Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Wearable Health & Activity Tracker

Project Engineering

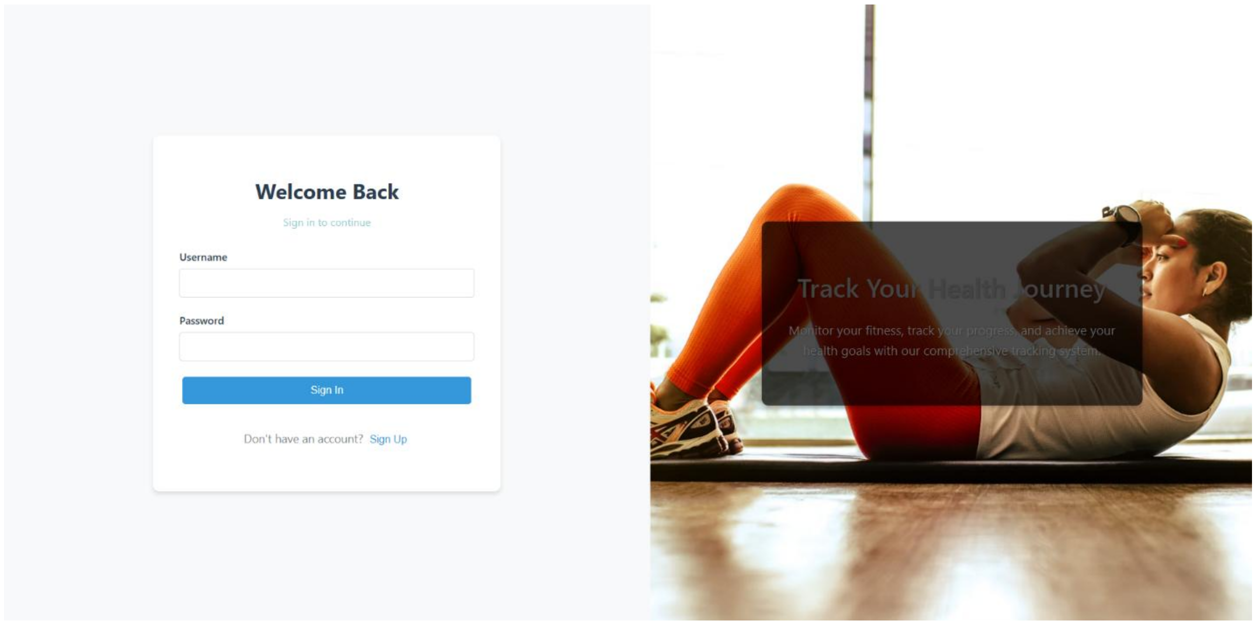
Year 4

Ross Armitage

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2024/2025



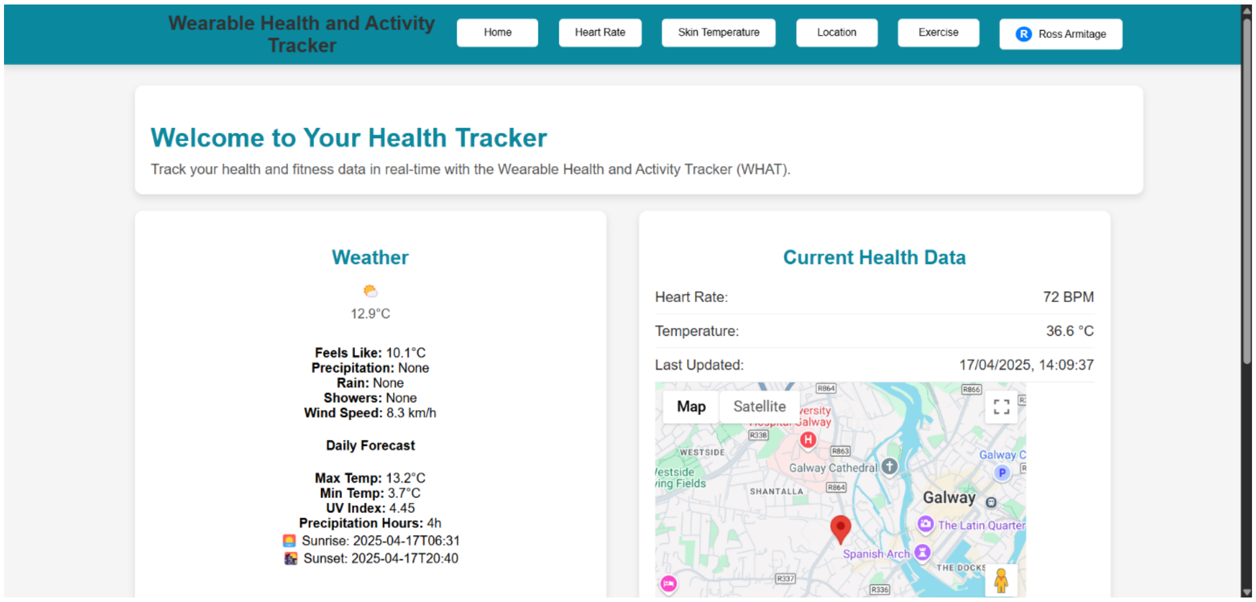
Welcome Back

Sign in to continue

Username

Password

Sign In
Don't have an account? Sign Up
Track Your Health Journey
Monitor your fitness, track your progress, and achieve your health goals with our comprehensive tracking system.



Welcome to Your Health Tracker

Track your health and fitness data in real-time with the Wearable Health and Activity Tracker (WHAT).

Weather

12.9°C

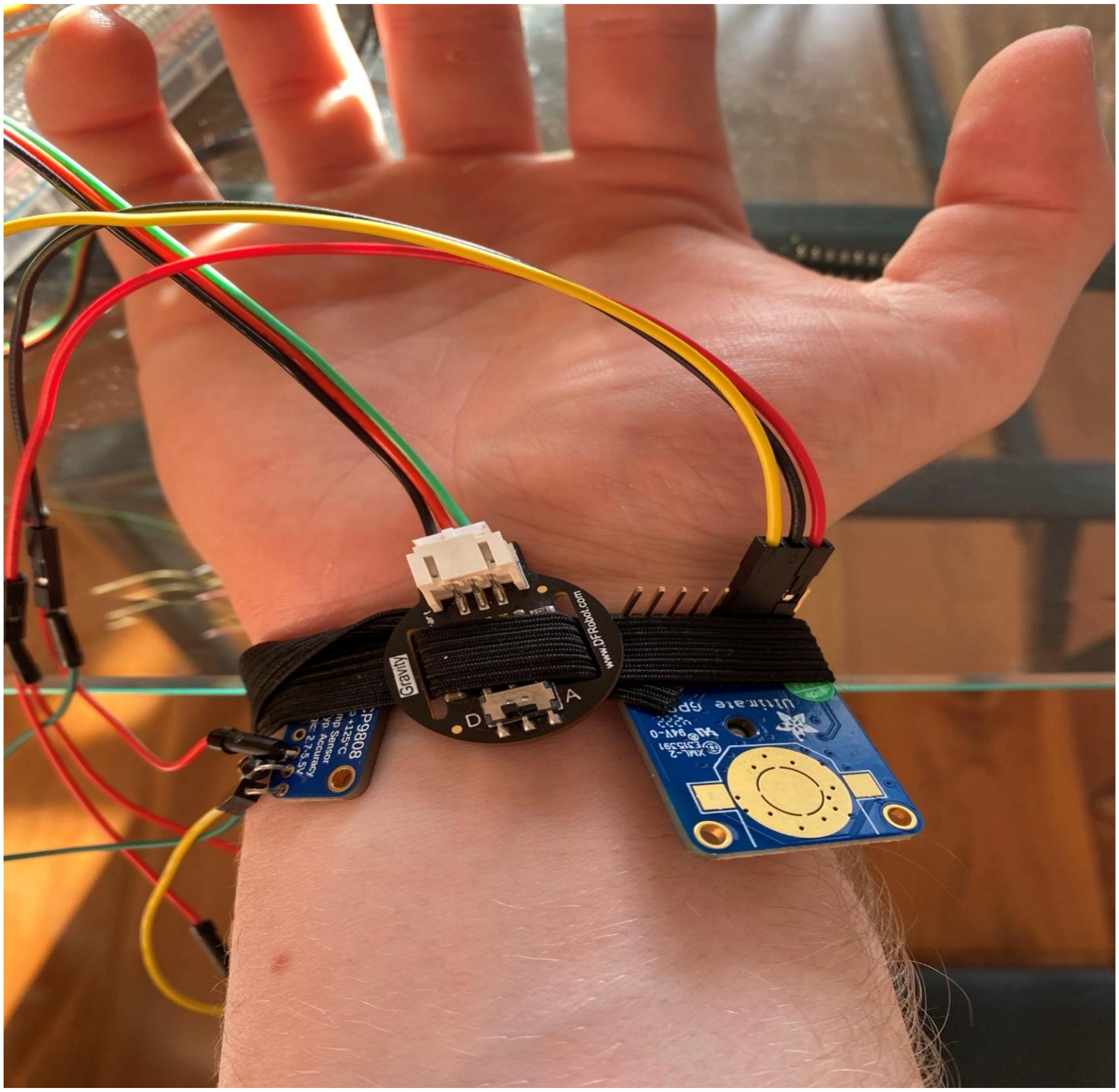
Feels Like: 10.1°C
Precipitation: None
Rain: None
Showers: None
Wind Speed: 8.3 km/h

Daily Forecast

Max Temp: 13.2°C
Min Temp: 3.7°C
UV Index: 4.45
Precipitation Hours: 4h
Sunrise: 2025-04-17T06:31
Sunset: 2025-04-17T20:40

Current Health Data

Heart Rate:	72 BPM
Temperature:	36.6 °C
Last Updated:	17/04/2025, 14:09:37


Gravity
www.DFRobot.com
D
A
Ultimate GPS

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Ross Armitage

Acknowledgements

I would like to express my gratitude to all the lecturers who have shared their knowledge and provided guidance throughout the development of this project. Your support has been very important in helping me develop the skills necessary to complete this project.

A special thanks to my project supervisor Ben Kinsella, for his continuous guidance and encouragement throughout the development of this project. His expertise and support have been invaluable.

Finally, I would like to extend my heartfelt thanks to all my classmates for their collaboration, encouragement and the positive learning environment they helped create. Your support made this journey rewarding.

Table of Contents

1 Summary

The goal of this project was to create a wearable device that monitors key health metrics and location data in real time, helping users track their physical wellbeing during daily life and activities. The project aimed to create a reliable, portable and connected health tracking system that integrates seamlessly with a web interface.

The scope of the project included the development of both hardware and software components, a wearable device built around the ESP32 microcontroller and a web application to display the collected data. The device measures heart rate, body temperature, and GPS location, transmitting this data wirelessly to the cloud.

Key features of the project include:

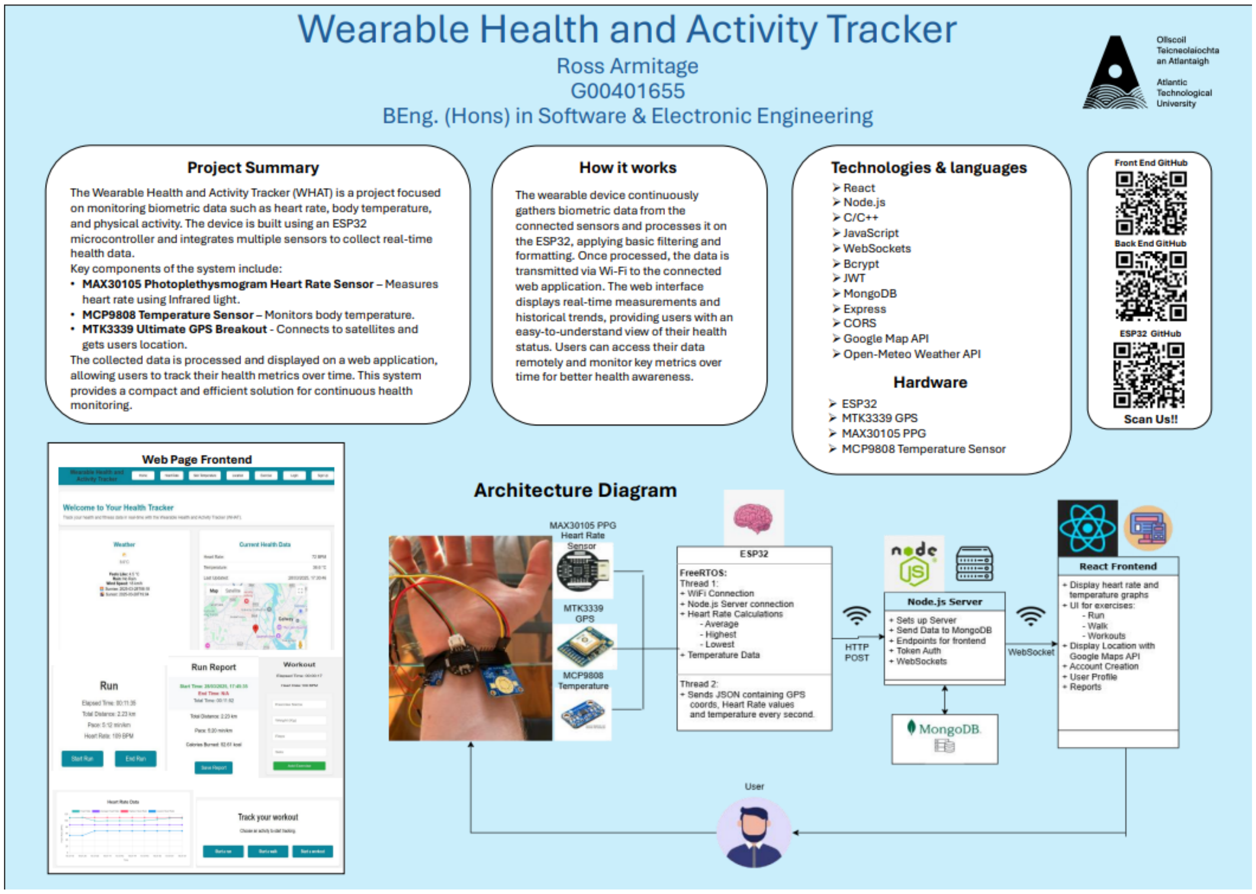
- Real-time monitoring of heart rate using the **MAX30105** PPG sensor
- Accurate body temperature tracking using the **MCP9808** sensor
- Live **MTK3339** GPS tracking for logging location
- **Run/Walk/Workout** tracking features
- Wireless communication over Wi-Fi for seamless data transfer
- A responsive web app for data visualization
- Weather information based on location
- Secure accounts with activity history

The approach involved writing custom firmware in C/C++ for the ESP32 using the Arduino IDE. Sensor data is collected, formatted, and transmitted over Wi-Fi using WebSockets. The web interface was built with React to display the data in a user-friendly way.

The project demonstrates continuous data collection and live streaming to the web app. All sensors were integrated effectively, and the device was tested under various conditions to ensure reliability.

In conclusion, the project achieved its goal of creating a functional, internet-connected wearable capable of health and location tracking, providing a foundation for further development in mobile health monitoring systems.

2 Poster



Wearable Health and Activity Tracker

Ross Armitage

G00401655

BEng. (Hons) in Software & Electronic Engineering

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Project Summary

The Wearable Health and Activity Tracker (WHAT) is a project focused on monitoring biometric data such as heart rate, body temperature, and physical activity. The device is built using an ESP32 microcontroller and integrates multiple sensors to collect real-time health data.

Key components of the system include:

- **MAX30105 Photoplethysmogram Heart Rate Sensor** – Measures heart rate using Infrared light.
- **MCP9808 Temperature Sensor** – Monitors body temperature.
- **MTK3339 Ultimate GPS Breakout** - Connects to satellites and gets users location.

The collected data is processed and displayed on a web application, allowing users to track their health metrics over time. This system provides a compact and efficient solution for continuous health monitoring.

How it works

The wearable device continuously gathers biometric data from the connected sensors and processes it on the ESP32, applying basic filtering and formatting. Once processed, the data is transmitted via Wi-Fi to the connected web application. The web interface displays real-time measurements and historical trends, providing users with an easy-to-understand view of their health status. Users can access their data remotely and monitor key metrics over time for better health awareness.

Technologies & languages

- React
- Node.js
- C/C++
- JavaScript
- WebSockets
- Bcrypt
- JWT
- MongoDB
- Express
- CORS
- Google Map API
- Open-Meteo Weather API

Hardware

- ESP32
- MTK3339 GPS
- MAX30105 PPG
- MCP9808 Temperature Sensor

Front End GitHub

Back End GitHub

ESP32 GitHub

Scan Us!!

Web Page Frontend

Architecture Diagram

Figure 2-1 Poster

3 Introduction

The goal of this project is to develop a wearable health and activity tracker that monitors key biometric data and GPS location in real time. The system aims to provide users with insights into their physical wellbeing through continuous tracking of heart rate, body temperature, and movement, displayed via a responsive web interface.

The motivation for this project stems from a personal interest in both fitness and biometric data. As someone who regularly goes to the gym and runs, I’ve always been curious about how the body responds to different activities and how data can be used to optimize performance and recovery. This interest in health tracking and technology inspired the development of the wearable health & activity tracker tailored to active lifestyles.

4 Background

4.1 What is MongoDB?

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format (called BSON). Instead of tables like SQL, it uses collections and documents, which makes it good for storing data that changes structure often.

Great for fast development, flexible schemas, and handling lots of data. [7]

4.2 How Node.js works?

Node.js [16] is a **runtime environment** that lets you run **JavaScript on the server** (not just in the browser). It uses an **event-driven**, **non-blocking I/O** model, which makes it **fast and efficient**, especially for real-time apps like chat or live data. [7]

4.3 What is React?

React is a **JavaScript library** for building **user interfaces** (UIs), especially **single-page applications**. It lets you create **components,** reusable bits of UI and update them efficiently when data changes. [7]

4.4 PPG Heart rate sensor

A PPG (Photoplethysmography) sensor uses **light to measure your pulse**. It shines light into your skin and detects changes in blood flow, which corresponds to your **heartbeats.** [7]

4.5 What is AWS?

AWS (Amazon Web Services) is a cloud platform by Amazon. It offers tools like servers, storage, databases, and more, all accessible online. You can use AWS to host websites, store files, or run machine learning models. [7]

4.6 What is an ESP32?

The **ESP32** is a powerful, low-cost microcontroller with built-in **Wi-Fi and Bluetooth**. It’s commonly used in IoT (Internet of Things) projects to collect data from sensors and send it to the internet. [7]

4.7 What is WebSockets?

WebSockets allow for **real-time communication** between a client (like your browser) and a server. Unlike HTTP, which is one-way and short-lived, WebSocket connections stay open and let both sides send messages at any time. [7]

4.8 Open-Meteo API

The Open-Meteo [20] API is a free, fast, and open-source weather API that provides accurate weather forecasts. It supports global coverage and allows users to retrieve hourly, daily, and real-time weather data, including temperature, precipitation, wind, and more. Designed for developers, it offers a simple RESTful interface and does not require authentication. [7]

5 Project Architecture

This diagram illustrates the full architecture of the WHAT project, breaking down its hardware, software, and data flow components from the wearable device to the frontend user interface.

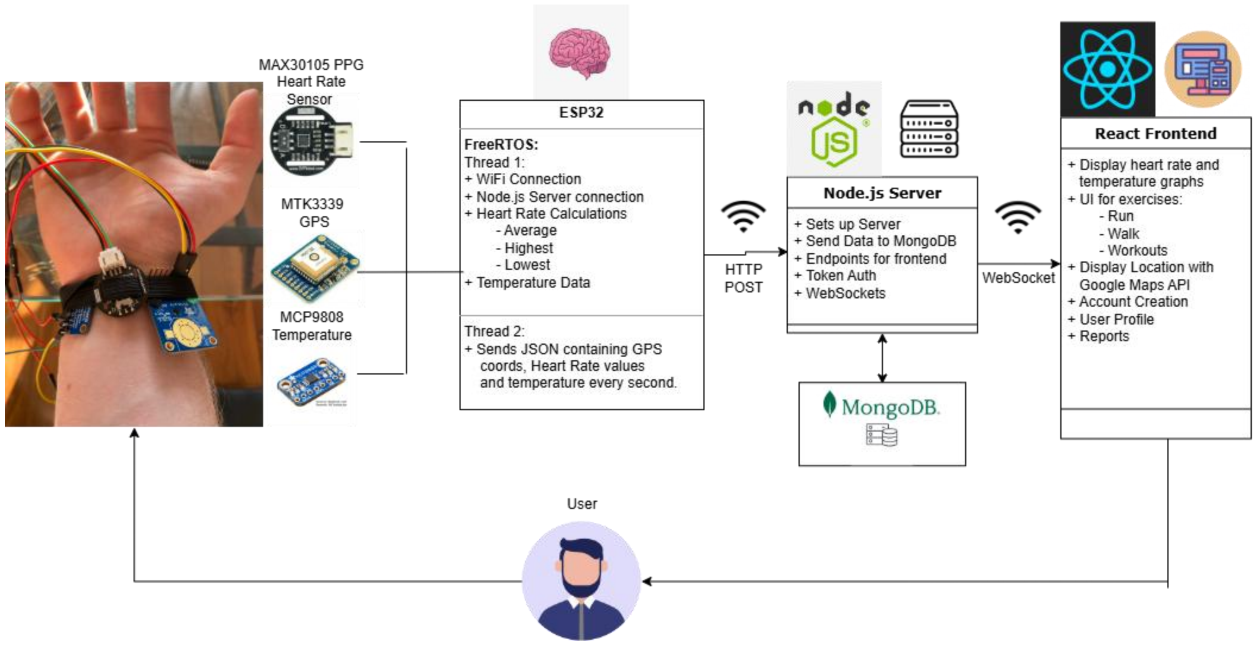


Figure 5-1 Architecture Diagram

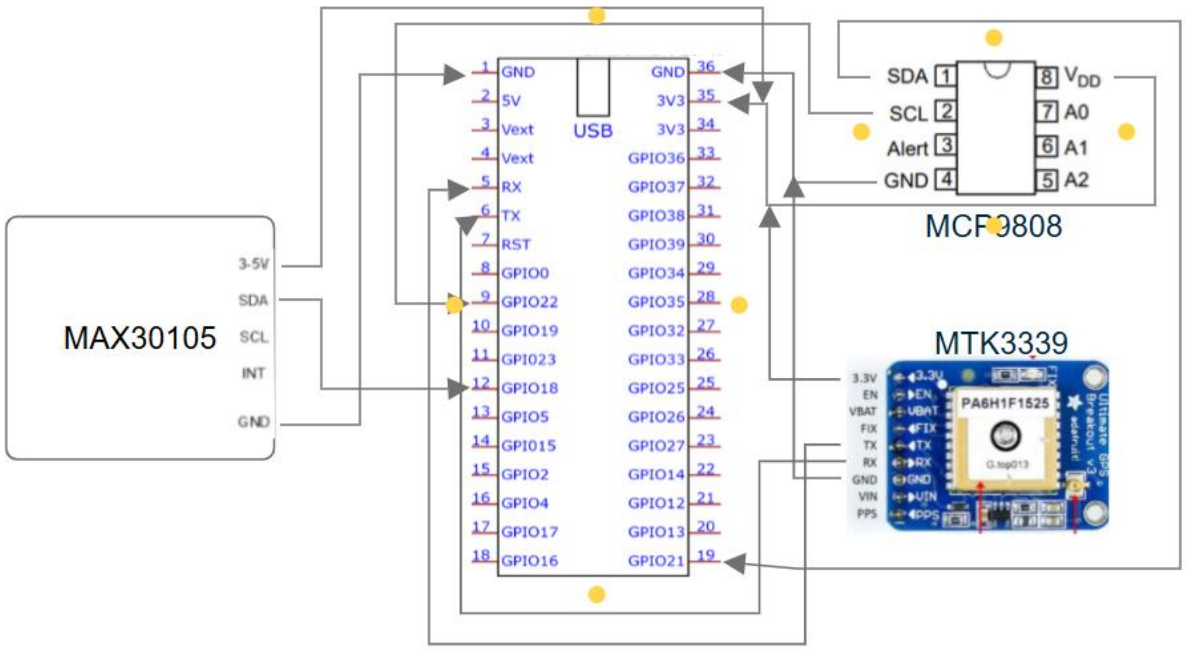


Figure 5-2 Schematic Diagram

5.1 Hardware (Left Side)

The wearable device is built around the **ESP32** microcontroller and includes three key sensors:

- **MAX30105**: Used for heart rate monitoring.
- **MCP9808**: Used for accurate body temperature measurement.
- **MTK3339 GPS Module**: Used to acquire the user's live location coordinates.[4]

These sensors are physically connected to the ESP32, worn on the user's wrist as shown in the image.

5.2 Firmware (ESP32 with FreeRTOS - Center):

The ESP32 runs **FreeRTOS** with two main threads:

- **Thread 1** handles:
    - Wi-Fi connection
    - Communication with the Node.js backend server
    - Heart rate calculations (average, highest, lowest)
    - Reading and processing temperature data
- **Thread 2**:
    - Packages data into JSON format every second, including:
        - GPS coordinates
        - Heart rate metrics
        - Temperature
    - Sends this JSON data to the backend via Wi-Fi

5.3 Backend (Node.js - Middle Right):

The **Node.js server** is responsible for:

- Receiving JSON data from the ESP32
- Storing the data in **MongoDB**
- Providing API endpoints for the frontend to fetch data
- Handling user authentication via tokens
- Managing **WebSocket** communication for real-time updates

5.4 Database (MongoDB):

- Stores all the collected sensor and GPS data.
- Organizes data per user, activity, and time for easy retrieval.

5.5 Frontend Web App (React - Far Right):

Built with **React and Next.js**, the UI offers:

- Real-time heart rate and temperature graphs
- Exercise tracking modes (Run, Walk, Workouts)

- Google Maps API [10] integration to display location history
- Features like:
    - Account creation
    - User profiles
    - Health reports

6 Project Plan

6.1 Objectives

Build a Wearable Device Using ESP32:

Design and assemble a compact and lightweight wearable device based on the ESP32 microcontroller. The device interfaces with multiple sensors to gather biometric and location data in real time.

Monitor Heart Rate, Temperature and GPS location:

Integrate the MAX30105 PPG sensor for heart rate monitoring, the MCP9808 for body temperature monitoring and MTK3339 GPS for tracking of the user's location.

Develop a Full Stack solution:

Implement an end-to-end system combining embedded firmware, a backend server, and a frontend web interface. The embedded system handles data calculations and communication. The backend server (Node.js and MongoDB) manages data storage and processing. The frontend (React) allows users to interact with and visualise their data.

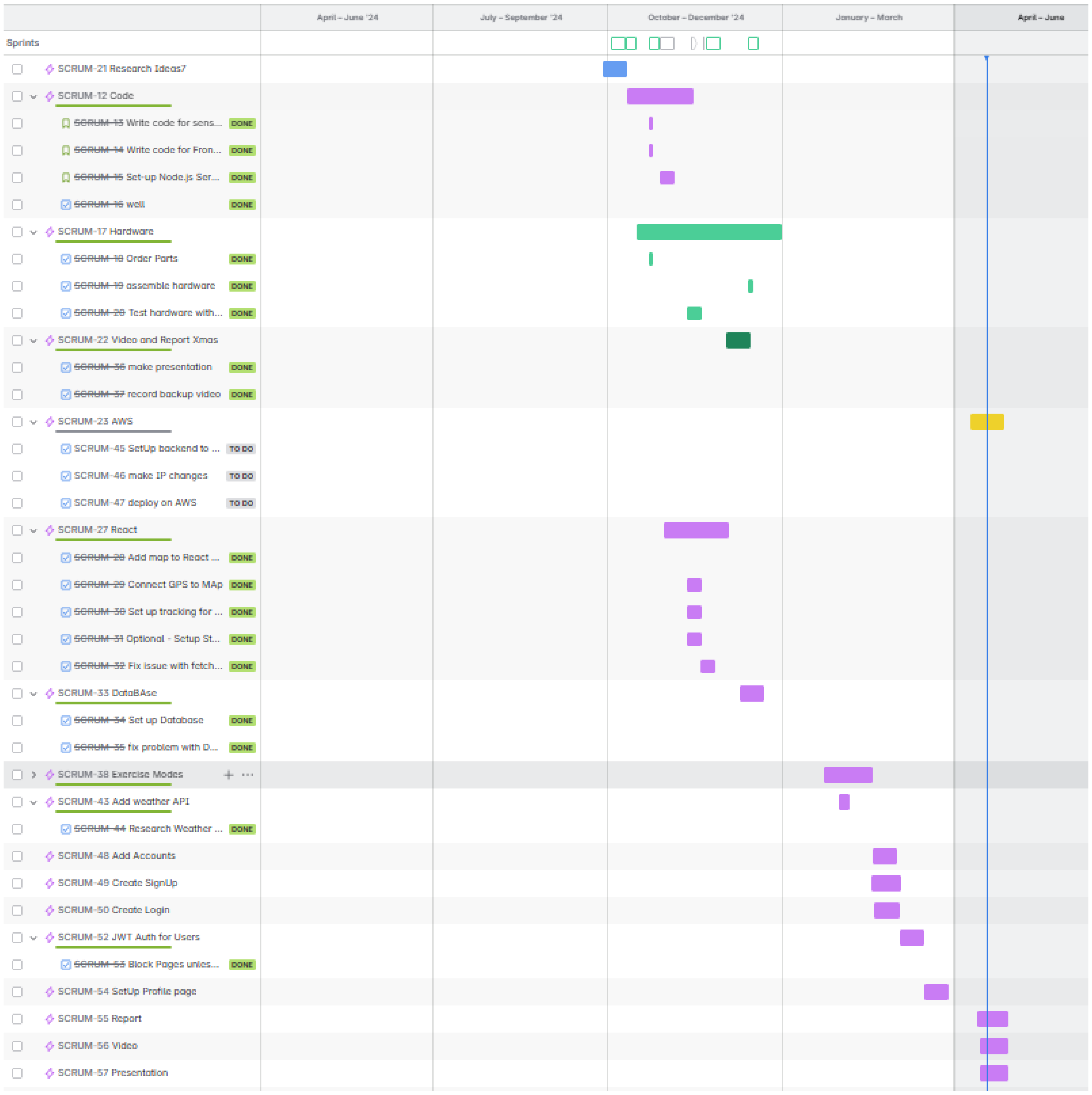
Enable Real-time data transfer and Visualization:

Use Wi-Fi and WebSocket protocols to ensure fast and reliable communication between the wearable device and the web app. Display live updates of biometric and GPS data through graphs and maps.

User Authentication and Data privacy:

Implement secure token-based authentication and user management features. Each user's data is isolated and protected, ensuring the personal health information remains private and accessible to only authorised users.

6.2 Project Phases & Timeline


April – June '24
July – September '24
October – December '24
January – March
April – June

Sprints

- SCRUM-21 Research Ideas7
- SCRUM-12 Code
  - ~~SCRUM-13~~ Write code for sens... DONE
  - ~~SCRUM-14~~ Write code for Fron... DONE
  - ~~SCRUM-15~~ Set-up Node.js Ser... DONE
  - ~~SCRUM-16~~ well DONE
- SCRUM-17 Hardware
  - ~~SCRUM-18~~ Order Parts DONE
  - ~~SCRUM-19~~ assemble hardware DONE
  - ~~SCRUM-20~~ Test hardware with... DONE
- SCRUM-22 Video and Report Xmas
  - ~~SCRUM-36~~ make presentation DONE
  - ~~SCRUM-37~~ record backup video DONE
- SCRUM-23 AWS
  - SCRUM-45 SetUp backend to ... TO DO
  - SCRUM-46 make IP changes TO DO
  - SCRUM-47 deploy on AWS TO DO
- SCRUM-27 React
  - ~~SCRUM-28~~ Add map to React ... DONE
  - ~~SCRUM-29~~ Connect GPS to MAp DONE
  - ~~SCRUM-30~~ Set up tracking for ... DONE
  - ~~SCRUM-31~~ Optional - Setup St... DONE
  - ~~SCRUM-32~~ Fix issue with fetch... DONE
- SCRUM-33 DataBAse
  - ~~SCRUM-34~~ Set up Database DONE
  - ~~SCRUM-35~~ fix problem with D... DONE
- SCRUM-38 Exercise Modes
- SCRUM-43 Add weather API
  - ~~SCRUM-44~~ Research Weather ... DONE
- SCRUM-48 Add Accounts
- SCRUM-49 Create SignUp
- SCRUM-50 Create Login
- SCRUM-52 JWT Auth for Users
  - ~~SCRUM-53~~ Block Pages unles... DONE
- SCRUM-54 SetUp Profile page
- SCRUM-55 Report
- SCRUM-56 Video
- SCRUM-57 Presentation

Figure 6-1 Jira Board

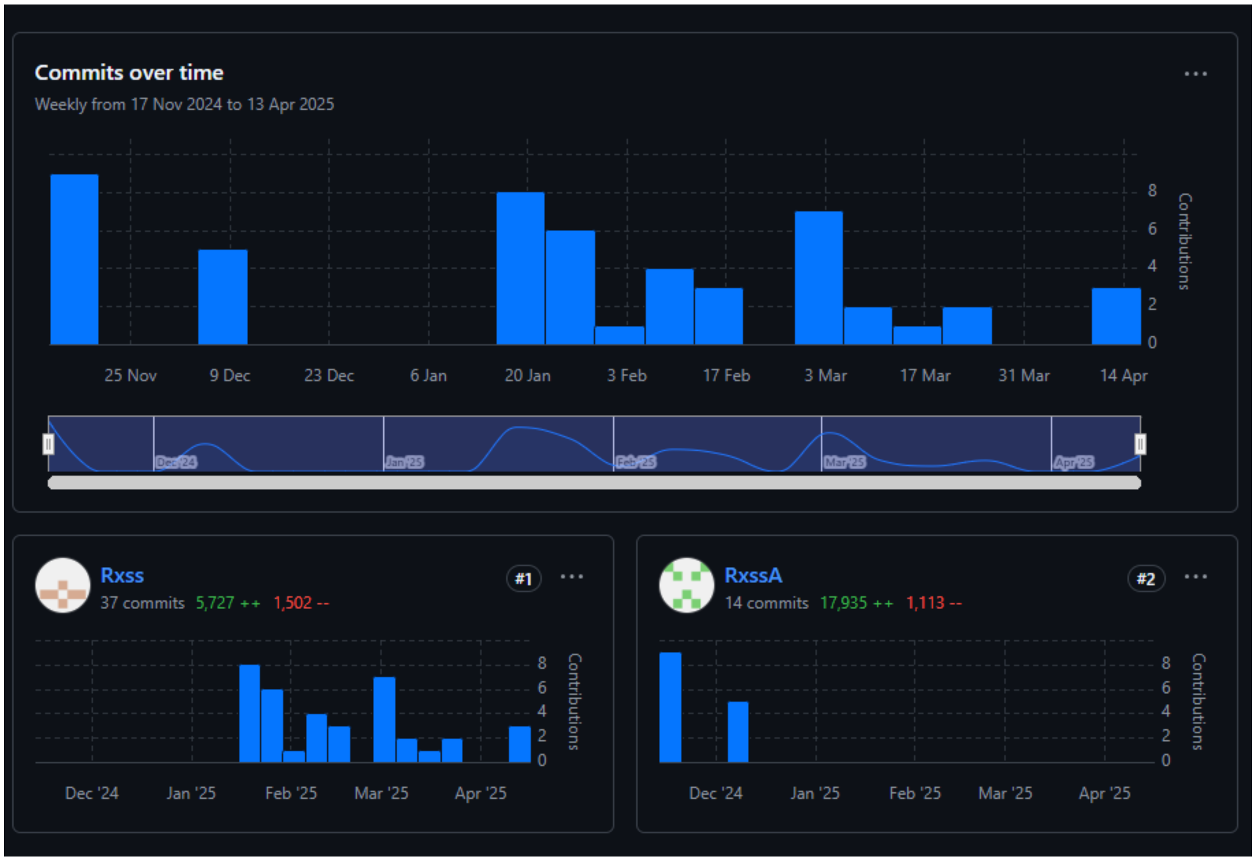
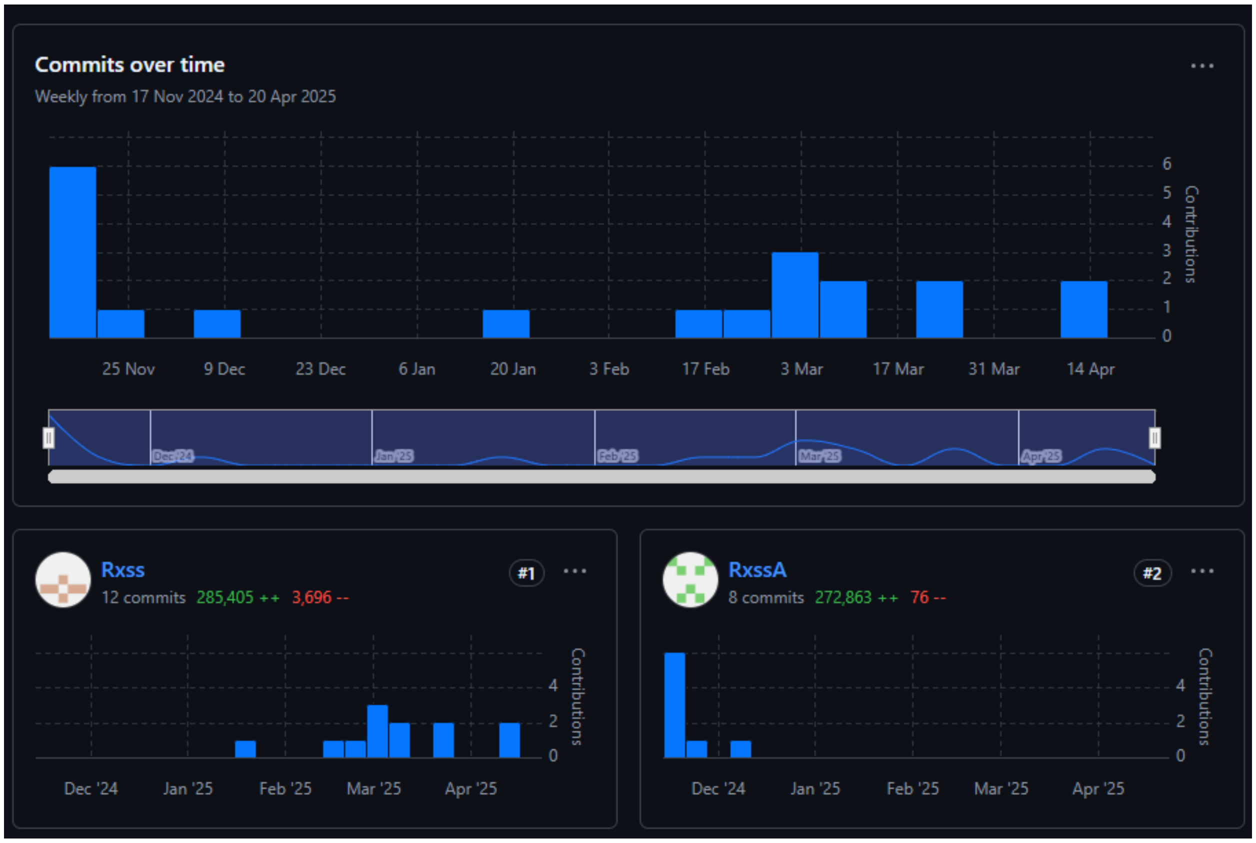


Figure 6-2 GitHub Activity


Commits over time

Weekly from 17 Nov 2024 to 20 Apr 2025

Rxss #1

12 commits 285,405 ++ 3,696 --

RxssA #2

8 commits 272,863 ++ 76 --

Figure 6-3 GitHub Activity

6.2.1 Research & Planning

The first step in the project involved research into suitable components for building the wearable device. The ESP32 microcontroller was chosen as the core of the system due to its built-in Wi-Fi capabilities, low power consumption, and support for multithreading with FreeRTOS. For heart rate monitoring, the MAX30105 was selected as it is compact and can measure pulse through photoplethysmography (PPG). The MCP9808 was selected for temperature monitoring due to its high accuracy and I2C interface, making it easy to integrate with the ESP32. The MTK3339 GPS was selected for real-time location tracking, and ATU already had many available as 2nd years from my course were working with it.

Each component was selected based on factors like reliability, ease of integration, power efficiency, and compatibility with the ESP32. Cost and availability were also considered to ensure the project remained within budget and feasible for prototyping.

After selecting components, a detailed architecture plan was created for both hardware and software. The hardware architecture diagram included wiring planning for the sensor. The software was split into 3 parts:

- **Embedded System (ESP32 + FreeRTOS):** Handles sensor readings, data formatting, Wi-Fi connection, and JSON message transmission.
- **Backend (Node.js + MongoDB):** Receives data from the device, stores it securely in a database, and exposes endpoints and WebSocket [19] channels for the frontend.
- **Frontend (React):** Displays health and activity data through a responsive web interface, using charts and maps for visualization.

6.2.2 Hardware Development

The hardware development phase began with individually testing each sensor to verify its accuracy, and compatibility with the ESP32, this was done with the sample code provided by the required libraries needed for these sensors. The **MAX30105[13]** heart rate sensor was tested under various lighting and motion conditions to ensure consistent PPG readings. Basic filtering techniques were implemented to reduce noise and improve reliability, such as Moving average filters, Peak detection with edge slope check, BPM smoothing with Rolling average buffer. [11]

The MCP9808 [14] temperature sensor was validated for accuracy by comparing its readings with those from a Fitbit device. It consistently produced similar values, indicating that it is well suited for tracking body temperature.

The MTK3339 GPS was tested with miniGPS, this software shows how many satellites are connected to the GPS and is an easy way to test functionality.

These tests ensured that all hardware components performed reliably before moving on to full system integration.

Firmware for the ESP32 was developed using the Arduino framework with **FreeRTOS**[8] [9[12] for task scheduling and multitasking. Two main FreeRTOS tasks were implemented:

- **Task 1 (Data Collection & Processing):**
  Responsible for reading data from the MAX30105, MCP9808, and MTK3339 GPS module. Heart rate values were averaged over a time window, and temperature readings were smoothed to reduce jittering.
- **Task 2 (Data Packaging & Transmission):**
  Compiles sensor data into a JSON object every second and transmits it to the backend using WebSocket over Wi-Fi.

Using FreeRTOS allowed for clean separation of responsibilities between data collection and communication, ensuring efficient real-time performance without blocking. Proper use of semaphores and queues helped manage shared resources and avoid conflicts between tasks.

This phase concluded with a stable, real-time firmware capable of reliable sensing and communication with the backend server.

6.2.3 Backend Development

Node.js Setup

The backend was developed using **Node.js** due to its non-blocking, event-driven architecture, which is ideal for handling real-time data from the wearable device. An Express [17] server was implemented to handle HTTP requests, and CORS was configured to allow communication with the frontend hosted separately.

The backend runs a lightweight RESTful service that receives JSON-formatted sensor data from the ESP32 over HTTP POST requests. It also provides endpoints for authenticated users to fetch historical data.

MongoDB Database design

MongoDB was chosen for its flexibility and scalability. A schema was designed to store user data along with the following fields:

- Heart rate (average, max, min)
- Body temperature

- GPS coordinates (latitude, longitude)
- Timestamps

Each reading is stored as a document, allowing for easy aggregation and visualization later. The data model supports scalability and future additions like activity logs or session summaries.

In addition to the REST API, a **WebSocket connection** was established to enable real-time updates between the backend and the web application. When new sensor data arrives from the device, the server immediately pushes it to connected frontend clients via the WebSocket, allowing for live updates on charts and maps without polling.

6.2.4 Frontend Development

UI design with React

The frontend of the application was developed using **React** [19] with the **Next.js** framework to create a fast, responsive, and user-friendly interface. The design follows a clean, modern aesthetic, making it suitable for use across devices.

Data Visualization (Graphs, Google Maps API)

Real-time sensor data is visualized using dynamic charts and maps:

- **Heart rate** and **temperature** trends are displayed with interactive line graphs, allowing users to track their biometric changes over time.
- The **Google Maps API[10]** is integrated to show live GPS tracking on a map. The user's location updates as the ESP32 sends new coordinates, providing context to outdoor activity data.
- Additional UI elements like cards and charts display summary statistics (e.g. average, max, and min values).

User account system

A **user authentication system** was implemented using JSON Web Tokens (JWT)[21] to allow secure login and protect personal health data. Users can sign up, log in, and view their historical data.

Authentication ensures that only authorized users can access their data, maintaining privacy and security across the platform.

6.3 Tools & Technologies

Embedded hardware & firmware:

- **ESP32 with FreeRTOS (C/C++)**: The core microcontroller used for sensor data acquisition and Wi-Fi communication. FreeRTOS was utilized for efficient task management.
- **Sensors**:
- **MAX30105**: For heart rate measurement using photoplethysmography (PPG).
- **MCP9808**: High-accuracy digital temperature sensor.
- **MTK3339 GPS Module**: For real-time location tracking using NMEA data streams.

Backend:

- **Node.js & Express**: Lightweight and scalable server-side framework used to handle HTTP requests and real-time WebSocket connections.
- **WebSocket**: Enables real-time bi-directional communication between the server and web clients.
- **MongoDB**: NoSQL database for storing time-stamped biometric and location data efficiently.

Frontend:

- **React & Next.js**: Frameworks used for building a responsive, modern user interface with server-side rendering and fast client-side navigation.
- **Google Maps API**: Used to visualize real-time GPS location on an interactive map, enhancing context for user activity.

- **Open-Meteo API:** Used to displaying weather information based on users' location

6.4 Risk Management

Effective risk management was essential to ensure the successful delivery of the wearable health and activity tracking system. The following potential risks were identified and addressed:

Hardware Failure or Sensor inaccuracies

- **Risk**: Sensors may produce inaccurate readings or fail due to hardware limitations or improper handling.
- **Mitigation**: Components were individually tested during development. Data smoothing techniques and calibration were applied to improve reliability.

WiFi connectivity issues

- **Risk**: The wearable device relies on Wi-Fi to transmit data. Connectivity loss could disrupt data flow.
- **Mitigation**: The firmware includes basic reconnection logic. Future improvements could add offline data caching.

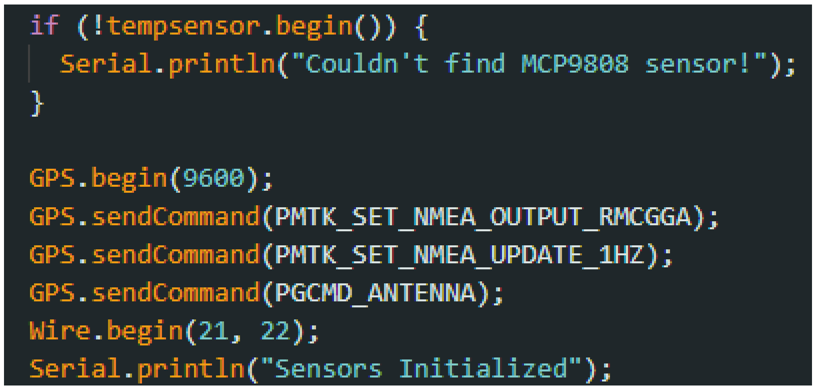
Data security and user authentication

- **Risk**: Sensitive health and location data must be protected from unauthorized access.
- **Mitigation**: A user login system using JWT tokens was implemented. HTTPS and secure storage mechanisms can be incorporated in future versions.

7 Deep Dive into the WHAT

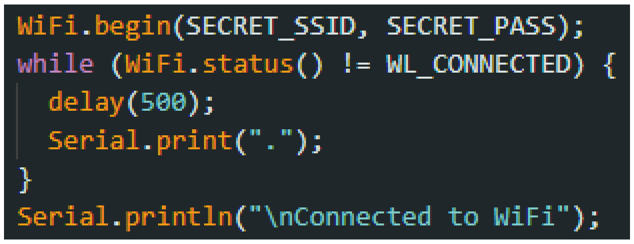
7.1 ESP32 Firmware and System Integration

The ESP32 microcontroller acts as the central hub of the Wearable Health and Activity Tracker system. Its firmware is written in C/C++ using the Arduino IDE, it integrates multiple sensors to collect biometric and environmental data and transmits this data to a cloud backend via HTTP POST requests. The main components include the **MAX30105** heart rate sensor, **MCP9808** temperature sensor, and the **MTK3339** GPS module, all of which are initialized and managed in the `setup()` function.



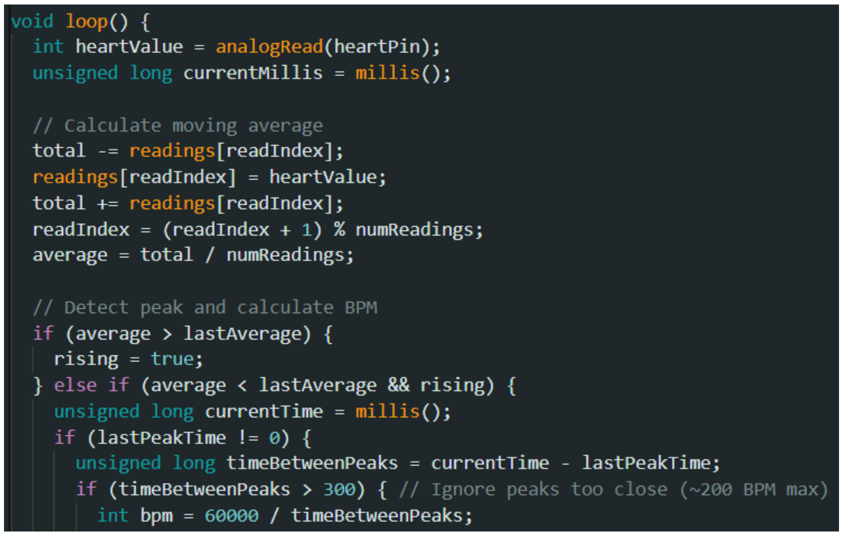
```
if (!tempsensor.begin()) {
  Serial.println("Couldn't find MCP9808 sensor!");
}

GPS.begin(9600);
GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
GPS.sendCommand(PGCMD_ANTENNA);
Wire.begin(21, 22);
Serial.println("Sensors Initialized");
```

Wi-Fi is configured with credentials using `WiFi.begin()`, and the ESP32 attempts to connect before proceeding with data collection. This enables the microcontroller to send data to the backend server hosted on a local network.

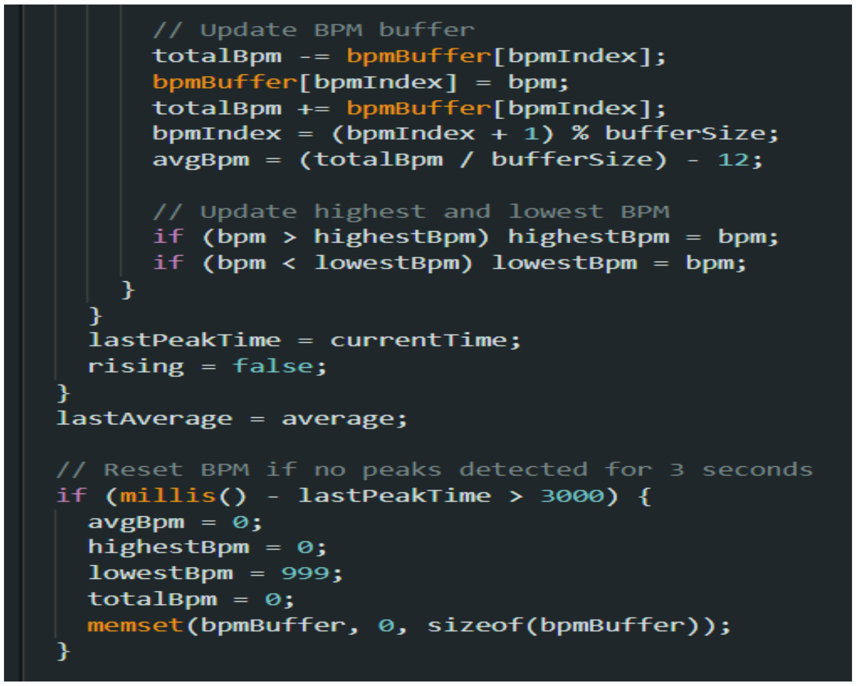


```
WiFi.begin(SECRET_SSID, SECRET_PASS);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("\nConnected to WiFi");
```

The BPM is calculated by detecting peaks in the heart rate sensor's signal. The sensor's raw readings are first smoothed using a moving average over 20 samples to reduce noise. A peak is detected when the smoothed value stops rising and begins to fall, indicating a heartbeat. The time interval between peaks is measured, and the BPM is computed as `60000 / timeBetweenPeaks`, converting the interval from milliseconds to beats per minute. To stabilize the BPM value, the last 10 BPM readings are stored in a buffer and averaged. Additionally, the code tracks the highest and lowest BPM values. If no peak is detected for 3 seconds, the BPM is reset to indicate a lack heartbeat detection.

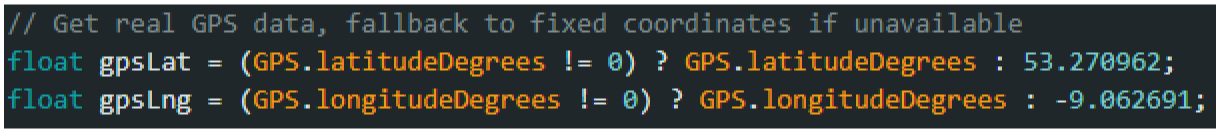


```
void loop() {
  int heartValue = analogRead(heartPin);
  unsigned long currentMillis = millis();

  // Calculate moving average
  total -= readings[readIndex];
  readings[readIndex] = heartValue;
  total += readings[readIndex];
  readIndex = (readIndex + 1) % numReadings;
  average = total / numReadings;

  // Detect peak and calculate BPM
  if (average > lastAverage) {
    rising = true;
  } else if (average < lastAverage && rising) {
    unsigned long currentTime = millis();
    if (lastPeakTime != 0) {
      unsigned long timeBetweenPeaks = currentTime - lastPeakTime;
      if (timeBetweenPeaks > 300) { // Ignore peaks too close (~200 BPM max)
        int bpm = 60000 / timeBetweenPeaks;
```



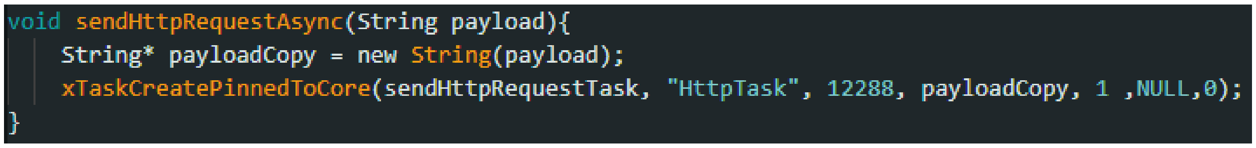
```
        // Update BPM buffer
        totalBpm -= bpmBuffer[bpmIndex];
        bpmBuffer[bpmIndex] = bpm;
        totalBpm += bpmBuffer[bpmIndex];
        bpmIndex = (bpmIndex + 1) % bufferSize;
        avgBpm = (totalBpm / bufferSize) - 12;

        // Update highest and lowest BPM
        if (bpm > highestBpm) highestBpm = bpm;
        if (bpm < lowestBpm) lowestBpm = bpm;
      }
    }
    lastPeakTime = currentTime;
    rising = false;
  }
  lastAverage = average;

  // Reset BPM if no peaks detected for 3 seconds
  if (millis() - lastPeakTime > 3000) {
    avgBpm = 0;
    highestBpm = 0;
    lowestBpm = 999;
    totalBpm = 0;
    memset(bpmBuffer, 0, sizeof(bpmBuffer));
  }
```

GPS data is read using the `Adafruit_GPS` library and parsed for latitude and longitude. If the GPS signal is unavailable, hardcoded fallback coordinates are used:



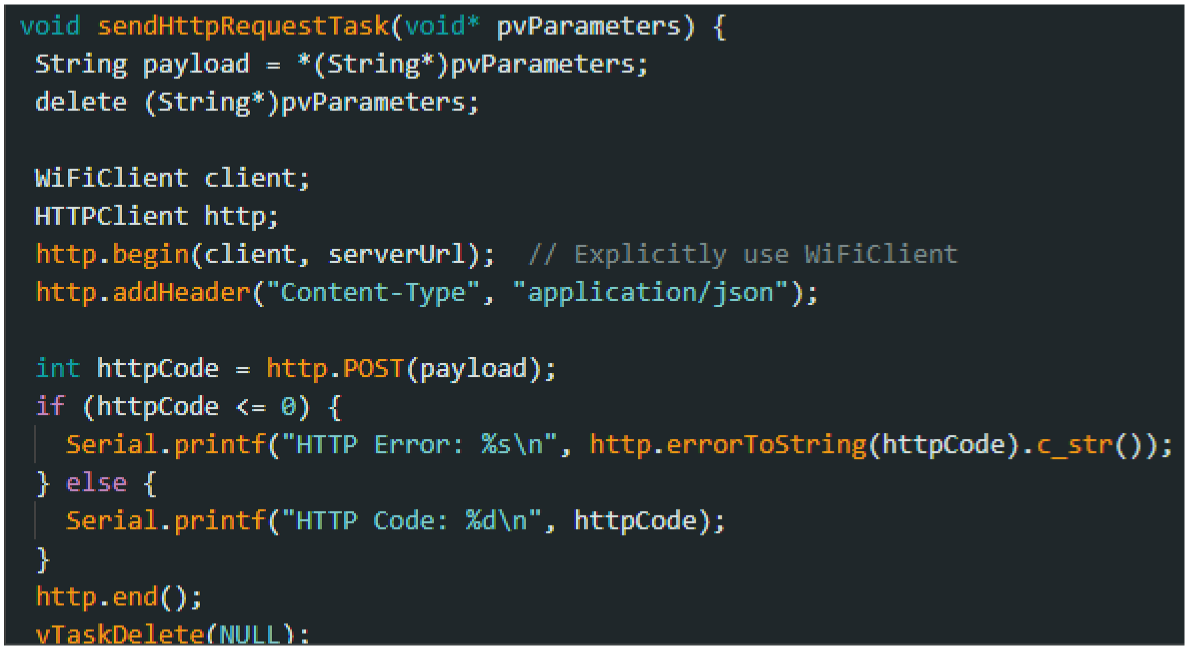
```
// Get real GPS data, fallback to fixed coordinates if unavailable
float gpsLat = (GPS.latitudeDegrees != 0) ? GPS.latitudeDegrees : 53.270962;
float gpsLng = (GPS.longitudeDegrees != 0) ? GPS.longitudeDegrees : -9.062691;
```

To avoid blocking the main loop when sending the JSON post request, the ESP32 uses FreeRTOS to create a separate task for HTTP communication. The payload is constructed as a JSON string containing heart rate, temperature and GPS data. The `sendHttpRequestAsync()` function enables non-blocking communication between the wearable device and the backend server. It creates a new FreeRTOS task that runs independently of the main sensor loop, ensuring that time-sensitive operations like heart rate calculation and GPS readings are not interrupted by slow network calls. The function first creates a dynamically allocated copy of the JSON-formatted sensor data (`payload`) and then uses `xTaskCreatePinnedToCore()` to launch the `sendHttpRequestTask` on core 0 of the ESP32. By offloading the HTTP POST request to a separate core, the device can continue collecting real-time data without blocking the JSON post requests.



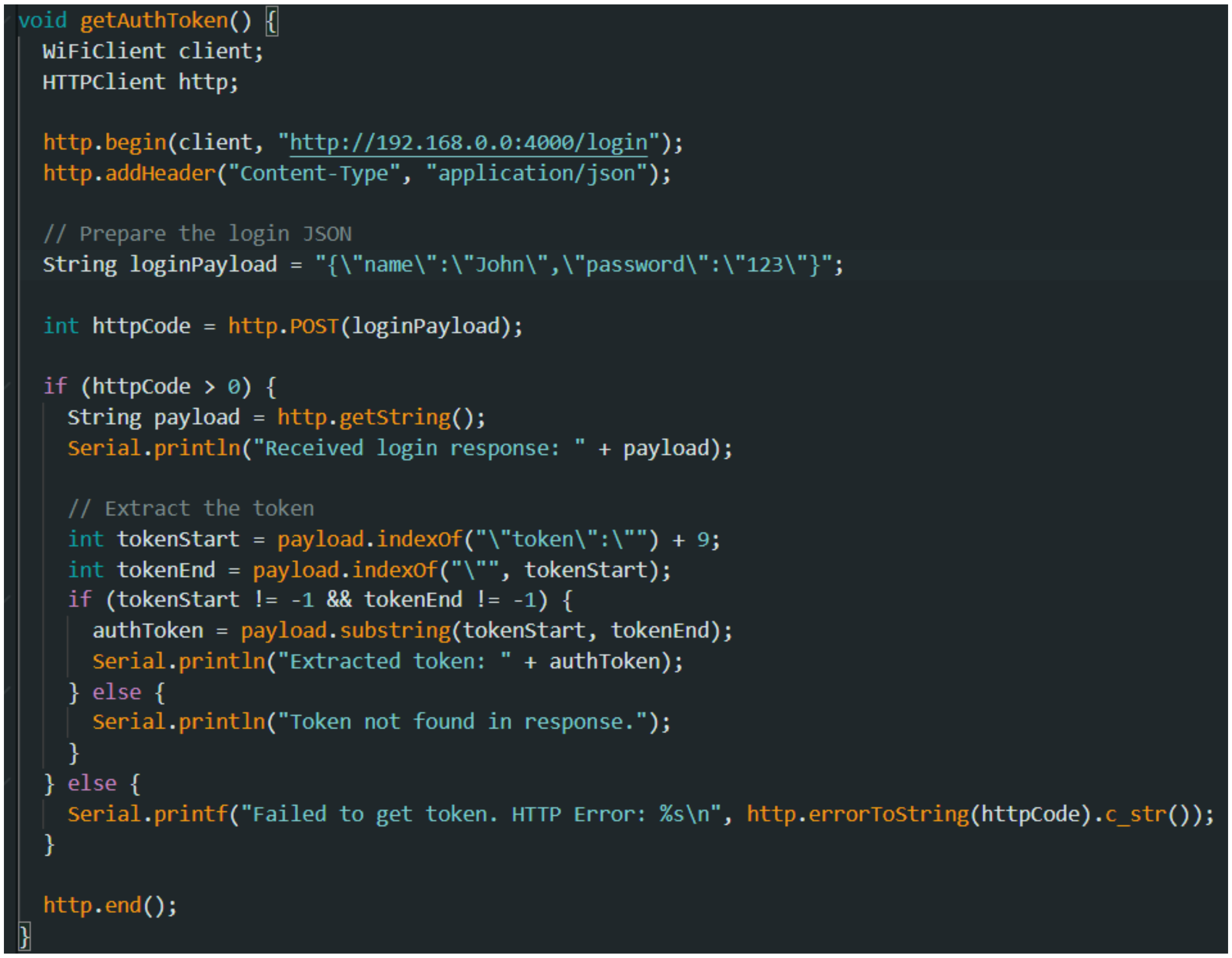
```
void sendHttpRequestAsync(String payload){
    String* payloadCopy = new String(payload);
    xTaskCreatePinnedToCore(sendHttpRequestTask, "HttpTask", 12288, payloadCopy, 1 ,NULL,0);
}
```

The sendHttpRequestTask function uses HTTPClient to send data to the Node.js backend



```
void sendHttpRequestTask(void* pvParameters) {
  String payload = *(String*)pvParameters;
  delete (String*)pvParameters;

  WiFiClient client;
  HTTPClient http;
  http.begin(client, serverUrl);  // Explicitly use WiFiClient
  http.addHeader("Content-Type", "application/json");

  int httpCode = http.POST(payload);
  if (httpCode <= 0) {
    Serial.printf("HTTP Error: %s\n", http.errorToString(httpCode).c_str());
  } else {
    Serial.printf("HTTP Code: %d\n", httpCode);
  }
  http.end();
  vTaskDelete(NULL);
```

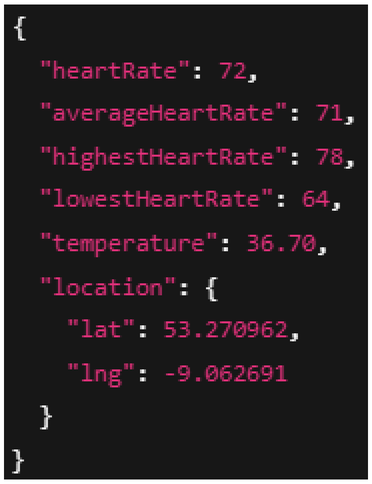
Authentication with the backend server is handled through the getAuthToken() function. This function connects to the server's /login endpoint using an HTTP POST request and sends login credentials in JSON format. After establishing a connection with WiFiClient and HTTPClient, it constructs a JSON payload containing a username and password and posts it to the server. If the server responds successfully, the function reads the server's response, extracts the token field manually by locating the "token" key within the returned JSON string, and stores it in a global variable authToken. This token is then used to authenticate HTTP requests.



```
void getAuthToken() {
  WiFiClient client;
  HTTPClient http;

  http.begin(client, "http://192.168.0.0:4000/login");
  http.addHeader("Content-Type", "application/json");

  // Prepare the login JSON
  String loginPayload = "{\"name\":\"John\",\"password\":\"123\"}";

  int httpCode = http.POST(loginPayload);

  if (httpCode > 0) {
    String payload = http.getString();
    Serial.println("Received login response: " + payload);

    // Extract the token
    int tokenStart = payload.indexOf("\"token\":\"") + 9;
    int tokenEnd = payload.indexOf("\"", tokenStart);
    if (tokenStart != -1 && tokenEnd != -1) {
      authToken = payload.substring(tokenStart, tokenEnd);
      Serial.println("Extracted token: " + authToken);
    } else {
      Serial.println("Token not found in response.");
    }
  } else {
    Serial.printf("Failed to get token. HTTP Error: %s\n", http.errorToString(httpCode).c_str());
  }

  http.end();
}
```

On the backend, the Node.js server receives this JSON data through the API endpoint `/data`. It parses the content and stores it in the MongoDB database, allowing the system to store user biometric records for visualization.

The frontend, built with React uses REST API calls to retrieve historical and real-time data. Graphs display average heart rate, highest, and lowest over time using charting libraries, and Google Maps API is used to visualize user location.

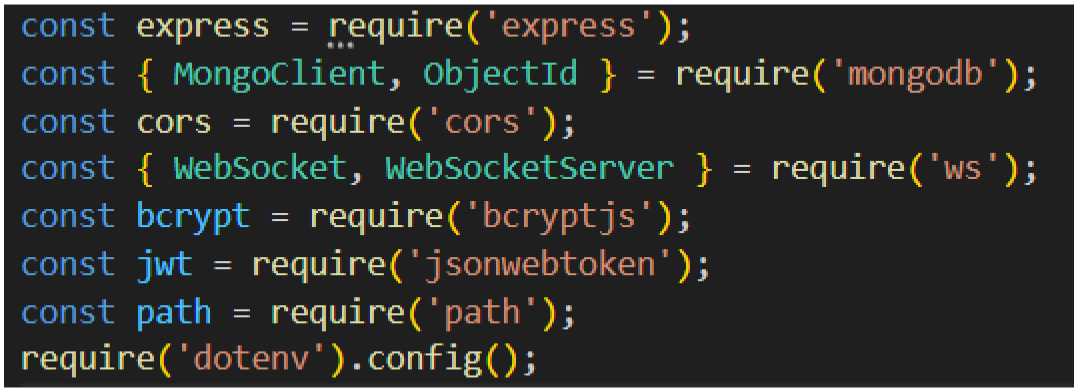


```json
{
  "heartRate": 72,
  "averageHeartRate": 71,
  "highestHeartRate": 78,
  "lowestHeartRate": 64,
  "temperature": 36.70,
  "location": {
    "lat": 53.270962,
    "lng": -9.062691
  }
}
```

7.2 Node.js Backend

The backend for the wearable health and activity tracker is a Node.js server built using the Express.js framework. It acts as the central hub between the three major components: the **ESP32 device**, the **frontend React app**, and the **MongoDB database**. This backend handles incoming data from the ESP32, manages user authentication, stores data and reports, and provides real-time updates to the frontend using **WebSockets**.

The server begins by importing essential modules:



```javascript
const express = require('express');
const { MongoClient, ObjectId } = require('mongodb');
const cors = require('cors');
const { WebSocket, WebSocketServer } = require('ws');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const path = require('path');
require('dotenv').config();
```

- **Express** is used to handle HTTP routes and RESTful API endpoints.
- **MongoDB** stores user data, sensor readings, and activity reports.

- **CORS** allows the frontend to make requests to the backend from a different origin.
- **bcryptjs** and **jsonwebtoken** are used to securely handle user authentication.
- **WebSockets** enable live communication between the server and frontend without needing repeated HTTP requests.

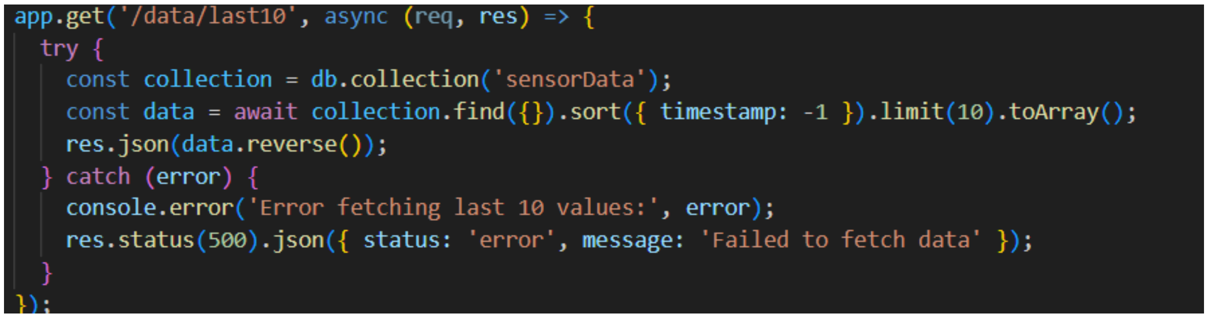
Handling HTTP Requests with Express:

The backend uses Express to define REST API routes for various purposes:

- /signup and /login handles user registration and login using hashed passwords with bcrypt.js and JWT-based authentication.
- /profile authenticates and returns user profile information.
- /data receives heart rate, temperature, and GPS data from the ESP32 device via HTTP post.
- /data/last10 sends the last 10 sensor data entries to the frontend, used for the graphs.
- /runreport, /walkreport, workoutreport endpoints to save different types of activity reports, authenticated with JWT.

Example:

This route is where the ESP32 sends data using an HTTP post request, it saves the sensor values in MongoDB and broadcasts them to the frontend via WebSockets.[1][2][3]



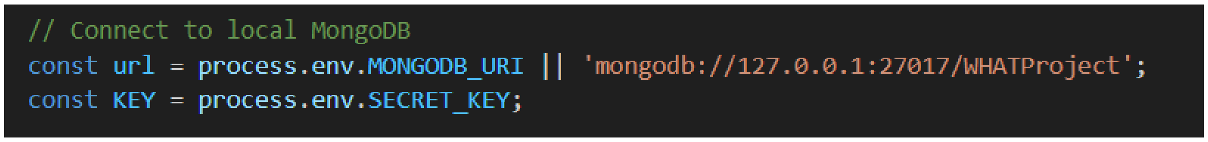
```
app.get('/data/last10', async (req, res) => {
  try {
    const collection = db.collection('sensorData');
    const data = await collection.find({}).sort({ timestamp: -1 }).limit(10).toArray();
    res.json(data.reverse());
  } catch (error) {
    console.error('Error fetching last 10 values:', error);
    res.status(500).json({ status: 'error', message: 'Failed to fetch data' });
  }
});
```

MongoDB integration:

The server uses MongoDB to store all sensor and user-related data. The collections used include:

- `sensorData` for heart rate, temperature, and location data.
- `users` for user credentials and personal details.
- `runReports`, `walkReports`, and `workoutReports` for fitness data.

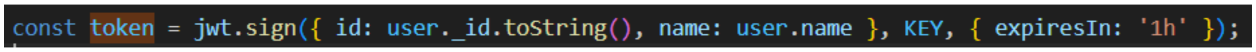
The connection is initialized here, the "SECRET_KEY" is stored in the .env which is also in the .gitignore file, so it is not accessible by the public through GitHub.



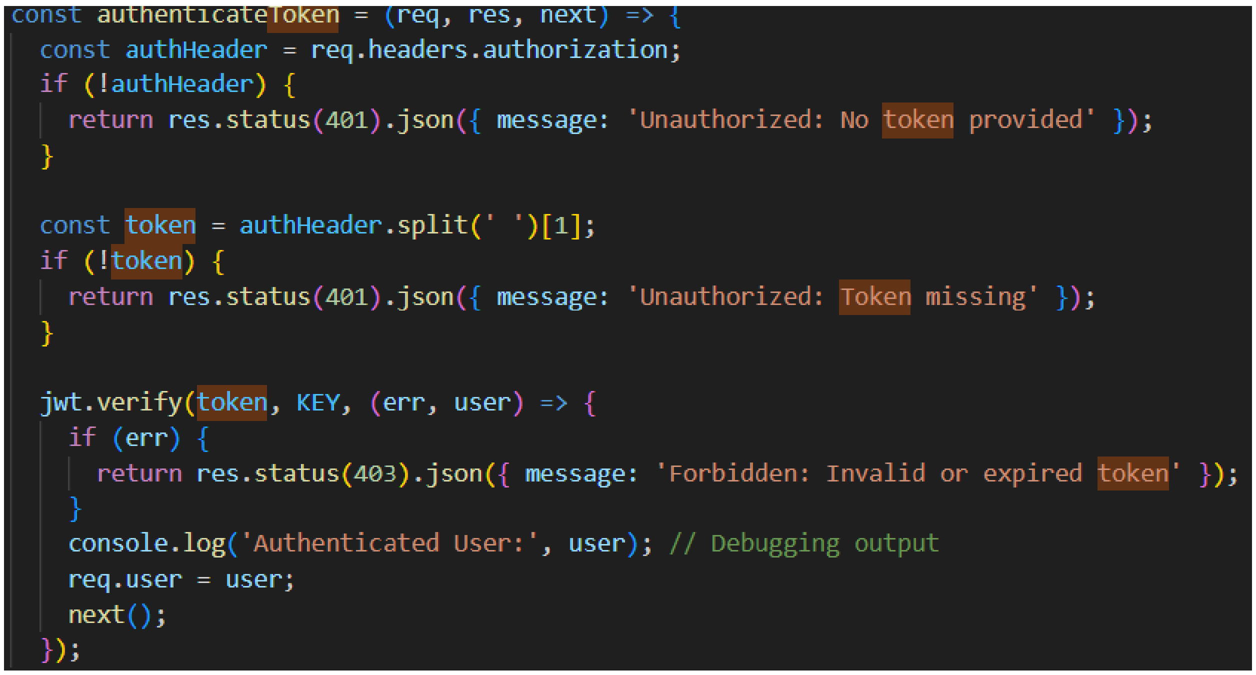
```
// Connect to local MongoDB
const url = process.env.MONGODB_URI || 'mongodb://127.0.0.1:27017/WHATProject';
const KEY = process.env.SECRET_KEY;
```

User Authentication:

JWT tokens are used to secure endpoints and verify users. Tokens are signed with a secret key:

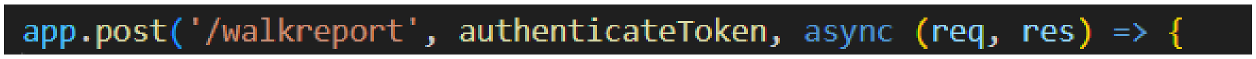


```
const token = jwt.sign({ id: user._id.toString(), name: user.name }, KEY, { expiresIn: '1h' });
```



```
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).json({ message: 'Unauthorized: No token provided' });
  }

  const token = authHeader.split(' ')[1];
  if (!token) {
    return res.status(401).json({ message: 'Unauthorized: Token missing' });
  }

  jwt.verify(token, KEY, (err, user) => {
    if (err) {
      return res.status(403).json({ message: 'Forbidden: Invalid or expired token' });
    }
    console.log('Authenticated User:', user); // Debugging output
    req.user = user;
    next();
  });
```

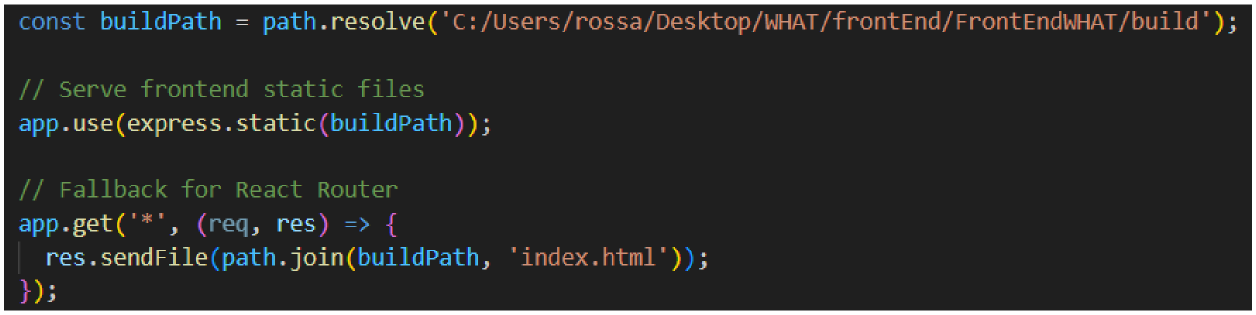
This token is sent in the `Authorization` header for protected routes like `/walkreport`.



```
app.post('/walkreport', authenticateToken, async (req, res) => {
```

Serving the React Frontend:

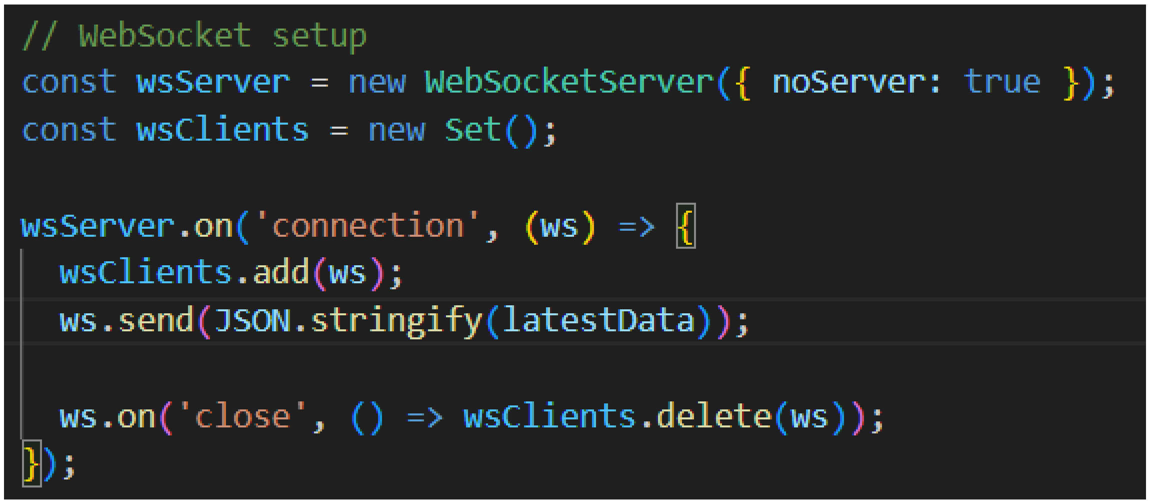
The server hosts a production build of the React frontend, so when starting up the server, there is no need to start up the frontend.



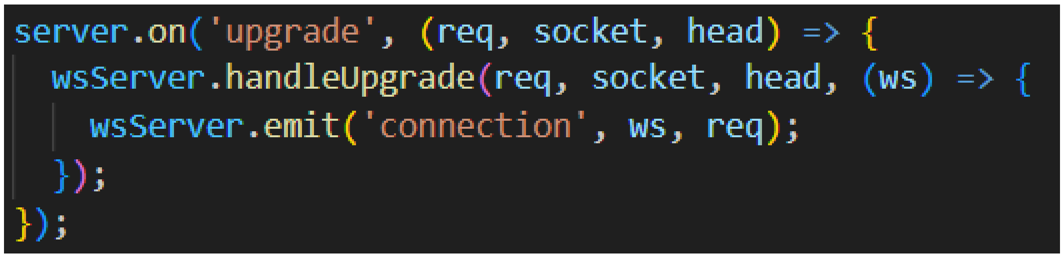
```
const buildPath = path.resolve('C:/Users/rossa/Desktop/WHAT/frontEnd/FrontEndWHAT/build');

// Serve frontend static files
app.use(express.static(buildPath));

// Fallback for React Router
app.get('*', (req, res) => {
  res.sendFile(path.join(buildPath, 'index.html'));
});
```

WebSockets for real-time updates

The WebSocket server is created using the ws library, it listens for upgrade requests from the HTTP server, when a connection is established, the most recent sensor reading is sent immediately.

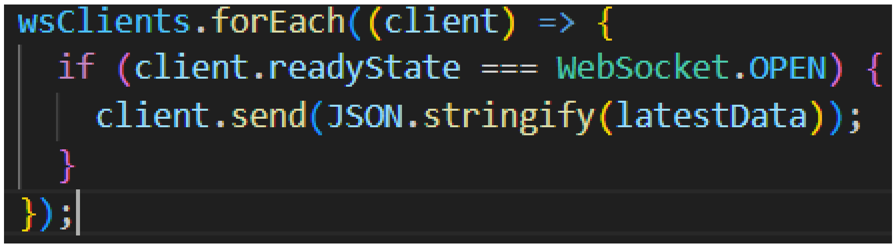


```
// WebSocket setup
const wsServer = new WebSocketServer({ noServer: true });
const wsClients = new Set();

wsServer.on('connection', (ws) => {
  wsClients.add(ws);
  ws.send(JSON.stringify(latestData));

  ws.on('close', () => wsClients.delete(ws));
});
```



```
server.on('upgrade', (req, socket, head) => {
  wsServer.handleUpgrade(req, socket, head, (ws) => {
    wsServer.emit('connection', ws, req);
  });
});
```

WebSockets are used to **push live sensor data to the frontend** without requiring constant polling. When the ESP32 sends a POST to `/data`, the latest data is sent to all connected clients via WebSocket:



```
wsClients.forEach((client) => {
  if (client.readyState === WebSocket.OPEN) {
    client.send(JSON.stringify(latestData));
  }
});
```

This enables **real-time display of sensor data on** dashboards in the frontend, which improves user experience for live health monitoring.

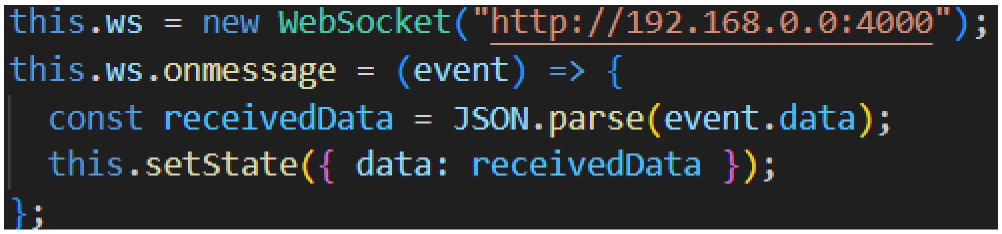
7.3 React Frontend

The frontend of the Wearable Health and Activity Tracker project is developed using **React** and functions as the user interface for viewing biometric and activity data collected by the ESP32 device. It provides real-time feedback using WebSockets, supports user authentication, and visually presents health and location information through an interactive interface.

The HomePage component serves as the central entry point of the application and has the following key functionalities:

7.3.1 Real-Time Data Reception via WebSocket

a WebSocket connection is established with the backend server, when data is received through this connection, the component updates its state



```
this.ws = new WebSocket("http://192.168.0.0:4000");
this.ws.onmessage = (event) => {
  const receivedData = JSON.parse(event.data);
  this.setState({ data: receivedData });
};
```

The incoming payload contains:

- Bearer Token
- Heart rate (bpm)

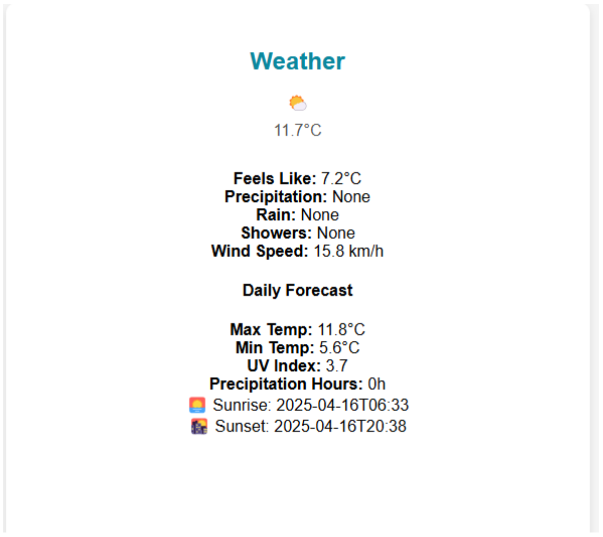
- Body temperature (°C)
- GPS location (latitude and longitude)
- Timestamp of last update

This approach allows the frontend to see the current sensor readings without requiring the user to refresh or manually poll the server.

7.3.2 Weather Integration

The component fetches weather data from the Open-Meteo API on initial mount. The request includes parameters for:

- Current Temperature
- Daily minimum and maximum temperatures
- Rains percentage
- Wind Speeds
- UV index
- Sunrise & Sunset



Weather

11.7°C

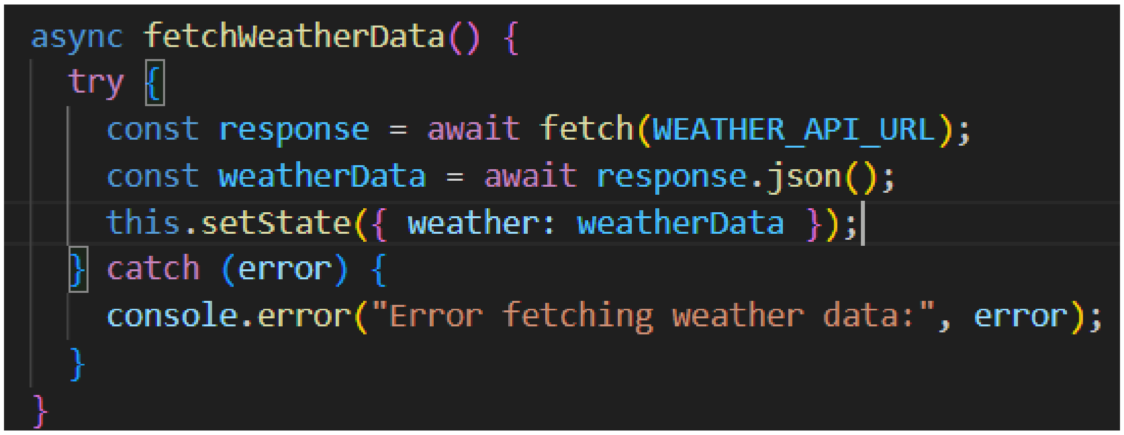
Feels Like: 7.2°C
Precipitation: None
Rain: None
Showers: None
Wind Speed: 15.8 km/h

Daily Forecast

Max Temp: 11.8°C
Min Temp: 5.6°C
UV Index: 3.7
Precipitation Hours: 0h
Sunrise: 2025-04-16T06:33
Sunset: 2025-04-16T20:38

The weather data is retrieved using the following logic:

```
const WEATHER_API_URL = "";
```



```
async fetchWeatherData() {
  try {
    const response = await fetch(WEATHER_API_URL);
    const weatherData = await response.json();
    this.setState({ weather: weatherData });
  } catch (error) {
    console.error("Error fetching weather data:", error);
  }
}
```

7.3.3 Authentication Management

The frontend manages user authentication with localStorage and tracks login state in the component's state:

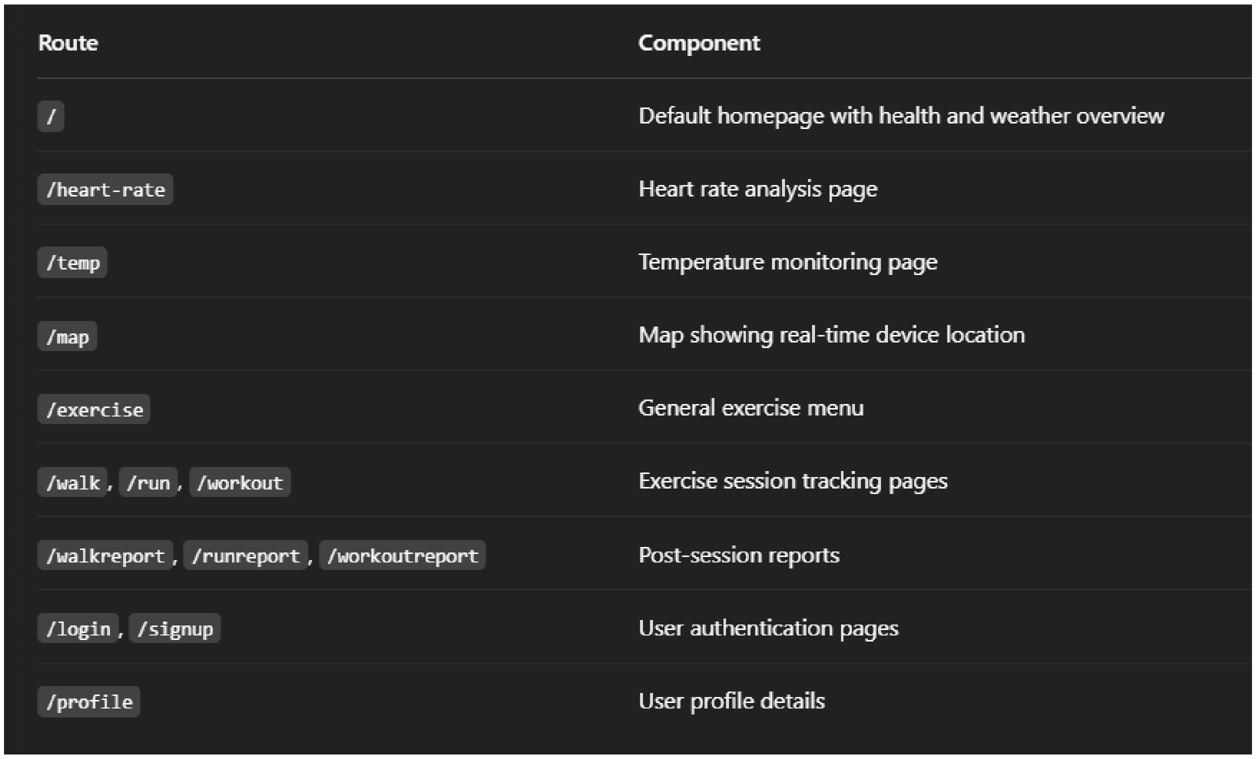
- handleAuthSuccess(username) sets the login flag and stores the username.
- handleLogout() clears the session.

Authentication UI elements include:

- A profile avatar in the navbar that displays the user's initial of their first name.
- A dropdown menu for accessing the profile page and logging out.
- Conditional rendering of "Login" and "Signup" routes if the user is not logged in.

7.3.4 Navigation and Routing

Routing is handled using react-router-dom. The HomePage component defines the core layout and sets up navigation to subpages, each responsible for aspects of the health tracking data:



Route	Component
`/`	Default homepage with health and weather overview
`/heart-rate`	Heart rate analysis page
`/temp`	Temperature monitoring page
`/map`	Map showing real-time device location
`/exercise`	General exercise menu
`/walk`, `/run`, `/workout`	Exercise session tracking pages
`/walkreport`, `/runreport`, `/workoutreport`	Post-session reports
`/login`, `/signup`	User authentication pages
`/profile`	User profile details

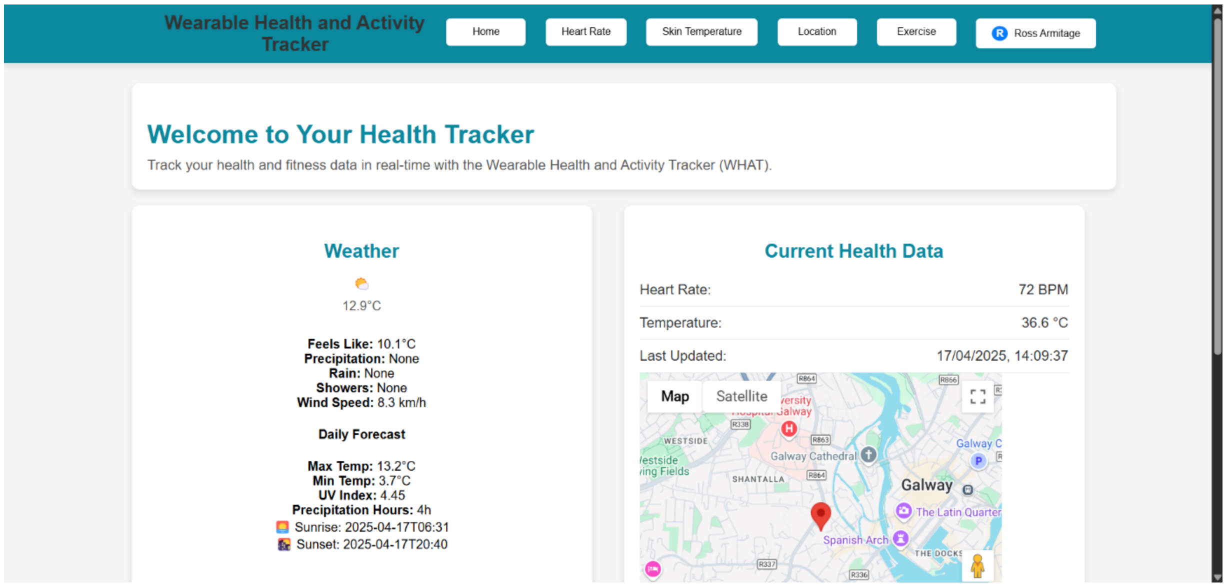
Figure 7-1 Routes [7]

Routing uses the `<Routes>` and `<Route>` components and passes down WebSocket data to subcomponents with props.

7.3.5 User Interface and Layout

The main HomePage layout consists of:

- A responsive navigation bar.
- A left panel containing weather data and a welcome message.
- A right panel showing real-time heart rate, temperature, and location on a map.



Welcome to Your Health Tracker

Track your health and fitness data in real-time with the Wearable Health and Activity Tracker (WHAT).

Weather

12.9°C

Feels Like: 10.1°C
Precipitation: None
Rain: None
Showers: None
Wind Speed: 8.3 km/h

Daily Forecast

Max Temp: 13.2°C
Min Temp: 3.7°C
UV Index: 4.45
Precipitation Hours: 4h
Sunrise: 2025-04-17T06:31
Sunset: 2025-04-17T20:40

Current Health Data

Heart Rate:	72 BPM
Temperature:	36.6 °C
Last Updated:	17/04/2025, 14:09:37

Figure 7-2 Home Page

8 Ethics

When developing the Wearable Health and Activity Tracker, several ethical considerations were considered, Firstly, data privacy was a concern, as the device collects sensitive biometric information such as heart rate, body temperature and location data. Additionally, the accuracy and reliability of the sensor readings were considered essential to avoid providing misleading health information.

9 Problems

One of the main challenges I encountered during the development of this project was getting accurate readings from the heart rate sensor (MAX30105). Initially, the BPM values were inaccurate. I suspected the issue was related to the signal processing and attempted several filtering techniques to improve the accuracy. These included a moving average filter, BPM smoothing algorithms and adjusting the peak detection logic, all to refine the heart rate calculation.

However, I discovered that the root of the problem was not in the sensor data processing itself, but in the way the data was being transmitted to the backend. The HTTP POST requests, which were being sent once every second. This blocking behaviour was interfering with the timing-sensitive heart rate calculations.

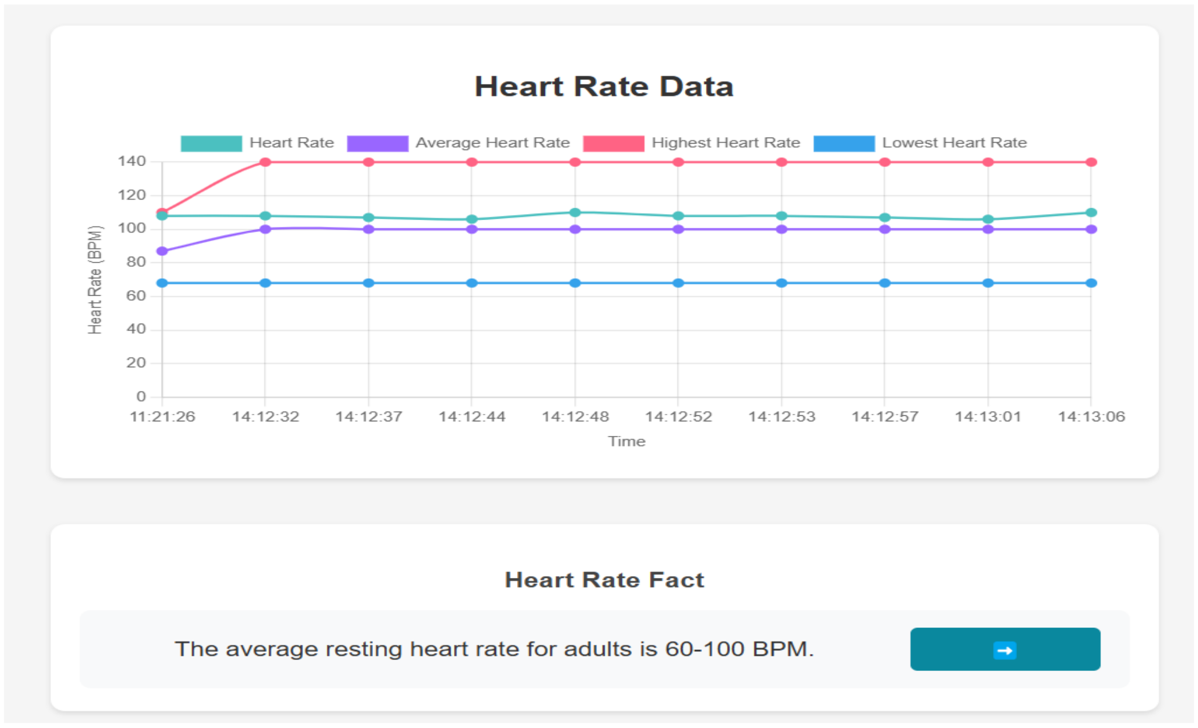
To resolve this issue, I integrated FreeRTOS into the ESP32 firmware. By setting the HTTP communication to a separate FreeRTOS task, I was able to ensure that the heart rate processing task could run independently and most importantly, uninterrupted.

Another major challenge I faced was when deploying my project to AWS. While the system ran perfectly on my local machine, it behaved differently once deployed to the cloud, even after updating all IP addresses to the AWS public IP. Debugging this was difficult because the errors were inconsistent and environment specific.

In hindsight, to avoid these deployment issues and ensure consistency across different environments, I should have containerized the backend using Docker. This would have created a predictable environment that behaves the same way locally and on AWS.

10 Conclusion

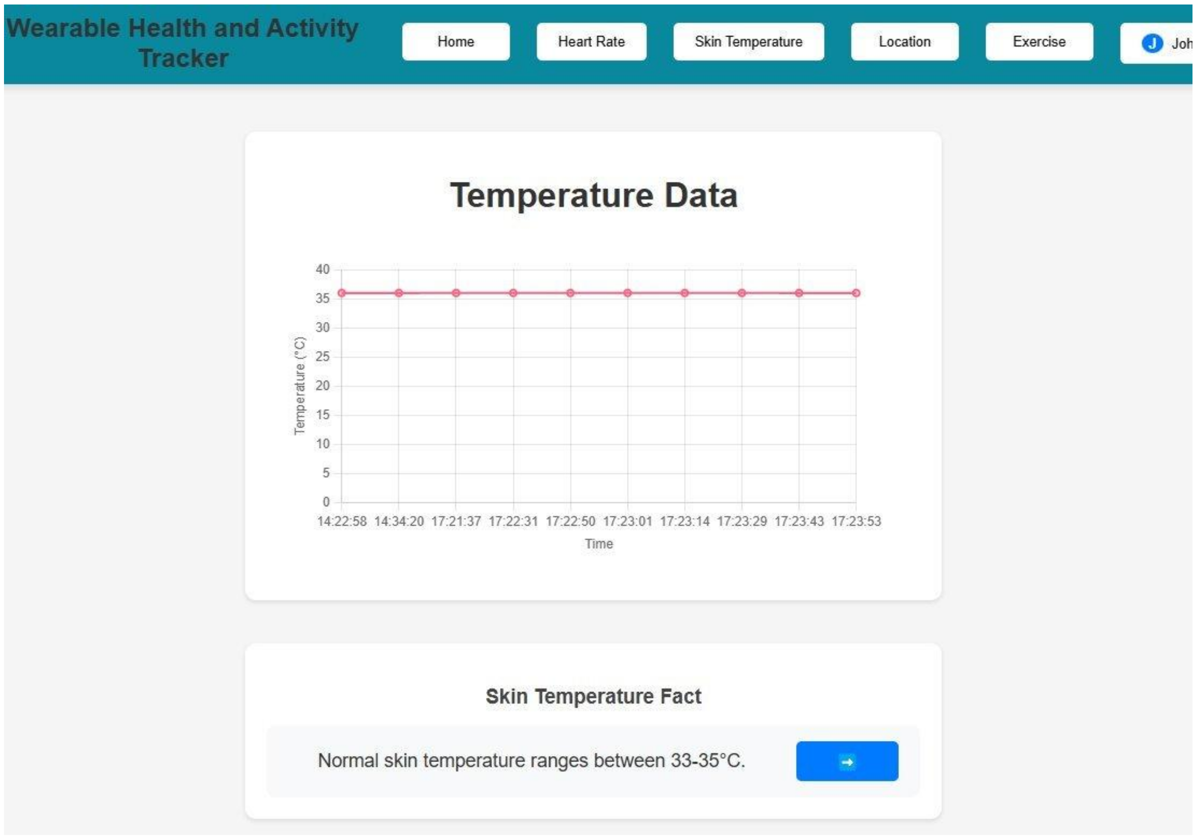
Overall, the Wearable Health and Activity Tracker project was a successful experience that allowed me to apply both hardware and software skills in a real-world context. I achieved all the core goals I set out at the start, including reading and processing biometric data such as heart rate and body temperature. This data was visualized in real-time through responsive graphs, helping users easily track their health trends over time.



Heart Rate Fact

The average resting heart rate for adults is 60-100 BPM.

Figure 10-1 Heart Rate Page



Skin Temperature Fact

Normal skin temperature ranges between 33-35°C.

Figure 10-2 Temperature Page

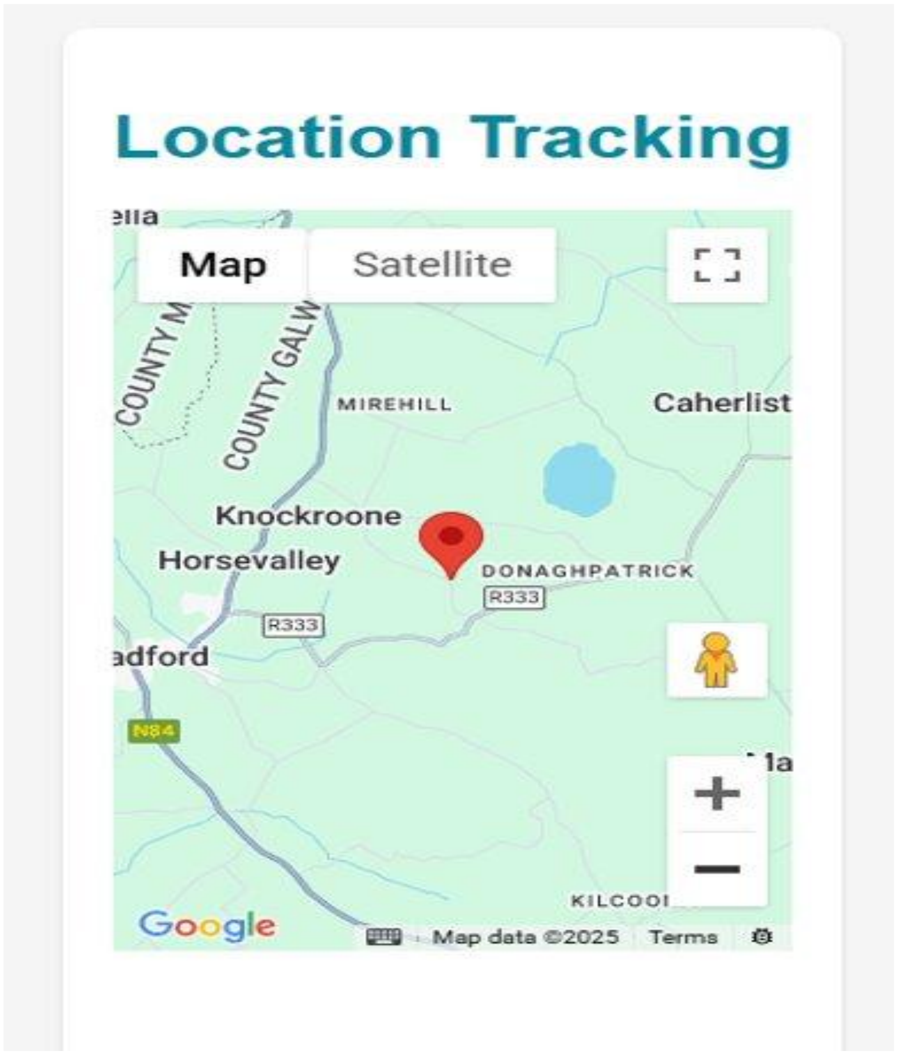
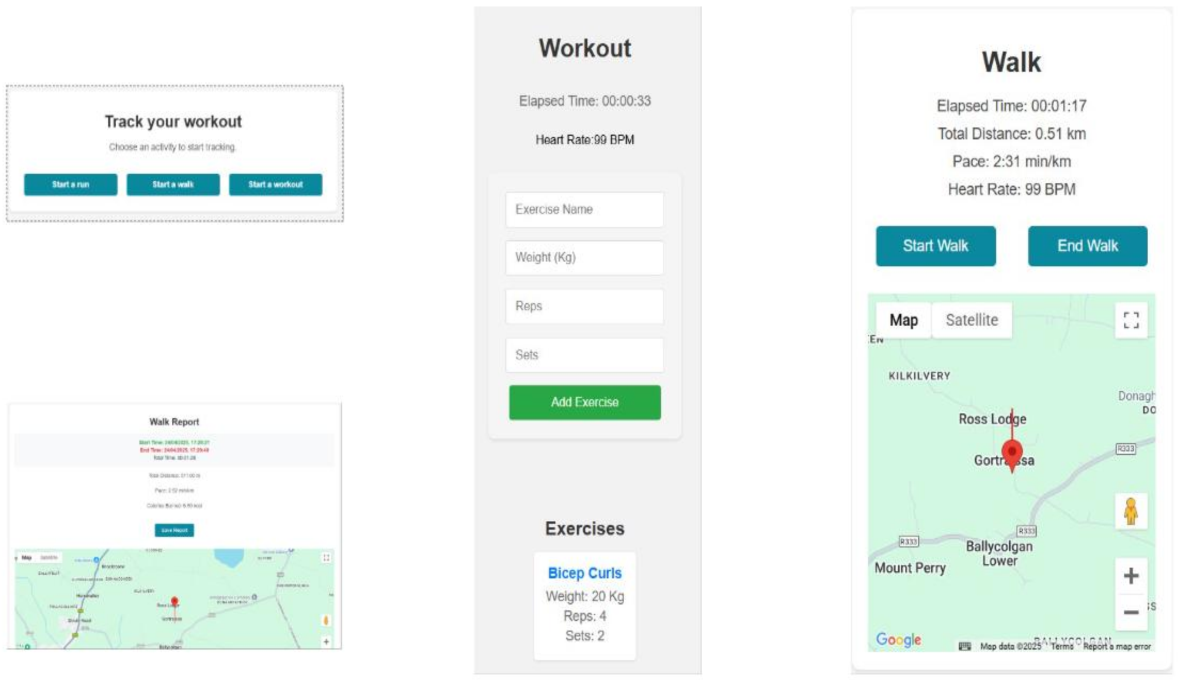


Figure 10-2 Location Page



Start Time: 24/04/2025, 17:28:21
End Time: 24/04/2025, 17:29:49
Total Time: 00:01:28

Total Distance: [illegible]

Pace: 2:52 min/km

Calories Burned: 6.60 kcal

Elapsed Time: 00:00:33

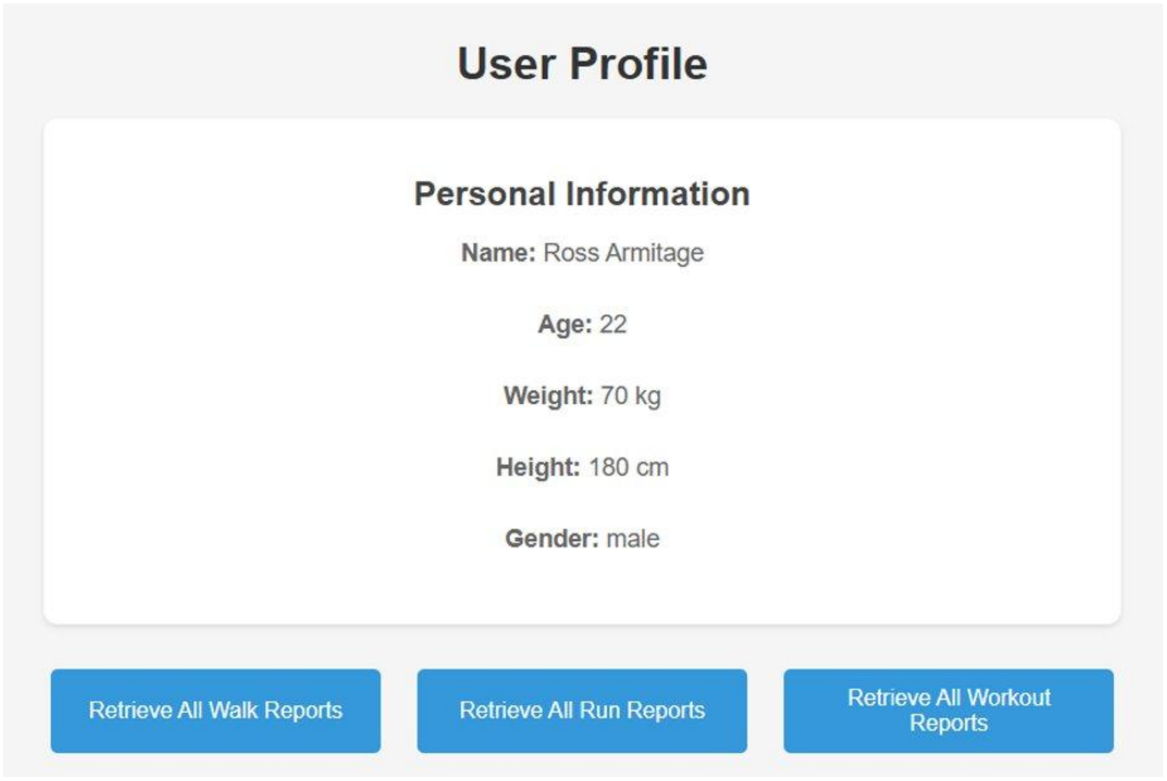
Heart Rate:99 BPM

Bicep Curls
Weight: 20 Kg
Reps: 4
Sets: 2

Elapsed Time: 00:01:17
Total Distance: 0.51 km
Pace: 2:31 min/km
Heart Rate: 99 BPM

Figure 10-3 Activities/Reports

On the software side, I developed a secure web application with account authentication, where users could log in to view their personalized data.



User Profile

Personal Information

Name: Ross Armitage

Age: 22

Weight: 70 kg

Height: 180 cm

Gender: male

Figure 10-4 User Profile Page

A backend and database system stored user information and sensor readings securely, and deployment on AWS [22] ensured scalability and reliable access. Overall, the project helped me grow technically and gave me practical experience in building full-stack systems.

11 Links

GitHub - https://github.com/RxssA/FYPWHAT#

YouTube video - https://youtu.be/vfVL7bx8cq8

12 References

[1] GeeksForGeeks, "How to get the last N records in MongoDB," *GeeksForGeeks*, [Online]. Available: https://www.geeksforgeeks.org/how-to-get-the-last-n-records-in-mongodb/. [Accessed: Apr. 19, 2025].

[2] T. Sookocheff, "How do WebSockets work?" *Sookocheff.com*, [Online]. Available: https://sookocheff.com/post/networking/how-do-websockets-work/. [Accessed: Apr. 19, 2025].

[3] GeeksForGeeks, "What is Web Socket and how it is different from the HTTP?" *GeeksForGeeks*, [Online]. Available: https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/. [Accessed: Apr. 19, 2025].

[4] Adafruit, "Adafruit Ultimate GPS," *Adafruit Learning System*, [Online]. Available: https://learn.adafruit.com/adafruit-ultimate-gps/overview. [Accessed: Apr. 19, 2025].

[5] Mozilla, "Protocol upgrade mechanism," *MDN Web Docs*, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Protocol_upgrade_mechanism. [Accessed: Apr. 19, 2025].

[6] Calculator Ultra, “R-R Interval Heart Rate Calculator,” *Calculator Ultra*, [Online]. Available: https://www.calculatorultra.com/en/tool/r-r-interval-heart-rate-calculator.html#gsc.tab=0. [Accessed: Apr. 19, 2025].

[7] DeepSeek, “DeepSeek AI Models,” *DeepSeek.AI*, 2024. [Online]. Available: https://www.deepseek.com/. [Accessed: Apr. 19, 2025].

[8] Hackster.io, “Getting started with FreeRTOS with ESP32,” *Hackster.io*, Apr. 28, 2023. [Online]. Available: https://www.hackster.io/485440/getting-started-with-freertos-with-esp32-500103. [Accessed: Apr. 19, 2025].

[9] Niall O’Keeffe "Embedded Systems". Lecture, ATU, Galway, ATU 2024

[10] Google Developers, “Overview | Maps JavaScript API | Google for Developers,” *Google Developers*, 2024. [Online]. Available: https://developers.google.com/maps/documentation/javascript/overview. [Accessed: 20-Apr-2025].

[11] Wikipedia contributors, “Moving average,” *Wikipedia, The Free Encyclopedia*, Apr. 18, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Moving_average

[12] FreeRTOS, *Mastering the FreeRTOS Real-Time Kernel*, FreeRTOS.org, 2024. [Online]. Available: https://www.freertos.org/Documentation/RTOS_book.html. [Accessed: Apr. 19, 2025].

[13] Maxim Integrated, *MAX30105 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor*, Datasheet, 2021. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX30105.pdf. [Accessed: Apr. 19, 2025].

[14] Microchip Technology, *MCP9808 High Accuracy I2C Temperature Sensor*, Datasheet, 2020. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/25095A.pdf. [Accessed: Apr. 19, 2025].

[16] Node.js Foundation, *Node.js v20 Documentation*, 2024. [Online]. Available: https://nodejs.org/docs/latest/api/. [Accessed: Apr. 19, 2025].

[17] Express.js, *Express API Reference*, 2024. [Online]. Available: https://expressjs.com/en/api.html. [Accessed: Apr. 19, 2025].

[18] React, *React Documentation*, 2024. [Online]. Available: https://react.dev/learn. [Accessed: Apr. 19, 2025].

[19] IETF, *The WebSocket Protocol (RFC 6455)*, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455. [Accessed: Apr. 19, 2025].

[20] Open-Meteo, *Weather API Documentation*, 2024. [Online]. Available: https://open-meteo.com/en/docs. [Accessed: Apr. 19, 2025].

[21] IETF, *JSON Web Token (JWT) RFC 7519*, 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Accessed: Apr. 19, 2025].

[22] AWS, *AWS IoT Core Developer Guide*, 2024. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html. [Accessed: Apr. 19, 2025].

[23] OWASP, *Authentication Cheat Sheet*, 2024. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. [Accessed: Apr. 19, 2025].