МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО
ЗАХИСТУ
Завідувач кафедри
Олександр ЛИТВИНЕНКО
"_____"_____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Програмний засіб управління елементами IoT

Виконавець: Каміла АХМЕДОВА

Керівник: Олена НЕЧИПОРУК

Нормоконтролер: Олена НЕЧИПОРУК

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри

Олександр ЛИТВИНЕНКО
«_____»_____ 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Ахмедової Каміли Михайлівни
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи: «Програмний засіб управління елементами IoT»

затверджена наказом ректора від «16» вересня 2022 р. № 1530/ст

2. Термін виконання роботи: з 5 вересня 2022 р. по 30 листопада 2022 р.

3. Вхідні дані до роботи: технічна документація, програмні продукти

4. Зміст пояснювальної записки:

1) Огляд архітекутри Інтернету речей

2) Вибір інструментів розробки програмного модуля

3) Проєктування додатку

5. Перелік обов'язкового графічного (ілюстрованого) матеріалу:

1) Алгоритм роботи додатку

2) Екранні форми додатку

3) Топологія *BLE Connect*

4) Архітектура додатку

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проаналізувати джерела по темі дипломної роботи	05.09.2022 – 07.09.2022	
2	Визначити архітектуру, протоколи, та технології, що використовуються в *IoT*	08.09.2022 – 13.09.2022	
3	Розглянути основні можливості застосування IoT у сфері охорони здоров'я	14.09.2022 – 15.09.2022	
4	Обрати технологію розробки та архітектурний патер програмного засобу	15.09.2019 – 16.09.2019	
5	Вивчити особливості технології моніторингу глюкози	19.09.2022 – 30.09.2022	
6	Розробити програмний засіб управління елементом *IoT*	3.10.2022 – 5.11.2022	
7	Оформити пояснювальну записку	7.11.2022 – 13.11.2022	
8	Оформити графічний та ілюстративний матеріал	14.11.2022 – 18.11.2022	

7. Дата видачі завдання: « 5 » вересня 2022 р.

Керівник дипломної роботи _____ Олена НЕЧИПОРУК

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Каміла АХМЕДОВА

(підпис студента) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний засіб управління елементами IoT»: 81 сторінки, 33 рисунки, 1 таблиця, 18 використаних джерел.

ІНТЕРНЕТ РЕЧЕЙ, ПРОГРАМНИЙ ЗАСІБ, *DART*, *FLUTTER, BLE.*

Об’єкт дослідження – управління елементом *IoT*.

Предметом дослідження кваліфікаційної роботи – програмний засіб управління елементами IoT.

Мета роботи – розробити програмний засіб управління елементом *IoT* на прикладі інсулінової помпи.

Методи дослідження. Для виконання кваліфікаційної роботи використано наступні методи: теоретико-емпіричний, аналіз та синтез.

Практичне значення отриманих результатів. Розроблено програмний засіб для віддаленого моніторингу рівня інсуліну в крові. Результати розробки рекомендується впровадити у сферу охорони здоров’я, так як це зможе суттєво підвищити рівень надання медичних послуг та поліпшити життя людей, хворих на цукровий діабет. Дана робота потребує проведення клінічних тестів та тестування із різними видами сенсорів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – *Application Programming*

BLE – *Bluetooth Low Energy*

CGM – *Continuous Glucose Monitor*

IoT – *Internet of Things*

ВСТУП

Актуальність теми. Інтернет речей *(Internet of Things)* – це мережа пов’язаних фізичних об’єктів (речей), які мають індивідуальні ідентифікатори та можуть взаємодіяти між собою без втручання ззовні, спілкуючись через Інтернет. Пристрої Інтернету речей збирають і діляться даними з іншими пристроями, системами та програмами, а також «спілкуються» з людьми та іншими речами, до яких вони підключені. по своїй сутті є мережею мереж,

До пристроїв *IoT* відносяться мільярди пристроїв по всьому світу. Вони об’єднуються в єдину мережу, де відбувається обмін даними, обробка, аналітика та керування процесами, таким чином формується самостійна екосистема, що потребує мінімального втручання людини. Екосистем здатна приймати рішення та керувати об’єктами в залежності від їх призначення.

На сьогоднішній день, пристрої мережі *IoT* активно впроваджують у сферу охорони здоров’я. Пристрої з Інтернету речей зробили можливим віддалений моніторинг в здоров'я пацієнтів, розкриваючи потенціал для забезпечення їх безпеки і здоров'я, а також надаючи лікарям можливість надавати високоякісну медичну допомогу. Серед багатьох доступних медичних пристроїв *IoT*: ті, які забезпечують дистанційний моніторинг температури для вакцин; засоби передачі медичних даних; датчики якості повітря; відстеження ефективності препарату; збір даних життєво-важливих показників; монітори сну; технологія нагадування про поповнення ліків; біометричні сканери дистанційного догляду; і інструменти для сну та безпеки для немовлят. *IoT* змінив життя людей, особливо пацієнтів із захворюваннями, для яких вимірювання показників є критично важливим для життя. Про будь-які порушення або зміни в організмі, механізм оповіщення надсилає сигнали самому пацієнту та відповідним медичним працівникам. Повністю автономні бездротові системи моніторингу стану здоров'я можуть мати багато корисних застосувань. Серед таких застосувань – вимірювання рівня глюкози для діабетиків. Діабет є серйозною проблемою охорони здоров'я. Згідно з доповіддю ВООЗ, кількість людей, хворих на діабет, перевищила 422 мільйони, а в 2012 році понад 1,5 мільйона людей померли

від діабету. ВООЗ віднесла діабет до першої десятки причин смертності. Діабет має серйозні наслідки для благополуччя людини і суспільства. На жаль, досі не існує відомих постійних ліків від діабету. Однак, одним із рішень цієї проблеми є постійне вимірювання рівня глюкози в крові та замикання циклу відповідною доставкою інсуліну. Таким чином, глюкометри, що під'єднані до мережі Інтернету речей, оснащені системами сповіщень, можуть допомогти пацієнтам вжити коригувальних заходів, таких як рішення про дієту, фізичні вправи та час прийому ліків.

Виходячи з проведених досліджень, було прийняте рішення створити програмний засіб для управління інсуліновою помпою за допомогою мови програмування *Dart* у поєднанні із фреймворком *Flutter*. Представлена робота полягає в розробці мобільного застосунку для керування пристроєм *IoT* та представлення даних про рівень глюкози, температуру тіла та контекстні дані (наприклад, темпер, зрозумілій кінцевим користувачам, таким як пацієнт та лікар.

Мета і завдання виконання кваліфікаційної роботи. Мета кваліфікаційної роботи – дослідження та розробка програмного засобу для управління інсуліновою помпою у мережі *IoT*.

Відповідно до мети роботи поставлено наступні завдання:

– дослідити мережу пристроїв *IoT*;

– проаналізувати архітектуру мережі інтернету речей, технології та протоколи передачі даних;

– проаналізувати принципи застосування елементів *IoT* у сфері охорони здоров'я;

– провести аналіз існуючих рішень для контролю рівню інсуліну в крові пацієнта;

– розробити алгоритм роботи програмного засобу;

– обрати мову програмування та визначити інші технології необхідні для програмної реалізації;

– розробити програмний засіб, який буде зручний та зрозумілий у користуванні пацієнтами та забезпече швидку взаємодію із сенсором та його показниками.

Об'єкт дослідження – управління елементом *IoT*.

Предметом дослідження кваліфікаційної роботи – програмний засіб управління елементами IoT.

Мета роботи – розробити програмний засіб управління елементом *IoT* на прикладі інсулінової помпи.

Методи дослідження. Для виконання кваліфікаційної роботи використано наступні методи: теоретико-емпіричний, аналіз та синтез.

Практичне значення отриманих результатів. Розроблено програмний засіб для віддаленого моніторингу рівня інсуліну в крові. Результати розробки рекомендується впровадити у сферу охорони здоров'я, так як це зможе суттєво підвищити рівень надання медичних послуг та поліпшити життя людей, хворих на цукровий діабет. Дана робота потребує проведення клінічних тестів та тестування із різними видами сенсорів.

Наукова новизна отриманих результатів. Розроблено мобільний додаток для керування елементу Інтернету речей, у якості якого виступає інсулінова помпа, із застосуванням технології *BLE Connect* та мови програмування *Dart*.

РОЗДІЛ 1
ОГЛЯД АРХІТЕКТУРИ МЕРЕЖІ ІНТЕРНЕТУ РЕЧЕЙ

Термін, що визначає концепцію «Інтернету речей» вперше з'явився у промові Пітера Т. Льюїса на 15-му щорічному законодавчому засіданні Конгресу *Black Caucus Foundation* у Вашингтоні, опублікованому у вересні 1985 року. За словами Льюїса, «Інтернет речей», або *IoT*, –це інтеграція людей, процесів і технологій із підключеними пристроями та датчиками для забезпечення віддаленого моніторингу, стану, маніпуляції та оцінки тенденцій таких пристроїв [1].

Вперше дав сучасну оцінку Інтернету речей у 1991 році Марк Вайзер. У 1999 році Білл Джой написав про зв'язок між пристроями у своїй таксономії Інтернету, а пізніше цього ж року Кевін Ештон запропонував поняття «Інтернет речей» для пристроїв, що пов'язані між собою. Хоча Ештон віддає перевагу фразі «Інтернет для речей». У той момент він вважав радіочастотну ідентифікацію (*RFID*) важливою для Інтернету речей, що дозволить комп'ютерам керувати всіма окремими речами. Основна тема Інтернету речей полягає в тому, щоб вбудувати мобільні трансивери малого радіусу дії в різноманітні гаджети та предмети повсякденної потреби для забезпечення нової форми спілкування між людьми та речами, а також між самими речами.

Інакше кажучи, Інтернет речей –це мережа пов'язаних фізичних об'єктів, які спілкуються через Інтернет. Пристрої Інтернету речей збирають і діляться даними з іншими пристроями, системами та програмами, а також «спілкуються» з людьми та іншими речами, до яких вони підключені. (рис. 1.1)

Основною ідеєю *IoT* є надання можливості автономного обміну інформацією між унікальними ідентифікованими пристроями реального світу. Такі пристрої оснащені новітніми технологіями, такими, як радіочастотна ідентифікація (*RFID*) і бездротові мережі датчиків (*WSNs*). В залежності від того, яка автоматизована дія виконується, вони надалі отримують можливість приймати самостійні рішення.

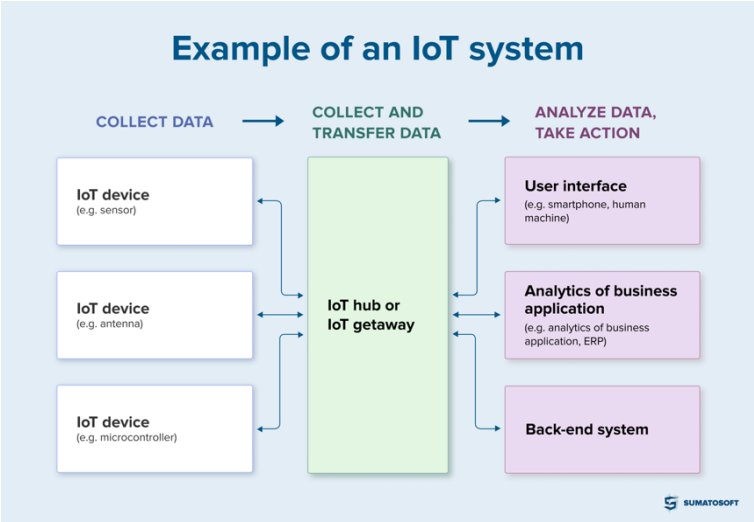

Рис. 1.1. Приклад мережі *IoT*

Пристрої *IoT* виробляють сенсорні, біотелеметричні та широкий спектр інших типів даних, від переносних до промислових датчиків.

Основна концепція мережі інтелектуальних пристроїв обговорювалася ще в 1982 році. Першим пристроєм, що був підключений до *IoT* став модифікований торговий автомат з напоями в Університеті Карнегі-Меллона. Він був здатний повідомляти про свої запаси та про температуру напоїв.

Упродовж наступного десятиліття суспільний інтерес до технології Інтернету речей почав зростати, оскільки на ринку з'являлося все більше підключених пристроїв. У 1991 році компанією *HP* було представлено мережевий принтер, у 2000 році *LG* анонсувала перший розумний холодильник, у 2007 році був випущений перший *iPhone*, у 2009 році *Google* почав тестувати безпілотні автомобілі, а в 2011 році на ринок вийшов розумний термостат *Google Nest*, який дозволяв дистанційно керувати центральним опаленням. Таким чином, до 2008 року кількість підключених пристроїв перевищила кількість людей на планеті.

За даними *Statista* прогнозується, що кількість підключених пристроїв у споживчому секторі збільшиться більш ніж утричі порівняно з 2019 роком та у 2030 році становитиме 15,9 мільярдами підключених пристроїв до Інтернету речей у всьому світі. *IoT* буде покривати кожен сегмент у сфері бізнесу, торгівлі, охорони здоров'я та промисловості [2].

1.1 Екосистема *IoT*

Говорячи про будь-яку екосистему, ми говоримо про складну систему взаємопов'язаних компонентів і середовища, де вони існують і з яким взаємодіють. Усі компоненти пов’язані між собою через потік енергії, деякі цикли та середовище. Точка взаємозв’язку між усіма елементами та навколишнім середовищем є найважливішою в екосистемі, і це те, що відрізняє систему від екосистеми. Сама система утворює складне та єдине ціле, тоді як екосистема тісно пов’язана з навколишнім середовищем.

Використовувати термін «екосистема *IoT*» замість «система *IoT*» обґрунтовується тим, що пристрої *IoT* не мають цінності без середовища, в якому вони існують. Основна перевага, яку пристрої *IoT* приносять людям, –це дані. Ці дані пов’язані з умовами навколишнього середовища або зовнішніми явищами, а також з чимось усередині системи. Окрім зв’язку з навколишнім середовищем, усі пристрої пов’язані один з одним. Кінцевим пунктом призначення даних є люди, які потім ці дані використовують.

Ці три фактори (середовище, дані, люди) підводять до визначення терміну екосистеми Інтернету речей – це мережа взаємопов’язаних пристроїв, яка існує в певному середовищі, збирає дані та передає їх людям, які обробляють та аналізують їх за допомогою сучасних технологій для досягнення певної мети, наприклад, як побудова розумного дому.

Оскільки різні групи людей створюють різні програми *IoT* для своїх цілей, розробка програмного забезпечення *IoT* створює численні екосистеми *IoT*. Ці екосистеми можуть бути простою мережею з 20 підключеними пристроями, такими як «розумний дім», або багаторівневою структурою зі складною та широкою мережею пристроїв, яка потребує складної платформи для керування всіма рівнями.

Якщо розкласти найскладнішу екосистему *IoT* із проміжним рівнем на її блоки, отримаємо наступну схему: пристрої *IoT* збирають дані та безпечно передають їх через мережу до шлюзу, підключеного до Інтернету, який стискає інформацію та передає її до хмари для подальшого аналізу, який пізніше буде відображено в додатку, щоб надати користувачам важливу інформацію. (рис. 1.2)

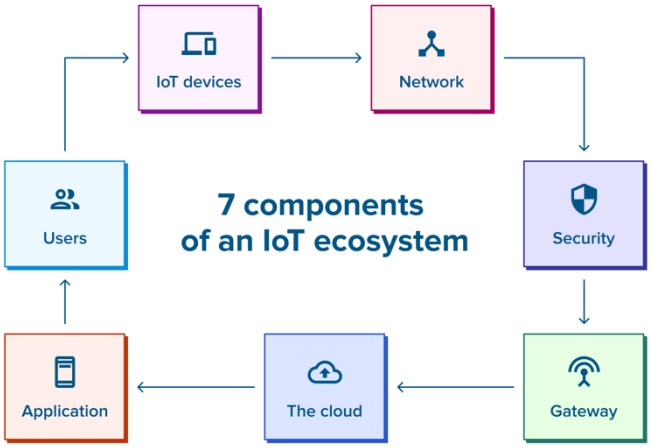

Рис. 1.2. Компоненти екосистеми *IoT*

Таким чином, до екосистеми *IoT* можна віднести наступні 7 компонентів [3]:

— Пристрої *IoT* –це рівень датчиків, приводів і розумних об'єктів, які збирають інформацію про навколишнє середовище та вимірюють фізичні параметри. Основними елементами екосистеми Інтернету речей є датчики та виконавчі пристрої (або просто «речі»). Датчики –це приймачі системи *IoT*, основною функцією яких є отримання інформації з середовища та перетворення її в дані. Зазвичай, в екосистемі Інтернету речей використовується більше ніж один тип датчика. Найпоширенішими датчиками в *IoT* є: датчик температури, датчики наближення, датчики якості, хімічні датчики.

— Безпека –це компонент, який поєднує всі інші компоненти, забезпечує безпеку передачі даних та запобігає несанкціонованим підключенням за межами екосистеми Інтернету речей. В останні роки кількість *DDoS*-атак на основі *IoT* різко зросла. Саме тому будь-яка *IoT*-система потребує потужного рівня безпеки, який захищає хоча б від найпоширеніших вразливостей. Рівень безпеки має широкий спектр обов'язків, таких як: контроль доступу до мережі *IoT*, запобігання витоку даних під час передачі даних через мережу, сканування на наявність шкідливого програмного забезпечення. Для захисту екосистеми Інтернету речей існують різні постачальники прошивок та вбудованих засобів безпеки, такі як *Azure Sphere*, *LynxOS, Mocana, Spartan, Forescout, Symantec* та інші.

— Мережа –основна логістична частина екосистеми Інтернету речей. Мережу також називають рівнем зв'язку. Вона відповідає за всю комунікацію всередині системи *IoT*: взаємозв'язок розумних речей, передачу даних і команд між рівнями IoT, підключення до хмари.

Існує два способи комунікації. Перший спосіб зв'язку відбувається локально в межах локальної мережі (*LAN*) між пристроями *IoT* і смарт-шлюзами через протоколи бездротового зв'язку короткого радіусу дії. Цей спосіб зв'язку є необов'язковим, оскільки датчики можуть підключатися до хмари безпосередньо через Інтернет за допомогою протоколу *TCP/IP*. Однак підключення не через *IP* протоколи споживає менше енергії, оскільки пристрої підключаються до локальних смарт-шлюзів, а не намагаються отримати доступ до головного сервера в хмарі. Найбільш популярними протоколами короткого радіусу дії для архітектури Інтернету речей є наступні: *Wi-Fi*; *Bluetooth* і *Bluetooth Low Energy* (або *Bluetooth LE* для малопотужних пристроїв, які генерують менше даних); *ZigBee* - універсальне рішення, що об'єднує всі "розумні" пристрої; *Near Field Communication* (*NFC*); радіочастотна ідентифікація (*RFID*); *Sigfox*; *LoRaWAN*. Якщо системі необхідно долати великі відстані в діапазоні кілометрів, вона може використовувати малопотужну широкосмугову мережу (або *LPWAN*), яка призначена для бездротової передачі даних на великі відстані.

Другий спосіб зв'язку виникає, коли дані передаються від речей до хмари у випадках, коли відсутній смарт-шлюз або у випадку зв'язку між смарт-шлюзами та хмарою. Мережевий рівень встановлює зв'язок між локальною мережею та Інтернетом. Базовим протоколом тут є протокол *IPv6*.

– Шлюз –це фізичний пристрій або віртуальна платформа, яка служить посередником між пристроями Інтернету речей та хмарою. Існує кілька основних функцій *IoT*-шлюзів: управління потоком даних в екосистемі Інтернету речей; забезпечення безпеки передачі інформації в обох напрямках; передача команди з хмари на *IoT*-пристрої; попередня обробка даних перед відправкою в хмару; економія енергії *IoT*-пристроїв, оскільки зв'язок через Інтернет споживає багато енергії, в той час як технології з низьким енергоспоживанням, такі як *Bluetooth Low Energy*, не споживають її; зменшення затримки відповідей на запити IoT-пристроїв. Зв'язок з

хмарою може зайняти деякий час, в той час як шлюзи можуть бути запрограмовані на вирішення деяких типових проблем і забезпечення миттєвої реакції на пристрої.

— Хмара – це хмарний комп'ютерний ресурс, що відповідає за зберігання даних, їх глибокий аналіз та управління. Іншими словами, це група комп'ютерів, до яких люди отримують доступ через Інтернет для використання їх обчислювальних потужностей з певною метою. Хмара може бути оснащена аналітичним програмним забезпеченням, інструментами візуалізації, штучним інтелектом та машинним навчанням для глибокого аналізу та обробки даних. Однією з головних переваг хмарних рішень є те, що вони легко масштабуються, що є життєво важливою вимогою для побудови ефективної системи *IoT*.

— Програма –це місце, де користувачі можуть взаємодіяти з екосистемою Інтернету речей. Взаємодія стає можливою завдяки графічному інтерфейсу користувача, де користувачі можуть переглядати аналітику та звіти, контролювати систему та керувати пристроями.

— Користувачі є найбільш важливим компонентом серед усіх 7 компонентів екосистеми Інтернету речей. Користувачі виконують дві ролі: вони використовують екосистему Інтернету речей для своїх цілей та визначають, що буде робити екосистема *IoT*. Можливості, які надає екосистема Інтернету речей, стають основою для отримання цінної інформації для всіх типів користувачів. *IoT* має великий вплив на бізнес і світову економіку, він змінює існуючі бізнес-моделі і призводить до появи нових. Екосистема Інтернету речей повинна служити людям, задовольняти їхні потреби та надавати конкретну інформацію, яка допомагає досягти цілей користувачів.

1.2 Огляд архітектури Інтернету речей

Архітектура Інтернету речей складається з пристроїв, мережевої структури та хмарної технології, яка дозволяє пристроям *IoT* спілкуватися один з одним.

Найбільш базовою архітектурою є тришарова архітектура. Вона була представлена на ранніх стадіях досліджень у цій галузі [4]. Вона має три рівні, а саме: рівень сприйняття, мережевий та прикладний рівні.

–Рівень сприйняття –це фізичний рівень, який має датчики для зондування та збору інформації про навколишнє середовище. Він вимірює деякі фізичні параметри або ідентифікує інші інтелектуальні об'єкти в навколишньому середовищі.

Рівень сприйняття представляє самі фізичні пристрої *IoT*. Це можуть бути монітори здоров'я, системи освітлення, автономні транспортні засоби, робототехніка і системи безпеки. Кожен пристрій *IoT* збирає дані, які потребують обробки. Датчики *IoT* недорогі і можуть збирати сотні різних точок даних для обробки. Зібрані дані будуть відрізнятися в залежності від цілей організації.

Головна мета рівня датчиків – ідентифікація різних явищ навколишнього середовища за допомогою периферійних пристроїв та отримання даних з реального світу. Датчики в *IoT* пристроях зазвичай інтегруються за допомогою концентратора датчиків, він акумулює та передає дані з датчиків до блоку обробки даних в пристрої.

Датчики в *IoT* пристроях можуть бути класифіковані на 3 широкі категорії: датчики руху, які вимірюють зміни в русі, а також орієнтацію пристроїв; датчики навколишнього середовища – до них відносяться такі пристрої, як датчики світла, датчики тиску та інші, які вбудовані в *IoT* пристрої реагують на зміни в параметрах навколишнього середовища за допомогою периферійних пристроїв; датчики локації , що взаємодіють з фізичним місцеперебуванням та розташуванням самого пристрою. Найбільш поширеними датчиками місцеперебування, що використовуються в *IoT* є магнітні датчики та *Global Position System* (*GPS*) датчики.

–Мережевий рівень відповідає за з'єднання з іншими розумними речами, мережевими пристроями та серверами. Його функції також використовуються для передачі та обробки даних з датчиків. Рівень використовується, як комунікаційний канал для передачі даних, зібраних на рівні датчиків, до інших підключених пристроїв. В *IoT* пристроях, мережевий рівень реалізований для використання різноманітних комунікаційних технологій (наприклад, *Z-Wave*, *LoRa*, *Wi-Fi*, *Bluetooth* та ін.), що дозволяють передавати дані між різними пристроями всередині самої мережі.

–Прикладний рівень відповідає за надання користувачеві специфічних послуг. Він визначає різні додатки, в яких може бути розгорнутий Інтернет речей, наприклад,

"розумні будинки", "розумні міста" і "розумне здоров'я". Прикладний рівень відповідає за форматування та представлення даних. Прикладний рівень в Інтернеті зазвичай базується на *HTTP*. Однак *HTTP* не підходить у середовищах з обмеженими ресурсами, оскільки він досить складний за своєю природою і, отже, вимагає великих витрат на аналіз. Було розроблено багато альтернативних протоколів для середовищ *IoT*, таких як *CoAP* (*Constrained Application Protocol*) і *MQTT* (*Message Queue Telemetry Transport*).

Трирівнева архітектура визначає основну ідею Інтернету речей, але її недостатньо для дослідження Інтернету речей, оскільки дослідження часто зосереджуються на більш тонких аспектах *IoT* [5]. Тому в літературі пропонується набагато більше багаторівневих архітектур. На рисунку 1.3 наведено чотиришарову архітектуру *IoT.*

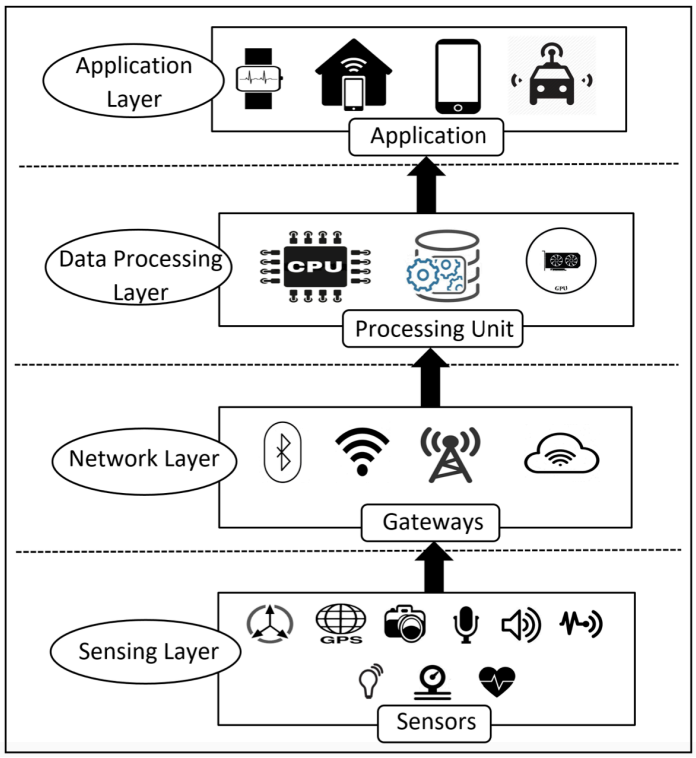

Рис. 1.3. Архітектура *IoT*

Однією з поширених архітектур є п'ятирівневв архітектура, яка додатково включає в себе рівень обробки та бізнес-рівень. П'ять рівнів –це сприйняття, транспортний, рівень обробки, прикладний та бізнес-рівень. Роль рівнів сприйняття

та прикладного така ж, як і в архітектурі з трьома рівнями. Функції решти трьох рівнів розглянуто нижче.

– Транспортний рівень відповідає за відправку зібраних даних в хмару або на периферійний пристрій для обробки. Транспортний рівень покладається на периферійний пристрій для обробки. Транспортний рівень покладається на Інтернет-шлюзи для переміщення даних з рівня фізичного сприйняття в фазу обробки. Зазвичай при розробці використовують мережі стільникового зв'язку та *Wi-Fi* для переміщення даних через транспортний рівень. Існує кілька різних технологій, які можна використовувати на цьому етапі: стільниковий зв'язок *4G LTE*/5G, *Wi-Fi*, *Bluetooth*, широкомасштабні мережі з низьким енергоспоживанням. Архітектуру *IoT* можна розробляти за допомогою комбінації транспортних протоколів. В кінцевому підсумку транспортний протокол, який використовується, повинен бути в змозі надійно підтримувати дані від датчика до найближчого Інтернет-шлюзу.

–Рівень обробки також відомий як проміжний рівень. Він зберігає, аналізує і обробляє величезні обсяги даних, які надходять з транспортного рівня. Він може управляти і надавати різноманітний набір послуг нижчим рівням. Він використовує багато технологій, таких як бази даних, хмарні обчислення і модулі обробки великих даних. Як тільки дані досягають хмари або периферійного пристрою, сервер може перетворити ці дані в інформацію. Рівень обробки є мозком екосистеми *IoT*. Як правило, дані аналізуються, попередньо обробляються та зберігаються тут перед надсиланням до центру обробки даних, де до них отримують доступ програмні додатки, які відстежують та керують даними, а також готують подальші дії. Сучасні архітектури *IoT* використовують машинне навчання і штучний інтелект, які створюють цінність, аналізуючи ці дані. Наприклад, якщо датчик *IoT* реєструє високі коливання температури, штучний інтелект може попередити про цю аномалію, відстежуючи поточну температуру в порівнянні з її базовим значенням. У цьому випадку сервер може відправити команду на блок *HVAC*, щоб знизити температуру і вирішити проблему. Обробка зазвичай відбувається без втручання людини, але люди все одно повинні повідомляти серверу, що робити, коли виконуються певні правила або порушуються порогові значення. На рівні додатків адміністратори керують

налагодженням пристроїв *IoT*, створюють набори правил і встановлюють угоди про рівень обслуговування для своєї архітектури. Надійні архітектури Інтернету речей використовують прикладний рівень для контролю і управління своїми мережами з централізованої інформаційної панелі. Ця централізація знижує складність, особливо в корпоративних мережах *IoT*, що, в свою чергу, підвищує ефективність і безпеку.

– Бізнес-рівень керує всією системою IoT, включаючи додатки, бізнес-моделі і моделі отримання прибутку, а також конфіденційність користувачів. На бізнес-рівні інформація перетворюється на бізнес-аналітику, яка сприяє прийняттю рішень. Зацікавлені сторони та керівники можуть використовувати інформацію, зібрану на прикладному рівні, для прийняття кращих бізнес-рішень. Бізнес-рівень, як правило, покладається на звіти та інформаційні панелі для бізнес-аналітики в реальному часі.

Архітектура Інтернету речей робить можливим коректне отримання даних та відправлення їх на сервер, гарантуючи, що дані потрапляють туди, куди їм потрібно, і обробляються правильно. Без належної архітектури *IoT* мережі стануть ненадійними.

1.3 Огляд протоколів передачі даних у мережі Інтернету речей

Протоколи *IoT* є невід'ємною частиною стеку технологій *IoT*. Їх необхідність пояснюється тим, що протоколи *IoT* дозволяють обладнанню обмінюватися даними. І з цих переданих фрагментів даних, кінцевий користувач може отримувати корисну інформацію.

Протоколи і стандарти *IoT* часто не беруться до уваги, коли люди думають про Інтернет речей. Найчастіше увага індустрії прикута до комунікації. І хоча взаємодія між пристроями, датчиками, шлюзами, серверами і призначеними для користувача додатками є важливими компонентами *IoT*, без правильних протоколів зв'язок не буде ефективним [6].

Протоколи та стандарти *IoT* широко класифікуються на дві окремі категорії. До них можна віднести протоколи передачі даних *IoT* (прикладний рівень), мережеві протоколи для *IoT* (фізичний рівень).

Протоколи даних *IoT* використовуються для підключення малопотужних пристроїв *IoT*. Вони забезпечують зв'язок з обладнанням на стороні користувача - без необхідності будь-якого підключення до Інтернету.

Підключення в протоколах і стандартах даних *IoT* здійснюється через дротову або стільникову мережу. Деякі приклади протоколів передачі даних *IoT*:

– *MQTT* (*Message Queuing Telemetry Transport*) – це спрощений протокол передачі даних *IoT*. Він має модель обміну повідомленнями між відправником і отримувачем і забезпечує простий потік даних між різними пристроями. Основною перевагою *MQTT* є його архітектура. Його генетичний склад є базовим і легким, а тому він здатний забезпечити низьке енергоспоживання для пристроїв. Він також працює поверх протоколу *TCP/IP*.

Протоколи передачі даних *IoT* були розроблені для боротьби з ненадійними мережами зв'язку. Це стало необхідністю в світі *IoT* через зростаючу кількість невеликих, дешевих і малопотужних об'єктів, які з'явилися в мережі за останні кілька років. Незважаючи на широку адаптацію *MQTT* - в першу чергу як стандарту *IoT* з промисловим застосуванням - він не підтримує визначеного режиму представлення даних і структури управління пристроями. В результаті, реалізація можливостей управління даними і пристроями повністю залежить від платформи або постачальника.

– *CoAP* (*Constrained Application Protocol*) – це протокол прикладного рівня. Він розроблений для задоволення потреб систем *IoT* на основі *HTTP* (*Hypertext Transfer Protocol*). *HTTP* є основою передачі даних у Всесвітній мережі.

Хоча існуюча структура Інтернету знаходиться у вільному доступі і може використовуватися будь-яким пристроєм *IoT*, вона часто занадто важка і енергоємна для більшості додатків *IoT*. Це призвело до того, що багато представників спільноти *IoT* враховують HTTP як протокол, який не підходить для *IoT*.

Однак, *CoAp* вирішив це обмеження, перевівши модель *HTTP* у використання в обмежених пристроях і мережевих середовищах. Він має неймовірно низькі накладні витрати, простий у використанні і має можливість включити підтримку

багатоадресної розсилки. Тому він ідеально підходить для використання в пристроях з обмеженими ресурсами, таких як мікроконтролери *IoT*.

– *Advanced Message Queuing Protocol* (*AMQP*) – це відкритий стандартний протокол прикладного рівня, який використовується для передачі транзакційних повідомлень між серверами. Основні функції цього протоколу IoT полягають в наступному: отримання та розміщення повідомлень в черзі; зберігання повідомлень; встановлення взаємозв'язку між цими компонентами.

Завдяки своєму рівню безпеки і надійності, він найчастіше використовується в середовищах, які вимагають серверних аналітичних середовищ, таких як банківська сфера. Однак, в інших сферах він не набув широкого застосування. Через свою важкість він не підходить для сенсорних пристроїв *IoT* з обмеженою пам'яттю. В результаті, його використання все ще досить обмежене в світі *IoT*.

– *DDS* (*Data Distribution Service*) – це ще один масштабований протокол *IoT*, який забезпечує високоякісний зв'язок в *IoT*. Подібно до *MQTT*, *DDS* також працює за моделлю "відправник-отримувач".

Він може бути розгорнутий в різних умовах, від хмари до дуже маленьких пристроїв. Це робить його ідеальним для систем реального часу і вбудованих систем. Більш того, на відміну від *MQTT*, протокол *DDS* дозволяє здійснювати інтероперабельний обмін даними, який не залежить від апаратної та програмної платформи. Фактично, він вважається першим відкритим міжнародним стандартом проміжного програмного забезпечення IoT.

– *HTTP (HyperText Transfer Protocol).*Як вже згадувалося, цей протокол не є кращим стандартом *IoT* через його вартість, час автономної роботи, величезне енергоспоживання і проблеми з вагою.

Тим не менш, він все ще використовується в деяких галузях промисловості. Наприклад, виробництво і 3-*D* друк покладаються на протокол *HTTP* через великі обсяги даних, які він може публікувати. Він дозволяє підключати ПК до 3-*D* принтерів в мережі та друкувати тривимірні об'єкти.

– *WebSocket* був розроблений ще в 2011 році в рамках ініціативи *HTML5*. За допомогою одного *TCP*-з'єднання можна відправляти повідомлення між клієнтом і

сервером. Як і *CoAp*, стандартний протокол підключення *WebSocket* допомагає спростити багато складнощів і труднощів, пов'язаних з управлінням з'єднаннями і двостороннім зв'язком в Інтернеті. Він може бути застосований до мережі *IoT*, де дані безперервно передаються між декількома пристроями. Він найчастіше використовується в місцях, які виступають в якості клієнтів або серверів. Сюди входять середовища виконання або бібліотеки.

1.3.1 Мережеві технології IoT

Оскільки Інтернет речей розвивається дуже швидко, з'являється велика кількість різнорідних інтелектуальних пристроїв, що підключаються до Інтернету. Пристрої Інтернету речей живляться від батареї, мають мінімальні обчислювальні ресурси та ресурси для зберігання даних. Через їх обмежений характер виникають різні комунікаційні проблеми, які полягають у наступному:

– адресація та ідентифікація: оскільки мільйони розумних речей будуть підключені до Інтернету, їх доведеться ідентифікувати за допомогою унікальної адреси, на основі якої вони спілкуються один з одним, для цього потрібен великий адресний простір, і унікальна адреса для кожного розумного об'єкта;

– комунікація з низьким енергоспоживанням: передача даних між пристроями є енергоємним завданням, особливо, бездротовий зв'язок;

– протоколи маршрутизації з низькими вимогами до пам'яті та ефективними схемами зв'язку;

– висока швидкість та відсутність втрат;

– мобільність розумних речей.

Пристрої *IoT* зазвичай підключаються до Інтернету через стек *IP* (*Internet Protocol*). Цей стек дуже складний і вимагає великої кількості енергії та пам’яті від підключених пристроїв. Пристрої *IoT* також можуть підключатися локально через мережі, не пов’язані з *IP*, які споживають менше енергії, і підключатися до Інтернету через розумний шлюз. Не *IP*-канали зв’язку, такі як *Bluetooth*, *RFID* і *NFC*, досить популярні, але мають обмежений діапазон (до кількох метрів). Тому їх застосування обмежене малими персональними мережами. Персональні мережі широко

використовуються в додатках Інтернету речей, таких як переносні пристрої, підключені до смартфонів. Для збільшення радіусу дії таких локальних мереж виникла потреба модифікувати стек *IP*, щоб полегшити зв'язок із низьким енергоспоживанням за допомогою стека *IP*. Одним із рішень є *6LoWPAN*, який об'єднує *IPv6* із персональними мережами низької потужності. Радіус дії PAN із *6LoWPAN* подібний до локальних мереж, а споживання електроенергії значно нижче.

Провідними комунікаційними технологіями, що використовуються в світі *IoT*, є *IEEE* 802.15.4, *Wi-Fi* з низьким енергоспоживанням, *6LoWPAN*, *RFID*, *NFC*, *Sigfox*, *LoraWAN* та інші власні протоколи для бездротових мереж.

В *IoT* переважно використовують мережі з низьким енергоспоживанням і малим діапазоном. Такі мережі підходять для різних приміщень, таких як будинкі, офісні приміщення та інші невеликі простори. Як правило, ці досить економічні в експлуатації. Для використання мереж з низьким енергоспоживанням достатньо невеликих акумуляторів, а інколи налаштування можливе і без використання додаткових елементів живлення.

Нижче наведено декілька мережевих технологій, що використовуються в Інтернеті речей:

– *Z-Wave* –бездротова радіо технологія з низьким енергоспоживанням. Z-Wave працює у діапазоні частот до 1 ГГц. ЇЇ оптимізовано для передавання простих команд управління з досить малими затримками (перемикання каналів, вимикання приладів, т. ін). Система, що працює за допомогою дистанційного керування і користується радіохвилями малої потужності, є також і надійною майже на 100%, оскільки, ця мережа покриває всі області будинки, радіохвилі можуть проходити через стіни, поверхи та меблі.

–*NFC* (*Near Field Communication*), або система зв'язку на невеликих відстанях, призначена для обміну різною інформацією, наприклад, картинками, музичними файлами, номерами телефонів, або ключами цифрової авторизації між двома розташованими близько один до одного пристроями з підтримкою *NFC*. Такою технологією можна користуватися як ключем доступу до даних або служб. Технологія також дозволяє передавати дані не тільки від активного пристрою до пасивного, але і

між двома активними пристроями. *NFC* застосовують для взаємодії з пристроями радіочастотної ідентифікації *RFID*. Для забезпечення сумісності між картами *RFID* та мобільним телефоном різних виробників перевіряють цифровий протокол і вимірюють усіх важливих властивості радіочастотного сигналу: тимчасові характеристики, чутливості та амплітуди приймача в активному режимі, частоти несучої амплітуди сигналу.

–RFID (Radio Frequency IDentification) –метод автоматичної ідентифікації об'єктів, у якому за допомогою радіосигналів зчитують або записують дані, котрі зберігають у транспондерах, або *RFID* - мітках. Будь-яка *RFID*-система складається із зчитувального пристрою (зчитувач, рідер) та транспондеру (він же *RFID*-мітка, іноді також називають *RFID*-тег). Більшість *RFID*-міток складається з двох частин. Перша – інтегральна схема для обробки та зберігання інформації, демодулювання та модулювання радіочастотного сигналу і деяких інших функцій. Друга –антена для прийому і передачі сигналу. А також, для роботи цих міток потрібно програмне забезпечення –програми, за допомогою яких збирають інформацію та аналізують, отриману із *RFID*-міток. *RFID*-мітки можуть керувати товарними запасами або відстежувати час на спортивних змаганнях. Магнітні мітки не замінюють штрих-коди, а доповнюють їх можливістю дистанційного зчитування. Мітками можуть маркувати велику рогату худобу для запису інформації про проходження ветеринарного огляду. Рішення для транспорту допомагають ідентифікувати автомобіль, навіть якщо він рухається з великою швидкістю.

–BLE (Bluetooth Low Energy) –бездротова технологія з низьким енергоспоживанням *Bluetooth*. Пристрої, котрі користуються *BLE*, споживають менше енергії, ніж інші версії *Bluetooth*-пристрої попередніх поколінь. *Bluetooth Low Energy*, також відомий як "*Bluetooth Smart*". Він має відносно менший радіус дії і споживає менше енергії у порівнянні з конкуруючими протоколами. Стек протоколу *BLE* схожий на стек, що використовується в класичній технології *Bluetooth*. Він складається з двох частин: контролера та хоста. Фізичний та канальний рівні реалізовані в контролері. Контролер, як правило, являє собою *SOC* (*System on Chip*) з радіоприймачем. Функціональні можливості верхніх рівнів включені в хост. *BLE* не

сумісний з класичним *Bluetooth*. Основна відмінність полягає в тому, що *BLE* не підтримує потокову передачу даних. Замість цього він підтримує швидку передачу невеликих пакетів даних (розмір пакетів невеликий) зі швидкістю передачі даних 1 Мбіт/с.

Існує два типи пристроїв в *BLE*: ведучий і підлеглий. Ведучий виступає в ролі центрального пристрою, до якого можуть підключатися різні підлеглі. У сценарії *IoT*, де телефон або ПК виступають в ролі ведучого, а мобільні пристрої, такі як термостат, фітнес-трекер, смарт-годинник або будь-який пристрій моніторингу, виступають в ролі підлеглих. У таких випадках підлеглі пристрої повинні бути дуже енергоефективними. Тому для економії енергії ведені пристрої за замовчуванням знаходяться в сплячому режимі і періодично прокидаються для отримання пакетів від ведучого.

У багатьох випадках пристрої зможуть працювати більше року на одній невеличкій батарейці типу “таблетка” без підзарядки. Завдяки цьому, можна користуватися датчиками невеликих розмірів, котрі постійно працюватимуть та взаємодіятимуть із іншими пристроями. *BLE* призначено для тих пристроїв, котрі мають невеликі розміри, тобто для пристроїв, у яких важлива компактність і куди не можна встановити повноцінний акумулятор або батарею великого об'єму. *Bluetooth LE* споживає в 10-20 разів менше енергії і здатний передавати дані в 50 і більше разів швидше та на відстані більше 100 метрів, ніж класичні *Bluetooth* рішення. Крім перерахованих вище переваг, *BLE* має високу безпеку, надійність, низьку затримку при підключенні та низьку споживчу потужність.

Ще одна важлива особливість стандарту полягає в адаптивності переналаштування частоти, тобто, відбувається захист від помилок при передачі сигналу, *BLE* швидко змінює свою робочу частоту, вибираючи найоптимальнішу для усунення перешкод, проблем переповнення і для зниження інтерференції.

–*Wi-Fi HaLow* –протокол бездротової мережі, котрий опубліковано у 2017 році як доповнення до стандарту бездротової мережі *IEEE*_802.11. Протокол працює на частоті 900 МГц, котру не потрібно ліцензувати і забезпечує розширений діапазон *Wi-Fi* мереж, порівняно із звичайними мережами *Wi-Fi*, котрі працюють у діапазонах 2,4

ГГц і 5 ГГц. Його низьке енергоспоживання також є перевагою, котра дозволяє створювати великі групи станцій або датчиків, які взаємодіють і поширюють сигнали, підтримуючи концепцію *IoT*. Низьке енергоспоживання протоколу конкурує з *Bluetooth* і має додаткову перевагу –вищі швидкості передачі даних і більш широкий діапазон покриття. Як і інші *Wi-Fi*-пристрої, пристрої з підтримкою *Wi-Fi HaLow* також підтримують IP-з'єднання, що важливо для додатків *IoT*. Розглянемо специфікації стандарту IEEE 802.11ah. Цей стандарт був розроблений для вирішення сценаріїв бездротових сенсорних мереж, де пристрої обмежені в енергоспоживанні і вимагають відносно великої дальності зв'язку. *IEEE 802.11ah* працює в субгігагерцовому діапазоні (900 МГц). Через відносно низьку частоту, діапазон довший, оскільки більш високочастотні хвилі страждають від більшого загасання. Ми можемо збільшити радіус дії (наразі 1 км) шляхом подальшого зниження частоти, але швидкість передачі даних також буде нижчою, а отже, такий компроміс не є виправданим. *IEEE 802.11ah* також призначений для підтримки великих зіркоподібних мереж, де багато станцій підключено до однієї точки доступу.

1.4 Використання *IoT* у сфері охорони здоров’я

IoT трансформує галузь охорони здоров'я, перевизначаючи простір взаємодії пристроїв та людей у наданні медичних рішень. Використання датчиків, які можуть вимірювати і контролювати різні медичні параметри в організмі людини значно покращує рівень надання медичних послуг. Додатки *IoT* можуть бути спрямовані на моніторинг здоров'я пацієнта, коли він не перебуває в лікарні або коли він знаходиться на самоті. Згодом вони можуть надавати зворотній зв'язок в режимі реального часу лікарю, родичам або самому пацієнту [7].

До появи Інтернету речей взаємодія пацієнтів з лікарями обмежувалася візитами, теле- та текстовими повідомленнями. Лікарі та лікарні не мали можливості безперервно контролювати стан здоров'я пацієнтів та надавати відповідні рекомендації.

Пристрої з підтримкою Інтернету речей зробили можливим віддалений моніторинг в секторі охорони здоров'я, розкриваючи потенціал для забезпечення безпеки і здоров'я пацієнтів, а також надаючи лікарям можливість надавати високоякісну медичну допомогу. Це також підвищило залученість і задоволеність пацієнтів, оскільки взаємодія з лікарями стала простішою і ефективнішою. Крім того, віддалений моніторинг стану здоров'я пацієнта допомагає скоротити тривалість перебування в лікарні і запобігає повторним госпіталізаціям. *IoT* також має великий вплив на значне скорочення витрат на охорону здоров'я та покращення результатів лікування.

На ринку є багато сенсорних пристроїв, які можна носити на собі. Вони оснащені медичними датчиками, які здатні вимірювати різні параметри, такі як частота серцевих скорочень, пульс, артеріальний тиск, температура тіла, частота дихання та рівень глюкози в крові. Серед багатьох доступних медичних пристроїв *IoT*: ті, які забезпечують дистанційний моніторинг температури для вакцин; засоби передачі медичних даних; датчики якості повітря; відстеження ефективності препарату; збір даних життєво-важливих показників; монітори сну; технологія нагадування про поповнення ліків; біометричні сканери дистанційного догляду; і інструменти моніторингу для сну та безпеки для немовлят [8].

Додатки для охорони здоров'я роблять незалежне життя можливим для людей похилого віку та пацієнтів з серйозними захворюваннями. Наразі датчики Інтернету речей використовуються для безперервного моніторингу та реєстрації стану їхнього здоров'я, а також для передачі попереджень у разі виявлення будь-яких аномальних показників. Якщо є незначна проблема, *IoT*-додаток сам може запропонувати пацієнту рецепт лікування.

Дуже важливим моментом тут є контекст. Дані, зібрані медичними датчиками, повинні поєднуватися з контекстною інформацією, такою як фізична активність. Наприклад, частота серцевих скорочень залежить від контексту. Воно збільшується, коли людина тренується. У цьому випадку твердження про аномальний пульс не буде вірним. Тому потрібно поєднати дані з різних датчиків, щоб зробити правильний висновок.

Пристрої *IoT* для пацієнтів у вигляді натільних аксесуарів, таких як фітнес-браслети та інші бездротові пристрої, такі як манжети для вимірювання артеріального тиску і частоти серцевих скорочень, глюкометри тощо, надають пацієнтам доступ до персоналізованої уваги. Ці пристрої можуть бути налаштовані на нагадування про підрахунок калорій, перевірку фізичної активності, призначення, коливання артеріального тиску та багато іншого.

IoT змінив життя людей, особливо пацієнтів із захворюваннями, для яких вимірювання показників є критично важливим для життя. Про будь-які порушення або зміни в організмі, механізм оповіщення надсилає сигнали самому пацієнту та відповідним медичним працівникам. Пацієнти, які страждають на цукровий діабет, можуть мати пристрої з датчиками, імплантованими безпосередньо під шкіру. Датчики в пристроях надсилатимуть інформацію на мобільний телефон пацієнта, коли його або її рівень глюкози стає занадто низьким, а також записуватимуть історичні дані для них. Таким чином, пацієнти також зможуть визначити, коли у них найімовірніше буде низький рівень глюкози в майбутньому, а також зараз. Догляд за пацієнтами може здійснюватися більш ефективно в режимі реального часу без необхідності відвідування лікаря. Це дає їм можливість робити вибір, а також забезпечує науково обґрунтовану медичну допомогу.

IoT дозволяє контролювати стан здоров'я пацієнтів у режимі реального часу, таким чином значно скорочуючи непотрібні візити до лікарів, перебування в лікарні та повторні госпіталізації. Наявність точних показників дозволяє лікарям приймати обґрунтовані рішення на основі доказів і забезпечує абсолютну прозорість. Постійний моніторинг пацієнтів і збір даних в режимі реального часу допомагає діагностувати захворювання на ранній стадії або навіть до того, як хвороба розвинеться на основі симптомів.

Інтернет речей у сфері охорони здоров'я не позбавлений проблем [9]. Підключені пристрої з підтримкою Інтернету речей збирають величезні обсяги даних, у тому числі конфіденційну, що викликає занепокоєння щодо безпеки даних. Реалізація відповідних заходів безпеки має вирішальне значення. *IoT* відкриває нові

аспекти догляду за пацієнтами за допомогою моніторингу здоров’я в режимі реального часу та доступу до даних про здоров’я пацієнтів.

Лікарі, використовуючи носимі пристрої та інше обладнання для домашнього моніторингу, вбудоване в *IoT*, можуть більш ефективно відстежувати стан здоров'я пацієнтів. Вони можуть відстежувати дотримання пацієнтами планів лікування або будь-яку потребу в негайній медичній допомозі.

IoT дозволяє медичним працівникам бути більш пильними і активно взаємодіяти з пацієнтами. Дані, зібрані з пристроїв Інтернету речей, можуть допомогти лікарям визначити найкращий процес лікування для пацієнтів і досягти очікуваних результатів.

Пристрої Інтернету речей з датчиками у лікарнях використовуються для відстеження в режимі реального часу місцезнаходження медичного обладнання, такого як інвалідні візки, дефібрилятори, небулайзери, кисневі насоси та інше обладнання для моніторингу. Розгортання медичного персоналу в різних місцях також може бути проаналізовано в режимі реального часу.

1.5 Переваги та недоліки Інтернету речей

Основні переваги Інтернету речей наступні:

– мінімізація людських зусиль: оскільки пристрої Інтернету речей взаємодіють і спілкуються один з одним, вони можуть автоматизувати завдання, допомагаючи підвищити якість послуг та зменшити потребу у втручанні людини;

– покращений збір даних: інформація легко доступна, навіть якщо людина знаходимося далеко від фактичного місцезнаходження пристрою *IoT*, і вона часто оновлюється в режимі реального часу;

– наявність більшої кількості інформації допомагає приймати кращі рішення;

– моніторинг: знаючи точні показники, може додатково надати точні рекомендації.

Інтернет речей сприяє створенню переваг, але водночас створює і значний набір недоліків. Деякі з недоліків *IoT* наведені нижче:

–питання безпеки: Системи *IoT* взаємопов'язані і взаємодіють через мережі. Таким чином, система пропонує мало контролю, незважаючи на будь-які заходи безпеки, і це може призвести до різного роду мережевих атак [10];

–занепокоєння щодо конфіденційності: система *IoT* надає критичні персональні дані в повному обсязі без активної участі користувача;

–складність системи: проектування, розробка, підтримка та підключення великої кількості технологій до системи *IoT* є досить складним завданням;

–висока ймовірність пошкодження всієї системи: якщо в системі є помилка, існує ймовірність того, що кожен підключений пристрій буде пошкоджений;

–відсутність міжнародної стандартизації: оскільки не існує міжнародного стандарту сумісності для *IoT*, пристроям різних виробників проблематично спілкуватися один з одним;

–висока залежність від Інтернету: пристрої залежать від Інтернету і не можуть ефективно функціонувати без нього.

1.6 Висновки до розділу

У розділі розглянуто та проаналізовано архітектуру мережі Інтернету речей. Під час аналізу актуальності предметної області було виявлено щорічне збільшення кількості підключених до мережі пристрої, що в свою чергу призводить до збільшення сфер, проблеми яких можна вирішити використанням мережі IoT. Наведено основні протоколи, котрі застосовують для побудови таких мереж. На малій дистанції для комунікації між пристроями найкраще використовувати техногогії *BLE* та *NFC*, оскільки вони надають найкраще співвідношення швидкості передачі даних та енергоспоживання. Для передачі даних на середніх дистанціях найкращим вибором буде *Wi-Fi HaLow* оскільки при споживанні енергії на рівні *Bluetooth* має вищу швидкість передачі даних та більший радіус дії.

Також розглянуто можливість використання пристроїв Інтернету речей у сфері охорони здоров’я. Пристрої з підтримкою *IoT* зробили можливим віддалений моніторинг в секторі охорони здоров'я, розкриваючи потенціал для забезпечення

безпеки і здоров'я пацієнтів, а також надаючи лікарям можливість надавати високоякісну медичну допомогу. *IoT* змінив життя людей, особливо пацієнтів із захворюваннями, для яких вимірювання показників є критично важливим для життя. Про будь-які порушення або зміни в організмі, механізм оповіщення надсилає сигнали самому пацієнту та відповідним медичним працівникам.

Незважаючи на усі переваги, пристрої з мережі Інтернету речей не позбавлені недоліків. До основних можна віднести складність проектування, необхідність високого рівня безпеки та залежність від Інтернету.

РОЗДІЛ 2
ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ ПРОГРАМНОГО ЗАСОБУ

Для забезпечення функціонування системи управління та моніторингу інсулінової помпи, необхідно обрати засоби та розробити таке програмне забезпечення, яке буде виконати поставлені задачі та цілі.

Інтернет речей – це постійне спілкування між людьми та пристроями. Пристрої представляють собою різноманітні прилади та датчики, зо поєднані між собою провідними та безпровідними каналами зв'язку для підключення до мережі Інтернет.

Рішення на базі Інтернету речей дозволяють виконати досить складну роботу набагато простішими способами та без втручання людини, а також уникнути помилок через «людський фактор».

2.1 Обґрунтування обраної технології розробки

З ціллю забезпечення моніторингу інсуліну у крові, шляхом отримання даних від інсулінової помпи, було прийнято рішення розробити мобільний додаток для моніторингу та управління інсуліновою помпою.

Мобільний додаток – це програмне забезпечення, яке виконується на мобільних пристроях. В *IoT*, мобільний додаток – це посередник між пристроєм Інтернету речей та мобільним телефоном. Додаток працює як основний інтерфейс, за допомогою якого є можливість керування розумними речами.

Мобільні додатки *IoT* доповнюють і розширюють можливості використання *IoT*, щоб зробити його роботу більш ефективною.

Смартфони насичені великою кількістю різноманітних датчиків. Вони мають більше можливостей підключення, таких як *Wi-Fi*, *Bluetooth* та інші. Ці функції роблять смартфони найзручнішими пристроями для управління технологіями *IoT*.

З метою забезпечення швидкого доступу до даних та миттєвого реагування на них, було прийнято рішення розробити кросплатформений мобільний додаток, що буде виконувати поставлені функцію моніторингу та управління елементом *IoT*, а саме інсуліновою помпою.

У якості інструменту розробки було обрано мову програмування *Dart* у поєднанні із фреймворком *Flutter*, оскільки *BLE* – це нативний канал передачі даних, який за замовчуванням присутній як в *Android*, так і в *iOS*. В кожній операційній системі він різний, тому з метою покриття пристроїв обох операційних систем, необхідно розробляти два різних додатки, в яких буде реалізовано *BLE* канал. Фреймворк *Flutter* у поєднанні із мовою програмування *Dart* дозволяє отримати доступ до нативних каналів усіх платформ, на яких він підтримується.

Dart – об'єктно-орієнтована мультипарадигмова мова програмування, що була представлена компанією *Google* у 2010 році для створення серверних і настільних додатків[11]. Основними задачами розробки *Dart* були:

– створення структурованої, але гнучкої мови програмування для веб-розробки;

– створення мови, що схожа на існуючі, з метою спрощення навчання;

– висока швидкодія програм, що розробляються.

Кодування в *Dart* стає природним, коли ознайомлено із загальними об'єктно-орієнтованими принципами. *Dart* має вбудовану підтримку для юніт-тестування, немає потреби додавати нові бібліотеки або фреймворки.

Dart має підтримку з боку компанії *Google*. Якщо у розробника є пропозиція щодо покращення *Dart SDK*, її можна подати на розгляд до *Google*. Це саме стосується і помилок. На відміну від патентованого ПЗ, мови із відкритим кодом завжди оновлюються та доповнюються. *Dart* має логічну структуру та інтуїтивний синтаксис, схожий на *C*, *Java* та *C#*. Можливість автоматичного виведення типів полегшує багато завдань програмування, дозволяючи програмісту вільно опускати анотації типів, дозволяючи перевірку типу. Це полегшує розробникам перехід на *Dart* незалежно від рівня їх програмування.

Flutter – це кросплатформений набір інтерфейсів, призначений для повторного використання коду в таких операційних системах, як *iOS* та *Android*, а також дозволяє додаткам взаємодіяти безпосередньо з базовими службами платформи [12]. Мета його створення полягає в тому, щоб дозволити розробникам доставляти високопродуктивні програми, які почуваються природньо на різних платформах,

враховуючи відмінності там, де вони існують, одночасно надаючи якомога більше коду. *Flutter* використовується розробниками та організаціями з усього світу і є безкоштовним та відкритим.

За даними *StatCounter* [13], ринок мобільних операційних систем наступний:

– *Android* – 71.81%;

– *iOS* – 27.43%;

– *Samsung* – 0.38%;

– *KaiOS* – 0.14%;

– *Linux* – 0.02%;

– Інші – 0.14%.

Таким чином один *Flutter* додаток зможе покрити 99,24% мобільних пристроїв користувачів, а також це зменшить вартість та тривалість розробки.

2.2 Архітектурний шаблон програмування

Для розробки додатку необхідно визначитися із ключовим архітектурним рішенням – обрати архітектурний шаблон програмування, який передбачає високий рівень абстракції. Вибір архітектури є надзвичайно важливим, оскільки це безпосередньо впливає на швидкість розробки, гнучкість додатку, його стабільність.

Одним із паттернів, який використовується при розробці на *Flutter* є *Model-View-Controller* (*MVC*) (рис. 2.1). Цей шаблон призначений для розробки мобільних та веб-додатків. У цьому паттерні всі об’єкти мають одне з трьох призначень: *model* (модель), *view* (представлення), *controller* (контролер). *MVC* не тільки встановлює призначення для кожного з об’єктів, а й описує взаємодію між ними.

MVC працює за шаблоном “Контролер - Модель - Вид”. З використанням цього паттерну враховуються окремо різні елементи:

– логіка інтерфейсу користувача;

– логіка введення;

– бізнес-логіка.

Згідно шаблону, кожен елемент існує в додатку, але вони не взаємопов’язані. Логіка інтерфейсу користувача взаємопов’язана із *View*. Логіка введення має справу із *Controller*, а бізнес-логіка працює із *Model*.

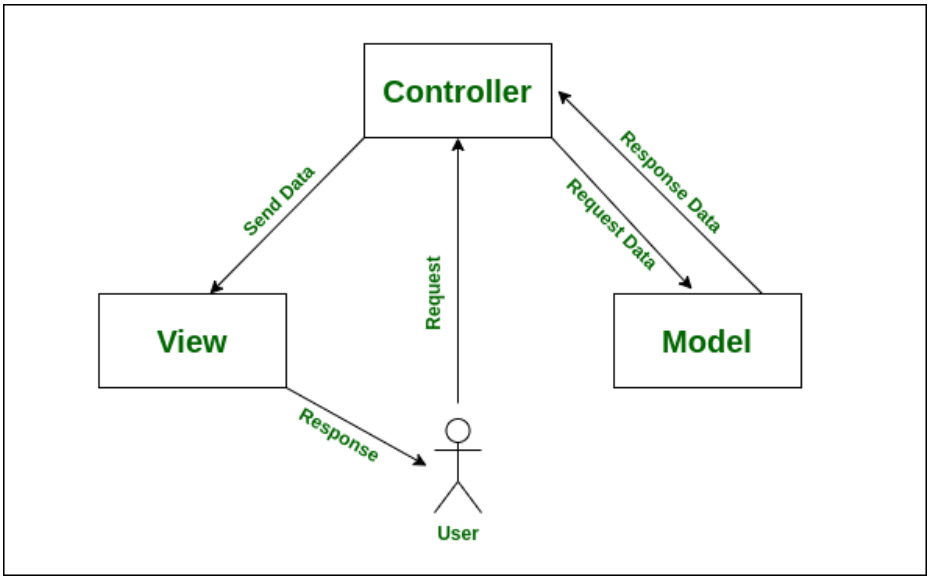

Рис. 2.1. Архітектура *MVC*

Модель – являє собою модель деякої предметної області, яка зберігає у собі дані та методи їх обробки, реагує на запити із контролера та повертає дані або змінює свій стан. Модель описує функціональну бізнес-логіку проєкту, при цьому вона повністю незалежна від інших частин програми, містить опис лише самої себе.

Представлення – відображення даних моделі користувачу. Цей елемент стосується того, як пов'язати дані моделі, проте не містить жодної логіки щодо того, що з себе представляють ці дані або про те як користувач їх може використовувати.

Контролер – знаходиться між Моделлю та Представленням. Він відстежує всі події, які викликані Представленням та виконує відповідну до цих подій реакцію.

Основною перевагою використання даного патерну є незалежність Представлення від Моделі. Це робить код модульним і дозволяє повторно використовувати компонент, а також змінювати частини кода, не впливаючи при цьому на інші.

У розробленому програмному засобі, MCV архітектура буде мати вигляд, представлений на рисунку 2.2.

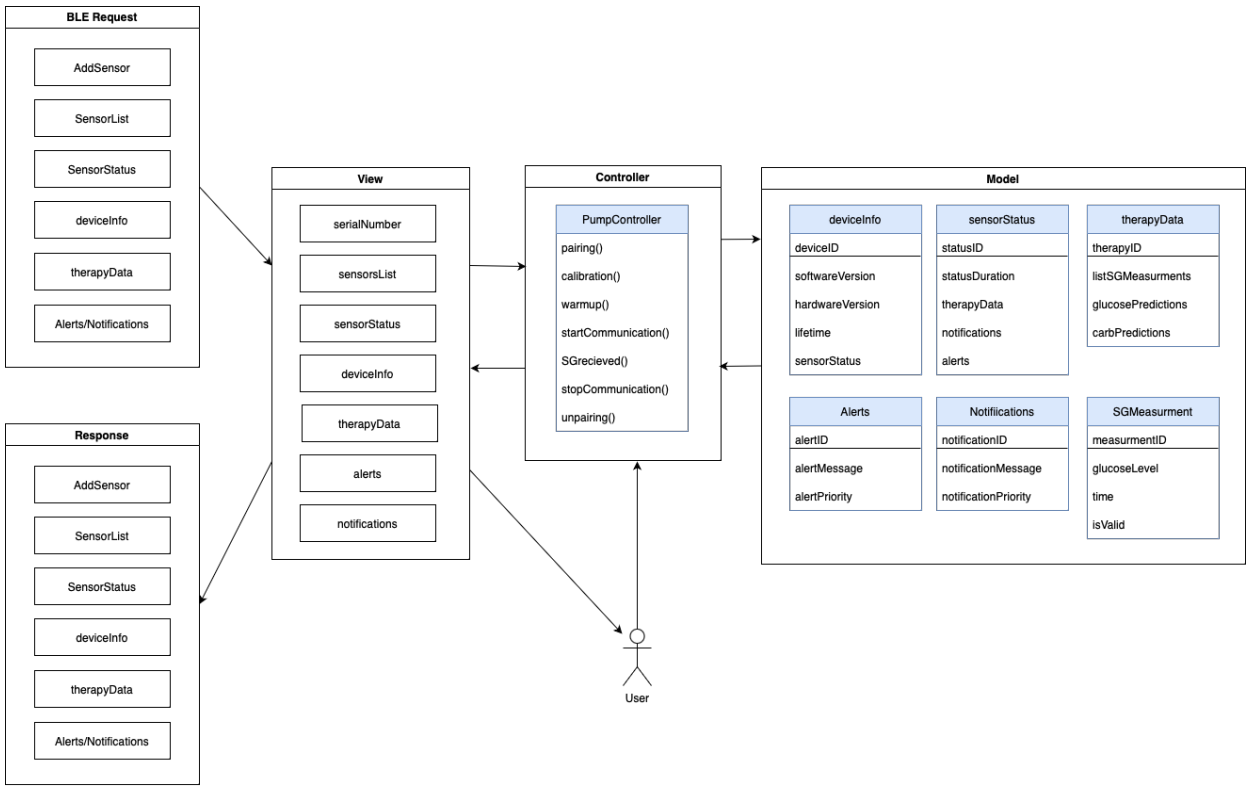

Рис. 2.2. Архітектура додатку

2.3 *Bluetooth Low Energy*

Bluetooth Low Energy (*BLE*) є одним з найбільш широко застосовуваних стандартів зв'язку з низьким енергоспоживанням. Це, не в останню чергу, відбулося завдяки популярності Інтернету речей, який породив масу розумних персональних пристроїв, які потребували загальних і ефективних засобів зв'язку [14]. Існує багато комерційних і бізнес-додатків для цього стандарту, які зробили його таким популярним.

Bluetooth Low Energy –це бездротова персональна мережа з низьким енергоспоживанням, яка працює в діапазоні 2,4 ГГц *ISM*. Її метою є з'єднання пристроїв на відносно невеликій відстані. *BLE* була створена з урахуванням додатків *IoT*, що має особливі наслідки для її дизайну. Наприклад, пристрої *IoT*, як правило, обмежені і вимагають тривалого використання батареї, тому *BLE* надає перевагу низькому енергоспоживанню над безперервною передачею даних. Іншими словами: коли пристрій не використовується, він переходить у сплячий режим для економії енергії.

Що важливо розуміти, говорячи про пристрої, обладнані *BLE*, –це архітектура, що лежить в основі технології, зокрема її асиметрія. Пристрій може функціонувати як в центральній, так і в периферійній ролі. Якщо розглядати смартфон та смарт-браслет: більш функціональний та складний смартфон є центральним пристроєм; смарт-браслет, який має обмежену функціональність –периферійним. Ні два центральних, ні два периферійних пристрої не можуть розмовляти між собою (рис.2.2). Комунікація можлива лише між центральним та периферійним пристроєм. Для подолання цього обмеження в пристрої може бути налаштований як центральний, так і периферійний режим, як це зроблено в багатьох смартфонах.

Рис. 2.2. Асиметрична архітектура *BLE*

Режим пристрою не слід плутати з його загальним атрибутивним профілем, де пристрій може виступати або як сервер, або як клієнт. Як тільки два пристрої встановлюють з'єднання, той, що відправляє дані, є сервером, а той, що їх отримує, - клієнтом. Тобто, якщо смарт-браслет відправляє, скажімо, показники пульсометра на смартфон, то він виступає в ролі сервера. А от якщо смартфон надсилає смарт-браслету оновлення програмного забезпечення, то сервером виступає смартфон.

2.3.1 Відмінність між *Bluetooth Classic* та *Bluetooth Low Energy*

BLE –це незалежний стандарт, який несумісний з "класичним" *Bluetooth*. Останній був вперше представлений на комерційній основі більше 20 років тому і зараз більше не розробляється. Проте він активно використовується в пристроях, які вимагають постійного з'єднання, переважно аудіопристроях, таких як бездротові колонки або навушники.

Тим часом, компанія *SIG* представила *Bluetooth Low Energy* у своїй специфікації *Bluetooth* 4.0 у 2010 році. Основна увага була зосереджена на зростаючому ринку

пристроїв, пов'язаних зі здоров'ям та фітнесом, а також "розумним" будинком та локацією в приміщенні.

Bluetooth Classic призначений для безперервного двостороннього зв'язку, тоді як *BLE* передає менші пакети даних за короткі проміжки часу; як випливає з назви, *Bluetooth Low Energy* має набагато менше енергоспоживання, але з іншого боку, оскільки *Bluetooth Classic* не настільки обмежений, він має більший радіус дії і вищу пропускну здатність.

Bluetooth LE споживає в 10-20 разів менше енергії і здатний передавати дані в 50 і більше разів швидше та на відстані більше 100 метрів, ніж класичні *Bluetooth* рішення. Крім перерахованих вище переваг, *BLE* має високу безпеку, надійність, низьку затримку при підключенні та низьку споживчу потужність.

У порівнянні з класичним *Bluetooth*, *Bluetooth Low Energy* має на меті забезпечити значно менше енергоспоживання та вартість при збереженні аналогічної дальності зв'язку. Мобільні операційні системи, включаючи *iOS*, *Android*, *Windows Phone*, а також *macOS*, *Linux*, *Windows* 8, 10*,* 11, підтримують *BluetoothLE*.

На основі аналізу функціональних можливостей розглянутих технологій було складено порівняльну таблицю (таблиця 2.1).

Таблиця 2.1

Порівняльна таблиця технлогоій *Bluetooth Classic* та *Bluetooth Low Energy*

Функціональні можливості	***Bluetooth Classic***	***Bluetooth Low Energy***
Комунікація	безперервна, двонаправлена	коротка передача даних в одному напрямку
Діапазон	100 м	<100 м
Енергоспоживання	1 Вт	0.01 –0.50 Вт
Швидкість передачі даних	1-3 Мбіт/с	125 кбіт/с –2Мбіт/с
Затримка	100 мс	6 мс
Підтримка голосових команд	так	ні

Найпоширенішою у мережі *IoT* у сфері контролю показників глюкози є топологія з інсуліновою помпою в якості центрального вузла та передавачами, такі як датчик глюкози, глюкометр, смартфон та передавач, у якості периферійних пристроїв. Ця конфігурація називається мережею *BLE Connect* та включає пристрої, наведені на рисунку 2.3.

Друга топологія мережі утворюється зі смартфоном як центральним вузлом і та датчиком глюкози як периферійним пристроєм.

Рис. 2.3. Топологія *BLE Connect*

Інтерфейси *BLE* в мережі *IoT* використовують специфікації, надані *Bluetooth SIG*, з додатковими власними розширеннями та послугами за необхідності. Обсяг даних і функцій, що використовуються цими інтерфейсами, може бути підмножиною того, що передбачено та визначено профілями та послугами *SIG*. Якщо дані та функції, необхідні системі, не передбачені специфікаціями *Bluetooth SIG*, вони можуть бути додані або як розширення до існуючих послуг, або як абсолютно нові власні послуги.

Очікується, що інсулінова помпа з *BLE Connect* буде підтримувати безпеку та ефективність активної інсулінотерапії. Помпа, як центральний пристрій в мережі, може активно керувати мережевою поведінкою периферійних пристроїв для досягнення цієї мети. Пристрої мережі *Bluetooth Low Energy* спілкуються через єдиний радіоприймач на центральному пристрої, який стає слабким місцем для пропускної здатності мережі. Незважаючи на те, що мережа може підтримувати кілька з'єднань, дані повинні послідовно передаватися і прийматися центральним

пристроєм. Коли трафік з різних периферійних пристроїв конкурує за увагу центрального процесора, можуть виникнути перегони. Щоб забезпечити передачу критично важливих даних, коли це необхідно, треба впровадити суворий протокол для управління цими конкуруючими пріоритетами. Помпа може регулювати свій мережевий режим, відключаючись від периферійного пристрою, ігноруючи запити на з'єднання або регулюючи інтервали між з'єднаннями. Помпа також може вибрати пріоритетний зв'язок з певними периферійними пристроями для покращення досвіду користувача або затримок у прийнятті терапевтичних рішень.

Первинною функцією мережі є забезпечення терапії діабету для пацієнта. Вторинною функцією мережі є моніторинг на вторинних та віддалених дисплеях. Третинні функції включають ретроспективний аналіз даних про пацієнта. (Винятком є випадок ручного ініціювання завантаження історії хвороби в кабінеті лікаря, коли ретроспективний аналіз тимчасово стає первинною функцією системи).

Через тривалий час, необхідний для передачі великих обсягів історичних даних, система повинна дозволяти призупиняти і відновлювати третинні функції для підтримки функцій терапії (наприклад, коли помпа передає історичні дані до мобільного додатку, а датчки глюкози намагається надіслати значення глюкози у крові).

Алгоритми помпи досить надійні, щоб підтримувати безпеку та ефективність терапії під час перерв у зв'язку з іншими пристроями.

2.3.2 Протокол мережевих додатків

Протокол мережевих додатків –це протокол, що використовується для обміну даними між пристроями, визначається в термінах послідовностей повідомлень, транзакцій та робочих процесів транзакцій. Всі взаємодії між пристроями складаються з цих елементів.

Послідовність повідомлень –це набір обмінів повідомленнями *BLE*, які забезпечують функціонування пристрою та/або обмін інформацією. Послідовність

повідомлень охоплює обмін командами та відповідями на рівні загального профілю атрибутів, а також будь-які допоміжні або подальші повідомлення.

Послідовність повідомлень є базовим структурним блоком для транзакцій та робочих процесів вищого рівня. Послідовність повідомлень розроблена таким чином, щоб бути надійною та стійкою до проблем на рівні додатків. Будь-які збої в послідовності, ймовірно, відбуваються через несправності на рівні протоколу *BLE*, такі як неочікувані розриви зв'язку та відповіді загального профілю атрибутів з вичерпаним часом. Будь-яка нездатність належним чином і повністю виконати визначену послідовність призводить до збою послідовності. Деталі для відновлення після збою послідовності визначаються не на рівні послідовності, а на рівні транзакції.

2.3.3 Транзакції

Транзакція –це атомарна або безперервна операція, в якій відбувається обмін однією або декількома послідовностями повідомлень *BLE* між центральним та периферійним пристроями. Транзакція визначає порядок і час кожної послідовності повідомлень, а також пом'якшення наслідків відновлення у разі збою послідовності повідомлень. Пом'якшення наслідків можуть включати ігнорування збою, повторну спробу послідовності, виконання іншої послідовності або роз'єднання.

У межах транзакції очікується, що всі визначені послідовності повідомлень будуть успішно завершені у встановленому порядку. Будь-яке відхилення від визначеного порядку транзакції вважається збоєм і буде оброблятися будь-яким визначеним пом'якшенням наслідків збою. Якщо не визначено жодного пом'якшення наслідків, поведінка за замовчуванням полягає в тому, що пристрій, який спостерігає збій, негайно від'єднується від іншого.

Кожна послідовність транзакцій має визначений час очікування. Неможливість завершити транзакцію протягом визначеного вікна часу очікування призводить до збою транзакції. Конкретні часи очікування визначаються для кожного інтерфейсу пристрою.

Не існує загальної поведінки обробки помилок при збої транзакції. Реакція кожного пристрою залежить від типу транзакції та ролі пристрою (периферійний або центральний). Деякі транзакції можуть вимагати повторних спроб, в той час як деякі можуть бути відновлені шляхом ініціювання інших транзакцій.

2.4 *Continuous Glucose Monitor*

Технологія моніторингу глюкози, або *CGM* (*Continuous Glucose Monitor),* використовується хворими на цукровий діабет для контролю рівня глюкози в крові протягом останніх трьох десятиліть. Найбільш поширеним і широко використовуваним методом є інвазивний метод, який вимагає від користувача уколу пальця для забору крові [15]. Однак, останнім часом було розроблено багато нових технологій для неінвазивної техніки моніторингу рівня глюкози в крові, і дослідження в цій галузі стрімко зростають. Серед них оптичний та трансдермальний підходи є двома найбільш потенційними методами неінвазивного моніторингу рівня глюкози, які мають дуже хороші перспективи. Рівень глюкози в крові людини можна виміряти за допомогою інфрачервоного випромінювання. Концентрація глюкози в крові залежить від інтенсивності довжини хвилі, характерної для випромінювання. Виявлений рівень глюкози в крові передається на смартфон по бездротовому каналу, а смартфон керує критично важливими для безпеки пристроями, такими як інсулінова інфузійна помпа, за допомогою мобільних медичних додатків на базі *IoT*.

Прилад для моніторингу рівня глюкози в крові використовується для контролю рівня концентрації глюкози в крові. Зазвичай такий прилад використовується хворими на цукровий діабет протягом дня, щоб допомогти їм зрозуміти поведінку концентрації глюкози в крові в залежності від прийому їжі та активності. Зазвичай, вимірювання рівня глюкози в крові здійснюється за допомогою глюкометра. При інвазивному моніторингу рівня глюкози в крові користувачеві необхідно взяти кров і нанести її на тест-смужку. Цю тест-смужку потрібно вставити в глюкометр, щоб перевірити показання. Для кожного тесту також потрібна нова тест-смужка, що збільшує вартість такого приладу. Оптимальне дозування інсуліну вимагає частого або безперервного моніторингу рівня глюкози в крові. Але безперервний моніторинг

необхідно здійснювати під шкірою, що спричиняє травми та потребує заміни щотижня. Альтернативою є неінвазивні монітори рівня глюкози в крові. Запропонований метод використовує ближній інфрачервоний датчик. В основному, цей типовий глюкометр виробляється кількома різними виробниками з подібним підходом, де в більшості систем вимірюється електрична характеристика напруги на виході з біосенсора або тест-смужки, яка проходить через механізм фільтрації з метою покращення сигналу для кращого перетворення напруги в концентрацію глюкози в крові. Зазвичай, хворим на цукровий діабет рекомендується мати відповідний режим моніторингу рівня глюкози в крові для того, щоб відслідковувати рівень глюкози в крові [16].

CGM працює за допомогою крихітного датчика, введеного під шкіру, як правило, на животі або руці (рис. 2.4). Датчик вимірює рівень інтерстиціальної глюкози, тобто глюкози, яка міститься в рідині між клітинами. Датчик тестує рівень глюкози кожні кілька хвилин. Передавач бездротовим способом надсилає інформацію на монітор. Монітор може бути частиною інсулінової помпи або окремим пристроєм, який можна носити в кишені чи сумочці. Деякі CGM надсилають інформацію безпосередньо на смартфон або планшет.

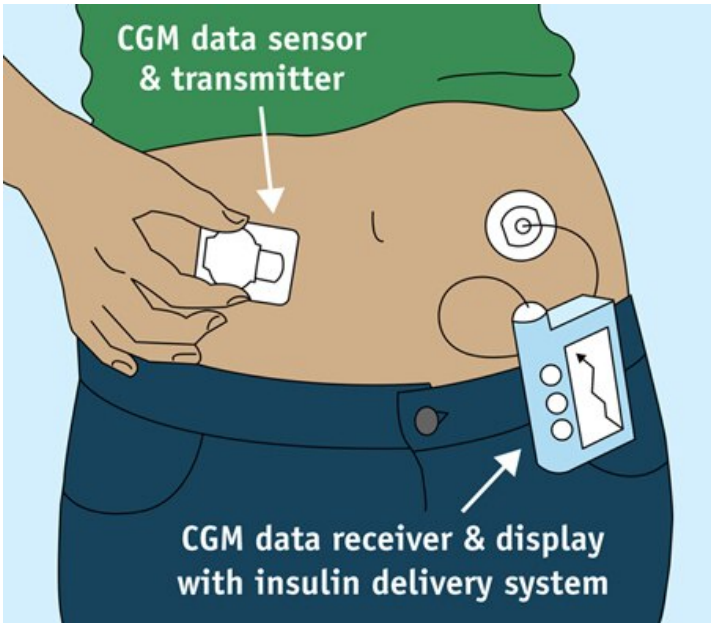

Рис. 2.4. Інсулінова помпа та датчик моніторингу глюкози

2.4.1 Особливості роботи *CGM*

CGM завжди увімкненутий та реєструє рівень глюкози –незалежно від того, що робить людина. Багато глюкометрів мають спеціальні функції, які працюють з інформацією, отриманою з ваших показників рівня глюкози:

–сигнал тривоги може пролунати, коли рівень глюкози стає занадто низьким або занадто високим;

–існує можливість відмічати в приладі *CGM* свої прийоми їжі, фізичну активність та ліки, а також рівень глюкози;

–завантаження даних на комп'ютер або смарт-пристрій, щоб легше бачити тенденції зміни рівня глюкози.

–двічі на день вам необхідно перевіряти *CGM*, для цього необхідно тестувати краплю крові на стандартному глюкометрі, показники глюкози повинні бути однаковими на обох пристроях;

Для безпеки важливо вжити заходів, коли *CGM* сигналізує про високий або низький рівень глюкози в крові. Деякі моделі можуть негайно надсилати інформацію на смартфон другої особи –можливо, лікаря, партнера або опікуна. Наприклад, якщо рівень глюкози у дитини вночі впаде небезпечно низько, *CGM* можна налаштувати так, щоб розбудити батьків у сусідній кімнаті.

2.4.2 Архітектура *CGM*

Для забезпечення безперервного моніторингу рівня глюкози в реальному часі локально та віддалено, використовується архітектура *CGM*, показана на рисунку 2.5, базується на архітектурі *IoT*. Система включає три основні компоненти, такі як портативний сенсорний пристрій, шлюз і серверну систему.

Сенсорний пристрій, структура якого наведена на рисунку 2.6, складається з основних компонентних блоків, таких як датчики, мікроконтролер, блок бездротового зв'язку, компоненти збору інформації та керування. Мікроконтролер виконує основні завдання пристрою, такі як збір та передача даних.

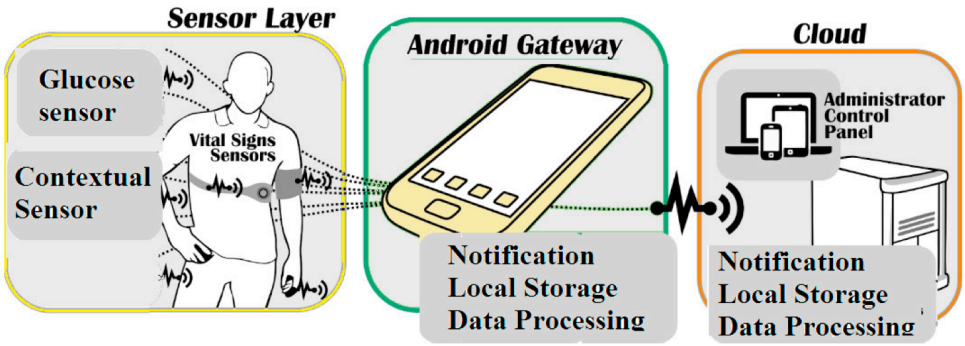

Рис.2.5. Безперервний моніторинг рівня глюкози за допомогою IoT

Зменшення енергоспоживання мікроконтролера дозволяє значно заощадити енергоспоживання пристрою. Мікроконтролер з наднизьким енергоспоживанням, здатний працювати в сплячому режимі, є підходящим для досягнення цієї мети. У пристрої мікроконтролер отримує дані про рівень глюкози від імплантованого датчика глюкози через бездротовий приймач індуктивного зв'язку, в той час як він збирає дані про температуру навколишнього середовища і тіла через канали передачі даних.

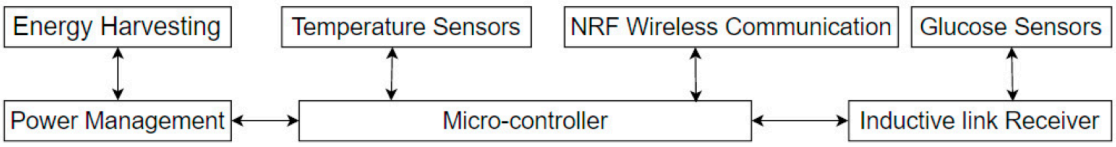

Рис.2.6. Структура сенсорного пристрою

Блок бездротового зв'язку відповідає за передачу даних від мікроконтролера до шлюзу, оснащеного радіочастотним приймачем. Блок включає в себе мікросхему радіочастотного приймача для діапазону 2,4 ГГц *ISM* та вбудовану антену. Завдяки підтримці швидкості передачі даних 2 Мбіт/с, блок повністю задовольняє вимоги до швидкості передачі даних в системі *CGM*. Швидкість передачі даних може бути налаштована для досягнення певного рівня енергоефективності. Наприклад, замість 2 Мбіт/с можна використовувати швидкість передачі даних 256 кбіт/с для економії електроенергії при передачі даних про рівень глюкози, температуру та контекстних даних. Залежно від конкретного застосування, діапазон передачі та потужність

передачі можуть бути налаштовані. При короткій дальності зв'язку *BLE* споживає менше енергії.

У сенсорному вузлі блок зберігання енергії та блок керування живленням, описані нижче, є двома найважливішими компонентами, оскільки вони безпосередньо впливають на споживання енергії та тривалість роботи сенсорного вузла.

Експоненціальний розвиток бездротових телемереж і нової галузі *IoT* широко відкрив двері для численних інтелектуальних застосувань. На жаль, цей розвиток не відобразився на ємності акумуляторів. Основним обмеженням незв'язаних вузлів є обмежена ємність батареї, яка обмежує час роботи вузлів. Обмежений термін служби вузла означає обмежений термін служби додатків або додаткові витрати і складність для регулярної заміни батарей. Вузли можуть використовувати великі батареї для збільшення терміну служби, але тоді необхідно зіткнутись із збільшенням розміру, ваги і вартості. Вузли також можуть використовувати малопотужне обладнання, таке як малопотужний процесор і радіо, ціною меншої обчислювальної здатності і меншої дальності передачі. Для максимізації терміну служби сенсорних вузлів, що живляться від батарей, було запропоновано кілька методів вирішення. Деякі з них включають енергозберігаючі протоколи *MAC*, енергозберігаючі протоколи зберігання, маршрутизації і розповсюдження даних, стратегії робочого циклу, адаптивну швидкість зондування, багаторівневі системні архітектури і надлишкове розміщення вузлів. Хоча всі вищезгадані методи оптимізують і адаптують використання енергії для максимізації терміну служби сенсорного вузла, термін служби залишається обмеженим і німим. Вищезазначені методи допомагають продовжити термін служби застосування та/або інтервал часу між замінами батарей, але не виключають енергетичних перешкод.

Збір енергії може бути рішенням вищезгаданої дилеми. Збір енергії означає використання енергії з навколишнього середовища або інших джерел енергії та перетворення її в електричну енергію. Якщо зібране джерело енергії є великим і періодично/безперервно доступним, сенсорний вузол може живитися безперервно.

Для живлення вузла датчика глюкози обирається комбінація джерел енергії

навколишнього середовища та людини. Завдяки своїй повсюдній доступності радіочастотна енергія є адекватним джерелом для цього застосування. Крім того, оскільки датчик встановлюється на тілі людини, є сенс використовувати це середовище як джерело енергії. За допомогою термоелектричного генератора теплова енергія може бути перетворена в електричну. Процес перетворення заснований на ефекті Зеєбека, де електрика може генеруватися за рахунок градієнта температури між двома провідниками, з'єднаними разом. В даній роботі представлена система збору радіочастотної енергії, збір теплової енергії буде інтегрований в систему збору в подальших роботах.

У сенсорному пристрої використовується мікроконтролер *ATMega328P*, оскільки він дозволяє досягти високого рівня енергоефективності. Мікроконтролер може працювати на частоті 16 МГц. Однак для роботи на такій частоті йому необхідно використовувати зовнішній генератор і потрібне джерело живлення 5В. На відміну від цього, він використовує лише внутрішній генератор на 1 МГц і потребує живлення 2 В для роботи на частоті 1 МГц. На підставі того, що сенсорний пристрій не виконує ніяких важких обчислень, тактова частота 1 МГц і джерело живлення 2 В є придатними. У реалізації сенсорний пристрій більшу частину часу знаходиться в режимі глибокого сну. Він регулярно пробуджується, наприклад, для отримання вхідних даних від датчика глюкози та датчиків температури. Потім пробуджується радіочастотна складова для передачі даних на шлюз. Після чого знову переходить в режим глибокого сну.

Шлюз включає в себе радіочастотний приймач та смартфон, в якому радіочастотний приймач з'єднаний з телефоном через *BLE*. Мікроконтролер живиться напругою 3,3 В та для економії енергоспоживання більшу частину часу він знаходиться в режимі глибокого сну. Прокидається лише за перериванням для отримання вхідних даних від сенсорного пристрою та негайної пересилки на смартфон. Після виконання цих завдань він знову занурюється у глибокий сон. Коли він знаходиться в сплячому режимі, радіочастотний компонент також знаходиться в сплячому режимі.

У шлюз вбудовано кросплатформений додаток для отримання даних від радіочастотного компонента та виконання інших сервісів. Коли дані доступні на одному кінці, додаток автоматично зчитує дані та виконує їх обробку. Крім того, програма здатна представляти оброблені дані у текстовому та графічному вигляді, а також запускати сервіс push-повідомлень.

Сервіс пуш-сповіщень реалізований за допомогою *API (Application Protocol Interface)* пуш-сповіщень *Google*. Коли мобільний додаток виявляє аномальні ситуації (наприклад, занадто низький або занадто високий рівень глюкози), в шлюзі спрацьовує сервіс пуш-повідомлень для відправки повідомлень в хмару, яка потім повідомляє лікарів і кінцевого користувача, який носить сенсорний пристрій.

2.5 *Google Cloud Platform*

Для створення мобільного додатку використовується сервіс *Google Cloud Platform*. *Google Cloud Platform* – це пакет хмарних обчислювальних послуг, які надає компанія *Google*. *Google Cloud Platform* є частиною *Google Cloud*, яка також включає корпоративні версії ОС *Android* та *Chrome*, а також інтерфейси прикладного програмування (*API*) для машинного навчання. Платформа пропонує сервіси для обчислення даних, зберігання та розробки додатків, які будуть працювати на операційній системі *Google*. Для використання сервісу необхідно створити акаунт адміністратора та зареєструвати проєкт.

2.5.1 *Cloud Firestore*

Cloud Firestore – це розміщена у хмарі база даних *NoSQL*, до якої *iOS*, *Android* та веб-додатки можуть отримувати безпосередній доступ через власні *SDK*. Як і *Firebase Realtime Database*, вона підтримує синхронізацію даних між клієнтськими додатками в реальному часі і пропонує автономну підтримку для мобільних пристроїв і Інтернету, для створення адаптивних додатків, які працюють незалежно від затримки в мережі або підключення до Інтернету [17]. *Cloud Firestore* також пропонує

безшовну інтеграцію з іншими продуктами *Firebase* і *Google Cloud*, включаючи *Cloud Functions*.

Доступ до даних у *Cloud Firestore* захищається за допомогою аутентифікації *Firebase* та правил безпеки *Cloud Firestore* для *Android*, *iOS* та *JavaScript*.

Cloud Firestore проєкту має наступні колекції: measurmentStandarts (стандарти вимірювань), devices (пристрої), users (користувачі). У кожному документі колекції користувача містяться наступні класи:

— *DeviceInfo*;

— *SensorStatus*;

— *TherapyData*;

— *Alerts*;

— *Notifications*;

— *SGMeasurments*.

На рисунку 2.7. наведено колекції *Cloud Firestore*.

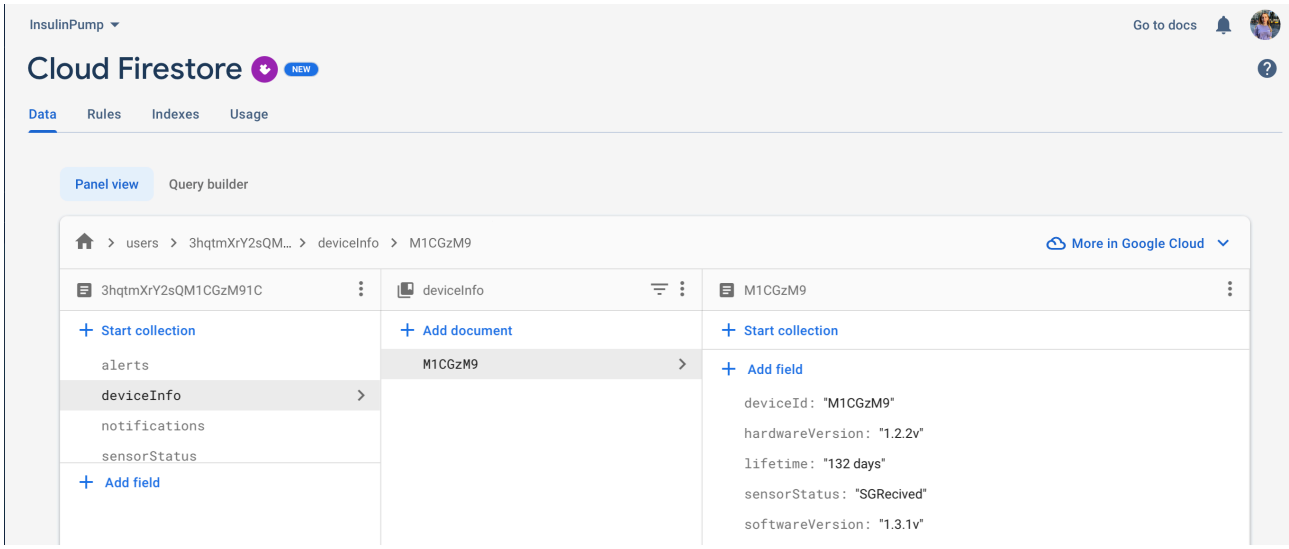

Рис. 2.7. Колекції *Cloud Firestore*

2.5.2 *Cloud Storage*

Cloud Storage для *Firebase* – це послуга зберігання об’єктів, створена компанією *Google*. Пакети *Firebase* для хмарного сховища додають безпеку *Google* для завантаження файлів для програм *Firebase*, незалежно від якості мережі.

Cloud Storage для *Firebase* створено для розробників додатків, яким потрібно зберігати та обслуговувати вміст, створений користувачами.

2.6 Висновки до розділу

У даному розділі обґрунтовано вибір технології розробки та описано архітектуру програмування *MVC*, розкрито сутність технології *BLE* для інсулінової помпи, описано технологію моніторингу глюкози та її особливості, а також налаштовано хмарну базу даних, яка використовується для подальшої розробки додатку.

У якості інструменту розробки було обрано мову програмування *Dart* у поєднанні із фреймворком *Flutter*, оскільки *BLE* – це нативний канал передачі даних, який за замовчуванням присутній як в *Android*, так і в *iOS*. Фреймворк *Flutter* у поєднанні із мовою програмування *Dart* дозволяє отримати доступ до нативних каналів усіх платформ, на яких він підтримується.

Найпоширенішою у мережі *IoT* у сфері контролю показників глюкози є топологія з інсуліновою помпою в якості центрального вузла та передавачами, такі як датчик глюкози, глюкометр, смартфон та передавач, у якості периферійних пристроїв. Ця конфігурація називається мережею *BLE Connect.*

Технологія моніторингу глюкози, або *CGM* (*Continuous Glucose Monitor),* використовується хворими на цукровий діабет для контролю рівня глюкози в крові. Для забезпечення безперервного моніторингу рівня глюкози в реальному часі локально та віддалено, використовується архітектура *CGM*, яка базується на архітектурі *IoT*. Система включає три основні компоненти, такі як портативний сенсорний пристрій, шлюз і серверну систему.

У якості бази даних було обрано хмарну БД *Cloud Firestore*, яка є одним із плагінів сервісу *Firestore*. *Firestore* має велику кількість бібліотек для мобільних пристроїв. Прості дані можна зберігати в документах, які подібних до *JSON*, складні ієрархічні впорядковуються у підколекції, що є зручним при великих масштабах баз даних. Використання хмарних баз даних дозволяє робити додаток гнучким, а розробку швидшою та економічно вигіднішою.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Отримання необхідних доступів до функцій операційних систем

Оскільки додаток є медичним та використовує елемент *IoT*, для коректної роботи необхідно отримати певні доступи у операційних систем. При запуску додатку користувач побачить діалогове вікно з текстом-описом. Якщо дозвіл наданий, додаток зможе отримати доступ даних, що зазначені, поки дозвіл не буде відхилено. У разі відхилення запиту на доступ, програма не зможе отримати доступ до цих даних, якщо тільки ви пізніше дозвіл користувач не надасть доступ власноруч. Якщо у доступі було відмовлено, програма все одно буде запущена, але вона може працювати некоректо. Виконання функцій належним чином буде залежати від того, наскільки важливим є цей дозвіл для її роботи.

Для операційної системи *Android* необхідними доступами є:

– доступ до *Bluetooth*;

– доступ до геолокації пристрою;

– доступ до мережі;

– контроль над вимиканням девайсу;

– доступ до фонових процесів.

Для цього необхідно запитати *permission* (дозвіл) у смартфона користувача для використання цих даних. Для цього необхідно використовувати код, що наведений на рисунку 3.1.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

Рис. 3.1. Створення запитів на доступ до використання функцій

Розглянемо кожен доступ більш детально.

– *android.permission.BLUETOOTH* є доступом до програмного забезпечення *Bluetooth* та *BLE*, встановленого на пристрій. Без цього доступу не можливо працювати з сервісами *BLE*.

– *android.permission.BLUETOOTH_ADMIN* – це батьківський доступ, який надає можливість шукати інші пристрої, маніпулювати налаштуваннями *Bluetooth* на поточному пристрої, впливати на живлення пристрою тощо. Містить в собі *BLUETOOTH_ADVERTISE*, *BLUETOOTH_CONNECT*, та *BLUETOOTH_SCAN* дозволи. Дозволи *BLUETOOTH_ADVERTISE*, *BLUETOOTH_CONNECT* та *BLUETOOTH_SCAN* є дозволами під час виконання. Тому необхідно явно запитувати дозвіл користувача у програмі, перш ніж шукати пристрої *Bluetooth*, робити пристрій доступним для виявлення іншими пристроями або спілкуватися з уже підключеними пристроями *Bluetooth*. Коли програма запитує хоча б один з цих дозволів, система пропонує користувачеві дозволити вашій програмі доступ до пристроїв поблизу. При першому старті додатку користувач побачить вікно, яке запитує чи хоче користувач взаємодіяти з пристроями довкола. Окрім цього всі бажані дозволи треба прописати окремо, адже *BLUETOOTH_ADMIN* надає можливість ним користуватись, та не дає до них доступу.

– *android.permission.ACCESS_NETWORK_STATE* надає можливість дізнатисть стан мережі (*Wi-Fi*, *3G*, *4G*), а також перевірити, чи підключено пристрій до мережі. Якщо немає Інтернету, запит *HTTP* завершиться невдачею, і буде отримано певний виняток, який доведеться обробити. Для роботи із помпою достатньо лише цього доступу, але так як додаток працює з базою даних, то необхідний і доступ *android.permission.INTERNET*.

– *android.permission.INTERNET* – дозволяє використовувати доступ пристрою до Інтернет та відкривати мережеві сокети., завдяки цьому доступ забезпечується передача даних зі смартфону до хмарної БД.

– *android.permission.ACCESS_FINE_LOCATION* – доступ до поточної локації, на пристроях з версією *ANDROID* 11 і нижче. Дозволяє *API* визначати якомога

точніше місцезнаходження за допомогою доступних постачальників даних про локацію, включаючи глобальну систему позиціонування (*GPS*), а також *Wi-Fi* і дані мобільного стільникового зв'язку.

– *android.permission.ACCESS_COARSE_LOCATION* – доступ до поточної локації, на пристроях з версією *ANDROID* 9 і нижче. Дозволяє *API* повертати приблизне місцезнаходження пристрою. Дозвіл надає оцінку місцезнаходження пристрою від служб визначення місцезнаходження, як описано в документації про точність приблизного місцезнаходження. На більш старих пристроях буде використовуватися *ACCESS_COARSE_LOCATION*, адже він потребує менше ресурсів операційної системи.

– *android.permission.WAKE_LOCK* – постійний доступ до стану пристрою. *WAKE_LOCK* –це потужна концепція в *Android*, яка дозволяє розробнику змінювати стандартний стан живлення свого пристрою. Оскільки додаток застосовується у медичній сфері, коректна робота важлива і необхідна, оскільки від цього безпосередньо залежить життя та стан здоров'я користувача. Доступ надає можливість працювати медичному додатку навіть в режимі польоту або сну, проте відбувається навантаження батареї та зменшення часу автономної роботи мобільного пристрою. Недоліком використання *WAKE_LOCK* у програмі полягає в тому, що це зменшить час строку служби акумулятора пристрою.

– *android.permission.FOREGROUND_SERVICE* – доступ до фонових процесів, дозволяє в фоновому режимі завантажувати або надсилати дані на помпу або до бази даних. Працює в комбінації з *WAKE_LOCK*, завдяки чому процес взаємодії з помпою перериваєтьсяся мінімальну кількість разів. Використання цього доступу призначено для додатків, що запускають службу фонового відтворення, які мають вимоги щодо створення повідомленнь та їх надсиланнь за допомогою *startForeground*. Спроба запуску сервісу *FOREGROUND* без оголошення цього дозволу в маніфесті призведе до аварійного завершення роботи програми з виключенням.

Варто зауважити, що операційна система *Android,* для оптимізації роботи пристрою та збереження заряду акумулятору, може відімкнути *Bluetooth*, а отже і *BLE*. У цьому випадку мобільні пристрої можуть запросити необхідний діапазон

даних у *BLE*-пристрою. Ця функція називається *Backfill*. В такому випадку додаток посилає звуковий сигнал для користувача, запускає перевірку *BLE* підключення, та запускає процес автоматичного перепідключення, в разі розірвання поточного з'єднання.

Для коректної та зручної для користувача роботи додатку необхідно запросити доступи до роботи пристрою, такі як доступ до системи розблокування дотиком пальця, вібрацію, тощо.

Окрім цього, для доступу до апаратного забезпечення необхідні *android.hardware.bluetooth* та *android.hardware.bluetooth_le*, які відповідають за апаратне забезпечення *Bluetooth* та *BLE* відповідно. Повний список запитів на доступ до необхідних нативних функцій у додатку наведено на рисунку 3.2.

```
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
<uses-permission android:name="android.permission.USE_FINGERPRINT" />

<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" android:maxSdkVersion="30"/>
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" android:usesPermissionFlags="neverForLocation" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" android:maxSdkVersion="30" />

<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />

<uses-feature android:name="android.hardware.bluetooth" android:required="false"/>
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

Рис. 3.2. Повний список запитів на доступ

Більшість програм, що написані на смартфони під операційною системою iOS запитують доступ одразу після втсановлення та першого запуску додатку. Основним запитом для цих мобільних девайсів є запит на підключення до мережі *BLE*. У фрагменті коду, що наведено на рисунку 3.3 наведено не повний список доступів, але продемонстровано основну структуру запитів доступ до нативних функцій на операційній системі *IOS*.

Для доступів, що запитуються у користувача, обов’язково додається строка, з описом для чого використовується цей доступ.

```
<key>LSRequiresIPhoneOS</key>
<true/>
<key>LSSupportsOpeningDocumentsInPlace</key>
<true/>
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
<key>NSBluetoothAlwaysUsageDescription</key>
<string>APP would like to receive data from the pump, like notifications and sensor glucose event alerts, when you’re
<key>NSBluetoothPeripheralUsageDescription</key>
<string>The app uses bluetooth to find, connect and transfer data between different devices</string>
<key>NSFaceIDUsageDescription</key>
<string>Why is my app authenticating using face id?</string>
<key>UIBackgroundModes</key>
<array>
    <string>bluetooth-central</string>
    <string>fetch</string>
    <string>remote-notification</string>
</array>
<key>UIFileSharingEnabled</key>
<false/>
<key>UILaunchStoryboardName</key>
<string>LaunchScreen</string>
```

Рис. 3.3 Фрагмент коду запитів на операційній системі *IOS*

Для використання пристроїв через *Bluetooth* використовуються *NSBluetoothAlwaysUsageDescription* та *NSBluetoothPeripheralUsageDescription*.

NSBluetoothAlwaysUsageDescription після надання дозволу, надає можливість керувати станом *Bluetooth* навіть без відкритого додатку. Доступ *NSBluetoothAlwaysUsageDescription* із зрозумілим для користувача рядком призначення, який чітко і повністю пояснює, навіщо додатку потрібні дані. Починаючи з весни 2019 року, всі додатки, представлені в *AppStore*, які отримують доступ до даних користувача, повинні включати рядок призначення. Якщо використовуються зовнішні бібліотеки або *SDK*, вони можуть посилатися на *API*, які вимагають рядок призначення. Хоча додаток може не використовувати ці *API*, рядок призначення все одно необхідний.

NSBluetoothPeripheralUsageDescription – це повідомлення, яке надає користувачеві інформацію, чому програма запитує можливість підключення до периферійних пристроїв Bluetooth. Завдяки отриманню цього доступу реалізується

можливість користуватись помпою без перерви, опираючись на *NSBluetoothAlwaysUsageDescription*.

Варто зауважити що гайдлайни компанії *Apple* дозволяють завершити роботу додатку, якщо він був довгий час відкритий у фоновому режимі. Це може викликати серйозні наслідки для користувача і, нажаль, через політику компанії програмно це вирішити не можливо. Додаток надсилає звуковий сигнал про втрату *BLE* з'єднання аналогічно до поведінки в операційній системі *Android*, однак користувач повинен самостійно перезавантажити додаток та відновити з'єданяння із помпою, у разі якщо воно було втрачено.

3.2 Підключення помпи до мобільного пристрою

Підключення помпи до мобільного пристрою можна розділити на три етапи:

– отримання доступів;
– перевірка доступів;
– підключення помпи.

Після запуску додатку, користувач потрапляє на спеціальну сторінку, де мусить надати всі доступи, що були розглянуті пункті 3.1. На цій сторінці детально описано, для чого необхідні доступи, що запитуються. Якщо усі доступи було надано, ця сторінка більше не відображається при наступних запусках. У разі, якщо доступи не було надано, це впливає на належне виконання функцій додатком та користувач не зможе користуватись їм коректно, тому цей екран завантажуватись при кожному перезапуску, до поки усі доступи не будуть надані.

Перевірка доступів – незалежний від попереднього пункту процес. Це обумовлено тим, що помпа працює "у вакуумі", вважаючи що всі необхідні доступи отримано. З метою уникнення помилок, при кожній новій взаємодії користувача з помпою виконується перевірка на наявні доступи, та повторний запит в разі відсутності останніх.

Для перевірки доступів створено спеціальний репозиторій, який містить посилання на нативний *Bluetooth Channel* та потоки, що містять дані про статус *Bluetooth* дозволів (рис. 3.4). Важливим уточненням є той факт, що перевірка дозволу

необхідна тільки на операційній системі *iOS*, оскільки на Android цей дозвіл надається за замовчуванням.

```
class BluetoothSettings extends Repository {
  BluetoothSettingChannel _channel;

  BluetoothSettings({BluetoothSettingChannel channel}) {
    _channel = channel != null ? channel : BluetoothSettingChannel.shared;
  }

  Stream<BluetoothState> getBluetoothState() {
    return _channel.getBluetoothState();
  }

  Stream<bool> getBluetoothEnabledStream() {
    return getBluetoothState().map((state) => state == BluetoothState.on);
  }

  Future<bool> isBluetoothSupported() {
    return _channel.isBluetoothSupported();
  }

  Future<bool> checkBluetoothPermission() {
    if (Platform.isIOS) {
      return _channel.getBluetoothPermission();
    }
    return Future.value(true);
  }

  @override
  void dispose() {
    _channel?.stopBluetoothStateMonitor();
    _channel = null;
  }
}
```

Рис. 3.4. Репозиторій перевірки доступів

Після вдалої перевірки на наявність доступів починається процес підключення помпи. Для цього виконується код, зо наведено на рисунку 3.5.

Першочергово очищується підписка на попередні пошуки. Це необхілно для уникнення зайвого відкриття *BLE* каналів, та підвищення продуктивності додатку. Після цього починається пошук пристроїв з технологією *BLE*, що можуть буті підключені, при цьому список починається з вже підключених девайсів, якщо вони

існують. Наступним кроком є перевірка, чи новий знайдений пристрій не дублює жодний із вже підключених девайсі та чи є його статус *isConnectable*. Статус *isConnectable* означає, чи може даний девайс бути підключеним, адже його розміщення поруч не обов'язково означає готовність до нового підключення. Важливим є і процес перехоплення помилок, оскільки у разі помилки, потрібно перезапустити цю функцію (рис. 3.5).

```
try {
  await _clearScanningSubscription();
  _scanningSubscription =
      _scanBleDevicesService.start().listen((bleDevice) {
    final List<cgm_models.BLEDeviceModel> newDevicesList =
        List.of(_devicesReadyToPairStream.value);
    if (isConnectable &&
        newDevicesList.none(
            (element) => element.identifier == bleDevice.identifier)) {
      newDevicesList.add(bleDevice);
      _devicesReadyToPairStream.add(newDevicesList);
    }
  });
} catch (e) {
  _logger.e(tag, 'Failed start sensor scanning', error: e);
}
```

Рис. 3.5. Функція прехоплення помилок при підключенні нового пристрою

Після того, як додаток знайшов помпу, потрібно ввести унікальний пароль, що написаний на помпі. Введений пароль перевіряється за допомогою функції *RegEx* на наявність заборонених символів (рис. 3.6). В перевірці також перевіряється співпадіння коду сенсора з введеним, основуючись на інформації з бази даних. Враховується і те, чи код вже було введено не правильно, так як від цього залежить вигляд інтерфейсу і видання помилки. Якщо паролі не співпали, встановлюється значення неправильно введеного паролю.

Якщо пароль введено вірно, відбувається безпосереднє підключення до помпи. Якщо дані не було отримано, запит на сполучення отримує відповідь зі статусом *undefined*. Якщо дані не дорівнюють *null*, перевіряється чи пароль не є *null*. Після

цього створюється асинхронна змінна *pairingStatus*, яка пробує виконати сполучення, основуючись на даних девайсу та паролі.

```
void onPairClicked(String pairingCode) {
  if (RegExp(r'^[0-9]+$').hasMatch(pairingCode)) {
        sendAction(PairAction(
          discoveredSensor,
          pairingCode,
          isAfterWrongCode,
        ));
  } else {
    _stateSubject.value = SensorPairingCodeState.wrongPassCodeInit;
  }
}
```

Рис. 3.6. Підключення помпи та перевірка паролю

В разі якщо підключення відбулось успішно, дані *BLE* підключення зберігаються в *KeyChain*, щоб надалі можна було перепідключатись до помпи без введеня коду. Функція повертає статус *accepted*, і додаток переходить на екран з успішним підключенням. Інакше відбувається обробка помилок. Програма розпізнає 4 типи помилок:

– не правильний пароль;

– перевищено кількість спроб підключення;

– пристрій не підтримується;

– невідома помилка.

При не правильно введеному паролі, відбувається повторний перехід на екран введення паролю. При перевищенні кількості спроб правильного введення паролю, користувач отримає повідомлення про необхідність спробувати повторно через 15 хвилин. Помилку, що пристрій не підтримується, можна отримати, якщо телефон користувача не має *BLE* каналу, або не відповідає мінімальним необхідним версіям операційної системи. Всі інші помилки рахуються як невідома помилка. В разі трьох невідомих помилок, з’являється повідомлення з інформацією про те, що перевищено кількість спроб підключення.

У разі якщо користувач зміг успішно пройти з’єднання, він потрапляє на екран з повідомленням, що все майже готово до роботи. Протягом відображення цього

екрану відбувається асинхронне збереження даних в локальну та серверну базу даних (рис.3.7). Після цього відбувається навігація на головний екран.

```
final cgm_models.BLEDeviceModel? deviceModel =
    _devicesReadyToPairStream.value.firstWhereOrNull((element) =>
        element.serialNumber ==
        sensorPairingRequest.discoveredSensor.name);
_pairingDevice = deviceModel;
if (deviceModel == null) {
  return SensorPairingResponse(SensorConnectionResult.undefined);
}

final int passkey = int.tryParse(sensorPairingRequest.pairingCode) ?? 0;
final cgm_models.PairResponse pairingStatus = await _pairBleDevicesService
    .pairWithPasskey(deviceModel, passkey, () {});
_pairingDevice = null;

if (pairingStatus.bleDeviceConnectionState ==
    cgm_models.ConnectionStatus.connected) {
  await BLESecureChannel.shared.savePasskeyToKeychain(
    deviceModel.identifier,
    passkey,
  );
  return SensorPairingResponse(SensorConnectionResult.accepted);
} else {
  if (pairingStatus.exception is cgm_models.InvalidPasskeyException) {
    emitConnectionEvent(SensorConnectionEvent.passkeyRequired);
  } else if (pairingStatus.exception
      is cgm_models.MaximumRetriesException) {
    emitConnectionEvent(SensorConnectionEvent.connectionTimeout);
  } else if (pairingStatus.exception is cgm_models.BleException &&
      (pairingStatus.exception as cgm_models.BleException).type ==
          cgm_models.BleExceptionType.unsupported) {
    emitConnectionEvent(SensorConnectionEvent.deviceNotCompatible);
  } else {
    emitConnectionEvent(SensorConnectionEvent.undefined);
  }
  return SensorPairingResponse(SensorConnectionResult.undefined);
```

Рис. 3.7. Асинхронне збереження даних в локальну та серверну базу даних

3.3 Поєднання додатку з помпою

Поєднання означає перетворення незалежних подій в цілісний потік, що починається з підключення сенсора. Після підключення сенсора, додаток отримує дані про стан пацієнта, на основі яких формуються дані для лікування. Дані для

лікування динамічно змінюються, що в свою чергу потребує постійної презентації кінцевому користувачу. Окрім цього, в ході експлуатації може виникнути потреба відключити сенсор, або підключити новий. Весь цей потік подій – це залежні одна від одної події, які сумарно представляють собою додаток. Усі події зберігаються у хмарній базі даних, що була створена раніше. Це необхідно для того, щоб додаток був менш об'ємним, адже дані завантажуються у базу даних та не зберігаються локально на пристрої користувача. Зберігання у хмарній базі даних також забезпечує відновлення даних. Якщо користувач втратить телефон, або видалить додаток – дані терапії будуть автоматично завантажені знов при повторній авторизації у додатку.

3.3.1 Комунікація між помпою та додатком

В попередніх розділах зазначалось, що комунікація між сенсором і мобільним пристроєм відбувається за допомогою технології *BLE*. Комунікацію можна розділити на дві частини: прийом даних та відправка команд.

За замовчуванням сенсор готовий до підключення та чекає на команди від користувача. Зважаючи на це, при спробі підключити сенсор користувач запускає пошук девайсів навколо свого мобільного телефона. У разі, якщо користувач знайшов сенсор, та обрав його для підключення – сенсор отримує команду про початок підключення. Після спроби підключення, не залежно від результату, сенсор повертає дані. Якщо підключення пройшло успішно, а на сенсор надходять дані про стан пацієнта, він передає їх в додаток. Інакше сенсор надсилає дані з кодом помилки, або інформацію про те, що сенсор ще не використовується належним чином. Якщо сенсор було відключено, він автоматично переходить в стан готовності до команд.

На основі бажання або потреби користувач може:

– відключити сенсор;

– перепідключити підключений раніше сенсор;

– підключити новий сенсор;

– розірвати підключення з сенсором.

Виходячи з цих базових функцій, можна розглядати процес відправки повідомлення про обраний стан на сенсор.

Сенсор – це елемент *IoT*, який розуміє лише примітивний набір команд розміром до 128 біт. Для отримання даних використовується технологія *BLE*, однак для того щоб скористатись нею потрібно пройти наступні рівні абстракції:

– отримання даних від користувача;

– конвертація даних в байт код;

– відправка байт коду на нативний рівень операційної системи;

– відправка даних через *BLE.*

Джерелом даних від користувача в даному випадку виступає додаток. Залежно від обраного користувачем сценарію, додаток встановлю один з можливих станів, що описано у наступному програмному коді:

```
enum SensorStatus {
    sensorConnected,
    sensorDisconnected,
    sensorPairing,
    sensorUnpairing,
    sensorAvailable,
    sensorError
}
```

Після цього використовується *Mapper* функція, яка перетворює кожен можливий стан на байт код, в результаті для відправки на сенсор формується набір даних, з в казаним номером статусу, продемонстрований на рисунку 3.8.

```
const SensorGlucose$json = const {
  '1': 'SensorGlucose',
  '2': const [
    const {'1': 'header', '3': 1, '4': 1, '5': 11, '6': '.Header', '10': 'header'},
    const {'1': 'value', '3': 2, '4': 1, '5': 5, '10': 'value'},
    const {'1': 'isig', '3': 3, '4': 1, '5': 5, '10': 'isig'},
    const {'1': 'sensor_state', '3': 4, '4': 1, '5': 14, '6': '.SensorState', '10': 'sensorState'},
  ],
};
```

Рис. 3.8. Функція *Mapper*

Для відправки нативні платформи використовують *BLEConnectionChannel*. Його реалізація відрізняється на нативних платформах, однак ідея однакова. На рисунку

3.9 представлено нативний код написаний на *Swift* для *iOS.*

```
+ (void)registerWithRegistrar:(NSObject<FlutterPluginRegistrar>*)registrar {
    BLEConnectionChannel *connectionChannel = [BLEConnectionChannel new];
    connectionChannel.monitorHandler = BLEMonitorManager.shared;
    connectionChannel.flutterChannel = [FlutterMethodChannel methodChannelWithName:kConnectionChannelName
                                                                    binaryMessenger:[registrar messenger]];
    [registrar addMethodCallDelegate:connectionChannel channel:connectionChannel.flutterChannel];
}

- (void)handleMethodCall:(FlutterMethodCall*)call result:(FlutterResult)result {
    NSString *deviceIdentifier = nil;
    TransmitterType deviceType = kUnknown;
    if ([call.method isEqualToString:kRequestConnectDevice]) {
        deviceIdentifier = call.arguments[kDeviceIdKey];
        NSInteger deviceTypeIndex = [[call.arguments objectForKey:kDeviceTypeKey] integerValue];
        deviceType = (TransmitterType)deviceTypeIndex;
    } else if ([call.method isEqualToString:kRequestSensorCapabilities]) {
        RequestModel *request = [[RequestModel alloc] initFromDictionary:call.arguments];
        deviceIdentifier = request.deviceIdentifier;
    } else {
        deviceIdentifier = call.arguments;
    }
```

Рис. 3.9. Функція комунікації через *BLE* канал

Метод *handleMethodCall* виклається в *Dart* коді за допомогою команди *Invoke.MethodChannel()*.

Розглянемо більш детально отримання даних з сенсора. Механізм є аналогічним до відправки даних, однак є відмінності в отриманих даних. Вони можуть представляти собою як дані про сенсор та його статус, так й інформацію про стан здоров'я пацієнта. На рисунку 3.10 представлено дані про сенсор, які отримала операційна система *Android*.

```
// Define the Sensor Glucose Message Type
message SensorGlucose {//Event Type 214, event recorded by a device when it receives one or more Sensor readings.
  Header header = 1;
  uint32 value = 2; //SG value in set
  uint32 isig = 3; //Isig of SG value in set
  SensorState sensor_state = 4; //Sensor Status
  uint32 rate_of_change = 5; //Rate of change (RoC) associated with SG value in set
  uint32 predictive_sg = 6; //Predicted SG Value
  uint32 v_count = 7; //Vcount values in set
  string device_id = 8; //device_id = product name (GST, 5G, Synchrony) + device serial number
  bool backfill_indicator = 9; //Indicates whether the sg data is backfill data or not
  uint32 sensor_time = 10; //Ram needs to confirm the time format (rtc, utc or therapy?)
}
```

Рис.3.10 Дані про сенсор, отримані від операційної системи *Android*

У наступному коді дані про вимірювання, що були виконані сенсором:

final CgmMeasurements value = converter.unpac(newGattpayload (

new byte [] {

(byte) 0x0F, // Size

(byte) 0xFF, // Flags

(byte) 0xD2, (byte) 0xE4, // Glucose concentration

(byte) 0xD2, (byte) 0x04, // Time offset

(byte) 0x55, (byte) 0x55, (byte) 0x55, // Sensor status

(byte) 0x41, (byte) 0xE1, // Trend Info

(byte) 0x7B, (byte) 0xE0, // Quality

(byte) 0xB0, (byte) 0xA4, // E2E CRC

}

Після отримання інформації від сенсора, відбувається конвертація в рамках мови програмування Dart, та розпочинається підрахунок даних для терапії.

3.3.2 Підрахунок даних для терапії

Дані з сенсора оновлюються кожні 5 секунд. В разі якщо дані не надходять, протягом 15 хвилин додаток буде очікувати на оновлення даних, та демонструвати користувачеві план лікування, який був валідний для останнього отриманого вимірювання. Якщо оновлення даних відбулось, додаток автоматично запустить процес корегування історії терапії. Якщо оновлення не відбулось, додаток переходить до стану “сенсор відключено”, та в разі якщо сенсор буде підключено знову, дані про 15 хвилинний інтервал очікування буде знищено.

На основі даних з сенсора, додаток може робити наступні висновки:

– рівень глюкози пацієнта занадто низький;

– рівень глюкози пацієнта занадто високий;

– пацієнту не рекомендується фізичне навантаження протягом певного часу;

– пацієнту не рекомендується вживання вуглеводів протягом певного часу;

– пацієнту рекомендується ввести дозу глюкози в певному обсязі.

Всі ці висновки робляться на основі порівнянь даних з сенсора з константними значеннями, та на основі введених користувачем даних про себе: дата народження, стать, зріст, вага (рис.3.11).

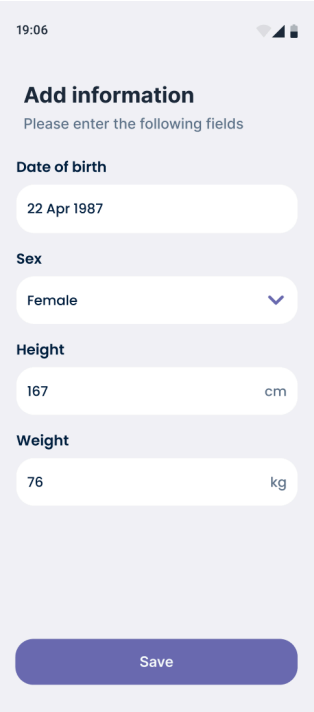

Рис. 3.11. Введення даних про користувача

Наприклад для всіх користувачів максимальна доза інсуліну за один прийом не може перевищувати 200 мг, тому додаток не може рекомендувати вводити дозу більшу ніж це значення. На у коді нижче розглядається, як відбувається перевірка валідності.

double get glucose => _service.measurment.glucose;

double get glucoseChangeRate => _service.measurmentState.glucoseChangeRate;

bool get isBelowValidRange => _service.measurmentState.isBelowValidRange;

bool get isAboveValidRange => _service.measurmentState.isAboveValidRange;

DateTime get lastUpdate => _service.measurmnetState.lastUpdate;

Окрім порівнянь, додаток виконує аналітичну та прогнозуючі функції. На основі даних, що отримуються протягом певного часу, точність аналітики та прогнозування покращується. Прогнозування дозволяє більш коректно працювати додатку в разі тимчасового розірвання комунікації .

3.3.3 Репрезентація даних терапії

Відображення даних про терапію можна розділити на два типи: постійний моніторинг та екстренні повідомлення.

Постійний моніторинг – це відображення даних з сенсора, що приходять і обробляються кожні 5 секунд, не залежно від того запущено додаток чи ні. Демонстрація цього механізму знаходиться на центральному віджеті рисунку 3.12.

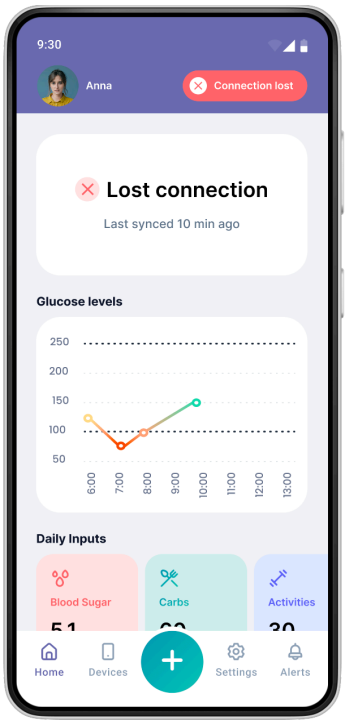

Рис. 3.12 Головний екран додатку у стані «З’єднання втрачено»

Окрім цього інформація про введені вуглеводи, спортивні зайняття та рівень інсуліну відображаться під центральним віджетом. На екрані терапії відображається поточний рівень глюкози, та його зміни залежно від подій, що ввів користувач.

На зображенні відображена структура графа, та рівні які на ньому відображаються. Код для граф віджету наведено на рисунку 3.13.

Рис. 3.13. Граф віджет

Реалізацію головного екрану відображенням даних описано у наступному коді:

return Scaffold(

appBaг: AppBar(),

body: Column (children: [const SensorStatus (),

Graph (

config: config, adapter: adapter, layers: [

SimpLeBackgroundLayer(), const SuspendedModeRangeLayer () ,

AutoModeRangeLayer (autoMode: adapter. autoMode),

GlucoseLimitationLayerO, const GlucoseTrendLayer), UserEventsLayer),

GlucoseEventsLayer(), const HorizontalTimeLayer (), const AxisVerticalLayer),

), //Graph

],

), // Column bottomNavigationBar: BottomNavigationBar (

items: const [

BottomNavigationBarItem(icon: Icon (Icons.home)),

BottomNavigationBarItem(icon: Icon (Icons.devices_fold)),

BottomNavigationBarItem(icon: Icon (Icons.hdr_plus)),

BottomNavigationBarItem(icon: Icon(Icons.settings)),

BottomNavigationBarItem(icon: Icon (Icons.add_alert)),

],

), // BottomNavigationBar

); // Scaffold

Коли дані були про інсулін були успішно додані, вони будуть відображатися на головному екрані наступним чином (рис. 3.14):

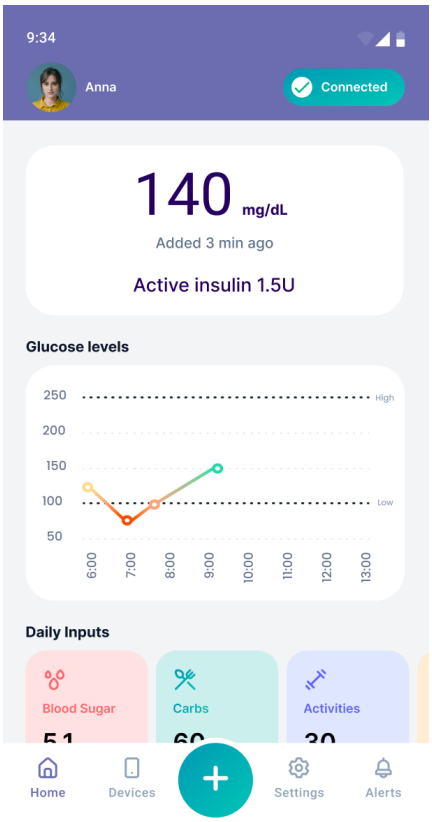

Рис. 3.14. Головний екран із введеними даними

Статус помпи також відображається як «підключена».

Критичні повідомлення – це частина терапії, що завжди буде найбільш пріоритетною для мобільного пристрою. Повідомлення на рисунку 3.13 повідомляє, що рівень глюкози критично низький. Оскільки ця інформація є критично необхідною для пацієнта, вона буде відображатись на екрані не залежно від того, запущено додаток чи ні. Також, користувач почує звуковий сигнал та вібрацію, навіть якщо його телефон в режимі польоту або без звуку.

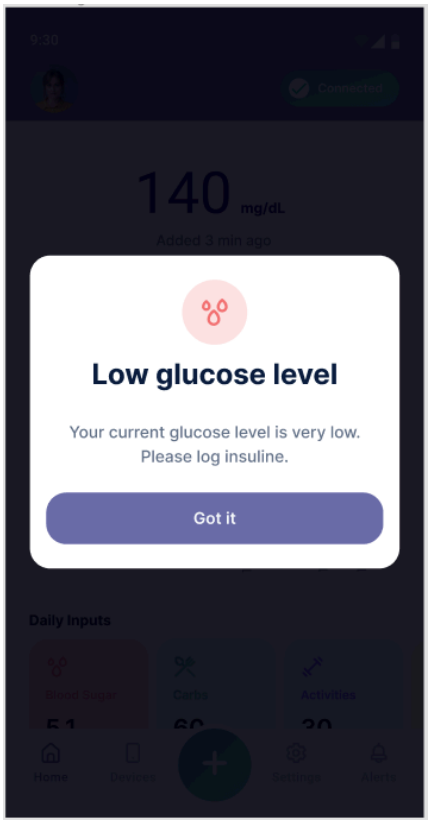

Рис. 3.13. Повідомлення про критично низький рівень глюкози

Усі повідомлення відображаються за допомогою функції *showDialog*, та повернння діалогу з цієї функції:

showDialog <void> (

userSafeArea: false,

context: context,

builder: (Builder context) {

String _dialogTitle = alertTitle,

String _headerText = alertDescription;

return Dialog (alertTitle, alertDescription)

}

)

3.4 Демонстрація результату

Наступним етапом після авторизації користувача є підключення помпи до додатку. Для цього запускається процес сканування для пошуку сенсору (рис. 3.15).

Рис. 3.15. Сканування для пошуку помпи

Після цього можливе настання двох подій. Якщо помпу не було знайдено, то викликається відповідне повідомлення (рис.3.16).

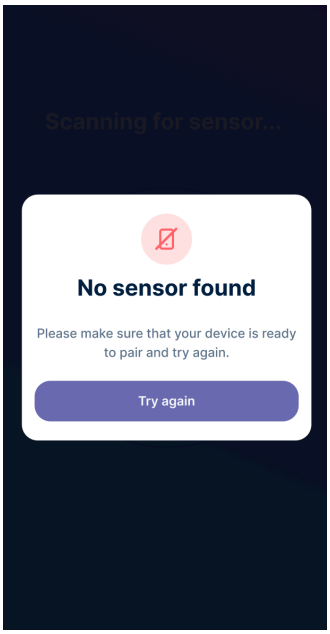

Рис. 3.16. Повідомлення, що сенсор не було знайдено

У разі знаходження сенсора поблизу, який готовий до підключення, відображається зображення цього сенсору та його серійний номер, що підтягуються із бази даних (рис. 3.17).

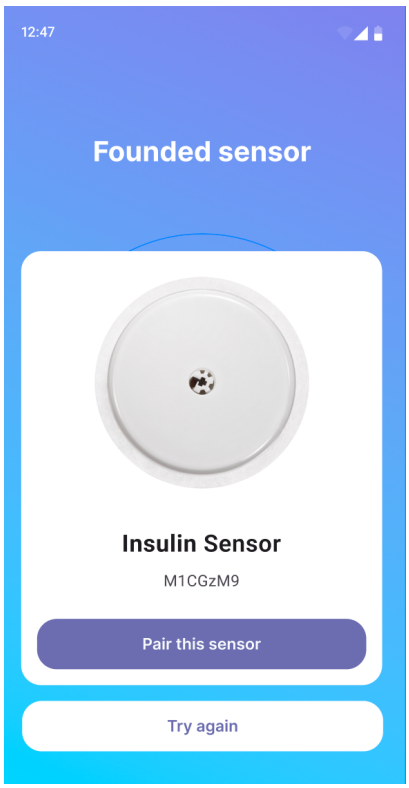

Рис. 3.17. Знайдений сенсор

Для підтвердження необхідно ввести код із 6 цифр, що знаходиться на інсуліновій помпі. Якщо код було введено невірно, повертається повідомлення про помилку (рис.3.18).

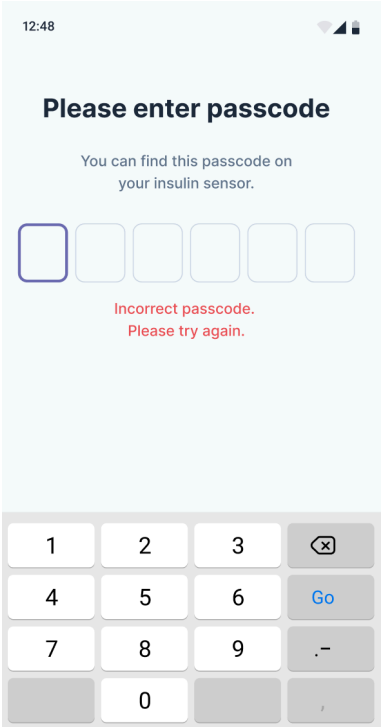

Рис. 3.18. Введеня коду для перевірки

Якщо процес перевірки пройшов успішно, відбувається процес підключення помпи до додатку. Так як завантаження даних відбувається із хмарної бази даних, процес займає деякий час (рис. 3.18).

Рис. 3.18. Процес підключення сенсора

Після завершення налаштувань користувач потрапляє на головний екран додатку.

Для введення даних необхідно натиснути на кнопку «додати». Реалізовано 5 типів подій, які можна додати:

– замітку;
– активність;
– вуглеводи;
– показники цукру у крові;
– інсулін.

Дані, введені до будь якої події, зберігаються та відправляються до бази даних, після чого відбувається оновлення віджетів головної сторінки та відображення

повідомлень-рекомендацій, якщо вони необхідні. Подія «додавання вуглеводів» представлена на рисунку 3.19.

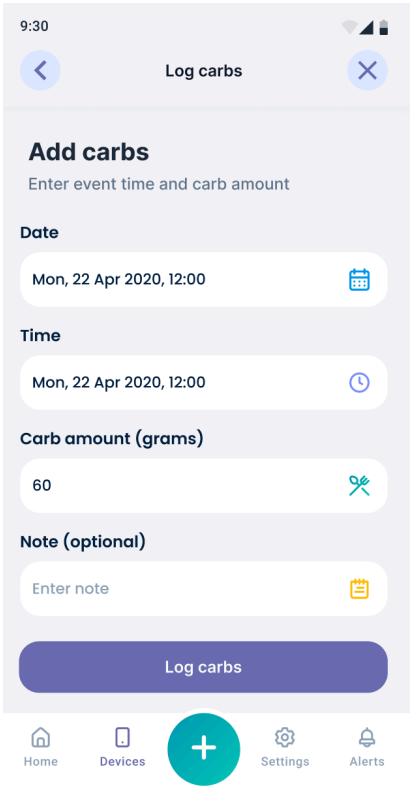

Рис. 3.19. Додавання вуглеводів

Після введення та збереження даних, на головному графі з’явиться іконка, що позначає цю подію. (рис.3.20).

Рис. 3.20. Відображення нової події на графі

У розділі «Пристрої» можна керувати помпою, а також дізнатись основну інформацію про неї, а саме:

– через скільки часу сенсор необхідно замінити;

– дату підключення сенсору;

– серійний номер;

– версії програмного та апаратного забезпечення.

Сенсор можна від’єднати або замінити (рис. 3.21).

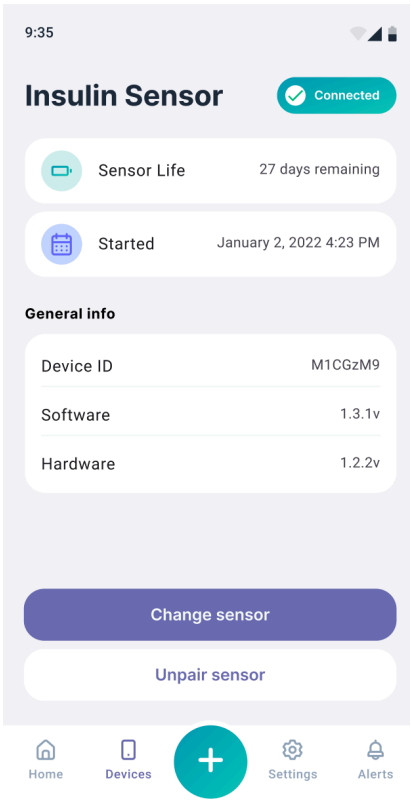

Рис.3.21. Вікно керування пристроями

З метою уникнення випадкового відключення сенсора, викликається діалогове вікно, на якому необхідно підтвердити розірвання з’єднання між сенсором та додатком. Діалогове вікно представлене на рисунку 3.22. Якщо сенсор буде відключено, весь обмін даними буде автоматично припинено, допоки з’єднання не відновиться. Якщо життя цього сенсору завершено, і його було відключено, при підключенні нового сенсору дані терапії продовжуються із моменту відключення попереднього сенсору, оскільки дані закріплюються не тільки за самим сенсором, а й за користувачем і завантажуються із бази даних. Якщо між відключенням

попередньої помпи та нової (або повторним підключенням попередньої) проходить не більше 15 хвилин, то дані для графу прогнозуються автоматично, такими чином інформація для користувача є послідовною та прогнозованою.

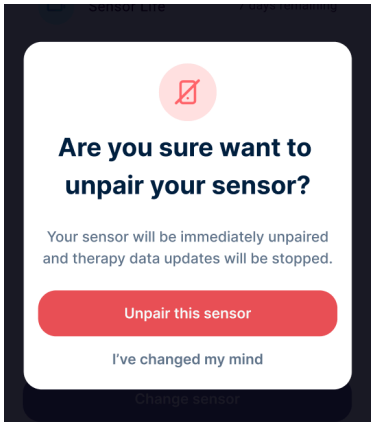

Рис.3.22. Підтвердження розірвання зв'язку

Після від'єднання сенсору з'являється повідомлення, що сенсор успішно від'єднано, а сам статус сенсору змінюється на «від'єднано» (рис.3.23). Також пролунає звуковий сигнал. На головному екрані інформація також буде змінена із статусом «втрата з'єднання».

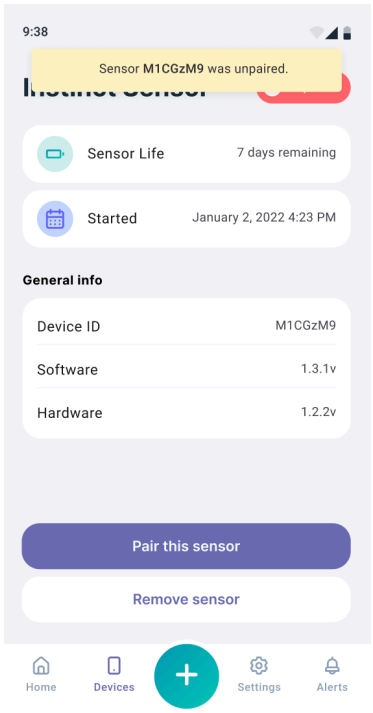

Рис. 3.23. Повідомлення про успішне від'єднання

3.5 Висновки до розділу

Даний розділ присвячено програмній реалізації проєкту. Окремо виділені питання отримання доступу до необхідних нативних функцій у додатку, підключення помпи до додатку, а також комунікація із ним.

У даному розділі також було продемонстровано основні екрани додатку, наведені програмні коди їх реалізації.

Для коректної роботи програми необхідно отримати доступи до нативних функції, більшість запитів пов’язана із *BLE* з’єднанням, проте є і менеш важливі, такі як дозвіл на разблокування пальцем.

Після успішного надання дозволів, необхідно пройти перевіркую. Перевірка доступів – незалежний від попереднього пункту процес. Це обумовлено тим, що помпа працює “у вакуумі”, вважаючи що всі необхідні доступи отримано. З метою уникнення помилок, при кожній новій взаємодії користувача з помпою виконується перевірка на наявні доступи, та повторний запит в разі відсутності останніх.

Програмна засіб надає можливість керувати елементом *IoT*, таким як інсулінова помпа. Її можна підключити, видалити, та перепідключити, а постійний моніторинг даних, що надходять із сенсору роблять моніторинг простим та зрозумілим.

ВИСНОВКИ

Кваліфікаційна робота присвячена темі «Програмний засіб управління елементом *IoT*». Актуальність даної теми обумовлена зростанню кількості елементів *IoT*, а також можливості, які вони надають. Неменш важливим фактором є підвищення впровадження пристроїв *IoT* у сферу охорони здоров'я та керування за критичними для життя показниками за допомогою останніх.

У кваліфікаційній роботі було створено програмний засіб, який дозволяє керувати таким пристроєм *IoT*, як інсулінова помпа, а також репрезентувати дані, що були отримані у зручному та зрозумілому для користувача форматі. Для вирішеня даної задачі було:

– оглянуто архітектуру Інтернету речей;
– описано протоколи та технології для взаємодії помпи та додатку;
– на основі Firebase реалізовано хмарну базу даних;
– реалізовано програмний засіб мовою програмування *Dart*.

У якості інструменту розробки було обрано мову програмування *Dart* у поєднанні із фреймворком *Flutter*, оскільки *BLE* – це нативний канал передачі даних, який за замовчуванням присутній як в *Android*, так і в *iOS*. В кожній операційній системі він різний, тому з метою покриття пристроїв обох операційних систем, необхідно розробляти два різних додатки, в яких буде реалізовано *BLE* канал. Фреймворк *Flutter* у поєднанні із мовою програмування *Dart* дозволяє отримати доступ до нативних каналів усіх платформ, на яких він підтримується.

У якості каналу з'єднання було обрано *BLE. BLE* передає менші пакети даних за короткі проміжки часу; як випливає з назви, *Bluetooth Low Energy* має набагато менше енергоспоживання, але з іншого боку, оскільки *Bluetooth Classic* не настільки обмежений, він має більший радіус дії і вищу пропускну здатність. У якості топології виступає *BLE Connect* з інсуліновою помпою в якості центрального вузла та передавачами, такі як датчик глюкози, глюкометр, смартфон та передавач, у якості периферійних пристроїв.

Було детально оглянуто технологію моніторингу глюкози, або *CGM* (*Continuous Glucose Monitor),* яка базується на архітектурі *IoT*. Система включає три основні компоненти, такі як портативний сенсорний пристрій, шлюз і серверну систему та використовується хворими на цукровий діабет для контролю рівня глюкози в крові. Було враховано особливості її роботи з метою подальшого застосування цих знань у розробці.

У якості бази даних було обрано хмарну БД *Cloud Firestore*, яка є одним із плагінів сервісу *Firestore*. *Firestore* має велику кількість бібліотек для мобільних пристроїв. Підключення бази даних надає можливість не зберігати дані локально на пристрої користувача та не робити додаток важким. Неменш важливим є те, що при втраті пристрою із додатком, дані завжди можна відновити, повторно встановивши додаток та авторизувавшись у нього.

Оскільки додаток є медичним та використовує елемент *IoT*, для коректної роботи необхідно було отримати певні доступи у нативних операційних систем. У разі відхилення запиту на доступ, програма не зможе отримати доступ до цих даних, якщо тільки ви пізніше дозвіл користувач не надасть доступ власноруч. Якщо у доступі було відмовлено, програма все одно буде запущена, але вона може працювати некоректо. Виконання функцій належним чином буде залежати від того, наскільки важливим є цей дозвіл для її роботи. Основними були запити на встановлення *BLE* з’єднання. Реалізація для операційних систем *iOS* та *Android* є різною та була розглянута у третьому розділі кваліфікаційної роботи. Недоліком є те, що смартфони *Apple* дозволяють завершити роботу додатку, якщо він був довгий час відкритий у фоновому режимі. Це може викликати серйозні наслідки для користувача, оскільки додаток має медичний характер. Нажаль, через політику компанії програмно це вирішити не можливо. Для оповіщення про втрату з’єднання реалізовано функцію надсиланя звукового сигналу про втрату *BLE*.

Реалізовано функціонал додатку, основними функціями якого є управління інсуліновою помпою, та відображення даних із неї.

На основі бажання або потреби користувач може:

– відключити сенсор;

– перепідключити підключений раніше сенсор;

– підключити новий сенсор;

– розірвати підключення з сенсором.

Виходячи з цих базових функцій, було розроблено процес відправки повідомлення про обраний стан на сенсор.

Дані з сенсора оновлюються кожні 5 секунд. На основі даних з сенсора, додаток може робити наступні висновки:

– рівень глюкози пацієнта занадто низький;

– рівень глюкози пацієнта занадто високий;

– пацієнту не рекомендується фізичне навантаження протягом певного часу;

– пацієнту не рекомендується вживання вуглеводів протягом певного часу;

– пацієнту рекомендується ввести дозу глюкози в певному обсязі.

Всі ці висновки робляться на основі порівнянь даних з сенсора з константними значеннями, та на основі введених користувачем даних про себе: дата народження, стать, зріст, вага.

Підбиваючи підсумки, можна підкреслити основні переваги даної програми: повне керування сенсором інсулінової помпи та зрозумілий інтерфейс, який надає наочне представлення що відбувається із помпою та із станом здоров’я пацієнта на даний момент.

Результати розробки рекомендується впровадити у сферу охорони здоров’я, так як це зможе суттєво підвищити рівень надання медичних послуг та поліпшити життя людей, хворих на цукровий діабет. Дана робота потребує проведення клінічних тестів та тестування із різними видами сенсорів.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *The Concept of IoT (Internet of Things)* [Електроний ресурс] – Режим доступу до ресурсу: *https://swifttalk.net/2021/10/06/the-concept-of-iot-internet-of-things/*
2. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/*
3. *IoT Ecosystem: Top 7 Components (+ Bonus)* [Електроний ресурс] – Режим доступу до ресурсу: *https://sumatosoft.com/blog/iot-ecosystem-top-7-components*
4. *Internet of Things: Architectures, Protocols, and Applications* [Електроний ресурс] – Режим доступу до ресурсу: *https://www.hindawi.com/journals/jece/2017/9324035/*
5. *What is iot architecture?* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.avsystem.com/blog/what-is-iot-architecture/*
6. *Creating secure IoT device identities* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.intertrust.com/resources/secure-iot-device-identities/*
7. *IoT Applications in Healthcare* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.wirelesswatchdogs.com/blog/iot-applications-in-healthcare*
8. *The Internet of Things for Healthcare:Applications, Selected Casesand Challenges* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.researchgate.net/publication/348220261_The_Internet_of_Things_for_Healthcare_Applications_Selected_Cases_and_Challenges*
9. *The Internet of Things for Healthcare:Applications, Selected Casesand Challenges* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.researchgate.net/publication/348220261_The_Internet_of_Things_for_Healthcare_Applications_Selected_Cases_and_Challenges*
10. *Internet of Things: Everything You Must Know About IoT* [Електронний ресурс] – Режим доступу до ресурсу: *https://positive-stud.medium.com/internet-of-things-everything-you-must-know-about-iot-879e60853577.*

11. *Dart overview* [Електронний ресурс] – Режим доступу до ресурсу: *https://dart.dev/overview*
12. *Flutter documentations* [Електронний ресурс] – Режим доступу до ресурсу: *https://flutter.dev/docs*
13. *Mobile Operating System Market Share Worldwide* [Електронний ресурс] – Режим доступу до ресурсу: *https://gs.statcounter .com/os-market-share/mobile/worldwide*
14. The Basics of Bluetooth Low Energy [Електронний ресурс] – Режим доступу до ресурсу: *https://www.novelbits.io/basics-bluetooth-low-energy/*
15. IoT-based continuous glucose monitoring system: A feasibility study [Електронний ресурс] – Режим доступу до ресурсу: *https://www.sciencedirect.com/science/article/pii/S1877050917310281?ref=pdf_download&fr=RR-2&rr=7677e00ffceb0063*
16. IoT Based Mobile Medical Application for Smart Insulin Regulation [Електронний ресурс] – Режим доступу до ресурсу: *https://ieeexplore.ieee.org/document/8869227*
17. *How to upload data to Firebase Firestore Cloud Database* [Електронний ресурс] – Режим доступу до ресурсу: *https://medium.com/@devesu/how-to-upload- data-to-firebase-firestore-cloud-database-63543d7b34c5*
18. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. правила оформлення. – 88 с.

ДОДАТОК А

Лістинг коду виклику функції Setup для різних операційних систем

```
if (Platform.isIOS) {
    await checkPermission();
  } else {
    setup();
  }
 }
 Future<void> checkPermission() async {
  if (!await bloc.isBluetoothPermissionEnable()) {
    showSettingsDialog();
  } else {
    setup();
  }
 }
      void setup() {
  if (bloc.foundDevices == null) {
    showScanProgress();
    bloc.bleDevicesStream.listen(
      (item) => setState(() {
        hideScanProgress();
      }),
      onError: (Object error) => handleScanError(error),
    );
    bloc.scanStatusStream.listen(
      (ScanStatus scanStatus) => onScanStatus(scanStatus),
    );
    bloc.initialize();
  }
```

```
  }
void showSettingsDialog() {
   showDialogWithActionOnContext(
     context: context,
     title: AppLocalizations.of(context)
        .LS_Transmitter_Pairing_Alert_Permissions_Title,
     leftButtonText: AppLocalizations.of(context)
        .LS_Transmitter_Pairing_Alert_Permissions_Settings_button,
     leftButtonAction: () {
       Navigator.of(context, rootNavigator: true).pop();
       Navigator.of(context)
          .popUntil(ModalRoute.withName(PairTransmitterPage.path));
       PermissionManager().openAppSettings();
     },
     rightButtonText: AppLocalizations.of(context).LS_Cancel,
     rightButtonAction: () {
       showSettingsDialog(); //Close pair screen
     },
   );
  }
```