Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-орієнтована програмна система генерації графіків поточного стану пацієнта для медичних закладів
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-3

Задорожний Н.С.
(прізвище, ініціали)

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

Керівник доц. кафедри ПІ Побіженко І.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р

Харківський національний університет радіоелектроніки

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

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Задорожному Нікіті Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-орієнтована програмна система генерації графіків поточного стану пацієнта для медичних закладів
Затверджена наказом по університету від 20.5.2024р. № 471 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 15.06.2024
3. Вихідні дані до роботи Розробити веб-орієнтовану програму систему з використанням фреймворку Next.js та React для генерації графіків поточного стану пацієнта для медичних закладів.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	10.04.2024	*виконано*
2	Створення специфікації ПЗ	12.04.2024	*виконано*
3	Проектування ПЗ	15.03.2024	*виконано*
4	Розробка ПЗ	26.05.2024	*виконано*
5	Тестування ПЗ	28.05.2024	*виконано*
6	Оформлення пояснювальної записки	29.05.2024	*виконано*
7	Підготовка презентації та доповіді	06.06.2024	*виконано*
8	Оцінка роботи рецензентом, отримання відзиву від керівника кваліфікаційної роботи, попередній захист роботи та проходження нормо-контролю	08.06.2024 - 15.06.2024	*виконано*
9	Здача роботи у електронний архів допуск роботи до захисту завідувачем кафедри	15.06.2024	*виконано*
10	Захист кваліфікаційної роботи	19.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка)_____ Задорожний Н.С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Побіженко І.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 61 стор., 12 рис., 1 табл, 11 джерел.

ПАЦІЄНТИ, ВЕБ-ЗАСТОСУНОК, ГРАФІКИ, МЕДИЧНИЙ ЗАКЛАД, NEXTJS, TYPESCRIPT, TRPC, SUPABASE

Об’єкт розробки – веб-орієнтована програмна система для генерації графіків поточного стану пацієнта у медичних закладах.

Мета розробки – створення веб-застосунку, який генеруватиме графіки стану здоров’я пацієнту для медичних закладів.

Метод рішення – мова програмування TypeScript, бібліотека React, фреймворк NextJs, середовище розробки Visual Studio Code.

У результаті розроблена веб-орієнтована програмна система для генерації графіків поточного стану пацієнта яка є важливим інструментом для медичних закладів, який допомагає оптимізувати та поліпшувати процес моніторингу пацієнтів.

PATIENTS, WEB APPLICATION, GRAPHS, MEDICAL DEPARTMENT, NEXTJS, TYPESCRIPT, TRPC, SUPABASE

The object of development is a web-based software system for generating graphs of the patient's current condition in medical institutions.

The purpose of the development is to create a web application that will generate graphs of the patient's state of health for medical institutions.

The solution method is the TypeScript programming language, the React library, the NextJs framework, and the Visual Studio Code development environment.

As a result, a web-oriented software system was developed for generating graphs of the patient's current condition, which is an important tool for medical institutions that helps to optimize and improve the process of monitoring patients.

Я, Задорожний Нікіта Сергійович, студент гр. ПЗПІ-20-3, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Веб-орієнтована програмна система генерації графіків поточного стану пацієнта для медичних закладів», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі 8
1.1 Аналіз предметної галузі 8
1.2 Виявлення та вирішення проблем 9
1.3 Постановка задачі 10
2 Формування вимог до програмної системи 12
2.1 Збір та аналіз вимог 12
2.2 Формування вимог 13
2.2.1 Функціональні вимоги 13
2.2.2 Нефункціональні вимоги 15
3 Архітектура та проектування програмного забезпечення 17
3.1 UML проєктування ПЗ 17
3.2 Проектування архітектури ПЗ 20
3.3 Проектування структури зберігання даних 22
3.4 Створення UI/UX 22
4 Опис прийнятих програмних рішень 26
4.1 Використання бібліотеки ReactTable 26
4.2 Використання бібліотеки RadixUI 27
5 Тестування програмного забезпечення 29
Висновки 33
Перелік джерел посилання 34
Додаток А Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ 36
Додаток Б Слайди презентації 37
Додаток В Специфікація вимог до програмного продукту ВСТУП 44
Додаток Г Код для створення WebSocket та підключення його 59

ВСТУП

У сучасному медичному середовищі генерація графіків поточного стану пацієнтів є ключовим аспектом у діяльності багатьох медичних закладів. Різноманітність і обсяг медичних даних ставлять перед закладами важливі завдання щодо зберігання, організації та оптимізації доступу до цих даних.

Розробка веб-орієнтованої програмної системи для генерації графіків поточного стану пацієнтів має на меті надати зручний і ефективний інструмент для візуалізації медичної інформації. Ця система спрямована на полегшення роботи з медичними даними, забезпечення швидкого доступу до них та покращення продуктивності медичного персоналу. Мета – зменшити кількість нещасних випадків (через невчасну реакцію лікарів), підвищити якість обслуговування пацієнтів, розвантажити і без того навантажених лікарів та інших медичних працівників, економія часу, зменшення використання паперу.

У цій роботі буде надано опис підприємства, що потребує програмну систему, проведений аналіз предметної галузі, визначені вимоги до програмної системи, розглянута архітектура та проектування системи, а також сформульовано висновки щодо роботи та перспектив її подальшого розвитку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У сучасному світі важко уявити функціонування медичних закладів без передових технологій, таких як веб-орієнтовані програмні системи, що поєднують у собі аналіз даних та генерування графіків. Ці технології змінюють підхід до управління і поліпшують якість медичного обслуговування.

Веб-орієнтована програмна система для медичних закладів забезпечує генерацію графіків [1], які відображають поточний стан пацієнтів. Завдяки постійному моніторингу, лікарі можуть отримувати тривожні сигнали та сповіщення щодо змін у стані пацієнта негайно. Це дозволяє швидше реагувати на можливі проблеми та призначити необхідне лікування.

Перш за все, такі програми дозволяють автоматизувати процес отримання повідомлень про стан пацієнтів, враховуючи їхні показники здоров'я. Вони аналізують дані про пацієнтів для побудови графіка [1], що забезпечує найефективнішу роботу медичного персоналу.

Вдруге, ці системи генерують індивідуалізовані підказки та рекомендації для медичного персоналу щодо обслуговування пацієнтів. Вони враховують дані про стан пацієнтів, щоб запропонувати найбільш ефективний підхід до медичного обслуговування.

На даний момент в України процес побудований на передачі інформації через чергових, які роблять обхід в зазначений час. Тому тут, по-перше, чергові лікарі можуть не встигнути побачити проблему через високу завантаженість, по-друге для цього використовують дуже багато паперу, що шкодить навколишньому середовищу, ну і також є людський фактор, тобто помиляються всі і цією програмною системою ми можемо зменшити відсоток помилок, що потягне за собой збереження коштів а також більш розумне лікування, скоротимо час на дослідження.

Ринок медичного програмного забезпечення на міжнародному рівні демонструє активний розвиток у галузі автоматизації. Веб-орієнтовані системи, які

забезпечують генерацію графіків, стають стандартом у багатьох країнах. Це сприяє покращенню ефективності медичних закладів і забезпечує високий рівень задоволеності як медичного персоналу, так і пацієнтів.

Такий аналіз підкреслює необхідність у впровадженні інноваційних веб-орієнтованих систем управління для поліпшення роботи медичних закладів, зменшення ризиків помилок і підвищення якості медичного обслуговування.

1.2 Виявлення та вирішення проблем

Після попереднього аналізу виділяються наступні проблеми, що виникають у медичних закладах та пацієнтів:

- Ручне створення графіків стану здоров’я пацієнтів часто не враховує всі показники та дуже швидко стає неактуальним;
- Ручний облік пацієнтів може призводити до помилок і вимагає багато часу;
- Складнощі подачі заявок на медичне обслуговування;
- Відсутність можливості перегляду графіку прийому пацієнтів;
- Відсутність централізованого інструменту управління інформацією про всіх пацієнтів;
- Ручне документування займає багато часу;
- Ризик помилок достатньо високий;
- Актуальність інформації має дуже малий проміжок часу;
- Лікар може не помітити різкі зміни в стану пацієнта;
- Відсутність змоги постійно контролювати деякі показники;
- Досить складно дізнатися за яким лікарем пацієнт записан.

Метою цієї роботи є розробка веб-орієнтованої програмної системи для автоматичного створення графіків стану здоров'я пацієнтів, враховуючи їхні медичні потреби та індивідуальні характеристики. Також система буде вести точний облік присутності пацієнтів, що підвищить безпеку медичного обслуговування. Впровадження веб-орієнтованої системи дозволить медичним закладам ефективніше виконувати завдання, покращить якість обслуговування

пацієнтів і сприятиме автоматизації облікових процесів. Крім того, система забезпечить зручний доступ до медичних даних, що полегшить роботу медичного персоналу та пацієнтів. Пацієнти можуть переглядати свою медичну історію та графіки прийому лікарських засобів у режимі онлайн.

Система також дозволить лікарям відстежувати динаміку стану здоров'я пацієнтів у режимі реального часу. Це сприятиме своєчасному коригуванню призначень та лікувальних заходів. Важливим аспектом є інтеграція з іншими медичними системами для забезпечення комплексного підходу до лікування. Впровадження такої системи також зменшить кількість помилок, пов'язаних з людським фактором, та забезпечить високий рівень конфіденційності даних.

1.3 Постановка задачі

Після аналізу сфери медичних закладів, існуючих викликів та вивчення реальних потреб ринку стає зрозумілою потреба у створенні веб-орієнтованої програмної системи, яка зможе ефективно відповідати на нагальні питання, що виникають у цій галузі. Веб-орієнтована програмна система "MediCareHub" надаватиме наступні можливості:

- Перегляд пацієнтів, які закріплені за користувачем;
- Налаштування та планування медичних процедур та обстежень;
- Перегляд медичного персоналу та спеціалізацій;
- Додавання та перегляд інформації про пацієнтів;
- Перегляд розкладу медичних процедур та візитів пацієнтів;
- Перегляд історії медичних візитів та процедур;
- Відображення медичних груп та спеціалізацій лікарів;
- Генерація розкладу медичних процедур та візитів;
- Перегляд списку пацієнтів у медичних групах;
- Генерація графіків поточного стану пацієнта;
- Отримання повідомлень про критичні показники пацієнта;
- Відсутність людського фактору та помилок;
- Більш детальний аналіз стану.

- Контроль пацієнтів закріпленими за лікарями.

Ця веб-орієнтована програмна система спрямована на полегшення доступу до медичних послуг, оптимізацію процесів запису та візитів пацієнтів, а також забезпечення зручності управління медичними процедурами як для пацієнтів, так і для медичного персоналу. Для розробки веб-застосунку використовується мова програмування TypeScript [2], бібліотека React [3], фреймворк Next.js [4], середовище розробки Visual Studio Code. Застосунок взаємодіє із сервером за допомогою TRPC [5]. Керування процесом розробки відбувається через інструмент для управління проектами Linear та використовує методологію Scrum.

Система дозволить медичному персоналу ефективніше керувати часом та ресурсами, а пацієнтам забезпечить швидкий і зручний доступ до медичних послуг. Крім того, використання сучасних технологій сприятиме підвищенню надійності та безпеки зберігання медичних даних. Інтеграція з іншими медичними системами забезпечить комплексний підхід до лікування та моніторингу здоров'я пацієнтів. Це також сприятиме покращенню комунікації між медичним персоналом та пацієнтами, роблячи процес лікування більш прозорим і зрозумілим.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Збір та аналіз вимог

Визначення вимог є критичним у процесі створення будь-якої системи, оскільки це передбачає глибоке розуміння сучасних ринкових умов і індивідуальних потреб користувачів. Успішність системи прямо залежить від аналітичного аналізу інформації про користувачів, ринкових тенденцій та діяльності конкурентів. Ця інформація визначає основні вимоги, які відповідають стратегічним цілям бренду й гарантують довгострокову ефективність системи. Глибоке розуміння вимог і очікувань цільової аудиторії є ключовим у формулюванні функціональних характеристик системи, що відповідають потребам ринку та сприяють збільшенню попиту. У рамках цього процесу була розроблена таблиця для детального аналізу цільової аудиторії, що дозволяє чітко визначити ключові параметри для розробки (див. табл. 3.1).

Таблиця 3.1 – Аналіз цільової аудиторії

Цільова аудиторія	Больова точка	Мета	Як система допоможе
Лікарі	− відсутність ефективного інструменту для керування графіками стану здоров'я; − складність у відстеженні критичних показників здоров’я;	− максимізація ефективності керування графіками; − покращення управління станом пацієнтів;	− автоматизація процесів отримання графіків; − датчики для відстеження показників здоров’я;

Створення таблиці здійснювалось за допомогою обговорення, де члени команди зустрілись для обміну думками щодо цільової аудиторії.

2.2 Формування вимог

2.2.1 Функціональні вимоги

Функціональні вимоги визначають необхідні функції та процедури системи, які мають бути реалізовані для того, щоб користувачі могли повноцінно використовувати її можливості. Документація цих вимог була виконана таким чином, щоб вона була доступною всім членам команди, використовуючи Linear, що є ключовим елементом в процесі створення ефективної системи.

Для визначення функціональних вимог використовувались User Story, доповнені критеріями прийняття для забезпечення детальності. User Story були організовані в Epics, що сприяло кращому управлінню робочим процесом та спрощувало роботу розробників.

Для формулювання функціональних вимог використовувались User Story, кожне з яких було доповнене критеріями прийняття для детальності та ясності. Ці User Story були інтегровані в спрінти, що дозволяло ефективно планувати та управляти робочим процесом, забезпечувало послідовність розвитку проекту та спрощувало взаємодію між членами команди. Крім того, кожен спрінт передбачав регулярні зустрічі для перегляду прогресу, що стимулювало швидке вирішення виникаючих проблем та адаптацію до змін у проектних вимогах, забезпечуючи гнучкість.

Перелічимо User Story для кожної ітерації системи «MediCareHub», які відображають основні функції та можливості веб-орієнтованої програми. Наприклад, перший спрінт пов’язаний з основами процесами, тобто веб функціоналом для адміністраторів. Він включає в себе наступні User Story:

- «Управління користувачами;
- «Управління пацієнтами»;
- «Управління персоналом».

Кожна User Story має свій детальний опис.

Веб-орієнтована програмна система має задовольняти низку затверджених функціональних вимог, які були чітко визначені та структуровані. Ці вимоги викладено у вигляді User Story, дозволяючи наочно представити основні функції та можливості, які система повинна надати своїм користувачам.

US-1: Як користувач, я хочу авторизуватися в системі, щоб мати доступ до персоналізованого інтерфейсу і функцій.

US-2: Як адміністратор, я хочу переглядати список користувачів до яких маю доступ, щоб переглядати і редагувати їх інформацію.

US-3: Як адміністратор, я хочу створювати облікові записи для нових пацієнтів.

US-4: Як адміністратор, я хочу створювати облікові записи для докторів.

US-5: Як адміністратор, я хочу переглядати профілі персоналу, щоб легше управляти ресурсами.

US-6: Як адміністратор, я хочу переглядати список поточних пацієнтів.

US-7: Як адміністратор, я хочу переглядати графіки стану пацієнта.

US-8: Як адміністратор, я хочу мати можливість зареєструвати новий сенсор для пацієнта.

US-9: Як пацієнт, я хочу отримувати дані про свій стан.

US-10: Як лікар, я хочу переглядати інформацію про стан здоров’я пацієнтів, яких я лікую.

US-11: Як лікар, я хочу отримувати сповіщення про критичний стан пацієнта.

US-12: Як лікар, я хочу переглядати графік важливих для життя показників.

US-13: Як лікар, я хочу мати можливість реєструвати пацієнта в системі.

US-14: Як лікар, я хочу отримувати сповіщення про відключення сенсорів.

US-15: Як лікар, я хочу мати можливість друку графіків.

Цей формат представлення вимог допомагає команді розробників більш ефективно організувати процес розробки та забезпечити високий рівень відповідності кінцевого продукту очікуванням користувачів.

2.2.2 Нефункціональні вимоги

Для веб-орієнтованої програмної системи генерації графіків «MediCareHub» нефункціональні вимоги відіграють ключову роль у забезпеченні якості, безпеки, та зручності використання. Варто зазначити, що нефункціональних вимог може бути дуже багато, тому було зосереджену увагу на тих, які відіграють ключову роль в функціонуванні нашого веб-застосунку. Такими вимогами є:

- продуктивність та швидкість відгуку;
- надійність;
- масштабованість;
- безпека;
- локалізація;
- інтернаціоналізація;
- підтримка та обслуговування.

Тепер, після того як ми визначили необхідні нефункціональні вимоги, можна описати кожну вимогу більш детально.

Продуктивність та швидкість відгуку. Програма має забезпечувати швидку відповідь на запити користувачів, не більше ніж за 3 секунди. Обробка та завантаження великого обсягу даних, таких як список всіх показників здоров’я пацієнта, повинні займати не більше 5 секунд. Для досягнення цих показників слід використовувати оптимізацію запитів до бази даних і кешування часто запитуваних даних. Також важливо мінімізувати кількість звернень до сервера, використовуючи асинхронні запити та технології, такі як WebSocket для реального часу.

Надійність. Веб-застосунок повинен забезпечувати високу доступність, працюючи неперервно 24/7 без значних перерв. Крім того, він повинен використовувати децентралізовані системи баз даних, щоб запобігти втраті інформації. Також система повинна мати механізми автоматичного відновлення після збоїв і регулярного резервного копіювання даних. Для підвищення надійності можна використовувати кластеризацію серверів і балансування навантаження.

Масштабованість. Система повинна мати можливість обслуговувати до 10,000 активних користувачів одночасно і зберігати до 1 мільйона записів, не погіршуючи продуктивність або швидкість відгуку системи. Важливо передбачити горизонтальне масштабування для збільшення кількості серверів у разі зростання навантаження. Крім того, використання мікросервісної архітектури дозволить легше розподіляти навантаження між різними компонентами системи.

Безпека. Захист даних має забезпечуватися за допомогою сучасних методів шифрування, наприклад, SHA-256. Також необхідно використовувати протоколи SSL/TLS для захищеного зв'язку між клієнтом і сервером. Важливим аспектом є впровадження багатофакторної автентифікації для підвищення безпеки доступу до системи. Регулярний аудит безпеки і тестування на проникнення допоможуть виявляти і усувати потенційні уразливості.

Локалізація. Усі елементи інтерфейсу користувача, такі як кнопки, меню, поля введення і повідомлення про помилки, повинні бути доступні як англійською, так і українською мовами. Також система має автоматично адаптувати формати дат, часу та чисел до культурних та регіональних відмінностей кожної з цих мов. Це забезпечить комфортне використання застосунку для користувачів з різних регіонів. Важливо також передбачити можливість додавання інших мов у майбутньому.

Інтернаціоналізація. Програма має використовувати I18n для підтримки різних мовних алфавітів. Веб-застосунок має підтримувати динамічну зміну мови на льоту, без необхідності перезавантаження застосунку. Це значно покращить користувацький досвід, дозволяючи швидко перемикатися між мовами залежно від потреб користувача. Для цього необхідно використовувати бібліотеки, такі як i18next, що підтримують ці функції.

Забезпечення усіх цих вимог дозволить створити надійну, ефективну та зручну для користувачів систему, що відповідатиме сучасним стандартам якості та безпеки.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Під час розробки веб-додатку MediCareHub, команда використовувала різноманітні UML-діаграми для кращого розуміння та документування архітектури системи. Серед ключових діаграм були такі:

- Діаграми використання (Use Case Diagrams), що показують основні функції, які система має виконувати для різних типів користувачів.
- Діаграми активності (Activity Diagrams), які відображають процеси та дії, що відбуваються в системі.
- Діаграми послідовності (Sequence Diagrams), що демонструють взаємодію між об'єктами в часі для виконання конкретних завдань.

Особлива увага була зосереджена на діаграмах використання, які допомагають виокремити основні сценарії взаємодії користувачів з веб-застосунком. Було створено кілька варіантів таких діаграм для адміністраторів садочків, вихователів і батьків, кожна з груп має свої унікальні функції системи і права доступу.

Інтеграція цих діаграм у процес розробки дозволяє систематично перевіряти відповідність функціоналу реальним потребам користувачів і оптимізувати процеси, усуваючи потенційні недоліки в інтерфейсі та взаємодіях ще до випуску продукту. Це важливо для створення зрозумілого та надійного продукту, який відповідає очікуванням користувачів і забезпечує їхнє задоволення.

На рисунку 3.1 зображено Use Case діаграму для адміністраторів, яка демонструє основні взаємодії з системою. Адміністратор може виконувати авторизацію, переглядати або редагувати списки користувачів, переглядати графіки, добавляти сенсори. Діаграма також показує можливість виходу з системи, що дає змогу адміністраторам зберігати безпеку доступу до своїх адміністративних функцій.

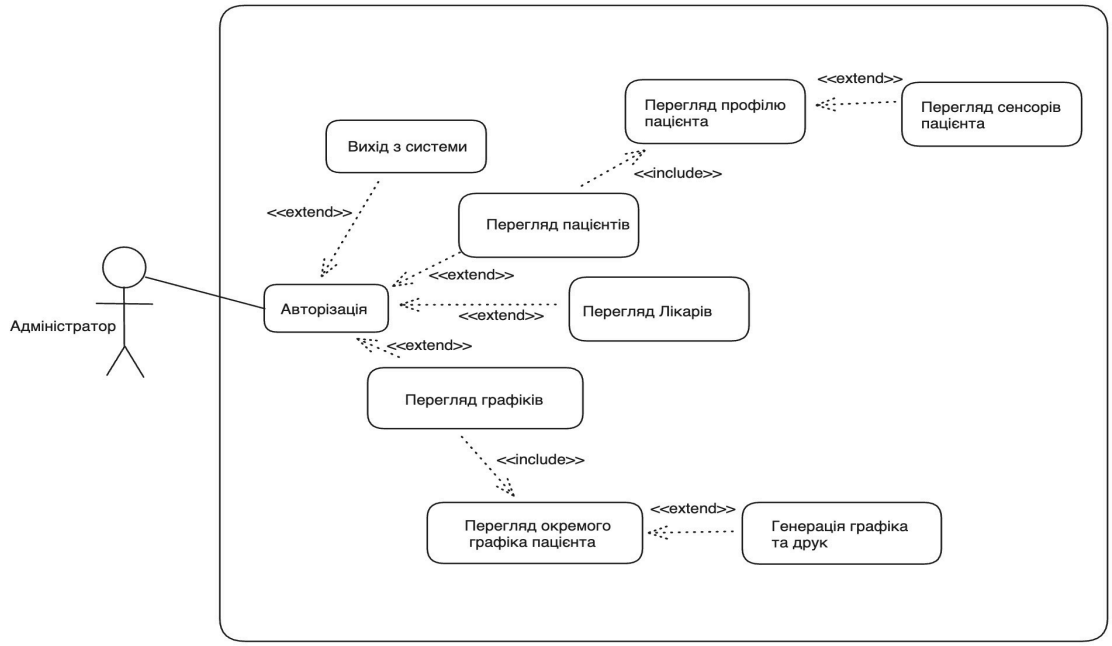


Рисунок 3.1 – Use-Case діаграма

На рисунку 3.2 зображено діаграму використання для лікарів. Від моменту авторизації, лікарі мають доступ до широкого спектру функцій.

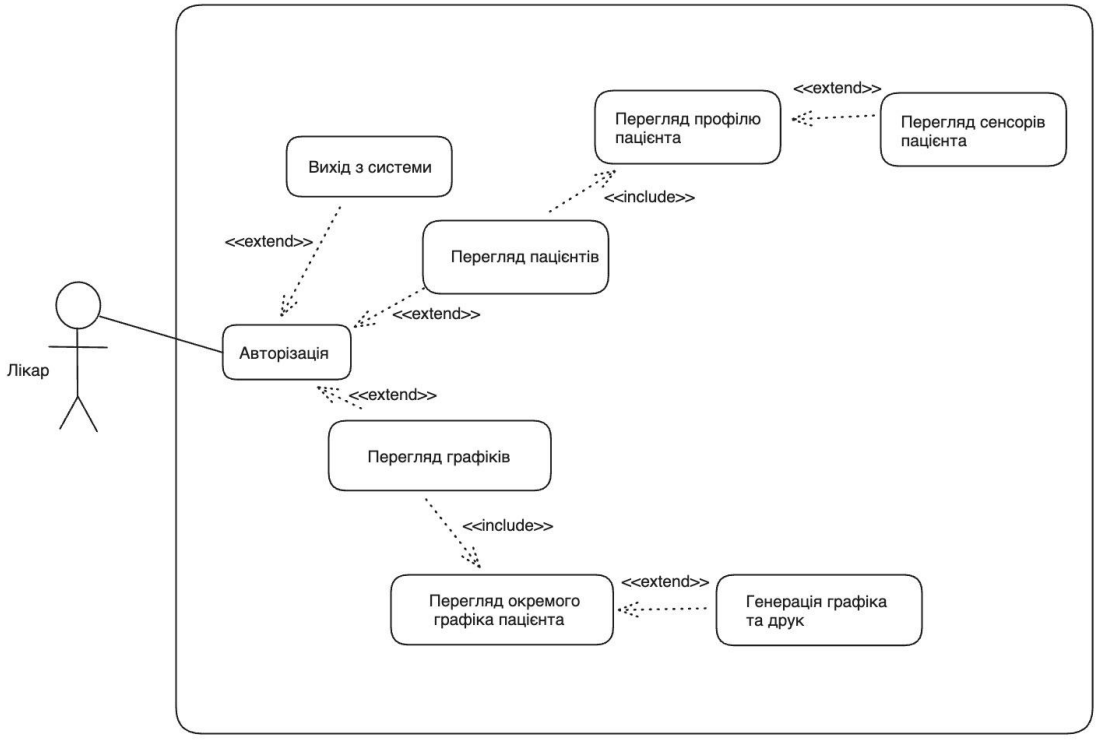


Рисунок 3.2 – Use Case діаграма лікарів (рисунок виконано самостійно)

Діаграма використання для лікарів також вказує на можливість керування профілями пацієнта, що включає додавання інформації про пацієнта, та реєстрування нових сенсорів. Важливою функцією є також перегляд графіків та отримання сповіщень, який дозволяє лікарю бути в курсі усіх станів пацієнтів пов'язаних із життям. Вихід з системи, як завершальний крок, забезпечує додаткову безпеку, гарантуючи, що особиста інформація залишається доступною лише під час активної сесії.

Діаграма розгортання, яка представлена на рисунку 3.3, відображає архітектуру системи веб-застосунку «MediCareHub», яка включає веб-застосунок, серверну частину та базу даних.

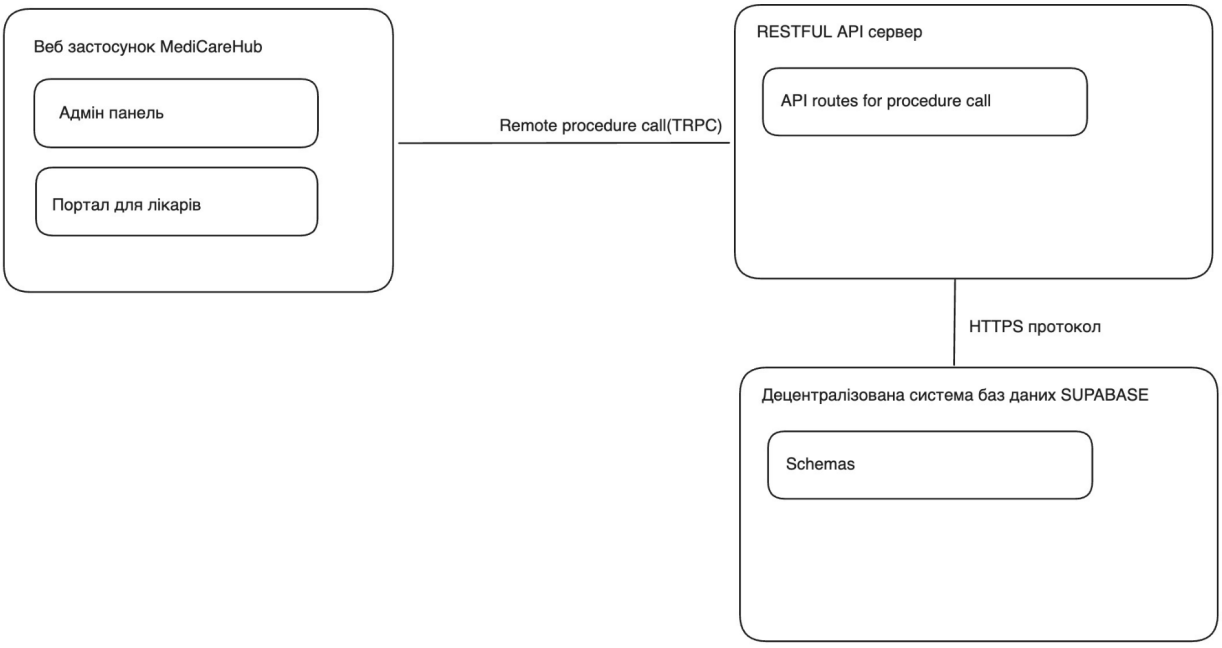


Рисунок 3.3 – Діаграма розгортання веб-застосунку «MediCareHub» (рисунок виконано самостійно)

Веб-застосунок «MediCareHub» складається з порталів для адміністратора та лікарів, з яких веб додаток взаємодіє з серверними компонентами через Remote Procedure Call.

Зображений на малюнку RESTFUL API сервер «MediCareHub» використовує TRPC API [5], що обробляє усі запити, після чого надсилає запит до бази даних. Децентралізована система баз даних SUPABASE [6] отримує дані з сенсорів.

Для обробки запитів авторизації і забезпечення безпекою застосунку використовується NEXT Auth [7]. Діаграма розгортання надає зрозуміле уявлення про те, як компоненти системи взаємопов'язані між собою для забезпечення правильної роботи веб-застосунку «MediCareHub». Діаграма підкреслює модульність та розподіленість системи, що є ключовими для сучасних веб-застосунків, гарантуючи користувачам швидкість і безпеку даних.

3.2 Проектування архітектури ПЗ

Архітектура веб-застосунку «MediCareHub» базується на клієнт-серверному підході, де веб-застосунок виступає як клієнт, а серверна частина обробляє бізнес-логіку і управління даними. Цей підхід дозволяє розділити відповідальності між клієнтськими пристроями та сервером.

Клієнт-серверна архітектура є дуже поширеним типом архітектури з численними перевагами. По-перше, ця архітектура дозволяє централізовано керувати даними та ресурсами. Сервер відповідає за обробку запитів, керування базою даних, ресурсами та бізнес-логікою, що спрощує взаємодію з різними клієнтами.

По-друге, така архітектура легко масштабується, дозволяючи збільшувати серверні ресурси для обслуговування більшої кількості клієнтів без змін у клієнтській частині.

По-третє, клієнти, що звертаються до сервера, потребують менше обчислювальних ресурсів, оскільки обробка даних відбувається на сервері. Це зменшує вимоги до апаратного забезпечення клієнтських пристроїв і поліпшує відгук системи, особливо на застарілих смартфонах.

Проте, не дивлячись на численні переваги, клієнт-серверна архітектура має основний недолік – система повністю залежить від стабільності та доступності

сервера. При відмові сервера може виникнути повна непрацездатність всіх клієнтів, що негативно впливає на користувачів.

Загалом, клієнт-серверна архітектура забезпечує високу ефективність, гнучкість та масштабованість веб-застосунку «MediCareHub». Вона дозволяє централізовано керувати даними та ресурсами, знижуючи вимоги до клієнтських пристроїв і забезпечуючи високий рівень безпеки та узгодженості даних. Незважаючи на залежність від стабільності сервера, правильне планування і впровадження резервних механізмів дозволить забезпечити безперебійну роботу системи і задовольнити потреби користувачів.

Коли користувач відвідує сторінку входу і вводить свій логін та пароль, ці дані надсилаються до сервісу NEXT Auth [7] для аутентифікації. Сервіс обробляє ці дані і, якщо логін і пароль введено правильно, повертає авторизаційний код. Цей код використовується веб-застосунком для отримання токена доступу. Токен використовується для запитів до API, де він перевіряється на валідність перед наданням доступу до даних у базі даних.

Під час запиту веб-застосунка до API-сервера, останній перевіряє токен. У разі успішної валідації API дозволяє доступ до запитаних даних і повертає відповідь клієнту. Усі дані зберігаються та обробляються в базі даних для централізованого управління і захисту інформації.

Основою веб-застосунку "MediCareHub" стала мова програмування TypeScript [2], яка є розширенням JavaScript і додає типізацію для зменшення помилок на етапі компіляції. Клієнтська частина застосунку була розроблена на основі популярної бібліотеки React [3] та фреймворку Next.js [4], які дозволяють створювати масштабовані та ефективні веб-додатки з високою оптимізацією.

Розробка проводилась у середовищі Visual Studio Code, яке дозволяє встановлювати різноманітні плагіни та інструменти для підвищення якості і швидкості розробки. Для контролю версій використовувалась система Git через її простоту та поширеність.

3.3 Проектування структури зберігання даних

Веб орієнтована програмна система MediCareHub покладається на сервер для зберігання і управління даними. Сервер взаємодіє із POSTGRES базою даних SUPABASE.

PostgreSQL була вибрана як основна SQL база даних через її потужні можливості управління транзакціями, що забезпечує надійність та консистентність даних навіть при великому обсязі операцій. Це вкрай важливо для нашого застосунку, оскільки штучний інтелект генерує значну кількість даних, і втрата або пошкодження будь-яких даних може серйозно вплинути на роботу системи.

Основні переваги PostgreSQL включають:

- Надійність та стійкість до високих навантажень: PostgreSQL добре справляється з великим обсягом транзакцій і запитів, забезпечуючи стабільну роботу системи навіть при інтенсивному використанні.
- Транзакційна підтримка: База даних PostgreSQL підтримує повне виконання ACID (Atomicity, Consistency, Isolation, Durability) для забезпечення надійності та цілісності даних.
- Розширені можливості: PostgreSQL підтримує багато функцій і розширень, таких як JSON-операції, геопросторові запити, аналітика та інші, що полегшують розробку складних додатків.
- Відкритий код та активна спільнота: PostgreSQL є вільним програмним забезпеченням з великою активною спільнотою розробників, що забезпечує швидкий розвиток і підтримку бази даних.

3.4 Створення UI/UX

User Experience (UX) і User Interface (UI) — це дві ключові складові, які відіграють важливу роль у розробці будь-якої клієнтської частини, з якою взаємодіє людина, особливо це важливо у веб-додатках. UX стосується загального враження користувача від використання додатка, включаючи те, наскільки зручно і інтуїтивно зрозуміло користувачам взаємодіяти з ним. Це охоплює всі аспекти

додатка: від навігації і швидкості роботи до зручності виконання необхідних дій та отримання необхідної інформації.

UI, з іншого боку, зосереджений на візуальних аспектах додатка — це дизайн інтерфейсу, включаючи розташування кнопок, кольорову гаму, шрифти і інші візуальні елементи, які впливають на те, як користувач сприймає додаток. Гарний UI робить додаток не тільки привабливим, але й забезпечує чіткість і простоту користування. Більше того, дуже ефективний і швидкий сервер не відчується користувачем так сильно, як гарний дизайн.

При розробці веб-додатків особлива увага до UX і UI є критично важливою, оскільки веб-додатки мають багато взаємодій із користувачем через браузер на різних пристроях. Важливо стежити за зручністю взаємодії та забезпечити відмінний дизайн, оскільки це покращує користувацький досвід і сприяє популярності додатка серед користувачів.

Інтерфейс веб-застосунку «MediCareHub» використовує дизайн систему Tailwind [8] для гарного та зрозумілого інтерфейсу. А також багато компонентів для побудови графіків, таких як REcharts [1].

На рисунку 3.4 зображено скріншот, де представлено декілька прикладів графіків «MediCareHub».

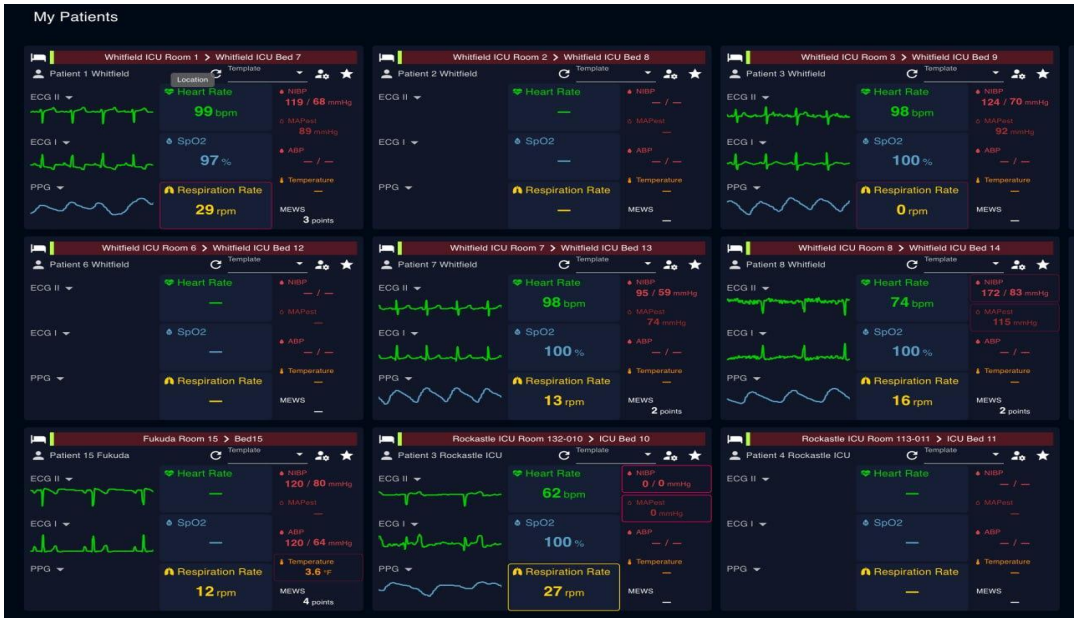


Рисунок 3.4 – Скріншот з наведеними графіками пацієнтів(рисунок виконано самостійно)

На знімках екрана видно графіки пацієнтів на яких зображені важливі життєві показники.

На рисунку 3.5 зображено скріншот, де представлено список пацієнтів «MediCareHub».

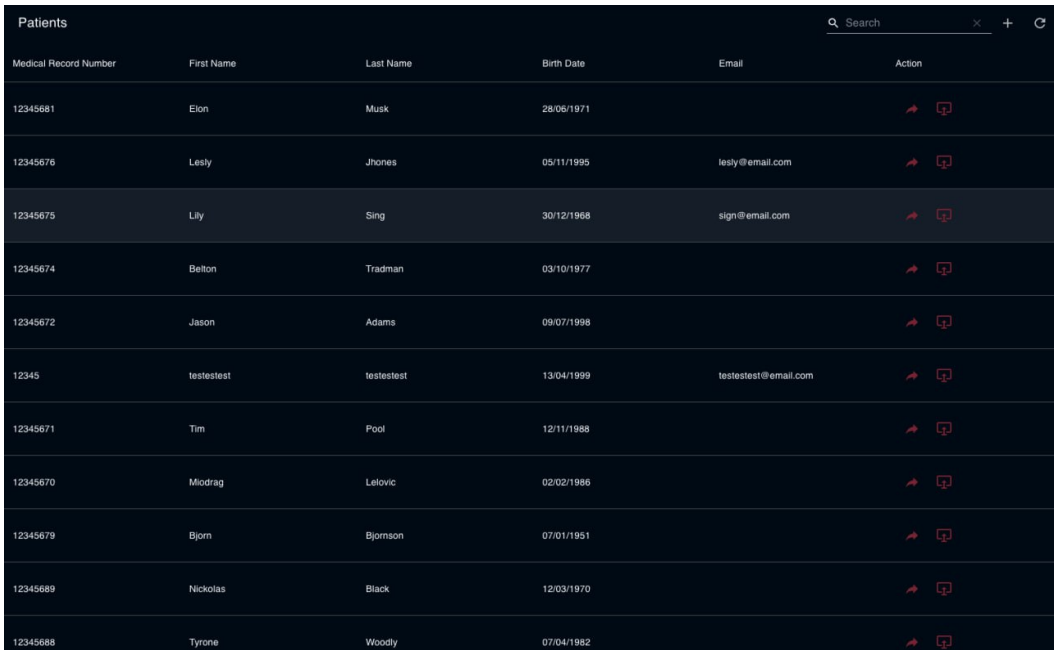


Рисунок 3.5 – Скріншот списку пацієнтів (рисунок виконано самостійно)

На рисунку 3.6 зображено скріншот, де можно побачити важливі повідомлення про стан пацієнта в застосунку «MediCareHub».

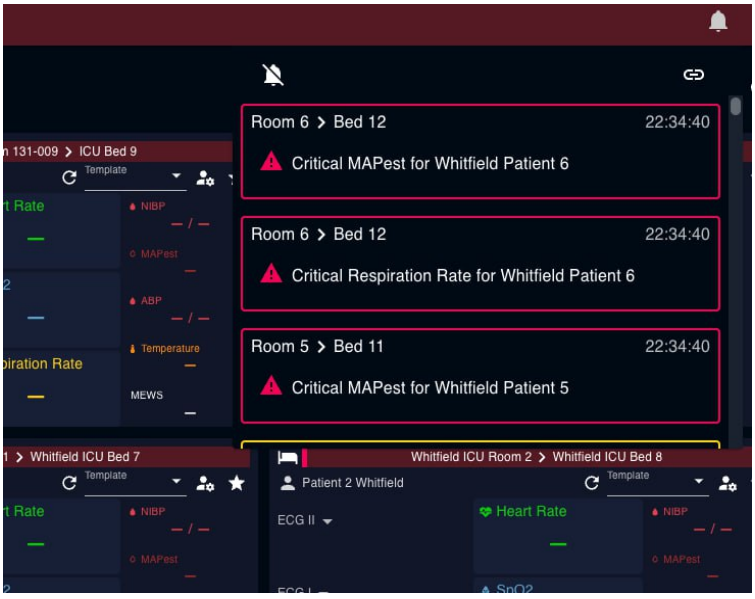


Рисунок 3.6 – Скріншот повідомлень про критичні показники (рисунок виконано самостійно)

На рисунку 3.7 зображено скріншот, на якому більш детально можно побачити графік стану пацієнта, на якому зазначено такі показники:

- NIBP
- MAPest
- Heart Rate
- Respiration Rate
- SpO2
- PVCs/min
- MEWS
- ST Elevation

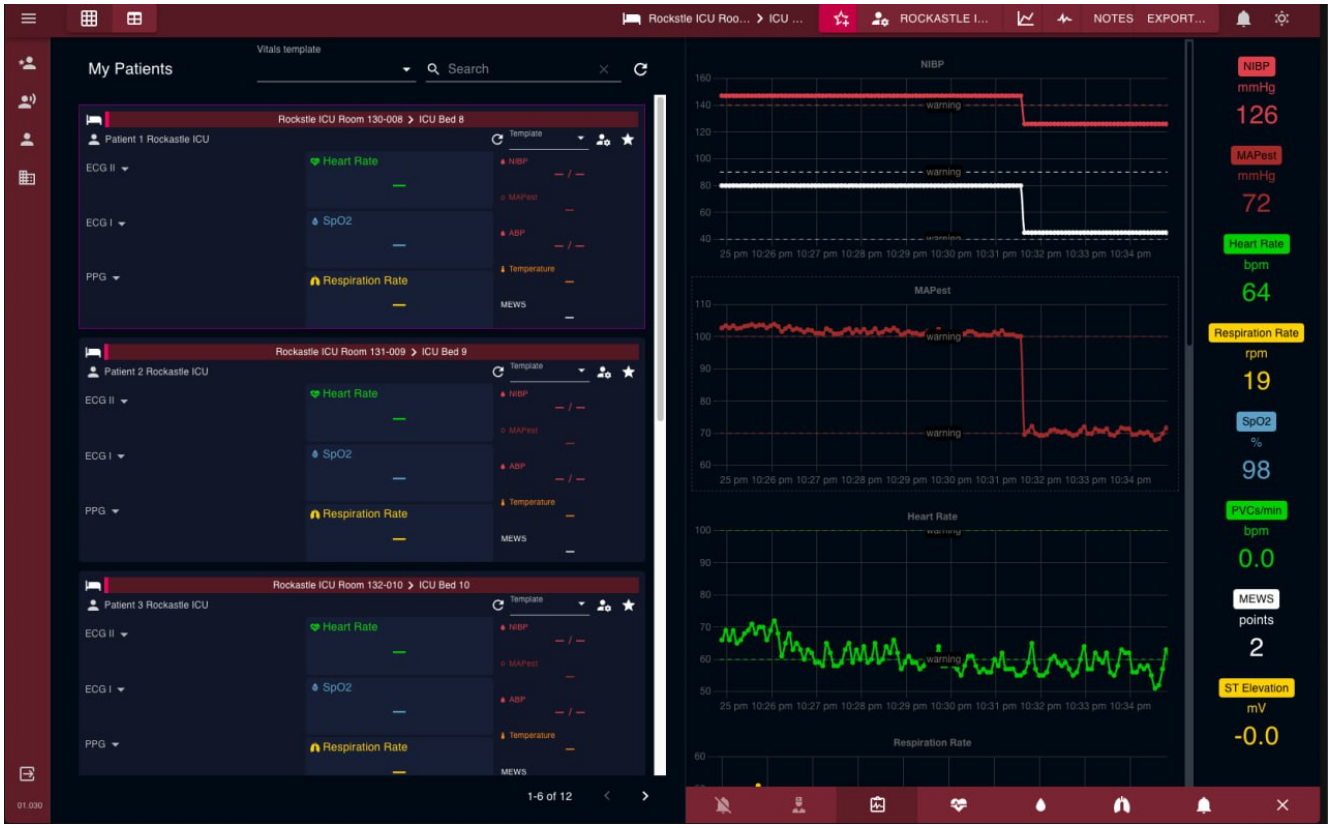


Рисунок 3.7 – Скріншот з детальними графіками всіх життєвих показників(рисунок виконано самостійно)

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Використання бібліотеки ReactTable

При розробці клієнтської частини на NextJs [4] використовуються різноманітні алгоритми та методи для вирішення складних завдань.

На рисунку 4.1 представлений код який забезпечує вивід пацієнтів в таблиці, для цього було використано бібліотеку “ReactTable” [9] яка відповідає за оптимізований рендерінг таблиць, а також добавляє безліч можливостей для кастомізації та перевикористування компонентів. Для стилізації використано Tailwind css [8].

```
const table = useReactTable({
  data: patientData,
  columns: patientColumns,
  getCoreRowModel: getCoreRowModel(),
  state: {
    columnVisibility: { id: false },
  },
});

return (
  <div className="grow space-y-4 pb-10">
    <Card
      className={cn(
        "w-screen max-w-full",
        sidebarVariant === "STRETCHED" && "lg:w-[calc(100vw-307px)]",
      )}
    >
      <div className="relative overflow-hidden">
        {patientData.length > 0 && (
          <Table className="w-full">
            <TableHeader className="sticky top-0">
              {table.getHeaderGroups().map((headerGroup) => (
                <TableRow className="border-0" key={headerGroup.id}>
                  {headerGroup.headers.map((header) => {
                    return (
                      <TableHead key={header.id}>
                        {header.isPlaceholder
                          ? null
                          : flexRender(
                              header.column.columnDef.header,
                              header.getContext(),
                            )}
                      </TableHead>
                    );
                  })}
                </TableRow>
              ))}
            </TableHeader>
```

Рисунок 4.1 – Реалізація списку пацієнтів (рисунок виконано самостійно)

В цьому коді також представлен кастомний хук useGetPatients для отримання даних пацієнтів. Хук useSidebarVariant для визначення варіанта бічного меню

(розтягнуте чи звичайне). Функція cn для умовного об'єднання класів CSS а також задля зменшення конфліктів між стилями. Функція flexRender [9] відповідає за флексовий рендерінг даних в комірках таблиці. Використання ReactTable дозволяє оптимізувати рендеринг таблиці навіть при великій кількості рядків, завдяки ефективному управлінню станом і видимістю колонок.

Також було використано Умовне завантаження даних через це таблиця рендериться лише тоді, коли є дані пацієнтів, що запобігає непотрібному рендерингу.

4.2 Використання бібліотеки RadixUI

На рисунку 4.2 представлено наступне рішення, де було використання RadixUI [10] для рендерингу діалогів а також використання TRPC [5] для виклики функцій бекенду.

```
const PatientDeviceDialog = ({ id }: { id: string }) => {
  const { mutate: connect, isPending } = useConnectPatientToDevice();
  const { data } = useGetDevices({ empty: true });
  const [selectedDevice, setSelectedDevice] = useState<string>("");
  const [isOpen, setIsOpen] = useState(false);
  return (
    <Dialog
      open={isOpen}
      onOpenChange={(open) => {
        if (isPending) return;
        else setIsOpen(open);
      }}
    >
      <DialogTrigger asChild>
        <div title="Connect to device">
          <Plus
            className={cn(
              buttonVariants({ variant: "default" }),
              "h-6 w-6 cursor-pointer p-1",
            )}
          />
        </div>
      </DialogTrigger>
      <DialogContent className="w-max">
        <DialogHeader>
          <DialogTitle>Choose a device to connect to</DialogTitle>
        </DialogHeader>
        <div>
          <Select value={selectedDevice} onValueChange={setSelectedDevice}>
            <SelectTrigger className="w-[180px]">
              <SelectValue placeholder="Select a device" />
            </SelectTrigger>
            <SelectContent>
              {data?.map((device) => (
                <SelectItem key={device.id} value={device.id}>
                  {device.name}
                </SelectItem>
              ))}
            </SelectContent>
          </Select>
        </div>
```

Рисунок 4.2 – Реалізація діалогу для підключення пацієнта (рисунок виконано самостійно)

На рисунку 4.2 можно побачити як саме відбувається виклик функції через TRPC, виклик цього метода повертає функцію mutate яка дозволяє зробити мутацію даних в базі даних. Також для роботи з TRPC була використана бібліотека ReactQuery для зберігання даних в кешу, це допомагає позбутися лишніх запитів до бази даних та ререндерінгу компонентів.

Наступне програмне рішення полягає в управлінні станом за допомогою Zustand, це можно побачити на рисунку 4.3. Ця бібліотека дозволяє ефективно керувати станом у складних модулях, де є багато взаємозалежних станів.

```
export type SidebarVariant =
  (typeof SIDEBAR_VARIANTS)[keyof typeof SIDEBAR_VARIANTS];

export interface SidebarLink {
  name: string;
  href: string;
  icon?: LucideIcon;
  disabled?: boolean;
  key: string | string[];
}

const useSidebarStore = create<{
  variant: SidebarVariant;
  actions: {
    changeSidebarVariant: () => void;
  };
}>((set) => ({
  variant: "SHORT",
  actions: {
    changeSidebarVariant: () => {
      set((state) => {
        const updatedVariant =
          state.variant === SIDEBAR_VARIANTS.STRETCHED
            ? SIDEBAR_VARIANTS.SHORT
            : SIDEBAR_VARIANTS.STRETCHED;

        return {
          variant: updatedVariant,
        };
      });
    },
  },
}));

export const useSidebarVariant = () =>
  useSidebarStore((state) => state.variant);

export const useSidebarActions = () =>
  useSidebarStore((state) => state.actions);
```

Рисунок 4.3 – Рішення для управління станом сайдбару, бокового меню (рисунок виконано самостійно)

Ці алгоритми, методи допомагають підвищити продуктивність, масштабованість та користувацький досвід при розробці застосунку на Next.js.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування всіх компонентів здійснювалося як вручну, так і за допомогою юніт-тестів, а також були проведені інтеграційні тести.

Для тестування в основному було залучено Playwright який забезпечує надійне наскрізне тестування сучасних веб-додатків. На рисунку 5.1 можно побачити як застосунок дозволяє робити кросс-платформе та кросс-браузерне тестування задля виключення випадків коли в різних браузерах чи на різних платформах користувацький інтерфейс буде відрізнятися.

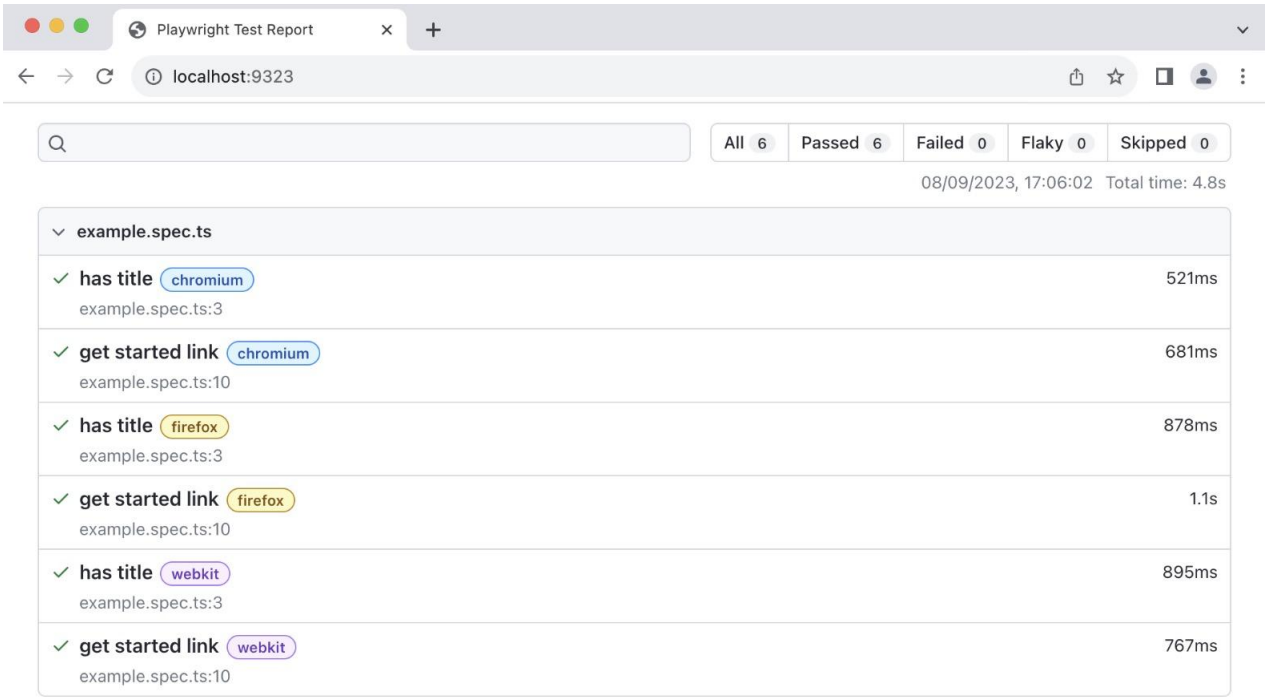


Рисунок 5.1 – Результат деяких тестів (рисунок виконано самостійно)

Для юніт-тестування та інтеграційного тестування було використано бібліотеку Jest, яка надає всі необхідні можливості для комплексного тестування. В додатку було запроваджено тестування всіх компонентів, включаючи не тільки базові, але й компоненти сторінок та модулів.

Під час тестування Next.js [4] додатку я застосував різні методи та інструменти для забезпечення якості продукту. Спочатку я розробив набір тестових кейсів, які включали перевірку функціональності, коректності відображення та взаємодії компонентів. На рисунку 5.2 представлено фрагмент тестів.

```
import React from 'react'
import { render } from '@testing-library/react' import dayjs from 'dayjs'
import '@testing-library/jest-dom' import 'jest-extended'
import { TestID } from '@resources/TestID' import i
LastSyncedNotification, LastSyncedNotificationProps,
} from '@/components/LastSyncedNotification'
describe('<LastSyncedNotification />', () => {
it('renders the Tab component', () => {
const enabledProps: LastSyncedNotificationProps = {
datetime: '', pending: false, syncing: true,
const component = render<LastSyncedNotification {...enabledProps} />)
expect (component). toBeTruthy )
子)
it ('Should display syncing ', () → f
const enabledProps: LastSyncedNotificationProps = {
datetime: "', pending: false, syncing: true,
const { getByTestId } = render<LastSyncedNotification {...enabledProps} />)
expect (getByTestId (TestID. LAST_SYNCED_NOTIFICATION_SYNCING) - innerHTML).
```

Рисунок 5.2 –Приклад юніт-тестування компонента (рисунок виконано самостійно)

Далі буде описано тест-кейси цього проекту

Тест-кейс №1 - Створення нового пацієнта

а) Послідовність роботи

1) Увійти в аккаунт Лікаря
2) Увійти на сторінку з таблицею пацієнтів
3) Нажати на кнопку “Створити пацієнта”
4) Вести усі потрібні дані про пацієнта
5) Нажати кнопку “Зберегти дані”

б) Використані дані

1) Імя = Daniel
2) Фамілія = Gotz
3) Дата народження = 23.03.1997
4) Стать = Male
5) Вага = 77 kg
6) Зріст = 185 cm
7) Зображення = daniel.jpg

8) Дата звернення = 03.05.2024

в) Що очікується в результаті

1) Форма закривається.
2) Нового пацієнта створено.
3) У лікаря з’являється кнопка закріплення цього пацієнта за собою.

г) Результат який отримали

1) Форма закрилася.
2) Новий пацієнт відобразився в списку.
3) Лікар має змогу закріпити пацієнта за собою.

д) Висновок

1) Тест-кейс №1 пройдено

Тест-кейс №2 -Перегляд графіка стану здоров'я пацієнта

а) Послідовність роботи

1) Увійти в аккаунт Лікаря
2) Увійти на сторінку з таблицею графіків
3) Нажати на графік пацієнта
4) Побачити в лайв-режимі поточний стан пацієнта
5) Побачити граничні показники кожного з датчиків

б) Використані дані

1) Пацієнт = Daniel Gotz

в) Що очікується в результаті

1) Відобразиться графік.
2) Граничні показники можно побачити.

г) Результат який отримали

1) Графік відобразився.
2) Граничні показники доступні для перегляду.

д) Висновок

1) Тест-кейс №2 пройдено

За результатами цих тест кейсів, а також за результатами PlayWright та Jest-тестів, програмна система пройшла перевірки та не викликає помилок при праці. Тести охопили основні функціональні можливості системи, включаючи взаємодію з інтерфейсом користувача, обробку запитів до сервера та коректність роботи бізнес-логіки.

Додатково були проведені навантажувальні тести для оцінки продуктивності та масштабованості системи, які підтвердили її здатність обслуговувати велику кількість одночасних користувачів без значного погіршення швидкості відгуку. Регресійне тестування показало, що жодні зміни або оновлення не вплинули на стабільність і функціональність веб-застосунку, забезпечуючи впевненість у його надійності.

ВИСНОВКИ

Під час виконання роботи в області автоматизації процесів у медичних закладах було визначено основні вимоги та сформульовано завдання для веб-орієнтованої програмної системи "MediCareHub". Розроблено аналіз користувачів та створено інтерактивний прототип, що забезпечує ефективну взаємодію між адміністраторами, медичним персоналом та пацієнтами системи. Зокрема, були визначені вимоги до дизайну, функціональності та технологій.

Для реалізації бекенду (серверної частини) системи було обрано TRPC для запитів, який забезпечує безшовну інтеграцію між клієнтом і сервером. Як базу даних було використано Supabase [6], яка відома своєю потужністю та надійністю, забезпечуючи широкі можливості для зберігання та оптимізації даних, що дозволяє ефективно управляти інформацією для потреб системи.

Для клієнтської частини було обрано Next.js, який є популярним фреймворком для розробки веб-додатків з використанням React. Використання Next.js дозволяє створювати швидкі, ефективні та масштабовані веб-додатки. Стилізація здійснюється за допомогою Tailwind CSS, що дозволяє швидко і гнучко налаштовувати зовнішній вигляд компонентів. Крім того, було використано Radix UI для створення доступних та настроюваних компонентів інтерфейсу.

Код написан на TypeScript, який транспілюється в JavaScript, забезпечуючи додаткову безпеку типів і полегшуючи розробку складних додатків.

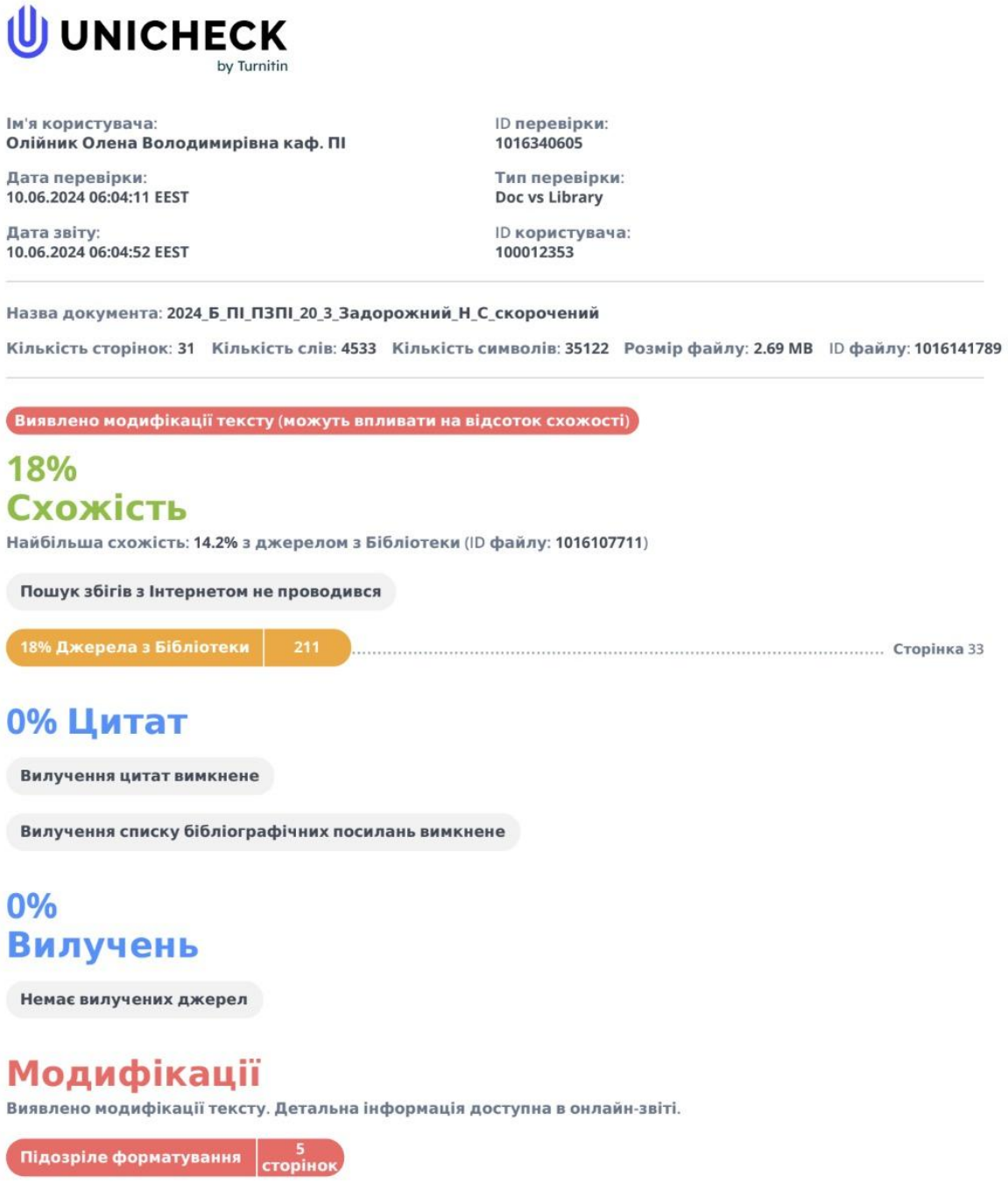
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Recharts API. [Електронний ресурс] – URL: https://recharts.org/en-US/api (дата звернення: 09.06.2024).
2. TypeScript documentation. [Електронний ресурс] – URL: https://www.typescriptlang.org/docs/ (дата звернення: 09.06.2024).
3. React | Getting Started. [Електронний ресурс] – URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 09.06.2024).
4. NextJs documentation. [Електронний ресурс] – URL: https://nextjs.org/docs (дата звернення: 09.06.2024).
5. Trpc documentation. [Електронний ресурс] – URL: https://trpc.io/docs (дата звернення: 09.06.2024)
6. Supabase documentation. [Електронний ресурс] – URL: https://supabase.com/docs (дата звернення: 09.06.2024).
7. NextAuth documentation. [Електронний ресурс] – URL: https://next-auth.js.org/getting-started/introduction (дата звернення: 09.06.2024).
8. Tailwind CSS documentation. [Електронний ресурс] - URL: https://tailwindcss.com/docs/installation (дата звернення: 09.06.2024).
9. React Table, Tanstack table - api and documentation. [Електронний ресурс] - URL: https://tanstack.com/table/latest/docs/introduction (дата звернення: 09.06.2024).
10. Radix UI. [Електронний ресурс] - URL: https://www.radix-ui.com/themes/docs/overview/getting-started (дата звернення: 09.06.2024).
11. Websockets MDN. [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 09.06.2024)
12. Лавріщева К. М. Програмна інженерія / К. М. Лавріщева. – К. : Академперіодика, 2008. – 319 с.
13. Nguyen H. Single-page application and front-end testing methods: built with React and React Router, tested with Jest and Cypress. – 2022. [Електронний

ресурс] – URL: https://www.theseus.fi/bitstream/handle/10024/745098/Nguyen_Huong.pdf (дата звернення: 09.06.2024).

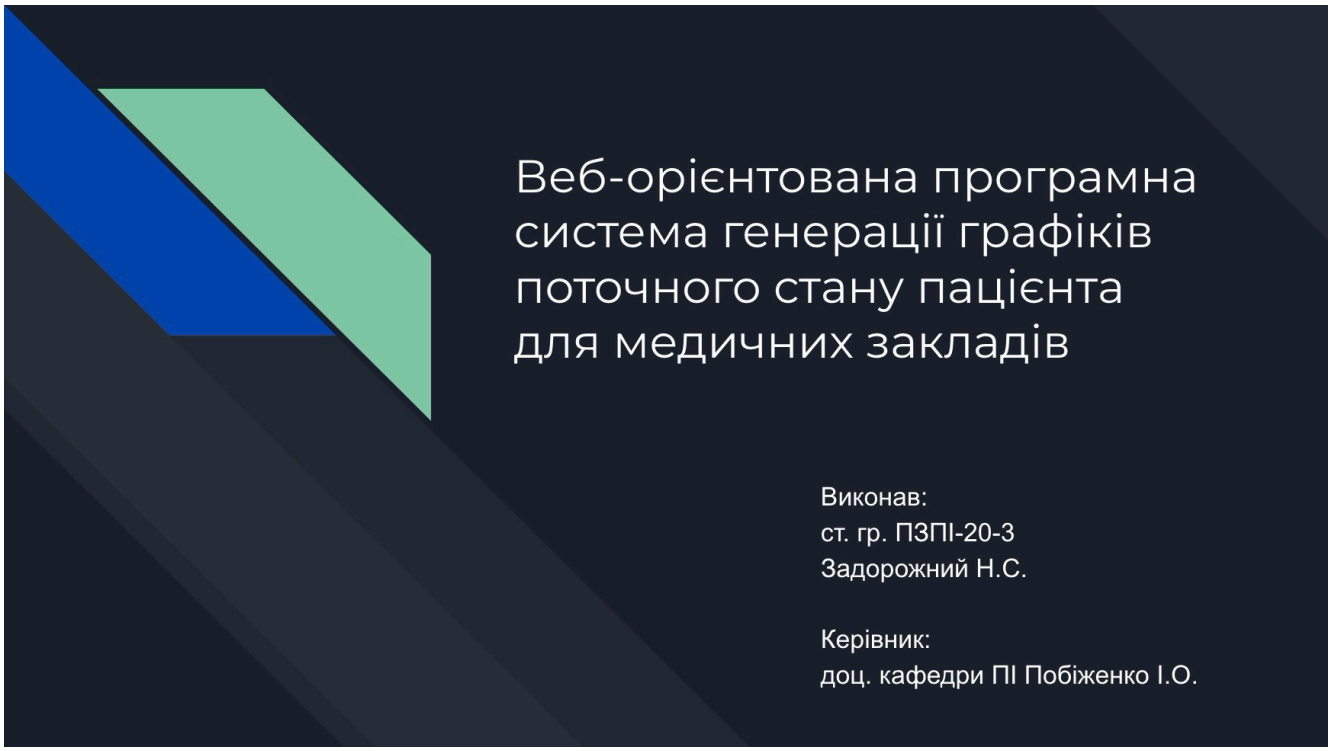
ДОДАТОК А

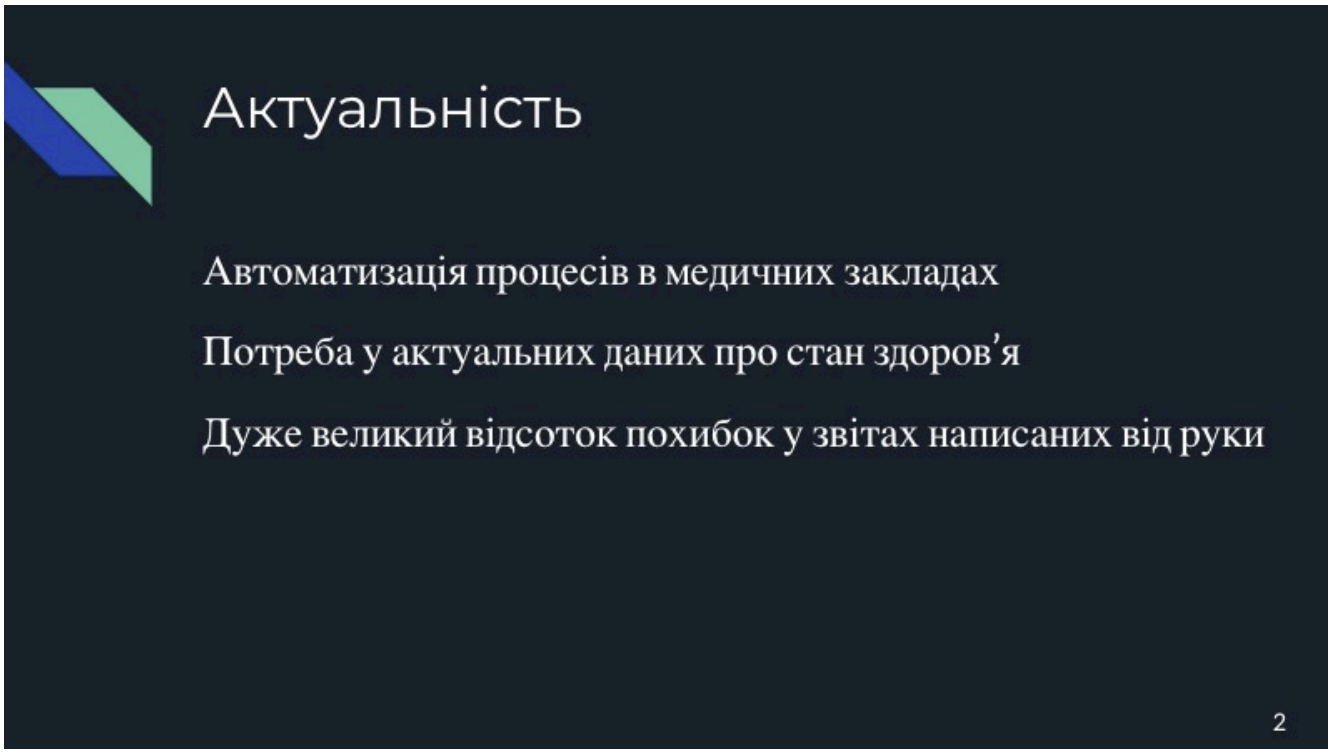
Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



ДОДАТОК Б

Слайди презентації





Об'єкт розробки – веб-орієнтована програмна система для генерації графіків поточного стану пацієнта у медичних закладах.

Мета розробки – створення веб-застосунку, який генеруватиме графіки стану здоров'я пацієнту для медичних закладів.

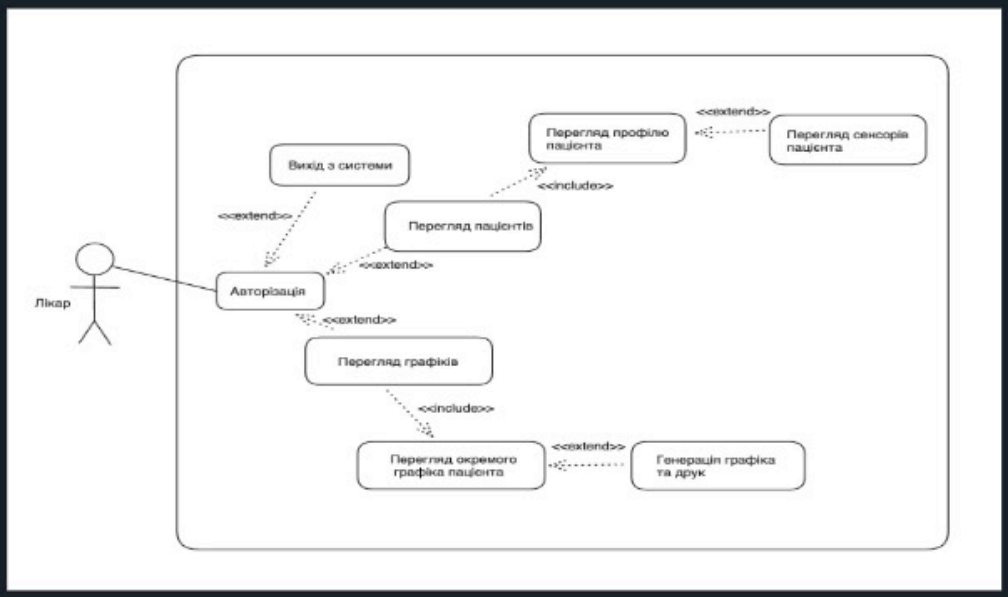
3

Постановка задачі

- Створення профілю пацієнта
- Підключення пацієнта до сенсорів
- Генерація графіків поточного стану показників
- Друк згенерованого графіка життєвого показника
- Менеджмент пацієнтами між лікарями

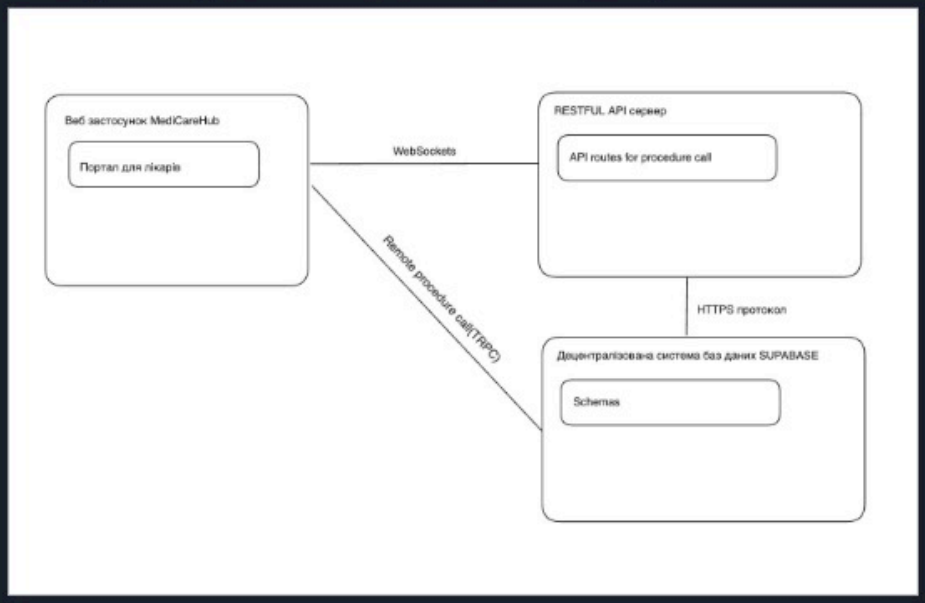
4

Use case для лікаря



5

Діаграма розгортання веб-застосунку



6

Архітектура

Архітектура веб-застосунку «MediCareHub» базується на клієнт-серверному підході, де веб-застосунок виступає як клієнт, а серверна частина обробляє бізнес-логіку і управління даними. Цей підхід дозволяє розділити відповідальності між клієнтськими пристроями та сервером.

Для прямих запитів в базу я використовую tRPC, який дозволяє уникати сервер.

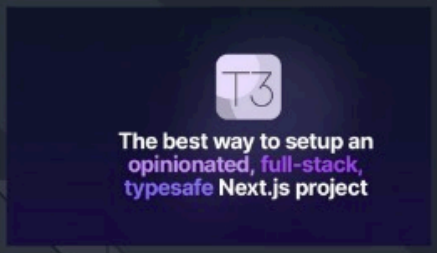
7

Архітектура

Загалом, клієнт-серверна архітектура забезпечує високу ефективність, гнучкість та масштабованість веб-застосунку «MediCareHub». Вона дозволяє централізовано керувати даними та ресурсами, знижуючи вимоги до клієнтських пристроїв і забезпечуючи високий рівень безпеки та узгодженості даних. Незважаючи на залежність від стабільності сервера, правильне планування і впровадження резервних механізмів дозволить забезпечити безперебійну роботу системи і задовольнити потреби користувачів

8

Чому саме NextJs?



1. Оптимізація для SEO
2. Гібридний статичний та серверний рендеринг
3. Побудова прогресивних веб-додатків (PWA)
4. Висока продуктивність із коробки
5. Безшовні інтеграції
6. Широка екосистема та модульність

9

Технології

10

Тестування

Було проведене:
Мануальне тестування таких тест-кейсів:

1. Створення нового пацієнта
2. Перегляд графіка стану здоров'я пацієнта

Навантажувальні тести для оцінки продуктивності та масштабованості системи, які підтвердили її здатність обслуговувати велику кількість одночасних користувачів без значного погіршення швидкості відгуку.

Такі функції як отримання даних для графіку, створення та підключення сенсорів покриті юніт тестами.

11

ВИСНОВКИ

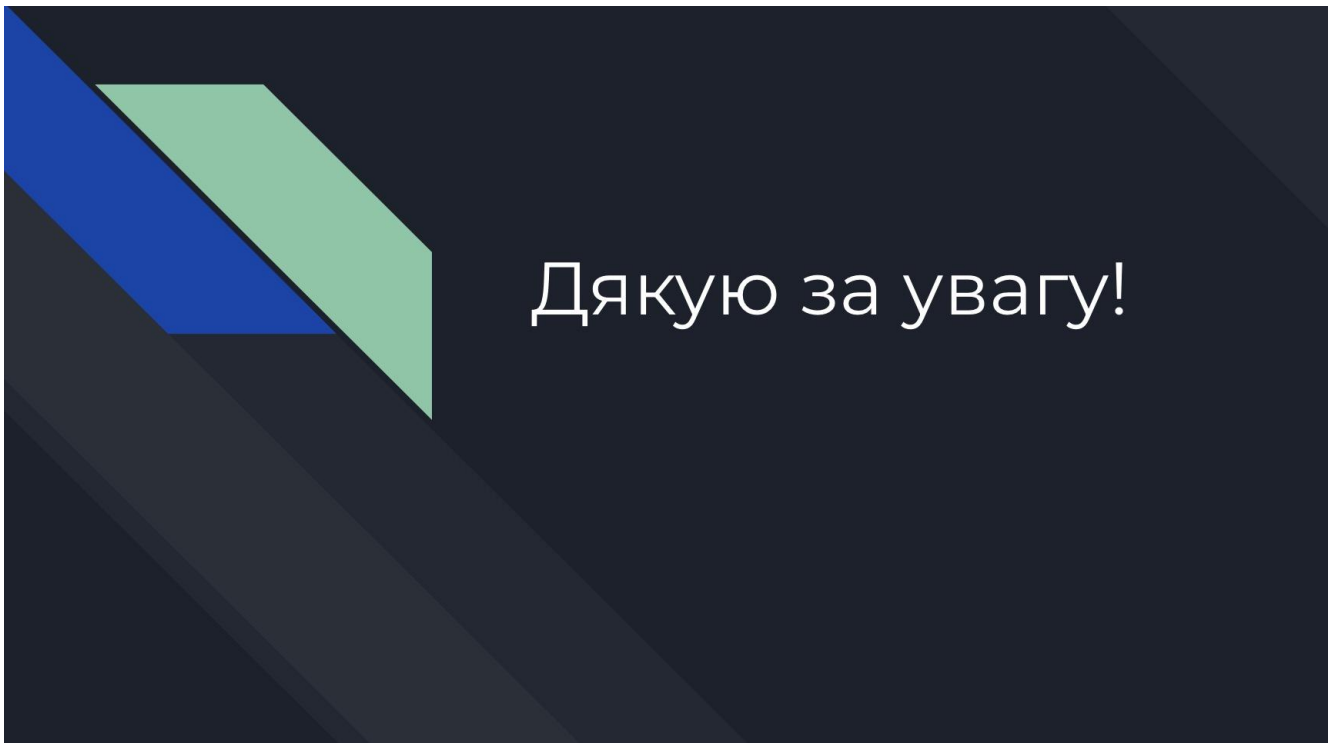
Було визначено мету теми та сформульовані завдання, що стоять перед метою

Проаналізовані сучасні підходи в розробці

Встановлено технології, що використовувались

Розроблено програмну систему та проведено тестування

12


Дякую за увагу!

ДОДАТОК В

Специфікація вимог до програмного продукту

ВСТУП

1.1 Огляд продукту

Програмна система "MediCareHub" з інтегрованим back-end рішенням представляє собою комплексний інструмент для медичних закладів, зацікавлених у моніторингу та відображенні поточного стану пацієнтів. Цей продукт включає в себе широкий спектр функцій для ефективного збирання, аналізу та візуалізації медичних даних пацієнтів.

Автентифікація та авторизація користувачів є ключовими елементами безпеки сервісу, що гарантує захист медичної інформації та контроль доступу до функцій залежно від ролі користувача. Ці можливості дозволяють медичним працівникам створювати персональні профілі, налаштовувати доступ до даних пацієнтів та отримувати персоналізовані рекомендації щодо лікування.

"MediCareHub" спрощує процес моніторингу та аналізу стану пацієнтів завдяки інтуїтивно зрозумілому інтерфейсу, де користувачі можуть переглядати детальну інформацію про поточний стан пацієнтів, включаючи показники життєдіяльності, історію хвороб, результати аналізів та діагнозів. Це забезпечує зручний доступ до важливої медичної інформації.

Система також має функцію керування медичними даними, де лікарі можуть додавати нові дані про пацієнтів, редагувати існуючі, управляти реєстраціями та відгуками від медичного персоналу. Це дає змогу ефективно організовувати та планувати лікування, а також збирати зворотний зв'язок для покращення медичного обслуговування.

Загалом, "MediCareHub" є інноваційним рішенням для медичних закладів, надаючи зручні інструменти для моніторингу, аналізу та візуалізації даних пацієнтів, а також для медичних працівників, щоб ефективно управляти та покращувати якість медичного обслуговування.

1.2 Мета

Основною метою проекту "MediCareHub" є створення універсального сервісу, який спрощує процес моніторингу, аналізу та візуалізації стану пацієнтів для медичних закладів. Цей сервіс надає можливість перегляду актуальних медичних даних у зручному графічному форматі, їх фільтрації за медичними показниками та пошуку.

Додатково, "MediCareHub" забезпечує верифікацію користувачів, можливість переходу до детальної інформації про пацієнта, виставлення оцінок ефективності медичних процедур.

Таким чином, мета проекту "MediCareHub" полягає у забезпеченні зручного, інтуїтивно зрозумілого інструменту для моніторингу та аналізу стану пацієнтів, сприяння підвищенню якості медичного обслуговування та ефективності лікування через використання сучасних технологій у медичній практиці.

1.3 Межі

У контексті проекту "MediCareHub", межі визначаються функціональними та технічними аспектами системи. Проект зосереджений на наданні медичним закладам інструментів для моніторингу, аналізу та візуалізації стану пацієнтів. Важливими аспектами є верифікація користувачів, надання доступу до медичних даних та оцінка ефективності лікування.

Технічні обмеження системи включають обмеження щодо ресурсів сервера, які можуть вплинути на її масштабованість та продуктивність при великій кількості одночасних запитів. Це також стосується обмежень на кількість одночасних користувачів та медичних даних, які можуть бути оброблені системою без зниження її ефективності.

З точки зору безпеки, проект має забезпечити захист даних пацієнтів, включаючи аутентифікацію, авторизацію, шифрування даних та захист від несанкціонованого доступу. Це вимагає використання сучасних протоколів

безпеки та методів шифрування, а також постійного моніторингу та оновлення системи безпеки.

Проект "MediCareHub" не займається безпосередньою діагностикою чи лікуванням пацієнтів, а лише надає платформу для моніторингу та аналізу медичних даних. Тому, відповідальність за точність медичної інформації, її застосування та якість лікування лежить на медичних працівниках та закладах.

1.4 Посилання

Проект "MediCareHub" прагне автоматизувати та оптимізувати процеси моніторингу та аналізу стану пацієнтів, мінімізуючи необхідність ручного відстеження медичних даних з різних джерел. Інтегровані функції верифікації користувачів, управління медичними записами, оцінки ефективності лікування, а також візуалізації даних на графіках надають медичним працівникам зручний і інтуїтивно зрозумілий доступ до необхідної інформації. Це значно спрощує процес взаємодії з медичними даними та покращує загальний досвід користувача.

"MediCareHub" також забезпечує медичним закладам ефективний інструмент для покращення якості медичного обслуговування та оптимізації внутрішніх процесів. Сервіс дозволяє швидко оновлювати інформацію про стан пацієнтів, управляти записами. Це створює можливості для покращення організації медичного обслуговування та підвищення його якості.

Таким чином, "MediCareHub" пропонує цілісний підхід до взаємодії з медичними даними, сприяючи покращенню якості лікування пацієнтів та ефективному управлінню медичними процесами, що покращує досвід як медичних працівників, так і пацієнтів.

1.5 Означення та абревіатури

Backend – це серверна частина програмного забезпечення, що відповідає за обробку даних, виконання бізнес-логіки та забезпечення безпеки. Вона керує

взаємодією з базою даних, виконує процеси автентифікації та авторизації користувачів і підтримує основні операції, необхідні для функціонування системи.

Frontend - це клієнтська частина програмного забезпечення, яка відповідає за інтерфейс і взаємодію з користувачем. Вона включає дизайн, розмітку та скрипти сторінок, які користувачі бачать та з якими взаємодіють через браузер або мобільні додатки.

API (Application Programming Interface) – це інтерфейс програмування застосунків, який дозволяє різним програмним компонентам взаємодіяти між собою. API визначає правила та методи, за допомогою яких можна отримувати доступ до функціоналу або даних однієї програми з іншої.

ORM (Object-Relational Mapping) – це технологія, що забезпечує мапінг реляційних баз даних на об'єктно-орієнтовану модель програми. Завдяки ORM, робота з базою даних стає більш інтуїтивно зрозумілою для розробників, оскільки вони можуть оперувати об'єктами замість використання SQL-запитів.

HTTP (HyperText Transfer Protocol) – це основний протокол передачі даних у Всесвітній павутині, який використовується для завантаження веб-сторінок з сервера на клієнтський браузер. HTTP визначає, як повинні бути сформовані та передані запити та відповіді між клієнтом і сервером.

CRUD (Create, Read, Update, Delete) – це концепція, яка описує чотири основні операції, що використовуються при роботі з даними: створення нових записів, читання (вибірка) існуючих даних, оновлення (модифікація) даних і видалення записів. Ці операції є фундаментальними для будь-яких систем, що працюють з базами даних.

2. ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Перспективи програмної системи “MediCareHub” щодо надання інструментів для моніторингу та аналізу стану пацієнтів є обнадійливими і відкривають багато можливостей для подальшого розвитку та розширення.

Зважаючи на зростаючу потребу у сучасних та ефективних методах медичного обслуговування, а також на необхідність персоналізованого підходу до лікування пацієнтів, "MediCareHub" може знайти своє місце в різних медичних закладах та сприяти розвитку сфери охорони здоров'я.

Перспективи продукту включають розширення функціональності та додавання нових модулів, щоб задовольнити специфічні потреби лікарень та медичних працівників. Наприклад, можливості планування лікування та розподілу медичних ресурсів, система відгуків про ефективність лікування, інтеграція з іншими медичними платформами або соціальними мережами, розширена аналітика та звітність.

Крім того, вдосконалення аналітичних інструментів для оцінки успішності різних методів лікування та медичних процедур. Розробка розширених звітів для медичних закладів та партнерів для підвищення ефективності медичного обслуговування. Розширення географічного охоплення для підтримки медичних закладів у різних регіонах та країнах. Активне залучення до міжнародних медичних конференцій та симпозіумів для розширення аудиторії.

Загалом, "MediCareHub" має великий потенціал стати ключовим гравцем у сфері медичного моніторингу та аналізу. Розширення функціональності, посилення географічного охоплення та впровадження новітніх технологій можуть вивести продукт на новий рівень та забезпечити йому стабільне місце на ринку медичних технологій.

2.2 Функції продукту

Основні функції цього продукту включають:

- US-1: Як адміністратор, я хочу переглядати список користувачів до яких маю доступ, щоб переглядати і редагувати їх інформацію.
- US-2: Як лікар, я хочу переглядати інформацію про стан здоров’я пацієнтів, яких я лікую.
- US-3: Як лікар, я хочу отримувати сповіщення про критичний стан пацієнта.
- US-4: Як лікар, я хочу переглядати графік важливих для життя показників.
- US-5: Як лікар, я хочу мати можливість реєструвати пацієнта в системі.
- US-6: Як лікар, я хочу мати можливість друку графіків.

2.3 Характеристика користувачів

- Характеристики користувачів програмної системи “MediCareHub” – для моніторингу та аналізу стану пацієнтів можуть бути різноманітними, оскільки система взаємодіє з різними типами користувачів. Основні характеристики користувачів включають:
- Пацієнти – це люди різного віку та статі, які є основними користувачами системи. Їх характеристики можуть включати їхні медичні історії, поточні стани здоров'я, вік, локацію проживання та індивідуальні медичні потреби.
- Лікарі – медичні працівники, які використовують систему для моніторингу стану пацієнтів, аналізу медичних даних та прийняття рішень щодо лікування. Вони мають доступ до детальної інформації про пацієнтів, можуть створювати та редагувати медичні записи, аналізувати звіти та надавати рекомендації.

- Адміністратори медичних закладів – відповідають за керування користувачами та лікарями в системі. Вони мають розширені права доступу, можуть додавати нових користувачів, редагувати права доступу, аналізувати звітність та здійснювати інші адміністративні функції для забезпечення ефективної роботи системи.

Ці користувачі разом забезпечують ефективне функціонування системи “MediCareHub” і сприяють покращенню медичного обслуговування та моніторингу стану пацієнтів.

2.4 Загальні обмеження

У програмній системі “MediCareHub” – для моніторингу та аналізу стану пацієнтів існують загальні обмеження, які можуть впливати на її функціонування та використання. Деякі з цих обмежень включають:

Безпека даних:

Система повинна дотримуватися вимог безпеки, щоб захистити медичні дані пацієнтів та запобігти несанкціонованому доступу. Обмеження безпеки можуть включати криптографічні вимоги, захист від атак, правила доступу до даних та контроль прав користувачів.

Застосування сучасних протоколів шифрування для забезпечення конфіденційності та цілісності даних під час передачі та зберігання.

Регулярний моніторинг та оновлення системи безпеки для виявлення та усунення потенційних загроз.

Масштабованість:

Система повинна бути здатна масштабуватися, щоб впоратися із зростаючим обсягом користувачів, медичних записів та аналітичних завдань. Обмеження масштабованості можуть впливати на продуктивність та швидкодію системи при збільшенні навантаження.

Використання хмарних технологій для динамічного масштабування ресурсів відповідно до навантаження.

Оптимізація бази даних та інфраструктури для підтримки високої продуктивності навіть при великій кількості одночасних запитів.

Сумісність та інтеграція:

Система повинна бути сумісною з різними медичними пристроями та платформами для забезпечення безперервного потоку даних. Обмеження можуть виникати через різні стандарти та протоколи, які використовуються у медичній сфері.

Забезпечення можливості інтеграції з іншими медичними системами та платформами для обміну даними та співпраці між різними медичними установами.

Юридичні та регуляторні вимоги:

Система повинна відповідати юридичним та регуляторним вимогам щодо обробки та зберігання медичних даних, таким як HIPAA (Health Insurance Portability and Accountability Act) або GDPR (General Data Protection Regulation).

Дотримання місцевих та міжнародних законодавчих норм для забезпечення правомірності та етичності використання системи.

Користувацький досвід:

Система повинна забезпечувати зручний та інтуїтивно зрозумілий інтерфейс для користувачів різного рівня технічної підготовки. Обмеження можуть виникати через необхідність уніфікації інтерфейсу для різних типів пристроїв та користувачів.

Забезпечення доступності системи для користувачів з обмеженими можливостями, відповідно до стандартів доступності.

Ці обмеження є важливими аспектами, які необхідно враховувати при розробці та впровадженні програмної системи “MediCareHub”, щоб забезпечити її надійне та ефективне функціонування.

2.5 Припущення й залежності

Для забезпечення оптимальної роботи системи "MediCareHub" важливо мати стабільне та надійне Інтернет-з'єднання, оскільки система взаємодіє з мережею для обміну даними, автентифікації, надсилання сповіщень тощо.

Наявність постійного з'єднання з Інтернетом є необхідною умовою для ефективної роботи всіх компонентів системи.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Інтерфейс користувача програмної системи генерації графіків поточного стану пацієнта для медичних закладів розроблений для забезпечення зручності, ефективності та простоти використання для користувачів. Він пропонує легке у використанні та привабливе середовище, яке дозволяє взаємодіяти з системою та виконувати необхідні завдання.

Інтерфейс користувача може мати форму веб-додатку або мобільного додатка в залежності від потреб користувачів і доступних платформ. Він має чітку структуру і навігаційні елементи, що спрощують орієнтування користувачів та виконання різноманітних дій.

Основні характеристики інтерфейсу користувача включають:

- Інтуїтивність: Забезпечення простоти розуміння та використання для користувачів без потреби у додатковому навчанні.
- Ефективність: Максимізація продуктивності користувачів і зменшення часу, необхідного для виконання завдань.
- Привабливість: Створення привабливого та естетичного вигляду інтерфейсу для залучення користувачів.
- Чітка структура: Організація інформації та функціоналу з метою легкого доступу і логічного розташування елементів.
- Навігаційні можливості: Наявність зрозумілих та легкодоступних елементів навігації для швидкого переміщення користувачів по системі.

Ці характеристики спрямовані на створення зручного та ефективного середовища для користувачів, що дозволяє легко взаємодіяти з програмною системою і виконувати потрібні завдання без зайвих ускладнень.

3.1.2 Апаратний інтерфейс

Апаратний інтерфейс для веб-орієнтованої програмної системи генерації графіків поточного стану пацієнта для медичних закладів включає необхідні апаратні компоненти та засоби, які забезпечують функціональність системи. Оскільки ця програма працює в онлайн-середовищі і доступна через веб-браузер або мобільний додаток, система потребує наявності серверів для зберігання та обробки даних. Це можуть бути фізичні сервери, віртуальні сервери або хмарні платформи, які забезпечують необхідні обчислювальні та ресурси для зберігання роботи системи.

3.1.3 Програмний інтерфейс

Програмний інтерфейс (API) веб-орієнтованої програмної системи генерації графіків поточного стану пацієнта для медичних закладів визначає набір правил та протоколів, які дозволяють взаємодіяти різним компонентам програми між собою. Цей інтерфейс надає зовнішнім системам або розробникам можливість використовувати функціональність системи та отримувати доступ до даних.

Програмний інтерфейс може бути реалізований у вигляді RESTful API, що базується на використанні HTTP-протоколу для передачі даних між клієнтами та сервером. Це забезпечує стандартизовану та просту взаємодію з системою шляхом використання HTTP-запитів (GET, POST, PUT, DELETE) та обміну даними у форматі JSON або XML.

3.1.4 Комунікаційний протокол

Комунікаційний протокол в програмній системі генерації графіків поточного стану пацієнта для медичних закладів визначає правила та формати обміну даними між різними компонентами системи. Він забезпечує швидку, надійну та безпомилкову передачу інформації між цими компонентами.

Один із найпоширеніших комунікаційних протоколів, який може бути використаний у такій системі, - це HTTP (Hypertext Transfer Protocol). HTTP використовується для передачі даних через мережу Інтернет і заснований на клієнт-серверній архітектурі.

HTTP визначає формат запитів та відповідей між клієнтом і сервером. Клієнт (наприклад, веб-браузер або мобільний додаток) робить запит до сервера, а сервер обробляє цей запит та повертає відповідь з необхідними даними.

3.1.5 Обмеження пам’яті

Обсяг оперативної пам'яті, доступний для виконання програм та зберігання даних, може бути обмеженим. Залежно від розміру доступної пам'яті, система може вміщувати обмежену кількість користувачів та обробляти лише обмежену кількість даних.

Наявність дискового простору для зберігання даних також може бути обмеженою. Дисковий простір використовується для зберігання файлів, баз даних, резервних копій та інших ресурсів. Обмеження дискового простору може впливати на масштабність системи та можливість зберігання великих обсягів даних.

3.1.6 Операції

Користувачі можуть шукати інформацію про графіки поточного стану пацієнтів, що їх цікавить. Це включає визначення основних показників, стану, лікування та інших важливих атрибутів медичного процесу.

Система надає можливості для створення медичним закладам або медичним групам, що мають права на створення нових графіків, а також доступ до існуючих записів про стан пацієнтів. Крім того, система надає можливість генерувати звіти для медичних працівників та адміністраторів.

3.2 Атрибути програмного продукту

3.2.1 Надійність

В медичній системі генерації графіків поточного стану пацієнтів надійність є ключовою характеристикою, оскільки вона забезпечує безперебійну та стабільну роботу системи. Надійність означає, що система працює відповідно до очікувань і надійно обробляє дані пацієнтів, запобігаючи можливим збоям або відмовам.

Для забезпечення надійності система повинна мати вбудований механізм обробки помилок та винятків, що можуть виникати під час роботи. Це допомагає вчасно виявляти та вирішувати проблеми, що можуть впливати на роботу системи, забезпечуючи її стабільну та надійну функціональність.

Також критично мати механізми резервного копіювання, які забезпечують збереження та захист медичних даних системи. Це може включати регулярне створення резервних копій даних, а також баз даних, що забезпечує можливість відновлення системи в разі втрати чи пошкодження важливої інформації.

3.2.2 Доступність

Система медичного моніторингу пацієнтів має мати адаптивний дизайн, який автоматично пристосовується до різних типів пристроїв, таких як комп'ютери, планшети або смартфони. Це дозволяє користувачам зручно взаємодіяти з системою на будь-якому пристрої, незалежно від розміру їхнього екрану. Такий підхід сприяє поліпшенню зручності використання та задоволенню користувачів, забезпечуючи оптимальний інтерфейс на різних пристроях.

3.2.3 Безпека

Система для медичного моніторингу пацієнтів повинна мати механізми для перевірки ідентифікації користувачів та авторизації їх доступу до різних функцій системи. Це може включати використання паролів, налаштування рівнів доступу та управління правами користувачів.

Для забезпечення конфіденційності даних, особливо під час їх передачі між клієнтом і сервером, використовується шифрування. Застосування шифрування, такого як SSL (Secure Sockets Layer) або TLS (Transport Layer Security), гарантує безпеку під час обміну даними.

Також система повинна мати заходи безпеки для захисту від зловмисницьких атак, таких як введення шкідливого коду, SQL-ін'єкції, переповнення буфера і т. д. Це може включати використання механізмів фільтрації введення користувачів,

валідацію даних та захист від переповнення, що забезпечує високий рівень безпеки системи.

3.2.4 Супроводжуваність

Система повинна бути документована належним чином, що чітко описує її архітектуру, конфігурацію, процедури встановлення, налаштування та підтримки. Це сприяє швидкому розумінню та вирішенню проблем розробниками і адміністраторами системи, а також дозволяє вносити необхідні зміни.

Крім того, система має мати механізми для регулярних оновлень та встановлення патчів для виправлення відомих помилок, усунення вразливостей та покращення функціональності. Це забезпечує надійність та стійкість системи, а також дозволяє впроваджувати нові функції та удосконалення.

3.2.5 Переносимість

Веб-орієнтована система для генерації графіків поточного стану пацієнтів у медичних закладах має бути переносимою, що означає її здатність працювати на різних платформах і середовищах без значних модифікацій. Це досягається за допомогою використання платформо-незалежних технологій, стандартизованих фреймворків та уникання глибокої залежності від конкретних платформ або систем управління базами даних (СКБД). Такий підхід забезпечує гнучкість та ефективність у впровадженні системи на різних середовищах без значних зусиль.

3.2.6 Продуктивність

Система для генерації графіків поточного стану пацієнтів у медичних закладах повинна демонструвати високу продуктивність. Це включає оптимізацію коду, здатність до масштабування та ефективне використання ресурсів. Розробка оптимізованого коду для виконання завдань, разом з масштабуванням системи для ефективного розподілу навантаження та оптимального використання ресурсів, сприяє досягненню високої продуктивності системи та задоволенню потреб користувачів.

3.2.7 Вимоги бази даних

Для системи генерації графіків поточного стану пацієнтів у медичних закладах вимоги до бази даних включають створення належної структури, нормалізацію даних та підтримку необхідних запитів і операцій. Забезпечення безпеки даних є пріоритетом і вимагає використання механізмів авторизації, шифрування та контролю доступу. Оптимізація бази даних для досягнення високої швидкодії та ефективності виконання запитів також є важливою вимогою.

ДОДАТОК Г

Код для створення WebSocket та підключення його

```
1  const { createServer } = require("http");
2  const WebSocket = require("ws");
3
4  const app = express();
5  const server = createServer(app);
6
7  const webSocketServer = new WebSocket.Server({ server });
8
9  webSocketServer.on("connection", (ws) => {
10   ws.on("message", (message) => {
11     const msg = JSON.parse(message);
12
13     if (msg.type === "high") {
14       setInterval(() => {
15         ws.send(JSON.stringify({ type: "high", objects:
   highfreq }));
16       }, 1000);
17     }
18
19     if (msg.type === "low") {
20       msg.devices.forEach((device) => {
21         setInterval(() => {
22           ws.send(
23             JSON.stringify({
24               type: "low",
25               objects: {
26                 [new Date().toISOString()]: {
27                   [device]: deviceData,
28                 },
29               },
30             }),
31           );
32         }, 4000);
33       });
34     }
35   });
36
37 ws.on("error", (e) => ws.send(e));
38 });
39
40 server.listen(7654, () => console.log("started at " + 7654));
```

Код для створення TRPC провайдера

```
41 "use client";
42
43 import { QueryClient, QueryClientProvider } from
   "@tanstack/react-query";
44 import { loggerLink, unstable_httpBatchStreamLink } from
   "@trpc/client";
45 import { createTRPCReact } from "@trpc/react-query";
46 import { type inferRouterInputs, type inferRouterOutputs } from
   "@trpc/server";
47 import { useState } from "react";
48 import SuperJSON from "superjson";
49
50 import { type AppRouter } from "@/server/api/root";
51
52 const createQueryClient = () => new QueryClient();
53
54 let clientQueryClientSingleton: QueryClient | undefined =
   undefined;
55 const getQueryClient = () => {
56   if (typeof window === "undefined") {
57     // Server: always make a new query client
58     return createQueryClient();
59   }
60   // Browser: use singleton pattern to keep the same query
   client
61   return (clientQueryClientSingleton ??= createQueryClient());
62 };
63
64 export const api = createTRPCReact<AppRouter>();
65
66 /**
67  * Inference helper for inputs.
68  *
69  * @example type HelloInput = RouterInputs['example']['hello']
70  */
71 export type RouterInputs = inferRouterInputs<AppRouter>;
72
73 /**
74  * Inference helper for outputs.
75  *
76  * @example type HelloOutput =
   RouterOutputs['example']['hello']
77  */
78 export type RouterOutputs = inferRouterOutputs<AppRouter>;
79
80 export function TRPCReactProvider(props: { children:
   React.ReactNode }) {
81   const queryClient = getQueryClient();
82
83   const [trpcClient] = useState(() =>
84     api.createClient({
85       links: [
86         loggerLink({
```

```
87          enabled: (op) =>
88            process.env.NODE_ENV === "development" ||
89            (op.direction === "down" && op.result instanceof
  Error),
90        }),
91        unstable_httpBatchStreamLink({
92          transformer: SuperJSON,
93          url: getBaseUrl() + "/api/trpc",
94          headers: () => {
95            const headers = new Headers();
96            headers.set("x-trpc-source", "nextjs-react");
97            return headers;
98          },
99        }),
100          ],
101        })
102      );
103
104      return (
105        <QueryClientProvider client={queryClient}>
106          <api.Provider client={trpcClient}
  queryClient={queryClient}>
107            {props.children}
108          </api.Provider>
109        </QueryClientProvider>
110      );
111    }
112
113    function getBaseUrl() {
114      if (typeof window !== "undefined") return
  window.location.origin;
115      if (process.env.VERCEL_URL) return
  `https://${process.env.VERCEL_URL}`;
116      return `http://localhost:${process.env.PORT ?? 3000}`;
```