Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-сервіс для дистанційного моніторингу стану здоров'я та взаємодії пацієнтів з лікарями
(тема)

Виконав:
здобувач 4 року навчання,
групи ПЗПІ-21-3

Катерина ВОСКОБОЙНІКОВА
(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Дмитро КОЛЕСНИКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Воскобойніковій Катерині Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-сервіс для дистанційного моніторингу стану здоров’я та взаємодії пацієнтів з лікарями
затверджена наказом університету від 19 травня 2025 р. № 397 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 13.06.2025 р.
3. Вихідні дані до роботи Створити веб-сервіс для дистанційного моніторингу стану здоров’я, який дозволяє пацієнтам відстежувати свої основні показники та взаємодіяти з лікарями. Лікарі, у свою чергу, матимуть змогу вести медичні записи, ставити діагнози та спілкуватися з пацієнтами через систему.
4. Перелік питань, що потрібно опрацювати у роботі Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	09.04.2025	*виконано*
2	Створення специфікації ПЗ	12.04.2025	*виконано*
3	Проєктування ПЗ	13.04.2025	*виконано*
4	Розробка ПЗ	20.05.2025	*виконано*
5	Тестування ПЗ	25.05.2025	*виконано*
6	Написання пояснювальної записки	30.05.2025	*виконано*
7	Підготовка презентації та доповіді	03.06.2025	*виконано*
8	Нормоконтроль, рецензування	09.06.2025	*виконано*
9	Здача роботи у електронний архів	10.06.2025	*виконано*
10	Допуск роботи до захисту	10.06.2025	*виконано*
11	Захист кваліфікаційної роботи	13.06.2025	*виконано*

Дата видачі завдання 4 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Дмитро КОЛЕСНИКОВ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 92 стор., 43 рис., 5 табл., 15 джерел.

АНАЛІТИКА, ВЕБ-СЕРВІС, ДИСТАНЦІЙНИЙ МОНІТОРИНГ, ДІАГНОСТИКА, ЗБІР ДАНИХ, ЛІКАР, МЕДИЧНІ ДАНІ, ПАЦІЄНТ, СТАН ЗДОРОВ’Я, ANGULAR 17, ASP.NET CORE, C#, MS SQL.

Об’єктом дослідження даної кваліфікаційної роботи бакалавра є галузь охорони здоров’я та розробка веб-сервісу для забезпечення ефективного та зручного дистанційного моніторингу стану здоров’я.

Метою кваліфікаційної роботи є проєктування та розробка програмного забезпечення, призначеного відстеження ключових показників здоров’я пацієнтів на відстані. Система забезпечує збір, надійне зберігання та аналіз медичних показників пацієнтів (таких як температура, пульс, тощо), управління електронними медичними картками та діагнозами, візуалізацію статистики стану здоров’я пацієнтів, а також ефективну комунікацію між пацієнтом та лікарем.

Для розробки використовуються такі технології, як C# та ASP.NET Core 8.0 для серверної частини, MS SQL Server 2019 для бази даних, Angular 17 для клієнтської частини, а також окремий сервіс для IoT-пристрою на базі ASP.NET Core 8.0.

У результаті виконання роботи було розроблено веб-систему, яка надає можливості для забезпечення моніторингу в дистанційному режимі.

ABSTRACT

ANALYTICS, WEB SERVICE, REMOTE MONITORING, DIAGNOSIS, DATA COLLECTION, DOCTOR, MEDICAL DATA, PATIENT, HEALTH STATUS, ANGULAR 17, ASP.NET CORE, C#, MS SQL.

The object of study in this bachelor's qualification work is the healthcare sector and the development of a web service to ensure effective and convenient remote health monitoring.

The aim of the qualification work is to design and develop software intended for the remote monitoring of key health indicators of patients. The system provides data collection, reliable storage, and analysis of patients' medical indicators (such as temperature, pulse, etc.), management of electronic medical records and diagnoses, visualization of health statistics, and effective communication between patients and doctors.

The development utilizes technologies such as C# and ASP.NET Core 8.0 for the backend, MS SQL Server 2019 for the database, Angular 17 for the frontend, and a separate service for an IoT device based on ASP.NET Core 8.0, with a Raspberry Pi used as a sample device.

As a result of the work, a web system was developed that provides opportunities for remote health monitoring.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IoT – Internet of Things

RBAC – Role-Based Access Control

REST – Representational State Transfer

ВСТУП

У наш час все більша кількість людей замислюється над питанням необхідності піклування про особисте здоров’я [1]. Якщо раніше це було модною тенденцією для малої кількості людей, то зараз це вже стало усвідомленою та обов’язковою складовою нашого щоденного життя. Тільки завдяки цьому ми можемо забезпечити собі гарне та повноцінне існування. Незупинний технологічний прогрес також сприяв цьому – щодня розробляються нові методи та системи у галузі медицини, відчиняючи широкий простір для покращення якості та доступності медичних послуг [2]. Важливість таких систем особливо зростає у випадках занятості медичних установ та для тих людей, які не можуть регулярно відвідувати лікарські заклади через велику відстань або нестачу часу [3]. Можна уявити собі літню жительку віддаленого села або молоду матір, якій складно знайти вільний час на прийом до лікаря: для них можливість дистанційного спілкування з лікарем стає справжнім порятунком.

Результати цієї кваліфікаційної роботи орієнтовано на використання у медичній галузі і можуть бути впроваджені у повсякденний робочий процес різних типів лікарень і корисними для приватних лікарів та пацієнтам, які потребують цілодобового нагляду.

Актуальність даного проєкту полягає у бажані оптимізувати роботу медичних установ та покращити якість лікування. Крім того, важливо забезпечити пацієнтам доступ до своєї медичної історії незалежно від їхнього місця перебування. Дистанційне спостереження допоможе вчасно виявляти проблеми на ранньому етапі – це ключовий аспект якісного та успішного лікування. Наприклад, при ранньому виявленні раку кишечника виживання упродовж наступних 5 років перевищує 90%, тоді як при виявленні на пізніх стадіях воно знижується приблизно до 10%. [4]. Такі цифри як ніхто інший демонструють ціну часу в питаннях медицини.

Саме тому завданням цієї роботи є проєктування та створення сучасного веб-сервісу для контролю за станом здоров’я пацієнтів на відстані. У це завдання також включено детальний аналіз існуючих рішень та конкурентів, формування вимог до майбутньої системи, а також розробку для неї оптимальної архітектури.

При виконанні проєкту були застосовані знання, отримані під час навчання за спеціальністю «Інженерія програмного забезпечення». Сама робота була виконана при дотримуванні принципів чистої архітектури, запропонованими Робертом Мартіном [5].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сучасні реалії кожного дня ставлять перед нами певні виклики, одним з яких та напевно найголовнішим є збереження свого здоров’я. Висока завантаженість на роботі, що особливо актуально у великих містах, та банальна географічна віддаленість від медичних установ часто стають тими викликами, що ускладнюють відвідування лікарень та отримання медичної допомоги. Також всім знайомі ситуації, коли люди, котрі навіть відчувають нездужання, відтягують чи взагалі відмовляються від візиту до лікаря через небажання відволікатися від «більш важливих» на їх погляд справ. Така ситуація може призводити до несвоєчасної діагностики захворювань на ранніх етапах та обмежувати доступ до дієвої та оперативної терапії [6]. Всім відомо, що більшість серйозних захворювань, наприклад серцево-судинні чи онкологія, найлегше лікуються саме на ранніх етапах. Дуже легко уявити людину, яка не може ніяк потрапити до лікаря через щільний робочий графік, бо, напевно, майже кожен стикався з такою ситуацією особисто, або спостерігав у своєму оточенні.

Окрім цього, навіть коли є бажання та час звернутися по допомогу, люди часто стикаються з нудним очікуванням у чергах, яке іноді може розтягнутися на години, труднощами із записом на прийом. Все це не лише знижує доступність медичних послуг, але й може створити додатковий стрес для людини, яка і без того має проблеми зі здоров’ям. Такий стан речей у галузі охорони здоров’я змушує деяких людей відмовлятися від візиту до лікаря або взагалі зайнятися самолікуванням, що при відсутності медичних знань часто тільки погіршує ситуацію. Також не слід забувати, що наразі в Україні присутній дефіцит медичних працівників. Медики змушені працювати в постійному напруженні та в умовах обмежених ресурсів. Згідно з даними Українського центру охорони здоров’я, у 2022 році загальна кількість лікарів в Україні зменшилася на 14% порівняно з

попереднім роком. Особливо відчутне скорочення спостерігалося серед молодшого медичного персоналу, чисельність якого знизилася на 16,5% [7].

І саме тому в останні роки з’являються та розробляються все більше нових систем та розробок для галузі охорони здоров’я, які допомагають подолати наявні проблеми. Діджіталізація вже стала глобальним трендом, охопивши за останні роки всі сфери нашого життя. Не оминула вона і медичну сферу, з’явився напрямок телемедицини та eHealth. Проєкт, який розробляється в рамках даної кваліфікаційної роботи, якраз і належить до подібних ініціатив. Його реалізація повинна не просто усунути декілька обмежень, а зробити стрибок у рівні надання медичних послуг, зробивши їх більш доступними для будь-кого. Окрім цього, паралельно це дозволить знизити навантаження та завантаженість лікарень та їхніх працівників. Таким чином, з’являться вільні людські ресурси для більш складних випадків, а це вже покращить ефективність медичної системи в цілому.

Як було вже сказано, саме для вирішення всіх зазначених проблем і розроблюється веб-сервіс, який повинен стати надійним мостом між пацієнтом та лікарем. Ключова ідея – надати кожному інструменти для ефективного віддаленого моніторингу показників власного здоров’я, незалежно від часу та місця перебування. Завдяки цьому проєкт має на меті усунути існуючі незручності традиційної моделі охорони здоров’я. Це буде реалізовано через створення інтуїтивно зрозумілої та безпечної системи, яка дозволить легко й регулярно передавати важливі дані про свій стан. З іншого боку, медичні працівники отримають можливість оперативно відстежувати динаміку цих показників, вчасно реагувати на тривожні зміни та надавати віддалені консультації, запрошувати на персональний прийом тільки коли на це є реальна необхідність. Також важливою складовою є бажання об’єднати всю важливу медичну інформацію, це включає, наприклад, історію хвороби, результати аналізів, в єдиному цифровому реєстрі, доступному як пацієнту, так і його лікарю. Це допоможе підвищити прозорість лікувального процесу та сприятиме кращій координації медичної допомоги [8].

1.2 Аналіз існуючих аналогів

Для визначення потенційних проблем, які можуть заважати проєкту в досягненні успіху та прийняття серед користувачів, слід спочатку провести аналіз поточної ситуації на ринку та наявних конкурентів. Це також буде корисно для виявлення їхніх слабких сторін, які можна використати як переваги для власного проєкту.

«Apple Health» можна розібрати як найбільш популярного конкурента. Ця платформа, створена Apple, слугує інструментом для власників пристроїв Apple, що дозволяє користувачам зберігати, відстежувати та керувати даними про власне здоров’я. За допомогою «Apple Health» користувачі можуть фіксувати різноманітні показники, як-от кількість кроків, пульс, якість сну, кількість різних харчових речовин та інше (див. рис. 1.1). Для збору більшості основних показників необхідний смарт-годинник або, у випадку з показниками по харчуванню, сторонній додаток.

Рисунок 1.1 – «Apple Health». Головна сторінка, інтерфейс виведення основних показників

Також у додатку присутній дуже детальний аналіз по великій кількості категорій показників, що дозволяє користувачу бачити ситуацію в його повсякденній поведінці та легко помічати зміни в ній (див. рис 1.2). Статистика зроблена дуже зручно та якісно, є порівняння даних за різні періоди, стислі висновки та рекомендації по зібраним даним. На мій погляд, це є основною сильною стороною додатку, оскільки перетворює набір незрозумілих цифр на структуровану інформацію, яку одразу зрозуміє будь-який користувач. Інтерфейс також інтуїтивно зрозумілий та зручний, у стилі продуктів Apple.

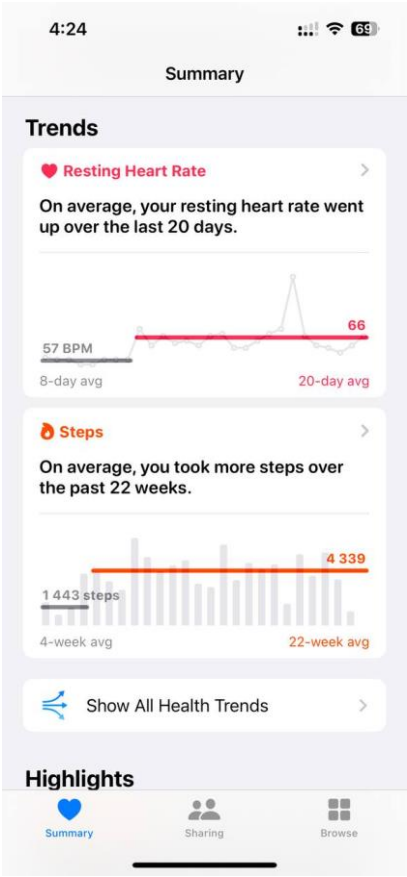

Рисунок 1.2 - «Apple Health». Головна сторінка, статистика та аналіз показників здоров’я

Однак, хоча додаток надає великий набір можливостей для власного моніторингу, функціонал цієї системи обмежений виключно особистим використанням та не передбачає прямої взаємодії з лікарнями чи іншими медичними закаладами. До того ж, користуватися додатком можуть виключно

власники їхніх пристроїв, так як додаток є вбудованим елементом IOS (операційна система Apple). Це дуже сильно звужує коло користувачів, тобто автоматично виключає власників пристроїв, які працюють на Android, та десктопні пристрої. Якщо розглядати кількість втрат користувачів на прикладі Android-пристроїв, то станом на 2025 рік пристроями Android користується приблизно 69.9% користувачів [9]. З цього можна зробити висновок, що «Apple Health» недоступний для переважної більшості користувачів. Користуватися повним функціоналом додатку «Apple Health» можна безкоштовно, а прибуток компанія отримує від продажу додаткових гаджетів у сфері здоров'я (наприклад, Apple Watch).

Таким чином, «Apple Health» є дійсно зручним та корисним сервісом для індивідуального контролю за своїм станом, але він не забезпечує повноцінний дистанційний моніторинг та будь-який контакт з лікарями, що є однією з головних цілей проєкту.

Далі розглянемо веб-сервіс «Doximity» (див. рис. 1.3). По суті, Doximity створена як професійна соціальна мережа для лікарів. Головна її мета – створити захищений простір, де фахівці можуть безпечно спілкуватися, обмінюватися досвідом та організовувати віртуальні консиліуми та консультації для пацієнтів. За допомогою платформи лікарі можуть швидко зв'язуватися з колегами, що є дуже важливим для координації лікування.

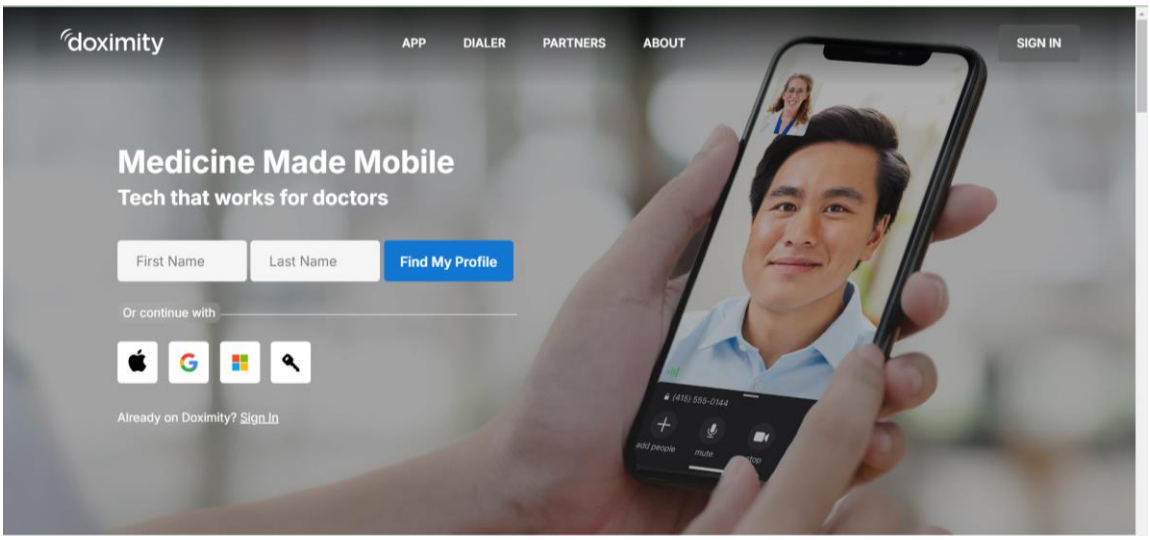

Рисунок 1.3 – Головна сторінка «Doximity»

Слід зазначити наявність вбудованих засобів захисту, а саме здійснення HIPAA-сумісних (тобто таких, що відповідають американським стандартам захисту медичної інформації) відеодзвінків та текстових повідомлень.

Однак сервіс має також певні недоліки, основний фокус зосереджено на лікарському середовищі і залучення пацієнтів як користувачів сайту досить приховане. Пацієнти можуть брати участь у консультаціях, але вони не є активними користувачами системи у контексті щоденного моніторингу здоров'я. Як наслідок, у Doximity практично відсутні функції для будь-якого моніторингу в реальному часі чи збору біометричних даних.

Ще одним конкурентом можна вважати «MyChart», який широко використовується американськими лікарнями (див. рис. 1.4). Ця платформа має функціонал, який дуже схожий на основні цілі даного проєкту і може вважатися головним конкурентом. Вона надає пацієнтам захищений онлайн-доступ до їхніх електронних медичних карток.

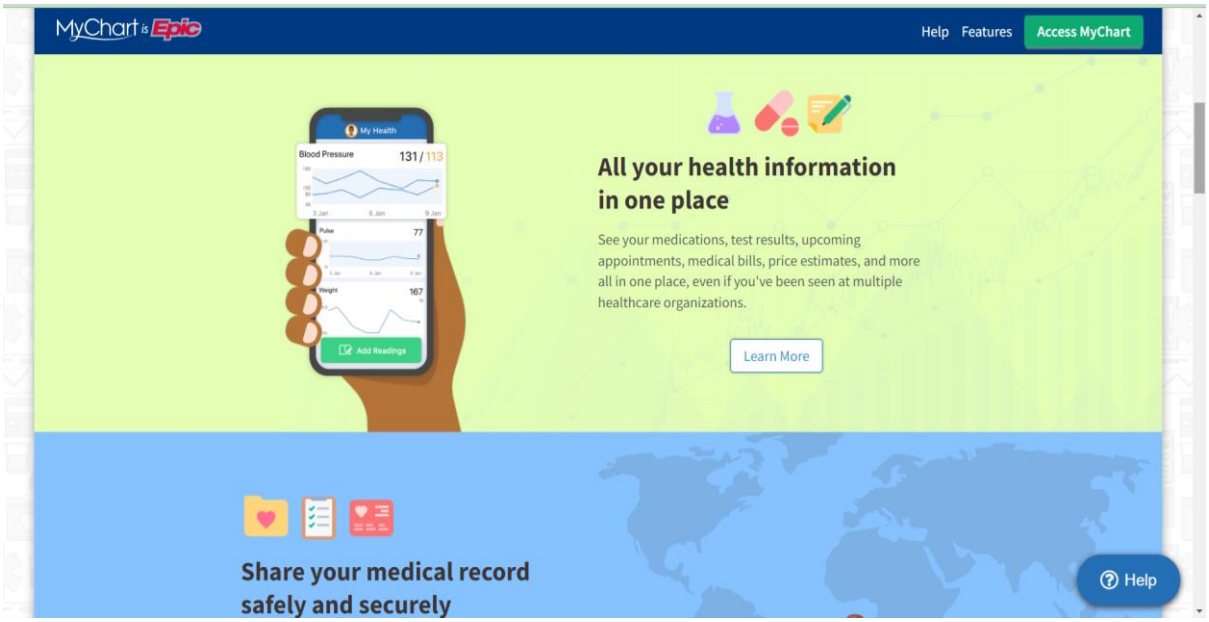

Рисунок 1.4 – Головна сторінка «MyChart»

Серед беззаперечних переваг зручний, простий та інтуїтивно зрозумілий інтерфейс. Він дозволяє пацієнтам переглядати результати лабораторних аналізів,

історію візитів до лікаря, список призначених ліків, тобто всю свою медичну історію, що зберігається в одному місці.

Проте, тут також немає функціоналу для моніторингу в реальному часі або введення даних з персональних медичних пристроїв, але система чудово відображає те, що вже було зафіксовано лікарями.

Отже, після проведення аналізу наявних аналогів, можемо зробити висновок, що об’єднання комунікації з лікарем та автоматичного збору показників здоров’я в одній платформі надасть даному проєкту значну перевагу на ринку, так як ще немає сервісів, які б мали одразу обидві частини цього функціоналу.

1.3 Виявлення проблем та ризиків програмної системи

При розробці будь-якого проєкту є велика вірогідність стикнутися з різними проблемами та ризиками. Для проєкту, який працює з чутливою медичною інформацією, найбільшою та першочерговою проблемою може стати витік даних, порушення конфіденційності, тому цьому питанню слід приділити особливу увагу. Так як витік особистої медичної інформації це дуже серйозна проблема, яка може зруйнувати репутацію та довіру до сервісу, у першу чергу треба подбати про методи захисту даних. Потрібно застосовувати сучасні методи захисту, наприклад, шифрування даних, перевірку доступу, а також час від часу проводити оновлення системи захисту, у випадку виявлення недосконалостей та вразливостей.

Також важливим ризиком є поява конкурентів, які можуть запропонувати користувачам подібний або навіть кращий функціонал. Ця ситуація цілком може виникнути, так як ринок телемедицини розвивається все більше і більше. Саме тому важливо постійно утримувати увагу користувачів, пропонувати їм щось нове та інноваційне, що вимагає відстеження та аналізу стану речей на ринку медичних програмних систем, досліджувати та аналізувати нові тенденції, додавати їх до власної системи при необхідності. Також важливо отримувати зворотний зв’язок від користувачів для вдосконалення сервісу.

Ще однією проблемою може стати негативна реакція від користувачів або несприйняття системи. З цією проблемою може стикнутися абсолютно будь-який новий проєкт. Навіть якщо система буде мати передовий функціонал, добре працювати, її можуть просто проігнорувати або не сприйняти. Щоб цьому запобігти, треба детально спланувати стратегію просування, рекламувати систему в найвідоміших соціальних мережах, брати участь у тематичних заходах, конференціях.

Ще один ризик – це технічні проблеми в процесі роботи. Це можуть бути, наприклад, збої в роботі серверів або повне їх відключення. Такі ситуації можуть зробити систему тимчасово недоступною, що буде негативно впливати на користувацький досвід та репутацію платформи. Щоб зменшити ймовірність таких ситуацій, краще мати резервні сервери, які будуть працювати у випадку відмови основних, а також передбачити відмовостійку архітектуру.

1.4 Постановка задачі

Провівши детальний аналіз предметної галузі, конкурентів та можливих ризиків і проблем, що можуть виникнути, як на етапах розробки, так і під час використання системи, ми можемо тепер сформулювати задачу, яка ляже в основі реалізації нашого веб-сервісу.

Як було вже визначено на попередньому етапі, забезпечення безпеки та конфіденційності медичних даних є головним пріоритетом. У систему, яка буде працювати з такою чутливою інформацією, обов’язково треба додати певні механізми захисту. У якості цих механізмів можна впровадити систему авторизації за певними ролями (у нашому випадку це адміністратор, лікар і пацієнт), дані користувачів повинні шифруватися під час їх передачі, додати хешування паролів, захист API і створити політику конфіденційності. У цій політиці прописати конкретні умови роботи веб-сервісу з інформацією користувачів, які дані ми збираємо та як їх зберігаємо.

Так як веб-сервіс буде орієнтовано для широкого кола звичайних людей, то також дуже важливо розробити зручний та зрозумілий інтерфейс, який підійде для всіх типів користувачів. Пацієнтам слід надати доступ до перегляду їхніх медичних показників (вага, пульс, насичення киснем, тощо), статистики на їх основі, а також можливість отримувати рекомендації та повідомлення від лікаря. Лікарі, у свою чергу, також повинні мати доступ до аналітики показників пацієнтів, його медичної карти, можливість створювати медичні записи й діагнози, надсилати повідомлення пацієнтам. І для адміністратора потрібно реалізувати окремий інтерфейс для управління всією системою. Це включає управління користувачами, записами в базі даних, правами доступу та загальною підтримкою системи.

Однією з ключових задач є організація ефективної комунікації між пацієнтами та лікарями. Це означає, що необхідно створити внутрішню просту систему повідомлень, яка і дозволить їм обмінюватися короткими текстовими повідомленнями, рекомендаціями, тощо. Також у майбутньому можна буде розглянути можливість розширити функціонал і додати відеозв'язок, якщо це буде актуально та того вимагатиме ситуація.

Окремим блоком задач виступає реалізація функціоналу збору та відстеження показників здоров'я, який також є однією з основних функцій і задач нашого веб-сервісу. У рамках даного проєкту це означає створення симулятора реальних пристроїв (наприклад, розумних годинників), які передають дані до бази даних в реальному часі або з певною періодичністю. Тобто задача полягає в тому, щоб дані збиралися, надходили до сервера, оброблялися там, зберігалися в базі даних та були доступними для перегляду в клієнтському інтерфейсі.

Також важливою задачею є реалізація багатомовності у системі. Тобто додати локалізацію принаймні українською та англійською мовами, що на першому етапі майбутнього впровадження веб-сервісу дозволить охопити досить широку аудиторію.

Для вирішення зазначених вище задач оптимальним рішенням буде використання ASP.NET Core 8.0 [10] та мови програмування C# для реалізації

серверної частини. Працювати з БД серверна частина зможе шляхом застосування Entity Framework Core 8.0. Для реалізації системи авторизації за певними ролями будемо використовувати JSON Web Token. Таким чином серверна частина зможе обробляти запити швидко й стабільно, навіть при великій кількості користувачів. Для зберігання всієї важливої інформації базою даних слугуватиме Microsoft SQL Server [11]. Робота з базою даних буде відбуватися у середовищі Microsoft SQL Server Management Studio 18, яке має всі необхідні для поставлених задач інструменти.

Так як з боку клієнтської частини необхідно реалізувати сучасний та адаптивний інтерфейс, будемо використовувати для цього Angular 17 [12], який працює на базі TypeScript. Задача тут полягає не лише в побудові зручного UI, а й у забезпеченні швидкої взаємодії з сервером і правильного відображення динамічних даних.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги

Веб-сервіс призначено для контролю стану здоров’я на відстані, полегшення комунікації між пацієнтами та лікарями, а також ефективного управління медичними записами. Основними користувачами системи є пацієнти, лікарі та адміністратори, кожен з яких має доступ до певного функціоналу відповідно до своєї ролі.

Враховуючи це, можемо визначити наступний набір функцій:

- реєстрація та вхід для користувачів з різними ролями, а саме пацієнтів, лікарів та адміністраторів;
- отримання та обробка фізіологічних показників користувача (пульс, температура, насичення киснем, вага тощо) з використанням IoT-пристроїв;
- перегляд всіх медичних показників конкретного пацієнта;
- створення та перегляд медичних записів лікарями для кожного пацієнта;
- обмін повідомленнями між пацієнтами та лікарями;
- перегляд профілів;
- редагування власного профілю;
- формування статистичних графіків та аналіз змін у стані здоров’я на основі даних, зібраних пристроями.

2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики якості веб-сервісу, які не залежать від конкретного функціоналу, але суттєво впливають на зручність, безпеку та продуктивність системи, що також відіграє ключову роль у створенні позитивного досвіду взаємодії з системою. Для визначення

нефункціональних вимог можемо використати висновки, отримані у результаті аналізу предметної галузі.

Отримуємо тоді наступний набір нефункціональних вимог:

– підтримка багатомовності інтерфейсу (локалізація українською та англійською мовами);
– зручний та зрозумілий інтерфейс користувача, без перевантаження великою кількістю різних елементів, розроблений у відповідності до принципів UI/UX-дизайну, у єдиній кольоровій палітрі, без різких та «агресивних» кольорів;
– наявність навігаційної панелі з основними сторінками, які потрібні користувачу, для легкої та швидкої навігації без обмежень;
– захищене з’єднання з сервером за допомогою протоколу HTTPS;
– миттєва реакція інтерфейсу, усі запити до сервера мають оброблятися з максимальною затримкою не більше 5 секунд;
– уся медична інформація повинна зберігатись у зашифрованому вигляді та бути доступною лише користувачам з відповідними правами;
– користувачі з різними правами повинні мати чітко обмежені дії у межах своєї ролі;
– система має коректно обробляти всі можливі помилки, не допускаючи критичних збоїв, користувач повинен отримувати відповідне повідомлення про помилку;

2.3 Припущення та залежності

Для кращої реалізації системи та її адаптації до всіх можливих ситуацій, можна визначити наступний набір припущень:

– користувачі мають базові навички, які необхідні для користування веб-браузером та інтернетом;

- користувачі погоджуються на обробку персональних та медичних даних відповідно до визначеної політики конфіденційності;
- усі учасники системи (пацієнти, лікарі, адміністратори) мають стабільне підключення до Інтернету;
- дані, отримані від IoT-пристроїв, є достовірними і не потребують додаткової верифікації на рівні системи;
- зміни до медичних записів можливі лише після авторизації користувача з відповідними правами;
- лікарі та пацієнти взаємодіють виключно в межах системи, сторонні канали спілкування не регламентуються;

Отже, маючи конкретний набір визначених вимог, ми можемо реалізувати веб-сервіс, який повністю вирішить потреби лікарів та пацієнтів у питаннях віддаленого спостереження за пацієнтами та їх станом здоров’я.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ

3.1 Проєктування структури бази даних

Основою будь-якої інформаційної системи, яка працює з великими обсягами даних, є добре продумана база даних. Від її структури залежить не лише ефективність зберігання та обробки інформації, але й цілісність даних та можливість реалізації всієї запланованої функціональності. Тому процес проєктування нашого веб-сервісу почався саме з проєктування бази даних.

На першому етапі цього процесу, з метою кращого розуміння та чіткого визначення основних сценаріїв взаємодії майбутніх користувачів з системою, змоделювали діаграму прецедентів (Use Case Diagram). Діаграма дозволяє ідентифікувати ключових акторів та дії, які вони можуть виконувати в рамках системи. Отримана діаграма прецедентів (див. рис. 3.1) ілюструє основні функціональні можливості, доступні для чотирьох типів користувачів: пацієнта, лікаря, адміністратора системи та неавторизованого користувача.

Рисунок 3.1 – Діаграма прецедентів веб-сервісу

Аналізуючи діаграму, можна бачити, що, зокрема, пацієнт як ключовий користувач системи, має можливість не лише переглядати свою власну медичну інформацію (що включає історію вимірювань, медичні записи, призначене лікування), але й активно взаємодіяти зі своїм лікуючим лікарем через систему повідомлень. Зі свого боку, лікарі отримують інструменти для ефективного керування медичними записами своїх пацієнтів, призначення консультацій, надсилання важливих повідомлень та, що не менш важливо, перегляду статистики та динаміки вимірювань пацієнтів для прийняття обґрунтованих рішень. Роль адміністратора є більш технічною та передбачає здійснення загального управління системою, включаючи менеджмент облікових записів користувачів, налаштування прав доступу та моніторинг стану системи. Навіть неавторизований користувач має визначений набір дій, таких як ознайомлення з загальною інформацією про платформу та можливості реєстрації або входу.

Після детального аналізу наведених вище діаграми прецедентів та функціональних вимог, ми можемо сформувати набір основних інформаційних об’єктів, які потребують постійного зберігання та управління, і, відповідно, мають бути реалізовані як сутності в базі даних. До цих ключових об’єктів належать: пацієнт (зберігає персональні та контактні дані пацієнтів), вимірювання (фіксує конкретні показники здоров’я, отримані з пристроїв), вимірювальний пристрій (описує конкретний екземпляр IoT-пристрою, закріплений за пацієнтом), тип вимірювального пристрою (довідник типів пристроїв, наприклад, розумний годинник, ваги), медичні записи (записи, створені лікарем, що стосуються діагнозів, призначень тощо), повідомлення (для комунікації між лікарем та пацієнтом), лікар (аналогічно до пацієнта, зберігає дані про медичних працівників), спеціалізація (довідник лікарських спеціалізацій) та групи крові (довідник). Кожна з цих сутностей інкапсулює набір атрибутів, що описують відповідний об’єкт реального світу.

На основі цих визначених сутностей та їхніх атрибутів далі була побудована ER-діаграма (див. рис. 3.2) для візуального відображення логічної структури бази

даних, показуючи сутності та зв’язки між ними [13, с. 405-430]. Як видно на діаграмі, здебільшого ці зв’язки мають тип «один до багатьох».

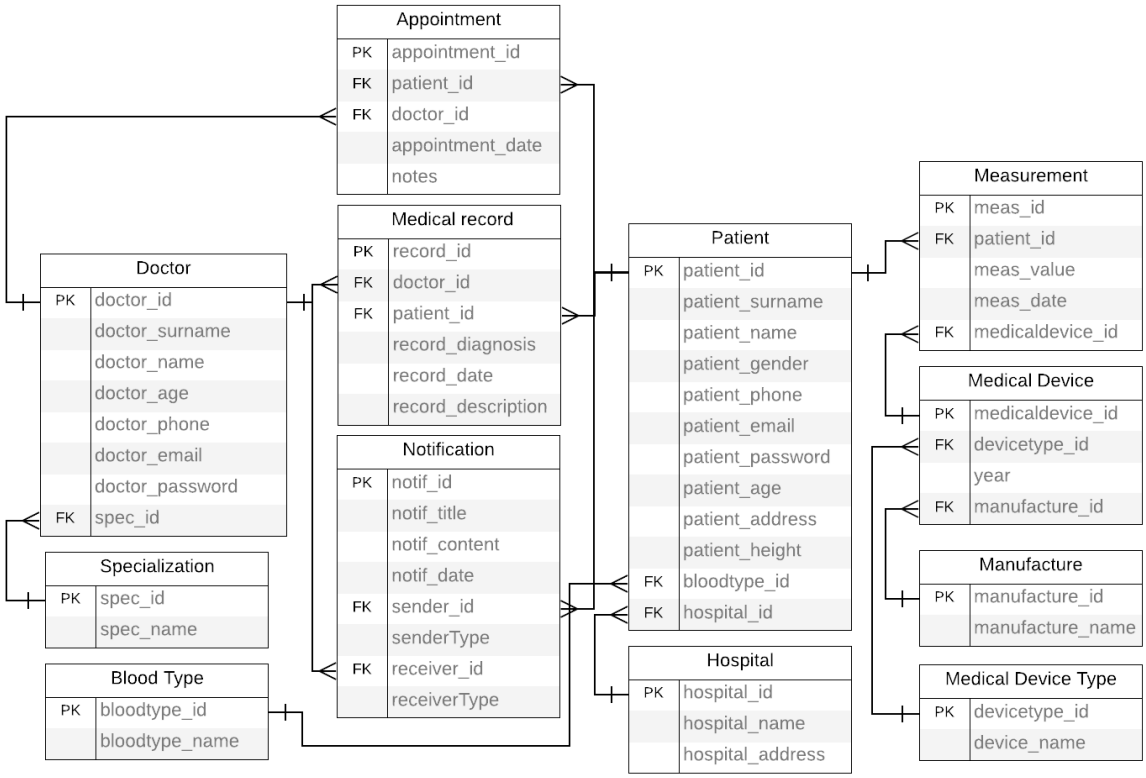

Рисунок 3.2 – ER-діаграма веб-сервісу

Далі розроблено діаграму розгортання (див. рис. 3.3), яка наочно ілюструє загальну архітектуру системи та взаємодію її ключових програмних компонентів.

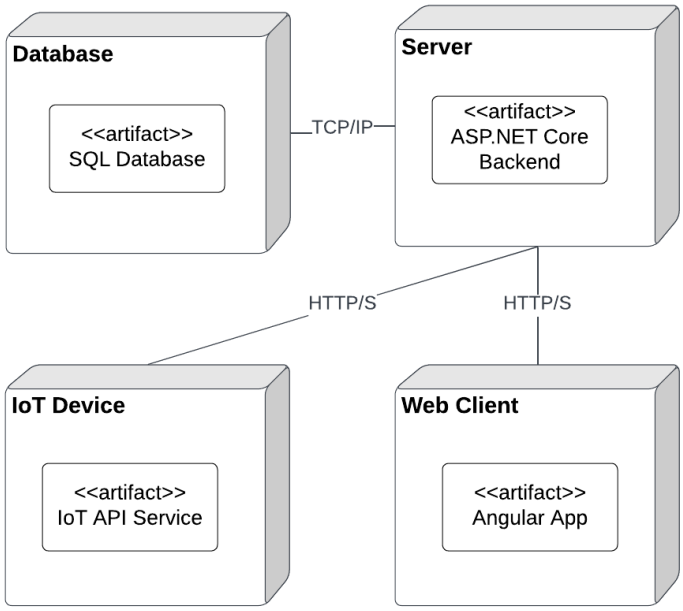

Рисунок 3.3 – Діаграма розгортання веб-сервісу

Діаграма показує, як клієнтська частина, реалізована у вигляді веб-додатку, взаємодіє з серверною частиною, через свій веб-інтерфейс надсилає HTTP-запити до серверної частини. Сервер, зі свого боку, відповідає за обробку цих запитів: виконує ключові операції, контактує з базою даних для збереження, обробки й доступу до даних, а також забезпечує отримання та обробку зчитаних даних від IoT-пристроїв. IoT-пристрої пацієнтів також виступають як джерела даних, що передають інформацію на сервер. Така трирівнева архітектура (клієнт-сервер-база даних) є дуже поширеним підходом для розробки веб-застосунків, забезпечуючи чіткий розподіл ролей та обов’язків між окремими компонентами, а також надаючи гнучкість у розгортанні та масштабуванні кожного з них незалежно.

Отримана структура бази даних та загальна архітектура системи проєктувалась з урахуванням майбутнього масштабування та розвитку системи. Усі паролі користувачів зберігатимуться у базі даних виключно у хешованому вигляді, що унеможливлює їх відновлення навіть у випадку несанкціонованого доступу до самої бази даних.

3.2 Серверна частина

3.2.1 Проєктування функціональності серверної частини

Для чіткого визначення того, як саме користувачі будуть взаємодіяти із системою на серверному рівні, та для забезпечення розділення доступу, функціональність серверної частини була спроєктована на основі ролей користувачів. Такий підхід, відомий як RBAC, є стандартом для побудови безпечних та керованих систем. Як було детально показано на діаграмі прецедентів (див. підрозділ 3.1, рис. 3.1), у системі передбачається чотири основні ролі, кожна з яких матиме свій унікальний набір дозволених операцій та доступ до певних даних:

а) неавторизований користувач: ця роль призначена для відвідувачів, які ще не увійшли в систему або не мають облікового запису. Для них серверна частина

повинна забезпечувати можливість перегляду загальної, публічної інформації про систему (наприклад, опис платформи, її переваги, контактна інформація). Ключовою функціональністю для цієї ролі є надання інструментів для реєстрації нових користувачів та авторизації вже існуючих користувачів через введення логіна та пароля;

б) пацієнт: після авторизації пацієнт отримує доступ до значно ширшого спектра функцій. Сервер має забезпечити доступ до його особистого профілю, де він може переглядати та редагувати свої контактні дані та деякі налаштування. Забезпечити доступ до власних медичних записів, залишених лікарем, можливість отримувати та надсилати повідомлення лікарям, перегляд статистики та візуалізації даних, отриманих зі своїх смарт-пристроїв;

в) лікар: авторизований лікар отримує інструменти для своєї професійної діяльності в рамках системи. Серверна частина повинна надати йому можливість керувати медичними записами та повідомленнями для своїх прикріплених пацієнтів. Це також включає створення нових записів, їх перегляд, редагування та, за потреби, видалення;

г) адміністратор: ця роль відповідає за загальне функціонування та адміністрування системи. Серверна частина повинна надати інструменти для управління користувачами системи (створення, блокування, видалення облікових записів, призначення ролей) та прямої управління даними, які зберігаються в базі даних.

3.2.2 Проєктування архітектури серверної частини

Було вирішено розробляти серверну частину за модульним принципом. Система буде складатися з набору відносно незалежних компонентів (модулів), кожен з яких відповідає за свою частину функціональності. Такий підхід дозволить зробити чіткий розподіл обов’язків між різними компонентами, спрощіть розробку,

тестування та підтримку системи, а також дозволить легше вносити зміни в один модуль, не чіпаючи інші.

Центральним елементом програми для ініціалізації та конфігурації в сучасних .NET-застосунках виступає клас Program.cs. В ньому і будуть налаштовані всі шляхи API та рядок підключення до бази даних.

Обробку HTTP-запитів, які будуть надходити від клієнтської частини здійснюватимуть спеціалізовані класи, які називаються контролерами (наприклад, UserAccountController для операцій з обліковими записами, MedicalRecordController для медичних записів, тощо). Основне завдання контролерів – приймати HTTP-запити (GET, POST, PUT, DELETE тощо), проводити їх первинну перевірку (валідацію вхідних даних на відповідність формату), витягувати необхідні дані із запиту та передавати їх для подальшої обробки до відповідних сервісів [14, с. 86-103]. Наприклад, GET-запит на отримання медичних записів конкретного пацієнта буде направлено до відповідного методу в MedicalRecordController, який потім викличе метод у MedicalRecordService для отримання цих даних.

Основна бізнес-логіка системи буде реалізована у сервісах. Сервісний шар є серцем додатку, де реалізуються всі складні операції та алгоритми. Щоб організувати обмін даними та роботу з базою даних, буде використано патерн Репозиторій (Repository).

Надзвичайно важливою складовою архітектури серверної частини є система авторизації. Для захисту API та даних користувачів планується використовувати механізм на основі JSON Web Token. При кожній авторизації користувач з певною роллю отримує унікальний JWT-токен. Сформований токен включає зашифровані дані про користувача та рівень його доступу, і надалі він буде додаватися до кожного запиту. Сервер буде валідувати токен і визначати, чи має користувач право на виконання запитуваної операції.

3.3 Клієнтська частина

3.3.1 Проєктування функціональності клієнтської частини

На початковому етапі проєктування клієнтської частини було важливо чітко визначити основні сценарії взаємодії різних типів користувачів з майбутнім веб-інтерфейсом. Хоча загальні ролі та їхні можливості вже були окреслені на етапі проєктування серверної частини, для фронтенду важливо було деталізувати саме те, як ці можливості будуть представлені та доступні через візуальні елементи. Тому для клієнтської сторони була розроблена окрема діаграма прецедентів (див. рис. 3.4).

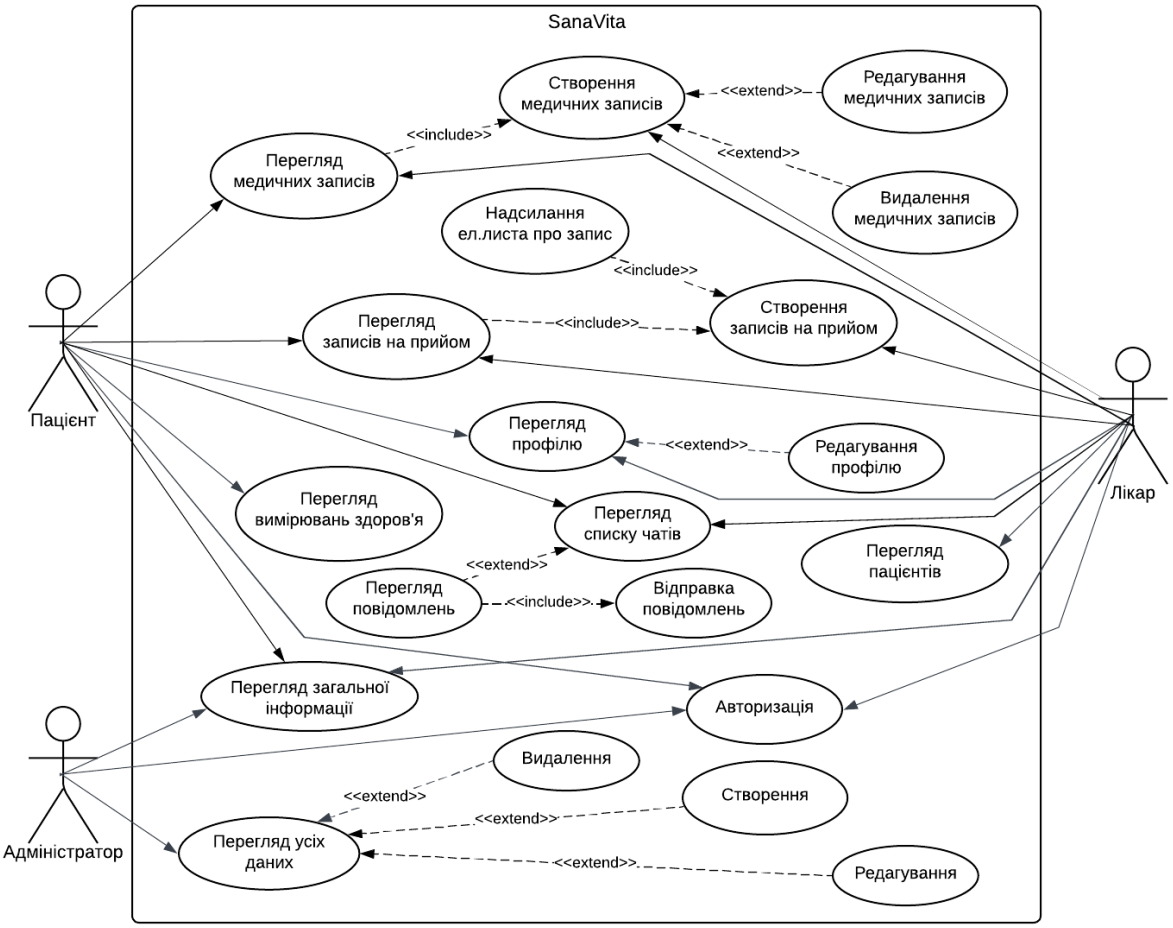

Рисунок 3.4 – Діаграма прецедентів клієнтської частини

Ця діаграма охоплює три основні типи авторизованих користувачів, які будуть взаємодіяти з веб-інтерфейсом: пацієнтів, лікарів та адміністраторів. Кожна

роль, як видно з діаграми, має свій унікальний набір функцій, реалізованих через відповідні сторінки та елементи інтерфейсу. Наприклад, пацієнти зможуть бачити свою медичну інформацію, яка була зібрана за весь час користування сервісом, отримувати візуалізовану статистику показників з IoT-пристроїв, спілкуватися з лікарями та керувати профілем. Для лікарів клієнтська частина надасть інструменти для ефективного ведення пацієнтів. Адміністратори системи через клієнтський веб-інтерфейс отримують доступ до спеціальної панелі, де вони зможуть здійснювати загальне управління системою.

Таким чином, основним завданням клієнтської частини є створення зручного, інтуїтивно зрозумілого та функціонально насиченого інтерфейсу для ефективної роботи з медичними даними та взаємодії між усіма учасниками процесу. Саме тому на етапі проєктування особлива увага приділялась логічній структурі майбутніх сторінок, продуманій навігації та створенню користувацького інтерфейсу, який є зручним та викликає бажання ним користуватися.

3.3.2 Проєктування архітектури клієнтської частини

Клієнтську частину буде розроблено з використанням Angular 17, на базі якого можна розроблювати модульні та продуктивні односторінкові додатки. Архітектура базуватиметься на компонентному підході: кожен елемент інтерфейсу (сторінки, панелі, форми) реалізується як окремий компонент з власною логікою, розміткою та стилем. Використання компонентів зможе допомогти у випадку повторного використання коду та спростить сам процес розробки.

Для забезпечення інтерактивності користувацького інтерфейсу застосовуватиметься двостороннє зв’язування даних (Two-Way Data Binding), яке дозволяє автоматично синхронізувати зміни між моделлю та інтерфейсом без необхідності оновлення сторінки. Архітектура проєкту буде побудована за модульним принципом: головний модуль (AppModule) відповідатиме за об’єднання компонентів та сторонніх бібліотек. Взаємодія з API серверної частини

буде реалізована за допомогою Angular-сервісів, які інкапсулюють HTTP-запити, ізолюючи їх від логіки компонентів.

3.3.3 Створення UX/UI дизайну користувацького інтерфейсу

Проєктування користувацького інтерфейсу (UI) та досвіду (UX) є ключовим для зручності та ефективності будь-якого веб-сервісу. У онлайн-сервісі Figma було створено прототипи інтерфейсу (див. рис. 3.5-3.10), базуючись на принципах:

а) Потреби кожного типу користувача: інтерфейс буде адаптований під функціональні потреби кожної категорії користувачів. Кожна роль отримуватиме лише ті функції та інформацію, що необхідні для її роботи;

б) Зручність та простота: інтерфейс розроблятиметься з акцентом на інтуїтивне сприйняття, повинен мати логічну структуру сторінок, зрозумілу навігацію та мінімальну кількість кроків для виконання звичних дій;

в) Єдиний стиль: для всіх сторінок та елементів інтерфейсу буде збережена стилістична цілісність, тобто однакова палітра кольорів, вигляд форм, кнопок тощо;

г) Наявність зворотнього зв’язку: користувач постійно отримуватиме чітке інформування про результати своїх дій. Наприклад, повідомлення про помилки чи успішне завершення якоїсь операції.

Першим елементом взаємодії користувача з системою є головна сторінка, на якій розміщується базова інформація про сервіс, кнопки для входу чи реєстрації, а також перемикачі для вибору мови інтерфейсу (див. рис. 3.5).

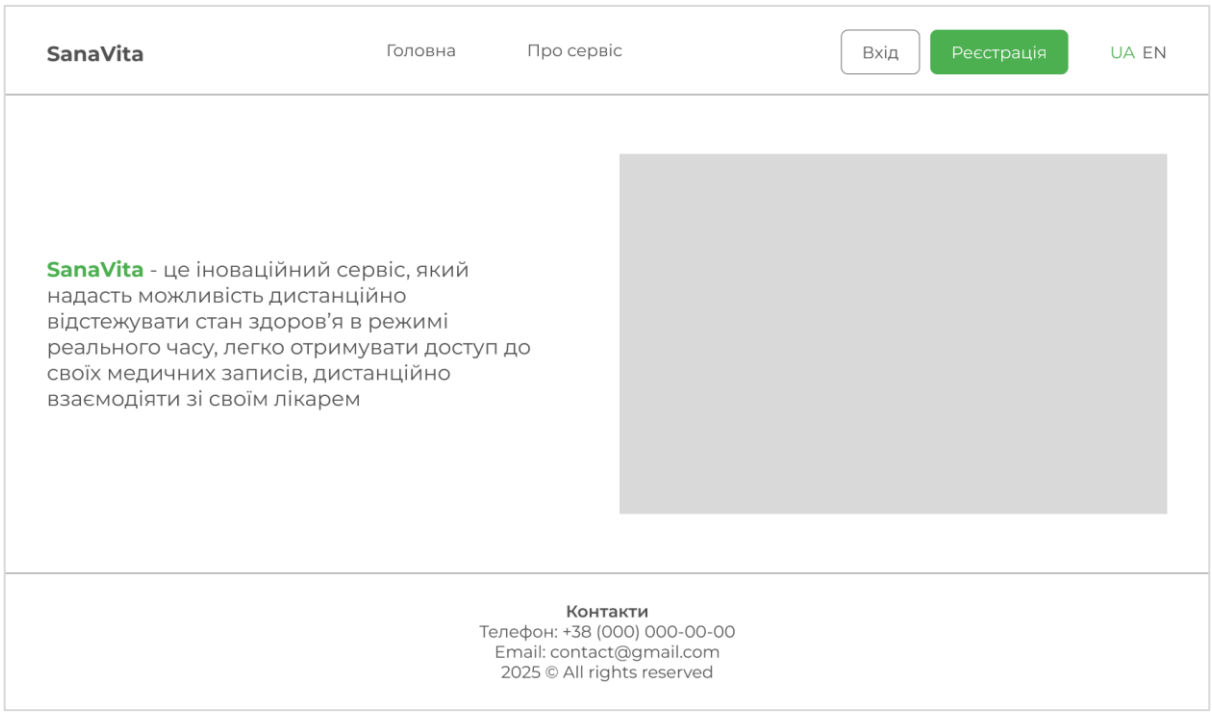

Рисунок 3.5 – Макет головної сторінки веб-сервісу

У якості наступного найбільш ймовірного кроку, користувач переходить на сторінку авторизації (див. рис. 3.6) або реєстрації (див. рис. 3.7), в залежності від своєї мети використання веб-сервісу.

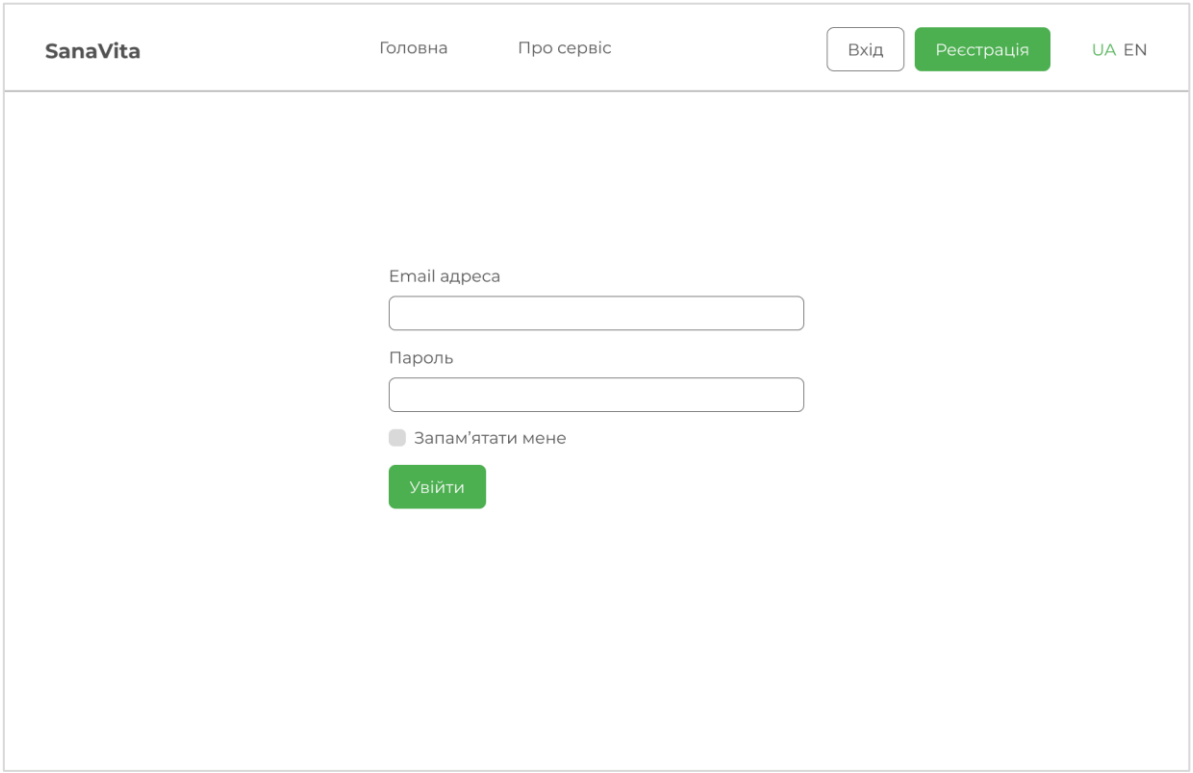

Рисунок 3.6 – Макет сторінки авторизації

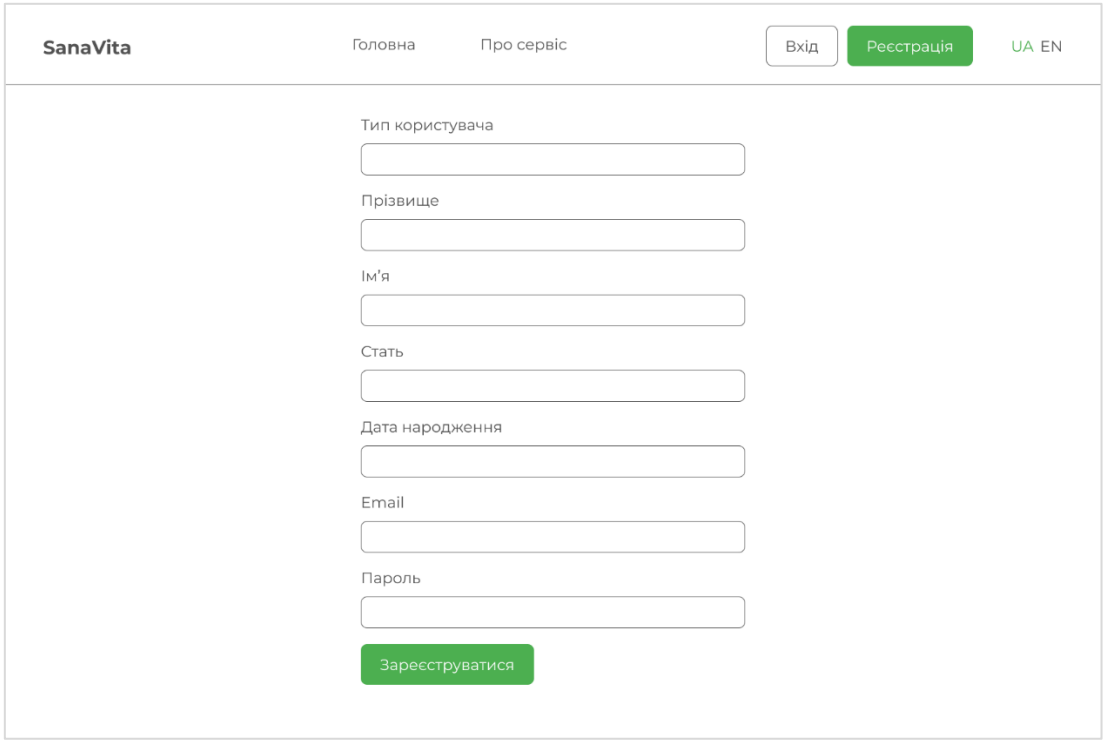

Рисунок 3.7 – Макет сторінки реєстрації

Після успішної авторизації перед користувачем з'являється сторінка, яка відповідає його ролі. Це може бути профіль пацієнта (див. рис. 3.8), інтерфейс лікаря (див. рис. 3.9) або панель адміністрування (див. рис. 3.10). Кожна зі сторінок оптимізована для виконання специфічних завдань певної ролі.

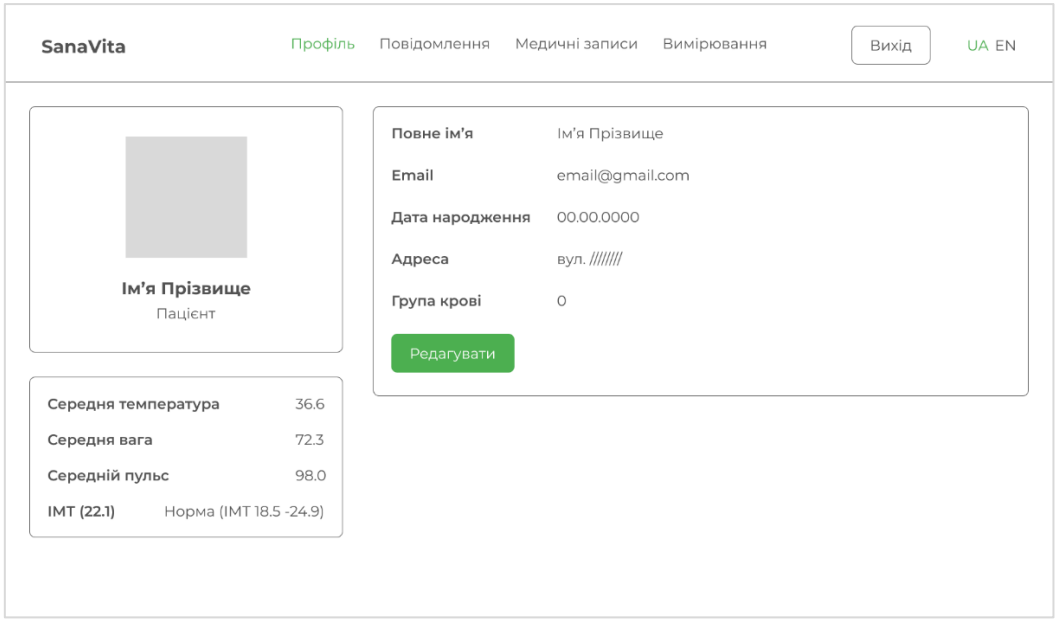

Рисунок 3.8 – Макет сторінки профілю та загального стилю інтерфейсу для пацієнта

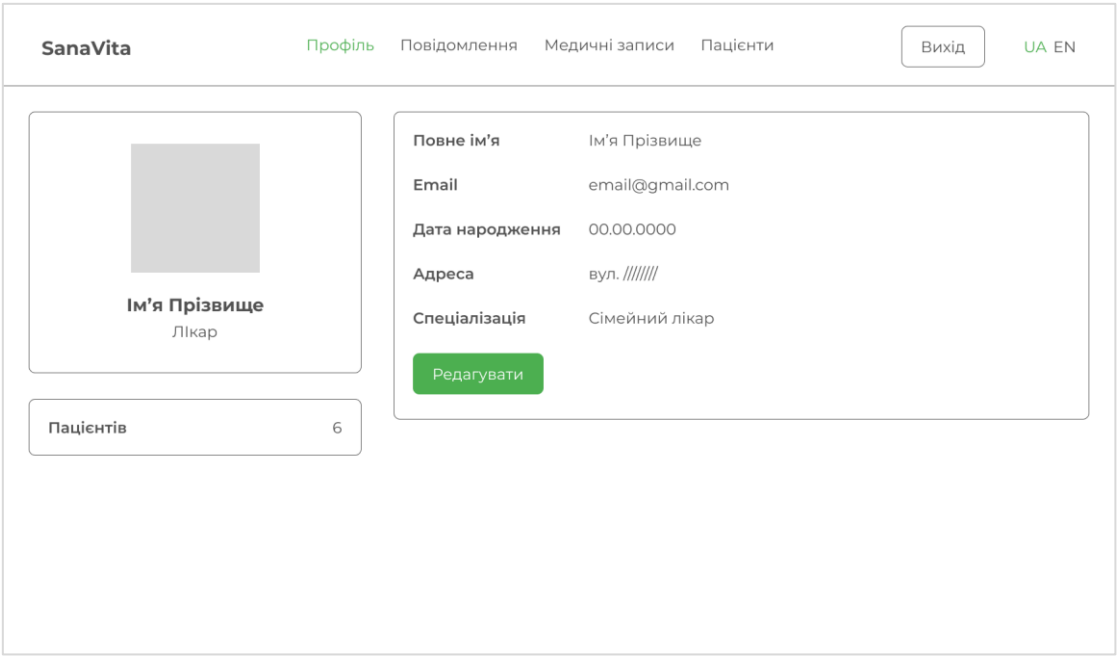

Рисунок 3.9 – Макет сторінки профілю та загального стилю інтерфейсу для лікаря

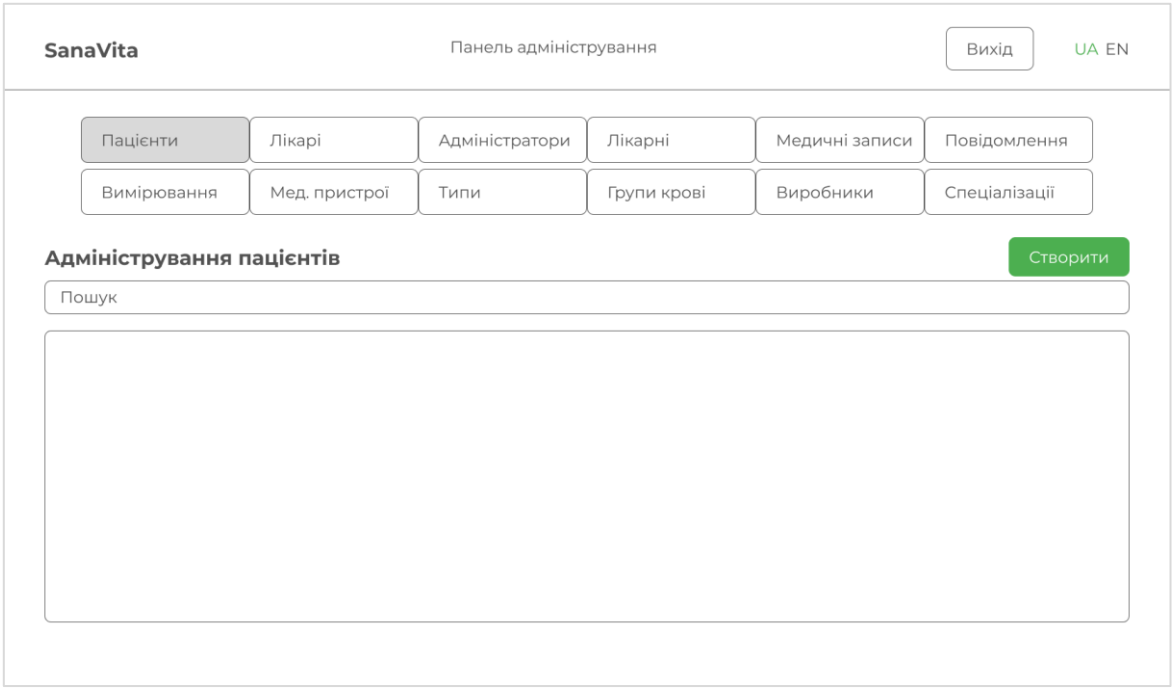

Рисунок 3.10 – Макет сторінки з панеллю для управління системою

Як можемо побачити, вхід у систему під обліковим записом адміністратора відкриває доступ до адміністративної панелі, яка досить сильно відрізняється від

інтерфейсу звичайних користувачів та дозволяє здійснювати повне керування всіма аспектами функціонування системи.

3.4 IoT пристрій

3.4.1 Проєктування функціональності ПЗ IoT пристрою

Аналогічно до проєктування інших компонентів системи, на першому етапі для програмного забезпечення IoT-пристрою було визначено ключові функції, які він повинен виконувати. Для наочного представлення цих функцій і способів взаємодії пристрою з навколишнім середовищем побудували діаграму прецедентів (див. рис. 3.11).

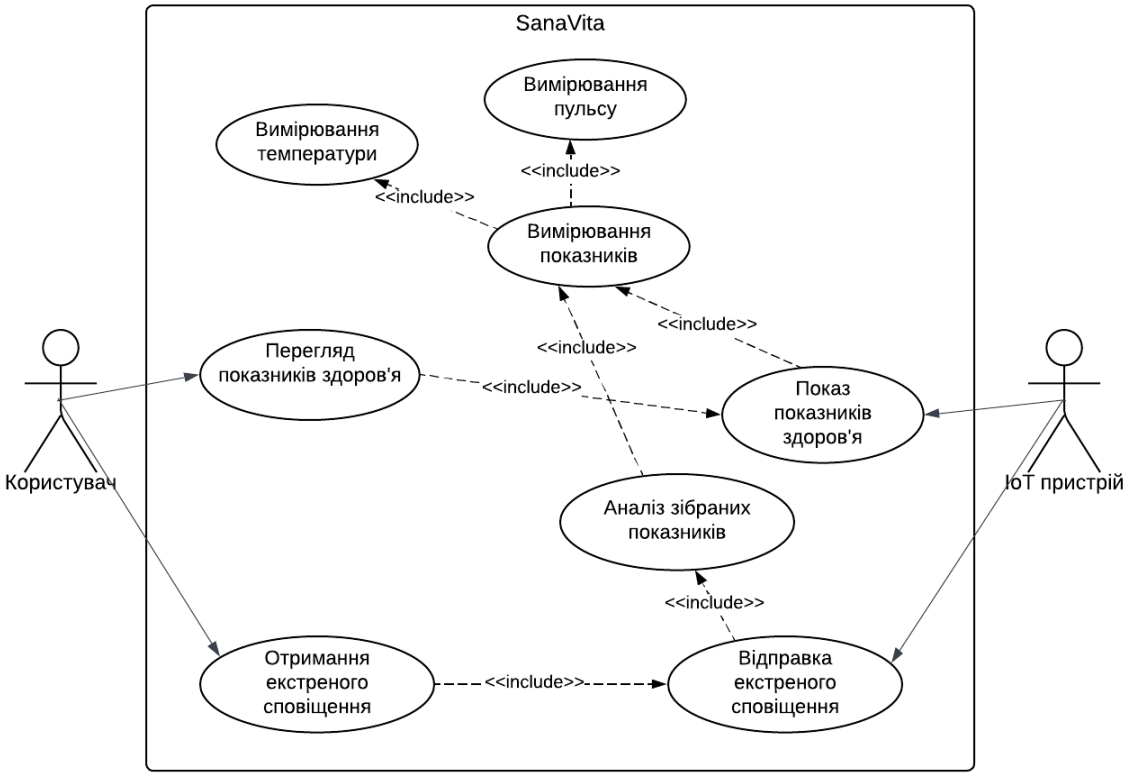

Рисунок 3.11 – Діаграма прецедентів для ПЗ IoT-пристрою

Діаграма демонструє ключові дії, що виконуються пристроєм або запускаються за його участю, а також показує взаємозв’язки з зовнішніми об’єктами. IoT-пристрій зчитує біометричні дані з певною періодичністю, виконує

їх первинну обробку, після чого надсилає результати до серверу. У реальному часі здійснюється перевірка отриманих значень на відповідність нормативним межам, і у разі виявлення критичних відхилень формується відповідне попередження. Пацієнт, у свою чергу, має доступ до актуальних біометричних показників та може побачити сповіщення, якщо якийсь з них вийде за допустимі межі.

3.4.2 Проєктування архітектури ПЗ IoT пристрою

Програмне забезпечення для IoT-пристрою створюється зі застосуванням модульної архітектури. Відповідальність за періодичне зчитування даних покладається на окремий вимірювальний модуль. Після отримання показників цей модуль виконує їх попередню обробку, перевіряючи дані на наявність можливих похибок та оцінюючи відповідність нормативним значенням. У разі виявлення відхилень, система автоматично генерує відповідне повідомлення. Для передачі та збереження оброблених показників використовується спеціалізована модель медичних показників, що визначає формат і структуру цих медичних параметрів.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Реалізація серверної частини

Серверна частина є невидимою для кінцевого користувача, але критично важливим ядром системи. Саме тут відбувається вся магія обробки даних, реалізується бізнес-логіка та забезпечується безпека. При її створенні ми керувалися принципами надійності, гнучкості та сучасними підходами до розробки програмного забезпечення, обравши для реалізації стек технологій .NET Core, що дозволяє створювати високопродуктивні та масштабовані веб-застосунки.

Для забезпечення ефективної комунікації з клієнтським додатком було обрано архітектурний стиль REST API. Такий підхід дозволяє розглядати всі ресурси системи як унікальні сутності, доступ до яких та маніпуляція якими здійснюється за допомогою стандартних HTTP-методів. Конкретно у даній системі це реалізовано наступними методами:

- GET використовується для отримання даних (наприклад, отримати конкретний медичний запис);
- POST слугує для створення нових ресурсів (наприклад, реєстрація нового користувача);
- PUT застосовується для повного оновлення вже існуючого ресурсу (наприклад, зміна даних у профілі пацієнта);
- DELETE відповідає за видалення ресурсу.

Така чітка відповідність HTTP-методів операціям CRUD робить API інтуїтивно зрозумілим та легким для інтеграції з різними клієнтами.

По-перше були створені моделі даних. Ці C#-класи є програмним представленням сутностей, визначених на етапі проєктування ER-діаграми бази даних. Вони описують структуру даних, з якими працює система. Щоб забезпечити комунікацію з базою даних для здійснення операцій та визначення зв’язків між моделями використовується ORM Entity Framework Core. Це дозволяє працювати з

даними як з об’єктами, а також легко визначати навігаційні властивості для зв’язків.

Приклад моделі медичного запису:

```
01 namespace stayhealth.Models
02 {
03     public class MedicalRecord
04     {
05         public int MedicalRecordId { get; set; }
06         public int PatientId { get; set; }
07         public Patient Patient { get; set; }
08         public DateTime VisitDate { get; set; } = DateTime.UtcNow;
09         public string Diagnosis { get; set; }
10         public string Medications { get; set; }
11         public int DoctorId { get; set; }
12         public Doctor Doctor { get; set; }
13     }
14 }
```

Можемо побачити, що рядки 06 та 07 відповідають за встановлення зв’язку з моделлю, яка описує сутність пацієнта.

Далі розглянемо інтерфейси, які описують набір методів, які повинні бути реалізовані для взаємодії з даними відповідної сутності (наприклад, отримати медичний запис, створити, видалити).

Приклад коду інтерфейсу медичного запису:

```
1 public interface IMedicalRecordRepository
2     {
3         MedicalRecord GetMedicalRecord(int mrecordId);
4         bool MedicalRecordExists(int mrecordId);
5         bool CreateMedicalRecord(MedicalRecord medicalRecord);
6         bool UpdateMedicalRecord(MedicalRecord medicalRecord);
7         bool DeleteMedicalRecord(MedicalRecord medicalRecord);
8         bool Save();
9     }
```

А сама реалізація цих інтерфейсів здійснена у відповідних класах-репозиторіях. Наприклад, для інтерфейсу медичного запису це реалізовано у MedicalRecordRepository. Саме ці класи інкапсулюють логіку безпосередньої роботи зі сховищем даних через Entity Framework, виконуючи запити на вибірку,

додавання, зміну чи видалення даних. Вони слугують абстракцією над сховищем даних.

Наприклад, код, який виводить всі медичні записи, які створив конкретний лікар для конкретного пацієнта:

```
1 public ICollection<MedicalRecord>
GetMedicalRecordsByPatientAndDoctor(int patientId, int doctorId)
2        {
3            return _context.MedicalRecords
4                .Where(e => (e.PatientId == patientId && e.DoctorId
== doctorId))
5                .ToList();
6        }
```

Такий шаровий підхід забезпечує хороший розподіл відповідальності, полегшує розуміння коду, його тестування та подальший розвиток системи.

Лікар також має функціонал, який дозволяє йому створювати записи на прийом. Після створення нового прийому пацієнт автоматично отримує сповіщення на електронну пошту. Це забезпечує оперативне інформування пацієнтів про майбутню консультацію, її дату, час та лікаря, до якого здійснено запис. Для реалізації цієї функції використовується сервіс EmailService, який через SMTP надсилає лист на вказану адресу. Нижче наведено метод SendAppointmentEmail, який відповідає за формування і надсилання повідомлення:

```
01 public async Task SendAppointmentEmail(string toEmail, string
patientName, DateTime dateTime, string doctorFullName, string notes)
02 {
03     string senderEmail = _configuration["EmailSettings:Sender"];
04     string password = _configuration["EmailSettings:Password"];
05
06     var message = new MailMessage();
07     message.To.Add(toEmail);
08     message.Subject = "Запис до лікаря";
09
10     string body = $"Шановний(а) {patientName},\n\n" +
11            $"Вас записано на прийом до лікаря: {doctorFullName}.\n"
+
12           $"Дата та час прийому: {dateTime:dd.MM.yyyy HH:mm}.\n\n";
13
14     if (!string.IsNullOrWhiteSpace(notes))
15     {
16         body += $"Нотатки від лікаря:\n{notes}\n\n";
```

```
17      }
18      body += "З повагою,\nСервіс SanaVita";
19
20      message.Body = body;
21      message.From = new MailAddress(senderEmail);
22
23      using var smtp = new SmtpClient("smtp.gmail.com", 587)
24      {
25          Credentials = new NetworkCredential(senderEmail, password),
26          EnableSsl = true
27      };
28
29      await smtp.SendMailAsync(message);
30  }
```

Цей функціонал підвищує зручність використання системи та зменшує ймовірність пропущених прийомів з боку пацієнтів.

4.2 Реалізація клієнтської частини

Клієнтська частина системи, розроблена на базі фреймворку Angular, є тим інтерактивним середовищем, через яке користувачі безпосередньо взаємодіють з усіма можливостями платформи. Основний акцент при її реалізації було зроблено на створенні інтуїтивно зрозумілого та функціонального веб-додатку, здатного забезпечити комфортну роботу з медичними даними.

Для забезпечення доступності використання веб-сервісу для ширшого кола користувачів, у нашій системі наявні українська (як стандартна) та англійська мови. Підтримку декількох мовних локалей було реалізовано через вбудовану бібліотеку @ngx-translate/core, яка дозволяє гнучко керувати перекладами текстових рядків інтерфейсу.

При першому відкритті сайту або при відсутності збережених налаштувань, система автоматично встановлює українську мову. Однак, користувач має можливість обрати бажану мову, і цей вибір зберігається у кеші браузера. Тому при наступних візитах додаток зчитує це значення та підвантажує обрану раніше мову.

Код, який реалізує цей функціонал:

```
01    constructor(public authService: AuthService, private translate:
TranslateService) {
02      const savedLanguage = localStorage.getItem('language') || 'ua';
03      this.translate.setDefaultLang(savedLanguage);
04      this.translate.use(savedLanguage);
05    }
06
07    switchLanguage(language: string) {
08      this.translate.use(language);
09      localStorage.setItem('language', language);
10    }
```

Самі переклади всіх текстових рядків для кожної мови зберігаються в окремих JSON-файлах (assets/i18n/ua.json та assets/i18n/en.json), що полегшує їх редагування та додавання нових мов у майбутньому. Для коректного відображення дат та часу, їх адаптації під кожного конкретного користувача, використовується вбудований в Angular DatePipe, який автоматично форматує їх відповідно до поточної обраної локалі користувача.

Забезпечення коректності даних, що вводяться користувачами, є важливим аспектом роботи будь-якої системи. У даному веб-сервісі реалізована клієнтська валідація форм, яка дозволяє надавати миттєвий зворотний зв’язок користувачеві та зменшити кількість некоректних запитів до сервера. Для цього використовуються як вбудовані засоби Angular для валідації форм, так і стандартні HTML5 атрибути (required, коли поле є обов’язковим для заповнення, maxlength, minlength). Якщо користувач вводить дані, що не відповідають встановленим правилам (див. рис. 4.1), або залишає обов’язкове поле порожнім (див. рис. 4.2), система відображає повідомлення про конкретну помилку.

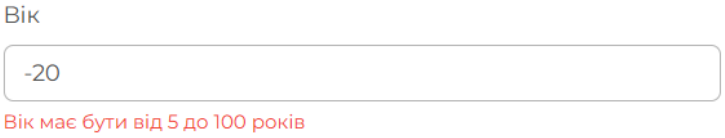

Рисунок 4.1 – Сторінка реєстрації, повідомлення про необхідність вести коректні дані

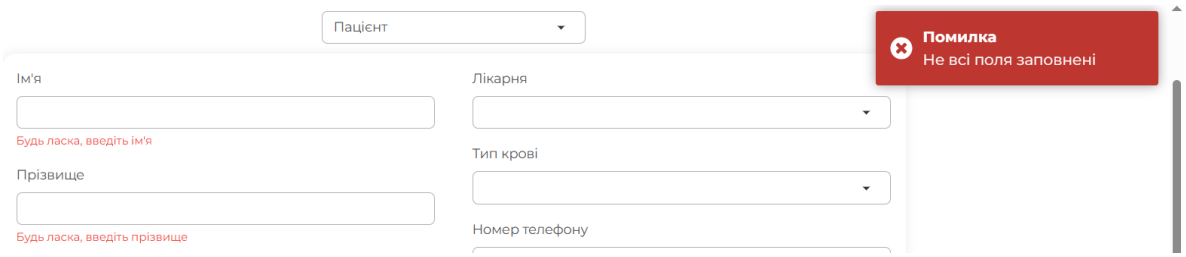

Рисунок 4.2 – Сторінка реєстрації, повідомлення про необхідність заповнити всі обов'язкові поля

Також при виконанні реєстрації для перевірки збігу пароля та його підтвердження використовується логіка на рівні компонента, яка порівнює значення відповідних полів форми:

```
01 <div class="form-group mb-4">
02     <label for="passwordInput">{{ 'FORM.PASSWORD' | translate }}</label>
03     <input #password="ngModel"
04            name="password"
05            [(ngModel)]="registrationData.password"
06            type="password"
07            id="passwordInput"
08            class="form-control"
09            required
10            minlength="8" />
11     <div *ngIf="password.invalid && password.touched" class="text-danger">{{ 'FORM.PASSWORD_MIN_LENGTH' | translate }}</div>
12   </div>
```

Функціонал обміну повідомленнями реалізований у вигляді стандартних чатів, де лікарі та пацієнти можуть надсилати повідомлення один одному. Повідомлення зберігаються у спільній таблиці Notifications і мають поля, що вказують на відправника, отримувача, тип користувача (лікар або пацієнт), вміст повідомлення та час відправлення. Виводяться повідомлення у вікно чату та розміщуються відносно до ролі відправника/отримувача, класи from-doctor та from-patient використовуються для стилізації повідомлень залежно від того, хто є їхнім відправником:

```
01 <div class="chat-window" *ngIf="messages">
02     <div *ngFor="let msg of messages; let isLast = last"
03         class="message"
04         [ngClass]="{
05           'from-doctor': msg.senderType === 0,
06           'from-patient': msg.senderType === 1
07         }">
08       <div class="message-content">
09         <div>{{ msg.content }}</div>
10           <div class="message-time">{{ msg.date | date:'dd.MM.yyyy
HH:mm' }}</div>
11       </div>
12
13       <div *ngIf="isLast" #lastMessage></div>
14     </div>
15   </div>
```

На TypeScript-стороні реалізовано завантаження повідомлень, які фільтруються за айді відправника/отримувача та типами користувачів. Цей метод забезпечує двосторонню фільтрацію повідомлень, щоб відображати тільки ті, які були надіслані між конкретним лікарем і пацієнтом. Повідомлення також сортуються за датою та часом для правильного порядку виводу в інтерфейсі чату:

```
01 loadMessages() {
02
this.http.get<Notification[]>(`${this.url}/Notifications`).subscribe((allNo
tifications: Notification[]) => {
03     this.messages = allNotifications.filter(m =>
04       (m.senderId === this.patient.patientId &&
Number(m.senderType) === UserType.Patient &&
05       m.receiverId === this.doctorId && Number(m.receiverType) ===
UserType.Doctor) ||
06       (m.senderId === this.doctorId && Number(m.senderType) ===
UserType.Doctor &&
07       m.receiverId === this.patient.patientId &&
Number(m.receiverType) === UserType.Patient)
08     ).sort((a, b) => new Date(a.date).getTime() - new
Date(b.date).getTime());
09   });
10 }
```

Також у застосунку реалізовано функціонал підрахунку статистичних показників здоров’я для кожного пацієнта, що потім використовується для оформлення візуалізації статистики пацієнта. Для цього в компоненті виконується обробка отриманих вимірювань, підраховуються середні, мінімальні та

максимальні значення температури, ваги та пульсу. Значення норми для температури та пульсу встановлені стандартні, а норма ваги розраховується для кожного індивідуально:

```
1 this.stats.tempNormal = 36.6;
2 this.stats.weightNormal = (this.user.height - 100) * 0.875
3 this.stats.pulseNormal = 75;
```

Окрім цього, для кожного користувача індивідуально розраховується індекс маси тіла за формулою:

```
1 let heightM: number = this.user.height / 100;
2 let BMI: number = this.stats.weight / (heightM * heightM);
```

Як було вже вказано раніше, клієнтська частина отримує інформацію з бекенду через HTTP-запити. Ця взаємодія реалізована за допомогою вбудованого в Angular модуля HttpClient. Приклад реалізації на прикладі отримання та обробки всіх вимірювань конкретного пацієнта:

```
01 getPatientAndMeasurements() {
02     this.http.get(this.url + "/Patients/email/" +
this.userInfo.email).pipe(
03       switchMap((patient: any) => {
04         this.patient = patient;
05         return this.http.get(this.url + "/Measurements/patient/" +
this.patient.patientId);
06       })
07     ).subscribe((measurements: any) => {
08       this.measurements = measurements;
09       for (let i = 0; i < this.measurements.length; i++) {
10
this.getMedicalDeviceName(measurements[i].medicalDevice.medicalDeviceTypeId
).subscribe(name => {
11           this.measurements[i].deviceName = name;
12         });
13       }
14     });
15   }
```

Спочатку запитується інформація про пацієнта за email користувача (рядок 02), потім медичні вимірювання цього пацієнта (рядки 03-06). Для кожного запису додатково завантажується назва пристрою за його типом (рядки 09-13).

4.2.1 Інтерфейс користувача

Інтерфейс користувача було реалізовано на основі прототипів, розроблених у Figma на етапі проєктування (див. розділ 3.3.3). Кожен екран та елемент інтерфейсу є реалізацією одного або декількох Angular компонентів, що забезпечує модульність та гнучкість.

При першому запуску веб-клієнту користувач потрапляє на головну сторінку (див. рис. 4.3-4.4). Ця сторінка містить опис веб-сервісу, його завдання, а також надає чіткі заклики до дії: кнопки для переходу до сторінок авторизації або реєстрації. Тут же розташовані елементи для зміни мови.

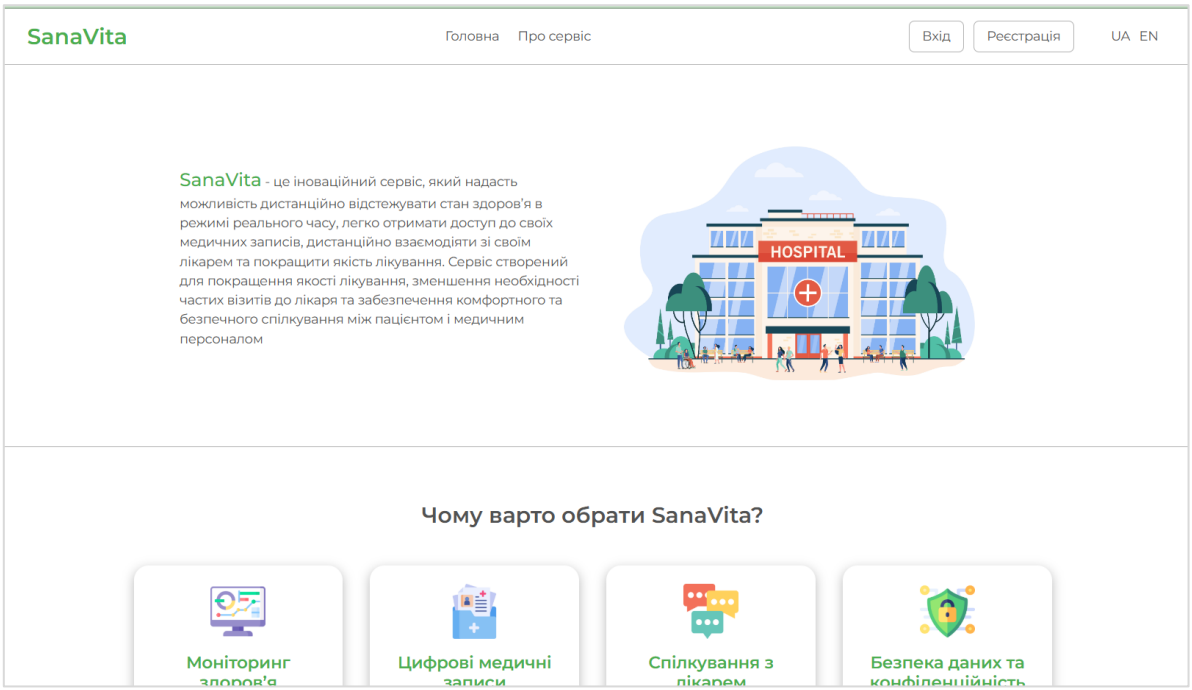

Рисунок 4.3 – Головна сторінка веб-сервісу

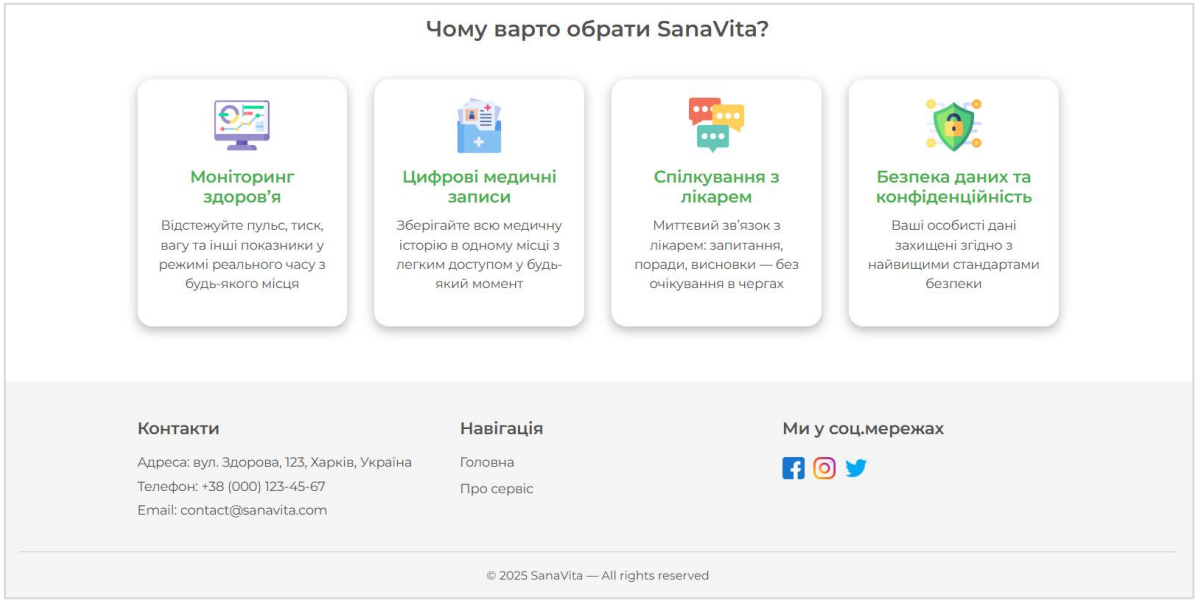

Рисунок 4.4 – Продовження головної сторінки веб-сервісу

Також сайт є адаптивним до планшетів та мобільних пристроїв (див. рис. 4.5-4.6).

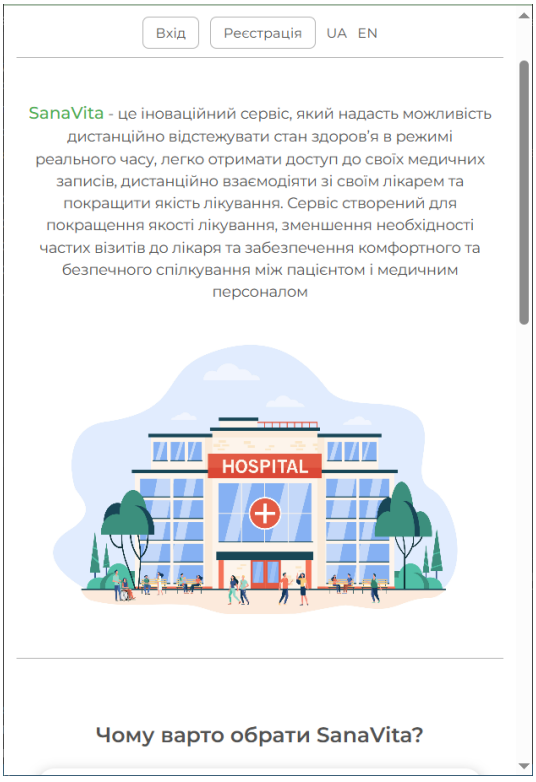

Рисунок 4.5 – Вигляд головної сторінки на мобільних пристроях

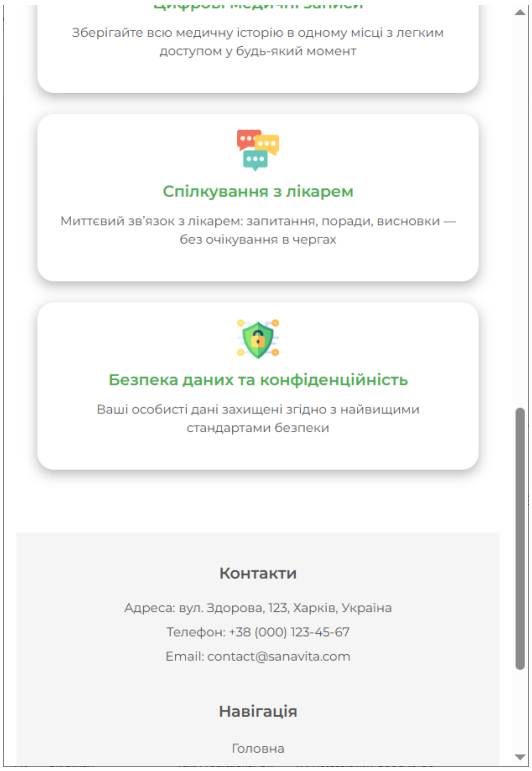

Рисунок 4.6 – Продовження головної сторінки на мобільних пристроях

Користувач, обравши відповідну дію, бачить сторінку авторизації (див. рис. 4.7) для входу в існуючий обліковий запис, або сторінку реєстрації (див. рис. 4.8) для створення нового.

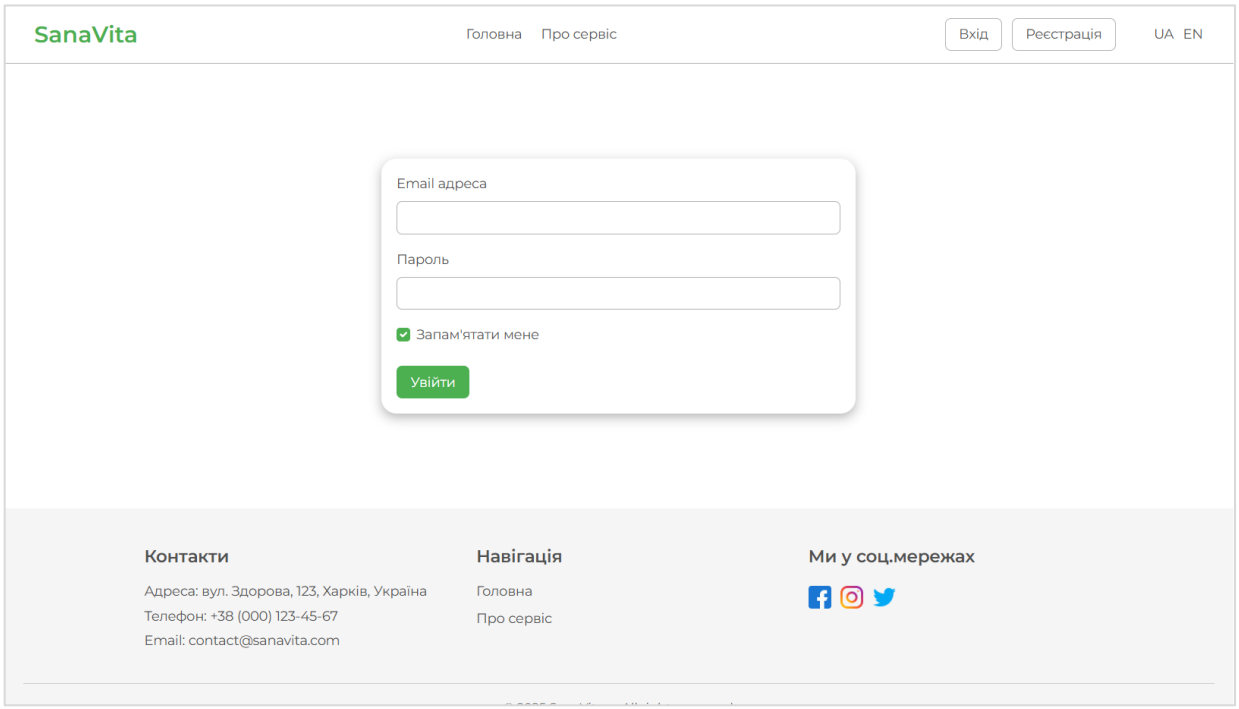

Рисунок 4.7 – Сторінка авторизації

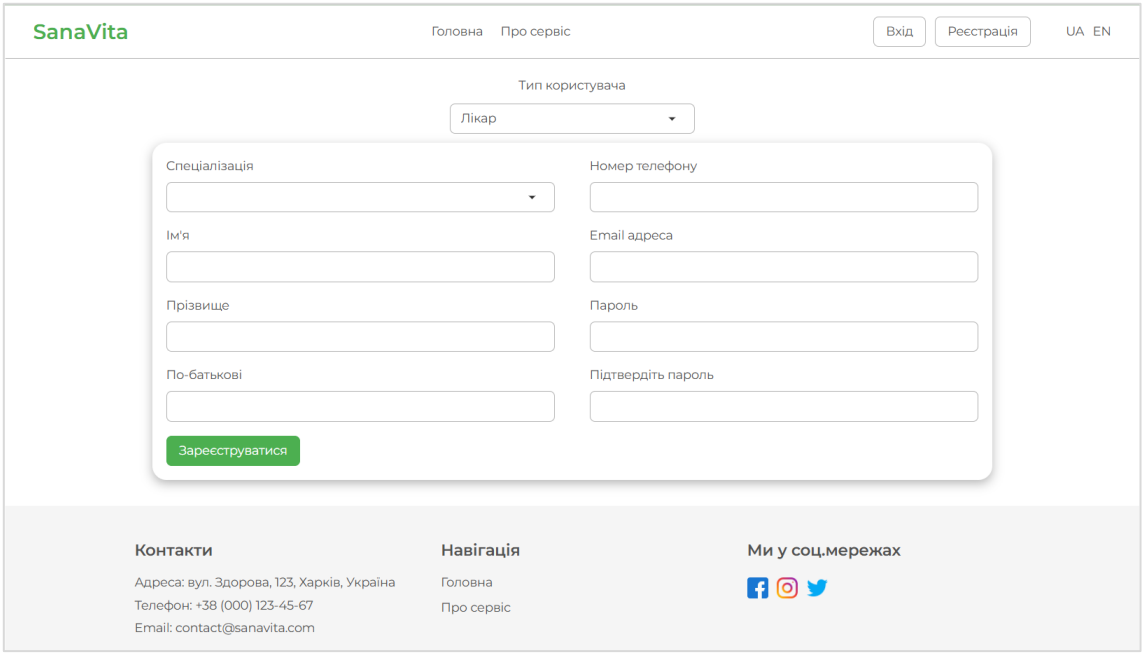

Рисунок 4.8 – Сторінка реєстрації

Після успішної авторизації користувач спрямовується до свого персонального робочого простору, вигляд та функціональність якого залежать від його ролі в системі. Так, пацієнт бачить свій профіль (див. рис. 4.9) з можливістю перегляду медичних записів (див. рис. 4.10), своїх показників (див. рис. 4.11), чатів з лікарями (див. рис. 4.12), записів на прийоми (див. рис. 4.13), та редагування особистої інформації. Сторінки також є адаптивними (див. рис. 4.15-4.16).

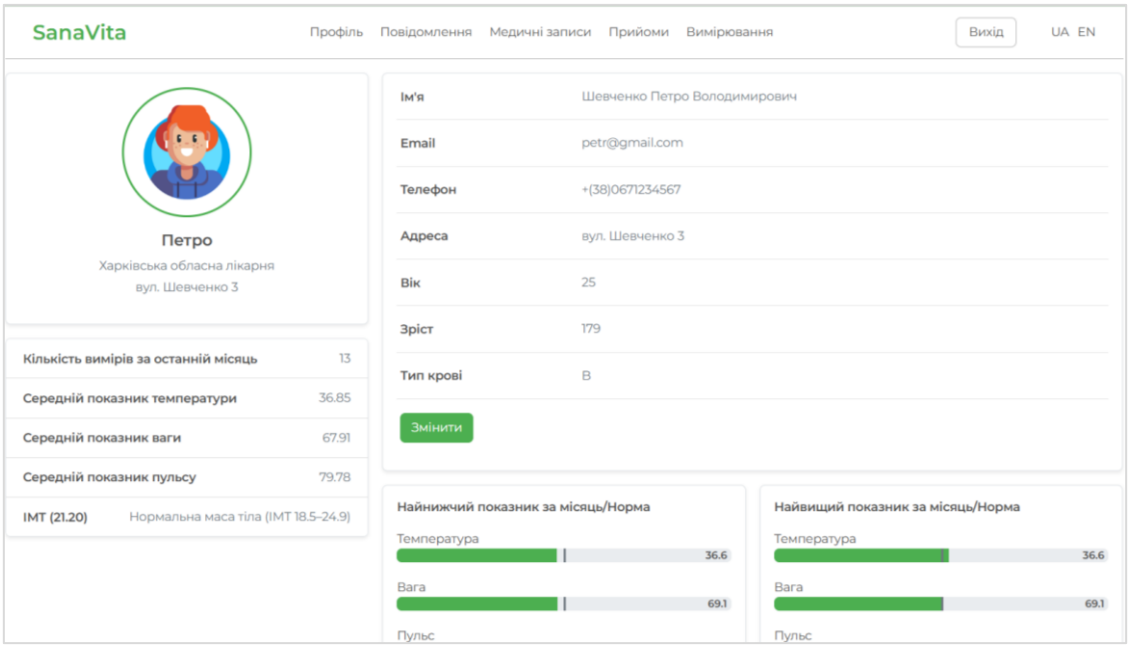

Рисунок 4.9 – Сторінка профілю користувача у ролі пацієнта

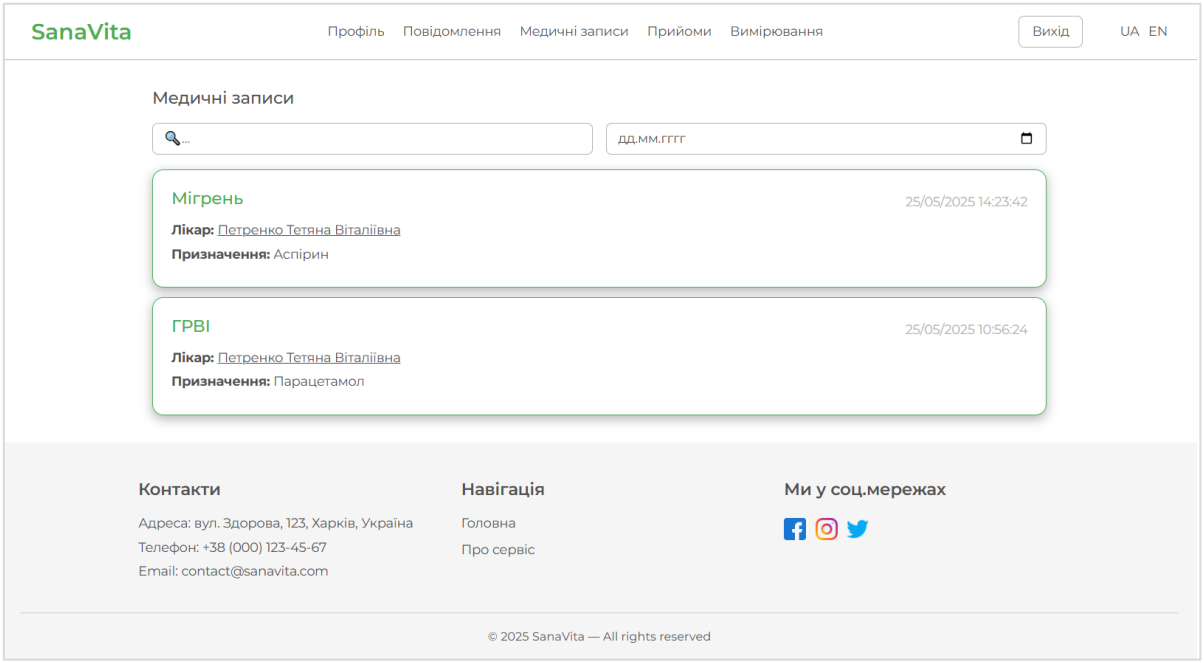

Рисунок 4.10 – Сторінка з медичними записами пацієнта

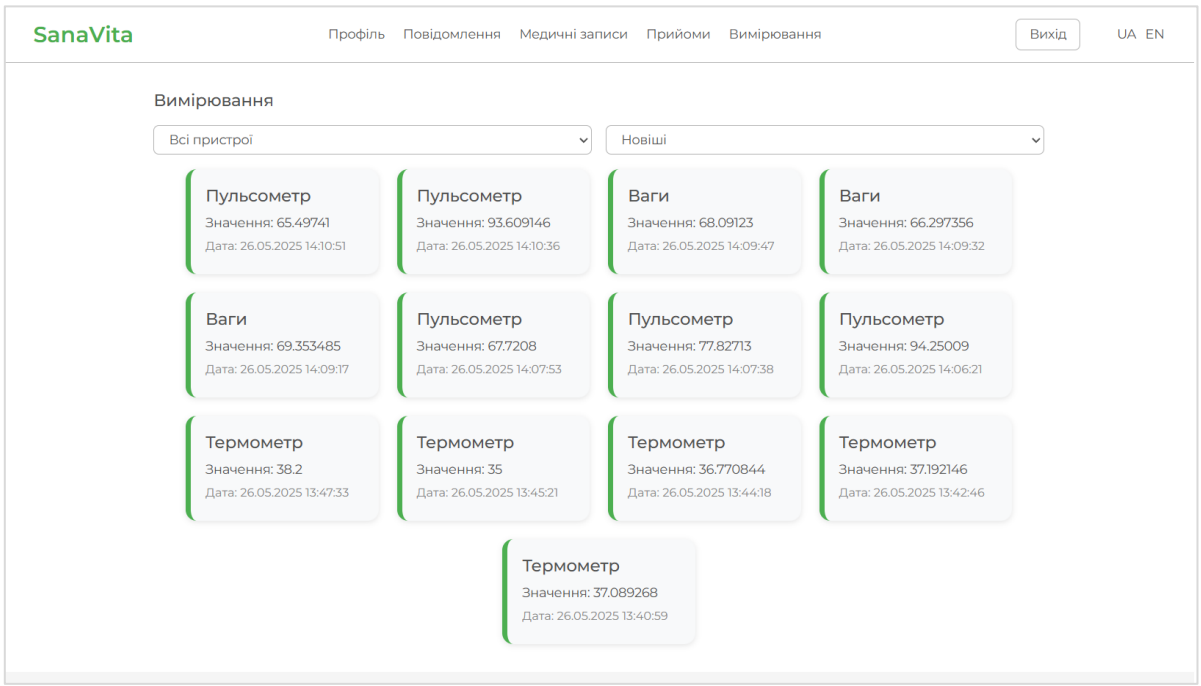

Рисунок 4.11 – Сторінка зі зібраними показниками пацієнта

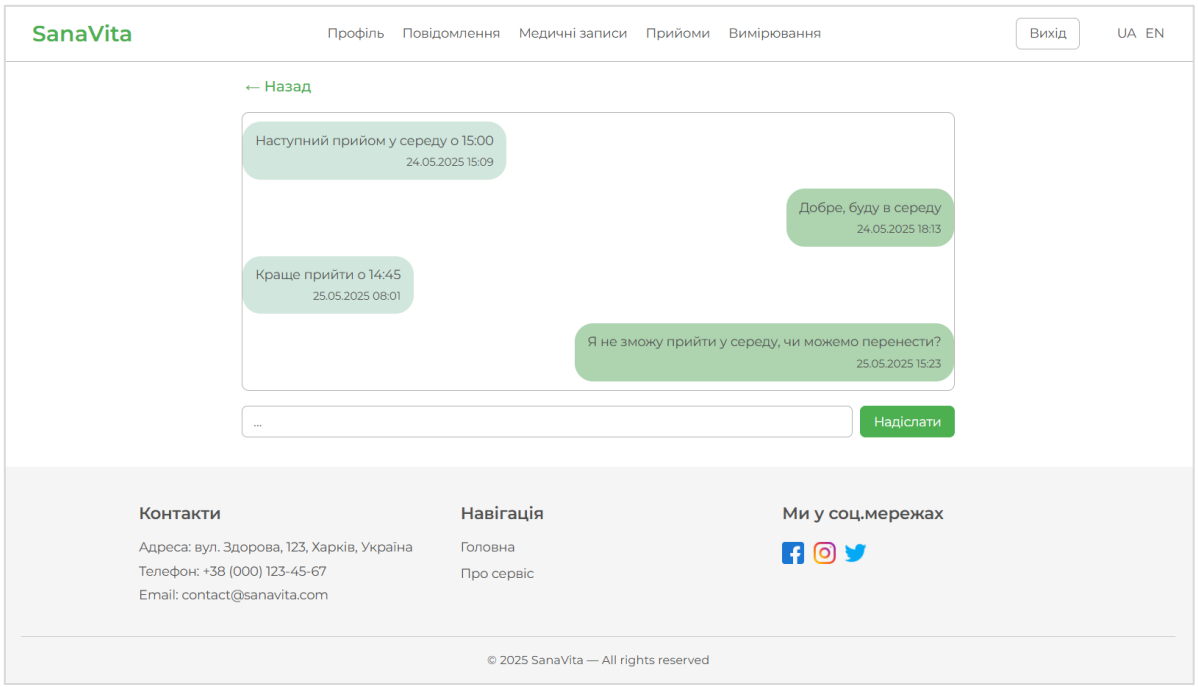

Рисунок 4.12 – Сторінка чату з лікарем

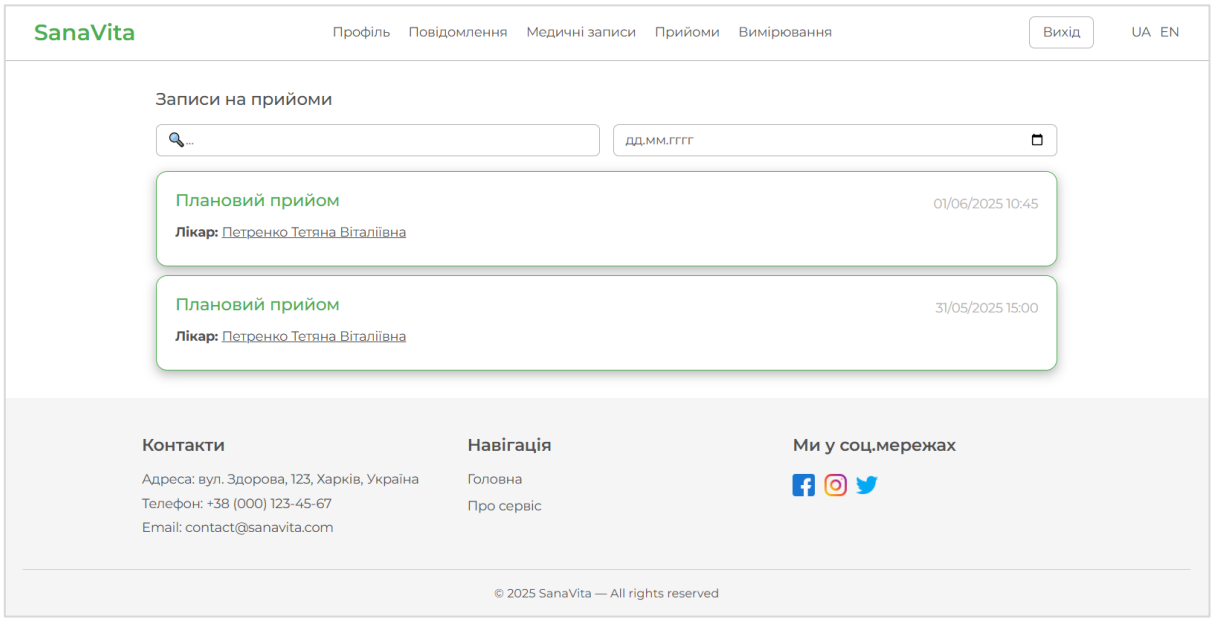

Рисунок 4.13 – Сторінка записів на прийоми

Після того як лікар створює новий прийом, пацієнт на свою електрону пошту отримує листа з усією інформацією про нього (див. рис. 4.14).

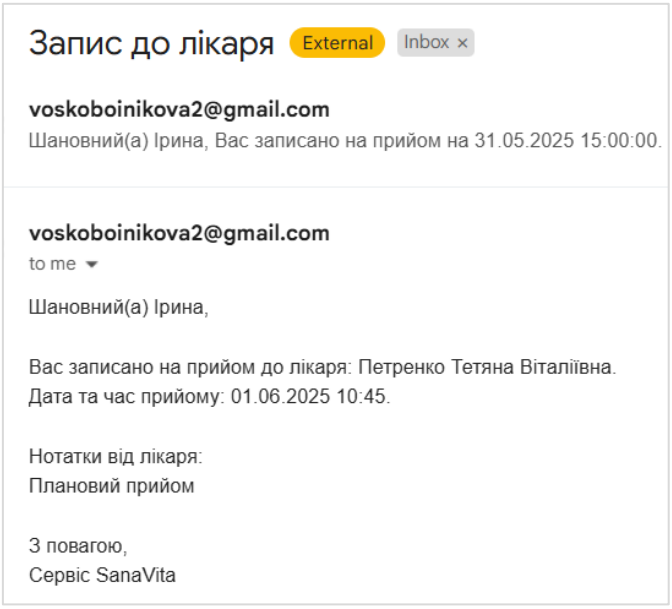

Рисунок 4.14 – Лист про новий запис на прийом до лікаря

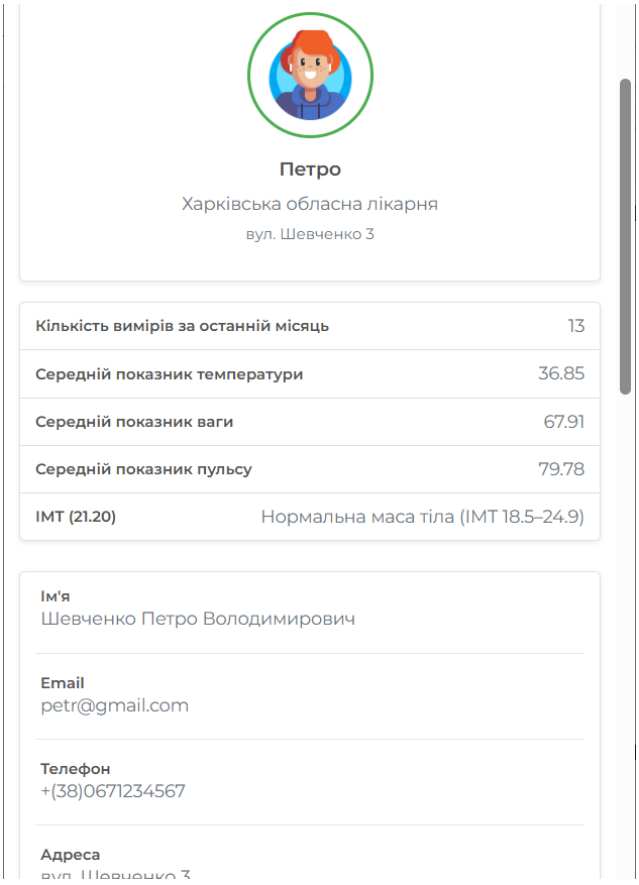

Рисунок 4.15 – Профіль пацієнта на мобільних пристроях

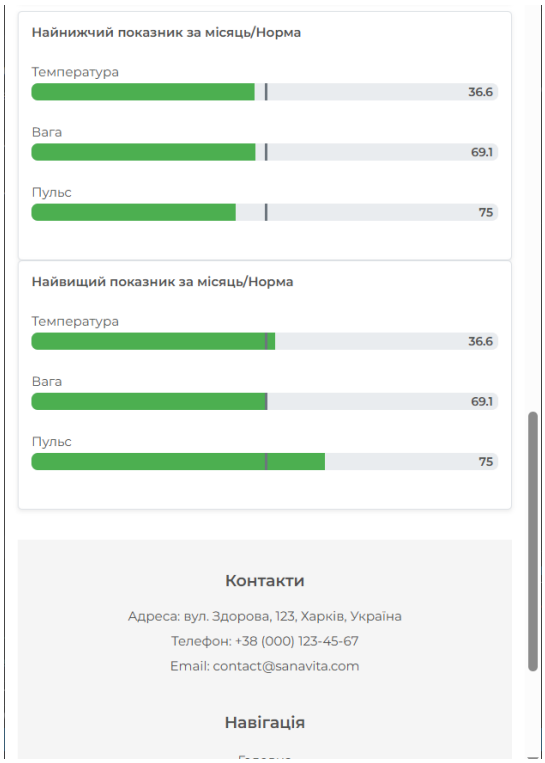

Рисунок 4.16 – Продовження профілю пацієнта на мобільних пристроях

Лікар отримує доступ до інтерфейсу (див. рис. 4.17-4.18), орієнтованого на управління пацієнтами: перегляд списку своїх пацієнтів, доступ до їхніх медичних карт (див. рис. 4.19), створення записів на прийоми (див. рис. 4.20), створення призначень (див. рис. 4.21) та обмін повідомленнями з ними (див. рис. 4.22).

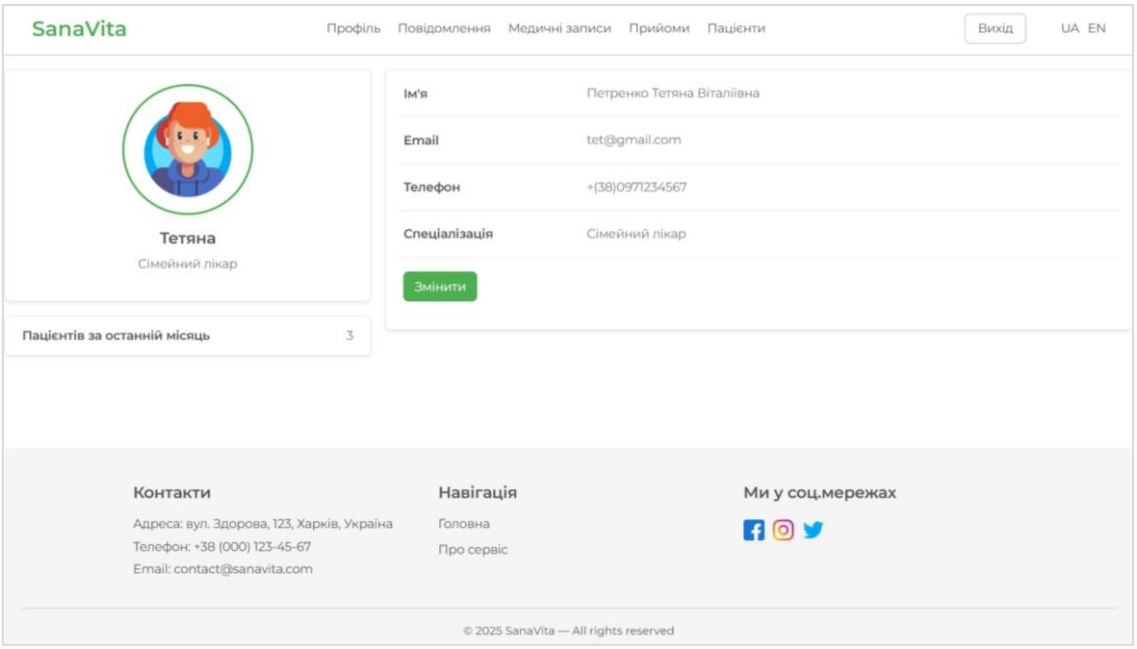

Рисунок 4.17 – Сторінка профілю користувача у ролі лікаря

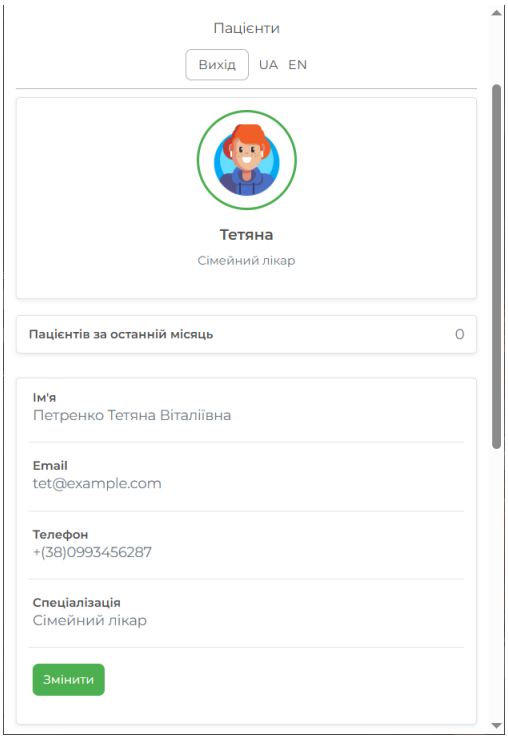

Рисунок 4.18 – Сторінка профілю користувача у ролі лікаря на мобільних пристроях

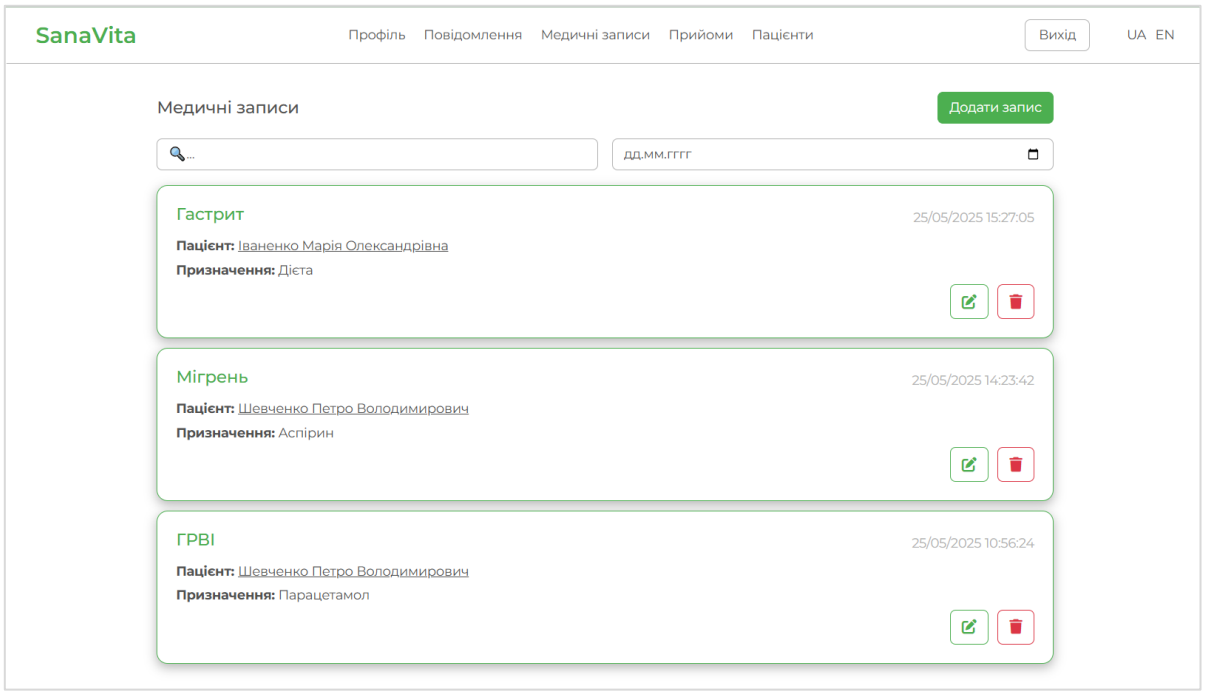

Рисунок 4.19 – Сторінка перегляду своїх медичних записів до пацієнтів

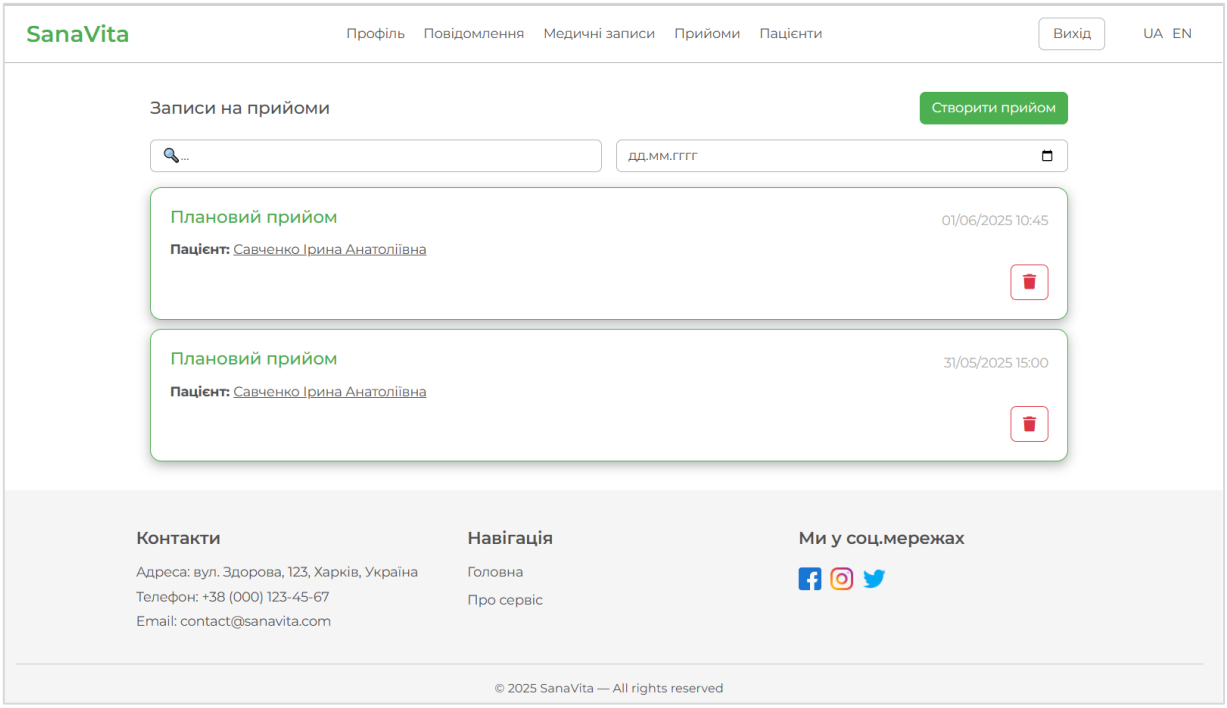

Рисунок 4.20 – Сторінка з записами на прийоми

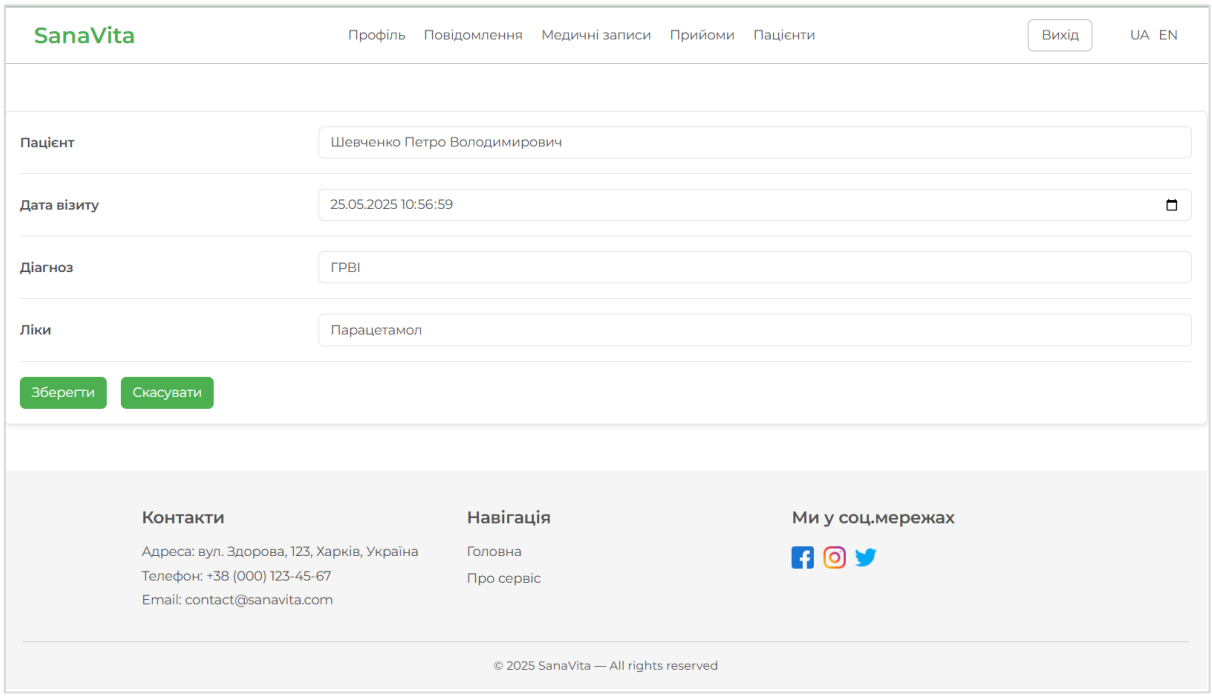

Рисунок 4.21 – Сторінка додавання нового медичного запису

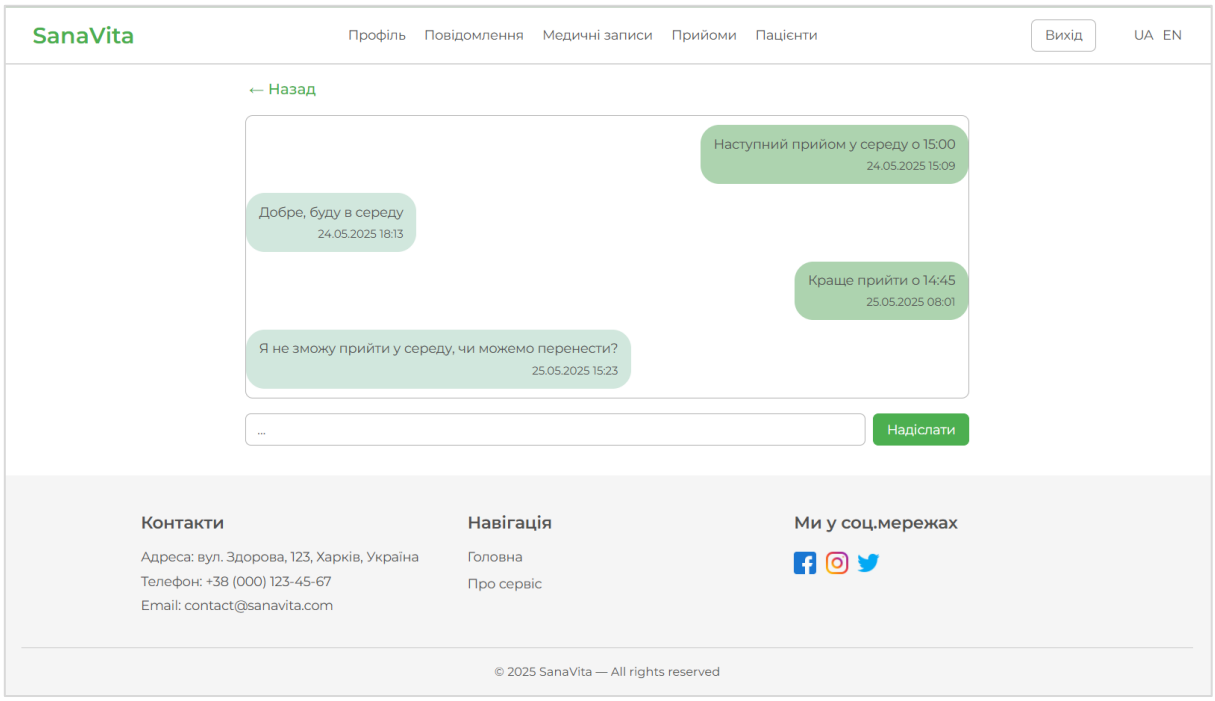

Рисунок 4.22 – Сторінка чату з пацієнтом

Користувач з роллю адміністратора після входу бачить спеціалізовану панель (див. рис. 4.23), з якої він може виконувати управління БД (див. рис. 4.24-4.25). В залежності від пристрою, сторінка також адаптується (див. рис. 4.26-4.27).

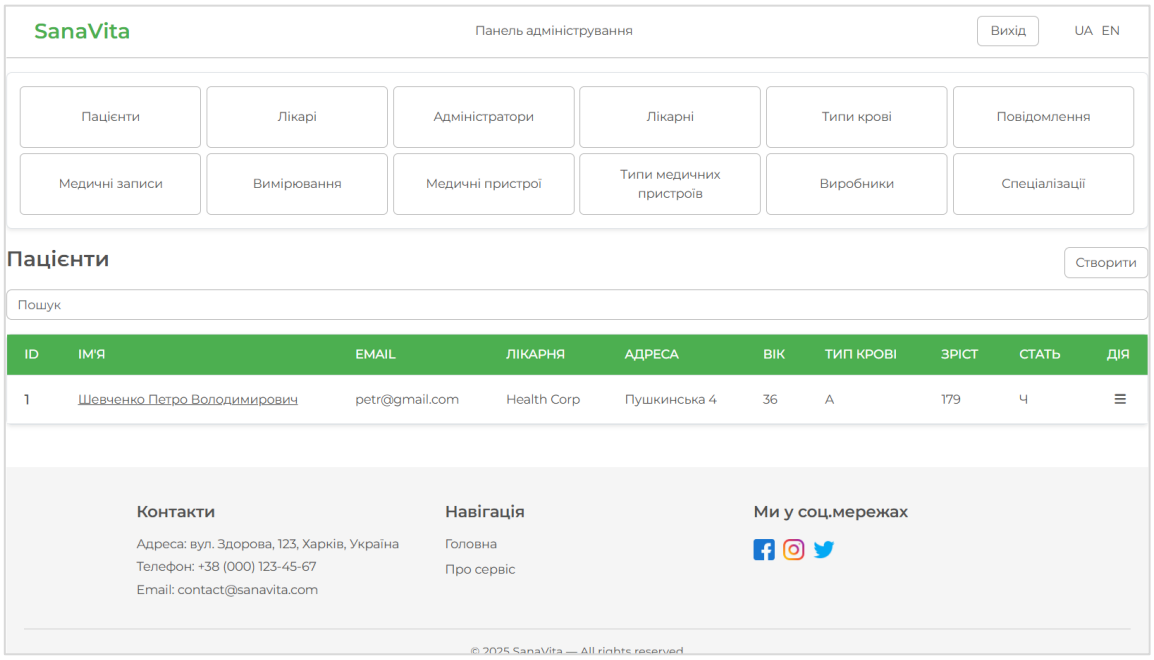

Рисунок 4.23 – Сторінка інтерфейсу для користувача у ролі адміністратора

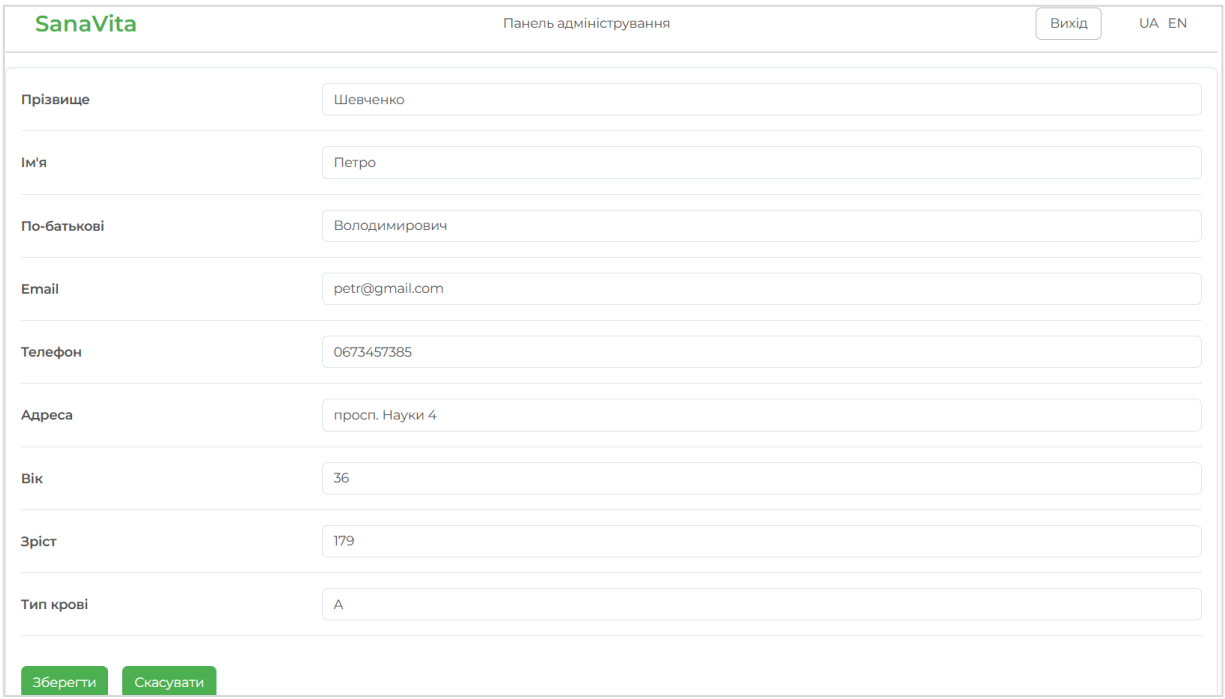

Рисунок 4.24 – Сторінка редагування інформації

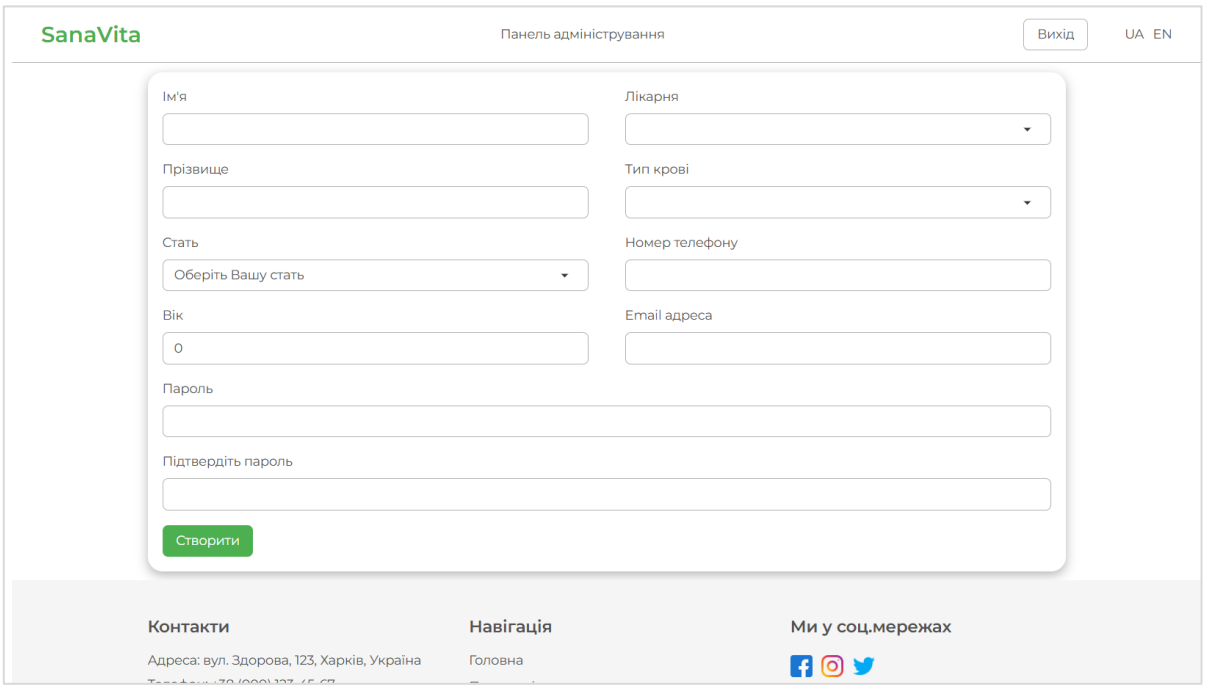

Рисунок 4.25 – Сторінка створення нового запису користувача

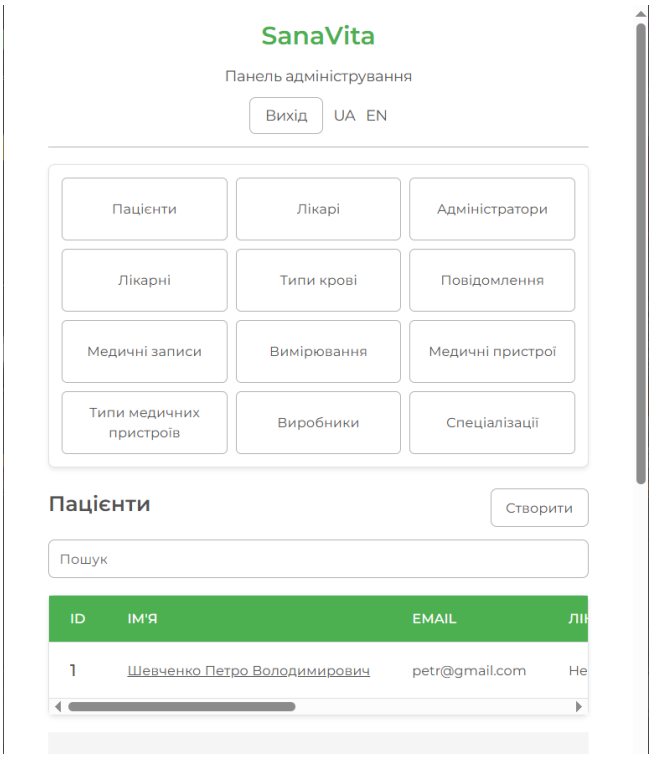

Рисунок 4.26 – Панель адміністрування на планшетах

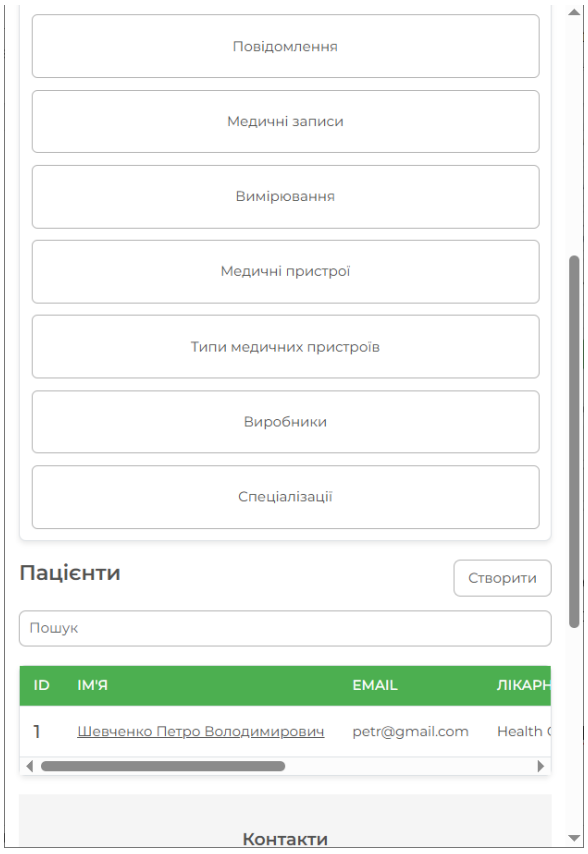

Рисунок 4.27 – Панель адміністрування на мобільних пристроях

Ця панель надає інструменти для управління користувачами та прямого доступу до управління даними, які зберігаються в базі даних, налаштування параметрів платформи.

4.3 Реалізація ПЗ для IoT-пристрою

В основі обробки даних у ПЗ для IoT-пристрою лежить модель Measurement:

```
01 namespace StayHealthIoT.Models
02 {
03     public class Measurement
04     {
05         public int MeasurementId { get; set; }
06         public int PatientId { get; set; }
07         public float MeasurementValue { get; set; }
08         public int MedicalDeviceId { get; set; }
09         public DateTime MeasurementDate { get; set; } =
DateTime.UtcNow;
10     }
11 }
```

Цей клас інкапсулює всю необхідну інформацію про одне конкретне зняття біометричного показника: ідентифікатор пацієнта, значення вимірювання, ідентифікатор медичного пристрою, а також точну дату й час вимірювання. Для забезпечення універсальності та уникнення проблем з часовими зонами, всі позначки часу фіксуються та зберігаються у форматі UTC.

Функціонал аналізу та виведення статистики по показникам реалізовано у окремому контролері. Наприклад, підрахунок середньої температури по всім попереднім вимірюванням користувача:

```
01 [HttpGet("average-temperature")]
02         public async Task<IActionResult> GetAverageTemperature(int
patientId, int medicalDeviceId)
03         {
04             var measurements = await _context.Measurements
05                 .Where(m => m.PatientId == patientId &&
m.MedicalDeviceId == medicalDeviceId)
06                 .ToListAsync();
07
```

```
08              if (measurements == null || !measurements.Any())
09              {
10                  return NotFound("No measurements found for the
given patient and device.");
11              }
12
13              var averageTemperature = measurements.Average(m =>
m.MeasurementValue);
14
15              return Ok(new { AverageTemperature =
averageTemperature });
16          }
```

Функціонал зчитування фізіологічного показника, наприклад, температури, реалізовано у окремому сервісі:

```
01 var measurement = new Measurement
02              {
03                  PatientId = Program.PatientId,
04                  MeasurementValue = (float)temperature,
05                  MedicalDeviceId = Program.MedicalDeviceId,
06                  MeasurementDate = DateTime.UtcNow
07              };
08
09              context.Measurements.Add(measurement);
10              context.SaveChanges();
11
12              if (temperature < MinTemp || temperature > MaxTemp)
13              {
14
Console.WriteLine(Program.LocalizationResources["TemperatureOutOfRange"],
temperature);
15              }
```

Можемо побачити, що після зчитування відбувається відправка нового запису до бази даних. Далі виконується аналіз на відповідність прийнятному діапазону, і у випадку виходу за межі норми сервіс повідомляє про це користувача.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Після завершення основних етапів проєктування та розробки, невід'ємною та критично важливою частиною життєвого циклу є фаза тестування. Основною метою тестування було всебічне дослідження розробленої системи для виявлення потенційних дефектів, помилок у логіці роботи, невідповідностей функціональним вимогам, а також для оцінки загальної якості та зручності користування.

Основним методом перевірки стало мануальне тестування через клієнтську частину. Цей підхід дозволив не лише перевірити коректність виконання розробленого функціоналу, але й оцінити систему очима кінцевого споживача: наскільки логічною є навігація, чи зрозумілий інтерфейс, наскільки зручно виконувати ті чи інші операції.

Також для серверної частини були створені юніт-тести, як позитивні, так і негативні, які перевіряють коректність роботи запитів до різних елементів API (див. рис. 5.1).

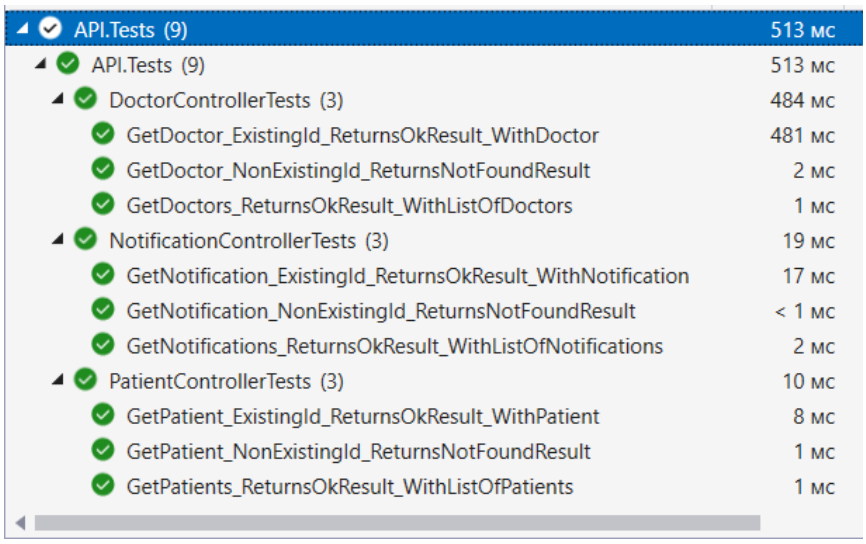

Рисунок 5.1 – Результати виконання юніт-тестів

Функціональне тестування було спрямоване на перевірку того, чи кожна функція системи працює так, як це було заплановано на етапах аналізу вимог та проєктування. Для зручності проведення цього процесу було розроблено набір тест-кейсів, що описують конкретні сценарії використання системи, кроки для їх

виконання та очікувані результати. Нижче у таблицях 5.1-5.4 наведено декілька прикладів тест-кейсів, що ілюструють процес перевірки певних функцій.

Таблиця 5.1 – Тест-кейс №1

Інформація про тест-кейс			
Ідентифікатор тесту	ТС-01		
Власник тесту	Воскобойнікова К.С.		
Опис	Реєстрація нового користувача		
Мета	Перевірити можливості успішної реєстрації нового пацієнта в системі		
Передумови:	Користувач не зареєстрований у системі		
№	Кроки виконання	Очікуваний результат	Відмітка
1	Відкрити головну сторінку веб-додатку	Користувач бачить головну сторінку сервісу	Пройдено
2	Натиснути кнопку «Реєстрація»	Виконався перехід на сторінку реєстрації	Пройдено
3	На сторінці реєстрації заповнити всі обов’язкові поля, обрати роль «Пацієнт»	Всі поля форми реєстрації заповнено коректними даними	Пройдено
4	Натиснути кнопку «Зареєструватися»	Система успішно створила новий обліковий запис пацієнта, користувач бачить сторінку профілю	Пройдено
Результати тестування			
Тестувальник: Воскобойнікова К.С.	Дата прогону тесту: 25.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.2 – Тест-кейс №2

Інформація про тест-кейс	
Ідентифікатор тесту	ТС-02
Власник тесту	Воскобойнікова К.С.
Опис	Створення нового медичного запису лікарем
Мета	Перевірити можливості створення лікарем нового медичного запису для пацієнта
Передумови:	Лікар авторизований у системі. У системі існує принаймні один пацієнт, призначений цьому лікарю

Продовження таблиці 5.2

№	Кроки виконання	Очікуваний результат	Відмітка
1	Увійти до системи у акаунт лікаря	Користувач бачить інтерфейс для лікаря	Пройдено
2	Відкрити сторінку «Медичні записи»	Відкрилась сторінка «Медичні записи»	Пройдено
3	Натиснути кнопку «Створити»	Відкрилась форма для створення медичного запису	Пройдено
4	Обрати конкретного пацієнта зі списку, заповнити всі необхідні поля текстовою інформацією	Всі поля форми заповнено	Пройдено
5	Натиснути кнопку «Створити»	У списку з’явився створений медичний запис	Пройдено
Результати тестування			
Тестувальник: Воскобойнікова К.С.	Дата прогону тесту: 25.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.3 – Тест-кейс №3

Інформація про тест-кейс			
Ідентифікатор тесту	ТС-03		
Власник тесту	Воскобойнікова К.С.		
Опис	Редагування даних профілю пацієнтом		
Мета	Перевірити можливості пацієнта редагувати власні дані профілю		
Передумови:	Пацієнт авторизований у системі		
№	Кроки виконання	Очікуваний результат	Відмітка
1	Відкрити сторінку профілю	Користувач бачить сторінку профілю пацієнта	Пройдено
2	Натиснути кнопку «Редагувати»	Відкрилась форма для редагування даних профілю	Пройдено
3	Змінити номер телефону	Введено новий номер телефону у відповідне поле	Пройдено

Продовження таблиці 5.3

4	Натиснути кнопку «Зберегти»	У профілі відображається новий номер телефону	Пройдено
Результати тестування			
Тестувальник: Воскобойнікова К.С.	Дата прогону тесту: 25.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.4 – Тест-кейс №4

Інформація про тест-кейс			
Ідентифікатор тесту	ТС-04		
Власник тесту	Воскобойнікова К.С.		
Опис	Отримання пацієнтом ел. листа про новий прийом		
Мета	Перевірити створення нових записів про прийоми та отримання пацієнтом ел. листа після цього		
Передумови:	Лікар авторизований у системі. Є доступ до пошти пацієнта, вказаної при реєстрації		
№	Кроки виконання	Очікуваний результат	Відмітка
1	Перейти на сторінку «Прийоми»	Користувач бачить сторінку «Прийоми»	Пройдено
2	Натиснути кнопку «Створити прийом»	Відкрилась форма для створення нового прийому	Пройдено
3	Обрати пацієнта зі списку, заповнити всі необхідні поля текстовою інформацією	Всі поля форми заповнено	Пройдено
4	Натиснути кнопку «Створити»	У списку прийомів лікаря з’явився створений прийом	Пройдено
5	Відкрити ел. пошту обраного раніше пацієнта	У списку листів є новий лист з інформацією про прийом	Пройдено
Результати тестування			
Тестувальник: Воскобойнікова К.С.	Дата прогону тесту: 25.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

У результаті проведення тестів помилки не були виявлені, фактичні результати співали з очікуваними, тобто усі тести пройшли успішно. За підсумками тестування можна зробити висновок, що розроблене програмне забезпечення в цілому відповідає поставленим функціональним вимогам.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи було спроєктовано та розроблено веб-сервіс, який забезпечує можливість віддаленого спостереження за станом здоров’я та комунікації пацієнтів з лікарями. Цей проєкт є актуальним рішенням на сучасні потреби сфери охорони здоров’я, що прагне до підвищення ефективності, доступності медичних послуг та залучення пацієнтів до активного піклування про власне здоров’я.

На початковому етапі було проведено всебічний аналіз предметної галузі та існуючих рішень, що дозволило чітко сформулювати вимоги до майбутньої системи. На основі сформованих вимог була створена серверна частина, яка має багатошарову архітектуру, створено її було за технологією ASP.NET Core 8.0, база даних зі SQL Server, інтерактивний клієнтський веб-додаток на базі Angular, а також програмне забезпечення для IoT-пристрою, відповідальне за збір біометричних даних.

Було проведено тестування, яке включало переважно ручну перевірку функціональності через клієнтський інтерфейс та юніт-тестування серверних компонентів. Це дозволило підтвердити працездатність ключових сценаріїв роботи з системою.

Таким чином, головним результатом кваліфікаційної роботи є створення веб-сервісу, що надає функціонал для контролю за здоров’ям на відстані, включаючи управління медичними записами, обмін повідомленнями між пацієнтом та лікарем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Health awareness in modern society. Times Health Mag. [Електронний ресурс] – URL: https://timeshealthmag.com/health-awareness-in-modern-society/ (дата звернення: 25.04.2025).

2. Revolutionizing healthcare and medicine: The impact of modern technologies for a healthier future–A comprehensive review. PMC Home. [Електронний ресурс] – URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11520245/ (дата звернення: 25.04.2025).

3. Kirimava T. Remote patient monitoring software development: a complete guide. Software Development Company – Custom Application Development Services | Orangesoft. [Електронний ресурс] – URL: https://orangesoft.co/blog/remote-patient-monitoring-software-development (дата звернення: 25.04.2025).

4. Why is early cancer diagnosis important?. Cancer Research UK. [Електронний ресурс] – URL: https://www.cancerresearchuk.org/about-cancer/spot-cancer-early/why-is-early-diagnosis-important (дата звернення: 25.04.2025).

5. Martin R. Clean architecture: a craftsman’s guide to software structure and design. Pearson Education, Limited, 2017. 432 p.

6. Medcenter Maria. Важливість ранньої діагностики. Медичний центр Марія. [Електронний ресурс] – URL: https://medcenter-maria.com/blog/vaznost-rannej-diagnostiki (дата звернення: 26.04.2025).

7. Минулого року загальна кількість медичних працівників в Україні знизилася на 14 відсотків. Український репортер. [Електронний ресурс] – URL: https://ukrreporter.com.ua/war/mynulogo-roku-zagalna-kilkist-medychnyh-pratsivnykiv-v-ukrayini-znyzylasya-na-14-vidsotkiv.html (дата звернення: 26.04.2025).

8. Systematic reviews and meta-analyses of home telemonitoring interventions for patients with chronic diseases: a critical assessment of their methodological quality. JMIR publications. 2013. Vol. 15, no. 7.

9. Android vs ios: mobile operating system market share statistics (updated 2025). AppMySite. [Електронний ресурс] – URL: https://www.appmysite.com/blog/android-vs-ios-mobile-operating-system-market-share-statistics-you-must-know/ (дата звернення: 26.04.2025).

10. ASP.NET Core 8.0 documentation. Microsoft. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0 (дата звернення: 12.04.2025).

11. SQL Server documentation. Microsoft. [Електронний ресурс] – URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (дата звернення: 12.04.2025).

12. Angular documentation. Angular. [Електронний ресурс] – URL: https://angular.dev/overview (дата звернення: 14.04.2025).

13. Connolly T., Begg C. Database systems: a practical approach to design, implementation, and management. 6th ed. Pearson, 2015. 1440 p.

14. Fielding R. T. Architectural styles and the design of network-based software architectures : Doctoral dissertation. Irvine, 2000. 162 p.

15. Електронні матеріали до кваліфікаційної роботи, GitHub. URL: https://github.com/NureVoskoboinikovaKateryna/2025_B_PI_PZPI-21-3_Voskoboinikova_K_S