FITNESS TRACKING INSIGHTS

MAJOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR
THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY
(Computer Science and Engineering)

Submitted By:

Mohit Kumar (2104225)
Naval Thanik (2104226)
Sonal Kumari (2104232)

Submitted To:

Dr. Kiran Jyoti
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA, 141006
April, 2024

ABSTRACT

The "Fitness Tracking Insights" project aims to revolutionise the health and fitness tracking industry by providing users with a comprehensive solution that combines advanced hardware and intuitive software. Traditional fitness tracking devices often fail to deliver accurate data and personalised recommendations, leading to user frustration and inefficiency. In response, our project integrates cutting-edge sensors into a wearable hardware prototype, ensuring precise health data collection. This hardware is complemented by the "WellWisher" mobile application, which offers a user-friendly interface for visualising health metrics and receiving personalised recommendations. By focusing on accuracy, personalization, and user experience, Fitness Tracking Insights strives to empower individuals to take control of their health and achieve their fitness goals effectively.

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to Dr. Sehijpal Singh, Principal of Guru Nanak Dev Engineering College (GNDEC), Ludhiana, for providing us with the opportunity to undertake our Major Project, "FITNESS TRACKING INSIGHTS".

We are deeply appreciative of the unwavering support and guidance received from Dr. Kiran Jyoti, our Head of Department (Computer Science and Engineering) and Project Guide at GNDEC Ludhiana. Her mentorship has been instrumental in navigating the complexities of our project journey, and we acknowledge her contributions with sincere thanks.

We would also like to express our gratitude to the other faculty members of the Computer Science and Engineering department at GNDEC for their valuable insights and support throughout this endeavour.

Furthermore, we are thankful to all individuals who have contributed to the success of this project, directly or indirectly.

Their collective efforts have significantly enriched our learning experience and enabled us to accomplish our project goals effectively.

Mohit Kumar

Naval Thanik

Sonal Kumari

LIST OF FIGURES

TABLE OF CONTENTS

Chapter 1: Introduction

IoT technology has opened the way for innovative approaches in a variety of areas, and our proposed project, "FITNESS TRACKING INSIGHTS" is at the leading edge of utilising these developments in the fields of personal wellness and physical activity. Our project, which seeks to transform the way individuals monitor and improve their health, collects and analyses critical health data utilising microcontrollers and sensors. By effectively integrating these components into a wearable device, users may easily track parameters like heart rate, steps taken etc. We're using cross-platform development frameworks to improve the user experience and availability of the WellWisher application. This user-friendly application provides one place for users to observe and learn about the fitness data received via their wearable prototype. Through personalised insights and recommendations tailored to individual fitness goals, the app empowers users to take proactive steps towards achieving optimum health and wellness.

In this chapter, we provide an overview of our project, highlighting the goals it seeks, technique, as well as its importance in fulfilling the increasing demand towards reliable but straightforward to use health monitoring solutions. We aim to improve the manner in which the people who are dedicated to their physical fitness and health journeys by combining our knowledge, sophisticated technology, and user-centred planning values.

1.1 Introduction to Project

Our initiative, "FITNESS TRACKING INSIGHTS" is all about allowing individuals to take control of their well-being in a seamless and personalised manner. Consider a future in which we easily track the heart rate, steps taken, and monitor our body temperature using a smart wearable gadget and a simple mobile app.

Our concept is built around the combination of modern Internet of Things (IoT) tools with focused on users standards of design. We developed a wearable device that serves as our personal health companion, offering immediate information into our physical activity and vital indicators using microcontrollers and sensors. There will be no more guesswork or ambiguous estimates; only clear, actionable data at our fingertips.

We've also created the WellWisher app, an easy-to-use platform for making sense of all that information. This app allows you to see your fitness stats, set personalised goals, and receive tailored advice to help you attain user's peak performance. This user-friendly mobile application serves as our virtual wellness coach, offering personalised recommendations, tracking our progress, and celebrating our victories along the way. Whether we're aiming to shed a few pounds, boost our energy levels, or simply lead a healthier lifestyle, "FITNESS TRACKING INSIGHTS" is here to support us every step of the way.

1.2 Project Category: Internet of Things (IoT)

Our project falls under the Internet of Things (IoT) domain, which deals with the internet-based interconnection of systems and objects. The automation and intelligence of Internet of Things (IoT) technology has transformed all aspects of human existence. IoT has the potential to alter a variety of sectors by optimising operations and extracting important data insights. It can make our lives better by fostering health and wellbeing, improving safety and security, and developing more sustainable and effective systems.

In our project, IoT technology is used to create a wearable prototype of the fitness monitoring gadget with a variety of sensors that monitor user health indicators. These devices communicate wirelessly with a central database, allowing immediate data collecting and processing.

Dedicated Mobile Application: Our project includes the development of mobile applications in addition to the Internet of Things. We are using frameworks such as Flutter to create the mobile application WellWisher. This app serves as a user interface for accessing and visualising the health data collected by the IoT devices. Through the WellWisher app, users can track their fitness progress, set goals, and receive personalised recommendations for improving their overall well-being.

1.2.1 IoT Architecture: Edge Computing

IoT architecture is crucial for fitness tracking, integrating multiple components to deliver a comprehensive health monitoring experience. It allows for better understanding with hardware, instant data presentation possible, data-driven personalisation, seamless device integration, and remote health monitoring, making it highly relevant to our project.

1. Data Collection:Microcontrollers and sensors in the wearable prototype gather critical health data from the user.
2. Data Processing:Collected data undergoes processing on the edge device, enabling rapid monitoring on database and feedback.
3. Data Transmission:Processed data is transmitted to a central server or cloud using protocols.
4. Data Analysis and Insights:Advanced data analysis techniques generate insights and personalised fitness plans.
5. User Interaction:Insights and recommendations are communicated back to the user's device, such as a smartphone app, facilitating user interaction and engagement.

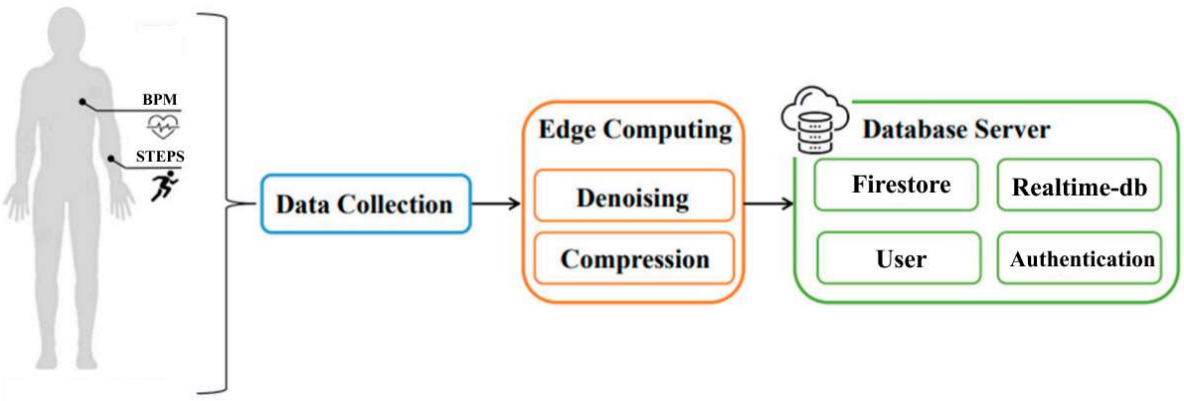

Fig 1.2.1 Reference Image for Edge Computing

The multiple benefits of implementing this architecture, such as fast monitoring, data-driven personalisation, seamless device integration, and remote health monitoring, justify its adoption. Our project's goal is to change how people monitor and improve their health by providing them with a complete and personalised fitness tracking solution that allows them to live better lives.

1.2.2 Research and Literature Review

Wearables and Internet of Things (IoT) Technologies for Fitness Assessment: This systematic review by Passos et al Ref.[6]. discusses the use of wearables and IoT technologies in sports for performance monitoring, evaluation, and fitness assessment. The study identifies and summarises recent studies that have used these technologies and discusses their applicability for fitness assessment. The results show that these technologies have been used not only for fitness assessment but also for monitoring the athlete’s internal and external workloads.

- An Exploration and Confirmation of the Factors Influencing User's Adoption of IoT-Based Wearable Fitness Trackers: This article explores the factors that influence the intentions of users that adopt IoT-based wearable fitness trackers Ref.[7]. Understanding these factors can help in designing and marketing your fitness tracking project.
- Tracking and Monitoring Fitness of Athletes Using IoT Enabled Wearables: This paper presents a theoretical model to integrate IoT and RF Algorithm to track and monitor fitness of athletes using wearables for activity recognition and performance prediction Ref.[8].

These studies provide a comprehensive overview of the current state of research in the field of fitness tracking using IoT technologies. They highlight the potential of these technologies in transforming personal wellness and physical activity, which aligns with the objectives of our project

1.3 Problem Formulation

After the COVID-19 Everyone is conscious about their health Ref.[5]. By Understanding the crucial value of information regarding health metrics (currently focusing) The Heart Rate & Steps, our project fills a significant gap in existing health monitoring systems. Our idea not only improves convenience by providing users with rapid feedback on their health data, but it also has the potential to save lives in crucial situations.

Also, our concept distinguishes itself by providing individualised health solutions that address each individual's specific needs and aspirations, raising it from a basic gadget to an invaluable health companion.

To reduce distractions and increase user engagement, the prototype of "FITNESS TRACKING INSIGHTS" will be screenless. A conscious design decision that demonstrates our commitment to providing affordable, seamless and focused health monitoring experience.

1.3.1 Problem Statement

Current fitness tracking solutions lack personalised recommendations and effective data display.

- Users serious about meeting fitness goals and monitoring health metrics face difficulties due to these distractions on screen.
- There exists a gap between current fitness tracking solutions and the needs of users seeking a more focused and personalised approach.

1.3.2 Significance

Fitness tracking is becoming more popular as people prioritise a healthier lifestyle. However, distracting screens limit the efficiency of existing fitness trackers in aiding consumers' fitness journeys.

Our proposal stems from the essential necessity of precise and personalised fitness tracking solutions. We hope to transform the way people track their fitness progress and improve their overall well-being by leveraging smart sensors and technology.

1.4 Recognition of Need

1.4.1 Market Inspection

Following extensive market research, it became clear that there is a growing demand for fitness tracking systems that are both effective and user-friendly. A complete solution that solves the drawbacks of current solutions and offers a smooth and user-friendly experience is what users are increasingly looking for.

1.4.2 User’s Need

Users are searching for a fitness tracking solution that goes beyond basic functionality and provides personalised recommendations based on their fitness levels, goals, and preferences. They also like being able to create and track fitness goals, as well as receive regular progress notifications. People appreciate an interface that is not only visually appealing but also simple to navigate and understand. A user-friendly layout improves the entire experience and encourages continuing use of the software.

1.4.3 Technological Advances

Recent advances in sensor technology, machine learning algorithms, and cloud-based infrastructure have created new opportunities for developing unique fitness tracking applications.

1.4.4 Industry Trends

The fitness sector is undergoing tremendous change, with a greater emphasis on digital health and wellness products. Wearable devices, smartphone applications, and cloud-based platforms are becoming more integrated, giving consumers easy ways to measure and assess their fitness progress. Keeping these industry trends in mind, the creation of Fitness Tracking Insights strives to capitalise on developing technology and market demands

1.5 Existing System

The current landscape of fitness tracking solutions presents several limitations that fail to meet the needs and preferences of users effectively. Traditional equipment such as Omron's BPM machine, while accurate, are often too complicated for the average user to operate comfortably. This complexity can lead to user frustration and ultimately discourages consistent usage.

- Fitness watches are often seen as status symbols rather than practical health monitoring devices.
- Their emphasis on aesthetics may compromise their ability to provide comprehensive fitness tracking features.
- Users may not receive the necessary functionality and accuracy required for effective health monitoring.
- They frequently include distracting screens and notifications that can disrupt the user experience.
- Concerns exist regarding user privacy and data security due to the potential for data leaks or unauthorised access to health information.

Similarly, fitness bands, while designed for health tracking, come with their own set of challenges. These devices often feature distracting screens and notifications that can detract from the user experience and hinder focus on fitness goals. Additionally, there are concerns regarding user privacy and data security, as fitness bands may be more prone to data leaks or unauthorised access to sensitive health information.

Overall, the existing system of fitness tracking solutions falls short in providing users with a seamless and user-friendly experience. The complexities, limited functionality, and privacy concerns associated with current options underscore the need for a more innovative and comprehensive solution like Fitness Tracking Insights.

1.6 Objectives

- 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.

1.6.1 Overview of Achieved Objectives

- **Developed a Functional Prototype of a Wearable Band:** The first objective of the project was to develop a functional prototype of a wearable band. This wearable band serves as the primary hardware component of the fitness tracking system. It incorporates essential sensors, such as heart rate monitors, accelerometers, and temperature sensors, to collect comprehensive health data from the user. The prototype is designed to be lightweight, comfortable to wear, and unobtrusive, ensuring user comfort and convenience during physical activities. Additionally, the wearable band features a sleek and ergonomic design, making it aesthetically pleasing and suitable for everyday use.
- **Developed a Dedicated Mobile Application:** The second objective focused on developing a dedicated mobile application, known as the WellWisher app. This app serves as the user interface for accessing and interpreting the health data collected by the wearable band. The "WellWisher" app features various profile views and a customizable dashboard that provides users with quick insights into their fitness metrics. Users can view their daily activity levels, monitor their heart rate trends, track their sleep patterns, and receive personalised recommendations for improving their overall health and well-being. The app is designed to be intuitive and user-friendly, ensuring that users can easily navigate through its features and access the information they need to achieve their fitness goals.
- **Validated the Reliability and Precision of the Hardware Device:** The third objective aimed to validate the reliability and precision of the hardware device developed for the fitness tracking system. This validation process involved comparing the performance of the wearable band with existing hardware devices available in the market. Through rigorous testing and analysis, we assessed the accuracy of the sensors integrated into the wearable band, evaluated its consistency in capturing health data, and verified its reliability in various real-world scenarios. The validation results provided valuable insights into the effectiveness of the hardware device and its suitability for use in fitness tracking applications.

Overall, achieving these goals marks an important step forward in the Fitness Tracking Insights project's growth. We have successfully designed and implemented a functional prototype of the wearable band, developed a dedicated mobile application, and validated the hardware device's reliability, laying the groundwork for an effective fitness tracking solution.

1.7 Proposed System

The proposed system, “Fitness Tracking Insights” is an integrated solution that combines cutting-edge hardware and software technologies to revolutionise the way individuals track and manage their fitness and health metrics. Building upon the limitations of existing fitness tracking systems, the proposed system offers a comprehensive approach to health monitoring, personalised recommendations, and user engagement.

1.7.1 Hardware Component

The hardware component of the proposed system consists of a wearable band equipped with advanced sensors, including heart rate monitors, accelerometers, and temperature sensors. This wearable band is designed to quickly collect health data from the user during various physical activities, such as exercise, and daily routines. The sleek and ergonomic design of the wearable band ensures user comfort and convenience, allowing for continuous monitoring without interference.

1.7.2 Software Component

The software component of the proposed system comprises the WellWisher mobile application, developed on cross-platform development frameworks such as Flutter. The WellWisher app serves as the user interface for accessing and interpreting the health data collected by the wearable band. Through intuitive profile views and a customizable dashboard, users can track their fitness metrics, monitor their progress towards health goals, and receive personalised recommendations for improving their well-being.

Key Features:

1. Personalized Health Insights: The WellWisher app provides users with personalised insights into their health and fitness metrics, allowing them to understand their current status and track their progress over time.

2. Fast Monitoring: The wearable band continuously monitors the user's vital signs, including heart rate, activity levels, sleep patterns, and more, providing immediate feedback and alerts as needed.

3. Personalised Experience: Based on the user's health data and goals, We can generate personalised recommendations for exercise routines, dietary habits, sleep patterns, and lifestyle changes to optimise overall health and well-being.

4. Seamless Integration: The proposed system seamlessly integrates with existing fitness tracking technologies, allowing users to sync their data across multiple devices and platforms for a unified experience.

In summary, the suggested approach provides a comprehensive and user-friendly solution for fitness tracking and health management, allowing people to take control of their health and achieve their goals with confidence and ease.

1.8 Unique features of the proposed system

Following are listed some Unique and Key features of the proposed system.

- Screenless Design: The proposed system features a screenless wearable band, eliminating distractions and enhancing user focus during fitness activities. This thoughtful design choice prioritises user engagement and convenience, distinguishing it from traditional fitness trackers with screen-based interfaces.

- Affordable Pricing: With sufficient production funding, the proposed system aims to offer an affordable solution for fitness tracking and health management. By minimising production costs and optimising manufacturing processes, the system seeks to make advanced health monitoring technology accessible to a wider audience.

- Cross-Platform Compatibility: The proposed system includes a dedicated mobile application developed on cross-platform development frameworks, such as Flutter. This cross-platform compatibility ensures that users can access their health data and track their fitness progress seamlessly across different devices and operating systems.

- Fitness-Centric Approach: Unlike generic health monitoring solutions, the proposed system is specifically designed for fitness-oriented individuals who prioritise health and wellness. The dedicated app offers tailored features and recommendations for exercise routines, dietary habits, and lifestyle changes, catering to the unique needs and goals of fitness enthusiasts.

Chapter 2: Requirement Analysis and System Specification

2.1 Feasibility study

The feasibility study assesses the "Fitness Tracking Insights" project's viability in terms of technological, economic, operational, and scheduling issues. It evaluates costs, resources, market demand, and technological feasibility in order to make informed decisions. This analysis assures effective resource allocation and project completion within specified restrictions.

2.1.1 Technical Feasibility

The technical feasibility of the project is high. The hardware components, including the ESP32 microcontroller, MAX30102 heart rate sensor, and MPU9250 accelerometer sensor, are readily available and easy to integrate. The firmware can be programmed using MicroPython, which is a simple and lightweight programming language. The data transmission mechanism is based on Wi-Fi, which is a widely available and reliable wireless communication protocol. Firebase, the cloud-based database, provides a scalable and secure platform for storing and retrieving data. The mobile application can be built using Flutter Flow or Flutter, which provides a robust and user-friendly development environment.

- Availability and suitability of microcontrollers and sensors: The project requires microcontrollers compatible with sensor integration, such as ESP32 for data processing and transmission.
- Capability of edge devices: Smartwatches and fitness bands must efficiently process collected data.
- Availability and reliability of data transmission protocols: Reliable protocols like HTTP, WiFi and Internet availability
- Technical skills: Adequate expertise in hardware and software development is necessary for successful implementation and maintenance.

2.1.2 Operational Feasibility

The operational feasibility of the project is also high. The wearable device is designed to be lightweight and comfortable to wear, making it easy for users to wear it throughout the day. The device can be powered by a small battery, which can last for several days depending on usage. The data transmission mechanism is automatic, and the mobile application provides data visualisation, making it easy for users to monitor their health and fitness goals.

- Compatibility with existing practices: The system integrates seamlessly with the user's daily routine and existing fitness tracking habits.
- User-friendliness: The interface must be intuitive and easy to use, minimising the need for extensive training.
- Training requirements: Minimal training is enough for users to navigate and utilise the system effectively.

2.1.3 Economic Feasibility

After conducting the extensive economic feasibility study for our project, We come at conclusion that our project is economically viable for 3 of us. The hardware components come under our budget and can be easily sourced from online e-commerce platforms. The programming language used, MicroPython, is open-source and free to use.

- Costs: Possible to Assess the expenses associated with hardware, software, development, and operational overheads.
- Financial resources Secure funding or allocate budgets to cover project costs and contingencies.
- Revenue: The potential streams from subscription models, partnerships, and from additional services the revenue is obtainable

2.1.4 Schedule Feasibility

- Project timeline: The 3 main milestones are achieved by following
- Initialization of Project is From Jan-2024 April-2024 and allocate time for planning, development, testing, and deployment.
- Team availability: Ensure team members are committed and available throughout the project duration.
- Risk assessment: Identify potential risks and mitigation strategies to minimise delays and ensure timely completion.

Each phase is represented by a bar, and the percentage completion of each phase is indicated. The timeline provides a visual representation of the project's progress and helps in tracking key milestones and deadlines.

The Fig 2.1.4 depicts a timeline representing the project schedule for a fitness tracking project. It is divided into months from January 2024 to July 2024.

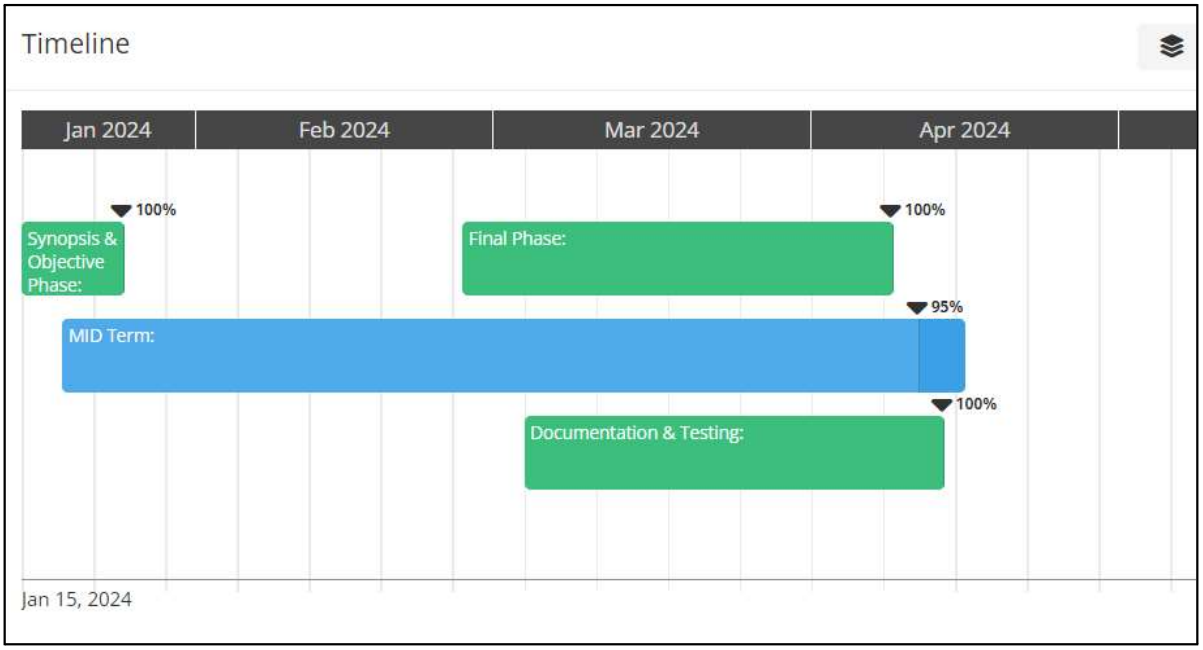

Fig 2.1.4 Project Timeline

- Key milestones and phases of the project are highlighted on the timeline:
- Synopsis & Objective Phase: This phase, marked as 100% complete, likely involved the initial planning and defining of project objectives.
- MID Term: This phase, currently in progress, appears to be a midterm evaluation or checkpoint in the project timeline.
- Final Phase: This upcoming phase indicates the final stages of the project, which is involves wrapping up development, conducting final testing, and preparing for deployment.
- Documentation & Testing: This phase, marked as 100% complete, likely involved documenting project details and conducting testing to ensure the system functions correctly.

2.1.5 Resource Feasibility

- Skilled personnel: Assess the availability of experts in hardware engineering, App development, and project management.
- Hardware and software availability: We Ensured access to necessary components, tools, and development environments mentioned in 2.2 SRS Document.
- After completing the project the overall progress is also tracked.

An overview of the project's progress and resource allocation mentioned in Fig 2.1.5, indicating that it is nearing completion and that the allocated resources have been utilised effectively.

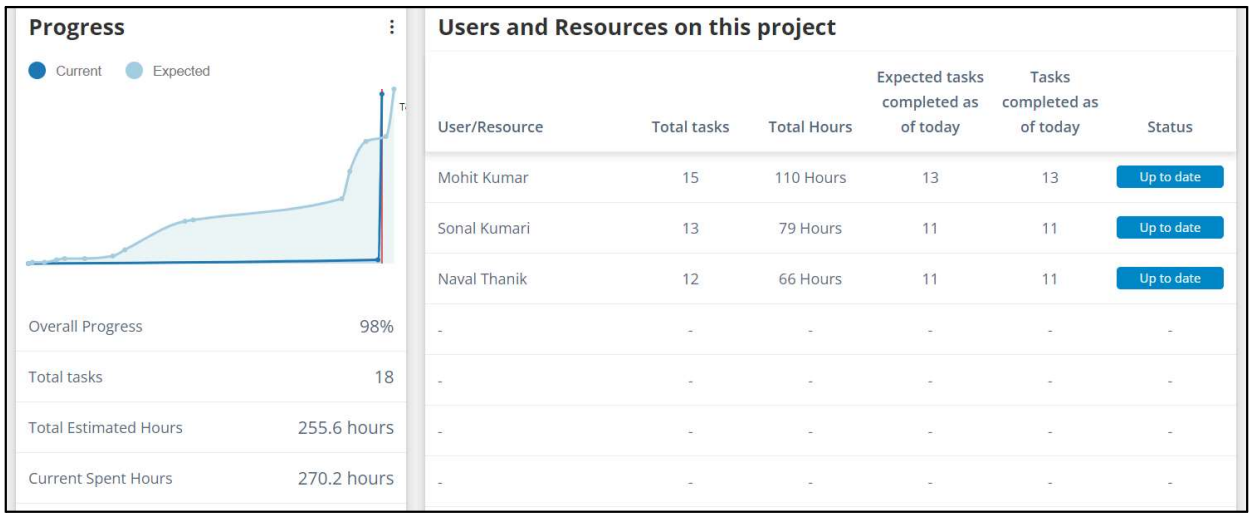

Fig 2.1.5 Progress View of Project

This image shows the progress and resource allocation for a project.

1. **Progress Chart:**
    - The blue line represents the actual progress of the project, while the light blue line represents the expected progress.
    - The overall progress is indicated as 98%, suggesting that the project is near completion.
    - Total tasks completed so far are 18 out of the total estimated tasks.
    - The total estimated hours for the project are 255.6 hours, with 270.2 hours already spent.
2. **Users and Resources on this Project:**
    - The table lists the users or resources allocated to the project along with their respective tasks and hours.
    - Each user's expected tasks, tasks completed as of today, and their status are provided.
    - The status for all users is "Up to date," indicating that they have completed their expected tasks as of today.

2.2 Software Requirement Specification Document

This document describes the software requirements for the "Fitness Tracking Insights" project. The project's scope comprises data collecting, processing, analysis, and display to users to enable effective health monitoring. Some of the precise requirements that we must achieve in order to develop this project.The primary objective of this project is to create a robust and user-friendly

platform that not only tracks but also interprets various health metrics. By doing so, we aim to provide users with actionable insights into their health status and progress towards their fitness goals.

In order to successfully develop and implement this project, there are several precise requirements that we must fulfil. These requirements will be further elaborated in the subsequent sections of this document.

A. Project Context

We defined the need for these requirements in the context of the project. which is situated within the context of the health and wellness industry, where there is a growing demand for effective fitness tracking solutions. Existing solutions are often misleading, also lack precision and user-friendly interfaces, At the end the user is dissatisfied with the outcome. The "Fitness Tracking Insights" project aims to address these shortcomings by providing a seamless and intuitive health monitoring experience.

- To develop the Hardware Prototype some of the High performing Microcontroller, Sensors and Other Electronic components are needed.
- To develop a Software Module technical skill and proper System requirement needs to be fulfilled.

B. Product Perspective

We divided our project into two different modules, Hardware Module includes ESP32, PCB, Sensors etc. and Software Module includes The WellWisher application based on Flutter Framework. The hardware collects data on user's physical activity and health metrics. The mobile application serves as a user interface, providing immediate insights and so the recommendations made based on the collected data.

Product Functions: The key functions of the system includes,

- Data collection from sensors embedded in wearable prototypes.
- Data transmission to a firebase server for further transmission to App and Analysis.
- Data processing on edge devices for Dashboard monitoring.
- On User’s end the data can be analysed to achieve personalised insights and recommendations.
- User interaction through a mobile application for viewing and managing health metrics.

2.2.1 Functional Requirements

The Hardware Module must perform these actions,

- Sense data from sensors accurately, including beats per minute and step count.
- Transmit the sensed values of BPM and step count to Firebase.
- Utilise the internet connection provided by the user's mobile hotspot for data transmission.
- Ensure that the data transmission process is reliable and secure.
- Handle potential network disruptions and resume data transmission seamlessly.

a. Data Collection: Sensors provide data and microcontrollers collect data on physical activity, heart rate, step patterns, timestamps etc. these health metrics from wearable prototypes.
b. Data Processing: The data further transmit to firebase to process collected data on edge devices for dashboard monitoring and feedback.
   Implement algorithms for data filtering and analysis to extract meaningful insights.
c. Data Transmission: Transmit processed data to a central server or cloud platform securely and efficiently.
   Utilise reliable communication protocols like HTTP and Wifi.
d. Analyze collected data to provide personalised insights and recommendations to users.
e. Present insights in an easy-to-understand format through the mobile application.
f. User Interaction: Developed an intuitive user interface for the Flutter application name “WellWisher”, allowing users to view and manage their health metrics.

2.2.2 Performance Requirement

The system meets the following performance requirements:

- The wearable device collects heart rate and step count data with at least 74% accuracy
- The data transmission mechanism is reliable, with a success rate of at least 90%
- The mobile application provides data visualisation with a latency of no more than 1 second
- The mobile application loads within 2 seconds on average.
- Data transmission from wearable sensors to the central server has a latency of less than 500 milliseconds.
- The system is able to handle concurrent requests from multiple users without experiencing performance degradation

Ensure fast and responsive performance of the mobile application, even under heavy load. Minimise latency in data transmission and processing to provide immediate response to users.

2.2.3 Dependability Requirement

Ensure the reliability and availability of the system for continuous health monitoring. Implement mechanisms for error detection, recovery, and fault tolerance.The system requires a reliable Internet which is core dependability

- The system have at least 99.9% uptime
- The system is be able to recover from any hardware or software failure within 5 minutes
- The system provides data backup and restore functionality to protect against data loss

2.2.4 Maintainability Requirements

The system is designed in a modular and extensible manner to facilitate future updates and enhancements. Provides comprehensive documentation for system architecture, components, and APIs. also for ease of extensibility

- The code is well-documented and adhere to industry best practices
- The system support future software upgrades without any significant changes to the existing codebase
- The system provides a mechanism for system administrators to update and deploy firmware updates to the wearable devices

2.2.5 Security Requirements

Implemented robust security measures to protect user data from unauthorised access or tampering. Encrypt data transmission and storage to prevent data breaches or leaks with the help of firebase.

- The system use google encryption standards to protect data transmission between the wearable device and the cloud-based database
- The App use authentication and authorization mechanisms to prevent unauthorised access to user data
- The project comply with industry security standards and best practices

2.2.6 Look and Feel Requirements

The user interface of the mobile application is designed by considering visually appealing and user-friendly. Also Ensured consistency in design elements and layout across different screens and devices.

- The WellWisher application has material UI with modern and intuitive user interface design
- The colours, fonts, and graphics chosen for the App with golden ratio and extensive analysis that provide an easy-to-use navigation system and a clear hierarchy of information.

2.2.7 User Characteristics

The system is designed for users who are interested in monitoring their fitness progress and improving their overall health. Users may vary in age, fitness level, and health goals but share a common desire for personalised health insights.

2.2.8 Constraints, Assumptions, and Dependencies

- Constraints: The system must comply with data privacy regulations and ensure the security of user data. The availability of reliable internet connectivity may affect the immediate transmission of data.
- Assumptions: Users will have access to compatible wearable devices and smartphones. The mobile application will be compatible with popular operating systems.
- Dependencies: The successful integration of hardware sensors with the mobile application relies on the availability of compatible software libraries and APIs.

2.3 SDLC Model : Agile Methodology

The Agile model provides us with iterative development, enabling continuous enhancement in small increments to deliver early value and adapt quickly to changing needs. By prioritising user collaboration and embracing changes, Agile ensures user-centred development, effective risk management, and high-quality software through transparent communication and continuous improvement practices. In the following points each we defined each phase of our “FITNESS TRACKING INSIGHTS” project using the Agile model:

Phase 1- Plan:

Start by gathering requirements and creating user stories. For Instance, “As a user, I want to monitor my heart rate so that I can track my health.” Prioritise these user stories based on their value to the user and the project’s objectives.

Phase 2- Design:

Design the system architecture and user interface based on the user stories. For our project, this involves designing the data collection, processing, transmission, analysis, and user interaction components.

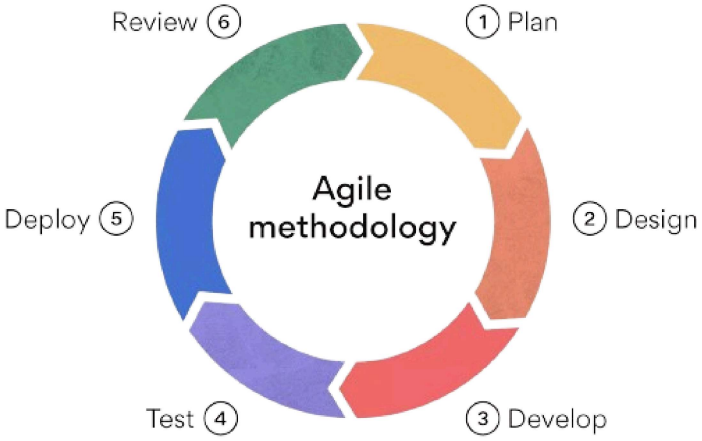

Fig 2.3 Agile Model block diagram

Phase 3- Development:

Developed the system in small, manageable units of work (or "sprints"). Each sprint results in a working version of the product that implements one or more user stories. Used paired programming, code reviews etc. These were the development practices to ensure high-quality code.

Phase 4- Test:

Continuously test the system throughout the development process. This includes unit testing, integration testing, and system testing. Use test-driven development (TDD) practices, where tests are written before the code, to ensure that all code is tested.

Phase 5- Deployment:

At present there was no deployment platform, but we regularly updated on the github and the working versions of the product to a staging environment for acceptance testing. Once the team accepts testing is The final code and considered passed, it will be uploaded to the Github.

Phase 6- Review:

After each sprint, hold a review meeting with our project guide and our fellow classmates in college to demonstrate the working product and gather feedback.We used their feedback to plan the next sprint and to continuously improve the project.

In conclusion, the Agile model presents an effective framework for the development of our fitness tracking project, providing & quality assurance the development lifecycle.

Chapter 3: System Design

3.1 Design Approach

For the development of our project, the most suitable design approach is Object-Oriented Design (OOD). Object-Oriented Design focuses on organising software systems around objects, which are instances of classes that encapsulate data and behaviour. This approach aligns well with the modular and component-based nature of our project, allowing for better manageability, reusability, and scalability of code.

The details of Object-Oriented Design in the context of our project:

1. Identify Objects: Identify the main entities or objects within the fitness tracking system, such as User, Sensor, DataProcessor, Database, and UserInterface.
2. Define Classes: Create classes to represent each identified object. For example, the User class has attributes like username, age, and fitness goals, while the Sensor class has attributes like sensorID and sensorType.
3. Establish Relationships: Define relationships between objects using associations such as aggregation and composition. For instance, the User object aggregates Sensor objects to track health metrics.
4. Encapsulate Behavior: Define methods within classes to encapsulate behaviour or functionalities associated with each object. For example, the Sensor class has methods for data collection, while the UserInterface class also has methods for displaying data to the user.
5. Implement Inheritance: Utilise inheritance to create hierarchies of classes and promote code reuse. For instance, different types of sensors inherited from a common Sensor class.
6. Ensure Modularity: Design the system to be modular, with each class encapsulating a specific set of functionalities. This allows for easier maintenance and modification of code.
7. Implement Design Patterns: Incorporate design patterns such as Observer pattern for data updates and Singleton pattern for managing database connections efficiently.

By following the Object-Oriented Design approach, we aim to create a well-structured and maintainable fitness tracking system that effectively meets the requirements of our targeted users while facilitating In-future enhancements and modifications as well

3.2 Detail Design

Fitness Tracking Insights involves the collection, processing, and analysis of health data from various sources, including sensors and actuators. Here's how our project aligns with the depicted architecture:

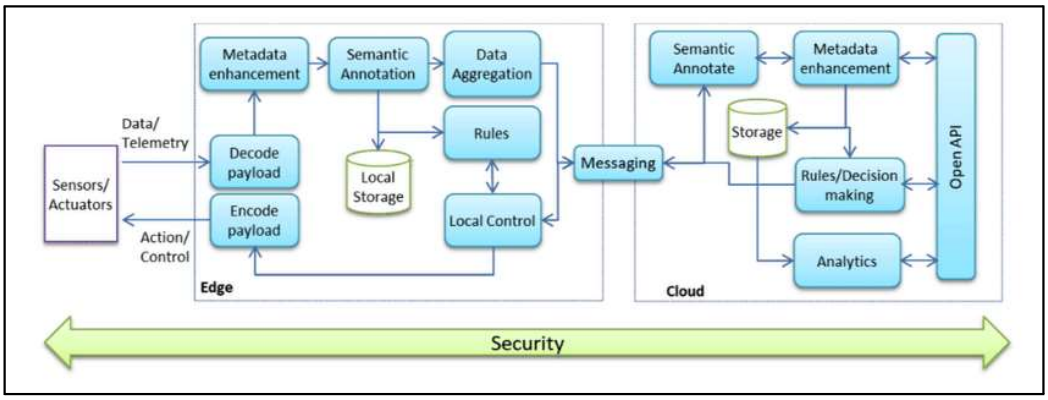

Fig 3.2 IoT Architecture in Project

3.2.1 Edge Design

- Data/Telemetry: Our project begins with the collection of health data, including heart rate, step count, and other relevant metrics, using sensors and actuators worn by the user.
- Edge: The collected data is then processed at the edge, where it undergoes decoding, encoding, and local storage. This ensures that data is prepared for transmission and stored securely before being sent to the cloud.
- Cloud: Once processed at the edge, the data is transmitted to the cloud for further processing and analysis. Here, the data undergoes semantic annotation, data aggregation, and metadata enhancement to extract valuable insights and patterns.
- Security: Throughout the entire process, security measures are implemented to protect the integrity and confidentiality of the health data. This includes encryption during transmission, access control mechanisms, and compliance with security standards.

3.2.2 Cloud/ FirebaseDesign

- Sensors/Actuators: Wearable devices equipped with sensors to capture health data.
- Edge Processing: Microcontrollers and edge devices process the collected data locally before transmission.
- Local Storage: Temporary storage of processed data at the edge for resilience and reliability.

- Cloud Processing: Centralised processing and analysis of data in the cloud for deeper insights.
- Security Measures: Implementation of encryption, access control, and compliance with security standards to protect user data.

This architecture and design, our project ensures efficient and secure handling of health data, enabling users to monitor their fitness progress effectively while safeguarding their privacy and security.

3.3 Design Diagrams

There are various diagrams and design architectures included in the section to explain the in-depth and detailed view of our "Fitness Tracking Insights".

3.3.1 Block Diagram

1. **Prototype Band:**
    - The hardware component of the system consists of an ESP32 microcontroller, MPU9250 accelerometer, and MAX30102 heart rate sensor.
    - Powered by a 3.7V battery.
    - Communicates with the mobile application via UART (Universal Asynchronous Receiver-Transmitter) for data transmission.
2. **User Interface:**
    - Consists of login and registration buttons, allowing users to access the system.
    - Displays the "WellWisher" logo, representing the App identity.
    - Provides access to the main features of the mobile application: Dashboard, BPM View, Steps View, and Profile View.
3. **Database and Firebase:**
    - Represents the backend infrastructure of the system, including a database for storing user data and Firebase for proper data synchronisation.
    - Enables secure storage and retrieval of user information, health metrics, and application settings.
4. **Dashboard:**Central hub of the mobile application, presenting users with an overview of their fitness metrics and progress towards their goals. Integrates data from the Prototype Band and Firebase, providing updates on heart rate, step count, and other health parameters.

5. **Navigation Views:** Individual views within the mobile application for accessing specific health data and user profiles. BPM View displays heart rate information, Steps View shows step count data, and Profile View allows users to view and manage their personal information.

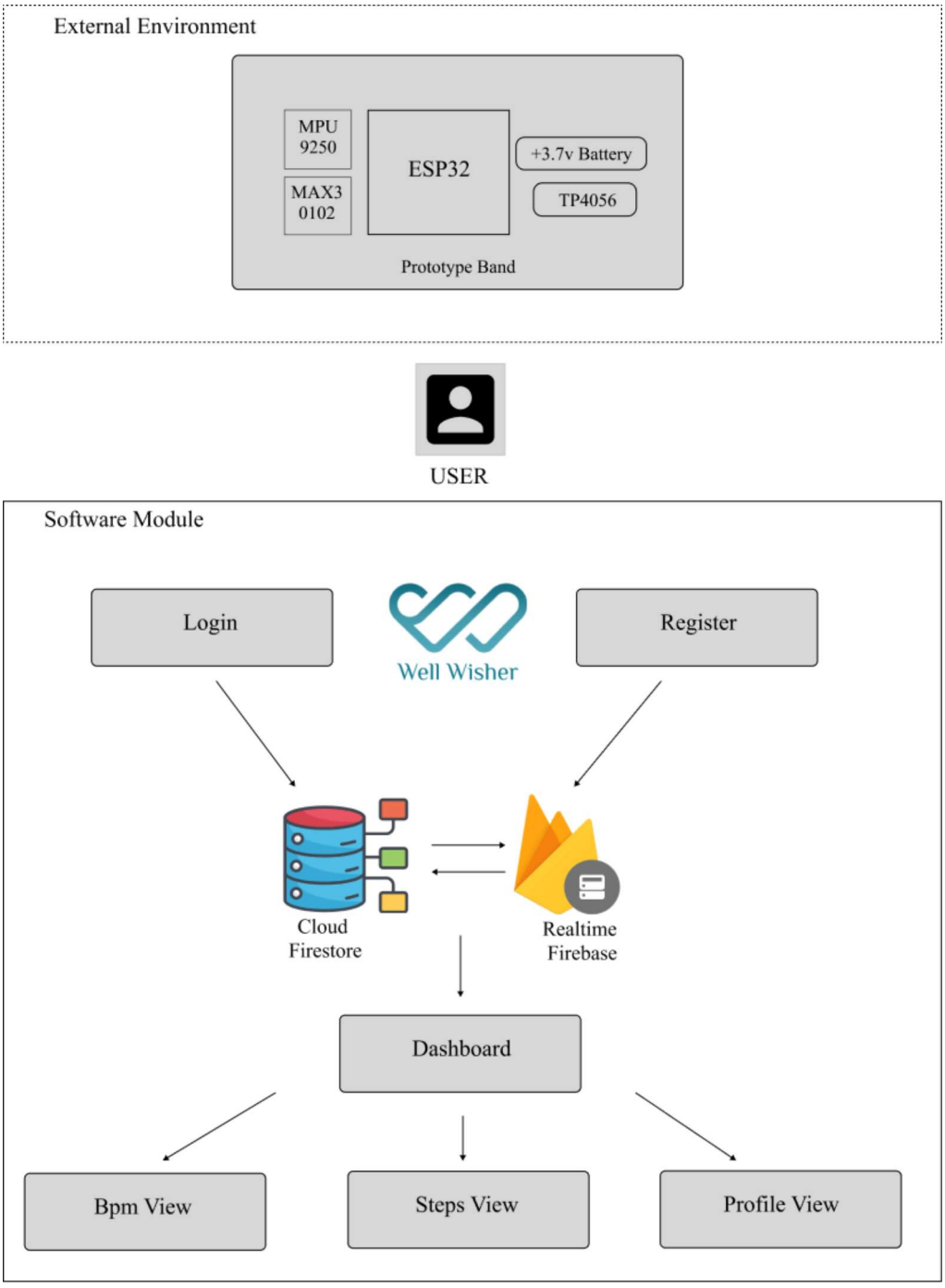

Fig 3.3.1 Block Diagram of the project

This block diagram illustrates the key components and interactions within the system is designed

Working

- The Prototype Band collects data on the user's physical activity and health metrics using the embedded sensors (MPU9250 and MAX30102).
- This data is transmitted wirelessly to the mobile application via UART communication.
- The mobile application processes and displays the received data in various views, such as the Dashboard, BPM View, Steps View, and Profile View.
- Users can interact with the application to view their fitness metrics, set goals, and track their progress over time.
- The backend infrastructure (Database and Firebase) ensures secure storage and synchronisation of user data, providing a seamless and reliable user experience.

3.3.2 Flow Chart

This flowchart outlines the sequence of actions that users can take within the WellWisher application, providing a clear and structured navigation path for accessing different features and functionalities.

1. Open the WellWisher mobile application on the device.
2. Proceed to the login screen.
3. Enter the username and password to access your account.
4. Upon successful authentication, User will be directed to the Dashboard.
5. From the Dashboard, we can navigate to different sections of the app, such as viewing our fitness data or accessing our profile.
6. To view step count, tap on the "STEPS" icon.
7. To view heart rate data, tap on the "BPM" icon.
8. If the user wishes to manage their profile, tap on the "Profile Page" icon.
9. Additionally, users can access other options by tapping on the "Options" icon.
10. From the options menu, users can choose to log out of the app by tapping on the "Logout" option.
11. Users can also update the profile information by selecting the "Update Profile" option.

After completing the desired actions, users can exit the app or continue exploring its features as needed.

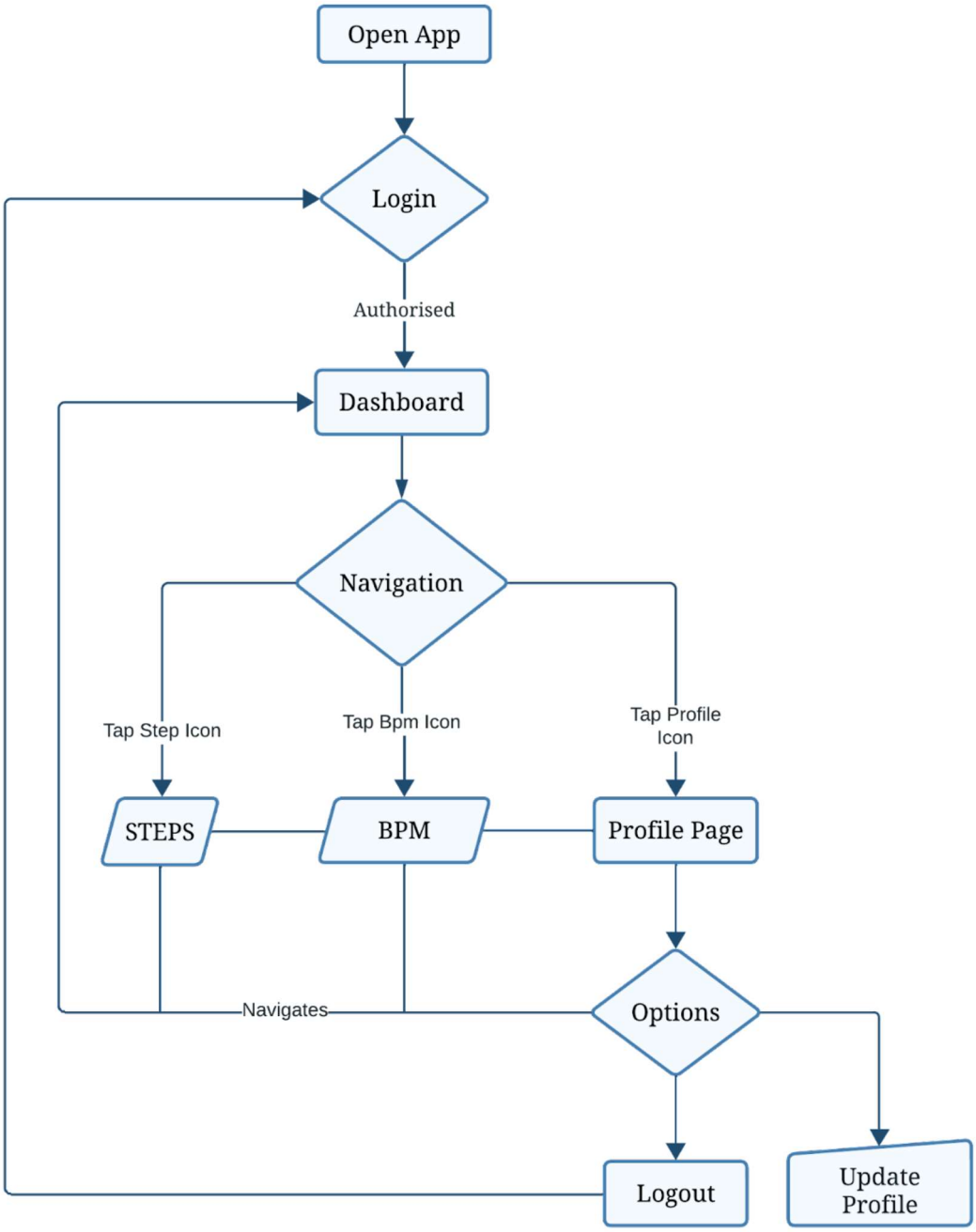

Fig 3.3.2 App Flow chart

These steps provide clear guidance for users on how to navigate through the WellWisher application. By following these intuitive steps, users can easily navigate the app and make the most out of its capabilities for tracking their fitness progress and managing their profile settings. This user-friendly approach enhances the overall user experience, making WellWisher a valuable tool for individuals striving to improve their health and wellness.by utilise its various functionalities.

3.3.3 UML Diagram

This UML diagram is important as it provides a visual representation of the system's architecture and component interactions in the fitness tracking project. It helps in understanding the relationships between different elements of the system, such as the microcontroller, sensors, and Firebase database.

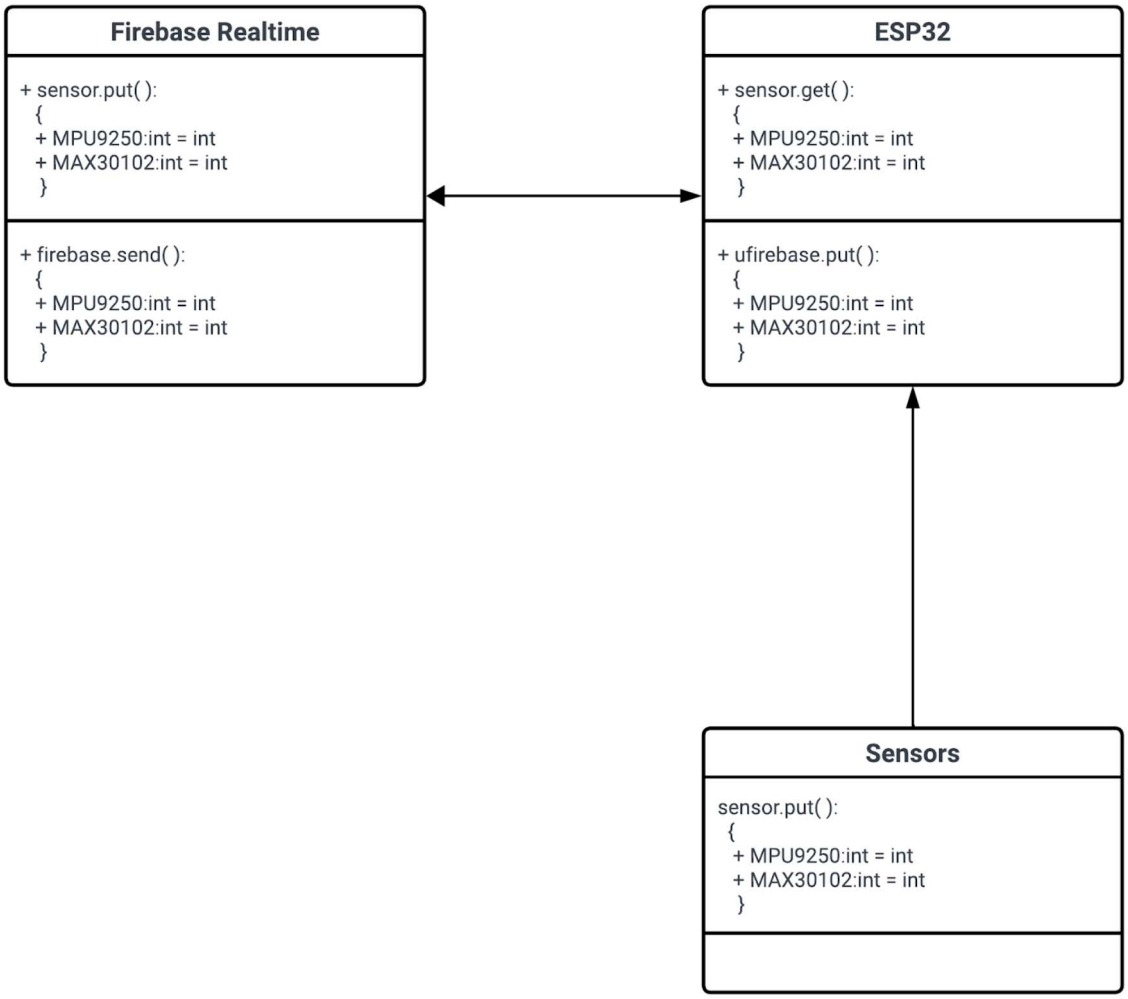

Fig 3.3.3 UML Diagram

The "Firebase Realtime" component represents Google's Firebase Realtime Database, which is used to store and synchronise sensor data in rapidly. It has methods for putting sensor data into the database and sending data to the cloud. The ESP32 component represents the ESP32 microcontroller, which is responsible for retrieving sensor data and interacting with the Firebase Realtime Database. It has methods for getting sensor data and putting data into the Firebase database.

The "Sensors" component represents the sensors used in the project, such as the MPU9250 and MAX30102 sensors. It defines the data structure for sensor data and the method for putting sensor data into the database.

Overall, this diagram shows how sensor data is collected by the ESP32, processed, and stored in the Firebase Realtime Database. It highlights the interaction between different components in the system and the flow of data between them.

3.3.4 Data Flow Diagrams

Level 0 DFD:

The Abstract Data Flow Diagram of WellWisher App is given below.

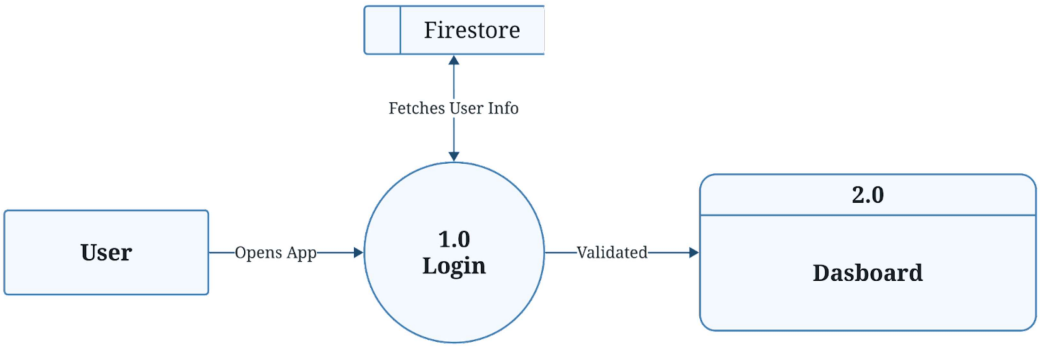

Fig 3.3.4 (a) Level 0 DFD of App

This Level 0 Data Flow Diagram (DFD) illustrates the flow of information within the system of the fitness tracking application:

User: The process "User" represents the user interacting with the application. It starts by opening the app, triggering the login process.

1.0 Login:

- The process "1.0 Login" handles the user authentication process.
- It fetches user information from Firestore, which likely includes credentials for validation.

Once the user is validated, the process moves to the next stage.

2.0 Dashboard:

- The process "2.0 Dashboard" represents the user interface where the user lands after successful login. And It displays the main dashboard of the application, providing various functionalities related to fitness tracking.

Firestore:

- Firestore is a cloud-based database used to store user information, including login credentials and other relevant data.
- It interacts with the login process to fetch user information for validation.

The Level 0 DFD outlines the core processes involved in user authentication and navigation within the fitness tracking application. It demonstrates the flow of data from the user to the authentication process and then to the main dashboard for further interaction.

Level 1 DFD:

This Level 1 Data Flow Diagram (DFD) provides a more detailed view of the processes involved in the fitness tracking application:

1. User: Represents the user interacting with the application.
2. Login:
    - Process for user authentication.
    - Checks user credentials for login.
3. Validation:
    - Sub-process of login.
    - Validates user credentials.
4. Forgot Password:
    - Allows users to reset their password if forgotten.
    - User enters their registered email for password reset.
5. Registration:
    - Process for new user registration.
    - User provides details for registration, and information is stored in the Firestore database.
6. Dashboard:
    - Represents the main dashboard of the application.
    - Users are redirected here after successful login or registration.
    - Provides various features and functionalities related to fitness tracking.
7. Update Profile: Allows users to update their profile information.
    - Users can modify their profile details as needed.

8. Log out: Process for user logout from the application.

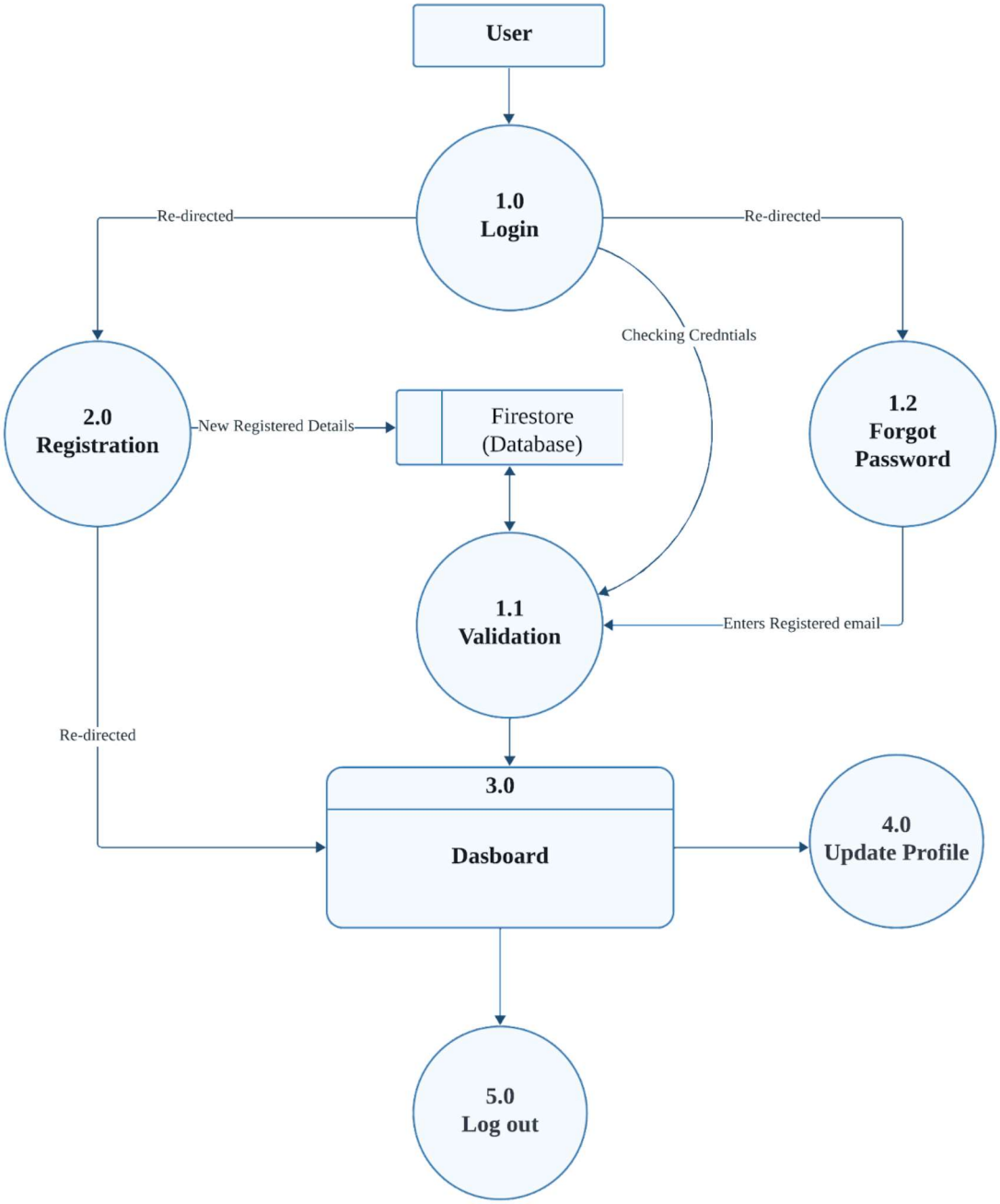

Fig 3.3.4 (b) Level 1 DFD of App

Contrasting with the Level 0 DFD, this diagram provides a more granular view of the login process, including validation and password reset functionalities. It also includes the registration process for new users and features for updating profile information and logging out. This level of detail helps in understanding the specific actions and interactions within the application's user management system.

Level 2 DFD:

This Level 2 Data Flow Diagram (DFD) provides a fully detailed view of the processes involved in the project:

1. User:
    - Represents the user interacting with the application.
2. 1.0 Login:
    - Process for user authentication.
    - Checks user credentials for login.
3. 2.0 Forgot password:
    - Allows users to reset their password if forgotten.
    - User enters their registered email for password reset.
4. 3.0 Registration:
    - Process for new user registration.
    - Includes sub-processes such as validation of user input.
5. 4.0 Dashboard:
    - Represents the main dashboard of the application.
    - Provides detailed sub-processes like fetching heart rate and steps taken data.
6. 4.1 Heart Rate:
    - Fetches heart rate data from the database.
7. 4.2 Steps Taken:
    - Fetches steps taken data from the database.
8. 4.3 Profile:
    - Allows users to view and update their profile information.
9. 4.3.1 Update Profile:
    - Process for updating user profile details.
10. 4.3.2 Logout:
    - Process for user logout from the application.
11. 5.0 Error Handler:
    - Handles errors encountered during various processes.
    - Includes sub-processes for displaying different error types like wrong input format or email not found.

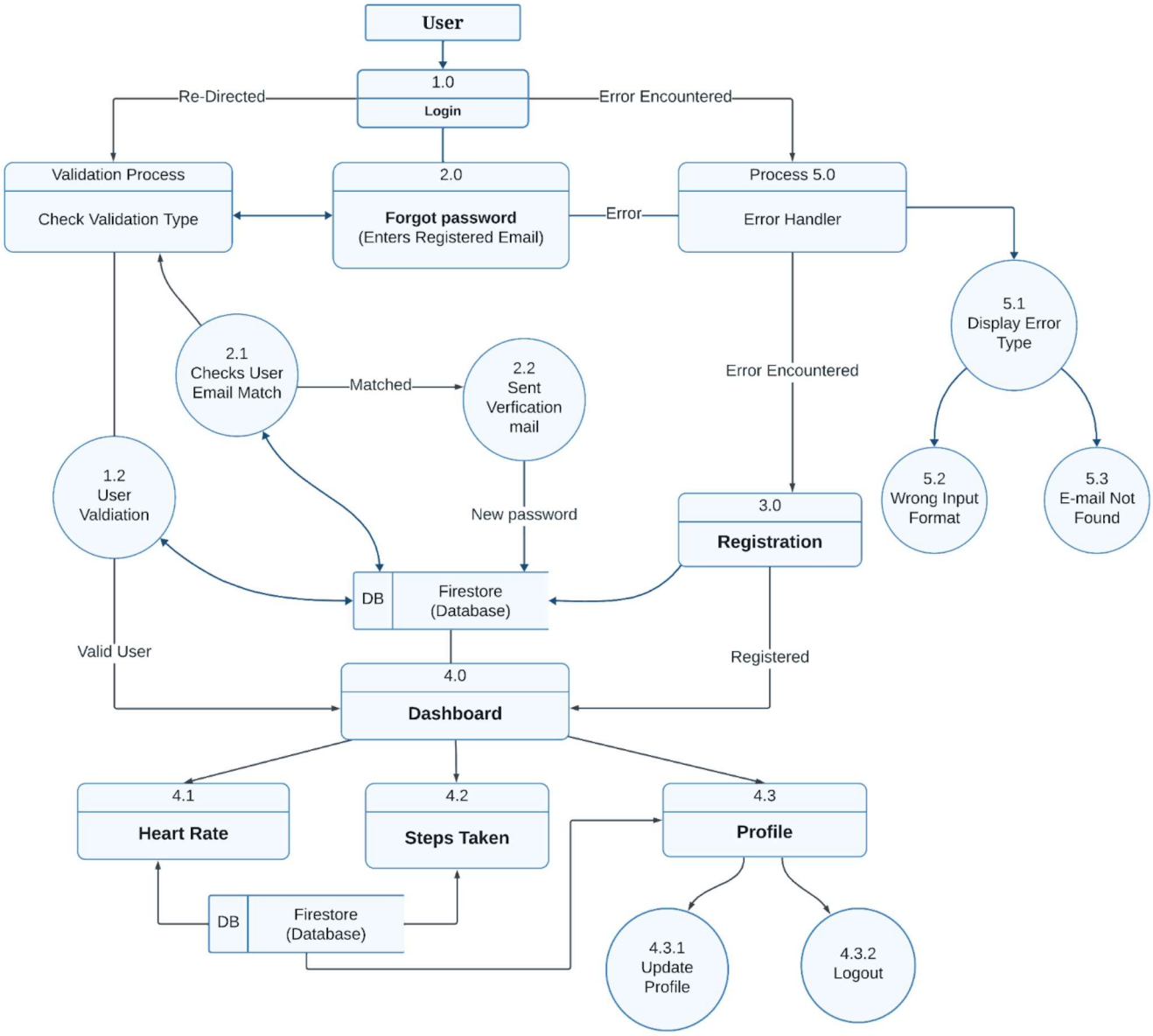

Fig 3.3.4 (c) Level 2 DFD of App

In comparison to the Level 1 DFD, this diagram provides an even more detailed breakdown of each process, including specific sub-processes and error handling mechanisms. It helps in understanding the flow of data and operations within the application at a more granular level, facilitating the development, testing, and maintenance of the application. The significance of this Level 2 DFD lies in its ability to provide a comprehensive blueprint of the application's functionality, aiding in the effective design, implementation, and management of the fitness tracking system.

3.3.5 Database Design

We are using a NoSQL database, which means it is a schema-less database. That means the data is NOT stored in the table format. we actually don't have any restrictions on how we store our data. The Firestore database uses the collection-document model to store the data.

Key terms of Firestore NoSQL Database:

- Collection: A collection is simply a set of 'documents'.
- Document: A document is a record that contains the 'fields'.
- Fields: The key-value pairs inside the document are called 'fields' e.g., name, place, age, etc.

To better understand, here is the figure below:

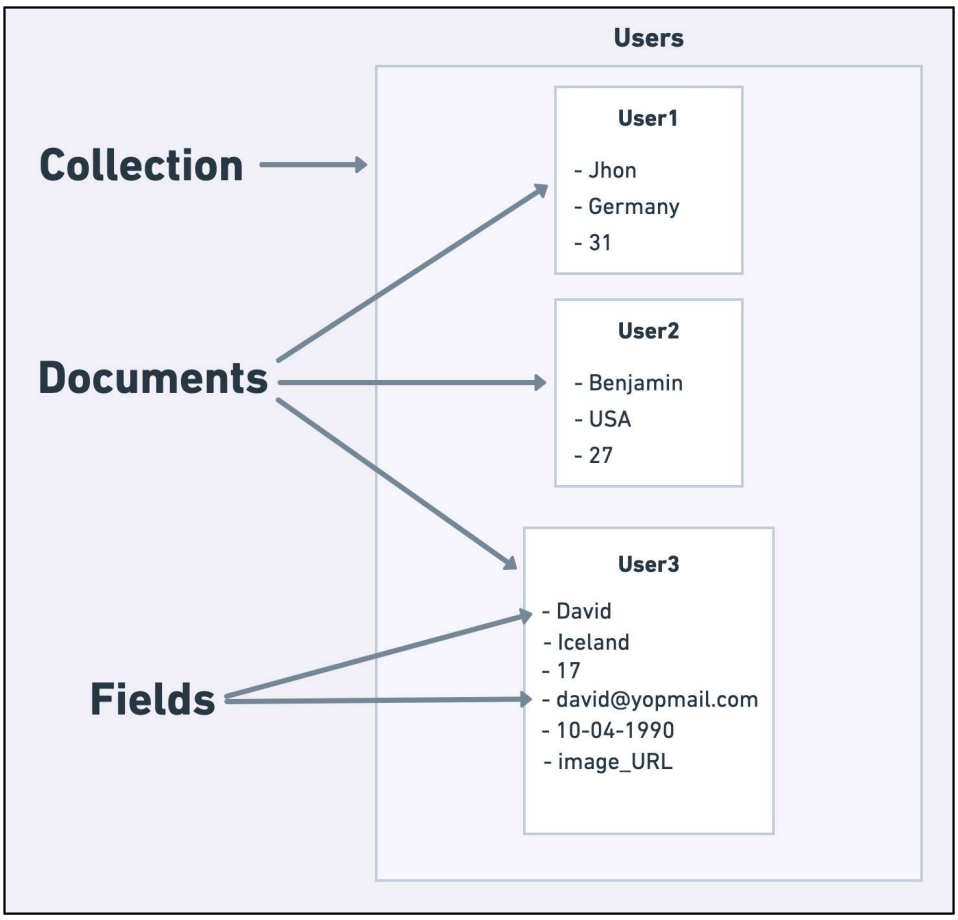

Fig. 3.3.5 (a) Database Design

Every user's information is kept in a unique document. Multiple of these documents come together to form a collection. The beauty of this system is that not all documents within a collection need to have identical fields. So, if we decide to add a new field (e.g., DOB, image) to a new document, there's no need to go back and add it to older ones.

A. Panel view

Also known as The Firestore Admin Panel is a tool for reading and writing customer data. It provides a spreadsheet-like UI for Firestore collections. It also allows us to share data access and invite team members with granular permission control at the table and field level.

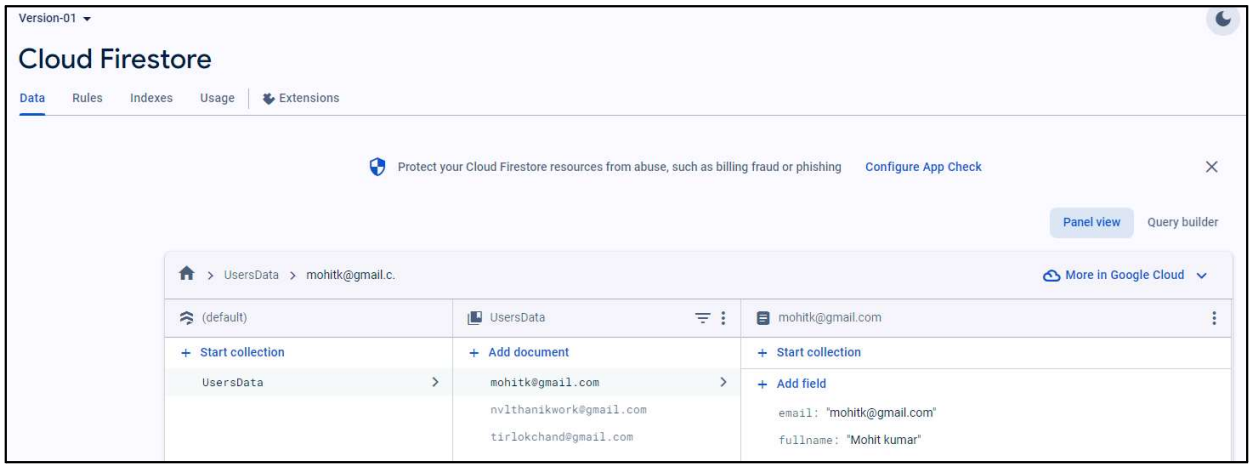

Fig. 3.3.5 (b) Panel View of Firestore

B. Fields Structure

As mentioned above the field-document data model is used to store the data All the documents must be stored in collections. Documents can contain sub-collections and nested objects, which can include primitive fields like string or complex objects like lists. Further Fields contain the actual data, like Tables in RDBMS

Here is a code snippet to get working on our database design. This code snippet is written in Dart and is likely part of a Flutter application using Firebase Firestore for data storage. Let's break down what each part of the code does:

1. final cities = db.collection("cities");
    - This line initialises a reference to a Firestore collection named "cities". The db variable likely represents an instance of Firestore.

2. final data1 = <String, dynamic>{ ... };
    - A variable named data1 is declared, which is a map containing information about a city. The keys in the map are strings, and the values can be of any type (dynamic). This map represents the data to be stored in Firestore for the city "San Francisco".
3. cities.doc("SF").set(data1);
    - This line sets the data represented by the map data1 to the document with the ID "SF" in the "cities" collection in Firestore. It effectively adds or updates the document with the specified data.
4. final data2 = <String, dynamic>{ ... };
    - Similar to data1, this line declares a variable data2 containing information about another city, "Los Angeles".
5. cities.doc("LA").set(data2);
    - This line sets the data represented by the map data2 to the document with the ID "LA" in the "cities" collection in Firestore. Similar to the previous line, it adds or updates the document with the specified data.

```
final cities = db.collection("cities");
final data1 = <String, dynamic>
{
  "name": "San Francisco",
  "state": "CA",
  "country": "USA",

  "capital": false,
  "population": 860000,
  "regions": ["west_coast", "normal"]
};
cities.doc("SF").set(data1);

final data2 = <String, dynamic>
{
  "name": "Los Angeles",
  "state": "CA",
  "country": "USA",

  "capital": false,
  "population": 3900000,
  "regions": ["west_coast", "social"],
};
cities.doc("LA").set(data2);
```

ER Diagram

The Entity-Relationship (ER) diagram for a Firestore database in a Flutter-Dart application visually represents the relationships and structure of the data model. Key points of the ER diagram:

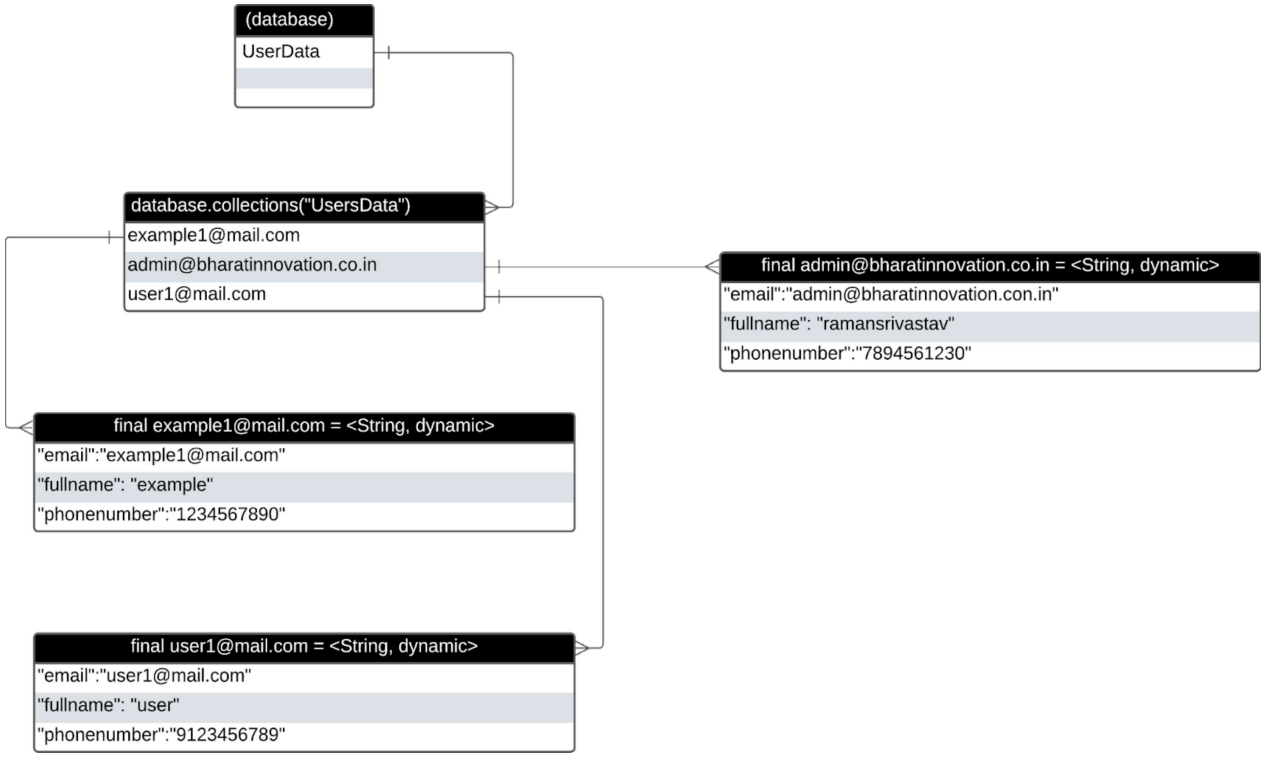

Fig.3.3.5 (c) ER Diagram of the project

This ER diagram visually conveys the structure of the Firestore database, defining entities, their attributes, relationships, and how the data is organised in the cloud-based NoSQL database. It serves as a valuable reference for developers and database administrators working on the Flutter application.

Entities:

- Users: Represents user profiles with attributes such as UserID, Username, Email, etc.
- Devices: Represents connected IoT devices with attributes like DeviceID, DeviceName, Status, etc.

Relationships:

- Owned Devices (One-to-Many): Connects Users to Devices, signifying that one user can own multiple devices. Depicts the DeviceID as a foreign key in the User entity.

Attributes:

- email: Username, Email: Attributes of the User entity.
- DeviceID, DeviceName, Status: Attributes of the Devices entity.

Primary Keys:

- UserID: Serves as the primary key for the Users entity.
- DeviceID: Functions as the primary key for the Devices entity.

Cardinality Notation:

- "1" and "M": Denote the one-to-many relationship between Users and Devices.

Firebase Collections:

- users: Represents the Firestore collection for user profiles.
- devices: Represents the Firestore collection for connected devices.

Cloud Firestore:

- Document Structure: Illustrates how user and device data is organised within Firestore documents.

3.3.6 Hardware Design

The Circuit Diagram of hardware prototype explains crucial details of the wearable band

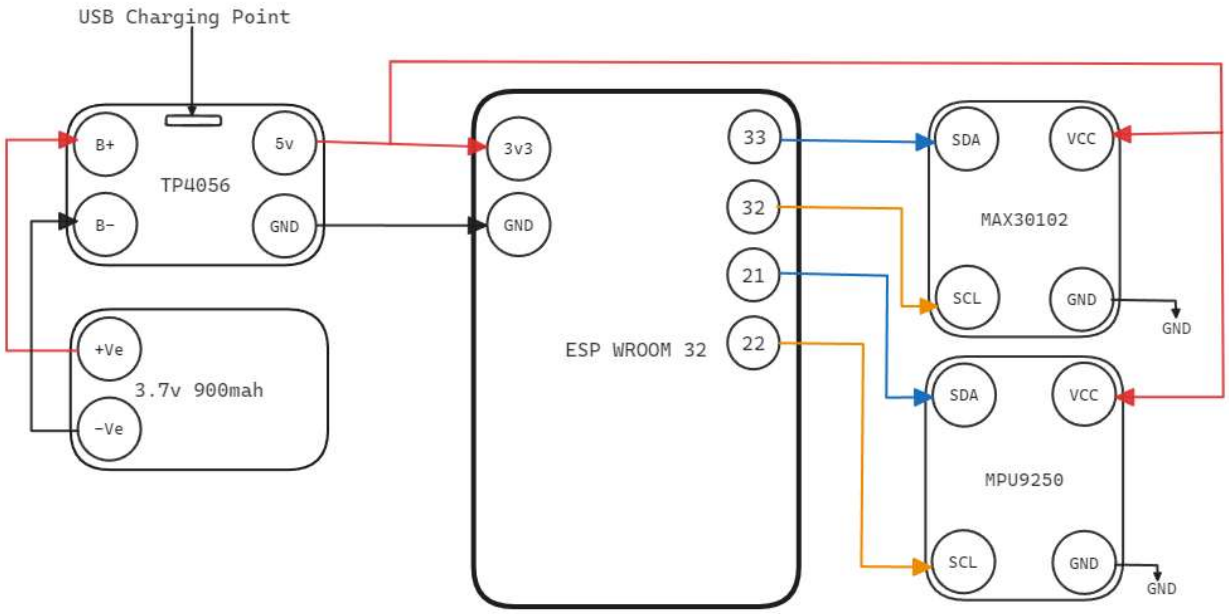

Fig.3.3.6 Circuit Diagram of the project

This hardware design depicts the circuitry and connections of a device, likely an IoT (Internet of Things) or embedded system. At its core is the ESP WROOM 32, a widely used microcontroller module for IoT applications, which interfaces with various sensors and components.

The power supply section includes a USB charging point connected to a TP4056 charging module, providing 5V output for charging a lithium-ion battery (3.7V, 900mAh) through a protection circuit. The 3.7V output is also supplied to the ESP WROOM 32 module for operation.

The ESP WROOM 32 interfaces with two sensors: the MAX30102 pulse oximeter sensor and the MPU9250 accelerometer and gyroscope sensor. These sensors communicate with the ESP module via the I2C protocol, with the SDA and SCL lines connected to GPIO pins of the ESP module.
Overall, this hardware design enables the ESP WROOM 32 to gather data from the sensors, process it, and potentially transmit it wirelessly to other devices or a central server for further analysis or action in IoT applications.

3.3.7 Assumptions and Dependencies

Assumptions

- Stable Internet Connection: The system assumes a stable and reliable internet connection for fast communication between the Flutter app and the Firestore database.
- User Authentication: Assumes that users will authenticate securely through Firebase Authentication to access the features of the Flutter app and control connected devices.
- Firebase Services Availability: Assumes the continuous availability and proper functioning of Firebase services, including Firestore, for seamless data storage and retrieval.

Dependencies

Firebase SDK: The system relies on the Firebase SDK for Flutter to integrate Firebase services, ensuring efficient communication with the Firestore database.

Flutter Framework: Dependencies include the Flutter framework and Dart programming language for app development, ensuring compatibility and optimal performance.

IoT Hardware Compatibility: Assumes compatibility with ESP32-based IoT hardware for successful integration with the Flutter app, enabling remote device control.

User Permissions: Dependencies on user permissions managed by Firebase Authentication to regulate access control and ensure data security.

Firestore Database Structure: Assumes adherence to the predefined structure of the Firestore database for consistent and accurate data storage.

External Libraries: Dependencies on external libraries used in the Flutter app for enhanced functionalities and UI components.

3.4 User Interface Design

User interface (UI) design plays a crucial role in creating an engaging and intuitive experience for users. Here's a detailed explanation of the user interface design elements and concepts used in our project:

1. AppBar: The AppBar is a fundamental UI component in our project, typically positioned at the top of the screen. It contains the app's title, navigation icons (e.g., menu icon for accessing additional options), and action buttons (e.g., search, settings).

2. TextField: TextFields are used for accepting user input, such as usernames, passwords, or search queries. They provide users with a visual cue for entering text and often include features like hint text, error messages, and input validation.

3. Buttons: Buttons are interactive UI elements used for triggering actions or navigating to different parts of the app. They can be styled with various colours, shapes, and text to indicate their purpose and importance. Common types of buttons include raised buttons, flat buttons, and floating action buttons (FABs).

4. ListView: ListViews are used to display a scrollable list of items, such as user data, messages, or product listings. They support vertical scrolling and can be customised to include different types of content, such as text, images, or icons.

5. Card: Cards provide a structured layout for presenting information in a visually appealing manner. They typically include a title, content area, and optional action buttons. Cards are commonly used for displaying user profiles, product details, or news articles.

6. AlertDialog: AlertDialogs are modal UI components used to display important messages, alerts, or prompts that require user attention. They typically include a title, message, and one or more action buttons for user interaction (e.g.OK, Cancel).

7. BottomNavigationBar: The BottomNavigationBar is a UI component located at the bottom of the screen, providing users with easy access to different sections or features of the app. It typically consists of multiple tabs, each representing a distinct area of the app, such as Home, Profile, Settings, etc.

8. SnackBar: SnackBars are temporary messages displayed at the bottom of the screen to provide brief feedback or alerts to users. They are commonly used to inform users about successful operations, errors, or other important events.

9. CircularProgressIndicator: CircularProgressIndicators are used to indicate that a task is in progress, such as loading data from a server or processing user input. They provide visual feedback to users to indicate that the app is actively working on their request.

10. Drawer: The Drawer is a UI component that slides in from the left edge of the screen, typically containing a menu or navigation panel. It allows users to access additional options, settings, or navigation links without cluttering the main screen.

By leveraging these UI design elements and principles, we aim to create a user-friendly, visually appealing, and intuitive interface for our project, enhancing the overall user experience and satisfaction.

3.4.1 UI Wireframe

This UI wireframe represents the initial layout and structure of a mobile application called "Well Wisher.his UI wireframe provides a basic visual representation of the app's key screens and functionalities, serving as a foundation for further design and development iterations. Starting from left to right It has 4 main Screens currently as follows:

1. Splash screen as Welcome screen
2. Login Screen
3. Dashboard Screen.
4. Data Screen for Profile etc.

In a typical application, the first screen we see is the Splash or Welcome Screen, often showcasing the app’s logo while loading necessary data. This transitions to the Login Screen where users enter their credentials to gain access. Once authenticated, the user lands on the Dashboard Screen, providing an overview of the app and its functionalities. Lastly, the Data Screen displays specific data, such as user profile information, enhancing the personalised experience of the app.

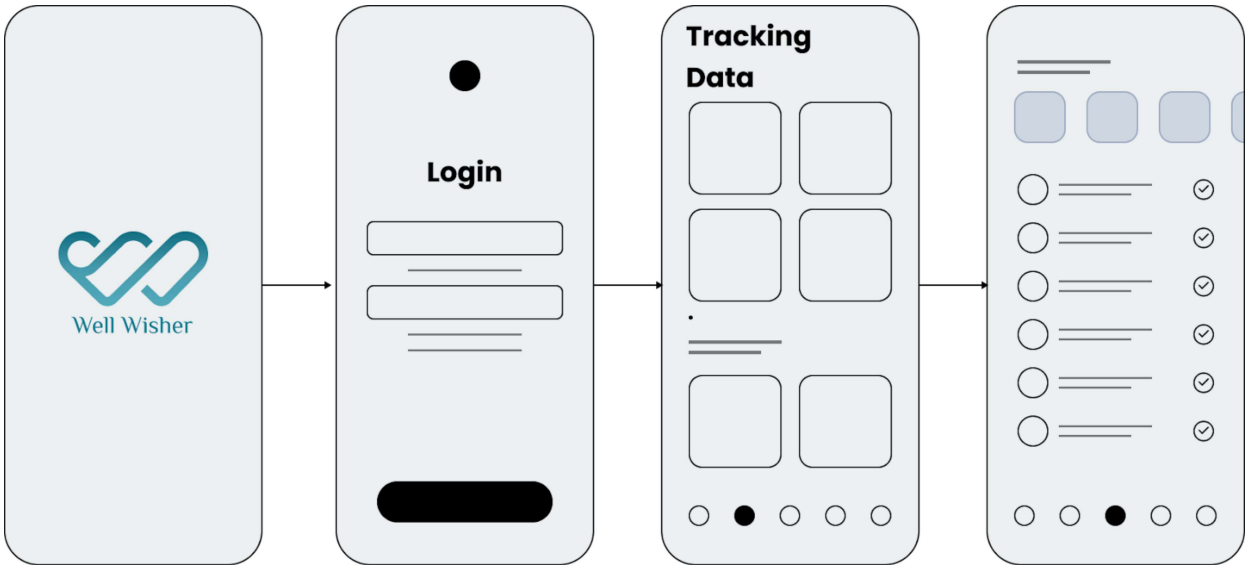

Fig 3.4.1 UI Wireframe of the WellWisher App

The detailer explains the above mentioned image, Firstly it establishes brand identity and creates a positive first impression.

Login Screen (Middle):

- The second screen is the login interface, featuring input fields for username and password.
- It allows users to authenticate themselves to access the app's features and functionalities securely.
- The login button at the bottom allows users to submit their credentials and proceed to the app's main content.

Dashboard Screen (Right):

- The third screen is the tracking data interface, designed to display various data points or metrics to users.
- It features multiple placeholders or containers where different types of data can be displayed, such as health or fitness metrics.
- This screen is likely part of the app's main functionality, providing users with valuable information or insights relevant to their well-being.

It helps users visualise the app's user flow and layout before investing resources into detailed design and implementation.

Chapter 4. Implementation and Testing

4.1 Hardware Technologies

4.1.1 Languages

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 us to transfer code with ease from the desktop to a microcontroller or embedded system.

Fig 4.1.1 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 licence, 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.

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.

Arduino IDE

Arduino Integrated Development Environment (IDE) is an open-source IDE that allows users to write code and upload it to any Arduino board. Arduino IDE is written in Java and is compatible with Windows, macOS and Linux operating systems.

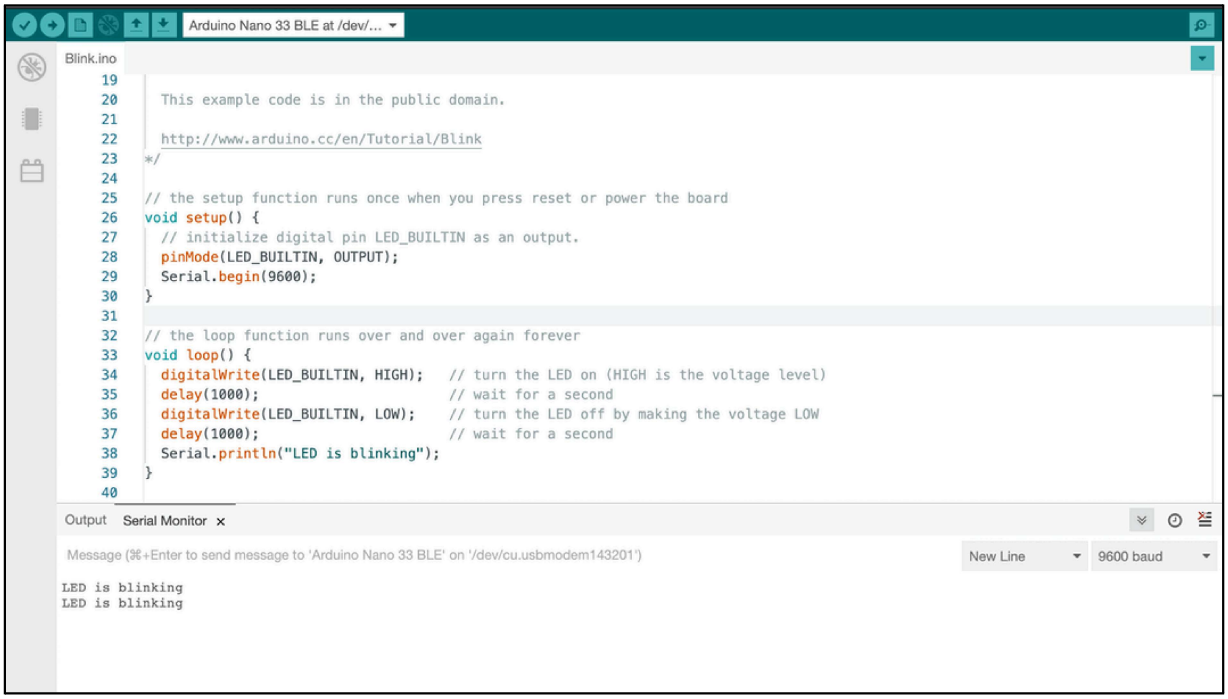

Fig 4.1.2 Arduino IDE Editor

Arduino is adding the Python language as an additional option for programming microcontrollers. Our platform of choice is MicroPython. Following is a short information about MicroPython and also the IDE used during training.

Thonny IDE

It is a free development program for PCs that was made by an independent dev who goes by the same name. It is an open-source integrated development environment (IDE) that can be used to create various applications using the Python programming language.

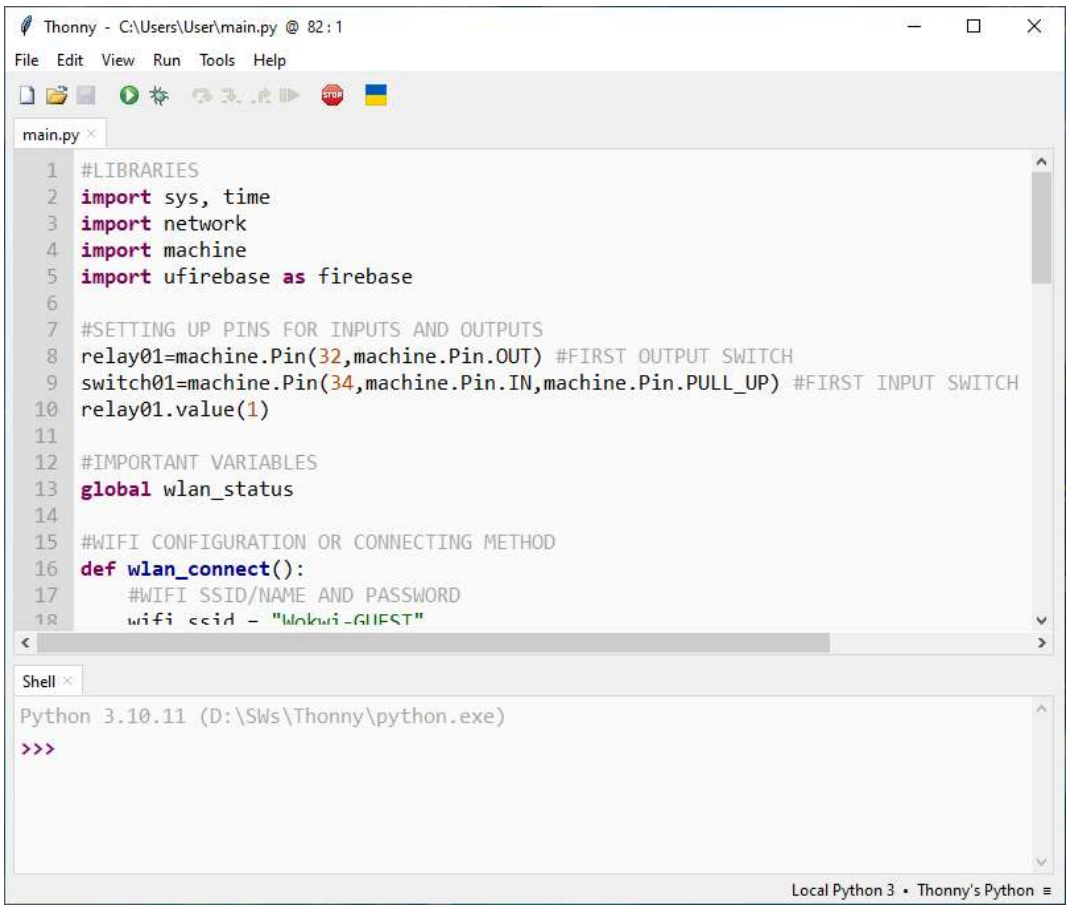

Fig 4.1.3 Thonny IDE Editor

- **MicroPython**: 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.

4.1.2 Hardware Equipments

Some of the tools that used for implementation of our prototype are as following:

ESP32 microcontroller

The ESP32 is a powerful microcontroller designed by Espressif Systems, which is based on a dual-core Xtensa LX6 processor and includes built-in Wi-Fi and Bluetooth connectivity. It has a clock speed of up to 240 MHz and can be programmed using various programming languages such as C, C++, and MicroPython. The ESP32 has a wide range of peripherals, including digital and analogue I/O pins, SPI, I2C, UART, and ADC interfaces. It also includes advanced features such as touch sensors, cryptographic hardware accelerators, and power management modules.

Fig 4.1.4 ESP-Wroom-32 Chip

MAX30102 heart rate sensor

The MAX30102 is a high-sensitivity pulse oximeter and heart-rate sensor module. It is designed to detect the pulse oximetry and heart-rate signals of a user noninvasively through a fingertip. The module integrates a red LED, an infrared LED, and a photodetector in a compact package. 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.

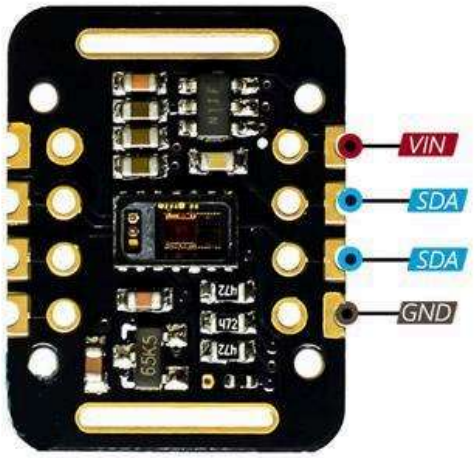

Fig 4.1.5 MAX30102 Heart Rate Sensor

MPU9250 accelerometer sensor

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 commonly used for motion detection, activity tracking, and orientation sensing in a wide range of applications, including drones, robotics, and wearable devices.

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 changes in orientation. 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.

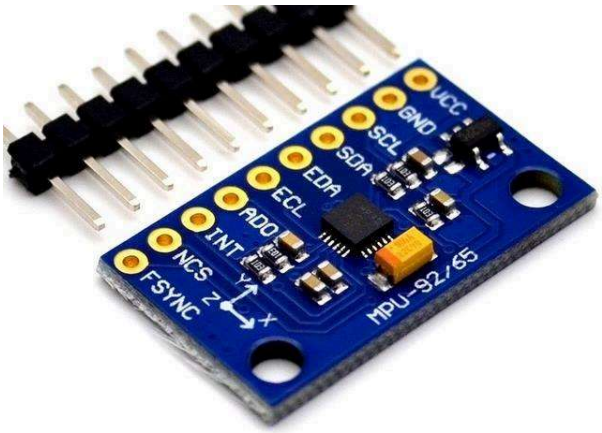

Fig. 4.1.6 MPU9250 9-Axis Sensor

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.

TP4056 charging module

The TP4056 charging module is a compact and versatile circuit board designed for charging lithium-ion or lithium polymer batteries. It can be integrated into various projects where a reliable and efficient battery charging solution is required. Here's how we used the TP4056 charging module in our project:

1. Power Source Connection: The module features a micro USB input port (labelled IN+) that allows us to connect it to a USB power source, such as a computer, USB wall adapter, or power bank.

2. Battery Connection: Connect the lithium-ion or lithium polymer battery to the module's battery terminals, which include B+ (positive terminal) and B- (negative terminal) connections. Ensure the correct polarity when connecting the battery.
3. Charging Control: The TP4056 charging IC on the module regulates the charging process, managing the charging current and voltage to safely and efficiently charge the connected battery. It automatically switches to trickle charging mode when the battery voltage reaches a certain level to avoid overcharging.

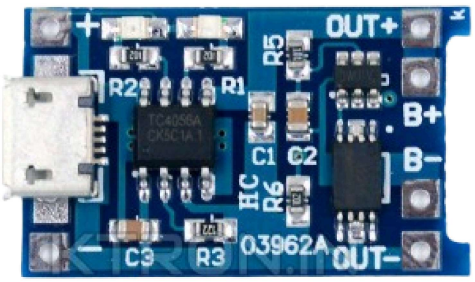

Fig 4.1.7 TP4056 charging module

4. Charging Status Indication: The module typically includes one or more LEDs to indicate the charging status. These LEDs illuminate to show whether the battery is charging, fully charged, or if there is an error during the charging process.
5. Output Connection: Once the battery is fully charged, We can disconnect it from the module and use it to power our project. The module's output terminals, labelled OUT+ and OUT-, provide the battery's output voltage for this purpose.

In our project, We incorporate the TP4056 charging module to provide a reliable and safe charging solution for any lithium-ion or lithium polymer batteries used to power our device. It ensures that our batteries are charged efficiently while protecting them from overcharging and other potential risks, making it suitable for a wide range of applications, including portable electronics, IoT devices, and hobbyist projects.

Rechargeable LiPo Battery

The image shows a lithium-ion polymer (LiPo) battery with a voltage rating of 3.7 volts. Here's how used this battery in our project:

- Power Source: The 3.7V LiPo battery serves as a power source for our electronic project. It can be used to provide a portable and rechargeable power supply.

- Capacity: The capacity of the battery, usually measured in milliampere-hours (mAh), indicates how much charge it can store. In this case, the battery has a capacity of 900mAh, which means it can supply a current of 900 milliamperes for one hour before it needs recharging.

Fig 4.1.8 Rechargeable LiPo Battery

- Voltage: The nominal voltage of the battery is 3.7 volts, which is typical for LiPo batteries. This voltage level is compatible with many electronic devices and microcontrollers.
- Charging: To use the battery in our project, we'll need a charging circuit, such as the TP4056 charging module mentioned earlier. Connect the LiPo battery to the charging module to recharge it when necessary. Ensured to follow proper charging procedures and precautions to prevent overcharging and ensure safety.
- Connections: The battery typically has two wires for connection: one for positive (+) and one for negative (-). Ensure correct polarity when connecting the battery to our project or charging circuit to avoid damaging the battery or our electronic components.

Overall, the 3.7V LiPo battery is a versatile and commonly used power source for various electronic projects due to its compact size, high energy density, and rechargeable nature. It provides a convenient solution for applications that require mobility or operate away from a traditional power outlet.

Jumper 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.

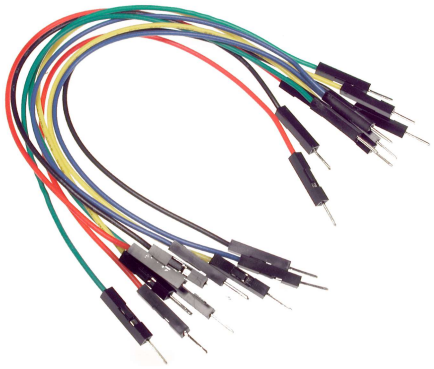

Fig 4.1.9 Jumper Wires (male-to-male jumper).

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.

4.2 Software Technologies

4.2.1 Dart Language

Dart is the primary language used for developing applications using the Flutter framework. Flutter is Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. 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 Context of our Project, The Dart becomes the unifying tool through the diverse development language of modern apps, providing us 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.

Fig 4.2.1 Dart Language Logo

- 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.

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.

4.2.2 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 our static files. Firebase provides an all-in-one solution for developers, including data storage and synchronisation, user authentication, analytics and reporting.

Fig 4.2.2 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.

4.2.3 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 our 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": "bee@example.com",
    "age": 24
  },
  "posts": {
    "title": "Connection 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 JSON format available in Firestore is incredibly useful for organising and storing data related to users and their activities. For example, we can structure our data in a way that each user's information, such as their email and age, is stored under a "users" collection. This makes it easy to retrieve and update user details whenever needed. Along with that, we can store information about each fitness activity or post, like the title and author ID, under a "posts" collection. This allows us to track users' workout sessions, progress, and any additional insights they share.

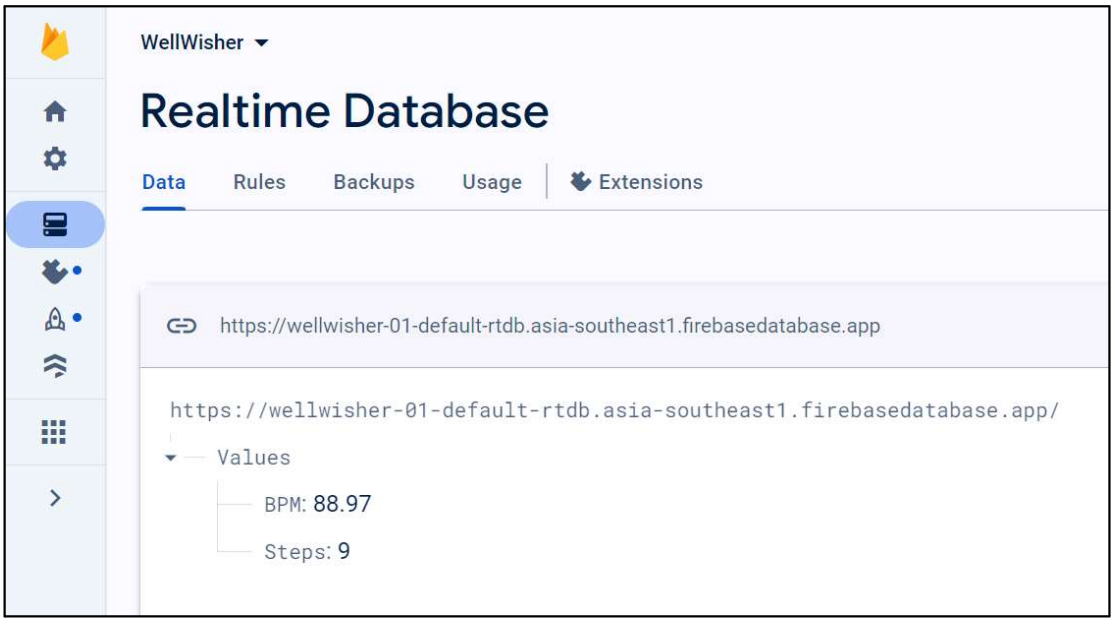

Fig 4.2.3 Firebase Console View

4.2.4 Supporting Languages & Tools

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 4.2.4 Flutter Framework Logo.

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 utilised, 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.

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 fast updates, while code templates and GitHub integration expedite feature building. Extensive testing tools, lint tools, and C++/NDK support ensure app robustness.

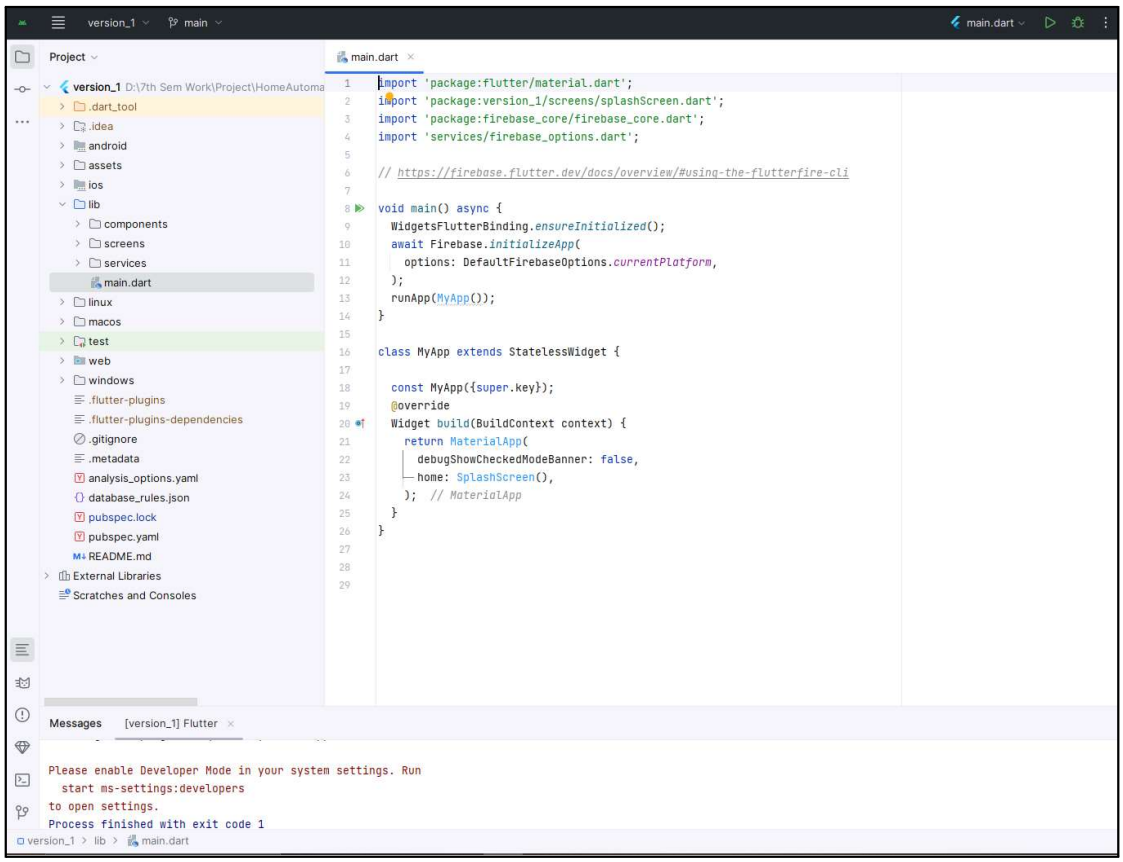

Fig. 4.2.5 Android Studio editor view

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.

4.3 Pseudocode in Hardware Module

The pseudocode outlines a program designed to create a comprehensive health monitoring system. It begins by establishing a WiFi connection using a WifiManager module, enabling communication with external services. The program then configures a Firebase URL using the ‘ufirebase’ module, setting up a cloud-based database to store and access health data.

```
# Initialize WiFi connection and Firebase URL
wifi_manager.connect()
firebase.setURL("https://your-firebase-url.com")

# Initialize and configure MAX30102 sensor
sensor = MAX30102()
sensor.setup()

# Define function to read MAX30102 sensor values
def read_max30102_values():
    while True:
        # Read sensor values
        red_reading, ir_reading = sensor.read()

        # Process readings to detect beats and calculate BPM
        if detect_beat(red_reading):
            bpm = calculate_bpm()

            # Update Firebase with BPM data
            firebase.put("Values", {"BPM": bpm})
# Initialize and configure MPU9250 accelerometer sensor
accel_sensor = MPU9250()
accel_sensor.setup()

# Define function to calculate steps using MPU9250 sensor
def calculate_steps():
    num_steps = 0
    while True:
        # Read accelerometer data
        ax, ay, az = accel_sensor.read_acceleration()

        # Calculate steps based on orientation changes
        num_steps += detect_orientation_changes(ax, ay, az)

        # Update Firebase with steps data
        firebase.put("Values", {"Steps": num_steps})

# Start separate threads for reading MAX30102 values and calculating
steps
thread.start_new_thread(read_max30102_values, ())
thread.start_new_thread(calculate_steps, ())

# Main loop for handling exceptions and maintaining program execution
while True:
    try:
        # Handle exceptions gracefully
        handle_exceptions()
    except Exception as e:
        print("Exception occurred:", e)
```

Next, the program initialises and configures two primary sensors: the MAX30102 for heart rate monitoring and the MPU9250 for step counting. The MAX30102 sensor is fine-tuned with parameters like sample rate, LED brightness, and active LED mode to optimise heart rate detection. Simultaneously, the MPU9250 sensor is configured to read accelerometer data and detect changes indicating steps taken.

To process sensor data, the program defines functions responsible for continuous readings and data processing. One function reads values from the MAX30102 sensor, detects heartbeats, calculates BPM (beats per minute), and updates Firebase with the BPM values. Another function reads values from the MPU9250 sensor, detects changes in orientation (indicative of steps), counts steps, and updates Firebase with step count data.

Multithreading is employed to execute these sensor reading and processing functions concurrently, enhancing performance and responsiveness. Additionally, error handling mechanisms are implemented within the main loop to gracefully manage exceptions, ensuring the program continues running smoothly despite unexpected issues like network errors or sensor malfunctions.

In summary, the pseudocode represents a sophisticated health monitoring system that leverages WiFi connectivity, cloud-based storage, and multiple sensors to monitor vital health metrics like heart rate and physical activity. The program is designed for efficient operation, accurate sensor data processing, and robustness through comprehensive error handling.

4.4 Testing

4.4.1 Testing The Hardware Module

To perform experimental testing of the hardware module in the context of the project, we followed these steps:

1. Setup Hardware:
    - Connect all the hardware components according to the circuit diagram or schematic provided in the project documentation.
    - Ensure that all connections are secure and that there are no loose wires or components.
2. Power On:
    - Power on the hardware module by supplying the required power source, such as batteries or USB power.

- Verify that the power indicators or LEDs on the hardware components light up, indicating that they are receiving power.

3. Test Sensors and Modules:
    - Test each sensor and module individually to ensure they are functioning correctly.
    - For example, test the MAX30102 sensor by checking its readings using the provided code snippet and verifying that it detects heart rate accurately.

Similarly, tested the MPU9250 sensor for step detection by moving it around and observing the step count output.

1. Data Transmission:
    - Verify that data transmission between the hardware modules and the microcontroller (e.g., ESP WROOM 32) is working as expected.
    - Check that data is being collected from sensors and transmitted to the appropriate destination, such as a cloud server or database.
2. Error Handling:
    - Test error handling mechanisms by intentionally introducing faults or incorrect inputs.
    - Verify that the system can detect errors, display appropriate error messages, and recover gracefully without crashing.

4.4.2 Test Case

Title: Heart Rate Monitoring Accuracy Comparison

Test Case ID: WW002

Description: This test case compares the heart rate measurement accuracy of the "Hardware Prototype" with the HEM7120 and Firebolt Phoenix devices. The test is conducted under room temperature conditions of 28 degrees Celsius, with a male user aged 23 years.

Preconditions:

The "Hardware Prototype" fitness tracking band, HEM7120, and Firebolt Phoenix devices are powered on and functioning properly.

The user is wearing the "Hardware Prototype" band, and the HEM7120 and Firebolt Phoenix devices are placed according to their instructions.

Test Steps:

1. Start with all devices in standby mode.
2. Navigate to the heart rate monitoring feature on the "Hardware Prototype" band, HEM7120, and Firebolt Phoenix devices.
3. Ensure that the devices are correctly positioned on the user (wrist for "Hardware Prototype" band, appropriate body part for HEM7120, and Firebolt Phoenix).
4. Wait for the devices to display the current heart rate reading simultaneously.
5. Record the heart rate readings displayed by each device.
6. Calculate the average heart rate reading from all three devices.
7. Compare the average heart rate with the actual heart rate of the user (if available) or use the expected heart rate based on the user's age and sex.

Expected Result: The average heart rate reading from the "Hardware Prototype" band, HEM7120, and Firebolt Phoenix devices closely matches the expected heart rate for a 23-year-old male under room temperature conditions (28 degrees Celsius).

Pass Criteria

- The average heart rate reading from all devices is within an acceptable margin of error (e.g., ±5 bpm) of the expected heart rate.

Fail Criteria

- The average heart rate reading from any device deviates significantly from the expected heart rate outside the acceptable margin of error.

Postconditions: Record the heart rate readings from both devices, the calculated average heart rate, any discrepancies observed, and the overall comparison results for analysis. This test case allows us to evaluate how well the "Hardware Prototype" fitness tracking band performs compared to other heart rate monitoring devices in a controlled environment, taking into account.

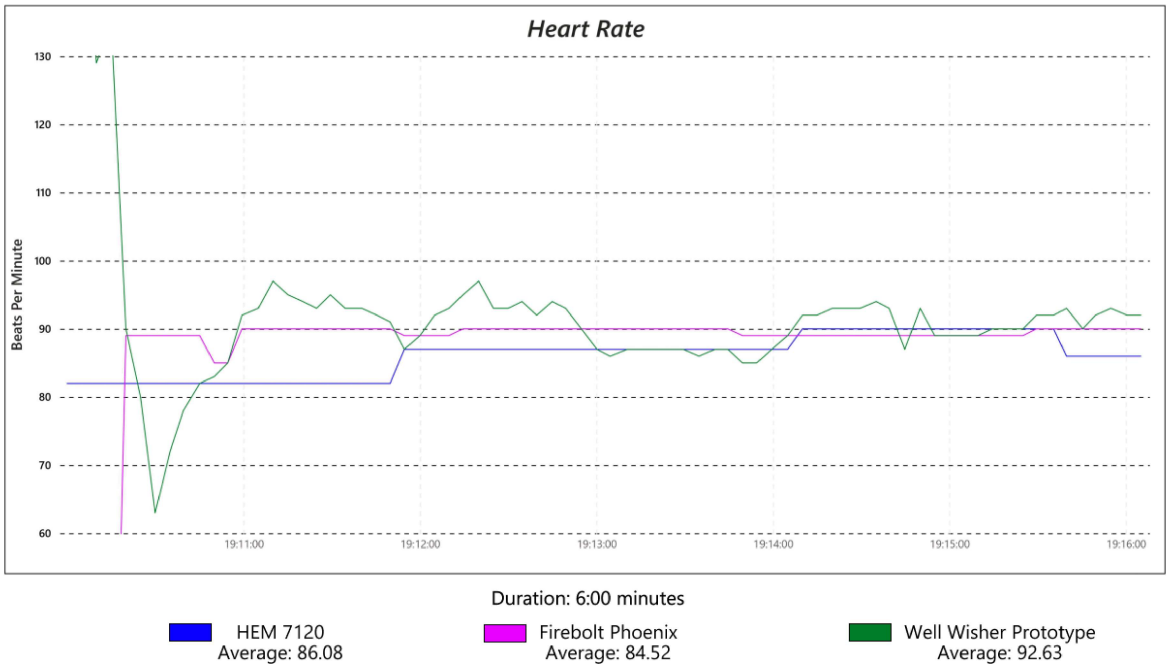

Fig 4.4.2 Beats Per Minute Graph

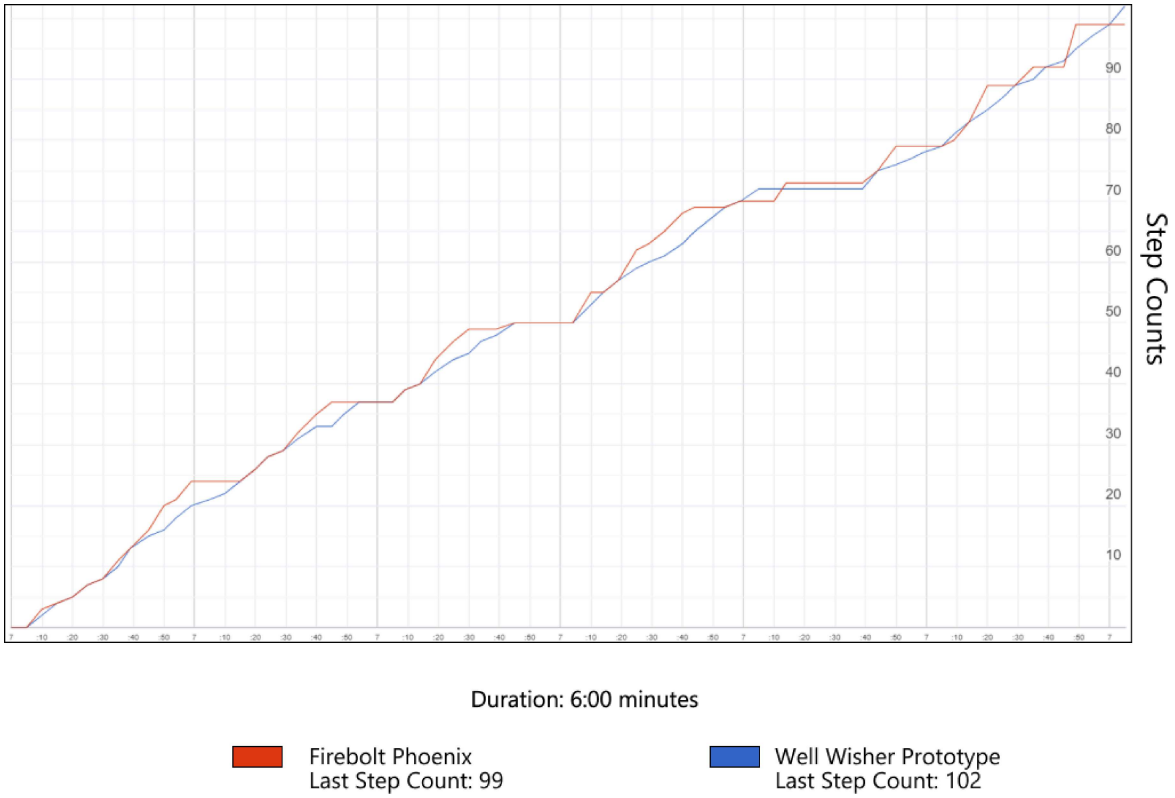

Fig 4.4.3 Steps Taken Graph

The Fig 4.4.2 depicts the heart rate monitoring data collected during Test Case WW002, which aims to compare the accuracy of heart rate measurement among the "Hardware Prototype" band, HEM7120, and Firebolt Phoenix devices. The test was conducted under room temperature conditions (28°C) with a male user aged 23 years.

Preconditions ensure that all devices are operational and correctly positioned on the user. Test steps include activating the heart rate monitoring feature, synchronising the readings, and recording the data from each device. The expected result is that the average heart rate readings closely match the expected heart rate for the user's demographic.

Pass criteria require the average heart rate readings from all devices to be within an acceptable margin of error of the expected heart rate, while any significant deviation constitutes a fail criterion. Postconditions involve recording and analysing the collected data to assess the performance of the "Hardware Prototype" band relative to the comparison devices.

The graph in Fig 4.4.3 depicts the performance comparison between two different algorithms or models, represented by the red and blue lines. The x-axis likely represents time or some other sequential measure, while the y-axis represents a certain metric, possibly accuracy, efficiency, or another performance measure.

In the context of the project, this graph could represent the comparison of different machine learning algorithms for predictive models used for heart rate monitoring. The red line represents the performance of one algorithm, while the blue line represents another algorithm. The upward trend in both lines indicates an improvement or increase in performance over time or with some other factor.

The legend at the bottom indicates which line corresponds to each algorithm or model. The differences in performance between the two algorithms can be observed and analysed to determine which one is more suitable for integration into the project. Further analysis involves examining specific data points or segments where one algorithm outperforms the other to understand the strengths and weaknesses of each approach.

Chapter 5: Results and Discussions

5.1 User Interface Representation

5.1.1 Splash Screen

The image shows a splash screen of an application or a mobile device interface. A splash screen is typically displayed when the application is launched, serving as an introductory screen that appears for a short period before the main content of the application loads.

In this splash screen:

- The name of the application "WellWisher" is prominently displayed at the centre, indicating the identity of the app.

Fig 5.1.1 Splash Screen

5.1.2 Sign Up Screen

The image depicts a signup screen for an application called "WellWisher". Here's the breakdown of the elements on the screen:

1. Signup Form: Below the app name, there's a signup form with fields to input the user's information. The fields include:
    - Full Name: This field is for the user to enter their full name.
    - Email Address: Users are prompted to enter their email address.
    - Password: This field is for users to create a password for their account.
    - Confirm Password: Users need to re-enter the password to confirm it.
2. Signup Button: At the bottom of the form, there's a prominent "Sign Up" button. Users can tap on this button to submit their information and create an account.
3. Login Link: Below the "Sign Up" button, there's a link that says "Already have an account? Login now." Users who already have an account can tap on this link to navigate to the login screen.

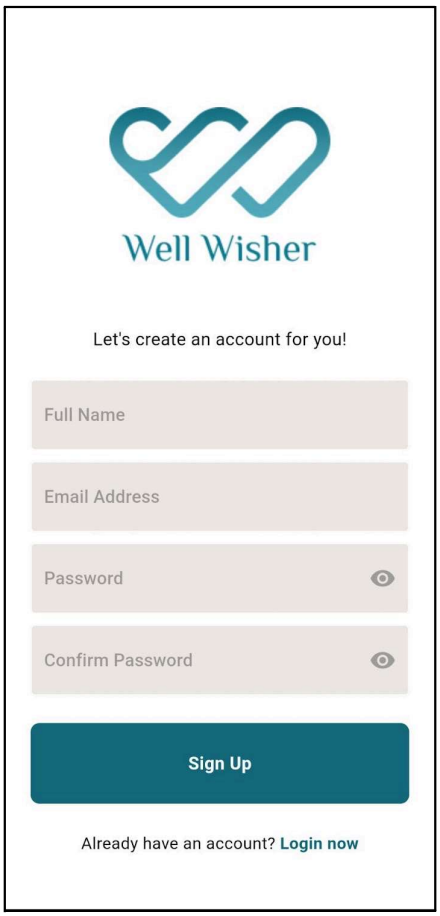

Fig 5.1.2 Signup Screen

5.1.3 Login Screen

This image shows the login screen for the "WellWisher" application. Here's a breakdown of the elements on the screen:

- Login Form: Below the greeting message, there's a login form with fields to input the user's credentials.
- Login Button: At the bottom of the form, there's a prominent "Log In" button. Users can tap on this button to submit their login credentials and access their account.
- Registration Link: Below the login button, there's a link that says "Not a member? Register Now." Users who do not have an account can tap on this link to navigate to the registration screen and create a new account.

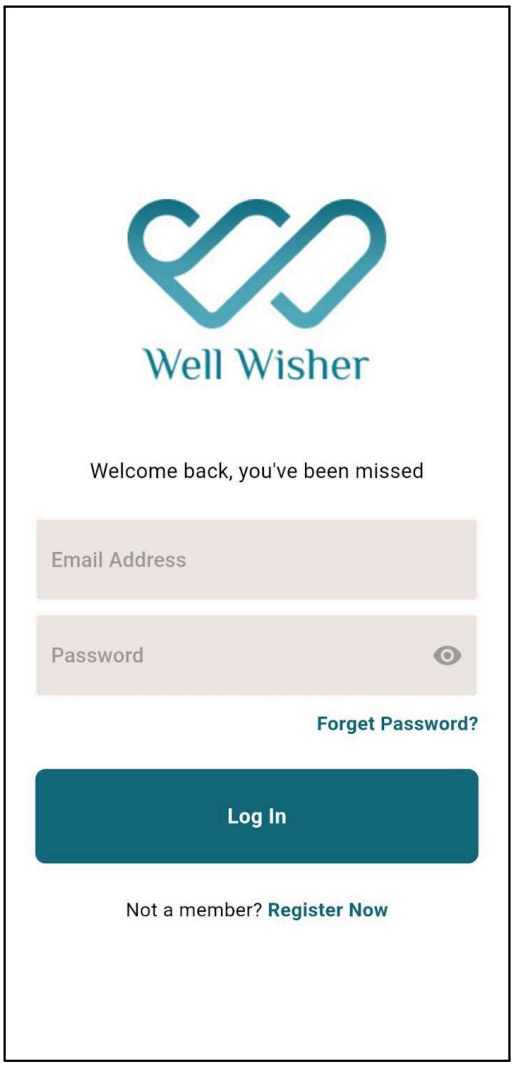

Fig 5.1.3 Login Screen

5.1.4 Forgot Password Screen

This image shows the "Forgot Password" screen of the "WellWisher" application. Here's an explanation of the elements present on the screen:

- Header: At the top of the screen, there's a header section with a back arrow icon on the left side. This icon allows users to navigate back to the previous screen.
- Title: Below the header, there's a title that says "Forgot Password".
- Email Address Field: Below the title, there's a text field where users can input their email address. This field is labelled "Email Address."

Get Verification Link Button: Beneath the email address field, there's a button labelled "Get Verification Link". Users can tap on this button to request a verification link that will be sent to their registered email address for the purpose of resetting their password.

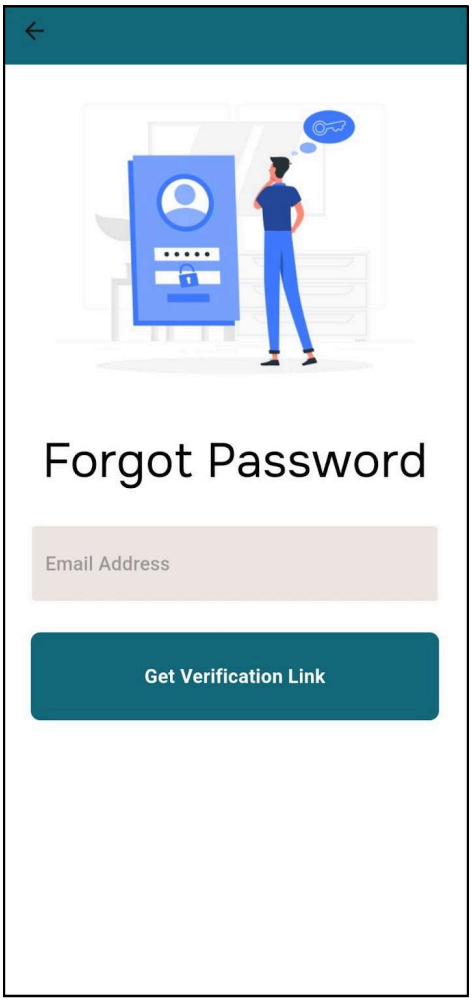

Fig 5.1.4 Forgot Password

5.1.5 User Dashboard Screen

The main dashboard of the app displayed in the image provides users with essential health metrics and navigation options. Here's a breakdown of its components:

1. Dashboard Title:
    - Below the header, there's a title section that says "Tracker Metrics".
2. Heart Rate Metrics:
    - In the first card below the title, there's information about the user's heart rate.
    - It displays the heart rate value (88.97 BPM) along with the label "Elevate Your Heart Rate".
    - The card also includes an illustration of a heart to represent the heart rate.
3. Steps Taken Metrics:
    - In the second card below the heart rate metrics, there's information about the steps taken by the user.
    - It displays the number of steps taken (9) along with the label "Stride Ahead".
    - The card also includes an illustration of a person jogging to represent steps taken.
4. Bottom Navigation Bar:
    - At the bottom of the screen, there's a navigation bar with icons representing different sections of the app:
    - "Home" icon: Likely navigates users to the home screen or main dashboard.
    - "Heart Rate" icon: Navigates users to the heart rate monitoring section.
    - "Steps" icon: Navigates users to the step tracking section.
    - "Profile" icon: Navigates users to their profile settings or account information.

Overall, this main dashboard provides users with a quick overview of their heart rate and steps taken, along with easy access to navigate to other sections of the app.

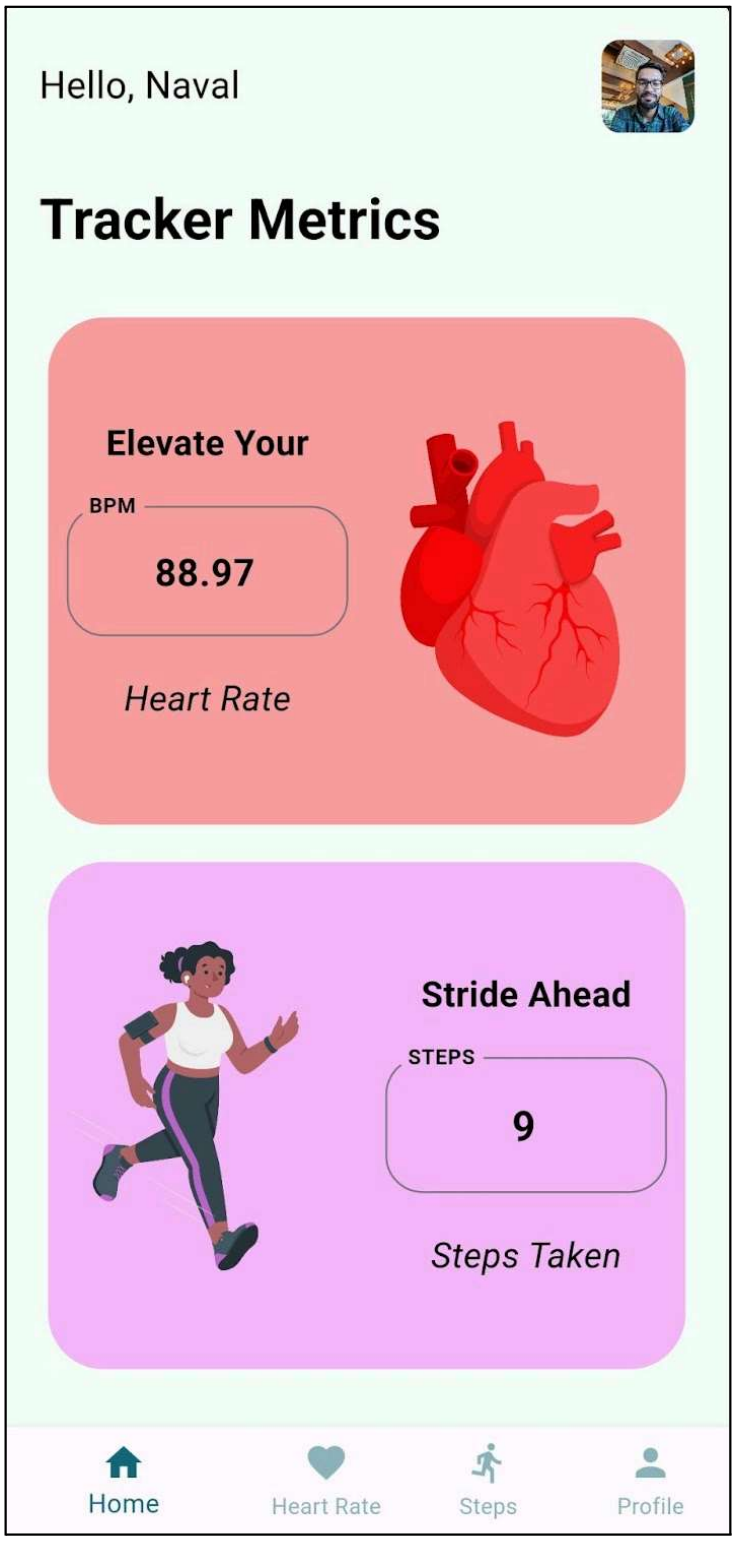

Fig 5.1.5 Dashboard Screen

5.1.6 User Profile Screen

The profile view screen of the app provides users with their personal information and the option to update their profile image. Here's a breakdown of its components:

1. Profile Picture:
    - A circular image representing the user's profile picture.
    - Tapping on the profile picture allows the user to update it by selecting a new image from their smartphone gallery.

2. User Information: Below the profile picture, there's a section displaying various details about the user:
    - Name: The user's name ("Naval" in this case).
    - Email: The user's email address ("bee@mail.com" in this case).
    - Country: The user's country ("India" in this case).
    - Gender: The user's gender ("Male" in this case).
    - Date of Birth: The user's date of birth ("16/9/1999" in this case).

3. Update Profile Button: A button labelled "Update Profile" is provided at the bottom of the screen.
    - This button likely allows users to make changes to their profile information, such as updating their name, email, country, gender, or date of birth.

4. Bottom Navigation Bar: Similar to other screens, the bottom navigation bar provides icons for navigating to different sections of the app:
    - "Home" icon: Likely navigates users to the home screen or main dashboard.
    - "Heart Rate" icon: Navigates users to the heart rate monitoring section.
    - "Steps" icon: Navigates users to the step tracking section.
    - "Profile" icon: Indicates that the user is currently viewing their profile settings.

After Scrolling Down there is a Logout Button which redirects users to the Login Screen.

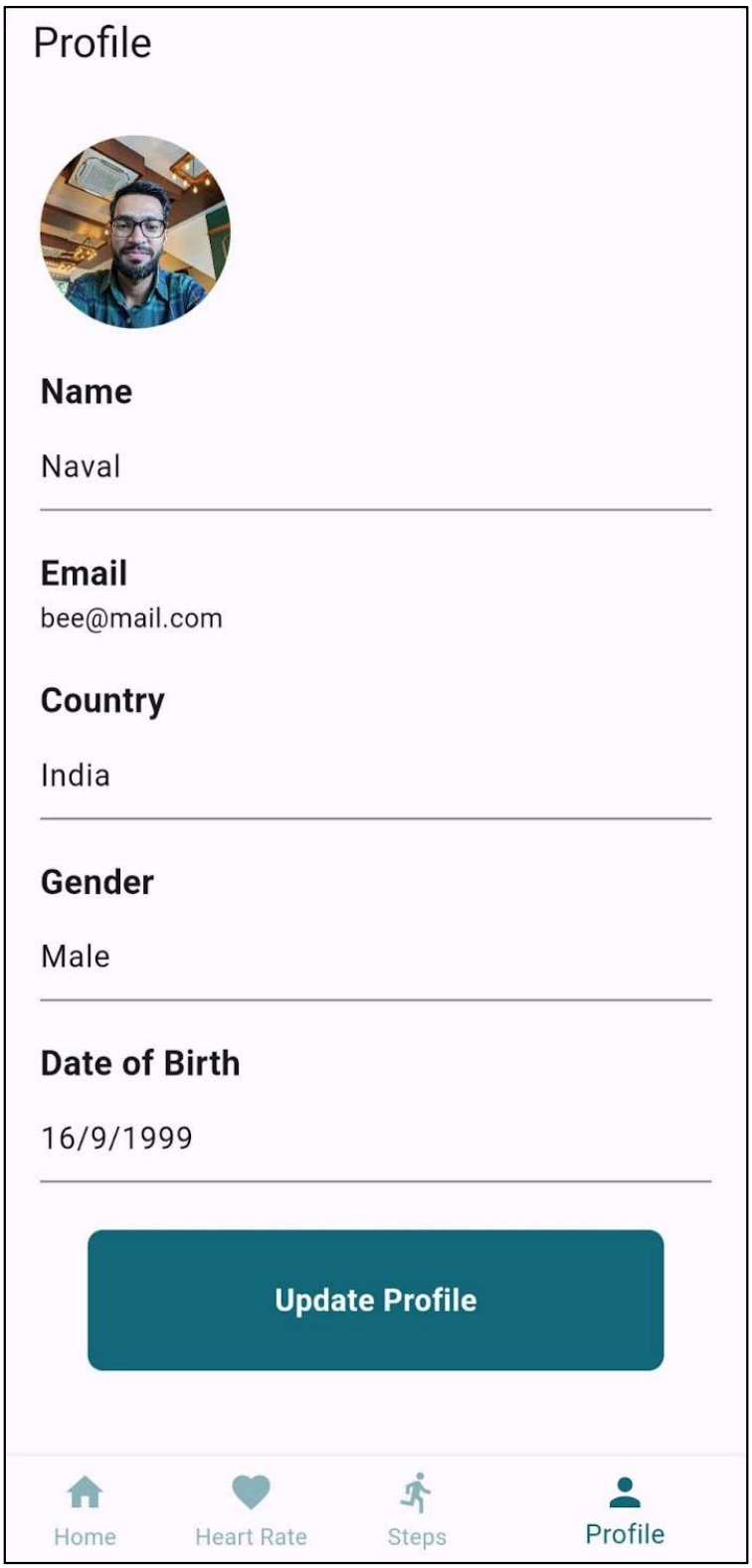

Fig 5.1.6 User Profile Screen

Chapter 6. Conclusion and Future Scope

6.1 Conclusion

In conclusion, our project aimed to create a wearable health monitoring system capable of accurately measuring heart rate, body temperature, and physical activity. The system is composed of an ESP32 microcontroller, MAX30102 heart rate sensor, MPU9250 accelerometer sensor, and a Wi-Fi module for transmitting data to the cloud-based Firebase database. We developed a mobile application using Flutter Flow to visualise and analyse data for users.

Throughout the project, we followed the Agile SDLC model to ensure flexibility, iteration, and responsiveness to changing requirements. We encountered expected challenges such as merging sensor programs, C programming issues, and Firebase connectivity problems, which we addressed using MicroPython and other troubleshooting methods. Overall, our system successfully met all requirements, offering a cost-effective and user-friendly solution for personal health monitoring.

Looking ahead, there are numerous opportunities to enhance and expand this project. One potential direction is to incorporate additional health parameters like sleep monitoring, female period tracking, and SpO2 level monitoring. Furthermore, implementing a recommendation system based on the user's historical data could offer personalised health and fitness advice.

In summary, there is significant potential for further development and enhancement of our project, allowing us to deliver a more comprehensive and personalised health monitoring solution to users.

6.2 Future Scope

6.2.1 Software Module

1. Enhanced User Experience: Continuously improve the user interface and experience based on user feedback and emerging design trends.
2. Additional Features: Integrate new features such as goal setting, personalised workout plans, nutrition tracking etc.
3. Machine Learning Integration: Implement machine learning algorithms to provide personalised insights, recommendations, and predictions based on user data and behaviour patterns.

4. Health Data Integration: Integrate with wearable devices and health tracking platforms to collect and analyse additional health data such as sleep patterns, stress levels, and blood glucose levels.
5. Gamification: Incorporate gamification elements such as challenges, badges, and rewards to motivate users and make the app more engaging and fun.
6. Accessibility Features: Implement accessibility features such as voice commands, screen readers, and text-to-speech functionality to make the app more inclusive and accessible to users with disabilities.

6.2.2 Hardware Module

Advanced Sensor Integration: Explore the integration of advanced sensors such as ECG sensors, blood pressure monitors, and temperature sensors to provide more comprehensive health monitoring capabilities.

1. Miniaturisation and Wearable Form Factors: Develop smaller and more lightweight hardware modules that can be easily integrated into wearable devices such as smartwatches, fitness bands, and clothing.
2. Long-Term Monitoring: Extend battery life and optimise power consumption to enable continuous long-term monitoring without frequent recharging or battery replacement.
3. Data Processing: Improve onboard processing capabilities to perform quickly analyse data and provide immediate feedback and insights to users.
4. Connectivity Enhancements: Enhance wireless connectivity options such as Bluetooth, Wi-Fi, and cellular connectivity to enable seamless data synchronisation and remote monitoring capabilities.
5. Security and Privacy Measures: Implement robust security and privacy measures to protect user data and ensure compliance with data protection regulations such as GDPR and HIPAA.
6. Customization and Personalization: Allow users to customise and personalise their hardware modules by choosing from different designs, colours, and accessories to suit their preferences and style.

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: 23-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: 23-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] A. Channa, N. Popescu, J. Skibinska, and R. Burget, "The Rise of Wearable Devices during the COVID-19 Pandemic: A Systematic Review," Sensors (Basel), vol. 21, no. 17, Aug. 2021, Art. no. 5787. DOI: 10.3390/s21175787. PMID: 34502679; PMCID: PMC8434481. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8434481/. [Accessed: 24-Mar-2024]

[6] "IoT for Fitness Assessment: A Systematic Review," MDPI Sensors, vol. 23, no. 20, 2023, Art. no. 8553. Available: https://www.mdpi.com/1424-8220/23/20/8553.[Accessed: 02-April-2024]

[7] "An Exploration and Confirmation of the Factors Influencing Adoption of IoT-Based Wearable Fitness Trackers," MDPI International Journal of Environmental Research and Public Health, vol. 16, no. 18, 2019, Art. no. 3227. Available: https://www.mdpi.com/1660-4601/16/18/3227. [Accessed: 12-April-2024]

[8] "Monitoring Fitness of Athletes Using IoT Enabled Wearables," Srinivas Publication, 2021. Available:https://srinivaspublication.com/wp-content/uploads/2021/04/6.-Tracking-and-Monitoring_Fullpaper.pdf. [Accessed: 01-April-2024]

[9] "A Novel Combination of Distributed Ledger Technologies on Internet of Things: Use Case on Precision Agriculture," ResearchGate,[Online]. [Accessed: 28-Mar-2024]

[10] 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].

[11] 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].

[12] 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].

[13] 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].

Appendix

A. Installation of Software Tools

A.1 Installing Thonny IDE – Windows PC

To install Thonny on wer Windows PC, follow the next instructions:

1. Go to https://thonny.org
2. Download the version for Windows(thony-ide-micropython-windows) and wait a few seconds while it downloads.
3. Run the .exe file.
4. Follow the installation wizard to complete the installation process. You just need to click “Next
5. After completing the installation, open Thonny IDE. A window as shown in the following figure should open.

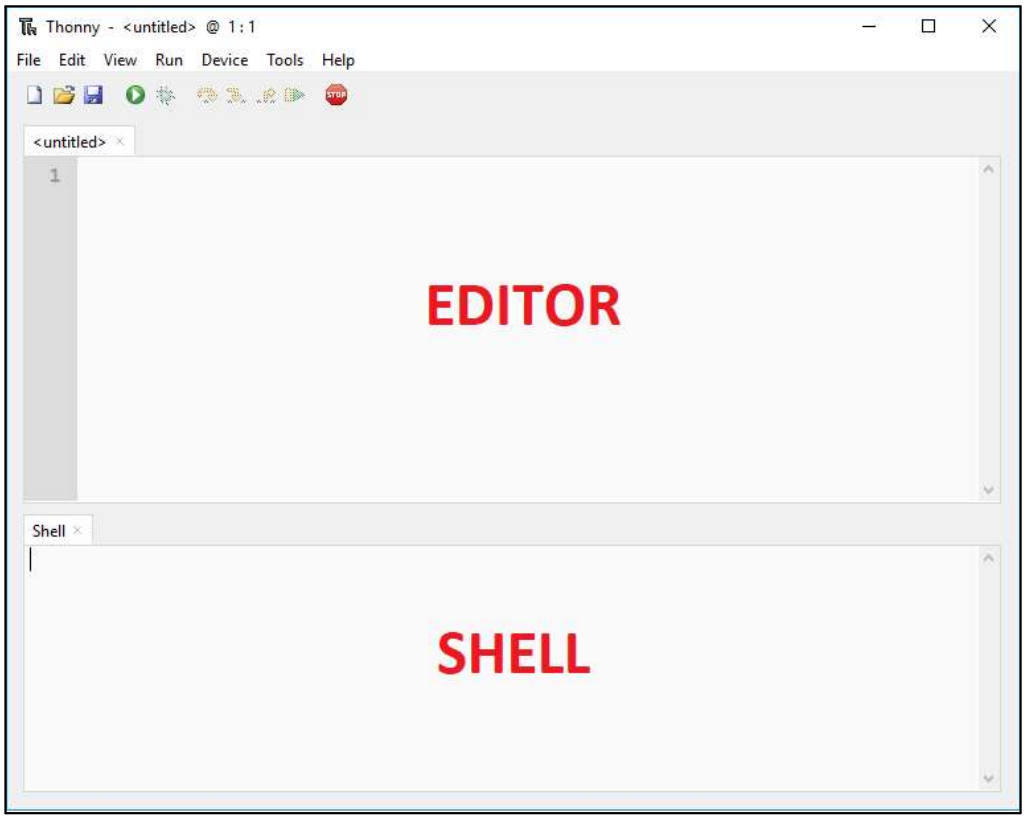

Fig. A.1 Thonny IDE

A.1.1 Installing esptool.py in System

To work with esptool.py, we'll need using Python 3.7.X, So we need to go Python's website and install it on our System:

With Python 3 installed, open a Terminal window and install the latest stable esptool.py release with,

- `pip install esptool`

Note: with some Python installations that command may not work and we'll receive an error. If that's the case, try to install esptool.py with:

- `pip3 install esptool`
- `python -m pip install esptool`
- `pip2 install esptool`

Setuptools is also a requirement that is not available on all systems by default. we can install it with the following command:

- `pip install setuptools`

After installing, we will have esptool.py installed into the default Python executables directory and we should be able to run it with the command esptool.py. In our Terminal window, run the following command:

- `python -m esptool`

A.1.2 Downloading and Flashing the MicroPython Firmware on ESP32

To download the latest version of MicroPython firmware for the ESP32, go to the MicroPython Downloads page and scroll all the way down to the ESP32 section.

we should see a similar web page (see figure below) with links to download .bin files. Download the latest release. At the time of writing this article, the latest release is v1.22.0 (2023-12-27) .bin.

A.1.3 Flashing MicroPython Firmware using Thonny IDE

In this section, we'll learn how to flash MicroPython firmware on our boards using Thonny IDE. Follow the next steps:

1. Connect our ESP32 or ESP8266 board to our computer.
2. Open Thonny IDE. Go to Tools > Options > Interpreter.
3. Select the interpreter we want to use according to the board we're using and select the COM port our board is connected to. Finally, click on the link Install or update firmware.

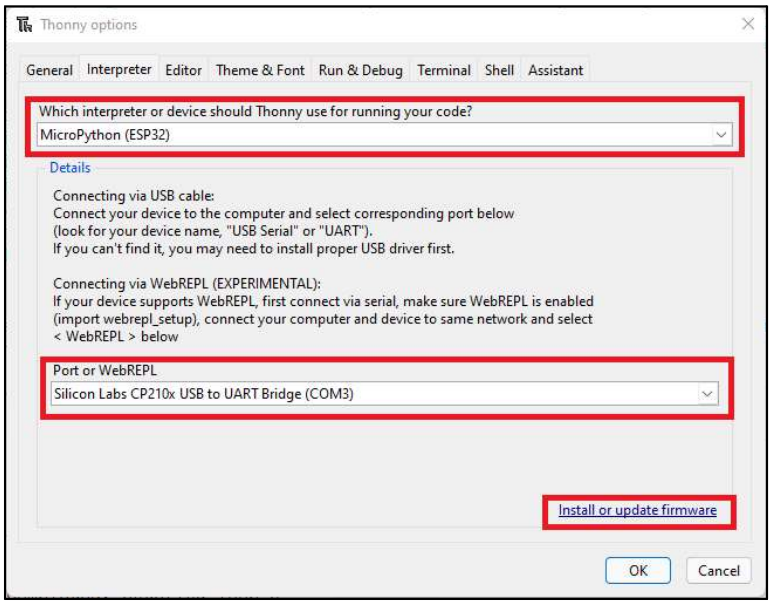

Fig. A.1.3 Thonny IDE Options View

4. Select the port once again, and then click on the Browse button to open the .bin file with the firmware we've downloaded on the previous step. Select the options and finally click on Install.

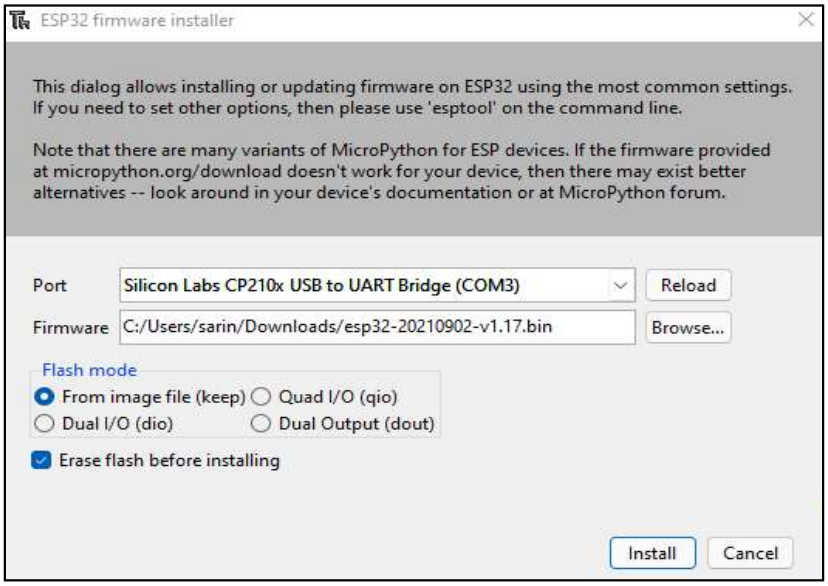

Fig. A.1.3.1 Firmware Installation on Thonny IDE

A.1.4 Testing the Installation

Connect the board to your computer using a USB cable. To test the installation, we need to tell Thonny that we want to run MicroPython Interpreter and select the board we are using.

1. Go to Tools > Options and select the Interpreter tab. Make sure we've selected the right interpreter for your board as well as the COM port.

- We can also select the "Try to detect automatically" option, but only if we just have one board connected to our computer at a time. Otherwise, select the specific port for the board you're using.

2. Thonny IDE should now be connected to our board and we should see the prompt on the Shell.

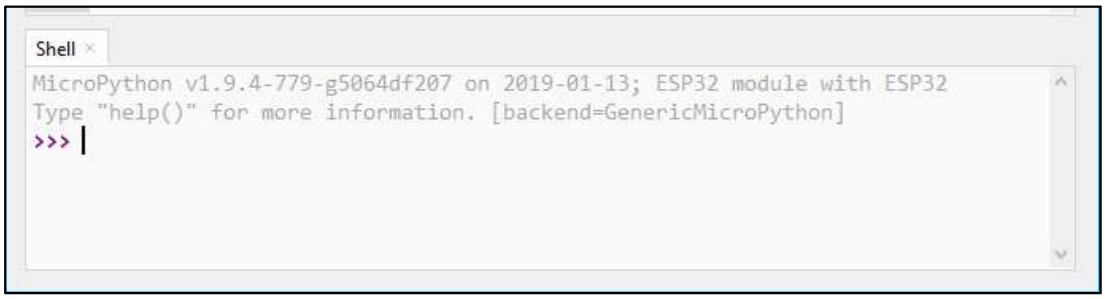

Fig. A.1.4 Thonny IDE Shell

3. Type the command help() in the Shell and see if it responds back.

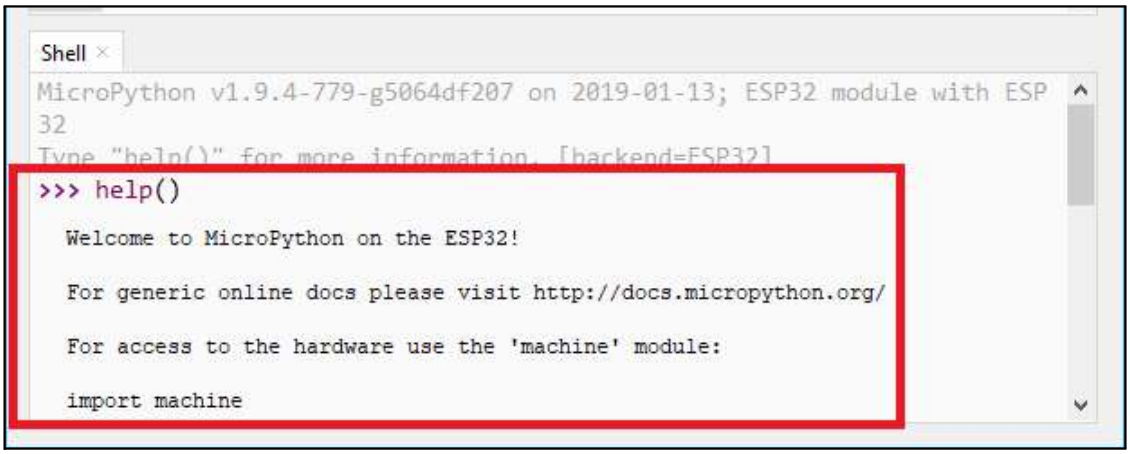

Fig. A.1.4.1 Thonny IDE Shell (Response)

4. If it responds back, everything is working fine. Now, we can send a few more commands to test.

Send the following commands to light up the on-board LED

```
>>> from machine import Pin
>>> Pin(2, Pin.OUT).value(1)
```

The on-board LED should light up.

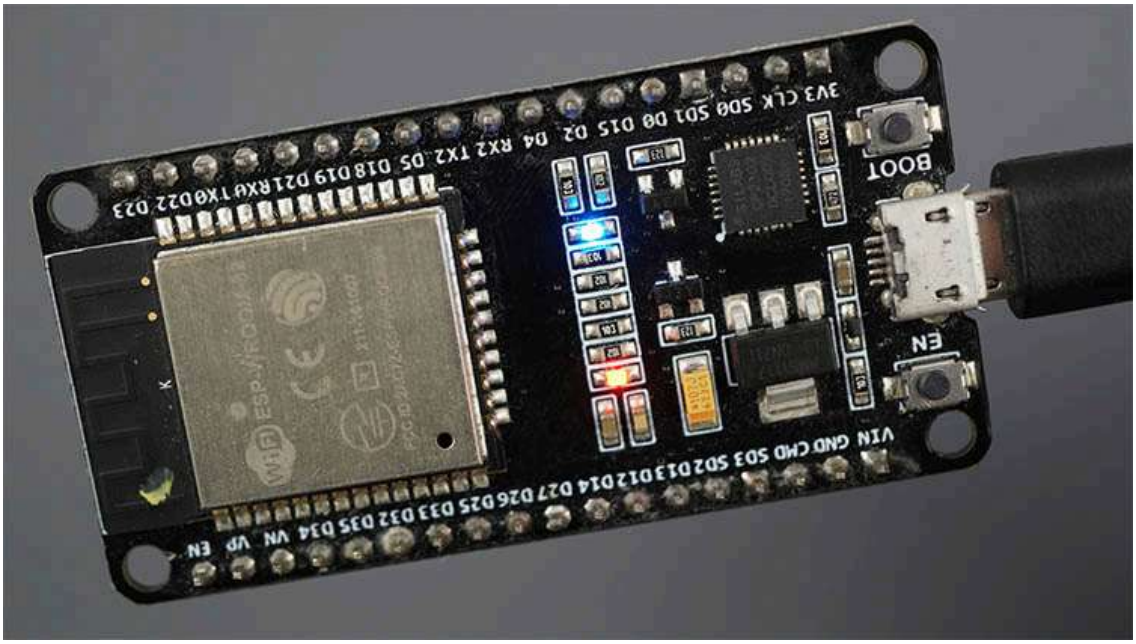

Fig. A.1.4.2 ESP32's LEDs Response

ESP32 board Built in LED turned on HIGH

- Then, turn off the led like this for the ESP32:

```
>>> Pin(2, Pin.OUT).value(0)
```

Congratulations! Your installation was successful!

A.2 Flutter Setup

A.2.1 Install and Configure Flutter SDK on System

Pre-installation Requirements

- Windows 10 operating system installed (Flutter will work on Windows 7 SP1 and later versions).
- At least 1.65 GB of free disk space (Additional free storage is needed for other tools and IDEs, if not already installed).
- Windows Powershell 5.0 or newer.
- Android Studio installed

After meeting all requirements, we can begin installing and configuring Flutter SDK. In today's tutorial, we will be installing a fixed installation of Flutter SDK, without using Git.

Step 1: Download Flutter SDK

Download the Flutter SDK package [flutter_windows_3.16.5-stable.zip] by clicking on the following button on the webpage.

Step 2: Extract the Files

Extract the downloaded zip file and move it to the desired location we want to install Flutter SDK. Do not install it in a folder or directory that requires elevated privileges, (such as C:\Program Files\) to ensure the program runs properly. For this tutorial, it will be stored in C:\development\flutter.

Step 3: Update Path Variable for Windows PowerShell

To run Flutter commands in PowerShell, add Flutter to the PATH environment variable.

1. Press Windows + S.
2. Type environment.
3. When Edit the system environment variables displays as the Best match, click Open under Edit the system environment variables.
4. Click About.
5. Click Advanced System Settings.
6. Click Environment Variables…
   The Environment Variables dialog displays.
7. Under User variables for <user> check for the Path entry.
   a. If the entry exists, click Edit….
   b. If the entry doesn't exist, click New….
   c. Click New.
   d. Type <install-directory>\flutter\bin.
   e. Click the <install-directory>\flutter\bin entry.
   f. Click Move Up until the Flutter entry sits at the top of the list.
   g. Click OK.
8. To enable these changes, close and reopen any existing command prompts and PowerShell instances.

If we have installed all prerequisites and the Flutter SDK, we should be able to start developing Flutter on Windows for desktop.

Step 4: Confirm Installed Tools for Running Flutter

The flutter doctor command validates that all components of a complete Flutter development environment for Windows.

1. Open PowerShell.
2. To verify your installation of all the components, run the following command.

```
C:\User\User\flutter> flutter doctor
```

```
C:\User\User\flutter> flutter doctor
Running "flutter pub get" in flutter_tools... 8,9s
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.10.4, on Microsoft Windows [Version
10.0.19041.746], locale en-US)
[X] Android toolchain - develop for Android devices
    X Unable to locate Android SDK.
    Install Android Studio from:
https://developer.android.com/studio/index.html
    On first launch it will assist you in installing the Android SDK
components.
    (or visit
https://flutter.dev/docs/get-started/install/windows#android-setup for
detailed instructions).
    If the Android SDK has been installed to a custom location, please use
    'flutter config --android-sdk' to update to that location.

[√] Chrome - develop for the web
[X]] Visual Studio - develop for Windows
     X Visual Studio not installed; this is necessary for Windows
development.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including
all of its default components
[!] Android Studio (not installed)
[√] Connected device (2 available)
[√] HTTP Host Availability

! Doctor found issues in 3 categories.
```

As visible, several components still need to be installed to complete the installation.

Step 5: Download and Install Android Studio

Continue by downloading Android Studio. In the setup, unless we have specific requirements, we can click Next on all screens leaving the default settings. Ensure that the Android Virtual Device option is selected on the Choose Components screen so that we can have an Android emulator running for Android app development.

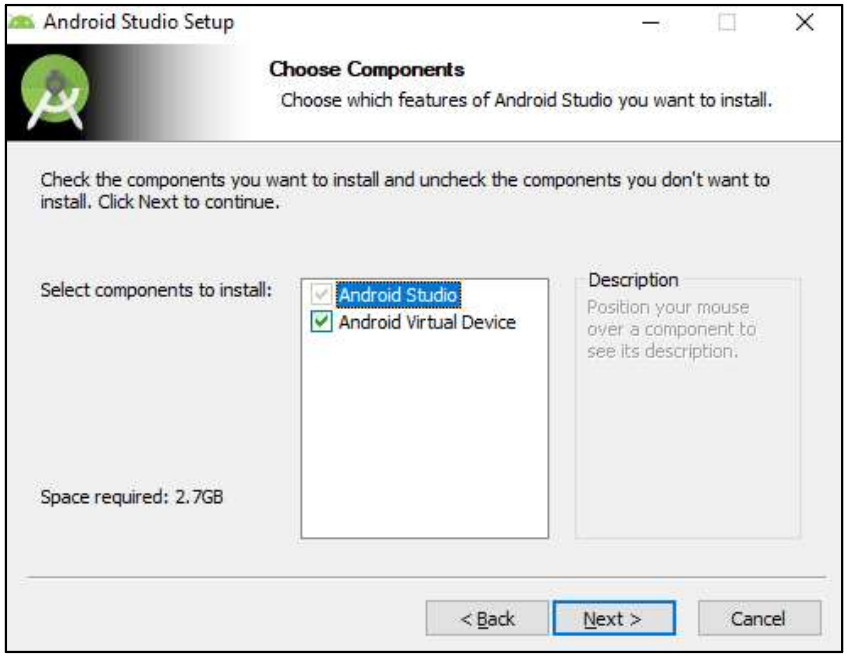

Fig A.2.1 Android Studio Installation

1. Select the installation location or leave the default path and click Next.
2. Select your SDK components and click Next.

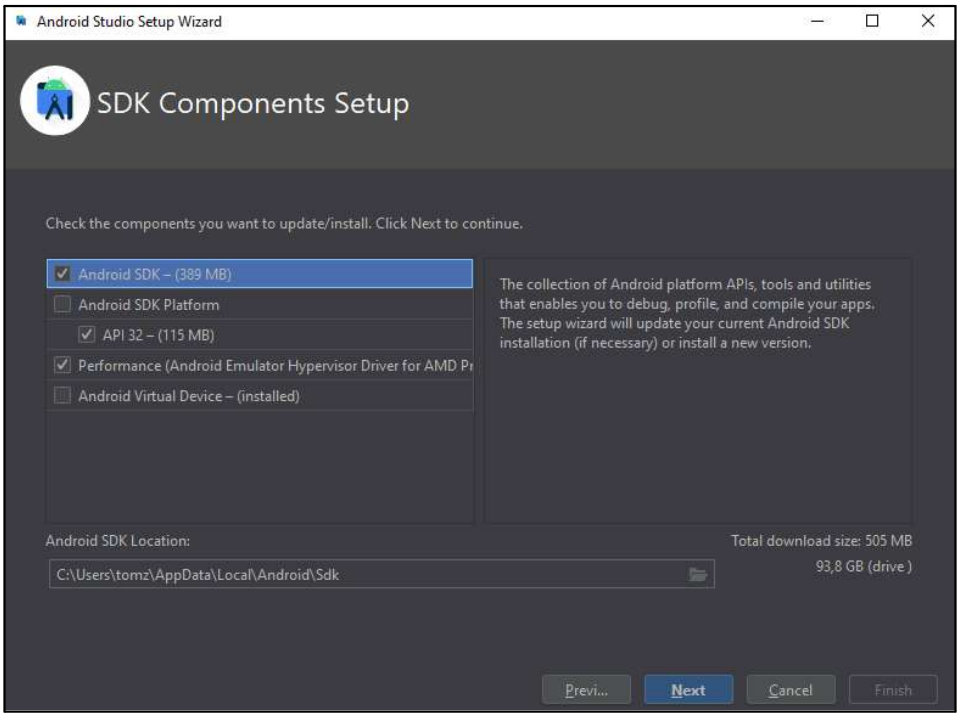

Fig A.2.1(a) Android Studio SDK Installation

3. On the next screen, accept the License Agreement and click Finish.

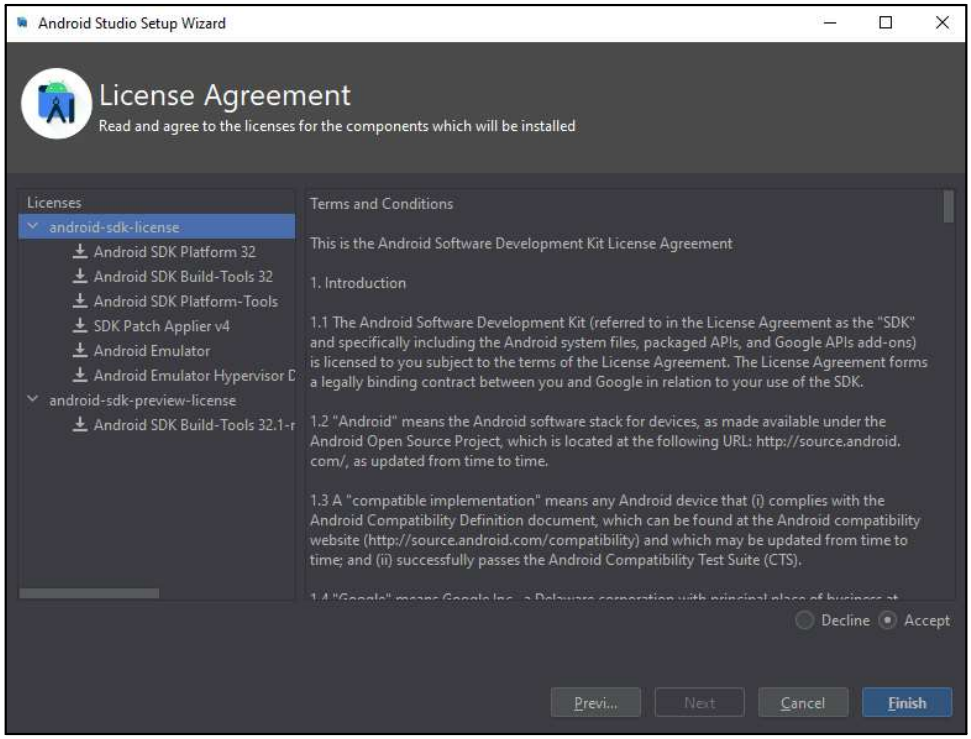

Fig A.2.1(b) Android Studio Licence Agreement

The download of the components will start and Android Studio will install. Once completed, click Finish.

4. After the installation, start Android Studio. On the left side, click Plugins. Search for Flutter and click Install to install the Flutter plugin

It will also prompt you to install Dart, a programming language used to create Flutter apps.

5. Click Install at the prompt.Finally, click Restart IDE so that the plugin changes are applied. Click Restart at the prompt to confirm this action.
6. In Android Studio, Go to Settings > Languages & Framework(expand) > Schemas and DTDs(expand) > Android SDK.
    - Here Go to SDK Tools, In This Check Following two and Install them.
        - ☑ Android SDK Command -line Tools (latest)
        - ☑ NDK (side by side)

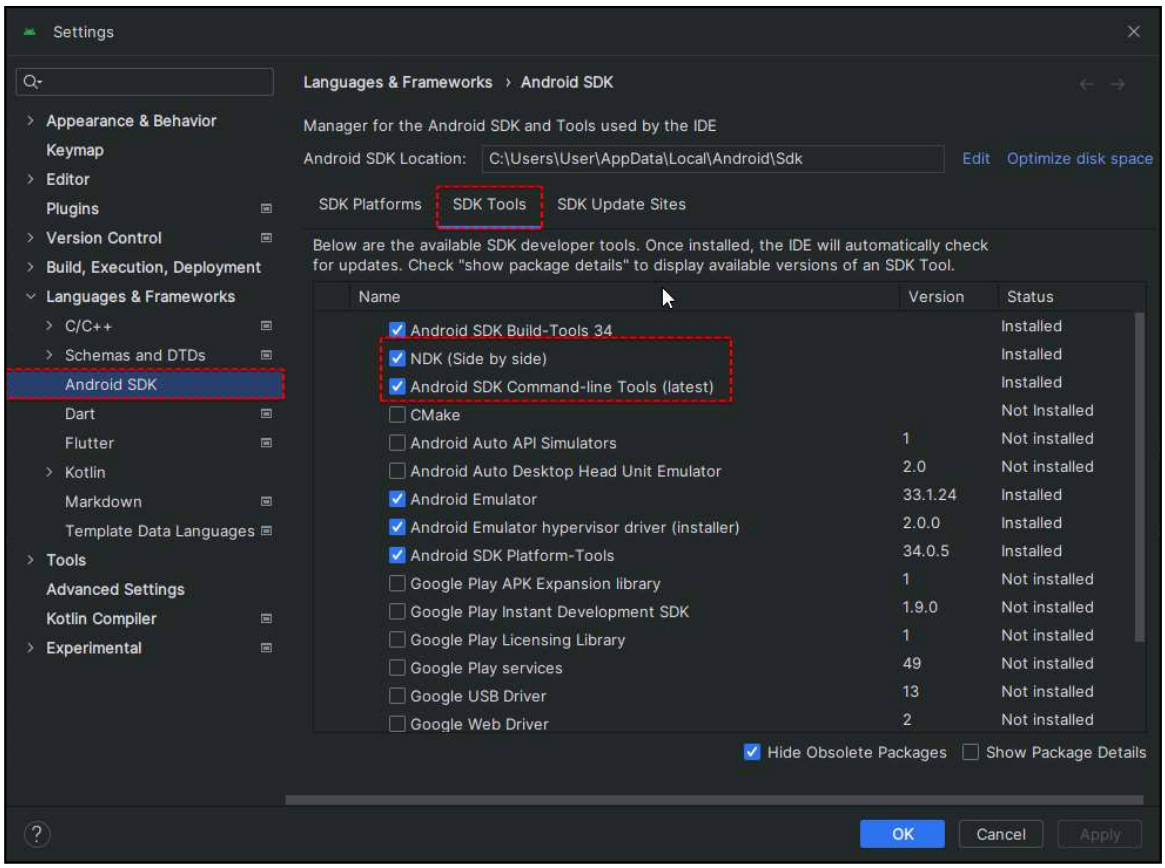

Fig A.2.1(c) SDK CL Tools in Android Studio

7. Open PowerShell.
8. Afterward, run the flutter doctor command in CMD to confirm the Android Studio installation.

```
C:\Users\User>flutter>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.10.4, on Microsoft Windows [version
10.0.19041.746), locale en-US)
[!] Android toolchain - develop for Android devices (Android SDK version
32.1.0-rc1)
    ! Some Android licenses not accepted. To resolve this, run: flutter
doctor --android-licenses
[√] Chrome - develop for the web
[X] Visual Studio - develop for Windows
    X Visual Studio not installed; this is necessary for Windows development.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++- workload, including
all of its default components
[√] Android Studio (version 2021.1)
[√] Connected device (2 available)
[√] HTTP Host Availability

! Doctor found issues in 2 categories.
```

9. Accept the Licence Agreements, run the following command.

```
C:\User\User\flutter> flutter doctor --android-licenses
```

10. When asked, input y to all prompts, to accept licenses.

```
C:\Users\User>flutter>flutter doctor --android-licences
5 of 7 SDK package licences not accepted. 100% Computing updates...
Review licences that have not been accepted (y/N)? y
```

A.3 Set Up Firebase with Flutter for Android Apps

A.3.1 Creating a New Flutter Project

Prerequisites

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
- Flutter and Dart plugins installed for Android Studio.
- Flutter extension installed for Visual Studio Code.

Step 1: Creating a New Flutter Project

Once you have your environment set up for Flutter, you can run the following to create a new application:

```
flutter create flutterfirebaseexample
```

Navigate to the new project directory:

```
cd flutter firebase example
```

Using flutter create will produce a demo application that will display the number of times a button is clicked.

Now that we've got a Flutter project up and running, we can add Firebase.

Step 2: Creating a New Firebase Project

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name:

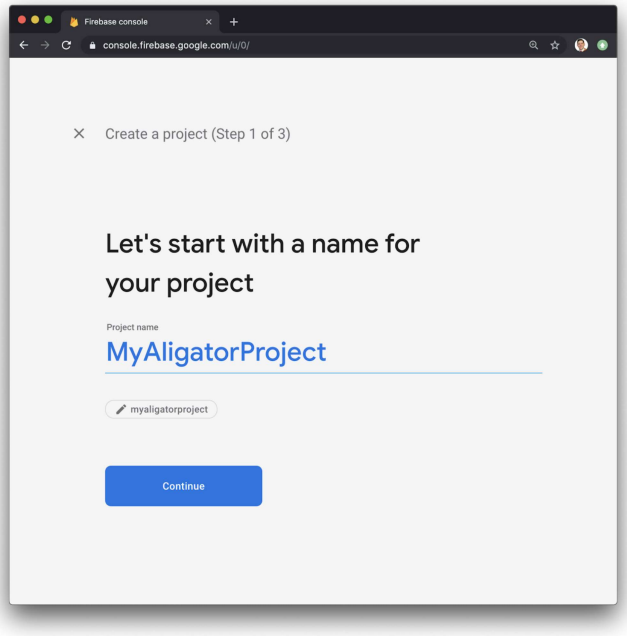

Fig A.3.1 Firebase Project Name

Next, we're given the option to enable Google Analytics. This tutorial will not require Google Analytics, but you can also choose to add it to your project.

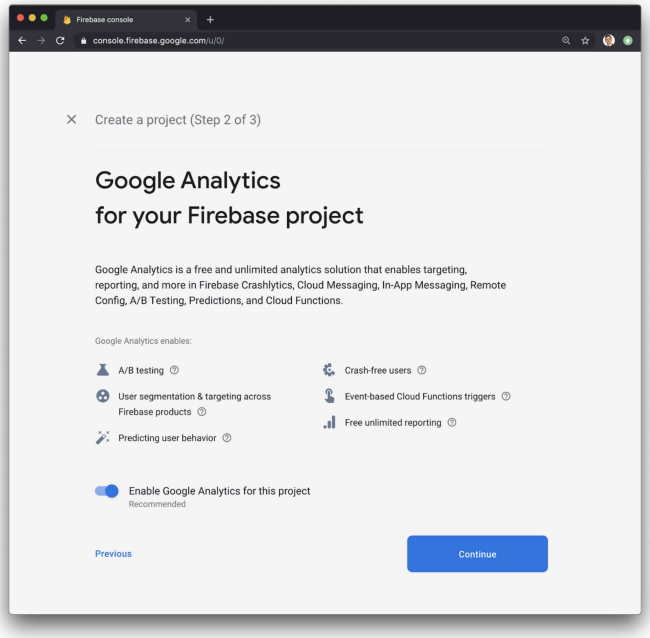

Fig A.3.2 Create Project

If you choose to use Google Analytics, you will need to review and accept the terms and conditions prior to project creation.

After pressing Continue, your project will be created and resources will be provisioned. You will then be directed to the dashboard for the new project.

Step 3: Creating a New Firebase Project

Registering the App

In order to add Android support to our Flutter application, select the Android logo from the dashboard. This brings us to the following screen:

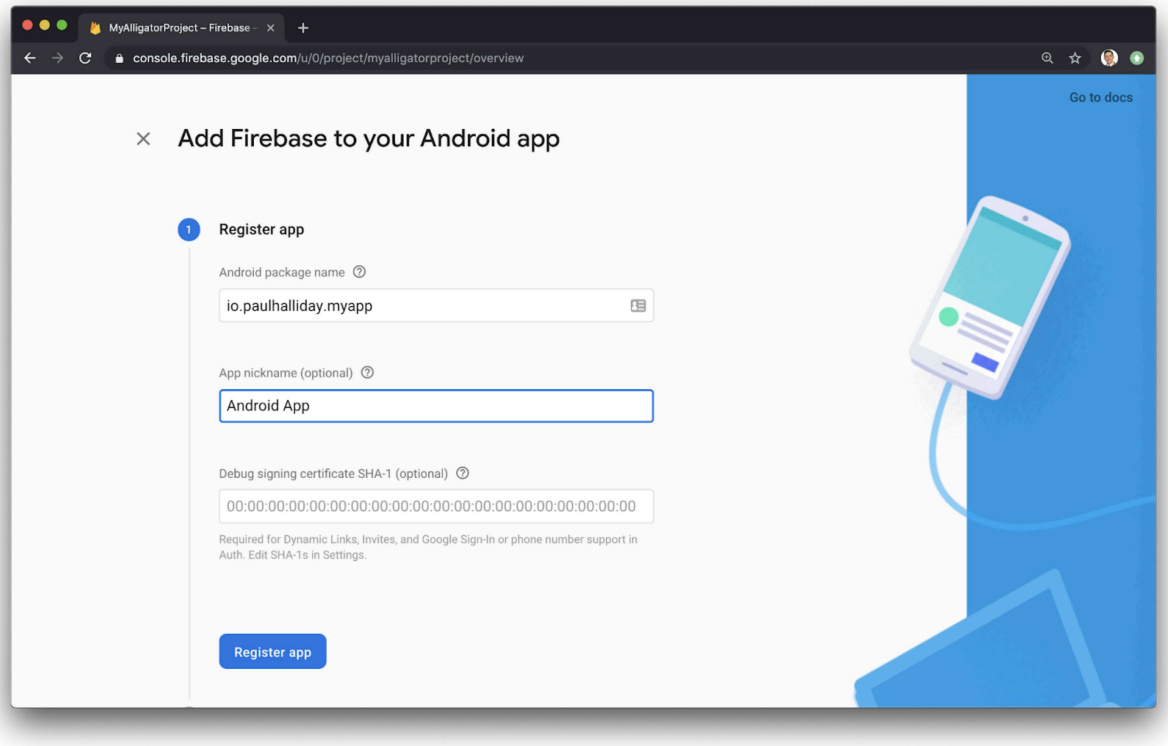

Fig A.3.3 Register The App

The most important thing here is to match up the Android package name that you choose here with the one inside of our application.

The structure consists of at least two segments. A common pattern is to use a domain name, a company name, and the application name:

```
com.example.flutterfirebaseexample
```

Once you've decided on a name, open android/app/build.gradle in your code editor and update the applicationId to match the Android package name:

```
...
defaultConfig {
    // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html)
.
    applicationId 'com.example.flutterfirebaseexample'
    ...
}
...
```

You can skip the app nickname and debug signing keys at this stage. Select Register app to continue.

Step 4: Downloading the Config File

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains the API keys and other critical information for Firebase to use.

Select Download google-services.json from this page:

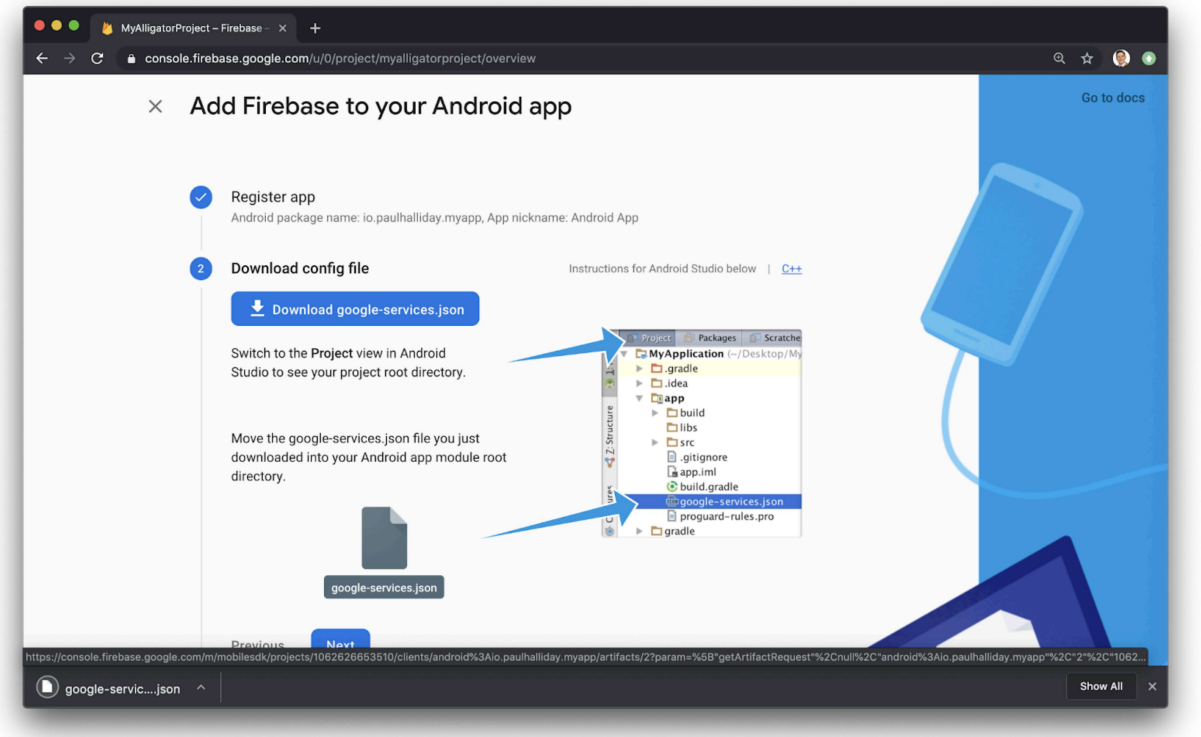

Fig A.3.4 Add The App

Next, move the google-services.json file to the android/app directory within the Flutter project.

Step 5: Adding the Firebase SDK

We'll now need to update our Gradle configuration to include the Google Services plugin.

Open android/build.gradle in your code editor and modify it to include the following:

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.6'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

Finally, update the app level file at android/app/build.gradle to include the following:

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation
platform('com.google.firebase:firebase-bom:28.0.0')

  // Add the dependencies for any other desired Firebase
products
  //
https://firebase.google.com/docs/android/setup#available-librari
es
}
```

With this update, we're essentially applying the Google Services plugin as well as looking at how other Flutter Firebase plugins can be activated such as Analytics.

From here, run your application on an Android device or simulator. If everything has worked correctly, you should get the following message in the dashboard:

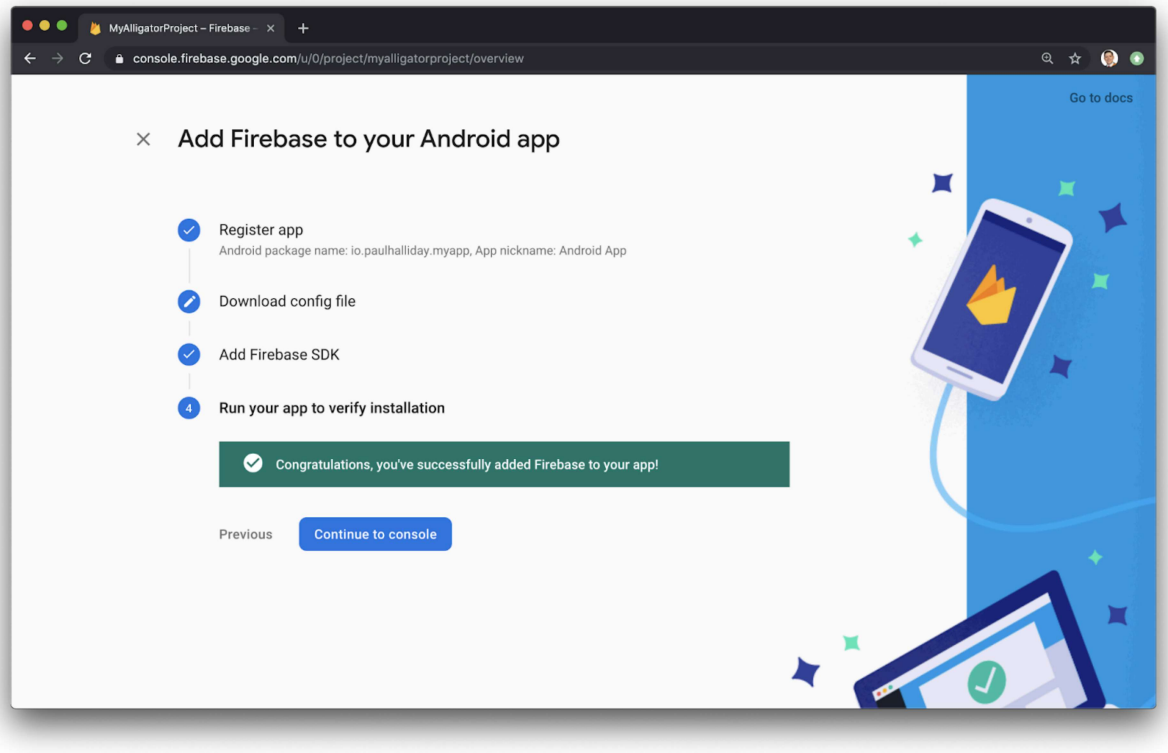

Fig A.3.5 Verify Installation