку запитів, які будуть надходити з клієнтської частини.
- Папка `models` – в даній теці знаходяться схеми для бази даних MongoDB. В цих окремих моделях зберігаються шаблони для створення документів для зберігання та з'єднання з базою даних.
- Папка `node_modules` – зберігаються всі підключенні конфігурації, бібліотеки для успішної роботи системи.
- Папка `routes` – містить файли, що відповідають за маршрутизацію для обробки HTTP-запитів, які надсилаються з клієнтської частини.

- Папка `utils` – місце, де зберігаються модулі, які допомагають коректній роботі. В нашому випадку це файл `checkAuth.js` в ньому відбувається декодування спеціального токена, який створюється при авторизації користувача.
- Файл `env` – зберігає та приховує значення констант, таких як: номер порту на якому запускається сервер, логін та пароль для доступу до бази даних та спеціальний ключ для декодування `jsonwebtoken`.
- Файл `index.js` – головний файл серверної частини, де виконується налаштування сервера.
- Файл `package.json` – відповідає за налаштування додатковими пакетами.
- Файл `package-lock.json` – зберігає версії бібліотек, які потрібні для роботи.

В загальному принцип роботи серверної частини полягає в тому, що `Node.js` в одному потоці обробляє декілька запитів і це дозволяє отримувати високу продуктивність. Коли сервер отримує запит від клієнта, то він обробляє цей запит за допомогою функції, яка відповідає за певний маршрут і виконує зазначену логіку. Після цього оброблений запит відправляється на клієнт у форматі JSON.

Варто пояснити алгоритм використання JWT-токена, оскільки він часто використовується у даній системі.

На прикладі авторизації можна навести такий алгоритм:

1. Надсилається запит з клієнта на сервер для автентифікації.
2. Сервер виконує перевірку введених даних і, якщо вони коректні, то генерується JWT-токен.
3. Сервер для генерації токена використовує секретний ключ.
4. Згенерований токен надсилається назад на клієнт, як результат успішної авторизації.
5. На клієнті зберігається отриманий токен.
6. При всіх наступних запитах на сервер клієнт використовує даний токен для автентифікації.

7. Сервер використовує секретний ключ для перевірки токена, якщо все добре, то обробляється запит.
8. Після закінчення терміну дії токена потрібно виконувати повторну аутентифікацію.

На рисунку 3.2.2 наведено приклад реєстрації з даною технологією.

```
try {
  const {username,password}=req.body
  const isUsed = await User.findOne({username})
  if(isUsed){
    return res.json({message:'Даний username зайнятий',})}
  const salt = bcrypt.genSaltSync( rounds: 10)
  const hash=bcrypt.hashSync(password,salt)
  const newUser=new User({
    username,
    password: hash,
  })
  const token = jwt.sign( payload: {id: newUser._id,},
    process.env.JWT_SECRET, options: {expiresIn:'30d'},)
  await newUser.save()
  res.json({newUser,message:'Реєстрація успішна'})
}
catch(error){res.json({message:'Помилка при створенні користувача'})}
```

Рисунок 3.2.2 – Реєстрація з JWT

Даний код перевіряє чи немає збігу в базі даних з отриманою інформацією, якщо все добре, то пароль шифрується за допомогою бібліотеки bcrypt. Після цього створюється json web token в який записується індивідуальний номер користувача, секретний ключ та вказується термін дії 30 днів. Останнім кроком, зберігається екземпляр модулі User та відправляються на клієнт відповідні повідомлення.

Процес створення хворого лікарем, якщо він попередньо не зареєстрований теж виконується за подібним алгоритмом. Але не використовується інструмент jwt, бо працівник медичного закладу вже є авторизованим в системі. Приймаються основні параметри, яких достатньо для додавання пацієнта – вік та ім'я. Після цього дані зберігаються в екземплярі моделі Patient та відправляються інформація про позитивну або негативну дію. На рисунку 3.2.3 наведено даний код.

```

try{
  const {patientName,age,doctorName} = req.body
  const newPatient = new Patient({patientName, age, doctorName,})
  await newPatient.save()
  res.json({newPatient, message:'Додано пацієнта',})
}catch(error)
{
  res.json({message:'Помилка при створенні пацієнта'})
}

```

Рисунок 3.2.3 – Додавання хворого лікарем

Всі інші методи збереження даних відбуваються за таким самим алгоритмом. Процес отримання даних трохи відрізняється. Потрібно використовувати вбудовані методи MongoDB, які стають доступними для створених моделей. Для прикладу виведення списку пацієнтів використовує вбудовану функцію `find`, яка викликається без параметрів і в такому випадку вона повертає всі записи. На рисунку 3.2.4 наведено код, який відправляє на клієнт всіх хворих.

```

try{
  const patients = await Patient.find().sort( arg: '-createdAt')
  if(!patients){
    return res.json({message:'Пацієнтів немає'})
  }
  res.json({patients})
}catch(error){
  res.json({message:'Щось пішло не так!'})
}

```

Рисунок 3.2.4 – Список хворих

У випадку, якщо буде помилка замість об'єкта з хворими відправитися повідомлення про проблему, яка виникла.

3.3 Розробка клієнтської частини

3.3.1 Архітектура клієнта

Для розробки клієнтської частини наведено структуру проєкту на рисунку 3.3.1 у середовищі розробки WebStorm 2022.

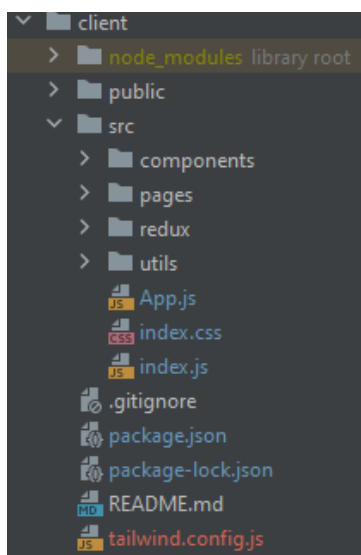


Рисунок 3.3.1.1 – Структура клієнта

В даній структурі можна помітити основні та стандартні елементи: теку для зберігання всіх бібліотек, які використовуються при розробці, теку для збереження компонентів на яких саме і базується фреймворк React, тека для збереження сторінок, які наявні у розробленому веб-додатку, тека для збереження стану, та основні файли для забезпечення роботи клієнтської частини.

Робота клієнта відбувається за принципом Flux-архітектури. Вона базується на односторонньому потоці, який проходить через такі етапи, як: action, dispatcher, store та відображення [14]. Диспетчер реагує на дію, яка була виконана користувачем та оновлює стан додатку. Наступний етап «Store» зберігає стан додатку. На рисунку 3.3.1.2 зображено схему роботи клієнтської частини.

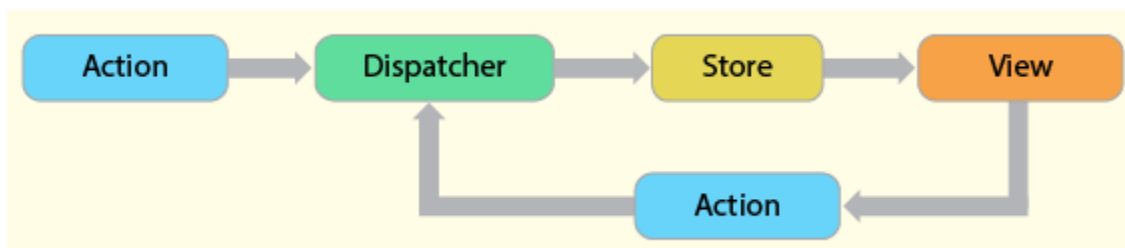


Рисунок 3.3.1.2 – Алгоритм flux архітектури

Дана архітектура легко реалізовується і є ефективною при розробці такими інструментами як React та Redux.

Тепер для кращого розуміння, кожний етап буде представлено частини лістингу коду. Для прикладу візьмемо сценарій для створення нового користувача в системі.

Спочатку система очікує якусь дію і знаходиться на етапі «Action». На рисунку 3.3.1.3 наведено код кнопки для підтвердження реєстрації.

```

<button type='submit'
  onClick={handleSubmit}
  className='flex justify-center items-
>
  Підтвердити
</button>
  
```

Рисунок 3.3.1.3 – Кнопка підтвердження реєстрації

В даному випадку система чекає на подію «handleSubmit», яка накладена на даний тег. Після цього почнеться етап «Dispatcher», який буде оброблений та асинхронною функцією дані відправляться на сервер. Код, який відправить на сервер об'єкт із значеннями «username», «password» та «typeUser» наведено на рисунку 3.3.1.4.

```
export const registerUser=createAsyncThunk(
  typePrefix: 'auth/registerUser',
  payloadCreator: async ({username,password})=>{
    try {
      //відправка на сервер паролю та логіна
      const {data} = await axios.post( url: '/auth/register', data: {
        username,
        password,
      })
      if(data.token){
        window.localStorage.setItem('token',data.token)}
      return data
    }catch (error){
      console.log(error)}
  },
)
```

Рисунок 3.3.1.4 – Асинхронна функція для реєстрації

Ця функція за допомогою інструментарію, який надає бібліотека Redux Toolkit створює HTTP-запит типу POST на адресу «/auth/register». Після вдалого запиту, перевіряється наявність токена у відповіді, яка надійшла з сервера, якщо він наявний, то зберігається в «Store». Далі зміни відображаються користувачу.

3.3.2 Компоненти

При розробці клієнтської частини додатку використовуються компоненти. Вони дозволяють скоротити час розробки проекту. Це окремі самостійні частини коду, які можна повторно використовувати.

На рисунку 3.3.2.1 зображено список компонентів, які створені для відображення та забезпечення надійної та зручної роботи клієнта.

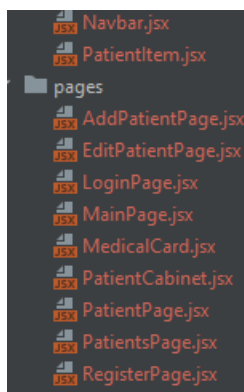


Рисунок 3.3.2.1 – Компоненти

Дані компоненти відповідають за такий інтерфейс:

- Navbar – вивід навігаційного меню.
- PatientItem – представлення інформації про пацієнта.
- AddPatientPage – створення нового пацієнта.
- EditPatientPage – редагування інформації хворого.
- LoginPage – авторизація користувача.
- MainPage – головна сторінка.
- MedicalCard – внесення інформації про стан та перебіг здоров'я.
- PatientCabinet – особистий кабінет хворого.
- PatientsPage – список пацієнтів медичного закладу.
- RegisterPage – реєстрація.

3.3.3 Маршрутизація

Для забезпечення маршрутизації використовується бібліотека React Router. Вона відповідає за відображення повної компоненти на конкретному URL-шляху.

Кожен елемент Route це певний маршрут до сторінки веб-додатка і він містить два атрибути: «path» (адреса) та «element» (назва компоненти). На рисунку 3.3.3.1 зображено всі маршрути, які використовує веб-сервіс

```

<Routes>
  <Route path="/" element={<MainPage />} />
  <Route path='patients' element={<PatientsPage />} />
  <Route path=':id' element={<PatientPage />} />
  <Route path=':medicalCard/edit' element={<EditPatientPage />} />
  <Route path='new' element={<AddPatientPage />} />
  <Route path='register' element={<RegisterPage />} />
  <Route path='login' element={<LoginPage />} />
  <Route path='medicalCard' element={<MedicalCard />} />
  <Route path='appointment' element={<Appointment />} />
  <Route path='patientCabinet/addAppointment' element={<AddAppointment />} />
  <Route path='patientCabinet' element={<PatientCabinet />} />
  <Route path='patientCabinet/medicalCard' element={<MedicalCard />} />
</Routes>

```

Рисунок 3.3.3.1 – Маршрутизація

Даний код є частиною файлу App.js повний код винесено в додаток А.

3.4 Створення бази даних

Для збереження даних використано популярну нереляційну базу даних MongoDB. Створено схему, яка містить шість моделей, а саме: Patients, Users, Doctor, InfoPatients, VisitDoctor та Prescription.

На рисунку 3.4.1 зображено модель для зберігання інформації про хворих. Містить індивідуальний номер, прізвище, ім'я, вік, дату створення та останнього оновлення та індивідуальний номер лікаря та медичної картки .


```

_id: ObjectId('64726dd79a7b2c0732feea7c')
patientName: "Василь"
age: 21
createdAt: 2023-05-27T20:53:43.017+00:00
updatedAt: 2023-05-27T20:53:43.017+00:00
__v: 0
MedicalCard: "64847887fe8badf8faa9897f"
doctorId: "64847887fe8badf8faacd43f"

```

Рисунок 3.4.1 – Модель хворого

На рисунку 3.4.2 зображено модель для обробки даних користувача. Містить роль користувача в системі, та стандартні дані для входу, а також час створення та оновлення. Тип даних користувач Boolean, тобто, якщо true – лікар, false – пацієнт. Пароль зашифрований для безпеки.

```

_id: ObjectId('647daca3bd6a25e8e36befb2')
username: "DcHause"
password: "$2a$10$gu4aR17yWBxrQJaVMSjGd.2oJSX9MiTZcDgYlgmKl31Vf1qKe9ICi"
createdAt: 2023-06-05T09:36:35.632+00:00
updatedAt: 2023-06-05T09:36:35.632+00:00
__v: 0
typeUser: "true"

```

Рисунок 3.4.2 – Модель користувача

На рисунку 3.4.3 зображено модель для зберігання даних для медичного працівника. Містить ім'я, індивідуальний номер та масив з індивідуальними номерами пацієнтів.

```

_id: ObjectId('64847887fe8badf8faa9897f')
arrayPatientsId: Array
doctorName: "Dc Hause"

```

Рисунок 3.4.3 – Модель Лікаря

На рисунку 3.4.4 зображено модель для медичної картки хворого. Містить назву діагнозу, інструкцію щодо лікування та індивідуальний номер.

```

_id: ObjectId('64847b2cfe8badf8faa98984')
diagnoz: "Грип"
prescription: "Інгаляції 2 рази на день
               Льодяники
               Чай з малини"

```

Рисунок 3.4.4 – Модель медичної картки

На рисунку 3.4.5 зображено модель для збереження даних про візити до лікаря. Вона містить дату і час та індивідуальні номери лікаря та пацієнта і свій власний.

```

_id: ObjectId('64847c9dfe8badf8faa98986')
doctorId: "64847887fe8badf8faa9897f"
patinetId: "64726dd79a7b2c0732feea7c"
visitDate: 2022-07-01T11:00:00.000+00:00

```

Рисунок 3.4.5 – Модель консультації

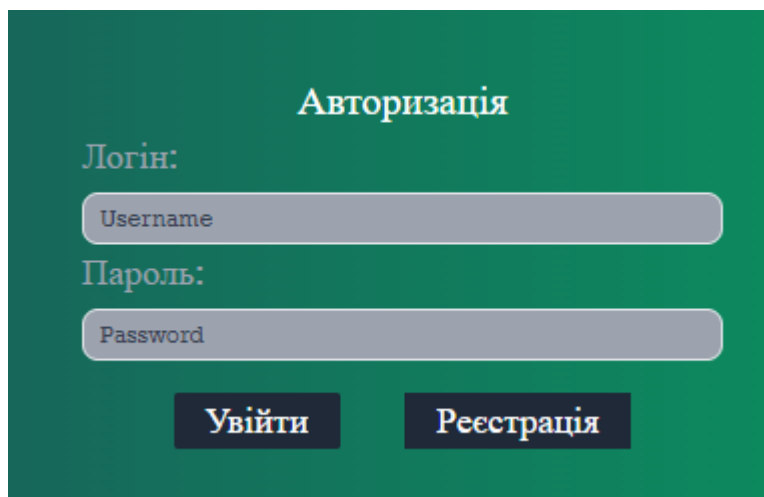
Дані схеми легко створити та настроїти, оскільки вони створюються за допомогою спеціального синтаксису в окремому файлі проекту.

4 ТЕСТУВАННЯ ТА ІНТЕРФЕЙС

4.1 Перевірка роботи системи

В даному розділі буде перевірено на коректність дії розроблений веб-додаток для медичного закладу. Тестування проведено в мануальному режимі.

На етапі авторизації користувач зустрінеться з відповідним інтерфейсом. Буде два сценарії: увійти в систему або створити дані для входу. На рисунку 4.1.1 зображено сторінку «Авторизації», на якій буде два поля для введення логіну та паролю і дві кнопки.



The image shows a login interface on a green background. At the top, the title 'Авторизація' is centered in white. Below it, the label 'Логін:' is followed by a light blue rounded rectangular input field containing the placeholder text 'Username'. Underneath, the label 'Пароль:' is followed by a similar light blue rounded rectangular input field containing the placeholder text 'Password'. At the bottom, there are two dark blue buttons with white text: 'Увійти' on the left and 'Реєстрація' on the right.

Рисунок 4.1.1 – Сторінка авторизації

Якщо натиснути кнопку «Реєстрація», то відкриється відповідна сторінка. На ній буде форма для вводу логіну, паролю та ролі користувача. Також можливість повернутися на попередню сторінку, якщо користувач вже зареєстрований, для цього потрібно натиснути на напис «Вже є акаунт?».

На рисунку 4.1.2 зображено сторінку для реєстрації.

Рисунок 4.1.2 – Сторінка реєстрації

Після підтвердження даної форми в базі даних MongoDB збережеться інформація. Пароль буде зашифрованим для забезпечення безпеки. На рисунку 4.1.3 зображено вигляд створеного користувача в базі даних. Якщо поле `typeUser` зберігає значення `true`, то це лікар в протилежному випадку пацієнт.

```
_id: ObjectId('647daca3bd6a25e8e36befb2')
username: "DcHause"
password: "$2a$10$gu4aR17yWBxrQJaVMSjGd.2oJSX9MiTZcDgYlgmKl31Vf1qKe9ICi"
createdAt: 2023-06-05T09:36:35.632+00:00
updatedAt: 2023-06-05T09:36:35.632+00:00
__v: 0
typeUser: "true"
```

Рисунок 4.1.3 – Дані користувача в БД

Після виконання основних дій програма інформує користувача про успішність виконання або помилку. Приклад повідомлення про «Успішну реєстрацію» зображено на рисунку 4.1.4.

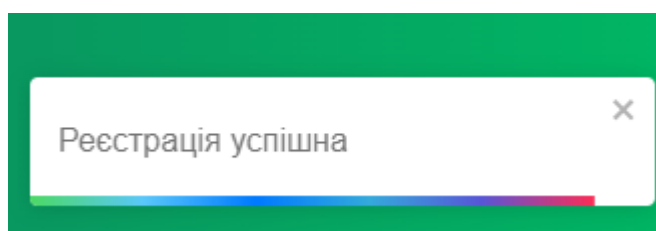


Рисунок 4.1.4 – Приклад повідомлення

Після успішної авторизації для лікаря буде відкрита головна сторінка. На ній присутня навігація для переходу на такі сторінки «Пацієнти» та «Добавити пацієнта». Назва активної сторінки підсвічується. Також присутній пошук пацієнта за фамілією та дві кнопки «Заплановані прийоми» та «Створити прийом». На рисунку 4.1.5 зображено даний інтерфейс.

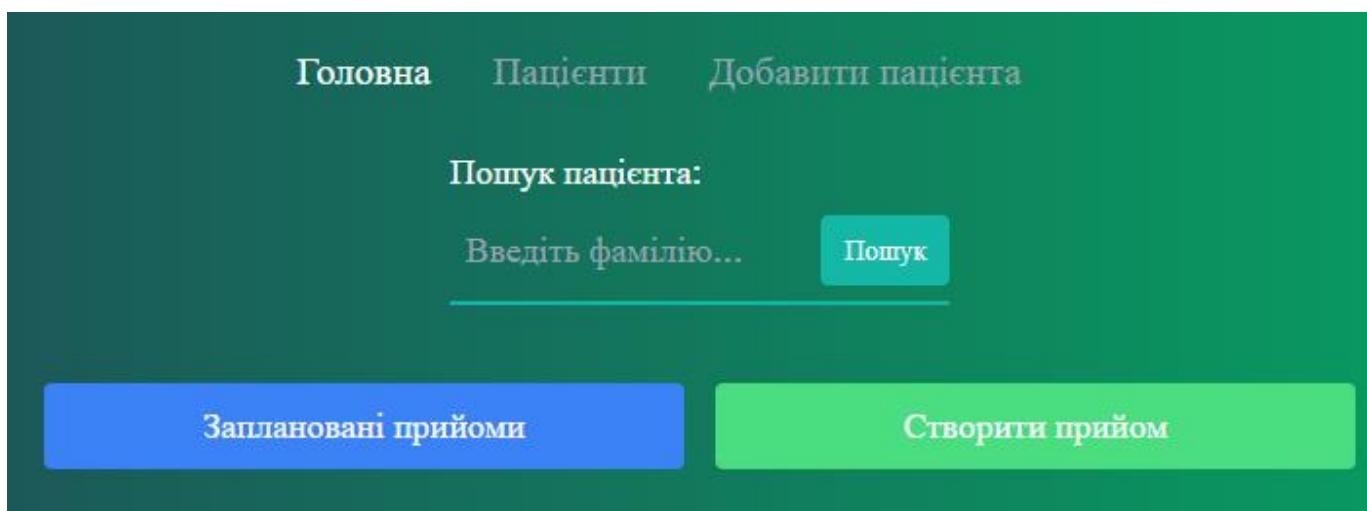


Рисунок 4.1.5 – Головна сторінка

Якщо перейти на сторінку «Добавити пацієнта» можна побачити форму для створення нового клієнта медичного закладу. Для цього потрібно ввести основну інформацію, таку як: ім'я, вік, та ім'я лікаря. Подальша інформація про хворого буде добавлятися у медичній картці лікарем під час консультації. На рисунку 4.1.6 зображено дану форму з відповідними полями. Також є кнопка для підтвердження та кнопка «Скасувати», яка очищує форму.

The screenshot shows a web interface with a green header bar containing three links: 'Головна', 'Пацієнти', and 'Добавити пацієнта'. Below the header, there are three input fields for patient information: 'Ім'я пацієнта:' with the value 'Василь', 'Вік пацієнта:' with the value '21', and 'Ім'я лікаря:' with the value 'Юлія'. At the bottom of the form are two buttons: a dark blue button labeled 'Добавити' and a red button labeled 'Скасувати'.

Рисунок 4.1.6 – Створення нового пацієнта

Після підтвердження вище згаданої форми, хворий з'явиться на сторінці «Пацієнти». Тут можна переглянути медичну картку та видалити пацієнта. На рисунку 4.1.7 зображено список пацієнтів.

The screenshot shows a web interface with a green header bar containing three links: 'Головна', 'Пацієнти', and 'Добавити пацієнта'. Below the header, the title 'Пацієнти' is displayed. The main content area shows a list of two patients. Each patient entry consists of the patient's name, the doctor's name, a 'Мед. карта' button, and a 'Видалити' button.

Пацієнт	Лікар	Мед. карта	Видалити
Пацієнт: Василь	Лікар: Юлія	Мед. карта	Видалити
Пацієнт: John	Лікар: Dr. Hause	Мед. карта	Видалити

Рисунок 4.1.7 – Список пацієнтів

Медична картка пацієнта головний документ, бо він містить інформацію про перебіг лікування та інструкції, яких потрібно дотримуватися для успішного вилікування. На рисунку 4.1.8 зображено приклад даної форми.

The image shows a web application interface with a green header bar containing three links: "Головна", "Пацієнти", and "Добавити пацієнта". Below the header is a white card titled "Медична карта пацієнта". The card contains the following text: "Ім'я та прізвище: Москалик Василь", "Діагноз: Запалення легенів", "Інструкції щодо лікування", "Інгаляції 2 рази в тиждень", "Імет 1 таблетка (при підвищені температури)", "Сироп Евкалор", and "Чай з малини". At the bottom of the card is a green button labeled "Редагувати".

Рисунок 4.1.8 – Приклад медичної картки

На рисунку 4.1.9 зображено форму для редагування медичної картки. Дана сторінка ідентична з попередньою, але є можливість редагування полів.

Головна Пацієнти Добавити пацієнта

Медична карта пацієнта

Ім'я та прізвище

Москалик Василь

Діагноз

Запалення легенів

Інструкції щодо лікування

Інгаляції 2 рази в тиждень
Імет 1 таблетка (при підвищені температури)
Сироп Евкалор
Чай з малини

Зберегти

Рисунок 4.1.9 – Редагування медичної картки

На вищезгаданій головній сторінці була можливість перегляду записів до лікаря. Отже на рисунку 4.1.10 зображено сторінку на якій лікар може подивитися графік запланованих до нього консультацій. На ньому відображається дата та час проведення зустрічі та фамілія пацієнта.

Головна	Пацієнти	Додати пацієнта
Дата	Час	Пацієнт
2023-06-05	10:00 AM	Стець
2023-06-06	11:30 AM	Москалик
2023-06-07	2:15 PM	Новосад

Рисунок 4.1.10 – Графік консультацій

Також наявний інтерфейс для пацієнта. Якщо в системі авторизується даний користувач, то він буде мати обмежений функціонал. Йому доступно лише переглянути медичну картку та записатися на прийом. На рисунку 4.1.11 зображено головну сторінку особистого кабінету пацієнта.

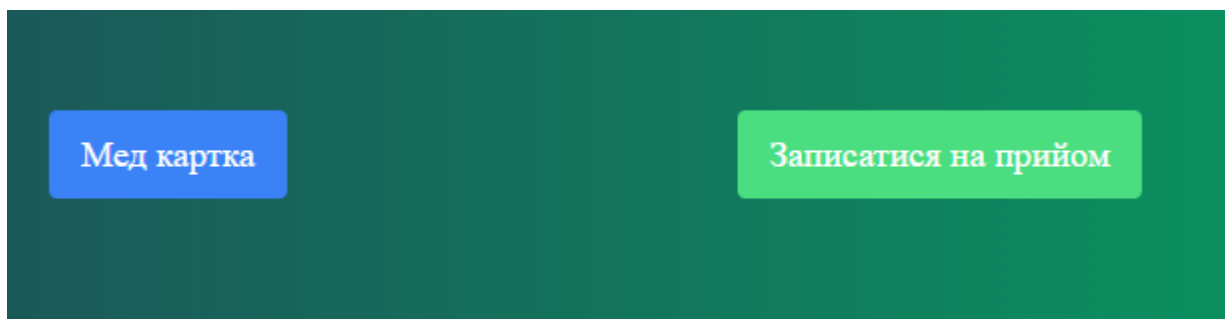
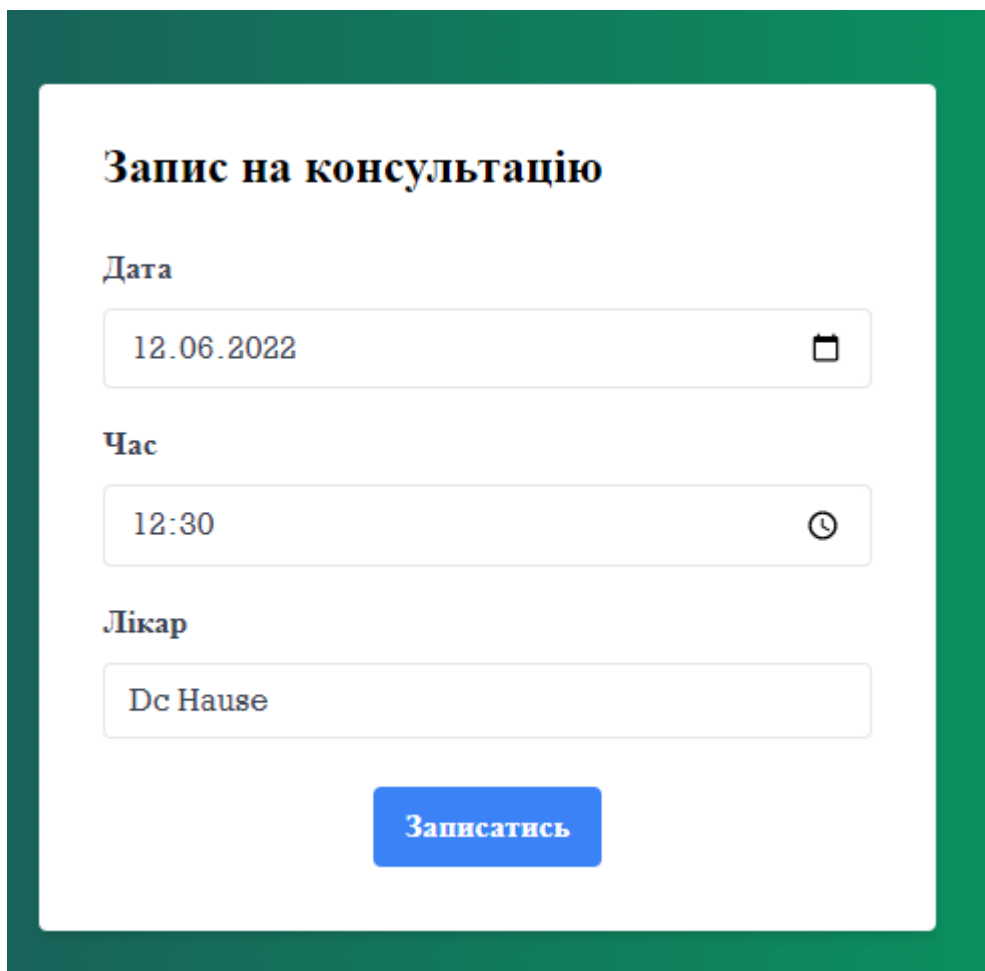


Рисунок 4.1.11 – Особистий кабінет пацієнта

Оскільки інтерфейс мед картки ідентичний вищеписаному інтерфейсу при авторизації зі сторони лікаря, то доцільно показати форму для запису на прийом. Вона містить такі поля: дата, час та ім'я лікаря до якого хворий має бажання потрапити на консультації. На рисунку 4.1.12 зображено дану форму.



The image shows a web form for booking a consultation. It is titled 'Запис на консультацію' (Consultation Booking). The form is set against a dark green background. It contains three input fields: 'Дата' (Date) with the value '12.06.2022', 'Час' (Time) with the value '12:30', and 'Лікар' (Doctor) with the value 'Дс Hause'. Each field has a corresponding icon (calendar, clock, and a person icon respectively). Below the fields is a blue button labeled 'Записатись' (Book).

Запис на консультацію

Дата

12.06.2022

Час

12:30

Лікар

Дс Hause

Записатись

Рисунок 4.1.12 – Створення консультації

Отже, після проведення тестування інтерфейсу користувачів виявлено, що він працює належним чином для двох ролей. Також можна додати, що використання даного інтерфейсу є зручним та інтуїтивним.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОХОРОНА ПРАЦІ

5.1 Долікарська допомога при ураженні електричним струмом

Охорона праці є невід'ємною складовою частиною будь-якої організації та діяльності, де здійснюються роботи. Вона має на меті забезпечити безпечні та здорові умови праці для всіх працівників, запобігти виникненню нещасних випадків та професійних захворювань, а також зберегти та зміцнити фізичне та психічне здоров'я працюючих.

Під час організації робочих місць та виконання робіт необхідно дотримуватися вимог безпеки, що забезпечуються на рівні національного законодавства, міжнародних стандартів та рекомендаційних документів.

Електробезпека при роботі. Заходи щодо усунення небезпеки ураження електричним струмом зводяться до правильного розміщення устаткування та електричних кабелів. Інші заходи щодо забезпечення електробезпеки, збігаються з загальними заходами пожежної та електробезпеки. В якості профілактичних заходів для забезпечення пожежної безпеки слід використовувати приховану електромережу, надійні розетки з пожежобезпечних матеріалів, силові мережі живлення устаткування виконувати кабелями, розрахованими на підключення в 3-5 разів більшого навантаження, включати й виключати живлення обладнання за допомогою штатних вимикачів. Треба регулярно робити очистку внутрішніх частин комп'ютерів, іншого устаткування від пилу, розташовувати комп'ютери на окремих неспалених столах. Для запобігання іскріння необхідно рідше встромляти і виймати штепсельні вилки з розеток [15].

Освітлення. Система освітлення повинна відповідати таким вимогам: освітленість на робочому місці повинна відповідати характеру зорової роботи, який визначається трьома параметрами: об'єктом розрізнення - найменшим розміром об'єкта, що розглядається на моніторі ПК; фоном, який характеризується коефіцієнтом відбиття; контрастом об'єкта і фону; необхідно забезпечити

достатньо рівномірне розподілення яскравості на робочій поверхні монітора, а також в межах навколишнього простору; на робочій поверхні повинні бути відсутні різкі тіні; в полі зору не повинно бути відблисків (підвищеної яскравості поверхонь, які світяться та викликають осліплення); величина освітленості повинна бути постійною під час роботи; слід обирати оптимальну спрямованість світлового потоку і необхідний склад світла [16].

Вимоги до монітору. Робочі місця мають бути розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1 м, між собою на відстані не менше 1,5 м. Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природне світло падало на нього збоку, переважно зліва. Робочі місця слід розташовувати так, щоб уникнути попадання в очі прямого світла. Джерела освітлення рекомендується розташовувати з обох боків екрану паралельно напрямку погляду. Для уникнення світлових відблисків екрану, клавіатури в напрямку очей користувача, від світильників загального освітлення або сонячних променів, необхідно використовувати антиполюскові сітки, спеціальні фільтри для екранів, захисні козирки, на вікнах - жалюзі. Екран дисплея повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він розташований під кутом, то стає причиною сутулості. Відстань від дисплея до очей повинна трохи перевищувати звичну відстань між книгою та очима. Перед екраном монітора, особливо старих типів, повинен бути спеціальний захисний екран. При його відсутності треба сидіти на відстані витягнутої руки від монітора. Фільтри з металевої або нейлонової сітки використовувати не рекомендується, тому що сітка спотворює зображення через інтерференцію світла. Найкращу якість зображення забезпечують скляні поляризаційні фільтри. Вони усувають практично всі відблиски, роблять зображення чітким і контрастним. Ще одним моментом, який стосується зору, є необхідність створення неоднорідного поля зору. Для цього можна розвісити на поверхнях (стінах) плакати та картини, виконані у спокійних тонах. Наприклад, пейзажі. При роботі з текстовою інформацією (в режимі введення даних та редагування тексту, читання з екрану) найбільш фізіологічним

правильним є зображення чорних знаків на світлому (чорному) фоні. Монітор повинен бути розташований на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані 400-700 мм від очей користувача. Рекомендується розміщувати елементи робочого місця так, щоб витримувалася однакова відстань очей від екрана, клавіатури, тексту.

Робоча поза. Зручна робоча поза при роботі з комп'ютером забезпечується регулюванням висоти робочого столу, крісла та підставки для ніг. Рациональною робочою позою може вважатися таке положення, при якому ступні працівника розташовані горизонтально на підлозі або підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук - вертикальні. Кут ліктьового суглоба коливається в межах 70-90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°. Важливою є форма спинки крісла, яка повинна повторювати форму спини. Висота крісла повинна бути такою, щоб користувач не почував тиску на куприк або стегна. Крісло бажано обладнати бильцями. Його потрібно встановити так, щоб не треба було тягтися до клавіатури. Періодично користувачу необхідно рухатися, вчасно змінювати положення тіла і робити перерви у роботі. При напруженій роботі за комп'ютером щогодини необхідно робити перерву на 15 хвилин через кожну годину і треба займатися іншою справою. Декілька разів на годину бажано виконувати серію легких вправ для розслаблення. Для нейтралізації зарядів статичної електрики в приміщенні, де виконується робота на комп'ютерах, в тому числі на лазерних та світлодіодних принтерах, рекомендується збільшувати вологість повітря за допомогою кімнатних зволожувачів. [17].

5.2 Вимоги безпеки до робочих місць користувачів ПК у медичних закладах.

При ураженні електричним струмом слід негайно звільнити потерпілого від його дії шляхом вимкнення електричного струму або відривання його від джерела

струму тримаючись за одяг потерпілого, якщо він сухий, чи ставши на гумову ковдру, суху дошку, картон, фанеру, брезент. Якщо потерпілий опинився в стані непритомності, слід забезпечити йому надходження свіжого повітря, розстібнути тісний одяг, дати нюхати нашатирний спирт, обприскати водою, розтирати і зігрівати тіло. Негайно викликати швидку медичну допомогу за номером 103 [17].

При рідкому та судорожному диханні потерпілого, слід робити йому штучне дихання. Штучне дихання – це лікарський засіб, метою якого є відновлення природного дихання потерпілого [18].

Засіб штучного дихання являє собою механічний, ритмічний вплив на грудну клітину та дихальні м'язи потерпілого. При цьому створюється газообмін у легенях і поступово може відновитися природне дихання.

Допомогу потерпілому повинні надавати дві особи:

- одна з них накладає долоню на нижню третину грудей і створює поштовхи на груди з частковою приблизно 50-ти поштовхів на хвилину. Стискання серця між грудьми та хребтом призводить до виштовхування крові із серця в судини і сприяє відновленню кровотоку;

- друга надає допомогу, щільно притуливши свої губи до рота потерпілого і вдуває в його легені повітря. При цьому грудна клітина розширюється і з утворюється пасивний вдих. Коли особа, яка надає допомогу відсторонюється, грудна клітина потерпілого спадає і в нього утворюється пасивний видих [19].

Надавати цю допомогу потерпілому продовжувати до прибуття кваліфікованої медичної допомоги.

При отриманні потерпілим опіків, не торкатися руками обпечених місць, обережно накласти на пошкоджені місця стерильну пов'язку і відвести потерпілого до медичного закладу.

ВИСНОВКИ

Розробка інтернет додатків для медичних закладів є потрібним та багатообіцяючим напрямком. Створення даних веб-сервісів оптимізують роботу медичного персоналу та обслуговування пацієнтів стане кращим.

Метою даної кваліфікаційної роботи було розробка веб-сервісу для медичного закладу, який забезпечить надійний та швидкий доступ до інформації хворих. Завдяки цьому додатку медичний персонал отримає зручний інструмент для лікування. Пацієнти в свою чергу, зможуть відслідковувати персональну медичну інформацію в будь-який час. Це допоможе покращити взаємозв'язок між лікарем та пацієнтом.

Проаналізовано та оцінено сильні та слабкі сторони конкурентів на ринку. Згідно даного аналізу головним конкурентом є компанія «Helsi».

Спроектовано та описано за допомогою UML-діаграм основні сценарії при використанні розробленого продукту. Також оформлено таблиці для класів, які реалізують серверну частину застосунку. В них описуються методи та їх призначення.

Для реалізації вибрано клієнт-серверну архітектуру та технології, які дозволяють легко реалізувати даний шаблон проектування. Для цього вибрано стек MERN. Під час виконання роботи було реалізовано клієнтську та серверну частину додатка.

Результатом кваліфікаційної роботи стало розробка веб-сервісу для обробки інформації про хворих у медичному закладі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HELSI - інформаційна система для пацієнтів [Електронний ресурс] – Режим доступу до ресурсу: <https://helsi.me/>.
2. Сервіс запису до лікаря на прийом в Києві онлайн [Електронний ресурс] – Режим доступу до ресурсу: <https://doc.ua/>.
3. Клієнт-серверна архітектура та ролі серверів. [Електронний ресурс] / Іван Змерзлий // Medium. – 2017. – Режим доступу до ресурсу: <https://medium.com/@IvanZmerzlyi>.
4. Zosym M. Діаграми послідовностей (Sequence Diagrams) [Електронний ресурс] / Махум Zosym // Махум Zosym. – 2022. – Режим доступу до ресурсу: <https://www.maxzosim.com/sequence-diagrams/>.
5. Best Programming Languages for Web Development [Електронний ресурс] // MOOC.org. – 2021. – Режим доступу до ресурсу: <https://www.mooc.org/blog/best-programming-languages-for-web-development>.
6. Сучасний підручник з JavaScript [Електронний ресурс] // Javascript.info – Режим доступу до ресурсу: <https://uk.javascript.info/>.
7. Веб-фреймворк Express (Node.js/JavaScript) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs.
8. Introduction to MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/manual/introduction/>.
9. Tutorial: Intro to React [Електронний ресурс] – Режим доступу до ресурсу: <https://legacy.reactjs.org/tutorial/tutorial.html>.
10. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>.
11. Getting Started with Redux [Електронний ресурс] – Режим доступу до ресурсу: <https://redux.js.org/introduction/getting-started>.

12. Introduction to Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/introduction-to-tailwind-css/>.
13. Getting Started [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>.
14. React Flux Concept [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/react-flux-concept>.
15. Правила техніки безпеки при роботі за комп'ютером [Електронний ресурс] – Режим доступу до ресурсу: <http://yu.mk.ua/print/news/19394>.
16. ДБН В.2.5-28 : 2018. „Природне і штучне освітлення” – К.: Мінрегіон України, 2018. 133 с.
17. Грибан В.Г., Негодченко О.В. ОГрибан В.Г., Негодченко О.В. Охорона праці. – К.: Центр учбової літератури, 2009. 209 с. охорона праці. – К.: Центр учбової літератури, 2009. 209 с.
18. Березюк О. В. Долікарська допомога при ураженні електричним струмом [Електронний ресурс] / О. В. Березюк, М. С. Лемешев – Режим доступу до ресурсу: https://web.posibnyky.vntu.edu.ua/fmbt/berezyuk_bezpeka_zhittyediyalnosti/76.htm.
19. П. Б. Волянський, С. О. Гур'єв. Домедична допомога на місці події. Видавничий дім «Гельветика», 2020. 221 с.

ДОДАТКИ

ДОДАТОК А

Лістинг коду розробленого веб-сервісу

Реалізація сервера наведена в наступних лістингах 1 – 13.

Лістинг 1 – Файл PatientItem.jsx

```
export const PatientItem=({patient})=>{
  //console.log(patient.newPatient.patientName)
  return<div
    className='flex flex-col basis-1/4 flex-grow'
  >
    <div className="flex justify-between items-center pt-1">
      <div className='text-white text-xl '> Пациент:
{patient?.newPatient.patientName}</div>
      <div className='text-white text-xl '> Лікар:
{patient?.newPatient.doctorName}</div>
      <div className="flex gap-8 items-center justify-center
mt-4">
        <NavLink to={'/medicalCard'}>
          <button className='flex justify-center items-center
bg-green-800 text-lg text-white rounded-sm py-2 px-4'
            >
            Мед. карта
          </button>
        </NavLink>
        <button className='flex justify-center items-center
bg-red-700 text-lg text-white rounded-sm py-2 px-4'
            >
            Видалити
          </button>
        </div>
      </div>
    </div>
  </div>
};
```

ЛІСТИНГ 2 – Файл AddAppointment.jsx

```

export const AddAppointment = ()=>{

  return (
    <div className="max-w-md mx-auto">
      <form className="bg-white shadow-md rounded px-8 pt-6
pb-8 mb-4">
        <h2 className="text-2xl font-bold mb-6">Запис на
консультацію</h2>
        <div className="mb-4">
          <label className="block text-gray-700 font-bold
mb-2" htmlFor="date">
            Дата
          </label>
          <input
            className="appearance-none border rounded w-
full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
            id="date"
            name="date"
            type="date"

          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700 font-bold
mb-2" htmlFor="time">
            Час
          </label>
          <input
            className="appearance-none border rounded w-
full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
            id="time"
            name="time"
            type="time"

          />
        </div>
      </form>
    </div>
  )
}

```

```

        <div className="mb-6">
            <label className="block text-gray-700 font-bold
mb-2" htmlFor="doctor">
                Лікар
            </label>
            <input
                className="appearance-none border rounded w-
full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
                id="doctor"
                name="doctor"
                type="text"

            />
        </div>
        <div className="flex items-center justify-center">
            <button
                className="bg-blue-500 hover:bg-blue-700
text-white font-bold py-2 px-4 rounded focus:outline-none
focus:shadow-outline"
                type="submit"
            >
                Записатись
            </button>
        </div>
    </form>
</div>
);
}

```

Лістинг 3 – Файл AddPatientPage.jsx

```

export const AddPatientPage=()=>{
    const [patientName,setPatientName]=useState('')
    const [age,setAge]=useState('')
    const [doctorName,setDoctorName]=useState('')
    const {loading}= useSelector((state)=>state.patient)
    const navigate = useNavigate()
    const dispatch = useDispatch()

```

```

useEffect(()=>{
  if(loading){
    toast("Додано пацієнта")
  }
}, [loading])

const submitHandler = ()=>{
  try{
    const data = new FormData()
    data.append('patientName',patientName)
    data.append('age',age)
    data.append('doctorName',doctorName)

    //Перетворення FormData в JSON об'єкт
    const jsonObject = {};
    for (const [key, value] of data.entries()) {
      jsonObject[key] = value;
    }
    dispatch(createPatient(jsonObject))
    setPatientName('')
    setAge('')
    setDoctorName('')
    navigate('/patients')
  }catch(error){
    console.log(error)
  }
}

const clearFormHandler=()=>{
  setPatientName('')
  setAge('')
  setDoctorName('')
}
return <form
  className='w-1/3 mx-auto py-10'
  onSubmit={ (e)=>e.preventDefault() }

>
  <label className='text-lg text-white opacity-70'>
    Ім'я пацієнта:
    <input type="text"
      value={patientName}
      onChange={ (e)=>setPatientName(e.target.value) }
      placeholder="Ім'я"
      className='mt-1 text-black w-full rounded-lg bg-gray-400

```

```

border py-1 px-2 text-xs outline-none placeholder:text-gray-700' />
</label>
<label className='text-lg text-white opacity-70'>
  Вік пацієнта:
  <input type="number"
    value={age}
    onChange={ (e) => setAge (e.target.value) }
    placeholder="Вік"
    className='mt-1 text-black w-full rounded-lg bg-
gray-400 border py-1 px-2 text-xs outline-none placeholder:text-
gray-700' />
</label>
<label className='text-lg text-white opacity-70'>
  Ім'я лікаря:
  <input type="text"
    value={doctorName}
    onChange={ (e) => setDoctorName (e.target.value) }
    placeholder="Ім'я лікаря"
    className='mt-1 text-black w-full rounded-lg bg-
gray-400 border py-1 px-2 text-xs outline-none placeholder:text-
gray-700' />
</label>
<div className="flex gap-8 items-center justify-center mt-
4">
  <button className='flex justify-center items-center bg-
gray-800 text-lg text-white rounded-sm py-2 px-4'
    onClick={submitHandler}
  >
    Додати
  </button>
  <button className='flex justify-center items-center bg-
red-700 text-lg text-white rounded-sm py-2 px-4'
    onClick={clearFormHandler}
  >
    Скасувати
  </button>
</div>
</form>
}

```

Лістинг 4 – Файл Appointment.jsx

```

const Appointment = () => {
  const consultations = [

```

```

    { id: 1, date: '2023-06-05', time: '10:00 AM', doctor: 'Dr
Hause', patient: 'Стець' },
    { id: 2, date: '2023-06-06', time: '11:30 AM', doctor:
'Юлія', patient: 'Москалик' },
    { id: 3, date: '2023-06-07', time: '2:15 PM', doctor:
'Запорожець', patient: 'Новосад' },
  ];

  return (
    <div className="max-w-lg mx-auto text-white">
      <table className="table-auto w-full">
        <thead>
          <tr>
            <th className="px-4 py-2">Дата</th>
            <th className="px-4 py-2">Час</th>

            <th className="px-4 py-2">Пацієнт</th>
          </tr>
        </thead>
        <tbody>
          {consultations.map((consultation) => (
            <tr key={consultation.id}>
              <td className="border px-4 py-
2">{consultation.date}</td>
              <td className="border px-4 py-
2">{consultation.time}</td>

              <td className="border px-4 py-
2">{consultation.patient}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default Appointment;

```

Лістинг 5 – Файл LoginPage.jsx

```

export const LoginPage=()=>{
  const [username,setUserName] = useState('')

```



```

const [password, setPassword] = useState('')

const {status}= useSelector((state)=>state.auth ||{})
const isAuth = useSelector(checkIsAuth)
const dispatch = useDispatch()
const navigate = useNavigate()

useEffect(()=>{
  if(status){
    toast(status)
  }
  if(isAuth)navigate('/')
},[status, isAuth, navigate])

const handleSubmit=()=>{
  try{
    dispatch(loginUser({username,password}))
  }
  catch (error){

    console.log(error)
  }
}

return <form onSubmit={e=>e.preventDefault()}
className="w-1/4 h-60 mx-auto mt-40"
>
  <h1 className='text-xl text-white text-
center'>Авторизація</h1>
  <label className='text-lg text-gray-400'>
    Логін:
    <input type="text"
      value={username}
      onChange={e=>setUserName(e.target.value)}
      placeholder='Username'
      className='mt-1 text-black w-full rounded-lg bg-
gray-400 border py-1 px-2 text-xs outline-none placeholder:text-
gray-700'
    />
  </label>
  <label className='text-lg text-gray-400'>
    Пароль:
    <input type="password"
      value={password}
      onChange={e=>setPassword(e.target.value)}

```

```

        placeholder='Password'
        className='mt-1 text-black w-full rounded-lg bg-
gray-400 border py-1 px-2 text-xs outline-none placeholder:text-
gray-700'
      />
    </label>

    <div className="flex gap-8 justify-center mt-4">
      <button type='submit'
        onClick={handleSubmit}
        className='flex justify-center items-center bg-gray-800
text-lg text-white rounded-sm py-w px-4'
      >
        Увійти
      </button>
      <Link to='/register'
        className='flex justify-center items-center bg-
gray-800 text-lg text-white py-w px-4'>
        Реєстрація
      </Link>
    </div>
  </form>
}

```

ЛІСТИНГ 6 – Файл MainPage.jsx

```

export const MainPage=()=>{
  return <div>
    <div className="flex justify-center items-center mb-10 ">

      <form className="w-64 text-white text-lg">
        Пошук пацієнта:
        <div className="flex items-center border-b border-b-
2 border-teal-500 py-2">
          <input
            className="appearance-none bg-transparent
border-none w-full text-gray-900 mr-3 py-1 px-2 leading-tight
focus:outline-none"
            type="text"
            placeholder="Введіть фамілію..."
          />
          <button
            className="flex-shrink-0 bg-teal-500
hover:bg-teal-700 border-teal-500 hover:border-teal-700 text-sm

```

```

border-4 text-white py-1 px-2 rounded"
      type="button"
    >
      Пошук
    </button>
  </div>
</form>
</div>
<div className="mx-auto max-w-2xl">
  <div className="grid grid-cols-2 gap-4">
    <NavLink to={'appointment'}>
      <button className="bg-blue-500 hover:bg-blue-700 text-lg
text-white py-2 px-4 rounded">
        Заплановані прийоми
      </button>
    </NavLink>
    <button className="bg-green-400 hover:bg-green-700 text-
lg text-white py-2 px-4 rounded">
      Створити прийом
    </button>
  </div>
</div>
</div>
}

```

Лістинг 7 – Файл MedicalCard.jsx

```

export const MedicalCard={()=>{

  return <div className="max-w-lg mx-auto bg-white shadow-md
rounded-lg overflow-hidden">
    <div className="px-6 py-4">
      <h2 className="text-2xl font-bold mb-2">Медична карта
пацієнта</h2>
      <div className="mb-4">
        <label className="text-gray-500 font-bold mb-2"
htmlFor="name">
          Ім'я та прізвище:
        </label>
        <p className="text-gray-900 font-bold mb-2"
htmlFor="name">
          Москалик Василь
        </p>
      </div>
    </div>
  </div>
}

```

```

        <div className="mb-4">
            <label className="text-gray-500 font-bold mb-2"
htmlFor="diagnosis">
                Діагноз:
            </label>
            <p className="text-gray-900 font-bold mb-2"
htmlFor="name">
                Запалення легенів
            </p>
        </div>
        <div className="mb-4">
            <label className="text-gray-500 font-bold mb-2"
htmlFor="instructions">
                Інструкції щодо лікування
            </label>
            <p className="text-gray-900 font-bold mb-2"
htmlFor="name">
                Інгаляції 2 рази в тиждень <br/>
                Імет 1 таблетка (при підвищені температури)<br/>
                Сироп Евкалор<br/>
                Чай з малини<br/>
            </p>
            <NavLink to={'edit'}>
                <button className="bg-green-500 hover:bg-green-700
text-lg text-white py-2 px-4 rounded">
                    Редагувати
                </button>
            </NavLink>
        </div>
    </div>
</div>
}

```

Лістинг 8 – Файл PatientsPage.jsx

```

export const PatientsPage=()=>{

    const dispatch = useDispatch()
    const {patients} = useSelector((state)=> state.patient)
    console.log(patients)
    useEffect(()=>{
        dispatch(getAllPatients)
    })
}

```

```

    },[dispatch])

    if(!patients.length){
      return (
        <div className='text-xl text-center text-white py-10'>
          Пацієнтів не існує
        </div>
      )
    }

    return <div className='max-w-[900px] mx-auto py-5'>
      <div className="flex justify-between gap-4">
        <div className="flex text-2xl text-white flex-col gap-5
basis-4/5">
          Пацієнти
          {
            patients?.map((patient,idx)=>(
              <PatientItem key={idx} patient={patient}/>
            ))
          }
          <PatientItem/>

        </div>
      </div>
    </div>
  }
}

```

ЛІСТИНГ 9 – Файл RegisterPage.jsx

```

export const RegisterPage=()=>{
  const [username,setUserName] = useState('')
  const [password,setPassword] = useState('')
  const {status}= useSelector((state)=>state.auth ||{})
  const dispatch = useDispatch()
  useEffect(()=>{
    if(status){
      toast(status)
    }
  }, [status])

  const handleSubmit=()=>{
    try{
      dispatch(registerUser({username,password}))
    }
  }
}

```

```

        setPassword('')
        setUsername('')
    }
    catch (error){

        console.log(error)
    }
}

return (<form onSubmit={e=>e.preventDefault()}
        className="w-1/4 h-60 mx-auto mt-40"
>
    <h1 className='text-xl text-white text-center'>Регістрація</h1>
    <label className='text-lg text-gray-400'>
        Ім'я:
        <input type="text"
            value={username}
            onChange={e=>setUsername(e.target.value)}
            placeholder='Username'
            className='mt-1 text-black w-full rounded-lg bg-gray-400 border py-1 px-2 text-xs outline-none placeholder:text-gray-700'
        />
    </label>
    <label className='text-lg text-gray-400 mt-2'>
        Пароль:
        <input type="password"
            value={password}
            onChange={e=>setPassword(e.target.value)}
            placeholder='Password'
            className='mt-1 text-black w-full rounded-lg bg-gray-400 border py-1 px-2 text-xs outline-none placeholder:text-gray-700'
        />
    </label>
    <label className="inline-flex items-center">
        <input
            type="checkbox"
            className="form-checkbox h-5 w-5 text-indigo-600"
            value="лікар"

        />
        <span className="ml-2">лікар</span>
    </label>

```

```

<label className="inline-flex items-center ml-6">
  <input
    type="checkbox"
    className="form-checkbox h-5 w-5 text-indigo-600"
    value="пацієнт"
  />
  <span className="ml-2">пацієнт</span>
</label>

<div className="flex gap-8 justify-center mt-4">
  <button type='submit'
    onClick={handleSubmit}
    className='flex justify-center items-center bg-
gray-600 text-lg text-white rounded-sm py-w px-4'
  >
    Підтвердити
  </button>
  <Link to='/login'
    className='flex justify-center items-center text-
lg text-white py-w px-4'>
    Вже є акаунт?
  </Link>
</div>
</form>)
}

```

Лістинг 10 – Файл authSlice.js

```

const initialState={
  user:null,
  token: null,
  isLoading: false,
  status:null,
}

export const registerUser=createAsyncThunk(
  'auth/registerUser',
  async ({username,password})=>{
    try {
      //Відправка на сервер паролю та логіна
      const {data} = await axios.post('/auth/register',{
        username,
        password,
      })
    }
  }
)

```

```

        if(data.token){
            window.localStorage.setItem('token',data.token) }
        return data
    }catch (error){
        console.log(error) }
    },
)

export const loginUser=createAsyncThunk(
    'auth/loginUser',
    async ({username,password})=>{
        try {
            const {data} = await axios.post('/auth/login',{
                username,
                password,
            })
            if(data.token){
                window.localStorage.setItem('token',data.token)
            }
            return data

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

export const getMe=createAsyncThunk(
    'auth/me',
    async ()=>{
        try {
            const {data} = await axios.post('/auth/me')
            return data

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

export const authSlice=createSlice({
    name:'auth',
    initialState,

```



```

reducers: {
  logout: (state) => {
    state.user = null
    state.token = null
    state.isLoading = false
    state.status = null
  }
},
extraReducers: {
  //Register user
  [registerUser.pending]: (state) => {
    state.isLoading = true
    state.status = null
  },
  [registerUser.fulfilled]: (state, action) => {
    state.isLoading = false
    state.status = action.payload.message
    state.user = action.payload.user
    state.token = action.payload.token
  },
  [registerUser.rejected]: (state, action) => {
    state.status = action.payload.message
    state.isLoading = false
  },
  //Login user
  [loginUser.pending]: (state) => {
    state.isLoading = true
    state.status = null
  },
  [loginUser.fulfilled]: (state, action) => {
    state.isLoading = false
    state.status = action.payload.message
    state.user = action.payload.user
    state.token = action.payload.token
  },
  [loginUser.rejected]: (state, action) => {
    state.status = action.payload.message
    state.isLoading = false
  },
  //Get me - перевірка авторизації
  [getMe.pending]: (state) => {
    state.isLoading = true
    state.status = null
  }
}

```

```

    },
    [getMe.fulfilled]: (state, action) => {
      state.isLoading = false
      state.status = null
      state.user = action.payload?.user
      state.token = action.payload?.token
    },
    [getMe.rejected]: (state, action) => {
      state.status = action.payload.message
      state.isLoading = false
    },
  },
})

export const checkIsAuth = (state) => Boolean(state.auth.token)

export const { logout } = authSlice.actions
export default authSlice.reducer

```

ЛІСТИНГ 11 – Файл PatientSlice.js

```

const initialState = {
  patients: [],
  loading: false,
}

export const createPatient =
createAsyncThunk('patient/createPatient', async (params) => {
  try {
    const {data} = await axios.post('/patient', params)
    return data
  } catch (error) {
    console.log(error)
  }
})

export const getAllPatients =
createAsyncThunk('patient/getAllPatients', async () => {
  try {
    const {data} = await axios.get('/patient')
    return data
  } catch (error) {

```

```

    }
  })

export const patientSlice = createSlice({
  name: 'patient',
  initialState,
  reducers: {},
  extraReducers: {
    //Create Patient
    [createPatient.pending]: (state) => {
      state.loading = true
    },
    [createPatient.fulfilled]: (state, action) => {
      state.loading = false
      state.patients.push(action.payload)
    },
    [createPatient.rejected]: (state) => {
      state.loading = false
    },
    //Get All Patients
    [getAllPatients.pending]: (state) => {
      state.loading = true
    },
    [getAllPatients.fulfilled]: (state, action) => {
      state.loading = false
      state.patients = action.payload.patients
    },
    [getAllPatients.rejected]: (state) => {
      state.loading = false
    },
  },
})

export default patientSlice.reducer

```

Лістинг 12 – Файл App.js

```

function App() {
  const dispatch = useDispatch()

  useEffect(() => {

```

```

        dispatch(getMe())
    })
    return (
    <Layout>
      <Routes>
        <Route path="/" element={<MainPage />} />
        <Route path='patients' element={<PatientsPage />} />
        <Route path=':id' element={<PatientPage />} />
        <Route path=':medicalCard/edit' element={<EditPatientPage
/>} />
        <Route path='new' element={<AddPatientPage />} />
        <Route path='register' element={<RegisterPage />} />
        <Route path='login' element={<LoginPage />} />
        <Route path='medicalCard' element={<MedicalCard />} />
        <Route path='appointment' element={<Appointment />} />
        <Route path='patientCabinet/addAppointment'
element={<AddAppointment />} />
        <Route path='patientCabinet' element={<PatientCabinet />}
/>
        <Route path='patientCabinet/medicalCard'
element={<MedicalCard />} />
      </Routes>

      <ToastContainer position='bottom-right' />
    </Layout>
  )
}

export default App;

```

Лістинг 13 – Файл index.js

```

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <Provider store={store}>
      <App />
    </Provider>
  </BrowserRouter>
)

```

Код який реалізовує роботу серверної частини додатку наведений в наступних лістингах 14 – 21 .

Лістинг 14 – Файл auth.js

```
import User from '../models/User.js'
import bcrypt from 'bcryptjs'
import jwt from 'jsonwebtoken'

//Register user
export const register=async (req,res)=>{
  try {
    const {username,password}=req.body
    const isUsed = await User.findOne({username})
    if(isUsed){
      return res.json({message:'Даний username зайнятий',})}
    const salt = bcrypt.genSaltSync(10)
    const hash=bcrypt.hashSync(password,salt)
    const newUser=new User({
      username,
      password: hash,
    })
    const token = jwt.sign({id: newUser._id},{
      process.env.JWT_SECRET, {expiresIn:'30d'},)
    await newUser.save()
    res.json({newUser,message:'Реєстрація успішна'})
  }
  catch(error){res.json({message:'Помилка при створені користувача'})}
}

//Login user
export const login=async (req,res)=>{
  try {
    const {username, password}=req.body
    const user = await User.findOne({username})
    if(!user){
      return res.json({message:'Даного користувача не існує',})
    }

    const isPasswordCorrect = await
    bcrypt.compare(password,user.password)
```

```

    if(!isPasswordCorrect){
        return res.json({
            message: 'Неправильний пароль',
        })
    }

    const token = jwt.sign({
        id: user._id,
    }, process.env.JWT_SECRET,
        {expiresIn: '30d'},
    )

    res.json({
        token, user, message: 'Ви зайшли в систему',
    })

}
catch(error){
    res.json({ message: 'Помилка при авторизації'})

}
}
//Get me
export const getMe=async (req,res)=>{
    try{
        const user= await User.findById(req.userId)

        if(!user){
            return res.json({message: 'Даного користувача не існує',})
        }

        const token = jwt.sign({
            id: user._id,
        }, process.env.JWT_SECRET,
            {expiresIn: '30d'},
        )

        res.json({
            user, token,
        })

    }catch (error){
        res.json({message: 'Немає доступу'})
    }
}

```

```

    }
  }
}

```

Лістинг 15 – Файл patients.js

```

//Create Patient
export const createPatient = async (req,res)=>{
  try{
    const {patientName,age,doctorName} = req.body
    const newPatient = new Patient({patientName, age,
doctorName,})
    await newPatient.save()
    res.json({newPatient, message:'Добавлено пацієнта',})
  }catch(error)
  {
    res.json({message:'Помилка при створені пацієнта'})
  }
}

//Get All Patients
export const getAll=async(req,res)=>{
  try{
    const patients = await Patient.find().sort('-createdAt')
    if(!patients){
      return res.json({message:'Пацієнтів немає'})
    }
    res.json({patients})
  }catch(error){
    res.json({message:'Щось пішло не так!'})
  }
}

```

Лістинг 16 – Файл Patients.js

```

const PatientSchema = new mongoose.Schema({
  patientName: {type: String, required: true},
  //diagnoz: {type: String, required: true},
  age: {type: Number, required: true},
  doctorName: {type: String, required: true},
},
  {timestamps: true},
)

```

```
export default mongoose.model('Patient', PatientSchema)
```

Лістинг 17 – Файл User.js

```
const UserSchema=new mongoose.Schema(
  {
    username:{
      type:String,
      required:true,
      unique:true,
    },
    password:{
      type:String,
      required: true,
    }
  },
  {timestamps:true},
)

export default mongoose.model('User',UserSchema)
```

Лістинг 18 – Файл auth.js

```
const router = new Router()

//Register
router.post('/register',register)
//Login
router.post('/login',login)
//Get me
router.post('/me',checkAuth,getMe)

export default router
```

Лістинг 19 – Файл patients.js

```
const router = new Router()

//Create Patient
//http://localhost:3002/api/patients
router.post('/',createPatient)
```



```
//Get All Posts
//http://localhost:3002/api/patients
router.get('/',getAll)

export default router
```

Лістинг 20 – Файл checkAuth.js

```
export const checkAuth=(req,res,next)=>{
  const token = (req.headers.authorization ||
  '').replace(/Bearer\s?/, '')

  if(token) {
    try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET)

      req.userId=decoded.id

      next()
    } catch (error) {
      return res.json({
        message: 'Немає доступу.',
      })
    }
  }
  else {
    return res.json({
      message:'Немає доступу.',
    })
  }
}
```

Лістинг 21 – Файл index.js

```
const app = express()
dotenv.config()

//Constants
```

```

const PORT=process.env.PORT ||3001
const DB_USER=process.env.DB_USER
const DB_PASSWORD=process.env.DB_PASSWORD
const DB_NAME=process.env.DB_NAME

// Middleware
app.use(cors())
app.use(express.json())

// Routes
app.use('/api/auth',authRoute)
app.use('/api/patient', patientRoute)
async function start(){
  try{
    await mongoose.connect(

`mongodb+srv://${DB_USER}:${DB_PASSWORD}@cluster1.cn5uxdq.mongodb.net/${DB_NAME}?retryWrites=true&w=majority`
    )

    app.listen(PORT,()=>console.log(`Server started on port:
${PORT}`))
    }catch(error){
      console.log(error)
    }
  }
}
start()

```