МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Веб-орієнтована система медичного закладу для запису до лікарів та управління записами клієнтів»

здобувачки групи ІН - 01 Новікової Катерини Дмитрівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Катерина НОВІКОВА

(підпис)

Керівник, Альона МОСКАЛЕНКО

доцент кафедри комп'ютерних наук,

кандидат технічних наук

(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Новікової Катерини Дмитрівни

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від *«22» квітня 2024 року № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для створення веб-сервісів медичних закладів. 3) Проектування бази даних. 4) Розробка макетів інтерфейсу. 5) Розробка веб-орієнтованої системи медичного закладу для запису до лікарів та управління записами клієнтів. 6) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	08.04.24-10.04.24	
2	*Огляд технологій, що використовуються для створення веб-сервісів медичних закладів.*	11.04.24-13.04.24	
3	*Проектування бази даних.*	14.04.24	
4	*Розробка макетів інтерфейсу*	15.04.24	
3	*Розробка веб-орієнтованої системи медичного закладу для запису до лікарів та управління записами клієнтів.*	16.04.24-11.05.24	
4	*Аналіз отриманих результатів*	12.05.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	13.05.24-16.05.24	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 64 стр., 30 рис., 3 додатки, 20 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі забезпечення пацієнтам зручного доступу до медичних послуг, автоматизації процесів управління записами.

Об'єкт дослідження — інформаційна система медичного закладу для управління інформацією про заклад, медичний персонал та пацієнтів, забезпечення можливості запису до лікарів та управління записами клієнтів.

Мета роботи — розробка веб-орієнтованої системи для медичного закладу, яка забезпечує легкий та вільний доступ до інформації про заклад, лікарів та послуги, а також спрощує процес запису на прийом, дозволяє керувати записам клієнтів.

Методи дослідження — веб-орієнтовані технології та методи, які застосовуються для створення систем управління записами пацієнтів.

Результати — розроблено веб-орієнтовану системи, яка надає можливість переглядати дані про медичний заклад, послуги, фахівців, клієнтів, записи до лікарів, має функціональність для реєстрації та авторизації клієнтів і медичного персоналу, можливість створення і управління записами, інформацією про фахівців. Проведено тестування проекту.

ВЕБ-ОРІЄНТОВАНА СИСТЕМА, МЕДИЧНИЙ ЗАКЛАД, ЗАПИС ДО ЛІКАРІВ, JAVA, SPRING, JAVASCRIPT, REACT, REST, MVC

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій зростаюча роль відводиться автоматизації процесів у всіх сферах людської діяльності. Медична сфера не є винятком, адже якісний та оперативний доступ до медичних послуг і інформації може вирішувати питання, що впливають на здоров'я та життя людей. Незважаючи на значні досягнення у цій області, все ще існує велика потреба у вдосконаленні інформаційних систем медичних установ.

Метою дипломної роботи є розробка веб-орієнтованої системи для медичного закладу, яка забезпечує легкий та вільний доступ до інформації про заклад, лікарів та послуги, а також спрощує процес запису на прийом, дозволяє керувати записам клієнтів.

Об'єктом дослідження є інформаційна система медичного закладу для управління інформацією про заклад, медичний персонал та пацієнтів, забезпечення можливості запису до лікарів та управління записами клієнтів.

Предметом дослідження є веб-орієнтовані технології та методи, які застосовуються для створення систем управління записами пацієнтів, що включають інтерфейси для користувачів, бази даних для зберігання медичної інформації та алгоритми обробки запитів.

Використання інформаційних технологій у медицині відкриває нові можливості для забезпечення більш доступного та якісного медичного обслуговування. Розробка такого веб-сервісу є актуальним і значущим кроком вперед у цьому напрямку.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Сучасний стан

Актуальність розробки веб-сервісу для медичного закладу у сучасних умовах є високою і зумовлена низкою соціальних, технологічних і медичних чинників. Зростання вимог до якості медичного обслуговування, потреба у оптимізації робочих процесів медичних установ та підвищення доступності медичних послуг в умовах глобалізації інформаційного суспільства вимагає впровадження новітніх технологій.

Серед ключових аспектів, що зумовлюють актуальність проекту, можна виділити наступні:

1. Задоволення потреб пацієнтів.

   Сучасні пацієнти вимагають більшої гнучкості та доступності при виборі медичних послуг, що може бути досягнуто за допомогою інтерактивних онлайн платформ.
2. Підвищення ефективності роботи медичних закладів.

   Автоматизація запису на прийом і управління медичною документацією може значно знизити адміністративне навантаження на персонал.
3. Забезпечення прозорості і довіри.

   Інформаційний портал дозволяє медичному закладу ефективно інформувати пацієнтів про доступні послуги, кваліфікацію персоналу та стандарти обслуговування, тим самим підвищуючи довіру та задоволеність серед користувачів.

Таким чином, створення веб-сервісу для медичного закладу відповідає поточним викликам медичної індустрії та відкриває нові можливості для покращення якості медичного обслуговування, роблячи його більш доступним, зручним і ефективним.

1.2 Аналіз цільової аудиторії та потреб користувачів

Детальний аналіз цільової аудиторії та вивчення їх потреб забезпечує створення продукту, який точно відповідає очікуванням і вимогам користувачів, підвищуючи шанси на його успішне впровадження та ефективне використання.

Цільова аудиторія веб-сервісу медичного закладу включає:

Пацієнти: основні користувачі послуг, які потребують зручного доступу до інформації про лікарів, послуги, графіки прийому та можливість онлайн-запису на прийом.

Медичний персонал: лікарі, медсестри та адміністративний персонал, які використовують систему для управління графіками, записами пацієнтів та іншими адміністративними задачами.

Аналізуючи категорії цільової аудиторії, можна виокремити потреби кожної з них.

Пацієнти

- Зручність та доступність: швидкий доступ до інформації про лікарів та послуги, можливість легкого запису на прийом онлайн в будь-який час, перегляд історії прийомів, отримання електронних направлень та рецептів, тощо.
- Інформаційна підтримка та комунікація: доступ до інформації про кваліфікацію лікарів, ціни послуг, контактні дані лікарів та адміністративного персоналу, відгуки пацієнтів, тощо.
- Конфіденційність: гарантія захисту персональних даних і медичної інформації.

Медичний персонал

- Ефективність управління: автоматизація рутинних задач, зниження паперової роботи, централізація управління записами.

Аналіз цільової аудиторії та їх потреб є критично важливим для проектування веб-сервісу, який ефективно відповідає на виклики і очікування сучасної медичної практики. Врахування цих аспектів допоможе забезпечити високу задоволеність користувачів і оптимізацію роботи медичного персоналу.

1.3 Огляд існуючих рішень

Для створення ефективного веб-сервісу для медичного закладу, варто врахувати досвід подібних систем.

Helsi (url: https://helsi.me) — це популярна медична інформаційна система, яка надає різноманітні послуги як пацієнтам, так і медичним закладам.

Переваги

1. Для користувачів:
   - запис на прийом онлайн через веб-сайт або мобільний додаток;
   - доступ до медичної інформації, включаючи історію відвідувань, результати аналізів і медичні рекомендації;
   - нагадування про прийоми.
2. Для медичного персоналу:
   - автоматизація процесів ведення графіків прийомів, управління чергами та обробки медичних даних;
   - інструменти для генерації звітів та аналітики.
3. Загальне:
   - підтримка національних програм охорони здоров'я;
   - сумісність з лабораторіями та аптеками.

Недоліки

- перевантаження серверів: у пікові години можливі затримки у роботі системи через перевантаження серверів;

- складність навігації: інтерфейс не завжди інтуїтивно зрозумілим, особливо для літніх людей або користувачів без досвіду роботи з цифровими технологіями;
- обмежені можливості кастомізації щодо налаштування інтерфейсу під конкретні потреби медичних закладів або користувачів;
- відсутність детальної інформації про клініки, їх послуги, ціни, тощо;

Медея (url: https://medeya.sumy.ua) – веб-сервіс для медичного закладу «Медея».

Переваги

- надання детальної інформації про заклад, фахівців, послуги, ціни, контактні дані, тощо;
- викладання новин про заклад, акції та події;
- інтуїтивно зрозумілий інтерфейс.

Недоліки

- можливість запису на прийом лише за номером телефону реєстратури або сторонні ресурси (Helsi).
- можливість ведення електронних карток, надання інформації про відвідування, надання електронних рецептів, формування звітності, тощо лише за допомогою взаємодії з системою Helsi.

Vidnova (url: https://vidnova.center) – веб-сервіс для реабілітаційного центру «Vidnova».

Переваги

- надання детальної інформації про заклад, фахівців, послуги, ціни, контактні дані, відгуки, тощо;

- чат-бот у Viber для запису на прийом та перегляд своїх записів;
- викладання новин про заклад, акції та події;
- інтуїтивно зрозумілий інтерфейс;
- ведення електронних медичних карток пацієнтів.

Недоліки

- використання месенджеру для запису на прийом, що не є універсальним рішенням через необхідність встановлювати месенджер;
- відсутність підтримки національних програм охорони здоров'я.

Медсоюз (url: https://med-soyuz.com.ua) – веб-сервіс для медичного закладу «Медсоюз».

Переваги

- надання детальної інформації про заклад, фахівців, послуги, ціни, контактні дані, тощо;
- викладання новин про заклад, акції та події;
- інтуїтивно зрозумілий інтерфейс.

Недоліки

- можливість запису на прийом лише за номером телефону реєстратури;
- підтримка національних програм охорони здоров'я.

Аналізуючи наведені приклади, можна охарактеризувати Helsi як потужну систему для медичних закладів та пацієнтів, проте через навантаження (як інформаційне, так і серверне) вона переважно використовується як допоміжний інструмент разом із веб-сайтом. У свою чергу веб-сайт є ресурсом для надання детальної інформації про заклад, якої бракує в системі Helsi. Така взаємодія є

ефективною, але може бути не дуже зручною для користувачів через необхідність використання сторонньої системи для запису на прийом, що також стосується і використання месенджерів.

Оптимальним для користувачів є наявність можливості запису на прийом безпосередньо на веб-сайті медичного закладу, а для медичного персоналу – можливості переглядати та керувати записами клієнтів.

1.4 Формалізована постановка задачі

Основними завданнями цієї роботи є:

1. Спроектувати і створити БД для збереження даних, необхідних для функціонування веб-сайту.
2. Розробити інтуїтивно зрозумілий інтерфейс користувача.
3. Розробити веб-сайт для медичної установи з системою управління записами на прийом.
4. Протестувати готовий проект.

Визначення функціональних вимог до веб-сайту

Реєстрація та автентифікація користувачів:

- можливість реєстрації нових користувачів (пацієнтів та персоналу);
- система входу в систему для пацієнтів і медичного персоналу.

Інформаційна платформа:

- публікація загальної інформації про медичний заклад, такою як адреса, години роботи, контактні дані, послуги, ціни, тощо;
- розміщення інформації про медичних спеціалістів, які працюють у закладі, зі зазначенням їхніх професійних кваліфікацій та досвіду.
- можливість пошуку спеціалістів за прізвищем або спеціальністю.

Система управління записами на прийом:

- інтерфейс для запису пацієнтів на прийом до лікарів, з можливістю вибору вільного часу та спеціаліста.

Особистий кабінет користувача:

- можливість перегляду своїх записів до лікарів користувачем.

Кабінет лікаря:

- можливість переглядати записи клієнтів лікарем;

Адміністративна панель:

- інструменти для адміністраторів та медичного персоналу для управління прийомами, надання ролей користувачам, додавання інформації про лікарів.

2 ВИБІР МЕТОДІВ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель

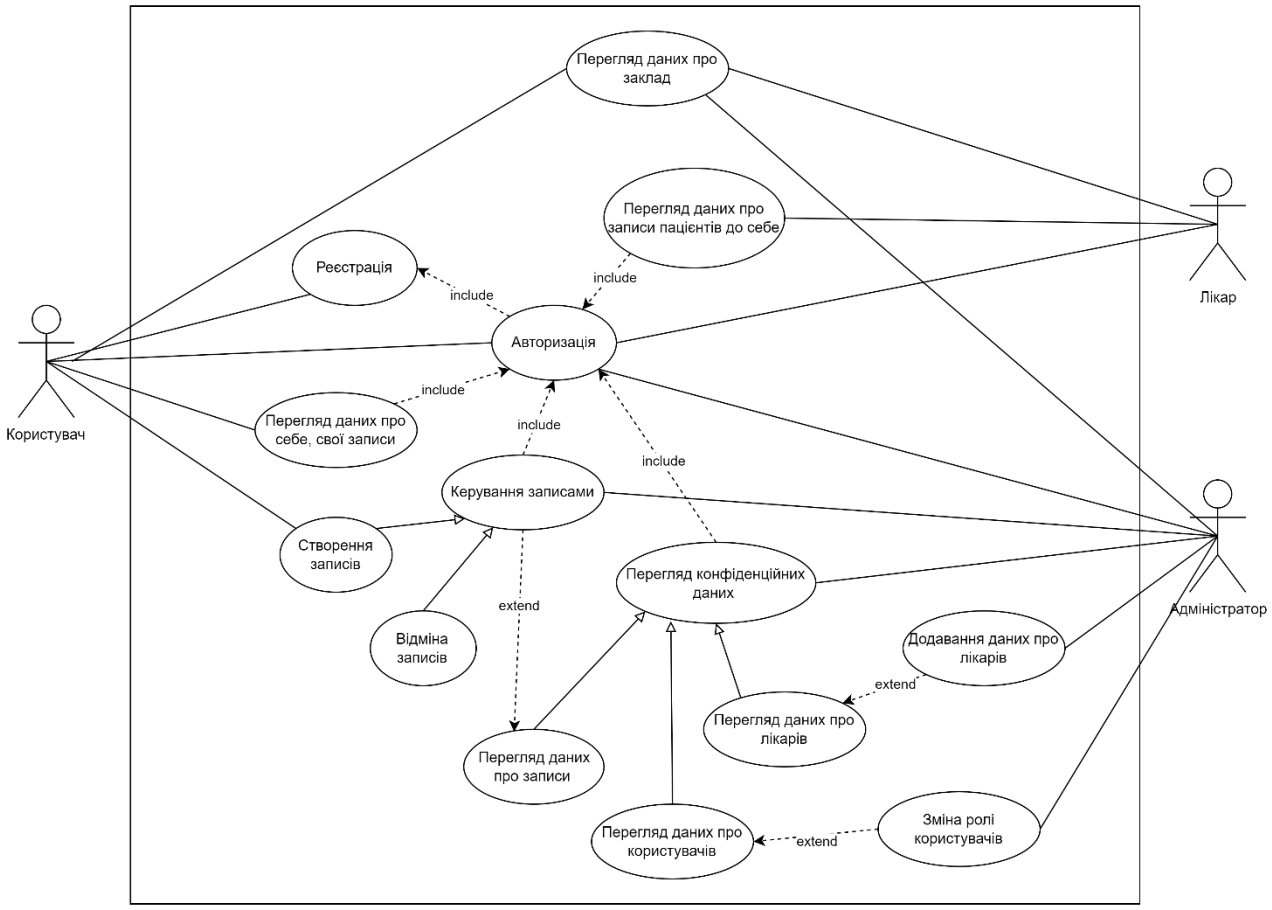


Рисунок 2.1 Use-case діаграма

Інформаційна система проекту має містити дані про:

- користувачів системи, їх ролі;
- лікарів, їх кваліфікацію;
- записи до лікарів;
- послуги, ціни на послуги.

2.2 Проектування бази даних

Структура сховища, що проектується, відображена на Рис.2.2.

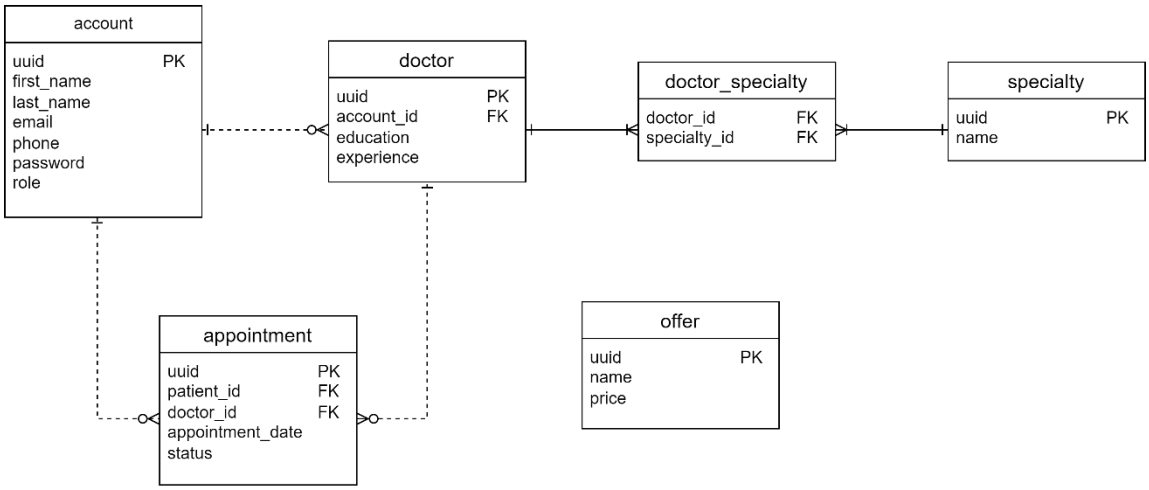


Рисунок 2.2 – ER-діаграма ІС «Медичний сервіс»

Таблиця 2.1 – Опис сутностей і атрибутів ER-діаграми

Атрибут	**Тип**	**Обмеження**
account – зберігає дані користувачів		
uuid	UUID	PK, NN
first_name	Varchar(40)	NN
last_name	Varchar(40)	NN
email	Varchar(200)	NN
phone	Varchar(12)	NN
password	Varchar(255)	NN
role	Numeric	NN, DEFAULT 0
doctor – зберігає дані лікарів		
uuid	UUID	PK, NN
account_id	UUID	FK, NN
education	Varchar(500)	
experience	Varchar(255)	
status	Boolean	NN, DEFAULT true
appointment – зберігає дані записів користувачів на прийом		
uuid	UUID	PK, NN
patient_id	UUID	FK, NN
doctor_id	UUID	FK, NN
appointment_date	Timestampz	NN

Продовження Табл. 2.1

Атрибут	**Тип**	**Обмеження**
status	Numeric	NN, Default 0
offer – зберігає дані про послуги		
uuid	UUID	PK, NN
name	Varchar(200)	NN
price	Numeric	NN
doctor_specialty – зберігає зв’язок лікаря зі спеціальностями		
doctor_id	UUID	PFK
specialty_id	UUID	PFK

Текст файлу міграції для створення таблиць БД наведено в Додатку А.

2.3 Архітектура застосунку

Веб-застосунок буде побудовано на основі REST API.

REST API (або RESTful API) — це інтерфейс прикладного програмування, який дотримується принципів архітектурного стилю REST і дозволяє взаємодіяти з RESTful веб-службами. API визначає набір правил та протоколів для створення та інтеграції програмного забезпечення. REST не є протоколом або стандартом, а представляє собою набір архітектурних обмежень, які розробники можуть реалізувати різними способами. Щоб API вважався RESTful, він повинен відповідати таким критеріям:

1. Клієнт-серверна архітектура, в якій система розділена на клієнти та сервери з взаємодією через HTTP.
2. Безстановий клієнт-серверний зв'язок, де дані про клієнта не зберігаються між окремими запитами.
3. Кешування даних.
4. Уніфікований інтерфейс: API має використовувати стандартні методи (наприклад, GET, POST, PUT, DELETE) та формат даних (наприклад,

JSON або XML) для доступу до ресурсів. ідентифікацію та відділення запитуваних ресурсів від представлень, які надсилаються клієнту;

5. Багаторівнева система: складається з серверів різних типів, ієрархічно організованих і невидимих для клієнта [15].

Архітектура REST API часто використовується разом з архітектурним підходом MVC (Model-View-Controller) у розробці веб-додатків, пропонуючи чітку організацію та розподіл відповідальності між компонентами системи.

Архітектурний підхід MVC (Model-View-Controller) використовується для розділення компонентів програми на три основні частини, що дозволяє краще організувати та керувати кодом:

- модель (Model): відповідає за обробку даних та бізнес-логіку. Вона представляє структуру та операції, що виконуються над даними, і реагує на запити від контролера, забезпечуючи доступ до даних та виконання необхідних операцій;
- представлення (View): відповідає за відображення даних моделі користувачеві. Вона представляє інформацію у вигляді, зрозумілому та зручному для користувача. Представлення реагує на зміни у моделі та оновлює відображення даних;
- контролер (Controller): відповідає за взаємодію з користувачем та координує роботу моделі та представлення. Контролер приймає вхідні дані від користувача та передає їх моделі для обробки. Він також відповідає за оновлення представлення після змін у моделі [10].

Розділення логіки додатку на легкі для розуміння та управління компоненти полегшує розробку, тестування та підтримку коду, а також сприяє його повторному використанню та модифікації. Цей підхід особливо корисний для веб-додатків, де важлива чітка організація та взаємодія між компонентами.

2.4 Мови програмування

Для реалізації даного архітектурного підходу буде використано мову програмування Java, зокрема фреймворк Spring. Spring - це потужний фреймворк у світі розробки програмного забезпечення, який надає широкі можливості для створення різноманітних веб-ресурсів та додатків.

Основні можливості Spring, необхідні для створення веб-ресурсів:

- Spring MVC (Model-View-Controller): модуль, який дозволяє легко створювати веб-додатки за архітектурним підходом MVC. Він надає інструменти для організації коду за принципами цієї архітектури [13];
- Spring Boot: інструмент для швидкої розробки веб-додатків, який дозволяє автоматизувати багато рутинних завдань та налаштувань, що допомагає розпочати роботу над проектом максимально швидко;
- Spring Security: модуль відповідає за забезпечення безпеки вашого веб-ресурсу, дозволяючи налаштовувати автентифікацію, авторизацію та управління правами доступу;
- Spring Data: модуль спрощує взаємодію з базами даних, надаючи спеціальні інструменти та абстракції для роботи з різними Системами Управління Базами Даних (СУБД) [17].

Spring Framework дозволяє розробникам створювати потужні, масштабовані та безпечні веб-додатки за допомогою широкого спектру інструментів та модулів, що спрощує розробку, підтримку та розширення таких систем.

Для створення клієнтської частини веб-сервісу використовуватиметься стандартний набір мов програмування: HTML для розмітки, CSS для оформлення та JavaScript для створення функціональності веб-сторінки. Ці

мови допоможуть створити зручний інтерфейс користувача, що є важливим критерієм для ефективності веб-сервісу.

Також для розробки користувацького інтерфейсу буде використано бібліотеку React, яка надає багато переваг, таких як:

- компонентна структура: React дозволяє будувати веб-додатки за допомогою компонентів - невеликих, відокремлених частин інтерфейсу, які можна повторно використовувати та збирати разом;
- віртуальний DOM (Document Object Model): React використовує віртуальний DOM, що дозволяє ефективно маніпулювати та оновлювати інтерфейс, перезавантажуючи тільки необхідні елементи при зміні стану додатку;
- односторінкові додатки (SPA): React дозволяє легко створювати односторінкові додатки, які забезпечують швидку навігацію без перезавантаження сторінок;
- розширюваність: є можливість використовувати різні бібліотеки та розширення для покращення функціональності додатку;
- спрощена синтаксична структура: JSX - розширення синтаксису JavaScript, що дозволяє писати HTML-подібний код безпосередньо в JavaScript, що полегшує розробку та розуміння коду [20].

2.5 База данних

Для роботи з базами даних було обрано PostgreSQL. Ця СУБД володіє значним набором переваг, серед яких ключовими для використання є:

- ефективні механізми транзакцій та реплікацій;
- підтримка роботи з даними у JSON-форматі;
- система вбудованих мов програмування, зокрема мови Java [11].

Важливо відзначити, що Spring має потужні інструменти для забезпечення надійної функціональності програми на основі архітектури MVC з використанням PostgreSQL.

2.6 Середовище розробки

Для розробки backend-частини використовуватиметься Intellij Idea - інтегроване середовище розробки (IDE), яке підтримує велику кількість мов програмування, включаючи Java, JavaScript, Python. Містить велику кількість інструментів для забезпечення зручного та швидкого процесу розробки, а саме:

- розумна система підтримки кодування: розумна система автозавершення коду, яка заснована на контексті та інтенціях користувача, не просто на синтаксисі. Це значно прискорює процес написання коду та знижує ймовірність помилок;
- глибока інтеграція з Java і JVM мовами: IntelliJ IDEA оптимізована для розробки в екосистемі Java, включаючи підтримку мов, що працюють на JVM, таких як Scala, Kotlin та інші. Це робить її ідеальним вибором для проектів, що використовують ці технології;
- вбудовані інструменти для розробки: IntelliJ IDEA має вбудовані інструменти для налагодження коду, тестування та управління залежностями, що включають Maven та Gradle, забезпечуючи комплексний підхід до управління проектами;
- підтримка баз даних і SQL;
- інтеграція з Git [5].

Intellij IDEA є потужним середовищем розробки, але може займати велику частину пам’яті при роботі з кількома проєктами одночасно. Для розробки frontend частини не потрібно так багато функціональних можливостей, тому для неї буде використано більш легковагу Visual Studio Code. Його перевагами є:

- легкість та швидкість: VS Code є легким і швидким редактором, що ідеально підходить для розробки фронтенду;
- багатий вибір розширень: велика кількість доступних розширень дозволяє легко інтегрувати підтримку різних фреймворків і бібліотек, таких як React, Angular і Vue.js;
- інтеграція з Git;
- інструменти для налагодження коду: VS Code надає вбудовані інструменти для налагодження, що дозволяють розробникам відстежувати та усувати помилки в коді [9].

2.7 Структура проекту

Проект складатиметься з двох додатків:

- backend додаток на основі SpringBoot (сервер), що буде обробляти HTTP запити, надсилати запити до бази даних і повертати дані клієнту;
- frontend додаток на основі React (клієнт), що надсилатиме HTTP запити, оброблятиме і відображатиме отримані із сервера дані.

Для зберігання даних буде використано PostgreSQL.

3 ПРАКТИЧНИЙ РОЗДІЛ

3.1 Розробка бекенду (Java, SpringBoot)

3.1.1 Налаштування додатку

Для спрощення налаштування проекту було використано онлайн інструмент Spring Initializr (https://start.spring.io). У ньому необхідно обрати інструмент для складання проекту, мову програмування та її версію, назвати проект, додати залежності. Після цього експортувати файл і відкрити в Intellij IDEA. Приклад заповнення зображено на Рис. 3.1

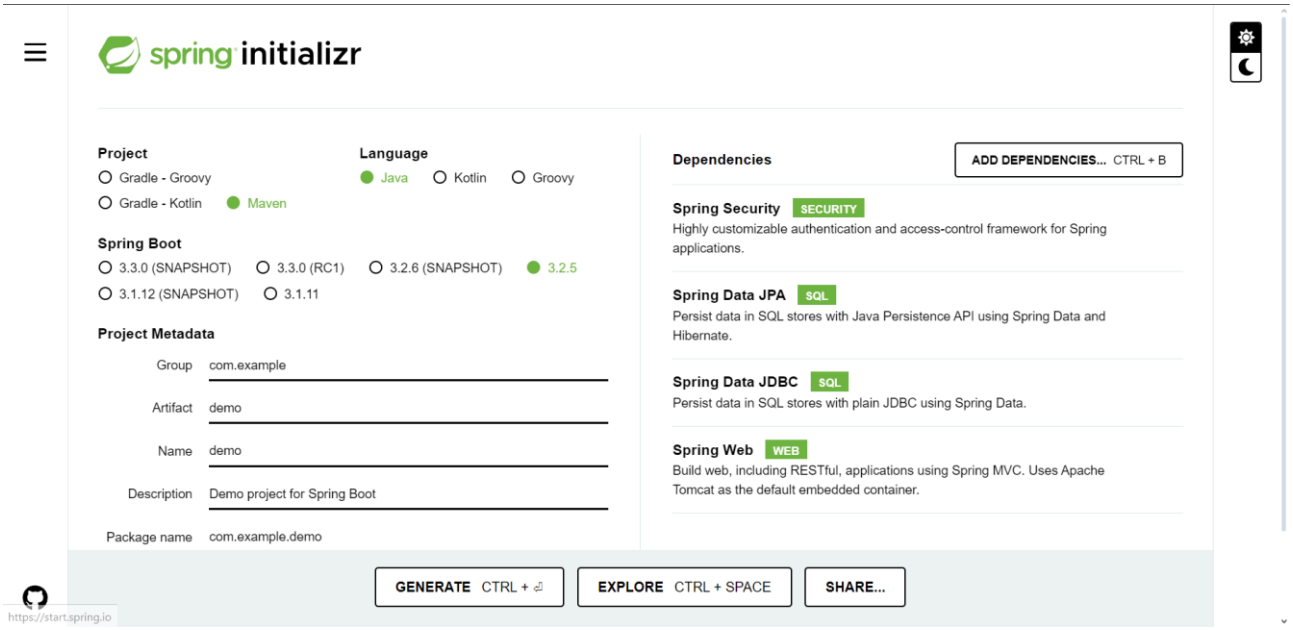


Рисунок 3.1 – Spring Initializr

Перед початком розробки була створена БД «med_application_db» з власником «postgres». Для налаштування підключення додатку до БД додано конфігурацію в application.yml файл.

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/med_application_db
    username: postgres
    password: katya
  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
```

Рисунок 3.2 – Конфігурація підключення до БД в application.yml

3.1.2 Структура додатку

Структура проекту ділиться на декілька папок

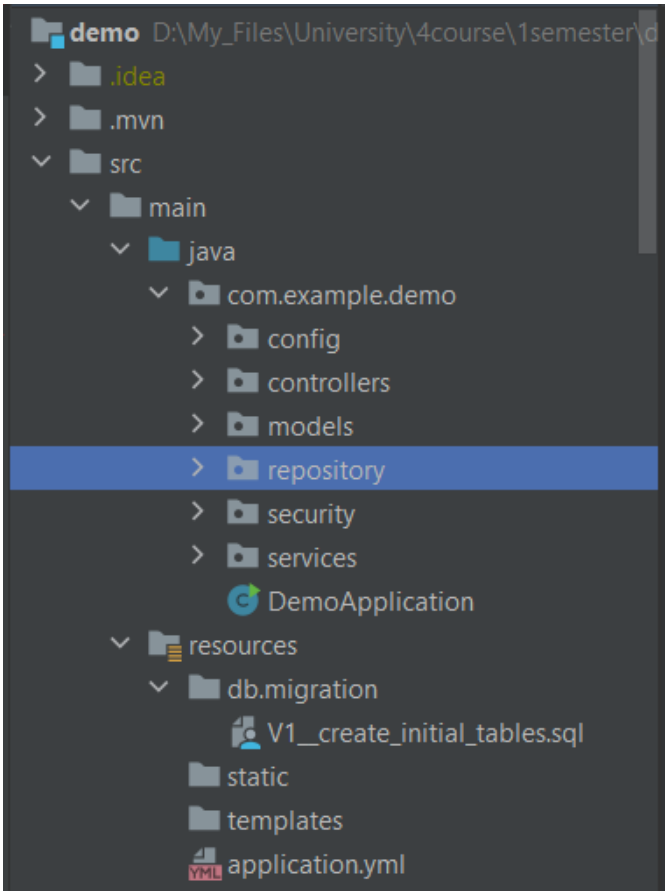


Рисунок 3.3 – Структура бекенд додатку.

Пакет «models» містить класи, що відображають структуру даних, які використовуються додатком, і зазвичай відповідають за опис таблиць бази даних у вигляді об'єктів. Окрім звичайних моделей, в цьому пакеті також містяться пакети «enum», «request», «response».

Пакет «enum» містить перелічення, які визначають набір констант або обмежених значень, які можуть використовуватися в аплікації. Це можуть бути статуси, типи ролей, тощо.

Пакет «request» містить рекорди, які використовуються для структурування даних, отриманих від клієнта через HTTP запити.

Пакет «response» містить рекорди для форматування даних, які будуть відправлені клієнту від сервера. Відповіді часто містять поля, які слід відображати, та можуть включати додаткові метадані, статус операції, а також фактичні дані або повідомлення про помилки.

Пакет «controllers» містить класи, які обробляють вхідні HTTP запити, взаємодіють з сервісами для отримання або зміни даних і повертають відповіді користувачам.

Пакет «services» містить інтерфейси та класи бізнес-логіки, які відокремлюють контролери від прямої взаємодії з моделями. Сервіси керують складними бізнес-операціями та транзакціями.

Пакет «repository» містить класи та інтерфейси для доступу до бази даних. Ці класи абстрагують взаємодію з джерелами даних. Репозиторії розширюють клас JpaRepository і використовуються для виконання CRUD операцій для спрощення реалізації доступу до даних.

Пакет «security» містить конфігурацію та класи, які забезпечують безпеку додатку. Це включає аутентифікацію, авторизацію, захист від загроз та керування сесіями користувачів.

Пакет «config» містить конфігураційні класи, які визначають налаштування додатку.

Пакет «db.migration» містить файли міграцій БД.

Приклади класів представлено в Додатку Б.

Повний код backend додатку наведено в проекті demo на GitHub [19].

3.2 Розробка фронтенду (HTML, CSS, JavaScript, React)

3.2.1 Розробка макетів інтерфейсу

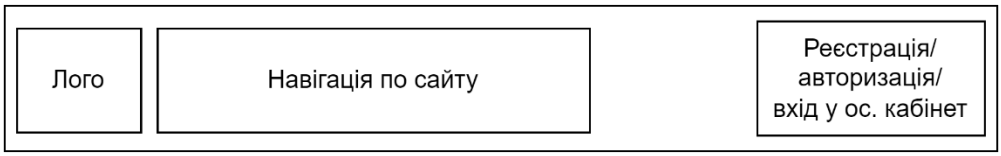


Рисунок 3.4 – Макет Хідера

Рисунок 3.5 – Макет Футера

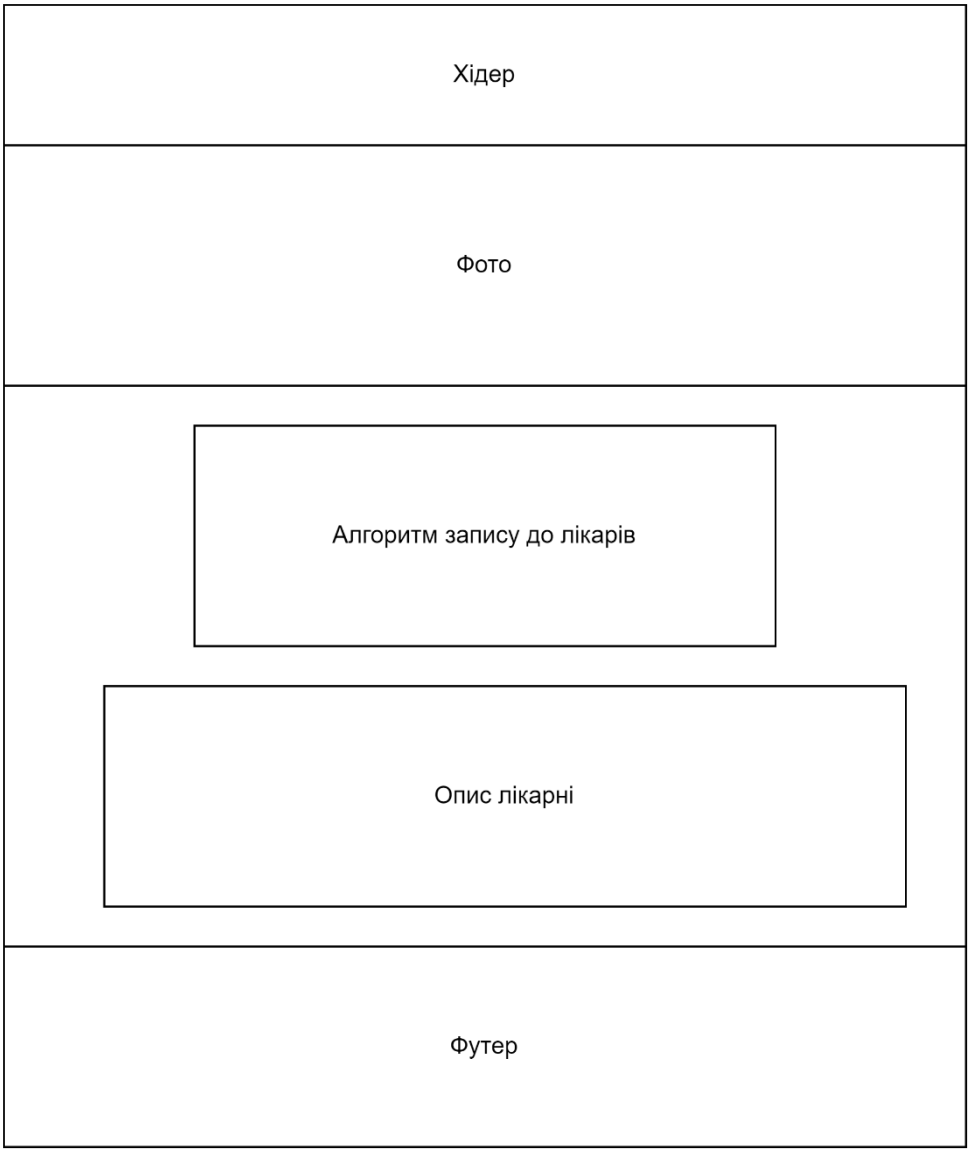


Рисунок 3.6 – Макет Головної сторінки

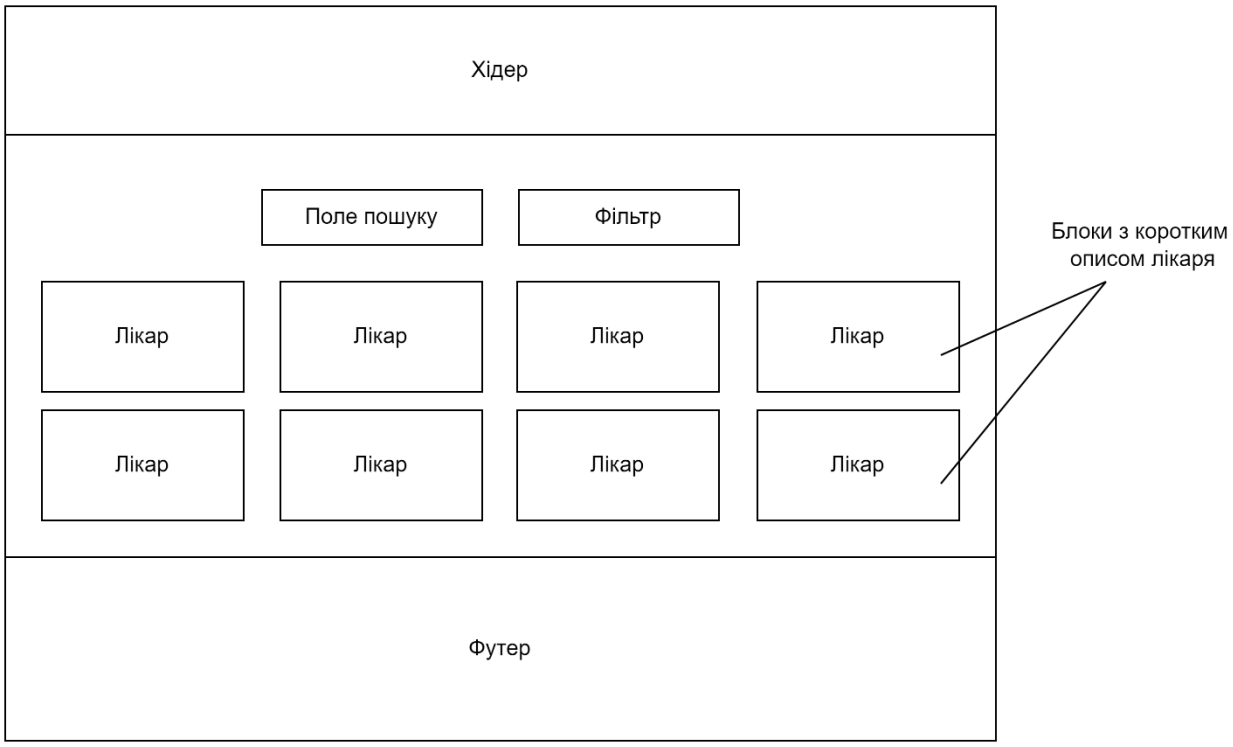


Рисунок 3.7 – Макет сторінки Фахівці

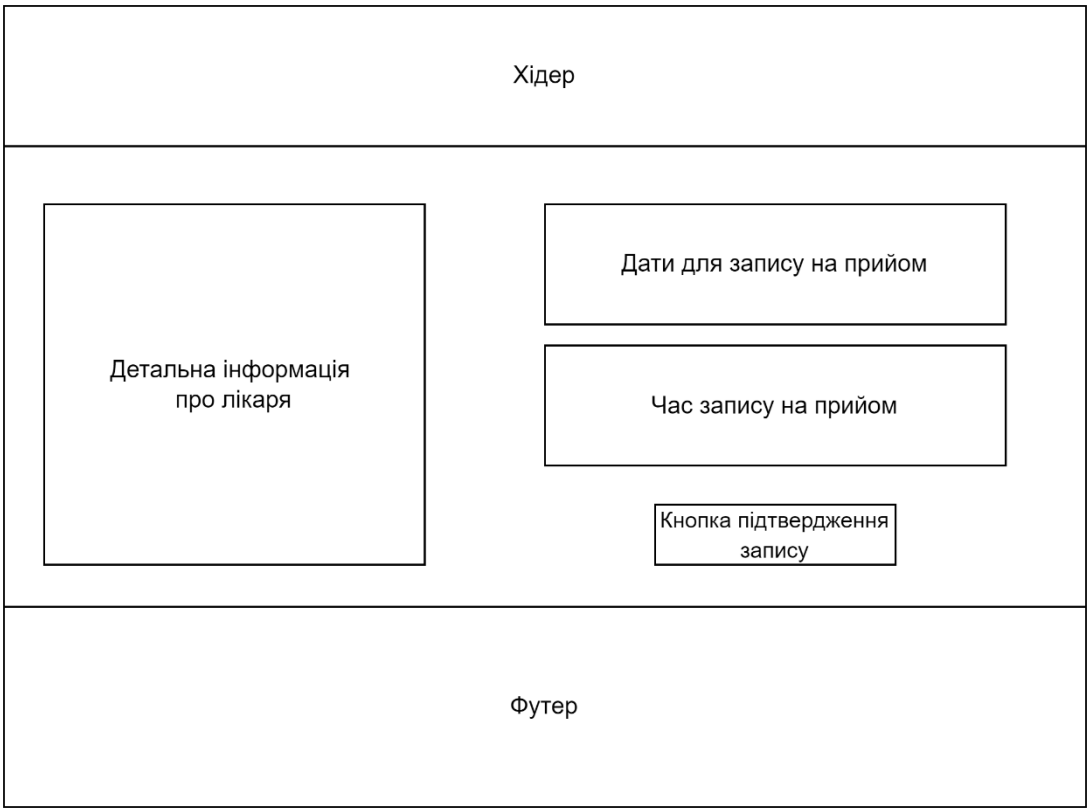


Рисунок 3.8 Макет сторінки Фахівця

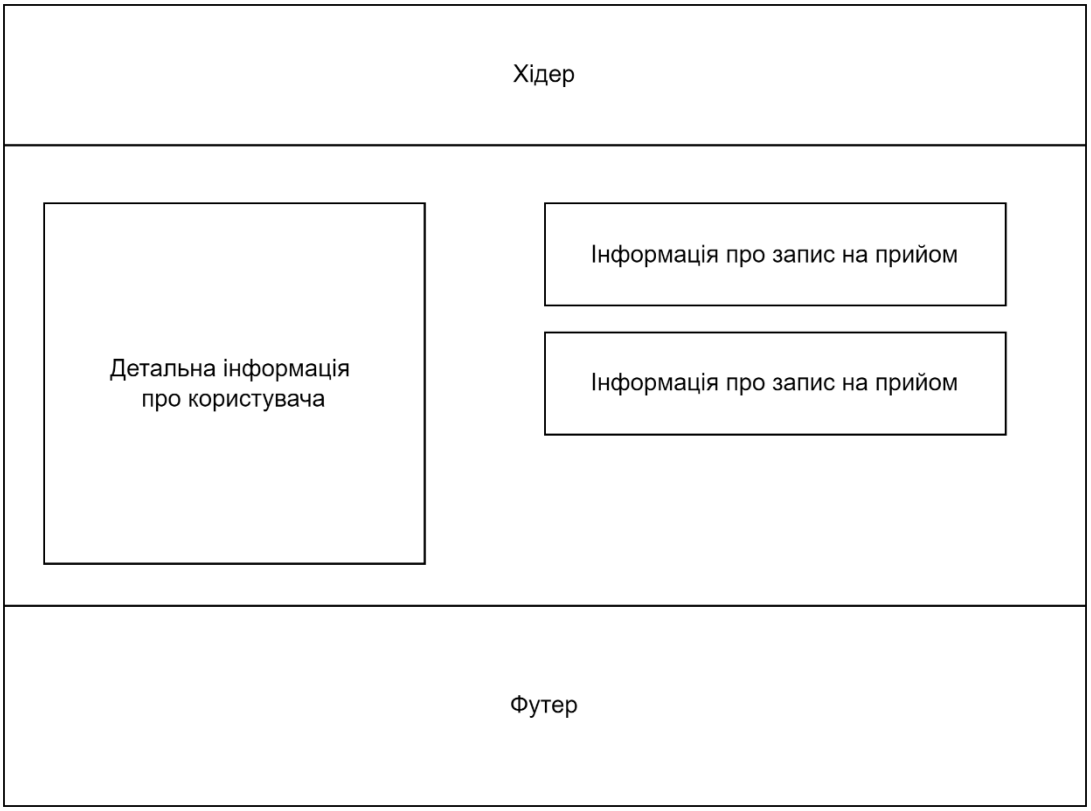


Рисунок 3.9 – Макет сторінки Особистий кабінет

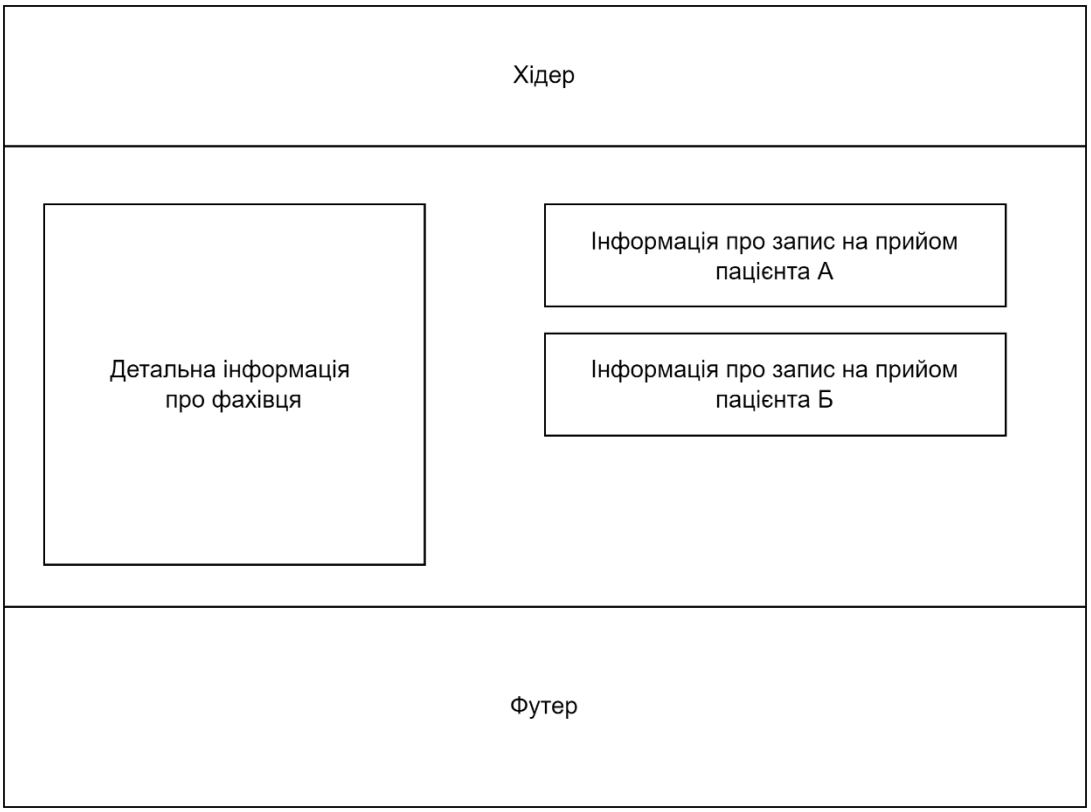


Рисунок 3.10 – Макет сторінки Кабінет лікаря

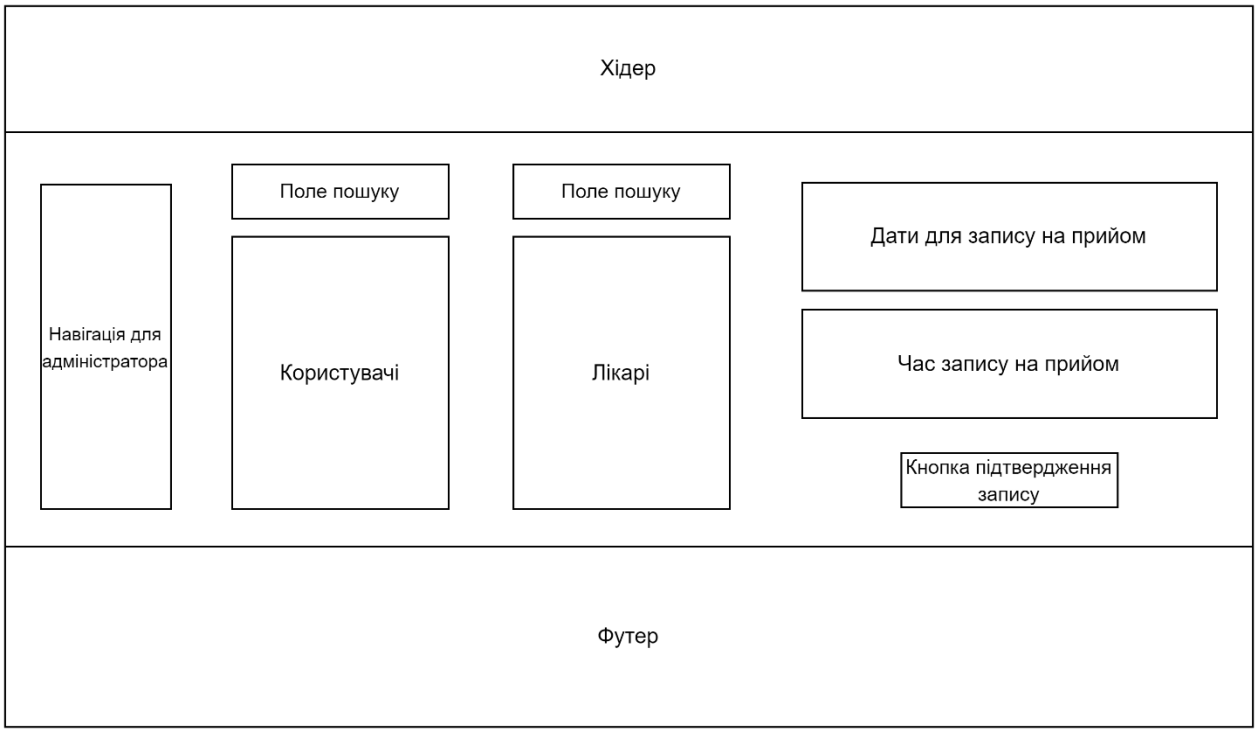


Рисунок 3.11 – Макет Адміністративної панеі

3.2.2 Налаштування додатку

Перед створенням проекту необхідно завантажити і інсталювати Node.js. Середовище розробки та структура проекту були автоматично налаштовані за допомогою інструменту Create-React-App командою

«npx create-react-app client-side».

Додатково необхідно завантажити бібліотеку «react-router-dom» для налаштування роутингу проекту за допомогою команди:

«npm install react-router-dom»

3.2.3 Структура додатку

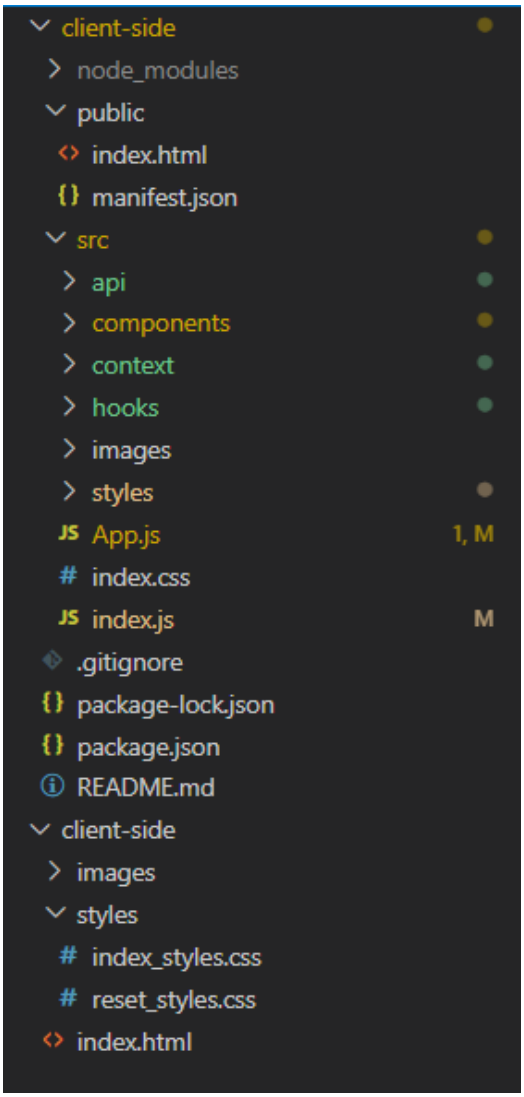


Рисунок 3.12 – Структура фронтенд проекту

Структура проекту ділиться на декілька папок.

Папка «public» містить статичні файли, такі як index.html, який є вихідним HTML файлом для додатку, маніфест.

Основна папка з джерельним кодом додатку – «src». Вона містить наступні підкомпоненти:

- «components»: для React компонентів;
- «api»: для всіх функції та файли, пов’язаних з взаємодією з зовнішніми API;

- «context»: для файлів, пов’язаних з React Context API, який використовується для управління глобальним станом додатку;
- «hooks»: для користувацьких хуків (custom hooks);
- «styles»: папка з css стилями;
- «images»: папка із зображеннями.

Роутинг сайту налаштовано у файлі App.js за допомогою компонентів Routes, Route, Link бібліотеки «react-router-dom» .

```
26  function App() {
27    return (
28      <>
29      <Routes>
30        <Route path='/' element={<Layout/>}>
31          <Route index element={<HomePage/>}/>
32          <Route path='doctors' element={<SpecialistsPage/>}/>
33          <Route path='doctors/:doctorId' element={<ActiveSpecialistPage/>}/>
34          <Route path='services' element={<ServicesPage/>}/>
35
36          <Route element={<RequiredAuths allowedRoles={["BASIC_USER"]}/>}>
37            <Route path='account' element={<PersonalPage/>}/>
38          </Route>
39
40          <Route element={<RequiredAuths allowedRoles={["DOCTOR"]}/>}>
41            <Route path='doctor_account' element={<DoctorPersonalPage/>}/>
42          </Route>
43
44          <Route element={<RequiredAuths allowedRoles={["ADMIN"]}/>}>
45            <Route path='admin/' element={<AdminPage/>}>
46              <Route path='create-appointment' element={<AdminCreateAppointments/>}/>
47              <Route path='appointments' element={<AdminAppointments/>}/>
48              <Route path='users' element={<AdminUsers/>}/>
49              <Route path='doctors' element={<AdminDoctors/>}/>
50              <Route path='doctors/:doctorId' element={<AdminDoctorDetails/>}/>
51            </Route>
52          </Route>
53        </Route>
54
55        <Route path='/account/signup' element={<SignUp/>}/>
56        <Route path='/account/login' element={<Login/>}/>
57      </Routes>
58      </>
```

Рисунок 3.13 – Налаштування роутингу в App.js

Взаємодію з серверною частиною проекту реалізовано за допомогою HTTP клієнта axios, який надсилає HTTP запити.

```
const DOCTORS_URL = 'doctors';
function AdminDoctors() {

    const [doctors, setDoctors] = useState([]);

    useEffect(() => {
        const getDoctors = async () => {
            try{
                const response = await axios.get(DOCTORS_URL);
                setDoctors(response.data);
            }
            catch(err){
                console.log(err);
            }
        }

        getDoctors();
    }, []);
```

Рисунок 3.14 – Приклад роботи axios.get

Приклади компонентів та стилів наведено в Додатку В.

Повний код фронтенд додатку наведено в проекті client-side на GitHub[18].

3.3 Тестування

1. Реєстрація.

Заповнюємо форму реєстрації та натискаємо Підтвердити.

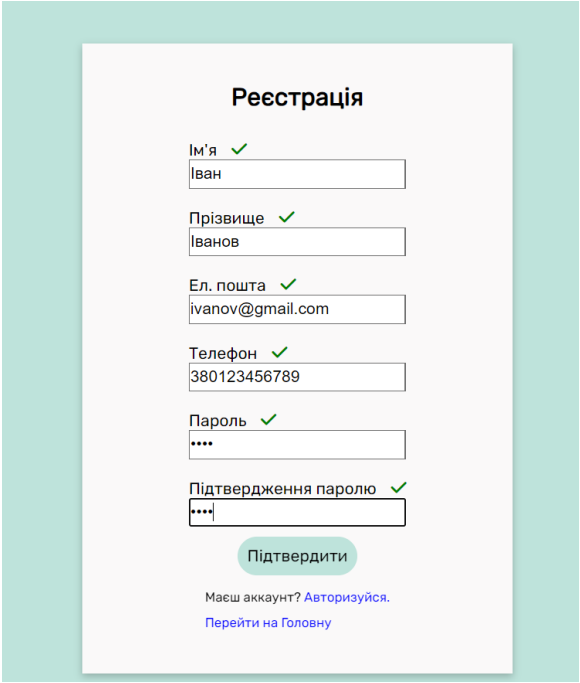


Рисунок 3.15 – Реєстраційна форма

Перевіряємо наявність користувача в базі даних.

```
1  select * from account where last_name = 'Іванов';
```

Data Output | Messages | Notifications

	uuid [PK] uuid	first_name character varying (40)	last_name character varying (40)	email character varying (200)	phone character varying (12)
1	0f9761dc-d80a-4280-9b4f-4fe3cc893786	Іван	Іванов	ivanov@gmail.com	380123456789

Рисунок 3.16 – Наявність нового користувача в БД

Новий користувач є в базі даних. Отже, реєстрація пройшла успішно.

2. Авторизація

Вводимо email і пароль користувача в форму авторизації і натискаємо підтвердити.

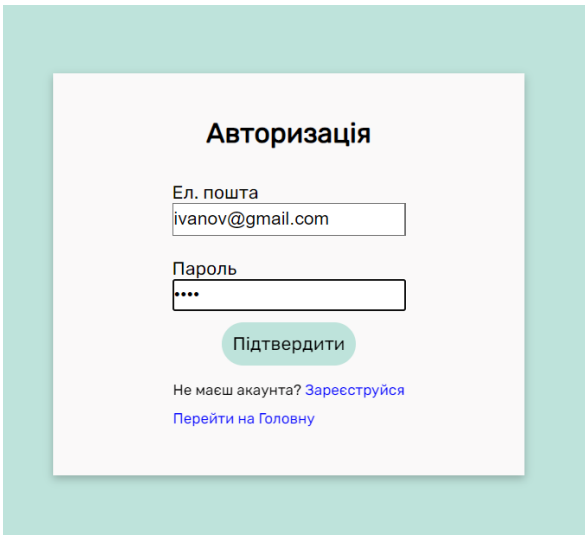


Рисунок 3.17 – Форма авторизації

Завантажується Головна сторінка. Можемо перейти в Особистий кабінет.

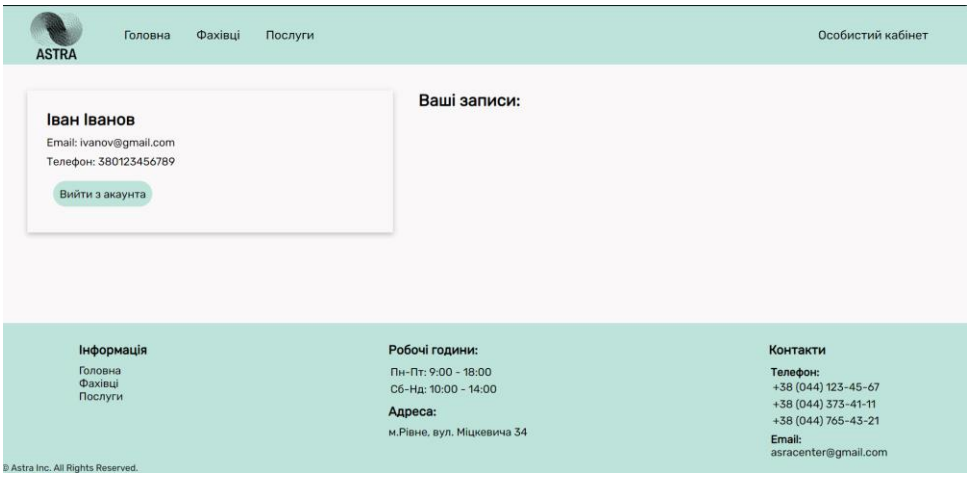


Рисунок 3.18 - Особистий кабінет нового користувача

В Особистому кабінеті відображаються відомості про авторизованого користувача. Отже, авторизація пройшла успішно.

3. Інформаційна платформа

Перейдемо на сторінку Фахівці.

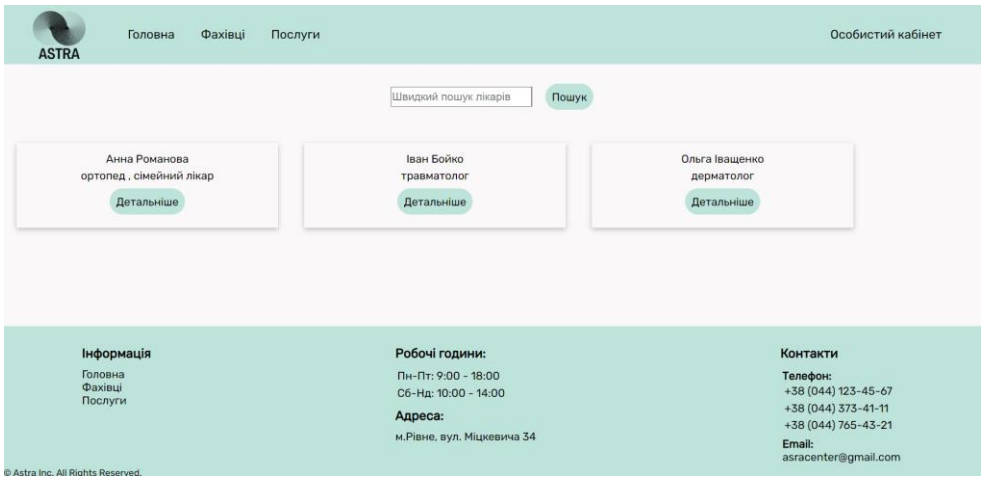


Рисунок 3.19 - Сторінка Фахівці

Фахівці відображаються на сторінці.

Перевіримо пошук за прізвищем «Бойко».

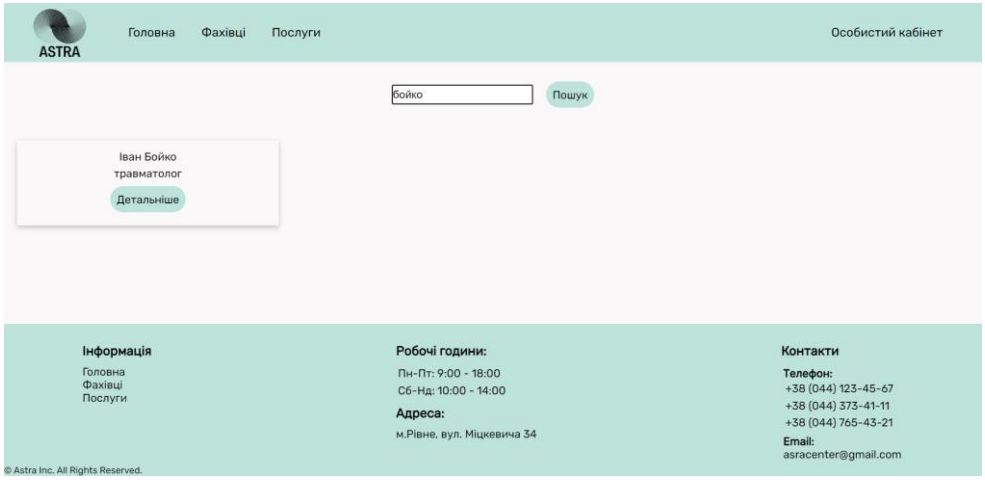


Рисунок 3.20 - Пошук за прізвищем

Пошук працює правильно.

4. Система управління записами на прийом, Особистий кабінет.

Перейдемо на сторінку лікаря, оберемо дату і час, підтвердимо запис.

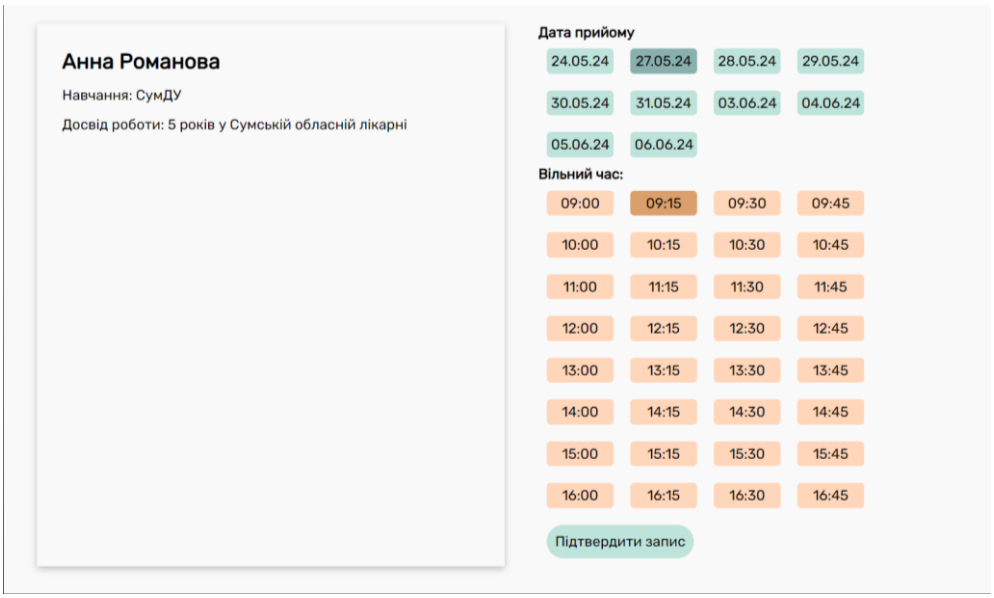


Рисунок 3.21 - Сторінка лікаря із формою запису на прийом

В особистому кабінеті має з’явитися запис до лікаря.

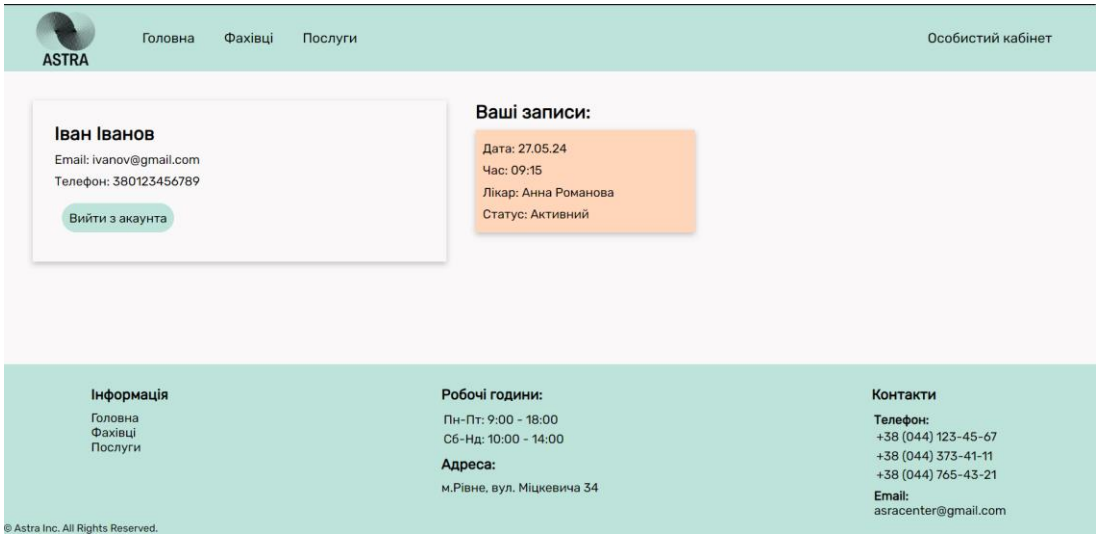


Рисунок 3.22 - Особистий кабінет із записами

5. Кабінет лікаря.

Авторизуємося як акаунт лікаря. Перейдемо в Кабінет лікаря.

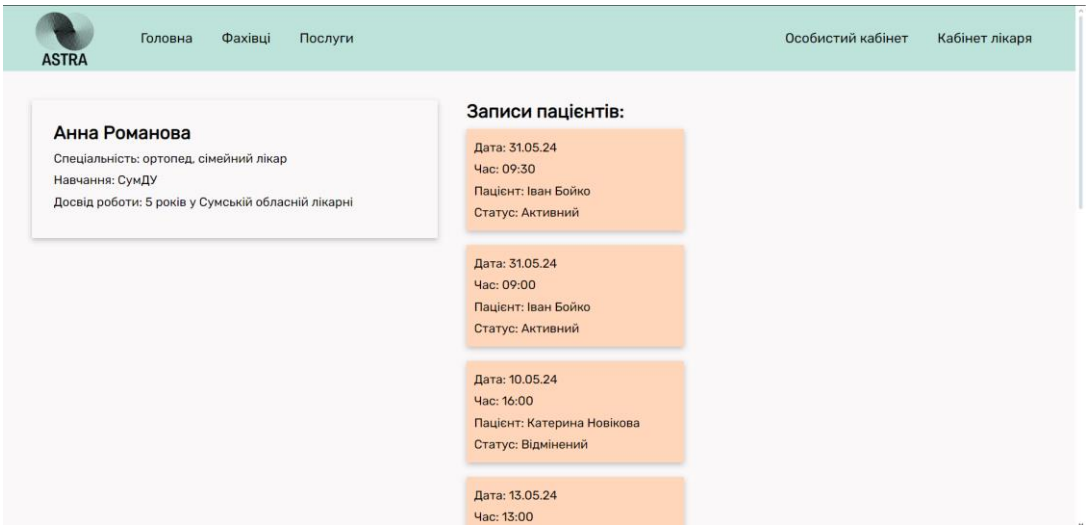
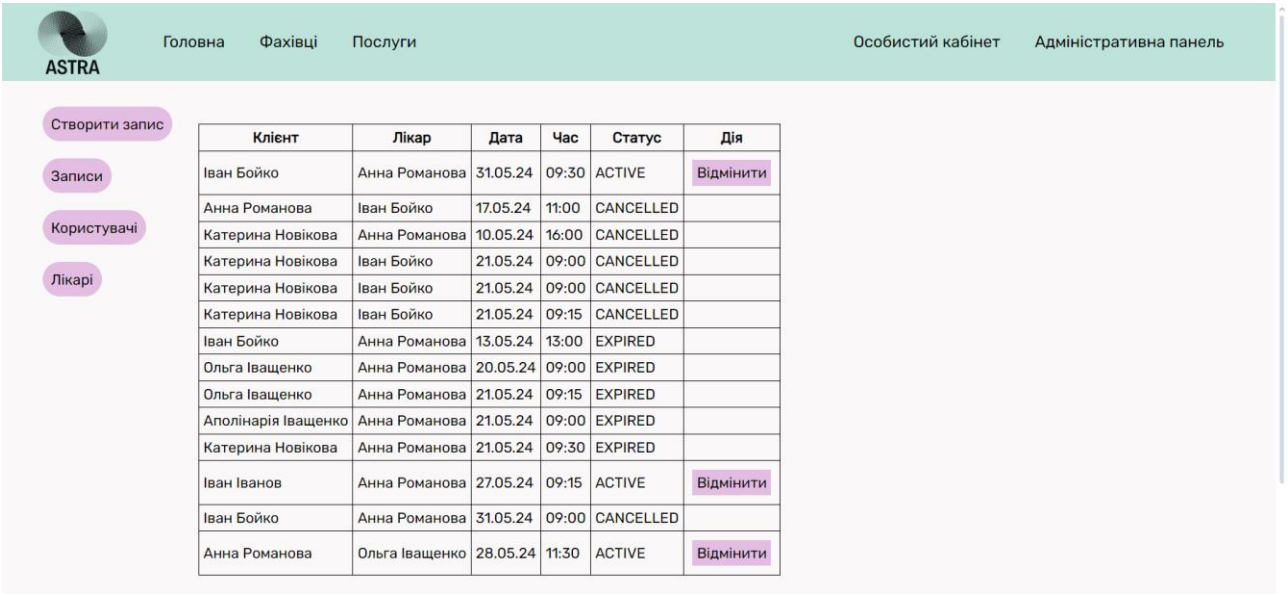


Рисунок 3.23 - Кабінет лікаря із записами на прийом

В кабінеті відображена інформація про лікаря та записи пацієнтів до нього.

6. Адміністративна панель.

Авторизуємось як акаунт адміністратора. Запис на прийом працює аналогічно до запису на прийом звичайним користувачем. Перевіримо можливість відміни прийому. Перейдемо на вкладу Записи.



Клієнт	Лікар	Дата	Час	Статус	Дія
Іван Бойко	Анна Романова	31.05.24	09:30	ACTIVE	Відмінити
Анна Романова	Іван Бойко	17.05.24	11:00	CANCELLED	
Катерина Новікова	Анна Романова	10.05.24	16:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:15	CANCELLED	
Іван Бойко	Анна Романова	13.05.24	13:00	EXPIRED	
Ольга Іващенко	Анна Романова	20.05.24	09:00	EXPIRED	
Ольга Іващенко	Анна Романова	21.05.24	09:15	EXPIRED	
Аполінарія Іващенко	Анна Романова	21.05.24	09:00	EXPIRED	
Катерина Новікова	Анна Романова	21.05.24	09:30	EXPIRED	
Іван Іванов	Анна Романова	27.05.24	09:15	ACTIVE	Відмінити
Іван Бойко	Анна Романова	31.05.24	09:00	CANCELLED	
Анна Романова	Ольга Іващенко	28.05.24	11:30	ACTIVE	Відмінити

Рисунок 3.24 - Адміністративна панель / Записи

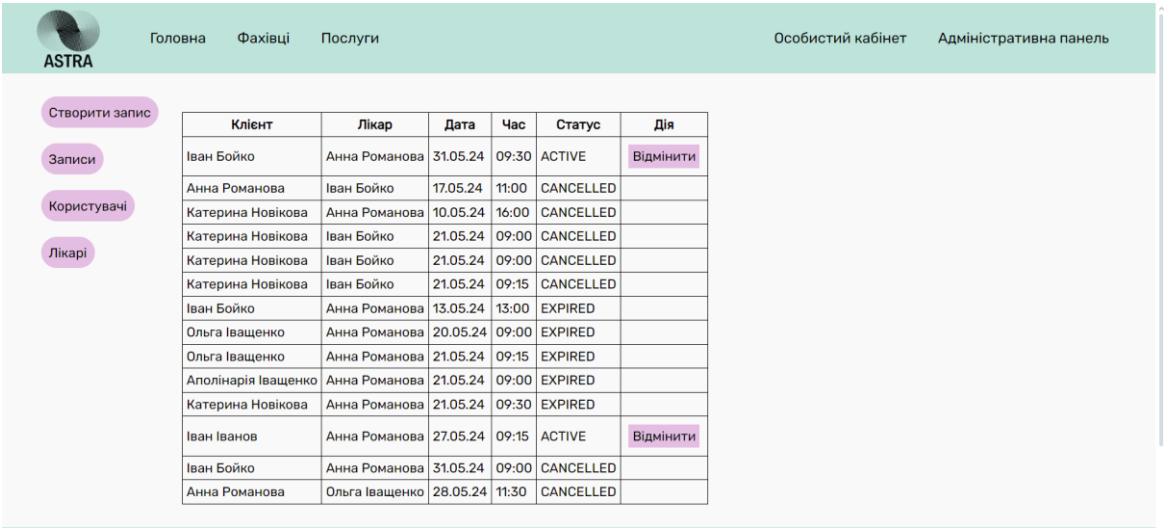
Натиснемо на кнопку Відмінити біля обраного запису:

Клієнт: Анна Романова

Лікар: Ольга Іващенко

Дата: 28.05.24

Час: 11:30

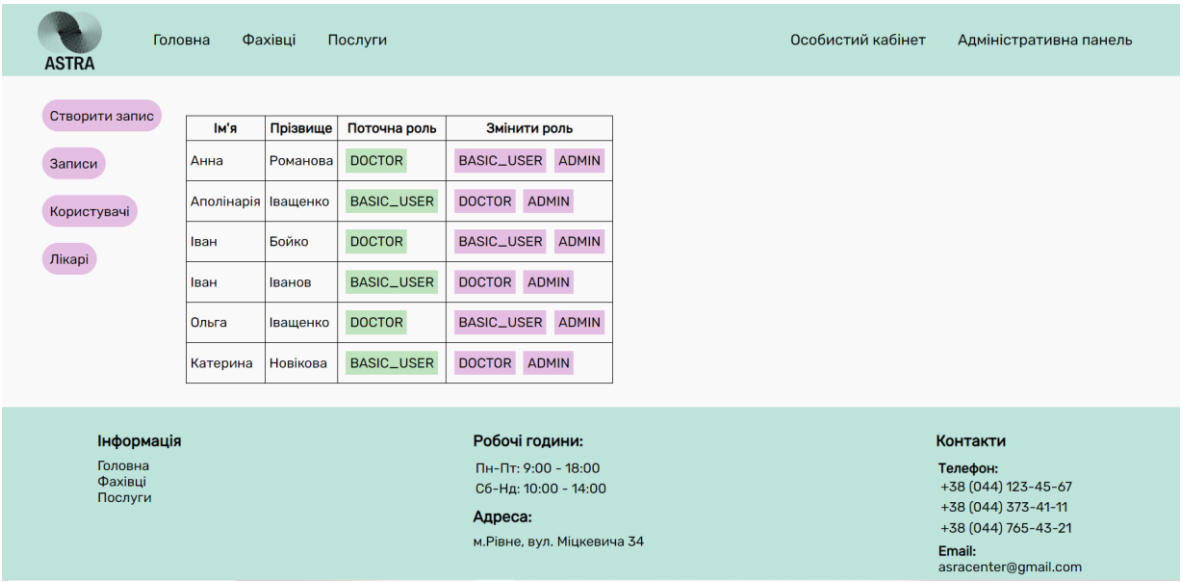


Клієнт	Лікар	Дата	Час	Статус	Дія
Іван Бойко	Анна Романова	31.05.24	09:30	ACTIVE	Відмінити
Анна Романова	Іван Бойко	17.05.24	11:00	CANCELLED	
Катерина Новікова	Анна Романова	10.05.24	16:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:00	CANCELLED	
Катерина Новікова	Іван Бойко	21.05.24	09:15	CANCELLED	
Іван Бойко	Анна Романова	13.05.24	13:00	EXPIRED	
Ольга Іващенко	Анна Романова	20.05.24	09:00	EXPIRED	
Ольга Іващенко	Анна Романова	21.05.24	09:15	EXPIRED	
Аполінарія Іващенко	Анна Романова	21.05.24	09:00	EXPIRED	
Катерина Новікова	Анна Романова	21.05.24	09:30	EXPIRED	
Іван Іванов	Анна Романова	27.05.24	09:15	ACTIVE	Відмінити
Іван Бойко	Анна Романова	31.05.24	09:00	CANCELLED	
Анна Романова	Ольга Іващенко	28.05.24	11:30	CANCELLED	

Рисунок 3.25 – Результат відміни прийому

Статус прийому змінився на Cancelled, отже прийом було відмінено.

Перевіримо функцію зміни ролі. Для цього перейдемо на вкладку Користувачі.



Ім'я	Прізвище	Поточна роль	Змінити роль
Анна	Романова	DOCTOR	BASIC_USER ADMIN
Аполінарія	Іващенко	BASIC_USER	DOCTOR ADMIN
Іван	Бойко	DOCTOR	BASIC_USER ADMIN
Іван	Іванов	BASIC_USER	DOCTOR ADMIN
Ольга	Іващенко	DOCTOR	BASIC_USER ADMIN
Катерина	Новікова	BASIC_USER	DOCTOR ADMIN

Рисунок 3.26 - Адміністративна панель/Користувачі

Змінимо роль користувача Новікова Катерина з BASIC_USER на DOCTOR.

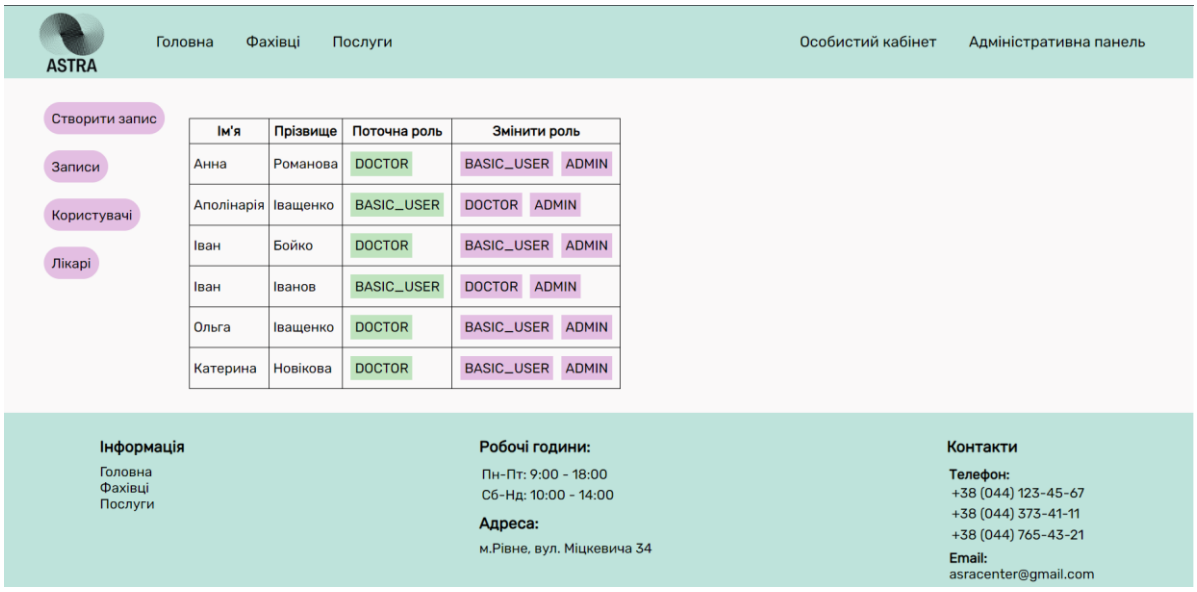


Рисунок 3.27 – Результат зміни ролі

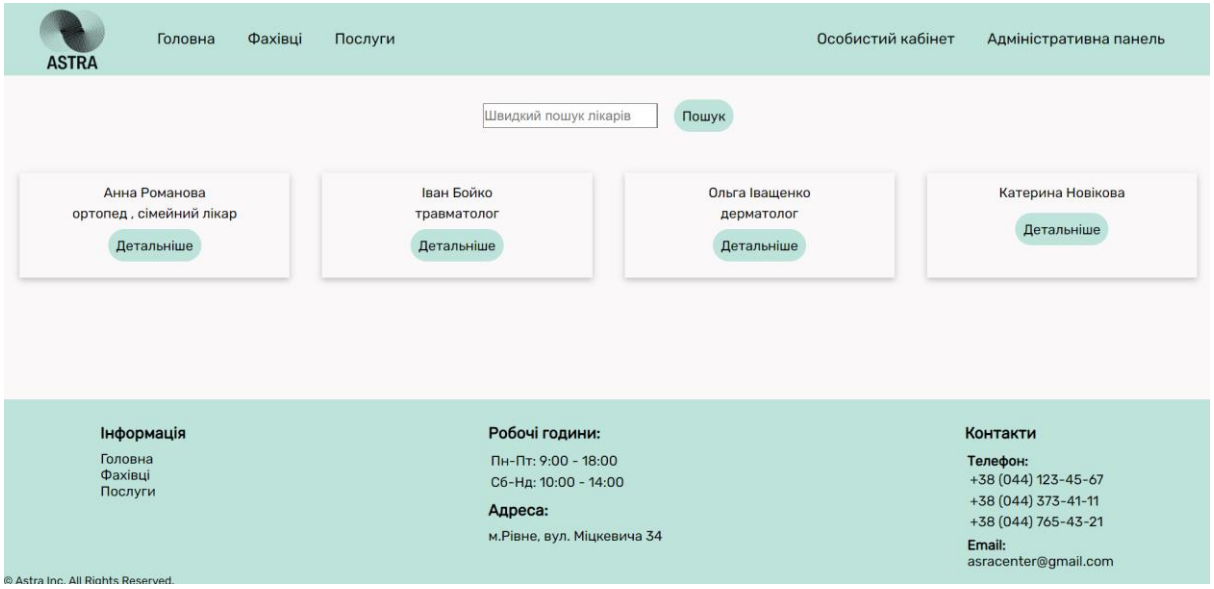


Рисунок 3.28 – Список лікарів

Роль змінилась і в списку лікарів з’явився користувач зі зміненою роллю.

Отже, всі основні функції проекту, зазначені у вимогах, працюють правильно.

ВИСНОВОК

Протягом дипломної роботи було виконано всі поставлені завдання, а саме: було спроектовано і створено БД, розроблено інтерфейс веб-сервісу для медичного закладу з повною інформацією про заклад, фахівців, послуги. реалізовано функціональність реєстрації та авторизації користувача, пошуку лікарів, створення записів до лікарів та їх перегляд, розроблено адміністративну панель для управління записами клінтів, ролями користувачів, інформацією про лікарів.

На допрацювання виноситься покращення системи безпеки, UX-дизайну сайту, додавання системи ведення медичних карток.

СПИСОК ЛІТЕРАТУРИ

1. A complete guide to flexbox | css-tricks. CSS-Tricks. URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (дата звернення: 02.12.2023).
2. A complete guide to routing in react. *The Next Generation GraphQL Headless CMS | Hygraph*. URL: https://hygraph.com/blog/routing-in-react#how-to-implement-dynamic-routing-with-react-router (date of access: 25.04.2024).
3. Anton Bacaj. React JS filter, search and sort items using react-router v6, 2022. YouTube. URL: https://www.youtube.com/watch?v=c3WSziz_u_o (date of access: 10.05.2024).
4. Dave Gray. React JS form validation | axios user registration form submit | beginners to intermediate, 2021. *YouTube*. URL: https://www.youtube.com/watch?v=brcHK3P6ChQ (date of access: 13.05.2024).
5. Features - IntelliJ IDEA. JetBrains. URL: https://www.jetbrains.com/idea/features/ (дата звернення: 01.12.2023).
6. Funda Of Web IT. React JS CRUD - How to fetch data from api in react js & display data in html table | React Tutorial, 2023. *YouTube*. URL: https://www.youtube.com/watch?v=T-r6xPM-qqE (date of access: 24.04.2024).
7. Getting Started | Building REST services with Spring. *Getting Started | Building REST services with Spring*. URL: https://spring.io/guides/tutorials/rest (date of access: 27.04.2024).
8. Login and registration REST API using spring boot, spring security, hibernate and mysql database. *Java Guides*. URL: https://www.javaguides.net/2021/10/login-and-registration-rest-api-using-spring-boot-spring-security-hibernate-mysql-database.html#google_vignette (date of access: 03.05.2024).
9. Microsoft. Visual studio code - code editing. redefined. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com (date of access: 02.05.2024).

10. Noodles P. H. Частина 7. Ознайомлення з патерном MVC (Model-View-Controller). JavaRush. URL: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller (дата звернення: 01.12.2023).

11. PostgreSQL: about. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (date of access: 20.04.2024).

12. Quick start – react. *React*. URL: https://react.dev/learn (date of access: 20.04.2024).

13. Spring MVC – основні принципи для початківців, приклади. Highload.today - медіа для розробників. URL: https://highload.today/spring-mvc/ (дата звернення: 02.12.2023).

14. Use case diagrams | unified modeling language (UML) - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/use-case-diagram/ (date of access: 14.04.2024).

15. What is a REST API?. *Red Hat - We make open source technologies for the enterprise*. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (date of access: 01.05.2024).

16. What is an entity relationship diagram (ERD)?. *Lucidchart*. URL: https://www.lucidchart.com/pages/er-diagrams (date of access: 13.04.2024).

17. Вибір правильного модуля Spring для ефективної розробки програм. Xentrict. URL: https://xentrict.com/вибір-правильного-модуля-spring-для-ефекти-3/ (дата звернення: 02.12.2023).

18. Новікова К. Д. GitHub - novikovaKat/client-side. *GitHub*. URL: https://github.com/novikovaKat/client-side.git.

19. Новікова К. Д. GitHub - novikovaKat/demo. *GitHub*. URL: https://github.com/novikovaKat/demo.git.

20. Чому і коли варто обирати розробку на React JS - IT рейтинг UA. IT рейтинг України. URL: https://it-rating.ua/news-3658 (дата звернення: 02.12.2023).

ДОДАТОК А.СКРИПТ СТВОРЕННЯ ТАБЛИЦЬ БД

```
CREATE TABLE "account"
(
  "uuid" UUID NOT NULL,
  "first_name" Varchar(40) NOT NULL,
  "last_name" Varchar(40) NOT NULL,
  "email" Varchar(200) NOT NULL,
  "phone" Varchar(12) NOT NULL,
  "password" Varchar(255) NOT NULL,
  "role" NUMERIC DEFAULT 0 NOT NULL
);
ALTER TABLE "account" ADD CONSTRAINT "PK_account" PRIMARY KEY ("uuid");

CREATE TABLE "doctor"
(
  "uuid" UUID NOT NULL,
  "account_id" UUID NOT NULL,
  "education" Varchar(500),
  "experience" Varchar(500)
);

ALTER TABLE "doctor" ADD CONSTRAINT "PK_doctor" PRIMARY KEY ("uuid");

CREATE TABLE "specialty"
(
    "uuid" UUID NOT NULL,
    "name" Varchar(100) NOT NULL
);

ALTER TABLE "specialty" ADD CONSTRAINT "PK_specialty" PRIMARY KEY ("uuid");

CREATE TABLE "appointment"
(
  "uuid" UUID NOT NULL,
  "patient_id" UUID NOT NULL,
  "doctor_id" UUID NOT NULL,
  "appointment_date" Timestamptz NOT NULL,
  "status" NUMERIC DEFAULT 0 NOT NULL
);
ALTER TABLE "appointment" ADD CONSTRAINT "PK_appointment" PRIMARY KEY ("uuid");

ALTER TABLE "doctor"
  ADD CONSTRAINT "Relationship1"
    FOREIGN KEY ("account_id")
    REFERENCES "account" ("uuid");

ALTER TABLE "appointment"
```

```
  ADD CONSTRAINT "Relationship2"
    FOREIGN KEY ("patient_id")
    REFERENCES "account" ("uuid");

ALTER TABLE "appointment"
  ADD CONSTRAINT "Relationship3"
    FOREIGN KEY ("doctor_id")
    REFERENCES "doctor" ("uuid");

CREATE TABLE "offer"
(
    "uuid" UUID NOT NULL,
    "name" Varchar(200) NOT NULL,
    "price" NUMERIC(32) NOT NULL
);
ALTER TABLE "offer" ADD CONSTRAINT "PK_offer" PRIMARY KEY ("uuid");
```

ДОДАТОК Б. ПРИКЛАДИ КЛАСІВ BACKEND ДОДАТКУ

Account

```
package com.example.demo.models;

import com.example.demo.models.enums.Role;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

import java.util.Objects;
import java.util.UUID;

@Entity
@Table(name = "account")
public class Account{
    @Id
    @Column(name = "uuid")
    private UUID uuid;
    @Column (name = "first_name")
    private String firstName;
    @Column (name = "last_name")
    private String lastName;
    @Column (name = "email")
    private String email;
    @Column (name = "phone")
    private String phone;

    @Column (name = "password")
    private String password;

    @Column
    private Role role;

    public UUID getUuid() {
        return uuid;
    }

    public void setUuid(UUID uuid) {
        this.uuid = uuid;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
```

```
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Role getRole() {
        return role;
    }

    public void setRole(Role role) {
        this.role = role;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Account account)) return false;
        return uuid.equals(account.uuid) && firstName.equals(account.firstName) &&
lastName.equals(account.lastName)      &&      email.equals(account.email)      &&
```

```
Objects.equals(phone, account.phone) && password.equals(account.password) && role
== account.role;
    }

    @Override
    public int hashCode() {
        return Objects.hash(uuid, firstName, lastName, email, phone, password,
role);
    }
}
```

AccountController

```
package com.example.demo.controllers;

import com.example.demo.models.request.CreateAccountRequest;
import com.example.demo.models.request.LoginRequest;
import com.example.demo.models.request.UpdateAccountRequest;
import com.example.demo.models.response.AccountResponse;
import com.example.demo.models.response.AppointmentResponse;
import com.example.demo.models.response.DoctorResponse;
import com.example.demo.services.AccountService;
import com.example.demo.services.AppointmentService;
import com.example.demo.services.DoctorService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController
@RequestMapping("/account")
@CrossOrigin("http://localhost:3000/")
public class AccountController {
    @Autowired
    private AccountService accountService;

    @Autowired
    private AppointmentService appointmentService;

    @Autowired
    private DoctorService doctorService;

    @PostMapping("/signup")
    public AccountResponse add(@RequestBody CreateAccountRequest
createAccountRequest) {
        return accountService.saveAccount(createAccountRequest);
    }

    @PostMapping("/login")
```

```
    public AccountResponse authenticateUser(@RequestBody LoginRequest loginRequest)
{
        return accountService.login(loginRequest);
    }

    @GetMapping("/data")
    public AccountResponse getAccountData(@RequestParam UUID uuid){
        return accountService.getAccountData(uuid);
    }
    @GetMapping("/appointments")
    public   List<AppointmentResponse>   getPatientAppointments(@RequestParam   UUID
uuid){
        return appointmentService.getAppointmentsByPatientId(uuid);
    }

    @GetMapping("/doctor/data")
    public DoctorResponse getDoctorData(@RequestParam UUID uuid){
        return doctorService.getDoctorByAccountId(uuid);
    }

    @GetMapping("/doctor/appointments")
    public   List<AppointmentResponse>   getDoctorAppointments(@RequestParam   UUID
uuid){
        return appointmentService.getAppointmentsByDoctorId(uuid);
    }

    @GetMapping("/all")
    public List<AccountResponse> getAllAccounts(){
        return accountService.getAllAccounts();
    }

    @PutMapping("/{uuid}")
    public    AccountResponse    updateAccount(@PathVariable    final    UUID    uuid,
@RequestBody UpdateAccountRequest updateAccountRequest){
        return accountService.updateAccount(uuid, updateAccountRequest);
    }
}
```

CreateAccountRequest

```
package com.example.demo.models.request;

public record CreateAccountRequest(
        String firstName,
        String lastName,
        String email,
        String phone,
        String password
) {
}
```

AccountService

```
package com.example.demo.services;

import com.example.demo.models.request.CreateAccountRequest;
import com.example.demo.models.request.LoginRequest;
import com.example.demo.models.request.UpdateAccountRequest;
import com.example.demo.models.response.AccountResponse;

import java.util.List;
import java.util.UUID;

public interface AccountService {
    public AccountResponse saveAccount(CreateAccountRequest createAccountRequest);

    AccountResponse login(LoginRequest loginRequest);

    AccountResponse getAccountData(UUID uuid);

    List<AccountResponse> getAllAccounts();

    AccountResponse updateAccount(UUID uuid, UpdateAccountRequest
updateAccountRequest);
}
```

AccountServiceImpl

```
package com.example.demo.services;

import com.example.demo.models.Account;
import com.example.demo.models.Doctor;
import com.example.demo.models.enums.Role;
import com.example.demo.models.request.CreateAccountRequest;
import com.example.demo.models.request.LoginRequest;
import com.example.demo.models.request.UpdateAccountRequest;
import com.example.demo.models.response.AccountResponse;
import com.example.demo.models.response.DoctorViewResponse;
import com.example.demo.repository.AccountRepository;
import com.example.demo.repository.DoctorRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```
import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
public class AccountServiceImpl implements AccountService{

    @Autowired
    private AccountRepository accountRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private DoctorService doctorService;
    @Override
    @Transactional
    public AccountResponse saveAccount(CreateAccountRequest request) {

        // add check for email exists in a DB
        if(accountRepository.existsByEmail(request.email())){
            throw new RuntimeException("Email already taken");
        }

        Account account = new Account();
        account.setUuid(UUID.randomUUID());
        account.setFirstName(request.firstName());
        account.setLastName(request.lastName());
        account.setEmail(request.email());
        account.setPhone(request.phone());
        account.setPassword(passwordEncoder.encode(request.password()));
        account.setRole(Role.BASIC_USER);

        Account savedAccount = accountRepository.save(account);
        return new AccountResponse(
                savedAccount.getUuid(),
                savedAccount.getFirstName(),
                savedAccount.getLastName(),
                savedAccount.getEmail(),
                savedAccount.getPhone(),
                savedAccount.getRole());
    }
```

```
    @Override
    public AccountResponse login(LoginRequest loginRequest) {
        Optional<Account>                          account                          =
accountRepository.findByEmail(loginRequest.email());
        if(account.isEmpty()){
            throw new RuntimeException("Account doesn`t exist");
        }
        UsernamePasswordAuthenticationToken          token          =          new
UsernamePasswordAuthenticationToken(
                loginRequest.email(), loginRequest.password());
        Authentication authentication = authenticationManager.authenticate(token);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        return new AccountResponse(
                account.get().getUuid(),
                account.get().getFirstName(),
                account.get().getLastName(),
                account.get().getEmail(),
                account.get().getPhone(),
                account.get().getRole());
    }

    @Override
    public AccountResponse getAccountData(UUID uuid) {
        Optional<Account> account = accountRepository.findByUuid(uuid);
        if(account.isEmpty()){
            throw new RuntimeException("Account doesn`t exist");
        }

        return new AccountResponse(
                account.get().getUuid(),
                account.get().getFirstName(),
                account.get().getLastName(),
                account.get().getEmail(),
                account.get().getPhone(),
                account.get().getRole()
        );

    }

    @Override
    public List<AccountResponse> getAllAccounts() {
        return this.accountRepository.findAll().stream()
                .map(account -> new AccountResponse(
                        account.getUuid(),
                        account.getFirstName(),
                        account.getLastName(),
                        account.getEmail(),
                        account.getPhone(),
                        account.getRole()))
```

```
                .collect(Collectors.toList());
    }

    @Override
    public    AccountResponse    updateAccount(UUID    uuid,    UpdateAccountRequest
updateAccountRequest){
        Account account = this.accountRepository.findByUuid(uuid)
                .orElseThrow(() -> new RuntimeException("No such account"));

Optional.ofNullable(updateAccountRequest.firstName()).ifPresent(account::setFirstNa
me);

Optional.ofNullable(updateAccountRequest.lastName()).ifPresent(account::setLastName
);

Optional.ofNullable(updateAccountRequest.email()).ifPresent(account::setEmail);

Optional.ofNullable(updateAccountRequest.phone()).ifPresent(account::setPhone);
        Optional.ofNullable(updateAccountRequest.role()).ifPresent(f             ->
this.changeRole(Role.valueOf(f), account));

        Account updatedAccount = this.accountRepository.save(account);



        return new AccountResponse(
                updatedAccount.getUuid(),
                updatedAccount.getFirstName(),
                updatedAccount.getLastName(),
                updatedAccount.getEmail(),
                updatedAccount.getPhone(),
                updatedAccount.getRole());
    }

    private void changeRole(Role role, Account account){
        switch (role){
            case BASIC_USER, ADMIN -> {
                if(account.getRole() == Role.DOCTOR) {
                    this.doctorService.disableDoctor(account.getUuid());
                }
            }
            case DOCTOR -> {
                Optional<Doctor>                        doctor                        =
this.doctorRepository.findByAccountUuid(account.getUuid());
                if(doctor.isPresent()){
                    doctor.get().setStatus(true);
                    this.doctorRepository.save(doctor.get());
                }
                else {
                    this.doctorService.createDoctor(account.getUuid());
```

```
                }
            }
        }
        account.setRole(role);
    }
}
```

AccountRepository

```
package com.example.demo.repository;

import com.example.demo.models.Account;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
import java.util.UUID;
@Repository
public interface AccountRepository extends JpaRepository<Account, UUID> {
    Optional<Account> findByEmail(String email);
    Optional<Account> findByUuid(UUID uuid);
    Boolean existsByEmail(String email);

}
```

AccountResponse

```
package com.example.demo.models.response;

import com.example.demo.models.enums.Role;

import java.util.UUID;

public record AccountResponse(
        UUID uuid,
        String firstName,
        String lastName,
        String email,
        String phone,
        Role role
) {
}
```

ДОДАТОК В. ПРИКЛАДИ КОМПОНЕНТІВ REACT ТА CSS СТИЛІВ

AdminCreateAppointments.jsx

```
import { useEffect, useState } from "react"
import axios from "../../../api/axios";
import AppointmentsByDoctor from "../appointments/AppointmentsByDoctor";


const USERS_URL = 'account/all';
const DOCTORS_URL = 'doctors';
const CREATE_APPOINTMENT_URL = '/appointment/create';

function AdminCreateAppointments(){
    const [searchUserInput, setSearchUserInput] = useState('');
    const [searchUser, setSearchUser] = useState('');
    const [searcDoctorInput, setSearchDoctorInput] = useState('');
    const [searchDoctor, setSearchDoctor] = useState('');

    const [users, setUsers] = useState([]);
    const [doctors, setDoctors] = useState([]);

    const [userId, setUserId] = useState();
    const [doctorId, setDoctorId] = useState();

    const [activeAppDate, setActiveAppDate] = useState(null);

    const [success, setSuccess] = useState(false);

    useEffect(() => {

        const getData = async () => {
            try{
                const resUsers = await axios.get(USERS_URL);
                setUsers(resUsers.data);
            }
            catch(err){
                console.log(err);
            }

            try{
                const resDoctors = await axios.get(DOCTORS_URL);
                setDoctors(resDoctors.data);
            }
            catch(err){
                console.log(err);
            }
        }
        getData();
```

```
}, []);

const handleClickOnUser = (e) =>{
    e.preventDefault();
    const row = e.target.parentElement;
    const userButtons = document.querySelectorAll('.user-row');

    userButtons.forEach(btn => {
        btn.classList.remove('active-row');
    })
    row.classList.add('active-row');

    setUserId(row.getAttribute('data'));
}

const handleClickOnDoctor = (e) =>{
    e.preventDefault();
    const row = e.target.parentElement;
    const doctorButtons = document.querySelectorAll('.doctor-row');

    doctorButtons.forEach(btn => {
        btn.classList.remove('active-row');
    })
    row.classList.add('active-row');

    setDoctorId(row.getAttribute('data'));
}


const createAppointment = async () =>{
    if(activeAppDate){
        setSuccess(false);
        try{
            console.log(activeAppDate);
            const response = await axios.post(CREATE_APPOINTMENT_URL,
                {
                    doctorId: doctorId,
                    patientId: userId,
                    startTime: activeAppDate
                }
            );
            console.log(response);
            setUserId();
            setDoctorId();
            setSuccess(true);

            const activeRows = document.querySelectorAll('.active-row');
            activeRows.forEach(row =>{
                row.classList.remove('active-row')
            });
```

```
            }
            catch(err){
                console.log(err);
            }
        }
        else{
            alert('Обери дату і час прийому!');
        }
    }

    return(
        <div className="create-app-layout">
            <section className="search-user main-section">
                <form
                    className="search-bar"
                    onSubmit={(e) => {
                        e.preventDefault();
                        setSearchUser(searchUserInput);
                    }}>
                    <input
                        type="text"
                        placeholder="Прізвище клієнта"
                        onChange={(e) => setSearchUserInput(e.target.value)}/>
                    <button className="basic-btn"><i class="fa-solid fa-magnifying-
glass"></i></button>
                </form>
                <section className="result-section">
                    <table className="result-table">
                        <thead>
                            <tr>
                                <th>Прізвище</th>
                                <th>Ім'я</th>
                            </tr>
                        </thead>
                        <tbody>
                            {users
                                .filter(item => {
                                    return searchUser === ''
                                        ? item
                                        :
item.lastName.toLowerCase().includes(searchUser.toLowerCase())
                                })
                                .map(user =>
                                    <tr        className="user-row        grid-row"
key={user.uuid} data={user.uuid} onClick={handleClickOnUser}>
                                        <td>{user.lastName}</td>
                                        <td>{user.firstName}</td>
                                    </tr>
                                )
                            }
```

```
                    </tbody>
                </table>
            </section>
        </section>
        <section className="search-doctor main-section">
            <form
                className="search-bar"
                onSubmit={(e) => {
                    e.preventDefault();
                    setSearchDoctor(searcDoctorInput);
                }}>
                <input
                    type="text"
                    placeholder="Прізвище лікаря"
                    onChange={(e) => setSearchDoctorInput(e.target.value)}/>
                <button className="basic-btn"><i class="fa-solid fa-magnifying-
glass"></i></button>
            </form>
            <section className="result-section">
                <table className="result-table">
                    <thead>
                        <tr>
                            <th>Прізвище</th>
                            <th>Ім'я</th>
                            <th>Спеціальність</th>
                        </tr>
                    </thead>
                    <tbody>
                        {doctors
                            .filter(item => {
                                return searchDoctor === ''
                                    ? item
                                    :
item.lastName.toLowerCase().includes(searchDoctor.toLowerCase())
                            })
                            .map(doctor =>
                                <tr       className="doctor-row       grid-row"
key={doctor.doctorId} data={doctor.doctorId} onClick={handleClickOnDoctor}>
                                    <td>{doctor.lastName}</td>
                                    <td>{doctor.firstName}</td>
                                    <td>{doctor.specialties.map((spec, i) => i
=== 0 ? spec.name : ', ' + spec.name)}</td>
                                </tr>
                            )
                        }
                    </tbody>
                </table>
            </section>
        </section>
        <section className="main-section">
```

```
                {doctorId
                ?                  <AppointmentsByDoctor         doctorId={doctorId}
updateDate={setActiveAppDate} createAppointment={createAppointment}/>
                : success
                    ? <div className="filler">Запис успішно доданий</div>
                    :  <div  className="filler">Оберіть  лікаря  і  користувача  для
створення запису</div>}
            </section>
        </div>

    )
}

export default AdminCreateAppointments;
```

index_styles.css

```
@import url('https://fonts.googleapis.com/css2?family=Rubik&display=swap');

*{
    font-size: 17px;
}
span{
    font-size: inherit;
    color:inherit;
}

body{
    font-family: 'Rubik', sans-serif;
    font-size: inherit;
}

h1{
    font-size: 200%;
    text-align: center;
    padding-bottom:30px;
    font-weight: bold;
}

h2{
    font-size: 150%;
    font-weight: bold;
    padding-bottom: 10px;
}

h5{
    font-size: 115%;
    font-weight: bold;
    padding-bottom: 10px;
    padding-top: 10px;
```

```
}

h6{
    font-weight: bold;
}

input{
    height: 30px;
}
.container{
    padding-right: 40px;
    padding-left: 40px;
    width: 100%;
}

#root{
    min-height: 100vh;
    background-color: #FAF9F9;
    display: flex;
    flex-direction: column;
}

/*---------header---------*/
header{
    background-color: #BEE3DB;
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items:center;
}
header a{
    font-size: 110%;
}

#content-nav{
    display: flex;
    flex-direction: row;

}
#header-menu{
    display: flex;
    flex-direction: row;
}
#logo-img{
    width: 90px;
    height: 90px;
    margin-right: 40px;
}
#main-img{
    width: 100%;
```

```
    height: 400px;
    object-fit: cover;
}
#header-menu, #autorization-nav a{
    align-items:center;
}
#header-menu a, #autorization-nav a{
    padding: 20px;
    border-radius: 30px;
}
#header-menu a:hover, #autorization-nav a:hover{
    background-color:#89B0AE;
}
.header-container{
    display: flex;
    flex-direction: row;
    justify-content: space-between;
}
/*---------main---------*/
#wide-section{
    position: relative;
}
#wide-section button{
    position: absolute;
    top: 175px;
    width: 50px;
    height: 50px;
    font-size: 25px;
    background-color: rgba(137, 176, 174, 0.3);
    border-radius:50%;
}
#wide-section button:hover{
    background-color: rgba(137, 176, 174, 0.8);
}

#wide-section i{
    font-size: 30px;
}
#left-arrow{
    left: 40px;
}
#right-arrow{
    right: 40px;
}

#tile-section{
    margin: 50px auto 50px auto;
    padding: 30px;
    box-shadow: rgba(0, 0, 0, 0.24) 0px 3px 8px;
    width: fit-content;
```

```
}
#tile-grid{
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items:center;
}
#tile-grid .arrow{
    width: 50px;
    height: 50px;
    font-size: 25px;
    text-align: center;
}
#tile-grid a{
    display: flex;
    justify-content: center;
}
#tile-grid .tile-icon{
    width: 150px;
    height: 150px;
    margin: 40px;
    font-size: 75px;
    display: flex;
    align-items:center;
    justify-content: center;
    background-color:#FFD6BA;
    border-radius:50%;
}
#tile-grid .tile-icon:hover{
    background-color:#DB9F6B;
}
.tile{
    display: flex;
    flex-direction: column;
    justify-content: center;
    width: 250px;
}
.tile h5, .tile span{
    text-align: center;
}
#about-center{
    padding: 0px 100px 40px 100px;
}
#about-center p{
    text-indent: 40px;
    line-height: 25px;
}
/*---------footer---------*/
footer{
```

```
    background-color:#BEE3DB;
    padding-top: 20px;
    margin-top: auto;
}
footer .container{
    display: flex;
    flex-direction: row;
    justify-content: space-between;
}
footer section{
    padding: 0px 80px 0px 80px;
}
footer a:hover{
    text-decoration: underline;
}
#footer-menu{
    display: flex;
    flex-direction: column;
}
footer section li{
    padding: 3px;
}
#rights{
    font-size: smaller;
}
```