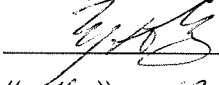


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ У МЕДИЦИНІ ТА
ТЕЛЕКОМУНІКАЦІЯХ

УДК 621.37

«Допущено до захисту»

Завідувач кафедри
комп'ютерних технологій у
медицині та телекомунікаціях
к.т.н., доц. Чухов В.В.

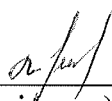

« 16 » 12 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему Дослідження можливості побудови медичної інформаційної системи
системного рівня

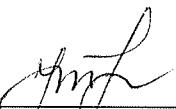
**за спеціальністю 163 «Біомедична інженерія»
освітній ступінь «магістр»**

Виконавець роботи
студент 2 курсу, гр.БІМ-22-1

19.12.2023 
(дата, підпис)

Луганський А.Г.

Керівник роботи
к.т.н., доц.

19.12.2023 
(дата, підпис)

Нікітчук Т.М.

Житомир
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ У МЕДИЦИНІ ТА
ТЕЛЕКОМУНІКАЦІЯХ

**Пояснювальна записка
до кваліфікаційної магістерської роботи**

на тему Дослідження можливості побудови медичної інформаційної
системи системного рівня

Виконав: студент 2 курсу,
групи БІМ-22-1 спеціальності
163 «Біомедична інженерія»

Луганський А.Г.

Керівник Нікітчук Т.М.

Рецензент Морозов Д.С.

«Затверджено»

Завідувач кафедри комп'ютерних
технологій у медицині та
телекомунікаціях

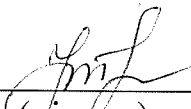
 Владислав ЧУХОВ
«27» 10 2022 р.

**Завдання
на кваліфікаційну магістерську роботу**

- Мурановський Андрій Григорович
(прізвище, ім'я, по батькові)
- Тема роботи Дослідження мотивації побудови
меритної інформаційної системи есесіонного рівня
 - Мета дослідження створення меритної інформаційної
системи для допомоги сучасним технологіям
 - Вихідні дані (характеристики об'єкта, умов дослідження та ін.)
меритна система для амбулаторій, відреструктур,
інтернет-екран
 - Основні задачі дослідження - дослідити тенденції сесіонної
побудови програмної системи; - спроектувати мережу
для лікування та моніторингу
 - Термін подання роботи до захисту липень 2023
 - Дата видачі завдання «27» 10 2022 р.

Науковий керівник

Завдання прийняв
до виконання


(підпис)

(підпис)

Глікітчук О.М.
(прізвище, ініціали)
Мурановський А.Г.
(прізвище, ініціали)

АНОТАЦІЯ

Розглянуто особливості сучасних прикладних медичних інформаційних систем та можливості медичних інформаційних систем як основного засобу зберігання медичних даних. Досліджено теоретичні основи створення медичної інформаційної системи за допомогою сучасних технологій, включаючи реєстрацію та передачу медичної інформації.

ANNOTATION

Features of modern applied medical information systems and possibilities of medical information systems as the main means of storing medical data are considered. The theoretical foundations of the creation of a medical information system using modern technologies, including the registration and transfer of medical information, have been studied.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1 МЕДИЧНІ ІНФОРМАЦІЙНІ СИСТЕМИ: ОГЛЯД МОЖЛИВОСТЕЙ І ПРИКЛАДИ ВИКОРИСТАННЯ.....	9
1.1 Аналіз розв’язуваної задачі й огляд наявних результатів.....	9
1.2 Класифікація МІС.....	11
1.3 Медичні інформаційні системи та компанії-розробники на ринку України.....	18
1.4 Висновки та основні задачі дослідження.....	26
РОЗДІЛ 2 ПРОЕКТНІ І ТЕХНІЧНІ РІШЕННЯ.....	28
2.1 Розробка МІС. Стандартний і кастомний софт.....	28
2.2 Види інформаційного та технічного забезпечення.....	31
2.3 Аналіз архітектури програмного забезпечення МІС.....	36
2.4 Архітектурні стилі та моделі.....	40
2.5 Деталі реалізації	47
Висновки до розділу 2.....	51
РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ОПИС РОБОТИ СИСТЕМИ.....	52
3.1 СУБД MongoDB та Redis	52
3.2 Порівняння реляційних і нереляційних баз даних.....	53
3.3 Створення інтерфейсу користувача.....	56
3.4 Оцінювання стану пацієнта на прикладі серцево-судинної системи.....	62
3.5 Елементи системи моніторингу та організації мережі передачі даних.....	63
3.6 Архітектурно-інформаційна модель обробки медичної інформації.....	64
Висновки до розділу 3.....	68
ВИСНОВКИ.....	69
ДОДАТОК А.....	72

ВСТУП

Розвиток сучасного суспільства відбувається під впливом глобалізації інформаційних процесів. Провідним проявом цих процесів є формування нового інформаційного простору, пов'язаного з появою мережі Інтернет. Формується нова культура, у якій ключовими словами є інформація, інформатизація, Інтернет і віртуалізація.

Поява нового інформаційного простору посприяла інтеграції інформаційних технологій у більшість соціальних, економічних і бізнес процесів. Загалом одним з основних проявів такої інтеграції є виникнення прикладних інформаційних систем, які покликані спростити життєдіяльність людей.

В сучасній державі прикладні інформаційні системи створені майже для всіх сфер суспільного життя. Охорона здоров'я є однією з найважливіших галузей діяльності держави. В сучасному інформаційному просторі охорона здоров'я є однією з найпоширеніших сфер впровадження провідних інформаційних технологій. В епоху розвинених інформаційних технологій державі необхідно забезпечити лікарів зручним і швидким засобом створення, зберігання і передачі медичної інформації. Для цього було створено медичні інформаційні системи.

Швидка поява на ринку все менш і менш нав'язливих, мініатюрних (медичних) носимих сенсорів (біосенсори та датчики тіла), зробила можливим дистанційне спостереження за пацієнтами та відкрила двері для безлічі нових бізнес моделей у галузі охорони здоров'я. Основною функцією датчиків тіла є контроль за життєвими показниками пацієнтів; потім ці дані обробляються та аналізуються програмною системою, яка візуалізує інформацію для лікарів, щоб вони могли надати віддалену допомогу. Найбільш поширеними біосигналами, що підлягають моніторингу, є вібрації, пульс, сатурація киснем крові, звуки, гази, рух або відсутність руху, температура тіла, кров'яний тиск, частота серцевих скорочень, електрокардіограма, гальванічна реакція шкіри (потовиділення) та положення тіла. Найдосконаліші системи можуть надати допомогу в процесі прийняття рішень під час визначення діагнозу та його

відповідного лікування, а також можуть забезпечити обґрунтування та автономну поведінку

Актуальність. Прикладні інформаційні системи з'явилися відразу після виникнення персональних комп'ютерів, з того часу роль прикладних інформаційних систем в житті людей постійно збільшується. Медичні інформаційні системи існують вже давно, вони є невід'ємною частиною взаємодії між сучасним пацієнтом і лікарем. Також медичні інформаційні системи забезпечують зручні способи зберігання і обробки медичної інформації та створення звітів.

В 2018 році в Україні розпочалась реформа системи охорони здоров'я. Одним з основних законів реформи був закон про обов'язкове подання електронних медичних даних до державної бази даних. На основі цих даних «Національна служба здоров'я України» регулює фінансування медичних закладів.

Таким чином, якщо раніше медичні інформаційні системи забезпечували лише прискорення і комфорт роботи, то тепер вони є обов'язковою частиною процесу отримання державного фінансування.

Одним із пріоритетних напрямів розвитку системи охорони здоров'я є створення єдиного медичного інформаційного простору, який забезпечить прийняття ефективних управлінських рішень на всіх рівнях. Це дасть змогу медичним закладам налагодити ефективний облік діяльності організації здійснювати на сучасному рівні менеджмент, своєчасно отримувати інформацію про передові досягнення в галузі медичної науки, використовувати всю медичну інформацію про пацієнта (за весь період його життя), накопичену зі всіх рівнів надання медичної допомоги для досягнення кращого лікувального ефекту. Наразі інформатизація системи охорони здоров'я України має хаотичний стан: відсутність галузевих стандартів і ліцензування програмного забезпечення, єдиних вимог щодо обсягів і форматів збереження інформації, уніфікованих форм статистичної звітності, не дають змоги сьогодні забезпечити системність, комплексність та узгодженість процесів інформатизації галузі. Подальшим напрямом досліджень є розроблення єдиного медико-

інформаційного простору, в основу якого покладено розглянуті механізми та враховані всі висвітлені зауваження. Єдиний медико-інформаційний простір має забезпечити достатність медико-статистичних даних для управління охороною здоров'я, задоволення інформаційних потреб про стан здоров'я населення та ресурси охорони здоров'я, згідно з вимогами ВООЗ та умовами приєднання до когорти країн з розвиненими медичними інформаційними технологіями.

Мета дослідження – створення медичної інформаційної системи за допомогою сучасних технологій, включаючи реєстрацію та передачу медичної інформації.

Для досягнення мети необхідно вирішити **задачі**:

- дослідити теоретичні основи побудови програмної системи «Медична інформаційна система для поліклініки та амбулаторій»;
- провести аналіз необхідних технічних рішень для зручної побудови системи;
- спроектувати систему на прикладі «Медична інформаційна система для поліклініки та амбулаторій» з можливістю передачі телемедичних даних;
- спроектувати модуль вибору лікаря;
- спроектувати модуль дистанційного спостереження за пацієнтами.

Об'єктом дослідження є медичні інформаційні системи.

Предметом дослідження - методи та засоби створення медичної інформаційної системи.

РОЗДІЛ 1

МЕДИЧНІ ІНФОРМАЦІЙНІ СИСТЕМИ: ОГЛЯД МОЖЛИВОСТЕЙ І ПРИКЛАДИ ВИКОРИСТАННЯ

1.1 Аналіз розв'язуваної задачі й огляд наявних результатів

Системою називають будь-який об'єкт, який одночасно розглядається і як єдине ціле, і як об'єднана в інтересах досягнення поставлених цілей сукупність різнорідних елементів. Системи значно відрізняються між собою як по складу, так і по головним цілям [1, 2, 3]. Поняття системи дуже поширене в галузі медичних інформаційних технологій. Найчастіше системою називають сукупності технічних, програмних і програмно-технічних засобів (рис. 1.1).



Рисунок 1.1 – Система як сукупність технічних, програмних і програмно-технічних засобів

Апаратну частину комп'ютера можна назвати системою. Сукупність комп'ютерних програм, метою яких є розв'язання конкретних прикладних задач, а також методи і процедури автоматизації документообігу і управління розрахунками. Одним з розповсюджених варіантів використання поняття система є інформаційні системи. Основним завданням інформаційної системи є

забезпечення збору, обробки, зберігання, видачі інформації, яка може бути використана на благо людини.

Інформаційна система – взаємозв'язана сукупність засобів, методів і персоналу, використовуваних для зберігання, обробки та видачі інформації для задоволення потреб користувача. В сучасних інформаційних системах використовують багато технічних засобів, однак в більшості випадків основним є персональний комп'ютер або сервер з підключеними до нього клієнтськими пристроями такими як: персональні комп'ютери, планшети або смартфони. На великих підприємствах або у крупних організаціях, у складі основного технічного комплексу можуть знаходитись мейнфрейми або суперкомп'ютери, також нещодавно популярним стало використання хмарних сервісів для розгортання і впровадження високонавантажених інформаційних систем. Не слід забувати що в більшості навіть сучасні інформаційні системи не можуть функціонувати без участі персоналу, а також споживачів для яких призначені результати обробки і видачі інформації в системі.

Медична інформаційна система (MIC) – інформаційна система для автоматизації лікувальної установи [1, 3].

Медична інформаційна система – комплексний програмний продукт, головне призначення якого автоматизація всіх основних процесів, пов'язаних із роботою медичних установ загальної і вузької спеціалізації. Автоматизовані медичні інформаційні системи дозволяють швидко й ефективно налагодити електронний документообіг, гнучко вибудовувати роботу з пацієнтами, вести оперативний облік роботи адміністративного персоналу, контролювати всі організаційні і фінансові питання.

За кордоном прийнято використовувати термін HIS (Hospital Information System) – госпітальна інформаційна система для комплексного управління всіма процесами медобслуговування, у тому числі юридичному аспекті. Доповненнями до неї можуть бути специфічні модулі, наприклад, RIS (Radiology Information System) – радіологічна інформаційна система або PACS (Picture Archiving and Communication System) – система збереження медичних зображень. Окремий вид MIC – лабораторні інформаційні системи (Laboratory

Information Management Systems) й аптечні інформаційні системи (AIC). Вони можуть частково або повністю реалізовуватися у вигляді окремих компонентів комплексної медичної інформаційної системи.

Отже, МІС – це спеціалізоване програмне забезпечення, розроблене спеціально під потреби системи охорони здоров'я. Від інформаційних систем для інших галузей МІС відрізняється тим, що одночасно зберігає і обробляє персональну, демографічну та медичну інформацію пацієнта.

1.2 Класифікація МІС

МІС класифікують за різними ознаками.

1. Для автоматизації процесів збору і обробки інформації можна виділити автоматизовані і автоматичні медичні інформаційні системи. В автоматизованих системах людина виконує частину операцій по збору й обробці інформації. В автоматичних системах людина не приймає участі у процесах, тобто процеси збору і обробки медичних даних повністю забезпечуються системою.

2. Залежно від типу інформаційної бази, МІС поділяються на системи, що оперують даними, та системи, що оперують знаннями. Системи другого типу - це експертні системи. Їхнє функціонування істотно спирається на знання, отримані від експертів, а результати функціонування близькі результатам аналітичної діяльності експертів.

3. Залежно від виду розв'язуваних задач, МІС можна розділити на такі групи:

- інформаційно-довідкові – системи автоматизованого пошуку, вимірювальні системи;
- інформаційно-логічні – діагностичні системи; системи прогнозу; системи моніторингу;
- керуючі або автоматизовані системи управління.

Інформаційно-логічна система призначена для перетворення інформації таким чином, щоб можна було одержати нову інформацію, відсутню в інформаційному масиві.

У системах управління реалізується принципово нова функція – прийняття керуючих рішень. Найбільш широке поширення в медичних установах одержали інформаційно-пошукові системи (ІПС), які у залежності від характеру інформації поділяються на фактографічні і документальні системи [3, 4].

Фактографічні ІПС містять інформаційні масиви фактичних даних. Аналогами таких систем виступають «паперові» довідники, каталоги, технічні паспорти. У комп'ютерних ІПС фактичні дані звичайно зберігаються в базах даних (БД) і являють собою таблиці, у колонках яких вказано назви різних характеристик об'єктів, а в рядках дані опису (значення характеристик) цих об'єктів. Документальні ІПС оперують з інформацією у вигляді документів. Прикладами таких систем можуть бути бібліографічна картотека, картотека з історіями хвороби, інші картотеки. Виконуючи пошук, документальна ІПС надає або номери необхідних документів, або список заголовків, або адреси зберігання шуканих документів. При цьому оцінку інформації, що знаходиться в знайдених документах, робить людина.

Керуючі системи реалізують збір інформації про об'єкт управління, обробку інформації, передачу даних в орган управління, формування керуючого рішення.

МІС або автоматизовані системи медичної документації класифікують на основі принципу ієрархії, які відповідають за це особливій структурі охорони здоров'я, що має кілька рівнів, існують:

- МІС, що мають початковий (базовий) рівень, завданням яких є певна підтримка роботи лікарів різної спеціалізації за допомогою ПК; такі системи можуть поліпшити якість діагностичної та профілактичної роботи, а особливо це стосується умов ведення великого прийому пацієнтів при нестачі часу лікарів. Тут можуть бути системи, призначені для пошуку необхідної інформації і даних медичного призначення за запитом лікуючого лікаря:

- системи, які призначені для того, щоб діагностувати різного роду патології, прогнозувати майбутній стан і вказувати рекомендації для лікування;
- системи, призначені для підтримки і автоматизації діагностики та лікувального процесу, які здійснюються при контакті з пацієнтом;
- автоматизовані робочі місця лікарів, для того, щоб повністю автоматизувати весь робочий процес лікуючого лікаря будь-якої спеціалізації та забезпечити підтримку інформацією для прийняття лікарем рішень діагностичного і тактичного характеру.

● МІС, що мають рівень лікувально-профілактичного закладу:

- інформаційні технології центрів консультації, призначених для підтримки функцій певних підрозділів і забезпечення інформацією медичного персоналу при консультації ними пацієнтів, прийнятті рішень та діагностуванні;
- сховище інформації медичних служб, які несуть в собі дані про число і якості складу медичного персоналу конкретної установи, населення, яке обслуговується ними, основні дані про статистику, характеристики районів та іншу важливу інформацію;
- спеціалізовані реєстри, які містять дані про певний контингент людей, заснованому на формалізації історії хвороби та медичній карті пацієнта;
- спеціалізовані сервіси, за допомогою яких проводиться огляд в якості профілактики до огляду лікарем і виявлення особливих груп ризику і пацієнтів, яким необхідна допомога лікаря;
- ІС науково-дослідних інститутів і медичних вищих навчальних закладів, які визначаються рішенням такі завдань як: інформатизація процесу технології навчання, науково-дослідних робіт, діяльність управління).

● МІС територіального рівня характеризуються такими пунктами:

- інформаційні системи територіального рівня охорони здоров'я;
- інформаційні системи завдань технологій в галузі медицини, які забезпечують інформаційну підтримку діяльності мед. персоналу і спеціалізованих служб;

- телекомунікаційні мережі медичного призначення, які створюють єдиний інформаційний простір в регіоні;
- рівень держави, які призначається для забезпечення інформаційної підтримки охорони здоров'я на рівні держави.

Сучасні медичні інформаційні системи піддаються взаємодії з неймовірно великою кількістю всіляких даних. Від ефективності використання цих даних лікарями, їх керівниками, керуючими підрозділами залежить якість послуг, що надаються в сфері охорони здоров'я. Необхідність використання потоку даних дуже великих масштабів, які постійно збільшуються через такі фактори як діагностика, статистика, управлінські рішення, терапія і так далі, на сьогоднішній день підтверджує необхідність створення автоматизованих інформаційних систем в медичних установах.

Деякий час назад система вітчизняної охорони здоров'я не мала практично ніяких яскравих ознак автоматизації. Такий документообіг як медичні амбулаторні карти, виписки, аналізи, висновки, звіти, облік пацієнтів та інше виконувався повністю вручну самими співробітниками медичних установ на папері. У свою чергу це сильно уповільнювало процес обслуговування пацієнтів, відбувалися помилки, витрачалась велика кількість часу на заповнення медичних карт і складання звітів, що в сукупності сильно впливало на якість охорони здоров'я в цілому. Все це ускладнювало роботу контролюючих органів. Крім того, ефективність використання медичних інформаційних систем в найближчому майбутньому буде обумовлена здоров'ям населення країни. Беручи до уваги цей факт, велика кількість лікувальнопрофілактичних установ активно застосовують у своїй діяльності комплексні медичні інформаційні системи (рис. 1.2). Вони є спеціальним продуктом, який може дозволити здійснювати управлінську діяльність медичного закладу на абсолютно іншому, більш високому рівні. Перевагою комплексних медичних інформаційних систем є поєднання найважливіших функцій. Так для багатьох закладів недостатньо системи яка може виконувати лише обмежене коло завдань.

В сучасних реаліях медичному закладу водночас потрібно автоматично формувати звітність для подання в державні установи, проводити операційні процеси в автоматичному режимі, зберігати медичні дані про своїх пацієнтів. Таким чином комплексні системи надають більшу гнучкість у використанні.



Рисунок 1.2 – Основні функції медичної інформаційної системи [4]

Відмінною рисою медичних інформаційних систем можна назвати перехід від локальної роботи з медичною інформацією до комплексної системи, в якій, вся необхідна інформація, що проходить через медичну установу, може бути доступна з єдиного інформаційного середовища. Крім того, з'являється реалізація технології, при якій немає ніякої необхідності використовувати ресурс паперу, але можливість отримання, при необхідності, паперового варіанту також зберігається. Якість надаваних лікарями послуг стає помітно вище при використанні сучасних технологій МІС. Також зростають показники оптимізації управління різними медичними відділеннями в установі [1,5].

Настав час, коли на заміну паперових документів лікувально-профілактичних установ приходять сучасні інформаційні технології, які спрямовані на здійснення внутрішніх функцій і рішення управлінських завдань в галузі медицини, процес інформатизації охорони здоров'я швидко набирає обертів, також збільшується кількість масштабних проектів.

В 2018 році в Україні почав діяти законопроект про реформу системи охорони здоров'я в Україні. Основною метою реформи є забезпечення всім

громадянам України рівного доступу до якісних медичних послуг та перебудови систему охорони здоров'я так, щоб у її центрі був пацієнт.

В рамках впровадження реформи було створено «Національну службу здоров'я України» (НСЗУ) (рис 1.3). Згідно з Законом, її основними функціями є:

- впровадження державної політики у галузі державних фінансових гарантій медичного обслуговування населення за програмою медичних гарантій;
- моніторинг, аналіз і прогнозування потреб населення України у медичних послугах та ліках;
- виконання функцій замовника медичних послуг та лікарських засобів за програмою медичних гарантій;
- розроблення проекту програми медичних гарантій, внесення пропозицій щодо тарифів;
- укладення, зміна та припинення договорів про медичне обслуговування населення та договорів про реімбурсацію;
- перевірка дотримання надавачами медичних послуг вимог, встановлених порядком використання коштів програми медичних гарантій і договорами про медичне обслуговування населення;
- забезпечення функціонування eHealth – системи охорони здоров'я (рисунок 1.4).

Електронна система охорони здоров'я eHealth – інформаційно-телекомунікаційна система, що забезпечує автоматизацію ведення обліку медичних послуг та управління медичною інформацією в електронному вигляді, до складу якої входять центральна база даних та електронні медичні інформаційні системи, між якими забезпечено автоматичний обмін даними через відкритий програмний інтерфейс (API).

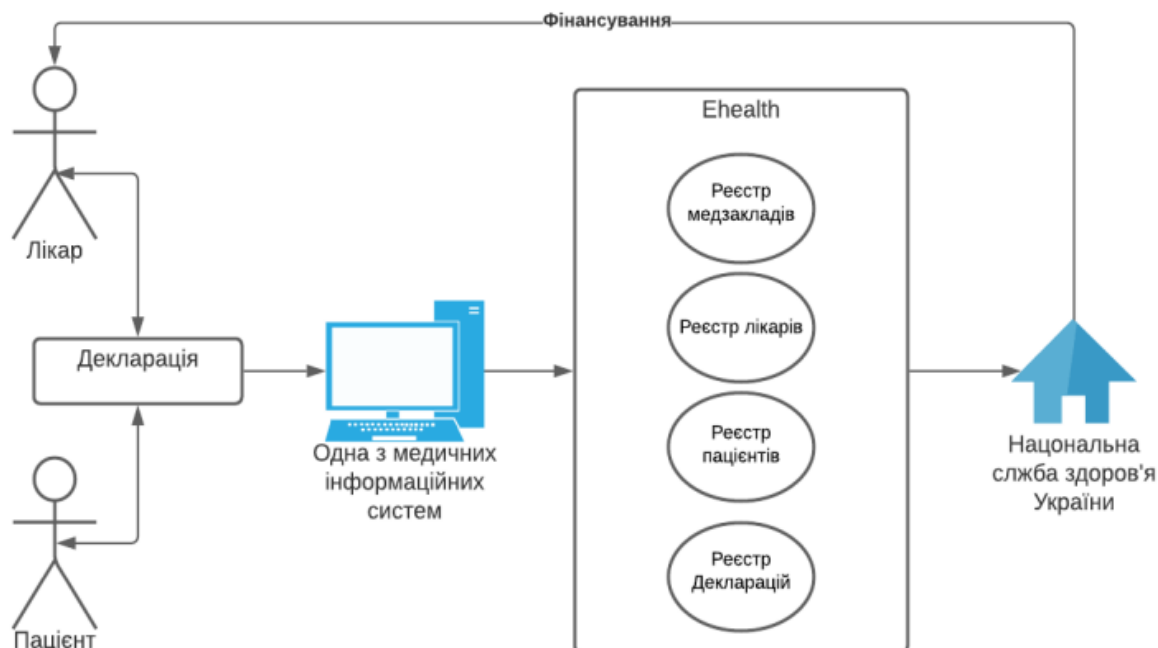


Рисунок 1.3 – Реформа первинної ланки системи охорони здоров'я

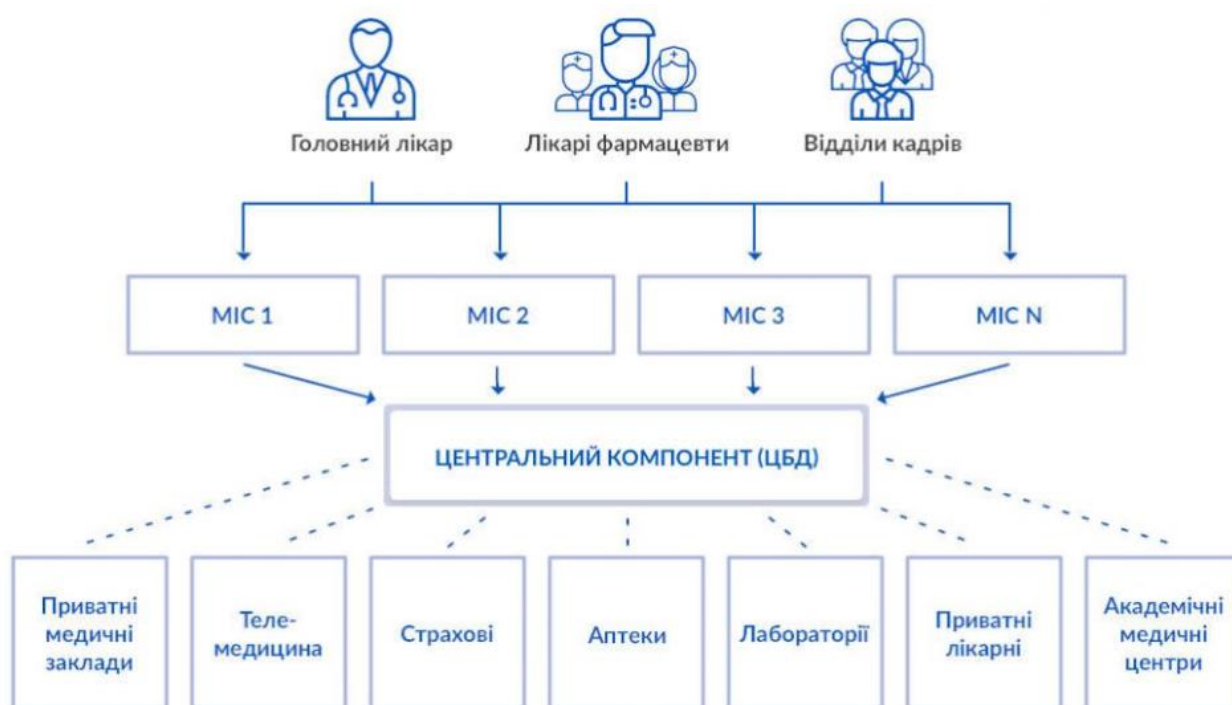


Рисунок 1.4 – Архітектура системи e-Health

Відповідно до [6, 7], медичні системи класифікуються за додатками (допоміжні, керовані), архітектурою (інфраструктура, потреба даних, система композиції), вимірюванням (тип датчика, метод, параметр), управління даними (інтеграція, зберігання, обробка даних), обчислення (моделювання, моніторинг), зв'язок (планування, протокол), безпека (конфіденційність,

шифрування) та контролем/активацією (прийняття рішень, механізм). Враховуючи важливість архітектури для приховування важливих рішень та характеристик систем, у цій роботі основна увага приділяється архітектурі програмного забезпечення для медичних систем, а саме програмним системам, які обробляють, аналізують, контролюють та візуалізують біосигнали для надання розумних медичних послуг, що є 2,3 та 4 рівнями структури ІМС (рис. 1.5).

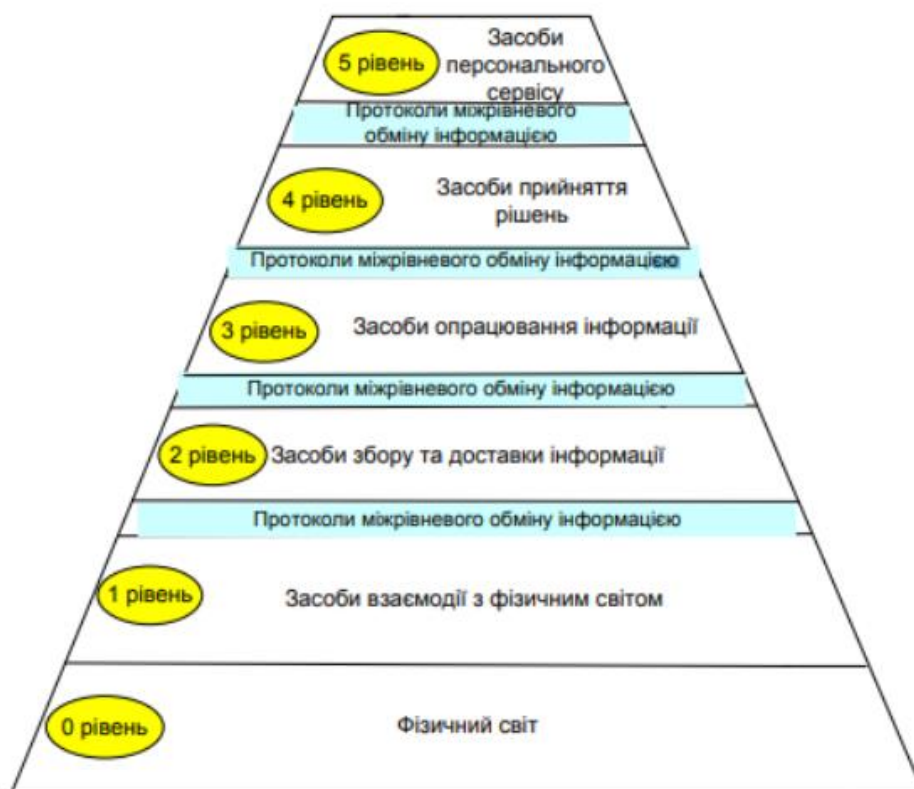


Рисунок 1.5 – Багаторівнева базова платформа ІМС та кіберфізичних систем

1.3 Медичні інформаційні системи та компанії-розробники на ринку України

Після створення державного центрального компоненту eHealth з'явилося багато медичних інформаційних систем. На даний момент більше 20. З найбільш популярних і досконалих слід зазначити: ● Helsi, ● Doctor Eleks, ● Emcimed, ● ПБЧ, ● MedStar, ● Medics.

Система Helsi – медична інформаційна система, що є однією з найбільших в Україні та покриває велику долю ринку. В переліку партнерів компанії Helsi знаходяться лікарі, пацієнти, медичні заклади з усієї України, фармацевтичні компанії та страхові компанії. За офіційними даними Національної служби здоров'я України через систему Helsi укладено 15,6 мільйона декларацій між лікарями та пацієнтами, що становить приблизно 44% всього обсягу укладених декларацій.

За даними самої компанії: через систему HELSI кожного місяця записується до лікаря більше, ніж 2 мільйони користувачів. Продуктом користується більш, ніж 16 тисяч лікарів по всій Україні. Система працює в більш, ніж 1400 медичних закладах. З HELSI співпрацюють більше 530 аптечних мереж, які за період співпраці погасили більше 6 мільйонів рецептів.

Система HELSI розроблена у форматі веб-застосунку. Веб застосунок складається з трьох частин: helsi.me; helsi.pro; reform.helsi.me.

[Helsi.me](https://helsi.me) [16] – це частина системи призначена для пацієнтів. Вона знаходиться у відкритому доступі (рис. 1.6). Основними функціями порталу helsi.me є перегляд каталогу лікарів, перегляд каталогу медичних установ, можливість попереднього заповнення декларації, яке дає змогу ввести всю необхідну для заключення інформацію на сайті і лише прийти до лікаря щоб підтвердити особу і отримати паперовий екземпляр, також для зареєстрованих користувачів доступна можливість запису на прийом до лікаря з зручним процесом вибору часу. Для користувачів, у яких укладена декларація з сімейним лікарем, в особистому кабінеті доступна інформація про історію їх візитів і взаємодій з лікарем, також вони можуть переглянути виписані їм направлення і в якому закладі ці направлення проходять обробку і погашення.

Не менш важливим є доступ до результатів діагностичних досліджень і процедур. У комунальних закладів є доступ до програми доступні ліки, тому пацієнти можуть отримувати безкоштовні ліки за унікальними ідентифікованими рецептами які доступні їм особистому кабінет.

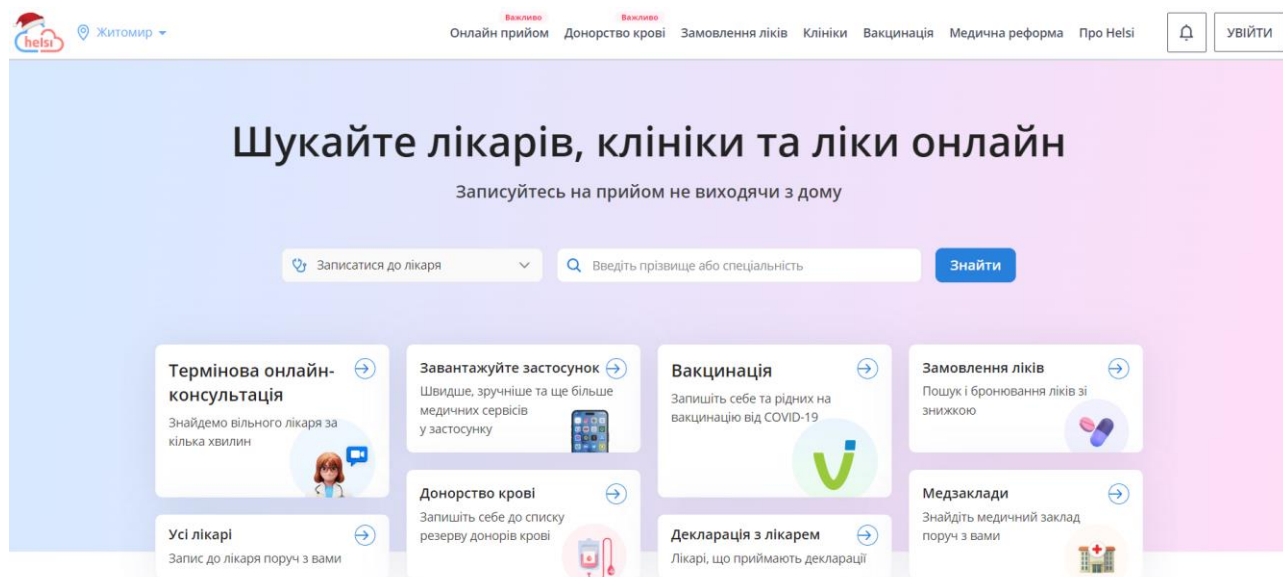


Рисунок 1.6 – Головна сторінка helsi.me [16]

Каталог лікарів є дуже гнучким, в ньому є маса фільтрів для вибору потрібного пацієнту лікаря за типами установи де працює лікар, можливістю онлайн прийому, географічним районом міста, а також не менш важливою є можливість фільтрація за вартістю прийому (рис. 1.7).

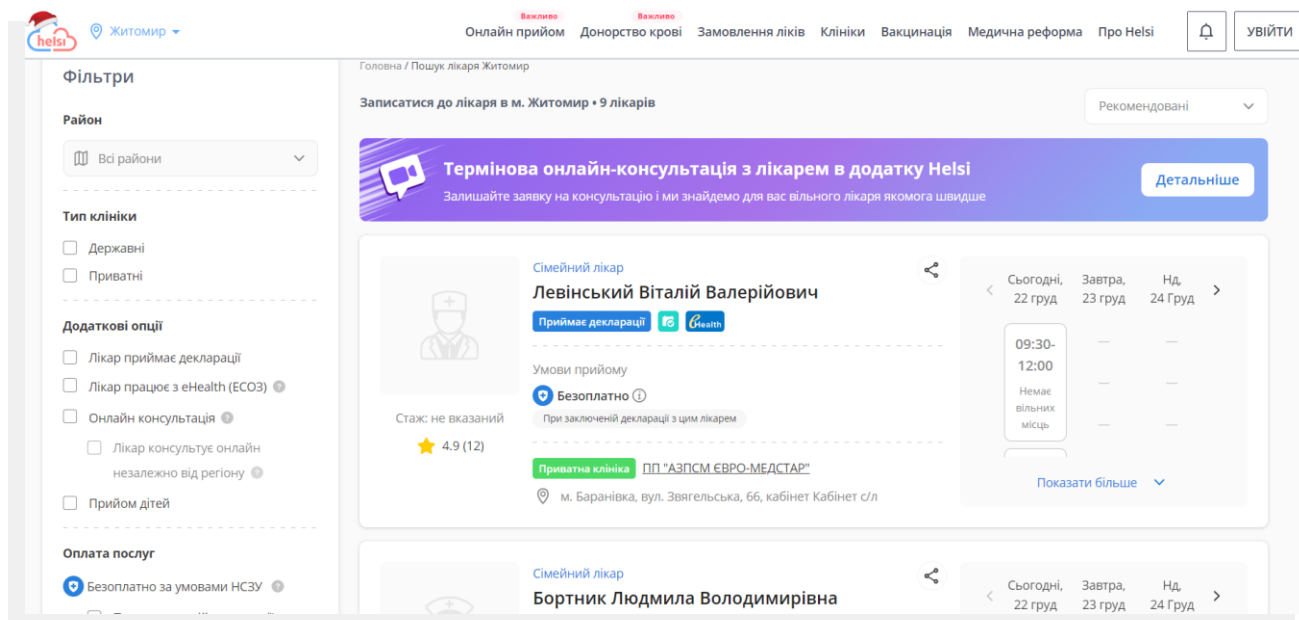


Рисунок 1.7 – Мторінки каталогу з фільтрами

Helsi.pro – це частина системи призначена для лікарів і персоналу медичного закладу. Вона є закритою, тобто доступ через інтернет є можливим, лише для виділених ір-адрес медичних установ які уклали договір з компанією. Основними функціями системи helsi.pro є адміністрування медичного закладу,

наприклад створення підрозділів, реєстрація і налаштування профілів лікарів, отримання звітності. Також це основна платформа для ведення прийому лікарем.

Доступне ведення прийому в форматі eHealth, з кодуванням діагнозів через систему ICPC2 або МКХ-11. Reform.helsi.me – це частина системи призначена для адміністрації мед закладу. Через цю систему здійснюється інтеграція закладів з центральним компонентом eHealth, а саме реєстрація медичних установ, реєстрація підрозділів медичних установи та укладення договору з Національною службою здоров'я України (рис. 1.8).

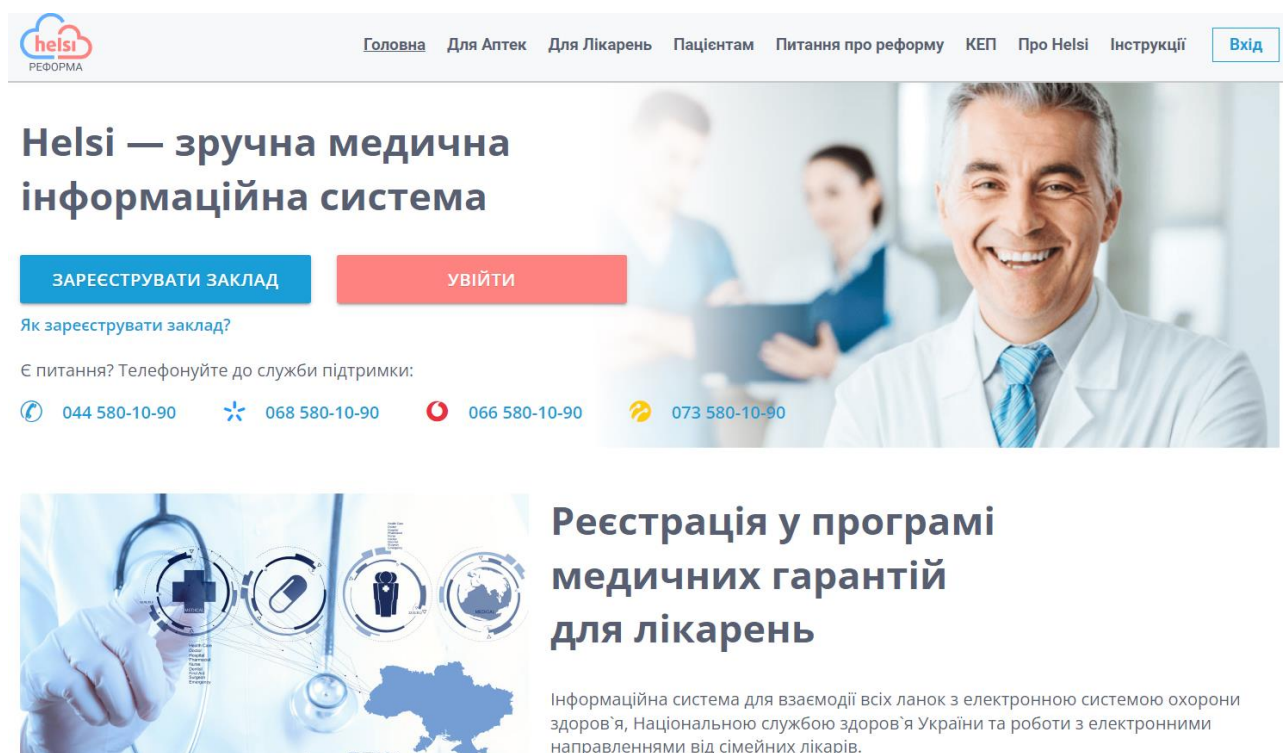


Рисунок 1.8 – Головна сторінка reform.helsi.me

Основними недоліками системи Helsi є:

- низька швидкість роботи підсистеми helsi.pro;
- не зовсім зрозумілий інтерфейс для лікарів, які не мали досвіду роботи з медичними інформаційними системами;
- занадто громіздка для малих установ і ФОПів;
- інтерфейс у трьох підсистемах різний, тому перехід з reform.helsi.me на helsi.pro викликає у лікарів труднощі.

Doctor Eleks – це друга по розповсюдженості медична інформаційна система. Вона з'явилась ще у 2006 році і є однією з перших великих медичних інформаційних систем на території України (рис. 1.9). Основними функційними модулями системи Doctor Eleks є:

- електронна медична картка для централізованого збереження інформації про пацієнта;
- розклад для ефективного планування та управління потоком пацієнтів;
- звіти та фінанси для оперативного доступу до аналітичних та статистичних даних;
- інтеграція з діагностичним обладнанням для легкого зберігання та швидкої обробки медичних зображень та відео;
- лабораторія для автоматизації збору, обробки та аналізу отриманих даних;
- послуги та їх оплата для безпечного управління грошовими потоками та контролю фінансів.

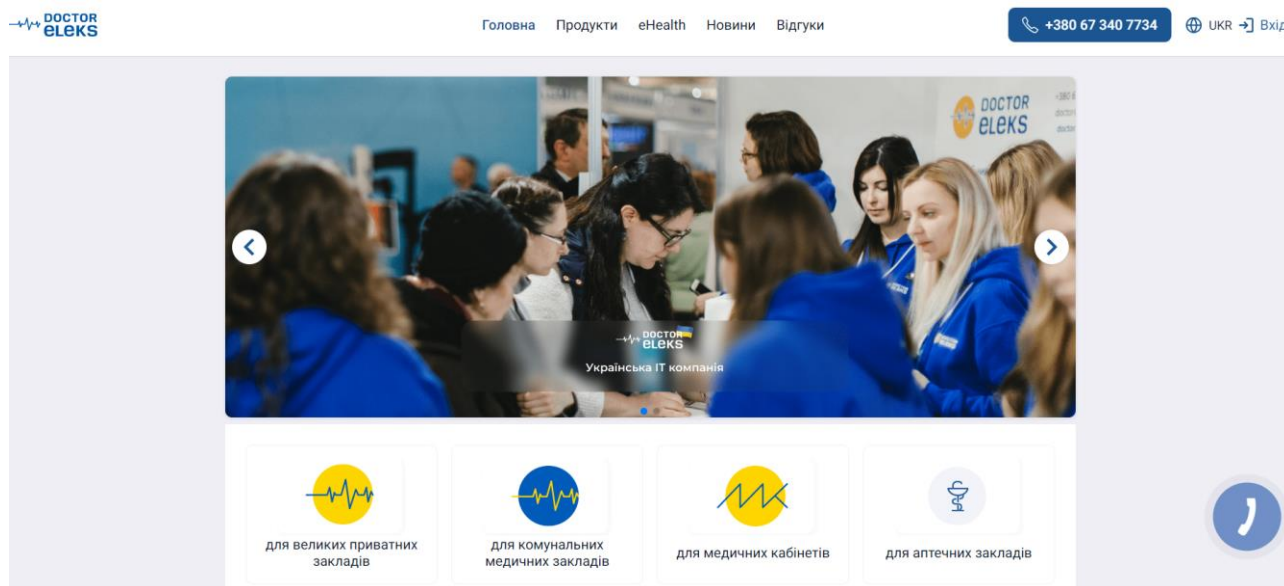


Рисунок 1.9 – Головна сторінка Doctor Eleks [17]

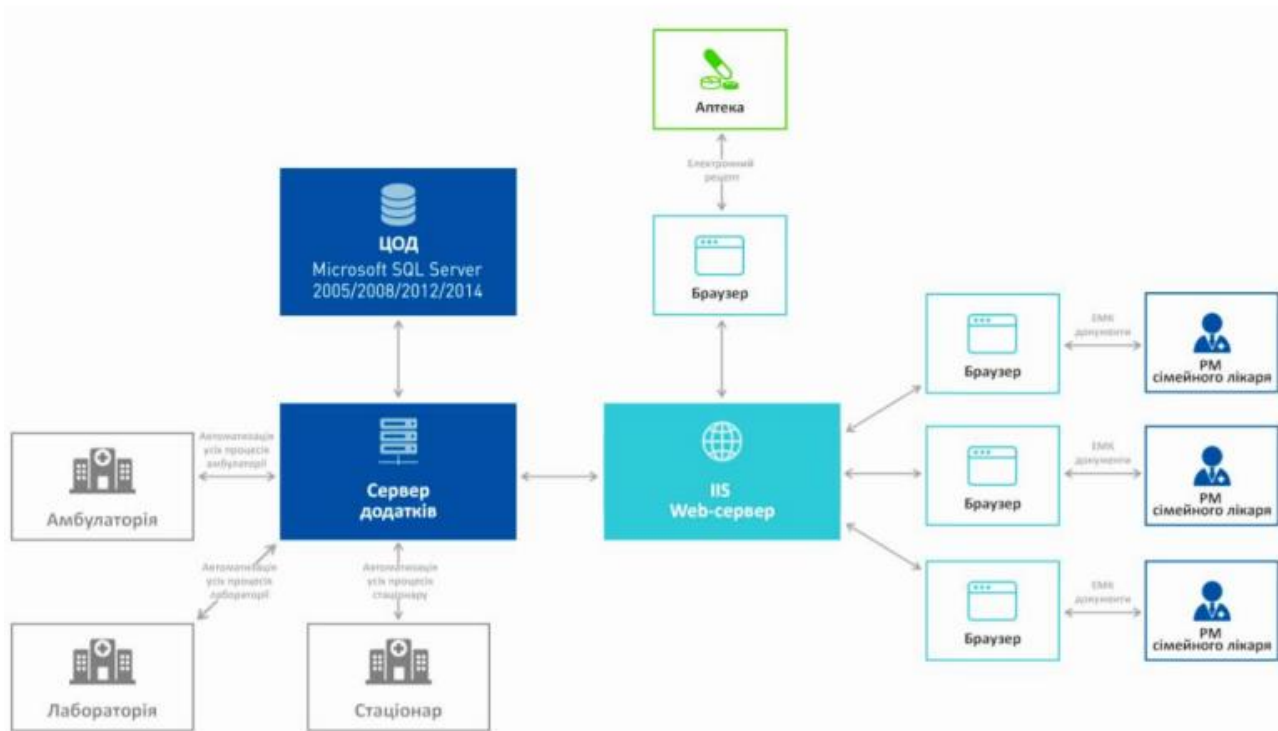


Рисунок 1.10 – Архітектура системи Doctor Eleks

Основними недоліками системи Doctor Eleks є:

- основний модуль системи для автоматизація робочого місця лікаря розповсюджується у форматі десктопного застосунку під операційну систему Windows, що робить неможливим його використання на інших платформах;

- десктопний застосунок має застарілий дизайн, незвичний і не зрозумілий для пересічного користувача;

- вартість впровадження системи є досить високою на одне робоче місце, урахування витрат на устаткування, сервери та технічну підтримку.

Ринок медичних інформаційних систем представлений всього декількома компаніями-розробниками програмного забезпечення. За кількістю впроваджень варто відзначити: «Медсистеми», CIET, «Укрмед соф», TherDer та «Медексперт».

Більшість систем побудовано на основі клієнт-серверної архітектури, яка забезпечує обмежену кількість функцій – переважно підготовку статистичних звітів і стандартних форм МОЗ. У цих системах ведеться електронна історія хвороби, внесення даних до яких здійснюється шляхом набору тексту або вибору фраз з випадających списків-довідників. Такий підхід не дає змоги в

подальшому здійснювати аналітику та поглиблений аналіз. Основним недоліком цих систем є потреба постійно звертатися до розробників для внесення змін у вхідні й вихідні форми.

ArchiMed – система автоматизації медичних установ будьякого профілю. Роботу над програмним комплексом ведуть з 2005 р., його основою є платформа «Archi», яка є розробкою компанії MLS IT Systems.

Одним з ключових моментів у впровадженні програмного комплексу є його проста установка, доступна користувачу з базовими знаннями персонального комп'ютера.

- Доктор Елекс – комплексна медична інформаційна система, яка містить всі необхідні для роботи модулі: реєстратура, фінанси, стаціонар, лабораторія та інші. Систему розробила компанія «Елекс».

- «TherDer» – медична інформаційна система, у якій існують функції табличного та графічного представлення, зокрема динаміки зміни цифрових показників. Колірне відображення результатів аналізів залежно від меж норми, можливості експорту текстової та табличної інформації в інші формати.

- Медична інформаційна система «Каштан», розробник компанія СІЕТ. Інформаційний медичний комплекс програмних систем для управління закладом охорони здоров'я. Комплекс призначений для оперативного аналізу та управління всіма сферами діяльності установи охорони здоров'я та охоплює медичні установи на всіх рівнях системи охорони здоров'я України.

- Комплекс від компанії «Укрмедсофт» – спрямований на повну комп'ютеризацію лікувально-діагностичного процесу в лікувальному закладі стаціонарного типу та на повну комп'ютеризацію лікувально-діагностичного процесу в поліклініці (табл.).

Сюди дописати інші системи

Зосередження уваги на цифровій охороні здоров'я, Connected Health [18] або eHealth (у різних її варіантах, таких як телемедицина, телездоров'я та mHealth), стосується медичних послуг та інформації, метою якої є покращення здоров'я, яка надається або покращується через Інтернет та супутні технології. Це спрямовано на зменшення витрат порівняно з традиційною медичною

послугою. Зокрема, телемедицина визначається як використання ІКТ для надання клінічних послуг на відстані, тоді як mHealth має на меті ту ж мету, але підтримується мобільними пристроями (стільниковими телефонами або цифровими персональними асистентами).

Таблиця 1.1 – Зведені характеристики системи

Характеристика	Назва МІС				
	ArchiMed	Doctor Eleks	TherDep	Каштан	Укрмед софт
Клієнт-серверна архітектура	+	+	+	+	+
Рівень вимог до апаратних засобів	середній	середній	низький	високий	високий
Здатність співпрацювати з медичним обладнанням	+	+	-	-	-
Наявність механізмів захисту інформації	+	+	-	-	-
Наявність веб-інтерфейсу	+	+	-	-	-
Підєднання додаткових модулів	+	+	+	+	+
Механізми статистичного оброблення баз даних	+	+	+	+	+
Використання баз даних	+	+	+	+	+
Можливість конфігурації системи	+	+	+	+	+

Зробивши висновки з проведеного аналізу наявних медичних інформаційних систем, їх переваг та недоліків, а також враховуючи існуючі вимоги НСЗУ можна зробити наступні висновки:

- всі системи розповсюджуються на платній основі;
- не мають модульної архітектури і тому не можуть бути налаштовані під вимоги конкретного закладу;
- через громіздкість не можуть швидко реагувати на нові вимоги НСЗУ і eHealth.

1.4 Висновки та основні задачі дослідження

Варто зазначити, що eHealth є досить складною системою, яка поступово розбудовується та впроваджується у роботу вітчизняних закладів охорони здоров'я. Цей процес розпочався із первинної ланки надання медичної допомоги: сімейних лікарів, терапевтів та педіатрів, а також пацієнтів які уклали декларації із обраними лікарями та закладів охорони здоров'я, які пройшли процес автономізації.

Запровадження медичних інформаційних систем та використання таких їх модулів, як аналітико-статистичного, управлінського обліку, управління якістю в режимі реального часу, покликані покращити ефективність прийняття управлінських рішень на усіх рівнях функціонування галузі охорони здоров'я. Відтак розбудова єдиної (інтегрованої) цифрової медичної платформи eHealth стає ключовим напрямом державної політики на шляху трансформаційних змін у сфері охорони здоров'я.

Електронна система охорони здоров'я являє собою інформаційно-комунікаційну платформу для здійснення обліку наданої медичної допомоги та послуг, керування інформацією медичного характеру шляхом її створення, обміну, оприлюднення та передачі у електронному вигляді. Зазначена платформа містить центральну базу даних, медичні інформаційні системи нижчого рівня з встановленою можливістю автоматизованого обміну інформацією з використанням програмного інтерфейсу API [8] (рис. 1.4). Національна електронна система охорони здоров'я задекларована з такою архітектурою: центральна компонента (Центральна база даних – ЦБД) забезпечує централізоване зберігання та обробку матеріалів медичних інформаційних систем, які забезпечують двосторонній обмін інформацією [9].

Центральна база даних – програмно-апаратний комплекс, за допомогою якого здійснюється формування реєстрів, збір даних медичних інформаційних систем та їх взаємодія із інформаційними модулями Національної служби здоров'я України (НСЗУ).

Медичні інформаційні системи переважно мають модульну архітектуру, яка дозволяє окремим модулям працювати як автономно, так і у складі комплексу, об'єднаного ядром системи, що зберігає ключову інформацію (реєстрація документів, подій), база електронних медичних карт, тощо. Найбільш затребувані модулі лікарняних МІС:

- амбулаторно-поліклінічна медична допомога (електронна медична карта, реєстратура, реєстрація медичних та діагностичних досліджень, робота з декретованим населенням, електронний рецепт);
 - стаціонарна медична допомога (планування госпіталізації, історія хвороби, протокол оперативного хірургічного втручання, тощо);
 - статистика-аналітика (формування державної статистичної та аналітичної і управлінської звітності, притому в режимі реального часу);
 - адміністрування (управлінський облік, управління якістю;
 - бухгалтерія (бухгалтерська звітність, облік ліків);
 - економіка (планування фінансування, облік та аналіз витрат, розрахунок вартості медичних послуг);
 - загально-системні (захист, обмін даними між ЗОЗ, базові класифікатори та довідники, протоколи лікування, структура території обслуговування).
- Повний перелік МІСів, яким МОЗ України надало можливість доступу до ЦБД, налічує понад 15 вітчизняних компаній-виробників.

Основними задачами роботи є розробка медичної інформаційної системи як веб-застосунку, що матиме функціональний модуль для проведення лікарських прийомів за стандартом ІСРС2 для медичних закладів первинної ланки та надання телемедичної допомоги.

РОЗДІЛ 2 ПРОЕКТНІ І ТЕХНІЧНІ РІШЕННЯ

2.1 Розробка МІС. Стандартний і кастомний софт

Відзначимо, що індивідуальна розробка МІС – це, в основному, окремі випадки для комерційних клінік. Більшість медзакладів використовують готові програмні рішення, їхній вибір залежатиме від типу закладу та повсякденних завдань, котрі належить оптимізувати за допомогою МІС.

Отримати доступ до можливостей МІС можна трьома способами:

1. **Дешево.** Обираємо готове програмне рішення і працюємо на базовому функціоналі, необхідному для виконання всіх поточних завдань. Бажано вибирати системи, розробники яких пропонують скористатися пробним доступом до всіх функцій – так ви можете зрозуміти, чи дійсно вам потрібно стільки модулів, або можна обійтися стандартними, що входять у базовий комплект поставки.

2. **Раціонально і з урахуванням потреб медустанови.** Йдеться про інтеграцію готового серверного рішення з подальшим «Допилюванням» його під конкретні потреби організації. На практиці користувачі можуть виявити, що звичні алгоритми їхніх дій або прийняття рішень розходяться з тими, що пропонує система. Набагато простіше найняти розробників і переписати/дописати функціонал, ніж перебудовувати роботу всієї організації під конкретне ПЗ.

3. **Індивідуально, під нестандартні завдання.** Тобто, замовити систему з нуля. Цей варіант може підійти установам з унікальною специфікою роботи. Сучасні комплексні МІС, як правило, пропонують інтеграцію або мають вбудовані модулі, заточені під специфічні завдання (управління лабораторіями, стаціонар, поліклініка, блоки живлення і т.д.). Можливо, необхідно доповнити вже існуючу систему новими можливостями, наприклад, додати портал для навчання персоналу або підключити до блоку з укладання договорів. Може виявитися, що постачальник ПЗ не передбачив наявності таких модулів, а підключення сторонніх програм до системи, що містить

вразливі персональні дані – досить ризикована затія. Розробка індивідуальної МІС для оптимізації рутинних операцій – рішення, де потрібно зважити всі за і проти. Однак у результаті можливо отримати результат, який необхідний.

Усі МІС поставляються у двох варіантах:

- **хмарні** (Saas) платформи;
- **стаціонарні** (з установкою на сервер компанії).

Хмарна медична інформаційна система. Підходить для будь-яких закладів охорони здоров'я та лікарів з приватною практикою. Перевага такої системи в тому, що вона доступна практично з будь-якого пристрою з доступом в інтернет. Для її впровадження не потрібно розгортати дороге серверне обладнання, а підтримку здійснює компанія-розробник.

Використання хмарних сервісів знижує вартість програм, вся підтримка, забезпечення безпеки лежить «на совісті» розробників, а принцип роботи простий: встанови і користуйся. Високий рівень конкуренції і наявність подібних продуктів на ринку сприяють тому, що майже всі пропонують максимально зручний інтерфейс, навчання і підтримку, тестовий доступ до функціонала. Мінус - вони не завжди здатні охопити всі процеси організації.

Серверна (або стаціонарна). Підходить в основному для дуже великих медичних установ через їхню складну інфраструктуру і величезний обсяг інформації. МІС встановлюється на сервер компанії. Збором інформації, організацією і адмініструванням системи в цьому випадку буде займатися установа охорони здоров'я самостійно. Перевагою серверної системи в тому, що, встановлюючи додаткові модулі під внутрішні потреби установи, можна зробити програму більш гнучкою.

Перевагою розгортання МІС на базі власного серверного обладнання буде те, що треті особи повністю виключаються з процесу збору та управління даними, набір сервісів розширюється шляхом придбання додаткових модулів, роблячи саму систему і її налаштування більш гнучкими.

Важливий етап розробки МІС – визначення вимог для системи. Залежно від установи, чи це стоматологічна клініка, чи санаторій, існують різні потреби

у функціональності системи, що впроваджується, наборі модулів. Про можливості типових МІС, а також про формати поставки ПЗ можна прочитати тут.

З урахуванням необхідного функціоналу, термінів і бюджету ми пропонуємо оптимальне рішення. Але при цьому завжди дотримуємося власної низки вимог при розробці медичного програмного забезпечення.

Вимоги:

1. Гнучка архітектура, яку можна масштабувати. Ми надаємо перевагу використанню хмарних рішень або рішень на private-серверах. Рішення перевірені на великих проєктах, де зберігаються дані мільйонів пацієнтів.

2. Захист персональних даних і безпека, що включає в себе документованість системи, моніторинг вразливостей елементів архітектури і т.п.

3. Омніканальні інтерфейси. Для кожного користувача медичної системи: пацієнт, лікар, лаборант, офісний працівник, кур'єр - повинен бути свій зручний клієнт - додаток, веб-кабінет, програма для робочого столу тощо.

4. Гнучкість у доопрацюваннях. Система автоматизації медичної установи повинна розвиватися разом зі зростанням бізнесу та розвитком всього ринку.

5. Можливість інтеграцій: відкриті високозахищені API, можливості для інтеграції сторонніх модулів та ін.



Рисунок 2.1 – Внутрішня структура медичної інформаційної системи

2.2 Види інформаційного та технічного забезпечення

Щоб реалізувати функціональну частину автоматизованої інформаційної системи потрібно мати відповідні ресурси, створені забезпечувальною частиною.

До складу забезпечувальної частини АІС входять окремі забезпечувальні підсистеми, об'єднуючи певний вид ресурсів та умови їх організації. Набір ресурсів, необхідних для функціонування АІС, абсолютно типовий, тому вони є практично однаковими для різних АІС ІЗ (інформаційне забезпечення) – це сукупність методів і засобів створення єдиного інформаційного фонду, систематизація та уніфікація показників і документів, розробка засобів формалізованого опису даних.

Інформаційне забезпечення є найважливішим елементом автоматизованої інформаційної системи, що наповнює управлінські задачі конкретним змістом. Від якості розробленого інформаційного забезпечення значною мірою залежить достовірність і ефективність прийнятих управлінських рішень.

ТЗ (технічне забезпечення) – сукупність технічних засобів, що забезпечують функціонування автоматизованої інформаційної системи, і містить пристрої, за допомогою яких виконуються типові операції опрацювання даних. При цьому, опрацювання даних може здійснюватися як поза ЕОМ (периферійні технічні засоби збору, реєстрації, первинного опрацювання інформації, засоби телекомунікації і зв'язку), так і на ЕОМ різних класів.

Технічні засоби можна поділити на п'ять груп:

1. Засоби збору та реєстрації інформації: пристрої підготовки даних, засоби збору інформації. Ці засоби призначені для приведення інформації у зручний для дистанційної передачі і подальшої обробки вигляд

2. Засоби передачі інформації. Ця група засобів призначена для передачі інформації у просторі. Це ті ж самі пристрої, що перераховані вище, за наявності інтернет з'єднання

3. Засоби зберігання інформації. При наявності інтернет з'єднання більшість даних переноситься на віддалений сервер, щоб не було можливості

перевантаження слабких пристроїв. При відсутності з'єднання інформація зберігається на пристрої до появи інтернету.

4. Засоби обробки інформації (засоби обчислювальної техніки з відповідним програмним забезпеченням) складають основу комплексу технічних засобів. Вони призначені для перетворення вихідних даних у результативну інформацію, необхідну для прийняття управлінських рішень.

Головні характеристики даної групи – швидкодія й обсяг пам'яті. Виділяють:

- універсальні – для багатьох користувачів і персональні;
- спеціалізовані – сервери та робочі станції.

5. Засоби видачі інформації: принтери, монітори – призначені для перетворення інформації у вигляд, зручний для сприйняття користувачем.

ПЗ (програмне забезпечення) – це сукупність програм для ефективної організації обчислювального процесу в автоматизованих інформаційних системах керування.

СПЗ (системне програмне забезпечення) – це сукупність програм, які управляють процесом обробки інформації в обчислювальних системах і забезпечують робоче середовище для функціонування прикладних програм.

Базові програмні засоби служать для автоматизації взаємодії людини і комп'ютера, організації типових процедур опрацювання даних, контролю і діагностики функціонування технічних засобів системи опрацювання даних. До їх складу входять: операційні системи, сервісні програми (оболонки, утиліти, антивірусні засоби), програми технічного обслуговування.

Проаналізувавши вимоги до медичної інформаційної системи, а також наявне у лікарів і пацієнтів ТЗ можна стверджувати, що найбільш зручною платформою буде веб-застосунок. Архітектурою веб-застосунку обрано клієнт-серверну архітектуру (рис. 2.2).

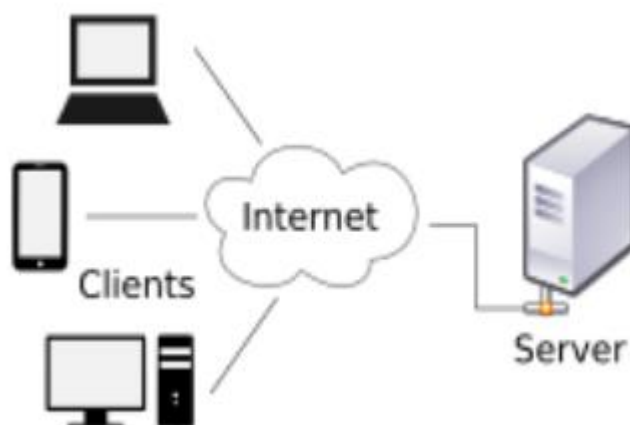


Рисунок 2.2 – Схема роботи клієнт-серверної архітектури

Модель клієнт-сервер – це розподілена структура додатків, яка розділяє завдання або робочі навантаження між постачальниками ресурсу або послуги, що називаються серверами, і запитувачами послуг, що називаються клієнтами. Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому апаратному забезпеченні, але і клієнт, і сервер можуть знаходитись в одній системі. Хост сервера запускає одну або декілька серверних програм, які діляться своїми ресурсами з клієнтами. Клієнт не ділиться жодним із своїх ресурсів, але він вимагає інформацію від сервера.

Таким чином, клієнти ініціюють сеанси зв'язку із серверами, які очікують на вхідні запити [1] (рис.2.3).

В якості серверної частини виступає бекенд з архітектурою REST API. REST розшифровується як Representational State Transfer, а API – Application Program Interface (програмний інтерфейс додатку). REST – це програмний архітектурний стиль, який визначає набір правил, які будуть використовуватися для створення веб-служб.

Веб-сервіси, що відповідають архітектурному стилю REST, відомі як RESTful веб-сервіси. Це дозволяє запитувати системи доступу та керувати веб-ресурсами, використовуючи єдиний і заздалегідь визначений набір правил. Взаємодія в системах на базі REST [20] відбувається через протокол передачі гіпертексту в Інтернеті (HTTP).

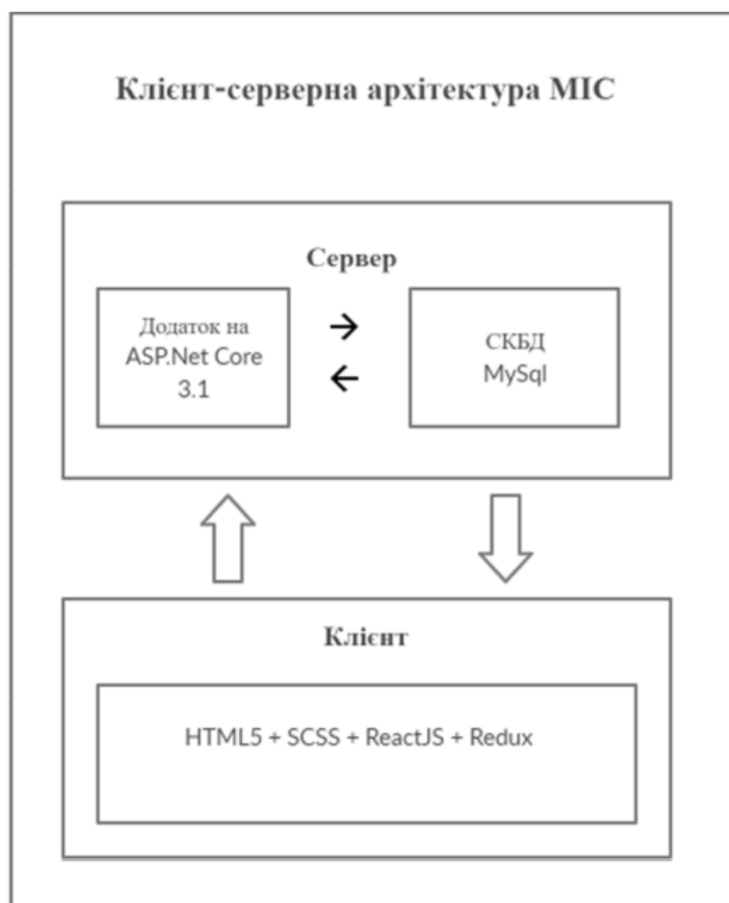


Рисунок 2.3 – Схема роботи клієнт-серверної архітектури у медичній інформаційній системі

Система REST складається з:

- клієнт, який запитує ресурси;
- сервер, який має ресурси.

Важливо створити API REST відповідно до галузевих стандартів, що призводить до полегшення розробки та збільшення кількості клієнтів.

Архітектурні обмеження API RESTful, які обов'язкові для будь-якої веб-служби перераховані нижче:

- Модель клієнт-сервер
- Відсутність стану
- Кешування
- Одноманітність інтерфейсу. Наявність уніфікованого інтерфейсу є фундаментальним вимогою дизайну REST-сервісів. Уніфіковані інтерфейси дозволяють кожному з сервісів розвиватися незалежно.

До уніфікованих інтерфейсів пред'являються наступні чотири обмежувальних умови:

1. Ідентифікація ресурсів – всі ресурси ідентифікуються в запитах, наприклад, з використанням URI в інтернет-системах. Ресурси концептуально відокремлені від уявлень, які повертаються клієнтам. Наприклад, сервер може надсилати дані з бази даних у вигляді HTML, XML або JSON, жоден з яких не є типом зберігання всередині сервера.

2. Маніпуляція ресурсами через представлення – якщо клієнт зберігає уявлення ресурсу, включаючи метадані – він володіє достатньою інформацією для модифікації або видалення ресурсу.

3. «Самодостатні» повідомлення – кожне повідомлення містить достатньо інформації, щоб зрозуміти, яким чином його обробляти. Наприклад, обробник повідомлення (parser), необхідний для отримання даних, може бути зазначений у списку MIME-типів.

4. Гіпермедіа як засіб зміни стану програми (HATEOAS) – клієнти змінюють стан системи тільки через дії, які динамічно визначені в гіпермедіа на сервері (наприклад, гіперпосилання в гіпертексті). Виключаючи прості точки входу в додаток, клієнт не може припустити, що доступна якась операція над якимось ресурсом, якщо не отримав інформацію про це в попередніх запитах до сервера.

- Багатошарова система. Клієнт зазвичай не здатний точно визначити, взаємодіє він безпосередньо з сервером або ж з проміжним вузлом, в зв'язку з ієрархічною структурою мереж (маючи на увазі, що така структура утворює шари). Застосування проміжних серверів здатне підвищити масштабованість за рахунок балансування навантаження і розподіленого кешування. Проміжні вузли також можуть підкорятися політиці безпеки з метою забезпечення конфіденційності інформації.

- Код на вимогу. REST може дозволити розширити функціональність клієнта за рахунок завантаження коду з сервера у вигляді аплетів або сценаріїв. Філдінг стверджує, що додаткове обмеження дозволяє спроектувати архітектуру, підтримуючу бажану функціональність в загальному випадку, але,

можливо, за винятком деяких контекстів Єдиним необов'язковим обмеженням архітектури REST є код на вимогу. Якщо додаток порушує будь-які інші обмеження, його не можна чітко називати RESTful. Для створення клієнтської частини було обрано архітектуру односторінковий додаток (single-page application, SPA).

Односторінковий додаток (SPA) – це веб-додаток або веб-сайт, який взаємодіє з веб-браузером, динамічно переписуючи поточну веб-сторінку новими даними з веб-сервера, замість стандартного способу, тобто завантаження цілих нових сторінок. Мета – швидші переходи, які дозволяють веб-сайту відчувати себе як рідний додаток. У SPA всі необхідні HTML, JavaScript та CSS-код отримують браузер із завантаженням однієї сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку за необхідності, як правило, у відповідь на дії користувача. Сторінка не завантажується в будь-який момент процесу, а також не передає керування на іншу сторінку, хоча хеш місцеположення або API історії HTML5 можна використовувати для забезпечення сприйняття та навігації окремих логічних сторінок у програмі. SPA повністю завантажується при початковому завантаженні сторінки, а потім частини сторінок замінюються або оновлюються новими фрагментами сторінок, завантаженими з сервера на вимогу. Щоб уникнути зайвого завантаження невикористаних функцій, SPA часто поступово завантажуватиме більше функцій, коли вони стануть необхідними, або невеликі фрагменти сторінки, або повно-екранні модулі. Таким чином існує аналогія між "станами" в SPA та "сторінками" на традиційному вебсайті. Оскільки "навігація за станом" на одній сторінці аналогічна навігації по сторінках, теоретично будь-який веб-сайт на основі сторінки може бути перетворений на односторінковий, замінивши на одній сторінці лише змінені частини.

2.3 Аналіз архітектури програмного забезпечення МІС

Архітектури програм описують структуру програмних систем шляхом приховування деталей низького рівня та абстрагування важливих функцій

високого рівня [22], що відповідає як функціональним, так і нефункціональним вимогам. Проектування, специфікація та аналіз структури програмноінтенсивних систем стали критичними питаннями при розробці програмного забезпечення [23], з'явившись як рішення для проектування та розробки великих та складних програмних систем. Насправді вже давно визнано, що архітектура має сильний вплив на життєвий цикл системи і є критично важливим елементом успішного розвитку, а також успішної еволюції програмно-інтенсивних систем [24] [25].

Зважаючи на важливість рівня архітектури розробки систем, потрібен надійний консенсус з точним визначенням архітектури системи. Стандарт IEEE 1471-2000 (2007) [26] та наступний стандарт ISO/IEC/IEEE 42010 (2011) забезпечують основу для архітектур програмно-інтенсивних систем з точки зору стандартизації елементів та практик для опису архітектури. Набір концепцій, пов'язаних з архітектурами програмного забезпечення, визначених стандартом IEEE 1471-2000, використовується наступним чином.

Система охоплює окремі додатки, системи, підсистеми, системи систем, лінійки продуктів або екосистеми. Система має одну або декілька зацікавлених сторін. Зацікавлена сторона системи – це особа, команда або організація, що мають інтереси в системі [26].

Фундаментальна організація системи проявляється через її архітектуру: компоненти системи, їх взаємозв'язок між собою та навколишнім середовищем, а також принципи, що керують розробкою та еволюцією системи. Компонент розглядається як чорний ящик, що показує високий рівень інкапсуляції та абстракції, а також взаємодії, з якими він обмежений через свої інтерфейси.

Відповідно до [27], архітектурні стилі можна класифікувати за комунікацією (наприклад, SOA (Service-oriented Architecture), message bus), розгортанням (наприклад, клієнт/сервер, багаторівнева архітектура (N-Tier), програмне забезпечення багатосторонніх програм (multitenant cloud) та структурою (наприклад, component-based, layered architecture). Service-oriented Architecture (SOA).

Сервісно-орієнтована архітектура пропонує переваги вільного з'єднання сервісу та його постачальника, пристосованості за допомогою композиції, організації та інкапсуляції сервісу для інтеграції у високо розподілену систему [28]. SOA прописує, як обчислювальні суб'єкти взаємодіють таким чином, що запитувач знає лише, що таке робота сервісу і як її запитувати; і сервіс єдиний, який знає про його реалізацію. Багато компаній знаходять, що зробити їх додатки високо доступними, масштабованими, модифікованими та гнучкими все ще залишається складним завданням.

Мікросервіси – це новий архітектурний стиль для розвитку розподілених систем, який має намір вирішити ці проблеми. Принцип архітектури мікросервісів подібний до принципу Unix: робіть одне і робіть це добре [29]. Кожна мікрослужба має чітко визначені контракти (як правило, RESTful) щодо інших мікросервісів для комунікації та обміну даними. Оскільки мікросервіси можуть бути розгорнуті незалежно один від одного, вони можуть масштабуватися незалежно, їх легко замінити та оновити.

Нарешті, **REST** – це програмний архітектурний стиль Всесвітньої павутини (WWW), який забезпечує узгоджений набір обмежень для проектування вебсервісів у розподіленій системі гіпермедіа, що може призвести до більш ефективної та ремонтпридатної архітектури [30]. Отже, REST – це архітектура, яка частіше використовується для створення API веб-служб (Прикладний програмний інтерфейс), яка може використовуватися будь-яким пристроєм або клієнтом, що реалізує HTTP (протокол передачі гіпертекстових документів). Це робить REST простішим, ніж попередні альтернативи, такі як SOAP (Простий протокол доступу до об'єкта) та XML-RPC (віддалений виклик процедури).

Шина повідомлень (Message bus). Архітектура, орієнтована на обробку повідомлень, передбачає використання орієнтованого на повідомлення підходу проміжного програмного забезпечення для централізації зв'язку між програмними системами або підсистемами таким чином, що деякі системи генерують повідомлення, а багато інших систем споживають ці повідомлення. Проміжне програмне забезпечення, орієнтоване на повідомлення, може

реалізовувати черги повідомлень, публікувати/підписуватись або пересилати повідомлення.

Багаторівнева архітектура розділяє функціональність на окремі сегменти, будучи рівнем, розташованим на фізично окремому комп'ютері. Клієнт-серверні архітектури або 2-рівневі архітектури розділяють систему на дві програми, де клієнт робить запити на сервер. У багатьох випадках сервер є базою даних з логікою додатків, представленою як збережені процедури [27]. Рівні, як правило, є монолітними, оскільки вони реалізують різноманітні функції, які об'єднані в єдиний пакет, розгорнутий на апаратному забезпеченні, попередньо масштабованому для пікових навантажень [31].

Cloud Computing – Multi-tenant SaaS architecture. Хмарні обчислення – це нова модель, яка забезпечує зручний мережевий доступ на вимогу до спільного пулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, програм, платформ та послуг), які можна швидко забезпечити та випустити з мінімальними зусиллями управління або взаємодії постачальника послуг. Постачальники хмарних обчислень, такі як Microsoft, Amazon та Google, пропонують різноманітні обчислювальні послуги, побудовані на основі власної інфраструктури, якими керують спеціальні глобально розподілені центри обробки даних, що забезпечують високу доступність, стійкість та масштабованість. Хмара сприймає всі виконані завдання як «послугу»: Інфраструктура як послуга (IaaS); Платформа як послуга (PaaS); та Програмне забезпечення як послуга (SaaS).

Багатокористувацький SaaS – це архітектурний стиль, заснований на модульному дизайні, в якому один екземпляр програми обслуговує декількох користувачів (орендарів). Орендарям може бути дозволено налаштовувати деякі частини програми для реалізації своїх індивідуальних та різноманітних вимог [32].

2.4 Архітектурні стилі та моделі

Більшість досліджуваних систем описують архітектури програм для медичних інформаційних систем з використанням моделі розгортання (відома як фізична модель) за допомогою архітектурних стилів багаторівневої архітектури (N-Tier), клієнт-серверної архітектури (Client-server) та хмарних обчислень (cloud computing); модель структури (модель реалізації) через компонентноорієнтовану архітектуру (component-based) та багат шарову архітектуру (layered architecture); та комунікаційна модель (модель обробки) через сервісноорієнтовану архітектуру (SOA), шину повідомлень, подійно-орієнтовану архітектуру та архітектуру REST. Архітектура, описана в [33], є трирівневою архітектурою: мережа бездротових датчиків (рівень сприйняття) взаємодіє з набором компонентів, які називаються драйверами (зазвичай називаються рівнями шлюзу), які згодом обмінюються даними через Інтернет з платформою EcoHealth (прикладний рівень).

Модель реалізації архітектури платформи EcoHealth [33] складається з набору модулів (сервіси проміжного програмного забезпечення). Модуль підключення пристроїв інтегрує фізичні пристрої (бездротові датчики тіла) до платформи за допомогою драйверів, які різні виробники розробляють відповідно до визначеного API (API EcoHealth). Ці драйвери можуть отримати доступ до даних шляхом постійного опитування або за запитом після повідомлення. Вони побудовані на принципах REST і засновані на протоколі HTTP (сумісний з парадигмою мережі речей).

Таким чином, датчики тіла являють собою веб-ресурси, які надають доступ, контроль і моніторинг через Інтернет. Модуль обробки даних реєструє дані, що надсилаються до EcoHealth через запити HTTP PUT від драйверів. Модуль зберігання включає реляційну базу даних та файлову систему для зберігання всіх даних, що використовуються в EcoHealth, які можуть скористатися перевагами хмарних інфраструктур зберігання.

Модуль загальних служб складається з набору загальних послуг, таких як безпека та сповіщення (наприклад, механізми сповіщення на основі подій, які

повідомляють про проблеми, симптоми або аномалії вимірюваних життєвих показників).

Нарешті, модуль візуалізації та управління забезпечує веб-інтерфейс для надання зацікавленим сторонам (лікарям, пацієнтам та системним адміністраторам) доступу до основних функціональних можливостей.

Архітектура, описана в [34], також є трирівневою: на борту смартфонів є датчики, а смартфони виступають як шлюзи, які надсилають дані до основного компонента, тобто до клінічного віддаленого сервера. Вид реалізації архітектури складається з наступних компонентів: додаток для смартфона, база даних та вебпрограма. Додаток для смартфона відстежує параметри ходи (рис. 2.4), виявляє епізоди замерзлої ходи, генерує сигнали тривоги у відповідь на них, зберігає ці дані в локальній базі даних і періодично завантажує ці дані в центральну базу даних клінічного віддаленого сервера.

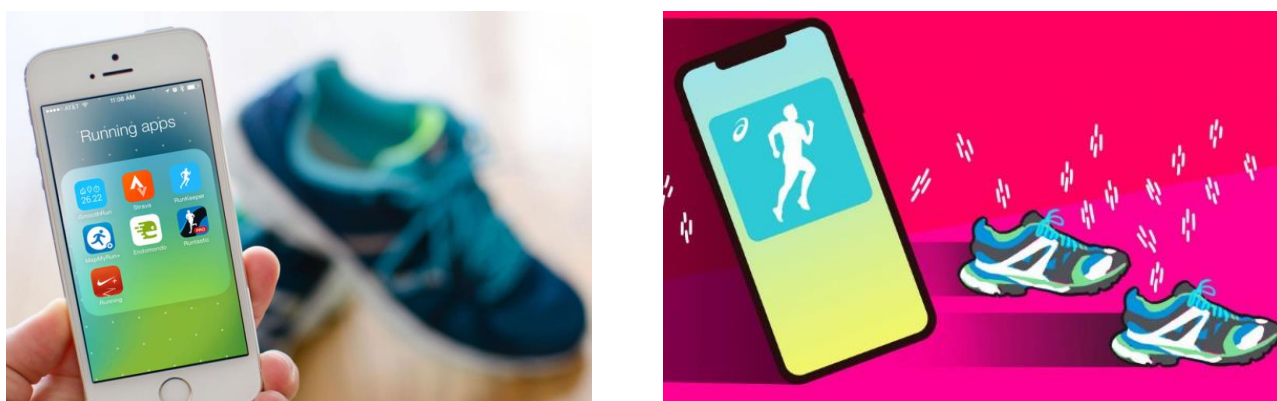


Рисунок 2.4 – Мобільні додатки для бігу

Архітектура, описана в [35], як [36] та [34], представляє три рівні: датчики життєво важливих знаків та їх зв'язок складають рівень сприйняття; розумний мобільний пристрій складає рівень шлюзу, який фіксує життєві показники датчиків і передає дані на медичну платформу; і нарешті, ця медична платформа становить прикладний рівень.

Модель реалізації прикладного рівня має такі модулі: розумний мобільний пристрій, який, крім фіксації життєво важливих показників від датчиків і надсилання цієї інформації на медичну платформу, реалізує систему генерації правил, здатну обробляти проміжні дані та генерувати тривожні повідомлення

направляти до опікунів пацієнта або медичного персоналу; модулі зберігання та обробки даних, які здатні управляти великими обсягами даних; платформа взаємодії та обміну повідомленнями для доставки інформації до всіх зацікавлених сторін за допомогою SMS, голосу та технології PUSH; платформа веб-сайту, що дозволяє отримати інформацію про пацієнта; і шукач інформації, пов'язаної зі здоров'ям, у різних медичних джерелах.

Платформа BodyCloud [37] – це чотирирівнева архітектура. Найнижчий рівень – це рівень сприйняття, який складається з мережі сенсорів тіла, який, у свою чергу, складається з набору сенсорних вузлів, які вимірюють біосигнали та надсилають необроблені або попередньо оброблені дані координатору. Вузли датчиків надають функції, які є більш досконалішими, ніж попередні роботи, такі як стандартний інтерфейс до різноманітних драйверів датчиків, а також налаштування та розширення платформи для підтримки нових фізичних датчиків та сервісів обробки сигналів. Рівень шлюзу призначений для координатора; він управляє вузлами датчиків (наприклад, координатор може повідомити про відкриття нових датчиків), збирає, зберігає та аналізує дані, отримані від датчиків, і діє як шлюз для підключення BSN до віддаленого клінічного сервера. Ці два рівні називаються стороною тіла. Прикладний рівень підтримує розподілену систему реального часу для моніторингу та аналізу потоків даних BSN, а також систему візуалізації результатів аналізу даних за допомогою вдосконаленої графічної звітності. Перший розгортається у хмарі через модель SaaS і називається стороною хмари, тоді як другий розгортається як клієнтська програма HTML / Android і називається стороною перегляду.

Проміжний рівень – це рівень, який складається з трьох підсистем: фреймворк-служби, що складаються із сервісів запитів, за допомогою яких можна отримати інформацію про конкретні параметри, сервіс диспетчера виконання, що відповідає за завантаження / розвантаження та запуск / зупинку компоненти та PRA, служба безпеки для управління дозволами на доступ та шифрування інформації та служба конфігурації для встановлення параметрів на низькому рівні; послуги зв'язку, сформовані з трьох модулів, що дозволяють виявляти ресурси низького рівня, виявляти та впливати на сервіс, а також

Інтернет зв'язок; та сервіси управління знаннями, реалізовані брокером та організатором сервісів, що реалізують API, запропонований розробникам PRA.

Верхній шар – шар агента, де визначені PRA. У роботі [23] пропонується розробка інтерфейсу для автоматизації зчитування біосигналів в системах EMR, описуючи трирівневу архітектуру. Необхідно підкреслити, що автори відносять рівні до шарів. Отже, вони описують наступну декомпозицію архітектури. Інтерфейсний рівень включає станцію зчитування життєвих ознак (додаток, що зчитує біосенсиори та зберігає інформацію у внутрішній пам'яті пристрою, де вона розгорнута), та спеціальну форму життєвих ознак (яка підтримує автоматичне читання біосигналів із читальної станції). Логічний рівень програми відповідає за збір даних із станції зчитування, їх відображення у формі та надсилання їх на рівень даних. Рівень даних позначає існуючу систему EMR.

PRIME [44] пропонує дворівневу архітектуру, що складається з комунікаційного рівня, який використовує накладення, використовуючи пристрої в накладеній мережі, побудованій поверх низькорівневих технологій бездротового зв'язку, та рівень програмування API, який забезпечує абстракції та операції реалізувати програми P-RESTful (Pervasive REST). E-SURAKSHAK [26] складається із сервісно-орієнтованої архітектури (SOA) з трьома рівнями.

Екологічний рівень (еквівалентний рівню сприйняття попередньої роботи) складається з WEID і маршрутизаторів і визначає життєво важливі параметри здоров'я пацієнтів. Рівень обслуговування складається з основних та складних функцій аналізу. Нарешті, рівень управління зберігає та обробляє зібрану інформацію та забезпечує веб-інтерфейс.

Система моніторингу пацієнтів (PMS) [38] також представляє SOA, яка реалізована за допомогою середовища OSGi. OSGi – це ініціатива, орієнтована на взаємодію програм та служб на основі платформи інтеграції компонентів (платформи обслуговування), що забезпечує сервісно-орієнтоване середовище на основі компонентів. Рівень послуг динамічно пов'язує набори – компоненти OSGi – за допомогою моделі публікації-пошуку-прив'язки для простих старих

об'єктів Java. Рівень життєвого циклу реалізує API для встановлення, запуску, зупинки, оновлення та видалення пакетів.

Рівень модулів визначає, як пакет імпортує та експортує код. Рівень безпеки гарантує безпечний доступ. Нарешті, середовище виконання визначає, які методи та класи доступні в конкретній структурі. PMS визначає набір пакетів для безпечного входу в систему, потокового передавання відео, медичного спостереження за пацієнтами, пристроїв для пацієнтів, аудіозв'язку та GPS-навігації.

Додаток ЕКГ для Android [39] (рис. 2.5) реалізує наскрізну архітектуру системи. Необхідно підкреслити, що автори описують трирівневу архітектуру, але посилаються на рівні як шари. Отже, вони описують таку декомпозицію архітектури: апаратний рівень включає мікроконтролери та датчики; прикладний рівень реалізує модель / вигляд / контролер програми Android для взаємодії з користувачами, отримує дані біосигналу з апаратного рівня, зберігає дані в буфері та двійковому файлі та передає файл із пристрою Android на FTP-сервер на хмарному рівні.

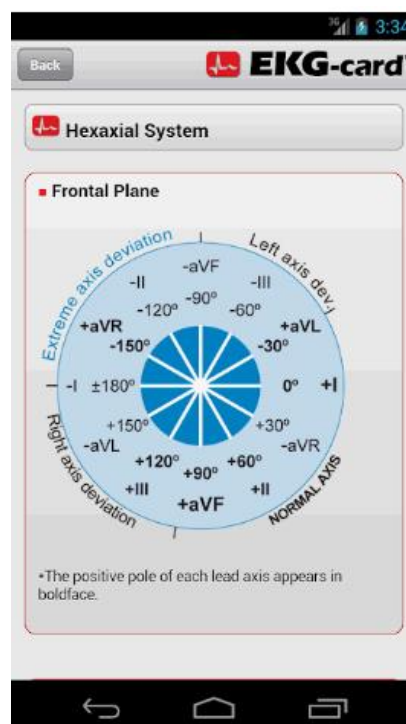
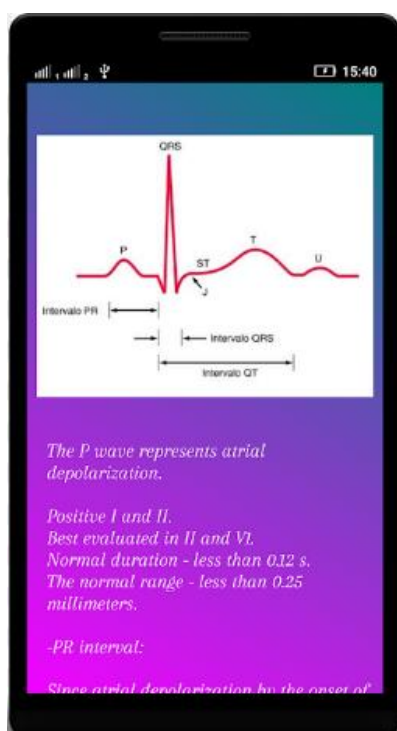




Рисунок 2.5 – Додаток ЕКГ для Android

Система iSenior [31] реалізує такі модулі: портал – модуль використовує інтерфейс, що дозволяє автентифікацію та доступ до iSenior. Портал відображає інформацію про життєво важливі ознаки від приладу для спостереження (Elderly Monitoring Device) за літніми людьми через записи та попередження, забезпечує аналіз історії та відстежує пацієнтів за допомогою Карт Google, конфігурації тривоги та функцій моніторингу; ядро: цей модуль керує зв'язком між віддаленим медичним сервером та EMD; обробка сигналів: цей модуль відповідає за активацію попереджень шляхом виявлення ненормальних умов, надсилання SMS-сповіщень та реалізації алгоритмів обробки необроблених даних [41] також пропонує трирівневу архітектуру, яка складається з модульного прототипу портативного пристрою (рівень сприйняття), який включає модуль обробки (відповідальний за підготовку та обробку сигналу), модуль збору даних, який вимірює параметри, і модуль передачі даних, який відповідає за передачу отриманих даних на шлюз або точку доступу. Шлюз управляє зв'язком; ця пропозиція розробила два протоколи зв'язку – пристрій до шлюзу та шлюз до сервера. Модулі для розпізнавання активності та складного моніторингу пацієнтів впроваджують послуги охорони здоров'я (прикладний рівень). Перший визначає діяльність, яку реалізують пацієнти (наприклад,

ходьба або біг), а другий контролює набір з восьми параметрів (наприклад, ЕКГ, артеріальний тиск, потік повітря та рівень кисню).

VIRTUS [40] пропонує багат шарову архітектуру (layered architecture), на базі SOA, яка складається з: носимих пристроїв, які збирають дані, пов'язані з діяльністю пацієнта; проміжне програмне забезпечення, що дозволяє здійснювати зв'язок між модулями системи; інтерфейс між користувачем, системою та віддаленим центром; а також програмне забезпечення центрального 40 підрозділу або менеджера проєктів, яке відповідає за класифікацію даних та формування щоденних звітів та відгуків для пацієнта. ІМС пропонує подібну архітектуру до VIRTUS [40] з наступними рівнями: рівень збору даних, що складається з вузлів даних, адаптерів та інтерфейсу для різномірних даних з різних джерел; рівень управління даними, який містить модуль розподіленого зберігання файлів (DFS) та модуль розподілених паралельних обчислень (DPC); і сервісний прикладний рівень, який забезпечує візуальну презентацію результатів та відкритий уніфікований API для розробників.

μWoTOP [41] реалізує багат шарову архітектуру (layered architecture): шар екосистеми IoT, який відповідає за адаптери, що визначають потенційні ресурси (датчики та виконавчі механізми), класифікують їх на основі їх топології та повідомляють про них верхнім шарам; рівень проміжного програмного забезпечення Web of Things, метою якого є надання функціональних можливостей, щоб дизайнери та розробники сервісів могли проектувати, розробляти та розгортати розумні простори; і склад ресурсів та рівень організації, який відповідає за розгортання необхідних компонентів, щоб можна було пропонувати сервіси та функціональні можливості. [37] пропонує децентралізовану хмарну архітектуру з трьома рівнями: датчики та виконавчі механізми (рівень сприйняття), рівень шлюзу, що відповідає за зв'язок між управлінням ресурсами та віддаленими одиницями телеметрії (RTU), а також послуги презентації та застосування (рівень додатків). [42] також пропонує багат шарову архітектуру: рівень збору даних відповідає за отримання та обробку даних; рівень зв'язку забезпечує наскрізне підключення, агрегування

вимірних даних з інших пристроїв та перетворення у формат; обробний шар виконує статистичні розрахунки; рівень управління складається з медичної експертної системи, яка пропонує дії на основі даних, що генеруються процесом обробки; а рівень обслуговування забезпечує доступ до кінцевого споживача (лікарні, служби екстреної допомоги, швидкої допомоги та відділення міліції), а також забезпечує аналіз на основі історії хвороби пацієнта.

2.5 Деталі реалізації

Платформа EcoHealth [40] реалізована на мові програмування Java і розгортається на прикладному сервері JBoss. Дані зберігаються в реляційній базі даних MySQL за допомогою специфікацій Java Persistence API (JPA), реалізованих в рамках Hibernate. Дані, якими слід керувати, описуються наступним чином: медичні записи, історичні дані від датчиків тіла, повідомлення, інформація, пов'язана з пацієнтами, та інформація про екстрені служби. Наданий веб-інтерфейс реалізований за допомогою технології JavaServer Faces (JSF). Відповідно до специфікацій авторів, драйвери повинні реалізовувати інтерфейс RESTful для клієнтів (людей чи додатків), зокрема реалізацію RESTEasy для архітектурного стилю REST та API Java для служб RESTfulWeb (JAX-RS). Нарешті, автори використовують специфікації служби автентифікації та авторизації Java (JAAS), реалізовані на сервері JBoss, для забезпечення автентичності, конфіденційності та цілісності користувачів.

В архітектурі, описаній у [34], є нечіткий логічний алгоритм та система нечітких висновків – реалізована за допомогою програми для смартфона – для визначення епізоду замерзлої ходи (FoG) на основі набору параметрів ходи: довжина кроку, крок частота, індекс замерзання та енергія. Додаток для смартфонів був розроблений для операційних систем iOS та Android. Дані зберігаються в реляційній базі даних MySQL на віддаленому клінічному сервері. Інформація, яку слід зберігати, така: особисті дані користувачів веб-додатків, дані моніторингу пацієнтів із програми для смартфона та фармакологічні та реабілітаційні методи лікування, призначені клініцистами

пацієнтам. Особливість [34] полягає у використанні розподіленої нереляційної бази даних для зберігання дуже великих обсягів даних наступним чином.

Розумний мобільний пристрій реалізує безперервне опитування датчиків. Дані зберігаються з використанням файлів, записи яких мають формат кількох елементів (ідентифікатор_сенсора, часовий_знак, значення), і кожен запис займає 10 байт пам'яті. Коли розмір файлу досягає 500 Кб, його закривають та створюють інший файл для збереження найсвіжіших даних. Файли даних, збережені на інтелектуальному пристрої, синхронізуються на сервері в Інтернеті, тобто дані передаються на сервер через захищений розподілений механізм синхронізації файлів P2P (протокол Bit-Torrent Sync). Через велику кількість даних, що підлягають обробці на стороні сервера, [38] використовує базу даних RIAC, яка є розподіленою, масштабованою нереляційною базою даних ключ / значення з відкритим кодом. У RIAC єдиною одиницею зберігання даних є об'єкти, які можна ізолювати за допомогою віртуальних просторів ключів (сегментів).

Отже, дані організовані таким чином: сегмент для користувачів, де ключовим значенням є ідентифікатор, а значенням є об'єкт, що містить дані користувача та список ідентифікаторів датчиків, пов'язаних з цим користувачем, і (II) сегмент для датчиків, де ключове значення формується ідентифікаторами користувача та пристрою, а значення є об'єктом, що містить необхідні дані для ідентифікації останніх даних, переданих одним датчиком. [39] інтегрує хмарні обчислення та BSN.

Хмарна сторона розроблена за допомогою Google App Engine (GAE). Зокрема, BodyCloud використовує API схожих даних GAE для збереження даних потоків, API чергових даних GAE для складеного поділу, що перетворює реалізацію додаткових програм для аналізу даних та інтеграції робочих процесів видобутки сторонніх даних (наприклад, WEKA) та (III) OAuth 2.0 [27] для систем авторизації. Взаємодія між компонентами цього рішення – тілом та хмарою (збір даних), хмарою та засобом перегляду (візуалізація даних) та хмарою та аналітиком (визначення нових служб аналізу даних) - підтримується через веб-службу RESTful, реалізовану за допомогою Restlet Framework.

KASO Middleware [22] полягає у можливості динамічної реконфігурації ресурсів (наприклад, датчиків) залежно від умов навколишнього середовища. З цією метою цей підхід визначає набір послуг управління знаннями (Knowledge Management), які залежать від інформаційної моделі системи, що описує ресурси низького та високого рівня. Дані зберігаються в базах знань (Knowledge Bases), контекстній базі знань (Contextual Knowledge Base) та системній базі знань (System Knowledge Base), які розміщуються на вузлах датчиків та виконавчих механізмів, контекстних вузлах та шлюзах відповідно. Бази знань структуровані через онтологію над RDF / OWL 2, а його послуги наносяться на стандартизовані веб-служби REST. [23] реалізував кіберфізичний інтерфейс для автоматизації зчитування біосигналів у системах EMR.

Реалізація цього інтерфейсу була зосереджена на власній та ліцензованій ERM: Centricity Practice Solution®, яка розгортається на сервері Citrix. Ця робота реалізувала обгортку (або адаптер) для розширення функціональності ERM, тобто для автоматизації генерування EMR від датчиків та зберігання даних у базах даних. Основними особливостями Системи моніторингу пацієнтів (Patient Monitoring System) [36] є орієнтація на сервіси та агрегація сервісів із використанням платформи Eclipse OSGi та архітектури завантажувача класу Java для розгортання служб під час виконання.

PMS включає GPS-навігаційну систему, доступ до якої здійснюється через Інтернет через TCP/IP за допомогою спеціальної мережі. Він підтримує потокове передавання відео через IP-камери з фільтрацією та співвідношенням передачі 1:1 (автори не використовують будь-яке власне програмне забезпечення), моніторинг стану здоров'я пацієнта шляхом попередньої конфігурації OSGi, щоб будь-який пристрій медичного моніторингу надсило дані через спеціальну мережу та відображає результати у вигляді графіку та аудіо зв'язку через VoIP (voice over IP).

Додаток ECG для Android [35] використовує технології Filestream та Filetable для зберігання неструктурованих даних. Він реалізує мікроконтролер, який отримує біосигнали пацієнта і передає їх на мобільний пристрій по бездротовій мережі за допомогою технології Bluetooth. Дані зберігаються у

двійковому файлі, зашифрованому на захищеній цифровій (SD) карті пристрою, після чого пацієнт може завантажувати двійкові файли до централізованої приватної хмари за допомогою захищеного сервера FTPES (через FTP). [41] представляє нейронну мережу для визначення виду діяльності, яку слід реалізувати пацієнтам, наприклад, сидячи, стоячи, ходьба, біг або невизначена діяльність. Відмінними рисами проміжного програмного забезпечення VIRTUS [28] є використання мови програмування Java, використання миттєвих повідомлень XMPP (eXtensible Messaging and Presence Protocol) та використання фреймворку OSGi. Нарешті, [41] використовує обробку в режимі реального часу та технології MapReduce (Apache Spark та Hadoop) для обробки даних та аналізу та обчислення статистичних параметрів.

Сумісність. У [40] автори визначають три рівні взаємодії: «на нижньому рівні необхідно безперешкодно інтегрувати безліч різнорідних фізичних пристроїв між собою», «на проміжному рівні, дані, надані пристроями повинні бути легко доступними в Інтернеті», «на більш високому рівні стандартизована модель програмування може забезпечити прозорість збору та перетворення інформації з пристроїв вимірювання, не вимагаючи якихось конкретних знань щодо особливостей цих фізичних пристроїв та мережного середовища.

Платформи EcoHealth [40], BodyCloud [39], KASO [22] та Activity-as-aService [43] мають справу з трьома рівнями, інкапсулюючи неоднорідність пристроїв у драйвери для зв'язку з API EcoHealth. Дані (канали) надаються пристроями, доступними для Інтернету через протокол HTTP та RESTful API, та структурувати канали за допомогою XML або JSON. Платформа EcoHealth використовує розширену мову розмітки середовищ (EEML) для вирішення питань взаємодії даних між датчиками тіла від різних виробників.

EEML – це мова, заснована на XML, для опису даних датчиків, зібраних з пристрою, побудови, системи або простору, у структурованому вигляді (у цьому випадку життєві показники людського тіла, зібрані з датчиків тіла).

Однак жодна з цих платформ не стосується стандартизації медичних записів, а також реляційні моделі історичних даних детально не описані. [36] та

[26] передбачають лише взаємодію на нижчому та середньому рівнях. У [36] інтеграція датчиків інкапсульована смартфонами – оскільки цей підхід заснований на датчиках вбудованих в смартфони, – і дані датчиків можуть бути доступні через HTTP. У [26] автори визначають взаємодію на фізичному та сервісному рівнях. У першому вони визначають Edge Routers та WEID, пов'язані з пацієнтами, тоді як у другому вони встановлюють послуги, незалежні від контексту або стану інших сервісів. [38] та [35], навпаки, реалізують смарт-мобільний пристрій на основі мікроконтролера з модулем Bluetooth, і, отже, не зосереджуються на підтримці неоднорідності датчиків. Запропоноване рішення в [23] базується на станції зчитування Spot Vital Signs LXi®, яка дозволяє зчитувати життєві показники із обмеженого набору сумісних датчиків. Для збереження взаємодії з іншими рішеннями розроблено механізми, що дозволяють VIRTUS обмінюватися даними з модулями платформ, що використовують інші протоколи зв'язку, ніж XMPP (eXtensible Messaging Presence Protocol), такі як веб-сервіси для SOA. Система [34] інтегрує домашній шлюз у хмарне проміжне програмне забезпечення, дозволяючи тим самим різноманітним пристроям інтегрувати дані з інших платформ охорони здоров'я. Система [16] пропонує інтеграцію із застосуванням стандарту CDA (Архітектура клінічних документів) HL7, оскільки це провідний світовий медичний стандарт ІКТ, який передбачається для семантичної сумісності медичних даних.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ОПИС РОБОТИ СИСТЕМИ

3.1 СУБД MongoDB та Redis

Основною задачею роботи є на основі порівняльного аналізу двох популярних СУБД MongoDB та Redis визначимо кращу для розширення функціоналу існуючої. Надалі створимо новий сервіс для забезпечення комунікації в реальному часі через вебсокет з'єднання для медичної інформаційної системи HELSI.

Для реалізації серверного застосунку з RESTful архітектурою було обрано мову C# 8.0 і фреймворк ASP.Net Core 5.0.

У якості бази даних обрано СУБД MySQL.

C# – об'єктно-орієнтована мова програмування. C# відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML [4].

ASP.NET Core – вільно розповсюджений крос-платформенний фреймворк для створення веб-додатків з відкритим вихідним кодом. Дана платформа розроблена компанією Майкрософт і має більшу швидкодію порівняно з ASP.NET. Має модульну структуру та сумісний з такими операційними системами як Windows, Linux та macOS. Незважаючи на те, що це новий фреймворк, побудований на новому веб-стеці, він має високу сумісність з концепціями з ASP.NET. Додатки ASP.NET Core підтримують паралельне управління версіями, в яких є різні програми, які працюють на одночасних комп'ютерах, і можуть орієнтуватися на різні версії ASP.NET Core. Це було неможливо в попередній версії ASP.NET.

У якості СУБД було обрано MySQL, його було порівняно з MongoDB. MySQL – вільна система управління базами даних, яка є власністю компанії

Sun Microsystems. СУБД розповсюджується по ліцензії GNU General Public License та під власною комерційною ліцензією. За умовами GPL, якщо якась програма вміщує вихідні коди MySQL, то вона також повинна розповсюджуватися за ліцензією GPL. Останнє може не відповідати планам розробників, які не бажають відкривати вихідні коди своїх програм. Для таких випадків передбачена комерційна ліцензія, яка, крім того, забезпечує якісну сервісну підтримку. Крім того, компанія розробляє різноманітну функціональність на замовлення ліцензійних користувачів. Завдяки одній із таких розробок ще в самих перших версіях з'явився механізм реплікації. MySQL переважно використовується для нескладних інформаційних систем. У більшості випадків MySQL використовується як сервер, до якого звертаються локальні та віддалені клієнти, хоча до дистрибутиву також входить бібліотека внутрішнього сервера, яка дозволяє включати MySQL в автономні програми.

MySQL підтримує багато типів таблиць, вона є мультиплатформенною СУБД. Завдяки широким можливостям, а також великій швидкодії MySQL часто використовується в якості СУБД, що забезпечує роботу динамічного Web-сайту в мережі Інтернет.

3.2 Порівняння реляційних і нереляційних баз даних

Реляційні бази даних використовують структурований мову запитів (Structured Query Language, SQL) для визначення і обробки даних. З одного боку, це відкриває великі можливості для розробки: SQL один з найбільш гнучких і поширених мов запитів, так що його вибір дозволяє мінімізувати ряд ризиків, і буде особливо до речі, якщо має бути робота з комплексними запитамі. З іншого боку, в SQL є ряд обмежень. Побудова запитів на цій мові зобов'язує визначати структуру даних, подальша зміна структури даних може бути згубним для всієї системи.

Нереляційні бази даних, в свою чергу, пропонують динамічну структуру даних, які можуть зберігатися кількома способами: орієнтовано по колонках,

документо-орієнтовано, в вигляді графів або на основі пар «ключ-значення». Така гнучкість означає наступне:

- можливо створювати документи, не ставлячи їх структуру заздалегідь;
- кожен документ може мати власну структуру;
- у кожній базі даних може бути власний синтаксис;
- можливо додавати поля прямо під час роботи з даними.

Масштабованість. У більшості випадків SQL бази даних вертикально масштабовані, тобто можливо збільшувати навантаження на окремо взятий сервер, нарощуючи потужність центральних процесорів, обсяги ОЗУ або системи зберігання даних. А NoSQL бази даних горизонтально масштабовані. Це означає, що можливо збільшувати трафік, розподіляючи його або додаючи більше серверів до вашої СУБД. У другому випадку, система може стати більша і потужніша.

Зробивши вибір NoSQL бази даних віддається перевага для великих або постійно мінливих структур даних.

Структура. У реляційних СУБД дані представлені у вигляді таблиць, в той час як в нереляційних – у вигляді документів, пар «ключ-значення», графів або wide-column сховищ. Це робить SQL бази даних кращим вибором для додатків, які передбачають транзакції з декількома записами, як, наприклад, система облікових записів або для застарілих систем, що були побудовані для реляційних структур. З діаграми на рис. 3.1 видно, що нереляційні бази даних, на прикладі MongoDB, значно повільніші при оновленні даних з індексом навіть на невеликих кількостях.



Рисунок 3.1 – Порівняння швидкості оновлення даних з індексом у СУБД MySQL і MongoDB

В основному в веб-додатках найпоширенішою операцією є читання даних. Тобто слід порівняти швидкість вибірки даних (рис. 3.2) .

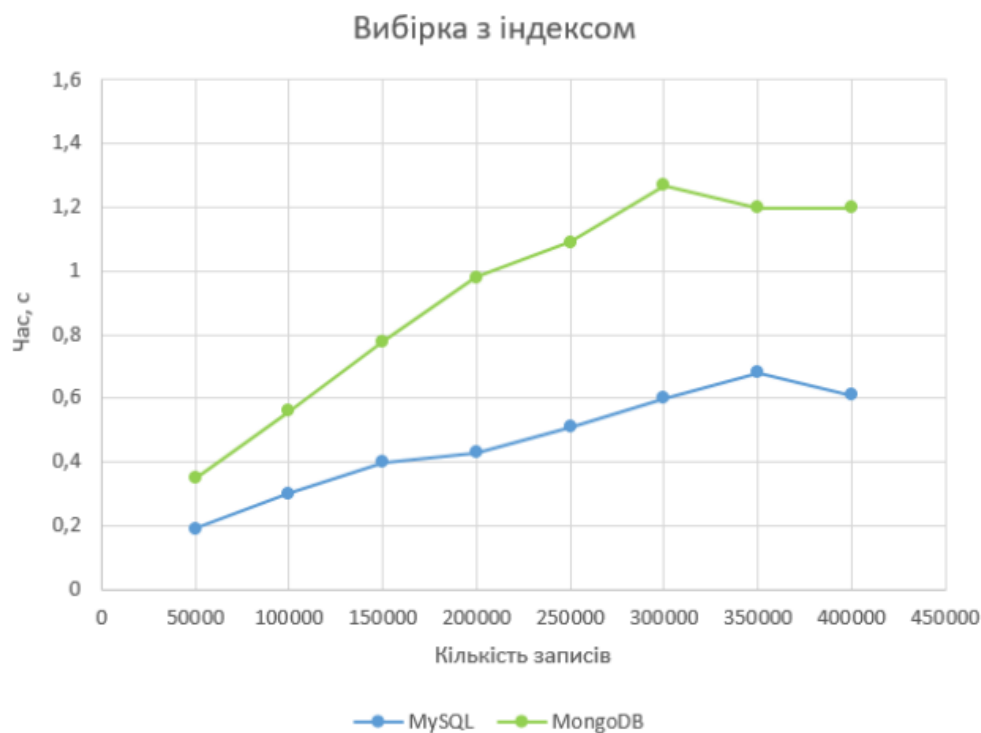


Рисунок 3.2 – Порівняння швидкості вибірки даних з індексом у СУБД MySQL і MongoDB

Отже, виходячи з результатів наведених вище порівнянь було обрано СУБД MySQL через швидкодію, велику розповсюдженість, широку підтримку і високу сумісність з об'єктно-орієнтованими мовами програмування.

3.3 Створення інтерфейсу користувача

Для розробки клієнтської частини використовувались HTML5 SASS, JavaScript, React та Redux. HTML (Hyper Text Markup Language) – це стандартна мова, яка призначена для створення гіпертекстових документів для середовища web сайтів. HTML документи можливо переглядати на різних пристроях та браузерів. Створений HTML документ браузер може інтерпретувати для виділення різних елементів та первинної їх обробки. Саме використання HTML дозволяє виконувати форматування документів для їх показу з використанням шрифтів, ліній та інших графічних елементів на будь-якому програмному забезпеченні та для перегляду web-сторінок.

Для виконання стилізування HTML документів використовуються каскадні таблиці стилів (CSS) – це технологія описування зовнішнього вигляду документа, який написаний мовою гіпертекстової розмітки HTML. CSS використовується авторами веб-сторінок для вказування кольорів, шрифтів, розташування блоків та інших аспектів представлення документа. Основною метою CSS є розділення вмісту (написаного на HTML) та подання документа (написаного на CSS). Цей поділ документа збільшує доступність документа, надає велику гнучкість та можливість виконувати управління його показу, а також зменшити складність та повторюваність в структурі самого документа.

Sass (Syntactically Awesome Stylesheets) – модуль, що входить в Haml. Sass – це метамова на основі CSS, призначена для збільшення рівня абстракції CSS коду та спрощення файлів каскадних таблиць стилів. Мова Sass має два синтаксису: sass – відрізняється відсутністю фігурних дужок, в ньому вкладені елементи реалізовані за допомогою відступів; SCSS (Sassy CSS) – використовує фігурні дужки, як і сам CSS.

Для створення інтерфейсу користувача буде використана технологія ReactJS – це бібліотека JavaScript, яка використовується для створення користувацьких інтерфейсів. З самого початку React був призначений для веб-сайтів, проте пізніше з'явилась платформа React Native, яка була призначена для створення додатків під мобільні пристрої. React представляється ідеальним інструментом для створення масштабних веб-сайтів, особливо в тих ситуаціях, коли веб-сайт має лише одну сторінку (landing page). Вся структура веб-сторінки може бути представлена завдяки допомозі DOM – організації HTML елементів, якими ми можемо маніпулювати, змінювати, видаляти або додавати нові.

Для самої взаємодії з DOM застосовується мова JavaScript. Однак коли ми намагаємося маніпулювати HTML-елементами, то можемо зменшити продуктивність веб-сторінки, особливо коли змінюємо велику кількість елементів. Саме для розв'язання проблеми продуктивності можна використати концепцію віртуального DOM.

Віртуальний DOM – це копія звичайного DOM. Тому React працює виключно з віртуальним DOM, а не звичайним. Якщо нам необхідно змінити елементи веб-сторінки, то зміни спочатку задаються в віртуальний DOM. Після цього новий стан віртуального DOM зрівнюється з поточним станом. Якщо ці стани відрізняються, то React виконує мінімальну кількість маніпуляцій, які необхідні для показу реального DOM до нового стану і виконує їх.

У результаті така схема взаємодії з елементами веб-сторінки працює набагато швидше та ефективніше, у порівнянні з JavaScript з DOM безпосередньо.

Redux – бібліотека для JavaScript з відкритим вихідним кодом, призначена для управління станом додаток. Частіше використовується в зв'язці з React або Angular для розробки клієнтської частини. Містить ряд інструментів, що дозволяють значно спростити передачу даних сховища через контекст. Для вирішення завдань проєктованої системи було проаналізовано декілька якісних наявних додатків з операціями та можливостями ведення сімейного бюджету та виділено функціонал, який найкраще підходить для проєктованої системи, не

ускладнює роботу користувачу та не зменшує продуктивність всього веб-сайту (табл. 3.1).

Таблиця 3.1 – Функціонал веб-додатку «Медична інформаційна система»

Назва функціоналу	Опис функціоналу
Авторизація користувача	Для можливості ідентифікації користувачів та збереженням їх даних на сервері кожен користувач повинен пройти авторизації та підтвердити свої реєстраційні дані.
Реєстрація	Функція яка дозволяє створювати облікові записи для кожного користувача з розділенням по ролям.
Створення медичного закладу	Функція створення медичного закладу дозволяє керівнику зареєструвати свій заклад.
Додавання лікарів до медичного закладу	Функція додавання лікарів до медичного закладу дозволяє керівнику зареєструвати свій заклад.
Редагування медичного закладу	Функція редагування гаманця дозволяє проводити зміни в назві адресі та інших властивостях медичного закладу.
Пошук лікарів	Функція пошуку лікарів дозволяє пацієнтам знаходити лікарів за прізвищем, а також фільтрувати результати.
Назва функціоналу	Опис функціоналу
Запис на прийом до лікаря	Пацієнт може записатись на прийом до лікаря
Скасування прийому	Пацієнт або лікар можуть скасувати запис на прийом
Ведення прийому	Лікар може провести прийом з пацієнтом який був записаний до нього. В рамках ведення прийому лікар може додавати причини, діагнози, послуги з кодуванням ICP2

Схема бази даних (рис. 3.3):

- Користувачі – таблиця для зберігання облікових записів користувачів, загальних даних про них (ПІБ, стать, дата народження, телефон, можна доповнити);
- Організації – таблиця для зберігання інформації про організації, їх адресу і код ЄДРПОУ
- Прийоми – таблиця для зберігання інформації про прийоми зі зв'язком з таблицею користувачів
- Довідник ICP2 – таблиця для зберігання довідника ICP2 кодів

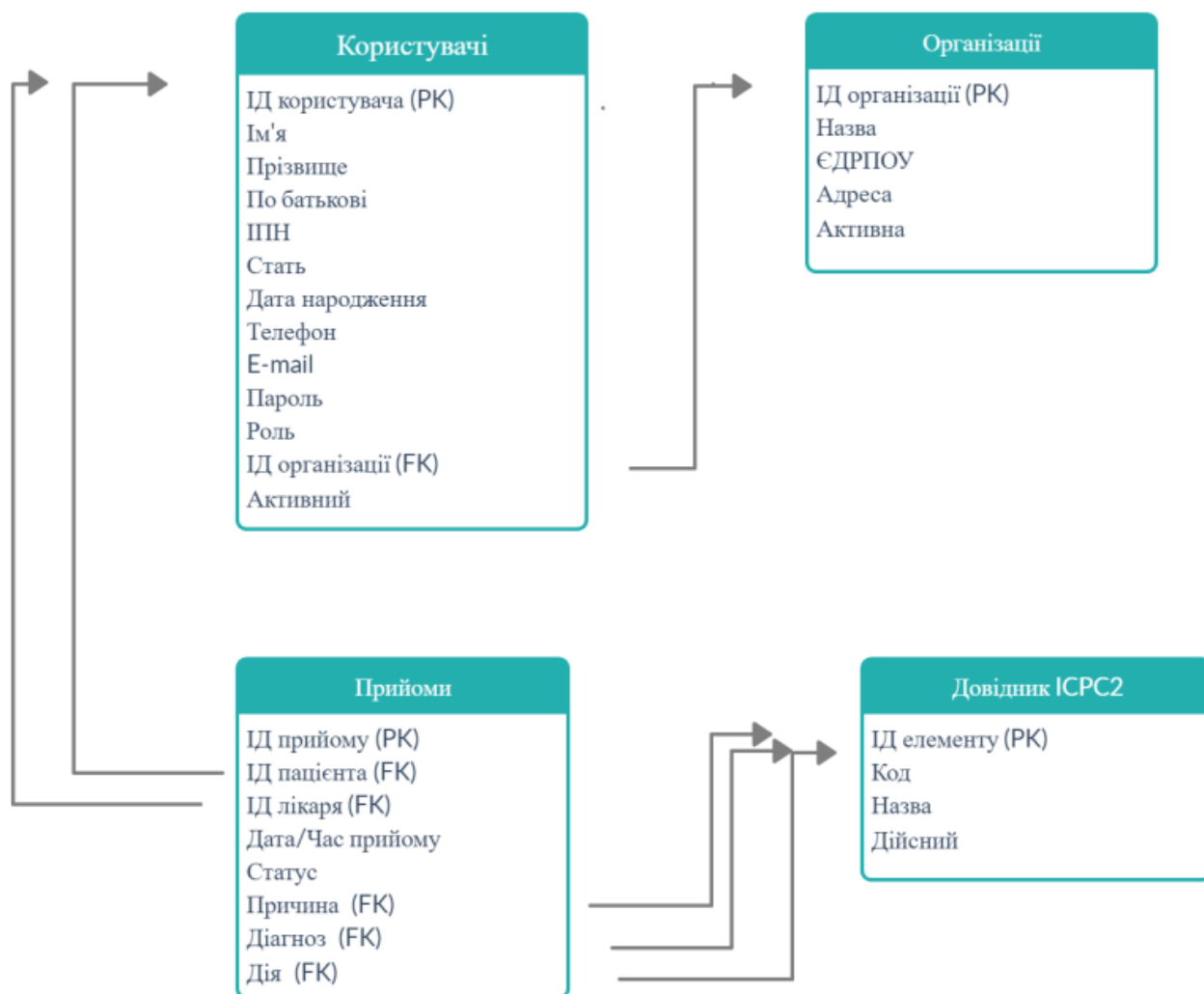


Рисунок 3.3 – Схема бази даних

Вся інформація складається з набору однотипних записів розташованих один за одним. Дані в цих таблицях можливо додавати, видаляти або змінювати. Також в кожній таблиці є набір іменованих полів, які зберігають корисну інформацію починаючи від імені користувача та закінчуючи довідником ICPC2 кодів.

Вікно запису прийому

На рисунку 3.4 зображено Usecase діаграму. Користувачами системи є пацієнт. Пацієнту доступний перегляд лікарів, пошук конкретного лікаря, після якої можливий перегляд сторінки лікаря, пошук по району, керування рейтингом лікарів.

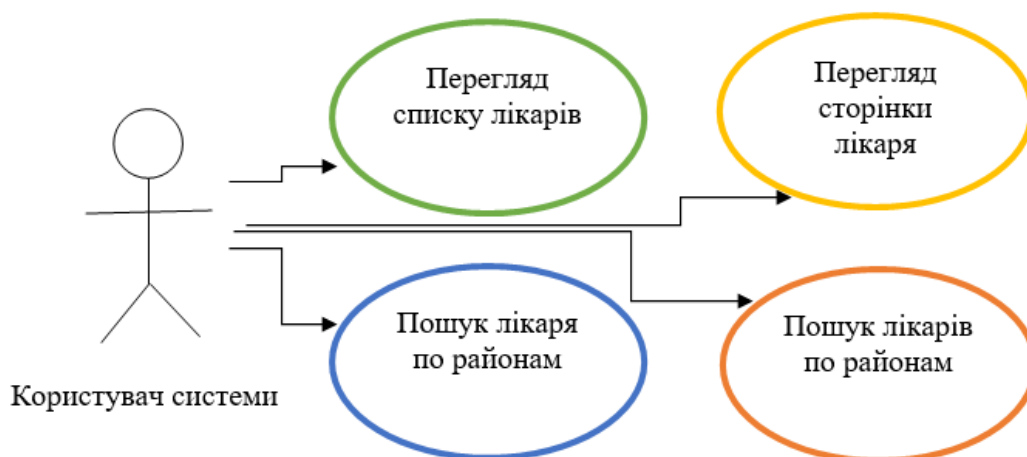


Рисунок 3.4 – Usecase діаграму

Після входу на сайт, користувач бачить головну сторінку сайту, де є варіант перейти до списку лікарів, перейти до форми пошуку по району чи перейти до форми пошуку конкретного лікаря (рис. 3.5).

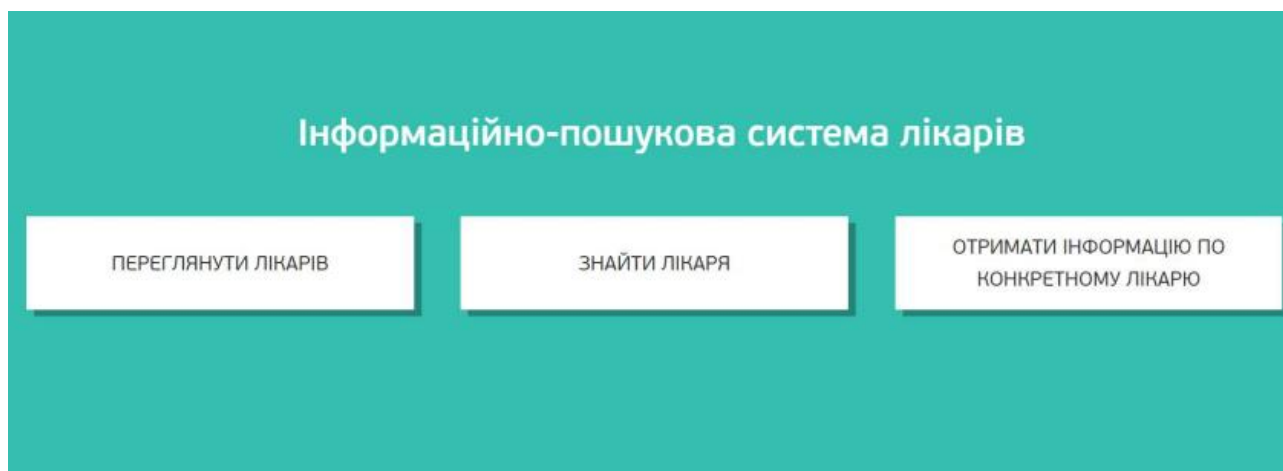


Рисунок 3.5 – Головна сторінка системи

Відповідно, при натисканні на кнопку «Переглянути лікарів» користувач переходить на сторінку всіх лікарів. На сторінці всіх лікарів, користувач може виконувати такі дії:

1. Переглядати список лікарів.
2. Фільтрувати та сортувати список відповідно до конкретних вимог.
3. Вибирати пріоритети критеріїв для розрахунку рейтингу.
4. Шукати конкретного лікаря по ініціалам.

5. Перейти на сторінку лікаря.

Картка лікаря, відображає основну інформацію про сімейного лікаря для користувача. На сторінці всіх лікарів, користувачу доступний список всіх лікарів, можливо створити рейтингування.

Користувачу також доступний попередній перегляд кожного з лікарів. Показники на картці лікаря:

- ініціали лікаря;
- фото лікаря;
- місце роботи лікаря (лікарня, поліклініка, тощо);
- лікарська категорія;
- дані про кількість підписаних декларацій з пацієнтами;
- відносний рейтинг лікаря (у балах);
- кнопка переходу на сторінку лікаря тощо.

На рисунку 3.6 зображена схема використання методу у даній системі.

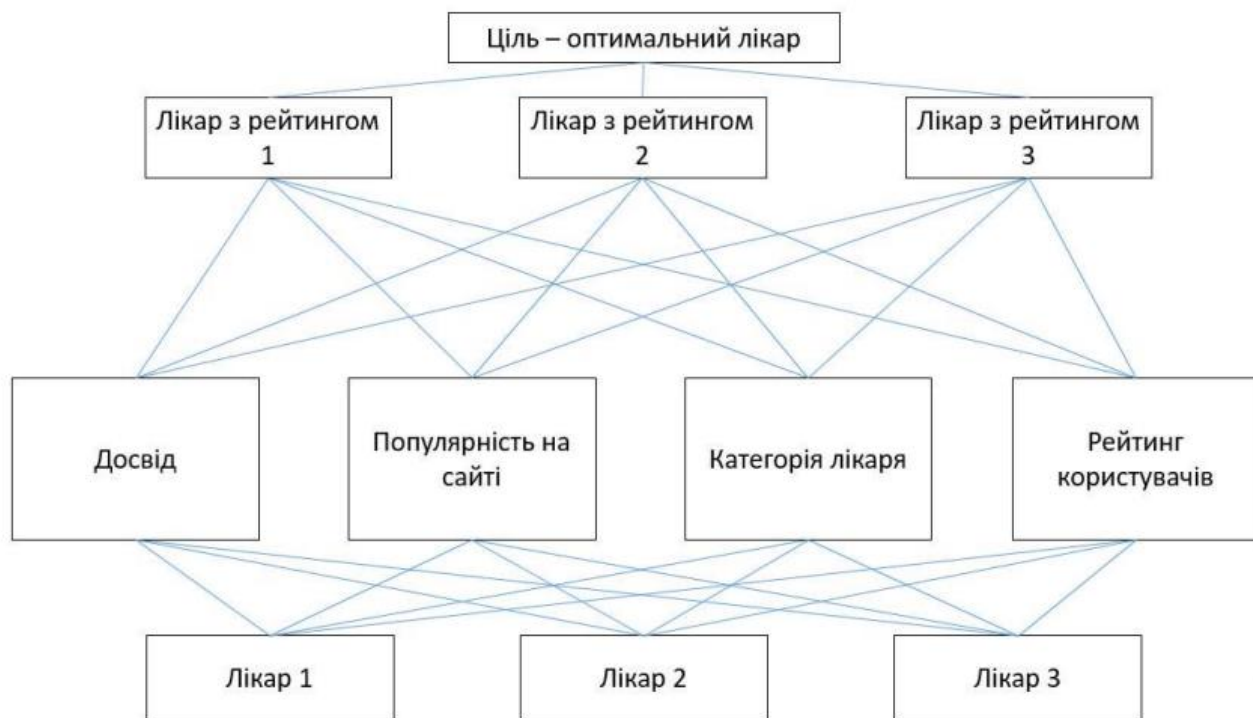


Рисунок 3.6 – Схема використання методу в системі

3.4 Оцінювання стану пацієнта на прикладі серцево-судинної системи

Для оцінювання стану серцево-судинної системи (ССС) необхідно використовувати методику, що поєднує в собі не лише засоби дослідження та обробку результатів, а й безпосередньо програмне забезпечення, що служить аспектом впливу на подальший діагноз. На рисунку 3.7 зображено розробку апаратно-програмного комплексу, що складається з трьох етапів: перший етап – збір персональних даних, другий – збір біологічних показників, третій – аналіз та надання рекомендацій.

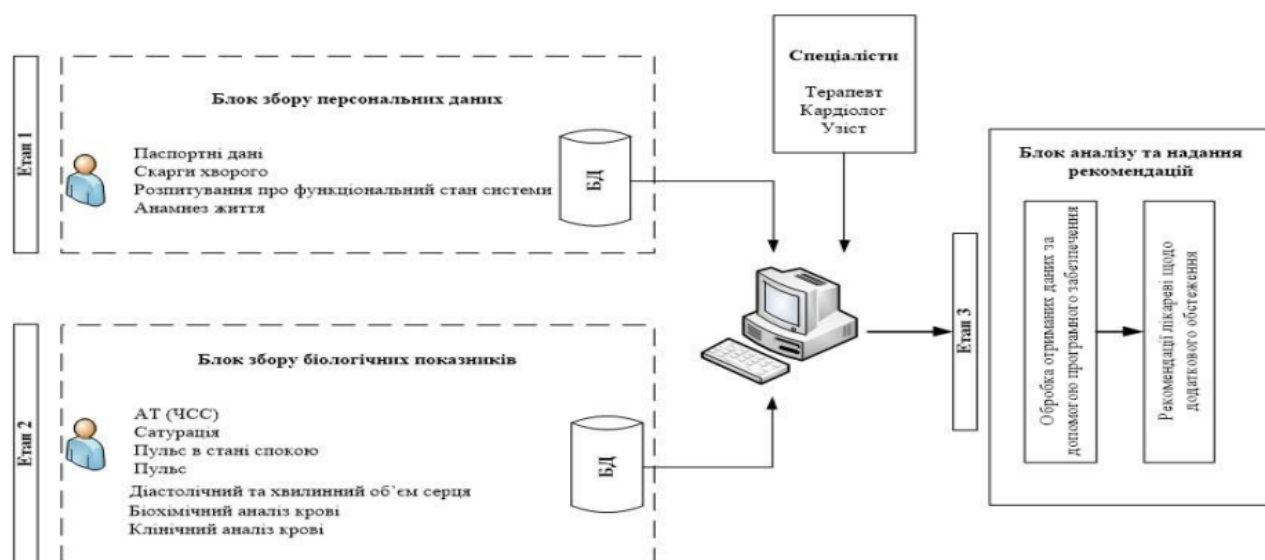


Рисунок 3.7 – Розробка апаратно-програмного комплексу для оцінювання стану пацієнта на прикладі ССС

На рисунку 3.8 зображено приклад надання рекомендацій, якщо визначеним діагнозом являється – серцева недостатність, а ймовірність – 98% та ступінь тяжкості – тяжка

Діагноз: Серцева недостатність

Ймовірність діагнозу: 98%

Ступінь тяжкості захворювання Sars-Covid-19: Тяжка

Рекомендації для лікаря:

Дотримання основних принципів, що стосуються способу життя, викладених в діючих рекомендаціях.

Планове відвідування медичних установ, соціальних структур або планову госпіталізацію під час пандемії COVID-19 доцільно відкласти.

Проведення планової вакцинації та санації порожнини рота недоцільно в зв'язку з високим ризиком контамінації COVID-19 в медичних установах. Хворим, які перебувають в листі очікування трансплантації серця, повинна бути забезпечена можливість телефонного контакту з лікарем.

Рисунок 3.8 – Рекомендації для лікаря щодо хвороби серцевої недостатності

Відповідно до біологічних значень визначаються не лише додаткові дослідження, а й додаткове визначення інших показників для дослідження порушення ритму серця. В результаті отримання рекомендацій полегшується процес дослідження серцево-судинної системи.

3.5 Елементи системи моніторингу та організації мережі передачі даних

Система моніторингу включає:

- мережу бездротових датчиків та/або пов'язаних смартфонів із встановленим мобільним додатком запропонованої системи, які використовуються для взаємодії з бездротовими датчиками та для обробки результатів моніторингу стану клієнтів;
- серверне програмне забезпечення для збору рівня кисню в крові в реальному часі, частоти пульсу та частоти дихання 24 години на добу від спостережуваних та важкохворих клієнтів, забезпечуючи захист персональних даних;

- мобільні програми для моніторингу функціонального стану клієнтів та попередження керівника про небезпечно низький рівень показників [64][65][66].

Структура цілодобової системи дистанційного моніторингу наведена на рисунку 3.9.

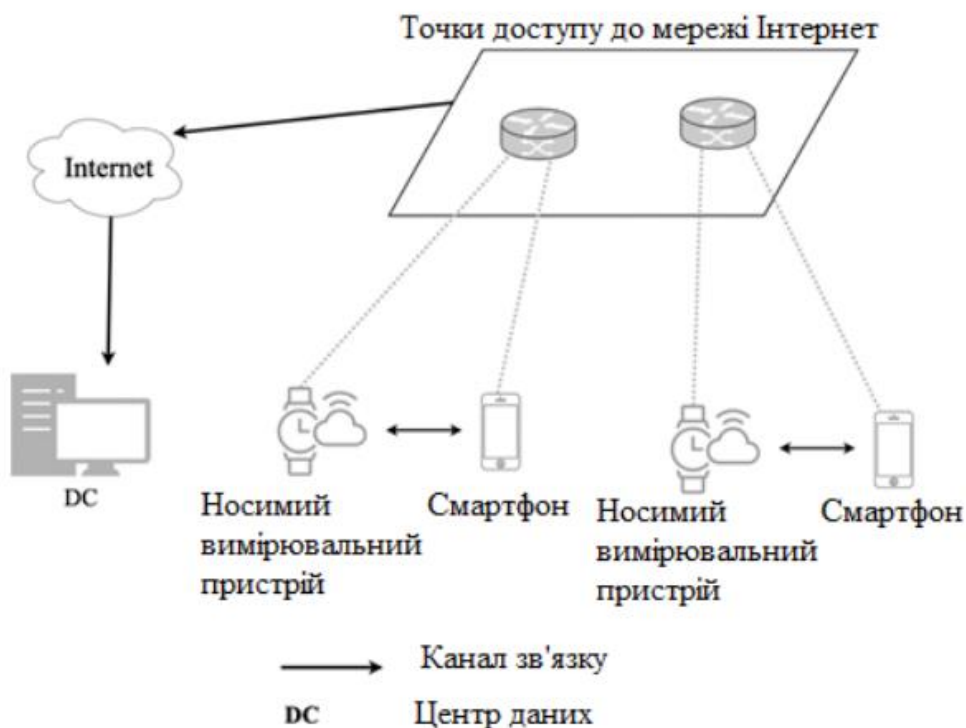


Рисунок 3.9 – Структура системи цілодобового дистанційного моніторингу

3.6 Архітектурно-інформаційна модель обробки медичної інформації

Згідно зазначеному вище, пропонується наступна структура медичної інформаційної системи (рис. 3.10).

З даної структури можна виділити архітектурно-інформаційну модель обробки інформації, яка складається з наступних елементів:

- шаблон спілкування;
- API Gateway (шлюз);
- сервіс запису даних та оперативної БД;
- сервіс автентифікації та контролю доступу;
- сервіс збору персональних даних та побудови зв'язків між користувачами та БД користувачів;

- сервіс прийняття рішень роботи мікросервісів та БД для черги процесів;
- сервіс зберігання та сповіщень про події та БД подій;
- сервіс виявлення критичних показників;
- сервіс класифікації пацієнтів та діагностики захворювань та БД кластеризованих юзерів;
- сервіс обробки даних для зберігання та БД зберігання оброблених актуальних даних;
- сервіс агрегації даних для подальшої візуалізації;
- сервіс збереження та доступу архівних даних та архівного сховища.

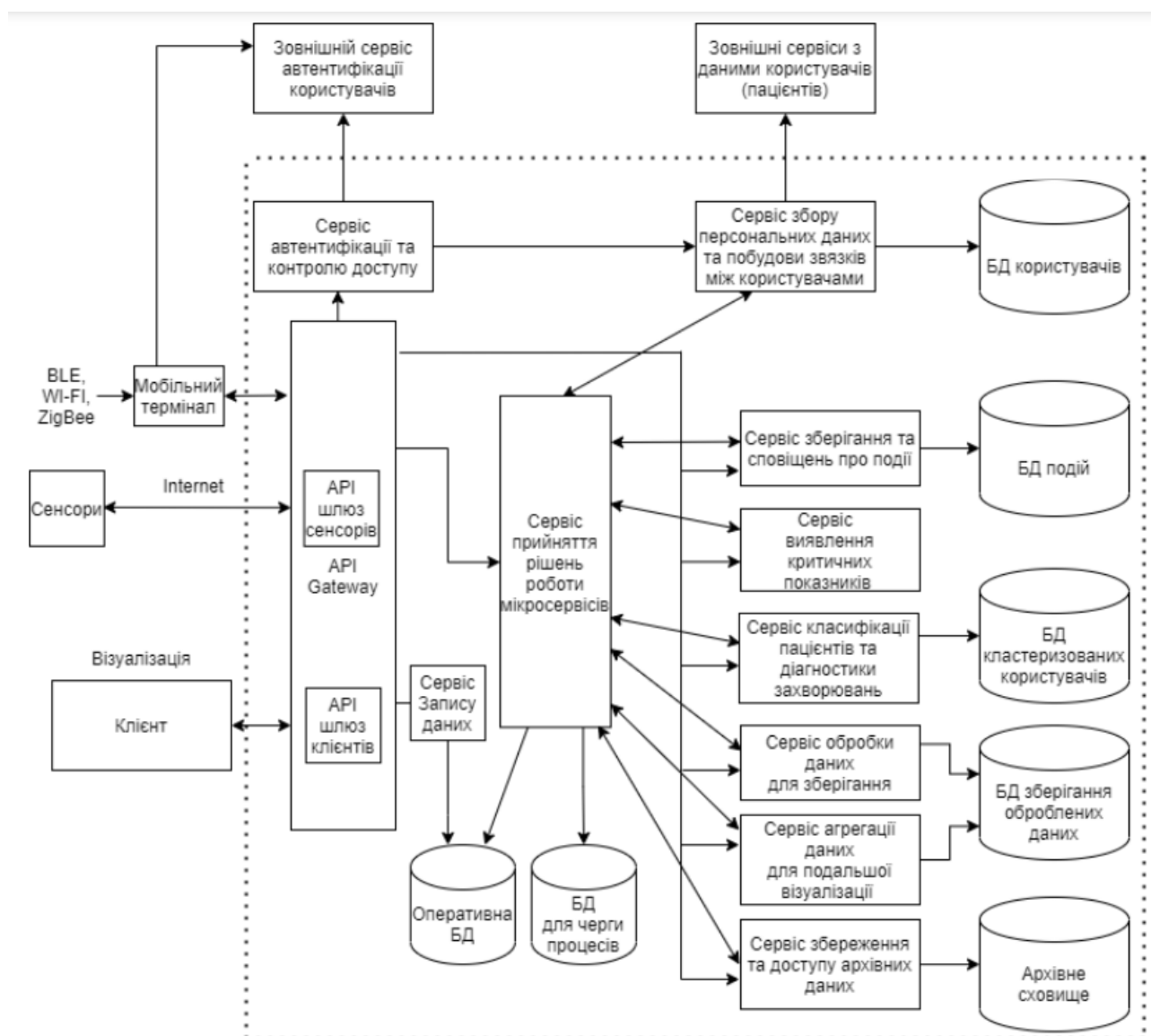


Рисунок 3.10 – Запропонована структура медичної інформаційної системи

Принцип роботи архітектурно-інформаційної моделі обробки інформації виглядає наступним чином. Життєві показники людини поступають на API шлюз. Після успішної автентифікації та авторизації цього запиту, дані записуються в оперативну базу даних через сервіс запису даних, а також інформується сервіс прийняття рішень роботи мікросервісів про надходження нових необроблених даних життєвих показників певної людини. Таким чином, джерело надходження даних отримує відповідь, що дані прийнято до обробки. Після цього, сервіс прийняття рішень роботи мікросервісів формує чергу процесів, які повинні виконати сервіси обробки інформації. Ця черга створюється в спеціальній базі даних для черги процесів.

Згідно вищезгаданої черги, мікросервіси виконують свою функцію обробки необроблених даних для оцінювання стану людини:

- виявлення критичних показників;
- класифікацію пацієнтів та діагностику захворювань;
- обробку даних для зберігання;
- збереження архівних даних.

Коли всі ці процеси виконуються, необроблені дані видаляються з оперативної бази даних. Процес отримання даних для візуалізації виглядає наступним чином. Клієнт виконує запит на конкретну кінцеву точку API шлюзу, де запит та користувач проходять процес автентифікації та авторизації. За умов, що користувач автентифікований та має право на отримання запитуваної інформації, API шлюз вступає до потрібного мікросервісу, який відповідає за запитувані дані, через сервіс прийняття рішень роботи мікросервісів. Дані надходять у відповідь запиту клієнта, де відбувається подальша візуалізація.

Коли проектуємо та створюємо великі або складні додатки на основі мікросервісу з кількома клієнтськими програмами, хорошим підходом може бути API-шлюз. Цей шаблон є послугою, яка забезпечує єдину точку входу для певних груп мікросервісів. Це схоже на візерунок "Фасад" з об'єктноорієнтованого дизайну, але в цьому випадку це частина розподіленої системи. Шаблон шлюзу API також іноді називають "серверним інтерфейсом"

(BFF), оскільки ви створюєте його, думаючи про потреби клієнтської програми. Тому шлюз API знаходиться між клієнтськими програмами та мікросервісами. Він діє як зворотний проксі-сервер, що спрямовує запити від клієнтів до служб. Він також може надавати інші наскрізні функції, такі як аутентифікація, кеш.

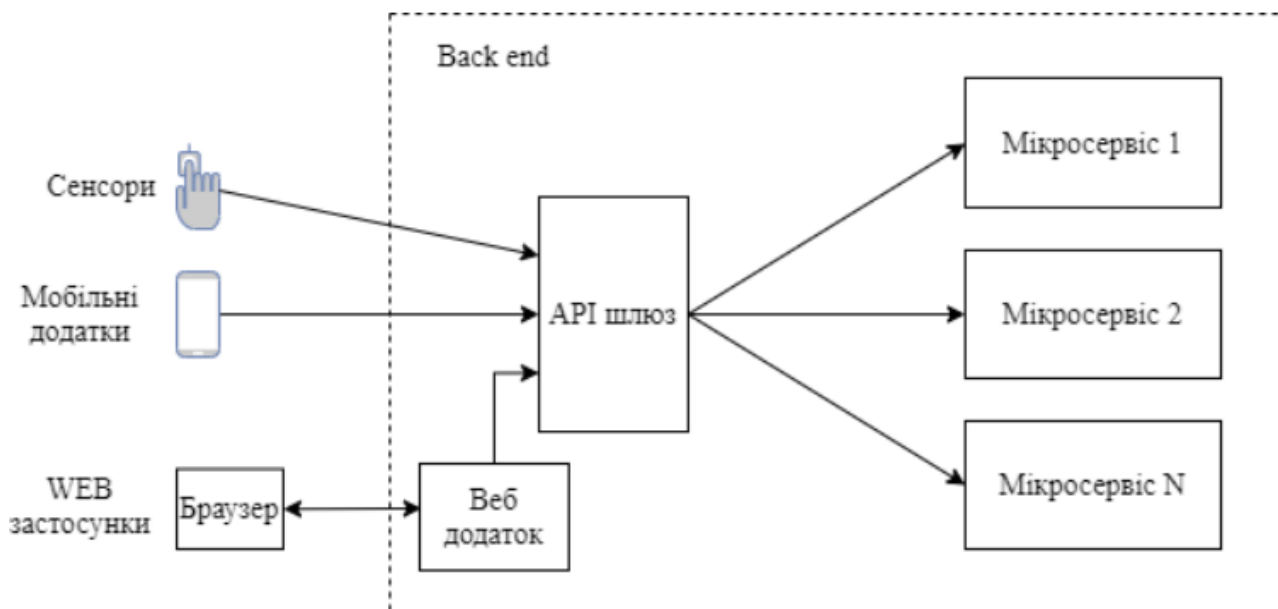


Рисунок 3.11 – Мікросервісна архітектура з зв'язком між клієнтом та мікросервісами через API шлюз.

Висновки до розділу 3

Запропонована реалізація та подано опис роботи системи, у якій є можливим модуль запису в поліклініку чи амбулаторію, вибір лікаря і запис до нього на прийом. Також запропоновано модуль оцінювання стану пацієнта на прикладі серцево-судинної системи, описано елементи системи моніторингу та організації мережі передачі даних, подано архітектурно-інформаційну модель обробки медичної інформації

ВИСНОВКИ

Інформатизація у сфері охорони здоров'я є актуальною в період реформ медичної сфери. Все більше аспектів надання медичних послуг потребують інформаційно-технічного забезпечення. Системне забезпечення для пошуку лікаря відносно потреб користувача є надзвичайно актуальним. Важливим фактором системи має бути об'єктивна оцінка лікарів, за якою користувач міг би обрати персонального лікаря, а також забезпечити дистанційний моніторинг за користувачем

У результаті проведеної роботи були виконані такі задачі:

- досліджено теоретичні основи побудови програмної системи «Медична інформаційна система для поліклініки та амбулаторій»;
- проведено аналіз необхідних технічних рішень для зручної побудови системи;
- запропоновано структуру систему на прикладі «Медична інформаційна система для поліклініки та амбулаторій» з можливістю передачі телемедичних даних;
- спроектувано модуль вибору лікаря;
- спроектувати модуль дистанційного спостереження за пацієнтами.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Севостьянов, І. В. Теорія технічних систем : підручник / Севостьянов І. В. – Вінниця : ВНТУ, 2014. – 181 с.
2. Технічні системи та їх робочі процеси. Режим доступу:
<https://elearn.nubip.edu.ua/mod/book/view.php?id=384918&chapterid=188350>
3. Microsoft Docs Microsoft Docs: веб-сайт. Режим доступу:
<https://docs.microsoft.com/enus/aspnet/core/security/authorization/policies?view=aspnetcore-5.0>
4. Панасюк О.І., Плєскач В.Л. Особливості побудови медичної інформаційної системи для поліклініки [Електронний ресурс]. Режим доступу:
https://er.knutd.edu.ua/bitstream/123456789/19933/1/SEIS_mono_2021_P061-089.pdf
- 5/. React 2020. React: website. Режим доступу: <https://uk.reactjs.org/>
- 6/ 1. Медичні інформаційні системи – впроваджуємо у вашому медичному закладі. URL: <https://www.medsprava.com.ua/article/544-qqq-17-m4-10-04-2017--medichnnformatsyn-sistemi-vprovadjumo-u-vashomumedichnomu-zaklad>
7. Дячук С. Ф. Медичні інформаційні системи /С. Ф. Дячук, А.Ю. Фесина, Д.Б. Ямщіков [Електронний ресурс]. Режим доступу:
https://elartu.tntu.edu.ua/bitstream/lib/23056/2/CAZST_2017v2_Dyachuk_S_F-Medical_information_systems_62-63.pdf
8. Про державні фінансові гарантії медичного обслуговування населення : Закон України від 19.10.2017 № 2168-VIII. Режим доступу:
<https://zakon.rada.gov.ua/laws/show/2168-19>
9. Деякі питання електронної системи охорони здоров'я: Постанова Кабінету Міністрів України від 25.04.2018 №411. Режим доступу:
<https://zakon.rada.gov.ua/laws/show/411-2018-%D0%BF>
10. Деякі питання електронної взаємодії державних електронних інформаційних ресурсів : Постанова Кабінету Міністрів України від 08.09.2016 № 606. Режим доступу:

<https://zakon.rada.gov.ua/laws/show/606-2016-%D0%BF/ed20160908#n17>

11. Про затвердження Порядку проведення пілотного проекту з продажу об'єктів малої приватизації із застосуванням електронної торгової системи “ProZorro.Продажі”: Наказ фонду державного майна від 19.07.2017 №1143. URL : <https://zakon.rada.gov.ua/laws/show/z1095-17/ed20170719#n29>

12. Про затвердження Порядку обміну електронними документами з контролюючими органами: наказ Міністерства фінансів України від 06.06.2017 № 557. URL : <https://zakon.rada.gov.ua/laws/show/z0959-17/ed20170606#n29>

13. Криничко Л.Р. Малігон Ю.М. Нормативно-правове забезпечення реформування системи охорони здоров'я в Україні. Вісник ХНАУ. Серія “Економічні науки” : зб. наук. пр. / Харк. нац. аграр. ун-т ім. В.В. Докучаєва. Харків : ХНАУ, 2020. № 3. С. 364-380.

14. Haque SA, Aziz, SM, Rahman M. Review of cyber-physical system in healthcare. International journal of distributed sensor networks 2014; 10(4):217415. DOI:10.1155/2014/217415

15. 43. Richardson I. Connected health: people, technology and processes, Lero-TR2015-03. Lero Technical Report Series, University of Limerick, 2015

16. Helsi.me. Режим доступу: <https://helsi.me/>

17. Doctor Eleks. Режим доступу: <https://doctor.eleks.com/>

18. Connected Health. Режим доступу: <https://connected-health.co.uk/>

19. НСЗУ. Режим доступу: <https://data.gov.ua/organization/natsionalna-sluzhba-zdorovia-ukrainy>

20. REST. Режим доступу: <https://foxminded.ua/shcho-take-rest-api/>

21. Perry DE, Wolf AL. Foundations for the study of software architecture. ACM SIGSOFT Software engineering notes 1992; 17(4):40–52.

22. Kruchten PB. The 4+1 view model of architecture. IEEE Software 1995; 12(6):42-50.

23. Garlan D. Software architecture. Wiley Encyclopedia of Computer Science and Engineering 2001.

24. Kruchten P, Obbink H, Stafford J. The past, present, and future for software architecture. IEEE Software 2006; 23(2):22-30.

25. Brown A, McDermid J. The art and science of software architecture. Proceedings of First European Conference on Software Architecture, ECSA'07, Springer-Verlag Berlin Heidelberg 2007; volume 4758 of Lecture Notes in Computer Science: 237-256

26. IEEE Std 1471-2000 (2007). ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of SoftwareIntensive Systems. ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15, (pp. c1-24)

27. Chapter 3: Architectural Patterns and Styles. Режим доступа: <https://msdn.microsoft.com/en-us/library/ee658117.aspx>

28. .Bazzani M, Conzon D, Scalera A, Spirito MA, Trainito CI. Enabling the IoT paradigm in e-health solutions through the VIRTUS middleware. 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012; 1954-1959. DOI:10.1109/TrustCom.2012.144

29. Lewis J, Fowler M.. Microservices, 2014 <http://martinfowler.com/articles/microservices.html>

30. Feng X, Shen J, Fan Y. REST: An alternative to RPC for web services architecture. First International Conference on Future Information Networks, ICFIN Oct 2009; 7-10.

31. Russinovich M. Microservices: An application revolution powered by the cloud, 2016. Режим доступа: <https://azure.microsoft.com/en-us/blog/microservices-anapplication-revolution-powered-by-the-cloud/>

32. Cai H, Cui L., Shi, Y, Kong L, Yan, Z. Multi-tenant service composition based on granularity computing. 2014 IEEE International Conference on Services Computing, SCC 2014; 669-676

33. Maia P, Batista T, Cavalcante E, Baffa A, Delicato FC, Pire PF, Zomaya A. A Web Platform for Interconnecting Body Sensors and Improving Health Care. Procedia Computer Science 2014; volume 40:135-142. DOI:<http://dx.doi.org/10.1016/j.procs.2014.10.041>

34. Pepa L, Capecci M, Verdini F, Ceravolo MG, Spalazzi L. An architecture to manage motor disorders in Parkinson's disease. 2nd World Forum on Internet of Things (WF-IoT), Milan, 2015 IEEE; 615-620. DOI: 10.1109/WFIoT.2015.7389124
35. Páez DG, Aparicio F, de Buenaga F, Ascanio J. Big Data and IoT for Chronic Patients Monitoring. International Conference on Ubiquitous Computing and Ambient Intelligence 2014; volume 8867:416–423. DOI:10.1007/978-3-319-13102-3_68
36. Maia P, Batista T, Cavalcante E, Baffa A, Delicato FC, Pire PF, Zomaya A. A Web Platform for Interconnecting Body Sensors and Improving Health Care. *Procedia Computer Science* 2014; volume 40:135-142. DOI:<http://dx.doi.org/10.1016/j.procs.2014.10.041>.
37. Fortino G, Parisi D, Pirrone V, Di Fatta G. BodyCloud: A SaaS approach for community Body Sensor Networks. *Future Generation Computer Systems* 2014; volume 35:62-79. DOI:<http://dx.doi.org/10.1016/j.future.2013.12.015>.
38. Ragavan SV, Haider K, Shanmugavel M. Healthcare Telematics Service Implementation Using OSGi Framework. *Procedia Computer Science* 2014; volume 42:168-174, ISSN 1877-0509. DOI:<http://dx.doi.org/10.1016/j.procs.2014.11.048>.
39. Mohammed J, Lung CH, Ocneanu A, Thakral A, Jones C, Adler A. Internet of Things: Remote Patient Monitoring Using Web Services and Cloud Computing. 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM), Taipei, 2014; 256-263. DOI:10.1109/iThings.2014.45
40. Bazzani M, Conzon D, Scalera A, Spirito MA, Trainito CI. Enabling the IoT paradigm in e-health solutions through the VIRTUS middleware. 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012; 1954-1959. DOI:10.1109/TrustCom.2012.144
41. Corredor I, Metola E, Bernardos AM, Tarrío P, Casar JR. A lightweight Web of Things open platform to facilitate context data management and personalized healthcare services creation. *International journal of environmental research and public health* 2014; 11(5):4676-4713.

42. Rodrigues A, Silva JS, Boavida F. iSenior—A Support System for Elderly Citizens. IEEE Transactions on Emerging Topics in Computing 2013; 1(2):207- 217. DOI:10.1109/TETC.2013.2294917

ДОДАТОК А

Для створення серверної частини медичної інформаційної системи HELSI на мові C# з використанням RESTful архітектури і взаємодії з СУБД MySQL, ми можемо використати ASP.NET Core для створення веб-сервера та Entity Framework Core для роботи з базою даних MySQL.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;

// Модель для зберігання інформації про ЕКГ
public class EkgMeasurement
{
    [Key]
    public int Id { get; set; }

    [Required]
    public int PatientId { get; set; }

    [Required]
    public string Data { get; set; } // Здесь можна зберігати дані ЕКГ, наприклад, у вигляді JSON

    [Required]
    public DateTime Timestamp { get; set; }
}

// Модель для пацієнта
public class Patient
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    public List<EkgMeasurement> EkgMeasurements { get; set; }
}
```

```

}

// Клас для доступу до бази даних
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    public DbSet<Patient> Patients { get; set; }
    public DbSet<EkgMeasurement> EkgMeasurements { get; set; }
}

// Контролер для обробки запитів
[ApiController]
[Route("api/[controller]")]
public class EkgController : ControllerBase
{
    private readonly ApplicationDbContext _context;

    public EkgController(ApplicationDbContext context)
    {
        _context = context;
    }

    // POST api/ekg
    [HttpPost]
    public IActionResult PostEkgMeasurement([FromBody] EkgMeasurement ekgMeasurement)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        _context.EkgMeasurements.Add(ekgMeasurement);
        _context.SaveChanges();

        return Ok();
    }

    // GET api/ekg/{patientId}
    [HttpGet("{patientId}")]
    public IActionResult GetEkgMeasurements(int patientId)
    {
        var patient = _context.Patients.Include(p => p.EkgMeasurements).FirstOrDefault(p => p.Id == patientId);

        if (patient == null)

```

```

        {
            return NotFound();
        }

        return Ok(patient.EkgMeasurements);
    }
}

// Конфігурація сервісів та налаштування
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseMySQL(Configuration.GetConnectionString("DefaultConnection")));

        services.AddControllers();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

class Program
{
    static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }
}

```

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
}
```

Цей код використовує ASP.NET Core для створення RESTful веб-сервісу з двома ендпойнтами: один для надсилання інформації про ЕКГ (POST api/ekg), інший для отримання інформації про ЕКГ для конкретного пацієнта (GET api/ekg/{patientId}). Дані зберігаються в базі даних MySQL за допомогою Entity Framework Core.

ДОДАТОК Б

Для серверу розроблено кросплатформенний клієнт, що працюватиме і на Android, і на IOs (за необхідності напиши два клієнта). Це додаток має містити:

1. Екран для входу лікаря в клієнт (поля логін і пароль, що використовуватимуться для авторизації на сервері),
2. Екран «Доступні пацієнти» зі списком пацієнтів, показниками їх ЕКГ (норма чи відхилення), часом останнього оновлення з моніторів ЕКГ
3. Екран «Додавання пацієнтів в систему» (введення Прізвища, ім'я, віку, захворювання, медичного ідентифікаційного номеру, номеру полісу соціального страхування, коду монітору ЕКГ)

Створення кросплатформенного клієнта для Android та iOS виконано за допомогою Xamarin.Forms – фреймворку для розробки мобільних додатків на C#. Xamarin.Forms застосовано в проекті для створення клієнта з трьома екранами: екраном входу, екраном доступних пацієнтів та екраном додавання пацієнтів.

В архітектурі клієнта використано класичний Model-View-ViewModel (MVVM) підхід для організації логіки додатка. Також було використано Xamarin.Essentials для роботи з локальним сховищем для збереження інформації про вхід.

ДОДАТОК Б.1

1. Екран для входу лікаря (LoginViewModel, LoginPage.xaml):

```
<!-- LoginPage.xaml -->
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:d="http://xamarin.com/schemas/2014/forms/design"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    x:Class="YourApp.Views.LoginPage">
    <StackLayout>
        <Entry Placeholder="Логін" Text="{Binding Username}" />
        <Entry Placeholder="Пароль" Text="{Binding Password}" IsPassword="True" />
        <Button Text="Увійти" Command="{Binding LoginCommand}" />
    </StackLayout>
</ContentPage>
```

ДОДАТОК Б.2

2. Екран "Доступні пацієнти" (PatientsViewModel, PatientsPage.xaml):

```

<!-- PatientsPage.xaml -->
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:d="http://xamarin.com/schemas/2014/forms/design"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    x:Class="YourApp.Views.PatientsPage">
    <StackLayout>
        <ListView ItemsSource="{Binding Patients}">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout>
                            <Label Text="{Binding Name}" />
                            <Label Text="{Binding EkgStatus}" />
                            <Label Text="{Binding LastUpdate}" />
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>

```

ДОДАТОК Б.3

3. Екран додавання пацієнтів (AddPatientViewModel, AddPatientPage.xaml):

```
<!-- AddPatientPage.xaml -->
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:d="http://xamarin.com/schemas/2014/forms/design"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    x:Class="YourApp.Views.AddPatientPage">
    <StackLayout>
        <!-- Введення даних пацієнта -->
        <Button Text="Додати пацієнта" Command="{Binding AddPatientCommand}" />
    </StackLayout>
</ContentPage>
```

ДОДАТОК Б.4

4. Клас PatientViewModel

Додамо клас PatientViewModel, який представлятиме модель пацієнта та буде використовуватися для відображення інформації на екрані "Доступні пацієнти".

```
// PatientViewModel.cs
public class PatientViewModel : INotifyPropertyChanged
{
    public string Name { get; set; }
    public string EkgStatus { get; set; }
    public DateTime LastUpdate { get; set; }

    // Реалізуйте решту логіки
}
```

ДОДАТОК Б.5

5. Комунікація з сервером

Додамо клас ApiService для взаємодії з сервером:

```
// ApiService.cs
public class ApiService
{
    private const string BaseUrl = "https://your-api-url.com/api/";

    public async Task<bool> AuthenticateAsync(string username, string password)
    {
        // Реалізуйте код для аутентифікації на сервері
        // Використовуйте HttpClient для взаємодії з сервером
        return true; // Повертаємо true у разі успішної аутентифікації
    }

    public async Task<List<PatientViewModel>> GetPatientsAsync()
    {
        // Реалізуйте код для отримання списку пацієнтів з сервера
        // Повертаємо список PatientViewModel
        return new List<PatientViewModel>();
    }

    public async Task<bool> AddPatientAsync(AddPatientViewModel newPatient)
    {
        // Реалізуйте код для додавання нового пацієнта на сервер
        // Використовуйте HttpClient для взаємодії з сервером
        return true; // Повертаємо true у разі успішного додавання
    }
}
```

ДОДАТОК Б.6

6. Оновлення екранів та навігація

Внесемо зміни в код ViewModel, щоб забезпечити навігацію між екранами та оновлення інтерфейсу при зміні даних.

```
// LoginViewModel.cs
public class LoginViewModel : INotifyPropertyChanged
{
    // ...

    private async void OnLogin()
    {
        bool isAuthenticated = await ApiService.AuthenticateAsync(Username, Password);

        if (isAuthenticated)
        {
            // Навігація до екрану доступних пацієнтів
            await Application.Current.MainPage.Navigation.PushAsync(new PatientsPage());
        }
        else
        {
            // Виведення повідомлення про помилку
            // Реалізуйте залежно від вашого варіанту виведення повідомлень
        }
    }
}

// PatientsViewModel.cs
public class PatientsViewModel : INotifyPropertyChanged
{
    public ObservableCollection<PatientViewModel> Patients { get; set; }

    public PatientsViewModel()
    {
        LoadPatients(); // Завантаження пацієнтів при створенні ViewModel
    }

    private async void LoadPatients()
    {
        Patients = new ObservableCollection<PatientViewModel>(await
        ApiService.GetPatientsAsync());
        OnPropertyChanged(nameof(Patients));
    }

    // ...
}
```

```

// AddPatientViewModel.cs
public class AddPatientViewModel : INotifyPropertyChanged
{
    // ...

    private async void OnAddPatient()
    {
        bool isAdded = await ApiService.AddPatientAsync(this);

        if (isAdded)
        {
            // Очистити дані форми після успішного додавання
            Name = string.Empty;
            Age = 0;
            Disease = string.Empty;
            MedicalId = string.Empty;
            InsuranceNumber = string.Empty;
            EkgMonitorCode = string.Empty;

            // Виведення повідомлення про успішне додавання
            // Реалізуйте залежно від вашого варіанту виведення повідомлень
        }
    }
}

```

ДОДАТОК Б.7

7. Налаштування App.xaml.cs для використання головної сторінки та запуску додатка:

```
public partial class App : Application
{
    public App()
    {
        InitializeComponent();

        // Встановлення головної сторінки
        MainPage = new NavigationPage(new LoginPage());
    }

    protected override void OnStart()
    {
        // Handle when your app starts
    }

    protected override void OnSleep()
    {
        // Handle when your app sleeps
    }

    protected override void OnResume()
    {
        // Handle when your app resumes
    }
}
```

Цей код створює основну структуру додатка та реалізує базовий функціонал для взаємодії з сервером.