

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Проект програмно-апаратного комплексу дис-
танційного моніторингу пацієнтів із серцевою недоста-
тністю»

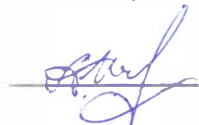
Виконав студент 2-го курсу групи КІм-22-1
спеціальності 123 «Комп'ютерна інженерія»

 М.Г. Венедчук

Керівник, старший викладач кафедри комп'ютерних
наук

 Петросян Р.В.

Рецензент, доцент кафедри КІ та КБ, к.т.н

 Шелуха О. О.

Житомир – 2023

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко

13 грудня 2023 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Венедчук Михайло Геннадійович

Тема роботи: **«ПРОЄКТ ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС ДИ-
СТАНЦІЙ-НОГО МОНІТОРИНГУ ПАЦІЄНТІВ ІЗ СЕРЦЕВОЮ НЕДО-
СТАТНІСТЮ»**

затверджена Наказом університету від 13.10 2023 р. № 127/с

Термін здачі студентом закінченої роботи 11.12 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Об'єкт дослідження – моніторинг та візуалізація дистанційного моніторингу пацієнтів із серцевою недостатністю.

Предметом дослідження є використання сучасних технологій розробки веб-додатків та організації дистанційного моніторингу пацієнтів із серцевою недостатністю. Обладнання та програмне забезпечення:

1. VSCode

2. Arduino IDE
3. Модуль ESP32 WiFi Bluetooth WROOM-32U CH9102X
4. Модуль серцевого ритму ЕКГ на базі чипу AD8232
5. Перемички для макетування 5 шт.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Бродський Ю. Б.	02.11.23	02.11.23
2	Бродський Ю. Б.	12.10.23	12.10.23
3	Бродський Ю. Б.	16.12.23	16.12.23

Студент  М.Г. Венедчук
Керівник  Р.В. Петросян

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз отриманого завдання та уточнення поставлених завдань	2 листопада 2023 – 5 листопада 2023	Виконано <i>М.Г. Венедчук</i>
2	Пошук, огляд та аналіз аналогічних розробок	6 листопада 2023 – 14 листопада 2023	Виконано <i>М.Г. Венедчук</i>
3	Аналіз літературних джерел	14 листопада 2023 – 20 листопада 2023	Виконано <i>М.Г. Венедчук</i>
4	Проектування структури програми, написання програмного коду на базі Arduino	21 листопада 2023 – 25 листопада 2023	Виконано <i>М.Г. Венедчук</i>
5	Розробка інтерфейс додатку засобами React Native	26 листопада 2023 – 14 грудня 2023	Виконано <i>М.Г. Венедчук</i>
6	Оформлення пояснювальної записки	14 грудня 2023 – 20 грудня 2023	Виконано <i>М.Г. Венедчук</i>
7	Підготовка презентації та виступу	21 грудня 2023 – 22 грудня 2023	Виконано <i>М.Г. Венедчук</i>
8	Захист	30 грудня 2023	Виконано <i>М.Г. Венедчук</i>

Студент *М.Г. Венедчук* М.Г. Венедчук

Керівник *Р.В. Петросян* Р.В.Петросян

АНОТАЦІЯ

Кваліфікаційна робота складається з вебсервера для Arduino, завданням якого є передача відцифрованого сигналу по Bluetooth; та клієнтського додатку, який показує ці дані на графіку, та у текстовому вигляді, з можливістю збереження отриманих даних у файл. Пояснювальна записка до роботи містить 64 сторінок та 28 ілюстрацій.

Метою роботи є розробка системи моніторингу серцевої активності.

В результаті роботи набуто навичок роботи з Arduino, створено вбудовану програму за допомогою якої відбувається передача отриманих відцифрованих даних по Bluetooth. Створено додаток за допомогою бібліотек React, які дають змогу підключати Arduino пристрій по мережі Bluetooth стандартом.

Ключові слова: ARDUINO, BLUETOOTH, BLE, REACT-NATIVE, NATIVE MODULE, LINECHART, FIRMWARE.

ANNOTATION

The qualifying work consists of a web server for Arduino, the task of which is to transmit a digitized signal via Bluetooth; and a client application that shows these data graphically and in text form, with the possibility of saving the received data to a file. The explanatory note to the work contains 64 pages and 28 illustrations.

The purpose of the work is to develop a system for monitoring cardiac activity.

As a result of the work, the skills of working with Arduino were acquired, a built-in program was created with the help of which the received digitized data is transferred via Bluetooth. An application was created using the React libraries, which allow you to connect an Arduino device via a standard Bluetooth network.

Keywords: ARDUINO, BLUETOOTH, BLE, REACT-NATIVE, NATIVE MODULE, LINECHART, FIRMWARE..

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП	9
РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ	11
1.1. Методи вимірювання біосигналів серця	11
1.2. Аналіз існуючих апартних рішень.....	11
1.3. Аналіз апаратного забезпечення та роботи з ним.....	13
1.4. Аналіз програмного забезпечення та роботи з ним	18
Висновок до розділу 1	24
2.1. Аналіз засобів для проектування додатку	25
2.2. Розробка вбудованої програми для роботи з Bluetooth та AD8232.....	28
2.3. Аналіз реалізації роботи з Bluetooth у додатку	30
2.4. Вибір бібліотеки для додатка, та створення Bluetooth клієнта.....	31
2.5. Особливості зв'язку BLE	32
2.6. Розробка функціоналу для відображення даних	34
2.7. Розробка функціоналу для отримання даних по Bluetooth	39
Висновок до розділу 2	41
РОЗДІЛ 3. ТЕСТУВАННЯ ТА НАЛАШТУВАННЯ ПРОГРАМНО- АПАРАТНОГО КОМПЛЕКСУ	42
3.1. Налаштування роботи з додатком	42
3.2. Приведення отриманих даних до одного формату	43
3.3. Налагодження відображення графічних даних, розробка функціоналу для збереження та показу збережених даних	46

Висновок до розділу 3	49
ВИСНОВКИ.....	50
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	51
ДОДАТКИ.....	52

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NPM – Node Package Manager

BLE – Bluetooth Low Energy

APK – Android Package Kit)

JSON – JavaScript Object Notation

MAC – Media Access Control

UUID – Universally Unique Identifier

IoT – Internet of Things

AD – Analog Devices

ADC – Analog-To-Digital Converter

АЦП – Аналого-цифровий перетворювач

ЦАП – Цифро-Аналоговий Перетворювач

DAC – Digital-to-Analog Convertor

API – Application Programming Interface

JS – JavaScript

JSON – JavaScript Object Notation

RSSI – Received Signal Strength Indication

ВСТУП

Актуальність теми. Тіло людини, нажаль, з часом не стає кращим, тому потребує догляду. Особливу увагу потребує серцево-судинна системи, так як великий відсоток смертей пов'язаних саме з нею, наприклад: від серцевої недостатності, серцевого нападу (відомого, як інфаркт міокарда) тощо.

В Україні було реформовано швидку медичну допомогу у 2016 році, в неї входить нове для нас поняття – парамедик. Парамедик – медичний працівник, що зазвичай надає допомогу при невідкладних станах на догоспітальному етапі (у тому числі виконує кардіографію). Також згідно Закону України «Про екстрену медичну допомогу», прописано ряд немедичних професій, які зобов'язані надавати домеди-чну допомогу людині у невідкладному стані, до неї входить дефібриляція в автоматичному режимі. Громадська ініціатива “РеаніМетро” добилася встановлення де-фібриляторів у громадських місцях.

В результаті глобалізації доступ до інформації не є проблемою, поширення даних набуло безпрецедентної швидкості, і кількості; не стоять на місці і технології відповідальні за їх обробку.

Серед таких даних є і медичні, цінність яких в тому, що їх теж можна проана-лізувати за допомогою машинного навчання, хоча є і відповідні ресурси де лікарі обговорюють електрокардіограми, серед таких є група у Facebook : «ЕКГ для заці-кавлених».

Аналіз таких даних відноситься до біоінженерії, яка об'єднує медицину, біо-логію та інженерію. В її рамках, розробляються новітні технології для діагностики та лікування захворювань, зокрема із застосуванням методів штучного інтелекту, розширюючи таким чином її можливості. До створення діагностичних пристроїв відносять біомедичну інженерію, до аналізу даних біоінформатику.

Поєднання досягнень медицини та техніки відкриває нові можливості у бо-ротьбі із серцево-судинними хворобами, та не тільки ними. Виявляти їх ознаки на ранніх стадіях, запобігати прогресуванню, підвищуючи, таким чином, шанси на одужання.

Обмін такою інформацією відповідає ідеї ноосфери, яка підкреслює спільну інтелектуальну силу в розвитку та забезпеченні здоров'я людства. Поява медичної, саме україномовної, спільноти для обміну такої інформації, змусило задуматись про ідеї, які підіймались понад 100 років тому. Звісно в ті часи люди теж обмінювались інформацією на відстані, проте зовсім не з такою швидкістю. Наприклад взяти мовний бар'єр, так як вивчення мов стало теж доступніше з появою інтернету. Для спілкування не обов'язково, навіть, досконало володіти мовою співрозмовника, онлайн перекладачі набагато спростили доступ до перекладів, які раніше являли собою пошук еквівалентних слів у паперових словниках. Це все досягнення спільного розуму, який розвивається відповідно до потреб людства.

В рамках даної роботи, порушується проблема підвищення рівня здоров'я всього населення, шляхом збору такої інформації, її збереженні, відповідно і передачі.

Об'єкт дослідження – моніторинг та візуалізація дистанційного моніторингу пацієнтів із серцевою недостатністю.

Предметом дослідження є використання сучасних технологій розробки веб-додатків та організації дистанційного моніторингу пацієнтів із серцевою недостатністю.

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ

1.1. Методи вимірювання біосигналів серця

Існують такі методи вимірювання серцевої активності:

- Електрокардіограма (ЕКГ) - це найпоширеніший метод, який реєструє електричну активність(сигнали) серця, що виникають під час скорочення серцевого м'яза. Проводиться як у стані спокою, так і під час фізичних навантажень (велоергометрія, тредміл-тест), за допомогою електродів, розміщених на грудній клітці, або на кінцівка. ЕКГ вимірює частоту серцевих скорочень, ритм і електричну провідність.
- Холтерівське моніторування - це безперервний запис ЕКГ протягом тривалого періоду часу. Він дозволяє виявляти будь-які зміни в роботі серця. Може поєднуватися з вимірюванням артеріального тиску. Являє собою пристрій для тривалого носіння(24-48 годин). Дозволяє виявити пароксизмальні порушення ритму, ішемічні зміни.
- Ехокардіографія - ультразвукове дослідження, яке створює зображення серця і дозволяє оцінити його структуру(розміри порожнини серця, товщину стінок, скоротливу функцію міокарда). Воно може виміряти частоту серцевих скорочень і їх силу. Проводиться у режимі реального часу. При підозрі на ураження клапанів серця, виявлення вроджених вад серця.
- Тонометрія - вимірювання артеріального тиску за допомогою манжети, яка накладається на руку. Артеріальний тиск залежить від сили і частоти серцевих скорочень.
- Пульсоксиметрія - неінвазивний метод, який вимірює насичення киснем артеріальної крові. Зміни насичення можуть відображати зміни в серцевому ритмі або частоті серцевих скорочень.

1.2. Аналіз існуючих апаратних рішень

На даний час існує кілька популярних пристроїв для реєстрації електрокардіограми для особистого користування.

KardiaMobile - це портативний одноканальний ЕКГ монітор для особистого користування розміром з кредитну картку. Він складається з двох електродів, які підключаються до смартфона чи планшета за допомогою Bluetooth. Дозволяє пацієнтам самостійно реєструвати ЕКГ протягом 30 секунд шляхом прикладання пальців до електродів. Дані передаються на смартфон, де за допомогою програми

відбувається їх аналіз та виявлення порушень ритму серця. Пристрій на рисунку 1.1.

Переваги: простота використання, портативність.

Недоліки: обмежена тривалість запису.



Рис. 1.1 KardiaMobile

ECG Belt (BioRadio) - це бездротовий грудний пояс з вбудованими електродами для довготривалого моніторингу ECG. Дані передаються на смартфон або ПК в режимі реального часу по Bluetooth. Пристрій на рисунку 1.2.

Переваги: зручність, можливість тривалого використання, можливість моніторингу під час руху.

Недоліки: висока вартість.



Рис. 1.2 ECG Belt

SEEQ MCT (Medtronic) - це пристрій для моніторингу ЕКГ у формі смарт-смужки, який кріпиться на грудній клітці. Дозволяє безперервно записувати

одноканальну ЕКГ протягом до 30 днів. Дані аналізуються в центрі моніторингу компанії, та в деяких випадках надсилаються лікарям. Пристрій на рисунку 1.3.

Переваги: тривалий запис ЕКГ, зручність, компактність, водонепроникність.

Недоліки: обмежена кількість каналів, висока вартість.



Рис. 1.3 ECG Belt

1.3. Аналіз апаратного забезпечення та роботи з ним

Для розробки апаратної частини для дистанційного моніторингу пацієнтів із серцевою недостатністю використовується мікроконтролер ESP-WROOM-32, який виробляється компанією Espressif Systems. Він дозволяє забезпечувати можливість підключення до бездротових мереж і виконання завдань Інтернету речей (IoT). Для його роботи може використовуватись апаратна обчислювальна платформа Arduino, перевага якої у простоті інструментів для розробки. Архітектура плат дозволяє, використовувати інші периферійні пристрої (датчики, світлодіоди, кнопки), або плати розширення, за допомогою цифрових та аналогових входів/виходів[1]. В основі ESP32 мікроконтролер, який містить в собі завантажувач, що дозволяє не використовувати зовнішніх програматорів. Пристрій на рисунку 1.4.



Рис. 1.4 Зовнішній вигляд ESP-WROOM-32

Архітектура плат дозволяє, використовувати інші пристрої як на передачу(згенерованих) даних (звук на аудіосистему, фотозображення на цифрові матриці, керування окремими світлодіоди), так і на прийом (передавати інформацію про: натискання кнопки; за допомогою додаткових модулів, зняття температури, витоків газу, відеопотоку).

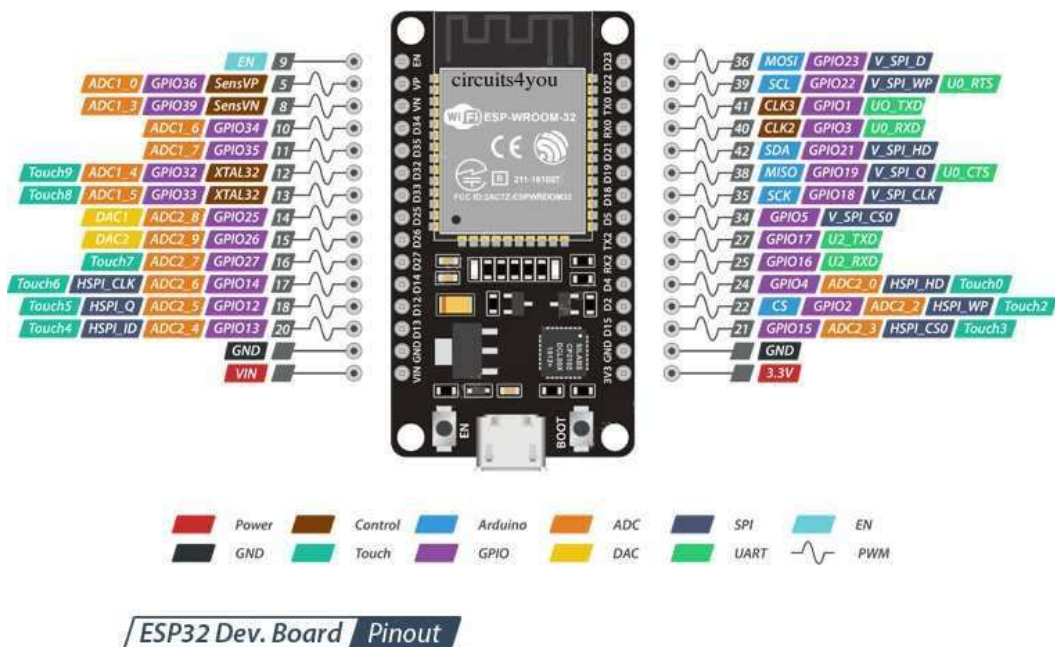


Рис. 1.5 Схема пінів ESP-WROOM-32

Для цього ESP32 має 34 цифрових пінів, які можуть використовуватися як на вхід, так і на вихід (GPIO - General Purpose Input/Output). Схема виходів на рисунку 1.5.

ESP32 також має 15 аналогових входів, ці піни можуть зчитувати аналогові сигнали, наприклад, з датчиків, і перетворювати їх на цифрові значення за допомогою АЦП (ADC - analog-to-digital converter).

Кожен аналоговий вхід може бути розглянутий як "канал". Канал - це конкретний аналоговий вхід, який може бути зчитаний. Та не одночасно. Справа в тому, що ESP32 має тільки 2 АЦП, один з яких має обмеження у разі використання Wi-Fi, тому варто це враховувати, що звісно, не заважає зчитувати кожен канал по черзі. В межах цього проекту треба буде приймати дані з іншого модуля(AD8232), тому це буде використовуватися.

Серед переваг можна відмітити ціну, вбудований Wi-Fi, Bluetooth; активну спільноту, так як часто використовується аматорами, та має багато прикладів використання, які підключаються в самому інтегрованому середовищі розробки. Так як для програмування треба тільки кабель, який на початках може виконувати і функцію живлення, крім програмування вбудованої програми, яка зазвичай пишеться на C/C++, хоча є і варіанти використання інших мов, для них можна, для зручності використовувати такі IDE(VS Code, Thonny IDE, Arduino Lab).

В рамках цього проекту було встановлене стандартну «Arduino IDE», яка має компілятор та можливість передачі прошивки в плату, у разі успішної компіляції. Також має можливість завантажувати бібліотеки, та готові приклади або користуватись встановленим разом з середовищем[1]. Для створення програми необхідно пишеться функція Setup, яка при ввімкненні живлення, виконається на початку роботи тільки раз; на відміну від функції Loop, яка буде виконуватися циклічно, до вимкнення живлення.

Для зчитування електричної активності серця скористаюсь модулем AD8232. У модулі AD8232 є окремі виводи RL, LA, RA для підключення електродів, а також mini-jack для стандартних кабелів, які йдуть в комплекті.

Переваги AD8232:

- Низька вартість
- Простота використання
- Невеликі габарити

- Низьке енергоспоживання

Недоліки:

- Обмежена точність вимірювань
- Потреба у зовнішніх компонентах (резистори, конденсатори)
- Одноканальний

Аналогічні рішення:

1. Модуль ECG BMD101(Neurosky) – можливість роботи 2 каналні ЕКГ, має вбудований акумулятор. Має кращу точність, але вищу ціну. Виріб побудований на ньому на рисунку 1.6.



Рис. 1.6 Виріб на основі BMD101

2. ADS1292 (Texas Instruments) - чіп для ЕКГ з 2 каналами та з 3,5,6 відведеннями. Містить АЦП 24 біт. Складніший у реалізації, теж потребує мікроконтролера.
3. Модуль e-Health (Cooking-hacks) - комплект для створення медичних пристроїв, на рисунку 1.7. Містить сенсори для ЕКГ, пульсоксиметрії тощо. Дорожчий, але більш функціональний. Позиціонує як open-source.



Рис. 1.7 Модуль e-Health (Cooking-hacks)

Отже, AD8232 - доступне і просте рішення для створення одноканального ЕКГ. Для складніших багатоканальних систем краще підійдуть спеціалізовані чіпи або модулі, незважаючи на вищу ціну.

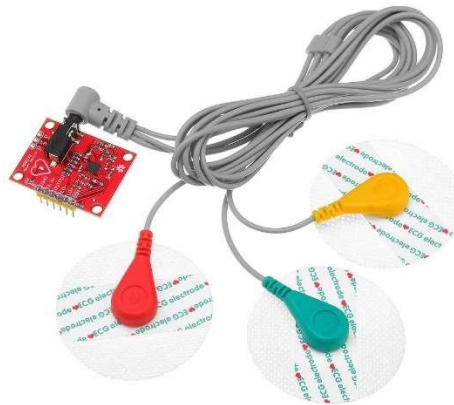


Рис. 1.8 Модуль AD8232 з трьома підключеними відведеннями

Датчик AD8232, на рисунку 1.8, має 3 електроди, які вимірюють електричну активність серця:

- Електроди RA (права рука) та LA (ліва рука) - вимірюють напругу між цими точками. Це основний ЕКГ сигнал.
- Електрод RL (права нога) - слугує як загальний, відносно якого вимірюється напруга на RA та LA.

Датчик перемикається між вимірами напруги RA-RL та LA-RL, формуючи один сигнал ЕКГ. Хоча фізично електродів 3, але сигнал ЕКГ формується як послідовність вимірів між 2 парами електродів і передається як один аналоговий сигнал на ESP32, тобто датчик одноканальний. Основні компоненти:

- Інструментальний підсилювач, має фіксоване підсилення 100;
- Операційний підсилювач - використовується для створення додаткового другого порядку низькочастотного фільтру. На платі встановлений коефіцієнт підсилення 11.
- RLD(Driven Right Leg circuit)– для подавлення шумів (синфазного сигналу).
- Фільтри, які дозволяють позбутися завад(шумів), і виділити саме корисний сигнал ЕКГ, з його типовою смугою частот
- Компоненти для виявлення від'єднання електродів.
- Загальне підсилення ланцюга - $100 \text{ (інструментальний підсилювач)} * 11 \text{ (операційний підсилювач)} = 1100$.

ESP32 може приймати сигнал з цього модуля, для цього його треба підключити до аналогового піна(тому, що з AD8232 він знаходить саме в такому вигляді), через нього відбувається зчитування напруги (в даному випадку біомедичного сигналу). Та конвертувати його за допомогою АЦП, вхідна напруга якого, в ESP32, до 3.3В, та отримується числове значення в діапазоні 0 до 4095(тому, що 12-розрядний АЦП), яке може відправитись різними способами: через UART (Serial) або Wi-Fi/Bluetooth на зовнішній пристрій.

1.4. Аналіз програмного забезпечення та роботи з ним

Ідея передачі в реальному часі даних з модуля AD8232 не нова[2]. Для приймання даних смартфоном використовується додаток BluetoothGraphics. Він має такі функції:

1. Сканування Bluetooth пристроїв, для можливості приєднання пристроїв з різними іменами, без необхідності жорсткого кодування у вбудованій програмі для мікроконтролера.
2. Вмикання Bluetooth, за допомогою до звернення до системних API
3. Підключення(створення пари)

4. Відображення вхідних даних можливе у двох форматах: текстовому, де дані відображаються у чисельному форматі; та графічному, де можливе відображення одразу декілька змінних. Приклад на рисунках 1.9 та 1.10.

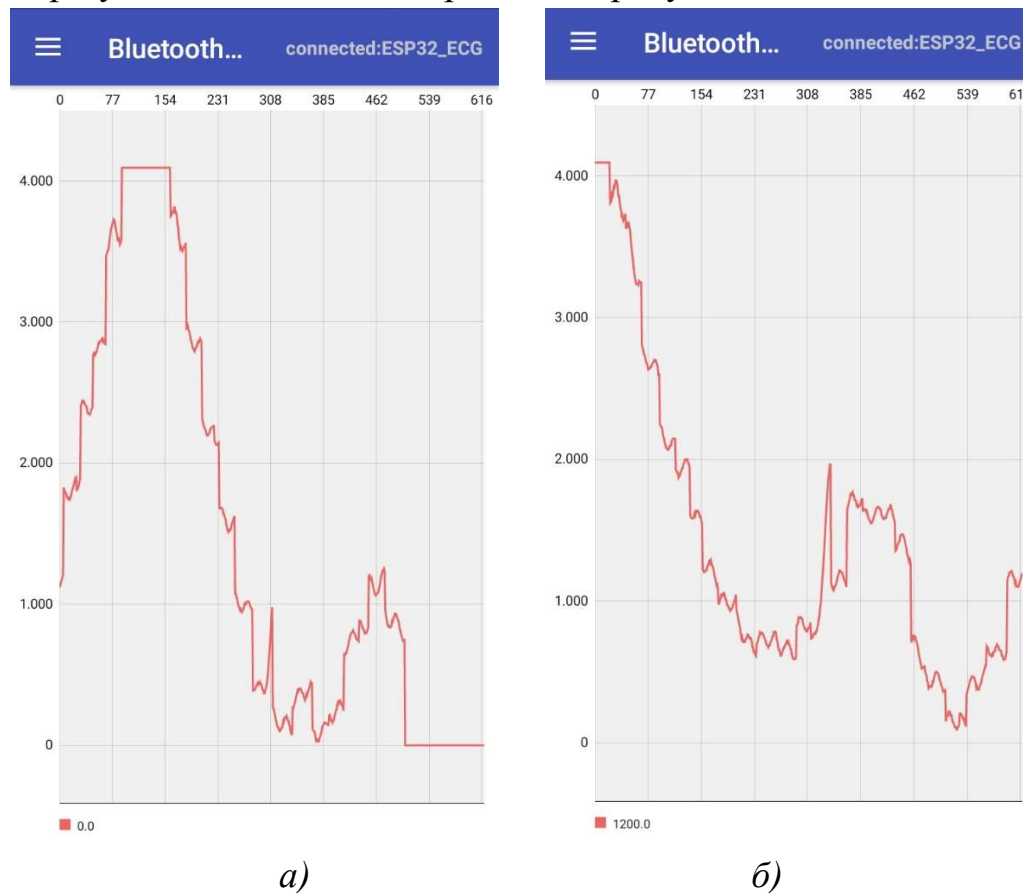


Рис. 1.9 Додаток, який приймає сигнал з ESP32, демонструючи напругу у графічному форматі

Формат відправлених даних має бути відповідним до інструкції, фрагмент якої у лістингу 1.1, яка йде до додатку: “This sequence is must be like that: Evalue1,value2,value...\n”. Тобто кожен рядок має починатися з “E” символу. Що й має бути зроблено при відправці даних, відповідно до прикладу програмного коду з інструкції у додатку, для прошивки. Результат відправки на рисунку 1.10

Лістинг 1.1 Відправка даних відповідно до інструкції

```
Serial.print('E');  
Serial.print("18.21") ;  
Serial.print(",");  
  
Serial.print("19");  
Serial.print("\n");
```

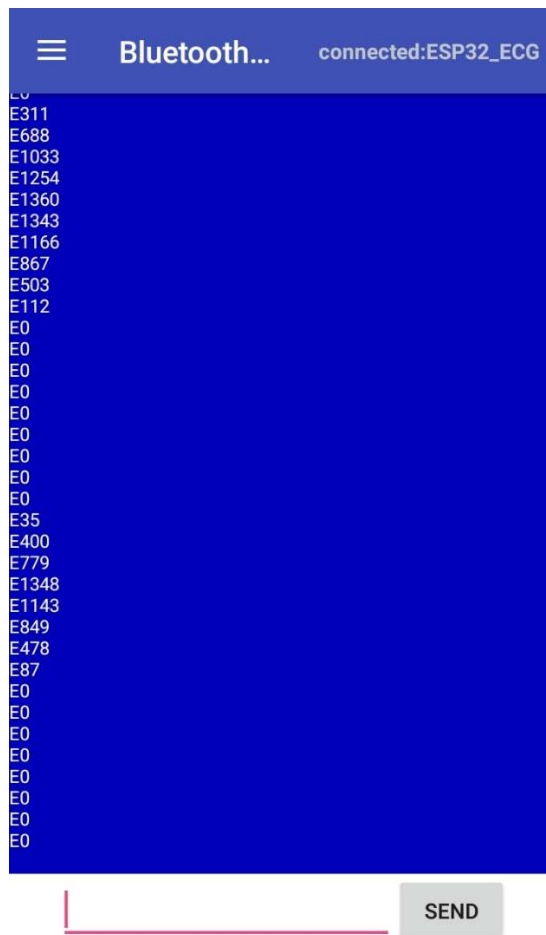


Рис. 1.10 Додаток, який приймає сигнал з ESP32, демонструючи напругу у тестовому форматі

Або ближче зі статті, використовуючи функцію для оцифровування аналогового сигналу: `ESP_BT.println(analogRead(A0));`

5. Додаток має функцію збереження потоку у файл, яка дає можливість експортувати дані

Він не має відкритого, публічного коду. Тому завантажую його на смартфон, також додаток для генерування файлу APK “Apk extractor”, щоб мати змогу розібрати його за допомогою декомпілятора. Це дозволить побачити декопільований `smali` код[3]. Також не менш цінним, є побачити які інструменти (бібліотеки) та дозволи використовуються. Звісно, всі дозволи можуть не відображатись, а запитуватись у процесі роботи з додатком(`Runtime Permission`), перед викликом конкретної функції, для якої потрібен буде доступ. Проте буде хоч якась поверхова інформація[3].

В випадку вищезгаданого додатку, не буде можливості навіть ознайомитись з інтерфейсом, без ввімкненого Bluetooth. З файлу, який можна витягнути відкривши APK-файл, як архів, в кореневій папці буде файл, який міститься в кожному Android додатку: AndroidManifest.xml. У ньому можна бачити, що запитуються дозволи, які в подальшому можна буде використати у своєму додатку. Формат XML, тому вся інформація у тегах та атрибутах.

Дозволяють вмикати Bluetooth, створювати пару, та проводити пошук доступних пристроїв:

- BLUETOOTH;
- BLUETOOTH_ADMIN;

Для збереження отриманих даних у файл:

- READ_EXTERNAL_STORAGE;
- ACCESS_NETWORK_STATE;

Для запобігання переходу в сплячий режим:

- WAKE_LOCK(в даному додатку необхідно для того, щоб продовжувати зберігати у файл, навіть після блокування, тільки з ініціативи користувача. На відміну від PARTIAL_WAKE_LOCK, в якому процесор буде працювати, та пристрій зможе при цьому перейти в сон, самостійно)

Після вказування дозволів у цьому файлі описуються компоненти додатку:

- activities – екрани, з якими користувач взаємодіє;
- services – класи, які працюють у фоновому режимі;
- receivers – класи, які взаємодіють з подіями системного рівня[2].

У тегах activities, є атрибут “android:name”, де вказуються назви класів, які використовуються, по їх назвах можна спробувати знайти, вихідний код, та використати його. Так було знайдено клас DeviceListActivity, який використовується у прикладі від Google Open Source, BluetoothChat. Цей додаток є демонстрацією того, як використовувати Bluetooth у розробці, та є готовим рішенням, яке показує доступні пристрої, та дозволяє під’єднатися до вибраного. Компанія Google має офіс відкритого коду(OSPO), який має на меті заохотити та підтримати розробників

відкритого коду. Можна побачити, що такі самі дозволи для роботи з Bluetooth, є і в цього додатку.

За допомогою декомпілятора Java decompiler online, який дозволяє побачити вихідний код Java, що набагато зручніше за smali. Також ресурс дозволяє завантажити його, для того, щоб відкрити в редакторі, та користуватися зручною навігацією, пошуком змінних у файлах.

Для перегляду декомпільованого коду можна скористатись Android Studio. Переглянувши файли стало зрозуміло, що графік малюється за допомогою MPAndroidChart.LineChart.

В процесі використання додатку, зроблено висновки, що іноді запис може відбутися некоректно, наприклад замість чисел у файл записуватимуться пусті рядки, або використовуватиметься такий же формат, який відправляється з Arduino.

Звісно формат теж не дуже інформативний, так як не містить даних про час, зменшуючи можливості для обробки сигналу.

Знайшовши файл DeviceListActivity, з OpenSource коду від Google, зрозумів, що він є частиною цілого додатка під назвою BluetoothChat, створеного в 2009 році, та поширюваного під ліцензією вільного програмного забезпечення Apache License, про що свідчить підпис вгорі файлу. Це додаток надає приклад того, як створювався тривалий зв'язок, для надсилання повідомлень між пристроями[4]. Крім офіційного ресурсу з репозиторієм[5], цей файл з проектом є і на Github, де зберігається архів проектів компанії, які вже не підтримуються її розробниками[6], про що й свідчить відповідний напис біля назви репозиторію, на рисунку 1.11.



Рис 1.11. Навігаційний рядок, який свідчить про припинення підтримки

Це також означає, що перевикористання можливо, проте, тільки в окремо створеному проекті, використовуючи форк(розгалудження)[7]. Про створення такого, в цьому випадку, й написано в архівному репозиторії, у файлі README.

Такі файли зазвичай створюють для подальшого вводу інших користувачі до проекту, можуть бути як пояснення до чогось, що не є очевидним.

Приклади використання написані на мові Kotlin, що дає зрозуміти, що підтримка проекту присутня. Також є ймовірність, що при побудові проекту не буде помилок, як це часто буває, якщо вона відсутня, ускладнюючи зворотну сумісність, пошуком помилок, та виправленню їх. Що потребує повторної збірки проекту, до наступної помилки, таким чином кешуючи процес, збільшуючи розмір відповідальної за це папки в корені диску: «.gradle»(звісно є можливість почистити її, та наступна збірка буде займати більше часу). Зазвичай, це пов'язано з тим, що у створених залежностях, є проекти, які не підтримуються, тому й виникає необхідність змінювати їх файли пов'язані з системою автоматичної збірки(Gradle). Та це не стосується цього випадку, так як основна мета, показати роботу з системними речами, тому й такі залежності відсутні. Проте файли збірки можуть бути не сумісними з останніми версіями Gradle, тому може бути

Прочитавши вихідний код стало зрозуміло, що додаток було перероблено під новий стандарт Bluetooth Low Energy. Проте використання першої версії додатку дає гарантію, на отримання даних від ESP32, відповідно до додатку зазначеного вище, звісно це не є аргументом, який покриває використання BLE стандарту, проте в рамках роботи досліджено можливість його використання. За допомогою пошуку в Github, можна дізнатися, чи були в когось спроби реалізувати такий код в React Native додатку, знайшовши два проекти, вирішив інтегрувати код, та зібрати проект. Та результат не втішний, при заході в додаток, застосунок завершує роботу, на рисунку 1.12, в якості експерименту, зробив декілька сторінок, щоб переконатися, що проблема точно в коді, який походить від Bluetooth Chat. При генеруванні відповідної сторінки, додаток завершує роботу, що відображено на рисунку 2. Звісно, при великій необхідності, можна було б спробувати зменшити версії залежностей, та розібратись в конфліктах, читаючи помилки за допомогою запуску на емуляторі, або перехоплювати виключення. Та реалізація нових стандартів звісно, є більш доцільнішою. Хоч перший варіант реалізації раніше і вважався за найоптимальніший, та з розвитком технологій, вже ним не є.

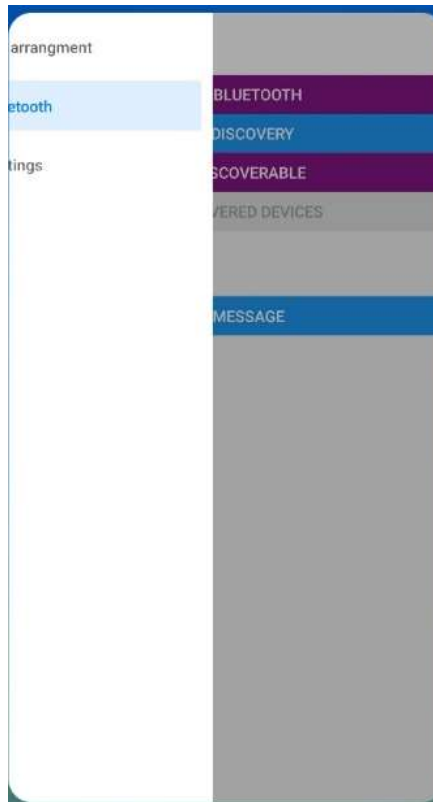


Рис. 1.12 Додаток, за мить до завершення роботи, не завантаживши решту сторінки

Висновок до розділу 1

Було розглянуто методи вимірювання серцевої діяльності. Також проаналізовано апаратні пристрої від приватних компаній, які продаються для моніторингу серцевої активності. Деякі з них дозволяють передавати дані на сервера компаній для їх аналізу, або на смартфон власника. Проаналізовано можливі рішення на платформі Arduino, які дозволяють оцифровувати сигнал. Проведено розбір програмних продуктів для приймання такого сигналу, вивчення технологій: бібліотек, інтерфейсних компонентів, методи передачі та прийому. Також знайдено частини коду існуючого додатку, який у відкритому доступі, та колись був еталоном написання рішень з технологією Bluetooth.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

2.1. Аналіз засобів для проектування додатку

React – це бібліотека JavaScript, створена компанією Facebook для розробки інтерфейсів користувача. Вона дозволяє створювати складні інтерфейси шляхом компонування малих частин коду, відомих як "компоненти", які можна використовувати повторно[8]. Цей підхід полегшує розуміння коду. Зазвичай компоненти написані за допомогою синтаксису JSX, який потім компілюється в виклики методів бібліотеки React. Також можна використовувати чистий JavaScript. Синтаксис може нагадувати HTML, оскільки компоненти створюються аналогічно, і атрибути передаються так само. Проте є синтаксичні відмінності, що роблять його розширенням синтаксису для JavaScript.

Крім того, React дозволяє розробникам створювати декларативні інтерфейси, де вони описують, як повинні виглядати елементи при різних станах програми, і бібліотека самостійно вирішує, як це досягти. Це полегшує управління станом програми і реагування на зміни в ній.

Однією з ключових особливостей React є віртуальний DOM (Document Object Model), який дозволяє ефективно оновлювати лише ті частини інтерфейсу, які зазнали змін, замість повного ререндерингу. Це призводить до оптимізації продуктивності за рахунок зменшення навантаження на браузер.

Бібліотека також підтримує реактивний підхід до програмування, де зміни в стані програми автоматично призводять до змін у відображенні, що спрощує реалізацію реактивних та взаємодіючих інтерфейсів.

React здобув широку популярність у розробників завдяки своїй ефективності, гнучкості та активній спільноті.

React Native – фреймворк для мобільної розробки, який використовується для створення нативних мобільних додатків за допомогою мови програмування JavaScript та бібліотеки React. Одна з головних переваг React Native полягає в тому, що він дозволяє розробникам використовувати знання JavaScript, для проектування додатків для платформ Android та iOS.

Архітектура React Native використовує ті ж принципи, що і оригінальний React, такі як компоненти та віртуальний DOM. Однак, в контексті мобільної розробки, React Native дозволяє використовувати компоненти, які відповідають нативним елементам інтерфейсу користувача. Це дозволяє досягати високої продуктивності, шляхом швидкого переходу від розробки вебсайтів, до додатків.

Однією з ключових особливостей React Native є можливість гарячого перезапуску (hot reload), що дозволяє вносити зміни в код додатку без повторного завантаження його в емулятор, чи на пристрій. Це полегшує і прискорює процес розробки.

React Native широко використовується у сфері мобільної розробки, спрощуючи створення кросплатформених додатків та забезпечуючи ефективне перевикористання коду між платформами.

Так як, в додатку з аналогічним функціоналом, використовувався клас з Open Source прикладу від Google, для роботи з Bluetooth. Так як є можливість перевикористати його при розробці додатку на React Native [9].

Загалом для розробки застосунку на основі цієї технології є два способи: з використанням Expo і React Native CLI.

Для розробки простих додатків використовується перший спосіб, він дозволяє пришвидшити розробку на початковому етапі. Переваги використання:

- абстрагуватися від нативних(платформених) залежностей, уникаючи необхідності налаштування конкретної платформи.

До таких залежностей входить:

- інтегроване середовище розробки (яке зазвичай містить компілятор, емулятор, інструменти для розробки, налагодження та відладки: Xcode, Android Studio)
- менеджер залежностей (спеціальний інструмент, який допомагає розробникам керувати зовнішніми бібліотеками та фреймворками, які використовуються у проектах. Має такі функції: встановлення бібліотек; використання їх конкретних версій, для уникнення конфліктів; оновлення для різних середовищ). Приклади менеджерів CocoaPods, npm,

Yarn; певною мірою Gradle, який частково робить це у файлі build.gradle Android версії.

- спрощений процес збірки, виконує компіляцію сам;
- розгорнути тестову версію додатку онлайн, в хмарі, за допомогою Expo Application Services ;
- можливість встановлення додатку Expo Go на свій смартфон, після цього підключитися до тієї ж мережі і протестувати свій додаток на будь-якому фізичному пристрої (як на Android, так і на iOS);
- доступ до готових вбудованих в бібліотеку рішень, які реалізують звернення до системних API: GPS-навігації, часткових сповіщень(при відкритому додатку), камери.
- Over-the-Air оновлення, дозволяє швидше застосувати зміни в додатку, реагуючи на зміни в коді;

Використання має й недоліки пов'язані з обмеженим функціоналом. Тобто інтеграція деяких бібліотек, може бути не доступною, особливо стосується тих, які роблять системні виклики, потребують низькорівневих речей. Відповідно, через додатковий шар абстракції, розмір програми теж буде більший. Звісно при роботі з Expo можна використовувати Native Modules з обмеженим функціоналом, наприклад MPAndroidChart, а от з DeviceListActivity – не вийде, так як потребує Bluetooth, тобто виклику системних API, відповідних дозволів.

Expo також дозволяє зробити перехід на React Native CLI, надаючи інструмент eject. При переході весь Native код розпаковується у папки ios та android, та створює в папці проекту файл кореневого компоненту, точку входу для додатку: index.js, використовуючи відповідний для цього клас з react-native бібліотеки AppRegistry, та його метод registerComponent[10]. Що відрізняється від Expo, де точка входу додатку знаходиться у \node_modules\expo AppEntry.js, та реєструється методом registerRootComponent, відповідним файлом з expo/build/launch.

Після цього можна керувати дозволами, так як тепер створився також файл `android/app/src/main/AndroidManifest.xml`, в якому можна додавати необхідні [10].

В рамках цього проекту необхідні дозволи на роботу з Bluetooth, для передачі інформації на смартфон, та файлової системи, для збереження та відтворення даних.

2.2 Розробка вбудованої програми для роботи з Bluetooth та AD8232

Розробка вбудованої програми для використання AD8232 з передачею даних по Bluetooth вимагає обробки біоелектричних сигналів, і їх безпечного передавання через бездротовий зв'язок. З цією метою використаю BLE стандарт, так він вважається новішим, та часто використовується в IoT, тому що зазвичай використовує менше енергії, що робить його ідеальним для пристроїв, які працюють в обмежених умовах енергопостачання, що є частим сценарієм для вбудованих систем, та використання в парі з AD8232, так як в інструкція часто вказують про безпеку, та необхідність підключення через ноутбук від автономного живлення або акумулятора. Прошивка включає в себе:

1. Підключення бібліотек у лістингу 2.1.

Лістинг 2.1 Бібліотеки необхідні для функціонування BLE

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
```

2. Ініціалізація сенсора AD8232 у лістингу 2.2.

Лістинг 2.2 Бібліотеки необхідні для функціонування BLE

```
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(38, INPUT); // L0+
  pinMode(35, INPUT); // L0-
} ...
void loop() {
  value = analogRead(A0);
  ...
}
```

```
}
```

3. Зчитування та відправлення даних через Bluetooth у лістингу 2.3.

Лістинг 2.3 Створення серверного з'єднання

```
#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b" // UUID сервісу
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8" // UUID характеристики

BLECharacteristic *pCharacteristic;
void setup() {
    Serial.begin(115200);
    // Ініціалізація Bluetooth
    BLEDevice::init("ZTU-ECG"); // Ім'я Bluetooth пристрою
    // Створення BLE серверу
    BLEServer *pServer = BLEDevice::createServer();

    // Створення BLE сервісу
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // Створення BLE характеристики
    pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_NOTIFY
    );

    // Старт BLE сервісу
    pService->start();

    // BLE транслює пакети на всі пристрої навколо нього
    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}
```

Також слід врахувати, що характеристика встановлена з властивістю `PROPERTY_NOTIFY`, що дозволяє використовувати систему сповіщень для передачі даних. У функції `setup()`, після ініціалізації Bluetooth, ми створюється сервер, сервіс та характеристику. Потім сервіс запускається, і чекає підключення (за допомогою `getAdvertising` та `pAdvertising->start()`). Після цього пристрій готовий приймати підключення та передавати дані через Bluetooth.

У основному циклі програми буде використовуватись `pCharacteristic->setValue()` та `pCharacteristic->notify()` для встановлення та відправлення значень даних через Bluetooth у лістингу 2.4.

Лістинг 2.4 Відправка даних, які зчитуються аналоговим входом та оцифровуються

```
void loop() {
    if(deviceConnected){
        if((digitalRead(41) == 1)|| (digitalRead(40) == 1)){
            Serial.println('!');
            std::string value = "!"; pCharacteristic->setValue(value);
            pCharacteristic->notify();
        }
        else { value = analogRead(A0);
```

```
Serial.println(value);  
pCharacteristic->setValue(value);  
pCharacteristic->notify();  
}  
}  
delay(3);  
}
```

2.3. Аналіз реалізації роботи з Bluetooth у додатку

Для можливої реалізації роботи з Bluetooth, як уже зазначалось, є декілька методів, більш старіша технологія яка й використовується в додатку з першого розділу, та новіший стандарт BLE(Bluetooth Low Energy). Під час перевірки додатку було істотно видно, що заряд батареї швидко знижується, після перегляду рішення, яке використовувало реалізацію звернень через Bluetooth, яка була написана 2009 року. Та знак того, що справа зниження заряду, точно від використання старої технології стало те, що проект заархівовано, а на його основі побудовано новіший, який вдосконалюється волонтерами Open Source.

Тому вирішено розробляти рішення з використанням новіших технологій, хоч вони й часто містять помилки(особливо при використанні новішої версії чогось, наприклад системи автоматичного збирання, або NPM пакет конфліктує, тому вихід тільки сподіватись, що хтось раніше стикнувся з цією проблемою та звернувся по допомогу, ресурси на які вказує пошуковий сайт можуть бути дуже різними: Github, StackOverflow, Twitter, Reddit, буває, що такі проблеми залишаються не вирішеними, через перехід на новіші рішення, зупинка підтримки власником), так як зазвичай написані, та підтримуються добровольцями. Та активна спільнота грає ключову роль у вирішенні проблеми з помилками, при зборі додатку, буває що подібні помилки можуть бути вже розв'язані тільки не на тій залежності яка використовується саме на такому, чи іншому проекті. Так як варто розуміти, що Bluetooth доволі специфічна річ, та проекти на її основі пишуться обмеженою кількістю людей, тому при подальшій розробці додатку ці речі були враховані, так як помилок не мало, та перезбирати додаток через кожну помилку є річчю не продуктивною, та на це варто розраховувати при використанні React Native CLI.

2.4. Вибір бібліотеки для додатка, та створення Bluetooth клієнта

Для застосування формату Bluetooth Low Energy в розробці додатку, є дві популярні бібліотеки react-native-ble-manager та react-native-ble-plx. Тому можна намагатися зрозуміти документацію, або подивитися їх вихідний код, і сконструювати відповідний функціонал власноруч, або знайти існуючі проекти де вони використовуються, та взяти доречні функції.

Таким чином, знайдено існуюче рішення[9]. Додаток виконує функціонал для знаходження пристроїв поблизу, які працюють з стандартом BLE, показуючи крім імені пристрою, також рівень потужності сигналу. Перед тим запитуючи дозвіл на включення Bluetooth, у разі його виключеного стану. Також додаток може створювати пару, та давати запит на підключатися до пристрою, якщо той є в зоні досягнення.

Тому взято проект існуючого додатку для подальшої розробки власного додатку на основі цього. За допомогою команди у лістингу 2.5 скопійовано проект.

Лістинг 2.5 Використання команди для копіювання віддаленого репозиторію на локальний простір

```
git clone [адреса репозиторію]
```

Після цього створено віддалений репозиторій та змінено адресу локальної системи контролю версій, на адресу створеного, видаливши перед цим прив'язку до оригінального, за допомогою команд у лістингу 2.6.

Лістинг 2.6 Видалення оригінальної адреси віддаленого репозиторію, та заміна на іншу, для того, щоб контроль версій не включав у себе оригінал

```
git remote rm origin  
git remote add origin [адреса віддаленого]
```

Також скористаюсь командою з лістингу 2.7, для перевірки, чи все додалось коректно.

Лістинг 2.7 Перевірка, чи попередня команда виконалась успішно

```
git remote -v
```

Після цього згенерую APK файл[10], для тестування додатку, звісно можна було б його не генерувати, а протестувати за допомогою функціоналу Arduino Studio, Android Device Manager, який дозволяє створювати емулятори пристроїв. Проте тут має бути звернення до апаратної частини, та функціонал залежить саме

від неї, бо має бути доступ до Bluetooth, а емулятор цього не передбачає. На рисунку 2.1 запит дозволу на включення Bluetooth, та початок сканування після цього.

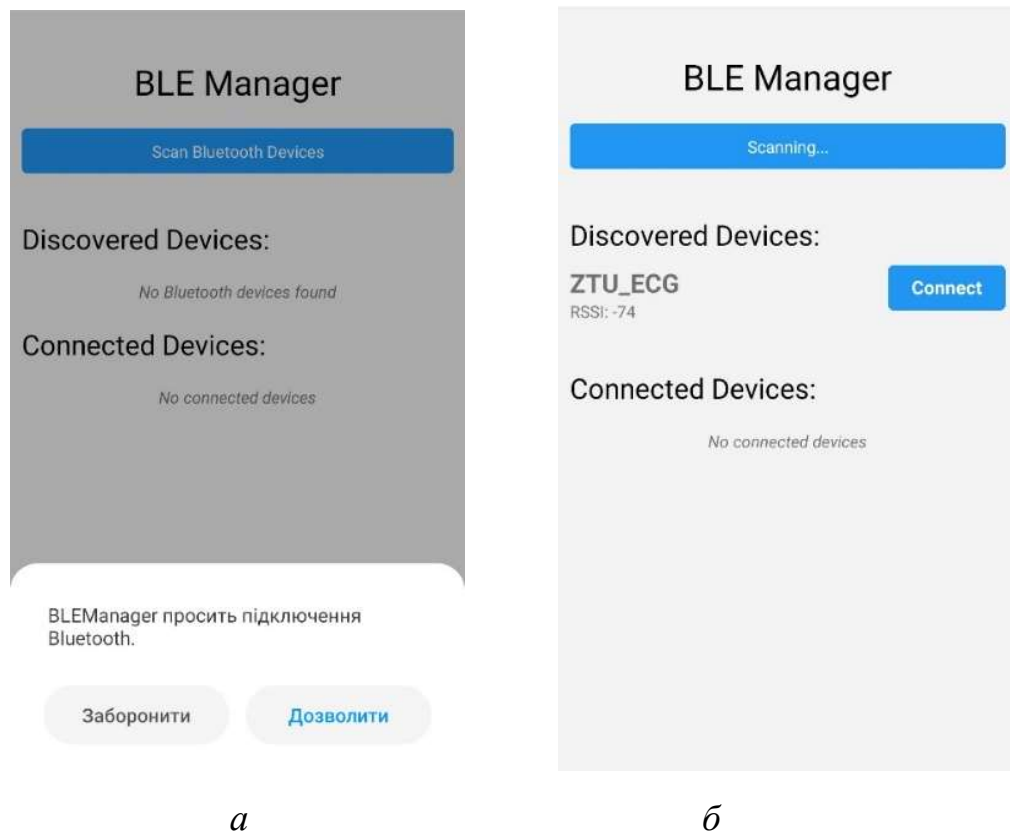


Рис. 2.1. Функціонал скомпільованого додатку:

a – запит дозволу при умові, що Bluetooth вимкнено; *б* – знайдені пристрої

Після цього можна спробувати підключитись до ESP32. Спробувати прийняти надіслані дані, та підключитись на обробник подій, який відпрацьовує, коли вони міняються. Для цього варто провести розбір, як саме працює код, з додатку, описаного вище, для початку приймання даних. Тому найголовнішим є бібліотека «react-native-ble-manager». Вона працює так, що треба підписуватись на події, які надсилає підключений пристрій(сервер, клієнту).

2.5. Особливості зв'язку BLE

Для зв'язку між двома пристроями, їм обом треба знати ServiceUUID, та CharacteristicUUID. Перше це унікальний ідентифікатор, який визначає послугу BLE. Друге - це дані або набір даних у межах послуги, які представляють конкретну

інформацію. Їх можна генерувати на спеціальних ресурсах, це робиться щоб не виникало конфліктів. BLE немає централізованого реєстру, як наприклад мають MAC адреси. Тому випадкові значення - найпростіший спосіб гарантувати унікальність.

Наприклад:

- 0000fee7-0000-1000-8000-00805f9b34fb – сервіс серцевого ритму
- 00002a37-0000-1000-8000-00805f9b34fb – UUID характеристики поточного показника серцевого ритму.

З них можна зрозуміти, що перший відноситься до вимірювання серцевого ритму, а другий містить конкретне значення серцевого ритму. Випадкові UUID в свою чергу не несуть інформації про призначення послуги чи характеристики. Не пов'язані UUID, ті які не несуть в собі певну семантичну інформацію про призначення сервісу або характеристики, ускладнюють сканування та аналіз пристрою зловмисниками. Хоча UUID можна зробити і змістовними, випадкові значення мають переваги за замовчуванням для більшості застосунків BLE. містить дані у вигляді атрибутів, згрупованих в сервіси та характеристики.

1. Клієнт спочатку сканує наявні BLE пристрої. Після виявлення потрібного сервера, клієнт надсилає запит на підключення.
2. Після успішного з'єднання, клієнт починає виявлення сервісів та характеристик на сервері за допомогою query.

Для обміну даними клієнт записує або читає значення характеристик, використовуючи відповідні UUID. BLE також підтримує шифрування та аутентифікацію за допомогою пари ключів. Для економії енергії BLE використовує короткі підключення, а не постійне з'єднання.

Для перевірки, чи інформація передається правильно, можна скористатись сервісами, які надають таку можливість. Справа в тому, що є експериментальна технологія Web Bluetooth API[11]. JavaScript API для веб-браузерів, що дозволяє веб-сайтам взаємодіяти з пристроями Bluetooth 4.0 та Bluetooth Low Energy (BLE), надає таким чином можливість взаємодіяти BLE периферійним пристроям з ПК. Так, як вона не є прийнятим стандартом, тому для її ввімкнення, може бути потрібно заходити до налаштувань браузера, в експериментальні налаштування.

Більше того, не всі браузери її підтримують[12]. Оскільки Web Bluetooth API ще перебуває в стадії розробки. його підтримка обмежена. В даний час, на рисунку 2.2 вказано, що він працює в Chrome, Opera та Samsung Internet на смартфонах Android та деяких настільних ОС. Це дозволяє створювати веб-додатки, які взаємодіють з BLE без необхідності нативного клієнта. Наприклад, зчитувати дані з фітнес-трекерів, поліпшувати взаємодію з IoT пристроями тощо.

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	Desktop					Mobile					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
Bluetooth	✓ 56 ...	✓ 79 ...	✗ No	✓ 43 ...	✗ No	✓ 56	✗ No	✓ 43	✗ No	✓ 6.0	✗ No
getAvailability	✓ 78	✓ 79	✗ No	✓ 65	✗ No	✓ 78	✗ No	✓ 56	✗ No	✓ 12.0	✗ No
getDevices	✗ 85 [icon]	✗ 85 [icon]	✗ No	✗ 71 [icon]	✗ No	✗ 85 [icon]	✗ No	✗ No	✗ No	✗ No	✗ No
requestDevice	✓ 56	✓ 79	✗ No	✓ 43	✗ No	✓ 56	✗ No	✓ 43	✗ No	✓ 6.0	✗ No

Рис. 2.2 Підтримка браузерами Web Bluetooth API

2.6. Розробка функціоналу для відображення даних

Для зручності користування, до взятого у розділі 2.4 рішення, збоку додано навігаційну панель, за допомогою додавання відповідної бібліотеки React Navigation. Таким чином створено іншу сторінку, для того, щоб можливо було побачити прогрес приймання даних в реально часу. Реалізую приймання до схожої реалізації додатку, розглянутого у першому розділі, де відбувається виведення на екран сигналу представленого у вигляді чисел від 0 до 4095.

Реалізація створення окремого вікна відбувається за допомогою елемента NavigationContainer з бібліотеки «@react-navigation/native» та функції

createDrawerNavigator з «@react-navigation/drawer». Також передбачено переключення між режимами текстового-графічного, тому для цього застосовано вже нижнє меню за допомогою функції «createBottomTabNavigator» з «@react-navigation/bottom-tabs». Дані будуть спільні для цих двох вікон, тому зберігатися у батьківському компоненті, нижнє меню для них зображено на рисунку 2.3. Код компонента у лістингу 2.8

Лістинг 2.8 Генерування штучних даних, їх збереження в стані компонента та передача до дочірніх елементів, які представляють собою екрани для відображення даних у текстовому та графічному форматі

```
export default function DataScreen({deviceId}) {
  const [data, setData] = useState([0]);

  useEffect(() => {
    var callCount = 1;
    var repeater = setInterval(function () {
      if (callCount < 30) {
        callCount += 1;
        let val = Math.floor(Math.random() * (10 - 1) + 1);
        setData(prev => [...prev, val]);
      } else {
        clearInterval(repeater);
      }
    }, 1000);
  }, [1000]);
  return (
    <NavigationContainer independent={true}>
      <Tab.Navigator
        screenOptions={({ route }) => ({
          tabBarIcon: ({ focused, color, size }) => {
            let iconName;

            if (route.name === 'Graphic') {
              iconName = 'heart-pulse';
              return <MaterialCommunityIcons name={iconName} size={size} color={color} />;
            } else if (route.name === 'Terminal') {
              iconName = 'code-greater-than';
              return <MaterialCommunityIcons name={iconName} size={size} color={color} />;
            }
          },
          tabBarActiveTintColor: '#2296f3',
          tabBarInactiveTintColor: 'gray',
          tabBarShowLabel: false,
          headerShown: false,
        })}
      >
        <Tab.Screen name="Graphic"
          options={{
            drawerIcon: ({color, size} ) => <MaterialCommunityIcons
              size={size}
              color={color}
              name="heart-pulse"></MaterialCommunityIcons>
          }}
          children={() => <GraphicScreen data={data}/>}
        />
        <Tab.Screen name="Terminal"
          options={{
            drawerIcon: ({color, size} ) => <MaterialCommunityIcons
```

```

        size={size}
        color={color}
        name="code-greater-than"></MaterialCommunityIcons>
      }}
      children={() => <TerminalScreen data={data}/>}
    />
  </Tab.Navigator>
</NavigationContainer>
);
}

```



Рис. 2.3 Нижнє навігаційне меню

Тепер ці дані можна буде відобразити в окремогих вікнах, та переключати режими перегляду, за допомогою нижнього меню. Нижче у лістингу 2.8 реалізація їх перегляду у текстовому форматі, на рисунку 2.4, компонент з даними.

Лістинг 2.8 Компонент, який являє собою вікно для відображення даних отриманих від батьківського компонента у текстовому форматі

```

export default function TerminalScreen({data}) {
  const [contentOffset, setContentOffset] = useState(0);

  return (
    <SafeAreaView >
      <ScrollView
        contentOffset={{x: 0, y: contentOffset}}
        onContentSizeChange={(_, height) => {
          setContentOffset(height);
        }}
      >
        <Text>{"  Дані:"}</Text>
        {!!data && data.map((value, index) => (
          <Text key={index}>{"    " + value}</Text>
        ))}
      </ScrollView>
    </SafeAreaView> ); }

```

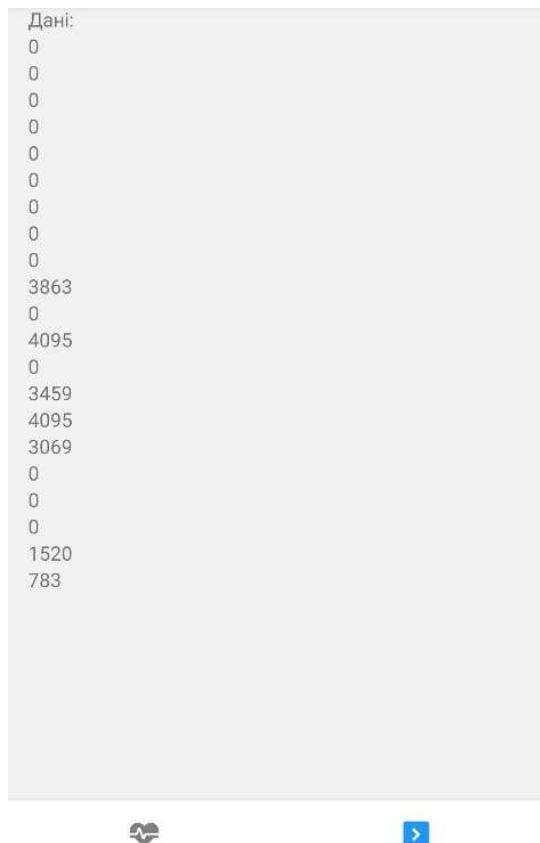


Рис. 2.4 Відображення даних у текстовому форматі

Також реалізація показу даних графічному форматі у лістингу 2.9. Для подальшого приймання даних про сигнал, дані будуть мати шуми, тому додано реалізацію фільтра Калмана, таким чином відбудеться відображення двох графіків, на рисунку 2.5 це зображено.

Лістинг 2.9 Вікно для відображення даних отриманих від батьківського компонента у графічному форматі

```
export default function GraphicScreen({data}) { return (
  <SafeAreaView >
    <ScrollView
      horizontal={true}
      contentOffset={{x: 26.5625 * data.length, y: 0}}
      showsHorizontalScrollIndicator={false}
    >
      <LineChart
        withDots={false}
        withVerticalLines={false}
        withHorizontalLines={false}
        withShadow={false}
        chartConfig={{
          backgroundColor: "#ffff",
          backgroundGradientFrom: "#ffff",
          backgroundGradientTo: "#ffff",
          decimalPlaces: 2, // optional, defaults to 2dp
          color: (opacity = 1) => `rgba(46, 204, 113 , ${opacity})`,
          labelColor: (opacity = 1) => `rgba(28, 40, 51 , ${opacity})`,
          style: {
```

```

        borderRadius: 16,
      },
      propsForDots: {
        r: "6",
        strokeWidth: "2",
        stroke: "#2ECC71",
      },
    },
  },
  opacity={0}
  data={{
    // labels: ["January", "February", "March", "April", "May", "June"],
    legend: [{legend}`],
    datasets: [ { data: data, backgroundColor: '#e26a00', } ]
  }} bezier width={chartWidth} height={Dimensions.get('window').height -
(Dimensions.get('window').height/100*15)} />
</ScrollView> </SafeAreaView> );
function calculateKalmanData(data) {
  // Параметри фільтра Калмана
  const R = 0.01; // Шум вимірювань
  const Q = 0.03; // Шум процесу
  const kalmanData = [];
  // Стан фільтра
  let x = 0; let p = 1;
  for (let i = 0; i < data.length; i++) {
    // Прогноз
    const x_prior = x;
    const p_prior = p + Q;
    // Оновлення
    const k = p_prior / (p_prior + R);
    x = x_prior + k * (data[i] - x_prior);
    p = (1 - k) * p_prior;
    // Збереження фільтрованого значення
    kalmanData.push(x);
  }
  return kalmanData;
}

```

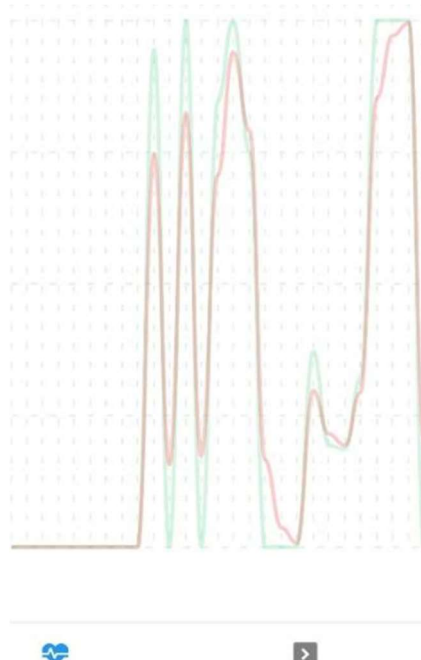


Рис. 2.5 Відображення згенерованих даних, з застосуванням фільтра Калмана

2.7. Розробка функціоналу для отримання даних по Bluetooth

Для додавання функціоналу по прийому даних через Bluetooth заміню функцію, по їх генеруванні штучно. Та спершу, треба після коду, який відповідає за створення пари, додати підписку та приєднання до того ж пристрою. Потім викликати метод `startNotification` з параметрами ідентифікаторів: пристрою, до якого відбувається підключення; сервісу, та характеристики. Після цього додано прослуховування на зміну значення для характеристики. Код реалізації у лістингу 2.10.

Лістинг 2.10 Підписка на зміну характеристики, та виклик `alert` функції

```
...  
const handleDidUpdateValueForCharacteristic = (receivedData) => {  
  alert(receivedData);  
};  
...  
  
BleManager.connect(deviceId);  
BleManager.startNotification(deviceId, ESPEcg.service, ESPEcg.characteristic);  
bleEmitter.addListener(  
  'BleManagerDidUpdateValueForCharacteristic',  
  handleDidUpdateValueForCharacteristic,  
);
```

`startNotifications()`, повідомляє пристрою, що він може почати надсилати дані[13]. Далі за допомогою підписки на `'BleManagerDidUpdateValueForCharacteristic'`, кожного разу, коли характеристика оновлює своє значення, вона попаде в методі, який передається, як `Callback` другим параметром. Виклик `alert` функції зі значенням, яке прийшло у `handleDidUpdateValueForCharacteristic` на рисунку 2.6.

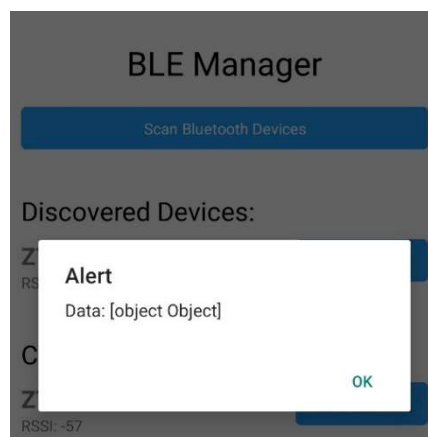


Рис. 2.6 Спроба отримання даних

Як можна бачити, прийшов об'єкт, тому для читання його значень, варто викликати `JSON.stringify()`, яке конвертує об'єкт JS значення в JSON рядок. Результат виконання на рисунку 2.7.

Alert

```
Data: {"value":[255,15],"service":"4fafc201-1fb5-459e-8fcc-c5c9c331914b","characteristic":"beb5483e-36e1-4688-b7f5-ea07361b26a8","peripheral":"A0:B7:65:4F:4B:7A"}
```

Рис. 2.7 Значення, яке надійшло від ESP32

Можна побачити, що крім значення, яке власне й було метою передачі, приходить і інша інформація: MAC-адреса, яка фігурує після створення пари у коді, як `deviceId`. Можна спробувати викликати і іншу службову інформацію, результатом такого виклику є рисунок 2.9.

Alert

```
{"advertising":{"serviceUUIDs":["4fafc201-1fb5-459e-8fcc-c5c9c331914b","4fafc201-1fb5-459e-8fcc-c5c9c331914b"],"txPowerLevel":3,"isConnectable":true,"serviceData":{"localName":"ZTU_ECG","manufacturerData":{"bytes":[2,1,6,17,7,75,145,49,195,201,197,204,143,158,69,181,31,1,194,175,79,8,9,90,84,85,95,69,67,71,2,10,3,17,7,75,145,49,195,201,197,204,143,158,69,181,31,1,194,175,79,0,0,0,0,0,0,0,0],"data":"AgEGEQdLkTHDycXMj55FtR8Bwq9PCAlaVFVfRUNHAgODEQdLkTHDycXMj55FtR8Bwq9PAAAAAAAAAAAAAAAA","CDVType":"ArrayBuffer"}},"rssi":-60,"id":"A0:B7:65:4F:4B:7A","name":"ZTU_ECG"}
```

OK

Рис. 2.8 Службова інформація, про пристрій, яку можна запитати після створення пари

Інформація запитана вище містить в собі крім вже знайомих даних, також і RSSI (рівень потужності), ідентифікатори сервісів, через який власне й було підключення, та отримання значення вище.

Допишу відповідну підписку до іншого екрану, для того, щоб отримані дані зберегти в стан, замість штучно генерованих, в результаті вийде код з лістинга 2.11.

Лістинг 2.11 Підписка на зміну характеристики, та зберігання отриманих значень у стані

```
export default function DataScreen({deviceId}) {
  const [data, setData] = useState([0]);
  const handleDidUpdateValueForCharacteristic = (receivedData) => {
    setData(prev => [...prev, receivedData]);
  };
  useEffect(() => {
    if (!deviceId) {return;}

    BleManager.connect(deviceId);
    BleManager.startNotification(deviceId, ESPEcg.service, ESPEcg.characteristic);
    bleEmitter.addListener(
      'BleManagerDisconnectPeripheral',
      handleDisconnectedPeripheral,
    );
    bleEmitter.addListener(
      'BleManagerDidUpdateValueForCharacteristic',
      handleDidUpdateValueForCharacteristic,
    );

    return () => {
      console.log('Unmount');
      bleEmitter.removeAllListeners();
    };
  }, [deviceId]);
}
```

Висновок до розділу 2

Оцінено переваги використання сучасних рішень для побудови додатків, а саме React Native, методи для їх розробки (з можливим використанням на початках Expo, та переходом на складнішу розробку у разі такої необхідності, за допомогою React Native CLI), можливості звернень до системних функцій за допомогою нативних модулів, які пишуть на мові програмування Java. Розглянуто структуру таких проектів, засоби керування залежностями. Розроблено вбудовану програму для ESP32, яка оцифровує дані від модуля AD8232, та передає їх, по Bluetooth, стандартом BLE, створюючи сервер. Після приєднання до якого створюється пара, та передаються дані. Також створено додаток, який може приєднуватись до ESP32, та отримувати передані дані.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА НАЛАШТУВАННЯ ПРОГРАМНО-АПА- РАТНОГО КОМПЛЕКСУ

3.1. Налаштування роботи з додатком

Для роботи з додатком, треба мати встановлену платформу Node.JS, яка дозволяє запускати програми написані мовою JavaScript. Після його встановлення в корені папки запустити командний рядок, «npm install» за допомогою якого встановляться пакети описані у файлі package.json. Далі, є два варіанти, або запускати на емуляторі, або генерувати APK файл, та подальшому тестування на смартфоні. Незалежно від того, який обраний варіант далі треба встановити програму Gradle, яка створена для автоматичного збирання додатку.

Для генерування файлу треба зайти в папку android, та запустити скрипт «\.\gradlew assemble», який згенерує APK файл у папці «android\app\build\outputs\apk». Після цього його передати смартфон.

Для тестування на емуляторі, треба встановити Android Studio, в якого є функція Virtual Device Manager, далі завантажити емулятор, також є можливість вибору розміру екрана, та Android версії. Після в корені проекту треба запустити «npm run android». Після цього має з'явитися вікно, як на рисунку 3.1, після якого треба натиснути один раз «a» для автоматичного відкривання емулятора, в який завантажиться додаток.

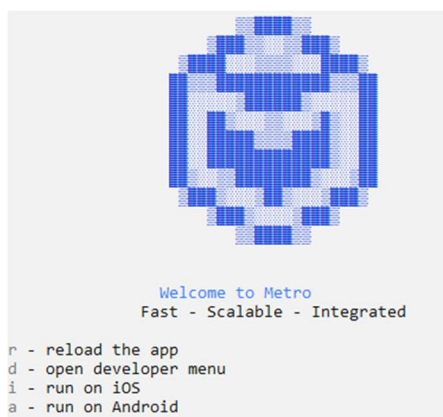
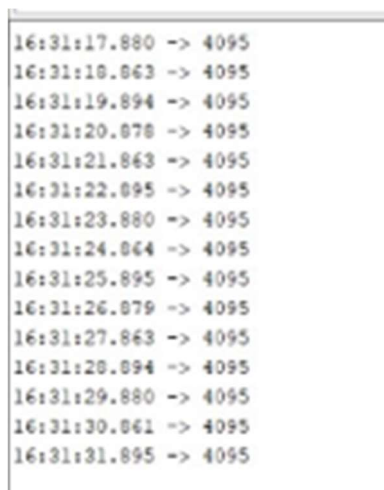


Рис. 3.1 Вікно для запуску додатка в емуляторі

3.2. Приведення отриманих даних до одного формату

Після генерування додатка, встановлення підключення, було отримані дані у вигляді [255, 15]. Для того, щоб зрозуміти яке значення малось на увазі, відкриву Arduino IDE, та запуску Serial Monitor, так як значення крім відправки, транслюється туди теж, про що було написано у коді прошивки. Serial Monitor з прийнятими значеннями на рисунку 3.2.



```
16:31:17.880 -> 4095
16:31:18.863 -> 4095
16:31:19.894 -> 4095
16:31:20.878 -> 4095
16:31:21.863 -> 4095
16:31:22.895 -> 4095
16:31:23.880 -> 4095
16:31:24.864 -> 4095
16:31:25.895 -> 4095
16:31:26.879 -> 4095
16:31:27.863 -> 4095
16:31:28.894 -> 4095
16:31:29.880 -> 4095
16:31:30.861 -> 4095
16:31:31.895 -> 4095
```

Рис. 3.2 Дані, які прийнялись додатком як [255, 15]

Тому для більшого уявлення, чи не містить додаток помилок здійснено перевірку через засоби веб браузера. Підключення на рисунку 3.3, після якого браузерна вкладка зміниться на рисунку 3.4, додавши символ про підключення.

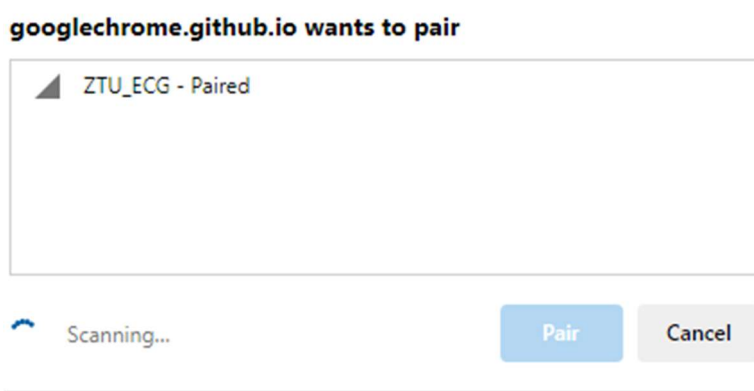


Рис. 3.3 Сканування доступних пристроїв

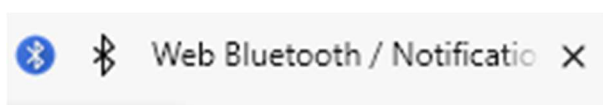


Рис. 3.4 Індикатор, праворуч який свідчить про підключення

Web Bluetooth / Notifications Sample

Available in [Chrome 48+](#) | [View on GitHub](#) | [Browse Samples](#)

The [Web Bluetooth API](#) lets websites discover and communicate with devices over the Bluetooth standard using the Generic Attribute Profile (GATT). It is currently partially implemented in Android, iOS, Mac, and Windows 10.

This sample illustrates the use of the Web Bluetooth API to start and stop characteristic notifications from a nearby Bluetooth Low Energy Device. You may want to try this demo with the BLE Peripheral on the [Google Play Store](#) and check out the [Notifications \(Async Await\)](#) sample.

4fafc201-1fb5-459e-8fcc-4 beb5483e-36e1-4688-b7f

Live Output

```
Requesting Bluetooth Device...
Connecting to GATT Server...
Getting Service...
Getting Characteristic...
> Notifications started
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
> 0xff 0x0f
```

Рис. 3.5 Початок передавання після вказування відповідних UUID

На рисунку 3.5, зображено процес передачі даних, тільки дані, не в такому форматі, якому читаються з SerialMonitor. Зі сторони смартфона, не буде проблемою спробувати їх конвертувати, знаючи оригінальні дані, які мають бути. На рисунку 3.6 зображено одночасне приймання через Bluetooth та Serial.Monitor.

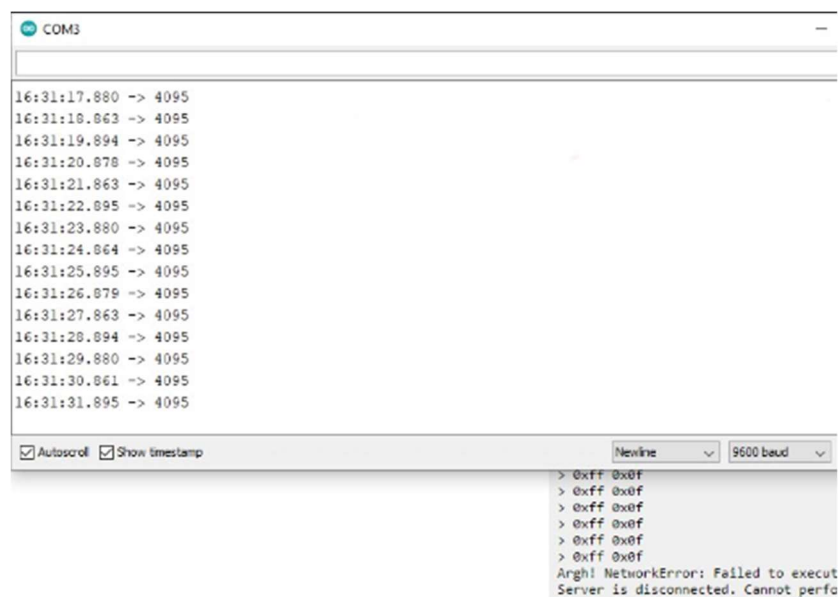


Рис. 3.6 Співставлення даних з SerialMonitor, та отриманих через браузер

Тож можна бачити, що інформація у всіх можливих підключеннях відрізняється. На смартфоні: [255, 15], у браузері: [0xff 0x0f], SerialMonitor: 4095.

Зі сторони смартфона, не буде проблемою спробувати їх конвертувати, знаючи дані, які мають бути. Така різниця сталась внаслідок того, що через Bluetooth передаються за допомогою бінарного кодування даних, які відображаються по різному. Тому у додатку треба їх привести до вигляду 4095, для забезпечення відповідності форматів даних на обох пристроях. Для приведення написано код з лістингу 3.1

Лістинг 3.1 Приведення даних до похідних

```
const parseData = (receivedData) => {  
  const first = parseInt(receivedData.value[0], 10);  
  const second = parseInt(receivedData.value[1], 10);  
  
  return (second << 8) + first;  
};
```

Після зчитування чисел, врахувавши порядок байтів(в даному випадку Великий порядок, Big-endian), відбувається приведення до початкового формату. Після цього можна перевірити правильність перетворення та бачити дані які прийшли на графіку, на рисунку 3.6.

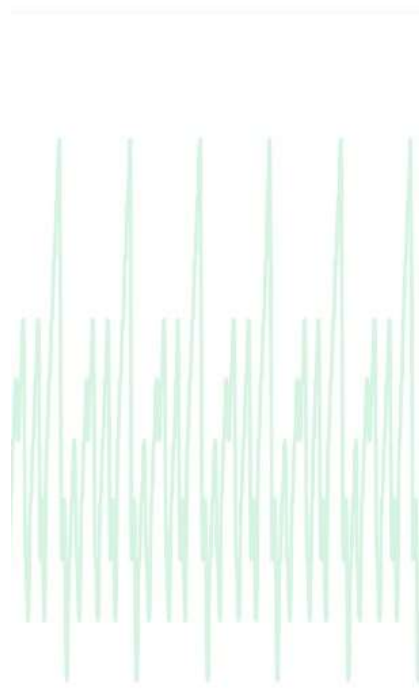


Рис 3.6 Відображення даних графічно в результаті розшифрування прийнятих даних

3.3. Налагодження відображення графічних даних, розробка функціоналу для збереження та показу збережених даних

Так, як при тривалому використанні лінії починають бути нерозбірливі, накладатися одна на одну було вирішено зробити, щоб діаграма розтягувалась, та сигнал можна було розрізнити. Для цього введено змінну, яка позначає, скільки приблизно буде поміщатись точок в межах одного екрану. Та замірено, в який саме момент ширина зміщатись. Це сталося при 48 точках на екрані, ціна екрану в цей момент була 1250 пікселів, тому поділивши перше на друге, отримав ціну поділки(26.56). За допомогою неї буде відбуватись розтягнення у випадку, якщо точок в даних буде більше ніж число, яке позначає кількість точок, які мають вміститись на одній ширині екрану. Щоб розтягнення не відбулося до моменту наближення до відстані ширини. Таким чином досягнувши навігації, на діаграмі. Цього б не варто було б робити, у випадку, якщо б відбувалося коректне реагування на Scroll, в такому випадку реалізацію цієї функції можна було б зробити інакше, не розтягуванням діаграми, а її перемальовуванням, відповідно реагувавши на слайд в бік.

Також в додатку, який було взято за приклад цього не було, по тій причині, що перегляд графічних даних назад не є можливим, тільки перегляд реальних за якийсь конкретний період. Тому додам функцію для зупинки слідуванням за реальними даними, щоб мати змогу повернутись на початок. Зупинка буде після натискання на кнопку. Результат її додавання на рисунку 3.7.

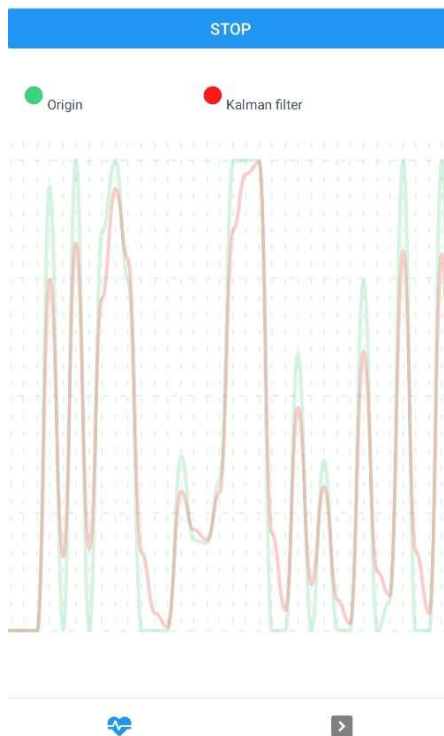


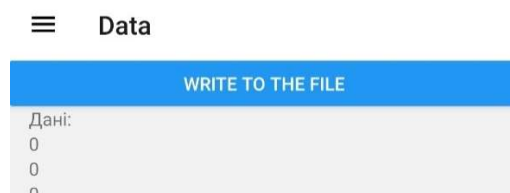
Рис. 3.7 Діаграма, з кнопкою, для можливості перестати керування навігацією

Також після цього було реалізовано збереження у файл (за допомогою «react-native-fs»[14]), результат на рисунку 3.8, додавши перед цим дозвіл на написання у файлову систему, у файлі Android.xml android.permission.WRITE_EXTERNAL_STORAGE[15]:

Лістинг 3.2 Збереження даних

```
import RNFS from 'react-native-fs';

<Button title={'Write to the file'} onPress={() => RNFS.write(RNFS.ExternalStorageDirectoryPath
+ '/data.txt', data.toString())}/>
```



a

< data

359,3376,385,1807,2702,2278,785,714,1296,108
8,2043,0,4095,934,391,3927,0,4095,0,1904,2110,
1437,3757,4095,0,4095

б

Рис. 3.8 Функція збереження даних

a – кнопка після якої відбувається збереження; *б* – файл збережених даних

Після цього було вирішено додати можливість завантажувати збережені файли, для цього продубльовано існуючий функціонал, та винесено в окреме вікно

на бічній панелі. Також зменшено висоту діаграми, та надано можливість скролити нижче. Вихідний код після змін на лістингу 3.3, прочитані дані на рисунку 3.9.

Лістинг 3.3 Регулювання ширини діаграми в залежності від кількості даних, можливість не слідувати за діаграмою

```

const windowHeight = Dimensions.get('window').width;
const [chartWidth, setChartWidth] = useState(windowWidth);
const [stopWidth, setStopWidth] = useState(false);
const pointsPerScreen = 50;
const [contentOffset, setContentOffset] = useState({ x: 26.5625 * data.length, y: 0 });
useEffect(()=>{
  if (data.length < pointsPerScreen) {return;}
  setChartWidth(data.length / pointsPerScreen * windowWidth);
  if (stopWidth)
    {setContentOffset({x:0, y:0});}
  else
    {setContentOffset({ x: 26.5625 * data.length, y: 0 });} }, [data.length]);
return (
  <SafeAreaView > <Button title={stopWidth ? 'Follow' : 'Stop'} onPress={() =>
setStopWidth(!stopWidth)}/> <ScrollView horizontal={true} contentOffset={contentOffset}
showsHorizontalScrollIndicator={false}>
    <LineChart opacity={0} data={{ legend: ['Origin', 'Kalman filter'], datasets: [
      { data: data, backgroundColor: '#e26a00', },
      { data: kalmanData, color: (opacity = 1) => `rgba(255,0,0,${opacity})`,
    },
    ]},
    bezier
    width={chartWidth}
    heightDimensions.get('window').height-(Dimensions.get('window').height/100*25)}
    verticalLabelRotation={-90}
  />
</ScrollView>
</SafeAreaView>
);
}

```



Рис. 3.9 Прочитані дані

Висновок до розділу 3

Було налаштовано програмні рішення для тестування роботи додатку, шляхом генерування окремого файлу, для тестування на смартфоні; та встановленням емулятора. Налагоджено додаток: перевірено передачу даних від ESP32 у веб браузері, додано приведення отриманих даних у додатку, до початкового формату. Так як, при пересилання по Bluetooth дані передаються за допомогою бінарного кодування даних, та формат змінюється.

Змінено показ графічних даних, для зручнішого перегляду, додано можливість навігації по графіку, також можливість не слідувати за потоком даних. Розроблено функціонал для зберігання отриманих даних, та можливість перегляду вже збережених

ВИСНОВКИ

Метою роботи є створення пристрою, який реєструє серцевий струм. Тому для його проектування було розглянуто методи дослідження серця, можливі апаратні реалізації. У ході роботи розроблено пристрій, який працює у мережі Bluetooth за допомогою енергоекономного стандарту BLE. Пристрій виступає у ролі передавача відцифрованих біомедичних даних на смартфон. Розроблено додаток для приймання цих даних, таким чином стало можливе спостереження серцевої активності в реальному часі.

Додаток до пристрою включає: сторінку для перегляду доступних BLE підключень, з можливістю створювати пару, та почати отримувати дані з датчика. Реалізовано зберігання отриманих даних у файл, також можливість відкривати збережені файли, та переглядати їх на графіку.

При розробці додатку використовувалися бібліотеки:

- для побудови графіка react-native-chart-kit;
- для збереження даних у файл react-native-fs.
- для обрання документу для прочитання react-native-document-picker

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Purdum J. Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino / Jack Purdum. – Cincinnati: Apress, 2015. – Chapter 12 Arduino Libraries; P 277-298.
2. ECG Monitoring system using AD8232 with Arduino or ESP32 IOT Based. Circuit Schools. URL: <https://www.circuitschools.com/ecg-monitoring-system-using-ad8232-with-arduino-or-esp32-iot-based/> (date of access: 28.12.2023).
3. Chio C., Freeman D. Machine Learning and Security: Protecting Systems with Data and Algorithms. Sebastopol : O'Reilly Media, 2018.
4. Smith D. Android Recipes: A Problem-Solution Approach for Android 5.0. Cincinnati : Apress, 2015.
5. Android Git repositories - Git at Google. <https://android.googlesource.com>
6. Google Archive. Github. URL: <https://github.com/googlearchive> (date of access: 26.12.2023).
7. Fork. URL: [https://en.wikipedia.org/wiki/Fork_\(software_development\)](https://en.wikipedia.org/wiki/Fork_(software_development)) (date of access: 26.12.2023).
8. Dabit N. React Native in Action. NY : Manning Publications, 2019.
9. Innocent C. Using react-native-ble-manager in a mobile application. URL: <https://blog.logrocket.com/using-react-native-ble-manager-mobile-app> (date of access: 26.12.2023).
10. Ward D. React Native Cookbook. 2nd ed. Mumbai : Packt Publishing, 2019.
11. Chong L. P. Mastering Front-End Web Development. Amazon Digital Services LLC - Kdp, 2020.
12. Web Bluetooth API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API (дата звернення: 29.12.2023).
13. Web Bluetooth by example. URL: <https://medium.com/@gerybbg/web-bluetooth-by-example-6d200fa9a3ed> (date of access: 26.12.2023).
14. Kuttig A. B. Professional React Native. Mumbai : Packt Publishing, 2022.
15. Burd B., Mueller J. P. Android Application Development All-in-One For Dummies. Wiley : 2020.

ДОДАТКИ

Батьківський компонент для інших файлів, вміщає в собі бокову навігацію додатку

```
import 'react-native-gesture-handler';

import React, { useState } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import ScanScreen from './ScanScreen';
import DataScreen from './DataScreen';
import AnalyzeScreen from './AnalyzeScreen';
import {
  createDrawerNavigator,
} from '@react-navigation/drawer';

import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';
const Drawer = createDrawerNavigator();

const App = () => {
  const [deviceId, setDeviceId] = useState();

  return (
    <NavigationContainer independent={true}>
      <Drawer.Navigator>
        <Drawer.Screen
          name="Scan"
          options={{
            drawerIcon: ({color, size} ) => <MaterialCommunityIcons
              size={size}
              color={color}
              name="bluetooth-audio" />,
          }}>
          {() => <ScanScreen setDeviceId={setDeviceId}/>}
        </Drawer.Screen>
        <Drawer.Screen name="Data" options={{
          drawerIcon: ({color, size} ) => <MaterialCommunityIcons
            size={size}
            color={color}
            name="heart-pulse" />,
        }}>
          {() => <DataScreen deviceId={deviceId}/>}
        </Drawer.Screen>
        <Drawer.Screen name="Analyze" options={{
          drawerIcon: ({color, size} ) => <MaterialCommunityIcons
            size={size}
            color={color}
            name="border-none"/>,
        }}>
          {() => <AnalyzeScreen/>}
        </Drawer.Screen>
      </Drawer.Navigator>
    </NavigationContainer>
  );
}; export default App;
```

Вікно сканування доступних пристроїв поблизу, з можливістю створення пари, при натисканні на кнопку

```

export default import React, {useState, useEffect} from 'react';
import { Text, Alert, View, FlatList, Platform, StatusBar, SafeAreaView, NativeModules,
  useColorScheme, TouchableOpacity, NativeEventEmitter, PermissionsAndroid,} from 'react-
native'; import {styles} from './src/styles/styles';
import {DeviceList} from './src/DeviceList';
import BleManager from 'react-native-ble-manager';
import {Colors} from 'react-native/Libraries/NewAppScreen';
const BleManagerModule = NativeModules.BleManager;
const BleManagerEmitter = new NativeEventEmitter(BleManagerModule);
const ScanScreen = ({setDeviceId}) => {
  const peripherals = new Map();
  const [chosenPeripheral, setChosenPeripheral] = useState();
  const [isScanning, setIsScanning] = useState(false);
  const [connectedDevices, setConnectedDevices] = useState([]);
  const [discoveredDevices, setDiscoveredDevices] = useState([]);
  const handleLocationPermission = async () => {
    if (Platform.OS === 'android' && Platform.Version >= 23) {
      try { const granted = await PermissionsAndroid.request(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
      ); if (granted === PermissionsAndroid.RESULTS.GRANTED) {
        console.log('Location permission granted');
      } else {
        console.log('Location permission denied');
      } } catch (error) {
        console.log('Error requesting location permission:', error);
      }
    }
  };
  const handleGetConnectedDevices = () => {
    BleManager.getBondedPeripherals([]).then(results => {
      for (let i = 0; i < results.length; i++) {
        let peripheral = results[i];
        peripheral.connected = true;
        peripherals.set(peripheral.id, peripheral);
        setConnectedDevices(Array.from(peripherals.values()));
      }
    });
  };
  useEffect(() => {
    handleLocationPermission();
    BleManager.enableBluetooth().then(() => { console.log('Bluetooth is turned on!'); });
    BleManager.start({showAlert: false}).then(() => {
      console.log('BleManager initialized');
      handleGetConnectedDevices(); });
    let stopDiscoverListener = BleManagerEmitter.addListener(
      'BleManagerDiscoverPeripheral', peripheral => {
        peripherals.set(peripheral.id, peripheral);
        setDiscoveredDevices(Array.from(peripherals.values()));
      }, );
    let stopConnectListener = BleManagerEmitter.addListener(
      'BleManagerConnectPeripheral', peripheral => {

```

```

        setChosenPeripheral(peripheral); }, );
let stopScanListener = BleManagerEmitter.addListener(
  'BleManagerStopScan',
  () => {
    setIsScanning(false);
    console.log('scan stopped');
  },
);
return () => {
  stopDiscoverListener.remove();
  stopConnectListener.remove();
  stopScanListener.remove();
};
}, []);

const scan = () => {
  if (!isScanning) {
    BleManager.scan([], 5, true)
      .then(() => {
        console.log('Scanning...');
        setIsScanning(true);
      })
      .catch(error => {
        console.error(error);
      });
  }
};

const connect = peripheral => {
  BleManager.createBond(peripheral.id)
    .then((id) => {
      setDeviceId(peripheral.id);
      alert('connected');
      peripheral.connected = true;
      peripherals.set(peripheral.id, peripheral);
      let devices = Array.from(peripherals.values());
      setConnectedDevices(Array.from(devices));
      setDiscoveredDevices(Array.from(devices));
      console.log('BLE device paired successfully');
    })
    .catch(() => {
      throw Error('failed to bond');
    });
};

const disconnect = peripheral => {
  BleManager.removeBond(peripheral.id)
    .then(() => {
      peripheral.connected = false;
      peripherals.set(peripheral.id, peripheral);
      let devices = Array.from(peripherals.values());
      setConnectedDevices(Array.from(devices));
      setDiscoveredDevices(Array.from(devices));
      Alert.alert(`Disconnected from ${peripheral.name}`);
    })
    .catch(() => {
      throw Error('fail to remove the bond');
    });
};

```

```

const isDarkMode = useColorScheme() === 'dark';
const backgroundStyle = {
  backgroundColor: isDarkMode ? Colors.darker : Colors.lighter,
};

return ( <SafeAreaView style={[backgroundStyle, styles.container]}>
  <StatusBar
    barStyle={isDarkMode ? 'light-content' : 'dark-content'}
    backgroundColor={backgroundStyle.backgroundColor}
  /> <View style={{paddingHorizontal: 20}}>
    <Text style={[
      styles.title,
      {color: isDarkMode ? Colors.white : Colors.black},
    ]}> BLE Manager </Text>
    <TouchableOpacity onPress={scan} activeOpacity={0.5} style={styles.scanButton}>
      <Text style={styles.scanButtonText}> {isScanning ? 'Scanning...' : 'Scan
Bluetooth Devices'} </Text> </TouchableOpacity>
      <Text style={[ styles.subtitle, {color: isDarkMode ? Colors.white : Colors.black},
    ]}> Discovered Devices: </Text>
      {discoveredDevices.length > 0 ? ( <FlatList data={discoveredDevices}
renderItem={({item}) => ( <DeviceList peripheral={item} connect={connect}
disconnect={disconnect} />
    )} keyExtractor={item => item.id}
/>):( <Text style={styles.noDevicesText}>No Bluetooth devices found</Text>)}
      <Text style={[styles.subtitle, {color: isDarkMode ? Colors.white : Colors.black},
    ]}> Connected Devices:</Text>
      {connectedDevices.length > 0 ? (
        <FlatList
          data={connectedDevices}
          renderItem={({item}) => (
            <DeviceList
              peripheral={item}
              connect={connect}
              disconnect={disconnect}
            />
          )}
          keyExtractor={item => item.id}
        />
      ) : (
        <Text style={styles.noDevicesText}>No connected devices</Text>
      )}
    </View>
  </SafeAreaView>
);
};

export default ScanScreen;

```

Екран для показу графіка, який з даних, які надходять, від ESP32, графік розширюється динамічно

```
import React, { useState, useEffect } from 'react';
import { Dimensions, ScrollView, SafeAreaView, Button } from 'react-native';
import { LineChart } from 'react-native-chart-kit';

export default function GraphicScreen({data, contentOffsetProp}) {
  const windowWidth = Dimensions.get('window').width;
  const [chartWidth, setChartWidth] = useState(windowWidth);
  const [stopWidth, setStopWidth] = useState(false);
  const pointsPerScreen = 200;
  const [kalmanData, setKalmanData] = useState([0,2]);
  const [contentOffset, setContentOffset] = useState({ x: 26.5625 * data.length, y: 0 });
  useEffect(()=>{
    const filteredData = calculateKalmanData(data);
    setKalmanData(filteredData);
    if (data.length < pointsPerScreen) {return;}
    setChartWidth(data.length / pointsPerScreen * windowWidth);
    if (stopWidth)
      {setContentOffset({x:0, y:0});}
    else
      {setContentOffset({ x: 26.5625 * data.length, y: 0 });}
  }, [data.length]);
  return (
    <SafeAreaView >
      <Button title={stopWidth ? 'Follow' : 'Stop'} onPress={() =>
setStopWidth(!stopWidth)}/>
      <ScrollView horizontal={true} contentOffset={contentOffset}
showsHorizontalScrollIndicator={false} >
        <LineChart withDots={false} withShadow={false}
          chartConfig={{ backgroundColor: '#ffff', backgroundGradientFrom: '#ffff',
            backgroundGradientTo: '#ffff', decimalPlaces: 2, // optional, defaults to 2dp
            color: (opacity = 1) => `rgba(46, 204, 113 , ${opacity})`, labelColor:
(opacity = 1) => `rgba(28, 40, 51 , ${opacity})`, style: { borderRadius: 16, },
            propsForLabels: { fontSize: 12, }} opacity={0} data={{ legend: ['Origin', 'Kalman
filter'], datasets: [ { data: data, backgroundColor: '#e26a00', },
          { data: kalmanData, //Kalman Values
            color: (opacity = 1) => `rgba(255,0,0,${opacity})`, }, ], }}
yAxisLabel={'mV '} bezier width={chartWidth}
          height={Dimensions.get('window').height - (Dimensions.get('window').height /
100 * 25)} verticalLabelRotation={-90} /> </ScrollView> </SafeAreaView> ); }
  function calculateKalmanData(data) {
    // Параметри фільтра Калмана
    const R = 0.01; // Шум вимірювань
    const Q = 0.03; // Шум процесу
    const kalmanData = [];
    // Стан фільтра
    let x = 0; let p = 1;
    for (let i = 0; i < data.length; i++) {
      // Прогноз
      const x_prior = x; const p_prior = p + Q;
      // Оновлення
      const k = p_prior / (p_prior + R);
      x = x_prior + k * (data[i] - x_prior);
      p = (1 - k) * p_prior; // Збереження фільтрованого значення
      kalmanData.push(x); } return kalmanData;
  }
}
```

Екран для показу даних, які прийшли від ESP32 у числовому вигляді, з
можливістю запису їх у файл

```
import React, { useState } from 'react';
import { Text, ScrollView, SafeAreaView, Button } from 'react-native';
import RNFS from 'react-native-fs';

export default function TerminalScreen({data}) {
  const [contentOffset, setContentOffset] = useState(0);

  return (
    <SafeAreaView >
      <Button title={'Write to the file'} onPress={() =>
RNFS.write(RNFS.ExternalStorageDirectoryPath + '/data.txt', data.toString())}/>
      <ScrollView
        contentOffset={{x: 0, y: contentOffset}}
        onContentSizeChange={(_, height) => {
          setContentOffset(height);
        }}
      >
        <Text>{'    Дані:'}</Text>
        {!!data && data.map((value, index) => (
          <Text key={index}>{'    ' + value}</Text>
        ))}
      </ScrollView>
    </SafeAreaView>
  );
}
```

Екран який міститься в боковому мені, містить в собі низове меню, для файлів з
двох попередніх додатків

```

ScanScreen import React, { useState, useEffect } from 'react';
import { NativeModules, NativeEventEmitter } from 'react-native';
import { ESPEcg } from './src/ESPEcgTag';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';
import BleManager from 'react-native-ble-manager';

const Tab = createBottomTabNavigator();

const BleManagerModule = NativeModules.BleManager;
const bleEmitter = new NativeEventEmitter(BleManagerModule);
import TerminalScreen from './TerminalScreen';
import GraphicScreen from './GraphicScreen';

export default function DataScreen({deviceId}) {
  const [data, setData] = useState([0]);
  const parseData = (receivedData) => {
    const first = parseInt(receivedData.value[0], 10);
    const second = parseInt(receivedData.value[1], 10);
    return (second << 8) + first;
  };
  const handleDisconnectedPeripheral = data => {
    alert('Disconnected from ' + data.peripheral);
  };
  const handleDidUpdateValueForCharacteristic = (receivedData) => {
    const result = parseData(receivedData);
    setData(prev => [...prev, result]);
  };
  useEffect(() => { if (!deviceId) {return;}
    BleManager.connect(deviceId);
    BleManager.startNotification(deviceId, ESPEcg.service, ESPEcg.characteristic);
    bleEmitter.addListener( 'BleManagerDisconnectPeripheral',
handleDisconnectedPeripheral,);
    bleEmitter.addListener( 'BleManagerDidUpdateValueForCharacteristic',
handleDidUpdateValueForCharacteristic,);
    return () => { bleEmitter.removeAllListeners(); }; }, [deviceId]);

  return ( <NavigationContainer independent={true}> <Tab.Navigator screenOptions={({ route
}) => ({ tabBarIcon: ({ focused, color, size }) => { let iconName; if (route.name ===
'Graphic') { iconName = 'heart-pulse'; return <MaterialCommunityIcons name={iconName}
size={size} color={color} />; } else if (route.name === 'Terminal') { iconName = 'code-
greater-than'; return <MaterialCommunityIcons name={iconName} size={size} color={color}
/>; },
tabBarActiveTintColor: '#2296f3', tabBarInactiveTintColor: 'gray', tabBarShowLabel:
false, headerShown: false, )}} >
    <Tab.Screen name="Graphic"
      options={({ drawerIcon: ({color, size} ) => <MaterialCommunityIcons
size={size} color={color} name="heart-pulse" />,
    }} children={() =>
    <GraphicScreen data={data}/> } />
    <Tab.Screen name="Terminal" options={({ drawerIcon: ({color, size} ) =>
    <MaterialCommunityIcons size={size} color={color} name="code-greater-than" />, }}
      children={() => <TerminalScreen data={data}/> } /> </Tab.Navigator>
    </NavigationContainer>
  );}

```

Екран для читання даних зі збережених раніше файлів

```

import React, { useContext, useState } from 'react';
import { Text, ScrollView, SafeAreaView, Button } from 'react-native';
import DocumentPicker from 'react-native-document-picker';
import RNFS from 'react-native-fs';
import { DataContext } from './AnalyzeScreen';

export default function CalculationScreen() {
  const [contentOffset, setContentOffset] = useState(0);
  const [result, setResult] = useState(null);
  const {setData} = useContext(DataContext);

  async function pickTxtFile() {
    try {

      const chosenFile = await DocumentPicker.pickSingle({
        type: ['text/plain'],
      });
      console.log(
        'Selected file:',
        chosenFile.size
      );
      const fileContent = await RNFS.readFile(chosenFile.uri, 'utf8');
      const parsed = fileContent.split(',').map(x => parseInt(x));
      console.log('parsed ' + parsed[20]);
      setData(parsed);
      setResult(parsed);
    } catch (err) {
      if (DocumentPicker.isCancel(err)) {
        // User cancelled the picker
      } else {
        // Error!
      }
    }
  }

  return (
    <SafeAreaView >
      <Button title={'Read the file'} onPress={pickTxtFile}/>
      <ScrollView
        contentOffset={{x: 0, y: contentOffset}}
        onContentSizeChange={(_, height) => {
          setContentOffset(height);
        }}
      >
        <Text>{'    Дані:'}</Text>
        {
          result != null ? (<Text>{'    ' + result}</Text>)
            : (<Text>No available</Text>)
        }
      </ScrollView>
    </SafeAreaView>
  );
}

```

Екран для встановлення обмеження на читання даних з графіка, який знаходить всередині цього компонента, для можливості перегляду збережених даних та передати їх до дочірнього компонента який містить графік

```

ScanScreenimport React, { useState, useEffect, useRef, createContext } from 'react';
import { ESPEcg } from './src/ESPEcgTag';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';
import { Pressable, Text, TextInput, View, Modal, TouchableOpacity,
TouchableWithoutFeedback } from 'react-native';
const Tab = createBottomTabNavigator();
export const DataContext = createContext();
import GraphicScreen from './GraphicScreen';
import CalculationScreen from './CalculationScreen';

export default function AnalyzeScreen() {
  const [data, setData] = useState([0]);
  const [modalVisible, setModalVisible] = useState(false);
  const [inputValue, setInputValue] = useState('20');
  const inputRef = useRef(null);
  const toggleModal = () => {
    setModalVisible(!modalVisible);
  };
  useEffect(() => {
    if (modalVisible) {
      inputRef.current.focus();
    }
  }, [modalVisible]);

  const handleOutsidePress = (event) => {
    const { locationX, locationY } = event.nativeEvent;
    console.log(locationX + ' ' + locationY);
    const modalView = inputRef?.current?.view;
    alert(modalView);
    if (modalView) {
      const isInsideModal = locationX >= modalView.x && locationX <= modalView.x +
modalView.width &&
      locationY >= modalView.y && locationY <= modalView.y + modalView.height;
      if (!isInsideModal) {
        toggleModal();
      }
    }
  };
  return (
    <DataContext.Provider value={{ data, setData }}>
      <NavigationContainer independent={true}>
        <Header toggleModal={toggleModal} />
        <Tab.Navigator
          screenOptions={({ route }) => ({
            tabBarIcon: ({ focused, color, size }) => {
              let iconName;
              if (route.name === 'Graphic') {
                iconName = 'heart-pulse';
                return <MaterialCommunityIcons name={iconName} size={size} color={color}
/>;
              } else if (route.name === 'Terminal') {
                iconName = 'code-greater-than';

```

```

        return <MaterialCommunityIcons name={iconName} size={size} color={color}
/>;
    }
  },
  tabBarActiveTintColor: '#2296f3',
  tabBarInactiveTintColor: 'gray',
  tabBarShowLabel: false,
  headerShown: false,
)}} <Tab.Screen
  name="Calculation"
  options={{ drawerIcon: ({ color, size }) => ( <MaterialCommunityIcons
size={size} color={color} name="code-equal" /> ), }}
  >
    {() => <CalculationScreen setData={setData} />}
  </Tab.Screen>
  <Tab.Screen
    name="GraphicSaved"
    options={{
      drawerIcon: ({ color, size }) => (
        <MaterialCommunityIcons size={size} color={color} name="heart-pulse" />
      ),
    }}
  >
    {() => <GraphicScreen data={data} contentOffsetProp={inputValue} />}
  </Tab.Screen>
</Tab.Navigator>
<TouchableWithoutFeedback onPress={handleOutsidePress}>
  <Modal animationType="slide" transparent={true} visible={modalVisible}>
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <TouchableOpacity
        onPress={toggleModal}
        style={{ flex: 1, width: '100%', backgroundColor: 'rgba(0,0,0,0.5)',
position: 'absolute' }}
      />
      <View style={{ backgroundColor: '#2296f3', borderRadius: 10, padding: 20,
width: '80%' }}>
        <TextInput
          ref={inputRef}
          inputMode={'numeric'}
          value={inputValue}
          onChangeText={({text}) => setInputValue(text)}
          onSubmitEditing={toggleModal}
          style={{ backgroundColor: 'white', borderRadius: 50, height: 40,
marginVertical: 12, borderWidth: 1, }} />
        <TouchableOpacity onPress={toggleModal} style={{ alignItems: 'center' }}>
          <Text style={{ color: 'white', fontWeight: 'bold' }}>Close</Text>
        </TouchableOpacity>
      </View> </View> </Modal> </TouchableWithoutFeedback> </NavigationContainer>
</DataContext.Provider>
));
function Header({ toggleModal }) {
  return (
    <View style={{ flex: 0.1, flexDirection: 'row', justifyContent: 'flex-
end', alignItems: 'center' }}>
      <Pressable onPress={toggleModal} style={{ padding: 10, borderRadius: 5,
backgroundColor: '#2296f3' }}>
        <Text style={{ color: 'white', fontWeight: 'bold' }}><MaterialCommunityIcons
color="white" name="dots-vertical" size={20} />
        Set graphic offset</Text>
      </Pressable>
    </View>
  );
}

```

Дозволи, які запитує додаток, пов'язані з роботою Bluetooth та зберігання у файл

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.blemanager">
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />
  <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <application
    android:name=".MainApplication"
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:allowBackup="false"
    android:theme="@style/AppTheme"
    android:requestLegacyExternalStorage="true">
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name"
      android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|screenSize|smallestScreenSize|uiMode"
      android:launchMode="singleTask"
      android:windowSoftInputMode="adjustResize"
      android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Підключення Native Module бібліотеки, та бібліотеки для іконок у файлі build.gradle

```
project.ext.vectoricons = [  
    iconFontNames: [ 'MaterialCommunityIcons.ttf' ]  
]  
apply from: file("../..../node_modules/@react-native-community/cli-platform-  
android/native_modules.gradle"); applyNativeModulesAppBuildGradle(project)  
apply from: "../..../node_modules/react-native-vector-icons/fonts.gradle"
```