Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Інтелектуальна система моніторингу та аналізу показників при реабілітації спортсменів
(тема)

Виконав:
здобувач І І року навчання, групи СКСм-23-2
Білич Ю.Є.
(прізвище, ініціали)

Спеціальність
123_*Комп'ютерна інженерія*
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Спеціалізовані комп'ютерні системи
(повна назва освітньої програми)

Керівник: доц. Філіппенко І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп’ютерні
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“ 8 ” 11 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Білич Юрію Євгенійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Інтелектуальна система моніторингу та аналізу показників при реабілітації спортсменів

затверджена наказом по університету від “ 08 ” 11 2024 р. № 1189 Ст

2. Термін подання студентом роботи до екзаменаційної комісії “ 9 ” 01 2025 р.

3. Вхідні дані до роботи

Пульсометр MAX30102

Мікроконтролер ESP32

4. Перелік питань, що потрібно опрацювати у роботі

Аналіз предметної області та постановка задачі

Огляд засобів вирішення задачі

Реалізація системи

Робота та тестування системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2024-08.09.2024	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2024-15.09. 2024	
3	Аналіз методів проектування цифрових фільтрів	16.09.2024-29.09.2024	
4	Вибір налагоджувальної плати та інструментарію для реалізації проекту	30.09. 2024-20.10.2024	
5	Фізична реалізація проекту в налагоджувальній платі	20.10. 2024-20.11. 2024	
6	Макетування спроектованих фільтрів	20.11. 2024-10.12. 2024	
7	Оформлення пояснювальної записки	10.12. 2024-30.12. 2024	
8	Захист проекту	02.01. 2025-25.01. 2025	

Дата видачі завдання 2 вересня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Філіппенко І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 54 сторінки, 9 рисунків, 5 джерел за переліком посилань.

ESP, GPS, BPM, МІКРОКОНТРОЛЕР, ПУЛЬСОМЕТР, МОДУЛЬ, ШВИДКІСТЬ

Розроблена інтелектуальна система моніторингу та аналізу показників при реабілітації спортсменів спортсменів на базі мікроконтролера ESP32

Було проведене дослідження та порівняння компонентів проекту, результатом якого є оптимальний вибір комплектації для створення прототипу. Була створення нейрона мережа для надання рекомендацій на основі отриманних данних. Створено мобільний застосунок для обробки інформації та надання результату користувачу. Було створено функціональну схему пристрою та розроблена програма для спідометру за допомогою мови C++ для отрмання частоти сердцевого скорочення та відправки його до застосунку. Для реалізації проекту були використано: плата розробника ESP32, смартфон на базі Android, пульсометр MAX30102. Розроблена схема підключення усіх елементів до плати.

В ході виконання роботи було зібрано та протестовано прототип.

ABSTRACT

Master’s thesis contains 54 pages, 9 figures, 5 sources according to the list of links.

ESP, GPS, BPM, MICROCONTROLLER, HEART RATE MONITOR, MODULE, SPEED

An intelligent system for monitoring and analyzing indicators during the rehabilitation of athletes based on the ESP32 microcontroller has been developed

A study and comparison of the project components was conducted, resulting in the optimal choice of equipment for the prototype. A neural network was created to provide recommendations based on the data obtained. A mobile application was created to process information and provide the result to the user. A functional diagram of the device was created and a program for the speedometer was developed using the C++ language to take the heart rate and send it to the application. To implement the project, we used the ESP32 development board, an Android smartphone, and a MAX30102 heart rate monitor. A scheme for connecting all the elements to the board was developed.

In the course of the work, the prototype was assembled and tested.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ДСР – датчик серцевого ритму

ICSP – in-circuit serial programming (технологія програмування електронних компонентів)

МК – мікроконтролер

ПЗ – програмне забезпечення

IoT – інтернет речей

ШІ – штучний інтелект

ЧСС – частоту серцевих скорочень

ВСТУП

Ефективне відновлення спортсменів після травм чи інтенсивних навантажень є ключовим фактором у забезпеченні їхньої подальшої продуктивності та зниженні ризику повторних ушкоджень. В останні роки технологічний розвиток у галузі інтелектуальних пристроїв відкриває нові можливості для моніторингу стану спортсменів у реабілітаційний період. Це сприяє глибшому розумінню фізіологічних змін у їхньому організмі та дозволяє оперативно реагувати на будь-які відхилення від норми.

Впровадження інтелектуальних систем у сферу спортивної реабілітації дозволяє не тільки відслідковувати показники здоров'я в режимі реального часу, але й автоматично обробляти ці дані для оцінки загального стану спортсмена та оптимізації процесу відновлення. Зокрема, важливим є моніторинг серцево-судинної активності, оскільки цей параметр є одним із найчутливіших індикаторів фізіологічного стану людини. На основі отриманих даних про частоту серцевих скорочень і рівень кисневого насичення можна виявляти тенденції відновлення, контролювати рівень навантажень і попереджати про можливі ризики перевтоми.

Метою цієї роботи є розробка інтелектуальної системи моніторингу та аналізу фізіологічних показників спортсменів у процесі реабілітації. Основними завданнями системи є безперервний збір даних, їхній аналіз та надання рекомендацій щодо інтенсивності навантажень для кожного спортсмена. Це забезпечить індивідуальний підхід до реабілітації, підвищить її ефективність і безпеку.

Актуальність роботи полягає в інтеграції сучасних технологій у реабілітаційні процеси, що дозволяє створити більш персоналізовану та адаптивну програму відновлення для кожного спортсмена.

1 ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ПРИ РЕАБІЛІТАЦІЇ СПОРТСМЕНІВ

Реабілітація спортсменів – це комплексний процес, спрямований на відновлення фізичних і психологічних сил після травм, оперативних втручань або значних фізичних навантажень. Вона є невід'ємною частиною спортивної діяльності та включає не лише медичні процедури, але й цілеспрямовані програми фізичних тренувань, які допомагають спортсмену повернутися до оптимальної фізичної форми. Ефективна реабілітація дозволяє мінімізувати ризик повторних травм і забезпечити поступове повернення до повноцінних тренувань та змагань.

1.1 Цілі та завдання реабілітації спортсменів

Основною метою спортивної реабілітації є максимальне відновлення функціональних можливостей спортсмена. Залежно від конкретного виду спорту, рівня фізичної підготовленості та характеру травми або навантаження, реабілітація може включати різноманітні процедури та методики. Ключові завдання реабілітації можна розділити на такі етапи:

Зниження болю та набряку – відразу після травми першим завданням є зменшення болю, набряку та запалення. Цього досягають за допомогою медикаментозної терапії, фізіотерапії та відпочинку.

Відновлення рухливості та гнучкості – на другому етапі реабілітації основною метою є повернення рухливості ураженій ділянці, що забезпечується за допомогою спеціальних вправ на розтягування, масажу та теплових процедур.

Покращення сили та витривалості – як тільки болючі відчуття зменшені, а рухливість відновлена, фокус переміщується на покращення

сили, витривалості та загального тонусу м'язів, щоб запобігти повторним травмам.

Повернення до спортивної форми – останній етап реабілітації спрямований на поступове повернення до повноцінних тренувань і змагань. На цьому етапі використовуються більш інтенсивні вправи, які імітують реальні спортивні навантаження.

Кожен з цих етапів вимагає точного контролю за станом спортсмена, що включає як візуальні спостереження фахівців, так і об'єктивні дані про фізіологічні показники, такі як частота серцевих скорочень, рівень кисневого насичення крові, артеріальний тиск, ступінь відновлення м’язів тощо.

1.2 Особливості моніторингу у реабілітації спортсменів

Моніторинг є надзвичайно важливою частиною реабілітаційного процесу, оскільки саме завдяки постійному спостереженню за показниками фізичного стану можна відстежувати прогрес спортсмена і коригувати навантаження. На сучасному етапі технологічного розвитку з'явилися інтелектуальні пристрої, здатні автоматично збирати та аналізувати фізіологічні показники, такі як пульс, насичення крові киснем та інші життєво важливі параметри. Це дозволяє створити більш точні та індивідуалізовані реабілітаційні програми.

Для моніторингу показників спортсменів під час реабілітації часто використовують носимі пристрої, які забезпечують безперервний контроль та миттєве зворотне сповіщення. Такі пристрої дозволяють не лише лікарям та тренерам, але й самому спортсмену розуміти поточний стан організму та коригувати рівень навантаження.

1.3 Інтелектуальні системи моніторингу у спортивній реабілітації

Інтелектуальні системи моніторингу, що базуються на технологіях інтернету речей (IoT) та штучного інтелекту (ШІ), мають значний потенціал для використання у реабілітаційній практиці. Вони забезпечують збір даних про фізіологічний стан спортсмена в режимі реального часу, їх аналіз та інтерпретацію, а також надання рекомендацій щодо подальшого відновлення. Така система може бути налаштована на автоматичне відстеження змін у показниках, таких як частота серцевих скорочень, рівень кисню у крові, температура тіла, активність м’язів та інші параметри.

Завдяки використанню штучного інтелекту можливим стає не тільки фіксувати фактичні значення показників, але й аналізувати їх динаміку та прогнозувати можливі ризики для спортсмена. Це особливо важливо в тих випадках, коли спортсмени проходять інтенсивну реабілітацію після серйозних травм або операцій, коли необхідно постійно контролювати реакцію організму на фізичні навантаження. Таким чином, інтелектуальні системи надають можливість більш персоналізовано підходити до процесу відновлення, мінімізуючи ризики та підвищуючи ефективність реабілітації.

1.4 Використання фізіологічних показників у реабілітації

Серед основних показників, що контролюються під час реабілітації, особливу увагу приділяють серцевому ритму та рівню кисневого насичення крові. Ці показники є критичними для визначення рівня фізичного навантаження, який може бути безпечним для спортсмена на різних етапах відновлення. Частота серцевих скорочень дозволяє оцінити реакцію серцево-судинної системи на навантаження і виявити, чи не перевищують вони допустимих меж. У свою чергу, рівень насичення киснем вказує на здатність організму постачати м’язи киснем під час фізичної активності.

Завдяки використанню пульсометра, який інтегрований у систему моніторингу, можливо не лише контролювати ці показники, а й миттєво реагувати на відхилення від норми. Після кожного сеансу реабілітаційних занять система може фіксувати зміни показників і автоматично рекомендувати коригування програми залежно від динаміки стану спортсмена.

1.5 Значення інтелектуальної системи для спортивної медицини

Запровадження інтелектуальної системи для моніторингу та аналізу показників у процесі реабілітації спортсменів сприятиме підвищенню точності діагностики і полегшенню процесу адаптації до спортивних навантажень. Така система забезпечить ефективне управління відновлювальним процесом, дозволяючи вчасно змінювати інтенсивність і обсяг навантажень. Це важливо для спортсменів, які часто працюють на межі своїх фізичних можливостей і ризикують зазнати повторних травм у разі недостатнього відновлення.

Окрім того, інтелектуальні системи надають можливість лікарям та тренерам аналізувати великий масив даних, який зберігається і може бути використаний для довгострокового моніторингу стану спортсмена. Це особливо корисно у випадках хронічних травм або проблем, які можуть проявитися у майбутньому.

1.6 Прилади для моніторингу серцевого ритму

Прилади для моніторингу серцевого ритму широко застосовуються як в медичній практиці, так і в спортивній реабілітації для контролю стану серцево-судинної системи та адаптації фізичних навантажень. Їх використовують не тільки лікарі для діагностики й спостереження за пацієнтами, але й спортсмени та тренери для оцінки інтенсивності тренувань

і відстеження відновлення організму. У реабілітаційних процесах, особливо після серцево-судинних захворювань або травм, такі пристрої допомагають забезпечити безпечне та ефективне повернення до фізичної активності. Існує кілька основних видів приладів для моніторингу серцевого ритму, кожен з яких має свої особливості та підходить для різних умов застосування. Розглянемо деякі з них, їх можливості та характеристики.

Рисунок 1.1 – Портативні пристрої для вимірювання пульсу

Холтерівські монітори є одним з найпоширеніших пристроїв для довготривалого моніторингу серцевого ритму. Це портативний електрокардіограф (ЕКГ), який записує електричну активність серця протягом 24-48 годин або навіть декількох днів. Пристрій фіксується на тілі пацієнта за допомогою електродів, які підключені до маленького

портативного записувального блоку. Холтерівський монітор дозволяє лікарям відстежувати зміни серцевого ритму протягом доби, включаючи періоди активності, сну та стресових ситуацій, що робить його цінним інструментом для діагностики аритмій та інших порушень.

Кардіомонітори, що використовуються спортсменами під час тренувань, зазвичай є компактними пристроями, що надягаються на груди або зап’ястя. Вони можуть показувати частоту серцевих скорочень (ЧСС) в реальному часі, що дозволяє контролювати інтенсивність тренування. Більшість таких пристроїв передають дані на смартфон або смарт-годинник через Bluetooth або ANT+, дозволяючи спортсменам та тренерам зручно спостерігати за показниками під час руху. Ці пристрої допомагають контролювати, щоб серце не перевантажувалося, що важливо для уникнення перенапруження й травм.

Пульсометри на зап'ясті такі, як фітнес-браслети та смарт-годинники з вбудованими пульсометрами є популярними як серед спортсменів, так і серед людей, які просто стежать за своїм здоров’ям. Вони працюють на основі оптичного методу, який використовує світлодіоди для визначення зміни об’єму крові в судинах на зап'ясті. Такі пристрої зручні у використанні, не вимагають спеціальної підготовки і дозволяють контролювати ЧСС в реальному часі. Деякі моделі також можуть фіксувати рівень насичення крові киснем і надавати оцінки рівня стресу. Незважаючи на зручність, точність таких пристроїв може поступатися спеціалізованим медичним приладам, особливо під час інтенсивних рухів або при певних типах шкіри.

Медичні монітори з функцією електрокардіограми (ЕКГ) забезпечують високоточний моніторинг серцевої активності та використовуються для діагностики серйозних захворювань. Такі пристрої можуть фіксувати не тільки ЧСС, а й повну кардіограму, що дозволяє лікарям аналізувати детальну картину електричної активності серця. Вони зазвичай використовуються в клініках, але деякі сучасні моделі дозволяють пацієнтам проводити вимірювання вдома і передавати дані лікарям для віддаленого

моніторингу. Такі пристрої особливо корисні в реабілітації пацієнтів з кардіологічними захворюваннями, оскільки дозволяють відстежувати прогрес і адаптувати терапію.

Пульсоксиметри використовують для вимірювання рівня насичення киснем у крові та частоти серцевих скорочень. Ці пристрої особливо важливі для людей із захворюваннями дихальної системи, оскільки низький рівень кисню в крові може бути небезпечним. Пульсоксиметри, що кріпляться на палець, забезпечують швидке вимірювання і можуть бути корисними під час фізичної активності, коли необхідно стежити не лише за серцевим ритмом, але й за станом дихання.

1.7 Мета роботи

Метою кваліфікаційної роботи є розробка системи моніторингу за станом здоров’я для реабілітації спортсменів. Завдяки використання мінімуму периферійних модулей, система проста та дешева в реалізації. Через велику кількість модулей та датчиків на ринку, прототип легко оновити та додати нові функції.

Об’єктом є система моніторингу за станом здоров’я.

Предметом дослідження є розробка інтелектуальної системи за станом здоров’я. В більшості систем використовуються мікроконтролери, які забезпечують інтеграцію датчиків і обробку даних у реальному часі. У поєднанні з нейронними мережами вони дозволяють створювати інтелектуальні рішення, які аналізують великі обсяги даних, прогнозують події та приймають автономні рішення, значно підвищуючи ефективність і точність систем моніторингу та управління.

2 РОЗРОБКА МОДЕЛІ МОНІТОРИНГУ ТА АНАЛІЗУ ПОКАЗНИКІВ

2.1 Модель моніторингу та аналізу показників

На основі проведеного аналізу різноманітних рішень щодо оптимізації та автоматизації процессу надання рекомендацій для реабілітації, було запропоновано наступну програмно-апаратну інтелектуальну модель. Вона складається з наступних модулей:

Модуль «Датчики» необхідний для збору інформації таких як показники частоти серцевих скорочень (пульсу) та насиченості крові киснем (SpO2). Від цього модулю через провідне підключення данні поступають до ESP32.

Модуль «Система сбору даних» ESP32 здійснює попередню обробку отриманих даних, включаючи фільтрацію шуму, аналіз стабільності показників і їх підготовку до подальшого аналізу.

Модуль «Нейромережа» Після отрмання даних, вони передаються в нейронну мережу (може бути розгорнута як локально, так і в хмарі), яка аналізує показники спортсмена та визначає рекомендації для реабілітації. Отримані результати аналізу передаються на мобільний телефон для візуалізації у вигляді графіків, сповіщень або звітів.

Модуль «ДАТАСЕТ» використовується для тренування нейронноъ мережі. Складається з даннихпро час тренування, пульсу протягом тренування та пройцденної відстані.

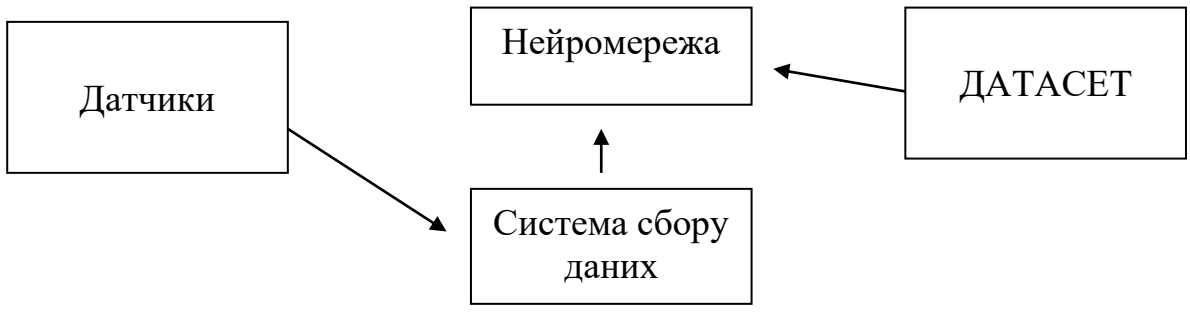

Рисунок 2.1 – Модель моніторингу та аналізу показникі

При розробці інтелектуальної системи моніторингу для реабілітації спортсменів було обрано мікроконтролери сімейства ESP, через вже вбуд інтегровані контролери Wi-Fi і Bluetooth. Було проведено порівняння між ESP32 та ESP8266.

ESP32 є наступником ESP8266. Він додає, ядро процесора, швидкий Wi-Fi, більше GPIO і підтримує Bluetooth 4.2 і низької енергії Bluetooth. Крім того, ESP32 постачається з датчиками дотику, вбудованим ефектом Холла, датчиками та датчиками температури. Обидві плати дешеві, але ESP32 трохи дорожчий. Обидва чіпи мають 32-бітний процесор. ESP32 - двоядерний процесор із частотою від 160 до 240 МГц, а ESP8266 - одноядерний, що працює на частоті 80 МГц.

Для даного проекту був обраний мікроконтролер ESP32, через наявність Bluetooth 4.2 та більшу кількість вбудованих інтерфейсів, таких як I²C, SPI, UART, які дозволяють легко інтегрувати модуль з різними датчиками та іншими периферійними пристроями.

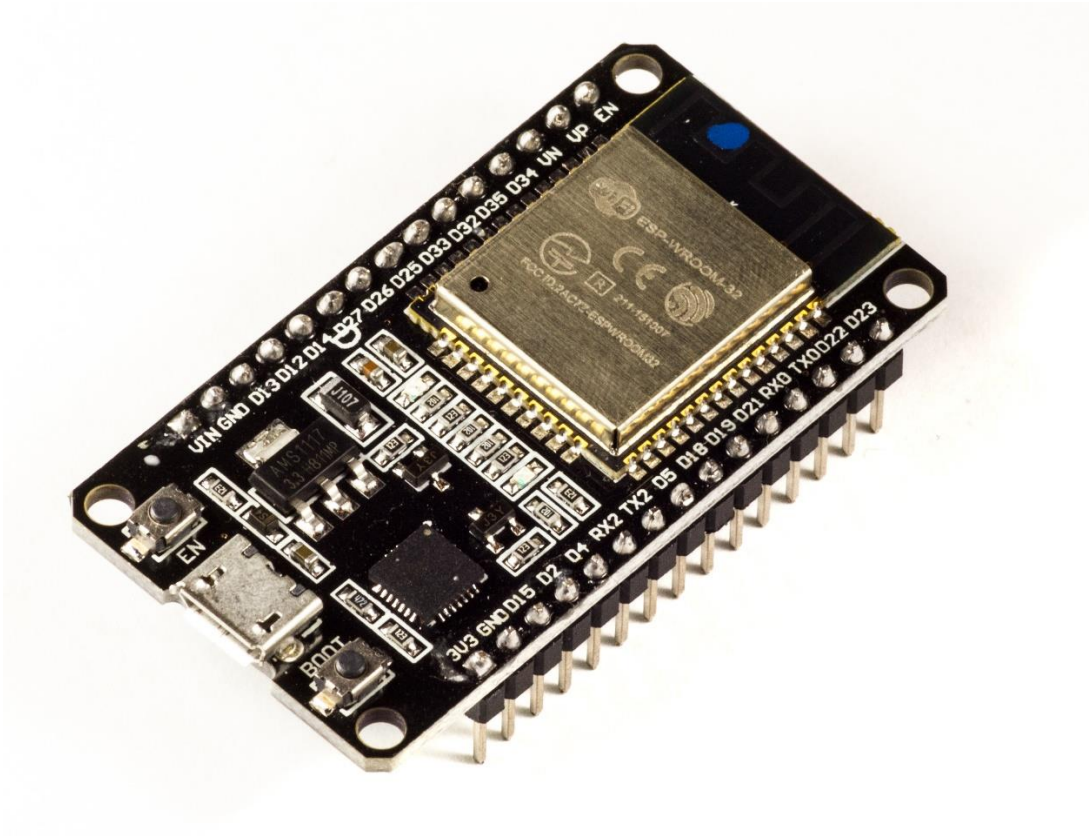

Рисунок 2.2 – Зовнішній вигляд ESP32

Технічні характеристики модуля:

- USB-UART конвертер: CP2102;
- напруга живлення: 5В;
- Wi-Fi Стандарти: FCC/CE/IC/TELEC/KCC/SRRC/NCC;
- протоколи: 802.11 b/g/n/d/e/i/k/r (802.11n до 150 Мбіт/с);
- частотний діапазон: ГГц 2.4 ~ 2.5;
- Bluetooth протоколи: Bluetooth v4.2 BR/EDR та BLE specification;
- апаратні засоби та інтерфейси: SD, UART, SPI, SDIO, I²C, LED PWM, Motor PWM, I²S, I²C, IR;
- датчики на борту: Hall sensor, температурний датчик;
- живлення мікромодуля: 2.2 ~ 3.6;
- робочий струм, середній: 80 мА;
- діапазон робочих температур: -40°C ~ +85°C;
- програмне забезпечення: Режими Wi-Fi Station/softAP/SoftAP+station/P2P;
- оновлення ПЗ: UART Download / OTA (по мережі) / download and write firmware via host;
- мережеві протоколи: IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT;
- налаштування користувача: AT instruction set, cloud server, Android/iOS App.

ESP32 може бути заживленний від USB або від зовнішнього джерела живлення - тип джерела вибирається автоматично. У разі живлення від акумулятора/батареї, її дроти необхідно підключити до висновків Gnd та Vin роз'єму POWER.

Напруга зовнішнього джерела живлення може бути в межах від 6 до 20 В. Однак зменшення напруги живлення нижче 7В призводить до зменшення напруги на виведенні 5V, що може стати причиною нестабільної роботи пристрою. Використання напруги більше 12В може призводити до перегріву стабілізатора напруги та виходу плати з ладу. З огляду на це рекомендується використовувати джерело живлення з напругою в діапазоні від 7 до 12В.

Порти вводу-виводу забезпечують універсальність підключення до різних датчиків, актуаторів та інших периферійних пристроїв. Вони підтримують широкий спектр функцій, включаючи цифровий і аналоговий ввід-вивід, ШІМ, I2C, SPI, UART.

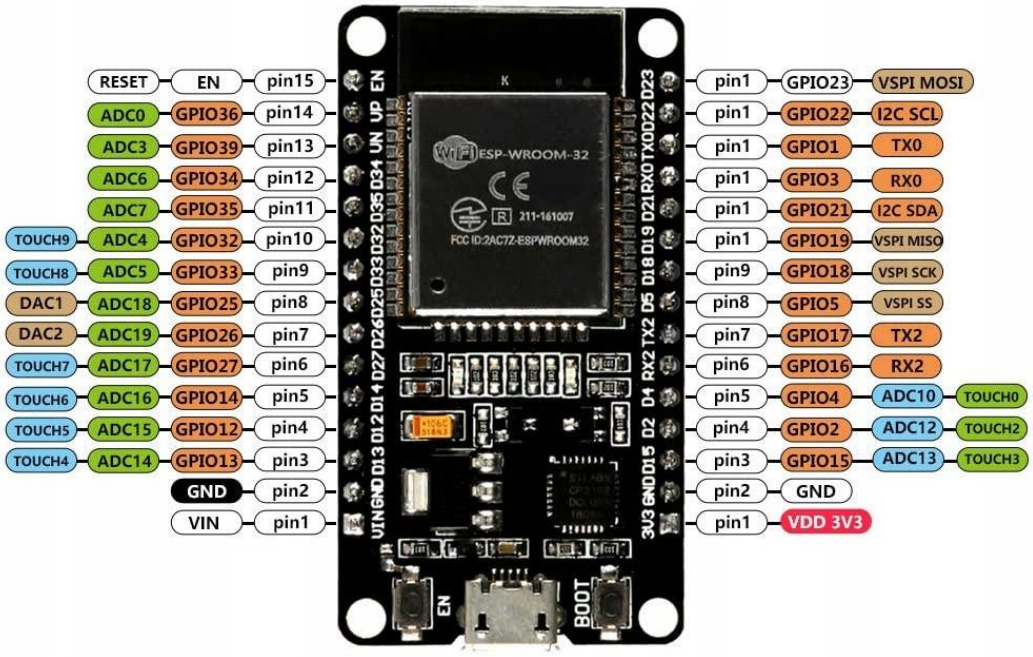

Рисунок 2.3 – Порти введення/виведення плати

2.1.1 Переваги використання ESP32 у спортивній реабілітації

ESP32 є ефективним для створення портативних та енергоефективних систем, які необхідні в реабілітаційних програмах. Його переваги для спортивної реабілітації включають:

Бездротове з'єднання – модуль підтримує Wi-Fi та Bluetooth, що дозволяє передавати дані на віддалений сервер, мобільний додаток або інші пристрої для моніторингу в реальному часі. Це забезпечує тренерам і лікарям доступ до актуальних даних незалежно від місцезнаходження спортсмена.

Підтримка інтелектуальних алгоритмів – завдяки своїй обчислювальній потужності ESP32 може працювати з простими алгоритмами для попередньої обробки даних, такими як фільтрація шумів або обчислення середніх значень, перед передачею їх на основний сервер або хмару для більш складного аналізу.

Низьке енергоспоживання – ESP32 розроблений для енергоефективного функціонування, що дозволяє створювати пристрої, які можуть працювати від батареї протягом тривалого часу. Це зручно для носимих пристроїв, які спортсмен може використовувати протягом усього реабілітаційного процесу.

ESP32 є ключовим елементом інтелектуальної системи моніторингу, оскільки забезпечує зв'язок між датчиками і кінцевими користувачами — лікарями, тренерами та самими спортсменами. Завдяки можливості обробки даних та бездротової передачі інформації, система на основі ESP32 може автоматично збирати та аналізувати дані про стан спортсмена в реальному часі, а також надавати зворотний зв'язок. Це дозволяє оперативно реагувати на зміни у стані спортсмена та коригувати реабілітаційну програму залежно від фізіологічних показників.

2.2 Датчик пульсу MAX30102

Для ефективного моніторингу фізіологічних показників спортсменів під час реабілітації необхідні надійні й точні датчики, здатні безперервно відслідковувати ключові параметри, такі як частота серцевих скорочень (ЧСС) та рівень насиченості киснем у крові. Одним із таких пристроїв є датчик MAX30102 – компактний, високочутливий оптичний пульсометр, який широко застосовується для медичних і спортивних цілей.

MAX30102 – це датчик оптичного типу, який використовує фотоплетизмографію для вимірювання змін у кровотоці, дозволяючи отримувати дані про ЧСС та SpO. Він складається з двох світлодіодів (червоного та інфрачервоного) і фотодетектора, які працюють разом для аналізу рівня поглинання світла кров’ю у різних фазах серцевого циклу. Завдяки такому принципу роботи MAX30102 може безперервно фіксувати зміни у частоті серцевих скорочень та рівень кисню, що робить його надзвичайно корисним у реабілітаційних програмах для спортсменів.

Цей пульсометр має компактні розміри та низьке енергоспоживання, що робить його придатним для інтеграції в портативні та носимі пристрої. Його особливості дозволяють проводити тривале безперервне спостереження за спортсменом без необхідності переривання реабілітаційного процесу.

Основна цінність датчика MAX30102 для спортивної реабілітації полягає у його здатності точно і швидко фіксувати важливі показники, що відображають реакцію серцево-судинної системи на навантаження. Ця інформація є критично важливою для адаптації індивідуальних програм реабілітації:

Безперервний моніторинг серцевого ритму – дозволяє лікарям і тренерам оперативно реагувати на відхилення у показниках серцевого ритму, що особливо важливо під час поступового збільшення фізичного навантаження.

Оцінка рівня насичення киснем – під час інтенсивних фізичних навантажень, особливо у процесі відновлення, дуже важливо відстежувати, чи забезпечується організм необхідною кількістю кисню.

Швидка реакція на зміни – завдяки високій чутливості MAX30102 забезпечує оперативний збір даних у режимі реального часу, що дозволяє своєчасно вносити корективи у реабілітаційний процес.

Технічні особливості та інтерфейс MAX30102 наступні. Він підтримує цифровий інтерфейс I^2C, що дозволяє інтегрувати його з різними мікроконтролерами, включаючи ESP32. Його спектральний діапазон спеціально налаштований на фільтрацію зовнішнього шуму, що забезпечує точні результати навіть у випадку стороннього освітлення. Це робить його надійним рішенням для моніторингу в умовах різного рівня освітлення, як-от у залі для тренувань чи на відкритому повітрі.

Завдяки малим розмірам і простоті інтеграції, датчик MAX30102 також може бути використаний у складі компактних носимих пристроїв, що значно підвищує комфорт спортсмена під час реабілітації. Це дозволяє спортсмену

не відчувати значного дискомфорту при носінні датчика і продовжувати фізичні заняття без додаткових обмежень.

Технічні характеристики:

- вимірювані параметри: частота пульсу і насичення крові та SpO2;
- напруга живлення: 3,3 В (внутрішній стабілізатор на 1,8 В);
- струм в режимі вимірювання: 1.2 мА;
- струм в режимі сну: до 10 мкА;
- інтерфейс: I2C;
- напруга інтерфейсу I2C: 3.3 В;
- максимальна частота інтерфейсу: 400 кГц;
- розміри: 18.5 х 14.5 х 3 мм.

MAX30102 надає можливість автоматично збирати й аналізувати дані про стан серцево-судинної системи спортсмена. За допомогою обчислювальних алгоритмів на основі штучного інтелекту система може відстежувати і прогнозувати фізіологічні зміни, допомагаючи уникати ризику перевантажень та запобігаючи можливим загостренням. Це дозволяє досягти вищого рівня безпеки у процесі реабілітації та підтримати спортивну форму на належному рівні.

Для коректної роботи модуля його необхідно правильно під’єднати. Пульсометр MAX30102 підтримує інтерфейс I^2C, що дозволяє легко підключити його до ESP32 для передачі даних про частоту серцевих скорочень та рівень кисню в крові.

Особливість реалізації полягає у переназначенні стандартних пінів ESP32 для підключення датчика. У стандартному випадку для інтерфейсу I^2C використовуються GPIO 21 (SDA) та GPIO 22 (SCL). Проте в рамках цього проєкту контакти були переназначені для забезпечення сумісності з іншими компонентами системи. Після переназначення сигнал SDA підключено до контакту D5, а сигнал SCL – до контакту D4 на платі ESP32. Це переназначення дозволяє більш гнучко налаштувати систему і зберегти її сумісність з іншими модулями або периферійними пристроями.

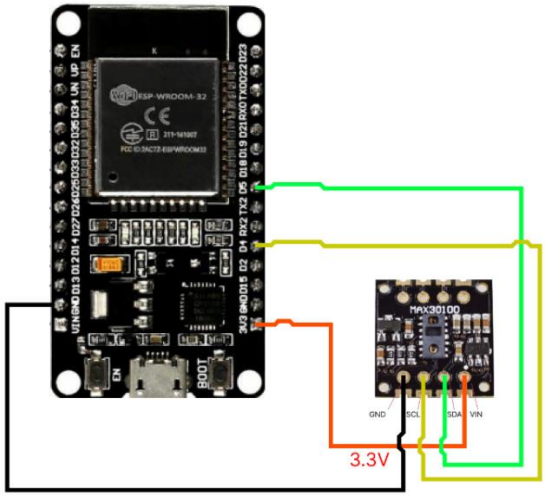

Рисунок 2.4 – Схема підключення

MAX30102 використовує оптичний метод вимірювання, що дозволяє йому точно фіксувати зміни у кровотоці, а з'єднання з ESP32 за допомогою інтерфейсу I²C забезпечує надійний обмін даними. Контакт SDA передає інформацію від датчика до ESP32, а SCL синхронізує цей процес. Додатково для коректної роботи система потребує підключення до живлення (3.3 В) та заземлення (GND).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Складові компоненті проекту

Для реалізації інтелектуальної системи моніторингу та аналізу показників при реабілітації спортсменів було розроблено програмний комплекс, що складається з кількох компонентів: мобільного додатку на основі React Native, прошивки для модуля ESP32, а також серверної частини з інтегрованою нейронною мережею для аналізу даних.

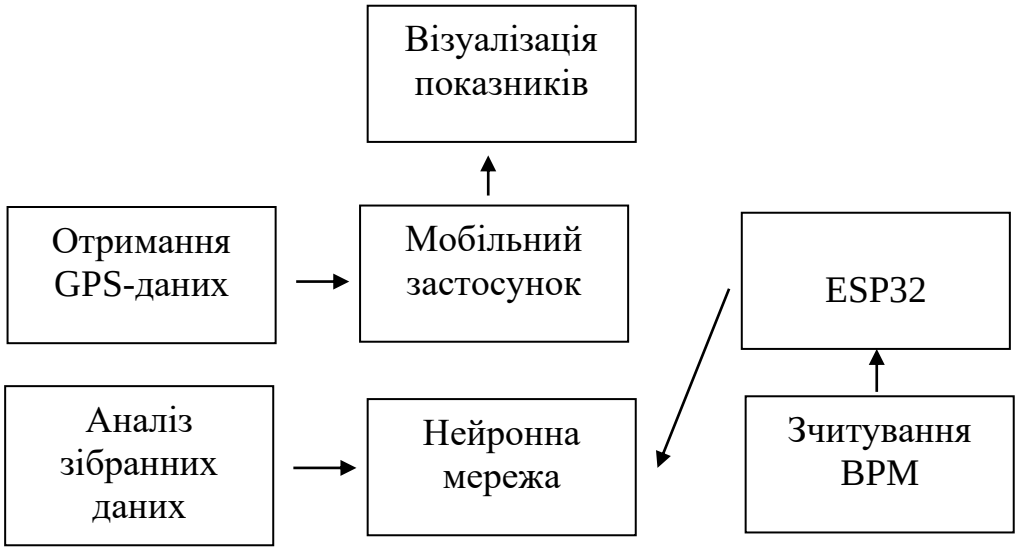

Рисунок 3.1 – Прогамна модель проекту

1. Мобільний додаток

Запропонований мобільний додаток є основним інтерфейсом для користувача. Його було реалізовано на базі React Native. Функціональність мобільного додатку включає в себе:

– збір даних з пульсометра (MAX30102) через модуль ESP32;
– отримання GPS-даних (координати та швидкість) зі смартфона;
– візуалізація реабілітаційних показників у реальному часі;
– передача зібраних даних на сервер для подальшого аналізу.

Для реалізації комунікації між мобільним додатком і ESP32 використовувалася точка доступу Wi-Fi. Взаємодія з GPS-модулем смартфона реалізована за допомогою стандартних API React Native, таких як react-native-geolocation-service. Дані про швидкість і координати передаються у вигляді JSON-пакетів.

2. Програмна частина для ESP32

ESP32 виступає посередником між пульсометром MAX30102 і мобільним додатком. Функції прошивки:

– читання даних із сенсора MAX30102 через інтерфейс I2C;

– обробка даних для отримання значень частоти серцевих скорочень (ЧСС) та рівня насиченості киснем крові (SpO2);

– передача оброблених даних на мобільний додаток через Bluetooth Low Energy (BLE).

Для точного вимірювання ЧСС і SpO2 були використані алгоритми обробки сигналів, надані бібліотекою для MAX30102.

3. Нейронна мережа для аналізу даних

Нейронна мережа може бути розгорнута як локально, так і в хмарі, та відповідає за обробку і аналіз даних, отриманих від користувачів. Основні завдання нейронної мережі:

– аналіз зібраних даних (ЧСС, SpO2, швидкість, маршрути) для виявлення відхилень у процесі реабілітації;

– надання рекомендацій щодо коригування фізичних навантажень.

Нейронна мережа була реалізована за допомогою фреймворку TensorFlow. Для навчання використовувалися попередньо зібрані дані спортсменів різних вікових і фізичних категорій. Модель інтегрована в сервер через REST API, що дозволяє обробляти дані у реальному часі.

Для забезпечення злагодженої роботи всіх компонентів була реалізована наступна модель (рис 3.1):

1) ESP32 збирає дані з пульсометра і передає їх на мобільний додаток;

2) мобільний додаток комбінує ці дані з GPS-координатами та швидкістю;

3) за допомогою нейронної мережі ці дані аналізуються та виводятся на головний екран мобільного додатку (наприклад, попередження про перевищення допустимих фізичних навантажень).

5. Тестування та оптимізація:

– програмний комплекс був протестований на різних пристроях і в різних умовах для забезпечення коректної роботи;

– перевірено стабільність зв’язку між ESP32 та мобільним додатком;

– тестування нейронної мережі на реальних даних для оцінки точності прогнозів.

3.2 Структура файлів у проекті

Структура файлів у проєкті Expo організована так, щоб забезпечити зручність роботи з різними компонентами додатку та їх інтеграцію. У центрі проєкту знаходиться папка app, яка визначає основну навігацію. Вона містить файли, що відповідають за різні екрани програми. Наприклад, файли index.tsx та explore.tsx представляють окремі маршрути, а файл _layout.tsx використовується для налаштування вкладок (табів), забезпечуючи їхнє правильне функціонування. Уся логіка навігації побудована на основі структури цієї папки.

Окрім папки app, проєкт включає папку assets, яка призначена для зберігання ресурсів додатку, таких як зображення, шрифти та інші медіафайли. Це дозволяє централізовано керувати медіаконтентом, що використовується в додатку. Для побудови інтерфейсу використовуються повторно застосовувані компоненти, які знаходяться в папці components. Вони забезпечують ефективність розробки, оскільки їх можна легко використовувати в різних частинах програми.

Конфігураційні файли, такі як app.json і package.json, забезпечують базові налаштування проєкту. Файл app.json містить параметри Expo, наприклад, назву, значки та інші глобальні налаштування додатку. Файл package.json відповідає за залежності проєкту, дозволяючи легко додавати нові бібліотеки чи скрипти.

Завдяки такій структурі забезпечується логічна організація коду, що спрощує як додавання нових функцій, так і підтримку існуючих. Розробники можуть швидко знаходити потрібні файли, вносити зміни та масштабувати проєкт без зайвих ускладнень.

3.3 Створення додатку

Лістинг 3.1 - приховування SplashScreen після завантаження додатку.

```
import {useCallback} from 'react';
import {Stack} from "expo-router";
import * as SplashScreen from "expo-splash-screen";
const Layout = () => {
    const onLayoutRootView = useCallback(async () => {
        if (SplashScreen) {
            await SplashScreen.hideAsync();
        }
    }, [])
    return <Stack onLayout={onLayoutRootView}/>
};
export default Layout;
```

Цей фрагмент коду є частиною React Native додатку, створеного за допомогою expo-router та expo-splash-screen. Він використовується для приховування SplashScreen після завантаження додатку.

Імпортуються функції useCallback з react для оптимізації виклику функції onLayoutRootView, а також SplashScreen з expo-splash-screen для роботи зі SplashScreen.

Компонент Layout повертає Stack, що використовується для створення навігаційної системи в додатку.

Функція onLayoutRootView викликається при рендері компонента Stack. Вона викликає SplashScreen.hideAsync(), щоб приховати SplashScreen, коли додаток повністю завантажений і готовий до використання.

Цей підхід дозволяє забезпечити плавний перехід від SplashScreen до основного контенту додатку після завантаження.

Лістинг 3.2 – Імпорти

```
import {useCallback, useEffect, useRef, useState} from 'react';
import {SafeAreaView, Text, ToastAndroid, TouchableOpacity} from
"react-native";
import {MaterialCommunityIcons} from "@expo/vector-icons";
import {Stack, useFocusEffect, useRouter} from "expo-router";
import * as Location from "expo-location";
import AsyncStorage from "@react-native-async-storage/async-
storage";
import {ActionButton, DataChart, StatsCardsList,
TrainingHistory} from "../components/home";

import {COLORS, SIZES} from "../constants";
import haversine from "haversine-distance";
import {formatTime} from "../utils";

import {getModel, predict} from "../model";
```

Імпорти (Imports):

- import {useCallback, useEffect, useRef, useState} from react: Імпорти з React, які дозволяють використовувати хуки (hooks) для управління станом та ефектами, а також рефами (refs) для доступу до об'єктів DOM;
- import {SafeAreaView, Text, ToastAndroid, TouchableOpacity} from react-native: Імпорти компонентів з бібліотеки React Native для побудови інтерфейсу користувача, таких як текст, кнопки, та системні сповіщення;
- import {MaterialCommunityIcons} from @expo/vector-icons: Імпорт бібліотеки іконок Material Community Icons, яка дозволяє використовувати різноманітні значки в додатку;

– import {Stack, useFocusEffect, useRouter} from expo-router: Імпорт компонентів з Expo Router для навігації в додатку та моніторингу фокусу на екран;

– import * as Location from expo-location: Імпорт бібліотеки Location з Expo для доступу до функцій GPS на мобільному пристрої;

– import AsyncStorage from @react-native-async-storage/async-storage: Імпорт бібліотеки AsyncStorage для збереження даних в локальному сховищі пристрою;

– import {ActionButton, DataChart, StatsCardsList, TrainingHistory} from ../components/home: Імпорт користувацьких компонентів, які належать до головної сторінки додатку (home).

Імпорт постійних і функцій:

– import {COLORS, SIZES} from ../constants: Імпорт постійних значень для кольорів та розмірів, які використовуються для уніфікації стилю інтерфейсу;

– import haversine from haversine-distance: Імпорт функції haversine-distance для обчислення відстані між двома точками на основі координат GPS;

– import {formatTime} from ../utils: Імпорт допоміжної функції для форматування часу.

Основна логіка:

– import {getModel, predict} from ../model: Імпорт функцій для роботи з нейронною мережею — завантаження моделі та здійснення прогнозів. Цей фрагмент коду налаштований для взаємодії з API Location, користування GPS і пульсометром для збору даних, а також для обчислення відстаней та аналізу зібраної інформації;

– useEffect і useFocusEffect: Використовуються для моніторингу стану екрана і виконання певних дій, коли екран набуває фокусу. Це дозволяє контролювати, чи додаток активно збирає дані і чи не слід призупинити процес, коли додаток не використовується;

– useState і useRef: Хуки React для управління станом (наприклад, поточне положення GPS) та рефами (наприклад, для збереження посилань на компоненти);

– Location.getCurrentPositionAsync(): Використовується для доступу до поточного положення GPS;

– AsyncStorage.getItem(): Використовується для збереження і читання даних з локального сховища для довготривалого зберігання даних, таких як історія тренувань.

Лістинг 3.3 – Декларація змінних і хуків

```
const HomePage = () =>
const router = useRouter();
const [history, setHistory] = useState([]);
const [bpm, setBpm] = useState(0);
const [speed, setSpeed] = useState(0);
const [_, setLastCoords] = useState({});
const [distance, setDistance] = useState(0);
const [permissionGranted, setPermissionGranted] = useState(false);
const [isRunning, setIsRunning] = useState(false)
const [time, setTime] = useState(0);
const [graphData, setGraphData] = useState([]);
const [chartMode, setChartMode] = useState("bpm");
const [model, setModel] = useState(null);
const [recommendation, setRecommendation] = useState('');
const locationSubRef = useRef(null);
const pulseSubRef = useRef(null);
const timerRef = useRef(null);
```

Опис логіки збору даних та обробки

Взаємодія з Location API: Через expo-location API цей фрагмент коду підписується на оновлення GPS координат користувача. Кожного разу, коли координати оновлюються, використовується haversine для розрахунку відстані між поточним і попереднім положенням. Додаток також слідкує за дозволом користувача на використання GPS і пульсометру, і оновлює відповідний стейт.Робота з пульсометром: Підписка на оновлення значень

пульсу дозволяє отримувати миттєві показники, які зберігаються в стейті bpm.

Таймер використовується для відслідковування часу тренування. Коли тренування починається або зупиняється, таймер буде ініціалізовано або зупинено відповідно.

Збір даних для графіка (graphData) дозволяє відстежувати тренування у реальному часі. Користувач може переключатися між режимами "bpm" (пульс) і "speed" (швидкість) для моніторингу різних параметрів.

Використання getModel() та predict() для обробки даних зібраних сенсорів дозволяє робити прогноз щодо стану спортсмена і надає рекомендації, які показуються в recommendation

Лістинг 3.4 – асинхронне отримання BPM

```
const fetchBpm = async () => {
        try {
            console.log("fetch bpm")
            const controller = new AbortController();
            const abort = setTimeout(() => {
                controller.abort();
            }, 5000);
            const res = await fetch("http://192.168.4.1/get-
bpm", {signal: controller.signal});
            const data = await res.json();
            setBpm(data?.bpm);
        } catch (err) {
            console.error("Error fetching BPM:", err);
            setBpm(0);
        }
    }
```

Метод fetchBpm використовує аборт-контролер для встановлення таймауту в 5 секунд, аби скасувати запит у випадку затримки. Запит здійснюється до модуля ESP32, який доступний за IP-адресою "http://192.168.4.1/get-bpm".

Під час запиту асинхронно чекаємо відповідь та перетворюємо її у форматі JSON. Якщо відповідь є успішною, дані зберігаються у змінній data,

і з неї ми отримуємо значення bpm, оновлюючи стан частоти серцевих скорочень за допомогою setBpm. Якщо ж запит завершився з помилкою або дані недоступні, ми відображаємо помилку в консолі і встановлюємо bpm на 0, що свідчить про відсутність коректної інформації. Це дозволяє користувачам застосунку знати, коли неможливо отримати актуальні дані про стан їхнього здоров’я.

Лістинг 3.5 – обробка даних геолокації

```
const locationProcessing = ({coords}) => {
        const {latitude, longitude, speed} = coords;
        setSpeed(speed || 0);
        setLastCoords(prevCoords => {
            if (prevCoords.latitude) {
                const distanceDifference = haversine(prevCoords, 
{latitude, longitude}) / 1000;
                setDistance(prevDistance => prevDistance + 
distanceDifference);
            }

            return {latitude, longitude};
        });
    };
```

Цей код відповідає за обробку даних геолокації, коли приходять оновлення від GPS. Функція locationProcessing отримує об’єкт coords, який містить поточні координати — широту (latitude), довготу (longitude) і швидкість (speed). Швидкість оновлюється за допомогою функції setSpeed(speed || 0), яка встановлює значення швидкості або 0, якщо швидкість не надходить.

Потім функція setLastCoords використовується для збереження нових координат у стані. Якщо вже є попередні координати, обчислюється різниця у відстані між попередніми і новими координатами за допомогою haversine, а результат конвертується з метрів в кілометри. Ця різниця додається до поточної загальної відстані, яку ми зберігаємо в setDistance(prevDistance => prevDistance + distanceDifference).

Нові координати повертаються для подальшого використання у застосунку, дозволяючи оновлювати інформацію про позицію користувача і обчислювати нові значення відстані.

Лістинг 3.6 – запит на дозвіл доступу до геолокації

```
const getLocation = async () => {
        try {
            console.log("fetching location started");
            if (!permissionGranted) {
                const {status} = await
Location.requestForegroundPermissionsAsync();
                if (status !== 'granted') {
                    ToastAndroid.show("Permission to access
location was denied", ToastAndroid.LONG);
                    return;
                }
                setPermissionGranted(true);
            }

            return await Location.watchPositionAsync(
                {accuracy: Location.Accuracy.Highest,
distanceInterval: 5},
                locationProcessing
            );


        } catch (error) {
            ToastAndroid.show('Error fetching location: ' +
error.message, ToastAndroid.LONG);
        }
    };
```

Цей блок коду в React застосунку виконує запит на дозвіл доступу до геолокації. Якщо дозвіл ще не надано, він запитує його у користувача. Якщо дозвіл не наданий, виводиться повідомлення через ToastAndroid.show, і функція завершиться. Якщо дозвіл отримано, встановлюється стан permissionGranted у true. Далі викликається Location.watchPositionAsync, яка починає моніторинг позиції з найвищою точністю і заданим інтервалом у 5 метрів. Цей моніторинг буде зупинено, якщо стан зміниться або в разі

помилки. У разі виникнення помилки, висвітлюється повідомлення через ToastAndroid.show, яке інформує користувача про проблему.

Лістинг 3.7 – отрмання історії

```
const getHistory = async () => {
        try {
            const storedHistory = await
AsyncStorage.getItem("history");
            const history = storedHistory ?
JSON.parse(storedHistory) : [];
            setHistory(history);
        } catch (error) {
            console.error("Error getting history:", error);
        }
    };
```

Цей блок коду отримує історію з локального сховища AsyncStorage. Він намагається взяти дані, які зберігаються під ключем "history". Якщо такі дані є, то вони парсяться з JSON у формат JavaScript, і результат зберігається у змінній history. Якщо дані не знайдено, то створюється порожній масив, що свідчить про відсутність попередньої історії. Далі, за допомогою setHistory(history), оновлюється стан з новими даними історії. Якщо виникає помилка під час виконання, вона логгиться у консолі.

Лістинг 3.7 – Збереження історії

```
const saveHistoryRecord = async () => {
        try {
            if (graphData.length === 0) {
                return;
            }
            const newRecord = {
                data: graphData,
                timestamp: Date.now(),
            };

            setHistory(prevHistory => {
                const updatedHistory = [...prevHistory,
newRecord];
```

```
                AsyncStorage.setItem("history",
JSON.stringify(updatedHistory));
                return updatedHistory;
            });

            console.log("History record saved successfully!");
        } catch (error) {
            console.error("Error saving history record:",
error);
        }
    };
```

Цей блок коду призначений для збереження нових записів історії в локальному сховищі через AsyncStorage. Якщо дані у graphData відсутні, функція нічого не робить. В іншому випадку створюється новий запис, який включає в себе дані з graphData і мітку часу, отриману через Date.now(). Цей запис додається до існуючої історії. Дані оновлюються в локальному сховищі за допомогою AsyncStorage.setItem("history", JSON.stringify(updatedHistory)). Якщо під час збереження виникає помилка, вона логгіться у консолі, що дозволяє розробнику відслідковувати проблему.

Лістинг 3.8 – обробка дії початку чи зупинки руху

```
const handleStartRunning = async () => {
        if (isRunning) {
            setIsRunning(false);
            locationSubRef.current.remove();
            locationSubRef.current = null;

            clearInterval(pulseSubRef.current);
            pulseSubRef.current = null;

            clearInterval(timerRef.current);
            timerRef.current = null;

            saveHistoryRecord().then();
            return;
        }

        setIsRunning(true);
        setTime(0);
        setDistance(0);
        setSpeed(0);
        setBpm(0);
        setGraphData([]);
```

```
        locationSubRef.current = await getLocation();
        pulseSubRef.current = getBpm();
        timerRef.current = setInterval(() => {
            setTime(prevTime => prevTime + 1);
        }, 1000);

    }
```

Цей код дозволяє керувати станом бігу. Якщо біг вже запущений (isRunning), то функція зупиняє всі потоки: зупиняє моніторинг геолокації (locationSubRef.current.remove()), припиняє оновлення частоти серцевих скорочень (clearInterval(pulseSubRef.current)), зупиняє таймер (clearInterval(timerRef.current)) і зберігає запис історії. Якщо біг не запущений, функція налаштовує новий запуск бігу: обнуляє час, відстань, швидкість, частоту серцевих скорочень та очищує графічні дані. Далі отримується поточна геолокація через locationSubRef.current = await getLocation() та частота серцевих скорочень через pulseSubRef.current = getBpm(). Таймер для відображення часу бігу активується через timerRef.current = setInterval(() => { setTime(prevTime => prevTime + 1); }, 1000).

Функція handleStartRunning (Додаток А) дозволяє користувачеві запустити або зупинити біг. Якщо біг вже запущений (isRunning), то вона зупиняє всі процеси: зупиняє моніторинг геолокації (locationSubRef.current.remove()), зупиняє відстеження частоти серцевих скорочень (clearInterval(pulseSubRef.current)), зупиняє таймер (clearInterval(timerRef.current)) і зберігає запис історії через saveHistoryRecord(). Якщо біг ще не розпочато, функція ініціалізує новий цикл бігу: обнуляє час, відстань, швидкість, частоту серцевих скорочень і очищує графічні дані. Потім активується моніторинг геолокації через locationSubRef.current = await getLocation(), відстеження частоти серцевих скорочень через pulseSubRef.current = getBpm() та таймер для відстеження часу бігу через timerRef.current = setInterval(() => { setTime(prevTime => prevTime + 1); }, 1000).

Функція showDataChart змінює режим відображення графіка залежно від поточного стану, переключаючи між показниками швидкості та пульсу.

Функція getHistory() викликається через useFocusEffect для завантаження історії бігу кожного разу, коли користувач повертається на домашню сторінку.

Функція getModel() викликається через useEffect, щоб завантажити модель прогнозування та встановити її в стан.

Функція useEffect обробляє оновлення графічних даних і надає рекомендації на основі прогнозованих значень частоти серцевих скорочень, швидкості і дистанції.

Функція useEffect очищує всі активні моніторинги і таймери при виході з компоненту.

Кінцевий результат відображається через JSX, включаючи компонент SafeAreaView, налаштовану шапку, статистичні карти, графіки, рекомендації та кнопку для запуску/зупинки бігу. Код інтегрує MaterialCommunityIcons, DataChart, TrainingHistory, StatsCardsList, ActionButton і router.push("/settings") для забезпечення всіх необхідних функцій на домашній сторінці.

Цей компонент забезпечує інтерактивний і інформативний інтерфейс для відстеження, управління і візуалізації даних під час тренування бігу.

Лістинг 3.9 – нормалізація даних

```
import * as tf from '@tensorflow/tfjs';
import "@tensorflow/tfjs-react-native";
import {dataset} from "./dataset.js";

function normalizeTensor(tensor) {
    const min = tensor.min(); // Минимальное значение
    const max = tensor.max(); // Максимальное значение
    return tensor.sub(min).div(max.sub(min)); // (value - min) /
(max - min)
```

Цей блок коду імпортує TensorFlow для роботи з моделями машинного навчання в середовищі React Native. Ось детальний опис без структурування на пункти:

Цей імпорт містить TensorFlow для React Native (@tensorflow/tfjs-react-native) і потім визначає функцію normalizeTensor, яка приймає тензор як аргумент. Функція визначає мінімальне (min) і максимальне (max) значення тензора, а потім нормалізує значення тензора за допомогою формули (value - min) / (max - min). Це дозволяє звести значення тензора до діапазону від 0 до 1, що часто використовується для підвищення ефективності навчання нейронних мереж.

Після цього нормалізований тензор можна використовувати для навчання моделі або передбачення, підготовленим до роботи з TensorFlow у вашій React Native середовищі.

Лістинг 3.10 – створення моделі та підключення датасету

```
function createModel() {
    const model = tf.sequential();
    model.add(tf.layers.dense({ units: 64, activation: 'relu',
inputShape: [3] }));
    model.add(tf.layers.dense({ units: 32, activation: 'relu'
}));
    model.add(tf.layers.dense({ units: 16, activation: 'relu'
}));
    model.add(tf.layers.dense({ units: 3, activation: 'softmax'
}));
    model.summary();
    return model;
}

function loadAndNormalizeDataset(dataset) {

    const inputs = dataset.map(item => [item.heart_rate,
item.speed, item.distance]);
    console.log(`inputs length: ${inputs.length}`)

    const inputTensor = tf.tensor2d(inputs);
    const normalizedInputTensor = normalizeTensor(inputTensor);
    normalizedInputTensor.print();
    const outputs = dataset.map(item => item.recommendation);
    const outputTensor = tf.tensor1d(outputs, "int32");
```

```
    const normalizedOutputTensor = tf.oneHot(outputTensor, 3);

    normalizedOutputTensor.print();

    return {
        input_tensor: normalizedInputTensor,
        output_tensor: normalizedOutputTensor,
    };
}
```

Цей блок коду визначає функції для створення моделі нейронної мережі та обробки даних для навчання моделі. Ось детальний опис без структурування на пункти:

Функція createModel створює нейронну мережу з використанням TensorFlow. Спочатку створюється порожня послідовна модель (tf.sequential()), яка дозволяє додавати шар за шаром. Спочатку додається шар з 64 нейронами, з функцією активації relu і з входом розмірністю 3, яка відповідає кількості параметрів у вашому наборі даних (серцевий ритм, швидкість, дистанція). Потім додаються ще два шари з 32 і 16 нейронами відповідно, також з функцією активації relu. Останній шар має 3 нейрони і функцію активації softmax, що дозволяє класифікувати в один з трьох варіантів рекомендацій. Після створення моделі, використовується метод .summary() для виведення інформації про структуру моделі у консоль, що дозволяє побачити кількість шарів, нейронів, а також розміри входу та виходу. Функція повертає створену модель.

Функція loadAndNormalizeDataset приймає набір даних і виконує наступні кроки:

– виводить вхідні дані для моделі як масив [серцевий ритм, швидкість, дистанція];

– перетворює їх у тензор tf.tensor2d для обробки з TensorFlow;

– нормалізує цей тензор за допомогою функції normalizeTensor, щоб усі значення в діапазоні 0 до 1;

– виводить результат на консоль для перевірки;

– виводить вихідні дані (рекомендації) як тензор tf.tensor1d з використанням int32 типу, і потім перетворює їх у oneHot формат для категоризації у три класи;

– нормалізує вихідний тензор, використовуючи tf.oneHot(outputTensor, 3), щоб отримати один гарячий тензор, який буде використовуватися для навчання;

– виводить нормалізований тензор на консоль для перевірки;

– повертає об’єкт з двома тензорами: input_tensor та output_tensor.

Цей код дозволяє підготувати набір даних для навчання моделі з використанням TensorFlow, нормалізуючи вхідні дані та перетворюючи вихідні дані у формат, що підходить для одно гарячого кодування.

Лістинг 3.10 – навчання моделі

```
async function trainModel(model, inputs, labels) {
    model.compile({
        optimizer: tf.train.adam(),
        loss: 'categoricalCrossentropy',
        metrics: ['accuracy'],
    });
    console.log("compile done")
    await model.fit(inputs, labels, {
        epochs: 128, // Adjust as needed for better results
        shuffle: true,
        callbacks: {
            onEpochBegin: (epoch, logs) => {
                console.log(`Epoch ${epoch}:`, logs);

            },
            onEpochEnd: (epoch, logs) => {
                console.log(`Epoch ${epoch}:`, logs);
            },
            onTrainBegin: (logs) => {
                console.log(`Training started...`);
            },
            onTrainEnd: (logs) => {
                console.log(`Training finished.`);
            },
        },
    }).catch((error) => {
        console.error("Error during model training:", error);
    });
```

```
    console.log("fit done")

    inputs.dispose();
    labels.dispose();
    console.log('Model training complete!');
}
```

Цей блок коду визначає функцію асинхронного навчання моделі з використанням TensorFlow. Ось детальний опис без структурування на пункти:

Функція trainModel приймає модель TensorFlow, вхідні дані (inputs) і мітки (labels). Спочатку модель компілюється з використанням оптимізатора Adam, функції втрат categoricalCrossentropy та метрики accuracy, щоб оцінити правильність прогнозів під час навчання. Після компіляції виводиться повідомлення, що процес компіляції завершений.

Потім викликається метод fit, який починає процес навчання. Вхідні дані та мітки передаються в якості параметрів. Кількість епох (epochs) можна регулювати в залежності від того, які результати ви хочете досягти. У цьому випадку, використовується 128 епох, але ви можете змінювати їх кількість.

Використовуються колбек-функції для виведення інформації під час початку та кінця навчання, початку та кінця кожної епохи. Ці колбек-функції дозволяють бачити деталі під час тренування, наприклад, на початку кожної епохи та в кінці кожної епохи.

Якщо під час навчання виникає помилка, вона перехоплюється через catch і виводиться повідомлення про помилку.

Після закінчення навчання тензори для вхідних даних і міток очищуються (dispose()), щоб звільнити ресурси.

Виводиться повідомлення про завершення навчання моделі.

Цей код дозволяє реалізувати процес навчання моделі, контролюючи основні етапи, таких як початок та кінець епохи, початок та кінець тренування, а також обробку помилок.

Лістинг 3.11 – функція прогнозу

```
export function predict(model, heart_rate, speed, distance) {
    return tf.tidy(() => {
        console.log(`predict start h:${heart_rate} s:${speed}
d:${distance}`)

        const input = tf.tensor2d([[heart_rate, speed,
distance]]);
        const normalizedSampleInput = normalizeTensor(input);
        const prediction = model.predict(normalizedSampleInput);
        const predictedClass = prediction.argMax(-
1).dataSync()[0];
        console.log("predict end ")
        console.log(predictedClass);
        return predictedClass;
    })
}
```

Цей блок коду визначає функцію для здійснення прогнозу з використанням вже натренованої моделі в TensorFlow.

Функція predict приймає модель, а також три параметри: heart_rate (серцевий ритм), speed (швидкість) і distance (дистанція). Спочатку за допомогою tf.tidy() створюється середовище для автоматичного очищення ресурсів TensorFlow після виконання всіх операцій всередині цієї функції. Це дозволяє уникнути витрат пам'яті під час виконання прогнозу.

Далі, з отриманих параметрів створюється тензор у вигляді двовимірного масиву (де кожен рядок містить значення серцевого ритму, швидкості та дистанції). Тензор нормалізується за допомогою функції normalizeTensor, яка приводить значення до діапазону від 0 до 1, щоб модель могла працювати з цими значеннями ефективно.

Після цього здійснюється прогноз за допомогою методу model.predict(), де передається нормалізований тензор як вхід для моделі. Результат прогнозу містить ймовірності для кожного класу. Оскільки модель має на виході три класи (представлені через softmax), використовуємо argMax(-1), щоб знайти індекс класу з найвищою ймовірністю (це клас, якому модель надає найбільшу ймовірність).

Метод dataSync() повертає значення передбаченого класу, яке потім виводиться в консоль для перевірки. Функція повертає індекс передбаченого класу.

Цей код дозволяє отримувати прогноз для конкретного зразка даних (серцевий ритм, швидкість, дистанція), використовуючи попередньо натреновану модель машинного навчання.

Лістинг 3.12 – Головна функція запуску моделі

```
export async function getModel() {
    await tf.ready().then(() => {
        console.log("TensorflowJS is ready");
    });
    const model = createModel();
    const { input_tensor, output_tensor } =
loadAndNormalizeDataset(dataset);
    console.log("start training")
    await trainModel(model, input_tensor, output_tensor);
    return model;
```

Цей блок коду визначає функцію getModel, яка готує та тренує модель машинного навчання за допомогою TensorFlow.

Функція getModel є асинхронною та починається з того, що чекає на готовність TensorFlow до роботи за допомогою методу tf.ready(). Після того, як TensorFlow підготувався, виводиться повідомлення в консоль, що TensorFlowJS готовий до використання. Потім створюється модель за допомогою функції createModel, яка була визначена раніше, і отримуються оброблені й нормалізовані дані для навчання за допомогою функції loadAndNormalizeDataset, передаючи їй набір даних dataset.

Далі виводиться повідомлення в консоль, що навчання починається. Викликається функція trainModel, де передаються створена модель, нормалізовані вхідні дані та мітки. Оскільки trainModel є асинхронною функцією, вона виконується з використанням await, чекаючи завершення навчання моделі.

Після того як навчання завершиться, функція повертає готову модель, яка тепер може бути використана для здійснення прогнозів.

Цей код забезпечує створення, тренування і повернення моделі машинного навчання з використанням TensorFlowJS у середовищі JavaScript (React Native).

3.4 Створення скетчу для ESP32

Лістинг 3.13 – Підлючення бібліотек

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"

// const char *ssid = "TP-Link_3F88";
// const char *password = "83825682";
const char *ssid = "esp32";
const char *password = "12345678";
AsyncWebServer server(80);

MAX30105 particleSensor;
long lastBeat = 0;
float beatsPerMinute;
```

Цей код ініціалізує базові компоненти для роботи ESP32 з Wi-Fi, веб-сервером, а також датчиком пульсу MAX30105.

У коді підключаються необхідні бібліотеки: WiFi.h для роботи з мережею Wi-Fi, AsyncTCP.h і ESPAsyncWebServer.h для налаштування асинхронного веб-сервера, Wire.h для роботи з протоколом I2C, а також MAX30105.h і heartRate.h для використання функціоналу сенсора MAX30105 і обробки даних пульсу.

Оголошуються змінні для налаштування Wi-Fi: ssid і password. Ці змінні містять ім'я мережі та пароль для створення точки доступу Wi-Fi, де ім'я мережі задано як "esp32", а пароль як "12345678".

Далі створюється асинхронний веб-сервер, використовуючи об'єкт AsyncWebServer на порту 80. Також ініціалізується об'єкт MAX30105 particleSensor для роботи з датчиком пульсу.

Змінна lastBeat використовується для зберігання часу останнього виявленого серцевого удару, а змінна beatsPerMinute призначена для збереження обчисленого значення частоти серцевих скорочень. Ці змінні будуть використовуватися для обробки та передачі даних пульсу.

Лістинг 3.14 – функція обробки HTTP-запиту до сервера

```
void mainPage(AsyncWebServerRequest *request) {
    Serial.print("send Main page\n");
    request->send(200,                          "application/json",
"{\"message\":\"Welcome\"}");
}
```

Функція приймає параметр AsyncWebServerRequest *request, який містить інформацію про HTTP-запит, надісланий клієнтом. При виклику функції в консоль за допомогою Serial.print виводиться повідомлення "send Main page\n", щоб розробник міг відстежувати, коли головна сторінка була відправлена.

Метод request->send використовується для відправки відповіді клієнту. У цьому випадку клієнт отримує HTTP-статус 200 (успішна операція), тип вмісту відповіді application/json і сам JSON-об'єкт у вигляді рядка {"message":"Welcome"}. Це повідомлення вказує, що клієнт успішно підключився до сервера і отримав привітальне повідомлення.

Лістинг 3.15 – функція обробки HTTP-запиту на отримання BPM

```
    void getBPM(AsyncWebServerRequest *request) {
      Serial.print("send BPM\n");
      request->send(200, "application/json", "{\"bpm\": " +
    String(beatsPerMinute) + "}");

}
```

Функція приймає параметр AsyncWebServerRequest *request, що представляє запит, надісланий клієнтом до сервера. При виклику функції в послідовний порт (Serial) виводиться повідомлення "send BPM\n" для відстеження моменту, коли сервер відповідає запитом частоти пульсу.

Далі за допомогою методу request->send клієнту відправляється HTTP-відповідь. Код відповіді — 200, що означає успішне виконання запиту, тип вмісту відповіді — "application/json", а сама відповідь — JSON-рядок у вигляді {"bpm": значення}, де значення — це поточне значення змінної beatsPerMinute. Значення beatsPerMinute перетворюється в рядок за допомогою функції String() для вставки у JSON.

Лістинг 3.16 – головна функція

```
void setup()
{
  Serial.begin(115200);
  Serial.println("Initializing...");

  WiFi.softAP(ssid, password);

  server.on("/", HTTP_GET, mainPage);
  server.on("/get-bpm", HTTP_GET, getBPM);
  server.begin();
  delay(5000);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  Wire.begin(5, 4);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST))  {
    Serial.println("MAX30105 was not found. Please check
wiring/power. ");
    while (1);
  }
  Serial.println("Place your index finger on the sensor with
steady pressure.");

particleSensor.setup();
particleSensor.setPulseAmplitudeRed(0x0A);
particleSensor.setPulseAmplitudeGreen(0);
}
```

Цей код визначає функцію setup, яка виконується одноразово під час ініціалізації мікроконтролера. Вона налаштовує серійний зв'язок, Wi-Fi, веб-сервер і датчик пульсу MAX30105. Ось детальний опис без структурованих пунктів:

Перш за все, викликається Serial.begin(115200), щоб налаштувати серійну передачу даних із швидкістю 115200 бод, після чого виводиться повідомлення "Initializing..." для інформування про початок ініціалізації. Потім налаштовується Wi-Fi, створюється точка доступу за допомогою WiFi.softAP, використовуючи ім'я мережі та пароль, збережені в змінних ssid і password.

Далі налаштовується веб-сервер. Метод server.on додає маршрути для обробки HTTP-запитів. Для запиту до кореневого шляху / сервер викликає функцію mainPage, а для шляху /get-bpm — функцію getBPM. Після цього сервер запускається викликом server.begin, і виконується затримка в 5 секунд для стабілізації. Потім у серійну консоль виводиться локальна IP-адреса за допомогою WiFi.localIP.

Ініціалізується з'єднання I2C через Wire.begin(5, 4) з використанням пінів 5 і 4 для передачі і прийому даних. Далі виконується перевірка, чи вдалося успішно підключити датчик пульсу MAX30105. Якщо датчик не виявлено, виводиться відповідне повідомлення, і програма застряє в нескінченному циклі while (1).

Після успішного підключення датчика виводиться повідомлення, що потрібно прикласти палець до сенсора. Потім датчик налаштовується за допомогою методу particleSensor.setup. Червоний світлодіод датчика вмикається на низький рівень яскравості через particleSensor.setPulseAmplitudeRed(0x0A), а зелений світлодіод вимикається через particleSensor.setPulseAmplitudeGreen(0). Ці дії завершують налаштування датчика для роботи.

Лістинг 3.17 – Алгоритм обчислення BPM

```
void countBPM() {
  long irValue = particleSensor.getIR();

  if(irValue > 50000){
    if (checkForBeat(irValue) == true){
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    Serial.print("IR=");
    Serial.print(irValue);
    Serial.print(", BPM=");
    Serial.print(beatsPerMinute);
    Serial.println();
    }
  }
   else {
        beatsPerMinute = 0;
  }

}
```

Функція починається зі зчитування інфрачервоного значення, яке генерує датчик, за допомогою particleSensor.getIR(), результат зберігається в змінну irValue. Потім перевіряється, чи перевищує це значення поріг у 50000. Це означає, що палець знаходиться на датчику, і дані є релевантними. Якщо умова виконується, викликається функція checkForBeat(irValue), яка визначає, чи був виявлений серцевий удар. У разі підтвердження виконується обчислення часу між останніми двома ударами серця, використовуючи різницю між поточним часом millis() і часом останнього удару lastBeat. Поточний час зберігається в lastBeat.

Обчислюється значення BPM за формулою 60 / (delta / 1000.0), де delta — це час між ударами в мілісекундах, перетворений у секунди. Отримане значення зберігається у змінній beatsPerMinute.

Для відладки в серійну консоль виводяться значення інфрачервоного сигналу (irValue) і обчисленого BPM. Якщо ж значення irValue не перевищує порогу, це вказує на відсутність пальця на датчику. У такому випадку змінній

beatsPerMinute присвоюється значення 0, що сигналізує про відсутність даних.

Ця функція використовується для постійного оновлення значення BPM, яке можна відобразити користувачеві або передати іншим компонентам системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ

У запропонованому проєкті було реалізовано інтелектуальну систему моніторингу та аналізу даних для підтримки реабілітації спортсменів, яку наведено на рисунку 4.1:

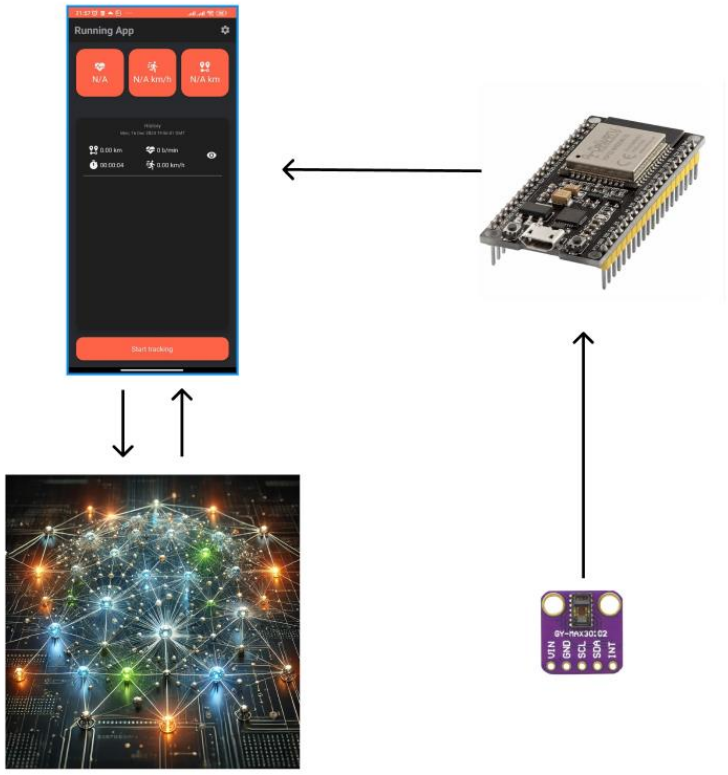

Рисунок 4.1 – Візуалізація роботи проекту

Для створення нейронної мережі, було обрана бібліотеку TensorFlow, через простоту підключення до ReactNative.

Основу мережі складає багатошаровий перцептрон, адаптований для аналізу фізіологічних і кінематичних показників, таких як пульс, швидкість та дистанція.

Дані, отримані з датчиків, перетворюються на тензори — багатовимірні масиви чисел, що виступають вхідними параметрами для нейромережі. У TensorFlow тензори використовуються для представлення та обробки цих

даних, забезпечуючи високу швидкість розрахунків. Нейронна мережа приймає ці тензори, проводить активацію через приховані шари і видає результати у вигляді ймовірності різних рекомендацій: знизити темп, підтримувати поточну швидкість або прискоритися.

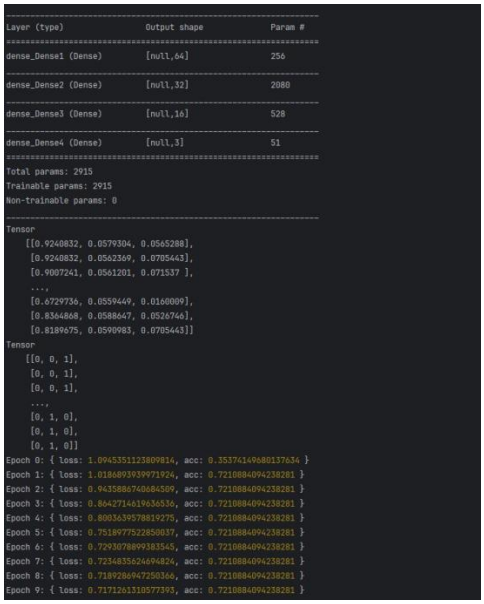

Рис 4.2 – Тренування моделі

Для підвищення точності передбачень була проведена попередня обробка даних, їх нормалізація та балансування набору даних під час навчання.

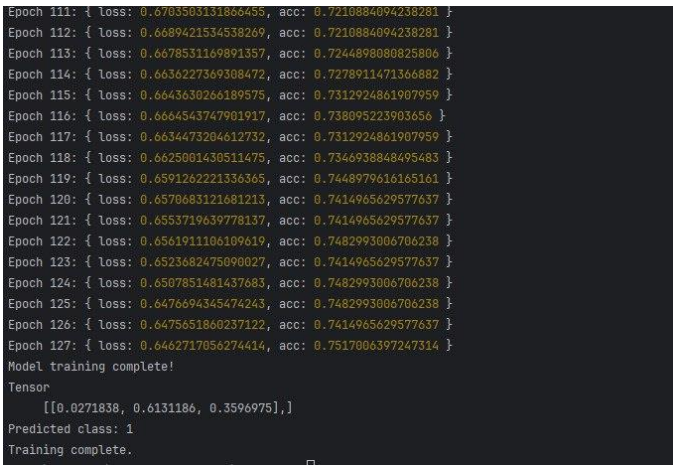

Рис 4.3 – Результат тренування моделі

Модель була попередньо навчена на зібраних даних, а потім оптимізована для роботи в реальному часі, щоб забезпечити високоточні рекомендації спортсменам, адаптовані до їхніх індивідуальних потреб і поточного стану.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи були детально розглянуті загальні поняття та підходи до реабілітації спортсменів. З’ясована роль та значення інтелектульних систем в спортивній медицині. Розглянуто існучуючі прилади та методи для реабілітації. Наведено основні фізіологічні показники, які будуть використовуватись в даному проекті.

Проведено аналіз-порівняння між платами розробки ESP32 та ESP8266. Сформований перелік переваг та недоліків цих мікропроцесорів та з’ясований найкращий для використання у цьому проекті.

Наведено перелік усіх комплектуючих для реалізації проекту. Розглянута детальна інформація щодо кожного модуля, надані повні технічні характеристики та схеми їх підключення.

Була створення нейрона мережа для надання рекомендацій на основі отриманних данних. Створено мобільний застосунок для обробки інформації та надання результату користувачу. Написано та відлагоджено код для отрмання частоти сердцевого скорочення та відправки його до застосунку. Для реалізації проекту були використано: плата розробника ESP32, смартфон на базі Android, пульсометр MAX30102.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цифровий спідометр для транспортних засобів на базі Arduino Uno: Кваліфікаційна робота бакалавра / Білич Ю.Є. – Харків: 2023. – 50 с., 13 іл.
2. Блум Джеремі Вивчаємо Arduino: інструменти та методи технічного чаклунства [Електронний ресурс] – Режим доступу: www / URL: https://www.rulit.me/books/izuchaem-arduino-instrumety-i-metody-tehnicheskogo-volshebstva/
3. Плата розробника ESP32 [Електронний ресурс] - Режим доступу: www / URL: https://arduino.ua/prod4846-keyestudio-esp32-io-shield-for-arduino-esp32-core-board
4. Датчик пульсу MAX30102 V2 [Електронний ресурс] - Режим доступу: www / URL: https://arduino.ua/prod2036-datchik-pylsy-max30102-v2
5. Фізична, реабілітаційна та спортивна медицина : Підручник для студентів і лікарів / За заг. ред. В.М.Сокрута. — Краматорськ: Каштан, 2019. — 480 с., 32 іл.