A Midterm Progress Report

On

FITNESS TRACKING INSIGHTS

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

Computer Science & Engineering

SUBMITTED BY

MOHIT KUMAR

2104225

NAVAL THANIK

2104226

SONAL KUMARI

2104232

UNDER THE GUIDANCE OF

DR. KIRAN JYOTI

(FEBRUARY-2024)

Department of Computer Science and Engineering

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Overview

Our proposed project “FITNESS TRACKING INSIGHTS" is an IoT-based project that will leverage the powerful capabilities of the microcontroller, which will work in conjunction with various sensors. It will collect data on the user's physical activity and health, such as heart rate, steps taken, body temperature etc. This data will be transmitted wirelessly to a centralized database.

In addition, the WellWisher app is under development for mobile. The app will provide users with a personalized experience by displaying fitness data collected by the band, enabling users to monitor their progress towards fitness goals, and offering personalized suggestions for improving their fitness levels.

Internet of Things (IoT) technology has revolutionized every field of Human life by making everything automated and intelligent. IoT refers to a network of things which make a self-configuring network. IoT has the potential to transform various industries by optimizing operations and generating valuable insights from data. It can improve our lives by creating more efficient and sustainable systems, enhancing safety and security, and promoting health and well-being.

1.2 Technical Terms Regarding the Project

Micro-Controller

A microcontroller is a small computer on a single integrated circuit chip that is designed to control specific devices and perform specific tasks. It typically consists of a central processing unit (CPU), memory, and input/output (I/O) ports.

Sensor: Sensors are pieces of hardware that detect changes in an environment and collect data. They work to bridge the digital world to the physical world.

Sensor fusion: This is a technique for combining data from multiple sensors to improve the accuracy and reliability of the data. Sensor fusion algorithms will integrate data from various sensors to provide a more complete picture of the user's physical activity and health.

Cross-platform development frameworks

They are software development tools that allow developers to create mobile applications that can run on multiple platforms such as iOS and Android. They enable us to write code once and deploy it on multiple platforms, which saves time and resources.

APIs: APIs are used to enable mobile applications to communicate with back-end servers and other services. APIs can provide access to data, functionality, and services that are not available on the device itself.

1.3 Objective

- To Develop a functional prototype of a wearable band, integrating essential sensors to ensure accurate health data of the user.
- To develop a dedicated mobile application, With enhanced user experience with diverse profile views & improved dashboard.
- To obtain the reliability and precision of the hardware device with benchmarking.

1.4 Objectives Overview

Develop a Functional Wearable Prototype

To design and fabricate a functional prototype of a wearable band is our one of objectives. This wearable device will integrate essential sensors capable of capturing accurate health data of the user. By incorporating advanced sensor technology, the prototype aims to provide reliable and real-time insights into various health metrics such as heart rate, activity level, sleep patterns, and more. The development process will prioritize the seamless integration of sensors to ensure optimal performance and user experience.

Create a Dedicated Mobile Application

The primary objective of the project is to develop a dedicated mobile application to complement the wearable device. The mobile app will serve as a user-friendly interface for accessing and interpreting the health data collected by the wearable band. With enhanced user experience features including diverse profile views and an improved dashboard, the app aims to provide users with a comprehensive overview of their health and fitness metrics. Additionally, the application will offer personalized recommendations and actionable insights to help users make informed decisions about their well-being.

Ensure Reliability and Precision

A critical aspect of the project involves assessing the reliability and precision of the hardware device through benchmarking. This objective entails conducting rigorous testing and validation procedures to evaluate the accuracy and consistency of the wearable prototype's health-monitoring capabilities. By comparing the performance of the device against established benchmarks and industry standards, the project aims to ensure that users can rely on the accuracy of the health data provided by the wearable device. This objective underscores the project's commitment to delivering a high-quality product that meets the needs and expectations of its users.

By developing a functional wearable prototype and a dedicated mobile application, the project endeavours to provide users with actionable insights and personalized recommendations to support their wellness goals. Furthermore, the emphasis on ensuring the reliability and precision of the hardware device underscores the project's commitment to delivering accurate and dependable health data. Through these objectives, the project seeks to empower individuals to take proactive steps towards optimizing their health and well-being.

1.5 Progress Abstract

As we approach February 2024, it's evident that participants, including myself, have made significant strides in technical proficiency, problem-solving, and adaptability. The ongoing project work reflects a synthesis of conceptual understanding and rapid development of the application, setting a strong foundation for the remaining duration.

The collaborative aspects of the program have not only enhanced technical skills but have also fostered effective teamwork, communication, and the ability to contribute meaningfully to shared objectives. The exposure to embedded hardware has provided valuable insights into the real-world implications of our work in IoT.

Looking ahead, the program continues to be a catalyst for individual growth, innovation, and the creation of impactful IoT solutions. The next phase promises deeper immersion, more complex projects, and continued exploration into the fascinating world of IoT technology.

CHAPTER 2: SYSTEM REQUIREMENT

2.1 Software Requirement

2.1.1 VS Code IDE

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

This editor stands out for its adaptability and extensibility. Users can open and save multiple directories, essentially functioning as a language-agnostic code editor. Its support spans various programming languages, tailoring features for each. Extensions from the VS Code Marketplace enhance the editor's functionality and language support. Project management is streamlined through workspaces, allowing users to save and reuse directories, excluding unwanted files and folders.

The platform's extensibility is further emphasized by its rich extension ecosystem. It supports popular version control systems like Git, Apache Subversion, and Perforce, facilitating repository creation and direct push/pull requests

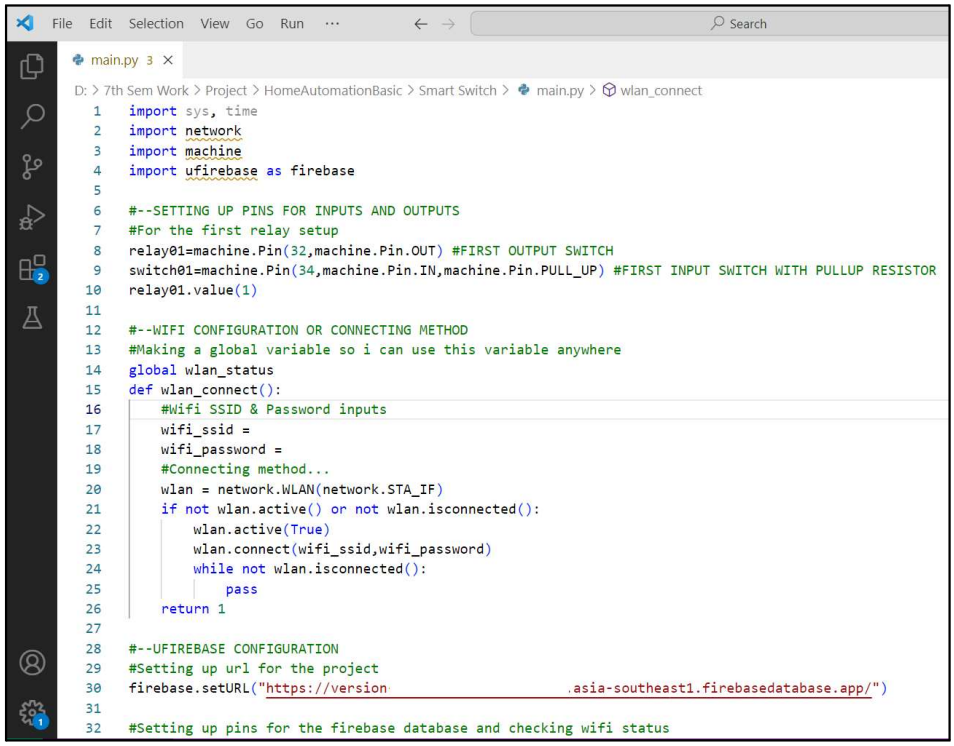

Fig. 2.1.1 VS Code editor view

2.1.2 Thonny IDE

Thonny IDE boasts an array of features that elevate the debugging experience. With the ability to step through code without breakpoints and observe live variables during debugging, users gain a dynamic perspective on their code's execution. This real-time insight is complemented by the option to step through expression evaluations, allowing users to witness the transformation of expressions into their corresponding values. Thonny also offers a simple way to install and manage Python packages, making it easy for users to work with third-party libraries. Thonny includes features such as syntax highlighting, auto-completion, debugging, and a built-in Python interpreter

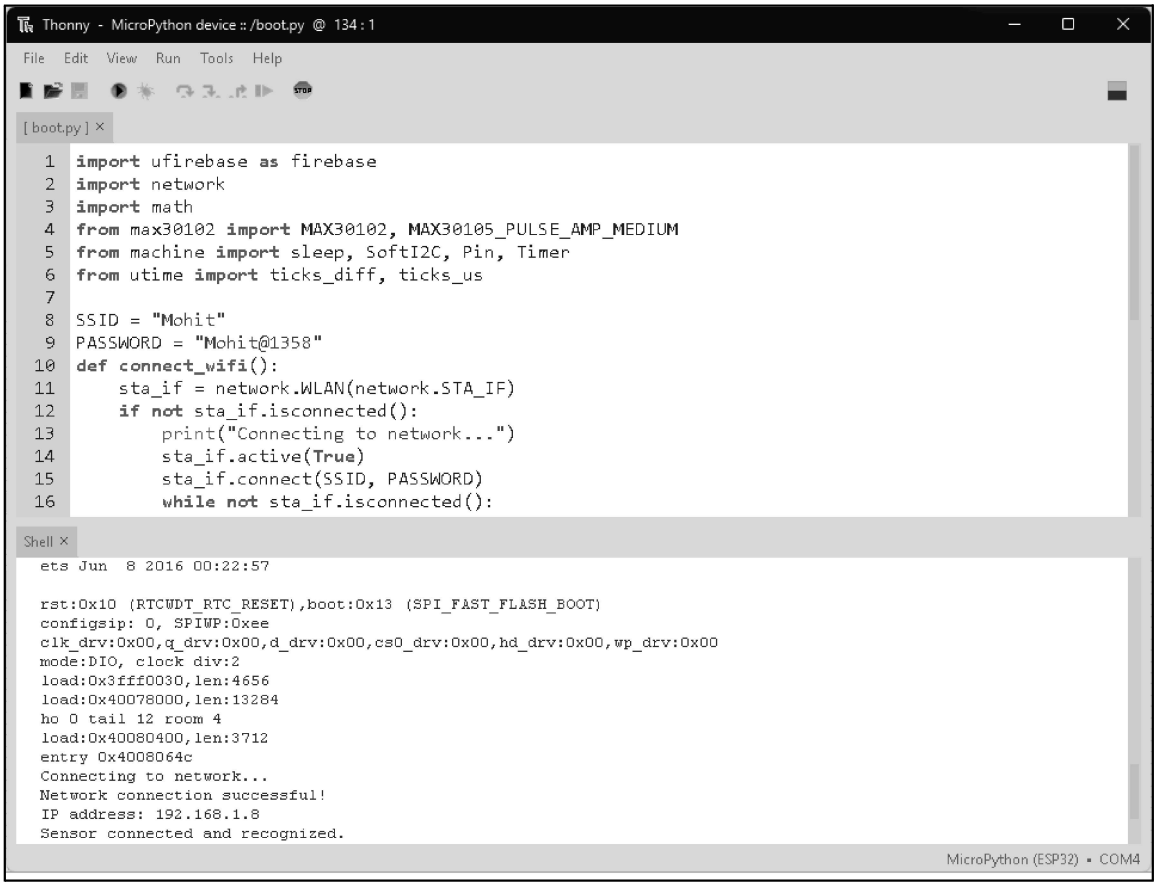

Fig 2.1.2 Thonny IDE Editor

The debugging interface in Thonny incorporates separate windows dedicated to executing function calls, elucidating local variables, and presenting the call stack. This modular approach enhances the comprehensibility of the debugging process, providing a clear and organized view of code execution. Thonny's debugging capabilities extend beyond local environments, offering support for both CPython and MicroPython. This versatility ensures compatibility with different Python implementations, catering to a broad spectrum of developers.

2.1.3 Android Studio

Android Studio, the official IDE for Android app development, builds upon IntelliJ IDEA's robust foundation. Offering a flexible Gradle-based build system, it provides a swift emulator and a unified environment for universal Android device development. Live Edit facilitates real-time updates, while code templates and GitHub integration expedite feature building. Extensive testing tools, lint tools, and C++/NDK support ensure app robustness.

The IDE's support extends to Google Cloud Platform integration, simplifying Google Cloud Messaging and App Engine integration. Beyond points, Android Studio encompasses project structure management, Gradle build system customisation, build variant handling and multiple APK support. Features like resource shrinking, dependency management, debug tools, and performance profilers enhance the development experience. Inline debugging, heap dumps, memory profilers, and data file access further contribute to Android Studio's comprehensive toolset. The IDE's continuous evolution is highlighted in the Android Studio release notes, providing developers with insights into the latest changes and improvements.

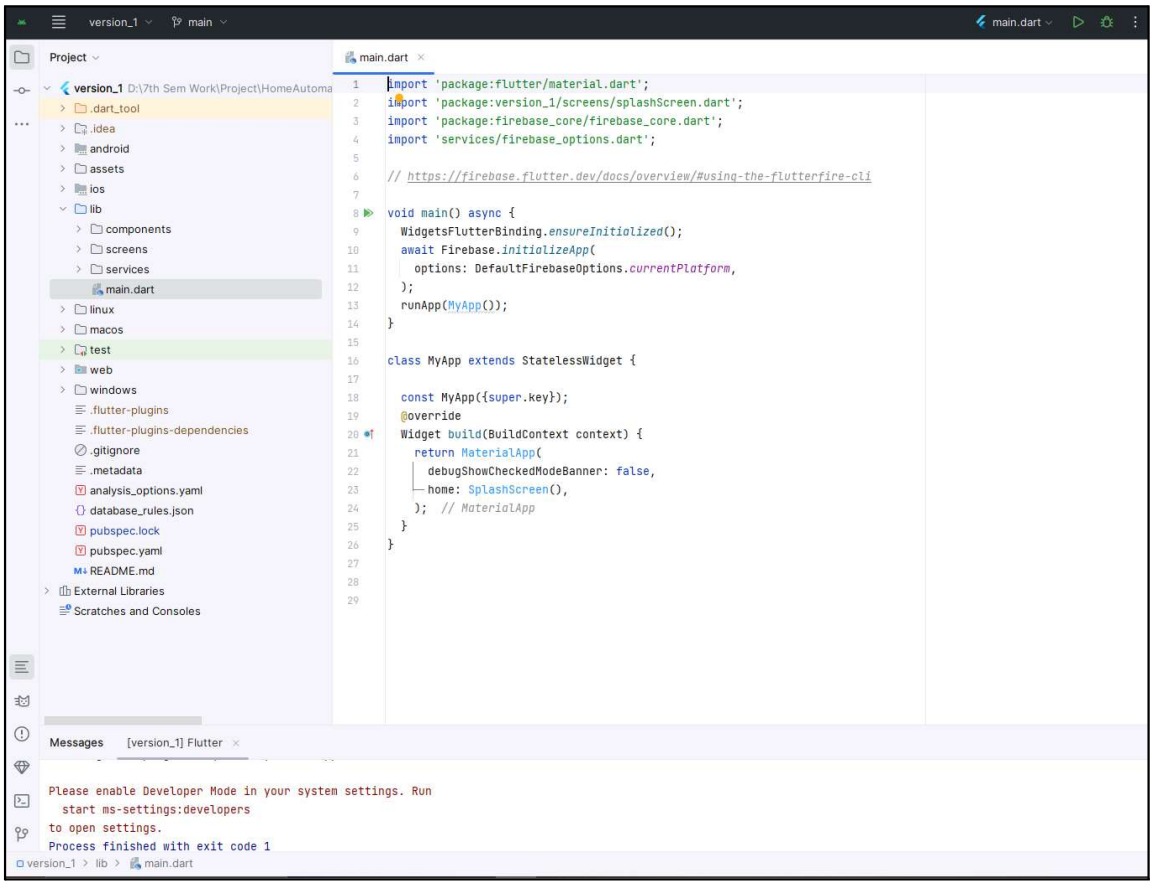

Fig. 2.1.3 Android Studio editor view

2.2 Programming Languages & Frameworks

2.2.1 MicroPython Language

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments. MicroPython is packed full of advanced features such as an interactive prompt, arbitrary precision integers, closures, list comprehension, generators, exception handling and more.MicroPython aims to be as compatible with normal Python as possible to allow you to transfer code with ease from the desktop to a microcontroller or embedded system.

Fig 2.2.1 MicroPython Language Logo

Key Features:

- Seamless code transfer between desktop and microcontroller environments with high compatibility with standard Python.
- Enhanced user interface with an interactive prompt (REPL) supporting history, tab completion, auto-indent, and paste mode.
- Open-source on GitHub with the entire core available under the MIT license, encouraging contributions for various uses.
- Highly configurable with support for multiple architectures, a configurable garbage collector, fast start-up time, and robust memory error handling.
- ESP32 integration replaces the MicroPython pyboard, offering a versatile platform for efficient embedded programming with low-level hardware access.

MicroPython exhibits seamless compatibility with standard Python (CPython), ensuring smooth code transfer between desktop and microcontroller environments. The interactive prompt (REPL) enhances user experience, enabling immediate command execution with features like history, tab completion, auto-indent, and paste mode. As an open-source project on GitHub, MicroPython embraces the MIT license, encouraging contributions and adaptations for personal, educational,

and commercial purposes. Leveraging advanced coding techniques, it remains highly configurable, supporting multiple architectures with features such as a configurable garbage collector, fast start-up time, memory error handling, and the ability to run Python code on hard interrupts. In the context of your training, the MicroPython integration with ESP32 replaces the pyboard, offering a versatile platform for efficient embedded programming. The ESP32's machine module facilitates low-level hardware access, making it a robust choice for IoT and embedded systems development.

Programming Example of MicroPython Language:

MicroPython makes sensor integration straightforward. For example, reading data from a digital temperature sensor (DS18B20):

```
import onewire
import ds18x20
import machine
import time

# Pin configuration
pin = machine.Pin(4)  # Use GPIO 4 for the sensor

# Create a one-wire bus object
ow = onewire.OneWire(pin)

# Create a DS18B20 sensor object
temp_sensor = ds18x20.DS18X20(ow)

# Scan for devices on the bus
devices = temp_sensor.scan()

while True:
    # Read temperature from the sensor
    temp_sensor.convert_temp()
    time.sleep_ms(750)

    # Wait for conversion
    temperature = temp_sensor.read_temp(devices[0])
    print("Temperature:", temperature, "C")
    time.sleep(5)
```

This code snippet initializes the LED pin, and in a loop, it toggles the LED state every second, creating a blinking effect.

MicroPython on ESP32 provides a powerful yet accessible platform for IoT development, combining the simplicity of Python with the flexibility of microcontroller programming.

2.2.2 Dart Language

Dart, a finely tuned language designed for clients, emerges as a standout choice for the dynamic landscape of swift multi-platform app development. Its prowess lies in the seamless transition it orchestrates—from the fluidity of desktop environments to the mobility demands of various devices. At the epicentre of this versatility is its role as the driving force behind my Flutter app, where it not only meets but exceeds expectations in delivering a cohesive and engaging experience across a spectrum of platforms.

Fig 2.2.2 Dart Language Logo

In essence, Dart becomes the unifying thread that threads through the diverse fabric of modern app development, providing a robust and adaptable foundation for my projects.

- Language Efficiency: Dart is type-safe with static checks and optional annotations, ensuring reliability. Sound null safety is built-in, guarding against null exceptions.
- Platform Diversity: Dart supports both native and web platforms. For mobile and desktop apps, there's native platform support with JIT and AOT compilation. On the web, Dart compiles to JavaScript, providing compatibility.
- Flutter Integration: Dart powers the UI toolkit of my Flutter app, seamlessly extending support across iOS, Android, macOS, Windows, Linux, and the web.
- Development Agility: Dart Native ensures a swift developer cycle during development with JIT compilation, supporting hot reload. For production, AOT compilation guarantees consistent, short startup times.
- Robust Runtime: The Dart runtime, embedded in self-contained executables for native platforms, efficiently manages memory, enforces type system rules, and controls isolates, providing a stable foundation for my Flutter app.

```
void main() {
  // Print a greeting
  print('Hello, Dart!');

  // Declare variables
  int number = 42;
  String message = 'Dart is awesome';

  // Perform a simple calculation
  int result = addNumbers(20, 22);

  // Conditionally print the result
  if (result > 40) {
    print('The result is greater than 40.');
  } else {

    print('The result is not greater than 40.');
  }
}
// Function to add two numbers

int addNumbers(int a, int b) {
  return a + b;
}
```

The Dart code prints a greeting, declares variables ('number' and 'message'), calculates using a function ('addNumbers') and employs a conditional statement for result-based message printing—illustrating fundamental programming concepts.

2.2.3 Flutter Framework

In the dynamic landscape of cross-platform mobile app development, our project has been propelled by the innovative Flutter framework, a creation of Google that has redefined the paradigm of building natively compiled applications. As an active participant in this transformative journey, I've harnessed the power of Flutter to craft visually stunning, high-performance apps for both iOS and Android platforms.

Fig 2.2.3 Flutter Framework Logo.

2.2.3 (a) Key Characteristics of Flutter:

- Dart Programming Language: At the core of Flutter is the Dart programming language, designed for ease of use, productivity, and optimal performance. My hands-on experience with Dart's succinct syntax has enabled me to express complex concepts with clarity and precision.
- Widget-Centric Development: Flutter's adoption of a widget-centric architecture treats everything as a widget, from structural elements to interactive components. This modular approach, which I actively utilized, enhances code reusability, facilitates maintenance, and allows for the creation of intricate UI structures.
- Consistent Cross-Platform Experience: Flutter ensures a consistent user experience across different platforms, maintaining a uniform look and feel on both iOS and Android devices. My engagement with the framework has showcased its ability to achieve cross-platform consistency, reducing the need for platform-specific adaptations.
- Rich Set of Customizable Widgets: These are easily customizable widgets that have allowed me to align the Flutter mobile app with specific design requirements, facilitating the creation of visually appealing and responsive interfaces.

2.2.4 Firebase

Firebase is a mobile app and web development platform created by Google. It is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a real-time database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files. Firebase provides an all-in-one solution for developers, including data storage and synchronization, user authentication, analytics and reporting.

Fig 2.2.4.a Firebase Logo

Firebase also supports authentication with email/password credentials and API authentication protocols from popular social media platforms like Google and Facebook. By using Firebase, we can quickly build powerful mobile and temporary web applications with features like real-time updates, data synchronisation across devices or servers, file storage services, and more. The platform allows them to focus on building the app instead of worrying about backend operations or infrastructure.

Realtime Database

It's a core component of the Firebase platform that revolutionises real-time data storage and synchronisation for web and mobile applications. Developed by Google, Firebase Realtime Database offers a NoSQL, cloud-hosted database that enables developers to build responsive and collaborative applications with minimal effort.

- Firebase Remote Config stores developer-specified key-value pairs to change the behaviour and appearance of your app without requiring users to download an update.

In the Realtime Database, the data is structured in a JSON-like format. Here's an example to illustrate the structure:

```
{
  "users": {
    "email": "john@example.com",
    "age": 30
  },
  "posts": {
    "title": "Introduction to Firebase",
    "author_id": "user_id_1"
  }
}
```

In this example:

- There is a "users" node containing user profiles, each identified by a unique user ID.
- Similarly, there is a "posts" node containing post information, each identified by a unique post ID.

The "author_id" in the "posts" node establishes a relationship with the "users" node, linking each post to its author.

The Firebase Realtime Database empowers you to create collaborative applications with secure client-side access. It ensures data persistence, even offline, and syncs local changes with remote updates upon reconnection. With Firebase Authentication integration, developers can precisely control data access.

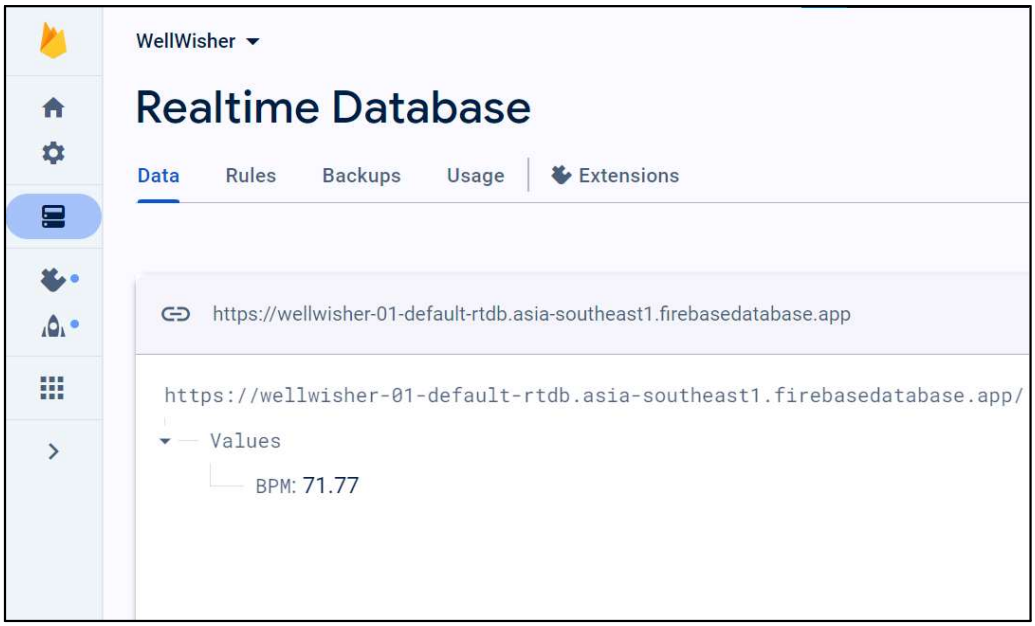

Fig. 2.2.4.b Firebase Realtime Database View

Featuring Firebase Realtime Database Security Rules, this NoSQL database offers flexibility in defining data structure and access permissions. The API prioritises quick operations, enabling a responsive real-time experience for millions of users. It's crucial to structure data thoughtfully based on user needs.

Cloud Firestore

Cloud Firestore is a flexible as well as scalable NoSQL cloud database. It is used to store and sync data for client and server-side development. It is used for mobile, web, and server development from Google Cloud Platform and Firebase. Like the Firebase Real-time Database, it keeps syncing our data via real-time listeners to the client app. It provides offline support for mobile and web so we can create responsive apps that work regardless of network latency or Internet connectivity.

Cloud Firestore also provides seamless integration with Google Cloud Platform products and other Firebase, including cloud functions.

2.3 Hardware Requirement

2.3.1 ESP32 board

The ESP32 stands out as a robust and versatile microcontroller widely embraced in the realm of embedded systems and IoT. Built on the Xtensa LX6 dual-core processor, it offers efficient multitasking and optimal performance. Noteworthy is its integrated Wi-Fi and Bluetooth connectivity, a key feature lending itself well to applications demanding wireless communication.

This microcontroller finds a sweet spot in IoT and home automation projects. With its inherent ability to connect to the internet, the ESP32 becomes a linchpin in devices communicating with each other or interfacing with cloud services. In the domain of home automation, it plays a pivotal role in powering smart lights, thermostats, and security systems, courtesy of its connectivity and processing prowess.

Programming the ESP32 is commonly done through the Arduino IDE or PlatformIO with the Arduino framework, catering to a broad spectrum of developers. The ESP-IDF provides a more sophisticated environment for those seeking advanced features and functionalities.

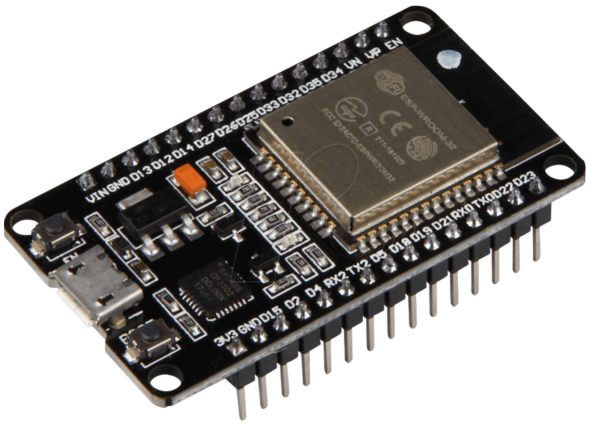

Fig 2.3.1 ESP WROOM 32 MCU Module Version: 1.1.

The ESP32 also boasts an array of GPIO pins and supports various peripherals like SPI, I2C, and UART, enhancing its versatility for interfacing with sensors, displays, and external components. Espressif, the driving force behind the ESP32, champions an open-source ecosystem. The ESP-IDF and related tools being open-source foster collaboration and innovation within the development community.

2.3.2 Sensors

MAX30102

The MAX30102 sensor is a key component of the project's hardware requirements. This sensor is specifically designed for pulse oximetry and heart-rate monitoring applications, making it ideal for capturing vital health data such as heart rates. With its integrated red and infrared LEDs, the MAX30102 sensor enables non-invasive measurement of physiological parameters, contributing to the accurate monitoring of user health. Its compact size and sensor offers programmable sample rates and LED current settings, allowing for optimized power usage and further enhancing energy efficiency. With a low-power heart-rate monitor consuming less than 1mW, the MAX30102 sensor minimizes energy consumption while delivering accurate heart-rate measurements. Additionally, its ultra-low shutdown current of 0.7µA (typical) ensures minimal power draw during inactive periods, contributing to overall power savings. The sensor's fast data output capability and high sample rates enable real-time data acquisition, facilitating rapid response to changes in health parameters.

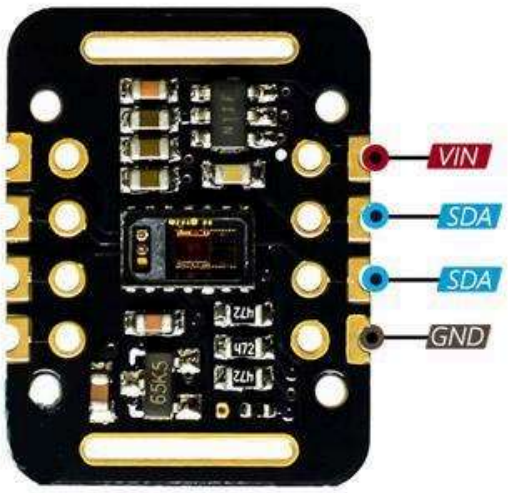

Fig 2.3.2 (a) MAX30102

The red LED emits light at a wavelength of 660nm, and the infrared LED emits light at a wavelength of 880nm. The photodetector measures the intensity of the reflected light from the user's fingertip, which varies with the volume of blood in the fingertip. The module then uses advanced algorithms to process the signals and determine the user's heart rate and blood oxygen saturation (SpO2) levels. It communicates with the ESP32 microcontroller through the I2C interface and provides reliable and accurate heart rate and SpO2 measurements. The MAX30102 sensor is widely used in wearable health and fitness applications and medical devices.

MPU9250

The MPU9250 is a 9-axis Motion Tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer in a single compact package. It is a vital component in the hardware requirements of our project, primarily for its essential role in step tracking. Its integration is indispensable as it provides crucial data on motion and orientation, enabling accurate step count calculations. This sensor's gyroscope features offer precise measurement of angular motion, ensuring accurate tracking of rotation and orientation changes. Likewise, its accelerometer features detect and measure linear acceleration, which directly translates into precise step count data.

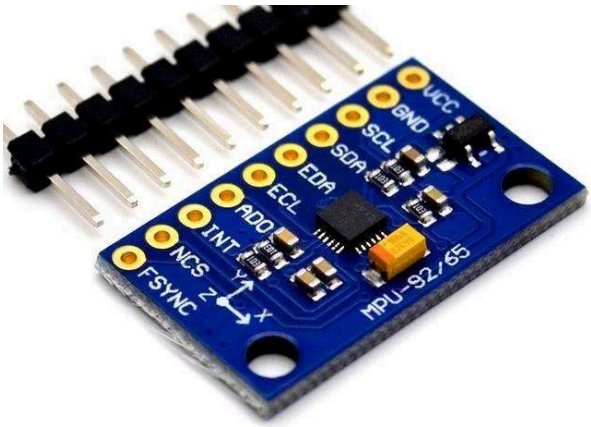

Fig. 2.3.2 (b) MPU9250

The accelerometer measures acceleration along the X, Y, and Z axes, allowing it to detect changes in motion and orientation. The gyroscope measures angular velocity around the same axes, allowing it to detect rotational motion and orientation changes. Finally, the magnetometer measures magnetic fields along these axes, which can be used to determine the orientation of the device relative to the Earth's magnetic field. Overall, the MPU9250 sensor's combination of gyroscope, accelerometer, and motion processing features makes it indispensable for our project. Its contribution ensures the accuracy and reliability of step tracking, which is integral to our fitness monitoring solution.

Moreover, the MPU9250's motion processing capabilities are invaluable, by performing complex calculations onboard which reduces the computational load on the main microcontroller.

The MPU9250 communicates with the ESP32 microcontroller using the I2C protocol and provides raw sensor data that can be processed and analysed to extract useful information about the device's motion and orientation.

2.3.3 Connecting wires

Jumper wires are electrical wires with connector pins at each end. They are used to connect two points in a circuit without soldering. Jumper wires are commonly used with breadboards and other prototyping tools like Arduino. They make changing circuits as simple as possible and come in a wide array of colours, but the colours do not mean anything.

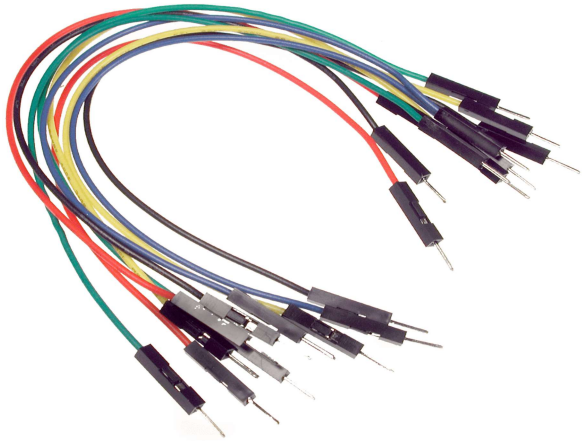

Fig 2.3.3 Jumper Wires (male-to-male jumper).

The wire colour is just an aid to help you keep track of what is connected to which. Jumper wires come in three versions: male-to-male jumper, male-to-female jumper, and female-to-female jumper, and two types of head shapes: square head and round head. The difference between each is in the endpoint of the wire. Male ends have a pin protruding and can plug into things, while female ends do not but are also used for plugging.

2.3.4 BreadBoard

A breadboard is a board used for testing or building circuits. It allows you to place components and connections on the board to make circuits without soldering. The holes in the breadboard take care of your connections by physically holding onto parts or wires where you put them and electrically connecting them inside the board. Breadboards are reusable and do not require soldering or destruction of tracks. They are great for hobbyists and tinkerers to set up projects as a standalone device or as a peripheral to an Arduino, Raspberry Pi, LaunchPad, BeagleBone, and many other development boards.

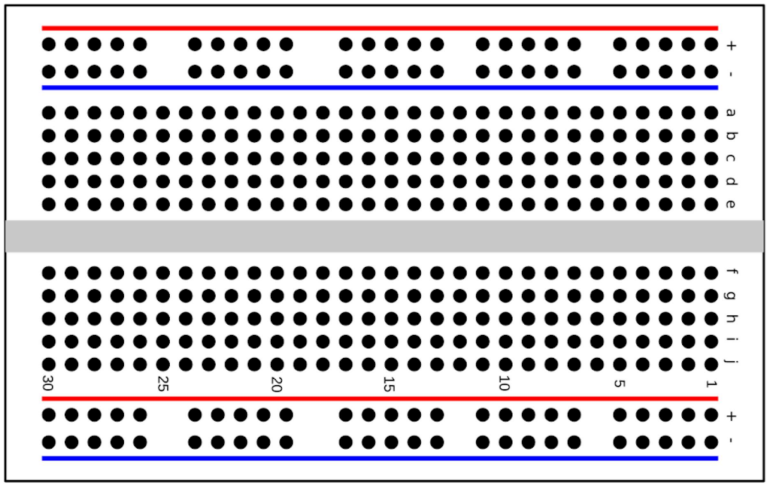

Fig 2.3.4 Breadboard.

Breadboard circuits are also not ideal for long-term use like circuits built on perfboard (protoboard) or PCB (printed circuit board). Still, they also don't have the soldering (protoboard), or design and manufacturing costs (PCBs).

2.4 Additional Hardware Requirement

This Hardware is required to achieve our secondary goal which is to build a wearable prototype for a band which works wirelessly. These components collectively contribute to the project's functionality, reliability, and usability, making them essential for successful post-midterm implementation.

2.4.1 Single-Cell Rechargeable Battery

The single cell Rechargeable LiPo Battery is an essential component in our hardware requirements, crucial for powering various components such as sensors and the ESP32 microcontroller. Its compact size and lightweight design make it ideal for integration into our wearable prototype, ensuring portability and convenience for users.

Fig 2.4.1 Single-Cell Rechargeable Battery

With a voltage rating of 3.7V and a capacity of 950mAh, this battery provides sufficient power to drive the operational needs of our device. The discharge rate of 0.2C ensures stable and consistent power delivery, while its lightweight construction at 16g and compact dimensions of 35 x 25 x 10mm contribute to the overall ergonomic design of our wearable device.

In summary, the single-cell Rechargeable LiPo Battery is instrumental in providing the necessary power for our project, enabling the seamless operation of sensors and microcontrollers while maintaining portability and user-friendliness.

2.4.2 TP4056 1A

The TP4056 1A Li-Ion Battery Charging Board Micro USB is an essential component in our hardware requirements, serving the critical function of recharging the single-cell Rechargeable LiPo Battery and powering various components such as sensors and the ESP32 microcontroller. Integrated onto the same PCB as the ESP32 for our wearable prototype, it ensures portability, wireless operation, and convenience for users.

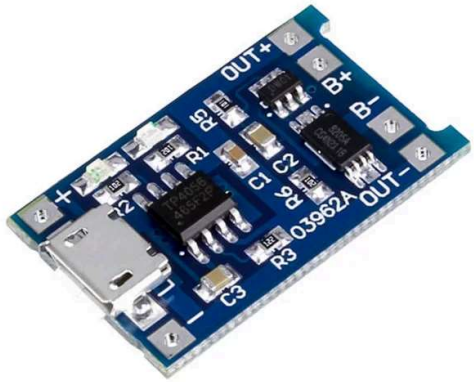

Fig 2.4.2 TP4056 1A

With compact dimensions of 25 x 19 x 10mm, the TP4056 board seamlessly integrates into our device. Its micro USB input interface allows for easy charging, while the built-in current protection ensures safe operation during the charging process. The charging mode is linear, providing stable and reliable charging performance, with an adjustable current of 1A to suit different battery capacities.

The TP4056 board offers precise charging capabilities, with a charge precision of 1.5% to ensure optimal battery health and longevity. It operates within an input voltage range of 4.5V-5.5V and charges the battery to a full voltage of 4.2V, providing efficient and effective charging for our wearable device.

In summary, the TP4056 1A Li-Ion Battery Charging Board Micro USB is indispensable for recharging the battery and powering our project's components, contributing to the overall functionality and usability of our wearable prototype.

2.4.3 Adapter Breakout Board

The Adapter Breakout Board for ESP-32f ESP32 ESP-Wroom-32 Wireless Bluetooth Module is a crucial component in our hardware requirements, enabling us to pre-program the ESP32 microcontroller before integrating it onto the PCB. This pre-programming stage ensures seamless operation with other components such as sensors, facilitating efficient data collection and processing for our wearable prototype.

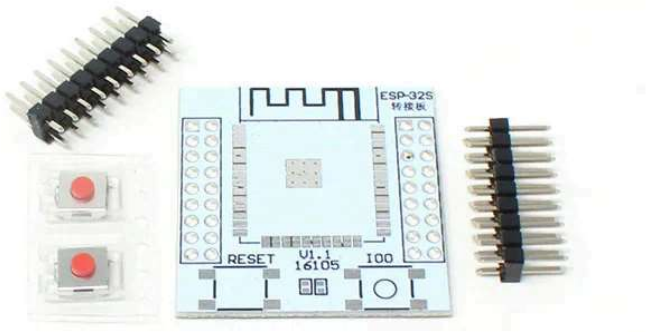

Fig. 2.4.3 Adapter Breakout Board

Featuring an onboard Reset button and flashing firmware button, the adapter board simplifies the programming process, allowing for easy firmware updates and troubleshooting. Additionally, it includes male double-row pins connectors (not soldered), providing flexibility in the integration process and allowing for custom configurations based on project requirements.

By facilitating the programming and integration of the ESP32 microcontroller, the Adapter Breakout Board streamlines the development process of our wearable prototype. Its user-friendly design and versatile features make it an essential component in ensuring the functionality and performance of our project.

In conclusion, the specified Software and Hardware requirements are indispensable for enhancing the functionality and performance of the wearable device prototype post-midterm. These components, along with the requisite software tools such as Arduino IDE, PlatformIO, and Python IDE, collectively contribute to the successful implementation and operation of the project. By fulfilling these hardware and software requirements, we aim to develop a robust and reliable wearable device prototype that meets the project objectives and user expectations effectively.

2.4.5 System Requirement For Development

Hardware Requirement

The Hardware components required to develop the fitness tracker, include the sensors, Jumper wires, Microcontroller, some other accessories etc. are mentioned already now these are the minimum system specification to develop the project is given below:

- 4-core CPU with x86_64 architecture; 6th generation Intel Core or newer, or AMD 2nd generation CPU or newer with support for a Windows Hypervisor.
- 8 GB RAM or more.
- 120 GB of available disk space minimum (IDE + Android SDK + Android Emulator)
- 1280 x 800 minimum screen resolution.
- Stable Ethernet or WiFi with minimum bandwidth of 10mbps.

If we utilize The centralized server facilitated by Firebase as the backend service for the Flutter application offers significant advantages. Firstly, it ensures seamless synchronization of data across various devices and platforms, enhancing user experience by providing real-time updates and consistent information. Secondly, Firebase's robust infrastructure guarantees high availability and scalability, enabling the application to handle increasing user loads without compromising performance. Moreover, the centralized server architecture simplifies maintenance and updates, as modifications can be implemented universally, reducing development overhead and ensuring uniformity across the application ecosystem. Lastly, Firebase's comprehensive set of features, including authentication, storage, and analytics, streamline backend development, allowing developers to focus on enhancing app functionality and user engagement.

CHAPTER 3: SOFTWARE REQUIREMENT ANALYSIS

3.1 Existing Problem

In India, Currently companies are still in the testing phase. At the international level, the companies already provide these services at a subscription base model which is expensive. So, it is harder to find a feasible model for this project.

We aim to provide an App for this which is IoT-based and justified based on several reasons:

1. User Interface: The hardware module will collect and transmit data to the cloud server, which will then be used for analysis and generating reports. The Flutter app can be used to provide a user-friendly interface for the end-user to access this data. The app can display the collected data in a graphical format, making it easy for the user to understand and interpret.
2. Real-time Database: The Flutter app can be used to provide a real-time database of the IoT device. The app can display the status of the device, including connectivity status, various level, and other vital parameters. This will enable the user to quickly identify and troubleshoot any issues that may arise.
3. Remote Control: The Flutter app can also be used to remotely control the IoT device. This can include features such as turning the device on/off, adjusting the data collection parameters, and other settings. This will provide greater flexibility and control to the end-user, making it easier to adapt the device to their specific needs.

3.2 Problem Definition

The problem lies in the lack of personalised recommendations and data display, which hinders users from effectively achieving their fitness goals and monitoring their health parameters. The inadequacies of current fitness tracking systems, which fail to provide personalised recommendations and effective data display, thus impeding users from achieving their fitness goals and monitoring their health parameters efficiently. Existing fitness trackers often feature distracting screens, limiting their effectiveness and user engagement. There is a clear need for a comprehensive solution that utilises advanced sensors to revolutionise the way individuals track their fitness progress and improve their overall well-being.

Significance: Fitness tracking has become increasingly popular, with individuals seeking to lead healthier lifestyles. However, current fitness has distracting screens which limit their effectiveness.

Motivation: The motivation behind this project stems from the growing need for accurate and personalised fitness tracking solutions. By developing a comprehensive solution that utilises advanced sensors, we aim to revolutionise the way individuals track their fitness progress and improve their overall well-being.

3.3 Modules Definition

3.3.1 Software Module

The absence of a streamlined and user-friendly health tracking solution has led to the development of the WellWisher App. Traditional fitness applications often lack cohesive interfaces and personalised insights.

Proposed Software Module: The WellWisher App, built on Flutter with a backend supported by Realtime Database and Cloud Firestore, serves as an innovative solution to bridge existing gaps in health monitoring applications.

Comparative Solution: In contrast to conventional health apps, WellWisher stands out by integrating a user-friendly interface with robust backend technologies. The use of Flutter ensures cross-platform compatibility, while Realtime Database and Cloud Firestore offer real-time data updates and seamless synchronisation across devices.

Functionality of the Module:

1. Authentication Module:
    - Login: Secure user authentication to access personalised health data.
    - Signup: User registration for new members.
    - Forgot Password: Account recovery functionality for enhanced user experience.
2. Dashboard Module:
    - Comprehensive Health Metrics: Provides a user-friendly dashboard for viewing data.
3. Data Storage and Retrieval:
    - Firestore Database: Efficiently stores and retrieves user-specific health data.

- Real-time Updates: Ensures continuous synchronization for up-to-date health insights.

By addressing these aspects, the WellWisher App aims to revolutionize the health tracking experience, setting it apart from traditional applications.

3.3.2 Hardware Module

Traditional methods of health parameter monitoring, such as manual recording or limited sensor-based devices, fail to provide comprehensive and timely information. Either the equipments is large and not smart enough for scalable with present digital infrastructure like IoT.

Proposed Hardware Module: The integration of the MAX30102 sensor with an ESP32 microcontroller serves as a groundbreaking solution to enhance health monitoring capabilities. This setup, coupled with backend support from Realtime Database and Cloud Firestore, revolutionizes data collection and analysis for the WellWisher App.

Comparative Solution: In contrast to conventional health monitoring devices, which rely on manual data entry or simplistic sensors, the MAX30102 sensor offers advanced capabilities for real-time physiological parameter monitoring. By leveraging the ESP32 microcontroller's processing power, the hardware module enables seamless integration with backend databases, ensuring timely data transmission and synchronization with the WellWisher App.

Functionality of the Module:

1. Sensor Integration:
    - The MAX30102 sensor captures physiological parameters, such as heart rate and oxygen saturation levels, with high reliability.
    - The sensor is seamlessly integrated with the ESP32 microcontroller, facilitating efficient data processing and transmission.
2. Microcontroller Control:
    - The ESP32 microcontroller acts as the central control unit, orchestrating the operation of the MAX30102 sensor and managing data transmission to the backend databases.
    - Preprogrammed with Micropython code in Thonny IDE, the ESP32 ensures optimal performance and compatibility with the WellWisher App.

3. Backend Integration:
    - Data collected by the hardware module is transmitted to the Realtime Database and Cloud Firestore, where it is stored securely and made accessible to the WellWisher App.
    - Instant updates and synchronization enable users to view their health metrics instantly on the app, fostering a seamless and intuitive user experience.

Through these functionalities, the hardware module enhances the accuracy, reliability, and efficiency of health parameter monitoring, empowering users to make informed decisions about their well-being.

3.3.3 Under progress Functionalities

Expansion of MPU9250

In addition to real-time physiological parameter monitoring facilitated by the MAX30102 sensor, the integration of the MPU9250 sensor further enriches the health monitoring capabilities of the WellWisher App. The MPU9250 sensor offers a comprehensive suite of features, including gyroscope and accelerometer functionalities, enabling the capture of additional health metrics such as activity levels, motion patterns, and orientation changes. Leveraging these capabilities, the WellWisher App can track user activity throughout the day, providing insights into daily step counts, physical activity levels, and sleep patterns. By incorporating data from the MPU9250 sensor into the backend databases, users can access a holistic view of their health metrics, enhancing their understanding of overall well-being.

Enhanced Software Module

To complement the expanded hardware functionalities, the software module of the WellWisher App introduces personalized recommendations and profile management features. Users are empowered to create personalized profiles with customizable settings and permissions, catering to different user roles such as parent, child, or healthcare provider. Through the app's intuitive interface, users can set health goals, receive personalized recommendations based on their activity and health metrics, and track progress over time. Moreover, the app facilitates seamless communication by enabling users to securely share their health data with designated healthcare professionals or family members, fostering collaborative care and support. With these enhanced functionalities, the WellWisher App strives to empower users to take proactive steps towards optimizing their health and well-being.

CHAPTER 4: SOFTWARE DESIGN

4.1 Design Approach

The design approach for the Flutter app of the WellWisher project emphasizes user-centricity, simplicity, and scalability. Following a user-centered design philosophy, the app interface is intuitively designed to ensure ease of navigation and accessibility for users of all demographics. The app adopts a minimalist aesthetic, with clean layouts, intuitive icons, and clear typography, enhancing user engagement and readability. Utilizing Flutter's widget-based architecture, the app offers seamless cross-platform compatibility, ensuring consistent performance across Android and iOS devices.

4.1.1 Key Design Principles(App)

User Experience (UX) Optimization: Prioritizing user experience, the app interface is designed to be intuitive and responsive, enabling users to effortlessly navigate through various features and functionalities.

Personalization and Customization: Recognizing the diverse needs and preferences of users, the app incorporates personalization features, allowing users to customize their profiles, set health goals, and receive tailored recommendations based on their individual health metrics and preferences.

Scalability and Flexibility: Built with scalability in mind, the app architecture is designed to accommodate future feature enhancements and updates seamlessly. Modular components and a flexible codebase ensure easy maintenance and extensibility.

Security and Privacy: Upholding stringent security and privacy standards, the app employs encryption protocols and secure authentication mechanisms to safeguard user data. User privacy is prioritized, with clear transparency regarding data collection, storage, and sharing practices.

4.1.2 Hardware Design

The hardware design for the WellWisher project encompasses the integration of sensors, microcontrollers, and power management components within a compact and ergonomic form factor. Key considerations include:

Component Selection: Careful selection of components, including sensors such as the MAX30102 and MPU9250, ensures accurate and reliable data acquisition. The integration of an ESP32 microcontroller facilitates seamless data processing and communication with the backend server.

Power Management: The inclusion of a rechargeable LiPo battery and TP4056 charging module ensures reliable power supply and efficient battery management. The hardware design prioritizes energy efficiency and longevity, enabling extended usage without frequent recharging.

Form Factor and Ergonomics: The hardware prototype is designed to be lightweight, portable, and comfortable for extended wear. Ergonomic considerations such as device size, weight distribution, and strap design are optimized for user comfort and convenience.

Integration and Connectivity: Seamless integration of hardware components and robust connectivity options, including Bluetooth and Wi-Fi, enable seamless communication between the wearable device and the companion mobile app. The hardware design ensures reliable data transmission and synchronization with the app backend in real-time.

4.2 System Architecture

System architecture refers to the structure or framework of a system, including its components, their relationships, and the principles guiding their design and evolution. It defines how various elements of a system work together to achieve the system's objectives, ensuring that the system is robust, scalable, and maintainable. The Architecture of “FITNESS TRACKING INSIGHTS” with its app. The system consists of the following components:

- End User: This is the person who will be wearing the fitness band and using the WellWisher App.
- Flutter App: This is the Android app that is used to view and track the fitness data. It is developed using the Flutter framework.
- Hardware Prototype: In This, The microcontroller that is used to collect data from the sensors in the fitness band. It is connected to the internet via a mobile hotspot.
- Also this module contains the sensors that are used to track the fitness activity of the end user. These sensors may include an accelerometer and a heart rate monitor.
- Firebase Realtime Database: This is a cloud-based database that is used to store the fitness data collected by the ESP32. The Android app can access this data to display it to the user.

The arrows in the diagram show how the different components communicate with each other. For example, the ESP32 collects data from the sensors and sends it to the Firebase Realtime Database. The Android app then retrieves the data from the database and displays it to the user.

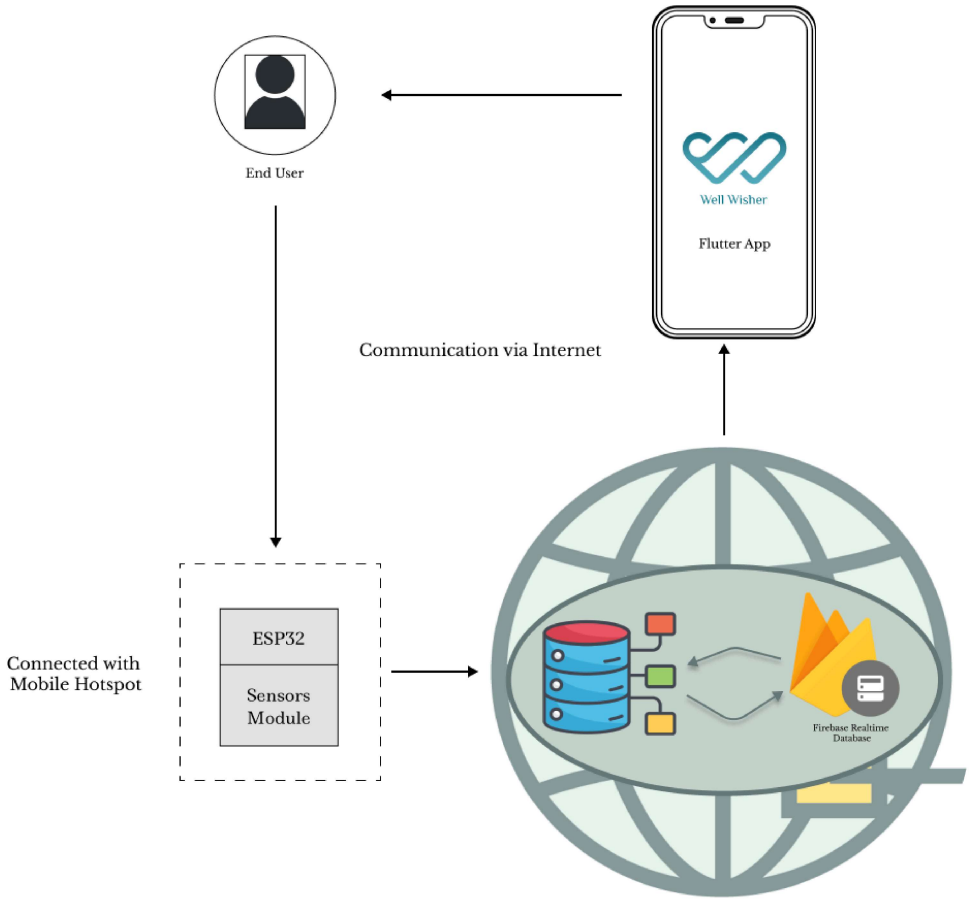

Fig. 4.2.1 System Architecture Diagram

The diagram illustrates the system architecture of the WellWisher project, showcasing the interaction between hardware, software modules, databases, and users. It visually represents how sensor data is collected by hardware components such as the MAX30102 and MPU9250, processed by microcontrollers like the ESP32, and transmitted to the backend system. The software modules, including the Flutter app and firmware, facilitate user interactions, data processing, and communication with the backend servers.

Databases, specifically Realtime Database and Cloud Firestore, store and manage user data, ensuring seamless synchronization and retrieval. Overall, the diagram provides a concise overview of the system's components and their interactions, highlighting the flow of data and user engagement.

4.3 Data Flow Diagrams

DFDs, or Data Flow Diagrams, are graphical representations that illustrate the flow of data within a project. In the context of our 'FITNESS TRACKING INSIGHTS' project, the DFDs section should include the following content:

4.3.1 Level 0 DFD

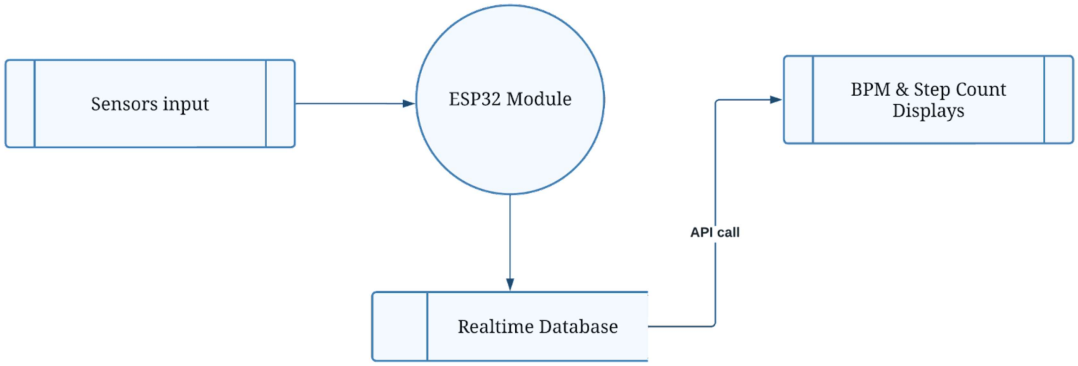

Fig. 4.3.1 Level 0 DFD

It is an abstract-level DFD which provides a high-level view of the system and its major processes or functions. For instance, The sensors take input from the user's body and upload it to the database with the help of ESP32.

4.3.2 Level 1 DFD

Break down the Level 0 DFD into more detailed Level 1 DFDs. There are two major sensors used MPU 9250 for Step counting and MAX30102 which takes input from the user's finger to sense heartbeat.

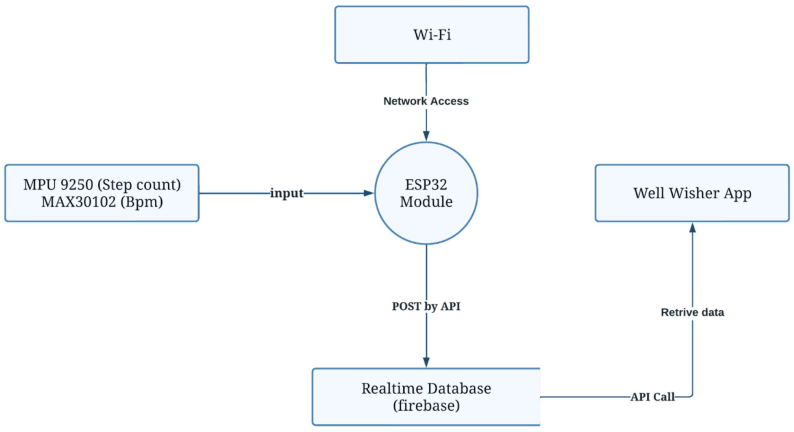

Fig. 4.3.2 Level 1 DFD

Further with the help of ESP32 raw values were computed and actual values were sent with the help of API on the real-time database(firebase) over the network. Now the API gets the values and displays the last value (bpm and step count) on WellWisher App.

4.3.3 Level 0 DFD App

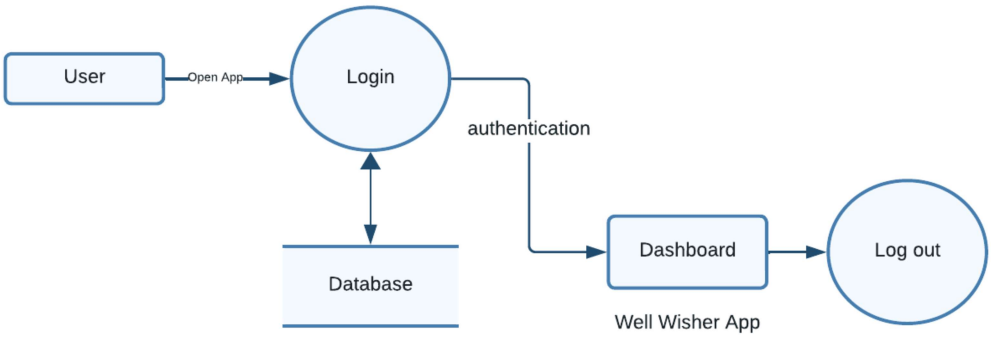

Fig. 4.3.3 Level 0 DFD App

In the Level 0 DFD of the app, the primary process is depicted as "Dashboard", which represents the main functionality of the application. The user initiates interaction with the app by opening it, leading to the login process for authentication.

Once logged in, the app accesses the database to retrieve user-specific data and information. The Dashboard process then presents this data to the user for viewing and interaction. Finally, the user can choose to log out of the application, completing the flow.

4.3.4 Level 1 DFD App

In above illustrated Level 1 DFD of WellWisher App, we can expand on the processes and data flows depicted in the Level 0 DFD to provide more detail:

User Login:

- The user initiates the login process by entering their credentials into the app.
- The app sends the login request to the Authentication process.
- If the credentials are valid, the Authentication process grants access to the Dashboard.
- If the credentials are invalid, an error message is returned to the user.

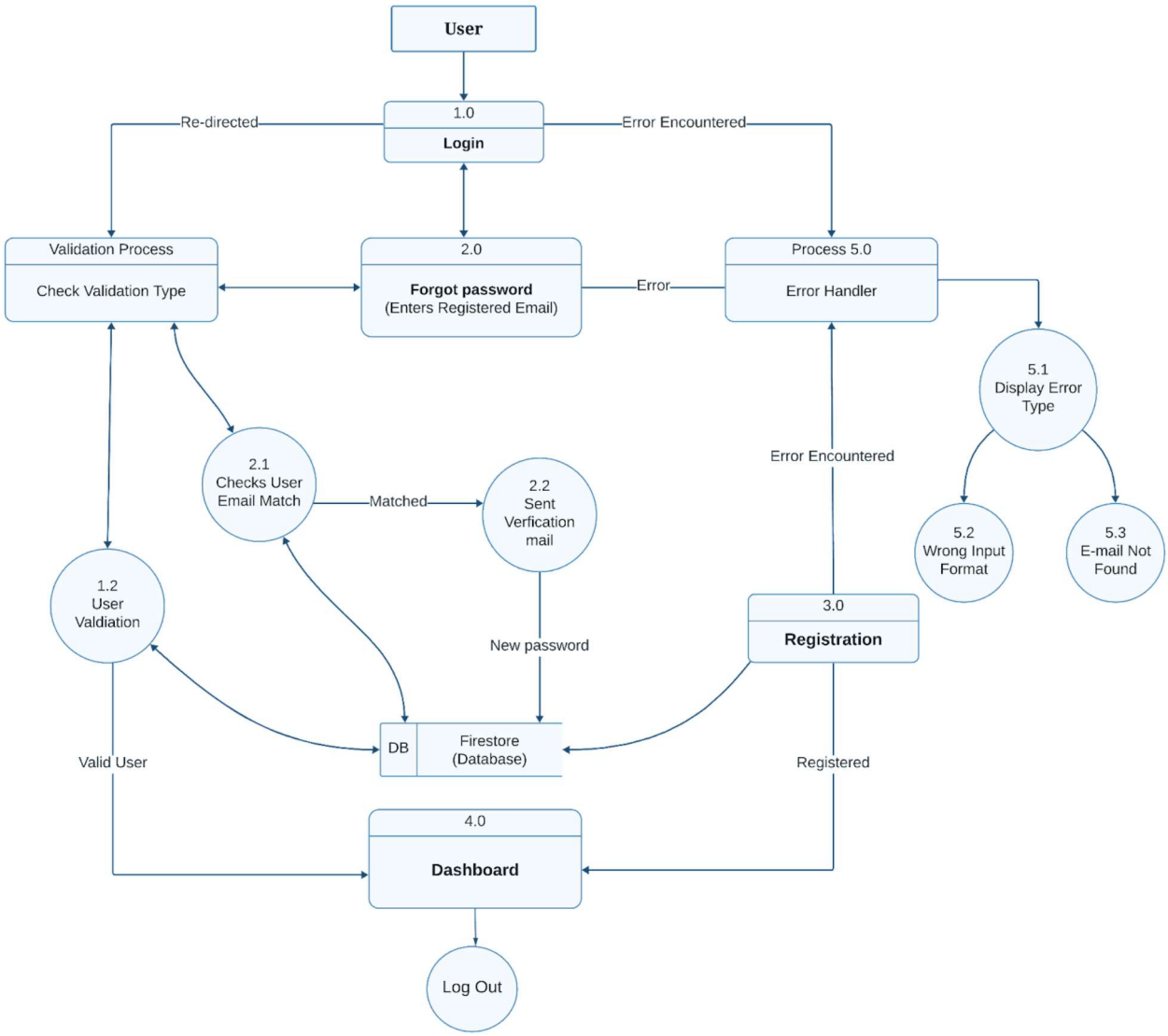

Fig. 4.3.4 Level 1 DFD App

Dashboard Display:

- Once logged in, the Dashboard process retrieves the user's personalised data from the database.
- This data may include health metrics, activity logs, recommendations, etc.
- The Dashboard process formats the data for display and presents it to the user in the app interface.

Logout:

- When the user chooses to log out of the app, the Logout process is triggered.
- The app clears the user's session data and returns to the login screen.

Authentication:

- Responsible for verifying the user's login credentials against the stored authentication data.
- If the credentials are valid, access is granted to the Dashboard; otherwise, access is denied.

Database:

- Stores user authentication data, including usernames, passwords, and access permissions.
- Also stores user-specific data such as health metrics, activity logs, etc.

These processes and data flows represent the core functionality of the app at a more detailed level, illustrating the interactions between the user, the app, and the database.

4.4 UML Diagram

In the UML Diagram, the main focus is on the "ESP32," depicted as a class. This controller embodies the ESP32 microcontroller and the Relay Module, A Class in MicroPython.This simplified UML diagram offers a clear overview of the core hardware components and their interactions within the circuit board.

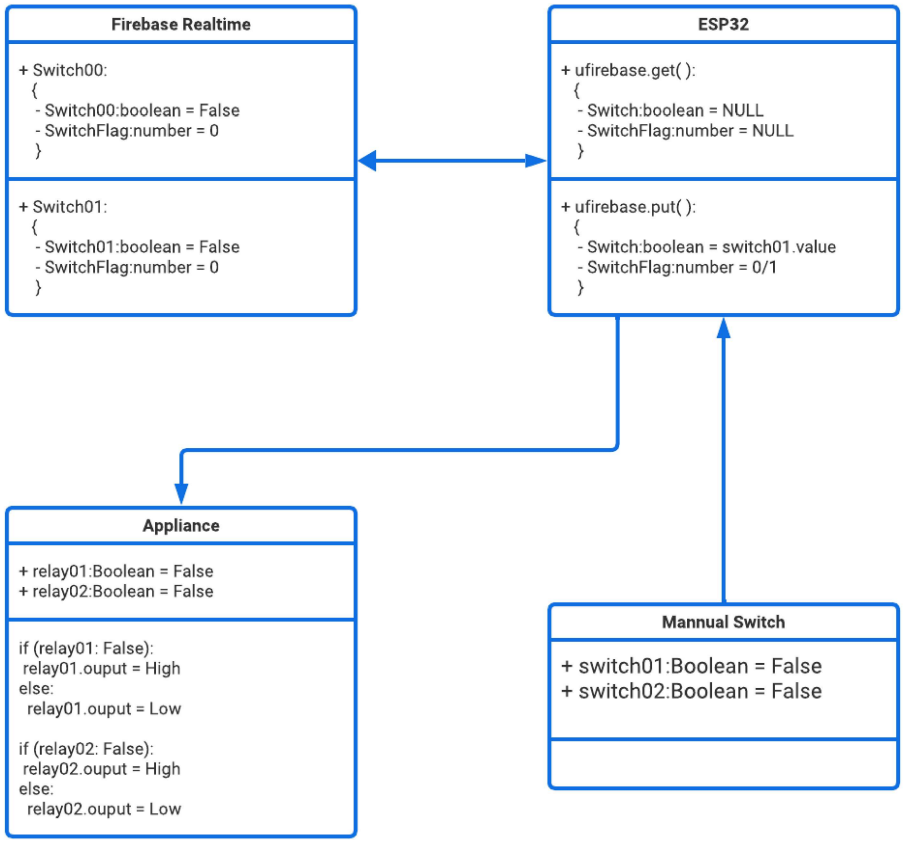

Fig. 4.4.1 UML Diagram

4.5 Database Design

Data Structure: Determine the entities, their attributes, and the relationships between them. Plan how the data will be organized and nested within the database.

JSON Structure: Firebase Realtime Database uses JSON as its data format. Design your data structure in a way that aligns with JSON's key-value pairs. Ensure that the data can be easily serialized and deserialized into JSON format.

Node Hierarchy: Firebase Realtime Database is organized as a hierarchical tree structure. Determine the hierarchy of nodes based on the relationships between entities. Each node should represent a specific entity or a collection of related entities.

Security Rules: Define appropriate security rules to restrict access to your database. Firebase provides a powerful security rule system that allows you to control read and write operations based on user authentication and data validation.

Real-time Updates: Leverage the real-time capabilities of Firebase by setting up listeners or observers to receive real-time updates whenever data changes in the database. Design your database structure to facilitate efficient and meaningful real-time updates for your application.

Testing and Iteration: As with any database design, it's crucial to test and iterate on your design to ensure it meets the requirements of your application. Make use of Firebase's testing tools and simulated data to verify the performance and functionality of your database design.

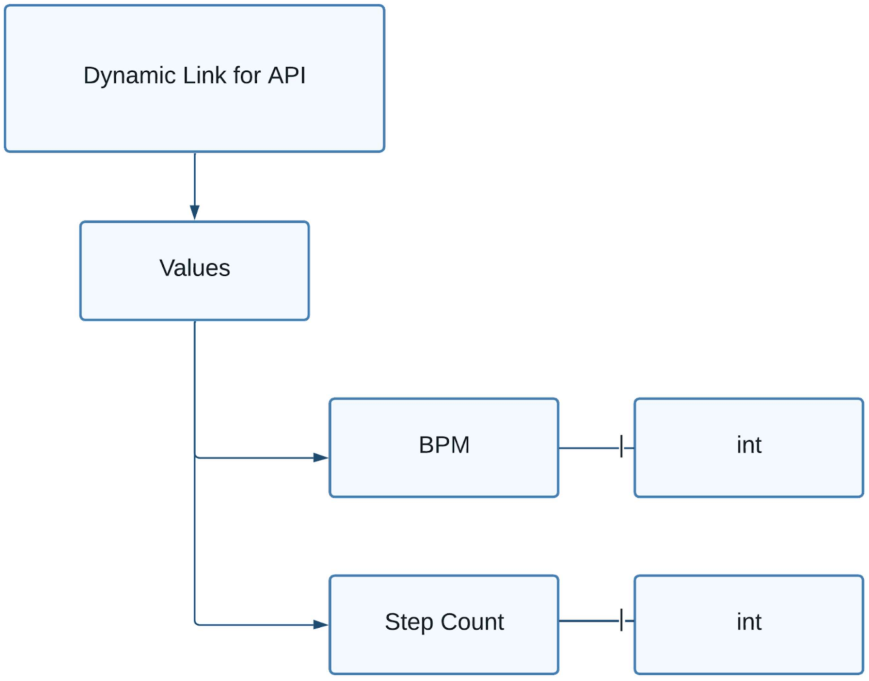

Fig. 4.5 Database Design in Firebase

4.5.1 ERD Diagram

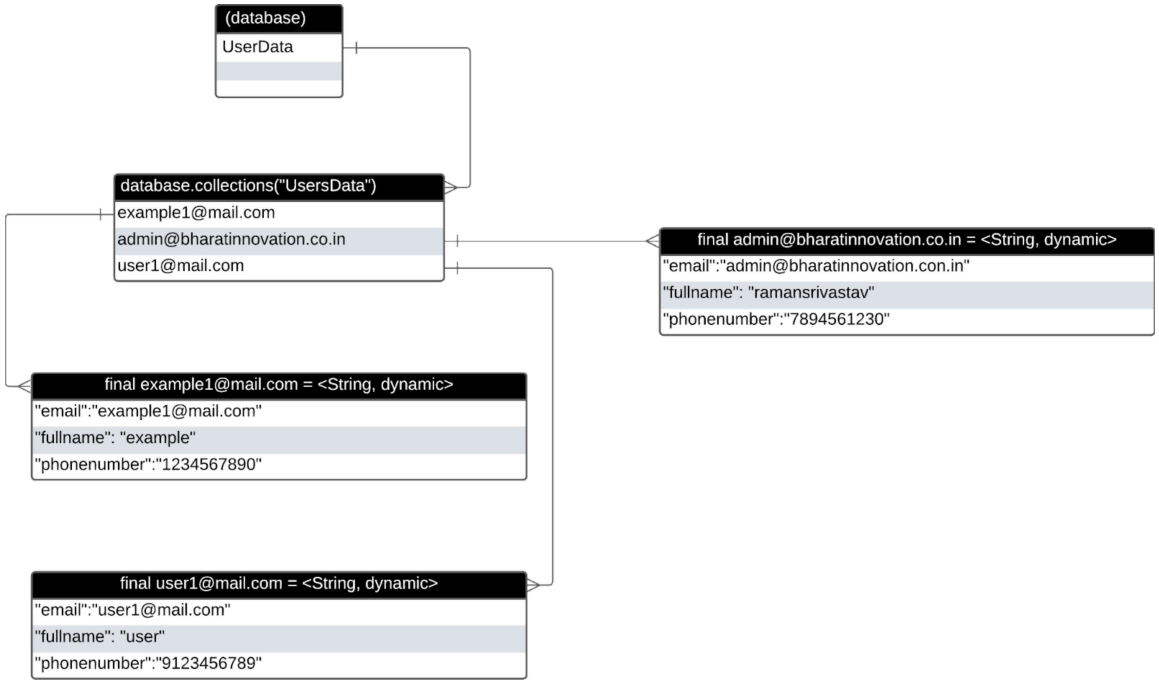

Fig. 4.5.1 ER Diagram of Firestore

The Entity-Relationship Diagram (ERD) for a Cloud Firestore database mentioned above shows a database schema for storing user data. It consists of a single entity called “UserData”. Each instance of UserData represents a user in the system and has the following attributes:

- email: The unique identifier for the user and the primary key for the entity.
- fullname: The full name of the user.
- phone number: The phone number of the user.

The Cloud Firestore database is a NoSQL database, so it does not have tables or rows in the traditional sense. Instead, it stores data in collections of documents. In this ERD, the UserData entity would be represented by a collection called “Users Data”. Each document in the collection would represent a single user and would contain the user’s email address, full name, and phone number as key-value pairs.

Firestore also supports nesting collections and documents. This means that you can create collections within collections to store hierarchical data. As we could create a subcollection within the “Users Data” collection to store each user’s posts.

CHAPTER: 5 TESTING MODULE

5.1 Software Testing

5.1.1 Testing Techniques

Unit Testing: Testing individual components/modules of the WellWisher App in isolation to ensure they function correctly. And Using the testing frameworks like Flutter's built-in testing framework or third-party packages like Mockito for mocking dependencies.

Integration Testing:

- Verifies that different modules/components of the app work together seamlessly.
- Ensure proper communication between frontend and backend systems.

User Interface (UI) Testing:

- Validate the app's user interface for responsiveness, layout, and visual consistency across different devices and screen sizes.
- Use tools like Flutter's widget testing framework or third-party tools like Appium for automated UI testing.

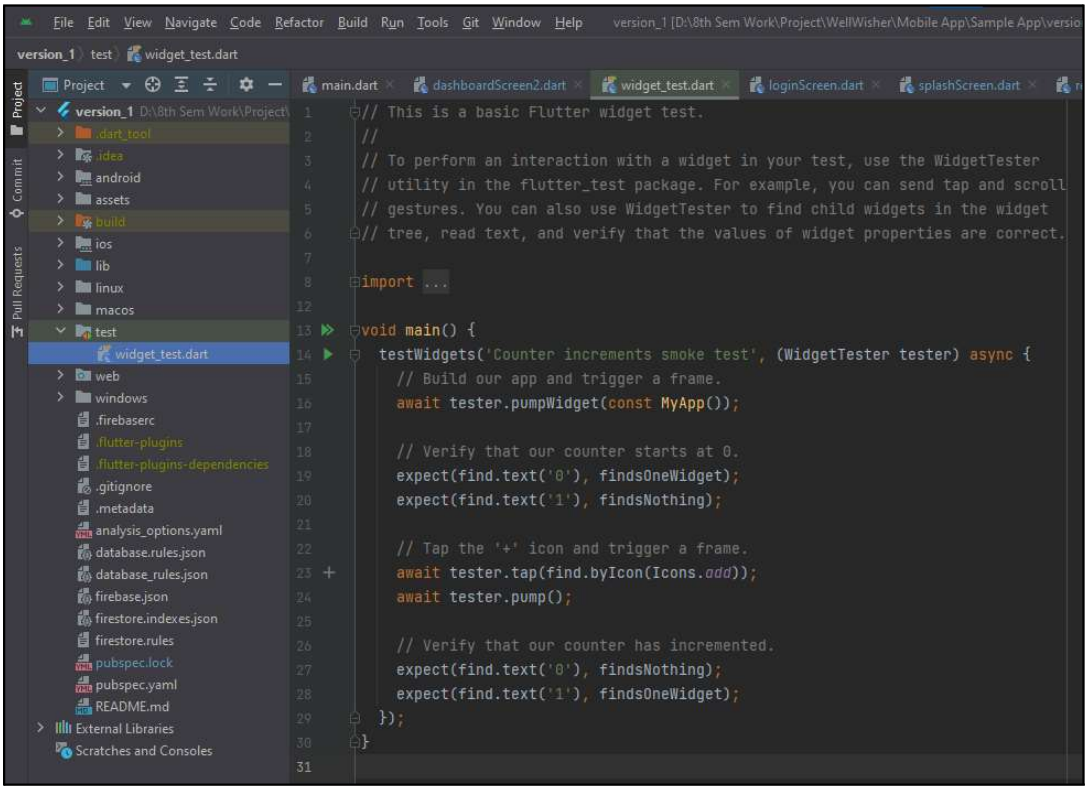

Fig. 5.1.1 Default Test File in Android Studio

5.1.2 Resource Consumption

Measured response time: 18-20-16 milliseconds

1. Hardware Resources:
    - CPU Usage: The WellWisher app utilizes CPU resources primarily for processing user interactions, data retrieval, and rendering user interface elements.
    - Memory Usage: The app consumes memory for storing application data, user preferences, and cached content. This includes RAM usage for runtime operations and storage space for app installation and data storage.
    - Battery Usage: The app may consume battery power, especially during active usage such as sensor data collection, background services, and network communication.
2. Software Resources:
    - Network Bandwidth: The app requires network resources for communication with remote servers, fetching data, and sending requests. This includes both cellular data and Wi-Fi bandwidth consumption.
    - CPU Processing Time: The app utilizes CPU processing time for executing application logic, handling user inputs, and performing background tasks. This includes time spent on computation-intensive operations such as data processing and algorithms execution.

5.1.3 Test Case

Title: User Login Authentication

Test Case ID: WWT001

Test Objective: To verify that the user authentication process functions correctly and grants access to authenticated users while denying access to unauthorized users.

Preconditions:

- The WellWisher app is installed and running.
- The user has valid login credentials (username and password).
- The user has an active internet connection.

Test Steps:

1. Open the WellWisher app.
2. Navigate to the login screen.
3. Enter valid login credentials (username and password).

4. Click on the "Login" button.
5. Verify that the app communicates with the authentication server.
6. Verify that the entered credentials are correct.
7. If the credentials are correct, verify that the app grants access to the user's dashboard.
8. If the credentials are incorrect, verify that the app displays an appropriate error message and does not grant access.

Expected Results:

- If the credentials are correct:
- The app should authenticate the user successfully.
- The user should be redirected to their dashboard.
- If the credentials are incorrect:
- The app should display an error message indicating invalid credentials.
- The user should not be granted access to the dashboard.

Test Data:

- Valid username: [mk4466378@mail.com]
- Valid password: [Mohit123]
- Invalid username: [mk4466@yahomail.com]
- Invalid password: [abc132]

Test Environment:

- Device: [Mi 11x]
- Operating System: [Android 12 ‘S’]
- WellWisher app version: [App version 1.0.0]

5.2 Hardware Testing

5.2.1 Sensor Testing

MAX30102

This test case initializes the I2C communication, initializes the MAX30102 sensor, and then reads the raw IR and red light data from the sensor for 10 iterations. It prints out the raw data for each iteration, allowing you to verify that the sensor is functioning and providing data.

```
from machine import I2C, Pin
import max30102
import time

# Function to perform a simple test of the MAX30102 sensor
def test_max30102()

    # Initialize I2C communication
    i2c = I2C(scl=Pin(22), sda=Pin(21))
    # Initialize MAX30102

    max30102_sensor = max30102.MAX30102(i2c)
    # Perform a series of readings and print the results
    for _ in range(10):  # Perform 10 readings
        # Read raw IR and red light data from MAX30102
        data = max30102_sensor.read_sensor()
        # Extract IR and Red value

        ir_data = data[0]
        red_data = data[1]
        # Print the raw dat

        print("IR Data:", ir_data)
        print("Red Data:", red_data)
        # Delay for a short while before the next reading
        time.sleep(1)  # You can adjust the delay based on your
requirements
# Run the test
test_max30102()
```

Here to adjust the pin numbers for the I2C communication (scl and sda) based on our specific hardware setup.

After running this test case, we observe the raw IR and red light data printed to the console for each iteration, indicating that the sensor is communicating properly with our ESP32 board. While we encounter any errors or unexpected behavior, it may indicate a problem with the hardware setup or the sensor itself.

MPU9250 accelerometer

This test case initializes the I2C communication, initializes the MPU9250 accelerometer, and then continuously reads the accelerometer data to calculate step counts. In this example, we use a simple threshold-based approach to count steps based on the acceleration along the X-axis. You may need to adjust the threshold value based on the sensitivity of your accelerometer.

```
from machine import I2C, Pin
import mpu9250
import time
# Function to perform a simple test of the MPU9250 accelerometer for
step counting

def test_mpu9250_step_count():
    # Initialize I2C communication
    i2c = I2C(scl=Pin(22), sda=Pin(21))
    # Initialize MPU9250
    mpu = mpu9250.MPU9250(i2c)
    # Initialize variables
    step_count = 0
    threshold = 2  # Adjust the threshold based on your specific
requirements

    # Main loop
    while True:
        # Read accelerometer data from MPU9250
        accel = mpu.acceleration
        # Calculate step count
        # For demonstration, let's use a simple threshold-based
approach
        if accel[0] > threshold:  # Check acceleration along X-axis
            step_count += 1

        # Print step count
        print("Step Count:", step_count)

        # Delay for a while before reading again

        time.sleep(1)  # You can adjust the delay based on your
requirements

# Run the test
test_mpu9250_step_count()
```

After running this test case, we observe the step count printed to the console, indicating that the MPU9250 accelerometer is functioning and counting steps. If you encounter any errors or unexpected behavior, it may indicate a problem with the hardware setup or the accelerometer itself.

CHAPTER 6: PERFORMANCE MEASURES

6.1 Evaluation of Project

Several key aspects need to be considered for The evaluation of the performance of the Fitness Tracking Insights project, including the WellWisher app, encompasses various aspects to ensure its effectiveness, reliability, and user satisfaction. Here's a comprehensive overview of the performance evaluation:

Functionality Testing: Verify that all features and functionalities of the WellWisher app, including user authentication, dashboard display, health metric tracking, and recommendation system, perform as intended without errors or glitches.

Usability Assessment: Conduct usability testing to evaluate the user experience of the WellWisher app. Assess factors such as ease of navigation, intuitiveness of the interface, clarity of information presentation, and overall user satisfaction.

- Performance Metrics: Define and monitor key performance indicators (KPIs) such as app responsiveness, loading times, and system stability.

Compatibility Testing: Test the WellWisher app on various devices, operating systems, and web browsers to ensure compatibility and consistent performance across platforms. Verify that the app functions correctly on different screen sizes, resolutions, and orientations.

- Security Assessment: Perform security testing to identify and mitigate potential vulnerabilities in the WellWisher app. Assess the app's adherence to security best practices, data encryption standards, and compliance with privacy regulations

Load and Stress Testing: Conduct load and stress testing to evaluate the app's performance under heavy usage and high traffic conditions.

Error Handling: Test the app's error handling mechanisms to ensure robustness and reliability in handling unexpected errors, exceptions, and edge cases. Verify that appropriate error messages are displayed, and the app gracefully recovers from failures without data loss or system instability.

Battery and Resource Consumption: Evaluate the app's impact on device battery life and resource consumption, particularly for features such as continuous sensor data monitoring and background

CHAPTER 7: OUTPUT SCREEN

7.1 WellWisher UI

Fig 7.1 App Splash Screen

The splash screen serves as the initial visual interface that appears when launching the WellWisher app. Its primary function is to provide users with immediate feedback that the app is loading and initializing.

7.2 Login Screen

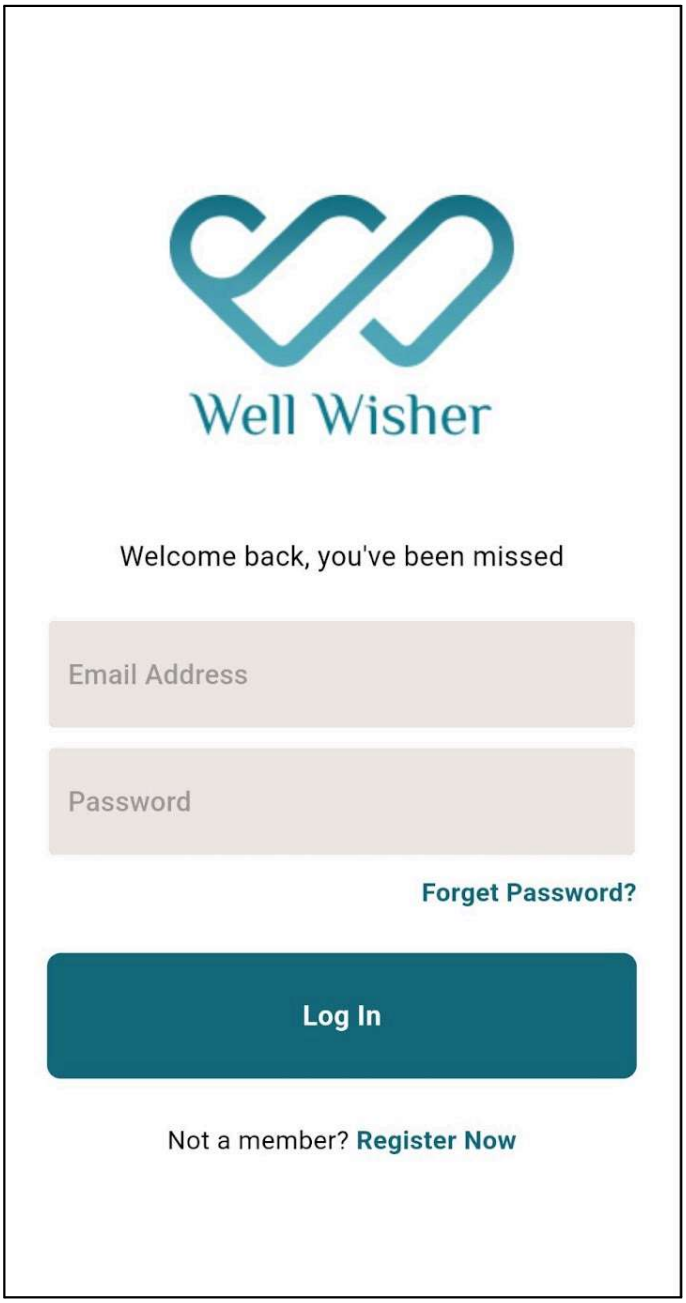

Fig 7.2 login Screen

The login screen of the WellWisher app presents users with a clean and intuitive interface designed for effortless authentication. The screen features prominently positioned input fields for entering login credentials, including username or email and password, ensuring easy access for returning users. Clear and concise instructions prompt users to input their credentials, while discreet error messaging provides real-time feedback to aid in resolving any authentication issues.

7.3 SignUp Screen

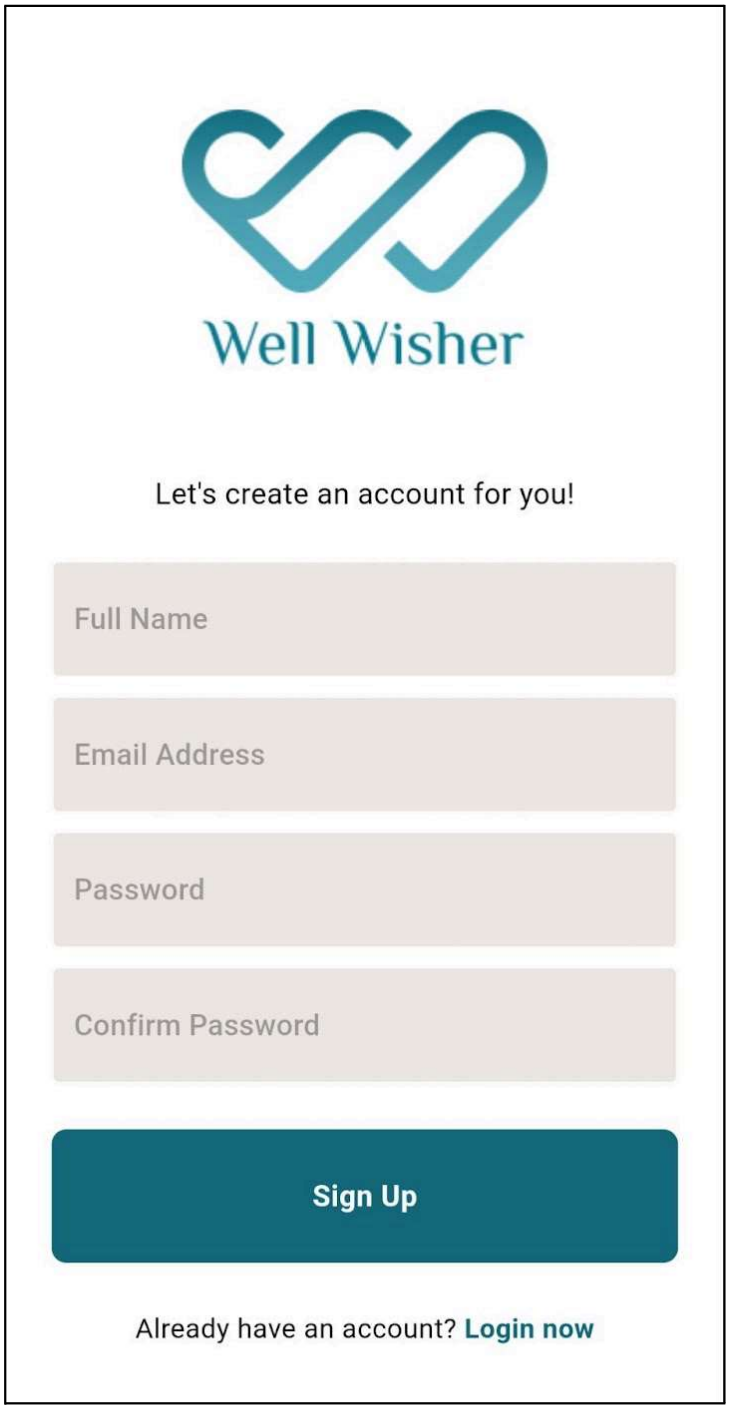

Fig 7.3 SignUp Screen

The signup screen of the WellWisher app offers users a straightforward and efficient means of creating a new account. Featuring prominently displayed input fields for essential registration details such as username, email address, and password, the screen ensures a user-friendly experience for new users. Clear instructions guide users through each step of account creation,

7.4 Dashboard View

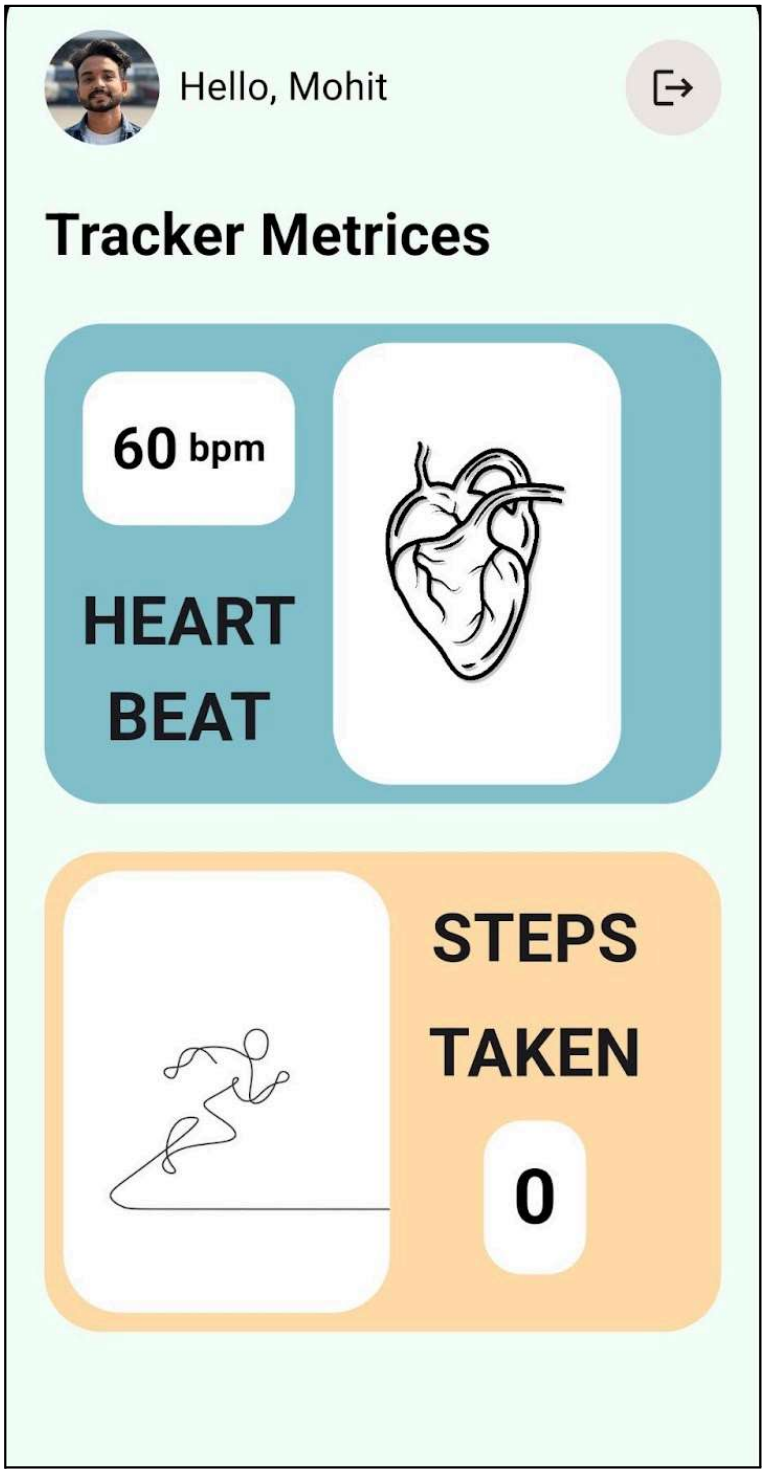

Fig 7.4 Dashboard View

The dashboard screen serves as the central hub for users to access and monitor their health metrics with ease and convenience. Featuring a clean and intuitive layout,overview of essential health data, including heart rate, step count, sleep patterns, and more.

REFERENCES

[1] A. Gupta and A. Gautam, "Health Monitoring of Elderly People Using IoT," SN Computer Science, vol. 1, no. 6, pp. 1-10, 2020. [Online].
Available: https://link.springer.com/article/10.1007/s42979-020-00195-y. [Accessed: 24-Jan-2024].

[2] P. C. Rout and S. K. Nanda, "IoT Based Patient Health Monitoring using ESP8266 and Arduino," ResearchGate, 2021. [Online].
Available:https://www.researchgate.net/publication/359024678_IoT_Based_Patient_Health_Monitoring_using_ESP8266_and_Arduino. [Accessed: 26-Jan-2024].

[3] Circuit Digest, "How MAX30102 Pulse Oximeter and Heart-Rate Sensor Works and How to Interface with Arduino," [Online].
Available:https://circuitdigest.com/microcontroller-projects/how-max30102-pulse-oximeter-and-heart-rate-sensor-works-and-how-to-interface-with-arduino. [Accessed: 26-Jan-2024].

[4] Onix-Systems, "How to Create a Fitness App and Generate Revenue with It," [Online].
Available:https://onix-systems.com/blog/how-to-create-a-fitness-app-and-generate-revenue-with-it. [Accessed: 31-Jan-2024].

[5] C. Norella, "ESP32 BLE Android App Development with Flutter," YouTube, 2022. [Online].
Available: https://www.youtube.com/playlist?list=PLw0SimokefZ3uWQoRsyf-gKNSs4Td-0k6. [Accessed: 31-Jan-2024].

[6] Last Minute Engineers, "MAX30102 Pulse Oximeter and Heart-Rate Sensor Arduino Tutorial," [Online].
Available:https://lastminuteengineers.com/max30102-pulse-oximeter-heart-rate-sensor-arduino-tutorial/. [Accessed: 06-Feb-2024].

[7] Analog Devices, "MAX30102 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor," Datasheet and Product Information, [Online]. Available:
https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf. [Accessed: 16-Feb-2024].

[8] TDK InvenSense, "MPU-9250 Nine-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking™ Device," Product Specification, [Online]. Available:
https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. [Accessed: 16-Feb-2024].