

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

Розробка сайту “Лікарня”

Виконав студент групи ІПЗ-19-1
ГОПЧУК Олександр Ярославович

Керівник роботи:
ГРИШКУН Євгеній Олександрович

Консультант:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2023

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

«ЗАТВЕРДЖУЮ»

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

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГОПЧУК Олександр Ярославович**

Керівник роботи: **ГРИШКУН Євгеній Олександрович**

Тема роботи: **Розробка сайту “Лікарня”,**

затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №70с

Вихідні дані для роботи: **методи та засоби розробки веб-додатків, аналіз проблематики додатку сайту лікарні, використання мови програмування JavaScript, фреймворку Express, бібліотеки React та Redux для розробки веб-сайту.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С. М., Гришкун Є. О.	16.02.2023р.	16.02.2023р.
2	Кравченко С. М., Гришкун Є. О.	06.03.2023р.	06.03.2023р.
3	Кравченко С. М., Гришкун Є. О.	05.05.2023р.	05.05.2023р.

РЕФЕРАТ

Кваліфікаційна робота складається з веб-додатку лікарні та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 80 сторінок та 53 ілюстрацій.

Метою даної роботи є створення веб-додатку, який буде забезпечувати доступ до медичних послуг та інформації про роботу медичного закладу. Головна мета полягає у створенні зручного та простого у використанні інструменту, який дозволить користувачам легко скористатись послугами лікарні.

У даній роботі було визначено основні завдання, які передбачали розробку інформаційної системи. Проведено детальний аналіз аналогічних програмних продуктів, а також ретельно обґрунтовано архітектуру проекту та надано алгоритми його функціонування. Крім цього, були презентовані приклади роботи програми та проведено тестування, щоб перевірити її функціональність та надійність.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ЛІКАРНЯ, ПОСЛУГИ, NODE, REACT, PRISMA.

					ІПЗ.КР.Б – 121 – 23 – ПЗ				
Зм.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Гопчук О.Я.			Розробка сайту «Лікарня» Пояснювальна записка		Літ.	Арк.	Аркуші
Керівник		Гришкун Є.О.						2	80
				Житомирська політехніка, група ІПЗ-19-1					
Рецензент		Чижморя О.Г.							
Зав. каф.		Морозов А.В.							

ABSTRACT

The qualification work consists of a hospital web application and an explanatory note. The explanatory note to the qualification work contains 80 pages and 53 illustrations.

The purpose of this work is to create a web application that will provide access to medical services and information about the work of a medical institution. The main goal is to create a convenient and easy-to-use tool that will allow users to easily use the services of the hospital.

In this paper, we have identified the main tasks that involved the development of the information system. A detailed analysis of similar software products was conducted, and the project architecture was thoroughly substantiated and algorithms for its functioning were provided. In addition, examples of the program's operation were presented and testing was conducted to verify its functionality and reliability.

KEYWORDS: WEB APPLICATION, HOSPITAL, SERVICES, NODE, REACT, PRISMA.

					ИПЗ.КР.Б – 121 – 23 – ИПЗ	
						3

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-САЙТУ ЛІКАРНІ.....	7
1.1. Постановка задачі	7
1.2. Аналіз аналогів програмного продукту.....	10
1.3. Вибір архітектури інформаційної системи.....	13
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	16
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-САЙТУ ЛІКАРНІ.....	19
2.1. Визначення варіантів використання та структури системи.....	19
2.2. Розробка бази даних системи.....	27
2.3. Проєктування та реалізація алгоритмів роботи системи.....	31
2.4. Реалізація функціоналу веб-сайту лікарні	32
Висновки до другого розділу	37
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ	38
3.1. Порядок встановлення та налаштування системи.	38
3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з веб-сайтом лікарні.	39
3.3. Тестування роботи веб-сайту	58
Висновки до третього розділу	59
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – Create, Read, Update, Delete (операції над даними: створення, читання, оновлення, видалення).

JSON – JavaScript Object Notation (формат збереження даних)

API – Прикладний програмний інтерфейс

ORM – Object-Relational Mapping (технологія, для взаємодії з реляційними базами даних)

REST – Representational State Transfer (підхід до архітектури мережеских протоколів, які надають доступ до інформаційних ресурсів)

БД – База даних

MUI – Material UI (бібліотека React компонентів, яка використовує Material Design)

SPA – Single Page Application (патерн для створення веб-додатків)

ВСТУП

Актуальність теми розробки веб-сайту для лікарні впливає з потреби впровадження сучасних технологій в медичну сферу. Все більше людей користуються Інтернетом для отримання інформації та виконання різних завдань, і лікарні повинні відповідати цьому попиту. Веб-сайт лікарні може надати зручний і доступний канал комунікації між медичним закладом і пацієнтами.

Завдяки веб-сайту, лікарня може представити свої послуги, створити графік роботи лікарів, надати інформацію про спеціалізації та кваліфікацію медичного персоналу. Пацієнти зможуть знайти необхідну інформацію перед візитом до лікарні, що полегшить їхній досвід і допоможе в зручний спосіб спланувати свої медичні послуги.

Метою даної кваліфікаційної роботи є розробка та проектування веб-сайту лікарні, створення інструменту для забезпечення зручної взаємодії між користувачами системи та надання необхідної інформації про медичні послуги, лікарів, події, контакти та інші корисні дані.

Об'єктом дослідження є система, яка надає медичні послуги та сприяє взаємодії між лікарями та клієнтами.

Предметом дослідження є застосування сучасних веб-технологій для розробки веб-сайту лікарні.

Кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел з 20 найменувань, 3 додатків, 51 ілюстрації по тексту. Повний обсяг роботи становить 81 сторінок, з них 61 сторінок основного тексту

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-САЙТУ ЛІКАРНІ

1.1. Постановка задачі

Мета розробки полягає у створенні веб-додатку лікарні, спрямованого на спрощення процесу замовлення та обробки медичних послуг, перегляд актуальних новин та отриманні інших даних про роботу медичного закладу та лікарів. Додаток надає можливість обробки замовлення клієнта та перегляду списку прийомів у лікаря. Основна мета веб-додатку – поліпшення процесу надання медичних послуг пацієнтам.

Тож основними етапами реалізації даного програмного продукту є:

1. Аналіз вимог та проектування:

1.1. Вивчення потреб користувачів та встановлення функціональних вимог.

1.2. Розробка архітектури системи та проектування бази даних.

1.3. Визначення структури веб-сайту та розробка інтерфейсу.

2. Побудова клієнтської архітектури додатку.

2.1. Визначення концепції дизайну системи

2.2. Розробка дизайну сторінок та компонентів

2.2.1. дизайн головного меню;

2.2.2. дизайн профілю користувача;

2.2.3. дизайн сторінок з інформацією про медичний заклад, контакти

2.2.4. дизайн сторінки з новинами, послугами, прийомами, лікарями;

2.2.5. дизайн сторінок з створенням та редагуванням послуг, прийомів, замовлень, новин

2.3. Інтеграція з сервером, встановлення механізмів передачі даних між клієнтом та сервером

3. Побудова серверної архітектури додатку.

- 3.1. Проектування архітектури, визначення структури серверної частини додатку, протоколів комунікації, забезпечення безпеки
- 3.2. Розробка серверних компонентів, модулі обробки даних, API
- 3.3. Інтеграція з базою даних
4. Розробка функціоналу користувача.
 - 4.1. Розробка функціоналу для створення, редагування або видалення користувача
 - 4.2. Розробка механізму перевірки та підтвердження даних про користувача
 - 4.3. Реалізація розподілення ролей користувачів
5. Розробка систем автентифікації та реєстрації користувача
 - 5.1. Розробка форм для входу та реєстрації
 - 5.2. Розробка механізму перевірки даних користувача та управління доступом
6. Розробка функціоналу послуг
 - 6.1. Розробка сторінок та компонентів для створення, редагування, видалення послуг, пакетів послуг
 - 6.2. Розробка функціоналу для створення, редагування, видалення послуг, пакетів послуг
7. Розробка функціоналу новин
 - 7.1. Розробка сторінок та компонентів для перегляду, створення, зміни, видалення новин
 - 7.2. Розробка функціоналу для створення, редагування, видалення новин
8. Розробка функціоналу замовлень та прийомів
 - 8.1. Розробка сторінок та компонентів для перегляду, створення, зміни, видалення замовлень та прийомів
 - 8.2. Розробка функціоналу для додавання замовлень клієнтом, зміна та створення прийомів консультантом та лікарем
9. Розробка функціоналу адміністратора

9.1. Розробка сторінок та компонентів для перегляду списку користувачів, зміни ролей

9.2. Розробка функціоналу для зміни ролей користувачів та додавання, зміни та видалення спеціалізацій

10. Тестування додатку

10.1. Тестування роботи графічного інтерфейсу

10.2. Тестування роботи функціоналу

У процесі проектування було вирішено реалізувати наступні ролі користувачів: адміністратор, модератор, консультант, лікар, клієнт,

Адміністратор

1. Можливість створення спеціалізацій
2. Можливість зміни ролей

Модератор

1. Можливість створення, редагування, видалення новин
2. Можливість створення, редагування, видалення послуг, пакетів послуг

Консультант

1. Можливість створення прийомів
2. Можливість редагування, відміни замовлення

Лікар

1. Можливість перегляду списку прийомів, редагування прийому
2. Можливість зміни персональних даних

Клієнт

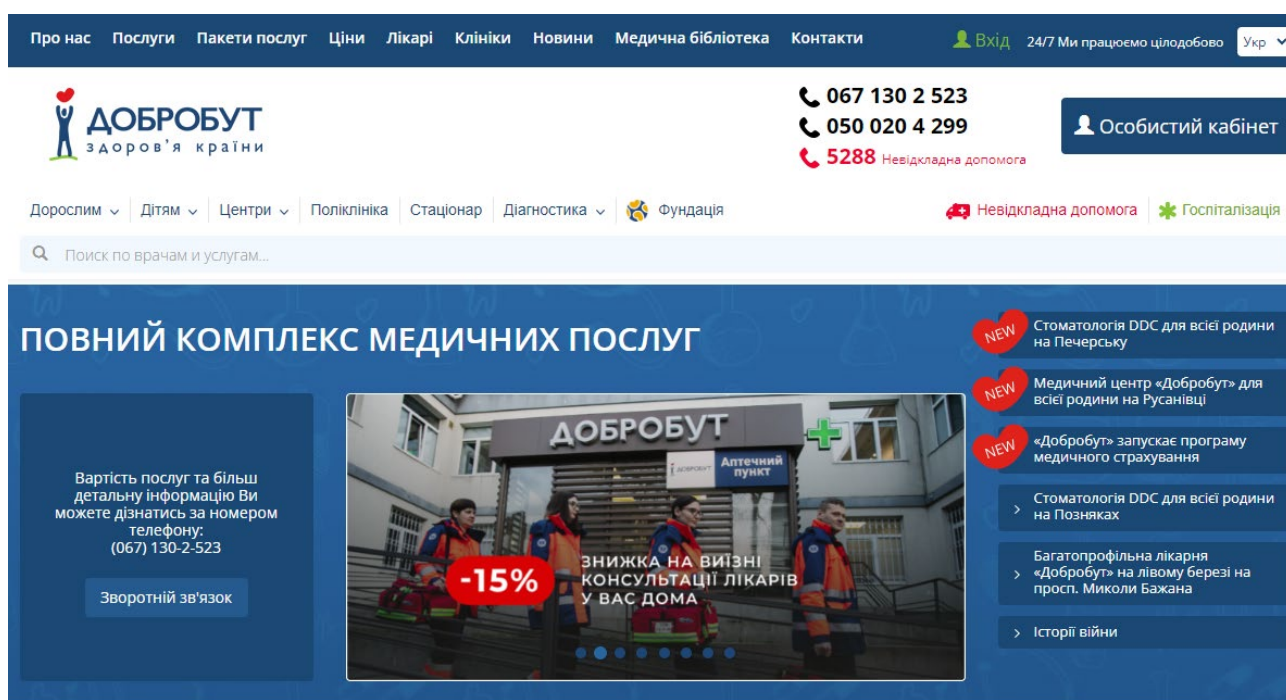
1. Можливість замовлення пакету послуг
2. Можливість перегляду списку лікарів, новин
3. Можливість зміни персональних даних
4. Можливість перегляду інформації про медичний заклад

1.2. Аналіз аналогів програмного продукту

На сьогодні існує велика кількість різних медичних закладів, які надають медичні послуги клієнтам, тому проаналізуємо деякі з них:

Медична мережа «Добробут» [1] – приватна медична клініка в Києві. У даній клініці можна переглянути список послуг, пакетів послуг, переглянути актуальні ціни на послуги, список лікарів. Є можливість перегляду актуальних новин та акцій на медичні послуги. Також є можливість перегляду карти з клініками у місті Києві. З переваг, велика кількість функцій, можливість перегляду медичної бібліотеки, детальної інформації за послугами. З недоліків, важкий у сприйнятті інтерфейс, деякі з компонентів написані іншою мовою, а не мовою інтерфейсу.

На рисунку 1.1. показано інтерфейс сайту «Добробут»



Переваги медичної мережі "Добробут"

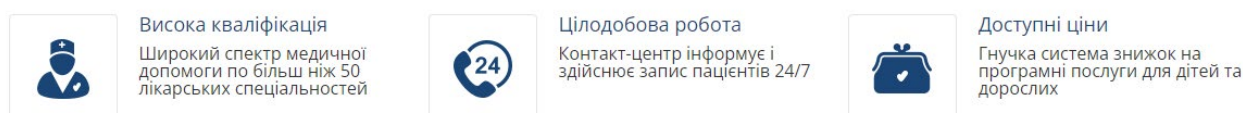


Рис. 1.1. Медична мережа «Добробут»

«Експерт» [2] – це приватна лікарня, яка надає послуги в декількох містах України. На сайті є можливість перегляду експертів, напрямків, та запису на

					ІПЗ.КР.Б – 121 – 23 – ПЗ	10

прийом по ним. Можна переглянути медичні центри, де надаються послуги, пакети послуг за типом або за назвою, ціни на послуги, інформацію про лікарню та контакти. З переваг, зручний та простий інтерфейс, широкий функціонал, можливість запису на консультацію. З недоліків, не можливість зміни мови інтерфейсу та теми оформлення.

На рисунку 1.2. показано лікарню «Експерт».

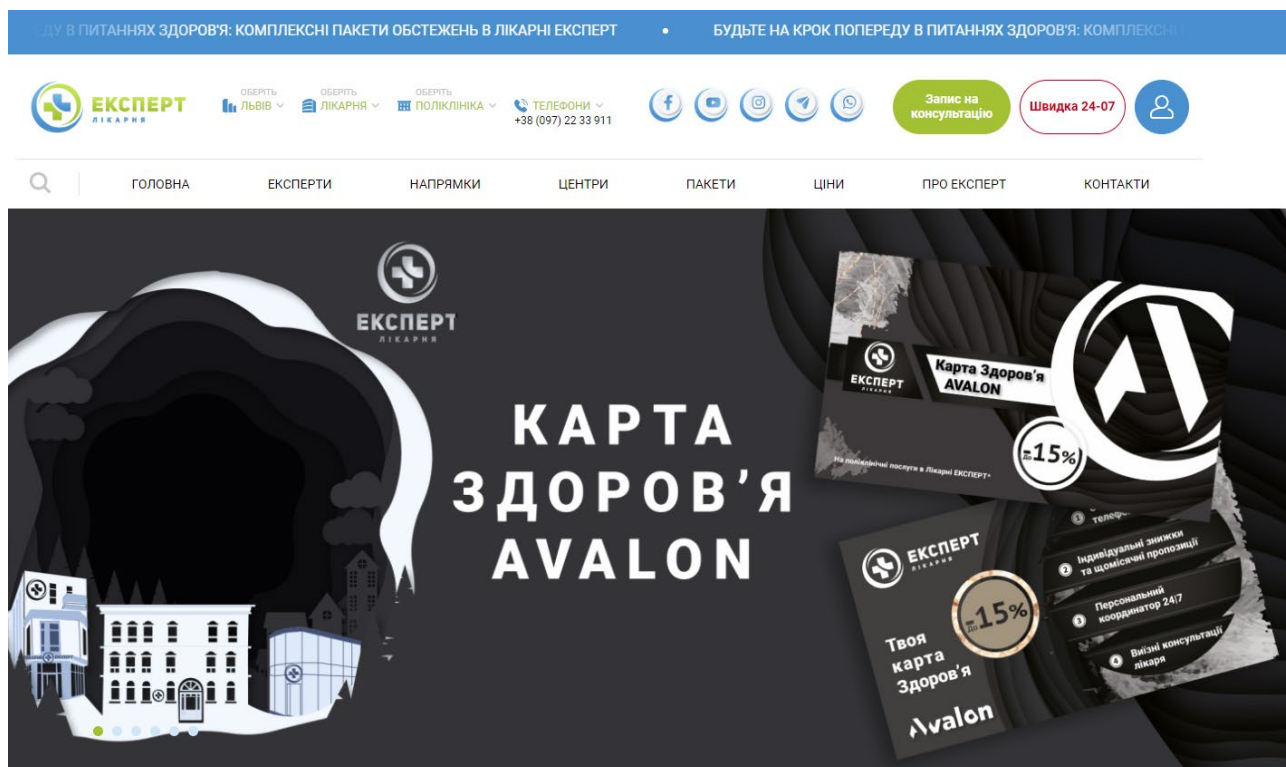


Рис. 1.2. «Експерт» лікарня

КНП СОР «Сумська обласна клінічна лікарня» [3] – обласна лікарня, має сторінки з новинами, списком відділень та інформацією про них. Також можна зв'язатися з консультантом лікарні за допомогою форми зворотнього зв'язку. Серед переваг можна зазначити доступну інформацію про роботу лікарні, існуючі відділення, лікарів, які там працюють. Натомість з недоліків це жахливий дизайн, незручний інтерфейс, різкі кольори, відсутність персонального кабінету. На рисунку 1.3. показано інтерфейс сайту КНП СОР «Сумська обласна клінічна лікарня»

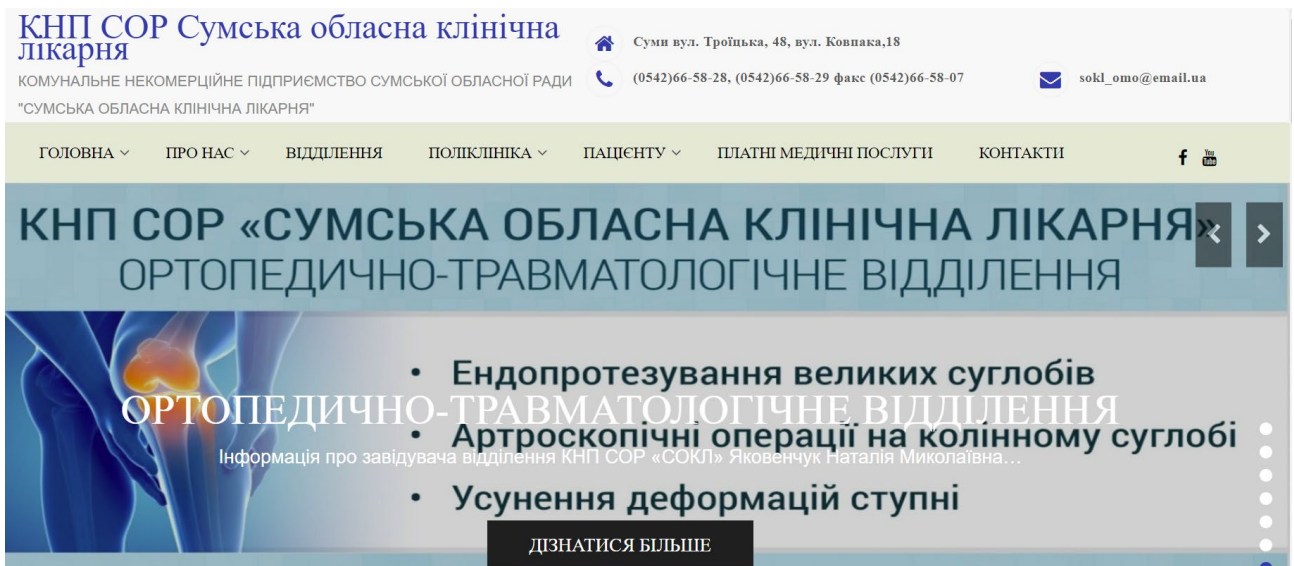


Рис. 1.3. КНП СОР «Сумська обласна клінічна лікарня»

КНП «Обласна клінічна лікарня ім. О. Ф. Гербачевського» [4] – обласна лікарня, має сторінки з новинами, інформацією про заклад, фотогалерею та відеогалерею. Серед переваг можна зазначити доступну інформацію про роботу лікарні, існуючі відділення. Натомість з недоліків – незручний дизайн, різкі кольори, відсутність персонального кабінету. На рисунку 1.4. показано інтерфейс сайту КНП «Обласна клінічна лікарня ім. О. Ф. Гербачевського»



Рис. 1.4. КНП «Обласна клінічна лікарня ім. О. Ф. Гербачевського»

Після аналізу подібних веб-додатків можна стверджувати, що у кожного з них є свої переваги та недоліки. Це дає користувачам можливість обирати, де замовляти послуги.

Враховуючи цю інформацію, виникає потреба в розробці веб-додатку з більш конкретною спрямованістю, що дозволить акцентувати увагу на основних особливостях продукту.

Отже, основними характеристиками цього продукту будуть:

- Простий та зрозумілий дизайн.
- Наявність різних послуг та пакетів послуг.
- Можливість здійснення замовлення.
- Перехід до особистого кабінету.
- Можливість перегляду списку прийомів.

1.3. Вибір архітектури інформаційної системи

Вибір архітектури додатку є надзвичайно важливим кроком у процесі розробки. Це має суттєвий вплив на продуктивність, розширюваність та підтримку проекту у майбутньому. Правильно підібрана архітектура забезпечує міцну та зручну основу для розробки, тестування, розширення та підтримки додатку на протязі всього його життєвого циклу.

У нашому випадку, програмний комплекс буде базуватися на клієнт-серверній архітектурі [5], з використанням веб-технологій, які не потребують додаткового ліцензування. Це означає, що функціонал додатку буде розподілений між клієнтською та серверною частинами. Шаблон клієнт-сервер є одним з найбільш поширених архітектурних шаблонів у розробці програмного забезпечення. Він застосовується для створення систем, в яких взаємодія між користувачем і сервером є основною та центральною.

Клієнтська частина буде взаємодіяти з користувачами через веб-браузер. Серверна частина буде відповідальна за обробку запитів користувачів, зберігання даних, бізнес-логіку та комунікацію з базою даних.

На рисунку 1.5 показано схему шаблону клієнт-сервер.

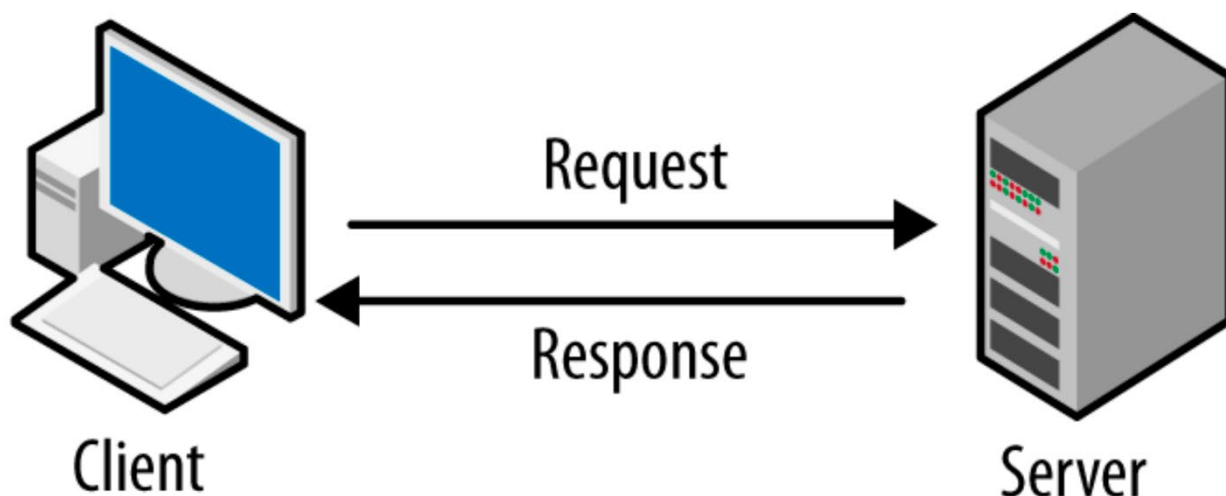


Рис. 1.5. Схема шаблону клієнт-сервер

У клієнт-серверній архітектурі система складається з двох компонентів:

- Клієнт: це програмна частина, що взаємодіє з користувачем. Клієнт може бути веб-браузером, мобільним або настільним додатком. Він відповідає за відображення інтерфейсу користувача та збір введених даних. Клієнт може надсилати запити до сервера і отримувати відповіді для відображення результатів користувачу.
- Сервер: це основна частина системи, що обробляє запити від клієнта і надає відповіді. Сервер містить бізнес-логіку, доступ до бази даних, обробку запитів та відповідей. Він може бути фізичним сервером або хмарним сервісом.

Клієнт-серверна архітектура є чудовим варіантом для реалізації веб-сайту лікарні з таких причин:

- Розподіл обов'язків: шаблон клієнт-сервер дозволяє чітко розділити функціональні обов'язки між клієнтською та серверною частинами системи. Клієнт відповідає за відображення інтерфейсу користувача, взаємодію з користувачем та передачу запитів серверу. Сервер відповідає за обробку запитів, бізнес-логіку, збереження даних та надання відповідей клієнту. Це спрощує розробку, підтримку та масштабування системи.

- **Забезпечення безпеки:** шаблон клієнт-сервер дозволяє акцентувати увагу на захисті серверної частини, де зберігаються конфіденційні дані та проводяться важливі обчислення. Це дозволяє використовувати різні механізми захисту, такі як шифрування даних, аутентифікація та авторизація користувачів, заходи проти атак та інші.

- **Розширюваність:** завдяки розділенню на клієнтську та серверну частини, легко розширювати функціональність системи, додавати нові функції або модулі. Це зменшує залежність між компонентами та спрощує розробку та підтримку. Отже, можна зробити висновок, що цей архітектурний шаблон ідеально підходить для створення обраного веб-додатку. Цей підхід забезпечить успішну реалізацію проекту та його подальшу модифікацію.

Отже, можна зробити висновок, що даний архітектурний шаблон чудово підходить для побудови обраного додатку.

Клієнтська частина буде використовувати архітектуру SPA. Single Page Application або односторінковий додаток є архітектурним підходом у веб-розробці, де весь контент і функціонал додатку завантажується одразу при першому запуску, а подальша навігація та взаємодія з додатком відбувається без перезавантаження сторінки.

Основні переваги архітектури SPA включають покращену продуктивність та швидкість завантаження, більший комфорт користувачів завдяки плавній навігації без перезавантаження сторінки, а також зниження навантаження на сервер завдяки зменшенню передачі даних. Окрім того, SPA спрощує розробку багатосторінкових додатків, дозволяючи розподіляти логіку між клієнтом і сервером.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При розробці веб-додатку потрібно обрати інструменти, які дозволять максимально спростити та пришвидшити процес розробки. Описані вимоги обмежують кількість варіантів, що можна вибрати для засобів розробки.

Оскільки на серверній частині додатку відбувається обробка запитів, бізнес-логік, збереження даних та надання відповідей клієнту, отже потрібно розумно обрати мову для написання серверу.

Для створення серверної частини додатку був вибраний фреймворк Express [12] для Node.js [11], який є одним з найбільш популярних інструментів для цієї мети. Серед переваг Express можна виділити наступне:

- Простота використання: Express є досить простим та легким у розумінні для розробки веб-додатків. Завдяки його мінімалістичній структурі можна досить швидко створювати маршрути, обробники запитів та відповідей.

- Висока швидкість: Express є досить швидким фреймворком, оскільки він базується на Node.js, який використовує неблокуючий ввід/вивід (non-blocking I/O). Це дозволяє обробляти багато запитів паралельно та ефективно використовувати ресурси сервера.

При виборі бази даних для інтернет-магазину слід звертати увагу на такі фактори як типи даних, обсяг даних, швидкодія а також навички та досвід розробників у роботі з даною базою даних, майбутні потреби.

Для фреймворку Express для Node.js найкраще підходять такі бази даних як MongoDB, PostgreSQL, MySQL, SQLite та FirebaseDB.

Було обрано систему керування базами даних MySQL [6], оскільки вона має декілька переваг порівняно з іншими системами:

- Надійність та стабільність: MySQL є дуже стабільною та надійною системою керування базами даних, яка вже довгий час використовується в багатьох веб-додатках.

– Швидкодія: MySQL має добре оптимізовану архітектуру, що дозволяє швидко виконувати запити до бази даних.

– Відкритий код: MySQL є вільним програмним забезпеченням з відкритим вихідним кодом, що дає можливість вносити зміни та доповнення до системи за потребою.

– Легка масштабованість: MySQL дозволяє легко масштабувати базу даних, що є важливим фактором для веб-додатків, які можуть зростати та змінюватися з часом.

– Для зручної взаємодії з базою даних MySQL, підходять такі ORM, як: Sequelize, Prisma ORM, TypeORM.

– Вирішено було обрати Prisma ORM [7], оскільки вона має декілька переваг порівняно з іншими рішеннями:

– Зручність у використанні: Prisma ORM надає високорівневий інтерфейс для взаємодії з базою даних, що дозволяє швидко та зручно розробляти функціонал, пов'язаний з базою даних.

– Безпека: Prisma ORM має вбудовану підтримку безпеки, що дозволяє уникнути багатьох типів атак на базу даних, таких як SQL-ін'єкції.

– Швидкодія: Prisma ORM побудована з використанням сучасних технологій та оптимізована для роботи з базою даних, що дозволяє забезпечити високу швидкодію взаємодії з базою даних.

Отже під час розробки буде використовуватися база даних MySQL, та Prisma ORM для взаємодії з базою даних.

Для написання клієнтської частини додатку було обрано бібліотеку React.js [10], оскільки він має такі переваги:

Широке співтовариство: React.js має дуже велике та активне співтовариство, що дає можливість знайти відповіді на більшість запитань та проблем, які виникають під час розробки. Будь-який розробник може швидко знайти документацію, підтримку, бібліотеки та інші ресурси, що допомагають розробляти додаток більш ефективно.

Легкість вивчення: React.js має досить просте API та добре документоване співтовариство, що дозволяє швидко вивчити бібліотеку та почати розробку. React.js використовує простий синтаксис JSX, що дозволяє розробникам легко створювати компоненти та взаємодіяти з ними.

Висока швидкодія: React.js має добре оптимізовану архітектуру, що дозволяє швидко відображати зміни на сторінці. React.js використовує Virtual DOM, що дозволяє швидко оновлювати сторінку без необхідності перезавантаження сторінки.

Також буде використовуватися бібліотека Redux [8] та Redux Toolkit [9] для взаємодії клієнтської і серверної частин та для управління отриманими даними.

Висновки до першого розділу

Після проведеного аналізу вже існуючих додатків було визначено їх переваги та недоліки, було виділено особливості, які мають бути присутні в майбутньому веб-додатку.

У результаті аналізу засобів розробки було вирішено використати фреймворк Express.js, базу даних MySQL та Prisma ORM, бібліотеки React.js, Redux, Redux Toolkit для клієнтської частини.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-САЙТУ ЛІКАРНІ

2.1. Визначення варіантів використання та структури системи.

Веб-сайт лікарні має за мету: забезпечення зручної взаємодії між користувачами системи та надання необхідної інформації про медичні послуги, лікарів, події, контакти та інші корисні дані

Вимоги користувачів.

Діаграма для гостя на рисунку 2.1.

Гість має доступ до наступних функцій:

1. До інформації з головної сторінки, контакти
2. До перегляду списку лікарів медичного закладу
3. До перегляду актуальних новин сайту
4. До перегляду медичних послуг, які надаються лікарнею
5. До реєстрації або авторизації

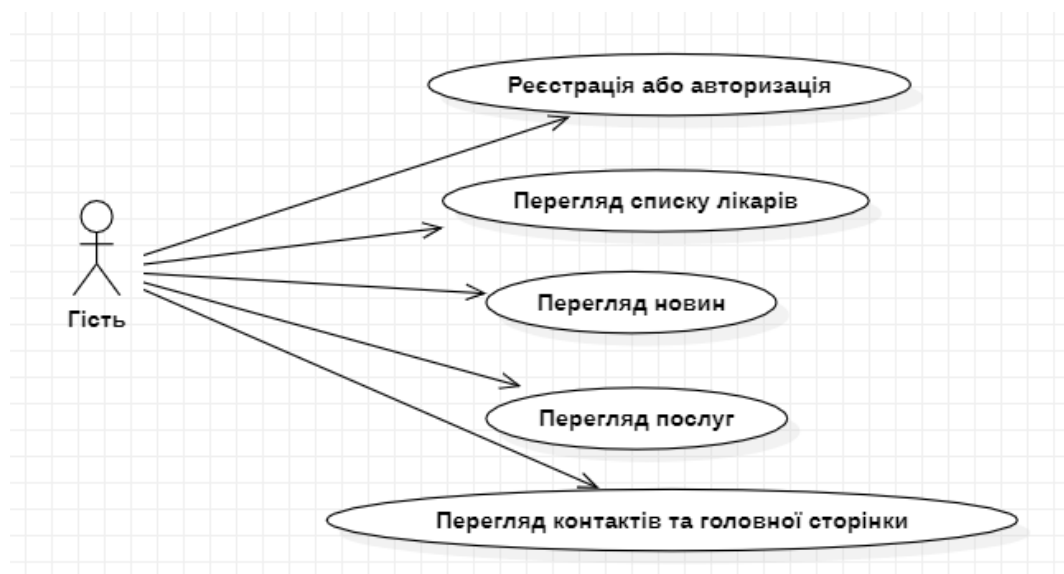


Рис. 2.1. Діаграма використання гостя

Діаграма для клієнта на рисунку 2.2.

Авторизований користувач (або клієнт) має доступ до наступних функцій:

1. До можливості замовлення послуги
2. До перегляду особистого кабінету
3. До редагування особистих даних в профілі

4. До перегляду графіку записів на прийом
5. До виходу з облікового запису
6. До перегляду списку замовлень
7. До відміни замовлення

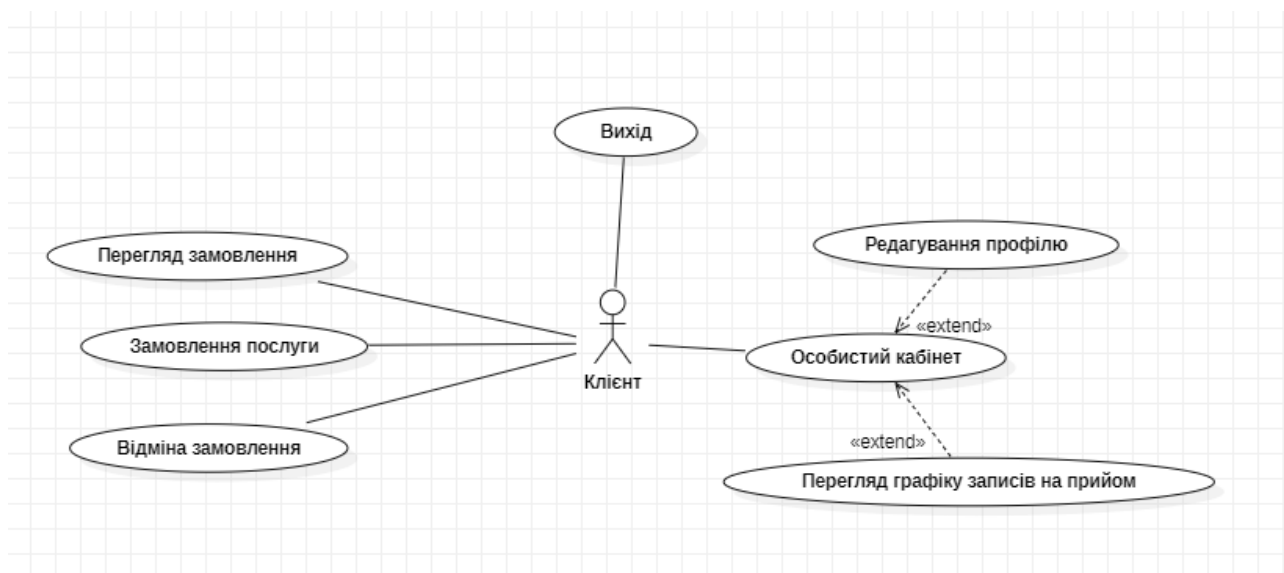


Рис. 2.2. Діаграма використання авторизованого користувача
Діаграма для модератора на рисунку 2.3.

Модератор має доступ до наступних функцій:

1. До публікації та редагування новин
2. До додавання і редагування послуг та пакетів послуг



Рис. 2.3. Діаграма використання модератора
Діаграма для адміністратора на рисунку 2.4.

Адміністратор має доступ до наступних функцій:

1. До управління ролями
2. До перегляду списку користувачів

3. До CRUD спеціалізацій лікаря



Рис. 2.4. Діаграма використання адміністратора

Діаграма для лікаря на рисунку 2.4.

Лікар має доступ до наступних функцій:

1. До перегляду графіку прийомів клієнтів
2. До перегляду прийому та його зміна або відміна
3. До зміни персональної інформації



Рис. 2.5. Діаграма використання лікаря

Діаграма для консультанта на рисунку 2.6.

Консультант має доступ до наступних функцій:

1. До перегляду списку поточних замовлень
2. До створення нового прийому у лікаря



Рис. 2.6. Діаграма використання консультанта

Функціональні вимоги:

1. Авторизація. Для користувачів необхідна можливість реєстрації.
2. Створення, редагування, перегляд та видалення послуг, пакетів послуг, новин, прийомів, замовлень.
3. Система повинна зберігати конфіденційність користувачів.
4. Система повинна коректно та автоматично відображати необхідну інформацію.

Нефункціональні вимоги:

Сприйняття

1. Сприйняття:
 - Час, потрібний для початку роботи - 2 хв.
 - Час відповіді системи для звичайних запитів не повинен перевищувати 0.500 мс, а для більш складних запитів – 1 с.
 - Інтерфейс повинен бути зрозумілим для користувачів.
2. Надійність:
 - Середній час безперервної роботи - 48 год.
 - Максимальна норма помилок та дефектів в роботі програми - 1 помилка на 2000 запитів користувача.
3. Швидкість. Сайт повинен швидко виконувати запити користувачів.
4. Можливості використання:
 - Масштабування - після додавання функціонала не повинно бути фатальних помилок.

Шляхом аналізу функціональних вимог було виявлено ключові елементи, які необхідні для розробки програмного комплексу для даної системи. На

рисунку 2.7 представлена діаграма класів, яка відображає контролери системи. Докладну структуру представлень розглянуто у третьому розділі.

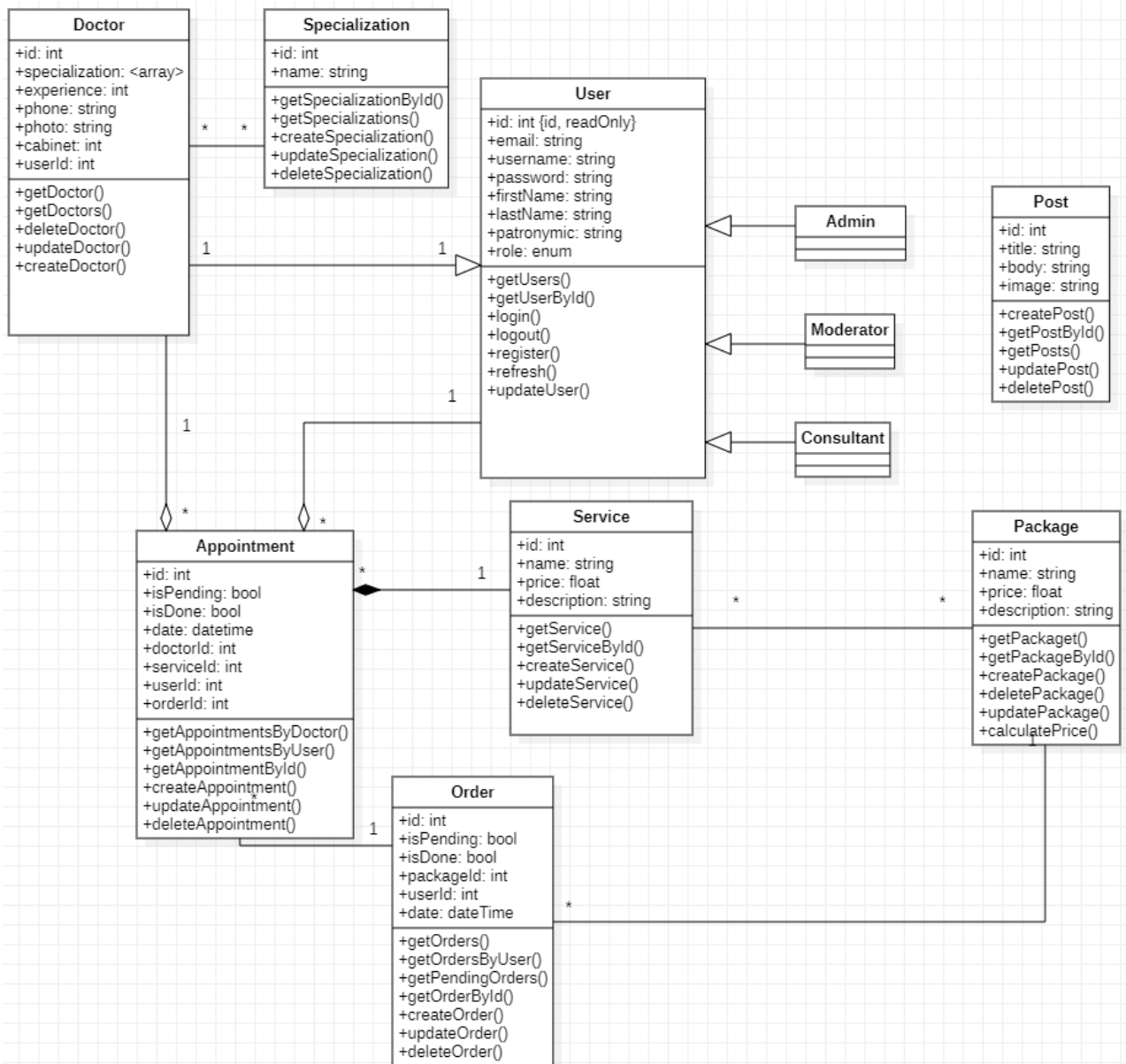


Рис. 2.7. Діаграма класів

На даному рисунку є наступні класи:

User – клас, який містить методи для роботи з користувачем.

Поля:

- id (тип int) – унікальний ідентифікатор;
- username (тип string) – ім'я користувача;
- email (тип string) – електронна пошта користувача;

- password (тип string) – пароль користувача;
- role (тип enum) – роль користувача.

Методи:

- register() – реєстрація користувача;
- login() – авторизація користувача;
- logout() – вихід з облікового запису;
- refresh() – оновлення унікального токена;
- updateUser() – редагування користувача;
- getUsers() – отримання всіх користувачів з бази даних;
- getUserById() – отримання користувача по ідентифікатору;

Post – клас, який містить методи для роботи із новинами.

Поля:

- id (тип int) – унікальний ідентифікатор;
- title (тип string) – заголовок новини;
- body (тип string) – контент новини;
- image (тип string) – прев'ю фото;

Методи:

- getPosts() – отримання всіх новин;
- getPostById() – отримання новини по ідентифікатору;
- createPost() – додати новину;
- updatePost() – редагувати новину;
- deletePost() – видалити новину;

Doctor – клас, який містить методи для роботи з лікарем.

Поля:

- id (тип int) – унікальний ідентифікатор;
- specialization (тип array) – спеціальність лікаря;
- experience (тип int) – досвід роботи;
- phone (тип string) – номер телефону;
- photo (тип string) – головне фото;

- cabinet (тип int) – номер кабінету;
- userId (тип int) – зовнішній ключ таблиці user;

Методи:

- getDoctor() – отримання лікаря;
- getDoctors() – отримання лікарів;
- deleteDoctor() – видалення лікаря;
- updateDoctor() – оновлення лікаря;
- createDoctor() – створення лікаря;

Specialization – клас, який містить методи для роботи з спеціалізацією лікаря.

Поля:

- id (тип int) – унікальний ідентифікатор;
- name (тип string) – назва спеціальність лікаря;

Методи:

- getSpecializationById() – отримання спеціальності по ідентифікатору;
- getSpecializations() – отримання спеціальностей;
- deleteSpecialization() – видалення спеціальності;
- updateSpecialization() – оновлення спеціальності;
- createSpecialization() – створення спеціальності;

Service – клас, який містить методи для роботи з послугою.

Поля:

- id (тип int) – унікальний ідентифікатор;
- name (тип string) – назва послуги;
- price(тип float) – ціна послуги;
- description(тип string) – опис послуги;

Методи:

- getService() – отримання послуги;
- getServiceById() – отримання послуги по ідентифікатору;
- createService() – створення послуги;

- updateService() – оновлення послуги;
- deleteService() – видалення послуги;

Package – клас, який містить методи для роботи з пакетами послуг.

Поля:

- id (тип int) – унікальний ідентифікатор;
- name (тип string) – назва пакету послуг;
- price(тип float) – ціна пакету послуг;
- description(тип string) – опис пакету послуг;

Методи:

- getPackaget() – отримання послуги;
- getPackageById() – отримання послуги по ідентифікатору;
- createPackage() – створення послуги;
- deletePackage() – оновлення послуги;
- updatePackage() – видалення послуги;
- calculatePrice() – обрахування ціни пакету послуг;

Order – клас, який містить методи для роботи з замовленнями.

Поля:

- id (тип int) – унікальний ідентифікатор;
- isPending (тип bool) – замовлення в процесі обробки;
- isDone (тип bool) – замовлення виконане;
- userId (тип int) – зовнішній ключ таблиці user;
- packageId(тип int) – зовнішній ключ таблиці package;
- date(тип dateTime) – бажана дата виконання замовлення;

Методи:

- getPackage() – отримання послуги;
- getPackageById() – отримання послуги по ідентифікатору;
- createPackage() – створення послуги;
- deletePackage() – оновлення послуги;
- updatePackage() – видалення послуги;

- calculatePrice() – обрахування ціни пакету послуг;

Appointment – клас, який містить методи для роботи з прийомом у лікаря.

Поля:

- id (тип int) – унікальний ідентифікатор;
- isPending (тип bool) – прийом в процесі очікування;
- isDone (тип bool) – прийом закінчено;
- userId (тип int) – зовнішній ключ таблиці user;
- serviceId (тип int) – зовнішній ключ таблиці service;
- doctorId (тип int) – зовнішній ключ таблиці doctor;
- orderId (тип int) – зовнішній ключ таблиці order;
- date (тип dateTime) – дата прийому;

Методи:

- getPackage() – отримання послуги;
- getPackageById() – отримання послуги по ідентифікатору;
- createPackage() – створення послуги;
- deletePackage() – оновлення послуги;
- updatePackage() – видалення послуги;

2.2. Розробка бази даних системи

Для збереження та подальшого використання даних в веб-сайті лікарні використовується структурована реляційна база даних MySQL, а також технологія Prisma ORM для управління даними.

На початку розробки бази даних було виконано два основних кроки. Першим кроком було створення самої бази даних в MySQL за допомогою MySQL Workbench, що є інтегрованим середовищем розробки баз даних для MySQL. Це дозволило створити необхідну інфраструктуру для збереження даних.

Другим кроком було створення файлу `prisma.schema`, в якому був визначений список моделей, за якими в подальшому будуть створені таблиці в базі даних під час міграції. Цей файл містить опис структури даних та їх зв'язків.

Для забезпечення під'єднання до бази даних MySQL через Prisma ORM, був створений файл `.env`, в якому були вказані наступні необхідні налаштування для здійснення з'єднання:

DB_CONNECTION=mysql – вказує на тип підключення до бази даних, в даному випадку це MySQL.

DB_HOST=localhost – вказує на адресу бази даних. Оскільки використовується локальний сервер, адреса дорівнюватиме localhost

DB_PORT=3306 – вказує на порт, через який відбувається підключення до бази даних.

DB_DATABASE=hospital – вказується назва бази даних, яка зазначена в MySQL.

DB_USERNAME=root – вказується логін користувача для авторизації в MySQL

DB_PASSWORD= вказується пароль, який повинен ввести користувач, щоб отримати доступ до бази даних.

В підсумку було створено файл конфігурації `.env` з таким лістингом:

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=hospital
DB_USERNAME=root
DB_PASSWORD=
```

Цей файл містить конфігураційні параметри, які дозволяють програмі взаємодіяти з базою даних.

Після того, як базу даних було підключено, ми створили моделі для компонентів, після чого потрібно було мігрувати зміни, за допомогою команди «`npx prisma migrate dev`».

Згодом, за вказаними моделями у файлі `prisma.scheme` створюються таблиці у базі даних MySQL. Такі моделі легше створювати та змінювати, а також це дозволяє простіше отримувати дані на серверній частині додатку. Ось приклад написання моделі, на прикладі таблиці `user`, повний список моделей наведено у ДОДАТКУ А:

```
model User {
  id          Int      @id @default(autoincrement())
  email       String
  username    String
  firstName   String
  lastName    String
  patronymic  String
  doctorProfile Doctor?
  Orders      Order[]

  role        Role      @default(USER)
  password    String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
  Appointment Appointment[]

  @@unique([email, username])
}
```

У результаті у базі даних створено наступні таблиці для кожного зі змінюваних сутностей:

1. `user` – таблиця в якій знаходиться інформація про користувачів;
2. `post` – таблиця в якій знаходиться інформація про новини;
3. `specialization` – таблиця в якій знаходиться інформація про спеціалізації лікарів;
4. `doctor` – таблиця в якій знаходиться інформація про лікаря;
5. `package` – таблиця в якій знаходиться інформація про пакет послуг;
6. `service` – таблиця в якій знаходиться інформація про послугу;
7. `order` – таблиця в якій знаходиться інформація про замовлення;
8. `appointment` – таблиця в якій знаходиться інформація про прийом до лікаря.
9. `_doctorspecializations` – таблиця яка пов'язує таблиці `doctor` та `specializations`

10. `_servicepackage` – таблиця яка пов’язує таблиці `service` та `package`
11. `_prisma_migrations` – таблиця, яка містить список міграцій Prisma ORM

Тож, база даних складатиметься з 11, таблиць: користувачі, новини, спеціалізації, лікарі, пакети послуг, послуги, замовлення, прийом. На рисунку 2.8 зображено діаграму бази даних.

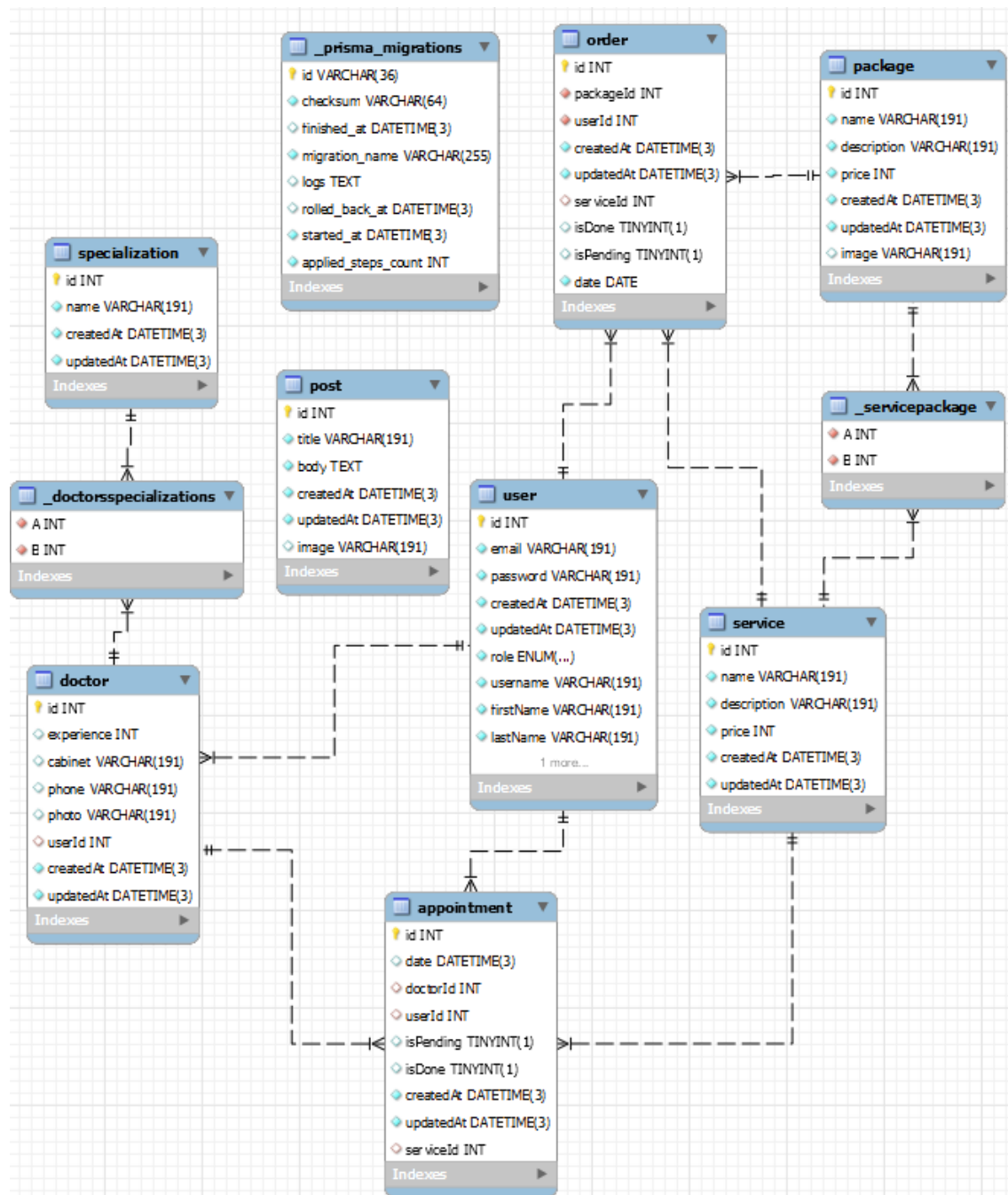


Рис. 2.8. Діаграма бази даних.

2.3. Проектування та реалізація алгоритмів роботи системи

Веб-додаток є простим у використанні, має всі основні функції веб-сайту лікарні, що дозволяє клієнтам зручно взаємодіяти з лікарями, переглядати новини та послуги. Алгоритм роботи веб-сайту лікарні зображено на рисунку 2.9, також описано детально його пункти.

Якщо користувач не авторизований, він має такі можливості:

- зареєструватися або авторизуватися на сайті
- переглянути список лікарів;
- переглянути список послуг;
- переглянути актуальні новини на сайті;
- переглянути контактні дані;

Якщо користувач авторизувався, він має такі можливості:

- перейти в особистий профіль;
- редагувати особисті дані;
- вийти з облікового запису;

Якщо авторизувався клієнт, він має такі можливості:

- замовити пакет послуг;
- переглянути або скасувати замовлення;
- перегляду списку прийомів до лікаря;

Якщо авторизувався консультант, він має такі можливості:

- переглянути список замовлень та згодом створити прийом у лікаря для клієнта;

Якщо авторизувався лікар, він має такі можливості:

- переглянути особистий профіль лікаря, редагувати його;
- перегляду графіку прийомів та змінювати прийом якщо виникла необхідність;

Якщо авторизувався модератор, він має такі можливості:

- додавати та змінювати новини;

- додавати та змінювати послуги, пакети послуг;

Якщо авторизувався адміністратор, він має такі можливості:

- перегляду списку користувачів
- зміни ролі користувача в системі
- додавання та зміни спеціальності

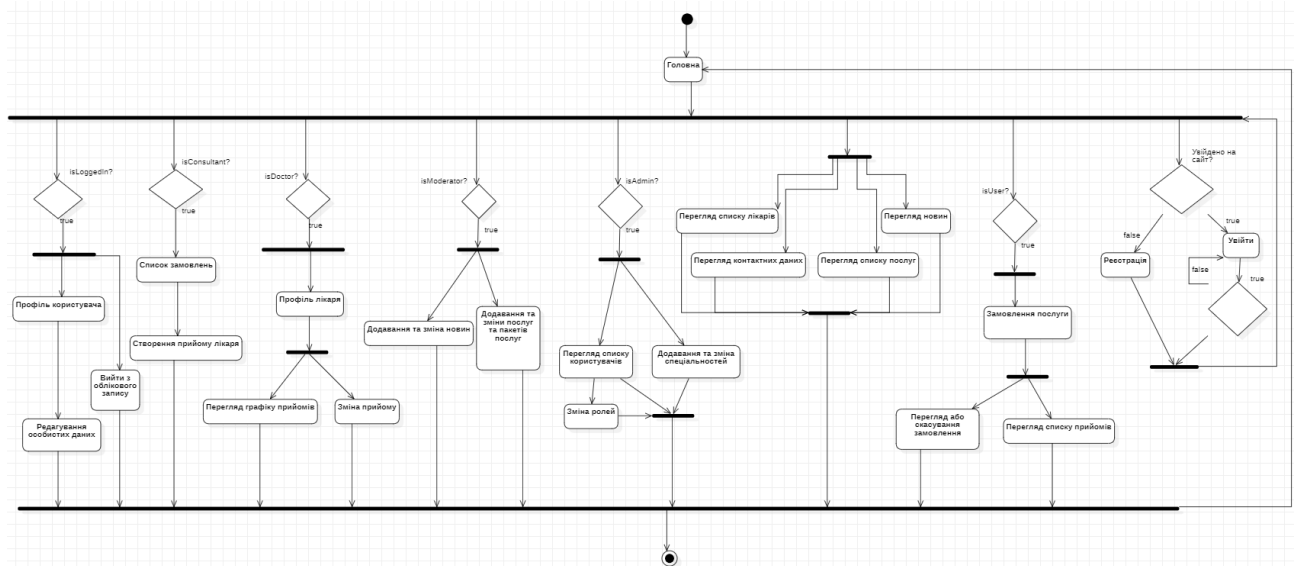


Рис. 2.9. Діаграма активності

2.4. Реалізація функціоналу веб-сайту лікарні

У попередніх розділах було зазначено, що для розробки веб-додатку буде використано: Express фреймворк на базі мови програмування JavaScript та платформи Node.js для серверної частини та React.js для клієнтської частини.

Для передачі даних між клієнтом та сервером використовуються бібліотеки Redux та Redux Toolkit. Було реалізовано код у файлі apiSlice.jsx, який забезпечить зручний і надійний механізм взаємодії між клієнтом та сервером

```
import { fetchBaseQuery } from '@reduxjs/toolkit/query/react'
const baseQuery = fetchBaseQuery({
  baseUrl: import.meta.env.BASE_URL/api',
})
```

Далі ми імпортуємо fetchBaseQuery з RTK Query, створюємо baseQuery, який будемо використовувати в запитах, встановлюємо baseUrl на наш localhost api. Наступним кроком вказуємо облікові дані, щоб дозволити відправку cookie

з кожним запитом, використовуємо функцію `prepareHeaders`, щоб налаштувати заголовок авторизації на включення токenu JWT.

```
const baseQuery = fetchBaseQuery({
  baseUrl: `${import.meta.env.BASE_URL}/api`,
  credentials: 'include',
  prepareHeaders: (headers, { getState }) => {
    const token = getState().auth.token

    if (token) {
      headers.set("authorization", `Bearer ${token}`)
    }
    return headers
  }
})
```

Оновимо дану функцію, використовуючи нову обгортку `baseQueryWithReauth()` для `baseQuery()`. Під час виконання запиту до API за допомогою функції `baseQuery()`, ми перевіряємо статус помилки, який може бути 403, що свідчить про неавторизованого користувача. У разі не авторизації, ми спробуємо оновити токен, викликаючи `setCredentials()`. Якщо оновлення токenu успішне, ми повторно виконуємо запит за допомогою `baseQuery()`. У разі невдалого оновлення токenu, ми повертаємо помилку. Якщо користувач авторизований, повертаємо отриманий результат.

```
const baseQueryWithReauth = async (args, api, extraOptions) => {
  let result = await baseQuery(args, api, extraOptions)
  if (result?.error?.status === 403) {
    const refreshResult = await baseQuery('/auth/refresh', api,
    extraOptions)
    if (refreshResult?.data) {
      api.dispatch(setCredentials({ ...refreshResult.data }))
      result = await baseQuery(args, api, extraOptions)
    } else {
      if (refreshResult?.error?.status === 403) {
        refreshResult.error.data.message = "Your login has
expired."
      }
      return refreshResult
    }
  }
  return result
}
```

Для керування ролями створено хук `useAuth()`, який можна викликати в будь-якому місці веб-додатку, та перевірити роль користувача, або його ідентифікатор

```
const useAuth = () => {
  const token = useSelector(selectCurrentToken)
  let isModerator = false
  let isDoctor = false
  let isAdmin = false
  let isConsultant = false
  let isUser = false
  let isLoggedIn = false
  if (token) {
    const decoded = jwtDecode(token)
    const { id, role } = decoded.UserInfo
    role === ROLES.MODERATOR ? isModerator = true : isModerator =
false
    role === ROLES.ADMIN ? isAdmin = true : isAdmin = false
    role === ROLES.DOCTOR ? isDoctor = true : isDoctor = false
    role === ROLES.CONSULTANT ? isConsultant = true : isConsultant =
false
    role === ROLES.USER ? isUser = true : isUser = false
    if (role !== undefined) isLoggedIn = true
    return { id, role, isLoggedIn, isModerator, isAdmin, isDoctor,
isConsultant, isUser }
  }
  return { id: '', role: '', isLoggedIn, isModerator, isAdmin,
isDoctor, isConsultant, isUser }
}
```

Ми імпортуємо хук `useSelector` з бібліотеки `react-redux` та функцію `selectCurrentToken` з файлу `authSlice`. Для перевірки стану авторизації користувача ми створюємо користувацький хук `useAuth`. Використовуючи хук `useSelector`, ми отримуємо поточний токен зі сховища `redux`.

Якщо токен присутній, ми використовуємо функцію `jwtDecode` для декодування токена та отримання ідентифікатора та ролі. За допомогою деструктуризації, ми встановлюємо значення змінних `isModerator`, `isAdmin`, `isDoctor`, `isConsultant` і `isUser` в залежності від ролі користувача.

Повертаємо об'єкт, який містить ідентифікатор, роль та змінні, що відображають стан авторизації та ролі користувача. Якщо токен відсутній,

повертаємо об'єкт зі значенням `isLoggedIn` в `false` та іншими змінними в `false`. Хук `useAuth` експортується для використання в інших компонентах.

Створимо хук для запам'ятовування логіну, для того щоб при кожному перезавантаженні сторінки користувач не виходив з системи.

```
const usePersist = () => {
  const [persist, setPersist] =
    useState(JSON.parse(localStorage.getItem("persist")) || false);

  useEffect(() => {
    localStorage.setItem("persist", JSON.stringify(persist))
  }, [persist])

  return [persist, setPersist]
}
```

За допомогою `useState` створюємо стан `persist` та функцію `setPersist` з початковим значенням, отриманим з `localStorage` або `false`. Використовуємо `JSON.parse(localStorage.getItem("persist"))` для отримання значення ключа `"persist"` з `localStorage` та його розпарсення у форматі `JSON`. За допомогою `useEffect` встановлюємо значення ключа `"persist"` у `localStorage` зі значенням стану `persist`. Ефект виконується при зміні значення стану `persist`. Функція `usePersist` повертає масив зі значенням стану `persist` та функцією `setPersist`. Тепер можна використовувати функцію `usePersist` для отримання значення ключа `"persist"` з `localStorage`. Розглянемо функцію відправлення запиту до серверу, на прикладі створення послуги. Наступний код наведено у ДОДАТКУ В

```
import { useCreateServiceMutation } from "../packagesApiSlice";
const [createService] = useCreateServiceMutation();
const handleCreate = (event) => {
  event.preventDefault();
  createService({
    name,
    description,
    price,
  });
}
```

В даному фрагменті коду імпортується хук, куди передаються дані про послугу, потім виконується дія, при якій на сервер відправляється запит

У наступному фрагменті створюється кінцева точка, при виконанні якої на сервер відправляються запит з даними у заголовку.

					ІПЗ.КР.Б – 121 – 23 – ІЗ	
						35

```
import { apiSlice } from "../api/apiSlice"

export const packagesApiSlice = apiSlice.injectEndpoints({
  endpoints: builder => ({
    createService: builder.mutation({
      query: data => ({
        url: '/services',
        method: 'POST',
        body: {
          ...data,
        }
      })
    })
  })
})
```

На серверній частині запит приходить по шляху (роутинг), після чого виконується функція з додаванням послуги. Код серверної частини наведено у ДОДАТКУ Б

Файл serviceController.js:

```
import { PrismaClient } from "@prisma/client";
const prisma = new PrismaClient();
export const createService = async (req, res) => {
  const { name, description, price } = req.body;
  try {
    const service = await prisma.service.create({
      data: {
        name: name,
        description: description,
        price: Number(price),
      }
    });
    res.status(201).json(service);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}
```

Файл serviceRoutes.js:

```
import express from 'express';
import {
  createService,
} from '../api/serviceController.js';

const router = express.Router();
router.post('/', createService);
```

Висновки до другого розділу

На даному етапі розробки дипломної роботи було проведено аналіз та визначено вимоги до розроблюваної системи. Крім того, були встановлені мінімальні системні вимоги, необхідні для належної роботи додатку. З метою більш детального уявлення про функціонал системи, були створені діаграми варіантів використання, які дозволили описати різні сценарії взаємодії з системою.

Для покращення розуміння та визначення взаємодії об'єктів веб-сайту лікарні, була розроблена об'єктно-орієнтована модель. Ця модель надала загальне уявлення про структуру системи та взаємозв'язки між її складовими.

Додатково, був розроблений загальний алгоритм, за яким буде виконуватися робота веб-сайту. Цей алгоритм був відображений у вигляді діаграми діяльності, що уявляє послідовність дій та переходів між ними.

Також була розроблена структура бази даних, яка описує схему зберігання і взаємозв'язки даних, необхідних для функціонування системи.

У заключному пункті детально розглядаються певні аспекти реалізації системи. Описується процес передачі даних між клієнтом та сервером, включаючи протоколи та формати обміну даними. Також згадується про виконання валідації даних та інші суттєві аспекти, що впливають на функціональність та надійність системи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи.

Оскільки додаток має клієнт-серверну архітектуру, це дозволяє розмістити серверну, клієнтську та адміністративну частини окремо. Мінімальні вимоги для розгортання серверної частини додатку залежать від передбачуваного обсягу трафіку, але рекомендується враховувати їх з додатковим запасом. Отже, для розгортання серверної частини додатку рекомендується використовувати сервер з такими характеристиками:

- двоядерний процесор з тактовою частотою 2 ГГц або вище;
- мінімум 2 ГБ оперативної пам'яті (ОЗУ).;
- мінімум 1 ГБ вільного місця на жорсткому диску.;

Щоб запустити серверну частину додатку потрібно виконати наступний ряд дій:

1. Встановіть Node.js на сервері.
2. Виконайте команду **npm install**, щоб встановити залежності додатку, які зазначені у файлі **package.json**.
3. Запустіть сервер, щоб додаток почав прослуховувати вхідні з'єднання.

Для розгортання клієнтської частини додатку також потрібно задовольнити та виконати аналогічні вимоги та дії.

Для встановлення та підключення бази даних MySQL, потрібно виконати такі дії:

1. Встановіть MySQL Workbench
2. Підключіться до локального або віддаленого сервера баз даних.
3. У верхньому меню оберіть пункт "Database" (База даних) і потім "Create New Schema".
4. Введіть назву **hospital**, пароль **password**, логін **root**
5. Натисніть кнопку "Apply" (Застосувати) для створення схеми бази даних.

Для взаємодії з додатком можна використовувати будь-який сучасний веб-браузер, такий як Google Chrome версії 45+, Mozilla Firefox версії 38+ або сумісні.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з веб-сайтом лікарні.

При першому заході на сайт, користувач потрапить на головну сторінку, де знаходиться основна інформація про лікарню. Користувач буде мати можливість переглянути сторінку новин, контактів, інформації про медичний заклад, про пакети послуг та список лікарів. (рис. 3.1.).

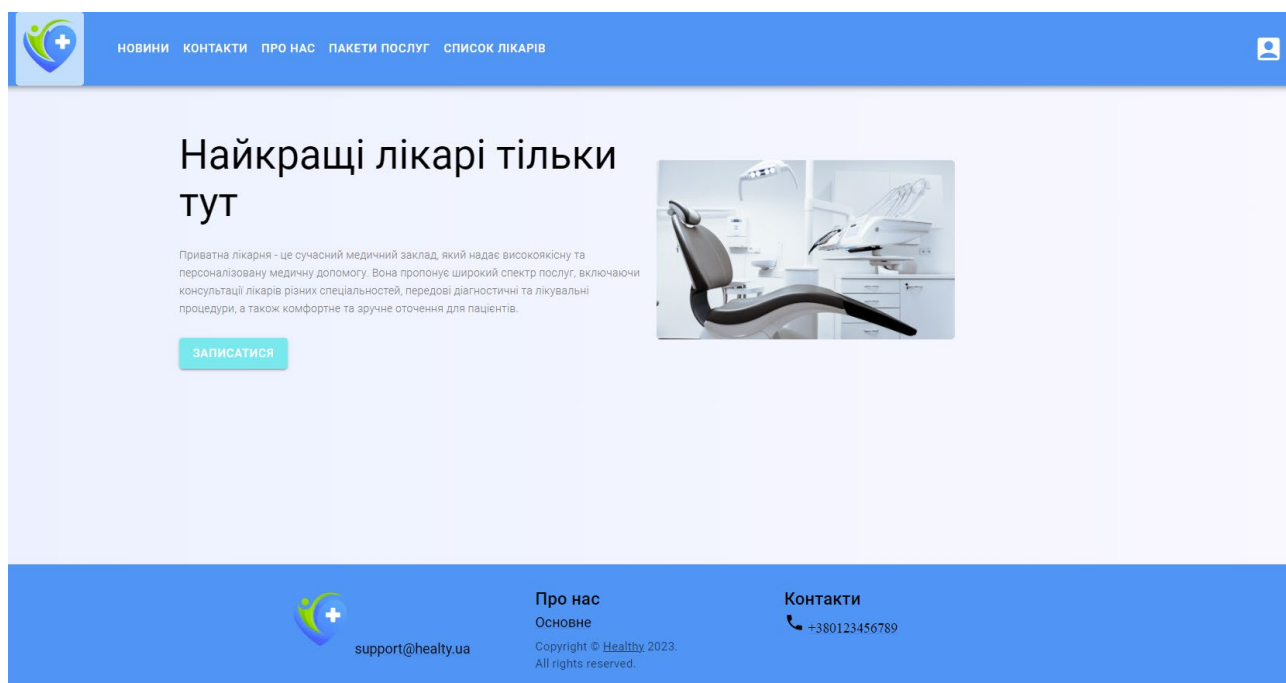


Рис. 3.1. Головна сторінка

Також є вікно входу в обліковий запис, де можна увійти в систему, при натисканні на кнопку, користувач потрапляє на сторінку входу (рис. 3.2.).

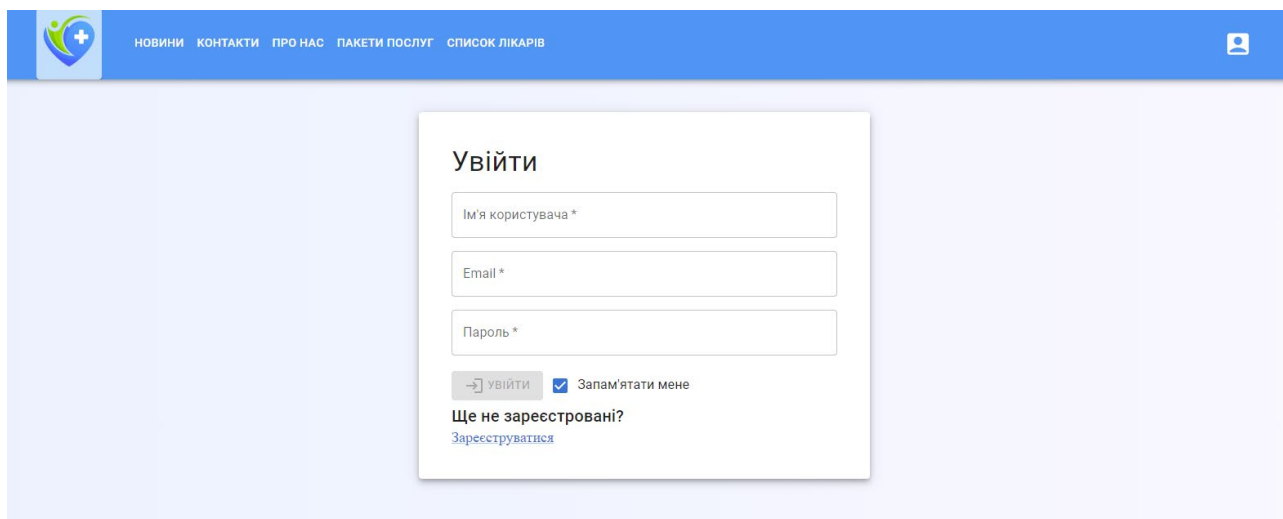


Рис. 3.2. Сторінка входу

Якщо ж користувач не зареєстрований, він може потрапити на сторінку реєстрації при натисканні на посилання. На сторінці реєстрації користувач має вказати ім'я, прізвище, по-батькові, логін, електронну пошту та пароль (рис. 3.3.).

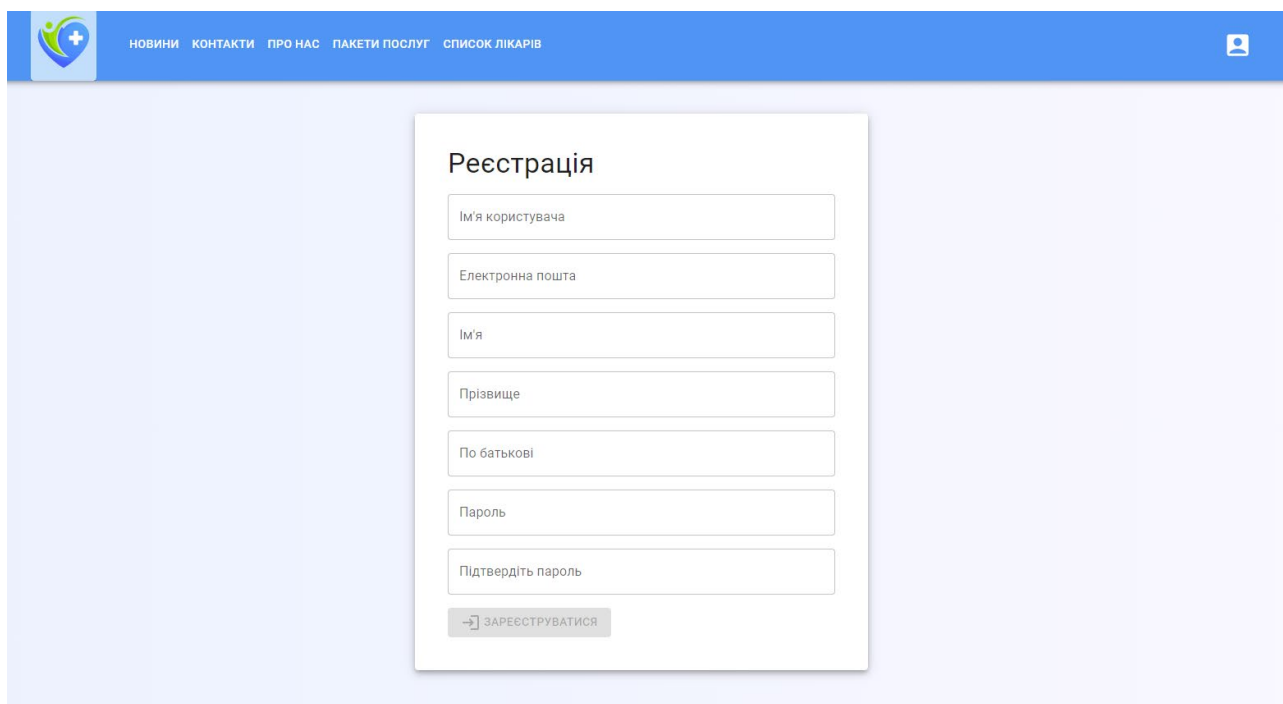


Рис. 3.3. Сторінка реєстрації

Спробуємо зареєструватися, вказавши необхідні дані. Після успішної реєстрації, користувач може зайти в особистий профіль та переглянути особисті дані (рис. 3.4.).

Рис. 3.4. Профіль користувача

Користувач може оновити особисті дані при переході на сторінку редагування даних у власному кабінеті (рис. 3.5.).

Рис. 3.5. Оновлення особистих даних

Користувач, перейшовши на сторінку пакетів послуг, може побачити медичні послуги які надаються лікарнею, та обрати будь-який з них. На сторінці працює пошук за назвою, а також присутня пагінація (рис. 3.6.).

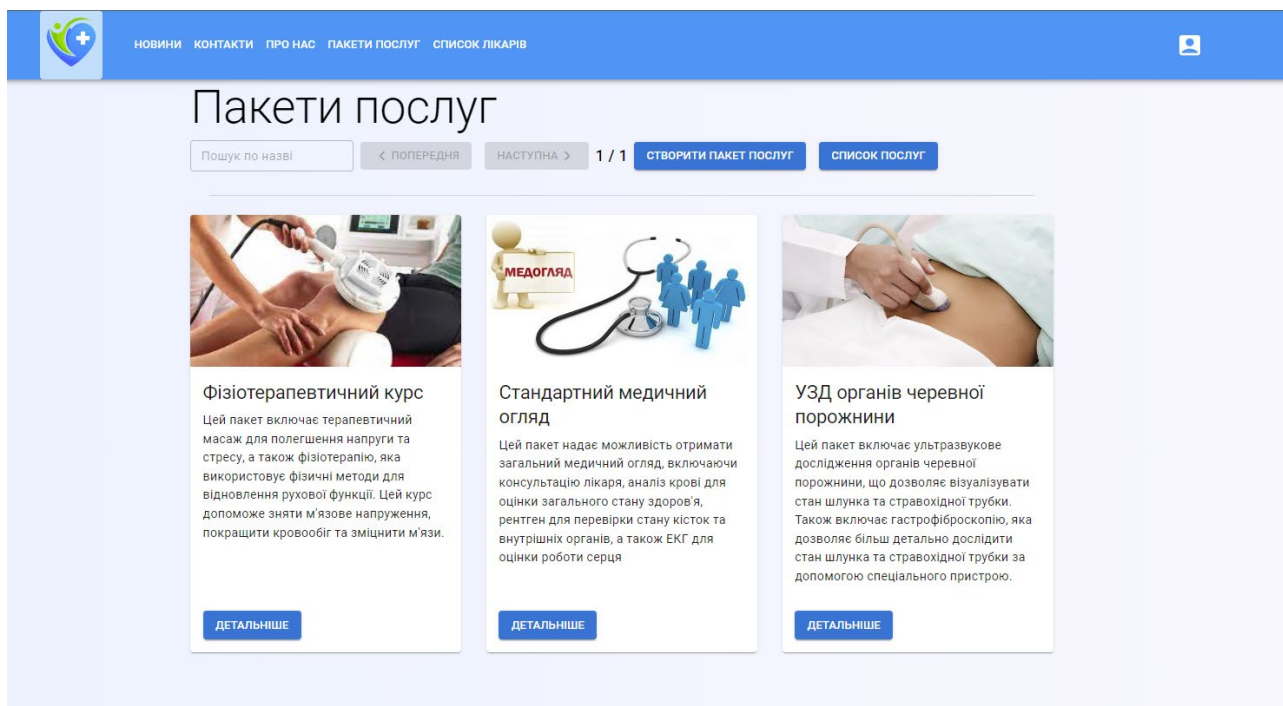


Рис. 3.6. Сторінка з пакетами послуг

Обравши пакет послуг, користувач потрапляє на сторінку замовлення пакету послуг, де вказані дані про послуги. При замовленні користувач вказує бажану дату, з якої він хоче почати отримувати послуги (рис. 3.7. – 3.8.).

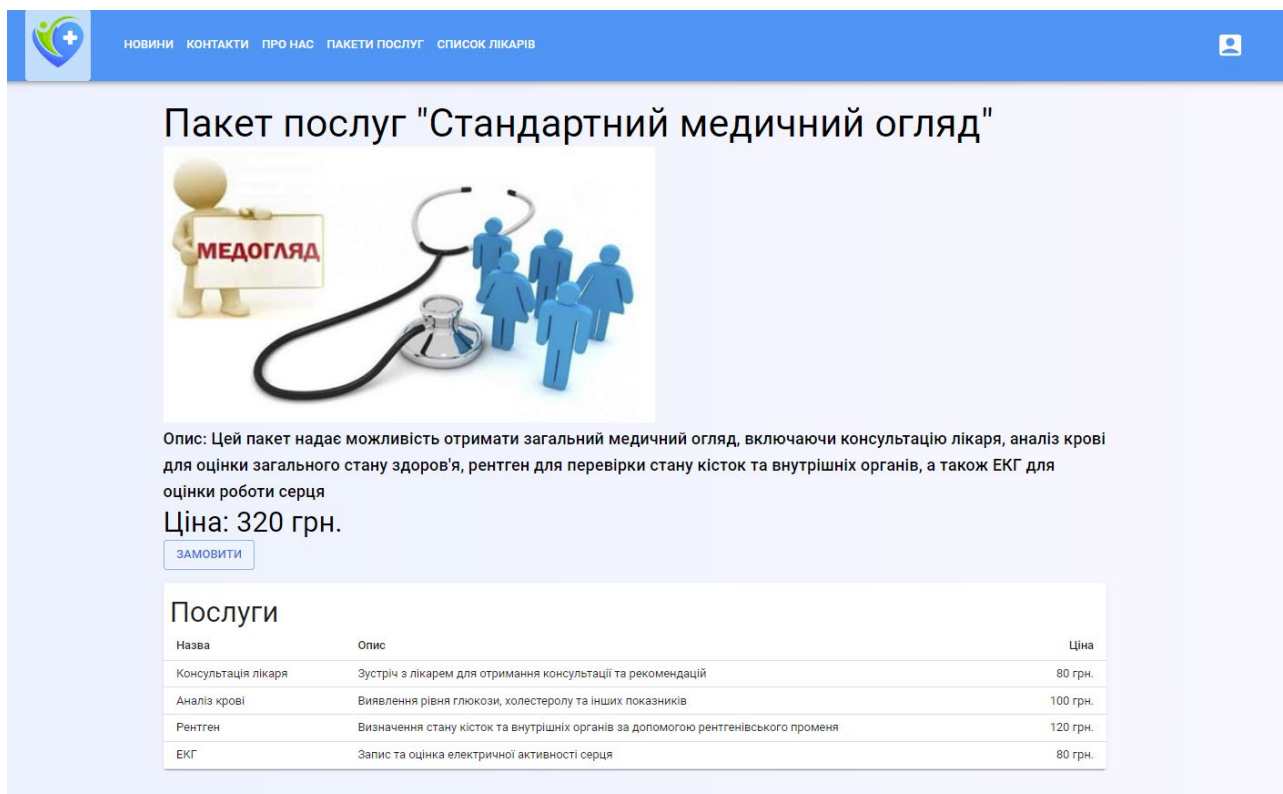


Рис. 3.7. Опис пакету послуг

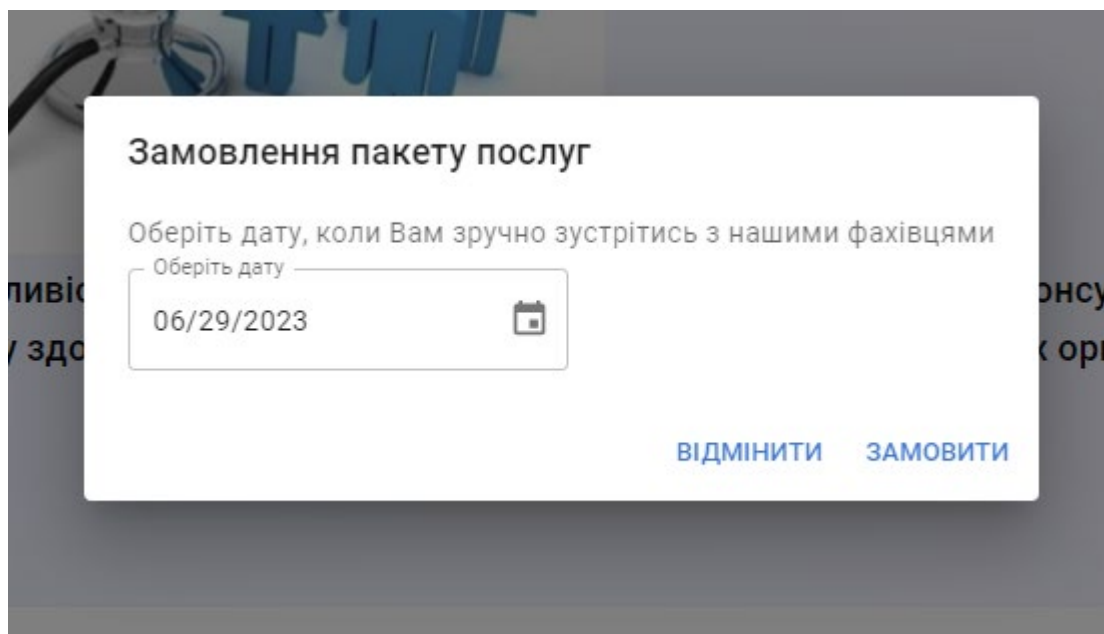
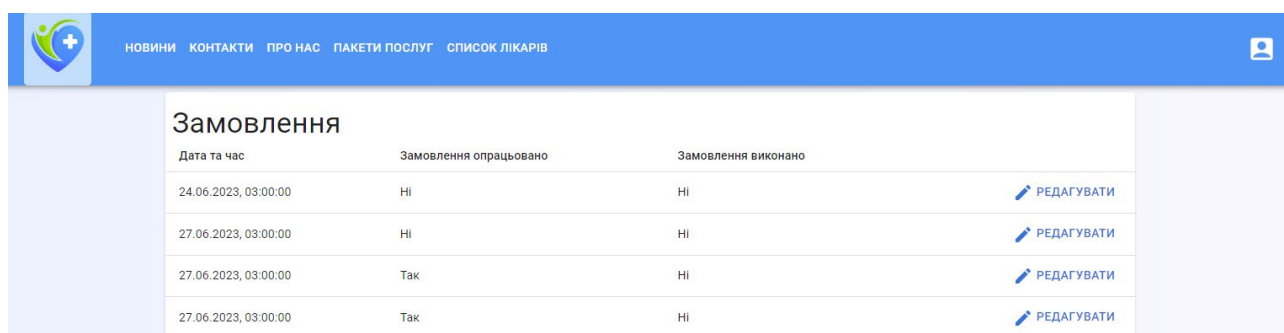


Рис. 3.8. Замовлення пакету послуг

Замовлені пакети послуг можна побачити в особистому кабінеті, у списку замовлень (рис. 3.9.).



Замовлення				
Дата та час	Замовлення опрацьовано	Замовлення виконано		
24.06.2023, 03:00:00	Ні	Ні		РЕДАГУВАТИ
27.06.2023, 03:00:00	Ні	Ні		РЕДАГУВАТИ
27.06.2023, 03:00:00	Так	Ні		РЕДАГУВАТИ
27.06.2023, 03:00:00	Так	Ні		РЕДАГУВАТИ

Рис. 3.9. Список замовлень

Якщо замовлення було опрацьоване консультантом, то можна буде переглянути графік прийомів у лікаря, де вказано дату та час прийому, назва послуги, лікар, а також кабінет (рис. 3.10.).

 НОВИНИ КОНТАКТИ ПРО НАС ПАКЕТИ ПОСЛУГ СПИСОК ЛІКАРІВ 						
Графік прийомів						
Дата та час	Послуга	Лікар	Спеціальність	Кабінет	Номер телефону	Досвід роботи
25.06.2023, 01:49:42	Ендоскопія шлунка			№		років
25.06.2023, 01:48:58	Фармацевтична консультація			№		років
24.06.2023, 23:20:00	Реабілітація післяопераційна	Литвиненко Дмитро Олегович	Оториноларинголог, Терапевт, Хірург,	№904	+3802312311	31 років
29.06.2023, 03:25:00	Ендоскопія шлунка			№		років
22.06.2023, 03:26:00	Реабілітація післяопераційна	Литвиненко Дмитро Олегович	Оториноларинголог, Терапевт, Хірург,	№904	+3802312311	31 років
29.06.2023, 03:26:00	Фармацевтична консультація	Литвиненко Дмитро Олегович	Оториноларинголог, Терапевт, Хірург,	№904	+3802312311	31 років

Рис. 3.10. Графік прийомів

Користувач має можливість перегляду новин про медичний заклад або його діяльність, перейшовши на відповідну сторінку. На цій сторінці є список новин, які відсортовані за датою створення, також присутня пагінація (рис. 3.11.).

 НОВИНИ КОНТАКТИ ПРО НАС ПАКЕТИ ПОСЛУГ СПИСОК ЛІКАРІВ 						
Новини						
← ПОПЕРЕДНЯ НАСТУПНА → 1 / 1						
		Лікарня "Здоров'я +" розпочинає спеціалізовану програму профілактики серцевих захворювань для місцевого населення! Лікарня "Здоров'я +", лідер у сфері медичних послуг та іннов				
		Лікарня отримала нагороду за високу якість медичної допомоги та інноваційний підхід! ЛІКАРНЯ ВІДОМА СВОЇМ ВИСОКИМ РІВНЕМ МЕДИЧНОГО ОБСЛУГОВУВАННЯ ТА ІННОВАЦІЙНИМИ ПІДХОДАМИ ДО ЛІКУВАН				

Рис. 3.11. Список новин

Обравши новину, користувач перейде на сторінку новини, де вказаний заголовок, час публікації та основний текст (рис. 3.12.).

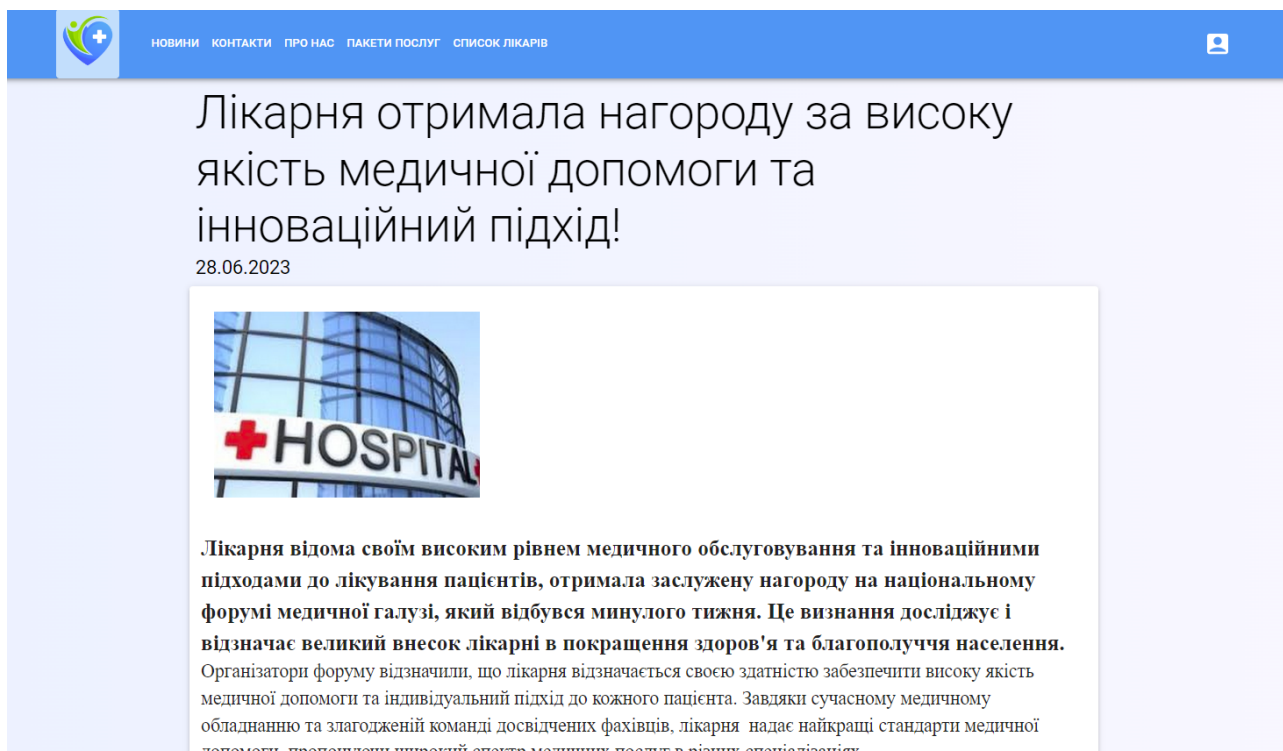


Рис. 3.12. Новина

Користувачі можуть переглянути список лікарів медичного закладу, перейшовши на відповідну сторінку. Тут вказані дані про лікарів, їхні спеціалізації, фотографії (рис. 3.13.).

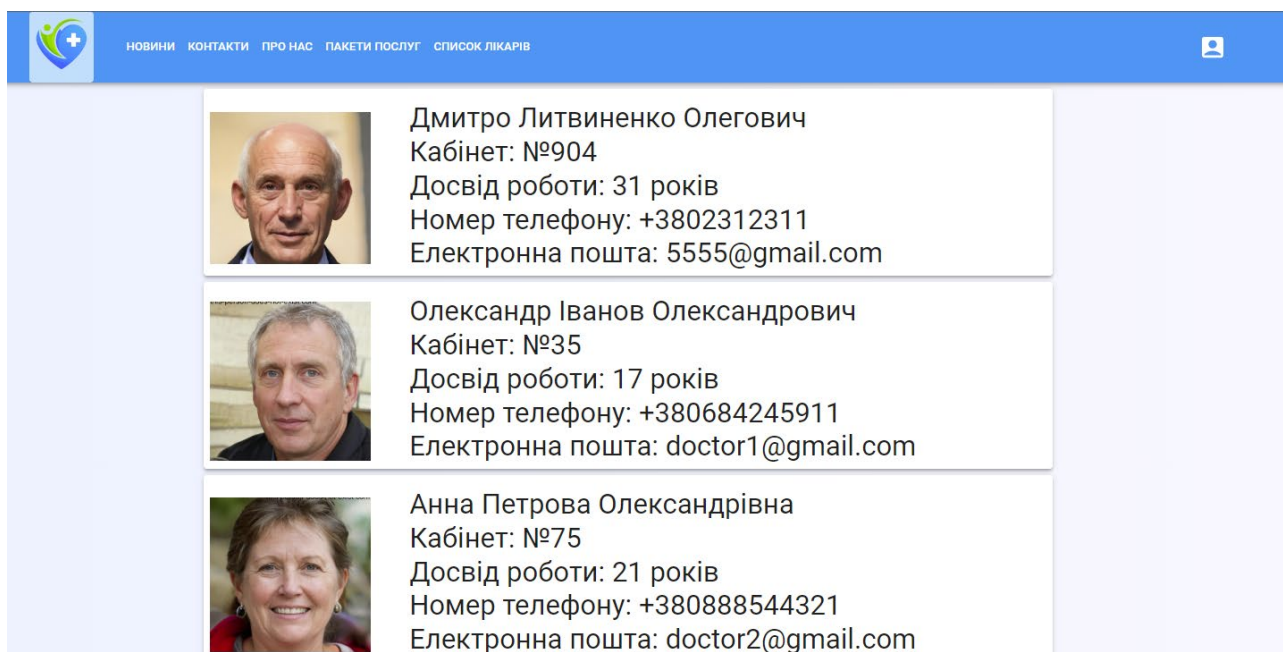


Рис. 3.13. Список лікарів

Також можна переглянути інформацію про медичний заклад, перейшовши на сторінку «Про нас» (рис. 3.14.)



[НОВИНИ](#)
[КОНТАКТИ](#)
[ПРО НАС](#)
[ПАКЕТИ ПОСЛУГ](#)
[СПИСОК ЛІКАРІВ](#)
[ЗАМОВЛЕННЯ](#)



Замовлення: Стандартний медичний огляд

Опис: Цей пакет надає можливість отримати загальний медичний огляд, включаючи консультацію лікаря, аналіз крові для оцінки загального стану здоров'я, рентген для перевірки стану кісток та внутрішніх органів, а також ЕКГ для оцінки роботи серця

Статус: У процесі

Ціна: 320 грн.

Назва	Опис	Ціна
Консультація лікаря	Зустріч з лікарем для отримання консультації та рекомендацій	80 грн.
Аналіз крові	Виявлення рівня глюкози, холестеролу та інших показників	100 грн.
Рентген	Визначення стану кісток та внутрішніх органів за допомогою рентгенівського променя	120 грн.
ЕКГ	Запис та оцінка електричної активності серця	80 грн.

✖ ВІДМІНИТИ ЗАМОВЛЕННЯ
✎ РЕДАГУВАТИ ЗАМОВЛЕННЯ
➕ ДОДАТИ ПРИЙОМ

Рис. 3.16. Сторінка замовлення

Натиснувши на кнопку «Редагувати замовлення», ми перейдемо на сторінку редагування замовлення, де можна змінити дату замовлення, або ж відмінити його (рис. 3.17.).



[НОВИНИ](#)
[КОНТАКТИ](#)
[ПРО НАС](#)
[ПАКЕТИ ПОСЛУГ](#)
[СПИСОК ЛІКАРІВ](#)
[ЗАМОВЛЕННЯ](#)



Редагування замовлення №7

Бажана дата та час: 27.06.2023, 03:00:00

🕒 ЗМІНИТИ ЧАС

Рис. 3.17. Редагування замовлення

Перейшовши на сторінку додавання прийомів до лікаря, з'являється список послуг, де можна обрати одну з них, після чого потрібно обрати лікаря, дату та час прийому, згодом додати прийом (рис. 3.18. – 3.20.).



НОВИНИ

КОНТАКТИ

ПРО НАС

ПАКЕТИ ПОСЛУГ

СПИСОК ЛІКАРІВ

ЗАМОВЛЕННЯ



Запис на прийом

Бажана дата: 27.06.2023

Замовлення #8

Пакет послуг "Стандартний медичний огляд"

Опис: Цей пакет надає можливість отримати загальний медичний огляд, включаючи консультацію лікаря, аналіз крові для оцінки загального стану здоров'я, рентген для перевірки стану кісток та внутрішніх органів, а також ЕКГ для оцінки роботи серця

Статус: У процесі

Ціна: 320 грн.

Послуги

Назва	Опис	Ціна	
Консультація лікаря	Зустріч з лікарем для отримання консультації та рекомендацій	80 грн.	ОБРАТИ
Аналіз крові	Виявлення рівня глюкози, холестеролу та інших показників	100 грн.	ОБРАТИ
Рентген	Визначення стану кісток та внутрішніх органів за допомогою рентгенівського променя	120 грн.	ОБРАТИ
ЕКГ	Запис та оцінка електричної активності серця	80 грн.	ОБРАТИ

Рис. 3.18. Обрання послуги зі списку

Замовлення #8

Пакет послуг "Стандартний медичний огляд"

Опис: Цей пакет надає можливість отримати загальний медичний огляд, включаючи консультацію лікаря, аналіз крові для оцінки загального стану здоров'я, рентген для перевірки стану кісток та внутрішніх органів, а також ЕКГ для оцінки роботи серця

Статус: У процесі

Ціна: 320 грн.

Оберіть лікаря

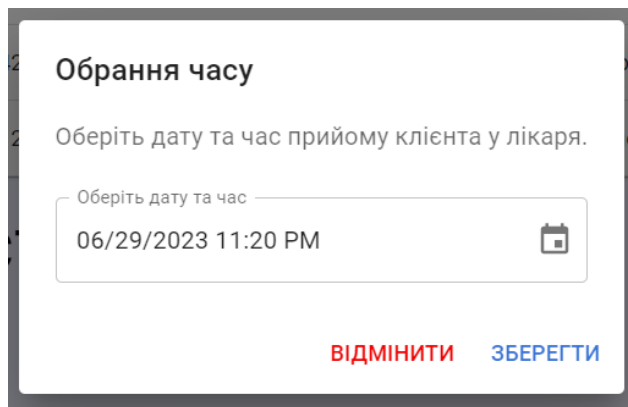
< ПОПЕРЕДНЯ

НАСТУПНА >

1 / 1

Ім'я	Номер телефону	Кабінет	Досвід роботи	Спеціальність	
Ковальська Анна йцуйцуйцуйц	+380634567890	№54	12 років	Офтальмолог, Оториноларинголог	ОБРАТИ
Григорова Марія Валеріївна	+380673332221	№33	7 років	Офтальмолог	ОБРАТИ
Литвиненко Дмитро Олегович					ОБРАТИ
Кравченко Юлія Віталіївна					ОБРАТИ
Зайцев Віктор Володимирович					ОБРАТИ

Рис. 3.19. Обрання лікаря



Обрання часу

Оберіть дату та час прийому клієнта у лікаря.

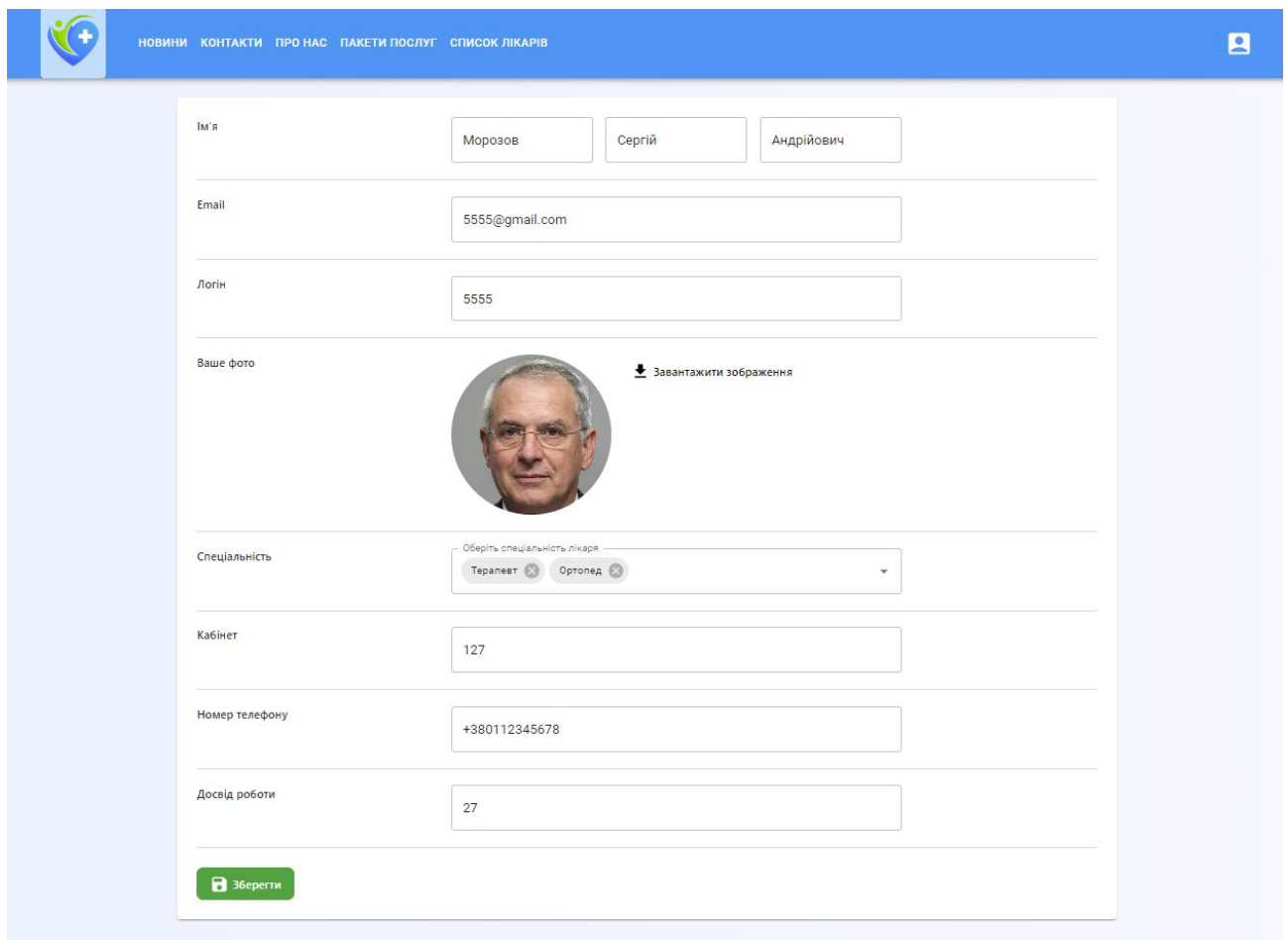
Оберіть дату та час

06/29/2023 11:20 PM

ВІДМІНИТИ **ЗБЕРЕГТИ**

Рис. 3.20. Обрання часу

Тепер вийдемо з облікового запису консультанта, і увійдемо у систему з лікарського акаунту. Перейшовши у особистий кабінет, ми бачимо дані про лікаря, а також кнопки переходу до редагування облікових даних, та перегляду графіку прийомів. Натиснувши кнопку «Редагувати», ми потрапимо на сторінку зміни облікових даних, де можна змінити інформацію про себе, спеціальності, додати фотографію (рис. 3.21.).



НОВИНИ КОНТАКТИ ПРО НАС ПАКЕТИ ПОСЛУГ СПИСОК ЛІКАРІВ

Ім'я Морозов Сергій Андрійович

Email 5555@gmail.com

Логін 5555

Ваше фото  Завантажити зображення

Спеціальність Оберіть спеціальність лікаря
Терапевт Ортопед

Кабінет 127

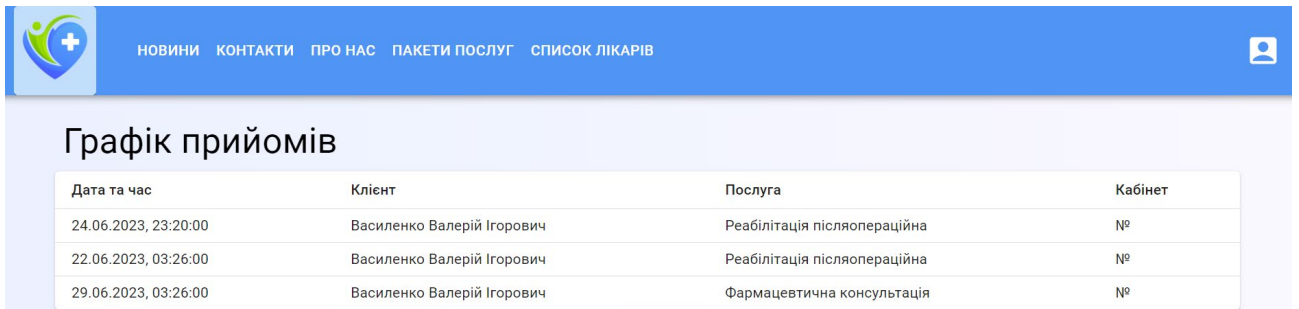
Номер телефону +380112345678

Досвід роботи 27

Зберегти

Рис. 3.21. Редагування даних про лікаря

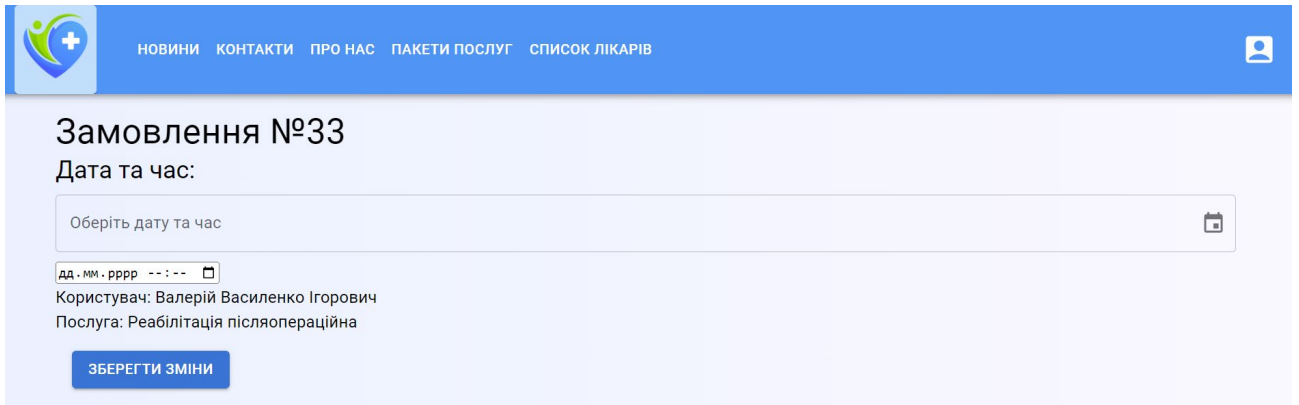
Перейдемо на сторінку графіку прийомів, де ми побачимо список прийомів клієнтів, їхнє ім'я, назву послуги, дату та час. Тут можна завершити прийом, якщо послугу було надано клієнту (рис. 3.22.).



Дата та час	Клієнт	Послуга	Кабінет
24.06.2023, 23:20:00	Василенко Валерій Ігорович	Реабілітація післяопераційна	№
22.06.2023, 03:26:00	Василенко Валерій Ігорович	Реабілітація післяопераційна	№
29.06.2023, 03:26:00	Василенко Валерій Ігорович	Фармацевтична консультація	№

Рис. 3.22. Графік прийомів

Також можна змінити дату та час прийому, перейшовши на сторінку редагування (рис. 3.23.)



Замовлення №33

Дата та час:

Оберіть дату та час

дд.мм.рррр --:--

Користувач: Валерій Василенко Ігорович

Послуга: Реабілітація післяопераційна

ЗБЕРЕГТИ ЗМІНИ

Рис. 3.23. Редагування прийому

Тепер увійдемо у систему з облікового запису модератора.

Перейшовши на сторінку з новинами, ми побачимо кнопку створення новини. Натиснувши на неї, ми потрапимо на форму створення новини, де вказується заголовок, основний текст, та обирається головне зображення (рис. 3.24.).



Рис. 3.25. Редагування новини

Перейдемо до сторінки пакетів послуг. Модератор має можливість створити пакет послуг та переглянути список послуг (рис. 3.26.).

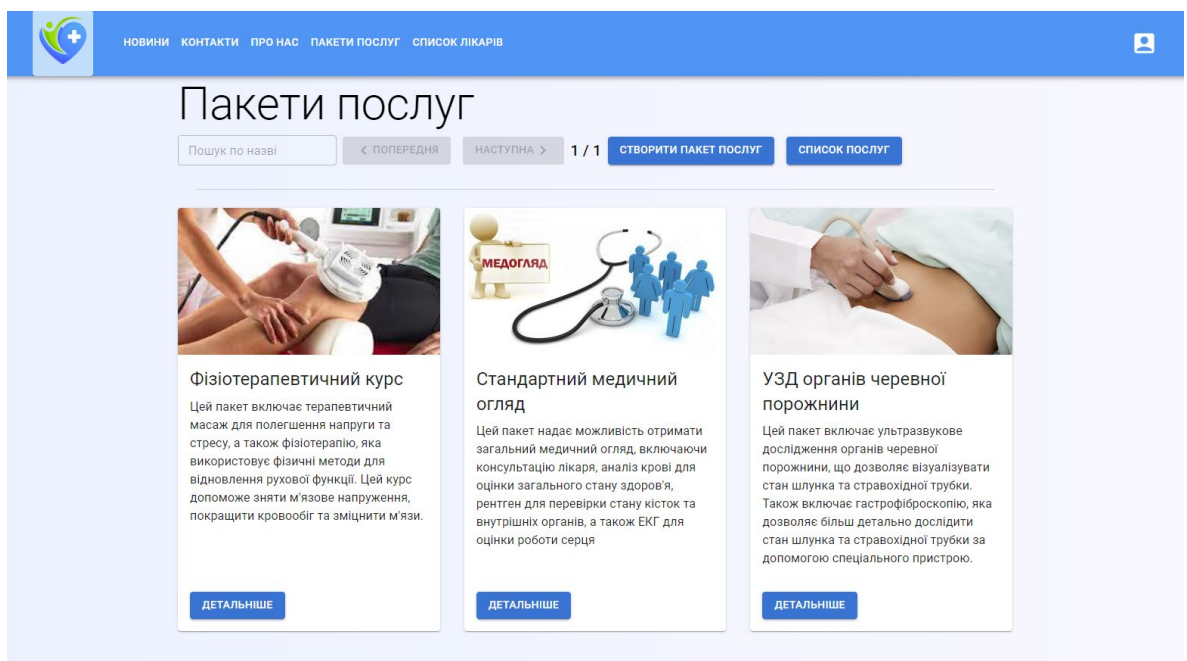


Рис. 3.26. Сторінка пакетів послуг

Перейдемо на сторінку створення пакету послуг. На цій сторінці ми можемо обрати послуги, які входитимуть у пакет, встановити назву, ціну, опис. Також можна обрати фотографію (рис 3.27.).

Назва пакету послуг

Індивідуальна реабілітаційна програма

Опис пакету послуг

Цей пакет створений для пацієнтів, які потребують комплексної реабілітації. Він включає фізіотерапевтичні процедури для відновлення рухової функції, психотерапію для підтримки психічного здоров'я

Ціна

250

Послуги

Послуги

Фізіотерапія × Психотерапія ×

Ваше фото

Завантажити зображення

Створити пакет послуг

Рис. 3.27. Створення пакету послуг

Також ми можемо відредагувати або ж видалити створений пакет послуг, перейшовши на відповідну сторінку (рис. 3.28.).

НОВИНИ

КОНТАКТИ

ПРО НАС

ПАКЕТИ ПОСЛУГ

СПИСОК ЛІКАРІВ

Назва пакету послуг

Індивідуальна реабілітаційна програма

Опис пакету послуг

Цей пакет створений для пацієнтів, які потребують комплексної реабілітації. Він включає фізіотерапевтичні процедури для відновлення рухової функції, психотерапію для підтримки психічного здоров'я

Ціна

250

Послуги

Оберіть послуги

Фізіотерапія

Психотерапія

Ваше фото

Завантажити зображення

Зберегти

Видалити

Рис. 3.28. Редагування пакету послуг

Перейшовши на сторінку перегляду списку послуг, ми побачимо наявні послуги у системі, які можна знайти по назві (рис. 3.29.).

НОВИНИ

КОНТАКТИ

ПРО НАС

ПАКЕТИ ПОСЛУГ

СПИСОК ЛІКАРІВ

Список послуг

Пошук по назві

< ПОПЕРЕДНЯ

НАСТУПНА >

1 / 2

ДОДАТИ ПОСЛУГУ

Назва	Опис	Ціна	
Маммографія	Рентгенологічне дослідження молочних залоз для виявлення ознак раку молочної залози	200	ОБРАТИ
Консультація гінеколога	Зустріч з гінекологом для отримання консультації та лікування жіночих захворювань	80	ОБРАТИ
УЗД серця	Ультразвукове дослідження серця для оцінки його структури та функцій	180	ОБРАТИ
Лабораторний аналіз	Дослідження біологічних зразків (крові, сечі, шлункового соку тощо) для діагностики та моніторингу стану здоров'я	200	ОБРАТИ
Психотерапія	Консультація та підтримка психотерапевта для покращення психічного здоров'я	150	ОБРАТИ
Фізіотерапія	Використання фізичних методів для відновлення рухової функції	90	ОБРАТИ
Дентальне чищення	Професійне чищення зубів, видалення нальоту та осаду для збереження здоров'я порожнини рота	120	ОБРАТИ
Імунізація	Введення вакцини для профілактики захворювань та зміцнення імунної системи	100	ОБРАТИ
Очна консультація	Консультація офтальмолога для діагностики та лікування очних захворювань	60	ОБРАТИ
УЗД	Візуалізація внутрішніх органів за допомогою ультразвуку	150	ОБРАТИ

Рис. 3.29. Список послуг

Також можна додати або ж відредагувати послугу, де потрібно вказати назву, опис, та ціну послуги. Можна видалити раніше створену послугу (рис. 3.30. – 3.31).

Створення нової послуги

Назва послуги

Опис послуги

Ціна

Зберегти

Рис. 3.30. Створення послуги

Редагувати послугу

Назва послуги

Опис послуги

Ціна

Зберегти

Видалити

Рис. 3.31. Редагування послуги

Увійдемо в систему з облікового запису адміністратора. Ми можемо потрапити на панель адміністрування, де можна переглянути список користувачів, спеціалізацій (рис. 3.32. – 3.33.).

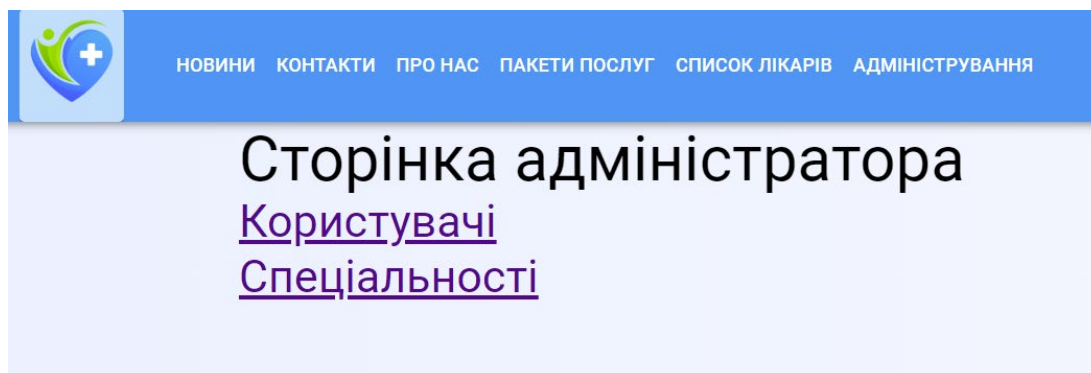


Рис. 3.32. Адміністративна панель

НОВИНИ КОНТАКТИ ПРО НАС ПАКЕТИ ПОСЛУГ СПИСОК ЛІКАРІВ АДМІНІСТРУВАННЯ							
Пошук по назві Попередня сторінка Наступна сторінка 1 / 1							
Username	Email	First name	Last name	Patronymic	Created at	Updated at:	Edit
1111	1111@gmail.com	Олександр	Гопчук	Ярославович	2023-06-20T15:26:17.711Z	2023-06-27T17:48:13.753Z	Edit user
2222	2222@gmail.com	Анна	Ковальська	Зінаїдівна	2023-06-20T15:48:48.898Z	2023-06-28T03:02:17.854Z	Edit user
3333	3333@gmail.com	Максим	Іванов	Ярославович	2023-06-20T15:49:01.112Z	2023-06-27T23:17:26.036Z	Edit user
4444	4444@gmail.com	Константин	Бобров	Ігорович	2023-06-21T18:50:04.020Z	2023-06-27T23:31:29.726Z	Edit user
5555	5555@gmail.com	Дмитро	Литвиненко	Олегович	2023-06-23T14:55:28.119Z	2023-06-28T05:51:54.666Z	Edit user
6666	6666@gmail.com	Михайло	Кошобинський	Володимирович	2023-06-23T15:13:58.530Z	2023-06-23T22:52:07.613Z	Edit user
7777	7777@gmail.com	Валерій	Василенко	Ігорович	2023-06-23T15:14:26.320Z	2023-06-28T06:22:22.268Z	Edit user
doctor1	doctor1@gmail.com	Олександр	Іванов	Олександрович	2023-06-27T14:29:21.099Z	2023-06-28T06:08:51.429Z	Edit user
doctor2	doctor2@gmail.com	Анна	Петрова	Олександрівна	2023-06-27T14:29:50.463Z	2023-06-28T06:10:00.375Z	Edit user
doctor3	doctor3@gmail.com	Михайло	Сидоров	Петрович	2023-06-27T14:30:12.589Z	2023-06-27T15:35:32.554Z	Edit user
doctor4	doctor4@gmail.com	Ірина	Ковальчук	Михайлівна	2023-06-27T14:30:36.970Z	2023-06-27T15:35:32.554Z	Edit user
doctor5	doctor5@gmail.com	Сергій	Морозов	Андрійович	2023-06-27T14:31:16.721Z	2023-06-27T15:35:32.554Z	Edit user
doctor6	doctor6@gmail.com	Ольга	Василенко	Ігорівна	2023-06-27T14:31:40.128Z	2023-06-27T15:35:32.554Z	Edit user
doctor7	doctor7@gmail.com	Віктор	Зайцев	Володимирович	2023-06-27T14:32:16.333Z	2023-06-27T15:35:32.554Z	Edit user
doctor8	doctor8@gmail.com	Юлія	Кравченко	Віталіївна	2023-06-27T14:32:44.137Z	2023-06-27T15:35:32.554Z	Edit user
doctor9	doctor9@gmail.com	Дмитро	Литвиненко	Олегович	2023-06-27T14:33:15.281Z	2023-06-27T15:35:32.554Z	Edit user
doctor0	doctor0@gmail.com	Марія	Григорова	Валеріївна	2023-06-27T14:33:36.791Z	2023-06-27T17:24:18.943Z	Edit user

Рис. 3.33. Список користувачів

Перейшовши на профіль користувача адміністратор може змінювати лише його роль у системі (рис. 3.34.)

НОВИНИ

КОНТАКТИ

ПРО НАС

ПАКЕТИ ПОСЛУГ

СПИСОК ЛІКАРІВ

АДМІНІСТРУВАННЯ

Ім'я

Анна

Ковальська

Зінаїдівна

Email

2222@gmail.com

Логін

2222

Пароль

Пароль

Підтвердіть пароль

Роль

DOCTOR

ADMIN

MODERATOR

USER

DOCTOR

CONSULTANT

Зберегти

Видалити

Рис. 3.34. Редагування ролі користувача

Також адміністратор має можливість переглянути спеціалізації, які є у системі, додати або відредагувати спеціалізацію (рис. 3.35. – 3.37.).

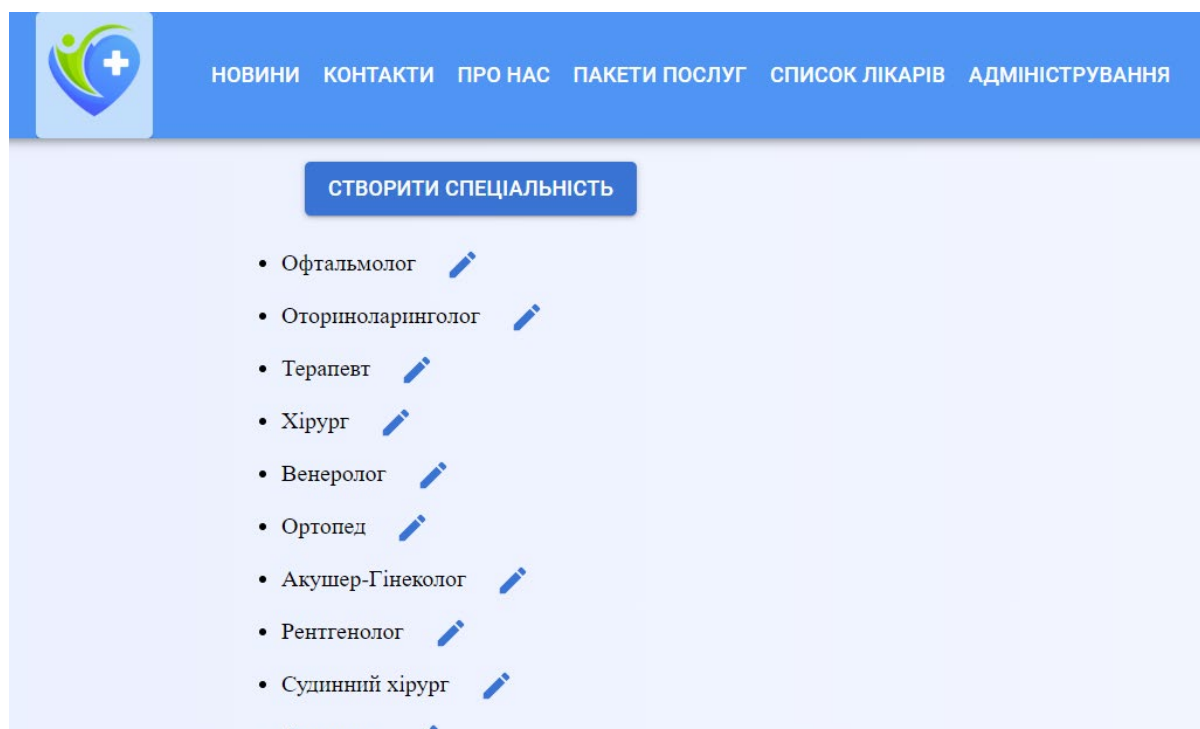


Рис. 3.35. Список спеціалізацій

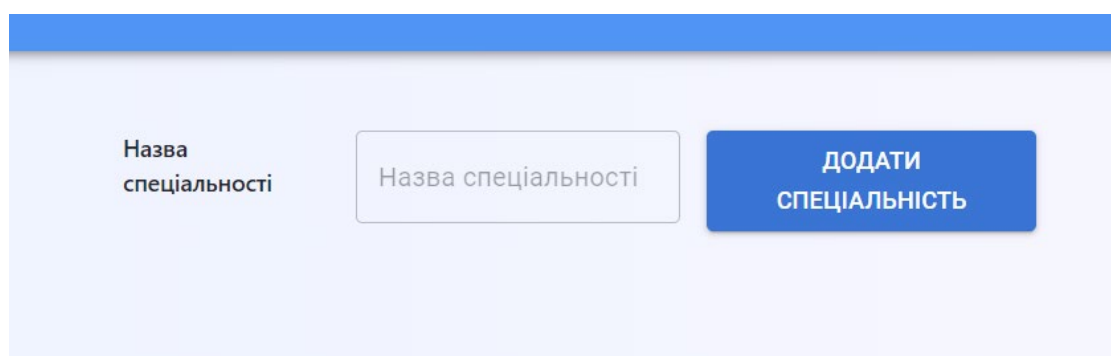


Рис. 3.36. Додавання спеціалізації

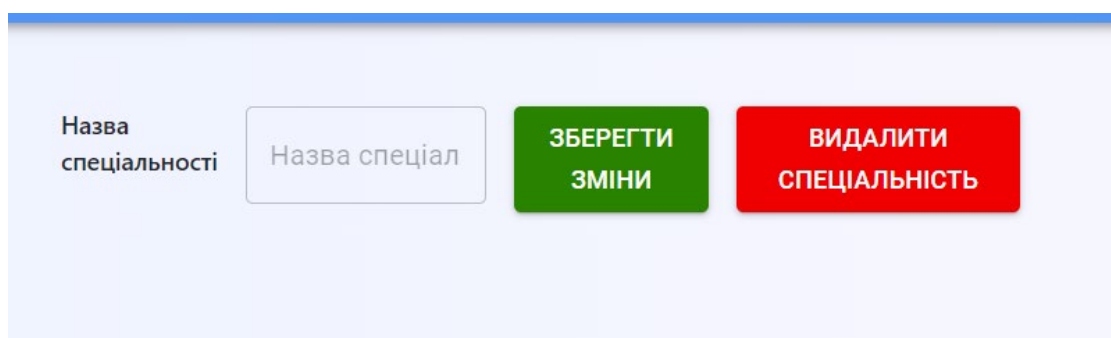


Рис. 3.37. Редагування спеціалізації

3.3. Тестування роботи веб-сайту

Після завершення розробки веб-застосунку, проводилося тестування, щоб переконатися, що він відповідає вимогам, поставленим у даній роботі. Для коректної роботи додатку необхідно проводити тести, які виявлять слабкі місця та проблеми у додатку. Для перевірки працездатності виконаємо дії, які повинен виконувати користувач.

1. Реєстрація облікового запису:

передумови: користувач, потрапивши на головну сторінку має можливість зареєструватися;

кроки: натиснувши “Зареєструватися”, користувач заповнює форму та відправляє дані;

очікуваний результат: користувач отримує лист з верифікацією на пошту, після верифікації обліковий запис ініціалізований в системі;

фактичний результат: користувач пройшов верифікацію.

2. Користувач заходить в персональний кабінет:

передумови: користувач переходить до акаунту з головної сторінки;

очікуваний результат: користувач перейшов до особистого кабінету і взаємодіє з його елементами;

фактичний результат: користувач потрапив до особистого кабінету.

3. Взаємодія з сутностями (Замовлення, Прийоми):

передумови: користувач повинен мати змогу для керування потрібними йому даними;

кроки: перехід на вкладку сутності (Замовлення, Прийоми), де користувач може переглянути, додати, змінити та видалити обрані дані;

очікуваний результат: користувач змінює необхідні йому дані й вони зберігаються;

фактичний результат: користувач має можливість взаємодіяти зі всіма елементами персонального кабінету.

4. Взаємодія з елементами головної сторінки:

передумови: користувач, який знаходиться на головній сторінці,

					ІПЗ.КР.Б – 121 – 23 – ІЗ	58

повинен мати змогу: переглядати сторінки “Про нас”, “Контакти”, “Новини”, “Пакети послуг”, “Список лікарів” та мати можливість взаємодіяти з інформацією;

передумови: користувач, який знаходиться на головній сторінці, відкриває обране посилання;

очікуваний результат: користувачу відкривається сторінка з завантаженими даними;

фактичний результат: користувач має змогу взаємодіяти з необхідними елементами.

5. CRUD сутностей (Новини, Пакети послуг, Послуги) користувачем з роллю модератор:

передумови: користувач (модератор), перейшовши на відповідну сторінку, повинен мати можливість для маніпулювання необхідними даними;

кроки: обравши додавання новин, модератор побачить форму з необхідними для заповнення полями;

очікуваний результат: після відправлення форми, новина збережена у базі даних;

фактичний результат: після відправлення форми, новина збережена у базі даних.

Висновки до третього розділу

У даному розділі було наведено порядок встановлення веб-додатку та визначено мінімальну конфігурацію системи для успішного розгортання додатку.

Оскільки були дотримані усі вимоги до розробки додатку, під час розробки та використання додатку було уникнуто проблеми.

Також було проведено тестування усіх частин додатку, в результаті виконаного тестування було знайдено та виправлено декілька помилок.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра було спроектовано та розроблено сайт лікарні. Для досягнення результатів під час розробки були виконано наступні кроки:

1. Проведено аналіз предметної області та існуючих аналогів з метою виявлення потреб користувачів і визначення основних функцій веб-сайту. Було обрано набір технологічних інструментів, які використовуються для розробки проекту.

2. Було спроектовано та розроблено проект, загальний алгоритм роботи веб-сайту у вигляді діаграм, де вказана взаємодія між користувачами і додатком. Розроблено структуру бази даних для загального функціонування роботи системи.

3. Було реалізовано програмний код, основні методи та функції, які використовуються для коректної роботи веб-сайту. Було представлено загальний алгоритм роботи сайту.

4. Було детально розглянуто процес користування системою, компонентів та сторінок на сайті. Розглянуто всі можливі дії для кожної з ролей (гість, клієнт, лікар, консультант, модератор, адміністратор)

У результаті проведеної роботи було реалізовано сайт лікарні, який може надавати послуги, інформацію, контакти медичного закладу. У майбутньому програмний продукт може покращуватись та розширюватись за рахунок введення додаткового функціоналу, поліпшення дизайну, розробки мобільного додатку, інтеграції з іншими системами та додатками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Медична мережа «Добробут» [Електронний ресурс] – Режим доступу до ресурсу: <https://www.dobrobut.com/ua/clinics/contacts>
2. «Експерт» лікарня [Електронний ресурс] – Режим доступу до ресурсу: <https://experthospital.com.ua/>
3. КНП СОР «Сумська обласна клінічна лікарня» [Електронний ресурс] – Режим доступу до ресурсу: <http://sokl.com.ua/>
4. КНП «Обласна клінічна лікарня ім. О. Ф. Гербачевського» [Електронний ресурс] – Режим доступу до ресурсу: <https://zhokl.com.ua/>
5. What is Client Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://intellipaat.com/blog/what-is-client-server-architecture/?US>
6. MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>
7. Prisma ORM [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io/docs>
8. Redux [Електронний ресурс] – Режим доступу до ресурсу: <https://redux.js.org/introduction/getting-started>
9. Redux Toolkit [Електронний ресурс] – Режим доступу до ресурсу: <https://redux-toolkit.js.org/introduction/getting-started>
10. React.js [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn>
11. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en/docs>
12. Express Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/en/5x/api.html>
13. MaterialUI [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/material-ui/getting-started/overview/>
14. MaterialUI React Templates [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/material-ui/getting-started/templates/>

15. How to Build an Authentication API with JWT Token in Node.js [Электронный ресурс] – Режим доступа до ресурсу: <https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/>
16. Proxying API requests in Development [Электронный ресурс] – Режим доступа до ресурсу: <https://create-react-app.dev/docs/proxying-api-requests-in-development/>
17. React Redux [Электронный ресурс] – Режим доступа до ресурсу: <https://react-redux.js.org/tutorials/quick-start>
18. React Router [Электронный ресурс] – Режим доступа до ресурсу: <https://reactrouter.com/en/main>
19. What is a RESTful API? [Электронный ресурс] – Режим доступа до ресурсу: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls
20. “npm install” [Электронный ресурс] – Режим доступа до ресурсу: <https://www.npmjs.com/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						63

Схема моделей

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

datasource db {
  provider = "mysql"
  url      = "mysql://root:password@localhost:3306/hospital"
}

enum Role {
  USER
  ADMIN
  DOCTOR
  MODERATOR
  CONSULTANT
}

model User {
  id          Int      @id @default(autoincrement())
  email       String
  username    String
  firstName   String
  lastName    String
  patronymic  String
  doctorProfile Doctor?
  Orders      Order[]

  role        Role      @default(USER)
  password     String
  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt
  Appointment Appointment[]

  @@unique([email, username])
}

model Post {
  id      Int      @id @default(autoincrement())
  title   String
  body    String   @db.Text
  image   String?

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Doctor {
```

```

    id            Int            @id @default(autoincrement())
    specializations Specialization[] @relation("DoctorsSpecializations")
    experience     Int?
    cabinet        String?
    phone          String?
    photo          String?
    user           User?          @relation(fields: [userId], references: [id])
    userId         Int?           @unique
    createdAt      DateTime       @default(now())
    updatedAt      DateTime       @updatedAt
    Appointment    Appointment[]
}

model Specialization {
    id            Int            @id @default(autoincrement())
    name          String
    doctors       Doctor[] @relation("DoctorsSpecializations")
    createdAt     DateTime       @default(now())
    updatedAt     DateTime       @updatedAt
}

model Service {
    id            Int            @id @default(autoincrement())
    name          String
    description    String
    price         Int
    Package       Package[] @relation("ServicePackage")
    Order         Order[]
    createdAt     DateTime       @default(now())
    updatedAt     DateTime       @updatedAt
    Appointment    Appointment[]
}

model Package {
    id            Int            @id @default(autoincrement())
    name          String
    description    String
    price         Int
    image         String?
    createdAt     DateTime       @default(now())
    updatedAt     DateTime       @updatedAt
    Service       Service[] @relation("ServicePackage")
    Order         Order[]
}

model Appointment {
    id            Int            @id @default(autoincrement())
    date          DateTime?
    isPending     Boolean? @default(true)
    isDone        Boolean? @default(false)
    doctor        Doctor? @relation(fields: [doctorId], references: [id])

```

```

    doctorId  Int?
    user      User?    @relation(fields: [userId], references: [id])
    userId    Int?
    service   Service? @relation(fields: [serviceId], references: [id])
    serviceId Int?
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt
}

model Order {
  id          Int        @id @default(autoincrement())
  isPending   Boolean?   @default(true)
  isDone      Boolean?   @default(false)
  date        DateTime   @db.Date
  package     Package    @relation(fields: [packageId], references: [id])
  packageId   Int
  Service     Service?   @relation(fields: [serviceId], references: [id])
  serviceId   Int?
  user        User       @relation(fields: [userId], references: [id])
  userId      Int
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt
}

```

ДОДАТОК Б

Файл маршрутизації

```

import express from "express";
import postRouter from "./newsRoutes.js";
import userRouter from "./userRoutes.js";
import authRouter from "./authRoutes.js";
import doctorRouter from "./doctorRoutes.js";
import serviceRouter from "./serviceRoutes.js";
import packageRouter from "./packageRoutes.js";
import specializationRouter from "./specializationRoutes.js";
import appointmentRouter from "./appointmentRoutes.js";
import orderRouter from "./orderRoutes.js";
import { verifyToken } from "../middleware/authJwt.js";
const router = express.Router();

router.use('/posts', postRouter);
router.use('/services', serviceRouter);
router.use('/packages', packageRouter);
router.use('/doctors', doctorRouter);
router.use('/specializations', specializationRouter);
router.use('/appointments', appointmentRouter);
router.use('/orders', orderRouter);
router.use('/users', userRouter);
router.use('/auth', authRouter);
router.use(verifyToken)

```

Файл маршрутизації послуг

```
import express from 'express';
import {
  getServices,
  getServiceById,
  getServicesByPage,
  createService,
  updateService,
  deleteService,
} from '../api/serviceController.js';

const router = express.Router();

router.get('/all', getServices);
router.get('/', getServicesByPage);
router.get('/:id', getServiceById);
router.post('/', createService);
router.patch('/', updateService);
router.delete('/', deleteService);

export default router;
```

Контролер послуг

```
import { PrismaClient } from "@prisma/client";
const prisma = new PrismaClient();

export const getServices = async (req, res) => {
  try {
    const services = await prisma.service.findMany();
    res.status(200).json(services);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
};

export const getServiceById = async (req, res) => {
  const { id } = req.params;
  try {
    const service = await prisma.service.findUnique({
      where: {
        id: Number(id)
      }
    });
    res.status(200).json(service);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const getServicesByPage = async (req, res) => {
```

```

const perPage = 10
const page = req.query.page || 1
const search = req.query.search || ''
const total = await prisma.service.count()
try {
  const data = await prisma.service.findMany({
    take: perPage,
    skip: Math.max(perPage * (page - 1), 0),
    where: {
      name: { contains: search },
    },
    orderBy: {
      createdAt: 'desc',
    },
  })
  res.status(200).json({
    data,
    page,
    totalPages: Math.ceil(total / perPage),
    total,
  })

} catch (error) {
  res.status(500).json({ msg: error.message });
}

export const createService = async (req, res) => {
  const { name, description, price } = req.body;
  try {
    const service = await prisma.service.create({
      data: {
        name: name,
        description: description,
        price: Number(price),
      }
    });
    res.status(201).json(service);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const updateService = async (req, res) => {
  const { id, name, description, price } = req.body;
  try {
    const service = await prisma.service.update({
      where: {
        id: Number(id)
      },
      data: {

```

```

        name: name,
        description: description,
        price: Number(price),
      }
    });
    res.status(200).json(service);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const deleteService = async (req, res) => {
  const { id } = req.body;
  try {
    const service = await prisma.service.delete({
      where: {
        id: Number(id)
      }
    });
    res.status(200).json(service);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const getPackages = async (req, res) => {
  try {
    const servicePackages = await prisma.package.findMany();
    res.status(200).json(servicePackages);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const getPackagesByPage = async (req, res) => {
  const perPage = 10
  const page = req.query.page || 1
  const search = req.query.search || ''
  const total = await prisma.package.count()
  try {
    const data = await prisma.package.findMany({
      take: perPage,
      skip: Math.max(perPage * (page - 1), 0),
      orderBy: {
        createdAt: 'desc',
      },
      where: {
        name: { contains: search },
      },
    })
    res.status(200).json({

```

```

        data,
        page,
        totalPages: Math.ceil(total / perPage),
        total,
    })
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const getPackageByName = async (req, res) => {
  const { name } = req.params;
  try {
    const servicePackage = await prisma.package.findUnique({
      where: {
        name: name
      },
      include: {
        Service: true
      }
    });
    res.status(200).json(servicePackage);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const getPackageById = async (req, res) => {
  const { id } = req.params;
  try {
    const servicePackage = await prisma.package.findUnique({
      where: {
        id: Number(id)
      },
      include: {
        Service: true
      }
    });
    res.status(200).json(servicePackage);
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

export const createPackage = async (req, res) => {
  try {
    const { name, description, price, services } = req.body;
    const JSONservices = JSON.parse(services);
    const image = req?.file?.filename;
    let servicePackage;
    if (!image) {

```

```

        servicePackage = await prisma.package.create({
          data: {
            name: name,
            description: description,
            price: Number(price),
            Service: {
              connect: JSONservices?.map((service) => {
                return {
                  id: service.id
                };
              })
            }
          }
        });
      } else {
        servicePackage = await prisma.package.create({
          data: {
            name: name,
            description: description,
            price: Number(price),
            image: image,
            Service: {
              connect: JSONservices?.map((service) => {
                return {
                  id: service.id
                };
              })
            }
          }
        });
      }
    } catch (error) {
      res.status(500).json({ msg: error.message });
    }
  }

export const updatePackage = async (req, res) => {
  try {
    const { id, name, description, price, services } = req.body;
    const image = req?.file?.filename;
    const JSONservices = JSON.parse(services);
    let servicePackage;
    if (!image) {
      servicePackage = await prisma.package.update({
        where: {
          id: Number(id)
        },
        data: {
          name: name,
          description: description,

```

```

        price: Number(price),
        Service: {
          set: JSONservices?.map((service) => {
            return {
              id: service.id
            };
          }),
        },
      }
    }
  });
} else {
  servicePackage = await prisma.package.update({
    where: {
      id: Number(id)
    },
    data: {
      name: name,
      description: description,
      price: Number(price),
      image: image,
      Service: {
        set: JSONservices?.map((service) => {
          return {
            id: service.id
          };
        }),
      },
    }
  });
}
res.status(200).json(servicePackage);
} catch (error) {
  res.status(500).json({ msg: error.message });
}
}

export const deletePackage = async (req, res) => {
  const { id } = req.body;
  try {
    const servicePackage = await prisma.package.delete({
      where: {
        id: Number(id)
      }
    });
  } catch (error) {
    res.status(500).json({ msg: error.message });
  }
}

```

Файл API послуг

```

import { apiSlice } from "../../api/apiSlice"

export const packagesApiSlice = apiSlice.injectEndpoints({
  endpoints: builder => ({
    getPackages: builder.query({
      query: () => ({
        url: `/packages`,
        method: 'GET'
      }),
      providesTags: (result, error, arg) => {
        if (result?.ids) {
          return [
            { type: 'Package', id: 'LIST' },
            ...result.ids.map(id => ({ type: 'Package', id }))
          ]
        } else return [{ type: 'Package', id: 'LIST' }]
      }
    }),
    getPackagesList: builder.query({
      query: (query) => ({
        url: `/packages?page=${query.page}&search=${query.search}`,
        method: 'GET'
      }),
      providesTags: (result, error, arg) => {
        if (result?.ids) {
          return [
            { type: 'Package', id: 'LIST' },
            ...result.ids.map(id => ({ type: 'Package', id }))
          ]
        } else return [{ type: 'Package', id: 'LIST' }]
      }
    }),
    getPackageById: builder.query({
      query: (id) => ({
        url: `/packages/${id}`,
        method: 'GET',
      }),
      providesTags: (result, error, arg) => {
        if (result?.ids) {
          return [
            { type: 'Package', id: 'LIST' },
            ...result.ids.map(id => ({ type: 'Package', id }))
          ]
        } else return [{ type: 'Package', id: 'LIST' }]
      }
    }),
    getServices: builder.query({

```

```

        query: () => ({
            url: `/services/all`,
            method: 'GET'
        }),
        providesTags: (result, error, arg) => {
            return [{ type: 'Service', id: 'LIST' }]
        }
    }),
    getServicesList: builder.query({
        query: (query) => ({
            url: `/services?page=${query.page}&search=${query.search}`,
            method: 'GET'
        }),
        providesTags: (result, error, arg) => {
            return [{ type: 'Service', id: 'LIST' }]
        }
    }),
    getServiceById: builder.query({
        query: (id) => ({
            url: `/services/${id}`,
            method: 'GET',
        }),
        providesTags: (result, error, arg) => {
            if (result?.ids) {
                return [
                    { type: 'Service', id: 'LIST' },
                    ...result.ids.map(id => ({ type: 'Service', id }))
                ]
            } else return [{ type: 'Service', id: 'LIST' }]
        }
    }),
    createPackage: builder.mutation({
        query: data => ({
            url: '/packages',
            method: 'POST',
            body: {
                ...data,
            }
        }),
        invalidatesTags: (result, error, arg) => [{ type: 'Package', id: 'LIST'
    ]
    }),
    createService: builder.mutation({
        query: data => ({
            url: '/services',
            method: 'POST',
            body: {
                ...data,
            }
        }),
    }),

```

```

        invalidatesTags: (result, error, arg) => [{ type: 'Service', id: 'LIST'
    ]}
  }},
  updatePackage: builder.mutation({
    query: data => ({
      url: '/packages',
      method: 'PATCH',
      body: {
        ...data,
      }
    }),
    invalidatesTags: (result, error, arg) => [{ type: 'Package', id: 'LIST'
  ]}
  }},
  updateService: builder.mutation({
    query: data => ({
      url: '/services',
      method: 'PATCH',
      body: {
        ...data,
      }
    }),
    invalidatesTags: (result, error, arg) => [{ type: 'Service', id: 'LIST'
  ]}
  }},
  deletePackage: builder.mutation({
    query: id => ({
      url: `/packages`,
      method: 'DELETE',
      body: { id }
    }),
    invalidatesTags: (result, error, arg) => [{ type: 'Package', id: 'LIST'
  ]}
  }},
  deleteService: builder.mutation({
    query: id => ({
      url: `/services`,
      method: 'DELETE',
      body: { id }
    }),
    invalidatesTags: (result, error, arg) => [{ type: 'Service', id: 'LIST'
  ]}
  })),
  })),
  })
})

export const {
  useGetPackagesQuery,
  useGetPackagesListQuery,
  useGetPackageByIdQuery,
  useCreatePackageMutation,

```

```

    useUpdatePackageMutation,
    useDeletePackageMutation,

    useGetServicesQuery,
    useGetServicesListQuery,
    useGetServiceByIdQuery,
    useCreateServiceMutation,
    useUpdateServiceMutation,
    useDeleteServiceMutation,
  } = packagesApiSlice

```

Сторінка послуги

```

import { useGetServiceByIdQuery } from "../packagesApiSlice"
import { useParams } from "react-router-dom"
import { useNavigate } from "react-router-dom"
import useAuth from "../../hooks/useAuth"

const Service = () => {
  const { id } = useParams()
  const {
    data: service,
    isFetching,
    isSuccess,
  } = useGetServiceByIdQuery(id, { refetchOnMountOrArgChange: true,
    refetchOnReconnect: true })

  const { isModerator } = useAuth()

  const navigate = useNavigate()

  const handleEditService = () => {
    navigate(`/packages/services/${id}/edit`)
  }

  let content

  if (isFetching) content = <p>Fetching...</p>

  if (isSuccess) content = (
    <>
      <h2>{service?.name}</h2>
      <ul>
        <li>
          {isModerator && (
            <button onClick={handleEditService}>Редагувати</button>
          )}
          <p>{service?.price}</p>
          <p>{service?.description}</p>
        </li>
      </ul>
    </>
  )

```

```

    )

    return content
}

export default Service

```

Сторінка список послуг

```

import { useGetServicesListQuery } from "../packagesApiSlice"
import { useState, useEffect } from "react"
import { useNavigate } from "react-router-dom"
import useAuth from "../../hooks/useAuth"

const ServiceList = () => {
  const [page, setPage] = useState(1)
  const [search, setSearch] = useState("")
  const query = { page: page, search: search }

  const {
    data: services,
    isFetching,
    isSuccess,
  } = useGetServicesListQuery(query, {
    refetchOnMountOrArgChange: true,
    refetchOnReconnect: true,
    refetchOnFocus: true,
  })

  const [totalPages, setTotalPages] = useState(services?.totalPages)
  useEffect(() => {
    if (search !== "") {
      setPage(1)
      setTotalPages(1)
    } else {
      setTotalPages(services?.totalPages)
    }
  }, [search, services?.totalPages])
  const { isModerator } = useAuth()
  const navigate = useNavigate()
  const handlePackage = (event, id) => {
    event.preventDefault()
    navigate(`/packages/services/${id}`)
  }

  const handleAddService = () => {
    navigate(`/packages/services/create`)
  }

  const handleSearch = (event) => {
    event.preventDefault()
    setSearch(event.target.value)
  }
}

```

```

let content
if (isFetching) content = <p>Fetching...</p>
if (isSuccess) content = (
  <>
    <h2>Список послуг</h2>
    {(isModerator) &&
      <>
        <button onClick={handleAddService}>Додати послугу</button>
      </>
    }
    <ul>
      <input
        type="text"
        placeholder="Пошук по назві"
        value={search}
        onChange={handleSearch}
        ref={input => input && input.focus()}
        autofocus
      />
      <button
        onClick={() => setPage(page - 1)}
        isLoading={isFetching}
        disabled={page === 1}
      >
        Попередня сторінка
      </button>
      <button
        onClick={() => setPage(page + 1)}
        isLoading={isFetching}
        disabled={page === totalPages}
      >
        Наступна сторінка
      </button>
      `${page} / ${totalPages}`
      {services?.data?.map(({ id, name, price }) => (
        <li key={id}>
          {name} {price}
          <button onClick={e => handlePackage(e, id)}>
            Детальніше
          </button>
        </li>
      ))}
    </ul>
  </>
)

return content
}

export default ServiceList

```

Сторінка редагування послуги

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { useParams } from "react-router-dom";
import {
  useGetServiceByIdQuery,
  useUpdateServiceMutation,
  useDeleteServiceMutation,
} from "../packagesApiSlice";
const EditService = () => {
  const { id } = useParams();
  const navigate = useNavigate();

  const {
    data: service,
    isFetching,
    isSuccess,
  } = useGetServiceByIdQuery(id, {
    refetchOnMountOrArgChange: true,
    refetchOnReconnect: true,
  });

  const updateService = useUpdateServiceMutation();
  const deleteService = useDeleteServiceMutation();
  const [name, setName] = useState(service?.name);
  const [price, setPrice] = useState(service?.price);
  const [description, setDescription] = useState(service?.description);
  const handleName = (event) => setName(event.target.value);
  const handlePrice = (event) => setPrice(event.target.value);
  const handleDescription = (event) => setDescription(event.target.value);

  const handleUpdate = (event) => {
    event.preventDefault();
    updateService({
      id,
      name,
      price,
      description,
    });
  };

  const handleDelete = (event) => {
    event.preventDefault();
    deleteService(id);
    navigate("/packages/services");
  };

  let content;
  if (isFetching) content = <p>Fetching...</p>;
  if (isSuccess) content = (
    <>
```

```

    <h2>Редагувати послугу</h2>
    <form onSubmit={handleUpdate}>
      <label htmlFor="name">Назва</label>
      <input
        type="text"
        value={name}
        onChange={handleName}
      />
      <label htmlFor="price">Ціна</label>
      <input
        type="text"
        value={price}
        onChange={handlePrice}
      />
      <label htmlFor="description">Опис</label>
      <input
        type="text"
        value={description}
        onChange={handleDescription}
      />
      <button type="submit">Зберегти</button>
    </form>
    <button onClick={handleDelete}>Видалити</button>
  </>
);
return content;
}
export default EditService;

```

Сторінка створення послуги

```

import { useCreateServiceMutation } from "../packagesApiSlice";
import { useState } from "react";
import { useNavigate } from "react-router-dom";

const CreateService = () => {
  const [name, setName] = useState("");
  const [description, setDescription] = useState("");
  const [price, setPrice] = useState("");
  const navigate = useNavigate();

  const [createService] = useCreateServiceMutation();

  const handleName = (event) => setName(event.target.value);
  const handleDescription = (event) => setDescription(event.target.value);
  const handlePrice = (event) => setPrice(event.target.value);

  const handleCreate = (event) => {
    event.preventDefault();
    createService({
      name,

```

					ІПЗ.КР.Б – 121 – 23 – ІІЗ	
						80

```

        description,
        price,
    });
    navigate("/packages");
}

let content;

content = (
    <>
        <h2>Створення нової послуги</h2>
        <form onSubmit={handleCreate}>
            <label htmlFor="name">Назва</label>
            <input
                id="name"
                type="text"
                value={name}
                onChange={handleName}
            />
            <label htmlFor="description">Опис</label>
            <input
                id="description"
                type="text"
                value={description}
                onChange={handleDescription}
            />
            <label htmlFor="price">Ціна</label>
            <input
                id="price"
                type="number"
                value={price}
                onChange={handlePrice}
            />
            <button type="submit">Створити</button>
        </form>
    </>
)
return content;
}

export default CreateService;

```