Project Documentation
“Health Monitoring System”

B.TECH in Information Technology
University of Calcutta
By

Kunal Pal	**University Roll:** T91/IT/216006
Indranil Kundu	**University Roll:** T91/IT/216005

Under the supervision
of
Dr.Himadri Nath Saha
HOD Computer Science, SNEC,
University of Calcutta
Kolkata, West Bengal, India

2024

Table of Contents:

1. Introduction

1.1 Purpose:

The purpose of this project is to develop a comprehensive health monitoring system that integrates real-time sensor data from DS18B20 (temperature), MPU-6050 (accelerometer), and MAX 30102 (heart rate) sensors. This data is processed using a Flask API hosted on a server and visualized through a Flutter application. The system aims to provide users with timely health insights and alerts based on machine learning predictions.

1.2 Scope:

The scope of this project includes hardware setup, software architecture, machine learning model development, API integration, and mobile application development. It covers data collection, transmission, processing, and visualization of health parameters in a user-friendly interface.

1.3 Audience:

This documentation is intended for developers, healthcare professionals, and stakeholders interested in understanding the technical implementation, functionality, and deployment of the health monitoring system.

1.4 Overview of Components:

- **Hardware:** DS18B20 Temperature Sensor, MPU-6050 Accelerometer, MAX 30102 Heart Rate Sensor, ESP8266 Microcontroller.
- **Software:** Flask API for backend prediction service, Flutter application for frontend user interface.
- **Integration:** ThingSpeak platform for real-time data storage and retrieval, scikit-learn for machine learning model training (RandomForestClassifier).

2.Hardware Setup

2.1 List of Sensors Used:

DS18B20 Temperature Sensor:

Description: The DS18B20 is a digital temperature sensor that uses the OneWire protocol to communicate with microcontrollers like the ESP8266. It provides accurate temperature measurements with a resolution of up to 12 bits.

Functionality:

- Measures ambient temperature in Celsius.
- Supports multiple DS18B20 sensors connected to a single microcontroller due to its unique address feature.

- Suitable for applications requiring precise temperature monitoring, such as health monitoring systems.

Integration:

- Connected to the ESP8266 microcontroller via GPIO pins using the OneWire communication protocol.
- Periodically reads temperature data and transmits it to the ThingSpeak platform for storage and further processing.

Usage: Provides real-time temperature data crucial for monitoring patient conditions and environmental factors affecting health.

MPU-6050 Accelerometer

Description: The MPU-6050 is a 6-axis accelerometer and gyroscope combination sensor module. It integrates a 3-axis accelerometer and a 3-axis gyroscope in a single chip.

Functionality:

- Measures acceleration along the x, y, and z axes in units of gravitational force (g-force).
- Provides motion detection, gesture recognition, and tilt sensing capabilities.

- Supports advanced motion processing algorithms to detect complex movements.

Integration:

- Connected to the ESP8266 microcontroller via GPIO pins to receive digital signals.
- Reads accelerometer data to monitor patient movements, including sitting, standing, walking, or running.
- Data transmitted to the ThingSpeak platform for storage and processing.

Usage: Enables monitoring of physical activities and movements, essential for assessing patient mobility and health status.

MAX 30102 Heart Rate Sensor

Description: The MAX 30102 is an integrated pulse oximeter and heart rate sensor module. It combines two LEDs, a photodetector, optimized optics, and low-noise analog signal processing to detect heart rate and blood oxygen saturation (SpO2).

Functionality:

- Measures heart rate in beats per minute (BPM) and estimates blood oxygen levels.
- Uses photoplethysmography (PPG) to detect changes in blood volume in microvascular tissue.

- Provides high accuracy and reliability for health monitoring applications.

Integration:

- Connected to the ESP8266 microcontroller via GPIO pins to receive and process analog signals.
- Measures heart rate continuously to monitor cardiovascular health and detect abnormalities.
- Data sent to the ThingSpeak platform for real-time visualization and analysis.

Usage: Critical for monitoring heart rate variations and detecting anomalies, providing early warnings for potential health issues.

ESP8266 Microcontroller

Description: The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability, widely used in IoT applications.

Functionality:

- Acts as the central device in the health monitoring system, collecting data from DS18B20, MPU-6050, and MAX 30102 sensors.
- Supports Wi-Fi connectivity to transmit sensor data to the ThingSpeak cloud platform.
- Executes sensor data collection routines, data preprocessing, and HTTP communication protocols.

Integration:

- GPIO pins are used to interface with DS18B20, MPU-6050, and MAX 30102 sensors.
- Runs firmware that periodically reads sensor data, formats it, and sends it to ThingSpeak via HTTP POST requests.
- Facilitates real-time monitoring and remote access to health data via the internet.

Usage: Central component enabling seamless integration of sensor data into the health monitoring system, ensuring continuous monitoring and timely alerts for healthcare providers and patients.

2.1 Configuration of ESP8266 as Central Device:

The ESP8266 microcontroller serves as the central device in the health monitoring system, responsible for interfacing with multiple sensors and transmitting their data over Wi-Fi to the ThingSpeak platform. Here’s how it is configured:

Data Acquisition from Sensors

DS18B20 Temperature Sensor:

- ➔ **Connection:** Connected to the ESP8266 via the OneWire protocol, utilizing GPIO pins for data communication.

- ➔ **Data Reading:** The ESP8266 periodically queries the DS18B20 sensor to obtain temperature readings in Celsius. It reads the digital signal output from the sensor, converts it to temperature values, and stores it locally.

MPU-6050 Accelerometer:

- ➔ **Connection:** Linked to the ESP8266 through GPIO pins to receive digital signals for accelerometer data.
- ➔ **Data Reading:** The ESP8266 gathers acceleration data along the x, y, and z axes from the MPU-6050 sensor. It interprets these readings to monitor movements such as walking, running, or stationary states based on predefined thresholds.

MAX 30102 Heart Rate Sensor:

- ➔ **Connection:** Interfaced with the ESP8266 via GPIO pins to capture analog signals for heart rate monitoring.
- ➔ **Data Reading:** The ESP8266 continuously samples the MAX 30102 sensor to measure heart rate in beats per minute (BPM). It utilizes photoplethysmography (PPG) to detect variations in blood volume, crucial for real-time monitoring of cardiovascular health.

Wi-Fi Connectivity and Data Transmission

- **Wi-Fi Integration:**

    - The ESP8266 connects to the local Wi-Fi network to enable seamless communication with the ThingSpeak cloud platform.
    - It leverages the Wi-Fi capabilities to establish HTTP connections for transmitting sensor data securely to ThingSpeak servers.

- **Data Uploading to ThingSpeak:**
    - HTTP Requests: Periodically, the ESP8266 formulates HTTP POST requests containing sensor data formatted as JSON.
    - Endpoint Communication: These requests are directed to ThingSpeak's API endpoint, ensuring real-time updates of health parameters such as temperature, accelerometer readings, and heart rate.

- **Data Logging and Storage:**

    - Local Buffering: In case of network disruptions, the ESP8266 employs local data buffering mechanisms to store sensor readings temporarily.
    - Retry Mechanisms: It implements retry mechanisms to resend data packets in case of failed transmission attempts, ensuring data integrity and reliability.

Firmware and Execution

- **Firmware Development:**
    - Developed firmware using Arduino IDE or similar environments, tailored to manage sensor interfaces, data acquisition routines, and Wi-Fi connectivity.
    - Integrated libraries and dependencies for OneWire protocol (DS18B20), I2C communication (MPU-6050), and analog signal processing (MAX 30102) into the firmware.
- **Execution and Control:**
    - Executes periodic sensor data collection tasks based on predefined intervals, ensuring timely updates without overwhelming the network bandwidth.
    - Implements error handling and logging mechanisms to monitor device health, network connectivity, and sensor performance.

3. Software Architecture

3.1 High-Level Overview:

The system architecture consists of three main components:

- **Sensor Interface:** ESP8266 collects sensor data and uploads it to ThingSpeak.
- **Flask API:** Hosted on a server, it fetches real-time sensor data from ThingSpeak, processes it using a

machine learning model, and provides predictions via HTTP endpoints.
- **Flutter Application:** Provides a user interface to visualize sensor data, display health predictions, and receive alerts based on thresholds.

3.2 Integration of Hardware and Software Components:

The ESP8266 communicates with sensors via GPIO pins, fetches data, and transmits it over Wi-Fi to ThingSpeak. The Flask API fetches this data, preprocesses it, and sends it to the machine learning model for predictions. The Flutter app fetches predictions from the Flask API and displays them to the user.

3.3 Communication Protocols Used:

- **HTTP:** Used for communication between ESP8266 and ThingSpeak, Flask API and Flutter app.
- **API:** Flask API provides endpoints for fetching predictions and sensor data.

4. Data Flow

4.1 Data Collection from Sensors:

- **DS18B20:** Measures temperature in Celsius.
- **MPU-6050:** Provides acceleration data in g-force units for x, y, and z axes.

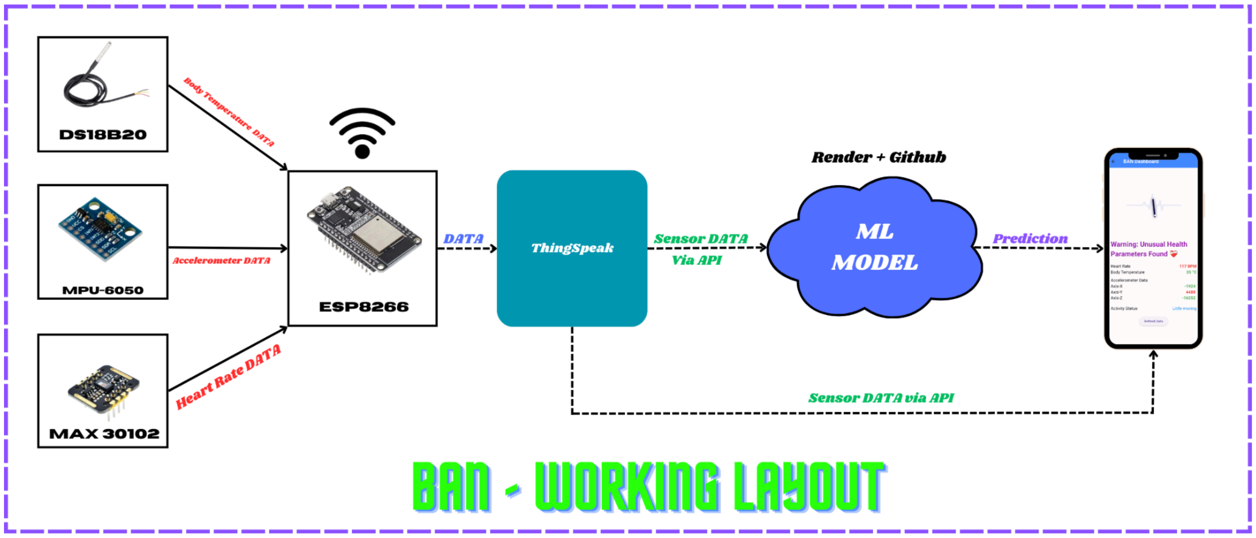
- **MAX 30102:** Measures heart rate in beats per minute (BPM).

4.2 Transmission to ThingSpeak Platform:

- ESP8266 collects sensor data and sends HTTP POST requests to ThingSpeak.
- Data is stored in ThingSpeak channels for real-time visualization and retrieval.

4.3 Integration with Flask API for Predictions:

- Flask API fetches real-time sensor data from ThingSpeak channels.
- Preprocesses data and sends it to a trained machine learning model (RandomForestClassifier).
- Receives predictions and serves them via HTTP endpoints (**"/predict"**).



5. Machine Learning Model

5.1 Dataset Description:

Utilized a collected dataset (**health_data.csv**) containing sensor readings (temperature, accelerometer data, heart rate) and corresponding health conditions (normal, abnormal) for model training.

5.2 Model Training Process:

Algorithm: Employed the RandomForestClassifier algorithm from the Scikit-Learn library for its robust performance in classification tasks involving structured data.

Training: Split the dataset into training and testing sets (80% training, 20% testing) using the train_test_split function. Trained the model on the training set to learn patterns and relationships between features and target labels.

- Split the dataset into training and testing sets using scikit-learn (train_test_split).
- Trained a RandomForestClassifier model on the training data.
- Evaluated model accuracy using accuracy_score.

5.3 Evaluation Metrics:

Accuracy: Measures the proportion of correct predictions out of all predictions made by the model. It is calculated as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Precision: Indicates the ability of the model to correctly identify positive predictions among all positive instances predicted. It is calculated as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall (Sensitivity): Measures the proportion of actual positive instances that were correctly predicted by the model. It is calculated as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

F1 Score: Harmonic mean of precision and recall, providing a balanced measure of the model's performance across both metrics. It is calculated as:

$$F1\,Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The integration of the RandomForestClassifier model into the Flask API allows for real-time prediction of health status based on sensor data collected by the ESP8266 microcontroller. This system provides valuable insights into users' health conditions, facilitating early detection and intervention when abnormalities are detected.

5.4 Model Deployment:

The trained RandomForestClassifier model is deployed using a Flask API hosted on Render for real-time predictions based on sensor data from the ESP8266 microcontroller.

Deployment Steps:

1. **Model Serialization:** The model is serialized into a binary file (model.pkl) using joblib to preserve its state and parameters.

2. **Flask API Development:** A Flask application creates a /predict endpoint to receive sensor data via HTTP requests from ThingSpeak.

3. **Model Loading:** During API startup, model.pkl is loaded into memory, ensuring rapid predictions without reloading for each request.

4. **Prediction:** Incoming sensor data is processed, transformed into a numpy array, and used to predict health status (normal or abnormal) using the loaded model.

5. **Real-time Updates:** Continuous updates ensure users receive timely health status notifications based on new sensor data.

Scalability and Maintenance

Scalability: The architecture supports horizontal scaling on Render's cloud infrastructure to handle increasing user demand and data volume efficiently.

Maintenance: Regular updates and monitoring maintain model performance, dependencies, and data integrity, ensuring reliable health predictions over time.

6. Flask API

6.1 API Structure and Endpoints:

The Flask API serves as the backend for receiving real-time sensor data from the ESP8266 microcontroller, making predictions using a trained

RandomForestClassifier model, and delivering health status updates.

Endpoint Details

Endpoint: “/predict”

Method: GET

Description: This endpoint receives sensor data from ThingSpeak via HTTP GET requests and predicts the health status based on the data received.

Functionality:

- **Data Fetching:** Fetches the latest sensor data (heart rate, body temperature, accelerometer readings) from ThingSpeak.
- **Data Transformation:** Extracts relevant fields (heart rate, body temperature, accelerometer readings) from the fetched data.
- **Feature Preparation:** Constructs a numpy array with the extracted features to feed into the pre-trained RandomForestClassifier model.
- **Prediction:** Uses the loaded model (model.pkl) to predict the health status based on the features.
- **Response:** Returns a JSON response containing the predicted health status (0 for normal, 1 for abnormal).

API Implementation Details

Model Loading:

- The Flask application loads the pre-trained model.pkl file during startup using joblib.
- This ensures that the model is ready to make predictions without the need for repeated loading, optimizing response time.

Dependencies:

- The API relies on Flask for web framework functionality.
- joblib is used to load the serialized model (model.pkl).

Integration with ThingSpeak:

- The API integrates with ThingSpeak via HTTP requests to fetch real-time sensor data.
- ThingSpeak provides the latest sensor readings from DS18B20, MPU-6050, and MAX 30102 sensors connected to the ESP8266 microcontroller.

6.2 Integration with Machine Learning Model:

- Loads the trained model.pkl to predict health conditions based on incoming sensor data.
- Handles data preprocessing (scaling, feature extraction) before making predictions.

6.3 Data Preprocessing for Prediction:

The Flask API preprocesses sensor data fetched from ThingSpeak to prepare it for health status prediction using a trained RandomForestClassifier model. The preprocessing steps include:

Data Extraction: The API fetches the latest sensor data (heart rate, body temperature, accelerometer readings) from ThingSpeak.

Normalization:

- Raw accelerometer readings are normalized to standardize their scale and ensure consistency across different sensors.
- Normalization involves converting raw values into a standard range suitable for machine learning models.

Feature Preparation:

- Extracted sensor readings (heart rate, body temperature, normalized accelerometer data) are organized into a numpy array format.
- Features are aligned and formatted according to the model's input requirements (model.pkl).

Prediction:

- Using the loaded RandomForestClassifier model, the API predicts the health status based on the preprocessed sensor data.
- This preprocessing ensures that the sensor data is transformed into a suitable format for accurate health status predictions by the machine learning model.

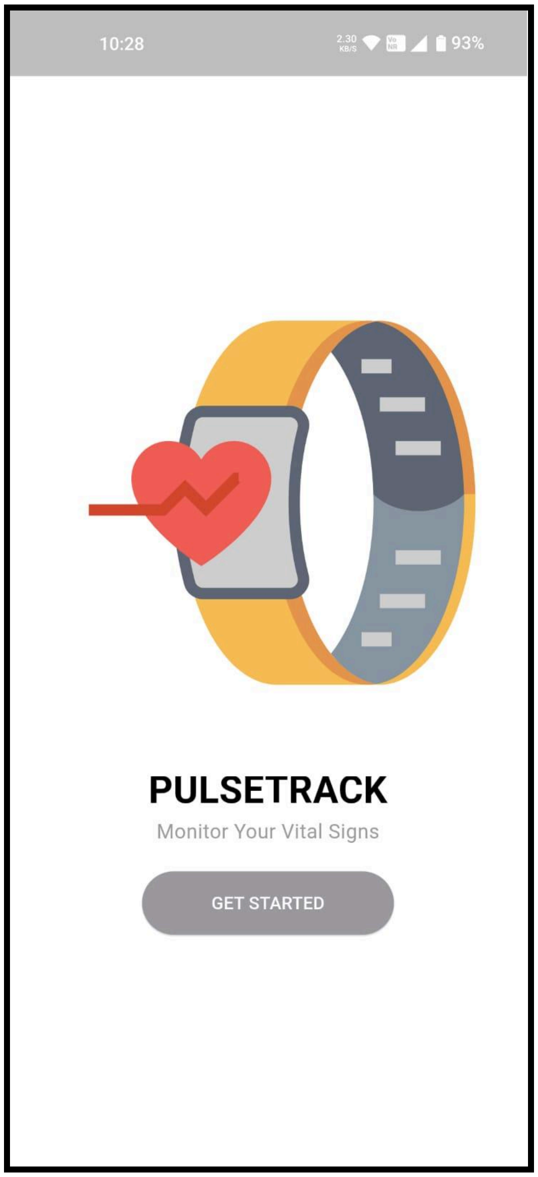
7. Flutter Application

7.1 User Interface Design:

The Flutter application provides a user-friendly interface to monitor real-time health parameters and receive predictive health status updates. Key design elements include:

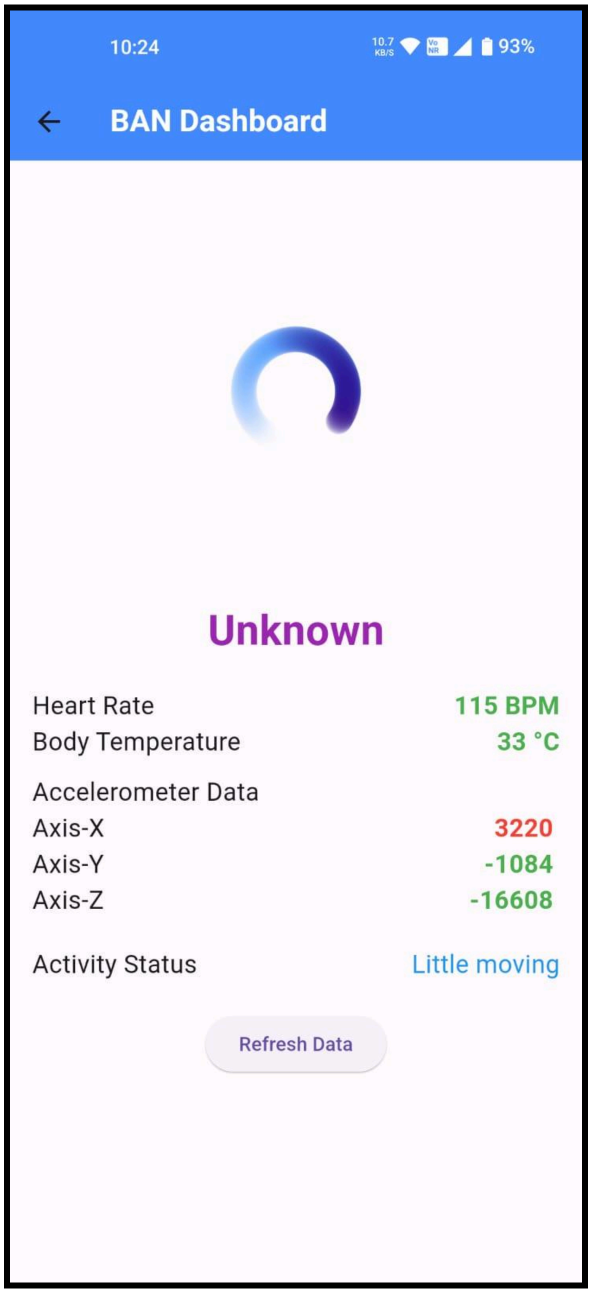
Welcome Screen:

Includes a welcome screen that greets the user.



Dashboard Overview:

Displays current health prediction status, including whether health parameters are within normal limits or if abnormalities are detected.


10:24 10.7 KB/S 93%

BAN Dashboard

Unknown

Heart Rate	115 BPM
Body Temperature	33 °C
Accelerometer Data	
Axis-X	3220
Axis-Y	-1084
Axis-Z	-16608
Activity Status	Little moving

Health Parameters:

- Visual representation of heart rate, body temperature, and accelerometer data.
- Each parameter is color-coded to indicate normalcy or potential health concerns.

Activity Status:

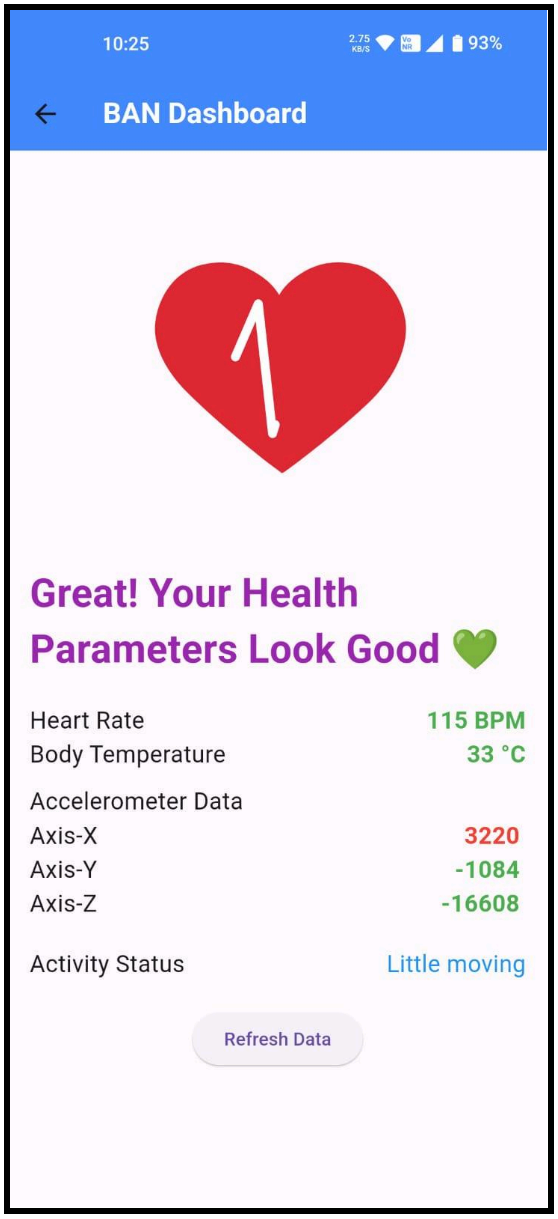
- Provides real-time status updates on user activity based on accelerometer data (e.g., walking, running, standing).
- Updates dynamically as sensor data changes.

Interactive Controls:

- Includes a "Refresh Data" button to manually update sensor readings and prediction status.
- Alerts or notifications are displayed if critical health thresholds are exceeded.

Lottie Animations:

Integrates animated feedback using Lottie for visualizing health status changes dynamically.



BAN Dashboard

Great! Your Health Parameters Look Good 💚

Heart Rate	115 BPM
Body Temperature	33 °C
Accelerometer Data	
Axis-X	3220
Axis-Y	-1084
Axis-Z	-16608
Activity Status	Little moving

7.2 Features and Functionality:

Real-time Health Monitoring: Fetches and displays live sensor data from DS18B20 (temperature), MPU-6050 (accelerometer), and MAX 30102 (heart rate) sensors.

Predictive Health Analysis:

- Utilizes a Flask API endpoint to predict health status based on sensor data fetched from ThingSpeak.
- Displays predictions indicating whether health parameters suggest normalcy or potential health issues.

Dynamic Updates:

- Automatically updates UI components based on new sensor data fetched periodically from ThingSpeak.
- Real-time updates ensure users have current health status information at all times.

Alerts and Notifications:

- Triggers alerts or notifications within the app if critical health thresholds (e.g., high heart rate, abnormal body temperature) are exceeded.
- Provides immediate feedback to users regarding potential health concerns.

7.3 Integration with Flask API:

API Communication:

- ➔ Establishes HTTP requests to the Flask API endpoint to fetch predictive health status based on sensor data.
- ➔ Handles JSON responses from the API to update UI components dynamically.

Data Processing:

- ➔ Preprocesses raw sensor data received from ThingSpeak within the Flutter app before sending it to the Flask API for prediction.
- ➔ Ensures data is formatted correctly to align with the model's input requirements.

Visual Feedback:

- ➔ Integrates Lottie animations in the UI to visually represent health status changes predicted by the Flask API.
- ➔ Animations change dynamically based on whether predictions indicate normal health or potential issues.

8. System Operation

8.1 How to Start and Stop the System:

Start: Power on the ESP8266. Launch the Flask API server,which is deployed in **Render (**www.render.com**)**.

The server is set to auto run when a request is given.Open the Flutter app.

Stop: Power off the ESP8266, stop the Flask server (Server sleeps automatically after a certain time), and close the app.

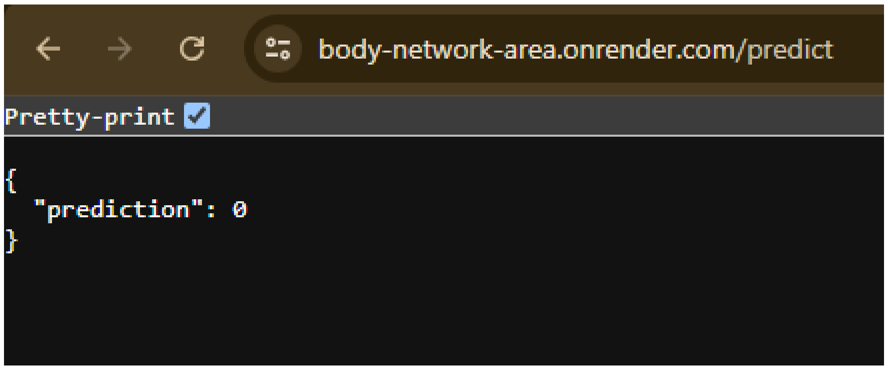
8.2 API Usage Instructions:

Endpoint:
“https://body-network-area.onrender.com/**predict**”

Method: GET

Request: Automatically initiated by the Flutter app.

Response: Contains a health prediction based on the latest sensor data.It’s either 1 or 0.



```
{
  "prediction": 0
}
```

8.3 Interaction with the Flutter App:

The Flutter application serves as the user interface for the BAN Health Monitoring System. It interacts with the Flask API to provide real-time health status updates and visual feedback to the user.

Data Fetching:

- **Periodic Updates:** The app fetches data from the Flask API every 30 seconds. It sends an HTTP GET request to the /predict endpoint of the Flask server. The server responds with the latest prediction based on the current sensor data.
- **Manual Refresh:** Users can manually refresh the data by pressing the 'Refresh Data' button in the app. This triggers immediate data fetching from the Flask API.

Displaying Data:

- **Heart Rate, Body Temperature, and Activity:** The app displays the current heart rate, body temperature, and physical activity status. It color-codes the values based on predefined thresholds (green for normal, red for high).
- **Health Prediction:** Based on the response from the Flask API, the app shows a health status message. If the prediction indicates potential health issues, an alert box is shown to warn the user.

Visual Feedback:

Lottie Animations: The app uses Lottie animations to enhance user experience. Different animations are shown based on the health prediction:

- **Healthy Prediction:** Displays a positive animation to indicate good health.
- **Unhealthy Prediction:** Displays a warning animation to alert the user of potential health issues.

Alerts:

Critical Health Parameters: If the heart rate or body temperature exceeds the defined thresholds, an alert box is triggered. This provides immediate feedback to the user, indicating that their health parameters are critical and may require attention.
Alert Box:

- Displays a message like "Alert: Your heart rate is too high!" or "Alert: Your body temperature is too high!" when thresholds are exceeded.
- Ensures the user is immediately informed of any critical health conditions.

9. Testing

9.1 Unit Testing of Components:

Sensors: Verify that each sensor (DS18B20, MPU-6050, MAX30102) provides accurate readings.

Data Upload: Ensure the ESP8266 correctly uploads data to ThingSpeak.

9.2 Integration Testing:

End-to-End: Test the complete workflow from sensor data collection to health prediction and display on the Flutter app.

9.3 Performance Testing:

Response Time: Measure the time taken for the Flask API to process a request and send a prediction.

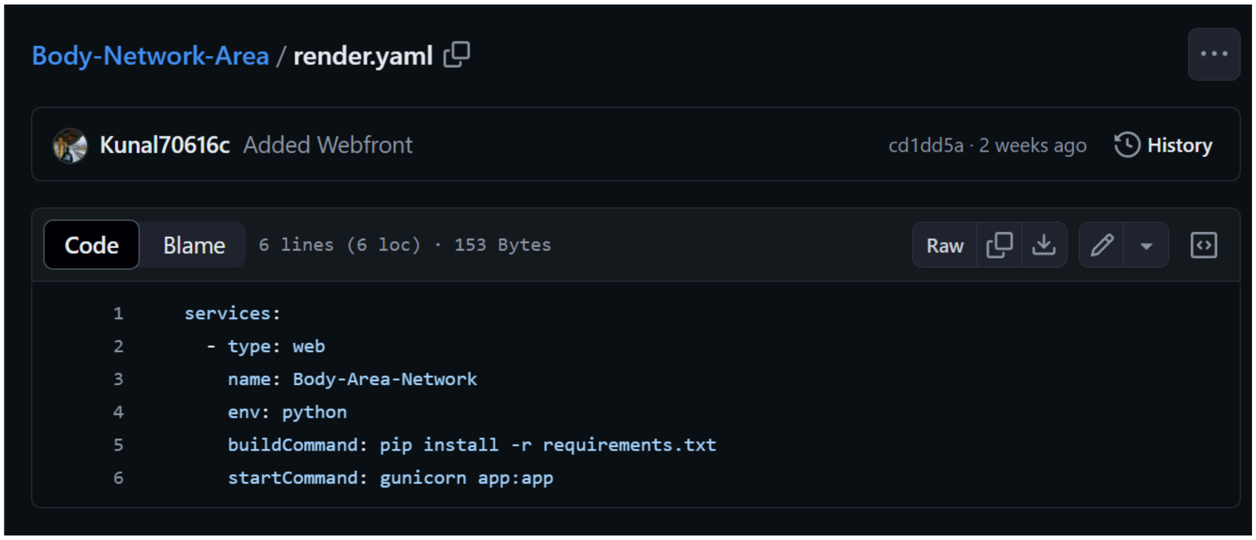
Scalability: Assess the system's performance under different loads to ensure it can handle multiple users.

10. Deployment

10.1 Server Deployment:

Render: Deploy the Flask API on Render for reliable hosting and scalability.Render clones all the files from github repo and runs on its server, and all the information of running the Flask API is stored in **“render.yaml”** file in the github repo.

Here is the content of **"render.yaml"** :


Body-Network-Area / render.yaml

Kunal70616c Added Webfront cd1dd5a · 2 weeks ago History

```yaml
services:
  - type: web
    name: Body-Area-Network
    env: python
    buildCommand: pip install -r requirements.txt
    startCommand: gunicorn app:app
```

The Github Repo used by Render:
https://github.com/Kunal70616c/Body-Network-Area

10.2 Scaling Considerations:

Ensure the deployment can scale to handle increased traffic and user base.

11. Security

11.1 Data Encryption:

HTTPS: Ensure all communications between the Flutter app, Flask API, and ThingSpeak are encrypted using HTTPS to protect data integrity and privacy.

11.2 Best Practices Implemented:

Data Privacy: Adhere to data privacy laws and guidelines to ensure user data is handled securely.
Error Handling: Implement comprehensive error handling to manage exceptions and provide informative error messages.

12. Maintenance and Support

12.1 Regular Maintenance Tasks:

Sensor Calibration: Periodically calibrate the sensors to ensure accurate readings.

API Monitoring: Regularly monitor the Flask API to ensure it is running smoothly and efficiently.

12.2 Troubleshooting Guide:

Connectivity Issues: Verify Wi-Fi connections and API endpoints if data is not being uploaded or fetched correctly.

Sensor Errors: Check sensor connections and ensure they are functioning properly if readings are inaccurate.

13. Appendices

13.1 Additional Resources:

GitHub Repository:
https://github.com/Kunal70616c/Body-Network-Area

Datasets: *health_data.csv*

Libraries Used: sklearn, joblib, Flask, requests, Flutter, Lottie

13.2 Glossary of Terms:

➔ **ESP8266:** A low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability.

➔ **ThingSpeak:** An IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams.

➔ **Lottie:** An open-source animation file format that's lightweight and can be easily integrated into apps.

➔ **RandomForestClassifier:** A machine learning algorithm for classification tasks.