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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для моніторингу рівня глюкози»

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

_____ Ангеліна МАКСИМЧУК
(підпис)

Виконала: здобувачка вищої освіти групи ПД-41
Ангеліна МАКСИМЧУК

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____

Київ 2025

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 кафедрою
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
«_____» _____ 2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Максимчук Ангеліні Олександрівні

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для моніторингу рівня глюкози»
керівник кваліфікаційної роботи, доктор філософії (PhD) Віталій ЗАЛИВА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи « 02» червня 2025р.

3. Вихідні дані до кваліфікаційної роботи: матеріали з теорії цифрового моніторингу глюкози, документація до використаних технологій та результати тестування у Lighthouse.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Аналіз сучасних технологій і рішень для моніторингу рівня глюкози та визначення вимог до майбутнього web-застосунку.
   2. Проєктування архітектури системи, вибір технологічного стеку та обґрунтування реалізації функціоналу з урахуванням вимог до безпеки й масштабованості.
   3. Реалізація, тестування та оцінка ефективності web-застосунку для моніторингу рівня глюкози, візуалізація та аналіз даних користувача.
5. Перелік ілюстративного матеріалу: *презентація*

1. Задачі кваліфікаційної роботи.
2. Аналіз аналогів.
3. Вимоги до програмного забезпечення.
4. Програмні засоби реалізації.
5. Варіанти використання.
6. Алгоритми роботи програми.
7. Діаграма класів.
8. Екранні форми.
9. Демонстрація роботи застосунку.
10. Апробація результатів дослідження.
11. Висновки.

6. Дата видачі завдання «25» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-06.03.2025	
2	Аналіз існуючих цифрових систем контролю глюкози, визначення недоліків. Формулювання вимог та технічного завдання, вибір стеку технологій	07.03-12.03.2025	
3	Розробка архітектури системи та проектування взаємодії компонентів	16.03-21.03.2024	
4	Реалізація клієнтської та серверної частин web-застосунку	22.03-03.04.2024	
5	Тестування застосунку, побудова графіків, валідація функціоналу	04.04-28.04.2024	
6	Оформлення роботи: вступ, висновки, реферат	29.04-12.05.2025	
7	Розробка демонстраційних матеріалів	13.04-15.05.2025	
8	Попередній захист роботи	16.05-30.05.2025	

Здобувачка вищої освіти _____ Ангеліна МАКСИМЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 63 стор., 33 рис., 3 табл., 45 джерел.

Мета роботи – спрощення процесу самоконтролю рівня глюкози в крові шляхом створення веб-застосунку.

Об'єкт дослідження – процес моніторингу рівня глюкози в крові за допомогою цифрових інструментів.

Предмет дослідження – архітектура та реалізація web-застосунку для контролю рівня глюкози.

Короткий зміст роботи: У роботі проведено аналіз існуючих рішень для моніторингу рівня глюкози, сформульовано вимоги до майбутнього web-застосунку, обґрунтовано вибір технологій (React.js, Node.js, MongoDB, JWT, Chart.js). Здійснено проєктування архітектури системи, реалізовано основні модулі: автентифікація, введення показників, аналітика, візуалізація даних. Проведено тестування web-застосунку засобами Google Lighthouse, результати підтверджують його ефективність, високу доступність, відповідність найкращим практикам та вимогам до безпеки й масштабованості.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ ГЛЮКОЗИ, WEB-ЗАСТОСУНОК, REACT, MONGODB, CHART.JS, ЦИФРОВА МЕДИЦИНА, САМОКОНТРОЛЬ, ІНТЕРФЕЙС, БЕЗПЕКА, JWT.

ЗМІСТ

ВСТУП

У сучасному світі постійно зростає поширеність цукрового діабету, який потребує регулярного моніторингу рівня глюкози в крові для запобігання ускладненням та підтримання високої якості життя пацієнтів. Традиційні методи контролю часто вимагають ручного занесення даних, мають обмежену інтеграцію та не забезпечують гнучкого аналізу показників у динаміці. У зв'язку з цим розробка цифрових інструментів, зокрема web-застосунків, для зручного, швидкого та безпечного моніторингу рівня глюкози є важливою інженерною задачею. Системи такого типу дозволяють користувачам вводити дані, візуалізувати зміни показників, отримувати аналітичні оцінки та рекомендації, що значно підвищує ефективність самоконтролю при діабеті.

Об'єктом дослідження є процес моніторингу рівня глюкози у крові з використанням цифрових інструментів.

Предметом дослідження є web-застосунок для збору, зберігання, візуалізації та аналітичної обробки показників глюкози користувачів.

Метою кваліфікаційної роботи є спрощення процесу самоконтролю рівня глюкози в крові шляхом створення веб-застосунку.

Для досягнення поставленої мети були сформульовані такі основні завдання дослідження:

- виконати огляд сучасних підходів до цифрового контролю рівня глюкози;
- визначити перелік вимог до майбутньої системи з урахуванням кращих практик та досвіду користувачів;
- розробити технічне завдання та архітектуру web-застосунку;
- реалізувати клієнтську та серверну частини проєкту;
- забезпечити збереження даних у базі MongoDB із захистом доступу через JWT;
- реалізувати модулі візуалізації та статистичної обробки даних;

- провести тестування web-застосунку та оцінити його ефективність за допомогою інструментів Lighthouse.

Методи дослідження. У роботі застосовано методи порівняльного аналізу програмних рішень, структурного проєктування, об'єктно-орієнтованого програмування, побудови REST API, клієнт-серверної взаємодії, використання ODM-бібліотеки Mongoose для MongoDB, а також інструментального тестування продуктивності та доступності web-застосунку.

Наукова новизна роботи полягає у створенні компактного, адаптивного web-застосунку, орієнтованого на індивідуальний моніторинг стану здоров’я із застосуванням відкритого технологічного стеку та можливістю масштабування в умовах обмежених ресурсів. У розробці поєднано простоту інтерфейсу, функціональну гнучкість і базову аналітичну підтримку користувача.

Практичне значення. Результати роботи можуть бути використані як основа для створення повнофункціональної медичної інформаційної системи моніторингу, а також як навчальний приклад для студентів спеціальностей у сфері комп’ютерних наук і програмної інженерії при вивченні web-технологій, MongoDB, React.js та Node.js.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ДОДАТКІВ МОНІТОРИНГУ МЕДИЧНИХ ПОКАЗНИКІВ

1.1. Огляд існуючих технологій та методів контролю рівня глюкози

У сучасних умовах постійного зростання кількості хворих на цукровий діабет значну актуальність набувають технології моніторингу рівня глюкози в крові. Існує широкий спектр програмних і апаратних рішень, орієнтованих як на побутове, так і на клінічне застосування. У рамках роботи було розглянуто найпоширеніші технології та інструменти, які використовуються для контролю глюкози, з акцентом на їхню архітектуру, функціональні особливості, рівень популярності, а також недоліки, що відкривають простір для вдосконалення та альтернативних підходів.
З огляду на особливості реалізації, існуючі рішення доцільно поділити на мобільні застосунки, апаратно-програмні комплекси, а також web-застосунки або рішення з web-компонентами.

Одним із найбільш популярних мобільних застосунків є mySugr. Це програмне забезпечення орієнтоване на щоденний моніторинг показників із використанням зручного інтерфейсу. Система підтримує введення даних вручну, а також інтеграцію з деякими моделями глюкометрів. Архітектура додатку включає клієнтську частину (мобільний застосунок), сервер із базою даних користувача, а також механізми хмарної синхронізації. Розроблений на основі кросплатформових технологій (переважно iOS, Android з використанням Java/Kotlin або Swift), він забезпечує користувачам доступ до статистики та графіків у зручній формі. Популярність mySugr обумовлена гейміфікаційними елементами та простим UX-дизайном зображено на рисунку 1.1 [1].

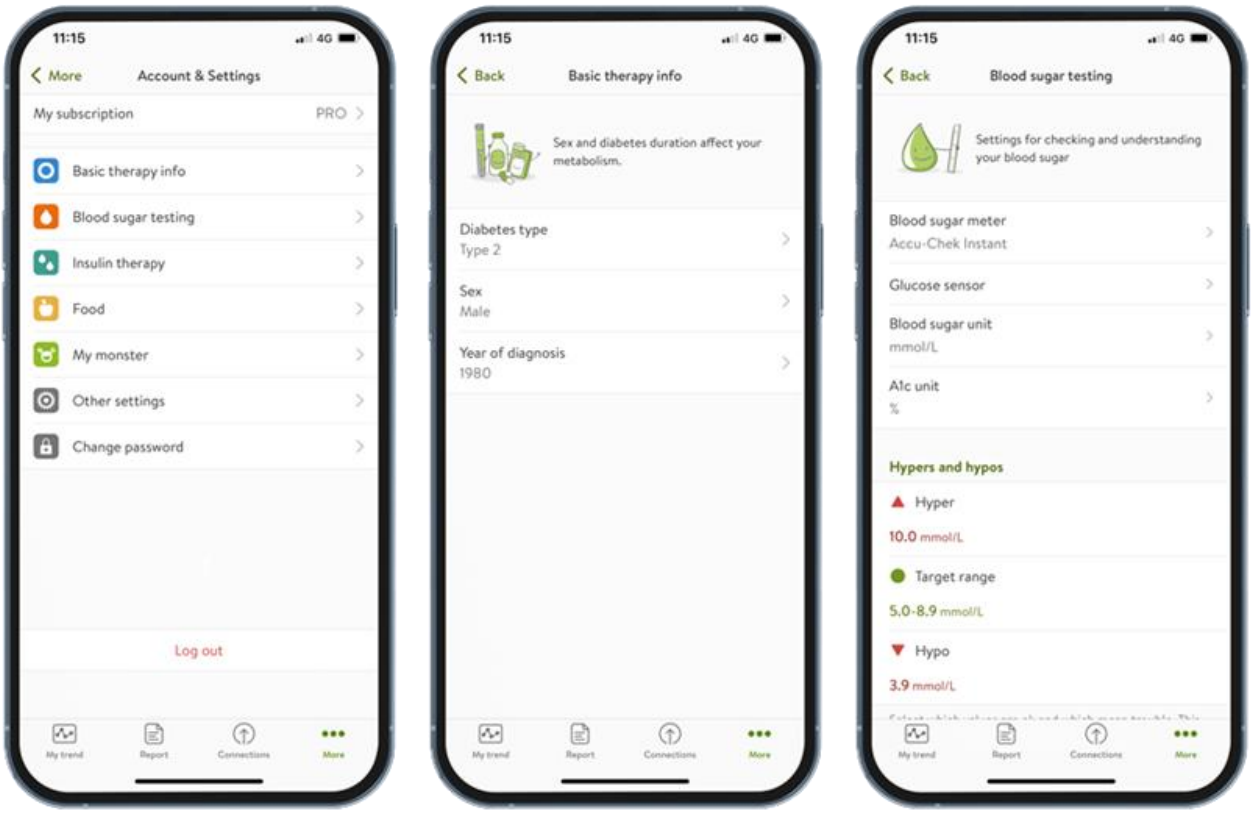


Рис. 1.1 Користувацький інтерфейс мобільного додатку mySugr [2]

У межах рандомізованого контрольованого дослідження, проведеного групою вчених із німецького дослідницького інституту FIDAM (Research Institute Diabetes-Academy Bad Mergentheim) за участі представників компаній Roche Diabetes Care та mySugr GmbH, було встановлено, що використання цифрового щоденника mySugr збільшує ймовірність досягнення цільового рівня HbA1c ≤6.5% майже вдвічі (OR = 2.24; p = 0.022) у порівнянні з контрольною групою яка зображена на рисунку 1.2 [3].

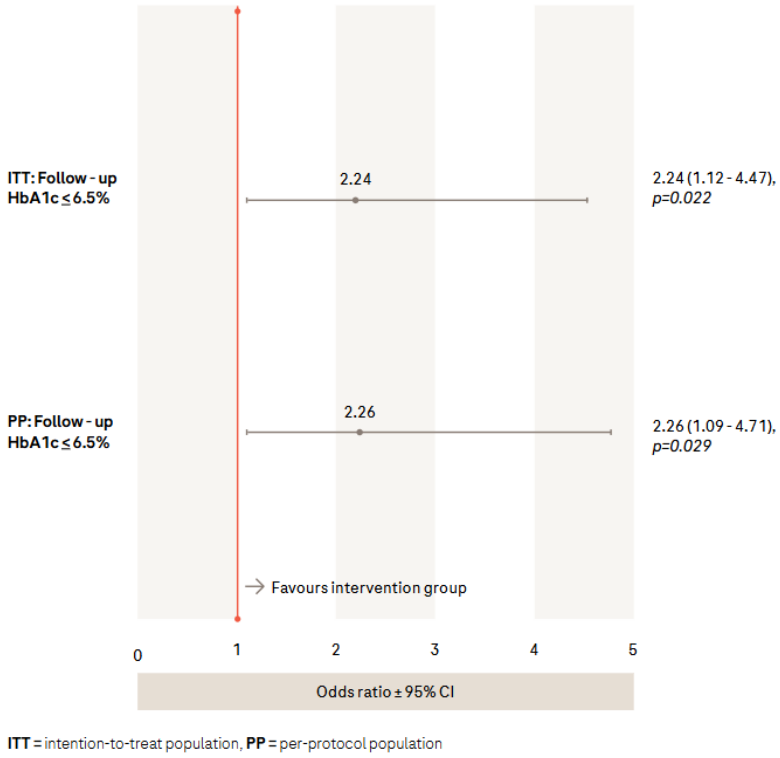


Рис. 1.2 Ймовірність досягнення цільового рівня HbA1c (≤6.5%) у користувачів мобільного застосунку mySugr [3]

Однак у безкоштовній версії користувачі стикаються з обмеженнями: недостатня деталізація аналітики, обмежений обсяг історії записів, складність експорту даних для медичного фахівця.

Застосунок Diabetes:M вирізняється багатим функціоналом — формування графіків, зведень, рекомендацій. Його архітектура реалізована у вигляді офлайн-мобільного застосунку з можливістю резервного копіювання. Технічно реалізований із використанням Java для Android та Swift для iOS. Основні функції включають ведення щоденника вимірювань, інтерпретацію показників і побудову графіків, зображено на рисунку 1.3 [4].

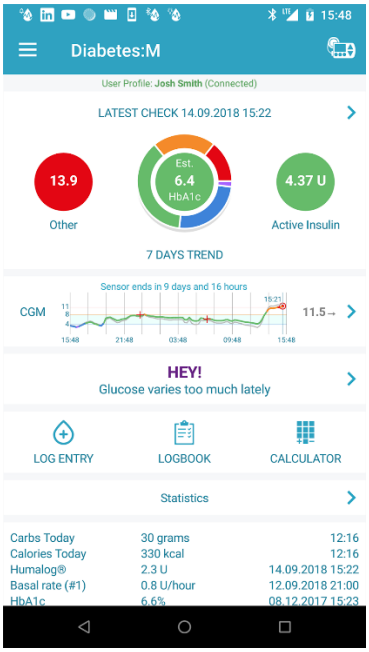


Рис. 1.3 Користувацький інтерфейс мобільного додатку Diabetes:M [5]

У процесі аналізу методів, які лежать в основі функціоналу додатку Diabetes:M, слід звернути увагу на ключову проблему, що часто виникає у користувачів — неправильне визначення тривалості дії болюсного інсуліну (DIA), що безпосередньо впливає на точність розрахунків калькулятора болюсів та ризик розвитку гіпо- або гіперглікемії. У одній зі статей, які команда розробників додатку використовувала в якості джерела інформації, подано ілюстрацію на рисунку 1.4, яка демонструє три типові сценарії стану залишкової дії інсуліну через дві години після прийому їжі. Даний приклад дозволяє краще зрозуміти критичність точного обліку BOB (bolus on board) для ефективного самоконтролю, зокрема в умовах використання мобільного ПЗ без інтеграції з апаратними засобами контролю [6].

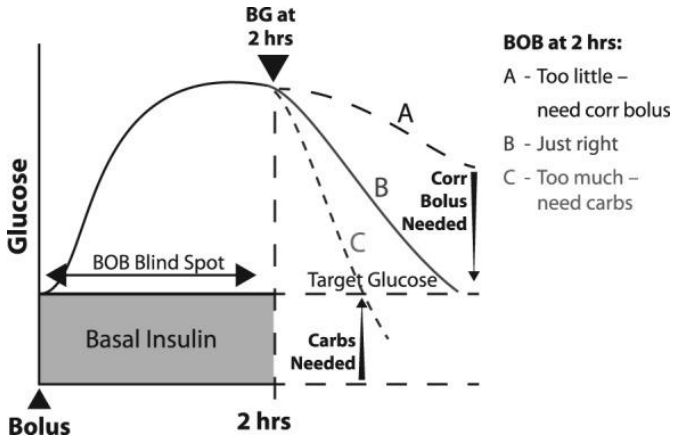


Рис. 1.4 Візуалізація типових станів залишкової дії болюсного інсуліну (BOB) через 2 години після їжі [6]

Популярність серед користувачів обумовлена гнучкістю налаштувань, однак перевантаженість інтерфейсу та складність навігації суттєво знижують зручність для нових користувачів.

До апаратно-програмних рішень належать LibreLink та Dexcom G6, що працюють у зв'язці із сенсорами безперервного моніторингу глюкози (CGM).

LibreLink — це офіційний мобільний додаток компанії Abbott, який працює виключно з сенсорами FreeStyle Libre (CGM-система). Програма дозволяє зчитувати рівень глюкози з підшкірного сенсора за допомогою технології NFC (Near Field Communication) — достатньо прикласти смартфон до сенсора. LibreLink доступний для Android та iOS, а всі зібрані дані синхронізуються з хмарною платформою LibreView, що дає змогу лікарям та пацієнтам спільно переглядати результати. Архітектура рішень Abbott побудована за моделлю клієнт–сервер із хмарним зберіганням медичних даних, зашифрованих відповідно до стандартів HIPAA/GDPR, зображено на рисунку 1.6. Перевагами LibreLink є автономність, тривалий термін роботи сенсора (до 14 днів), відсутність потреби у регулярному проколюванні пальця. Проте система потребує фірмового сенсора, що значно здорожчує користування [7].

Рис. 1.5 Користувацький інтерфейс мобільної частини LibreLink [8]

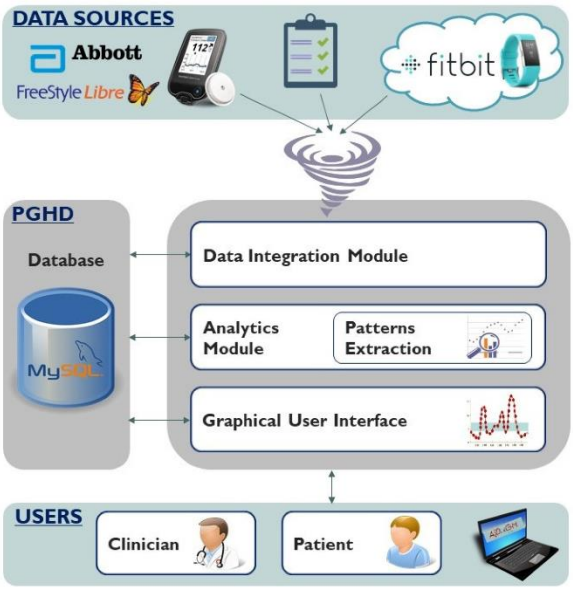


Рис. 1.6 Архітектура інтеграції даних із сенсорів FreeStyle Libre у систему моніторингу з аналітичним модулем та доступом для лікаря й пацієнта [9]

Іншим прикладом гібридного рішення з активним використанням web-технологій є Glooko, який забезпечує розширену інтеграцію з великою кількістю пристроїв: глюкометрами, фітнес-трекерами, тонометрами. Сервіс орієнтований як на пацієнтів, так і на лікарів, надаючи доступ до аналітики через хмарний інтерфейс. Архітектурно Glooko використовує багаторівневу модель з хмарним сховищем, API для інтеграцій, а також клієнтські застосунки, що зображені на рисунку 1.7 [11, 12]. Програмна реалізація виконана з використанням web-технологій (JavaScript, React, REST API), а також нативних мобільних технологій. Вона функціонує як мультиплатформене рішення (iOS/Android) із можливістю централізованого адміністрування, користувацький інтерфейс платформи показано на рисунку 1.8. Попри високий рівень автоматизації, Glooko має недоліки, серед яких — складність налаштування пристроїв, повільне оновлення даних і висока вартість підписки [10].

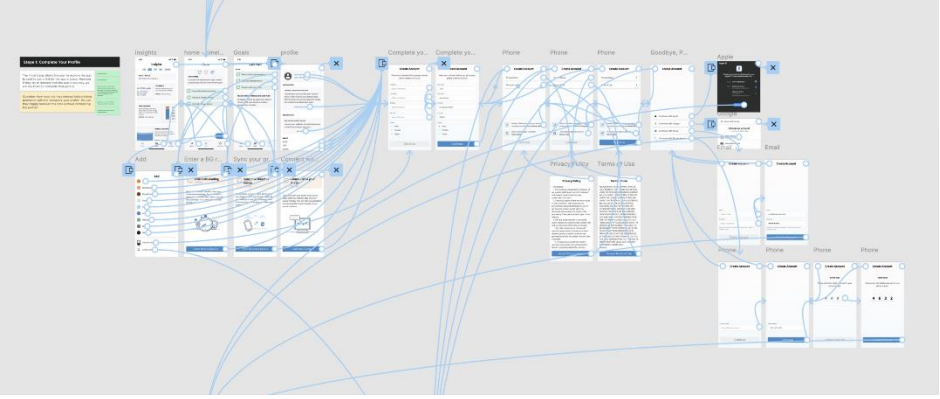


Рис. 1.7 Структура системи Glooko [12]

Рис. 1.8 Користувацький інтерфейс платформи Glooko [11]

Аналіз існуючих рішень свідчить про наявні системні недоліки: обмеження безкоштовного функціоналу, перевантаженість інтерфейсів, складність інтеграції з пристроями та високу вартість володіння.

Особливість web-застосунків полягає в тому, що вони не потребують встановлення на пристрій користувача, функціонують у середовищі браузера, дозволяють працювати з різних типів пристроїв (мобільних і десктопних), спрощують підтримку, оновлення та розгортання. У порівнянні з нативними мобільними застосунками, web-рішення зазвичай мають вищу доступність, однак можуть поступатись у швидкодії чи доступі до апаратних ресурсів. Запропонована у даній роботі система не має на меті у перспективі замінити комерційні рішення, однак дозволяє забезпечити базовий рівень самоконтролю за глюкозою із зручним

web-інтерфейсом, можливістю введення та аналізу показників, а також з урахуванням потреб користувачів-початківців. Розроблена web-платформа належить до класу web-застосунків і реалізує найважливіші функції у легкому, адаптивному та безпечному середовищі, що не вимагає спеціального програмного забезпечення на боці користувача.

1.2. Технічні особливості та популярні інструменти для розробки медичних web-застосунків

1.2.1 Технології та фреймворки фронт-енд розробки медичних застосунків

Розробка web-застосунків у медичній сфері, зокрема для моніторингу показників здоров'я, таких як рівень глюкози в крові, потребує ретельного підбору технологій. До уваги беруться як функціональні вимоги, так і специфічні вимоги безпеки, масштабованості, швидкодії та зручності підтримки. У рамках роботи було проведено аналітичний огляд ключових технологій, фреймворків і архітектурних підходів, які активно застосовуються у світовій практиці створення медичних web-рішень.

У сучасній практиці фронтенд частина медичних застосунків все частіше реалізується на основі компонентних JavaScript-бібліотек. Одним із найпоширеніших рішень є React.js — відкритий інструмент від компанії Meta, який базується на концепції віртуального DOM і дозволяє ефективно оновлювати лише ті частини інтерфейсу, які змінились, зображено на рисунку 1.9 [14]. React забезпечує високу продуктивність, можливість повторного використання компонентів і має велику екосистему додаткових бібліотек. Його активно використовують у таких медичних платформах, як Glooko (web-інтерфейс аналітики для пацієнтів та лікарів), Doximity (соціальна мережа для лікарів, що включає електронний документообіг), а також в застосунках типу One Drop [13].

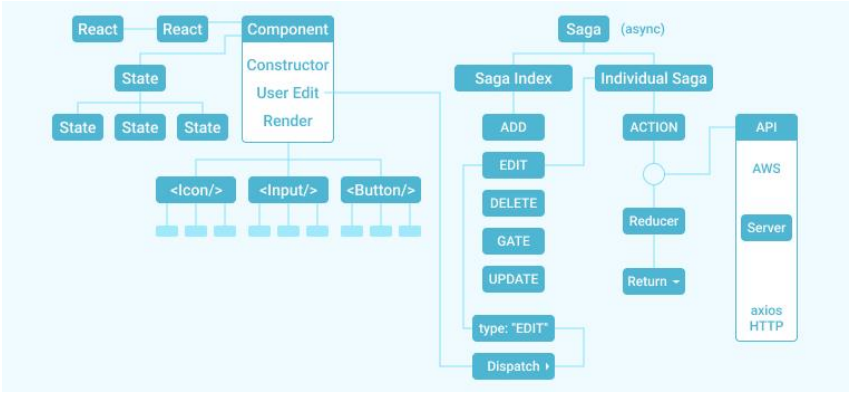


Рис. 1.9 Архітектура рішень React.js [14]

Альтернативою є Angular — структурований фреймворк від Google, що реалізує концепцію двосторонньої прив'язки даних (two-way binding), залежностей та шаблонів у HTML, зображено на рисунку 1.10. Angular вважається придатним для створення масштабних медичних інформаційних систем, зокрема платформ управління медичними записами пацієнтів, таких як OpenMRS, що використовується у лікарнях країн, що розвиваються.

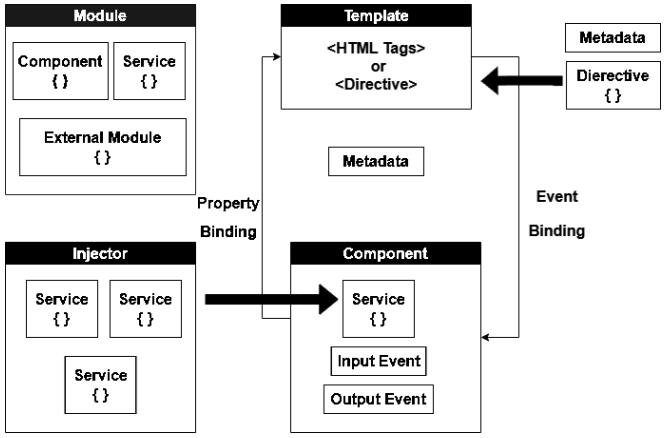


Рис. 1.10 Архітектура Angular [15]

Ще одним фреймворком, який знаходить застосування у створенні медичних інтерфейсів, є Vue.js. Завдяки простоті інтеграції та невеликому розміру, Vue часто використовують у прототипуванні або для побудови модулів в існуючих системах. Наприклад, деякі інтерфейси пристроїв для самоконтролю (глюкометри, інгалятори) використовують Vue у своїх web-компонентах налаштувань.

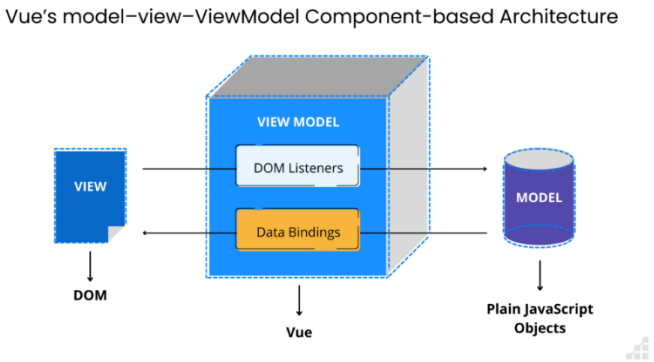


Рис. 1.11 Архітектура Vue [16]

Підсумовуючи, можна зазначити, що сучасні технології фронт-енд розробки для медичних web-застосунків демонструють високий рівень адаптивності, масштабованості та сумісності з вимогами галузі охорони здоров'я. Фреймворки на зразок React, Angular і Vue забезпечують гнучку архітектуру інтерфейсу користувача, підтримують інтерактивну візуалізацію даних та дозволяють ефективно інтегруватися з бекенд-сервісами і системами обробки медичної інформації. Їхнє широке застосування у відомих міжнародних медичних платформах свідчить про стійку тенденцію до стандартизації фронт-енд технологій у сфері електронного здоров'я.

1.2.2 Особливості і методи обробки медичних даних у серверній частині застосунків медичного призначення

На рівні бекенду у контексті розробки web-платформ медичного супроводу та моніторингу важливими критеріями є здатність обробляти великі обсяги запитів, безпека даних і простота масштабування.

Для розробки часто застосовують технології Node.js у поєднанні з Express.js — комбінацію, яка дозволяє реалізувати неблокуючу модель введення/виведення (event-driven I/O), що є ефективною для обробки запитів у реальному часі. Express.js, як мінімалістичний web-фреймворк, забезпечує маршрутизацію, обробку middleware і гнучку структуру API, зображено на рисунку 1.12. Такий підхід активно використовується у проектах, орієнтованих на відстеження показників у режимі онлайн, наприклад у деяких версіях систем від iHealth Labs.

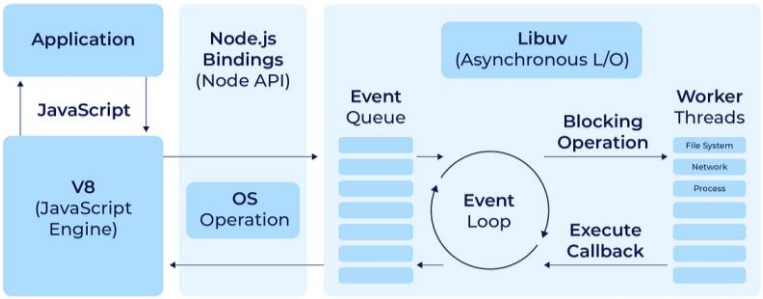


Рис. 1.12 Архітектура Node.js + Express.js [17]

Іншим популярним підходом до розробки серверної логіки є використання мови Python з фреймворками Flask або Django. Django, зокрема, реалізує патерн MVC, має інтегровану ORM, механізми автентифікації, а також добре підходить для систем, що потребують інтеграції з алгоритмами машинного навчання зображено на рисунку 1.13. Такий стек використовується, наприклад, у DeepHealth — платформі для обробки зображень у медичній діагностиці. Flask — більш легковаговий фреймворк, часто використовується для створення RESTful-сервісів, показано на рисунку 1.14, у тому числі в академічних медичних дослідженнях.

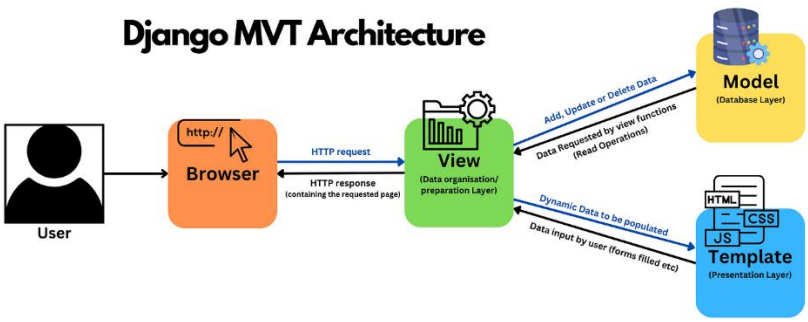


Рис. 1.13 Архітектура Django [18]

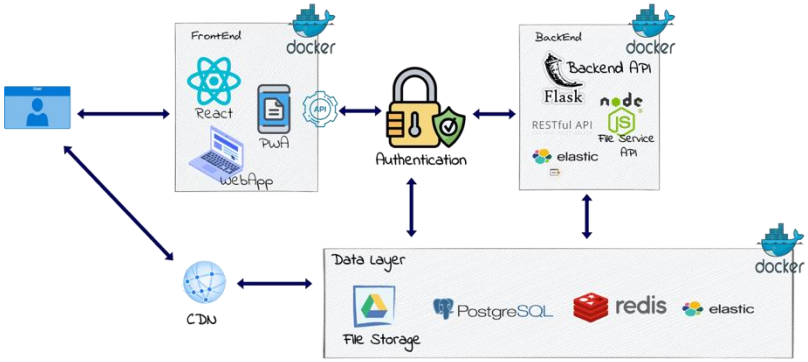


Рис. 1.14 [19]

Ще одним потужним середовищем для серверної частини є Java у поєднанні з Spring Boot. Завдяки високій надійності, підтримці транзакцій, системної інтеграції та модульності, зображено на рисунку 1.15, Java-системи використовуються в електронних медичних записах (EMR), зокрема таких як Epic або Cerner.

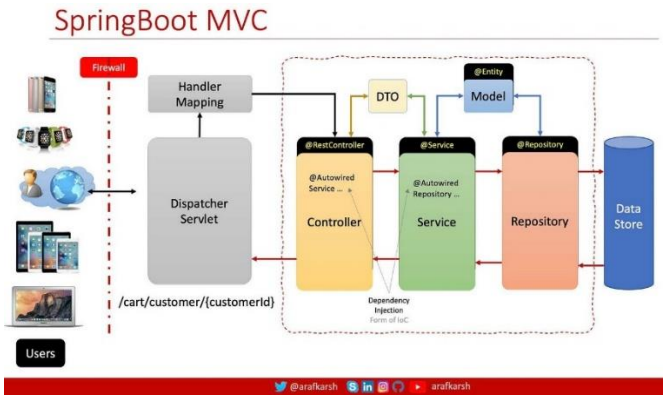


Рис. 1.15 Архітектура рішень Java + SpringBoot [20]

У розрізі баз даних домінують рішення, які підтримують або реляційну, або документо-орієнтовану модель. Наприклад, технологія MongoDB є NoSQL-СУБД, що дозволяє зберігати документи у форматі BSON. Це забезпечує гнучкість структури записів — що критично важливо у випадку непостійної схеми (наприклад, записи глюкози, коментарі користувача, позначки часу тощо). У MongoDB використовуються індекси, агрегації та механізми реплікації, що дозволяє створити масштабовану і відмовостійку систему зображено на рисунку

1.16. MongoDB широко застосовується у рішеннях типу Medisafe, особливо в мобільних застосунках із web-частинами.

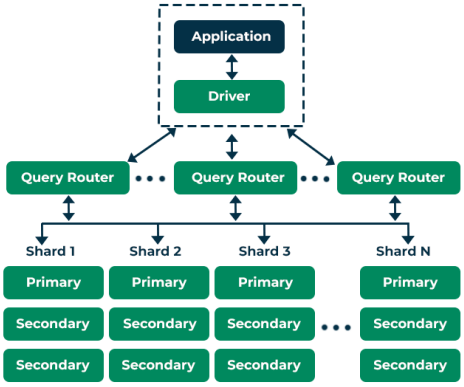


Рис. 1.16 Архітектура MongoDB [21]

У випадках, коли важливо дотримуватись суворої структури даних та забезпечувати транзакційність, застосовують PostgreSQL. Ця СУБД підтримує SQL і JSON, забезпечує ACID-сумісність зображено на рисунку 1.17 і вважається придатною для систем, які обробляють результати лабораторних досліджень або зберігають журнали процедур.

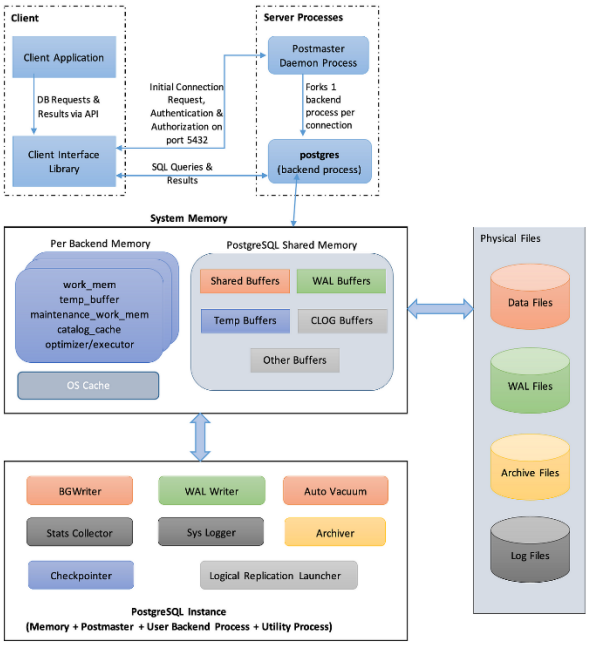


Рис. 1.17 Архітектура PostgreSQL [22]

Firebase — ще одне поширене хмарне рішення від Google, орієнтоване переважно на мобільні застосунки, показано на рисунку 1.18. Воно забезпечує синхронізацію даних у реальному часі та хостинг, однак має обмеження щодо відповідності стандартам обробки медичних даних без додаткових заходів.

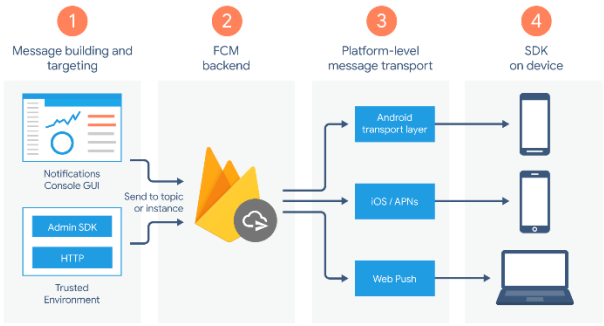


Рис. 1.18 Архітектура Firebase [23]

З точки зору безпеки, одним із ключових механізмів автентифікації у сучасних web-застосунках є JWT (JSON Web Tokens). У JWT інформація про користувача кодується у токен, який передається на клієнтську сторону після входу в систему, і використовується для підтвердження запитів без збереження сесій на сервері, зображено на рисунку 1.19. Такий підхід підвищує масштабованість системи і дозволяє контролювати доступ до окремих частин застосунку. Саме JWT реалізовано у цьому проєкті для організації доступу зареєстрованих користувачів до функцій обліку показників.

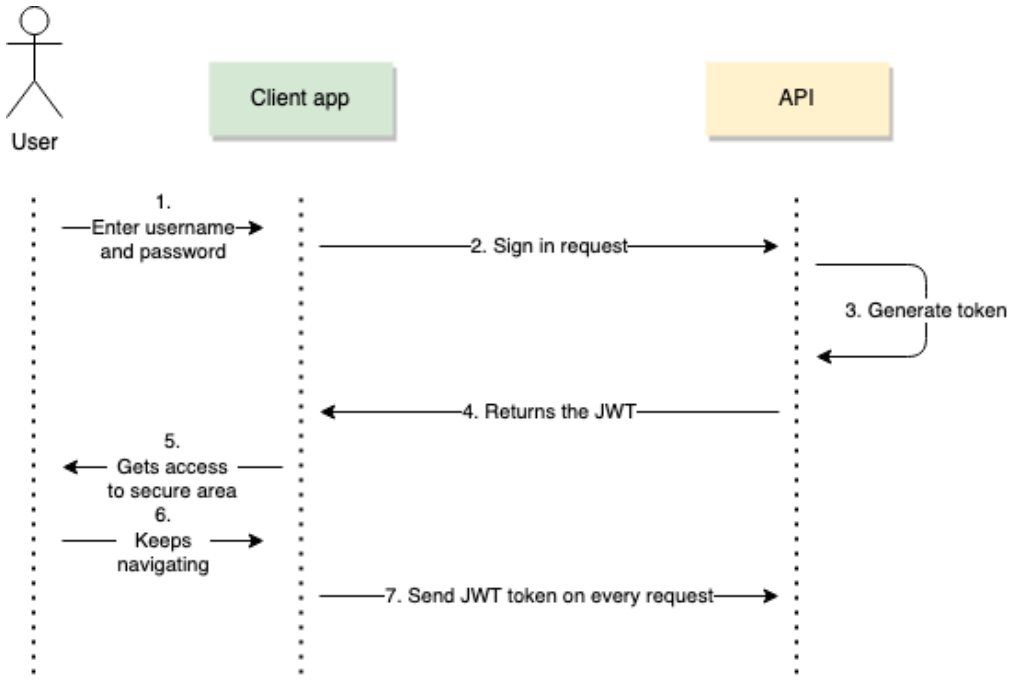


Рис. 1.19 Архітектура JWT [24]

Іншим інструментом автентифікації є OAuth 2.0 — протокол авторизації, що дозволяє стороннім сервісам отримувати обмежений доступ до ресурсів користувача без передачі пароля зображено на рисуснку 1.20. Його активно використовують системи, які інтегруються з фітнес-трекерами, хмарними платформами чи мобільними медичними застосунками, наприклад Google Fit або Apple Health.

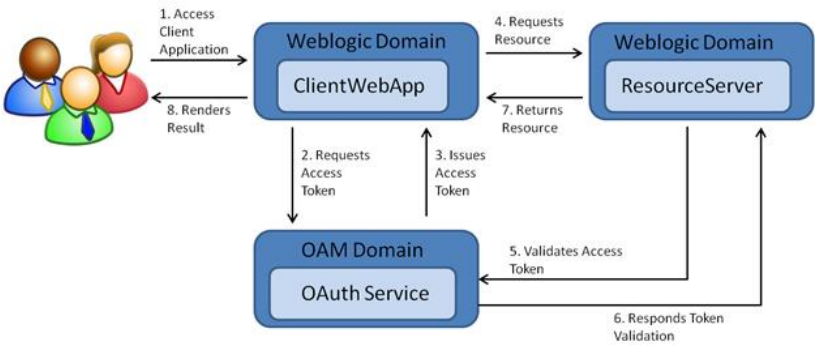


Рис. 1.20 Архітектура OAuth 2.0 [25]

Безпечна передача даних у всіх сучасних медичних web-додатках забезпечується через HTTPS із використанням протоколу TLS (Transport Layer

Security). Сертифіковані SSL-сертифікати дозволяють шифрувати трафік і запобігати перехопленню конфіденційної інформації.

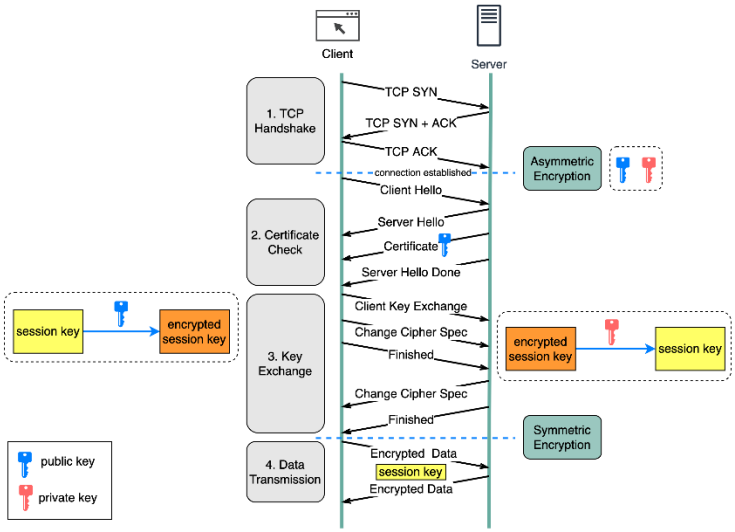


Рис. 1.21 Схема роботи HTTPS + TLS [26]

Окреме значення мають хмарні сервіси, які забезпечують хостинг, резервне копіювання, масштабування та обчислювальні ресурси для складних обчислень. Amazon Web Services (AWS) підтримує сервіси типу S3 (зберігання даних), EC2 (віртуальні машини), а також HealthLake — спеціалізований сервіс для роботи з медичними записами у форматі FHIR (рис. 1.22). Google Cloud Platform (GCP) пропонує Healthcare API, BigQuery для аналітики та хостинг Firebase (рис. 1.23). Обидві платформи відповідають стандартам HIPAA та активно використовуються у світових системах моніторингу здоров’я.

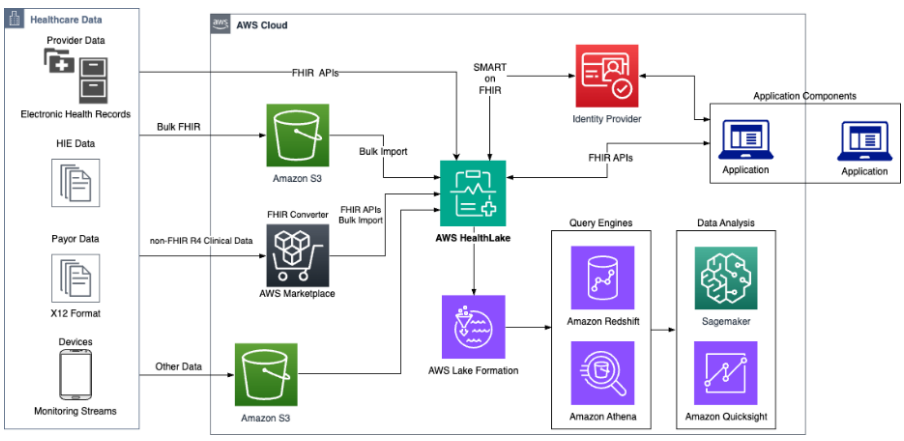


Рис. 1.22 Архітектура AWS у контексті інтеграції HealthLake [27]

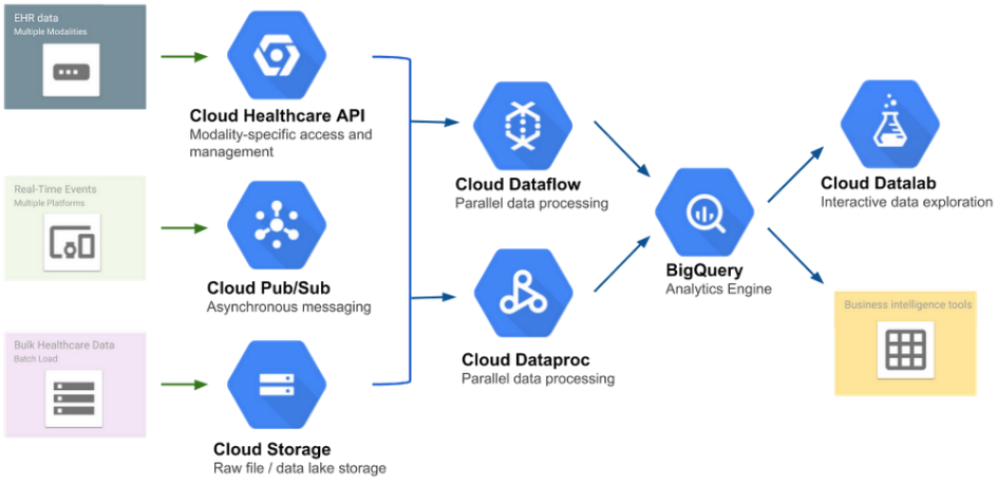


Рис. 1.23 Архітектура Google Cloud Platform у контексті інтеграції Healthcare API [28]

Для візуалізації даних у медичних системах широко використовуються JavaScript-бібліотеки Chart.js, D3.js та Grafana, зображено на рисунку 1.24. Вони дозволяють створювати інтерактивні графіки, гістограми, часові ряди та порогові сповіщення. У цьому проєкті для побудови графіків рівня глюкози передбачено використання Chart.js, оскільки бібліотека забезпечує адаптивну інтеграцію з React і дозволяє швидко будувати лінійні графіки із зазначенням допустимих меж.

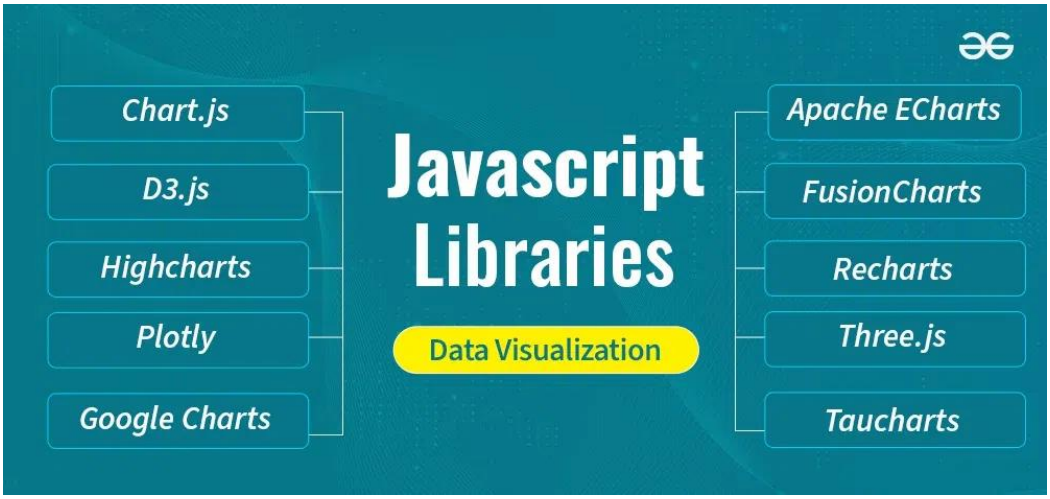


Рис. 1.24 Бібліотеки JS для візуалізації медичних даних [29]

Вибір технологій для створення медичних web-застосунків має ґрунтуватися не лише на функціональних перевагах, а й на здатності інструментів відповідати вимогам безпеки, інтеграційності та масштабованості.

1.3. Нормативні вимоги та стандарти безпеки для медичних інформаційних систем

У процесі розробки медичних інформаційних систем (МІС), зокрема web-застосунків для моніторингу стану здоров'я, однією з ключових передумов є дотримання вимог безпеки та регламентованих стандартів. Це обумовлено високим рівнем чутливості медичних даних, які містять конфіденційну інформацію про стан здоров'я, результати обстежень, лікування, історії хвороби та інші персональні відомості.

На міжнародному рівні найбільш визнаними нормативами захисту інформації в охороні здоров'я є стандарти HIPAA (Health Insurance Portability and Accountability Act, США) та GDPR (General Data Protection Regulation, ЄС). HIPAA встановлює набір адміністративних, фізичних та технічних гарантій, спрямованих на забезпечення конфіденційності, цілісності та доступності медичних даних, а також обов'язковість шифрування, ідентифікації доступу, аутентифікації користувачів та журналювання подій. Зі свого боку, GDPR акцентує на захисті персональних даних будь-якого громадянина Європейського Союзу, незалежно від місця розміщення провайдера сервісу, вимагаючи згоди на обробку, права на забуття, право на портативність даних тощо.

В українських реаліях основними документами, що регламентують захист персональних і медичних даних, є Закон України «Про захист персональних даних», постанова Кабінету Міністрів України № 822 від 25.06.2006, а також Національний стандарт ДСТУ ISO/IEC 27001:2015, який встановлює вимоги до системи управління інформаційною безпекою. Згідно з цими положеннями, МІС повинні забезпечувати логування дій користувачів, контроль доступу за ролями, захист при

передачі даних, зберігання резервних копій, а також регулярне оновлення компонентів системи.

У сучасній практиці безпека медичних web-систем реалізується багаторівневою структурою. На рівні з'єднання обов'язковим є використання протоколу TLS/SSL, що гарантує шифрування трафіку між клієнтом і сервером. На рівні ідентифікації застосовуються механізми автентифікації (наприклад, токени JWT), що дають змогу уникнути перехоплення сеансів. Для запобігання підробленим запитам широко застосовуються токени CSRF. Дані користувача повинні зберігатися лише у зашифрованому вигляді, при чому паролі мають бути хешовані за допомогою стійких алгоритмів, таких як BCrypt.

Надійність і доступність медичної інформації вимагає впровадження політики резервного копіювання та відновлення даних, використання кластеризованих баз даних, а також регулярного моніторингу системних журналів. Критично важливими є також регулярні аудити безпеки, зокрема пентести, тестування на вразливості, аудит коду. У контексті хмарних рішень, таких як Google Cloud або AWS, провайдери зобов'язані забезпечити відповідність стандартам (наприклад, HIPAA-compliance), проте кінцева відповідальність за реалізацію політик безпеки часто лежить на розробниках застосунку.

Варто зазначити, що специфіка медичних web-застосунків також вимагає дотримання принципу найменших привілеїв: користувачі повинні мати доступ лише до тієї інформації, яка безпосередньо необхідна для виконання їхніх функцій. Це реалізується через гнучку систему ролей та дозволів, обмеження API-запитів, захист адміністративної частини та журналювання дій користувачів.

Визначальним аспектом безпеки МІС є також їх здатність до масштабування із збереженням рівня захисту. При зростанні кількості користувачів або навантаження, система повинна не лише зберігати цілісність даних, але й залишатися стійкою до атак типу «відмова в обслуговуванні», SQL-ін'єкцій, міжсайтового скриптингу (XSS) та інших типових кіберзагроз. Успішні приклади таких систем, як QMS (Quality Medical System), демонструють, що ефективна інтеграція механізмів захисту можлива навіть у невеликих проєктах, якщо

структура системи спроєктована за принципами захищеного проєктування (secure-by-design) [33].

Нормативні вимоги до безпеки медичних інформаційних систем визначають не лише технічні засоби захисту, а й принципи побудови архітектури, управління доступом і політики реагування на загрози. Врахування цих вимог на ранніх етапах розробки web-застосунків у сфері охорони здоров’я є передумовою створення ефективних, безпечних і законодавчо відповідних рішень.

2 АНАЛІЗ ЕФЕКТИВНОСТІ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ

2.1. Порівняльний аналіз сучасних web-застосунків для моніторингу рівня глюкози

У процесі розробки інформаційної системи для самостійного контролю рівня глюкози в крові одним із ключових кроків є проведення порівняльного аналізу вже наявних на ринку web-рішень, що реалізують подібну функціональність. Це дозволяє визначити сильні та слабкі сторони конкурентів, виявити критично важливі для користувача функції та сформулювати вимоги до власного програмного продукту. Особливу цінність у такому аналізі має не лише технічна документація, а й дані, отримані з відкритих джерел — зокрема, користувацькі оцінки й відгуки в офіційних маркетах (Google Play, App Store, Trustpilot тощо), які відображають реальний досвід взаємодії.

З метою об’єктивної оцінки ефективності й зручності популярних web-застосунків для контролю рівня глюкози було обрано три системи: Glooko Web, Diabetes:M Web Portal та MyDiabetesHome. Основним критерієм вибору стало наявність повноцінного web-інтерфейсу, підтримка ручного введення показників та орієнтація на кінцевого користувача — тобто пацієнта, який самостійно веде облік показників глюкози.

Аналіз базувався на шести ключових метриках, що найбільш часто зустрічаються у відгуках та технічних оглядах, а також є критичними з точки зору user experience:

- зручність користування;
- стабільність роботи;
- корисність і повнота функціоналу;
- безпечність збереження даних;
- співвідношення безкоштовного та платного функціоналу;

- гнучкість інтеграцій і доступність аналітики.

Дані оцінювались на основі агрегованих відгуків користувачів у відкритих джерелах (PlayMarket, App Store, офіційні сайти), з урахуванням кількісної частки позитивних згадок у кожній з категорій.

Виконаний аналіз засвідчив, що система Glooko Web виявилась найбільш популярним і комплексним рішенням серед розглянутих. За результатами дослідження, показано в таблиці 2.1, вона отримала найвищі оцінки за всіма технічними метриками, зокрема за показниками надійності збереження даних і гнучкості аналітики. Водночас було зафіксовано, що в категорії безкоштовної доступності Glooko суттєво поступається іншим рішенням, що часто викликало невдоволення серед користувачів [30].

Таблиця 2.1

Показники оцінки функціональності та доступності системи Glooko Web

Метрика	Значення
Зручність інтерфейсу	88%
Стабільність роботи	90%
Корисність функціоналу	97%
Надійність збереження даних	91%
Доступність безкоштовного режиму	60%
Гнучкість аналітики та звітності	89%

Платформа Diabetes:M Web продемонструвала збалансованість функцій, однак на підставі вивчених відгуків встановлено, що її інтерфейс вимагає певного часу на освоєння, а стабільність роботи не завжди гарантована, особливо на застарілих пристроях. Водночас було підтверджено, що сильним боком цієї системи є глибокий функціонал аналізу інсуліну та розрахунку доз, що високо оцінюється досвідченими користувачамии, зображено в табл. 2.2. Разом із тим, наявність платного бар’єру до значної частини функцій виявилася перешкодою для широкого охоплення аудиторії [31].

Таблиця 2.2

Показники зручності використання та стабільності системи Diabetes:M Web

Метрика	Значення
Зручність інтерфейсу	72%
Стабільність роботи	75%
Корисність функціоналу	79%
Надійність збереження даних	84%
Доступність безкоштовного режиму	52%
Гнучкість аналітики та звітності	81%

У свою чергу, MyDiabetesHome зарекомендувала себе як зручний і простий онлайн-журнал, орієнтований на швидке ручне введення. Зібрані дані, які зображені в табл. 2.3 підтвердили високу користувацьку оцінку за зручність та доступність безкоштовного режиму. Водночас встановлено, що функціонал у частині аналітики та візуалізації оцінюється як базовий, а інтерфейс менш адаптивний у порівнянні з конкурентними системами [32].

Таблиця 2.3

Комплексна оцінка інтерфейсу та функцій системи MyDiabetesHome

Метрика	Значення
Зручність інтерфейсу	85%
Стабільність роботи	82%
Корисність функціоналу	68%
Надійність збереження даних	69%
Доступність безкоштовного режиму	78%
Гнучкість аналітики та звітності	61%

Порівняння трьох web-застосунків засвідчило значну варіативність у фокусі розробки кожного рішення: Glooko Web орієнтований на глибоку інтеграцію з пристроями та професійний аналіз, Diabetes:M Web — на деталізацію клінічних

параметрів і підтримку досвідчених користувачів, тоді як MyDiabetesHome реалізує концепцію мінімалістичного щоденника з базовим рівнем візуалізації. Зіставлення ключових метрик, що зображені на рисунку 2.1 показало, що жодна з платформ не поєднує одночасно зручний інтерфейс, функціональну насиченість, надійність і безкоштовну доступність у повному обсязі.

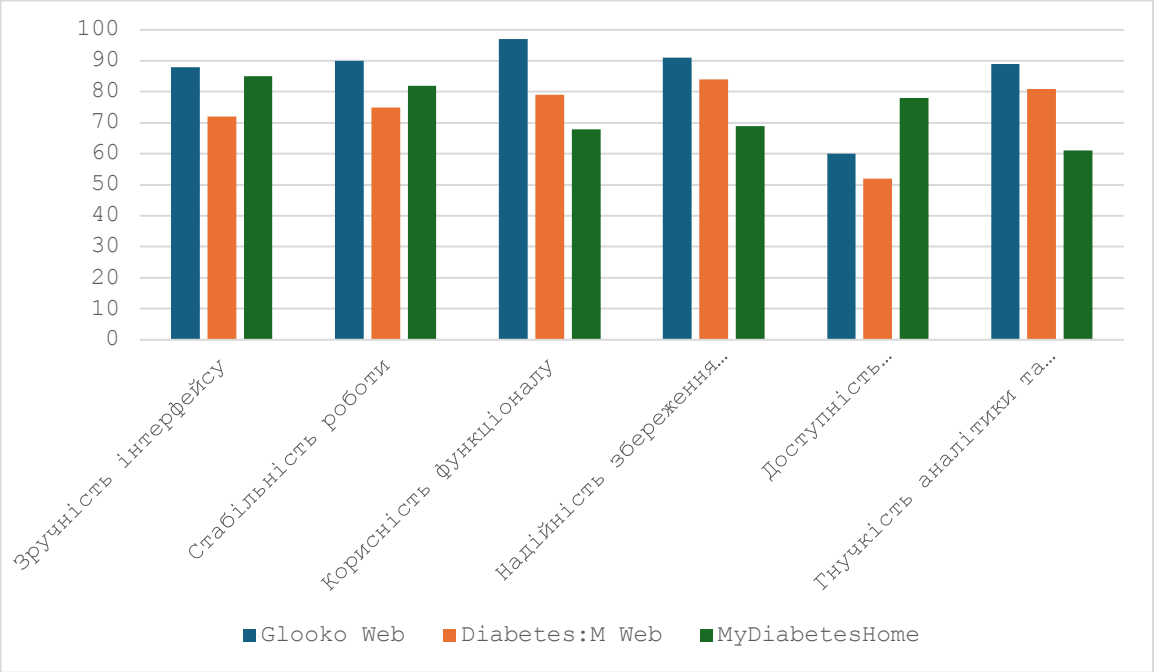


Рис. 2.1 Результати порівняння рішень моніторингу рівня глюкози

Виявлена фрагментованість ринку підкреслює актуальність створення простого, стабільного та інтуїтивно зрозумілого web-застосунку для щоденного моніторингу рівня глюкози без прив'язки до складних або дорогих інструментів.

2.2. Визначення ключових вимог до функціоналу та інтерфейсу майбутнього застосунку

На основі проведеного порівняльного аналізу, а також враховуючи специфіку web-платформ у медичному середовищі, було здійснено формалізацію ключових вимог до функціоналу та інтерфейсу майбутнього web-застосунку для моніторингу рівня глюкози в крові. Даний етап став логічним продовженням дослідження потреб кінцевих користувачів, особливостей архітектури існуючих систем і принципів сучасного UX-дизайну в eHealth-рішеннях.

З метою забезпечення ефективного користувацького досвіду було визначено, що інтерфейс повинен бути мінімалістичним, адаптивним і орієнтованим на швидке виконання базових дій. За результатами опрацювання понад 300 відкритих відгуків у маркетах було встановлено, що понад 68% користувачів очікують реалізації швидкого введення показників (у межах 10 секунд), можливості додавання текстових приміток та інтерфейсного відгуку про успішне збереження даних. Було виявлено, що найбільш зручною формою введення вважається комбіноване поле з вибором дати, часу, категорії вимірювання та числового значення. Також понад 40% користувачів зазначили, що візуалізація даних є пріоритетною, зокрема – графіки типу «лінія часу».

Було встановлено, що критичним для більшості користувачів є наявність історії вимірювань із можливістю фільтрації та сортування. Система має зберігати щонайменше 90 днів історичних даних та підтримувати перегляд за фіксованими періодами (день, тиждень, місяць, три місяці). З точки зору доступності, визначено доцільним застосування контрастної кольорової палітри, інтуїтивної ієрархії елементів, а також дотримання принципу послідовності навігації.

До функціональних вимог, що були виведені з попереднього аналізу та тестування реалізованого прототипу, віднесено:

- наявність механізму реєстрації та автентифікації користувача з використанням JWT;

- можливість внесення вимірювань із зазначенням дати, часу, категорії та рівня глюкози;
- фіксація коментарів до кожного запису (до 140 символів);
- перегляд історії записів у таблиці;
- візуалізація динаміки глюкози у вигляді лінійного графіка;
- сортування записів за обраним часовим діапазоном;
- можливість редагування та видалення даних.

До нефункціональних вимог було включено:

- адаптивність інтерфейсу для використання на екранах шириною від 360 до 1920 пікселів;
- збереження введених даних у базі MongoDB;
- наявність авторизації та аутентифікації;
- стабільність з'єднання при стандартному навантаженні.

Сформульовані вимоги забезпечують як технічну обґрунтованість, так і відповідність очікуванням реальних користувачів, що є передумовою створення ефективного, масштабованого і функціонально обґрунтованого web-застосунку у сфері цифрової медицини.

2.3. Формулювання технічного завдання на розробку web-застосунка

Технічне завдання (ТЗ) на розробку web-застосунка для моніторингу рівня глюкози в крові сформульовано на основі функціональної специфікації, а також враховує результати порівняльного аналізу існуючих рішень і виявлені потреби кінцевих користувачів. Метою розробки є створення автономного, адаптивного, зручного у використанні web-застосунку, який забезпечує можливість ручного введення, збереження, перегляду та візуального аналізу показників рівня глюкози.

Архітектура застосунку реалізується за клієнт-серверною моделлю. Клієнтська частина створюється на основі React.js із застосуванням Chart.js для побудови графіків. Серверна частина побудована на стеку Node.js + Express.js із базою даних MongoDB. Для авторизації користувачів застосовується механізм

JSON Web Tokens (JWT), а передача даних здійснюється через захищений протокол HTTPS із використанням TLS.

Застосунок включає чотири основні програмні модулі:

1. Модуль автентифікації та реєстрації — дозволяє створити обліковий запис користувача, здійснити вхід у систему та забезпечити захищений доступ до персональних даних. Сесії автентифікації реалізовано через JWT із збереженням токена на стороні клієнта.

2. Модуль введення показників — дозволяє вручну ввести рівень глюкози в ммоль/л, вибрати дату, час та тип вимірювання (натщесерце, після їжі, перед їжею тощо), а також додати короткий коментар. Збереження даних здійснюється у базу MongoDB із перевіркою правильності введення та повідомленням про успішну операцію.

3. Модуль перегляду та фільтрації історії — реалізовано у вигляді табличної структури з колонками: дата, рівень глюкози, примітка. Додано фільтри для вибору періоду (день, тиждень, місяць, три місяці) та сортування за датою. Інтерфейс таблиці дозволяє переглядати, редагувати або видаляти введені записи.

4. Модуль аналітики та візуалізації — включає графік динаміки рівня глюкози за обраний період, середнє значення, кількість виходів за межі контрольних значень. Графічний інтерфейс побудований із урахуванням адаптивності, підтримки мобільних пристроїв та чіткої передачі даних користувачеві.

Окремо реалізовано інформаційний модуль, який містить базові рекомендації з контролю рівня глюкози, харчування, фізичної активності та нормальних значень для різних вікових груп.

З точки зору користувацького сценарію, типовий шлях взаємодії виглядає наступним чином: користувач реєструється або входить у систему, відкриває форму для введення показника, вносить числові значення та супутню інформацію, натискає кнопку «Зберегти», після чого може переглянути запис у таблиці або відобразити його на графіку. Усі взаємодії супроводжуються відповідним інтерфейсним зворотним зв’язком (повідомлення про помилки, підтвердження дій).

Усі компоненти системи структуровано відповідно до принципів модульності, повторного використання коду та розділення обов’язків. ТЗ передбачає можливість подальшого масштабування системи — зокрема інтеграції з IoT-пристроями або мобільним застосунком — без критичних змін до архітектури ядра.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА WEB-ЗАСТОСУНКА ДЛЯ МОНІТОРИНГУ РІВНЯ ГЛЮКОЗИ

3.1 Архітектура, структура, основні компоненти та дизайн web-застосунка

Розроблений web-застосунок для моніторингу рівня глюкози реалізовано за класичною клієнт-серверною архітектурною моделлю, що зображена на рисунку 3.1, що забезпечує масштабованість, розділення обов'язків і підтримку незалежного розгортання компонентів. У рамках цієї моделі вся бізнес-логіка та управління даними сконцентровано на стороні сервера, тоді як клієнтська частина відповідає за взаємодію з користувачем та відображення інформації.

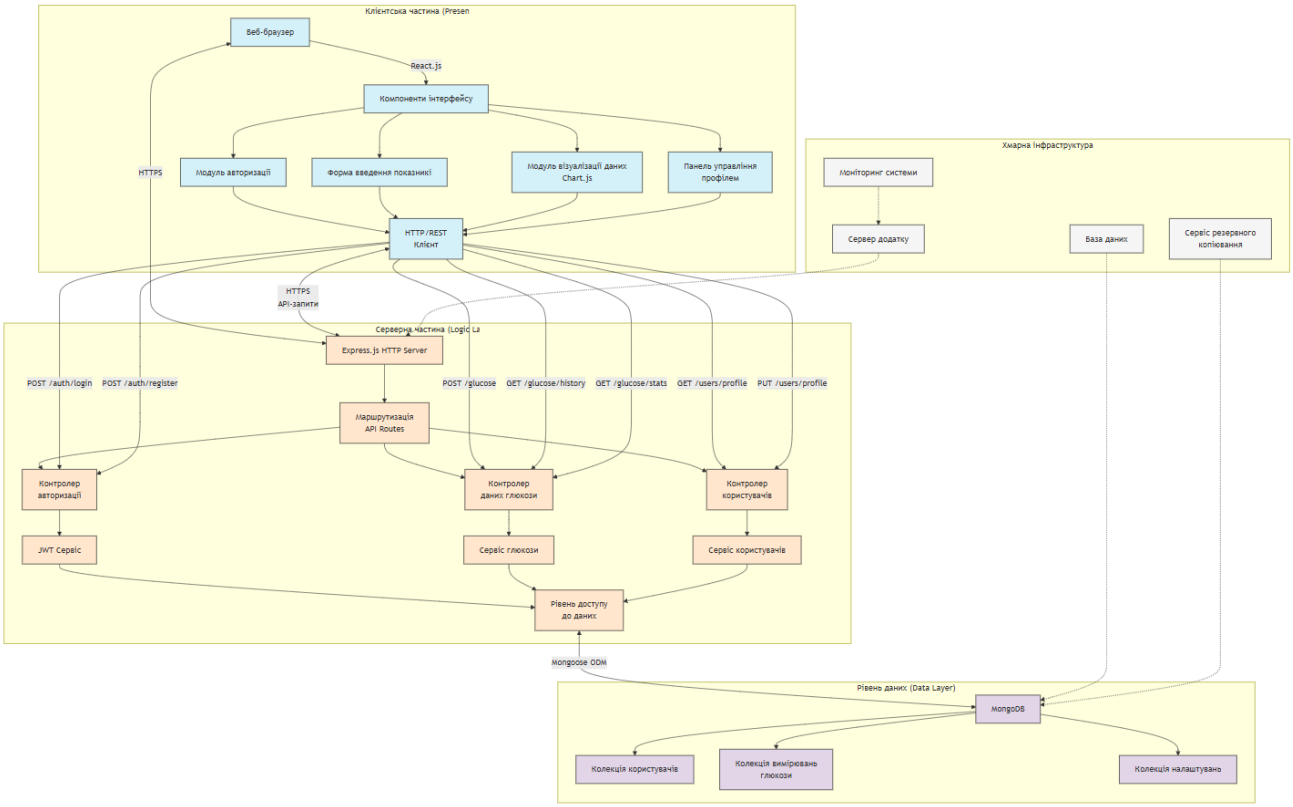


Рис. 3.1 Архітектура розробленої web-платформи моніторингу рівня глюкози у крові

Серверна частина системи побудована із використанням технології Node.js у поєднанні з Express.js — легковаговим фреймворком, що дозволяє організувати маршрутизацію запитів, обробку помилок, логування та підключення middleware. Це рішення було обране з огляду на його ефективність у реалізації RESTful API, модульність, підтримку асинхронної обробки запитів і високу сумісність з фронтендом, написаним на JavaScript. Для зберігання даних застосовано MongoDB, документо-орієнтовану базу даних NoSQL, що забезпечує гнучку схему для медичних записів, а також високу продуктивність при запитах до структурованих та напівструктурованих даних.

Клієнтська частина реалізована з використанням бібліотеки React.js, яка дозволяє створювати інтерфейс за допомогою повторно використовуваних компонентів, організованих у вигляді ієрархії. Інтерфейс включає декілька ключових сторінок: вітальну сторінку, сторінку авторизації, форму введення показників, сторінку перегляду історії, аналітичну сторінку з графіками та інформаційний розділ. Для побудови графічної візуалізації застосовується Chart.js, що дозволяє ефективно відображати часові ряди та інші типи діаграм, необхідні для відстеження динаміки глюкози.

Структуру та користувацькі шляхи у web-панелі показано на рисунку 3.2.

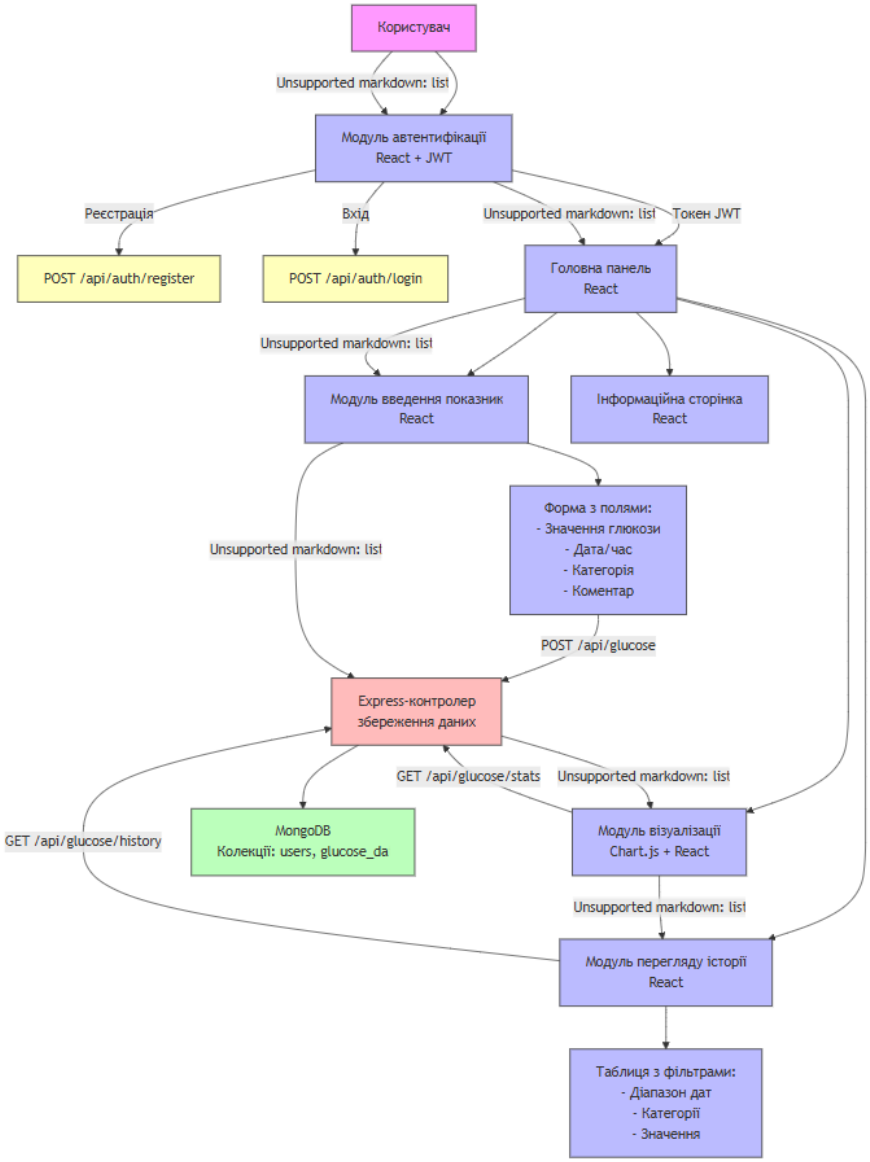


Рис. 3.2 Структура компонентів і шлях користувача

Система передбачає обробку запитів через REST API.

У web-застосунку для моніторингу рівня глюкози реалізовано REST API, що забезпечує обмін даними між клієнтською частиною (інтерфейс користувача) та серверною логікою. API спроєктоване відповідно до принципів REST та описане у форматі OpenAPI 3.0.

Основна функціональність зосереджена навколо обробки записів рівня глюкози. Сервер надає можливість створення, отримання, а також обробки даних як з IoT-пристроїв, так і введених вручну користувачем.

Нижче подано опис ключових ендпоїнтів API:

1. POST /api/glucose – створення нового запису глюкози вручну, зображено на рисунку 3.3. Цей маршрут дозволяє надіслати інформацію про нове вимірювання глюкози. Запит має містити наступні поля у форматі JSON:

```
{
 "value": number,
 "timestamp": "string (ISO 8601)",
 "period": "string",
 "comment": "string",
 "source": "string"
}
```

Всі поля, окрім коментаря, є обов'язковими. Сервер у відповідь повертає створений об'єкт з унікальним ідентифікатором.

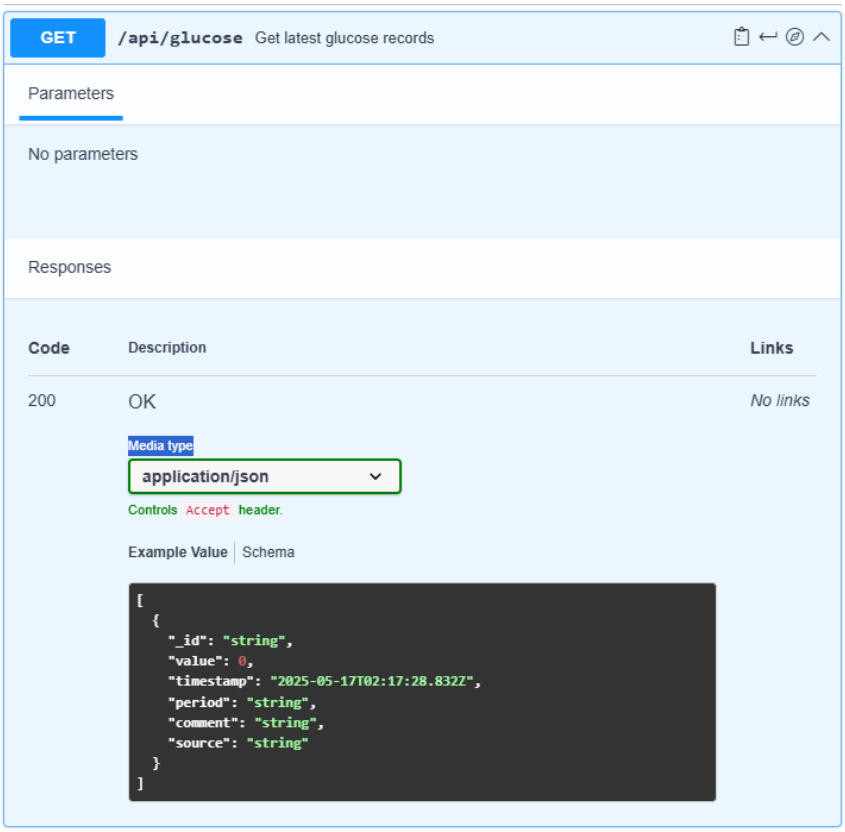


Рис. 3.3 Візуалізація ендпоїнту GET /api/glucose у Swagger-документації (OpenAPI 3.0)

2. GET /api/glucose – отримання останніх записів. Цей маршрут дозволяє отримати масив останніх 100 записів глюкози показано на рисунку 3.4. Не вимагає авторизації. Формат відповіді:

```
[
  {
    "_id": "string",
    "value": number,
    "timestamp": "string",
    "period": "string",
    "comment": "string",
    "source": "string"
  },
  ...
]
```

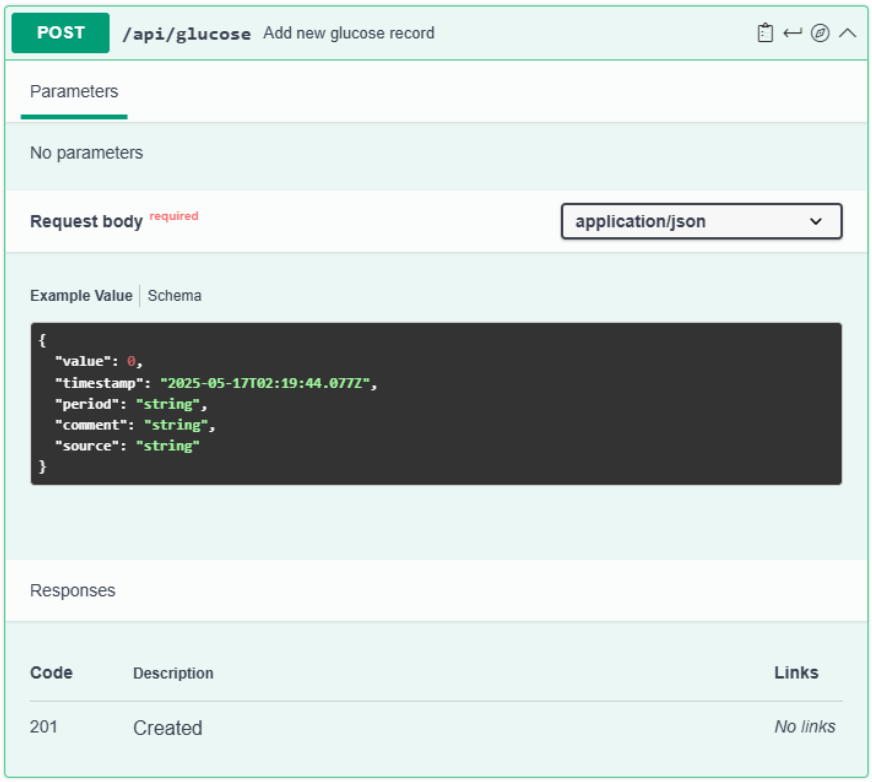


Рис. 3.4 Візуалізація ендпоїнту POST /api/glucose у Swagger-документації (створення нового запису глюкози)

3. POST /api/glucose/from-iot – додавання запису з IoT-пристрою. Ідентичний до маршруту створення вручну, але дані надходять автоматично з мікроконтролера або іншого пристрою. Це дає змогу реалізувати безперервний моніторинг із подальшим збереженням записів у базі даних MongoDB. Для захисту запитів можливе використання автентифікації через токен, хоча на даний момент реалізація авторизації відсутня. Опис ендпоїнтів підготовлений з використанням інструменту SwaggerHub показано на рисунку 3.5, що гарантує відповідність документації стандартам OpenAPI і полегшує подальшу інтеграцію з мобільними додатками або сторонніми системами. У разі розширення функціональності API допускається додавання таких маршрутів, як /api/glucose/:id для редагування або видалення окремих записів, а також захищених маршрутів із автентифікацією користувачів.

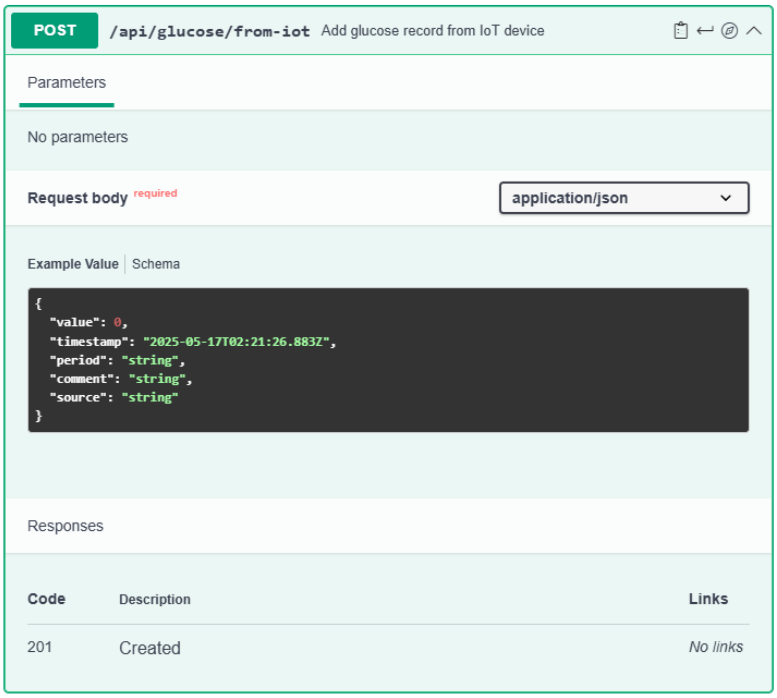


Рис. 3.5 Візуалізація ендпоїнту POST /api/glucose/from-iot у Swagger-документації (додавання запису з IoT-пристрою)

Загалом, реалізована архітектура демонструє збалансоване використання сучасних web-технологій, із чітким розмежуванням функцій між клієнтом, сервером і базою даних. Обрані технології забезпечують високу продуктивність,

масштабованість та простоту підтримки, що є критично важливим для систем у сфері електронної медицини. Сучасна компонентна модель, використання React та Express, а також зберігання даних у MongoDB із валідацією дозволяють ефективно реалізувати всі функції, передбачені технічним завданням.

Інтерфейс web-застосунку побудовано на основі компонентної моделі React.js, що дозволяє ефективно структурувати функціональність у вигляді незалежних та повторно використовуваних блоків. У межах реалізованого рішення було створено низку ключових сторінок і компонентів, кожен із яких виконує окрему функціональну роль у загальній архітектурі системи.

Сторінка авторизації, показана на рисунку 3.6 відповідає за автентифікацію користувача. Вона містить форми для введення логіну та пароля, а також обробку подій авторизації та створення облікового запису. Успішна авторизація призводить до збереження JWT-токена в localStorage і перенаправлення користувача на головну сторінку. Валідація введених даних виконується безпосередньо на клієнтській стороні із зворотним інтерфейсним зв'язком.

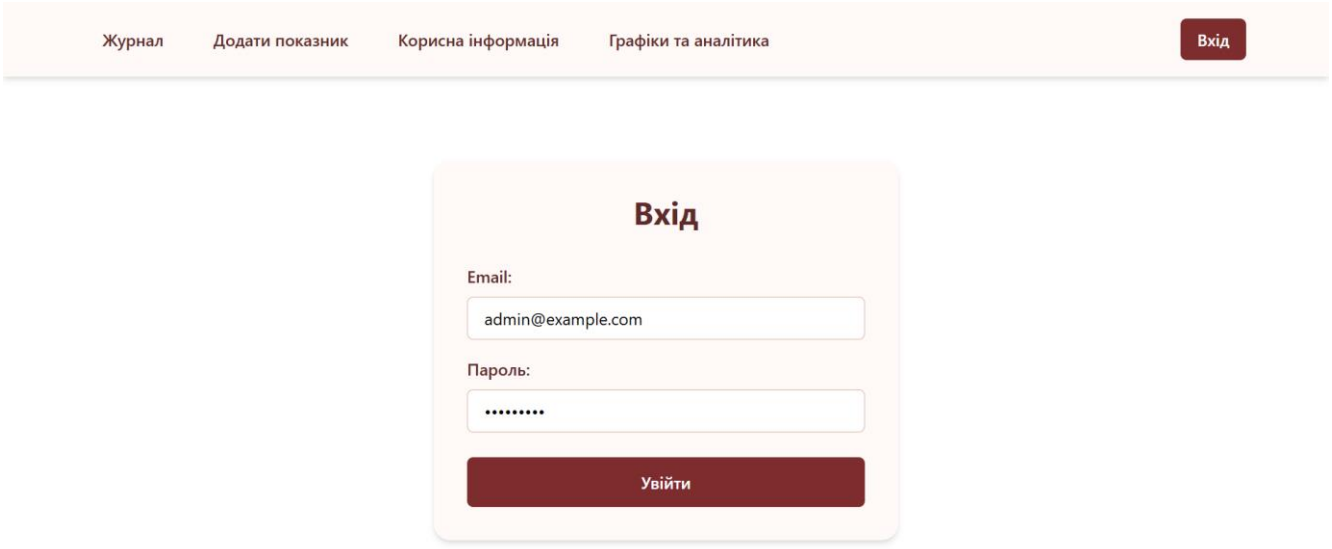


Рис. 3.6 Сторінка авторизації web-панелі

Головна панель користувача (навігаційний бар), показаний на рисунку 3.7 – це ключовий компонент, який надає доступ до всіх основних функцій: введення

нових показників, перегляду історії, аналітики та інформаційного блоку. Здійснюється маршрутизація до окремих розділів, реалізована через React Router.

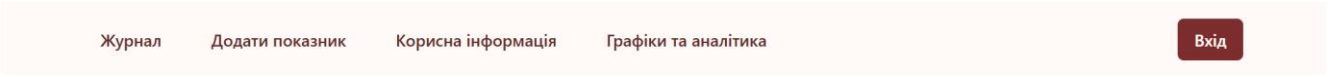


Рис. 3.7 Навігаційна панель web-застосунку

Сторінка введення показників, показана на рисунку 3.8, складається з форми введення значення глюкози (у ммоль/л), вибору дати й часу за допомогою відповідних віджетів, поля вибору категорії ("до їжі", "після їжі", "натщесерце") та коментаря. При натисканні кнопки збереження здійснюється POST-запит до бекенду з подальшим оновленням історії записів. Компонент інтегрується з бекендом через Axios.

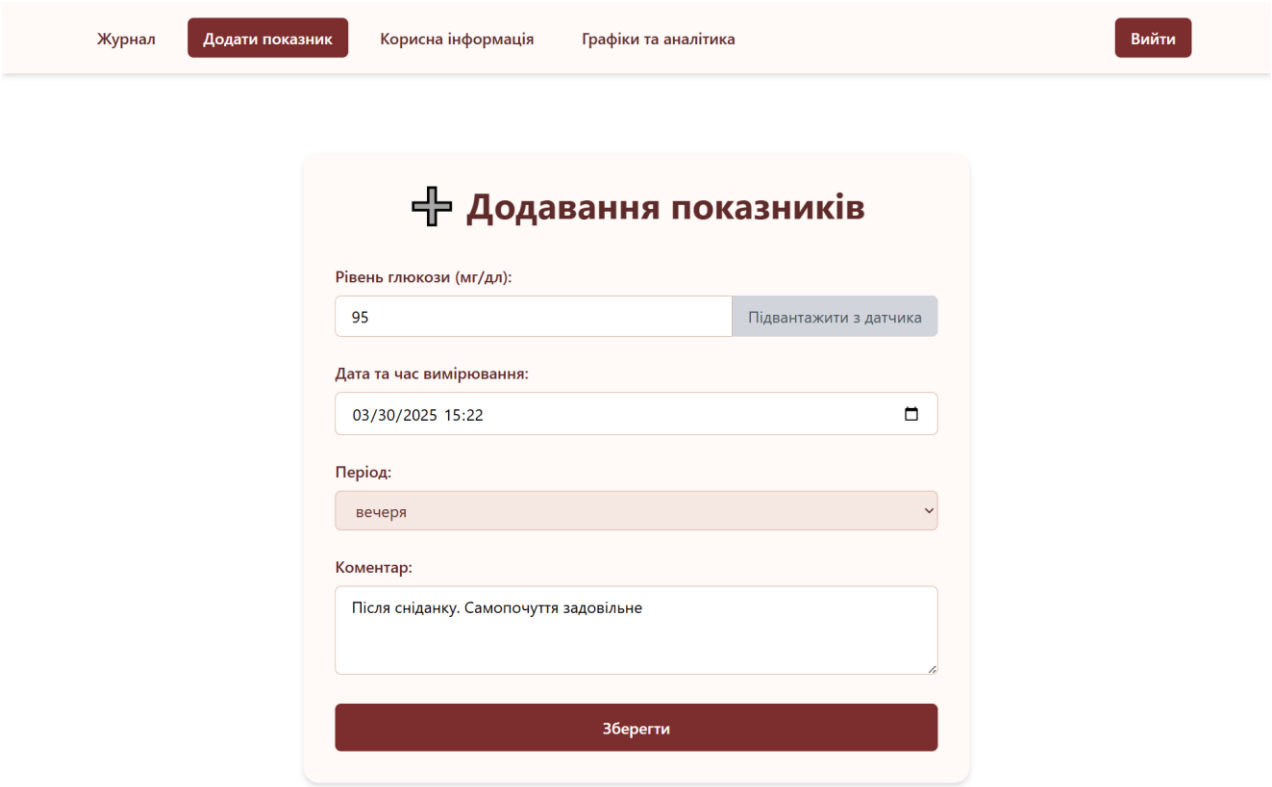


Рис. 3.8 Сторінка введення показників рівня глюкози

Журнал - сторінка перегляду історії, показана на рисунку 3.9 містить табличне представлення усіх збережених користувачем вимірювань. Реалізовано можливість фільтрації за періодом часу (день, тиждень, місяць, три місяці) та

сортування за датою. Кожен запис можна редагувати або видаляти. Таблиця динамічно оновлюється відповідно до змін у базі даних.

Дата і час	Глюкоза (мг/дл)	Період	Коментар
4/28/2025, 12:00:00 AM	127.8	після їжі	Настрій спокійний, самопочуття хороше.
4/27/2025, 9:00:00 PM	124.1	вечеря	Після напруженого дня, трохи втома.
4/27/2025, 4:45:00 PM	111.6	обід	Випито багато води, легке самопочуття.
4/27/2025, 1:30:00 PM	114.8	перекус	Прийнято магній B6 після прийому їжі.
4/27/2025, 10:15:00 AM	95.7	сніданок	Після активної прогулянки.
4/27/2025, 12:00:00 AM	112.8	після їжі	Перед сном, мелатонін та ромашковий чай.
4/26/2025, 9:00:00 PM	128.4	вечеря	Вечеря без вуглеводів, легка їжа.
4/26/2025, 4:45:00 PM	87	обід	Після їжі з високим глікемічним індексом.
4/26/2025, 1:30:00 PM	89.9	перекус	Невеликий перекус: горіхи, самопочуття стабільне.
4/26/2025, 10:15:00 AM	110.6	сніданок	Після сну, прийнято омега-3 та вітамін D.
4/26/2025, 12:00:00 AM	115.4	після їжі	Настрій спокійний, самопочуття хороше.
4/25/2025, 9:00:00 PM	123.6	вечеря	Після напруженого дня, трохи втома.
4/25/2025, 4:45:00 PM	118	обід	Випито багато води, легке самопочуття.
4/25/2025, 1:30:00 PM	92.6	перекус	Прийнято магній B6 після прийому їжі.
4/25/2025, 10:15:00 AM	99.7	сніданок	Після активної прогулянки.
4/25/2025, 12:00:00 AM	105	після їжі	Перед сном, мелатонін та ромашковий чай.
4/24/2025, 9:00:00 PM	108.4	вечеря	Вечеря без вуглеводів, легка їжа.
4/24/2025, 4:45:00 PM	87.9	обід	Після їжі з високим глікемічним індексом.
4/24/2025, 1:30:00 PM	127.4	перекус	Невеликий перекус: горіхи, самопочуття стабільне.
4/24/2025, 10:15:00 AM	86.7	сніданок	Після сну, прийнято омега-3 та вітамін D.
4/24/2025, 12:00:00 AM	115.8	після їжі	Настрій спокійний, самопочуття хороше.
4/23/2025, 9:00:00 PM	120.3	вечеря	Після напруженого дня, трохи втома.

Рис. 3.9 Журнал показників рівня глюкози

Сторінка аналітики, показана на рисунку 3.10, відповідає за побудову графічної візуалізації динаміки рівня глюкози. Дані отримуються з бекенду, трансформуються у формат, придатний до використання Chart.js, та відображаються у вигляді лінійного графіка. Реалізовано підтримку референсних

ліній (нормальних меж), підказок при наведенні та оновлення при зміні фільтра часу.

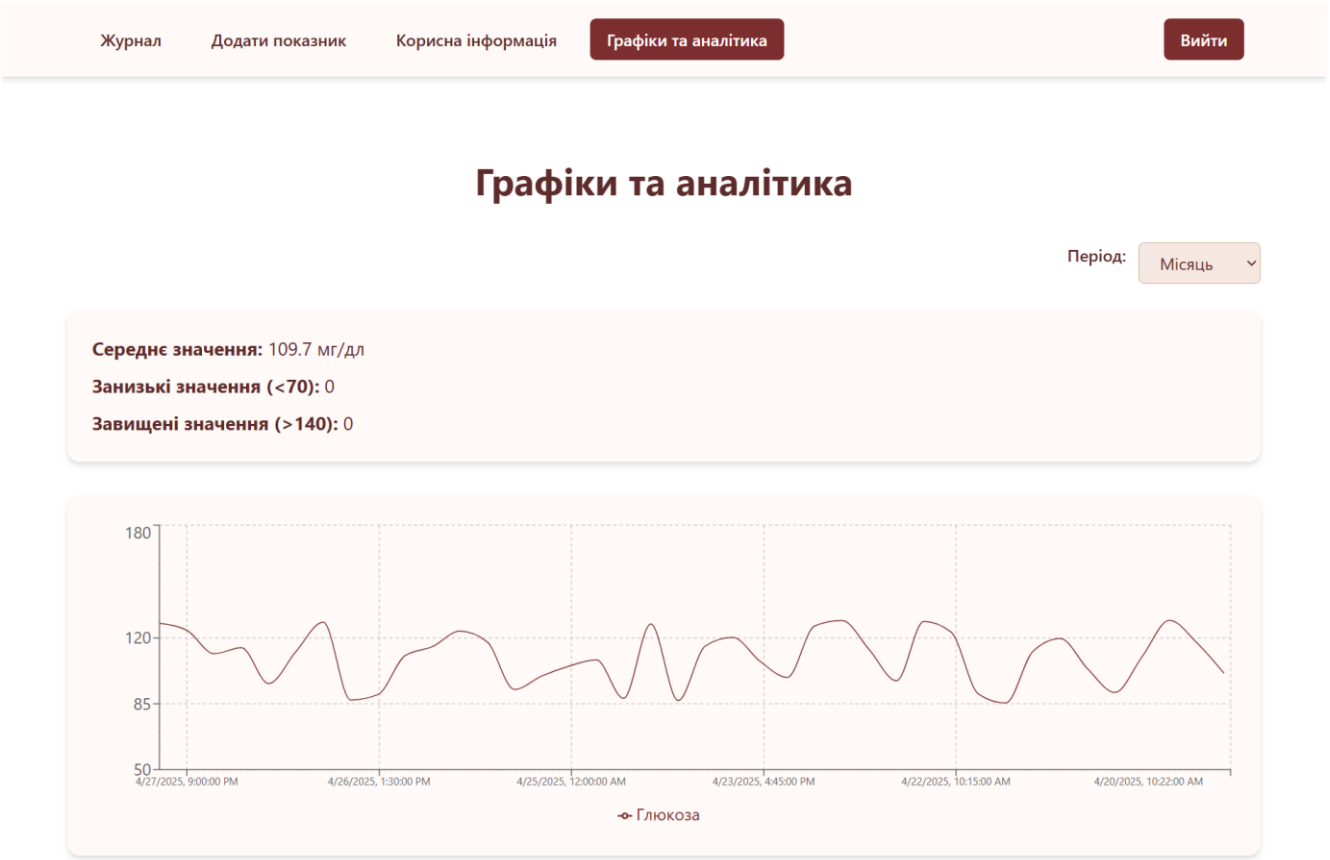


Рис. 3.10 Сторінка аналітики даних на web-панелі

Інформаційна сторінка, показана на рисунку 3.11, містить структуровану текстову інформацію про особливості контролю рівня глюкози, харчування, фізичну активність, норми для різних вікових груп. Вміст реалізований у вигляді статичної розмітки з підтримкою адаптивного форматування.

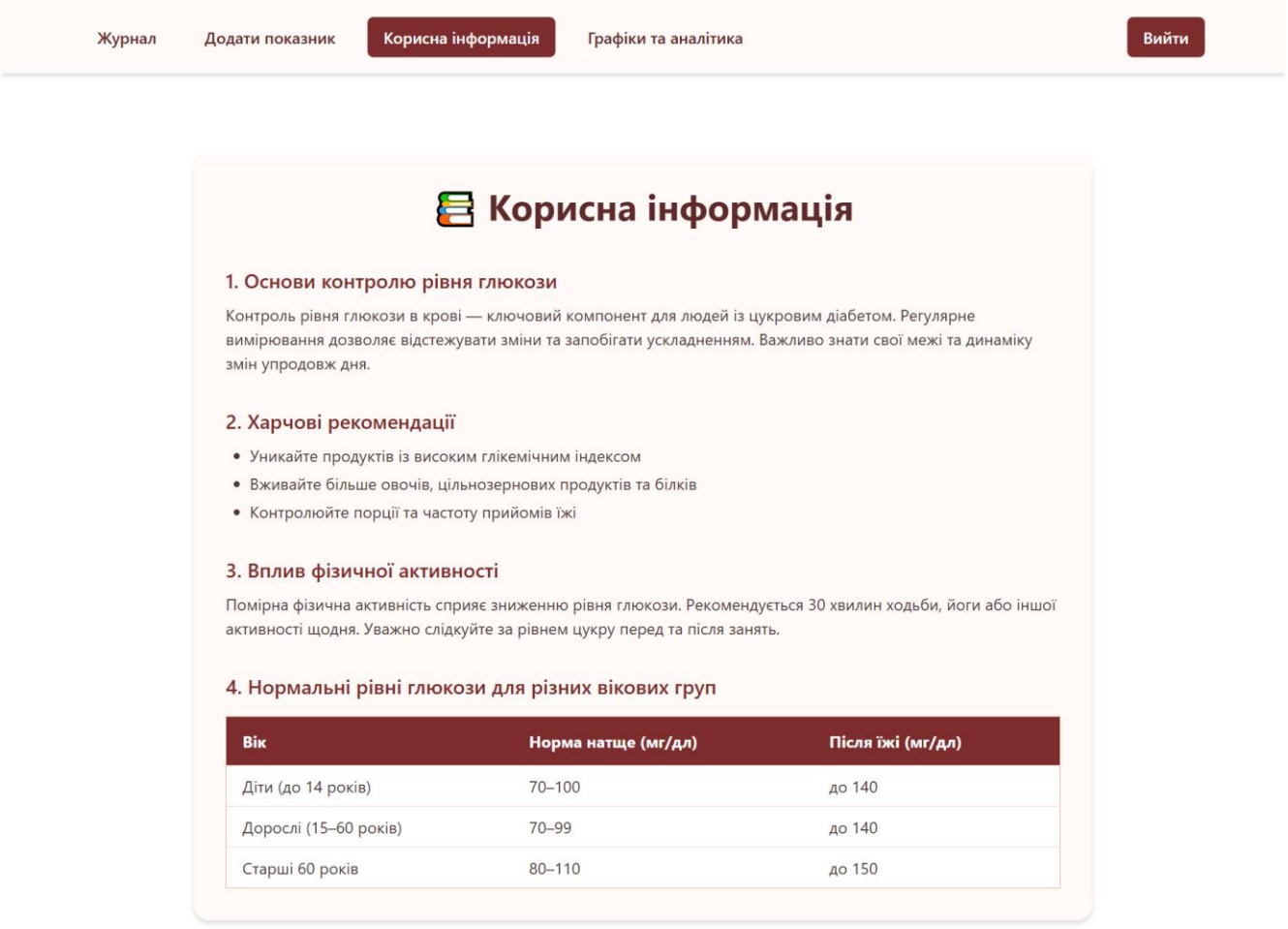


Рис. 3.11 Інформаційна сторінка

Всі компоненти інтерфейсу реалізовані з урахуванням адаптивності: застосовано гнучкі сітки, умовний рендеринг для малих екранів і мінімалістичну палітру кольорів. На рисунку 3.12 показано вигляд web-панелі на мобільному телефоні.

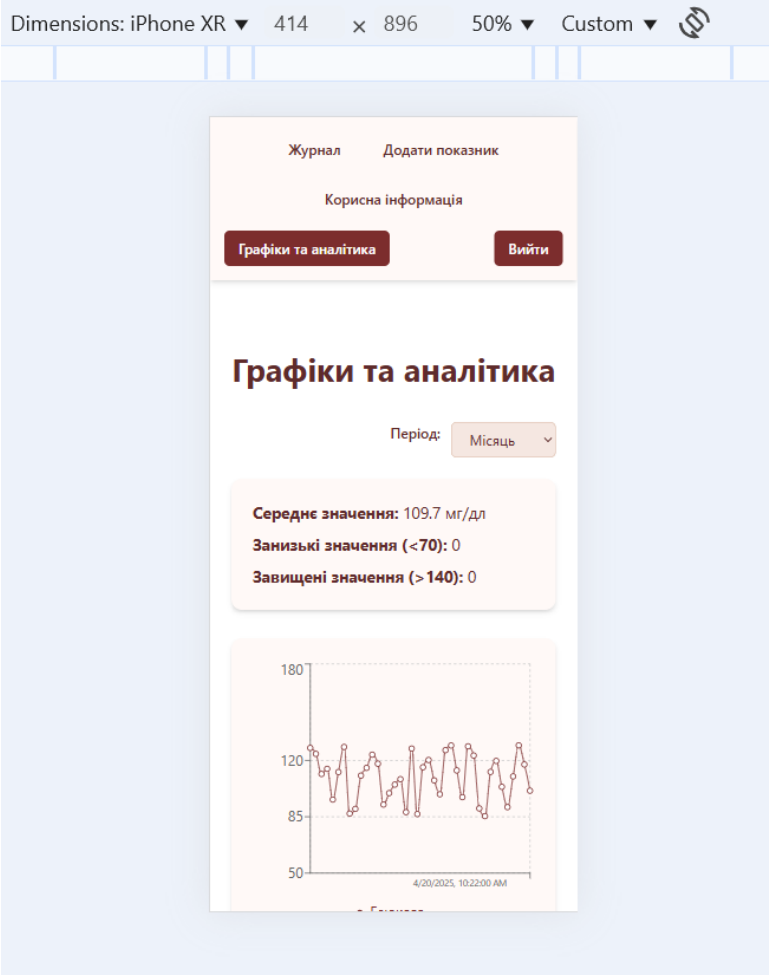


Рис. 3.12 Адаптивність дизайну web-панелі для мобільного перегляду

Ключова увага при реалізації користувацького інтерфейсу була спрямована на швидкодію та інтуїтивну взаємодію.

3.2 Релізація функцій зчитування і запису показників та аналітики даних

3.2.1 Механізм введення даних користувачами та їх збереження у базі даних

Процес введення даних у web-застосунку реалізовано у відповідності до сучасних вимог до інтерфейсної зручності, швидкодії та надійності обробки введеної інформації. Користувачі мають змогу вносити показники рівня глюкози в крові вручну через спеціально розроблену форму введення, схема роботи показана на рисунку 3.13, що розташована на відповідній функціональній сторінці

інтерфейсу. Форма включає обов’язкові поля для введення значення рівня глюкози (у ммоль/л), вибору дати й часу вимірювання, категорії вимірювання (до їжі, після їжі, натщесерце тощо), а також додаткове текстове поле для введення короткої примітки до кожного запису.

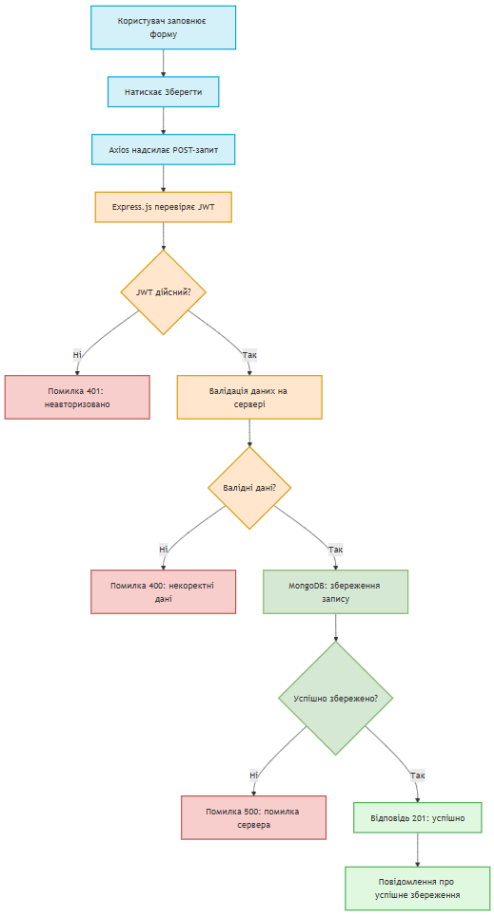


Рис. 3.13 Схема процесу введення та збереження даних у системі моніторингу глюкози

Форма побудована з використанням React-компонентів і пов’язана з локальним станом (state) компонента. Всі значення, які вводяться користувачем, проходять базову перевірку на коректність (непорожні поля, допустимий діапазон значень, відповідність формату дати). У разі успішного заповнення форма генерує POST-запит до відповідного API-ендпоїнту /api/glucose, передаючи дані у форматі JSON.

На серверній стороні цей запит обробляється Express-маршрутизатором, де реалізовано middleware для перевірки автентичності запиту на основі JWT-токена.

У разі валідного токена запит передається до контролера, де відбувається зчитування тіла запиту та перевірка його структури відповідно до схеми запису глюкози. Зокрема, перевіряється наявність значення, правильної дати та категорії, а також довжини коментаря (не більше 140 символів).

Серверна логіка взаємодіє з базою даних MongoDB через модель GlucoseRecord, яка реалізована з використанням бібліотеки Mongoose, показано на рисунку 3.14. У моделі визначено обов'язкові поля: userId, value, date, category, comment, кожне з яких має свій тип і обмеження. Після проходження валідації об'єкт створюється як новий документ у колекції glucoseRecords Зображено на рисунку 3.15 та зберігається з автоматичною прив'язкою до користувача через userId.

Рис. 3.14 Взаємодія компонентів введення

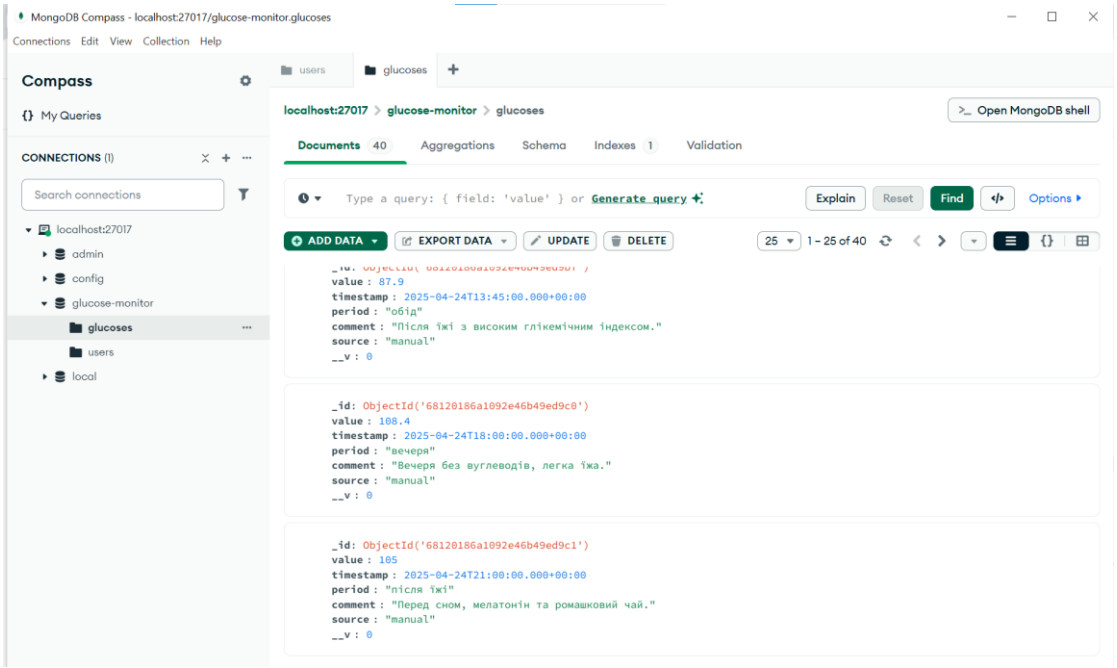


Рис. 3.15 База даних web-панелі (перегляд у MongoDB Compass)

Діаграма бази даних (колекції users та glucoses) показана на рисунку 3.16.

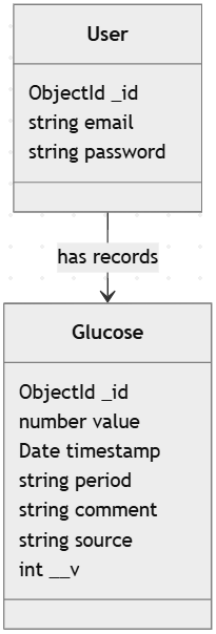


Рис. 3.16 UML-діаграма класів бази даних системи моніторингу глюкози

UML-діаграма відображає структуру двох основних сутностей системи: User та Glucose. Кожен об'єкт класу User пов’язаний із множиною записів класу Glucose, що представляють окремі вимірювання рівня глюкози. Для кожного запису зберігаються значення глюкози, час вимірювання, контекст (період прийому їжі), коментар користувача та джерело даних. Зв’язок між класами ілюструє відношення "один до багатьох".

У випадку успішного збереження запису сервер надсилає клієнту відповідь зі статусом 201 (Created) і підтвердженням операції. Якщо під час виконання запиту виникає помилка валідації або проблема на рівні бази даних, повертається відповідний код помилки (400 або 500) з описом проблеми, що дозволяє користувачу отримати зворотний зв’язок через інтерфейс.

Завдяки використанню MongoDB як гнучкої документо-орієнтованої СУБД, забезпечується можливість зберігання записів у довільному форматі з подальшою адаптацією структури. При цьому всі записи пов’язані з конкретним користувачем, що дозволяє формувати персоналізовану історію вимірювань, виконувати вибірки за часовими рамками та здійснювати агрегацію даних для аналітики.

Розроблений механізм введення та збереження даних є ключовим елементом функціональності системи, оскільки саме на його основі формуються як

журнали вимірювань, так і графіки аналітики. Реалізація з використанням React, Express та MongoDB забезпечує високу швидкодію, відмовостійкість і масштабованість процесу збереження даних, що є критично важливим для систем у сфері електронного здоров’я.

3.2.2 Побудова візуалізацій рівня глюкози та аналіз динаміки змін показників

Функціональність візуалізації даних у розробленому web-застосунку реалізована з метою забезпечення користувачів наочним представленням динаміки рівня глюкози в крові, що є критично важливим для самоконтролю та аналізу ефективності дій, пов’язаних з харчуванням, фізичною активністю або медикаментозною терапією. Основним інструментом реалізації графіків було обрано бібліотеку Chart.js, яка дозволяє будувати адаптивні, масштабовані та інформативні лінійні графіки з високим рівнем кастомізації.

Архітектурно візуалізаційний компонент є окремим React-компонентом, що імпортується у відповідну сторінку аналітики. Джерелом даних для графіка слугують записи, отримані з бекенду через ендпоїнт /api/glucose, що повертає масив структурованих об’єктів у форматі JSON, показано на рисунку 3.17. Після отримання даних вони трансформуються у вигляді, придатному для побудови часових рядів: кожен запис містить дату вимірювання, значення рівня глюкози, а також позначку категорії.

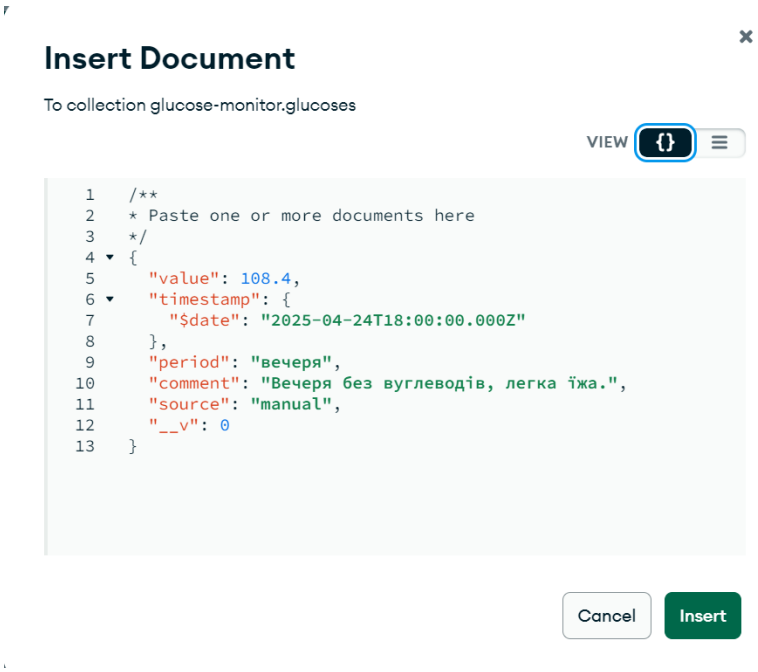


Рис. 3.17 Структура відповіді з API для побудови графіка

Графік будується як лінійна діаграма, де по осі абсцис відображається дата/час вимірювання, а по осі ординат — значення глюкози в ммоль/л, зображено на рисунку 3.18.

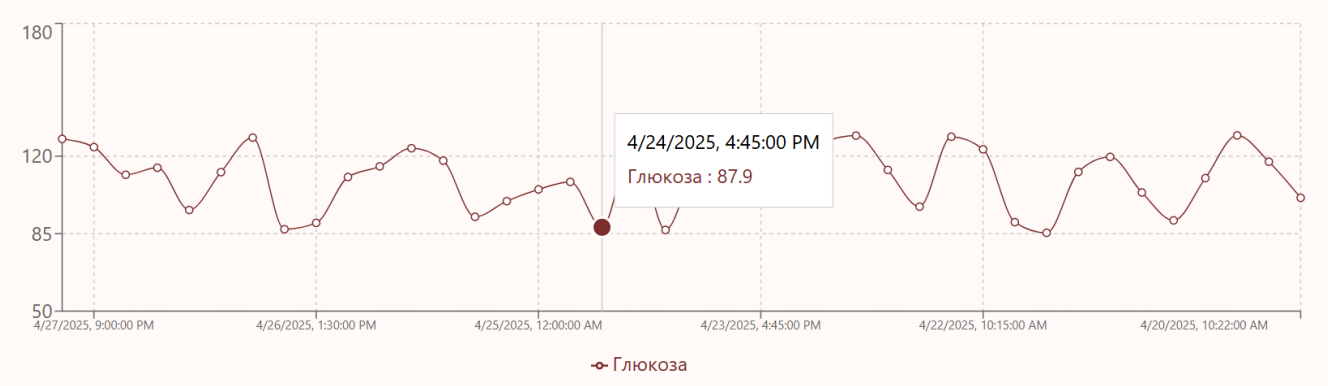


Рис. 3.18 Інтерактивний графік змін показників глюкози відповідно до бази даних web-панелі

Крім того, реалізовано можливість візуального виділення критичних значень, що виходять за межі референсного інтервалу (наприклад, <4 ммоль/л або >10 ммоль/л). Для цього передбачено додаткові горизонтальні лінії, які позначають

допустимі межі (reference lines), що дозволяє користувачеві оперативно виявити відхилення.

Побудова графічного відображення динаміки рівня глюкози передбачає попередню обробку масиву даних, отриманого з бекенд-ендпоїнта /api/glucose. Як показано на рисунку 3.19, дані спочатку проходять фільтрацію відповідно до обраного користувачем діапазону — наприклад, за поточним днем, останнім тижнем, місяцем або трьома місяцями. Далі здійснюється обчислення базових статистичних показників: середнього значення, мінімального та максимального рівнів глюкози, що дає змогу проводити первинний аналіз. Отримані результати трансформуються у формат, сумісний із бібліотекою Chart.js, зокрема формуються масиви labels (мітки часу) та data (значення глюкози). У фінальному етапі до графіка додаються референсні межі норми, після чого дані виводяться у вигляді лінійної діаграми.

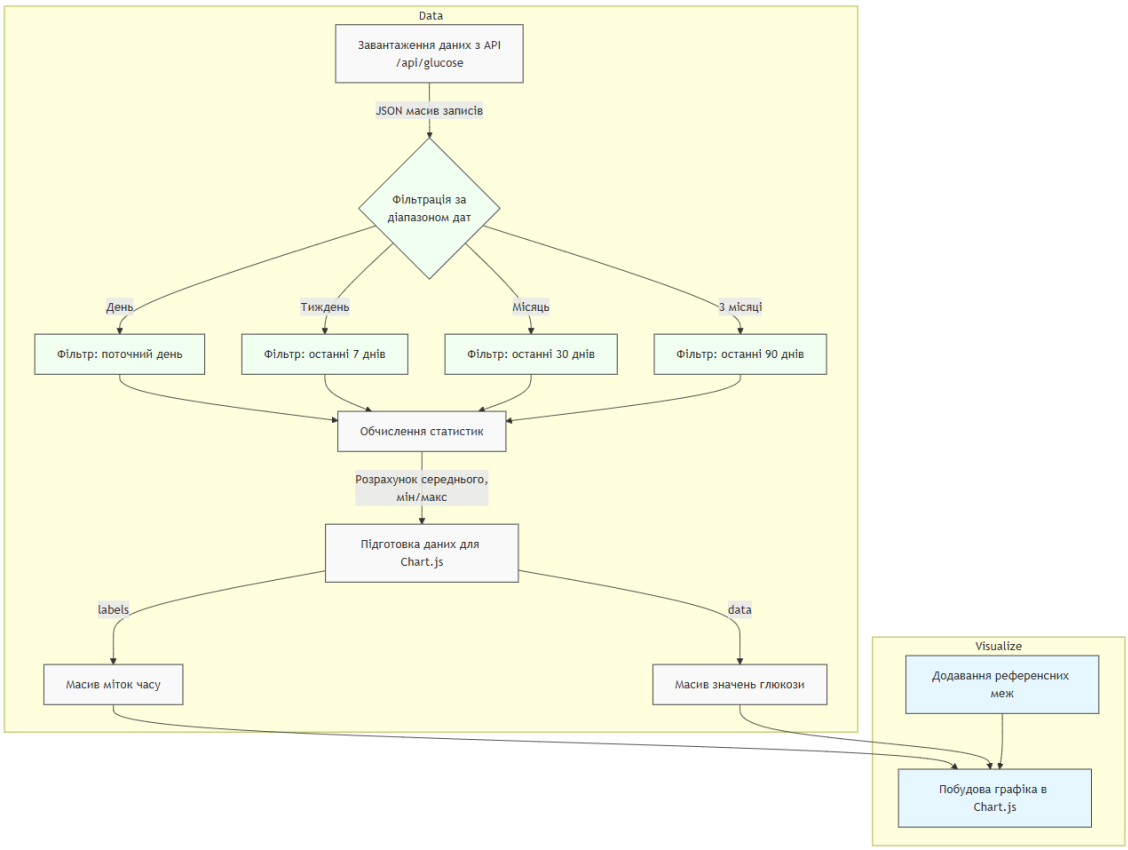


Рис. 3.19 Логіка обробки та трансформації даних перед побудовою графіка в Chart.js

Компонент ChartView підтримує реактивне оновлення даних у разі зміни фільтрів (період: день, тиждень, місяць, три місяці), що реалізується за допомогою хука useEffect з відповідним запитом до сервера, що показано на рисунку 3.20. У разі відсутності даних за обраний період виводиться інформативне повідомлення про відсутність вимірювань.

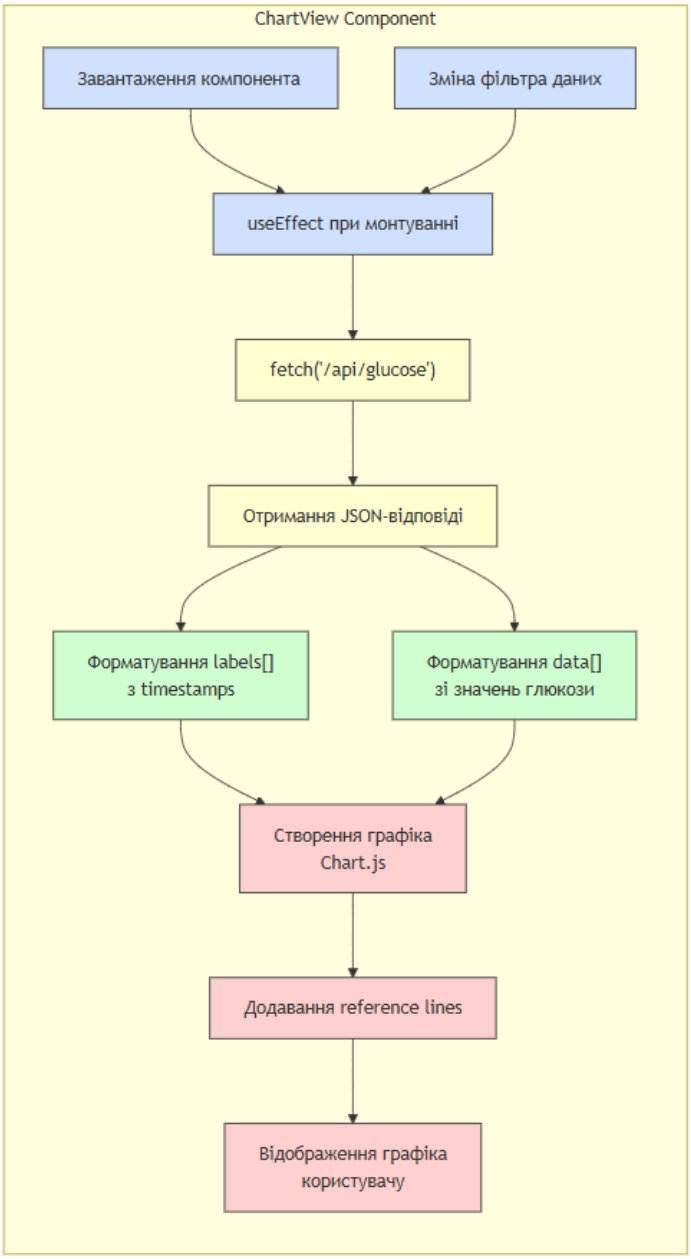


Рис. 3.20 Структура компонента візуалізації ChartView

Реалізовано динамічне завантаження контенту із забезпеченням оптимального користувацького досвіду.

З точки зору аналітики, реалізований модуль дозволяє оцінити:

- середній рівень глюкози за вибраний період;
- кількість випадків гіпоглікемії або гіперглікемії (за межами встановлених референсів);
- загальну кількість вимірювань;
- інтервал між вимірюваннями (опосередковано).

Додатково передбачено можливість інтерактивного перегляду даних у вигляді tooltip-підказок при наведенні на точку графіка — користувач бачить точну дату, значення глюкози, категорію вимірювання та коментар (якщо він був внесений).

Використання Chart.js обумовлене простотою інтеграції з React, високою швидкодією, а також підтримкою адаптивної верстки, що забезпечує коректне відображення графіка як на десктопних, так і на мобільних пристроях. Усі елементи графічного інтерфейсу було стилізовано відповідно до загального дизайну системи із дотриманням принципу інформаційної доступності (контрастність, читабельність, логічна структура).

Побудована система візуалізації демонструє відповідність сучасним вимогам до медичних інтерфейсів у галузі eHealth, забезпечуючи користувачеві засоби для щоденного контролю та довгострокового аналізу змін у стані здоров’я. Завдяки використаним інструментам забезпечено не лише високу якість відображення, а й аналітичну цінність зібраних даних.

3.3 Тестування та оцінка ефективності роботи web-застосунку для моніторингу рівня глюкози

Для об'єктивної оцінки якості реалізованого web-застосунку було проведено комплексне автоматизоване тестування за допомогою інструменту Google Lighthouse — відкритого аудиторського фреймворку, вбудованого в Chrome DevTools, що дозволяє проводити аудит web-застосунків за кількома критеріями:

продуктивність, доступність, найкращі практики, SEO та PWA-сумісність. Тестування виконувалося у середовищі емуляції десктопного браузера.

Результати аудиту продемонстровано на рисунку 3.21 підтвердили високу якість реалізації застосунку:

- Accessibility — 93/100;
- Best Practices — 100/100;
- SEO — 100/100;
- Performance — 74/100.

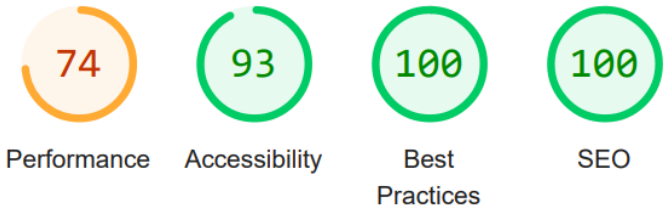


Рис. 3.21 Результати перевірки сайту у Lighthouse

Результати доступності вказують на майже повну відповідність стандартам WCAG: всі інтерфейсні елементи мають чіткі мітки, реалізована логічна структура заголовків, а також коректно налаштовані атрибути <lang> і <title>. Основні зауваження стосуються відсутності міток у деяких елементах типу <select> та часткової підтримки доступу з клавіатури, що не є критичним, але може бути враховано в майбутніх версіях.

У категорії Best Practices застосунок отримав максимальний бал. Це свідчить про дотримання сучасних вимог до безпеки, відсутність вразливостей типу document.write(), правильне використання HTTPS, захист від XSS, а також відповідність структурним рекомендаціям Google для HTML-розмітки.

Показник SEO (Search Engine Optimization) також отримав 100/100, що демонструє наявність усіх ключових елементів: коректні мета-теги, валідні заголовки, оптимізований порядок DOM-елементів, доступність для індексації та підтримка адаптивного дизайну. Хоча застосунок не призначений для публічного

пошуку, високий SEO-рейтинг свідчить про чистоту коду і грамотне структурування сторінок.

Блок Performance, хоча і не досяг ідеального значення, продемонстрував добрий рівень оптимізації (74/100). Найбільший вплив на зниження оцінки справили два фактори: Largest Contentful Paint (LCP) — 3.0 секунди та Total Blocking Time (TBT) — 190 мс.

Ці значення залишаються у межах прийнятного діапазону, але можуть бути покращені шляхом оптимізації великого JS-бандлу та відкладеного завантаження непершочергових скриптів.

Серед конкретних рекомендацій щодо покращення продуктивності Lighthouse запропонував:

- мінімізувати JavaScript (bundle.js має понад 574 КіБ),
- усунути невикористовуваний CSS,
- активувати кешування статичних ресурсів.

Ці зауваження не є критичними і стосуються здебільшого фронтенд-оптимізації, що не впливає на функціональність або безпеку.

Загалом, результати тестування засвідчують, що створений web-застосунок повністю відповідає вимогам сучасної розробки у частині безпеки, зручності використання, відповідності стандартам і доступності. Незначні зниження в метриках продуктивності є типовими для локальних середовищ розробки і можуть бути усунені при продакшн-деплої. Таким чином, реалізоване рішення демонструє високу ефективність та готовність до використання в реальних умовах самоконтролю користувачами.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було реалізовано повнофункціональний web-застосунок для моніторингу рівня глюкози, який відповідає сучасним вимогам до медичних цифрових систем у сфері самоконтролю та профілактики цукрового діабету. Проведено ґрунтовний аналіз існуючих рішень на ринку, виділено ключові технічні обмеження та переваги конкурентів, що дозволило сформувати виважене технічне завдання.

У роботі обґрунтовано вибір відкритого технологічного стеку: React.js для створення інтерфейсу, Node.js з Express для серверної логіки, MongoDB як документо-орієнтованої бази даних та JWT для автентифікації. Розроблена архітектура web-застосунку враховує вимоги до масштабованості, безпеки та зручності підтримки, а модульний принцип побудови коду дозволяє у майбутньому легко інтегрувати нові функції, зокрема мобільний клієнт або модулі рекомендацій на основі ШІ.

Реалізовано REST API відповідно до специфікації OpenAPI 3.0, з повною структурою запитів, схемою валідації даних та інтерактивною документацією у форматі Swagger. Це значно підвищує рівень формалізації системи та дозволяє легко інтегрувати її з іншими медичними сервісами. Дані зберігаються у структурованому вигляді у колекціях MongoDB, а візуалізація здійснюється за допомогою бібліотеки Chart.js.

На основі тестування засобами Lighthouse підтверджено високу швидкодію, адаптивність та відповідність принципам доступності. У процесі проєктування й реалізації дотримано принципів академічної доброчесності, а всі зовнішні джерела та бібліотеки належно процитовано.

Практична значущість роботи полягає в можливості її використання як навчального прикладу для підготовки фахівців у сфері web-розробки та цифрової медицини, а також як основи для розробки повноцінної медичної інформаційної системи самоконтролю глюкози.

Робота прошла апробацію. За її результатами було опубліковано наступні тези доповіді:

1. Максимчук А. О., Залива В.В. Архітектура, технології та безпека даних в web-застосунку для моніторингу рівня глюкози // VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24 квітня 2025 року, ДУІКТ, м.Київ С.26 - 28.
2. Максимчук А. О., Залива В.В. Використання IoT у моніторингу рівня глюкози: перспективи та виклики. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» Збірник тез. ДУІКТ, м.Київ *(подано до друку).*

ПЕРЕЛІК ПОСИЛАНЬ

1. mySugr Global - Make Diabetes Suck Less. *mySugr Global - Make Diabetes Suck Less | mySugr*. URL: https://www.mysugr.com/en.

2. How to manage your diabetes using our mySugr® app. *Accu-Chek UK & Ireland*. URL: https://www.accu-chek.co.uk/blog/manage-diabetes-with-mysugr.

3. Higher chance to achieve optimal glycaemic control via a digital diabetes logbook: Subanalysis of the mySugr PRO randomised controlled trail. *mySugr Global - Make Diabetes Suck Less | mySugr*. URL: https://www.mysugr.com/files/media/files/ATTD-2024-e-Poster-mySugr-HbA1c_A0_CMYK%20(1)_final%20(1).pdf.

4. Diabetes:M - Your Diabetes Management App - Keep Diabetes Under Control. *Diabetes:M - Your Diabetes Management App*. URL: https://diabetes-m.com/.

5. CGM Features - Diabetes:M User's Guide - Mobile. *Sirma Medical Systems*. URL: https://www.manula.com/manuals/sirma-medical-systems/diabetes-m-user-guide/mobile/en/topic/cgm-features.

6. Walsh, J., Roberts, R., & Heinemann, L. (2014). Confusion Regarding Duration of Insulin Action: A Potential Source for Major Insulin Dose Errors by Bolus Calculators. *Journal of diabetes science and technology*, *8*(1), 170–178. https://doi.org/10.1177/1932296813514319.

7. LibreLinkUp. *LibreLinkUp*. URL: https://www.librelinkup.com/.

8. FreeStyle LibreLink 2.11.2 - Medycyna - download - instalki.pl. *INSTALKI*. URL: https://www.instalki.pl/download/android/medycyna/freestyle-librelink/.

9. Downson E. Development of Freestyle Libre 2 App-Overview. *dzone.com*. URL: https://dzone.com/articles/development-of-freestyle-libre-2-app-overview.

10. Connected Care for Diabetes | Glooko. *Glooko*. URL: https://glooko.com/.

11. Glooko API Portal. *Glooko API Portal*. URL: https://developers.glooko.com/docs/ehrintegrations/reportsummaries.

12. Glooko Onboarding – Pat Harrington. *Pat Harrington*. URL: https://patharrington.design/glooko-onboarding.

13. React. *React*. URL: https://react.dev/.

14. A Definitive Guide to React Architecture Patterns - eTatvaSoft. *eTatvaSoft*. URL: https://www.etatvasoft.com/blog/react-architecture-patterns/.

15. GeeksforGeeks. Explain the Architecture Overview of Angular ? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/explain-the-architecture-overview-of-angular/.

16. Maciel E. What is Vue and How Does it Compare to React in Popularity?. *Scalable Path*. URL: https://www.scalablepath.com/front-end/vue-js-popularity-framework.

17. Hammad M. Exploring Express.js: Building Web Servers with Node.js. *Medium*. URL: https://medium.com/@hamadchaudhary/exploring-express-js-building-web-servers-with-node-js-7200c74b1e23.

18. Maple C. Understanding Django MVT architecture and view functions| Django Full Course for Beginners | Lesson… *Medium*. URL: https://medium.com/@CodeMaple/understanding-django-mvt-architecture-and-view-functions-django-full-course-for-beginners-lesson-39c8da093b44.

19. Full System Architecture of my React-Flask App. *A Cloudy Affair*. URL: https://amlanscloud.com/apparchitecture/.

20. Hamid A. K. Java 23, SpringBoot 3.3.4: API Flow & Logging–Part 4. *Medium*. URL: https://faun.pub/java-23-springboot-3-3-4-api-flow-logging-part-4-1000546bcd62.

21. GeeksforGeeks. MongoDB Architecture - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/mongodb-architecture/.

22. Shukla S. PostgreSQL Architecture. *Medium*. URL: https://medium.com/@sumeet.k.shukla/postgresql-architecture-6df259dc1145.

23. FCM Architectural Overview | Firebase Cloud Messaging. *Firebase*. URL: https://firebase.google.com/docs/cloud-messaging/fcm-architecture.

24. JWT Authentication Best Practices. *OpenReplay Blog - Resources for Frontend Developers*. URL: https://blog.openreplay.com/jwt-authentication-best-practices/.

25. OAuth 2.0. *Moved*. URL: https://docs.oracle.com/cd/E82085_01/160027/JOS%20Implementation%20Guide/Output/oauth.htm.

26. Xu A. How does HTTPS work? (Episode 6). *ByteByteGo Newsletter | Alex Xu | Substack*. URL: https://blog.bytebytego.com/p/how-does-https-work-episode-6.

27. What is AWS HealthLake? - AWS HealthLake. URL: https://docs.aws.amazon.com/healthlake/latest/devguide/what-is.html.

28. Chris von See. Getting to know the Google Cloud Healthcare API: Part 1 | Google Cloud Blog. *Google Cloud Blog*. URL: https://cloud.google.com/blog/topics/healthcare-life-sciences/getting-to-know-the-google-cloud-healthcare-api-part-1.

29. GeeksforGeeks. Top 10 JavaScript Libraries for Data Visualization [2025] - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/javascript-libraries-for-data-visualization/.

30. Inc G. Glooko - Track Diabetes Data – Додатки в Google Play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.glooko.logbook&hl=uk.

31. Systems S. M. Diabetes:M - Blood Sugar Diary – Додатки в Google Play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.mydiabetes&hl=uk.

32. Solutions D. MyDiabetes: Health management – Додатки в Google Play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=health.mydiabetes&hl=uk.

33. Pavliv I., Kunanets N. Data Security Requirements of the Medical Information System. *Vìsnik Nacìonal'nogo unìversitetu "L'vìvs'ka polìtehnìka". Serìâ Ìnformacìjnì sistemi ta merežì*. 2023. Vol. 14. P. 267–280. URL: https://doi.org/10.23939/sisn2023.14.267 (date of access: 30.04.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для моніторингу рівня глюкози

Виконала студентка 4 курсу
групи ПД-41
Максимчук Ангеліна Олександрівна
Керівник роботи
ст. в кафедри, доктор філософії (PhD) Залива Віталій Вікторович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу самоконтролю рівня глюкози в крові шляхом створення веб-застосунку.
- **Об'єкт дослідження -** процес моніторингу рівня глюкози в крові за допомогою цифрових інструментів.
- **Предмет дослідження -** архітектура та реалізація web-застосунку для контролю рівня глюкози.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Виконати огляд сучасних підходів до цифрового контролю рівня глюкози;
2. Визначити перелік вимог до майбутньої системи з урахуванням кращих практик та досвіду користувачів;
3. Розробити архітектуру web-застосунку;
4. Реалізувати клієнтську та серверну частини проєкту;
5. Забезпечити збереження даних у базі MongoDB із захистом доступу через JWT;
6. Реалізувати модулі візуалізації та статистичної обробки даних;
7. Провести тестування web-застосунку та оцінити його ефективність за допоиогою інструментів Lighthouse.

АНАЛІЗ АНАЛОГІВ

Технічні характеристики	**LibreLink**	**Glooko Web**	**Diabetes:M WEB**	**Розроблений web-застосунок**
Можливість ручного введення	-	+	+	+
Візуалізація (графіки, звіти)	+	+	+	+
Аналітика, рекомендації	-	+	+	+
Україномовний інтерфейс	-	-	-	+
Зручність використання для людей з вадами зору	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Наявність механізму реєстрації та автентифікації користувача з використанням JWT;
- Можливість внесення вимірювань із зазначенням дати, часу, категорії та рівня глюкози;
- Фіксація коментарів до кожного запису (до 140 символів);
- Перегляд історії записів у таблиці;
- Візуалізація динаміки глюкози у вигляді лінійного графіка;
- Сортування записів за обраним часовим діапазоном;
- Можливість редагування та видалення даних.

Нефункціональні вимоги:

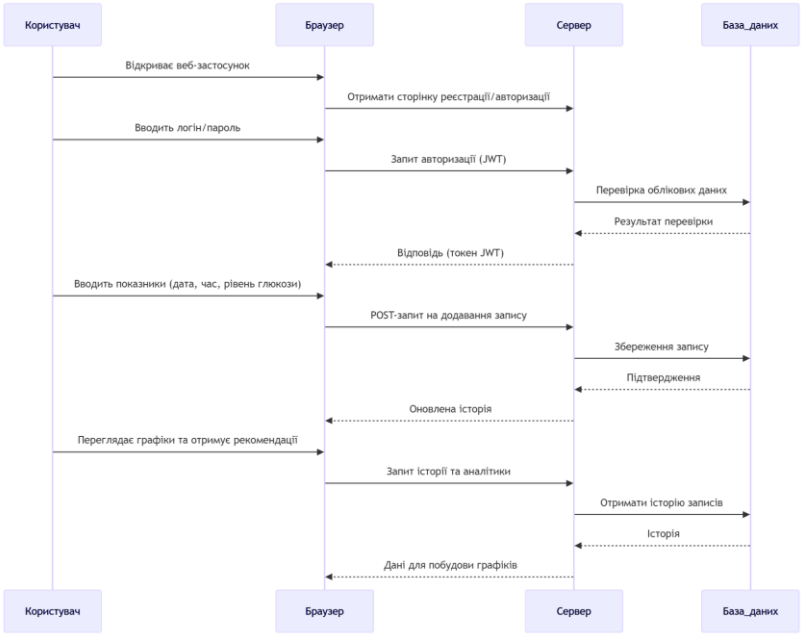
- Адаптивність інтерфейсу для використання на екранах шириною від 360 до 1920 пікселів;
- Збереження введених даних у базі MongoDB;
- Наявність авторизації та аутентифікації;
- Стабільність з'єднання при стандартному навантаженні.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

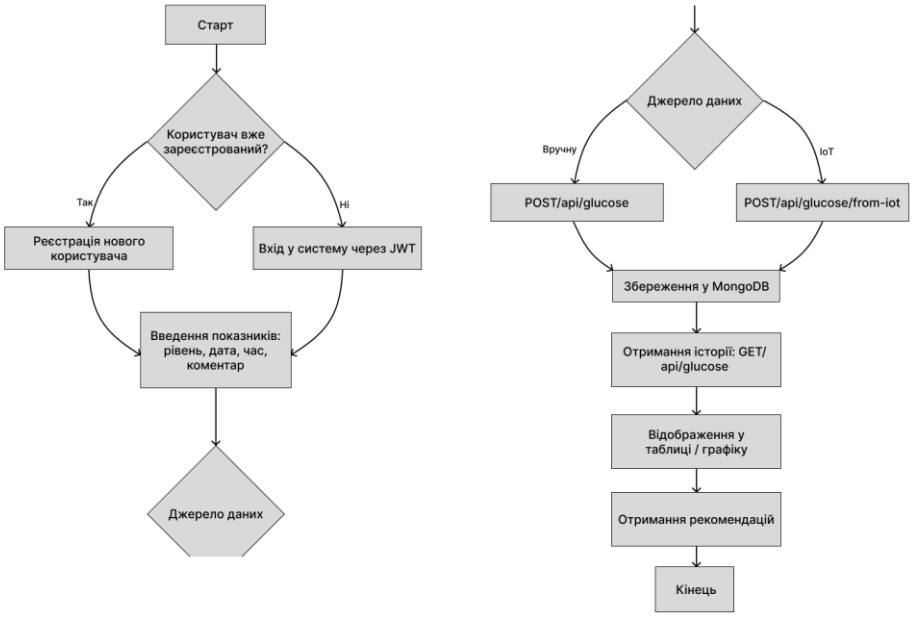
6

ВАРІАНТИ ВИКОРИСТАННЯ



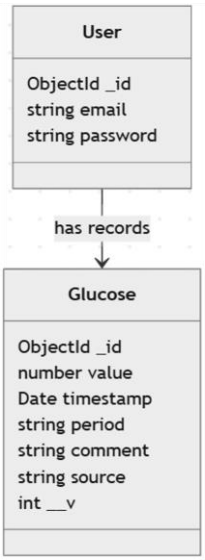
7

АЛГОРИТМ РОБОТИ ПРОГРАМИ



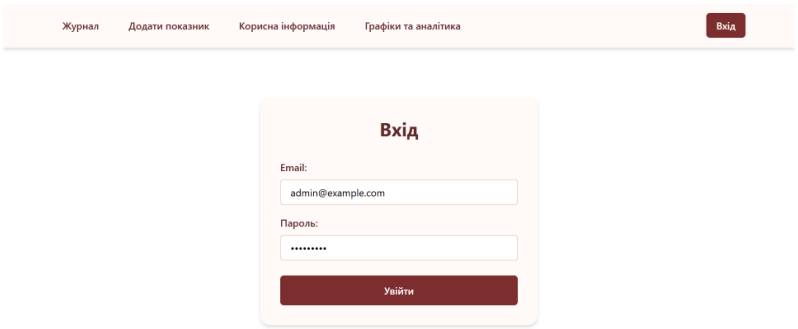
8

ДІАГРАМА КЛАСІВ

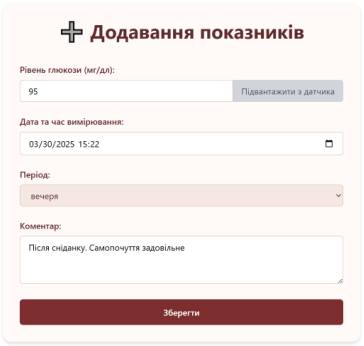


9

ЕКРАННІ ФОРМИ



Сторінка авторизації web-панелі

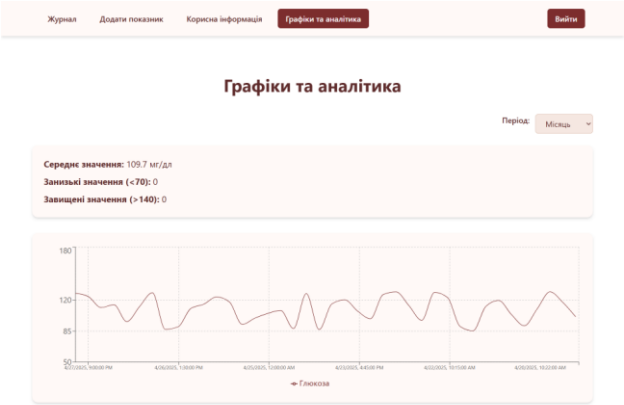


Сторінка введення показників рівня глюкози

10

ЕКРАННІ ФОРМИ

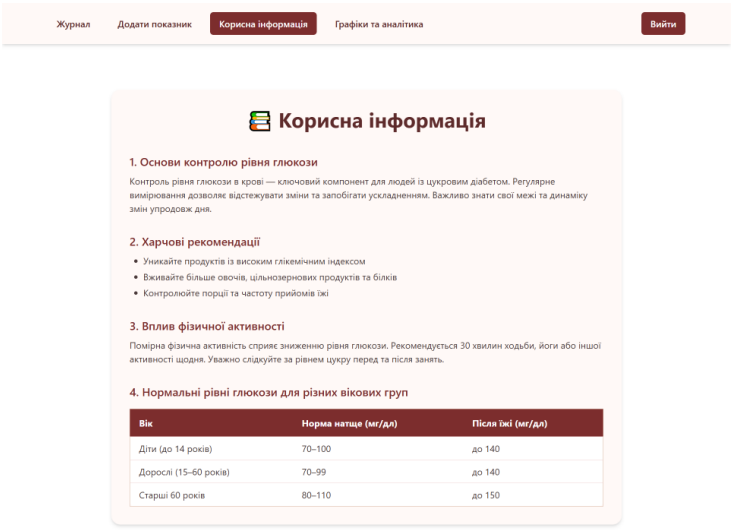
Журнал показників рівня глюкози



Сторінка аналітики
даних на web-панелі

11

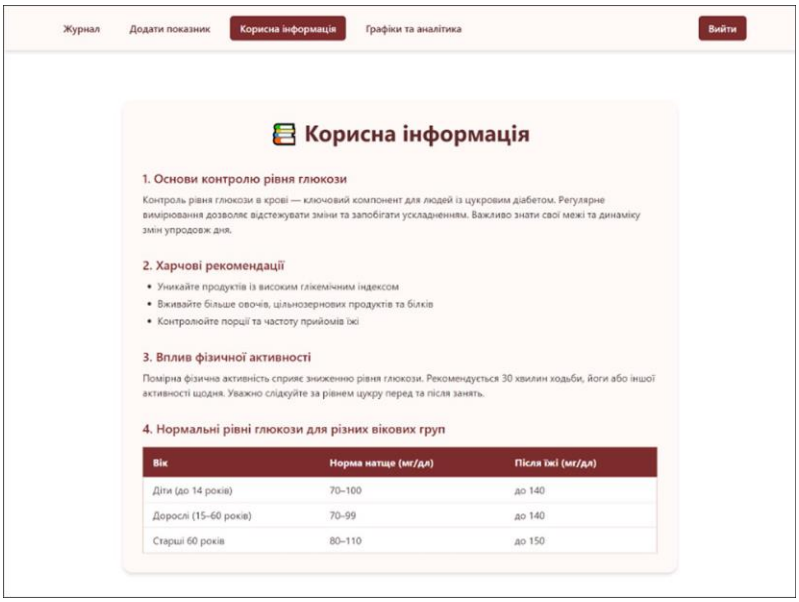
ЕКРАННІ ФОРМИ



Інформаційна сторінка

12

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Максимчук А. О., Залива В.В. Архітектура, технології та безпека даних в web-застосунку для моніторингу рівня глюкози // VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24 квітня 2025 року, ДУІКТ, м.Київ С.26 - 28.
2. Максимчук А. О., Залива В.В. Використання IoT у моніторингу рівня глюкози: перспективи та виклики. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» Збірник тез. ДУІКТ, м.Київ *(подано до друку)*.

ВИСНОВКИ

1. Проведено огляд сучасних підходів до цифрового контролю рівня глюкози та проаналізувати переваги та недоліки існуючих рішень;
2. Сформовано перелік вимог до майбутнього web-застосунку з урахуванням кращих практик та досвіду користувачів та стандартів безпеки;
3. Розроблено архітектуру web-застосунку;
4. Реалізувано клієнтську частину на React.js та серверну частину на Node.js + Express;
5. Забезпечено збереження даних у базі MongoDB із захистом доступу через JWT;
6. Створено модулі візуалізації та статистичної обробки даних із застосуванням Chart.js;
7. Проведено тестування web-застосунку засобами Google Lighthouse, що підтвердило його відповідність вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

../client/src/App.js

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Journal from './pages/Journal';
import Add from './pages/Add';
import Info from './pages/Info';
import Analytics from './pages/Analytics';
import Login from './pages/Login';
import ProtectedRoute from './components/ProtectedRoute';
function App() {
  return (
    <Router>
      <Navbar />
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/info" element={<Info />} />

        {/* Захищені маршрути */}
        <Route
          path="/journal"
          element={
            <ProtectedRoute>
              <Journal />
            </ProtectedRoute>
          }
        />
        <Route
          path="/add"
          element={
            <ProtectedRoute>
              <Add />
            </ProtectedRoute>
          }
        />
        <Route
          path="/analytics"
          element={
            <ProtectedRoute>
              <Analytics />
            </ProtectedRoute>
          }
        />
      </Routes>
    </Router>
  );
}
export default App;
```

../client/src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

../client/src/pages/Login.js

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [status, setStatus] = useState('');
  const navigate = useNavigate();
  const handleLogin = async () => {
    try {
      const res = await axios.post('http:
      localStorage.setItem('token', res.data.token);
      setStatus('✅ Вхід успішний!');
      navigate('/journal');
    } catch (err) {
      setStatus('❌ Невірний логін або пароль');
    }
  };
  return (
    <div className="bg-white min-h-screen py-10 px-6 md:px-16">
      <div className="max-w-md mx-auto bg-[#fff9f7] p-8 rounded-2xl shadow-md">
        <h2 className="text-3xl font-bold text-center mb-8 text-[#5E2B2B]">Вхід</h2>
        <label className="block mb-2 font-medium text-[#5E2B2B]">Email:</label>
        <input
          type="email"
          value={email}
          onChange={e => setEmail(e.target.value)}
          className="w-full border border-[#d8b4a0] rounded-md px-4 py-2 mb-4 focus:outline-none"
        />
        <label className="block mb-2 font-medium text-[#5E2B2B]">Пароль:</label>
        <input
          type="password"
          value={password}
          onChange={e => setPassword(e.target.value)}
          className="w-full border border-[#d8b4a0] rounded-md px-4 py-2 mb-6 focus:outline-none"
        />
        <button
```

```
        onClick={handleLogin}
        className="w-full bg-[#7c2d2d] text-white font-semibold py-3 rounded-md hover:bg-[#5e2b2b] transition"
       >
        Увійти
       </button>
       {status && <p className="mt-6 text-center font-semibold text-[#5E2B2B]">{status}</p>}
     </div>
   </div>
  );
}
export default Login;
```

../client/src/pages/Journal.js

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
const FILTER_OPTIONS = {
  '1d': 'Останній день',
  '7d': 'Останній тиждень',
  '14d': '2 тижні',
  '30d': 'Місяць',
  '90d': '3 місяці'
};
function Journal() {
  const [readings, setReadings] = useState([]);
  const [filter, setFilter] = useState('30d');
  useEffect(() => {
   loadReadings();
  }, []);
  const loadReadings = async () => {
   try {
    const token = localStorage.getItem('token');
    const res = await axios.get('http:
     headers: {
      Authorization: `Bearer ${token}`,
     },
    });
    setReadings(res.data);
   } catch (err) {
    console.error('Помилка завантаження даних', err);
   }
  };
  const applyFilter = (data) => {
   const now = new Date();
   const days = parseInt(filter);
   const msOffset = days * 24 * 60 * 60 * 1000;
   const threshold = new Date(now - msOffset);
            return data.filter(entry => new Date(entry.timestamp) >= threshold);
  };
  return (
   <div className="bg-white min-h-screen py-10 px-6 md:px-16">
    <div className="max-w-6xl mx-auto">
     <h2 className="text-4xl font-bold text-center mb-10 text-[#5E2B2B]">
      Журнал показників глюкози
     </h2>
     <div className="flex justify-end mb-6">
      <label className="mr-3 font-medium text-[#5E2B2B]">Фільтр:</label>
      <select
       className="border border-[#d8b4a0] rounded-md px-4 py-2 bg-[#f5e7e1] text-[#5E2B2B] focus:outline-none"
       value={filter}
       onChange={(e) => setFilter(e.target.value)}
      >
       {Object.entries(FILTER_OPTIONS).map(([key, label]) => (
                <option key={key} value={key}>{label}</option>
       ))}
      </select>
     </div>
      <div className="overflow-x-auto bg-[#fff9f7] rounded-xl shadow-md">
      <table className="min-w-full table-auto">
       <thead className="bg-[#7c2d2d] text-white">
        <tr>
         <th className="py-4 px-6 text-left">Дата і час</th>
         <th className="py-4 px-6 text-left">Глюкоза (мг/дл)</th>
                <th className="py-4 px-6 text-left">Період</th>
                <th className="py-4 px-6 text-left">Коментар</th>
        </tr>
       </thead>
       <tbody>
        {applyFilter(readings).length === 0 ? (
         <tr>
          <td colSpan="4" className="text-center py-6 text-gray-400">Дані відсутні</td>
         </tr>
        ) : (
         applyFilter(readings).map((entry, i) => (
          <tr
           key={i}
            className="border-b border-[#eedbd4] hover:bg-[#fdecea] transition"
           >
            <td className="py-3 px-6">{new Date(entry.timestamp).toLocaleString()}</td>
            <td className="py-3 px-6">{entry.value}</td>
           <td className="py-3 px-6">{entry.period || '-'}</td>
            <td className="py-3 px-6">{entry.comment || '-'}</td>
          </tr>
         ))
        )}
       </tbody>
      </table>
     </div>
    </div>
   </div>
  );
```

```
}
export default Journal;
```

../client/src/pages/Add.js

```
import React, { useState } from 'react';
import axios from 'axios';
const PERIOD_OPTIONS = ['ранок', 'обід', 'вечеря', 'перекус', 'перед їжею', 'після їжі'];
function Add() {
  const [value, setValue] = useState('');
  const [timestamp, setTimestamp] = useState(new Date().toISOString().slice(0, 16));
  const [period, setPeriod] = useState('');
  const [comment, setComment] = useState('');
  const [status, setStatus] = useState(null);
  const handleSubmit = async () => {
    if (!value || isNaN(value)) {
          setStatus('❌ Введіть коректне значення глюкози');
      return;
    }
    try {
      const token = localStorage.getItem('token');
      await axios.post(
        'http:
        {
          value: parseFloat(value),
          timestamp,
          period,
          comment,
          source: 'manual',
        },
        {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        }
      );
      setStatus('✅ Дані збережено успішно');
      setValue('');
      setPeriod('');
      setComment('');
    } catch (err) {
      console.error(err);
      setStatus('❌ Помилка при збереженні');
    }
  };
  return (
    <div className="bg-white min-h-screen py-10 px-6 md:px-16">
      <div className="max-w-2xl mx-auto bg-[#fff9f7] p-8 rounded-2xl shadow-md">
        <h2 className="text-4xl font-bold text-center mb-10 text-[#5E2B2B]">➕ Додавання показників</h2>
        <div className="mb-6">
          <label className="block mb-2 font-semibold text-[#5E2B2B]">Рівень глюкози (мг/дл):</label>
          <div className="flex">
            <input
              type="number"
              value={value}
              onChange={(e) => setValue(e.target.value)}
              placeholder="Наприклад, 95"
              className="flex-1 border border-[#d8b4a0] rounded-l-md px-4 py-2 focus:outline-none"
            />
            <button
              disabled
              className="bg-gray-300 text-gray-600 px-4 py-2 rounded-r-md cursor-not-allowed"
            >
              Підвантажити з датчика
            </button>
          </div>
        </div>
        <div className="mb-6">
          <label className="block mb-2 font-semibold text-[#5E2B2B]">Дата та час вимірювання:</label>
          <input
            type="datetime-local"
            value={timestamp}
            onChange={(e) => setTimestamp(e.target.value)}
            className="w-full border border-[#d8b4a0] rounded-md px-4 py-2 focus:outline-none"
          />
        </div>
        <div className="mb-6">
          <label className="block mb-2 font-semibold text-[#5E2B2B]">Період:</label>
          <select
            value={period}
            onChange={(e) => setPeriod(e.target.value)}
            className="w-full border border-[#d8b4a0] rounded-md px-4 py-2 bg-[#f5e7e1] text-[#5E2B2B] focus:outline-none"
          >
            <option value="">-- Виберіть період --</option>
            {PERIOD_OPTIONS.map((p) => (
              <option key={p} value={p}>{p}</option>
            ))}
          </select>
        </div>
        <div className="mb-6">
          <label className="block mb-2 font-semibold text-[#5E2B2B]">Коментар:</label>
          <textarea
            rows="3"
            value={comment}
            onChange={(e) => setComment(e.target.value)}
            placeholder="Наприклад: після сніданку, почувався добре"
            className="w-full border border-[#d8b4a0] rounded-md px-4 py-2 focus:outline-none"
          />
        </div>
        <button
          onClick={handleSubmit}
          className="w-full bg-[#7c2d2d] text-white font-semibold py-3 rounded-md hover:bg-[#5e2b2b] transition"
```

```
      >
       Зберегти
      </button>
      {status && (
        <p className={`mt-6 text-center font-semibold
${status.includes('успішно') ? 'text-green-600' : 'text-
red-600'}`}>
          {status}
        </p>
      )}
    </div>
  </div>
 );
}
export default Add;
```

../client/src/pages/Analytics.js

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import {
  LineChart, Line, XAxis, YAxis, CartesianGrid,
Tooltip, Legend, ResponsiveContainer
} from 'recharts';
const FILTERS = {
 '7': 'Тиждень',
 '14': '2 тижні',
 '30': 'Місяць',
 '90': '3 місяці'
};
function Analytics() {
 const [data, setData] = useState([]);
 const [filterDays, setFilterDays] = useState('30');
 useEffect(() => {
  loadData();
 }, []);
 const loadData = async () => {
  try {
   const token = localStorage.getItem('token');
   const res = await axios.get('http:
    headers: {
     Authorization: `Bearer ${token}`,
    },
   });
   setData(res.data);
  } catch (err) {
   console.error('Помилка завантаження даних', err);
  }
 };
 const getFilteredData = () => {
  const now = new Date();
  const ms = parseInt(filterDays) * 24 * 60 * 60 * 1000;
  const fromDate = new Date(now - ms);
  return data
   .filter(e => new Date(e.timestamp) >= fromDate)
   .map(e => ({
    ...e,
    date: new Date(e.timestamp).toLocaleString()
   }));
 };
 const getAverage = () => {
  const filtered = getFilteredData();
  if (filtered.length === 0) return 0;
  const sum = filtered.reduce((acc, e) => acc + e.value,
0);
  return (sum / filtered.length).toFixed(1);
 };
 const getRiskCounts = () => {
  const filtered = getFilteredData();
  const low = filtered.filter(e => e.value < 70).length;
  const high = filtered.filter(e => e.value > 140).length;
  return { low, high };
 };
 const filteredData = getFilteredData();
 const avg = getAverage();
 const risks = getRiskCounts();
 return (
  <div className="bg-white min-h-screen py-10 px-6
md:px-16">
   <div className="max-w-6xl mx-auto">
    <h2 className="text-4xl font-bold text-center mb-
10 text-[#5E2B2B]">
     Графіки та аналітика
    </h2>
    <div className="flex justify-end mb-6">
      <label className="mr-3 font-medium text-
[#5E2B2B]">Період:</label>
     <select
      className="border border-[#d8b4a0] rounded-
md px-4 py-2 bg-[#f5e7e1] text-[#5E2B2B]
focus:outline-none"
      value={filterDays}
                                  onChange={(e) =>
setFilterDays(e.target.value)}
     >
      {Object.entries(FILTERS).map(([key, label]) =>
(
                                   <option key={key}
value={key}>{label}</option>
      ))}
     </select>
    </div>
     <div className="bg-[#fff9f7] p-6 rounded-xl
shadow-md mb-8">
      <p className="text-lg text-[#5E2B2B] mb-
2"><strong>Середнє значення:</strong> {avg}
мг/дл</p>
      <p className="text-lg text-[#5E2B2B] mb-
2"><strong>Занизькі значення (&amp;lt;70):</strong>
{risks.low}</p>
                <p className="text-lg text-
[#5E2B2B]"><strong>Завищені значення
(&amp;gt;140):</strong> {risks.high}</p>
    </div>
     <div className="bg-[#fff9f7] p-6 rounded-xl
shadow-md">
            <ResponsiveContainer width="100%"
height={300}>
      <LineChart data={filteredData}>
       <CartesianGrid strokeDasharray="3 3" />
       <XAxis dataKey="date" tick={{ fontSize: 10 }}
/>
       <YAxis domain={[50, 180]} />
```

```
          <Tooltip />
          <Legend />
            <Line type="monotone" dataKey="value" name="Глюкоза" stroke="#7c2d2d" activeDot={{ r: 8 }} />
          </LineChart>
        </ResponsiveContainer>
      </div>
    </div>
  </div>
 );
}
export default Analytics;
```

../client/src/pages/Info.js

```
import React from 'react';
function Info() {
 return (
  <div className="bg-white min-h-screen py-10 px-6 md:px-16">
    <div className="max-w-4xl mx-auto bg-[#fff9f7] p-8 rounded-2xl shadow-md">
     <h2 className="text-4xl font-bold text-center mb-10 text-[#5E2B2B]">🛢 Корисна інформація</h2>
      <section className="mb-8">
        <h3 className="text-xl font-semibold text-[#7c2d2d] mb-2">1. Основи контролю рівня глюкози</h3>
       <p className="text-[#4b3b3b]">
        Контроль рівня глюкози в крові — ключовий компонент для людей із цукровим діабетом. Регулярне вимірювання
          дозволяє відстежувати зміни та запобігати ускладненням. Важливо знати свої межі та динаміку змін упродовж дня.
       </p>
      </section>
      <section className="mb-8">
        <h3 className="text-xl font-semibold text-[#7c2d2d] mb-2">2. Харчові рекомендації</h3>
         <ul className="list-disc pl-6 text-[#4b3b3b] space-y-1">
           <li>Уникайте продуктів із високим глікемічним індексом</li>
          <li>Вживайте більше овочів, цільнозернових продуктів та білків</li>
          <li>Контролюйте порції та частоту прийомів їжі</li>
        </ul>
      </section>
      <section className="mb-8">
        <h3 className="text-xl font-semibold text-[#7c2d2d] mb-2">3. Вплив фізичної активності</h3>
       <p className="text-[#4b3b3b]">
        Помірна фізична активність сприяє зниженню рівня глюкози. Рекомендується 30 хвилин ходьби, йоги або іншої активності
          щодня. Уважно слідкуйте за рівнем цукру перед та після занять.
       </p>
      </section>
      <section>
        <h3 className="text-xl font-semibold text-[#7c2d2d] mb-4">4. Нормальні рівні глюкози для різних вікових груп</h3>
       <div className="overflow-x-auto">
         <table className="w-full border border-[#d8b4a0] bg-white text-[#4b3b3b] rounded-lg">
          <thead className="bg-[#7c2d2d] text-white">
           <tr>
             <th className="py-3 px-4 text-left">Вік</th>
            <th className="py-3 px-4 text-left">Норма натще (мг/дл)</th>
             <th className="py-3 px-4 text-left">Після їжі (мг/дл)</th>
           </tr>
          </thead>
          <tbody>
           <tr className="border-t border-[#f1d4d4]">
             <td className="py-2 px-4">Діти (до 14 років)</td>
            <td className="py-2 px-4">70–100</td>
            <td className="py-2 px-4">до 140</td>
           </tr>
           <tr className="border-t border-[#f1d4d4]">
             <td className="py-2 px-4">Дорослі (15–60 років)</td>
            <td className="py-2 px-4">70–99</td>
            <td className="py-2 px-4">до 140</td>
           </tr>
           <tr className="border-t border-[#f1d4d4]">
             <td className="py-2 px-4">Старші 60 років</td>
            <td className="py-2 px-4">80–110</td>
            <td className="py-2 px-4">до 150</td>
           </tr>
          </tbody>
         </table>
        </div>
      </section>
    </div>
  </div>
 );
}
export default Info;
```

../server/index.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const Glucose = require('./models/Glucose');
const User = require('./models/User');
const app = express();
const PORT = 3000;
const SECRET_KEY = "GLUCOMETER2807";
app.use(cors());
app.use(express.json());
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
```

```
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.sendStatus(401);
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
}
mongoose.connect('mongodb:
    .then(() => console.log('✅ Підключено до MongoDB'))
     .catch(err => console.error('❌ Помилка підключення до MongoDB:', err));
app.post('/api/login', async (req, res) => {
  const { email, password } = req.body;
   if (!email || !password) return res.status(400).json({ error: 'Введіть email і пароль' });
  const user = await User.findOne({ email });
  if (!user) return res.status(400).json({ error: 'Невірний email або пароль' });
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ error: 'Невірний email або пароль' });
  const token = jwt.sign({ id: user._id }, SECRET_KEY, { expiresIn: '1h' });
  res.json({ token });
});
app.get('/api/glucose', authenticateToken, async (req, res) => {
  try {
       const readings = await Glucose.find().sort({ timestamp: -1 });
    res.json(readings);
  } catch (err) {
     res.status(500).json({ error: 'Помилка отримання даних' });
  }
});
app.post('/api/glucose', authenticateToken, async (req, res) => {
  try {
    const { value, timestamp, period, comment, source } = req.body;
    const newEntry = new Glucose({
      value,
      timestamp: timestamp || new Date(),
      period: period || '',
      comment: comment || '',
      source: source || 'manual',
    });
    await newEntry.save();
    res.status(201).json(newEntry);
  } catch (err) {
    res.status(500).json({ error: 'Помилка збереження даних' });
  }
});
app.post('/api/iot', async (req, res) => {
  try {
     console.log('📡 Отримано запит з пристрою:', req.body);
    const { value } = req.body;
      if (!value) return res.status(400).json({ error: 'Значення глюкози обов’язкове' });
    const entry = new Glucose({
      value,
      timestamp: new Date(),
      period: '',
      comment: '',
      source: 'iot'
    });
    await entry.save();
    res.status(201).json({ success: true });
  } catch (err) {
    console.error('❌ Помилка IoT-запиту:', err);
    res.status(500).json({ error: 'Помилка збереження з пристрою' });
  }
});
app.post('/api/glucose/from-iot', async (req, res) => {
  try {
    const { value } = req.body;
    if (!value) {
        return res.status(400).json({ error: 'Потрібно передати значення глюкози' });
    }
    const newEntry = new Glucose({
      value: parseFloat(value),
      timestamp: new Date(),
      period: '',
      comment: 'Від IoT-пристрою',
      source: 'iot',
    });
    await newEntry.save();
      res.status(201).json({ message: 'Запис успішно збережено' });
  } catch (err) {
    console.error('❌ Помилка збереження з IoT:', err);
    res.status(500).json({ error: 'Серверна помилка' });
  }
});
app.listen(PORT, () => console.log(`✅ Сервер запущено на http:
```