UNDERGRADUATE RESEARCH OPPORTUNITIES PROGRAMME (UROP)

PROJECT REPORT

Mobile Interface for Wearable Health Monitoring

Tong Zheng Hong

Department of Electrical Engineering, College of Design and Engineering

National University of Singapore

Semester 2, AY 2023/2024

Abstract

The rapid proliferation of mobile technology has propelled the development of mobile applications for health monitoring, revolutionizing the landscape of healthcare delivery. This report presents a comprehensive exploration of the design, implementation, and implications of such mobile applications, with a focus on interfacing with fiber-integrated wearable devices. Leveraging technologies such as React Native, Expo, and Skia, the development process prioritized the creation of an intuitive user interface (UI) to facilitate real-time health and environmental monitoring.

Key features of the mobile application include a summary page providing insights into vital health parameters, a listen page for real-time cardiogram visualization, and a history page for long-term data storage and analysis. The implementation of Bluetooth functionality using the React Native BLE PLX library enables seamless communication with wearable devices, while internal storage solutions such as Expo File System facilitate efficient file management.

Preface and acknowledgments

This project is supported by Assistant Professor Dr. Tural Khudiyev. I would like to thank my mentor Verdiyev Zulfugar for all the guidance that he has provided throughout the course of the project. I wish him all the best in his future endeavours.

Table of Contents

1. Introduction.. 4
2. Requirements..5
3. User Interface.. 6
  3.1 Design Framework.. 6
  3.2 Summary Page..6
  3.3 Listen Page..8
  3.4 History Page.. 8
4. Available Software Libraries.. 9
  4.1 React Native.. 9
  4.2 Expo.. 9
  4.3 React Native Paper.. 9
  4.4 Graphing Library..10
5. Software Implementation.. 11
  5.1 Bluetooth Implementation.. 11
  5.2 State Management.. 12
  5.3 Cardiogram Implementation.. 12
  5.4 Storage.. 13
6. Product (Vitasense).. 14
7. Conclusion.. 16

1. Introduction

As wearable technology becomes increasingly commonplace in the medical industry, there is a pressing need for efficient processing and analysis of the data they generate. This project addresses this by focusing on the development of a mobile application designed to interface with fiber-integrated wearable devices, streamlining the collection, processing, and visualization of health data.

The primary objective of this project is to gather data from wearable medical devices and present it in a format that is easily interpretable for medical personnel. This involves not only displaying current live data but also incorporating features to visualize processed data, facilitating the identification of long-term trends in the patient's health. By offering insights into both real-time and historical data, the application aims to empower healthcare professionals with the tools necessary to make informed decisions and manage patients.

Given the limited timeframe of one semester, the project prioritized key features to ensure timely delivery. Therefore, the primary focus was on user interface design and implementing the front-end software, emphasizing the importance of creating an intuitive and user-friendly interface. As a result, we are unable to incorporate code optimization and algorithmic enhancements due to the constrained timeframe. This report acknowledges the potential for further refinements in code optimization and algorithmic efficiency in future development.

In the subsequent sections of this report, we will focus the specific requirements of the mobile application, detailing the features and functionalities necessary to meet the objectives of the project. Additionally, considerable attention will be given to user interface design considerations, including the development of intuitive interfaces optimized for medical personnel's ease of use. Furthermore, the report will delve into the considerations of available software libraries, examining existing frameworks and tools utilized in the development process. This section will explore the rationale behind the selection of software libraries and their integration into the application architecture. Lastly, the report will provide an overview of the software implementation process, detailing the coding methodologies and integration of Bluetooth Low Energy (BLE) technology to facilitate seamless communication with wearable sensors. Through the examination of these key components, the report aims to offer insights into the practical aspects of translating project objectives into tangible software solutions.

2. Requirements

The development of the mobile application necessitates a clear understanding of the requirements which are dependent on the specifications of the wearable sensors and the desired outcomes of the project.

The hardware sensors will measure raw sound values in terms of voltage. These raw data inputs can be processed to derive essential health parameters such as heart rate, blood pressure, and respiratory rate. The mobile application will receive data from the hardware sensor via Bluetooth Low Energy (BLE). It is designed to handle both raw data and processed data streams, ensuring a comprehensive view of the patient's health status.

Regarding raw data, the sensors will sample sound/voltage data at a frequency of 50 kHz. Before transmission via Bluetooth, the raw data undergoes filtering and processing, resulting in an expected reception rate of 100Hz for the mobile application. Future developments in sensor hardware aspect may allow for higher transmission rates of up to 1 kHz. The raw data should be presented in a format similar to a cardiogram, enabling users to visualize the waveform of the signal.

Processed data will be presented in a manner that showcases the most recent information across various timeframes. This feature enables medical personnel to monitor patients' current health status efficiently.

Furthermore, the mobile application is tasked with storing processed data as historical records. Given the long-term nature of this data, the application will only display the range and average values of each type of processed data over different time frames. This historical data storage and visualization functionality provide valuable insights into patients' health trends, facilitating informed decision-making in patient care and management.

It is essential to note that the mobile application is intended for both Android and iOS devices. However, the development process will primarily focus on Android compatibility initially, with plans for subsequent adaptation to iOS platforms.

3. User Interface

In order to streamline the design process and visualize the UI layout efficiently, the application's UI is crafted using Figma, a UI design tool. UI prototyping allows for iterative design improvements and ensures alignment with user needs and expectations before software development begins.

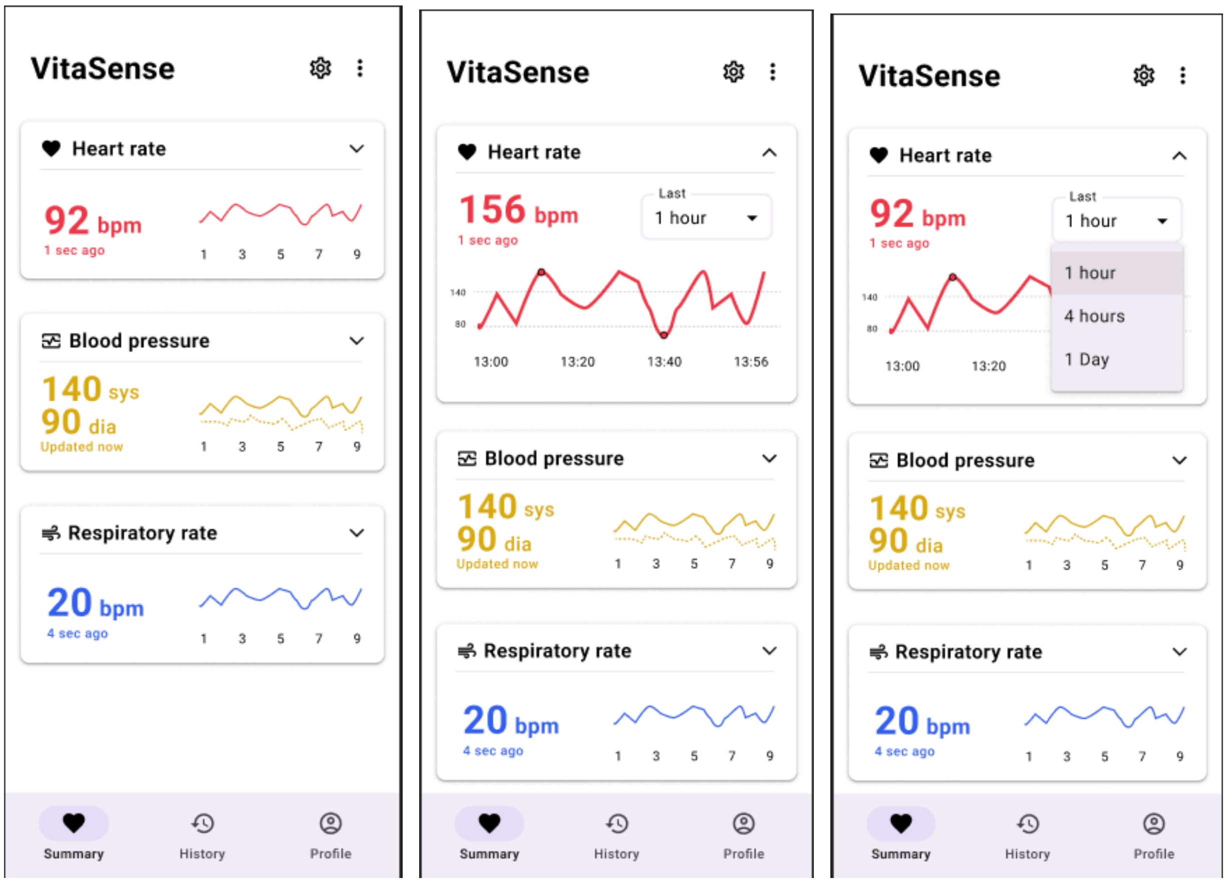
The mobile application will feature three main pages, summary, listen and history page, each tailored to fulfill specific user needs.

3.1 Design Framework

A design framework in UI refers to a collection of pre-defined guidelines, principles, and components that facilitate the creation of consistent and visually appealing user interfaces. We opted to utilize the Material Design framework. Material Design, developed by Google, offers a cohesive and visually appealing design language characterized by clean layouts, subtle animations, and intuitive interactions. This framework aligns with Android's design principles, enhancing the application's usability and familiarity for Android users.

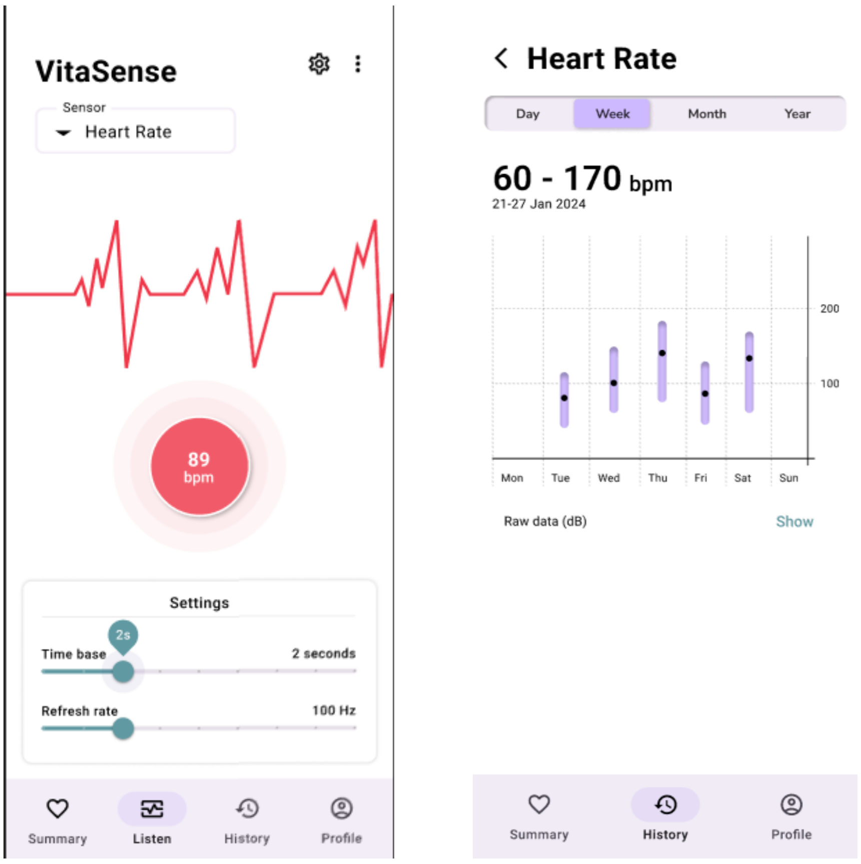
3.2 Summary Page

The summary page serves as a centralized hub, providing an overview of essential health parameters such as heart rate, blood pressure, and respiratory rate over the last 4 hours. The design of this page incorporates expandable cards for each processed data field, offering a concise yet comprehensive view of the patient's health status.



Summary pages with expandable cards

When expanded, users can pan over the graph to view the data value at a specific time, enhancing data interpretation and analysis capabilities. Additionally, the summary page features a time frame selector, enabling users to adjust the time frame displayed on the graphs. This empowers medical personnel to identify trends and fluctuations in health parameters over varying durations.



Time base 2 seconds

Refresh rate 100 Hz

Listen page (left) and history page (right)

3.3 Listen Page

The listen page offers medical personnel a comprehensive view of raw sound data measured by the sensor, presented in a cardiogram format. This presentation allows users to visualize the waveform of the signal, facilitating in-depth analysis and diagnosis.

In addition to the default presentation, the listen page features customizable settings that allow users to modify various parameters, providing a tailored view of the signal waveform.

3.4 History Page

Lastly, the history page functions as a repository for long-term data storage, drawing inspiration from Apple Health's interface. Data is presented using bars that represent the range of values observed during a specific interval. Additionally, a dot within each bar represents the average value recorded during that interval. This visual representation enables users to quickly grasp the variability and average trends of each health parameter over different time frames, enabling medical personnel to make informed decisions and interventions based on longitudinal data trends.

4. Available Software Libraries

This section aims to introduce the concept of leveraging software libraries, which offer ready-to-use components, to streamline the development process of the mobile application. These libraries provide pre-built functionalities that help implement various features without the need to create them from scratch.

4.1 React Native

The primary library utilized in this project is React Native, a cross-platform mobile application library that facilitates the development of applications for both Android and iOS platforms using TypeScript. Traditionally, Android applications are written in Kotlin, while iOS applications are developed using Swift. However, React Native allows for unified development using TypeScript, simplifying the process of building applications for multiple platforms.

4.2 Expo

To expedite development further, the project utilizes the Expo framework. Expo streamlines the process of building TypeScript code into the respective native code bases, enhancing development efficiency. Additionally, Expo's Expo Go app facilitates testing by enabling developers to view changes made to the application in real-time.

4.3 React Native Paper

For UI design consistency and efficiency, React Native Paper is employed. React Native Paper is a Material Design UI library that provides pre-styled components such as buttons, cards, and navigation bars according to Material Design guidelines. This library ensures a cohesive and visually appealing user interface across the application.

4.4 Graphing Library

To meet the visualization requirements outlined in the Figma design, a suitable graphing library is necessary. The chosen graphing library must be easy to implement, performant, and offer a wide range of features. The desired features include the ability to support hold gestures, allowing users to view current data points with ease. Additionally, the library should offer scrollable functionality, enabling users to navigate through data sets effortlessly. These features enhance user interaction and data exploration, facilitating intuitive and efficient analysis of health parameters within the mobile application.

During the development phase, it became evident that React Native Gifted Charts did not meet our performance expectations. Specifically, when users performed a pan gesture to view data, the graph re-rendered each time, resulting in a suboptimal user experience. To address this issue, we sought an alternative solution and implemented line graphs using another library called Skia.

Skia is a powerful 2D graphics engine which can render animations on a separate UI thread, distinct from the main JavaScript thread. This architecture enables smoother animations and pan gestures, as any changes to the graph's state are handled in the background. Moreover, Skia executes code in C++, which is significantly faster due to its closer resemblance to machine code, enhancing overall performance.

In conjunction with Skia, the D3 library was used to process the data into a format conducive to Skia operations. D3, known for its robust data visualization capabilities, proved invaluable in transforming raw data into a structured format suitable for rendering within the Skia environment.

However, leveraging Skia posed a challenge as it solely functions as a graphics engine. Consequently, every component of the graph, including axes, axis labels, line graphs, visual gradients, and the cursor used during pan gestures, had to be rendered from scratch. While this approach necessitated a longer development time, it ultimately yielded smooth animations with minimal dropped frames, greatly enhancing the user experience.

5. Software Implementation

5.1 Bluetooth Implementation

For the Bluetooth implementation, we employed the React Native BLE PLX library, a versatile tool for integrating Bluetooth Low Energy (BLE) functionality into our mobile application. The implementation consists of four main parts:

1. Requesting Permissions: This involves obtaining the necessary permissions from the user to access Bluetooth functionality on their device.
2. Scanning for Devices: The application scans for nearby Bluetooth devices and displays them to the user for selection.
3. Connecting to Device: Once a device is selected, the application establishes a connection to it, enabling communication.
4. Streaming Data from Device: Upon successful connection, the application streams data from the Bluetooth device, facilitating real-time data acquisition.

Due to time constraints, only the permissions and device scanning functionalities have been thoroughly tested in our implementation. The code for the Bluetooth functionality was referenced from the following GitHub page: BLESampleExpo.

Testing the Bluetooth functionality requires a physical device, as it cannot be adequately tested on emulators. Therefore, we utilized the Expo developmental build for testing purposes. However, testing on iOS devices presents challenges, as it requires an Apple Developer account to install apps on physical iOS devices. As a result, our testing and development efforts have primarily focused on Android devices, where no such restrictions exist.

In order to establish a connection and stream data from a Bluetooth device, we require specific UUIDs (Universally Unique Identifiers) and characteristic numbers to identify and communicate with the Bluetooth device effectively. These identifiers enable our application to establish a secure and reliable connection for data transmission.

5.2 State Management

State management in refers to the management of the data of an application, ensuring that it remains accessible throughout its lifecycle. It enables components within an application to communicate and interact effectively, ensuring that changes in one part of the application are reflected accurately in other parts.

In our implementation, we utilized the default React useState hooks to manage values that change dynamically. React components are structured in a tree format with hierarchies, and the useState hook provides components and their children access to a variable. When the variable changes, it triggers a rebuild of the entire component and its children. This process can be computationally expensive, particularly in more complex user interfaces.

Currently, the Bluetooth implementation has not been extensively tested, and as such, the visualizations are constructed using fixed random data. Therefore, the basic useState hook suffices for managing the state in our application. However, there is room for improvement in terms of data management.

Moving forward, it would be prudent to adopt a more comprehensive state management framework that re-renders specific components based on data changes rather than rebuilding entire components and their children. One such framework is Redux. Redux creates a global application state, allowing for more granular control over state changes and rendering only specific components affected by those changes. This approach enhances performance and scalability, particularly in applications with complex user interfaces and extensive data interactions.

5.3 Cardiogram Implementation

For the implementation of the cardiogram functionality, we devised a strategy to accommodate the limitations of Skia animations, which can run at a maximum of 60 frames per second (fps). Given that live data received from Bluetooth may arrive at varying frequencies, our current implementation is structured as follows:

1. Data received from Bluetooth is stored in an array buffer, ensuring efficient data management and accessibility.
2. With each frame drawn approximately every 17 milliseconds (1/60 Hz), we calculate the number of pixels to be displayed in each frame rerender.
3. In cases where the buffer contains multiple data points during a frame rerender, we evenly distribute the incoming data into groups. The number of groups corresponds to the number of pixels displayed in each rerender.
4. Subsequently, we compute the average value of each group and display the resulting value for the corresponding pixel.

By adopting this approach, we effectively adhere to the animation limit of 60 fps and ensure that the number of pixels displayed on a device remains within manageable bounds. This strategy optimizes the visualization of live data in the cardiogram, maintaining smooth animation and responsive user interaction while accurately representing the dynamic nature of the data being processed.

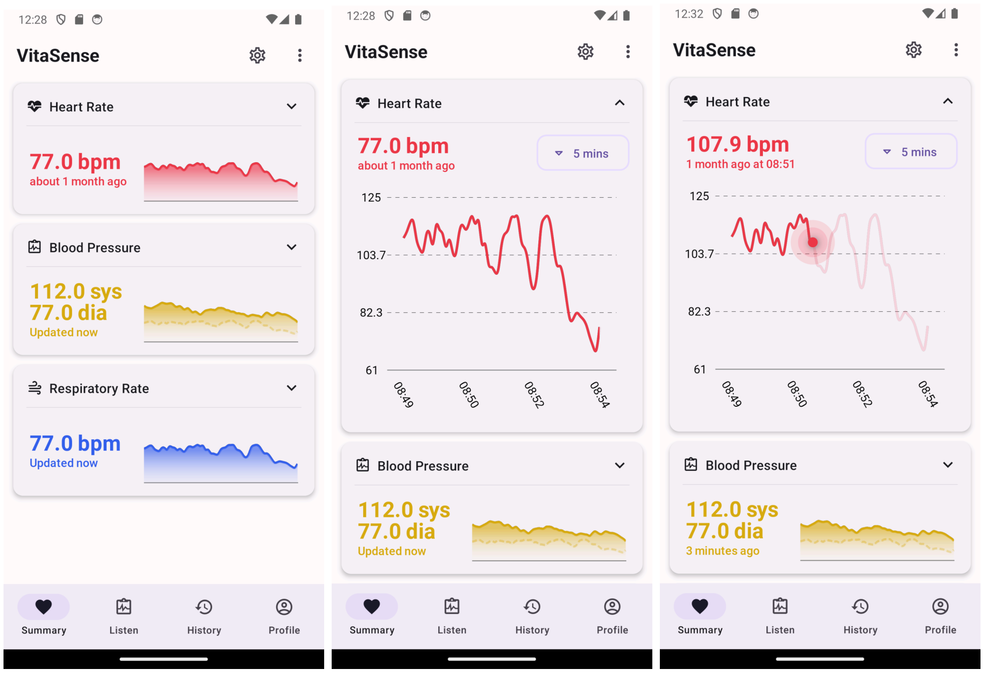
5.4 Storage

In terms of storage, mobile applications typically utilize both internal and external storage options. Internal storage refers to the dedicated storage space that each app possesses, inaccessible to other applications. Conversely, external storage encompasses the standard storage available on the physical device, accessible to all applications. Dealing with external storage can be more complex and is not currently necessary for our project's requirements. Therefore, our focus has been primarily on leveraging internal storage capabilities.

To facilitate file storage operations within the internal storage of the device, we utilized the Expo File System. This library provides functionalities to save and read files from the internal storage of the device seamlessly. By leveraging Expo File System, we ensure efficient and secure file management within our application, without the need to navigate the complexities associated with external storage handling.

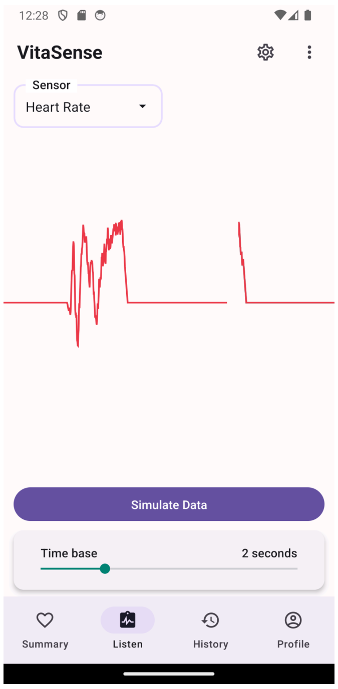
6. Product (Vitasense)

The current state of the mobile application reflects the culmination of our development efforts, incorporating various features to enhance user experience and functionality. The codebase for the application can be accessed on GitHub at VitaSense. The instructions to build and run the developmental build can be found in the README file.

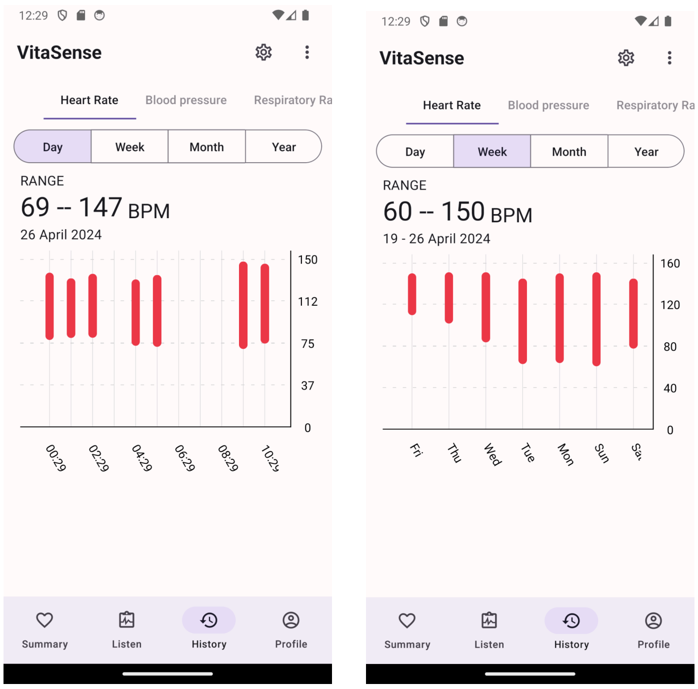


Implementation of Summary page

The summary page closely resembles the initial design, providing users with a comprehensive overview of vital health parameters such as heart rate, blood pressure, and respiratory rate over the last 4 hours. The implementation aligns with our design objectives, offering a concise and informative overview of the patient's health status.



Listen page



History page

The listen page features an animation that runs on randomly generated data when the "Simulate data" button is clicked. This functionality allows users to interact with the application and visualize the cardiogram representation of the raw data in real-time. On the history page, random ranges are used to simulate intervals, providing users with a glimpse of the long-term data storage and visualization capabilities of the application. While the data presented is randomly generated, it offers a preview of how health data will be displayed and accessed within the application.

7. Conclusion

In conclusion, notable progress has been made in the development of the mobile application interfacing with fiber-integrated wearable devices. Within the confines of a single semester, the majority of our efforts have been directed towards crafting a user interface (UI) that aligns with the project objectives. This focus on UI design has enabled us to create an intuitive and visually appealing platform for real-time health and environmental monitoring.

However, it is essential to acknowledge the inherent limitations of the short timeframe. As such, the Bluetooth implementation and data management within the application, have received less attention than desired. Moving forward, future development efforts should prioritize these critical areas, ensuring seamless connectivity with wearable devices and robust data management capabilities.

Additionally, while storage implementation was briefly explored using internal storage solutions, such as Expo File System, there is significant room for expansion. More extensive storage solutions, such as cloud storage, could be integrated to enhance data accessibility, scalability, and security.

Overall, the project has provided valuable insights into mobile app development, wearable technology integration, and the application of computer science in health monitoring.