ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для обліку та аналізу показників артеріального тиску»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Євгеній СУХОМІНСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Євгеній СУХОМІНСЬКИЙ

Керівник: Ірина ЗАМРІЙ
д-р техн. наук, проф.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сухомінському Євгенію Віталійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для обліку та аналізу показників артеріального тиску»
керівник кваліфікаційної роботи д-р техн. наук, професор Ірина ЗАМРІЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості систем дистанційного медичного моніторингу, методи захисту конфіденційних даних, технічна документація фреймворків Spring Boot 3 та Tailwind CSS, бібліотеки React 18, провайдеру Keycloak.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій інтелектуального моніторингу показників артеріального тиску.
2. Аналіз аналогічного програмного забезпечення.
3. Визначення функціональних і нефункціональних вимог.
4. Визначення технічних засобів розробки.
5. Проєктування застосунку для автоматизованого збору показників артеріального тиску.

6. Програмна реалізація застосунку для дистанційного моніторингу артеріального тиску.
7. Провести тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура застосунку.
6. Діаграма послідовності.
7. Діаграма класів.
8. Схема взаємодії з сервісом OpenRouter.
9. ER-діаграма бази даних.
10. Екранні форми.
11. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-08.03.2026	
3	Огляд існуючих методів та технологій інтелектуального моніторингу показників серцево-судинних захворювань.	09.03-16.03.2026	
4	Проектування застосунку для автоматизованого збору показників артеріального тиску.	17.03-31.03.2026	
5	Програмна реалізація застосунку	01.04-15.04.2026	
6	Тестування застосунку	16.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-07.05.2026	
8	Розробка демонстраційних матеріалів	08.05-14.05.2026	
9	Попередній захист роботи	15.05-29.05.2026	

Здобувач вищої освіти _____ Євгеній СУХОМІНСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 70 стор., 1 табл., 22 рис., 36 джерел.

Мета роботи – спрощення збору та аналізу статистичних даних показників артеріального тиску для вирішення проблеми віддаленого нагляду за пацієнтами із серцево-судинними захворюваннями.

Об’єкт дослідження – процес дистанційного моніторингу та контролю показників артеріального тиску пацієнтів із серцево-судинними захворюваннями.

Предмет дослідження – програмні засоби та технології розробки вебзастосунку для збору та аналізу показників артеріального тиску.

Короткий зміст роботи: у роботі проаналізовано методи дистанційного моніторингу артеріального тиску та підходи механізмів автоматизованого аналітики для прийняття клінічних рішень. Досліджено інструменти для розробки покращеної платформи. Розроблено архітектуру системи та програмно реалізовано ключові функціональні можливості, такі як: облік та візуалізація показників здоров’я, механізм знеособлення персональних даних у запитах до зовнішніх сервісах, автоматизований аналіз для клінічних висновків за допомогою ШІ-асистента, кабінети для взаємодії лікаря тат пацієнта. Проведено функціональне та інтеграційне тестування застосунку.

КЛЮЧОВІ СЛОВА: МЕДИЧНИЙ МОНІТОРИНГ, АРТЕРІАЛЬНИЙ ТИСК, ШТУЧНИЙ ІНТЕЛЕКТ, JAVA 21, SPRING BOOT, REACT 18, TYPESCRIPT, KEYCLOAK, POSTGRESQL, DOCKER, SAAS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JPA – Java Persistence API

SQL – Structured Query Language

API – Application Programming Interface

SPA – Single Page Application

LLM – Large Language Model

REST – Representational State Transfer

DTO – Data Transfer Object

JWT – JSON Web Token

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

ВСТУП

Актуальність: гіпертензія є одним із найбільш розвинутих серцево-судинних захворювань і посягає провідне місце серед смертності у світі. Захворювання, зазвичай, є хронічним, що потребує довготривалого контролю та лікування. Тому виникає невідкладна потреба в проведені цілодобового нагляду за пацієнтами. На жаль, традиційні методи не надають повноцінного вирішення проблеми, так як пацієнти часто пропускають прийоми препаратів, забувають зафіксувати показники, або просто втрачають записи. В той час, лікарі мають витрачати час прийому на аналіз зміни тиску для виявлення порушень. Тому створення дистанційної платформи є головним завдання для усунення поточних проблем.

*Об'єктом дослідже*ння є процес дистанційного моніторингу та контролю показників артеріального тиску пацієнтів із серцево-судинними захворюваннями.

Предметом дослідження є програмні засоби та технології розробки вебзастосунку для збору та аналізу показників артеріального тиску.

Метою роботи є спрощення збору та аналізу статистичних даних показників артеріального тиску для вирішення проблеми віддаленого нагляду за пацієнтами із серцево-судинними захворюваннями.

Методи дослідження: застосовано комплексний підхід, що включає: порівняльний аналіз існуючих програмних засобів для визначення та формування функціональних і нефункціональних вимог до системи; об'єктно-орієнтоване проєктування та системний аналіз для побудови архітектури на базі мови програмування Java; методи захисту через провайдер Keycloak; контейнеризація для забезпечення розгортання сервісів.

Наукова новизна роботи визначається інтеграцією наступних функціональних складових: впровадження механізмів промт інжирінгу для взаємодії з хмарними моделями штучного інтелекту шляхом генерації

автоматизованих аналітичних звітів для лікаря; створення умов захисту персональних даних для забезпечення безпечних запитів при використанні зовнішніх сервісів штучного інтелекту; створення єдиного цифрового середовища для постійної взаємодії дистанційного нагляду за пацієнтом.

Практична значущість результатів полягає в створенні надійного рішення, яке дозволяє лікарям швидко відстежувати динаміку стану пацієнта через інтерактивні графіки, мінімізуючи час на традиційний аналіз даних та знизити ризики виникнення критичних ускладнень у хворих на гіпертонію.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАДАЧІ МОНІТОРИНГУ СЕРЦЕВО-СУДИННИХ ЗАХВОРЮВАНЬ

Серцево-судинні захворювання посягають провідне місце в інвалідності та смертності населення всього світу та України. Згідно з опублікованою статтею опублікованою в 2023 році про смертність від хвороб системи кровообігу в Україні кожного року понад 2/3 смертей стаються через серцево-судинні захворювання, що в цифрах складає майже 430 тисяч осіб [1]. В умовах поступового прогресу в сфері цифровізації медицини рівень ризику все ще залишається високим. Тому впровадження систем дистанційного моніторингу пацієнтів є невідкладним інструментом для покращення якості лікування. Однак велика частина державних медичних закладів не притримується такого підходу через відсутність необхідного обладнання та програмного забезпечення [2].

Ключова проблема полягає в тому, що звичайний спосіб контролю тиску який пропонують лікарі просто не ефективний та має свої недоліки. Традиційні методи моніторингу артеріального тиску обмежені в здібності проведення повноцінного вимірювання, здатність пропустити фізіологічні коливання, які можуть негативного вплинути на стан хворого [3]. Також при веденні паперового щоденнику пацієнти часто помиляються, нерозбірливо пишуть, або просто забувають вносити свої показники. Існуючі програми також не вирішують дану проблему цілком, оскільки лише просто дублюють паперовий формат без додаткової аналітики. Через це лікарю доводиться опрацьовувати показники самостійно, витрачаючи час прийому.

Мета цього розділу полягає в дослідженні стану ринку існуючих застосунків для моніторингу кардіологічних показників та виявлення недоліків існуючих сервісів. Отримані результати стануть основою для розробки нової платформи, функціонал якого буде містити автоматизований збір медичних даних та механізми попередньої обробки створення висновків обстеження.

1.1 Роль дистанційного моніторингу в медицині

Використання цифрових інформаційно-комунікаційних систем вже стало інноваційним рішенням сприятливими для запобігання захворюваності серед населення. Вплив глобальної цифровізації зумовила появи нових форматів взаємодії між лікарем та пацієнтом. Згідно з “Глобальною стратегією цифрової охорони здоров’я на 2020-2025 роки”, розробленою Всесвітньою організацією охорони здоров’я, розвиток цифрового лікування буде і надалі мати ціну і інтегруватися в системи медичних установ для надання якісної та доступній допомозі [4]. Завдяки, використанню цієї технології електронні пристрої надають доступ та контроль до керування за станом здоров’я хворого без проведення очних зустрічей. Цей напрям було названо телемедициною, який активно почав інтегрувати цифрові трансформації в електронні системи охорони здоров’я України.

Згідно з означенням телемедицина – галузь медицини, яка надає можливість проводити прийоми у дистанційному форматі переважно для хронічних захворювань. Існують 4 ключові підходи надання телеметричних послуг:

– мобільні застосунки націлені на слідкування за здоров’ям;

– віддалене спостереження за пацієнтами;

– віддалено-синхронні зустрічі з хворими через телефонних перемовин чи з застосуванням платформ відеозв’язку;

– віддалено-асинхронні зустрічі з пацієнтами.

З-поміж усіх розглянутих варіантів для розроблюваного проєкту було обрано модель віддаленого спостереження за пацієнтом у поєднанні з використанням застосунку для слідкуванням за здоров’ям. Такий підхід забезпечує обмін структурованої інформації між пацієнтом та лікарем з метою проведення діагностики, обстеження або лікування. Найчастіше пацієнту потрібно буде надавати діагностичні дані в єдину систему для подальшого перегляду медичного персоналу. В той час, лікар надає в зворотному вигляді медичні консультації та рекомендації за допомогою електронного звіту. Такий підхід зменшує вплив

халатності медичного персоналу, покращує заохочуваність та вплив нервування очної консультації пацієнтів та збільшує відсоток впливу позитивного результату лікування, що значно збільшує якість надання сучасної медичної допомоги [5].

Можливість такої платформи дозволяє пацієнтам користуватися сервісом з будь-якої точки світу без особистої зустрічі з лікарем. Такий формат стає цільовим рішенням проблеми прозорості надання медичної допомоги, а також зменшення проявів виникнення корупції. До того ж, виникає ключовий вплив для людей з захворюваннями серця, так як успіх лікування залежить від тривалості терапії. Тому коли дані вносяться кожен день і лікар має можливість їх бачить, зменшується кількість непотрібних візитів, витрати коштів та часу пацієнта. Особливу роль такий принцип відіграє для працюючої категорії та літніми людьми, оскільки економить час обох категорій користувачів.

До того, телемедицина не лише допомагає пацієнтам, а й підвищує якість роботи медичних закладів. Лікар отримує доступ до контролю стану пацієнта одразу в любий період часу. Можливість оптимізації роботи системи дозволяє зменшити кількість персоналу особливо у період кадрового голоду через необхідність допомоги медиків під час військових дій. До того, телемедицина набуває особливого значення для критично важливої низки категорії, а саме: військовослужбовців, жителів окупованих територій та внутрішньо переміщених осіб [6].

Отже, впровадження систем дистанційного моніторингу є однією з ключових сучасних нововведень, яких сприяє переходу медицини на новий рівень. Формування надійного інструменту контролю покращує ефективність надання медичних послуг, який здатен трансформувати лікування у дистанційний формат, оптимізувати час роботи фахівців й сприяти ефективності до лікування хворого. Використання цифрових платформ здатний покращити точність та ефективності лікування, що підтверджується застосуванням сучасним методиками впровадження телемедицини.

1.2 Використання штучного інтелекту у медицині

Штучний інтелект все надалі займає нішу у сфері охорони здоров'я. Моделі аналізують великі обсяги несистематизованої інформації без людського контролю. Такі підходи дозволяють знаходити нетипові ознаки захворювань, там де традиційні методи часто помиляються. Статистика досліджень підтверджує, що в нейронні мережі працюють так само, як і спеціалісти, але з більшою точністю та швидкістю [7]. Завдяки цьому лікарі отримують дієвий додатковий інструмент який можна використовувати в різних сферах медицини, так як система здатна розраховувати ймовірності ускладнень, що робить лікування більш ефективним.

Однак архітектура нейронних мереж має технічні обмеження. Моделі ефективно знаходять закономірності в структурованих даних, але не здатні встановлювати неочевидні причинно-наслідкові зв'язки на відміну від людського мислення. З тієї причини такі системи не придатні для самостійного ухваленого рішення. Тому такі алгоритми мають виступати лише як додатковий інструмент для попереднього аналізу, тоді як винесення висновків повинна залишитися за лікарем [7].

Під час нагляду за пацієнтами машинний інтелект може збирати та структурувати інформацію, яку надалі оброблювати та швидко видавати свої висновки, що пришвидшує процес прийняття медичних рішень [8]. Оскільки система постійно оброблює потік даних, алгоритми самостійно перевіряють стабільність серцево-судинної системи та визначають показники, які вийшли за межі діапазону. Завдяки цьому впровадженні лікар не витрачає час, а фокусуються лише там де це потребує увага.

Одним з ключових ризиків при інтеграції моделей штучного інтелекту, є використання машинного навчання з хмарних сервісів. Треба уважно ставитися до відправки запитів та надання доступу до персональної та корпоративній інформації. Так як наразі відсутня будь-яка гарантія зловживання конфіденційними даними зі сторони власників хмарних сервісів [9]. Тож рішенням даної проблеми є використання методів шифрування. Контроль прав доступу да надійна анонімізація

стають мінімальним технічним завданням при побудові системи. Витік інформації має ризик оприлюднення конфіденційної інформації та тягне за собою юридичну відповідальність.

Отже використання моделей штучного інтелекту пришвидшує аналіз показників пацієнта та точність діагностики, завдяки делегації функції обробки медичних показників та пошуку ознак хвороби. Система автоматично відфільтрує показники пацієнта та проаналізує, лікарю достатньо відредагувати за потребою висновок та зворотно відправити у платформу. Однак, слід розуміти, що штучний інтелект повинен виступати лише як додатковий інструмент, який повинен виступати у ролі асистента, так як остаточне рішення повинно залишатися за медичним працівником. А використання зовнішніх хмарних сервісів штучного інтелекту потребує додаткової уваги, щодо забезпечення безпеки та конфіденційності персональних даних користувачів для запобігання небажаних випадків витоку.

1.3 Аналіз існуючих програмних рішень

Ринок застосунків медичного спектру перейшов на новий етап розвитку, демонструючи користувачам безліч рішень. Однак, в сфері серцево-судинного моніторингу більшість з них виконують лише функцію пасивного зберігання показників, що є недостатнім для лікування захворювання. Для покращення лікування необхідно побудувати інструменти для проведення більш детального аналізу. Потреба у створенні такої системи пояснюється через аналіз вже доступних аналогів. До порівняння яких залучені дієві рішення, що пропонують контроль за станом серцево-судинної системи на відстані. Проблеми, знайдені під час аналізу, стануть основою для подальшого проектування.

Огляд був приділений сумісностей застосунків від різних виробників, оскільки це є мінімальною умовою для створення подібного продукту. Враховано й наявність кросплатформеності, що регулює чи працює програма однаково добре

на всіх операційних системах. Через ці параметри і буде проводитися розрізнення між звичайних фітнес-програм та спеціалізованих засобів клінічного нагляду.

SmartBP – популярний мобільний застосунок, спеціалізований на контролі артеріального тиску. Програма підтримує ручне ведення даних та автоматичну синхронізацію із обмеженою кількістю брендів тонометрів. При цьому застосунок не має вебверсії та окремого кабінету лікаря для отримання глибокого аналізу даних. Інструменти алгоритмічного аналізу зібраних результатів у системі відсутні. Данні після додавання у систему записуються у локальну базу даних. Система надає доступ до історії змін, наочним графікам та статистичним даним. Крім того є можливість редагувати записи, додавати дату, час та нотатки. Інтерфейс програми більше орієнтований на візуалізацію базової статистики. Присутній експорт показників у різних форматах для подальшої взаємодії з лікарем [10]. Приклад екранних форм застосунку продемонстровано на рисунку 1.1.

Рис. 1.1 Екранні форми SmartBP

Apple Health – потужний застосунок медичних даних для екосистеми IOS. Його перевага полягає в повній незалежності від конкретних брендів медичного обладнання та безкоштовних базових функцій. Проте система перевантажена загальнооздоровчими метриками, де проходить паралельний збір інформації про сон, калорії та тренування. Це ускладнює фокус суто на кардіологічній статистиці. Окрім того, застосунок не надає лікарям інтерактивної інформації для спостереження за пацієнтом. Присутні вбудовані алгоритми, які мають виключно довідковий характер та не підходять для формування клінічних висновків. Закритість екосистеми унеможливлює інтеграції з іншими медичними системами [11]. Зовнішній вигляд робочих екранів системи проілюстровано на рисунку 1.2.

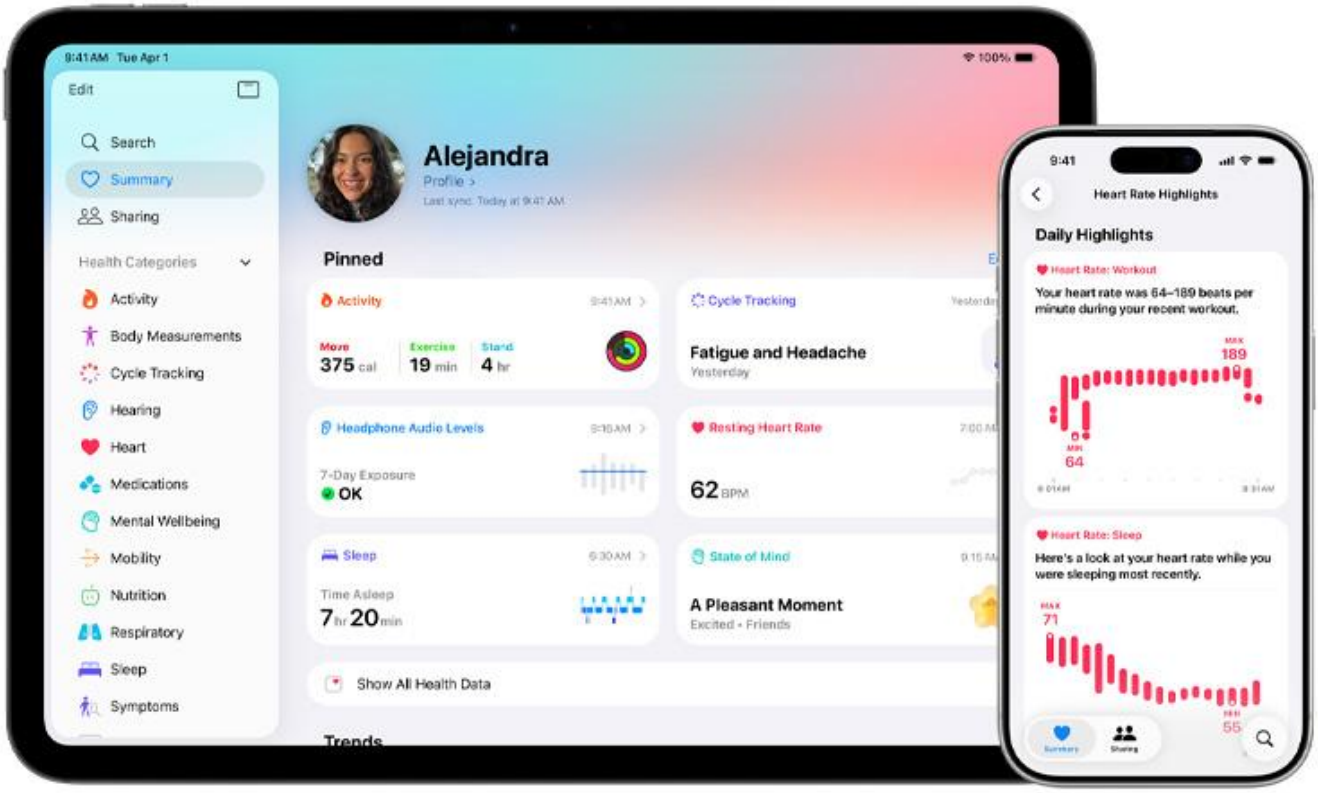


Рис. 1.2 Приклад віконних форм Apple Health

OMRON connect – фірмова розробка японського виробника медичної техніки. Програма функціонує в межах закритої екосистеми і приймає дані лише з пристроїв власного бренду. Подібна прив’язка блокує масштабування сервісу в реальних клінічних умовах, де зазвичай пацієнти користуються різними виробниками тонометрів. Присутня функція відстежування прийому ліків. Інтерфейс не

адаптований під професійну діагностику, так як обмежується базовими порадами щодо способу життя і не містить модулів машинного навчання здатних автоматизовано накладати шаблонних аномалій. Має розширений доступ до інформації пов’язаної про сон, вагу та активність. Незважаючи на вузьку спеціалізацію відсутня вебверсія застосунка [12].

Приклад екранних форм зображено на рисунку 1.3.

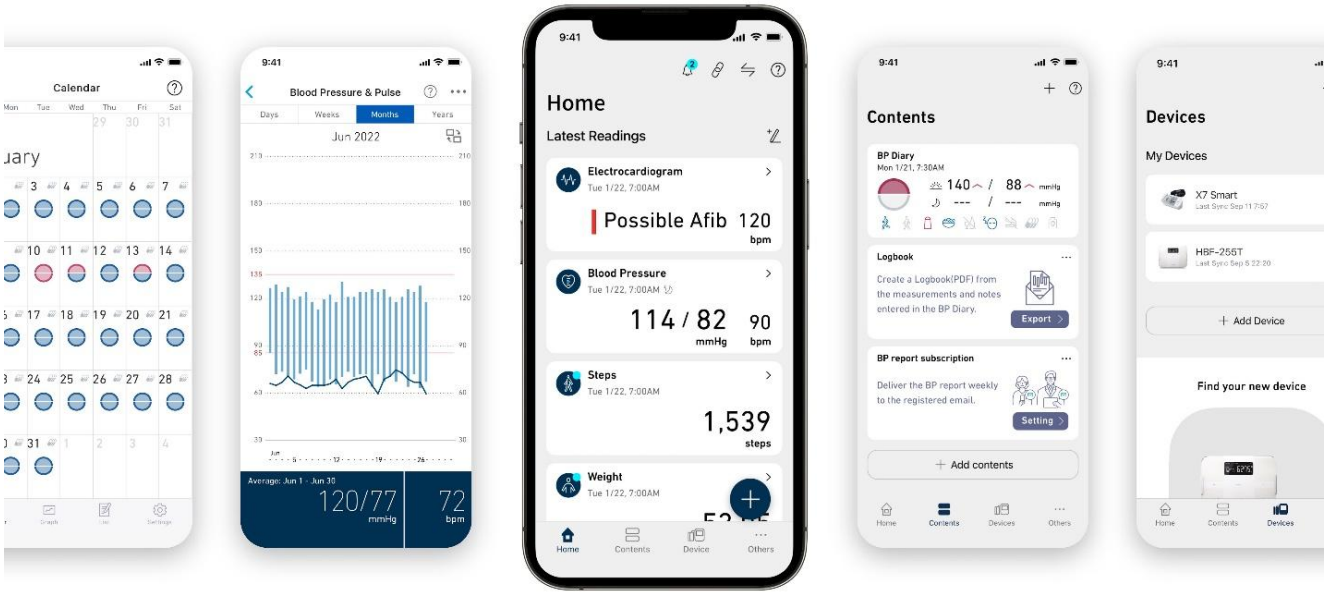


Рис. 1.3 Екранні форми Omron Connect

MedM Health – найбільш технічно сформоване рішення з точки зору телемедицини, що підтримує роботу через браузер та підходить для індивідуального користування. Призначено для покращення стану здоров’я та для лікування хронічних захворювань. Застосунок підтримує ручне та автоматизоване ведення показників з понад 100 різних пристроїв, включаючи тонометри, пульсометри та інші пристрої. Візуалізація даних відображається на інтегрованих дашбордах, що полегшує віддалений огляд. Однак більшість аналітичного функціоналу є платними.

Окремою технічною проблемою є повна відсутність алгоритмів автоматизації розмітки проблемних ділянок. Лікар змушений дивитися на показники на шукати відхилення. Великий об’єм інформації може негативно вплинути на результат через неуважність або втому лікаря [13]. Екранні форми проілюстровані на рисунку 1.4.

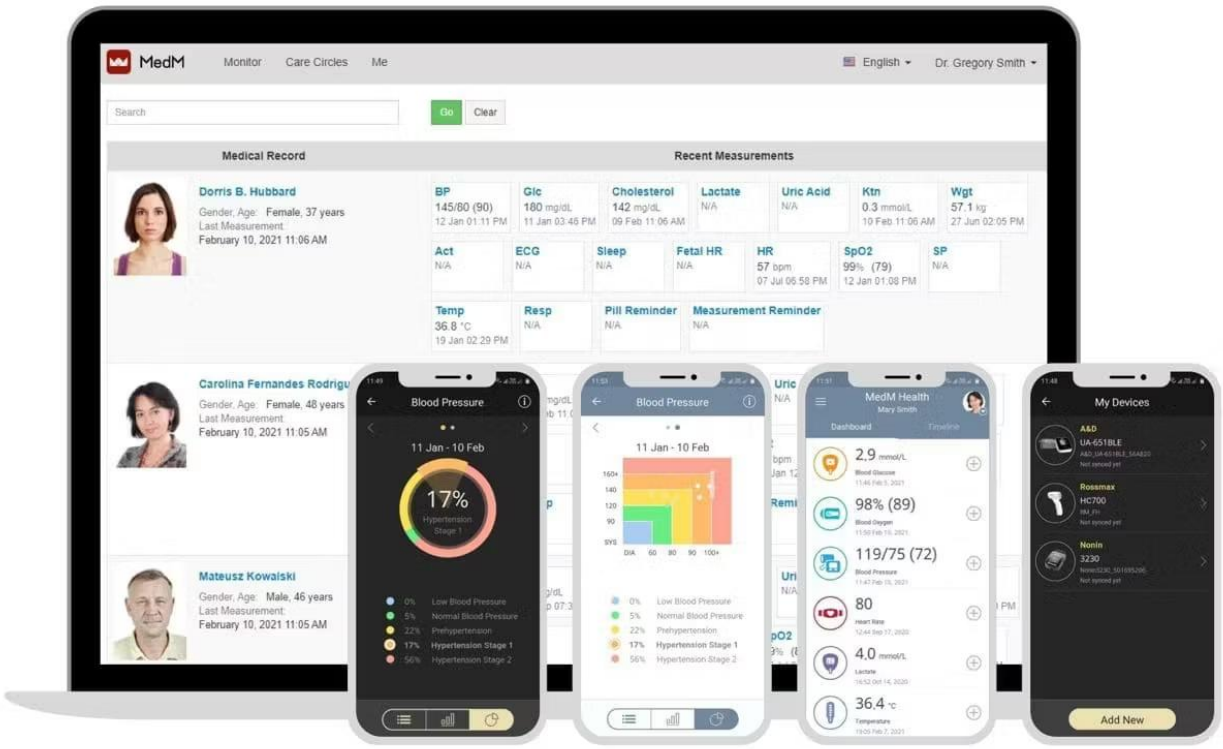


Рис. 1.4 Приклад екранних форм MedM Health

Проведення порівняльного аналізу існуючих аналогів на ринку, крім визначення основного функціоналу, дозволило виявити критичні недоліки в існуючих застосунках. Це дозволило відтворити формування з загальних потреб у базові умови для побудови мінімально життєздатного продукту. Головна ціль якого, отримання відгуку від користувачів для подальшої побудові та розвинення продукту враховуючи реальні потреби.

Таким чином, результати порівняльного аналізу визначили перелік умов до системи і були використані при розробці програмного забезпечення. Підсумовані характеристики продемонстровано в таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця аналізу програмних застосунків

Показник	SmartBP	Apple Health	OMRON connect	MedM Health	CardioCare
Окремий кабінет лікаря	-	-	-	+	+
Робота через браузер	-	-	-	+	+
Безкоштовний функціонал	-	+	-	-	+
Спеціалізація суто на серці (без фітнес даних)	+	-	+	-	+
ШІ-асистент для автоматичних висновків	-	-	-	-	+
ШІ-асистент для автоматичних висновків	-	-	-	+	+

1.4 Безпека та конфіденційність медичних даних

Цифровізаційні заходи переходу медицини на новий рівень потягнуло за собою велику кількість проблем пов’язаних з забезпеченням безпеки системи. Разом із тим велика кількість медичних даних наразі тягне за собою велику чутливість до кіберзагроз, що негативно впливає на активність переходу на дистанційний формат лікування. Тому надійний захист в розроблюваній телеметричній системі має закладатися на архітектурному рівні і жорстоко підпорядковуватися фундаментальним стандартам безпеки. Специфіка цієї галузі вимагає проєктування безпекових заходів на різних рівнях застосунку. Починаючи від моменту введення нових показників пацієнтом до перегляду висновків

сформовані за допомогою лікаря та ШІ-асистента. Лише за умовами повної взаємопов’язаності програмних та технічних рішень можливо досягнути ефективності інформаційної безпеки [14].

1.4.1 Розмежування доступу на ролі

Контроль доступу на основі ролей є механізмом для визначення та керування дозволами доступу за певними ролями. Для кожної ролі надається певні повноваження та визначеними рівнями доступу інформаційних ресурсів, гарантуючи надавання виключно визначені дозволи. Такий підхід ефективно мінімізує ризики, пов’язані з внутрішніми загрозами тим самим зміцнюючи загальну безпеку системи [15]. Для розроблюваного застосунку було відокремлено 3 основні ролі користувачів, кожна з яких має свій перелік дозволів:

– адміністратор володіє найвищим рівнем доступу, що включає керування системою, перегляд стану системи та створенню облікових записів лікарів;

– лікар має доступ до моніторингу закріплених пацієнтів, створення висновків, призначення прийому медикаментів та створення облікових записів пацієнтів;

– пацієнт найбільш обмежена роль, має доступ до давання нових показників артеріального тиску, перегляду висновків лікаря та власної статистики.

Тому, таке розмежування ролей дозволяє забезпечити чітку ієрархію, яка забезпечує особливий стан безпеки, зменшує рівень випадкового втручання інших ролей до керування платформи та надає гарантію того, що кожний користувач працює лише зі своїм функціональним рівнем у системі.

1.4.2 Керування облікових записів

Хоча технічні стандарти важливі, найчастіше проблеми починаються через помилки людей. Головні користувачі системи - це літні люди які не володіють цифровою грамотністю. Навіть найдосконаліший захист стає непрацездатним, коли хтось сам надсилає облікові дані шахраям, наприклад перейшовши на підроблене

посилання. Саме тому, безпека повинна організуватися через ієрархію: спочатку адміністратор додає лікарів у систему. Потім кожен лікар отримує можливість створення кабінету пацієнта. Такий порядок обмежує ризики несанкціонованого входу. Користувачі мають більше довіри до збереження інформації лікарями, ніж до автоматизацій системи [16].

Перехід від вільної реєстрації на користь контрольованого створення аккаунтів дозволяє сформулювати закритий медичний простір, де кожний новий профіль верифікований відповідною особою, що мінімізує ризики несанкціонованого доступу та появи фіктивних даних. Приклад створення облікових записів у платформі моніторингу продемонстровано на рисунку 1.5.

Рис. 1.5 Приклад каскадного створення облікових записів

1.4.3 Деперсоналізація даних в запитах штучного інтелекту

З тенденцією впровадження алгоритмів штучного інтелекту все частіше виникають нові загрози витоку конфіденційної інформації. В медичних системах може зберігатися не лише персональна інформація пацієнтів, а і конфіденційна інформація закладу, яка при витоку можуть використовуватися як проти закладу так і пацієнта.

Основа проблема полягає в тому, що при формуванні нового запиту до зовнішніх сервісів платформа втрачає контроль над даними та не може гарантувати витоку інформації назовні. Крім того, відправлена інформація з великою ймовірністю буде використовуватися для самонавчання алгоритмів [9].

Єдиним шляхом вирішення цієї проблеми полягає в застосуванні обмеженні персональних даних та використання методів анонімізаціїї даних. Такий підхід гарантує, що до хмарних сервісів надійдуть показники без вказування конкретної особи та унеможливлюють прив’язку до конкретної особи чи установи. Система

повинна повністю прибрати згадування пацієнта та усієї інформації, що може визивати загрозу витоку [9].

1.5 Аналіз потреб цільових груп користувачів

Перед початком розробки будь-якого продукту необхідно детально визначитись із потребами майбутніх користувачів. Так як від цього залежить наскільки точно буде побудовано технічне завдання та подальший успіх просування проєкту. Так як в основному задумі система орієнтована на взаємодію між пацієнтом та лікарем основний акцент буде заснований на цих групах. Також окремо треба зазначити додаткову група адміністрації, яка буде виступи у ролі керуючого над станом системи.

1.5.1 Пацієнти

Пацієнти із захворюваннями формують першу групу користувачів. Для цієї категорії обов'язковим є щоденний моніторинг тиску та пульсу. Так як, стандартне заповнення паперових медичних щоденників часто призводять до помилок запису або простої втрати записів, то перехід на цифрову платформу вирішує цю проблему і допомагає зібрати суцільну історію хвороби.

Окрім ведення метрик, пацієнту також додатково необхідний електронний графік прийому ліків з можливостю підтвердження споживання на кожен день. Від екранного інтерфейсу пацієнти очкуватимуть максимальної простоти, так як їм не потрібна складна аналітика, а присутність лише базової візуалізації власної динаміки буде достатнім. Зрозуміла архітектура програми особливо важлива для людей похилого віку, оскільки комфортність застосунку буде стимулювати пацієнта притримувати графіку лікування.

1.5.2 Лікарі

Наступною групою є лікарі. Головний недолік їх роботи полягає в обробці великого об'єму інформації, що супроводжується помилками та неуважністю. При

роботі лікаря для аналізу показників та виявленню стану здоров’я необхідно бачити зміни показників в часі. Тому виникає потреба розробити інструменти, що зможуть автоматично генерувати графіки показників для візуального спрощення сприйняття інформації та пошуку причин захворювань. А використання штучного інтелекту зможе підвищити швидкість оброки та надавати висновки, які подальше можуть бути відредаговані лікарем.

Крім того, велика незручність при роботі лікаря виникає при взаємодії з паперовими документами, що часто супроводжуються помилками та неточностями. Створення автоматичного рішення, дозволяє швидко формувати звіти на основі введених показників, що дозволяє оптимізувати час роботи медичного працівника та звільнити час при потребі на проведення консультацій з хворим.

1.5.3 Адміністратори

Системний адміністратор складає останню третю групу. Його завдання полягає в координації процесів всередині платформи. Оскільки система оперує медичними таємницями, вільна публічна реєстрація в системі має бути закрита, а контроль над створенням профілів повинен залишатися виключно за адміністратором.

Також за адміністративною групою закріплено керування медичною базою препаратів для подальшого призначення лікарями для пацієнтів та керуванню особистими кабінетами лікарів. Адміністратор має можливість не просто створювати нові, а і редагувати та видаляти існуючі картки. А загальна система моніторингу та аналітики за платформою має слугувати інструментом надійності підтримки системи від непередбачуваних ситуацій.

До того ж у разі виникнення технічних проблем з роботою системи роль адміністратора повинна виступати першочерговим рівнем контакту для інший ролей, що повинно забезпечити стабільність та безперебійність роботи застосунку. Усе це разом формує особливий простір управління де кожна дія контролюється адміністрацією.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Створення надійної платформи вимагає використання таких технологій, які здатні витримати роботу з критичними даними пацієнтів. Основна вимога полягає в надійності, щоб система не давала збоїв та конфіденційності для того щоб персональні дані користувачів були надійно захищені. Тому було обрано інструменти які дозволяють притримуватися плану для досягнення усіх цілей проєкту. Кожний варіант рішення було обрано з урахуванням доцільності використання інструменту чи технології та відповідності нормам захисту персональних даних.

Цій розділ буде присвячений детальному розбору кожної використаної технології. Буде розглянуто вибір мов програмування, фреймворків на якій будувалися серверний та клієнтський функціонал. Окрему увагу приділено зберіганню усіх даних в системі, впровадження провайдеру для безпечного входу до платформи та інтеграції хмарного сервісу штучного інтелекту, розгортання системи за допомоги контейнеризації. Наприкінці описано інструменти для проведення тестування системи. Обрані інструменти та технології формують єдину скомпоновану систему, що забезпечують стійкість до навантажень та безпеку роботи в розроблюваному застосунку.

2.1 Мови програмування

Вибір мови програмування займає високий рівень відповідальності на рівні з проєктування усієї системи. Вибір впливає на майбутню продуктивність системи, масштабованість та швидкість розробки. Особливу увагу треба звернути при розробці телемедичних платформ, де надійність та безпека системи займає найбільший ризик. Тому вибір найбільш доцільного інструменту реалізації поставлених задач є дуже відповідальним завдання поставленим перед розробником.

2.1.1 Java

Для створення серверної частини застосунку було обрано мову програмування Java. Ця мова є бездоганним лідером серед мов програмування для побудови надійних корпоративних рішень. Строга типізація критично важлива для медичних закладів, тож якщо замість цілого числа у показнику тиску прийде рядок, компілятор одразу помітить це та сповістить о виявленої помилці ще до падіння всього застосунку [17].

Найбільшою перевагою використання Java є можливість розгортання застосунку на різних операційних системах, так як програми написані цією мовою не залежать ні на рівні двійкового рівня, ні вихідного коду, що є дуже важливим фактором для серверних частинах застосунків. Для стабільності роботи було обрано 21 версію мови, яка гарантує стабільність та довгострокову підтримку розробниками. Саме тому ця мова ідеально вписується під вимоги для створення медичної платформи.

2.2.2 TypeScript

Для відтворення клієнтського інтерфейсу було використано мову програмування TypeScript. TypeScript є надбудовою мови JavaScript з численними перевагами. Мова дозволяє уникати великої кількості помилок за допомогою інтерфейсів. Цей інструмент жорстко вимагає описувати усі параметри зі строгою типізацією та методи яким буде притримуватись новостворений об’єкт. Таким чином за допомогою цього інструменту можна будувати клієнтський інтерфейс з чіткими правилами та передбачуваною поведінкою об’єктів [18].

Мова досі підтримується розробниками компанії Microsoft та гарантує стабільність екосистеми та наявності нових оновлень. А завдяки великому суспільству складнощі та помилки легко вирішується пошуком інформації в інтернеті, що пришвидшує процес розробки. Це робить його головним лідером серед мов для побудови надійної клієнтської архітектури.

2.2 Середовище розробки

Середовище розробки є фундаментальним інструментом, що визначає ефективність розробки продукту та комфортності використання. Сучасні інтегровані середовища коду надають великий спектр можливостей у єдиному середовищі, а саме:

- інтелектуальне автодоповнення коду за допомогою штучного інтелекту;
- вбудований відлагодник коду;
- підтримку контролю версій застосунку;
- інструменти збірки та компіляції.

Тому правильно обране інтегроване середовище має оптимізувати час розробки та зробити його максимально комфортним. Особливо важливим цей вибір стає при розробці багатошарового застосунку, де розробка відбувається як і на клієнтській так і на серверній частини платформи.

2.2.1 IntelliJ IDEA

Для створення надійної серверної частини платформи для екосистеми Java увага припала на використання професійного інтегрованого середовища розробки IntelliJ IDEA. Середовище підтримує великий вміст фреймворків та технологій для побудови надійного бекенду, а також включає підтримку інструментів для роботи з базами даних, підтримки спільної роботи та віддаленого доступу. А головною перевагою є інтуїтивно зрозумілий та привабливий інтерфейс, чого не вистачає у існуючих аналогів [19].

IntelliJ IDEA один з найкращих інструментів, який дозволяє корегувати безліччю параметрів, використовувати шаблони та автоматичний рефакторинг. Редактор дуже послідовний для інших інтегрованих середовищ компанії виробника JetBrains. Крім того цей продукт робить зручним вбудовані система контролю версій, інструмент збірки Maven та Gradle, а також підтримка системи контейнеризації Docker.

2.2.2 Visual Studio Code

Для побудови клієнтського інтерфейсу було обрано редактор коду Visual Studio Code, який використовується для різних мов програмування. Це легкий та максимально функціональний редактор, який став стандартом для веброзробки завдяки своєї гнучкості. Незважаючи на легкість та простоту редактор не поступається повноцінним редакторам завдяки розвитої системі розширень та вбудованих технологій веброзробки [20].

Редактор передбачає можливість легко розширюватися за допомогою встановлення необхідних розширень з офіційного магазину застосунку. Усі доступні доповнення поширюються безкоштовно. Таким чином використання цього інструменту дозволяє з легкістю розгорнути необхідну екосистему розробки у себе на локальній машині, забезпечуючи комфортність роботи.

2.3 Інструменти побудови серверної частини

Серверна частина застосунку є центральним компонентом, який відповідає за обробку медичних даних, забезпечення безпеки та взаємодією між компонентами для стабільної роботи застосунку. При виборі технологій розробки обиралась стабільність роботи та велика підтримка ком’юніті. Крім того, з головних факторів стала, наявність готових механізмів безпеки, інтеграції с системами керування базами даних та зовнішніх хмарних сервісів. Розглянуті технології в сукупності надають сформоване рішення для розробки та підтримки корпоративних, медичних рішень

2.3.1 Spring Boot

Для запуску застосунку з мінімальними налаштуваннями використовувався фреймворк Spring Boot. Цей фреймворк є інструментом надбудови над Java Spring Framework, який пришвидшує процес розробки вебзастосунків і використовується

для створення надійних корпоративних рішень. Переваги які пропонує Spring Framework:

– можливість компілювати виконавчих JAR-файлів, що місять усе необхідне для розгортання;

– вбудовані вебсервера для запуску проєкту під капотом середовища розробки;

– автоматичне конфігурування бібліотек на основі виявленні нових залежностей.

Крім того, в фреймворку присутній механізм автоматичного конфігурування для самостійного визначення залежностей та компонентів, що значно оптимізує роботу з обсягом коду проєкту. Тому використання цього фреймворку є необхідним для комфортної розробки та пришвидшенні роботи за рахунок автоматичного опрацювання та оптимізації певних речей на системному рівні [21].

2.3.2 Spring Security

Розробка рівня доступу даних є складним процесом, який містить великий вміст однотипного шаблонного коду. Для вирішення цієї проблеми безпеки існує Spring Security – це фреймворк екосистеми Spring, відповідальний за безпеку аутентифікації та авторизації розробляємого продукту. Завдяки екосистемі Spring, фреймворк просто інтегрується до основного проєкту без необхідності внесення суттєвих змін у проєкт [22].

Принцип роботи полягає в використанні ланцюжків фільтрів, кожен з яких має свої обов’язки, а кількість регулюється від потреб безпеки. Крім того, найголовніші можливості включають можливість використання шифрування паролів, захист від CSRF-атак і налаштуванні CORS. У розроблюваному проєкті фреймворк відповідає за перевірку JWT-токенів при кожному вході до системи. Крім того налаштування безпеки рольової моделі відбувається саме в цьому фреймворку надаючи та обмежуючи для кожної ролі свій перелік можливих властивостей.

2.3.3 Spring Data JPA

Spring Data JPA – інструмент, який являється частиною екосистеми Spring Boot і розроблений для пришвидшення роботи з інформацією у базах даних. Замість написання великого об’єму коду, достатньо створити інтерфейси репозиторіїв, а фреймворк бере на себе усю відповідальність за дані [23].

Суттєво, Spring Data JPA виступає абстракцією над стандартним JPA та ORM-фреймворком, найчастіше Hibernate, дозволяючи взаємодіяти з базою даних через Java-об’єкти, минув необхідності писати сирі запити SQL-запитів. Основні принципи базових елементів фреймворка:

– сутність – це клас, який повинен збігатися з назвою таблиці бази даних. Кожен екземпляр класу є моделлю, що зберігає стан окремого запису бази даних;

– первинний ключ – це поле у сутності поміченої анотацією @id, яке унікально ідентифікує кожен запис;

– зв’язки – це визначають як сутності зв’язані між собою відображаючи в базі даних через внутрішні ключі;

– EntityManager – це центральний компонент, відповіщаючий за керування об’єктом, їх хешуванням та зберіганням у базу даних. Але Spring Data JPA бере цю відповідальність на себе.

2.4 Інструменти побудови клієнтської частини

Клієнтська частина застосунку є представницьким рівнем для взаємодії користувача та серверу застосунку. Від якості реалізації зовнішнього виду залежить ефективність роботи медичних працівників та пацієнтів. Вибір інструментів зосереджувався на необхідності створені сучасного та ефективного рішення для відображення медичної інформації. Крім того, ще одним важливим критерієм є присутність гарантії в довгострокової підтримки та можливості розширення функціоналу застосунку.

2.4.1 React

Для пришвидшення написання клієнтської частини було прийнято рішення використовувати бібліотеку React. React – це інструмент який значно спрощує створення та подальше впровадження елементів інтерфейсу користувача. Спочатку інструмент використовувався лише для внутрішніх потреб компанії Facebook, які і є розробниками бібліотеки. Згодом було прийнято рішення опублікувати бібліотеку до загального кола користувачів. Основні переваги цієї бібліотеки:

– все побудовано на компонентах, що робить структуру застосунку логічною та інтуїтивно зрозумілою;

– існує велика інформативна база де можна знайти відповіді на ключові питання;

– використання віртуального DOM, що дозволяє оптимізувати оновлення інтерфейсу та скоротити час рендеренгу;

– підтримка роботи з мовою програмування TypeScript;

– гнучкість керуванням додатковими пакетами, надаючи повний контроль над екосистемою застосунку.

Головна ідея React відтворення компонентного підходу. Це означає, що розробники створюють самостійні блоки інтерфейсу які в подальшому можна комбінувати для побудови складних інтерфейсів [24].

2.4.2 Vite

Vite є інструментом для швидкого налаштування середи розробки часто використовуючись у поєднанні з бібліотекою React. Від оптимально підходить для розробки комплексних проєктів передбачуваних взаємодію з сервером. Переважною особливістю є простота інтеграції і швидкість розгортання за допомогою термінала Node.js, що виділяє його серед других допоміжних інструментів [25].

Інструмент пришвидшує створенню ефективних і легко переданих робочих середовищ, полегшуючи командну роботу для нових учасників розробки. Для інтеграції достатньо встановити усі залежності через термінал Node.js та швидко

розпочати роботу. Різниця між стандартними інструментами збірки полягає в використанні нативних модулів, що пришвидшують старт сервера незалежно від розмиру проєкту.

2.4.3 Tailwind CSS

Tailwind CSS – це CSS-фреймворк, побудований на принципах утилітарності, який дозволяє створювати стилізовані інтерфейси, використовуючи готові класи одразу в файлах розгортки HTML. Замість того, щоб одразу пропонувати певні компоненти, фреймворк демонструє набір із гнучких інструментів, які пришвидшують і стандартизують процес розробки. Використання цього інструменту надає повний контроль розробнику на зовнішнім виглядом користувацького інтерфейсу, не обмежуючи визначеними стилями та існуючими компонентами [26].

Існує схожий інструмент Bootstrap, який за певними характеристиками нічім не гірший за новостворений фреймворк. Але Tailwind часто компілюється набагато швидше, так як не має JavaScript компонентів, які можуть уповільнювати завантаження сторінок. А також підтримує теми, що легко міняють зовнішній вигляд сайту без необхідності вносити зміни в код. Крім того, Tailwind CSS автоматично видаляє невикористані стилі під час фінального компілювання та збірки проєкту. Таким підходом легко виконується оптимізація вихідних стильових файлів, зменшується об’єм, що добре впливає на швидкість завантаження продукту.

2.5 Керування даними

Сьогодні існує безліч систем керування даними від класичних реляційних до сучасних NoSQL-рішень. Серед лідерів цього ринку виділяється PostgreSQL, який і був обраний основною базою даних платформи. PostgreSQL – це об’єктно-реляційна система управління базою даних з відкритим кодом, яка успішно

користується в різних сферах діяльності. Система має широкий спектр можливостей, серед яких:

– забезпечення автоматичними обновлюваними представлення віртуальних таблиць, дозволяючи відстежувати зміни в реальному часі;

– гарантування цілісності даних через підтримку транзакцій з ACID-властивостями;

– ефективність обчислень збережених процедур, які виконують складні операції на стороні серверу;

– можливість модифікування вихідного коду і створення особистих функцій та процедур на різних мовах програмування;

– підтримка корисних розширень стандартного SQL.

У розроблюваному проєкті важливо використовувати ACID-транзакції, так як вони гарантують що показники артеріального тиску та медичні висновки будуть збережені коректна, навіть у випадках системних збоїв. Щодо сфер діяльності, PostgreSQL активно використовується в веброзробці для створення різних сайтів та застосунків. В аналітиці даних вона демонструє високу продуктивність при роботі з великими обсягами інформації [27].

2.6 Системи ідентифікації та контролю доступом

При створенні корпоративних вебзастосунків, розробники зазвичай уникають розробку трудомістких власних систем авторизації, тому використання вже існуючих сервісів є необхідною умовою для безпеки власної платформи. На допомогу приходить провайдер Keycloak. Це комплексна платформа керування ідентифікацією та доступом, що централізує і спрощує процеси аутентифікації та авторизації. Ключові можливості платформи включають:

– підтримка інтеграції аутентифікації через соціальні мережі;

– конфігурування інтерфейсу для реєстрації та входу користувачів та передачі функцій авторизації стороннім сервісам;

– централізоване керування правами доступу;
– використання токенів для забезпечення безпеки.

Архітектура Keycloak побудована за принципом клієнт-сервер. Серверна частина виступає готовим пакетом, який можна встановити на будь-яку операційну систему або використовувати у вигляді Docker-образу. Для інтеграції особистого проєкту з сервером потребується використовувати спеціальний модуль клієнтської інтеграції OAuth2/OIDC [28].

2.7 Керування версіями

Керування версіями проєкту є важливим фактором для стабільності та продуктивності розроблюваного застосунку. Відсутність цього інструменту підвищує ризик зламати роботу, а подальший пошук причини займатиме велику частку часу. Для цього проєкту було обрано системою керування версіями Git, основною задачею якого є слідкувати за новими змінами в файлах. Інструмент дозволяє зберігати повно історію коду, що надає можливість повернутися до попередньої стабільної версії за потреби, або порівняти поточну версію з минулими варіантами.

Крім цього цей інструмент може використовуватися і для локального зберігання без використання додаткових хмарних сервісів. А все ж таки для зручності зберігання написаного коду було застосовано платформу GitHub. Призначення цього хмарного сервісу полягає в для демонстрації коду в відкритому доступі та спільно працювати над проєктом не хвилюючись, що змінений код якось матиме вплив для інший пристроїв та учасників розробки.

До того, існує функція створення гілок проєкту, що надає ізольований доступ для нового функціоналу, який не буде порушений і не буде входити до основної гілки. Також існує спеціалізований механізм під назвою запит на вилучення, що забезпечує перевірку усіх змін перед періодом інтеграції до основної гілки проєкту [29].

2.8 Сторонні сервіси штучного інтелекту

Для створення вбудованого асистента в платформі кардіологічного моніторингу для прискорення роботи лікаря з показниками пацієнтів, слід використовувати моделі штучного інтелекту. Але виникають труднощі зв’язані з використанням декількох різних моделей машинного навчання у одному проєкті. Для вирішення даної проблеми використано хмарний сервіс OpenRouter, який виступає дистриб’ютором між користувачем та сервісами штучного інтелекту.

OpenRouter – сервіс, який керує єдиним доступом до великої бази сервісів машинного навчання, централізовано керувати ключами та лімітами інтегруючи в свої застосунки. Сервіс також надає можливість реалізовувати автоматичний перехід від високопродуктивних на більш економічні моделі зберігаючи роботу сервісу [30]. Завдяки легкої інтеграції достатньо проводити звичайні HTTP запити для швидкого зв’язку з машинним навчанням. Таким чином легко проводиться інтеграція, підключення та налаштування сервісу. Серед переваг сервісу можна винести такий список:

– інтегрування велику базу мовних моделей через єдиний API;
– підтримка генерації тексту, коду, зображень;
– прозора система оплати лімітів для повного контролю бюджету;
– відслідковування активностей і отримання детальної статистики по запитам.

Таким чином, інтеграції такого хмарного сервісу у медичну платформу дистанційного формату відкриває можливість комунікації з різними моделями штучного інтелекту з єдиного ресурсу. Застосування цього сервісів дозволяє зосередитись на розробці програмного забезпечення. Єдиний ресурс доступу до великого списку різних моделей, що забезпечує гнучкість системи у виборі більш оптимальної моделі для кожного індивідуального випадку. Єдиний центр для керування ключами та токенами спрощує керування та використання цього хмарного сервісу в умовах реального медичного застосунку.

2.9 Контейнеризація та розгортання

Для автоматизації розгортанні та управлінні платформою було обрано Docker. Застосунок є вкрай популярним програмним забезпеченням з відкритим кодом, що дозволяє пакувати застосунок та усі його залежності в ізольований контейнер. Контейнери працюють незалежно друг від друга використовував ресурси комп'ютера, що дозволяє вирішувати проблема при збірці, запуску та управлінні програмного забезпечення. Різниця між звичайними віртуальними машинами, вище згадані Docker контейнери є більше легшими і оптимізованими, що значно пришвидшує запуск. Це зумовлено тим, що застосунок використовує єдине ядро операційної системи, замість того щоб розвертати повноцінне віртуальне забезпечення [31].

При створенні програмного продукту часто виникає проблема, зв'язана з варіативністю середи розробки. Різні версії використовуваних бібліотек та залежностей на різних пристроях призводять до непередбачуваних наслідків. Програма може належно працювати на одному пристрої, а на іншому з помилками. Пропонований контейнер забезпечує кросплатформеність, що дозволяє запускати застосунок на різних типах пристроїв гарантуючи результат виконання.

Крім того, для роботи одразу з кількома контейнерами одразу можна використовувати Docker Compose, що дозволяє описати усі контейнери та залежності, що будуть запускатися з єдиного файлу конфігурування. Тому запуск клієнтської та серверної частини, системи керування базою даних та сервісом запровадження безпеки при авторизації та аутентифікації виконується однією командою.

2.10 Засоби тестування

Проведення тестування є необхідною умовою для розробки програмного забезпечення. Особливу увагу має для коректності та надійної роботи у медичній галузі, де неправильна робота якогось модулю може призвести до надання

негативної медичної допомоги. Для вибору надійних інструментів тестування визначається охопленням ключових вимог для покриття потреб усіх компонентів системи. Необхідно провести ізольовані тестування сервісів серверної частини застосунку та перевірку клієнтського інтерфейсу. Застосування того підходу має визначити помилки ще на стадії розробки і мінімізувати витрачання часу для вирішення проблем на пізніх стадіях.

2.10.1 Junit 5

Junit 5 – це стандартний засіб для проведення юніт-тестування для мови програмування Java. Також, використовується для перевірки стабільності роботи серверних застосунків. Фреймворк надає можливість створювати спеціальні ізольовані тести. Тестування зазвичай проводиться для сервісів або репозиторіїв серверної частини застосунка. Різниця між попередниками полягає в застосуванні модульній архітектурі, що складається з 3 компонентів кожна з який має свою межу відповідальності: платформи для запуску тестів, рушія для створення тестів та спеціалізований механізм розширення тестів [32].

Фреймворк підтримує анотації дозволяючи швидко описувати на налаштовувати правильну поведінку проведення тестування. Крім того, технологія має функції тестувань з параметрами з можливістю проводити перевірку при різних значеннях. Таким чином прибрати з коду дублювання і зробити код тестування більш читабельним.

2.10.2 Mockito

Mockito є спеціалізованою бібліотекою для створення мок-об’єктів, що дозволяють для тестування відокремлених компонентів. Головна ідея полягає в внесенні заглушок взамін реальних залежностей. Це створено для імітування поведінки залежностей незалежно від реальних залежностей таких як: база даних, зовнішніх API та інших. Використання цього інструменту спрощує тестування з реальним використання хмарного сервісу OpenRouter, оскільки симулює різноманітні сценарії, без виконання реальних запитів до сервісу. Такий інструмент

надає можливість перевіряти різні ситуації без залежності від реальних сервісів [33].

2.10.3 Postman

Цей застосунок використовується для проведення тестування створювання документації REST Api. Застосунок надає графічне відображення для зручного опрацювання ендпоінтів серверної частини застосунку. На відміну від інших типів тестування Postman виконується в ролі зовнішнього клієнта імітуючи реальні запити. Використання застосунку використовується для перевірки коректності роботи ендпоінтів на етапі інтеграційного тестування [34].

Зі сторони функціювання є зручне функціювання для комплексного перевірки одразу усіх тестів. Для цього достатньо створити колекцію поділену на групи тестів для повторного проведення тестування чи подальшої корегуванням. Також, присутня функція створення змін середовища для зручного доступу до різних версій тестування між локальними та опублікованим середовищем без змін адресів запитів. Такий функціонал прискорює проведення тестування на усіх етапах розробки.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення завжди починається з етапу проєктування. Це необхідно для формування чіткого технічного завдання, що дозволяє приблизно уявити кінцевий результат ще до старту розробки. Результатом є деталізований план, який надає розробникам чітке розуміння завдань, вибору інструментів та технологій для реалізації задуманого. Проведення проєктування застосунку дозволяє виявити усі неточності на початковому рівні розробки. Особливо важливо проводити проєктування архітектури заступнику для медичної галузі, так як поява помилки при проєктуванні може призвести до критичного стану проблем застосунку.

Розділ присвячений визначенню технічних специфікацій платформи, обґрунтуванню архітектурних підходів, та опису логіки взаємодії компонентів системи. Розглянуті рішення в цьому розділі демонструють рішення для загального стану системи так і до окремих компонентів, які взаємодіють між собою для досягнення спільної мети розроблюваного продукту.

3.1 Технічні вимоги

Технічні вимоги до програмного забезпечення визначають детальний опис того, як система повинна функціонувати та якими якостями володіти. Чіткі вимоги допомагають уникати непорозумінь між замовником та розробниками, гарантуючи, що фінальний продукт буде саме таким, як і очувалось. Вимоги поділяються на функціональні, що описують дії системи, та нефункціональні, що стосуються її характеристиками.

У сфері створення програмних продуктів, чітке розмежування між функціональними та нефункціональними вимогами є фундаментальними. Обидві групи доповнюють один-одного і мають першочергове значення для забезпечення успіху проєкту. Правильно сформовані вимоги вирішують подальшу долю

створюваного застосунку, проєктування архітектури, розробки та подальшого тестування. Особливо важливо закласти вимоги до забезпечення безпеки та конфіденційності, оскільки система буде оперувати медичними операціями. Окрім цього для визначення вимог доречно розробити візуальну діаграму, яка окреслюватиме поведінку системи по ролях. Для цього на рисунку 3.1 представлено діаграму варіацій класів для демонстрації роботи застосунку зсередини.

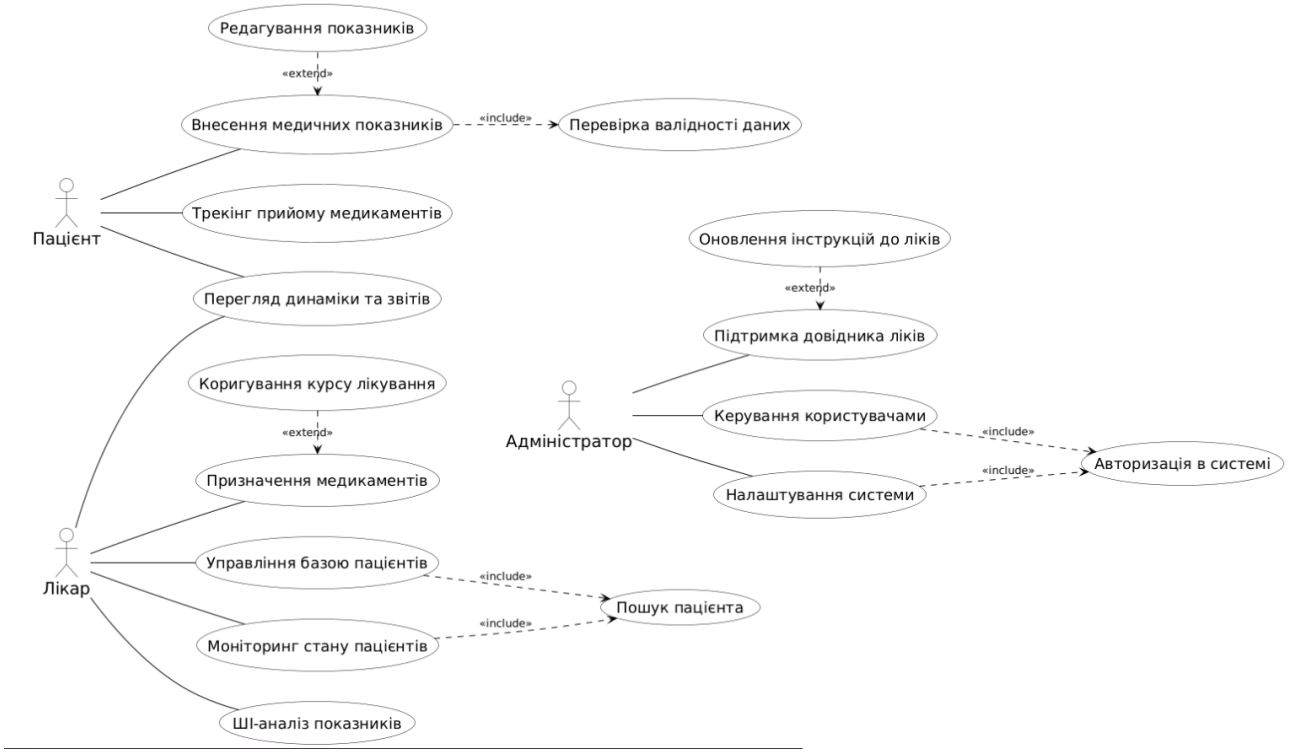


Рис. 3.1 Діаграма варіантів використання

3.1.1 Функціональні вимоги

Функціональні вимоги описують, що саме повинна робити система програмного забезпечення для досягненні своєї мети. Вони визначають її основні функції та особливості, а також як вона взаємодіє з користувачем та поводиться в різних ситуаціях. Правильно сформовані вимоги слугують початком для проєктування архітектури, так як дуже важливо, щоб вимоги відповідали реальною поведінкою системи. При розробці платформи медичного моніторингу було визначено вимоги за такими критеріями:

– реєстрація та авторизація різних типів користувачів (пацієнт, лікар, адміністратор) за допомогою єдиної системи управління ідентифікацією;

– можливість вносити, зберігати і переглядати історію вимірювань (пульс, артеріальний тиск) на інтерактивних вебдашбордах у реальному часі;

– призначення лікарем схеми лікування та автоматизований трекінг прийому медикаментів пацієнтом;

– формування електронних медичних звітів лікарем;

– інтеграція з уніфікованою платформою штучного інтелекту для автоматичного підготовки аналітичних звітів для лікаря.

Визначені функціональні вимоги застосовують повний цикл і формують чітку структуру для подальшої розробки програмного забезпечення. Кожна з перелічених умов відповідають потребам користувачів у ході аналізу реальних потреб.

3.1.2 Нефункціональні вимоги

Нефункціональні вимоги доповнюють функціональні, пояснюючи, як система повинна виконувати свої завдання. Вони встановлюють стандарти якості, тоді як швидкість роботи, надійність, безпека та легкість використання, а не перераховують їх дії. Основними нефункціональними вимогами є:

– застосування Keycloak для забезпечення безпечної автентифікації та надійного захисту облікових записів користувачів;

– анонімізація персональних даних шляхом заміни ПІБ на випадкові UUID перед їх відправкою запитів до зовнішніх ШІ-сервісів;

– розподіл системи на незалежні модулі використовуючи Docker;

– відображення логування та моніторингу стану системи.

Нефункціональні вимоги визначають якості якими повинна володіти система. Без дотримання цих умов повність завершений продукту по функціоналу не забезпечить високого рівня безпеки та стабільності продукту в умовах реального запуску застосунку.

3.2 Структурна побудова системи

Для проєктування складних програмних рішень, важливо використовувати готові архітектурні шаблони, для уникнення типових помилок. Для реалізації даного проєкту було обрано клієнт-серверну модель, що є стандартом для сучасних застосунків. Така модель стала широко використовуваною завдяки розширенню мережі інтернет, що справило до масштабного зберігання інформації у базах даних серверів.

Клієнт-серверна архітектура – це модель побудови інформаційних мереж, де сервери виступають у ролі основних ресурсів, що обслуговують запити від клієнтів. Розподіл функцій між клієнтом та сервером зумовлений гнучкості системи, що дозволяє масштабувати та розвивати частини системи окремо. Взаємодія передусім визначається тим, що розподілені завдання між ролями, можна логічно поділити на три основні операції:

– рівень представлення даних, що фактично є інтерфейсом для користувача. Він відповідає за відображення інформації користувачеві та прийом його команд;

– рівень прикладної обробки, де реалізована основна логіка програми, де відбувається необхідна обробка даних;

– рівень управління даних, що забезпечує зберігання даних та контрою до них.

Кожний рівень може функціонувати лише с сусіднім, таким чином маючи ізоляцію. Відповідальність перекладається на кожній рівень і гарантує пряму залежність між інтерфейсом користувача та зберігання інформації в базу даних. Організація такого підходу забезпечує систему надійною безпекою, так як фронтенд частина застосунку не матиме прямого доступу до бази даних, а взаємодія буде проходити лише через захищений протокол на бекенд частину застосунку. На рисунку 3.2 продемонстрована структурна схема клієнтської частини застосунку.

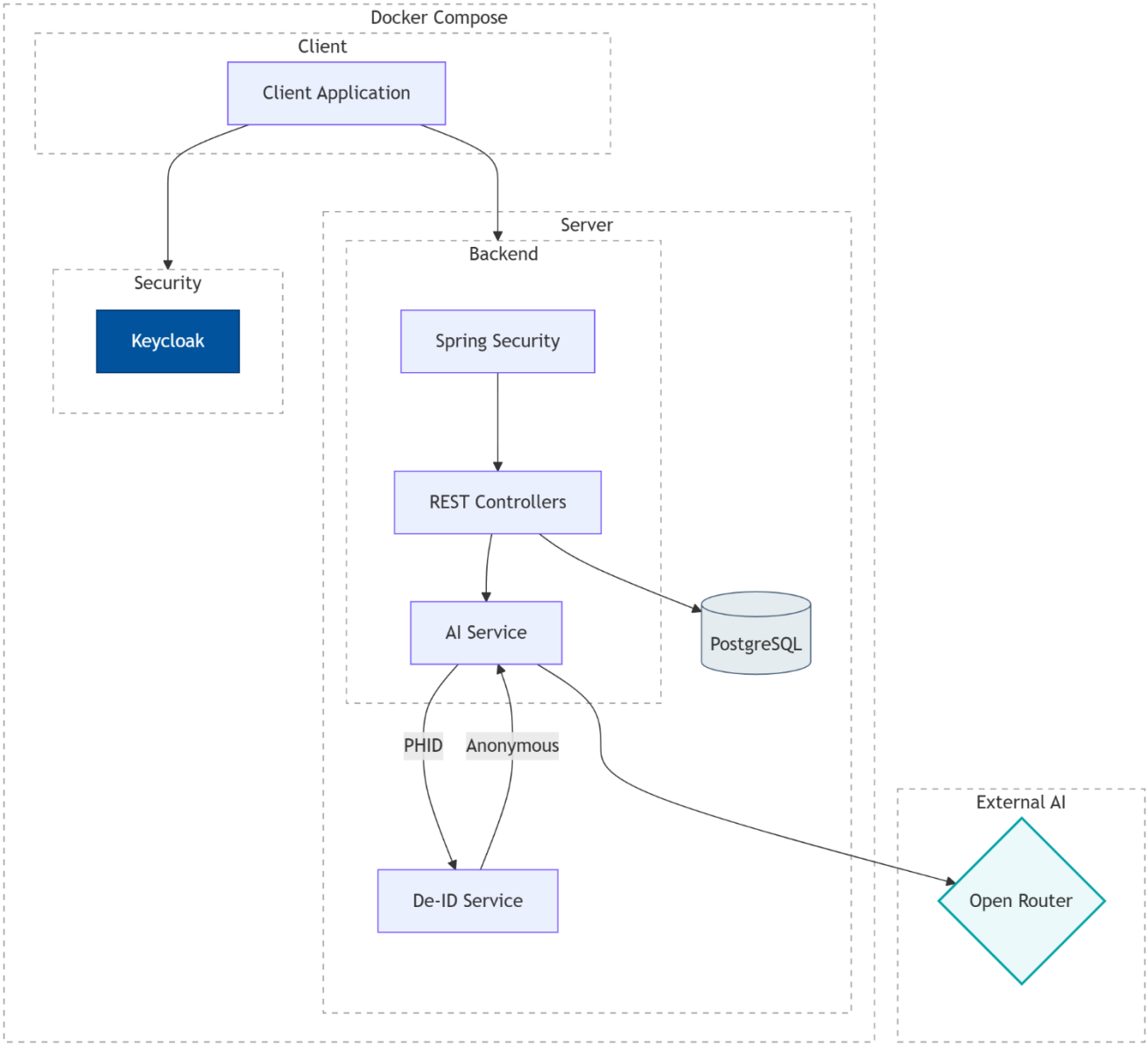


Рис. 3.2 Структурна схема клієнтської частини

3.3 Архітектура клієнтської частини

Головною метою вебзастосунків є забезпечення максимального комфорту взаємодії з користувачем. Одним із шляхів досягнення мети – швидке завантаження вебсторінки. Саме тому для розробки нових проєктів частіше всього обирають архітектурний підхід Single Page Application.

SPA – популярний архітектурний метод, який функціонує без необхідності перезавантаженні усієї сторінки. При першому запиті усі необхідні ресурси завантажуються один раз, а надалі керування усіма елементами сторінки

проводиться динамічно. Користувач у свою чергу бачить це як перехід між сторінками, хоча фактично змінюються лише компоненти. Як результат пришвидшується взаємодія між користувачем та платформою. Крім того, використання такого методу дозволяє розробити більш складний інтерфейс, ніж традиційним багатосторінковим підходом. Це пришвидшує відображення динамічного контенту важливого для інформаційної платформи такого як: графіки, діаграми, картки та інші [35].

На рисунку 3.3 зображено структурно схему клієнтської частини платформи, розробленої за архітектурним підходом SPA. Схема демонструє організацію клієнтського відображення та логіку взаємодії компонентів.

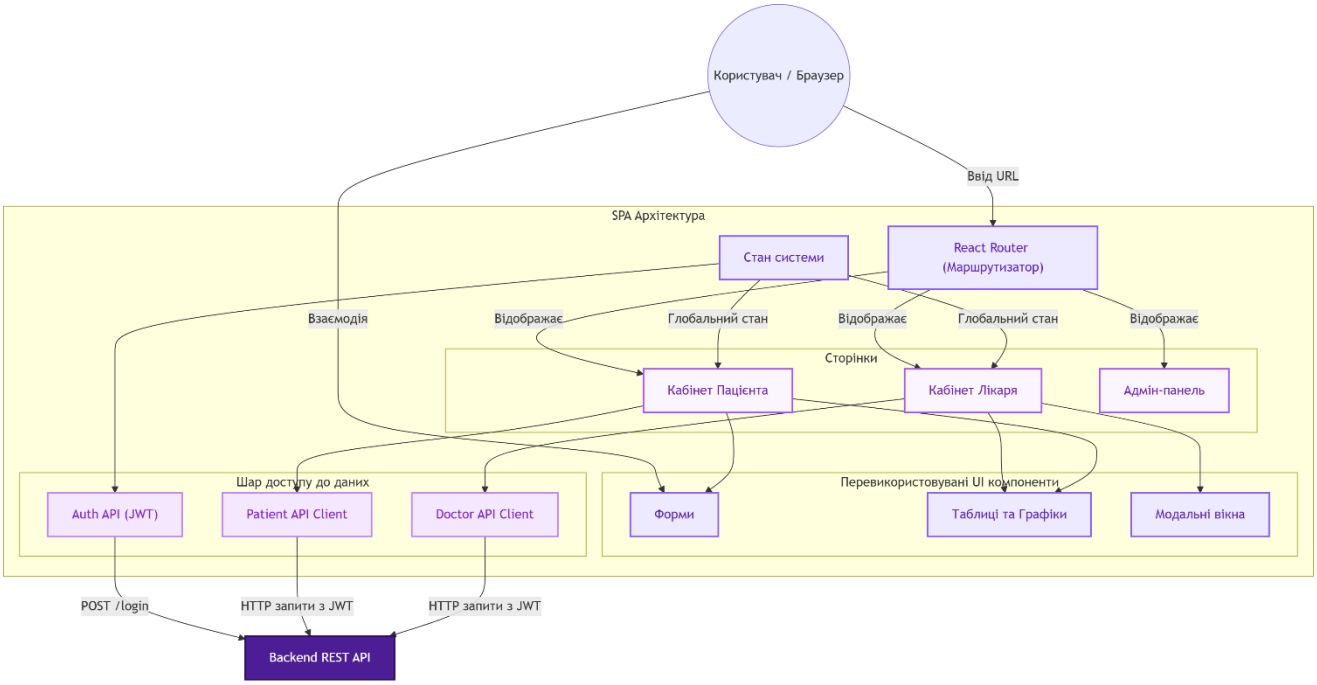


Рис. 3.3 Структурна схема клієнтської частини

Керування усіма елементами інтерфейсу, такими як форми, кнопки, меню та інші, зазвичай здійснюється за допомогою фреймворків, які сприяють модульної структури і дозволяють створювати багаторазово використовувані компоненти. Для цього проєкту було обрано React, що став першим фреймворком з інтеграції компонентного підходу користувацьких інтерфейсів. Де кожен компонент є самостійним елементом, який можна повторно використовувати для побудови

складних систем. Серед цього можна виділити ключові переваги таких односторінкових застосунків:

– застосунки універсально гарно працюють на різних типах пристроїв та операційних системи;

– архітектура дозволяє реалізувати біліший спектр функцій, недоступний для традиційних багатосторінкових сайтах. Можливістю відтворити вебзастосунок на мобільному пристрої у ролі імітаційного нативного застосунку;

– функціювання вебзастосунку без стабільного інтернет з’єднання, за допомогою механізмам хешування;

– при зміні сторінок завантажується лише актуальний контент, що пришвидшує процес взаємодії.

Тому використання Single Page Application у сучасній веброзробці є необхідним архітектурним підходом, що створює швидкі та динамічні вебзастосунки, забезпечуючи білиш високий статус застосунку. Переваги його використання підтверджені опрацюванням ролями застосунку при роботі з інформацією пов’язаною з медициною. Такі компоненти як: показники артеріального тиску, графіками стану здоров’я та призначень лікарських засобів вимагають від себе швидкості відображення, при перезавантаженні сторінки платформи наново.

Використання архітектурного підходу Single Page Application з бібліотекою React закладено на фундаментальному рівні. Це рішення відповідає технічним вимогам оптимізації процесів для відображення складного контенту в реальному часі.

3.4 Архітектура серверної частини

При розробці серверної частини застосунку часто виникає проблема в забезпеченні гнучкості системи, так як бізнес-вимоги, які є першопричиною розробки, постійно еволюціонують. Тому для проєктування серверної частини

застосунку необхідно здійснювати з урахуванням архітектурних принципів, що розвивають окремі компоненти системи незалежно один від одного.

3.4.1 Розмежовування рівнів

Для пришвидшення процесу розробки серверної частини виникає необхідність в використанні принципу Clean Architecture. Правила які визнають бізнес-процеси повністю незалежні від будь-яких зовнішніх елементів, таких як фреймворки, бази даних та інші. Ключова ідея цієї моделі полягає в поділу застосунку на рівні з чітким розподілом відповідальності [36]. Типова структура включає:

– рівень доменів, що є серцем застосунку, де зосереджена уся основна бізнес-логіка. Зазвичай визначаються правила, процеси та сутності, які складають суть застосунку. Шар повинен бути незалежним і не повинен знати про існування інших шарів;

– рівень представлення, що відповідає за взаємодію з користувачем і з зовнішніми сервісами. Усі механізми приймають дані від користувача або системи, а також відображають інформацію назад. Шар не містить складної бізнес-логіки. Його єдина задача полягає в керуванні потоком даних;

– рівень застосунку, що виступає в ролі посередника між рівнями представлення та доменним. Отримує запити від представлення і виконує їх обробку та підготовку, а після делегує виконання основної бізнес логіки доменному рівню;

– рівень конфігурації, що відповідає за взаємодію з зовнішніми системами, сервісами і технологіями. Він інкапсулює деталі реалізації пов’язані з базою даних, мережевих запитів чи файловою системою.

Особливу увагу приділено проєктування рівня домену. Оскільки він є фундаментом усієї архітектури проєкту, що втілює ключові правила системи. На цьому рівні закладаються основа логіки між пацієнтом та лікарем, яка повинна залишатися стійкою до будь-яких змін у виборі фреймворків чи баз даних. Для детального представлення об’єктної структури платформи між основними

компонентами медичної платформи, продемонстрована доменна модель, представлена діаграма класів на рисунку 3.4.

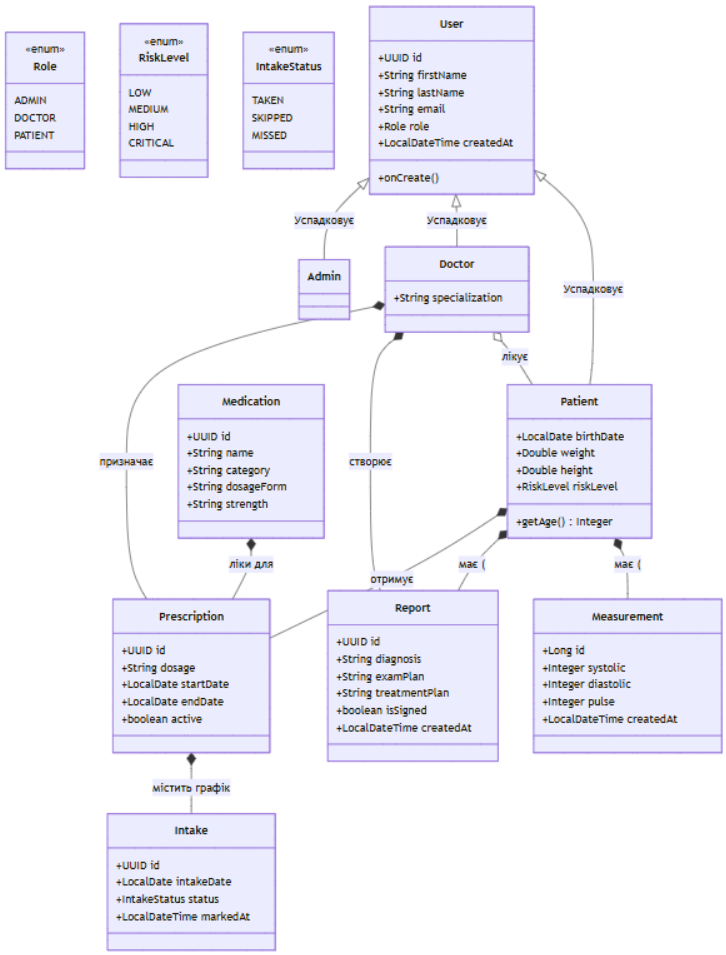


Рис. 3.4 Діаграма класів доменного рівня

3.4.2 Взаємодія між компонентами системи

Для розуміння взаємодії між компонентами системи важливо розглянути не лише структуру доменного рівня, а і конкретний приклад HTTP-ендпоінт взаємодії, через які відбувається обмін даними. REST Api є основним інтерфейсом комунікації між фронтенд та бекенд частиною, тож важливо дотримуватися конкретності проєктуванні маршрутів. Для більш глибокого розуміння представлення послідовностей викликів та взаємодії між пацієнтом та лікарем на рисунку 3.5 зображено діаграму послідовностей збереження та отримані зворотної відповіді користувачу.

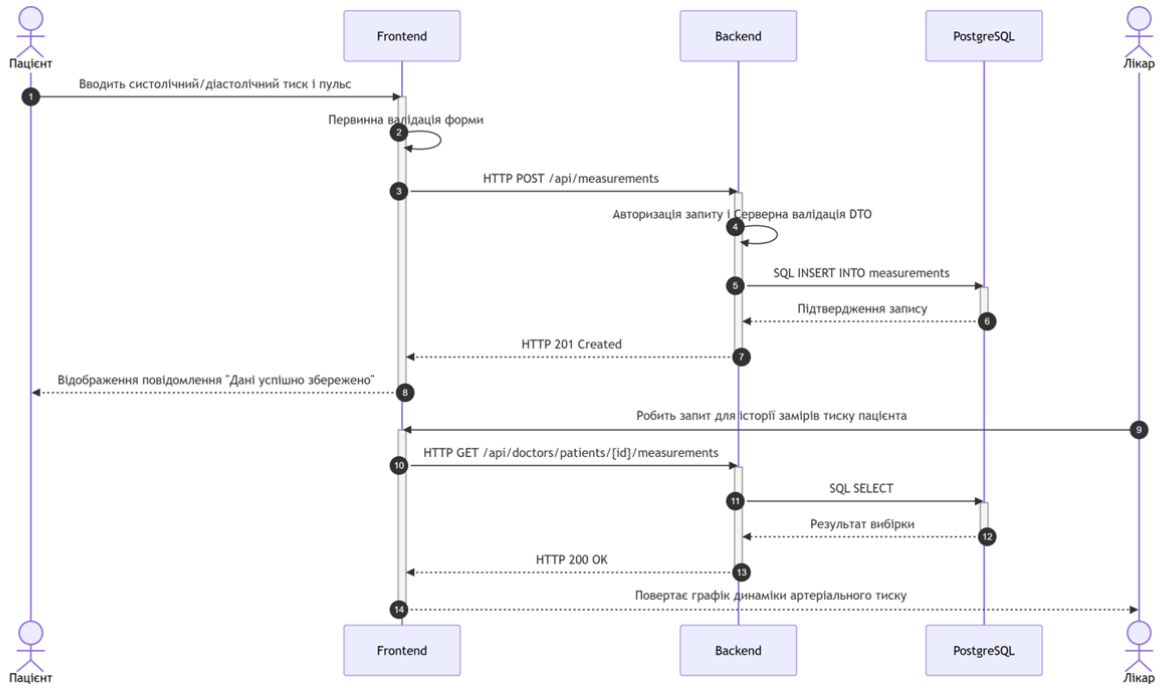


Рис. 3.5 Діаграма послідовностей

Пацієнт вводить показники тиску після чого фронтенд виконує першочергову валідацію форми та надсилає запит до бекенду. Бекенд у свою чергу авторизує запит і виконує валідацію DTO та зберігає дані у базу даних PostgreSQL, після чого повертає успішний код HTTP 201 Created, а пацієнт підтвердження про успішність операції. Паралельно лікар може відкрити панель моніторингу де відтворюється запит до серверу за отримання результатів вибірки показників конкретного пацієнта. Після успішності операції повертається код HTTP 200 Ok, а на фронтенді відображається графік стану пацієнта.

3.5 Реалізація вбудованого аналізу на базі штучного інтелекту

Платформа оснащена модулем автоматизованого аналізу медичних даних пацієнтів, що надає лікарям інтелектуальну підтримку в прийнятті рішень. Цей модуль використовує мовні моделі LLM через платформу OpenRouter. Архітектура побудована на взаємодії між інтерфейсом користувача та сервером застосунка з зовнішнім API хмарного сервісу. Динаміку роботи модуля та послідовність

взаємодії компонентів під час генерування висновків моделями штучного інтелекту зображено на рисунку 3.6 діаграми послідовностей.

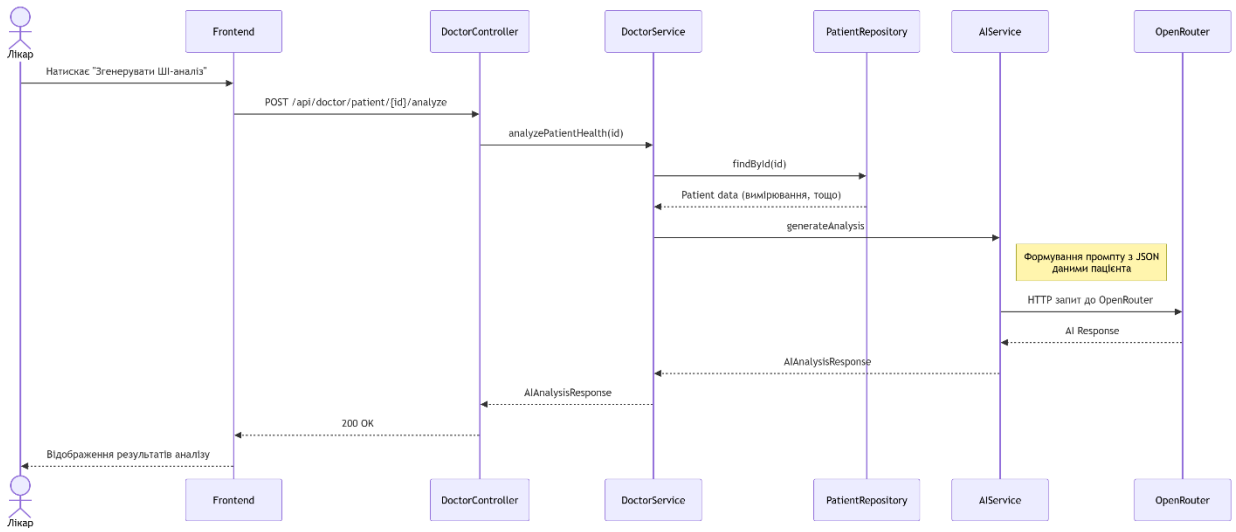


Рис. 3.6 Взаємодія вбудованого аналізу показників з сервісом OpenRouter

Процес генерації інтелектуального аналізу можна розділити на кілька логічних етапів:

1. Запуск аналізу. Лікар ініціює процес у клієнтській частини платформи, яка надсилає запит на сервер для отримання повної медичної історії пацієнта;

2. Підготовка даних. Медичні показники оброблюються та формується запит для мовної моделі;

3. Звернення до OpenRouter. Сформований запит надсилається до хмарного сервіса до обраної моделі штучного інтелекту. Такий принцип полегшує зміну мовних моделей без зайвої модифікації коду;

4. Відображення результату. Отримана відповідь від хмарного сервісу повертається до клієнтського інтерфейсу.

Такий підхід дозволяє ефективно використовувати зовнішні обчислювальні ресурси для аналізу, забезпечуючи безпеку даних та швидкість роботи інтерфейсу завдяки швидкої обробці даних на стороні зовнішнього сервісу.

3.6 Інформаційне забезпечення моделі даних

Для забезпечення функціоналу медичної платформи та надійності системи до високих навантажень необхідною умовою є правильно спланувати модель бази даних на основі реляційної системи управління базами даних PostgreSQL. Проєктування було основане на принципі від сутності до зв'язку, що дозволило створити гнучку структуру, яка легко масштабується та легко інтегрується до застосунку. Використання реляційної бази даних є оптимальним рішенням для медичної платформи, так як медична інформація завжди відображається лише чіткими показниками, логічними зв'язками між сутностями.

3.6.1 Методи збереження та безпеки даних

Для підвищення безпеки застосунку було прийнято рішення про підвищення надійності системи:

– для усіх ключових сутностях замість послідовних числових значень обрано спеціальний тип UUID. Це забезпечує анонімність даних при передачі до хмарних сервісів та спрощує майбутню синхронізацію між різними серверами;

– впроваджено суворі обмеження на рівні зовнішніх ключів. Наприклад, видалення профілю пацієнта автоматично видаляє і усі пов'язані журнали моніторингу, що запобігає повні застарілих та неактуальних даних;

– інформація про лікарські препарати було винесено в окрему таблицю Medications, що дозволило уникнути дублюванні текстових описів.

Перелік цих рішень надає багаторівневий захист ще на рівні бази даних для захисту персональних иедичним даних пацієнтів. Таким чином рівень безпеки відбувається на кожному рівні, що дуже впливово для медичної платформи. Підхід забезпечує відповідність системи до стандартизованих норм.

3.6.2 Структура інформаційного середовища

Архітектура даних платформи побудована на принципі розподілу інформації за чітко визначеними функціональними областями, що відображено на розробленій

ER-діаграмі рисунку 3.7. На відміну від традиційних лінійних структур, обрана модель використовує централізовану топологію, де центральним елементом системи виступає сутність User. Навколо цього і сформовано три основні рівня взаємодії.

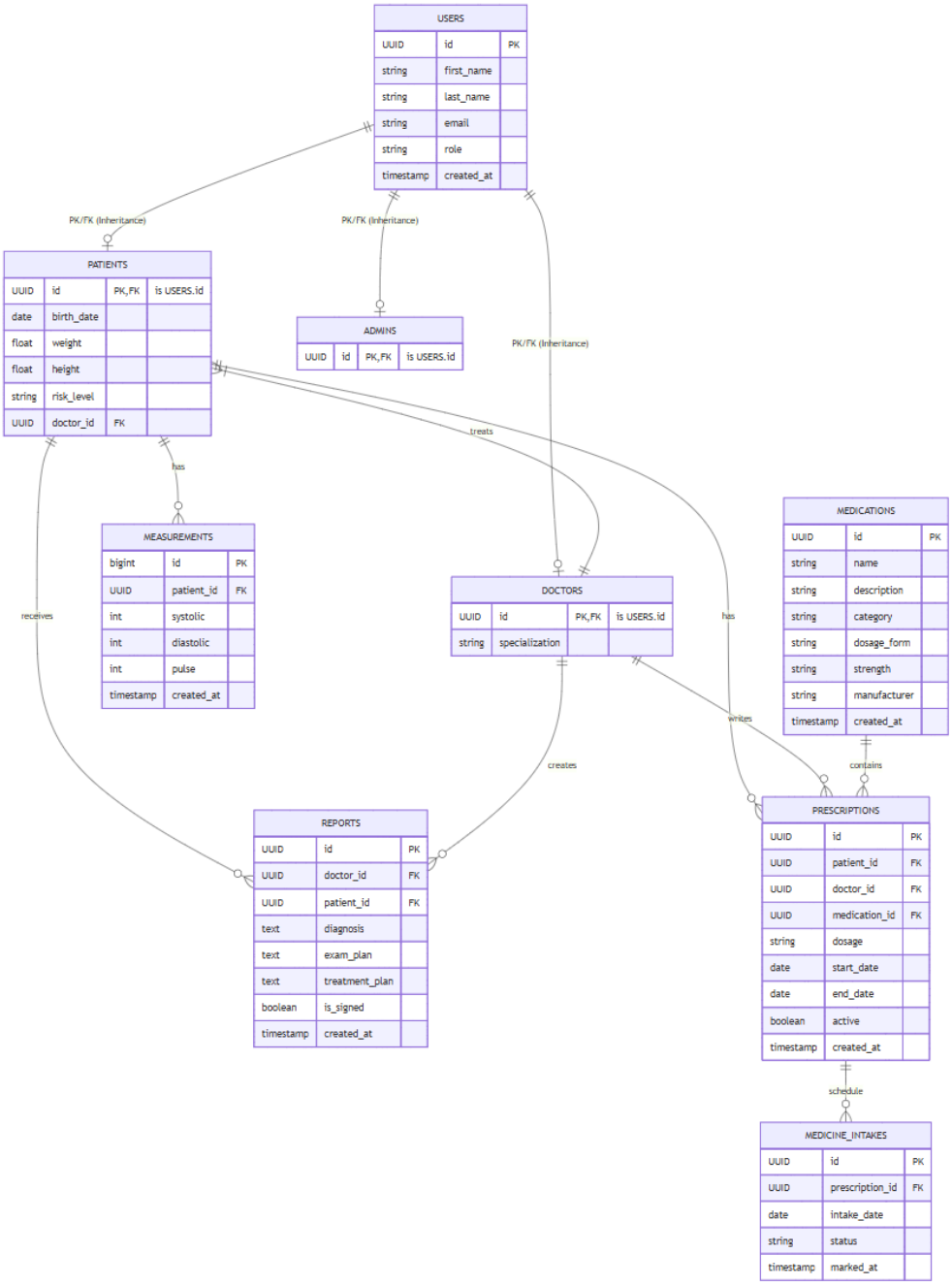


Рис. 3.7 ER-діаграма бази даних

Перший рівень, виступає в ролі ідентифікації. Його стратегія полягає в тому, що він успадковує. Це дозволяє відокремити загальні профілі користувачів від специфічних даних лікаря, пацієнта або адміністратора. Другий рівень є аналітичний. Забезпечує зв'язок між пацієнтом та набором динамічних показників таблиці Measurements, а також відповідними експертними звітами таблиці Reports. Завершальним структури є терапевтичний рівень, який охоплює весь життєвий цикл призначення лікарських засобів. Він починається з моменту створення рецепту таблиці Prescriptions і триває до фіксації кожного окремого випадку прийому препарату таблиці Medicine_intakes.

Для більш детального опису кожної таблиці бази даних потрібно описати призначення кожної сутності. Першим рівнем виступає сутності рольового доступу:

– users: виступає базовою таблицею, яка зберігає загальну інформацію користувачів не включаючи конфіденційної інформації. Крім того, виступає основою для успадкування більш конкретних сутностей рольового доступу до платформи;

– patients: розширює базову таблицю користувача і виступає в для зберігання особистої інформації пацієнта. Зберігає конфіденційні характеристики про дату народження, зріст, вагу та спеціалізований індексатор прикріпленого лікуючого фахівця;

– doctors: зберігає інформацію особистого кабінету медичного працівника із зазначенням професійного напрямку лікування в кардіології для правильного розподілу обов'язків;

– admins: використовується для зберігання інформації про особисті кабінети технічних спеціалістів адміністрування системою, які здійснюють нагляд за коректністю роботи системи та наповненості інформаційного реєстру (нових лікарів або списку ліків).

Наступна група демонструє сутності терапевтичного рівня відповідачі за збереження реєстрових даних:

– measurements: виступає в ролі журналу для фіксації показників артеріального стану пацієнтів. Використовує поля для збереження інформації про систолічний та діастолічний тиск, пульсу, та часу коли було проведення замір. Таблиця виступає в ролі центрового джерела для відображення інформації та побудови графіків;

– reports: зберігає фінальні висновки зроблені лікарем за результатом дослідження показників хворого. Включає збереження встановленого діагнозу, плану обстеження та плану подальшого лікування, а також статус електронного підпису медичного фахівця.

Останньою групою є сутності аналітичного рівня з яким працює сервіси роботи з штучним інтелектом:

– medication: виступає в ролі централізованого довідника лікарських препаратів. Містить інформацію про назву, форму випуску, дозування та виробника препаратів.

– prescription: спеціалізований реєстр призначений ліків. Пов’язує сутності лікаря та пацієнта призначенням. Фіксує препарат та правила його прийому, дозування та період прийому.

– medicine_intakes: журнал фактичного прийому ліків пацієнтом. Звіряє фактичний запланований час прийому з реальним часом прийому. Має статус підтвердження інформації для відображення для пацієнта та окремо для лікаря.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

Розробки та тестування програмного забезпечення являється фінальним етапом втілення переходу до конкретного результату. Це сама ресурсномістка і тривала частина проєкта, де обов'язково важливо притримуватися наміченого плану для досягнення очікуваного результату. Успіх на цьому етапі залежить від того, наскільки якісно проведена підготовча робота та складено план.

Цей розділ описує процеси програмної реалізації системи, охоплюючи усі важливі етапи: починаючи з організації середи розробки та конфігуруванні проєкту закінчуючи демонстрації роботи важливих екранних форм системи. Розглядається особливості налаштуванні інструментів контейнеризації, розробка ключових компонентів фронтенда та бекенда, а також методи забезпечення безпечного входу в систему. Особливу увагу приділено перевірці роботи та тестуванні для забезпечення надійності та стабільності роботи застосунку.

4.1 Організація та конфігурування проєкту

Проєкт є модальним, що забезпечує гнучкість та можливість легкого розширення. Кожен розроблений модуль системи має свою мету. Кожен розроблений елемент системи має свою чітко визначену функцію. Такий підхід розділення відповідальності дозволяє корегувати зовнішній вигляд або серверну логіку застосунку, не ризикуючи пошкодити систему в цілому. Структура спрощує процес проведення тестування, так як кожний компонент перевіряється окремо один від одного, зменшуючи проявів помилок та зменшити час для їх вирішення. Завдяки такому підходу розмежовуванні продукту на рівні, масштабування системи відбувається без змін існуючого.

Розробка базується на моделі, де частини для користувацького інтерфейсу та серверної обробки працюють незалежно, але зв'язок відбувається за допомогою спеціального інтерфейсу RESTful API. Це означає, якщо в майбутньому виникне

потреба в розробці мобільного застосунку, достатньо буде розробити лише новий клієнтський інтерфейс, а основна серверна логіка залишиться без змін. Основний акцент при розробці нового продукту спрямований на відтворення описаної концепції, яка одночасно зможе опрацювати багато користувачів і обробляти медичні запити.

4.1.1 Структура директорій бекенду

Серверна частина розроблена з дотриманням принципів багатошарової архітектури, що є основою для створення надійних корпоративних систем. Кожен рівень бекенд-застосунку має чітке розмежовування відповідальності, що дозволяє відокремити складну логіку аналізу медичних показників від технічних аспектів зберігання інформації в базах даних. Перевага такого підходу полягає в тому, що кожний шар не залежить від інших і спрощується редагування бізнес-логіки без додаткового втручання в інші.

Така декомпозиція проєкту на окремі пакети забезпечує високе тестування системи. Таким чином кожен сервіс або репозиторій може бути перевірений окремо. Використання шаблону DTO гарантує безпечне транспортування даних запобігаючи прямого доступу внутрішніх структур бази даних до зовнішнього джерела. Використання подібного підходу гнучко формує відповіді від сервера, передаючи клієнту лише ті, які були задумані у контексті передачі. Логічна структура серверної частини представлена такими пакетами:

– entity: містить класи-моделі, які відповідають таблиці бази даних і описують сутність структури зберігання інформації про користувачів, вимірювання та призначень;

– repository: реалізує рівень доступу до даних за допомогою інтерфейсів фреймворку Spring Data JPA, що забезпечує виконувати операції створення, читання, оновлення та видалення, а також складати запити без необхідності написання низькорівневого коду SQL;

– service: виступає централізованим рівнем системи, де зосереджена уся бізнес-логіка, алгоритми обробки медичних показників та інтеграція зі штучним інтелектом через зовнішні хмарні сервіси;

– controller: визначає усі REST ендпоінти застосунку, приймає вхідні запити від клієнтської частини, перевіряючи авторизацію та переадресовуючи відповідні завдання на сервіси;

– dto: використовується для ефективного обміну даних між сервером і клієнтом, приховуючи чутливі дані, які не потрібно передавати;

– mapper: автоматизує перетворення даних між різними об’єктами, що покращує читабельність коду;

– config: містить загальні налаштування проєкту, такі як конфігурація безпеки, права доступу до ресурсів та параметри автентифікації сервісу Keycloak.

4.1.2 Структура директорій фронтенду

Для створення інтерфейсу користувача було застосовано підхід, заснований на компонентах. Це дозволяє розбити складні графічні елементи на окремі, незалежні функціональні блоки. Організація файлової системи таким чином спрямована на досягнення максимальної оптимізації відображення інформації та полегшення супроводу коду.

Основний акцент зосереджений на розмежовуванні логіки роботи системи від шару візуального представлення. Такий підхід спрямований на масштабування проєкту, додаючи нові можливості без ризиків порушення вже існуючого. Організація коду такого підходу покращує читабельність коду проєкту. У межах проєкту розмежовано такі основні директорії:

– pages: містить компоненти-контейнери, які відповідають за окремі сторінки програм;

– components: зберігає набір самостійних елементів інтерфейсу, які можна комбінувати та використовувати повторно;

– api: об’єднує логіку взаємодії з бекендом, де містяться всі запити, обробка статусів відповідей та управління токенами доступу;

– types: містить описи інтерфейсів та типів даних, що забезпечує строгу типизацію та відповідальність структур даних тим об’єктам, які приходять від серверної частини у форматі DTO;

– hooks: містить сценарії для управління станом компонентів.

4.1.3 Застосування системи контролю версій

Для цілісності роботи сервісу застосовано систему контролю версій Git. Цей інструмент був обраний завдяки його гнучкості, можливості створення незалежних середовищ для розробки окремих функцій. У проєкті було дотримано спрощеної моделі, що забезпечує стабільність основної гілки коду та дозволяє одночасно інтегрувати нові технології, наприклад штучний інтелект. Структура розгалуження організована наступним чином:

– гілка main: призначена для зберігання перевіреного коду готового до запуска. Будь-які зміни в цій гілці заборонені для підтримки стабільності системи;

– гілка develop: слугує центральним вузлом для інтеграції всіх нових розроблених функцій. Саме тут відбувається первинне тестування взаємодії між клієнтською і серверною частиною.

Особлива увага приділяється веденні комітів та захисту репозиторію. Для підвищення зрозумілість введення історії використовувався стандарт Conventional Commits, де кожний коміт має чіткий префікс, що вказує на його тип (наприклад feat: для нових функцій, fix: для виправлень, refactor: для оптимізації). Крім того для захисту вразливих файлів з конфіденційною інформацією налаштовувано файл .gitignore. Він включає всі конфігураційні файли з ключами доступу до хмарних сервісів та паролями. Такий підхід запобігає потраплянню секретних даних у відкриті сховища коду.

4.1.4 Ізоляція та контейнеризація

Для уникнення проблем, пов’язаних з відмінностями в налаштуванні локальних середовищ, використано технологію контейнеризації Docker. Підхід дозволяє запакувати кожний компонент разом з усіма необхідними бібліотеками, залежностями та конфігураційними файлами в окремі ізольовані контейнери. Це гарантує, що система буде працювати однаково на різних етапах розробки і дозволить швидко масштабувати окремі частини платформи при необхідності.

Для управління цими контейнерами та їх взаємодії використано Docker Compose. Весь опис зберігається в одному файлі налаштуванні у зрозумілому форматі. Це дозволяє запустити всю систему, включаючи: базу даних, сервер авторизації, та інші сервіси однією командою. Такий підхід спрощує початок роботи і відповідає принципу інфраструктура як код. В рамках проєкту Docker Compose розгортає наступні сервіси:

– база даних PostgreSQL: слугує центральним сховищем для всій системи. Використовує ізольований том, для того щоб дані зберігалися окремо від контейнера. Тому при оновлені образу бази даних чи перезапуску системи вся інформація залишається недоторканою;

– сервер ідентифікації Keycloak: відповідає за безпеку системи, керуючи процесами перевірки особи та надання доступу. У межах єдиної мережі взаємодіє з базою даних для зберігання інформації про користувачів та налаштувань безпеки, що підвищує захист від несанкціонованого доступу до системи;

– бекенд системи: центр, що керує усіма бізнес-процесами. Автоматично визначає мережеві адреси інших контейнерів. Забезпечує безпечний вихід в мережу для взаємодії з сторонніми сервісами.

– фронтенд системи: відповідає за інтерфейс користувача та взаємодію з системою. Забезпечує стабільну доставку статичних файлів та перенаправляє запити користувача до бекенд контейнера. Ізоляція у власному контейнері запобігає конфліктам з іншими сервісами. Налаштування порту 3000 або 80 дозволяє лікарям та пацієнтам отримувати доступ до інтерфейсу через веббраузер.

Окремо варто зазначити про налаштування мережі для контейнерів. Усі компоненти системи інтегровані в єдину віртуальну мережу. Це означає, що вони можуть проводити комунікацію за допомогою імен сервісів, усуваючи необхідності в використанні динамічних IP-адресів. Наприклад, бекенд може легко знайти базу даних за адресом db:5432. Такий підхід робить систему безпечнішою, адже компоненти залишаються прихованими від зовнішнього доступу, доступні лише всередині цієї мережі.

4.2 Програмне втілення серверної бізнес-логіки

Після налаштування інфраструктури та організації структури файлів необхідною умовою є створення надійної бізнес-логіки проєкту. Серверна частина застосунка відповідає за повний цикл обробки медичних даних: від отримання показників з клієнтського інтерфейсу до аналітичних механізмів обробки даних. Алгоритми було реалізовано, щоб усі бізнес-правила були реалізовані в окремому сервісному шарі. Так забезпечується надійність даних та зберігається їх цілісність, навіть якщо до системи підключається багато користувачів.

4.2.1 Процес прийому та перевірки медичних даних

Основний потік даних у застосунку пов’язаний з обробкою та зберіганням показників артеріального тиску пацієнтів. Цей функціонал реалізовано в сервісі MeasurementService. Він діє посередником між запитами, що надходять з REST Api та базою даних. Коли пацієнт додає або редагує показники артеріального тиску, система запускає багатоступеневу обробку.

Спочатку проводиться первинна обробка через DTO за допомогою стандартних інструментів валідації, що пропонує Bean Validation API. Це дозволяє відсортувати некоректні запити, ще до того як вони попадуть з JSON. Проводиться порівняння отриманих значень тиску та пульсу з допустимими фізіологічними межами. Таким чином унеможливлюється внесення помилкових даних, наприклад, нульового пульсу. Якщо всі перевірки пройдені, дані з DTO конвертуються в об’єкт

сутності MeasurementEntity за допомогою бібліотеки MapStruct. Наостанок дані зберігаються у базі даних PostgreSQL через MeasurementRepository. Ця операція завдяки анотації @Transactional гарантує, що дані будуть збережені повністю, або не будуть збережені взагалі, забезпечуючи цілісність зв'язку між вимірюванням та профілем пацієнта.

4.2.2 Інтеграція сервісу штучного інтелекту

Найбільш складним технічним компонентом серверної частини є модуль AIService. Він є зв'язковим модулем системи з хмарними моделями штучного інтелекту через хмарну платформу OpenRouter. Робота сервісу базується на гнучкому формуванні контексту. Коли лікар ініціює аналіз, сервіс збирає та оброблює дані за певний період, структурує їх за датою та використовує як основу для запиту до хмарного сервісу штучного навчання.

Важливою частиною реалізації є техніка Prompt Engineering. AIService використовує шаблон, що визначає роль штучного інтелекту в запиті як медичного помічника і динамічно вставляє в нього дані в форматі JSON. Сформований запит надсилається до API OpenRouter. Отримана відповідь обробляється для видалення зайвої інформації та структуруються для зручного відображення. Це дозволяє системі надавати не просто числові дані, а аналітичні висновки щодо стану пацієнта та потенційних ризиків стану.

4.2.3 Захист медичних даних

Зважаючи на надзвичайну чутливість медичної інформації, у застосунку застосовані жорсткі заходи для ізоляції даних на рівні окремих функцій. Простої перевірки наявності авторизаційного токену JWT недостатньо для забезпечення надійного рівня безпеки. Тому в сервісному рівні реалізовано принцип власника ресурсу, який гарантує, що доступ мають лише авторизовані особи. Кожна функція, яка працює з історією вимірювань або іншими персональними даними, перевіряє, чи автор запиту є власником цих даних.

Технічно унікальний ідентифікатор користувача витягується з контексту безпеки, який заповнюється після успішної перевірки токена в Spring Security. Потім порівнюється з ідентифікатором пацієнта patient_id або лікаря doctor_id у базі даних. Якщо пацієнт намагається отримати доступ до даних іншого пацієнта, або лікар до пацієнтів, які за ним не закріплені система відправить помилку заборони доступу і зупинить операцію. Такий метод забезпечить надійний захист від несанкціонованого доступу всередині системи.

4.2.4 Логіка призначення та моніторингу прийому ліків

В розроблюваній платформі присутній модуль управління лікуванням медикаментами**.** Модуль відтворює безперервний зв’язок між лікарем та пацієнтом прийому ліків. Реалізовано через сервіс PrescriptionService, який повністю контролює процес лікування. Коли лікар виписує новий список прийому, система автоматично створює в базі даних список завдань для пацієнта під назвою MedicineIntake. Ці завдання формують індивідуальний графік прийому ліків на весь період лікування. Це автоматично відстежуває, чи приймає ліки пацієнт і готує дані для аналітики, позбавляючи від необхідності лікарю створювати нагадування для кожного прийому.

Пацієнт зі свого боку підтверджує, що прийняв ліки через інтерактивну функцію. Кожне підтвердження змінює статус відповідного запису про прийом ліків і записує точний час події. Це дозволяє системі в реальному часі оцінювати, наскільки пацієнт дотримується призначень. Лікар отримує реальну інформацію про процес лікування позитивно впливаючи на лікування та коригування терапії.

4.3 Реалізація користувацького інтерфейсу

Клієнтська частина системи реалізована односторінковим вебзастосунком SPA. Використання цієї архітектури дозволяє уникнути перевантаження сторінки при навігації між іншими сторінками. Таким чином оптимізується робота клієнтської частини застосунку. Основна частина обробки даних та формування

інтерфейсу проходить на стороні браузера користувача. Крім того, особлива увага приділена безпеці навігації, автоматизації взаємодії з сервером шляхом використання протоколу передачі даних HTTP та відображені медичної інформації.

4.3.1 Механізми захищених переходів

Для навігації використовується бібліотека react-router-dom. Однак, для медичної системи стандартних можливостей маршрутизації виявилося недостатнім. Щоб створити приватні розділи, був розроблений спеціальний компонент-обгортка. Цей компонент діє як контролер доступу для захищених сторінок. Перед тим як показати якийсь захищений маршрут, компонент перевіряє глобальний стан системи. Він шукає JWT токен і перевіряє чи роль користувача відповідає дійсності відповідно його ролі в системі Keycloak.

Якщо хтось намагатиметься отримати доступ без належний прав доступу система автоматично перенаправить користувача на сторінку входу /login, запам’ятавши минулий запитуваний шлях. Такий архітектурний підхід надає доступ пацієнтів та лікарів на рівні навігації. Це гарантує, що користувач отримає лише ті можливості, які йому дозволені відповідно до його ролі.

4.3.2 Автоматизація роботи з API

Для спрощення розробки було інтегровано механізм перехоплення запитів Axios Interceptors. Це дозволило автоматично додавати необхідний токен авторизації до кожного HTTP-запиту до бекенду, усуваючи потребу ручного додавання до кожного виклику, прибираючи випадок коли окремі запити надсилаються без необхідних заголовків авторизації.

Крім того, перехоплювач ефективно обробляють відповіді серверу. При отриманні помилки авторизації (статус 401 Unauthorized), що означає про закінчення терміну дії токену, система автоматично очищає локальне сховище та виходить з облікового запису користувача. Це запобігає подальшим запитам з недійсними даними та захищає від потенційних вразливостей. Також перехоплювач оброблює і інші помилки такі як: відсутність відповіді сервера чи

внутрішньої помилки бізнес-логіки формуючи користувачу зрозумілі відповіді замість технічного відображення помилки.

4.3.3 Відображення медичних графіків та аналітичних інструментів

Ключовим етапом інтерфейсу є функціонал, який дозволяє візуалізувати аналітичні дані. Переведення числової інформації на візуальний графік починається з підготовки даних. На цьому етапі фронтенд впорядковує записи, отримані від серверу за часом та стандартизує часові позначки, для коректного відображення на графіку. Для побудови використовується бібліотека Recharts. Приклад відображення графіку відображено на рисунку 4.1.

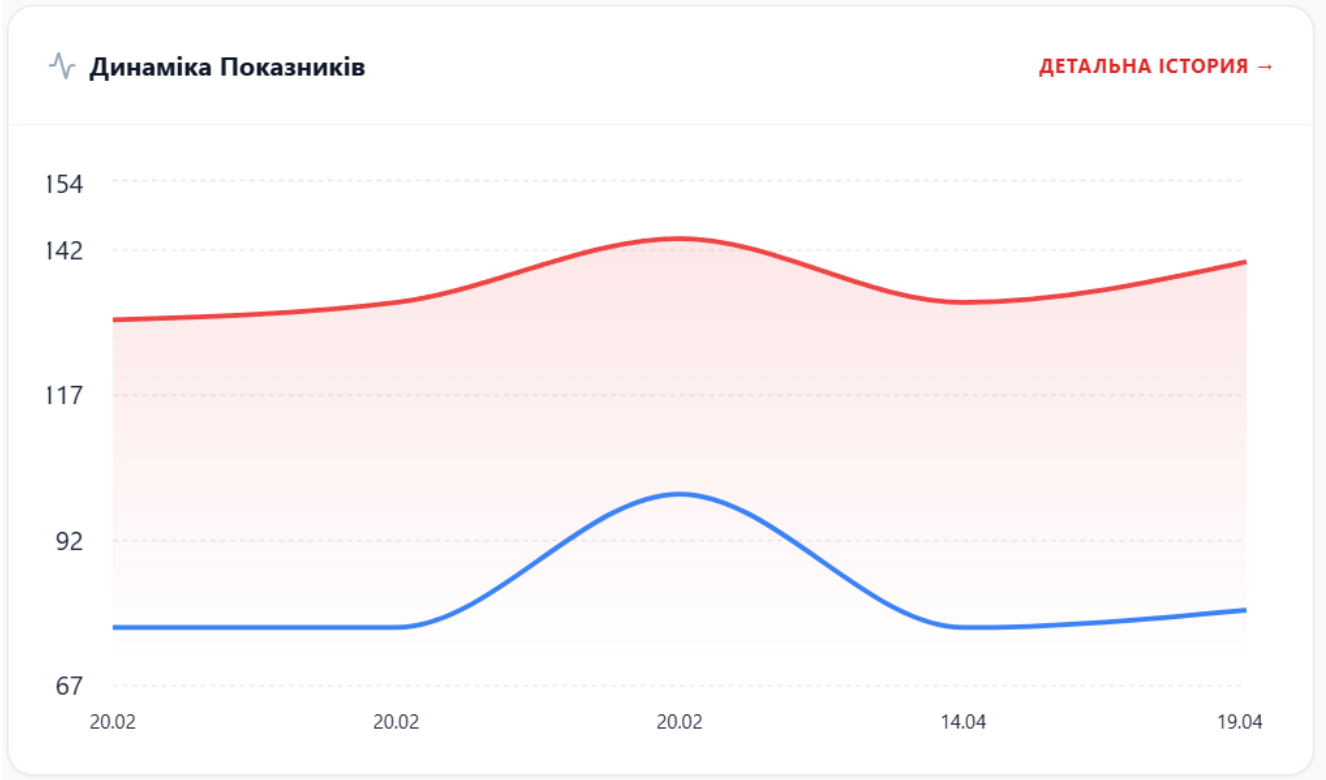


Рис. 4.1 Приклад відображення графіку

Алгоритми побудови графіка налаштовані на одночасне відображення двох окремих ліній, що представляють систолічний та діастолічний тиск. Це надає швидкість оцінки діапазону коливань показників та виявляти небезпечні стрибки або кризи. Графік підтримує інтерактивні підказки, які з’являються при наведенні курсору. Вони дозволяють побачити точні значення пульсу та часу вимірювання

для кожної точки даних, перетворюючи суху статистику на цінний інструмент для клінічного аналізу.

4.3.4 Побудова єдиної дизайн-системи

Для забезпечення єдиного та тематичного вигляду проєкту було запроваджено централізований механізм для дизайну системи. В основі механізму CSS-змінні, що дозволило відмовитися від хаотичного стилізуванні окремих компонентів. Створено єдину універсальну палітру кольорів, які застосовувалися з чітким призначенням – створення довіри для користувача. Це забезпечило єдиний стиль усіх компонентів та прискорило відображення нових компонентів через миттєве підхоплення вже існуючих стилів.

Для покращення ергономіки роботи медичного персоналу було реалізовано механізм зміни тем оформлення з світлого на темний. Враховуючи специфіку роботи фахівців, зокрема в нічну зміну та умов недостатнього освітлення, темна тема була впроваджена для запобігання зниження зорової напруги. Технічна реалізація передбачає зміну значень CSS змінних. Вибір користувача зберігається в localStorage та автоматично застосовує обрану тему при наступному вході в систему, що забезпечує персоналізований та користувацький досвід незалежно від часу доби.

4.4 Тестування компонентів

Для перевірки системи на коректність роботи було запроваджено систему тестування. Система охоплює ключові рівні платформи, а особливий акцент було зроблено на перевірці надійності алгоритмів звертання до хмарних систем у різних ситуаціях. Для проведення тестування було використано юніт-тестування для окремих компонентів для забезпечення стабільності системи. Крім того, було використано мок-заглушки при тестуванні зовнішніх залежностей. Такий підхід є важливим для стабільності роботи програмного забезпечення, призначеного для медичної сфери.

4.4.1 Тестування бізнес-моделей

На етапі перевірки окремих модулів головним завданням стало забезпечення цілісності роботи бізнес-логіки у сервісному рівні системи. Використання ізольованих тестів дозволило переконатися в коректності роботи системи без змін у базі даних. За допомогою інструментів Junit 5 та бібліотеки Mockito було створено імітаційні заглушок для тестування таких функціональних блоків:

– оцінка реакції системи на спробу введення фізіологічно неможливих показників тиску та пульсу;

– перевірка логіки формування графіків прийому ліків і автоматичного розрахунку відсотка дотримання пацієнтом призначень;

– контроль запобігання несанкціонованого доступу до медичної інформації з боку сторонніх осіб, які не мають на це відповідних дозволів.

4.4.2 Тестування REST API

Наступна стадія тестування полягала в перевірці ендпоінтів REST API серверної частини застосунку. Для проведення тестування було застосовано застосунок Postman, який дозволяє надсилати HTTP-запити до сервера і видавати відповідь. Результати тестування показали позитивний результат роботи серверної частини з фронтендом. У ході тестування було проведено:

– перевірку на коректність отримання полів JSON-об’єктів для усіх ендпоінтів;

– перевірка токенів JWT на авторизацію та блокування запитів з простроченим терміном токена;

– верифікація прав доступу до функціоналу різних ролей.

У висновку проведеного тестування, усі ендпоінти пройшли перевірку і повернули очікуваний результат за відповіддю. Перевірка на розмежування по ролям показала, що інші ролі не мають доступу і виконують функціонал належним чином. Усіма іншими функціями, ненаділеними ролі користувача застосунок повертає відповідь блокуванням.

4.4.3 Тестування інтерфейсу та клієнтської логіки

Останнім етапом перевірки є тестування клієнтської частини застосунку для забезпечення стабільності взаємодії з користувачем системи. Перевірка була зосереджена на обробці статистичних даних та подальшої візуалізації медичної інформації, що в результаті показала високу чутливість інтерфейсу та відсутність критичних помилок при взаємодії з фальшивими показниками. В межах тестування було виконано:

– перевірка React компонентів на правильність візуалізації при отриманні різних типів даних;

– ручне та автоматизоване тестування форм введення показників на предмет фізіологічних відхилень;

– перевірка на відповідність точок графіку з реальними значеннями з бази даних;

– перевірка цілісності інтерфейсу у різних браузерах та мобільних пристроїв.

Результати тестування продемонстрували коректність роботи усіх компонентів системи. Стабільність поведінки показують очікуваних значень. Таким чином, тестування засвідчило готовність клієнтської частини до використання в медичному середовищі.

4.5 Демонстрація застосунку

Як результат розробки програмного застосунку для моніторингу артеріального тиску було створено комплексну інформаційну систему, яка надає перевагу для переходу на новий рівень медицини. Для демонстрації роботи застосунку було поділено на основні ролі та обрано ключові екранні форми, які наочно описують та показують розроблений функціонал. Кожна з ролей отримала свій персональній кабінет для зручного керування набором існуючих інструментів. Підхід для демонстрації надає оцінку взаємодії між усіма користувачами, та наділяє можливість відстежити повну картину графічної частини застосунку.

4.5.1 Опис екранних форм адміністратора

Після входу в систему адміністратор потрапляє на головну сторінку для керування та оглядом за системою. Сторінка містить статистичні дані про загальну кількість пацієнтів, навантаження на систему та використану пам’ять серверу. Саме ця сторінка відповідає за загальний стан роботи системи.

У верхній частині панелі розміщено 4 основні інформаційні картки з ключовими показниками, такими як: загальна кількість зареєстрованих лікарів, загальна кількість створений карток пацієнтів, поточне навантаження на процесор сервера та загальний відсоток використання оперативної пам’яті комп’ютера. Оновлення показників цих карток відбувається в реальному часі. У центральній частині відображено блок з швидким доступом до такого функціоналу: реєстрація нового лікаря, навігація до сторінки довідки для адміністраторів. Наостанок, розміщено журнал подій для відстеження критичних подій пов’язані з станом користувачів у системі. На рисунку 4.2 продемонстровано екранна форма адміністративної панелі.

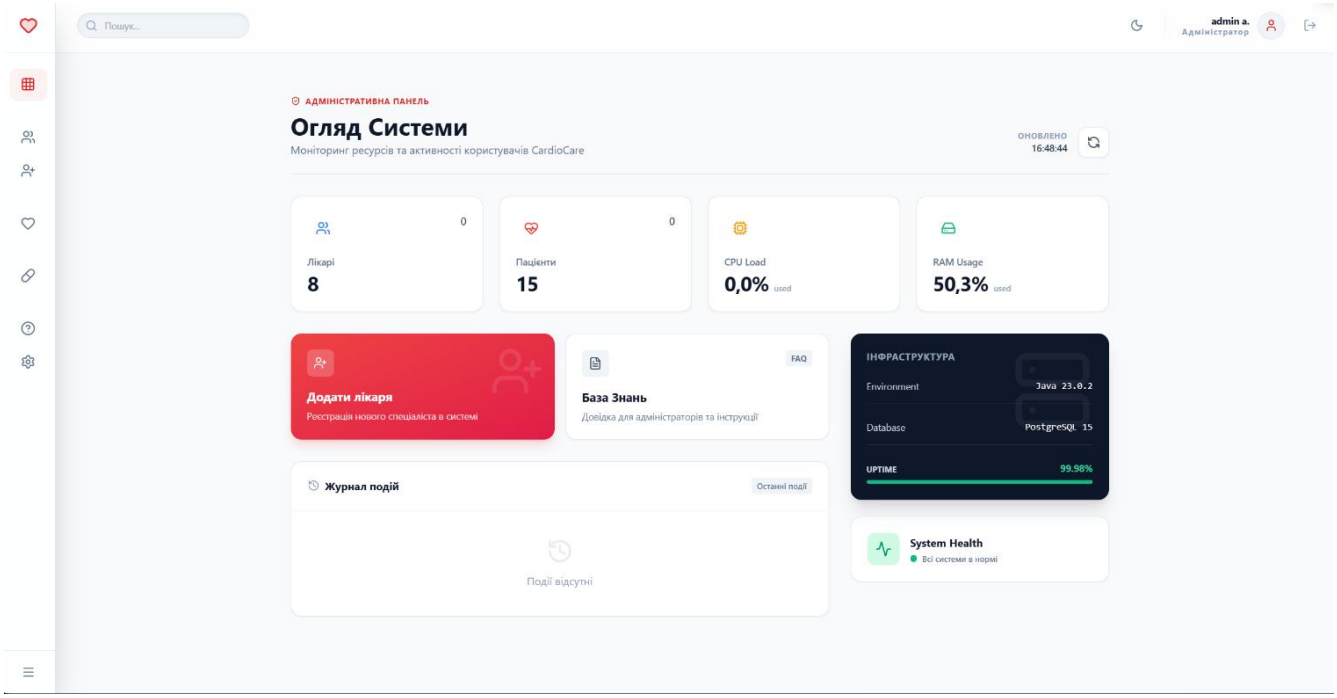


Рис. 4.2 Екранна форма адміністративної панелі

Наступною ключовою функцією є додавання нового лікаря. Ця можливість окремо надана ролі адміністратора для відтворення архітектурного підходу для

захисту системи. Це гарантує цілісність даних, так як лікар або пацієнт не мають доступ до керування свої обліковим записом. При створенні нового облікового запису у системі автоматично створюється новий лікар, якому надається права доступу, пов’язані з керування медичних карток пацієнтів. Усі поля форми наділені валідацією, що підтверджує що було введено коректні дані у потрібному форматі перед відправкою на сервер застосунку. Крім того, уся надана інформація використовується для відображені у платформі, а також для підвищення успіху відправки запиту до хмарного сервісів штучного інтелекту.

Візуальне відображення створення нового облікового запису лікаря продемонстровано на рисунку 4.3.

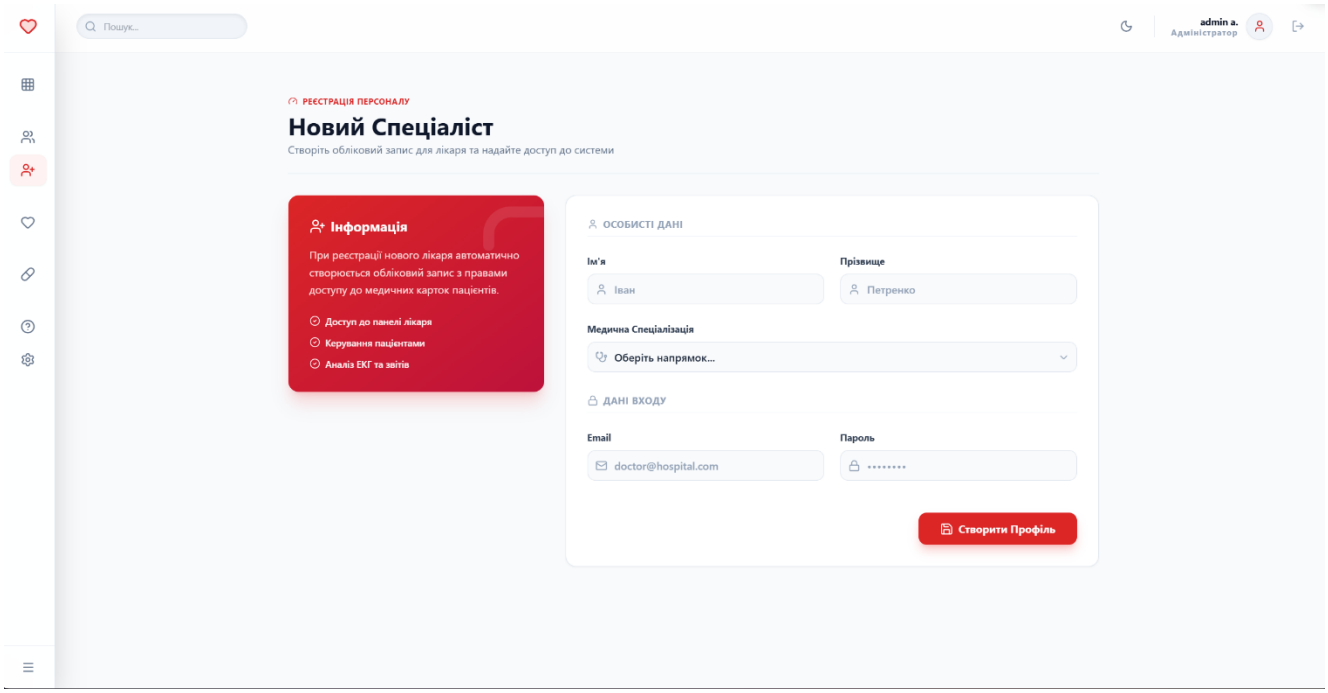


Рис. 4.3 Створення нового облікового запису лікаря

Крім того, адміністратору надано функціонал, дозволяючий керувати загальною базою препаратів у системі. Саме від цього списку залежить, які ліки будуть назначені пацієнту для останнього етапу лікування. Візуально сторінка представлена у інтерактивної табличній формі, що дозволяє швидко шукати та керувати певним препаратом. Для пошуку достатньо почати введення назви препарату, після чого уся збігла текстова інформація буде відображена у таблиці. Для додавання нового медичного препарату передбачено заповнити інформацію

про назву лікарського засобу, форму та дозування, а також короткий, але інформативний опис. Приклад інтерфейсу бази медичних препаратів представлено на рисунку 4.4.

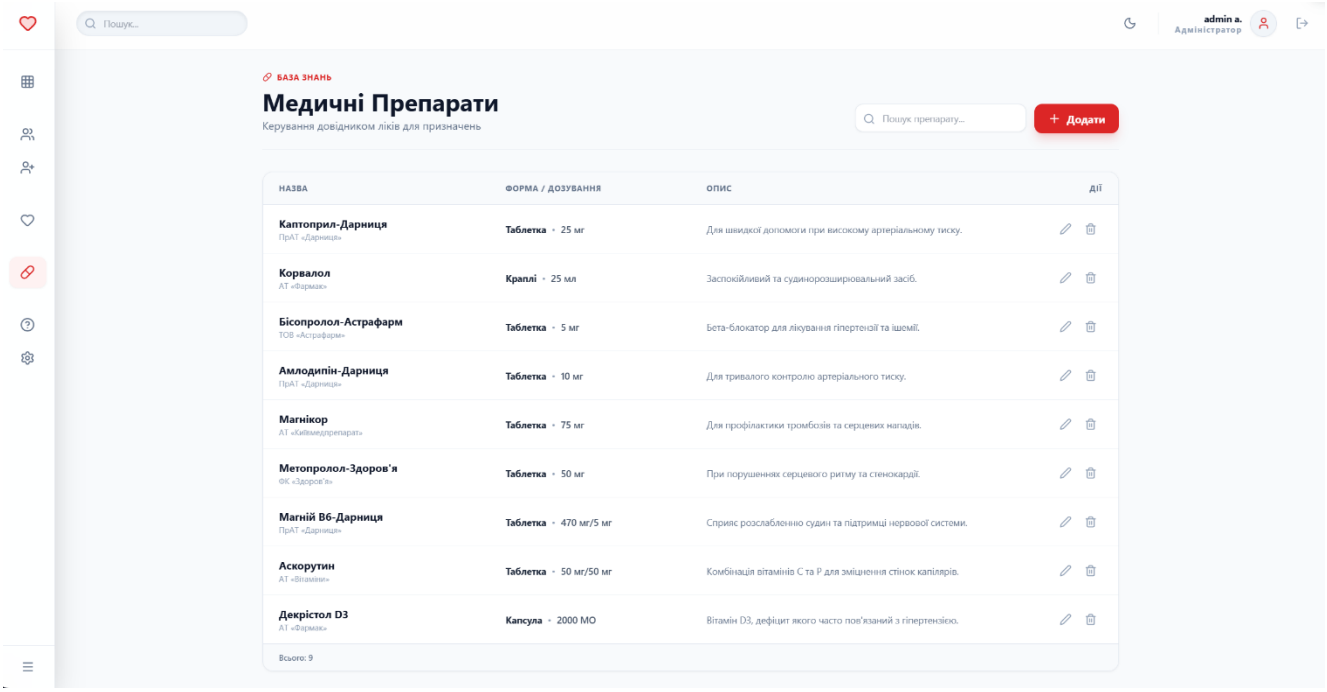


Рис. 4.4 Інтерфейс бази медичних препаратів

4.5.2 Опис екранних форм пацієнта

Після успішного входу в систему, пацієнт потрапляє на головну сторінку, де відображено ключову узагальнену інформацію, щодо лікування. Графічне відображення спроєктовано за принципом Dashboard first. Пацієнт одразу бачить поточний стан без необхідності переходити в інші сторінки платформи. Це спрощує користування системою для користувачів з низьким рівнем цифрової грамотності.

Основні продемонстровані блоки показують: останній доданий тиск, пульс пацієнта, також кількість висновків розроблений лікарем та відсоток прийому ліків. Поміж цього, присутній узагальнений графік останніх доданих показників артеріального тиску і можливість перейти до детального перегляду історії. Останній блок присвячений нагадуванню регулярності прийому назначених медичних засобів і своєчасного вимірювання тиску. На рисунку 4.5 продемонстровано головний екран пацієнтів.

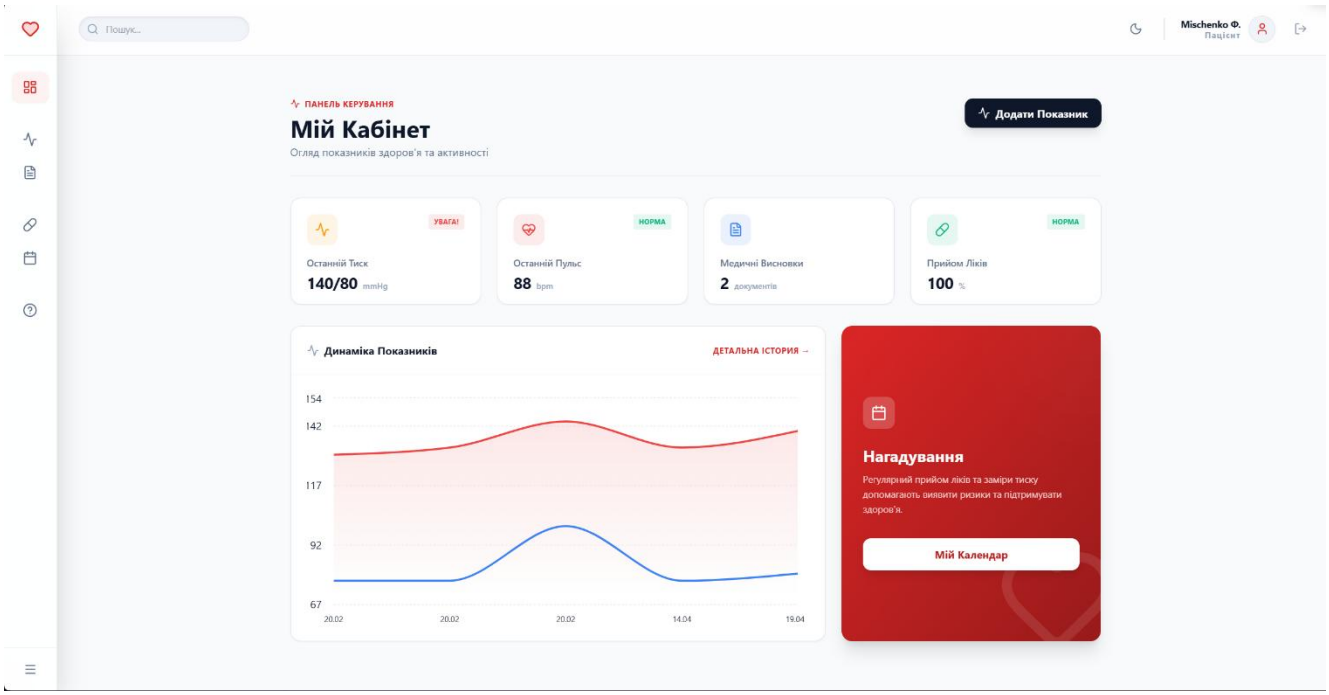


Рис. 4.5 Головний екран панелі керування кабінету пацієнта

Головним завданням пацієнта при лікуванні артеріальних захворювань є відслідковування та додавання в систему нових показників тиску. Для цього було створено екранну форму, яка відповідає за перегляд та керування таких показників. Це основний інструмент для формування єдиного списку між пацієнтом та лікарем, яка використовуватиметься для аналізу та призначені наступного етапу лікування.

Інтерфейс побудований за принципом табличного журналу, що надає зручність при перегляді історії замірювань. Кожний пункт списку синхронізують з базою даних серверу платформи та виступають у ролі об’єктів. Для додавання нового достатньо заповнити інформацію про систолічний та діастолічний тиск, а також про пульс пацієнта. Уся інша інформація по типу дати та часу створення нового пункту проставляється автоматично системою. Крім того, наявний швидке фільтрування для необхідного сортування даних.

Візуальне відображення щоденнику показників пацієнта зображено на рисунку 4.6.

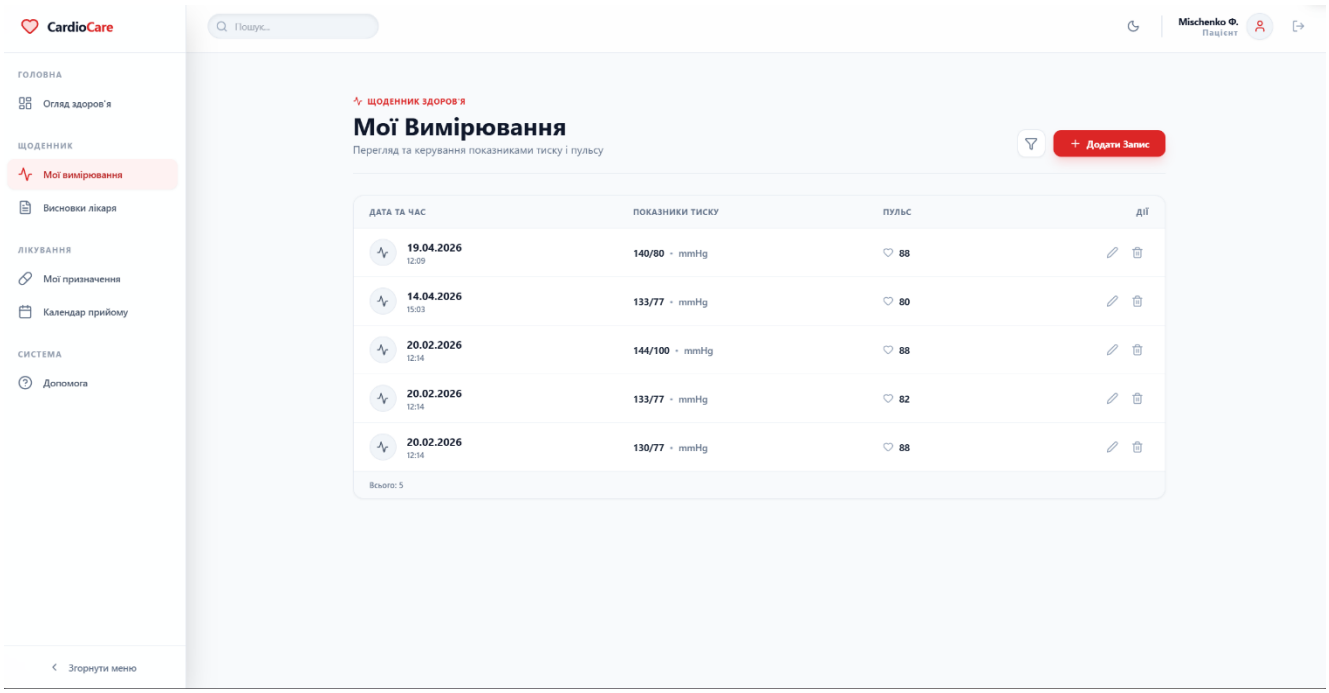


Рис. 4.6 Щоденник вимірювань тиску та пульсу пацієнтів

Форма перегляду медичного висновку є однією з головних екранних форм, що показує клінічний висновок зроблений лікарем. Зазвичай висновок розробляється при першої консультації з медичним фахівцем, або при повторному дообстеженні за умовою тривалого фіксування показників артеріального тиску. Ця сторінка слугує офіційним документом, замінюючи паперові довідки.

Екран реалізовано у форматі спливаючого модального вікна, що дозволяє швидко ознайомитися з рекомендаціями лікаря. Екран поділено на основні 4 параметри, що дозволяють розробити вигляд висновка структурованим. Верхня частина містить ідентифіковану інформацію про лікуючого спеціаліста. Наступним блоком є лаконічно сформований діагноз, згідно зафіксованих показників наданих пацієнтом, що виступили для обґрунтуванні клінічного рішення. План обстеження містить інформацію про подальші додаткові дії для проведення детального дообстеження.

На останок зазначена інформація про план лікування, призначення медичних препаратів. Усе це автоматично фіксується датою та електронним підписом лікаря для закріплення медичного висновка юридичною силою.

Приклад інтерфейсу бази медичних препаратів представлено на рисунку 4.7.

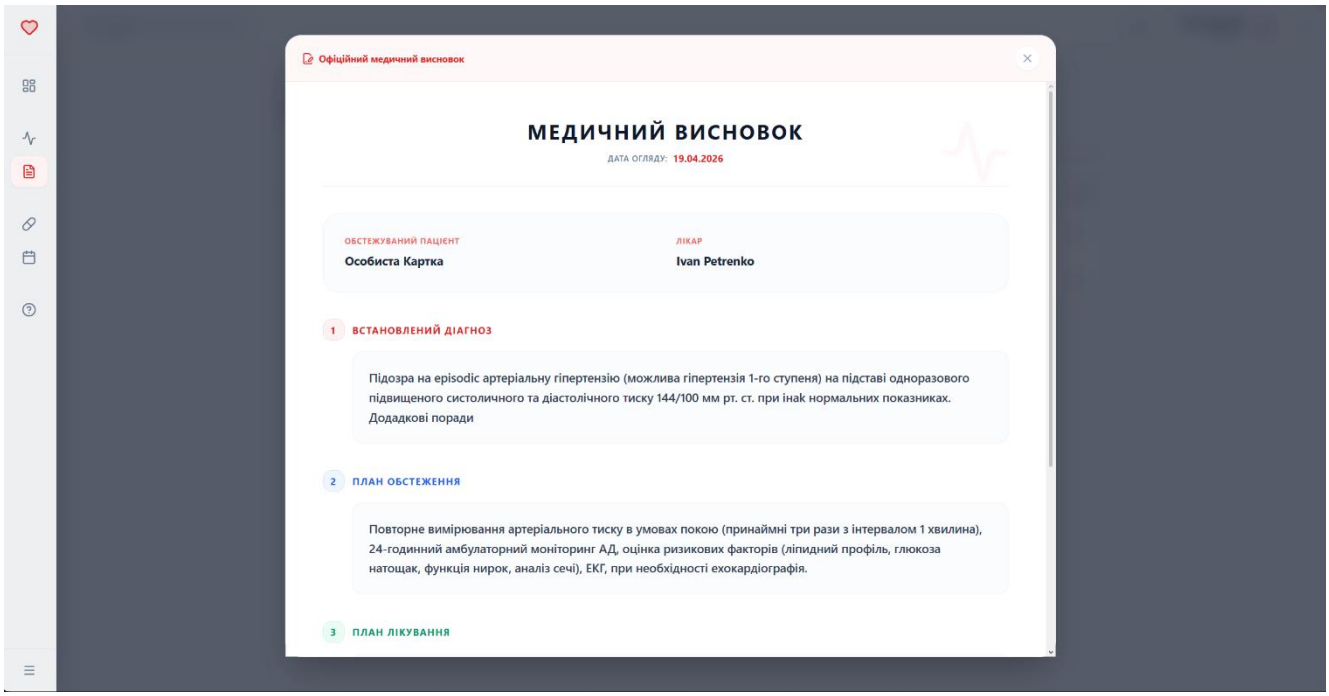


Рис. 4.7 Відображення медичного висновку для пацієнта

4.5.3 Опис екранних форм лікаря

Увійшовши до системи, лікар опиняється на головній сторінці панелі керування лікаря. Сторінка відображає усю важливу інформацію для швидкого огляду активності притримуванні лікування пацієнтів та за їх станами. Дизайн реалізовано за принципом модульності. Тому верхня і нижня частина відповідає за відображення ключових метрик з урахуванням моніторингового стану за пацієнтами.

З головних показників на сторінці відображено: закріплені за лікарем пацієнти, кількість пацієнтів з високим тиском для інформування пацієнтів із критичними випадками стану, кількість пацієнтів, що очікують на проведення аналізу та статистика прийомів за день. Крім того, з сторінки легко переглядати останніх доданих пацієнтів, що забезпечує швидкий перехід до персональних карток користувачів, що нещодавно вносили нові зміни у систему.

Така дизайнерська структура екранної форми надає реалізувати ефективний розподіл часу для лікаря і сфокусувати його увагу на більш потребуючі функції. На рисунку 4.8 зображено приклад екранної форми панелі керування лікаря за пацієнтами.

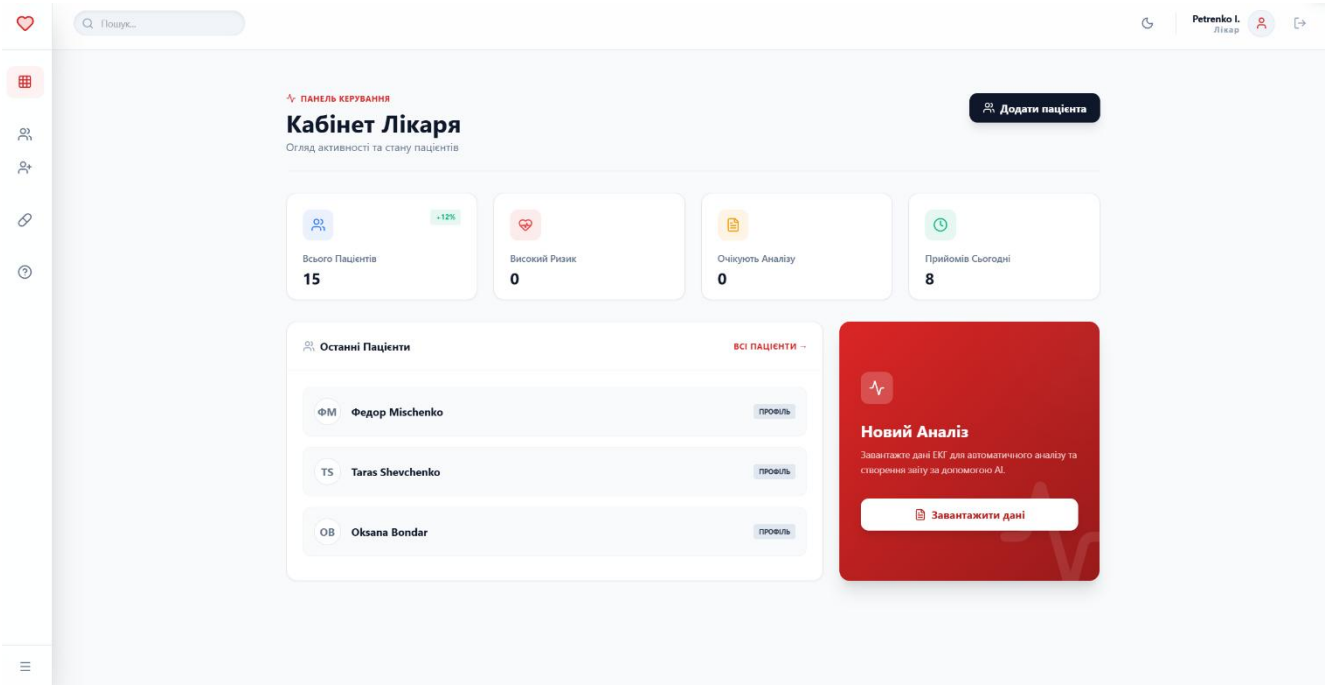


Рис. 4.8 Панель керування лікаря за пацієнтами

Лікарю наділена функціональна вимога для створення нових облікових кабінетів пацієнтів з ціллю зменшення введення помилкових значень та відсутності необхідності проведення верифікації особи. Усі поля форми мають особливе значення, так як для збільшення ймовірності відтворення правильного висновку лікаря недостатньо даних про артеріальний тиск.

Тому було додано поля дати народження, ваги за зросту. Це надає можливість для сервісу штучного інтелекту збільшити точність розрахунку шляхом введення індивідуальних показників окрім таких, які можуть впізнати пацієнта.

Для форми було реалізовано валідацію для перевірки полів форм на правильність заповнення клінічних показників. Таким чином це дозволяє запобігти заповненням бази даних серверної частини застосунку від некоректних даних.

Приклад форми реєстрації нових пацієнтів проілюстровано на рисунку 4.9.

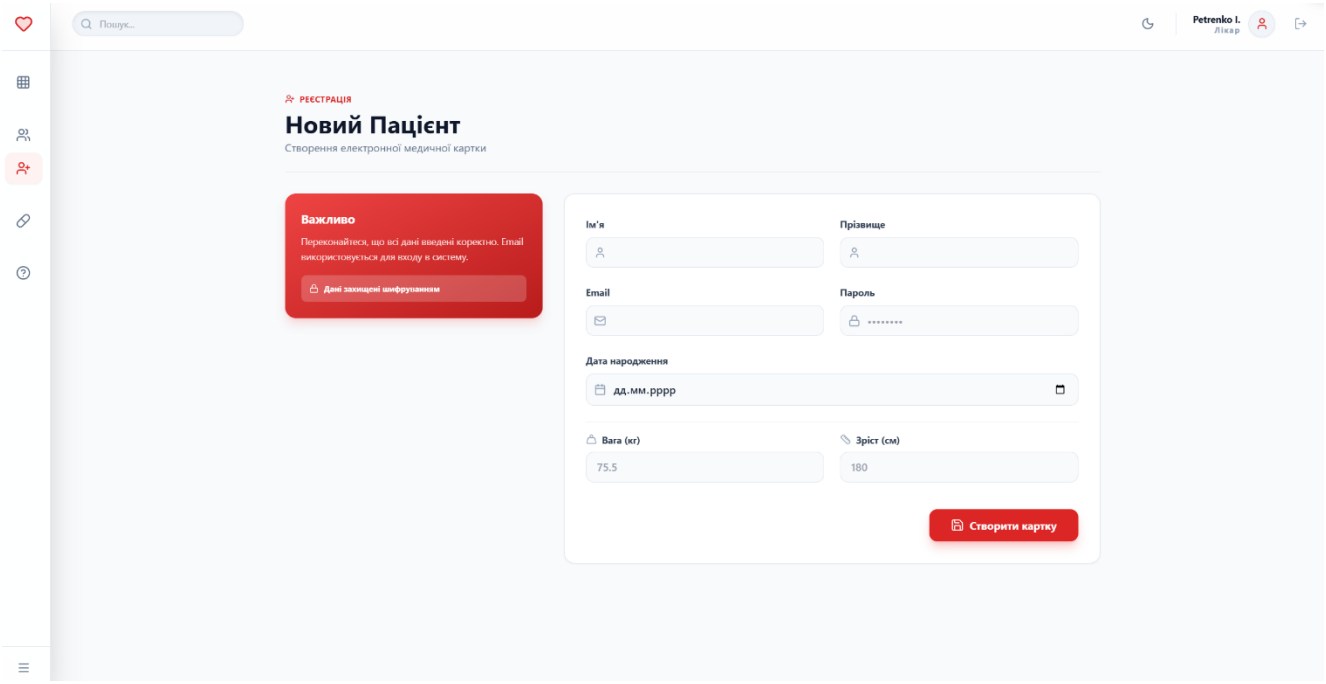


Рис. 4.9 Форма реєстрації нового пацієнта

Форма медичної карти надається для всіх пацієнтів, на якому зафіксовано поточний стан та усю необхідну інформацію пов’язану з пацієнтом. Сторінку логічно було поділено на основні логічні блоки для більш швидкої взаємодії з лікарем.

З самого верху відображено ключові блоки останніх записів у систему та вік хворого, для того щоб лікар одразу був ознайомленим з ключовими параметрами з, яким йому прийдеться працювати. Наступний блок відповідає за призначення нових, або заміни існуючих препаратів для проходження курсу лікування. Кожний медичний засіб має свій термін прийому та обмежено дозування. У свою чергу пацієнт має подібне вікно в якому він має відмічати прийом.

Далі йде блок з графіком усіх вимірювань для того, щоб лікар мав змогу візуально передивитися усі коливання та швидко визначитися з ймовірним станом пацієнта. При наведенні на кожну дату графіка відображається модальне вікно з цифровими характеристиками кожного дня.

Наостанок, модуль демонструє усі пропуски та прийоми призначених ліків. Сірі - відповідають за пропущений день, жовтий – неповний прийом, зелений – за прийом усіх ліків.

Візуальне представлення екранної форми персональної картки пацієнта зображено на рисунку 4.10.

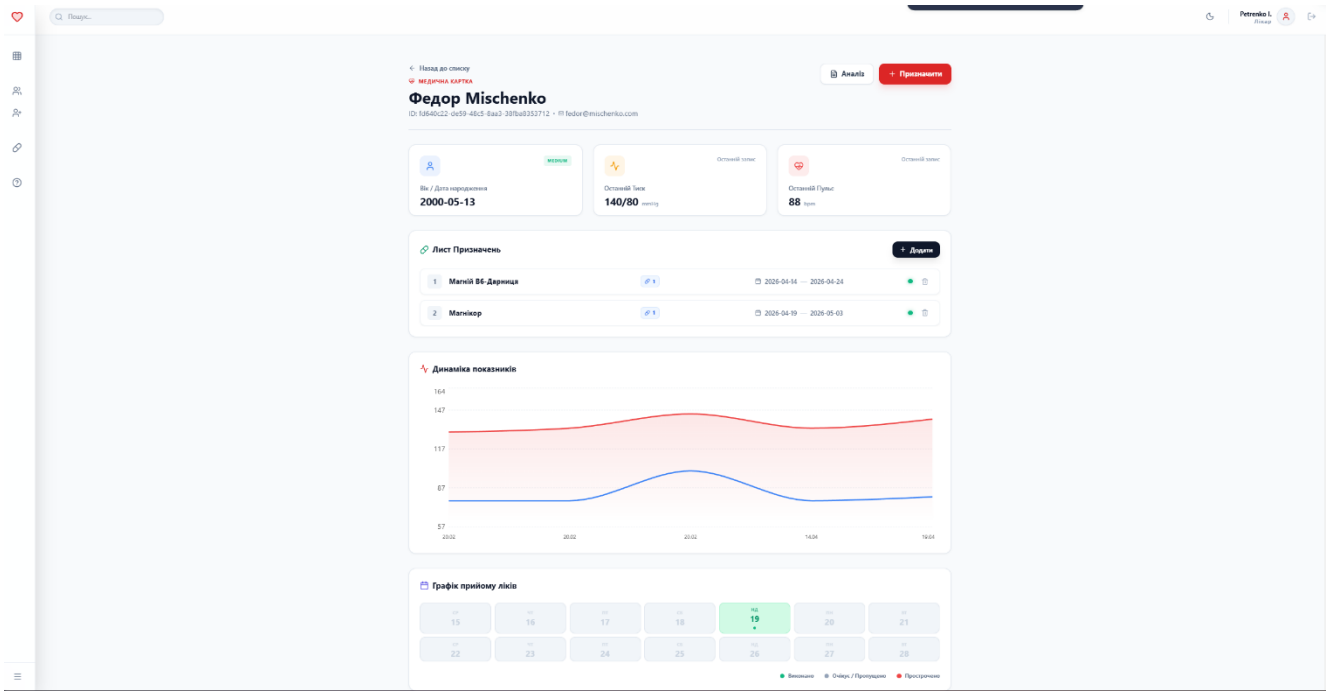


Рис. 4.10 Екранна форма персональної картки пацієнта

ВИСНОВКИ

Результатом виконаної роботи став сучасний веборієнтований застосунок призначений для клінічного моніторингу медичних показників серцево-судинної системи. Система реалізовує повний цикл супроводу взаємодії пацієнта починаючи від додавання нового клінічного показника, закінчуючи оглядом висновків лікаря. Застосунок орієнтований на спрощення проведення традиційного лікування артеріальних захворювань шляхом зберігання медичної інформації у єдиної платформі керування.

В застосунку створено окремі кабінети для кожної ролі зі своїм функціоналом, а також інтегровано хмарний сервіс керування моделями штучного інтелекту для швидкої допомозі в аналізі стану пацієнта. Використання розмежовування по ролях забезпечила гнучкий розподіл між основним ролями такими як: адміністратор, лікар та пацієнт, що створює гарантію безпеки застосунку і зменшує ризик прояву стороннього доступу іншої ролі до можливостей користувача.

Завдяки впровадженню модуля взаємодії створення клінічного висновку з хмарною платформою штучного інтелекту, дозволило пришвидшити та систематизувати процес попередньої обробки показників та своєчасного прийнятті діагностичного рішення. При цьому остаточний висновок залишився під контролем лікаря, який може змінювати або доповнювати згенерований результат, що робить баланс між професійністю медичного працівника та автоматизованою гнучкістю системи. Такий підхід є важливим, так як модель штучного інтелекту виступає у ролі інструменту для прийняття рішення, а не його повноцінною заміною, залишаючи відповідальність за остаточним рішенням лікаря.

Створений інструмент є ефективним рішенням для професійного використання в медичних установах. Застосунок розроблено з можливістю подальшого масштабування використовуючи інші медичні показники та інтеграції сторонніх пристроїв під різні потреби. Модульна архітектура платформи та

контейнеризація модулів забезпечують технічну готовність системи до майбутнього розширення без необхідності суттєвого втручання в існуючу розробки медичної платформи. Розроблені механізми закладають надійність системи та гарантує відповідність платформ до усіх технологічних вимог. У ході розробки кваліфікаційної роботи було досягнуто поставлених задач:

1. На основі проведеного аналізу методів та технологій інтелектуального моніторингу показників артеріального тиску, визначено переваги і недоліки процесу обліку та аналізу показників артеріального тиску.

2. На основі аналізу існуючого програмного забезпечення обліку та аналізу показників артеріального тиску, визначено перелік функціональних та нефункціональних вимог.

3. Обґрунтовано технологічних стек для реалізації сервісу, що включає React для швидкої побудови інтерфейсу, Spring Boot для побудови серверної логіки, PostgreSQL для зберігання та обробки даних системи.

4. На основі визначених вимог та стеку технологій спроєктовано ієрархічну модель доступу до системи, розроблено архітектуру застосунку та структуру базу даних, що забезпечує надійне розмежування прав користувачів та гарантує конфіденційність медичної платформи.

5. На основі проведеного мануального та інтеграційного тестування, визначено, що застосунок працює стабільно, а взаємодія модулів відповідають вимогам тестування.

6. Робота пройшла апробацію. За результатами було опубліковані наступні тези доповідей:

1. Сухомінський Є.В., Замрій І.В. Проєктування платформи для оцінки ризиків серцево-судинних захворювань на основі аналізу рішень аналогів. II Всеукраїнська науково-технічна конференція «Виклика та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез К.: ДУІКТ, 2025. С.95-97.

2. Сухомінський Є.В., Замрій І.В. Захист персональних даних у вебзастосунку для моніторингу серцево-судинної системи. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2026. С.225-228.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ковтун Г. І., Орлова Н. М. Смертність від хвороб системи кровообігу в Україні: медикостатистичний аналіз її динаміки та регіональних особливостей у 2010-2020 рр.. *Вісник Вінницького національного медичного університету*. 2023. С. 110–118.
2. Шпикуляк А. В. Системи для дистанційного моніторингу у медичних цілях. *Вінницький національний технічний університет*. 2024. С. 1–2. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/47144/21622.pdf?sequence=3&isAllowed=y.
3. Pugalenthi L., Senapati S., Gohri J. Blood Pressure Variability (BPV) as a Novel Digital Biomarker of Multisystem Risk and Diagnostic Insight: Measurement, Mechanisms, and Emerging Artificial Intelligence Methods. *Biomedicines*. 2026. С. 317. URL: https://www.mdpi.com/2227-9059/14/2/317.
4. Global strategy on digital health 2020-2025. *World Health Organization*. 2021. С. 1–44. URL: https://www.who.int/docs/default-source/documents/gs4dhdaa2a9f352b0445bafbc79ca799dce4d.pdf.
5. Булеца С. Телемедицина: переваги та недоліки в правовому полі. *Право України*. 2020. С. 49–60. URL: https://dspace.uzhnu.edu.ua/server/api/core/bitstreams/ff639526-92c1-4b2e-8608-c61cfd7e447a/content.
6. Богомолов Д. А. Телемедицина як елемент електронного врядування у сфері охорони здоров’я. *Науковий вісник Міжнародного гуманітарного університету.* 2023. С. 14–18. URL: https://vestnik-pravo.mgu.od.ua/archive/juspradenc65/65_2023.pdf#page=14.
7. Свінціцький А., Климова В., Сендецький С. Використання штучного інтелекту в медицині, хірургії, стоматології, онкології. *Державне некомерційне підприємство «Національний інститут раку»*. 2024. С. 1–4.

URL: https://www.clinicaloncology.com.ua/wp/wp-content/uploads/2025/01/955.pdf.

8. Вовк А. Використання штучного інтелекту в медицині. *Вінницький національний технічний університет*. 2024. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/47016/21267.pdf?sequence=3&isAllowed=y.
9. Бакуменко А., Загребельна Н. Захист конфіденційності та приватності в умовах розширеного застосування штучного інтелекту. *Legal Bulletin №1*. 2024. С. 78–85. URL: https://lbku.krok.edu.ua/index.php/legal-bulletin/article/view/442/385.
10. A smarter way to manage your blood pressure. *SmartBP*. URL: https://www.smartbp.app/.
11. Meaningful insights. Backed by science. *Apple Health*. URL: https://www.apple.com/health/.
12. Дані про Ваше здоров'я доступні в будь-який час. *OMRON Connect*. URL: https://omron-ukraine.com.ua/uk/omron-connect.
13. About MedM Health Cloud. *OMRON Connect*. URL: https://health.medm.com/en/about.
14. Bosanac D., Stevanovic A. Trust in E-Health System and Willingness to Share Personal Health Data. *Informatics and Technology in Clinical Care and Public Health*. 2022. С. 256–259.
15. Ямнич А., Коробейнікова Т. Модель контролю доступу персоналу до інформаційних ресурсів підприємств на основі RBAC та технології blockchain. *Technical sciences*. 2024. С. 380–386. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/866/902.
16. Кобрусєва Є. Інформаційна безпека електронних систем охорони здоров'я: теоретичні та практичні аспекти. *Юридичний науковий електронний журнал*. 2025. С. 310–312. URL: https://www.lsej.org.ua/1_2025/71.pdf.
17. JDK 21 Documentation - Home. *Oracle Help Center*. URL: https://docs.oracle.com/en/java/javase/21/.

18. TypeScript Documentation. *TypeScript*. URL: https://www.typescriptlang.org/docs/.
19. IntelliJ IDEA - in Java and Kotlin. *JetBrains*. URL: https://www.jetbrains.com/idea/.
20. Visual Studio Code - The open source AI code editor. *Visual Studio Code*. URL: https://code.visualstudio.com/.
21. Spring Boot. *SpringSource*. URL: https://docs.spring.io/spring-boot/index.html.
22. Spring Security. *SpringSource*. URL: https://docs.spring.io/spring-security/reference/index.html.
23. Spring Data JPA. *SpringSource*. URL: https://docs.spring.io/spring-data/jpa/reference/index.html.
24. Quick Start - React. *React*. URL: https://react.dev/learn.
25. Vite | Next Generation Frontend Tooling. Vite. URL: https://vite.dev/.
26. Documentation. *Tailwind CSS*. URL: https://v2.tailwindcss.com/docs.
27. PostgreSQL: Documentation. *PostgreSQL*. URL: https://www.postgresql.org/docs/.
28. Documentation. *Keycloak*. URL: https://www.keycloak.org/documentation.
29. GitHub Docs. *GitHub Docs*. URL: https://docs.github.com/en.
30. OpenRouter Quickstart Guide | Developer Documentation. *OpenRouter*. URL: https://openrouter.ai/docs/quickstart.
31. Reference documentation | Docker Docs. *Docker Docs*. URL: https://docs.docker.com/reference/.
32. JUnit 5 User Guide. *JUnit*. URL: https://docs.junit.org/5.5.0/user-guide/.
33. Mockito framework site. *Mockito framework*. URL: https://site.mockito.org/.
34. Postman Docs | Postman Docs. *Postman*. URL: https://learning.postman.com/.
35. Architectures. *Distributed Systems (4nd edition)* / M. Van Steen, A. S. Tanenbaum. Maarten van Steen, 2023. С. 55–108.
36. Hombergs T. Get Your Hands Dirty on Clean Architecture (2nd edition). Packt Publishing, 2022. 156 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для обліку та аналізу показників артеріального тиску

Виконав студент 4 курсу
групи ПД-44
Сухомінський Євгеній Віталійович
Керівник роботи
завідувач кафедри ІПЗ, д-р техн. наук, професор Замрій Ірина Вікторівна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення збору та аналізу статистичних даних показників артеріального тиску для вирішення проблеми віддаленого нагляду за пацієнтами із серцево-судинними захворюваннями.

Об'єкт дослідження – процес дистанційного моніторингу та контролю показників артеріального тиску пацієнтів із серцево-судинними захворюваннями.

Предмет дослідження – програмні засоби та технології розробки вебзастосунку для збору та аналізу показників артеріального тиску.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Огляд та аналіз існуючих методів та технологій інтелектуального моніторингу показників артеріального тиску.
2. Аналіз аналогічного програмного забезпечення.
3. Визначення функціональних і нефункціональних вимог.
4. Визначення технічних засобів розробки.
5. Проєктування застосунку для автоматизованого збору показників артеріального тиску.
6. Програмна реалізація застосунку для дистанційного моніторингу артеріального тиску.
7. Провести тестування застосунку.

3

АНАЛІЗ АНАЛОГІВ

	SmartBP	Apple Health	OMRON connect	MedM Health	CardioCare
Окремий кабінет лікаря	-	-	-	+	+
Робота через браузер	-	-	-	+	+
Генерація медичних звітів	-	-	+	-	+
Спеціалізація суто на серці (без фітнес даних)	+	-	+	-	+
ШІ-асистент для автоматичних висновків	-	-	-	-	+
Інтерактивний дашборд динаміки показників	-	-	-	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

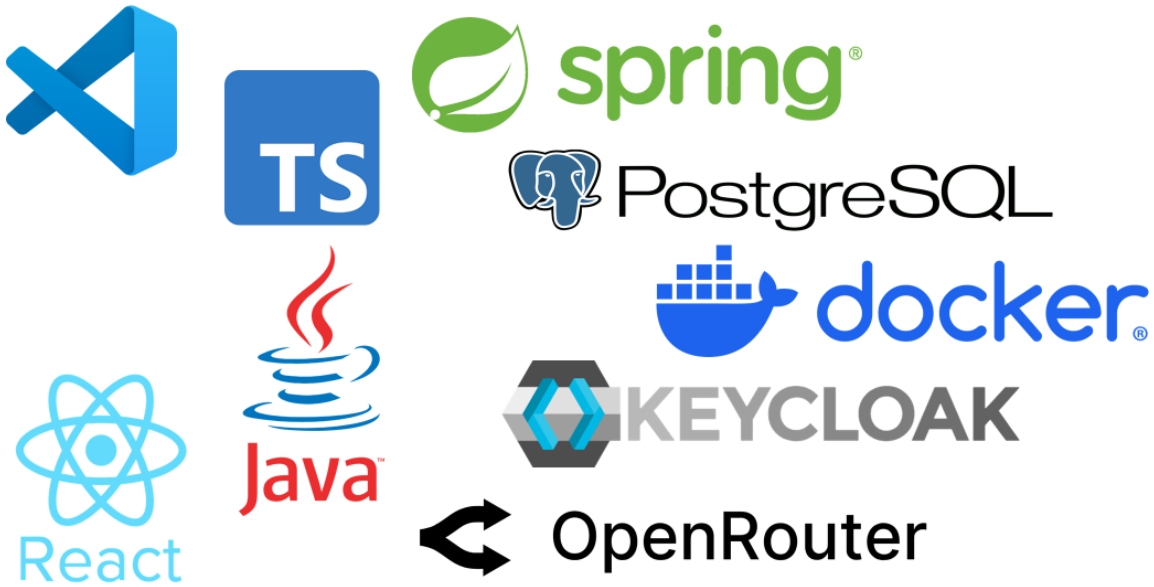
1. Реєстрація та авторизація різних типів користувачів (пацієнт, лікар, адміністратор) за допомогою єдиної системи управління ідентифікацією
2. Можливість вносити, зберігати і переглядати історію вимірювань (пульс, артеріальний тиск) на інтерактивних веб-дашбордах у реальному часі
3. Призначення лікарем схеми лікування та автоматизований трекінг прийому медикаментів пацієнтом
4. Інтеграція з уніфікованою платформою штучного інтелекту для автоматичного підготовки аналітичних звітів для лікаря

Нефункціональні вимоги:

1. Застосування Keycloak для забезпечення безпечної автентифікації та надійного захисту облікових записів користувачів
2. Анонімізація персональних даних шляхом заміни ПІБ на випадкові UUID перед їх відправкою запитів до зовнішніх ШІ-сервісів
3. Розподіл системи на незалежні модулі використовуючи Docker
4. Відображення логування та моніторингу стану системи

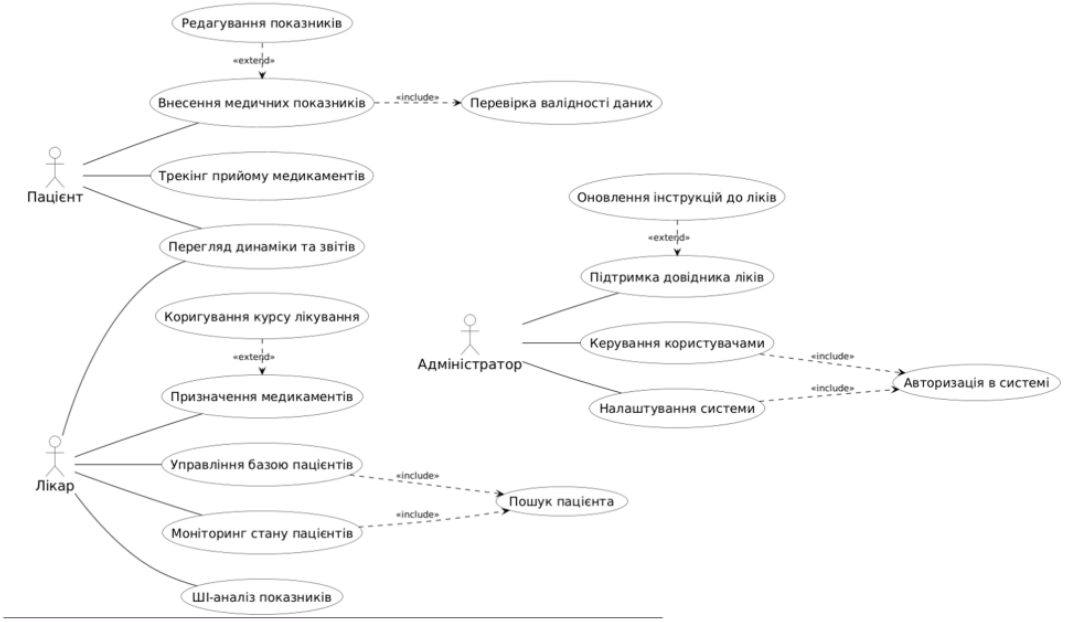
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



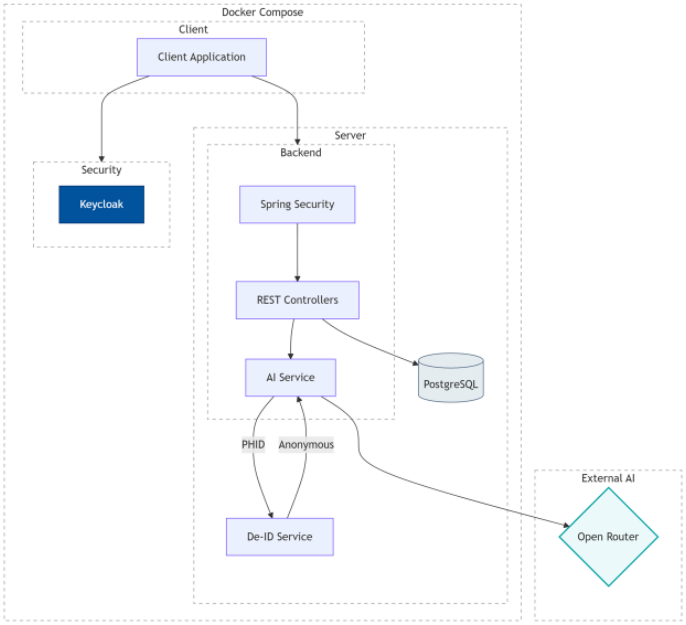
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



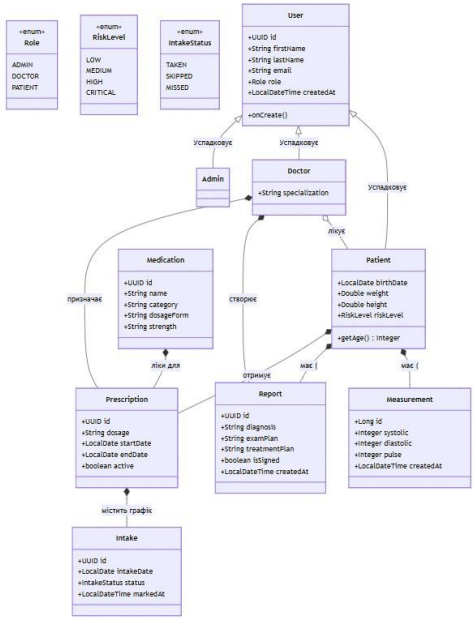
7

АРХІТЕКТУРА ЗАСТОСУНКУ



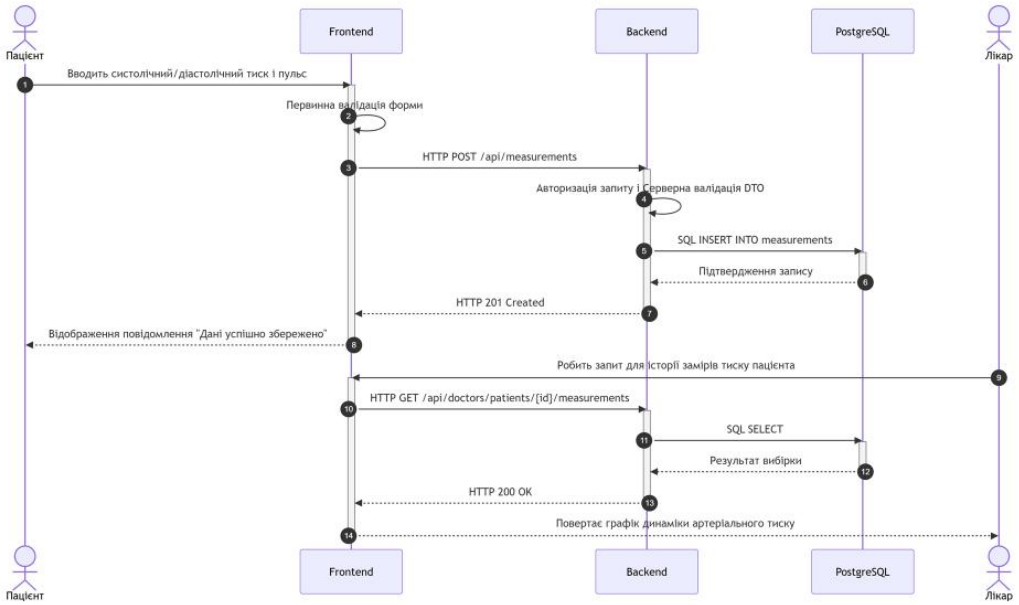
8

ДІАГРАМА КЛАСІВ



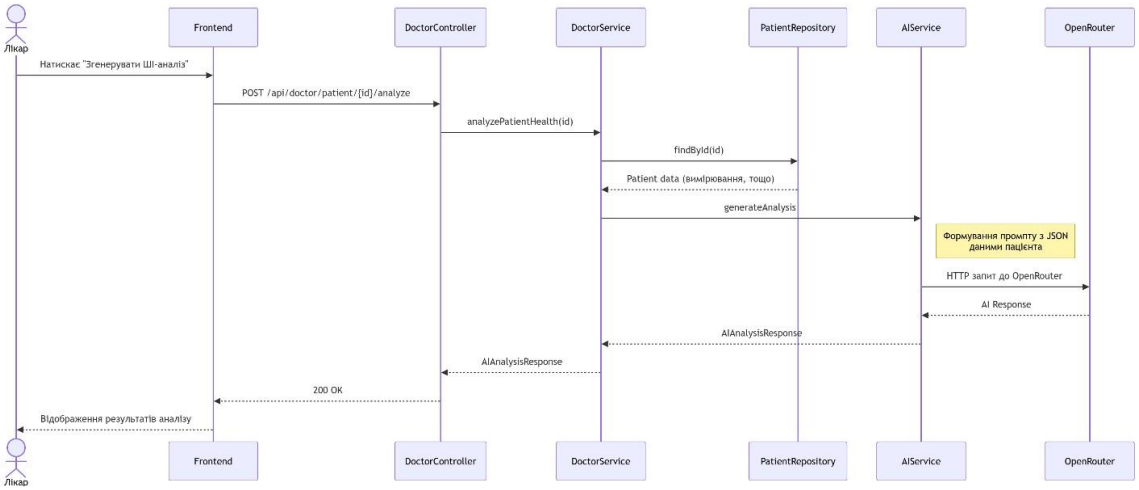
9

ДІАГРАМА ПОСЛІДОВНОСТЕЙ



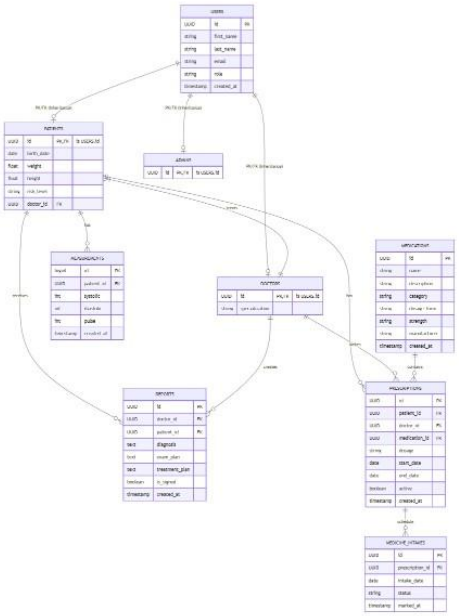
10

СХЕМА ВЗАЄМОДІЇ З СЕРВІСОМ OPENROUTER



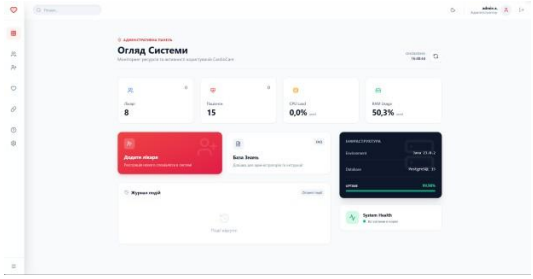
10

ER-ДІАГРАМА БАЗИ ДАНИХ

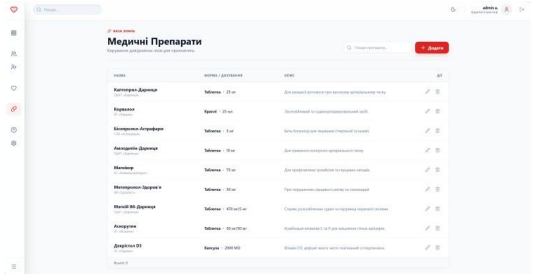


11

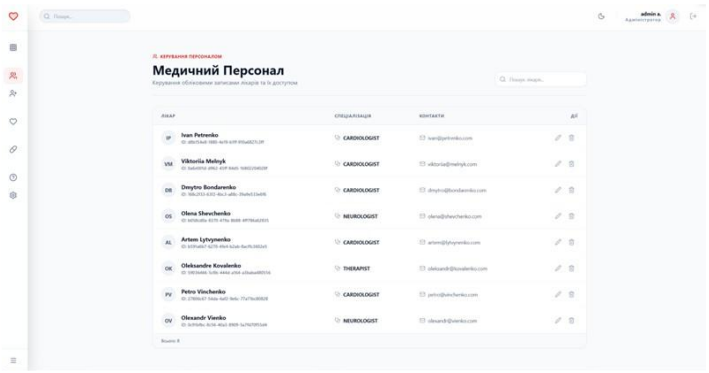
ЕКРАННІ ФОРМИ



Головна панель адміністратора

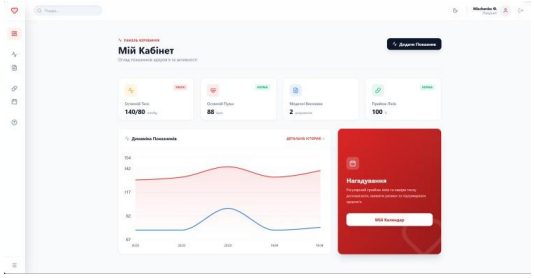


База знань лікарських препаратів

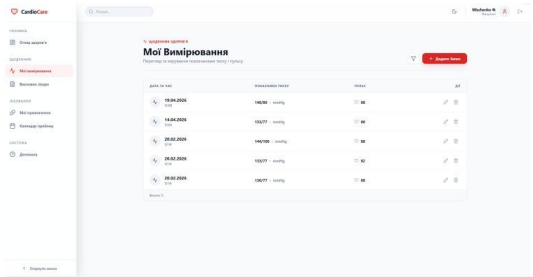


Інтерфейс управління обліковими записами медичного персоналу

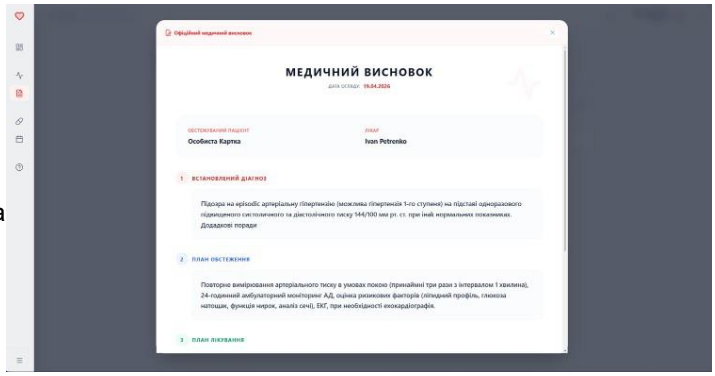
ЕКРАННІ ФОРМИ



Головна сторінка особистого кабінету пацієнта

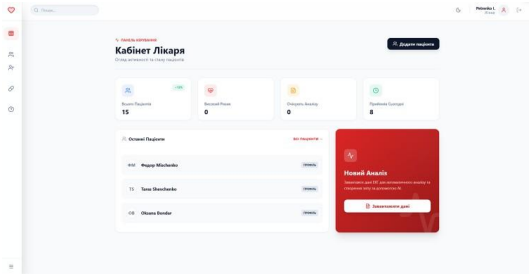


Журнал моніторингу показників

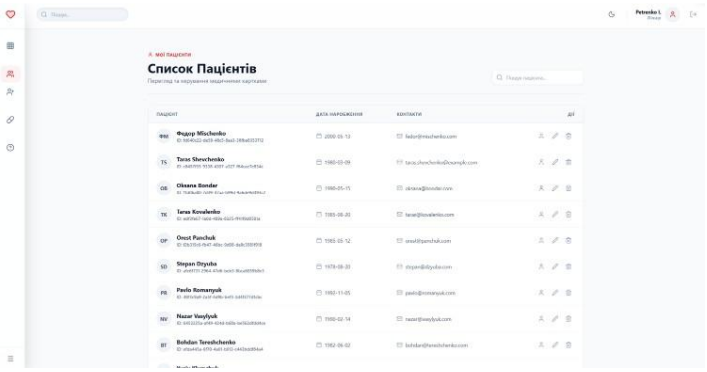


Візуалізація сформованого цифрового медичного висновку

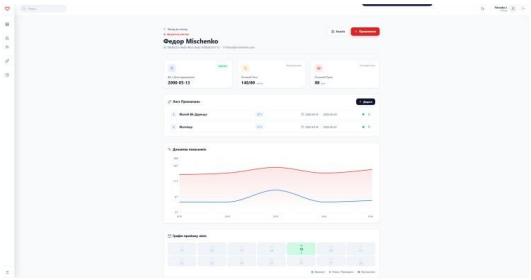
ЕКРАННІ ФОРМИ



Панель моніторингу пацієнтів



Загальний реєстр пацієнтів



Деталізована електронна медична картка

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Сухомінський Є.В., Замрій І.В. Проєктування платформи для оцінки ризиків серцево-судинних захворювань на основі аналізу рішень аналогів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, ДУІКТ. Збірник тез. К.: ДУІКТ, 2025. С.95-97.
2. Сухомінський Є.В., Замрій І.В. Захист персональних даних у вебзастосунку для моніторингу серцево-судинної системи. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. К.: ДУІКТ, 2026. С.225-228.

16

ВИСНОВКИ

1. На основі проведеного аналізу методів та технологій інтелектуального моніторингу показників артеріального тиску, визначено переваги і недоліки процесу обліку та аналізу показників артеріального тиску.
2. На основі аналізу існуючого програмного забезпечення обліку та аналізу показників артеріального тиску, визначено перелік функціональних та нефункціональних вимог.
3. Обґрунтовано технологічних стек для реалізації сервісу, що включає React для швидкої побудови інтерфейсу, Spring Boot для побудови серверної логіки, PostgreSQL для зберігання та обробки даних системи.
4. На основі визначених вимог та стеку технологій спроєктовано ієрархічну модель доступу до системи, розроблено архітектуру застосунку та структуру базу даних, що забезпечує надійне розмежування прав користувачів та гарантує конфіденційність медичної платформи.
5. На основі проведеного мануального та інтеграційного тестування, визначено, що застосунок працює стабільно, а взаємодія модулів відповідають вимогам тестування.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
package com.cardiocare.cardiocarebackend.entity;

import jakarta.persistence.*;
import lombok.Data;
import java.time.LocalDateTime;
import java.util.UUID;

@Entity
@Table(name = "users")
@Inheritance(strategy = InheritanceType.JOINED)
@Data
public class User {
    @Id
    private UUID id;

    private String firstName;
    private String lastName;
    private String email;

    @Enumerated(EnumType.STRING)
    private Role role;

    private LocalDateTime createdAt;

    @PrePersist
    public void onCreate() {
        this.createdAt = LocalDateTime.now();
    }

    public enum Role {
        ADMIN, DOCTOR, PATIENT
    }
}
```

```
package com.cardiocare.cardiocarebackend.service;

import
com.cardiocare.cardiocarebackend.dto.response.AIAnalysisRes
ponse;
import
com.cardiocare.cardiocarebackend.entity.IntegrationSettings;
import com.cardiocare.cardiocarebackend.entity.Measurement;
import com.cardiocare.cardiocarebackend.entity.Patient;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestClient;

import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
@Slf4j
public class AIService {
    private final RestClient restClient;
    private final ObjectMapper objectMapper;
    private final SettingsService settingsService;

    private static final String SYSTEM_PROMPT = "You are a
helpful medical assistant that outputs strictly valid JSON. No
markdown, no extra text.";

    private static final String USER_PROMPT_TEMPLATE =
"""
            Anonymous Patient ID: %s
            Patient age: %s years, Height: %s cm, Weight: %s kg.
            Blood pressure measurements (date: systolic/diastolic,
pulse):
            [%s]

            Based on the measurements and patient data, provide a
medical analysis.
            Return ONLY this exact JSON structure with Ukrainian
text:
            {
              "patientId": "ID that was provided above, strictly copy
it here",
              "diagnosis": "Попередній клінічний висновок та
діагноз...",
              "examPlan": "План необхідних обстежень...",
              "treatmentPlan": "План лікування та
рекомендації..."
            }
            """;

    private static class AnonymousClinicalData {
        private final UUID anonymousId;
        private final String age;
        private final String height;
        private final String weight;
        private final String measurementsInfo;

        public AnonymousClinicalData(UUID anonymousId,
String age, String height, String weight, String
measurementsInfo) {
            this.anonymousId = anonymousId;
            this.age = age;
            this.height = height;
            this.weight = weight;
            this.measurementsInfo = measurementsInfo;
        }

        public UUID getAnonymousId() { return anonymousId; }
        public String getAge() { return age; }
        public String getHeight() { return height; }
        public String getWeight() { return weight; }
        public String getMeasurementsInfo() { return
measurementsInfo; }
```

```
    public static AnonymousClinicalData decompose(Patient
patient, List<Measurement> measurements) {
        String measurementsText = measurements.stream()
            .limit(30)
            .map(m -> String.format("%s: %d/%d (Pulse:
%d)",
                m.getCreatedAt().toString().substring(0, 10),
                m.getSystolic(), m.getDiastolic(),
m.getPulse()))
            .collect(Collectors.joining("; "));

        String ageStr = patient.getBirthDate() != null
            ?
String.valueOf(java.time.Period.between(patient.getBirthDate()
, java.time.LocalDate.now()).getYears())
            : "невідомо";

        return new AnonymousClinicalData(
            UUID.randomUUID(), ageStr,
            patient.getHeight() != null ?
String.valueOf(patient.getHeight()) : "невідомо",
            patient.getWeight() != null ?
String.valueOf(patient.getWeight()) : "невідомо",
            measurementsText
        );
    }
  }

  public AIAnalysisResponse analyzePatient(Patient patient,
List<Measurement> measurements) {
    IntegrationSettings settings =
settingsService.getSettings();
    AnonymousClinicalData anonymousData =
AnonymousClinicalData.decompose(patient, measurements);

    log.info("Initiating secure AI session with anonymous
UUID: {}", anonymousData.getAnonymousId());

    String prompt =
String.format(USER_PROMPT_TEMPLATE,
        anonymousData.getAnonymousId().toString(),
        anonymousData.getAge(),
anonymousData.getHeight(),
        anonymousData.getWeight(),
anonymousData.getMeasurementsInfo());

    Map<String, Object> requestBody = Map.of(
        "model", settings.getAiModelName(),
        "messages", List.of(
            Map.of("role", "system", "content",
SYSTEM_PROMPT),
            Map.of("role", "user", "content", prompt)),
        "response_format", Map.of("type", "json_object"),
        "temperature", 0.2);

    try {
      String rawResponse = restClient.post()
          .uri(settings.getAiProviderUrl())
          .header("Authorization", "Bearer " +
settings.getAiApiKey())
          .contentType(MediaType.APPLICATION_JSON)
          .body(requestBody)
          .retrieve()
          .body(String.class);

      return parseResponse(rawResponse,
anonymousData.getAnonymousId());
    } catch (Exception e) {
      log.error("AI Error", e);
      return new AIAnalysisResponse("Помилка AI",
"Перевірте API ключ у налаштуваннях", e.getMessage());
    }
  }

  private AIAnalysisResponse parseResponse(String
rawResponse, UUID expectedAnonymousId) {
    try {
      JsonNode root = objectMapper.readTree(rawResponse);
      if (root.has("error")) {
        return new AIAnalysisResponse("Помилка AI",
root.path("error").path("message").asText(""), rawResponse);
      }

      JsonNode choicesNode = root.path("choices");
      if (choicesNode.isMissingNode() ||
choicesNode.isEmpty()) {
        return new AIAnalysisResponse("Некоректна
відповідь ШІ", "Відсутні дані", rawResponse);
      }

      String content =
choicesNode.get(0).path("message").path("content").asText();
      String cleanJson = content.replace("```json",
"").replace("```", "").trim();

      JsonNode jsonContent =
objectMapper.readTree(cleanJson);
      return new AIAnalysisResponse(
          jsonContent.path("diagnosis").asText("Не вдалося
сформувати діагноз"),
          jsonContent.path("examPlan").asText(""),
          jsonContent.path("treatmentPlan").asText(""));
    } catch (Exception e) {
      log.error("JSON Parse Error", e);
      return new AIAnalysisResponse("Помилка формату",
"AI повернув некоректний JSON", rawResponse);
    }
  }
}

package com.cardiocare.cardiocarebackend.controller;

import com.cardiocare.cardiocarebackend.dto.request.*;
import com.cardiocare.cardiocarebackend.dto.response.*;
import com.cardiocare.cardiocarebackend.entity.*;
import
com.cardiocare.cardiocarebackend.mapper.MapperService;
import
com.cardiocare.cardiocarebackend.repository.MedicationRepos
itory;
```

```
import com.cardiocare.cardiocarebackend.service.*;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPr
incipal;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.web.bind.annotation.*;
import jakarta.validation.Valid;

import java.time.LocalDate;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/doctor")
@PreAuthorize("hasRole('DOCTOR')")
@RequiredArgsConstructor
public class DoctorController {

    private final DoctorService doctorService;
    private final PatientService patientService;
    private final MapperService mapperService;
    private final PrescriptionService prescriptionService;
    private final MedicationRepository medicationRepository;

    @PostMapping("/patient")
    public PatientResponse createPatient(@RequestBody
PatientRequest request, @AuthenticationPrincipal Jwt jwt) {
        UUID doctorId = UUID.fromString(jwt.getSubject());
        Patient saved = doctorService.createPatient(doctorId,
request);
        return mapperService.toDto(saved);
    }

    @GetMapping("/patients")
    public List<PatientResponse>
getMyPatients(@AuthenticationPrincipal Jwt jwt) {
        return
doctorService.getMyPatients(UUID.fromString(jwt.getSubject(
)))

.stream().map(mapperService::toDto).collect(Collectors.toList(
));
    }

    @GetMapping("/patient/{id}/measurements")
    public ResponseEntity<List<MeasurementResponse>>
getPatientMeasurements(
            @PathVariable UUID id,
            @PageableDefault(size = 50, sort = "createdAt",
direction = Sort.Direction.DESC) Pageable pageable) {
        List<Measurement> measurements =
doctorService.getPatientMeasurements(id, pageable);
        List<MeasurementResponse> response =
measurements.stream()

.map(mapperService::toDto).collect(Collectors.toList());
        return ResponseEntity.ok(response);
    }

    @PostMapping("/patient/{id}/analyze")
    public ResponseEntity<AIAnalysisResponse>
analyze(@PathVariable UUID id) {
        return
ResponseEntity.ok(doctorService.analyzePatientHealth(id));
    }

    @PostMapping("/report")
    public ResponseEntity<ReportResponse> createReport(
            @RequestBody ReportRequest request,
            @AuthenticationPrincipal Jwt jwt) {
        UUID doctorId = UUID.fromString(jwt.getSubject());
        Report savedReport =
doctorService.createClinicalReport(doctorId, request);
        return
ResponseEntity.ok(mapperService.toDto(savedReport));
    }

    @PostMapping("/prescription")
    public PrescriptionResponse createPrescription(
            @Valid @RequestBody PrescriptionRequest request,
            @AuthenticationPrincipal Jwt jwt) {
        UUID doctorId = UUID.fromString(jwt.getSubject());
        return prescriptionService.createPrescription(request,
doctorId);
    }
}

import axios from 'axios';

const api = axios.create({
    baseURL: 'http://localhost:8081/api',
});

api.interceptors.request.use(
    (config) => {
        const token = localStorage.getItem('token');
        if (token) {
            config.headers.Authorization = `Bearer ${token}`;
        }
        return config;
    },
    (error) => {
        return Promise.reject(error);
    }
);

  api.interceptors.response.use(
    (response) => response,
    (error) => {
        if (error.response && error.response.status === 401) {
            localStorage.removeItem('token');
            window.location.href = '/login';
```

```
    }
    return Promise.reject(error);
  }
);

export default api;
```

```
import { BrowserRouter, Routes, Route, Navigate } from
'react-router-dom';
import LoginPage from './pages/LoginPage';
import AdminDashboard from
'./pages/admin/AdminDashboard';
import DoctorsList from './pages/admin/DoctorsList';
import DoctorDashboard from
"./pages/doctor/DoctorDashboard.tsx";
import DoctorPatientsList from
"./pages/doctor/PatientsList.tsx";
import PatientAnalysis from "./pages/doctor/PatientAnalysis";
import PatientDashboard from
"./pages/patient/PatientDashboard.tsx";
import PatientMeasurements from
"./pages/patient/PatientMeasurements.tsx";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<LoginPage />} />
        <Route path="/" element={<Navigate to="/login"
replace />} />

        {/* --- АДМІН --- */}
        <Route path="/admin" element={<AdminDashboard
/>} />
        <Route path="/admin/doctors"
element={<DoctorsList />} />

        {/* --- ЛІКАР --- */}
        <Route path="/doctor" element={<DoctorDashboard
/>} />
        <Route path="/doctor/patients"
element={<DoctorPatientsList />} />
        <Route path="/doctor/patient/:id/analysis"
element={<PatientAnalysis />} />

        {/* --- ПАЦІЄНТ --- */}
        <Route path="/patient" element={<PatientDashboard
/>} />
        <Route path="/patient/measurements"
element={<PatientMeasurements />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

```
import React, { useEffect, useState, useMemo } from 'react';
import MainLayout from '../../components/MainLayout';
import api from '../../api/axiosInstance';

const PatientMeasurements = () => {
  const [measurements, setMeasurements] = useState([]);
  const [form, setForm] = useState({ systolic: '', diastolic: '',
pulse: '' });
  const [isSubmitting, setIsSubmitting] = useState(false);

  const fetchMeasurements = async () => {
    try {
      const response = await
api.get(`/patient/measurements?page=0&size=10`);
      setMeasurements(response.data);
    } catch (error) {
      console.error(error);
    }
  };

  useEffect(() => {
    fetchMeasurements();
  }, []);

  const handleSave = async (e: React.FormEvent) => {
    e.preventDefault();
    setIsSubmitting(true);
    try {
      await api.post('/patient/measurement', {
        systolic: parseInt(form.systolic),
        diastolic: parseInt(form.diastolic),
        pulse: parseInt(form.pulse),
        createdAt: new Date().toISOString()
      });
      setForm({ systolic: '', diastolic: '', pulse: '' });
      fetchMeasurements();
    } catch (error) {
      alert("Помилка збереження");
    } finally {
      setIsSubmitting(false);
    }
  };

  return (
    <MainLayout role="patient">
      <div className="max-w-7xl mx-auto p-8">
        <h1 className="text-3xl font-bold">Мої
Вимірювання</h1>

        <form onSubmit={handleSave} className="mt-6
space-y-4 bg-white p-6 rounded-xl shadow">
          <input type="number" required
value={form.systolic}
            onChange={e => setForm({ ...form, systolic:
e.target.value })}
            placeholder="Систолічний тиск (120)"
className="border p-2 w-full" />

          <input type="number" required
value={form.diastolic}
```

```
                    onChange={e => setForm({ ...form, diastolic:
e.target.value })}
                    placeholder="Діастолічний тиск (80)"
className="border p-2 w-full" />

                <input type="number" required
value={form.pulse}
                    onChange={e => setForm({ ...form, pulse:
e.target.value })}
                    placeholder="Пульс (70)" className="border
p-2 w-full" />

                <button type="submit" disabled={isSubmitting}
className="bg-red-600 text-white px-4 py-2 rounded">
                  Зберегти вимірювання
                </button>
              </form>

              <div className="mt-8">
                {measurements.map((m: any) => (
                  <div key={m.id} className="p-4 border-b flex
justify-between">
                    <span>Тиск: {m.systolic}/{m.diastolic}
mmHg</span>
                    <span>Пульс: {m.pulse}</span>
                  </div>
                ))}
              </div>
            </div>
          </MainLayout>
    );
};

export default PatientMeasurements;
```