

CodeBlue

Design Document

Modular Phone App for Cardiac Arrest Detection

PREPARED BY

CPEN/ ELEC 491 Team JY-41

Akash Randhawa, Emily Lukas, Gurman Toor, Sean Garvey, Stella Wang

Version 4.0

April 10, 2023

Table of Contents

1. Overview	7
2. Technological Decisions	7
2.1 Operating System	7
2.2 Framework	8
2.2.1 Expo	10
2.3 Wireless Communication	10
2.3.1 Type of connection	11
2.3.2 Bluetooth type	11
2.4 CA Monitoring Algorithm Implementation	12
2.5 Accessibility Features	13
2.5.1 Audible Ping	14
2.5.2 Vibration	14
3. High-Level Design	14
3.1 System Diagrams	14
3.1.1 User Interaction Flow Chart	14
3.1.2 CodeBlue System Diagram	15
4. User Interface (UI)	17
4.1 Architecture	17
4.1.1 Modified MVVM Example	18
4.1.2 Alternative Architectures	19
4.2 Navigation	20
4.3 Design Considerations	21
4.4 Sequence Diagrams	22
4.4.1 Onboarding Flow	22
5. Key Components	23
5.1 Data Processing	23
5.1.1 Motivation	23
5.1.2 PapaParse	23
5.1.3 Alternatives	24
5.2 CA Monitoring Algorithm	25
5.2.1 Motivation	25
5.2.2 Fourier Transform	25
5.2.3 Peak Detection	25

5.2.4 Confidence	26
5.3 Cloud Server Hosting	26
5.3.1 Motivation	26
5.3.2 Amazon Elastic Compute Cloud (EC2)	26
5.3.3 Node and Express server	27
5.3.4 Alternative cloud hosting options	28
5.4 EMS Communication	29
5.4.1 Motivation	29
5.4.2 React Native Immediate Phone Call	30
5.4.3 React Native Phone Call Options	30
5.4.4 Enhanced 911	31
5.5 Local Data Storage	31
5.5.1 Motivation	31
5.5.2 React Native MMKV	31
5.5.3 Comparison of Local Storage Libraries	32
5.6 Push Notification	33
5.6.1 Motivation	33
5.6.2 Firebase Cloud Messaging	33
5.6.3 Triggering Push Notifications	35
5.6.4 Receiving FCM notifications	35
5.6.5 Alternative Push Notification Services	37
5.7 Background Processing	38
5.7.1 Motivation	38
5.7.2 Notiffee	39
5.7.3 Comparison of Background Processing Approaches	40
5.8 Bluetooth Low Energy (BLE)	41
5.8.1 Motivation	41
5.8.2 React-Native-Ble-Plx	41
5.8.3 Alternative Bluetooth Low Energy Libraries	42
6. Artifacts	42
6.1 UI Components	42
6.1.1 Onboarding	43
6.1.2 Home screen	43
6.1.3 Add New Sensor Device	44
6.1.4 Settings	44
6.1.5 Emergency Protocol	45

6.2 Repository Structure	45
6.3 Source Code	46
6.4 User Guide	46
7. References	47
8. Appendix	48
A. Contributions	48

List of Tables and Figures

Table 2.1 - Comparison of mobile operating systems	7
Table 2.2 - Comparison of mobile development frameworks	9
Table 2.2.1 - Comparison of using Expo framework	10
Table 2.3.1 - Comparison of Bluetooth and Wifi	11
Table 2.3.2 - Comparison for Classic Bluetooth and BLE	11
Table 2.4.1 - Comparison for using ML in the CA monitoring algorithm	12
Figure 3.1.1 - User interaction flow-chart for CodeBlue	15
Figure 3.1.2 - System Diagram for CodeBlue	16
Figure 4.1.1 - Modified MVVM architecture	17
Figure 4.1.2 - Onboarding Required Info screen	18
Table 4.1.3 - Comparison of alternative React Native architectures	19
Figure 4.2 - CodeBlue navigation structure	21
Figure 4.4.1 - Onboarding Sequence Diagram	22
Table 5.1.3 - Comparison of alternative CSV parser libraries	24
Figure 5.2.1 - Signal Analysis	25
Table 5.3.3 - High level overview of CA monitoring algorithm	28
Table 5.3.4 - Comparison of cloud hosting options for hosting monitoring algorithm	28
Table 5.4.3 - Comparison of React Native phone call options	30
Table 5.5.4 - Comparison of local storage libraries	32
Figure 5.6.2 - High-level overview of push notification architecture	34
Figure 5.6.4.1 - Cardiac Arrest Background Notification	36
Figure 5.6.4.2 - EP screen with running countdown	37
Table 5.6.5 - Comparison of push notification services	37
Figure 5.7.3 - Comparison of Background Processing Implementations	40
Figure 6.1.1 - Onboarding flow	43

Figure 6.1.2 - Home screen	43
Figure 6.1.3 - Add New Device flow	44
Figure 6.1.4 - Settings flow	44
Figure 6.1.5 - Emergency Protocol flow	45
Figure 6.2 - CodeBlue repository structure	45

1. Overview

CodeBlue is a mobile app built for the Android operating system using the React Native framework without Expo that supports Bluetooth connections and makes decisions based on available sensor data to monitor for Cardiac Arrest (CA) and contact Emergency Medical Services (EMS).

2. Technological Decisions

CodeBlue is built for the Android operating system using the React Native framework without Expo.

2.1 Operating System

The **Android** platform was chosen to satisfy all functional and non-functional requirements. Pros and cons for the choices of mobile operating system can be seen in Table 2.1.

Option	Pros	Cons
Android	<ul style="list-style-type: none">• Straightforward integration with Bluetooth• All members are able to access tools to develop on the Android platform	<ul style="list-style-type: none">• Building specifically for Android may limit user base

iOS	<ul style="list-style-type: none"> • Straightforward integration with Apple products • More widely used in North America than Android 	<ul style="list-style-type: none"> • Previous capstone team ran into issues pulling reliable sensor data from Apple watch (could not access implementation details due to Apple secrecy) • Not all team members are able to develop for iOS since Xcode is only available on Macs
-----	---	---

Table 2.1 - Comparison of mobile operating systems

The Android operating system was chosen so all team members could participate in developing the mobile app, and to avoid the difficulties the previous capstone team encountered with iOS.

2.2 Framework

React Native without Expo was chosen to satisfy functional requirement F2 and achieve non-functional requirement NF3. A comparison of the frameworks considered is shown below in Table 2.2.

React Native is a mobile app development framework that allows cross-platform apps to be built using a single codebase written in JavaScript or TypeScript. React Native is part of the React ecosystem and shares many similarities with its popular web counterpart, React.js. Flutter is a mobile app development framework that allows cross-platform apps to be built using a

single codebase written in Dart. Lastly, Android Studio is a native development platform that specifically targets the Android operating system.

Option	Pros	Cons
React Native	<ul style="list-style-type: none">• Group members have previous experience with React Native• Future cross-platform compatibility, maximizes reach• Large component library• Good community support	<ul style="list-style-type: none">• TypeScript is single threaded, run the risk of interfering with UI performance if the app frequently polls for a bluetooth response
Flutter	<ul style="list-style-type: none">• Faster performance than React Native• Future cross-platform compatibility, maximizes reach• Up-to-date, modern libraries• Trendy, ready to install visual components	<ul style="list-style-type: none">• App sizes are considerably large• Platform has risks due to slower growing community• Library support is not as rich as native development

Android Studio	<ul style="list-style-type: none"> • Out-of-the-box support for Bluetooth • Simple to use 	<ul style="list-style-type: none"> • Does not have cross-platform compatibility
----------------	---	--

Table 2.2 - Comparison of mobile development frameworks

2.2.1 Expo

Expo is a React Native framework that enables fast development by pre-configuring settings, removing the need for Xcode and Android Studio.

Pros	Cons
<ul style="list-style-type: none"> • Able to develop for both platforms without Android Studio or Xcode • Comes pre-packaged with tools and services aimed to speed up development time 	<ul style="list-style-type: none"> • Some iOS and Android Application Programming Interfaces (APIs) are not available • No Bluetooth API; requires several workarounds to develop for Bluetooth

Table 2.2.1 - Comparison for using Expo framework

React Native without Expo was chosen because of its cross-platform compatibility and ability to support all of CodeBlue's key functionalities.

2.3 Wireless Communication

Bluetooth Low Energy (BLE) was chosen to satisfy functional requirements F1 and F2 and non-functional requirements NF1 and NF3. A comparison of the

different wireless communication types are shown below in Table 2.3.1 and Table 2.3.2.

2.3.1 Type of connection

Specifications	Bluetooth	Wi-Fi
Bandwidth	Low	High
Ease of use	Simple, easy to switch devices	Complex, requires configuration
Hardware requirement	Bluetooth adapter on device	Wireless adapter on device and wireless router
Range	Most devices operate between to 10-100m	100 m
Power consumption	Low	High
Frequency range	2.4~2.483 GHz	2.4 GHz and 5 GHz
Flexibility	Supports limited number of users	Provides support for a large number of users

Table 2.3.1 - Comparison of Bluetooth and Wifi [1]

2.3.2 Bluetooth type

Specifications	Classic Bluetooth	Bluetooth Low Energy
Range	Up to 100m	Greater than 100m
Data Rate	1-3 Mbps	1 Mbps
Application Throughput	0.7 - 2.1 Mbps	0.27 Mbps
Frequency	2.4 GHz	2.4 GHz
Security	56/128-bit	128-bit AES with Counter Mode CBC-MAC

Robustness	Adaptive fast-frequency hopping, FEC, fast ASK	24-bit CRC, 32-bit Message Integrity Check
Latency	100 ms	6 ms
Time Lag	100 ms	3 ms
Voice Capable	Yes	No
Network Topology	Star	Star
Power Consumption	1W	0.01W to 0.5W
Peak Current Consumption	Less than 30 mA	Less than 15 mA

Table 2.3.2 - Comparison for Classic Bluetooth and Bluetooth Low Energy [2]

Bluetooth allows short-range wireless connection for sensors to send cardiovascular data to CodeBlue and has lower cost and power than Wi-Fi, as shown by Table 2.3.1. Additionally, BLE was chosen because it has significantly less power consumption than classic Bluetooth and is therefore better suited for a long task like cardiac monitoring.

2.4 CA Monitoring Algorithm Implementation

Monitoring without Machine Learning (ML) was chosen to satisfy functional requirements F1 and F3 and non-functional requirements NF1 and NF2. A comparison of the different possible implementation alternatives are shown below in Table 2.4.1 and Table 2.4.2.

Option	Pros	Cons
Using ML	<ul style="list-style-type: none"> Model would be constantly learning 	<ul style="list-style-type: none"> Out of scope for CodeBlue

	providing better predictions of cardiac arrests <ul style="list-style-type: none"> • Could lead to less false positives 	<ul style="list-style-type: none"> • Requires a large dataset to produce accurate results • Not necessary at this point for the sensor fusion algorithm to monitor for cardiac arrests
No ML	<ul style="list-style-type: none"> • Does not require a large amount of test data for training purposes • Faster implementation time, no training period required 	<ul style="list-style-type: none"> • Does not have continuous adaptability and accuracy improvement

Table 2.4 - Comparison for using ML in the CA monitoring algorithm

Monitoring without ML was chosen because our client does not have enough data to train a model that can accurately monitor for cardiac arrest.

2.5 Accessibility Features

The usage of **audible pings and vibrations** was chosen to satisfy functional requirement F4 and non-functional requirement NF3. Adding these accessibility features makes the Emergency Protocol more effective.

2.5.1 Audible Ping

Implementing audible ping helps CodeBlue alert bystanders and guide EMS responders to the victim in the case of a CA.

2.5.2 Vibration

Implementing vibration allows for better accessibility for the hearing impaired. It also provides an additional alert mechanism for CAs. However, we run the risk of high power usage.

3. High-Level Design

3.1 System Diagrams

3.1.1 User Interaction Flow Chart

The diagram outlines the interaction flow for a CodeBlue user. The arrows represent the possible routes a user might observe based on their actions.

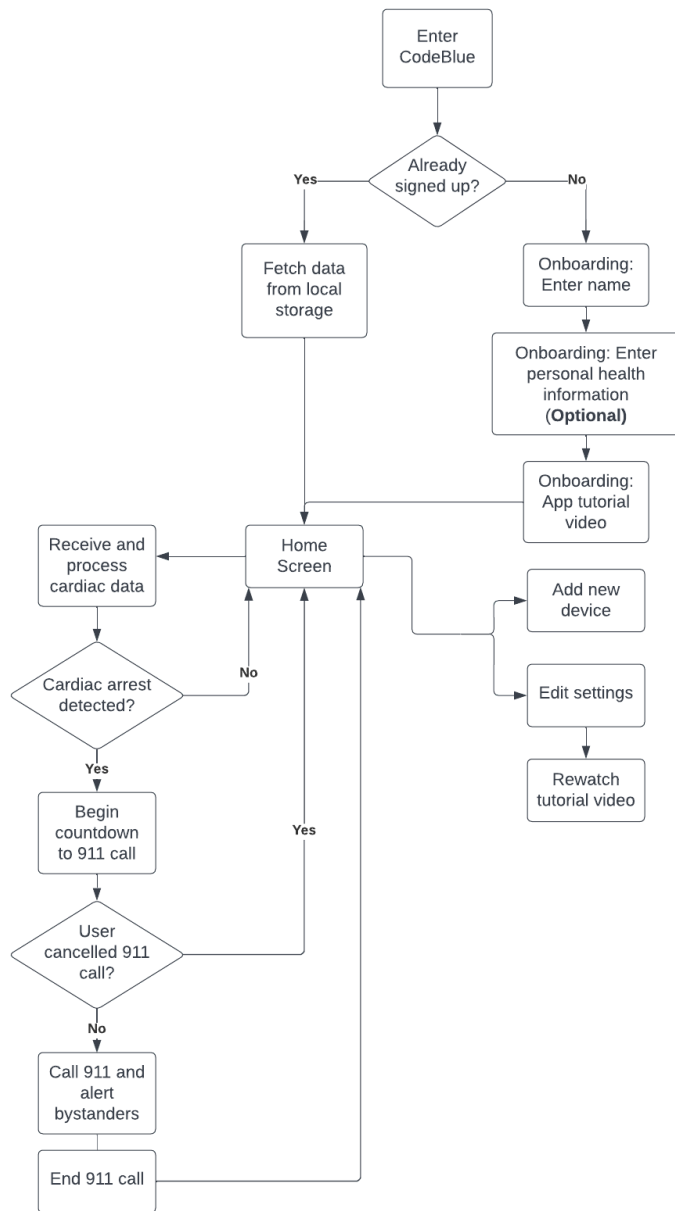


Figure 3.1.1 - [User interaction flow-chart](#) for CodeBlue

3.1.2 CodeBlue System Diagram

The diagram outlines the system architecture of the CodeBlue project.

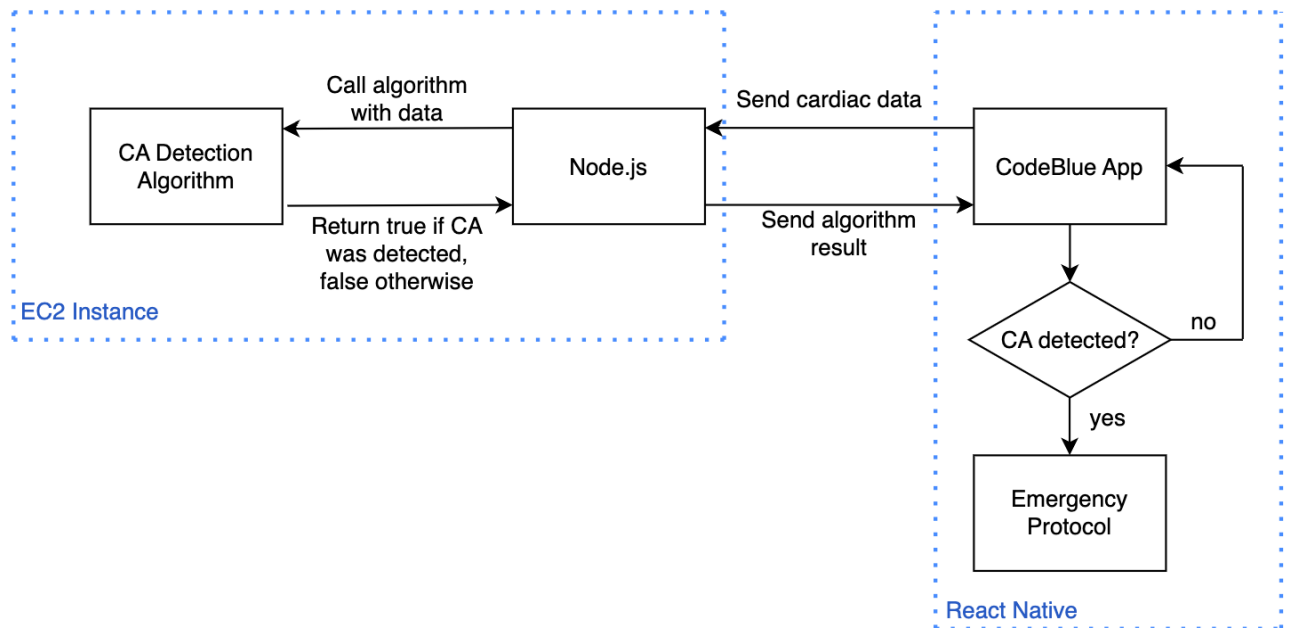


Figure 3.1.2 - System Diagram for CodeBlue

4. User Interface (UI)

The UI code for the app implements the user interaction flow-chart seen in Figure 3.1. It includes all React Native screens and design system components.

The mobile app UI consists of three major flows:

1. Onboarding flow and tutorial
2. Home screen and device management
3. Emergency Protocol (EP)

After the first installation of the app, the user is taken through a tutorial of how to use CodeBlue and given the option to enter some basic information if they choose to. The app's main content is rooted on the Home screen, which allows users to monitor their heart rate, manage their Bluetooth devices, and edit their app settings. Lastly, the EP flow is responsible for alerting the user in the case of a cardiac arrest and calling EMS.

4.1 Architecture

CodeBlue uses a **modified Model-View-ViewModel (MVVM)** architecture to implement the UI. Figures 4.1 and 4.1.1 demonstrate the MVVM architectures in practice, and Table 4.1.2 outlines the alternative options.

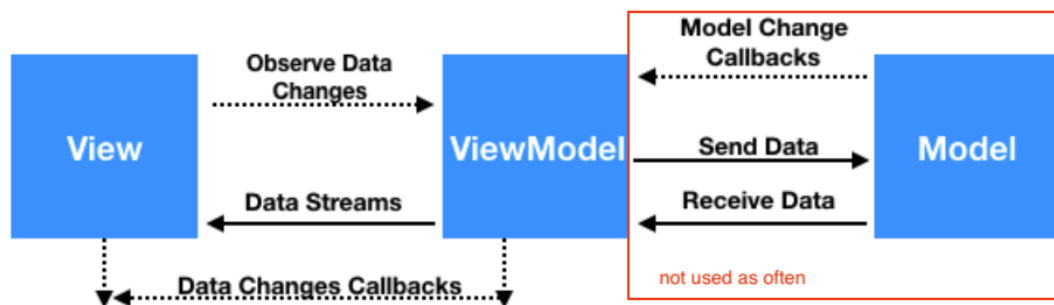


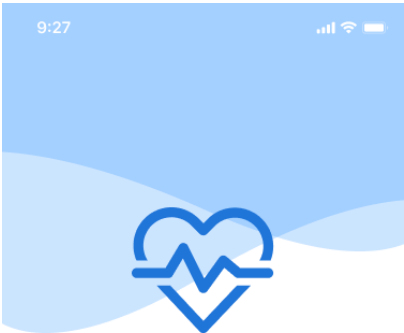
Figure 4.1.1 - Modified MVVM architecture [3]

Modified MVVM is chosen because the app's logic does not live inside the React Native codebase in most cases. For example, the CA monitoring algorithm will be implemented outside the app code. The only instance of a true Model layer is the local storage wrapper that saves cardiac and app data.

This pattern is applied for the entire UI, where design system components (i.e. text boxes, buttons, etc) act as Views and screens are synonymous with ViewModels. The local storage module functions similarly to the Model.

4.1.1 Modified MVVM Example

In the following example, the InputText components act as the View and the RequiredInfoScreen component as the ViewModel.



9:27

Lets get started

First Name

Last Name

Next

RequiredInfoScreen.tsx

```
const [firstName, setFirstName] = useState('');  
...  
<InputText  
  placeholder="First Name"  
  text={firstName}  
  onChangeText={setFirstName}  
  width={windowWidth * 0.9}  
>  
</>
```

InputText.tsx

```
const InputText = ({  
  text,  
  onChangeText  
): Props => {  
  ...  
<TextInput  
  value={text}  
  onChangeText={onChangeText}  
>  
</>
```

Figure 4.1.2 - Onboarding Required Info screen

The RequiredInfoScreen component feeds data to the InputText component via the text and onChangeText fields. InputText implements an input text field, using text to represent the current user input and onChangeText to update the value of text. Note that internally, text points to RequiredInfoScreen's firstName and onChangeText points to setFirstName. Therefore, after RequiredInfoScreen (ViewModel) feeds initial values into InputText (View), any changes seen by InputText are immediately sent back to RequiredInfoScreen, completing the ViewModel-View lifecycle.

4.1.2 Alternative Architectures

Option	Pros	Cons
Redux	<ul style="list-style-type: none">• Ideal for Emergency Procedure where there are clear states and transitions• Guarantees a source of truth across the app	<ul style="list-style-type: none">• High overhead to set up; lots of boilerplate code• Best suited for complex apps with many states• Hard to determine how often the store should be updated with incoming data
React Context	<ul style="list-style-type: none">• Easy to set up, little extra code required• Can assign each flow to its own Context and easily store flow-specific global variables	<ul style="list-style-type: none">• Can get bulky very fast if there are many global variables (such as user information)• State transitions are done asynchronously

Table 4.1.3 - Comparison of alternative React Native architectures

The CodeBlue UI does not contain enough complex logic to justify the high implementation overhead of Redux, as the monitoring algorithm operates outside of the mobile codebase. Therefore, Modified MVVM is implemented because it provides the best modularity for UI components and integrates easily with local storage, replacing the need for React Context.

4.2 Navigation

CodeBlue navigation uses the React Navigation library. Each major flow either has its own Stack or Tab navigator, and all screens are tied together using a global Stack navigator. The navigation flow can be seen in Figure 4.2 below.

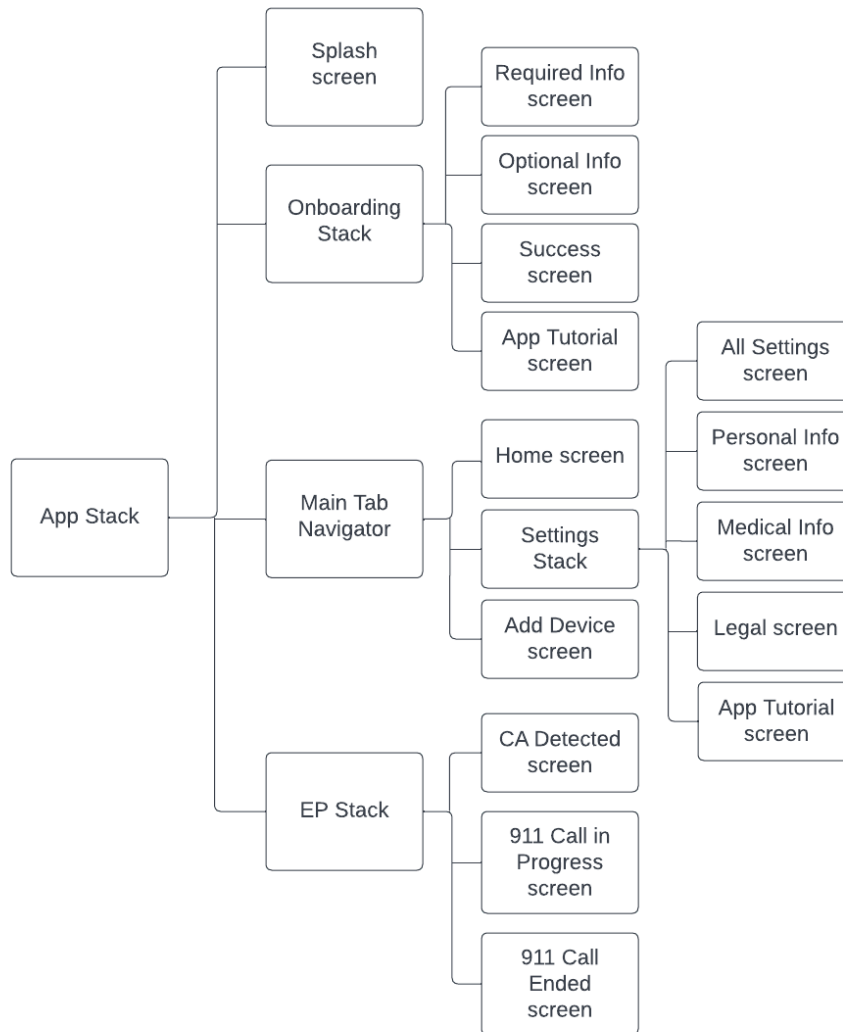


Figure 4.2 - CodeBlue navigation structure

4.3 Design Considerations

There are a number of design considerations when constructing the UI:

- Design should be simple and easy to navigate for users who are less adept with smartphones and wearable technology
- Colour scheme should feature muted tones for a professional, medical feel
- Design should be fool-proof and convey information users need to see

- Design must account for accessibility needs
 - Haptic feedback for hearing impaired users
 - Accessible font size and choice
 - Colours must be chosen in a manner where colourblind users are able to easily navigate the app and view all information

4.4 Sequence Diagrams

CodeBlue's onboarding flow communicates with the user's device local storage. The onboarding flow sequence can be seen in Figure 4.4.1 below.

4.4.1 Onboarding Flow

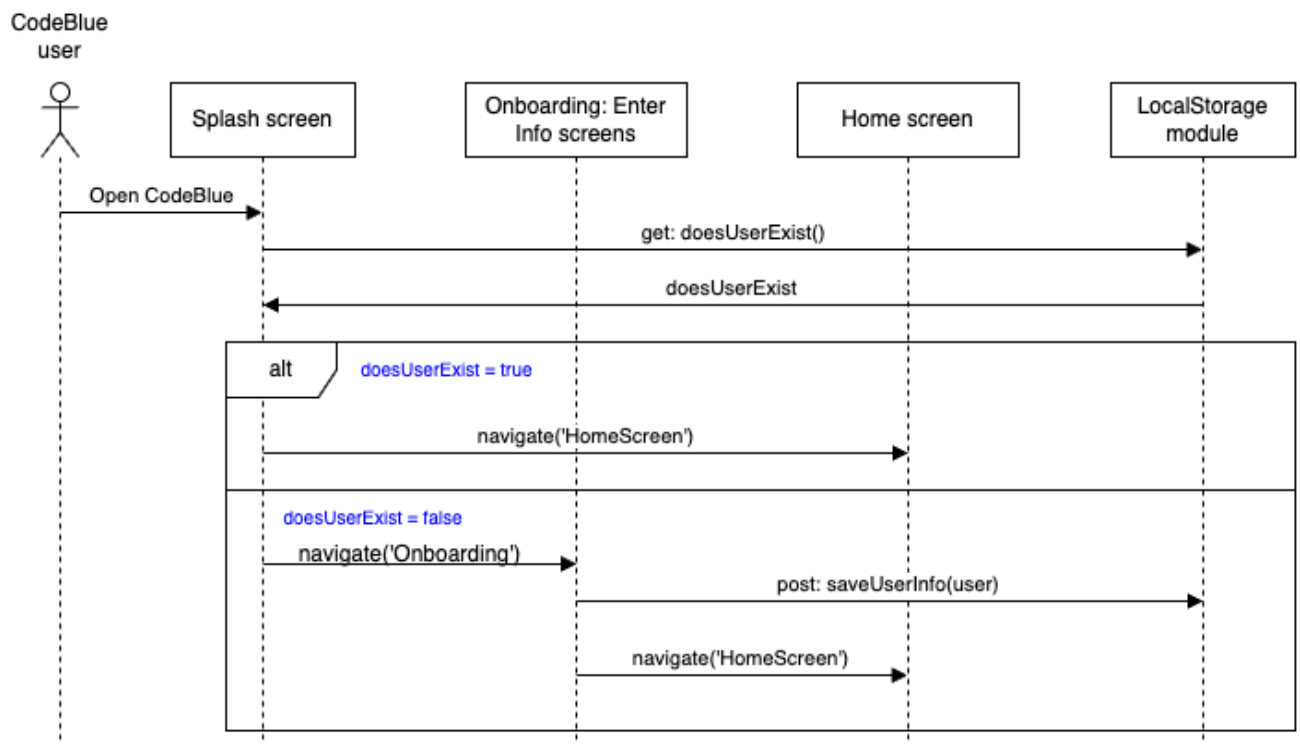


Figure 4.4.1 - Onboarding Sequence Diagram

5. Key Components

5.1 Data Processing

5.1.1 Motivation

Processing incoming data allows CodeBlue to ensure that the data being received is in the correct format and can be used in our CA monitoring algorithm.

5.1.2 PapaParse

During the MVP phase, CodeBlue consumes data in the Comma Separated Values (CSV) format. In order to process this data, CodeBlue utilises the React Native library, [PapaParse V5.0](#). PapaParse has the compatibility to convert the data from an entire CSV file into JSON format along with having the ability to read each individual line one at a time. In addition to this conversion, PapaParse has sync, stream, callback, and async iterators and provides support for large datasets. The alternatives to PapaParse can be seen in Table 5.1.3.

For the purposes of CodeBlue, the data will be parsed one line at a time to mimic the incoming data from a wearable device. Once the data is received from the CSV file and converted into JSON format, it is passed onto the CA Monitoring Algorithm.

5.1.3 Alternatives

Option	Pros	Cons
PapaParse	<ul style="list-style-type: none">• Supports data streaming from CSV• Fastest library available	
CSV-Parser	<ul style="list-style-type: none">• Supports data streaming from CSV• Has callback functions to modify header and values	<ul style="list-style-type: none">• Performance and speed is slower than PapaParse
Fast-CSV	<ul style="list-style-type: none">• Supports data streaming from CSV• Has callbacks to transform rows	<ul style="list-style-type: none">• Slower parsing speed than PapaParse
Dekkai	<ul style="list-style-type: none">• Automatic type conversion in binary mode• Iterates over data and invokes callback for each row	<ul style="list-style-type: none">• Still in pre-release, so there may be bugs with certain features• Has issues when dealing with large datasets

Table 5.1.3 - Comparison of alternative CSV parser libraries

Although all of the options were quite similar in terms of their functionality, PapaParsers ability to parse both small and large datasets at a faster speed set it apart from the others.

5.2 CA Monitoring Algorithm

5.2.1 Motivation

CodeBlue's purpose is to act as a supervisor of the user's cardiac health. In order to do so, it receives semi-processed data from sensors located on the user in order to calculate the user's heart rate and monitor other signs of declining cardiac function, satisfying F3.

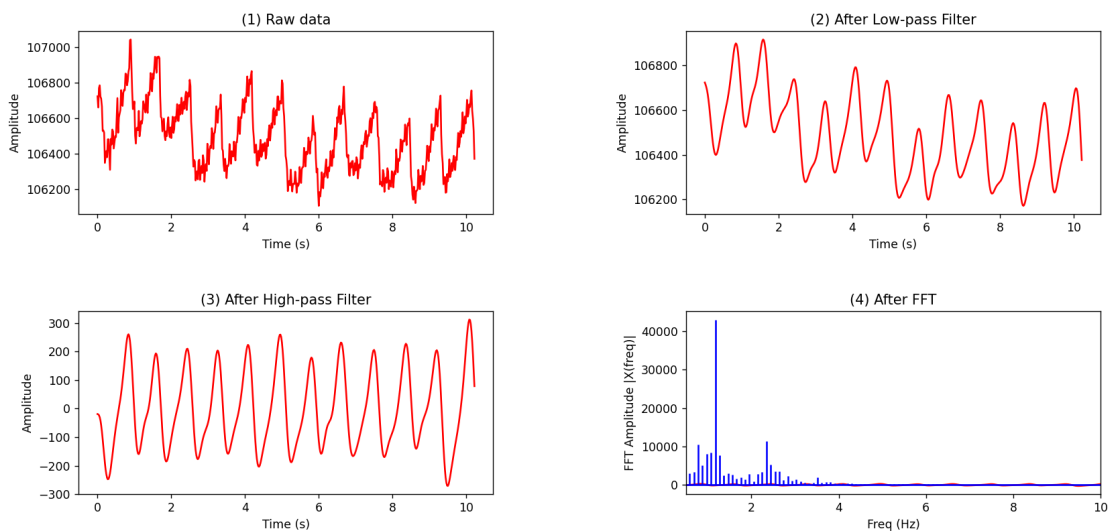


Figure 5.2.1 - Signal Analysis

5.2.2 Fourier Transform

CodeBlue performs a Discrete Fourier Transform in order to have data in the frequency domain. By doing so, CodeBlue builds a baseline for the user and monitors if/when it has entered an unsafe level. High-pass and low-pass Butterworth filters are used to eliminate noise.

5.2.3 Peak Detection

CodeBlue also watches over incoming data and applies a peak-detection algorithm. This algorithm is designed to watch for 2 things: the period of the

peaks, and the height of the peaks. Monitoring the period of the peaks is another way to determine the heart rate of the user, while monitoring the peak heights is used to determine if the user's heart is functioning at normal strength. Due to the algorithm's use of a sliding window to detect a peak, it is not very susceptible to noise that may give a false peak. We also implement high-pass and low-pass filters to eliminate noise.

5.2.4 Confidence

CodeBlue is compatible with two types of sensors: electrocardiogram (ECG) and photoplethysmography (PPG). The confidence of a sensor is determined by the type of data that it is tracking (ECG or PPG) and the location of the sensor on the user's body. This confidence level is then used to determine which readings to trust when monitoring the user's cardiac state.

5.3 Cloud Server Hosting

5.3.1 Motivation

The CA monitoring algorithm is written in Python and needs to be hosted on the cloud in order for CodeBlue to send data and receive cardiac arrest decisions. Hosting the monitoring algorithm on a cloud server satisfies functional requirements F6 and F7.

5.3.2 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (EC2) is a web service that provides compute capacity in the cloud. This allows a server to continuously parse requests with cardiac data from CodeBlue. CodeBlue sends requests through the Internet to the EC2 server's specified IP address with cardiac data attached to its request body.

5.3.3 Node and Express server

Express is a framework for building Representational State Transfer (RESTful) APIs with Node.js. Combining Express, Node.js and EC2 allows us to host the monitoring algorithm such that it is always running and always able to receive requests with cardiac data. The Node.js server accepts Hypertext Transfer Protocol (HTTP) requests with cardiac data with the request JSON body format:

```
{
  "sensors": [
    {
      "data": [v1, v2, v3, ...],
      "location": "wrist"
    }, {
      "data": [...],
      "location": "..."
    }, {
      ...
    }, ...
  ],
  "pastFrames": [v(-1), v(-2), v(-3)],
  "device_id": "abcdef"
}
```

The server will send a response back to CodeBlue with the following response JSON body format:

```
{
  "status": 200,
  "ca": True,
  "heartrate": 60
}
```

Express handles the routing for incoming API requests and is hosted on port 3000 by default. Since we only have one route, we use the root url to parse incoming requests. Namely, requests should be sent to the url:

```
http://<EC2_IP_ADDRESS>:3000/
```

Node.js handles spawning child Python processes which take cardiac data as input, run the monitoring algorithm, and output a True or False response based on whether or not an unhealthy heart rate is detected based on the data.

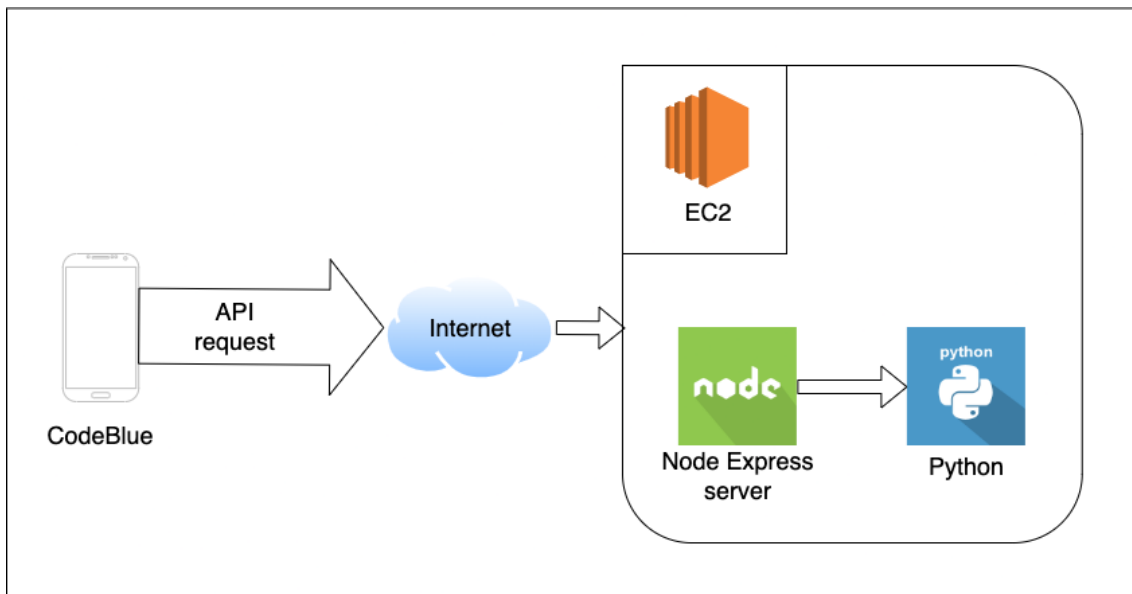


Figure 5.3.3 - High level overview of CA monitoring algorithm

5.3.4 Alternative cloud hosting options

Option	Pros	Cons
Local hosting	<ul style="list-style-type: none">Doesn't require any connection to the internet	<ul style="list-style-type: none">Does not allow for the utilization of Python libraries

	<ul style="list-style-type: none"> • Good for testing purposes 	<ul style="list-style-type: none"> • React Native cannot call localhost directly
Lambda	<ul style="list-style-type: none"> • Cheaper than EC2 • Serverless • Easy setup 	<ul style="list-style-type: none"> • More restrictions on dependencies
EC2	<ul style="list-style-type: none"> • Easy setup • Allows scalability and flexibility for future development • Customizable dependencies 	<ul style="list-style-type: none"> • More expensive

Table 5.3.4 - Comparison of cloud hosting options for hosting monitoring algorithm

Of all the alternative options, EC2 was chosen to host the Python monitoring algorithm due to its ability to scale. This gives the greatest flexibility to our client in the future.

5.4 EMS Communication

5.4.1 Motivation

Implementing EMS communication allows CodeBlue to place a call to 911, alert them of a possible cardiac arrest, and send the location of the user to the EMS dispatcher using enhanced-911 features. This feature also satisfies functional requirement F4.

5.4.2 React Native Immediate Phone Call

In order to make a call to EMS, we are using the [React Native Immediate Phone Call](#) Library. This library is chosen because it does not require any user input or confirmation before placing a call. The library implements a fast and reliable method for placing calls which is necessary for an emergency situation such as a CA. The alternative options are shown in Table 5.4.3 below.

5.4.3 React Native Phone Call Options

Option	Pros	Cons
React Native Immediate Phone Call	<ul style="list-style-type: none">• Able to place a call without any user action• Simple implementation steps	<ul style="list-style-type: none">• Requires additional configuration in Java
React Native Phone Call	<ul style="list-style-type: none">• Simple implementation steps• Uses native linking to place the call	<ul style="list-style-type: none">• Requires the user to press a prompt in order to place a call, which is not possible during a CA
Twilio API	<ul style="list-style-type: none">• Simple implementation steps• Calls handled externally by a 3rd party• Programmable voice	<ul style="list-style-type: none">• Expensive to use for emergency procedures• 3rd party privacy concerns

Table 5.4.3 - Comparison of React Native phone call options

Although the other options will place a call, the RN-immediate-phone call package is the option we chose because it is easy to use and does not require a user prompt to place a call.

5.4.4 Enhanced 911

In order to send the location of the user to the EMS dispatcher, we are using a feature called enhanced-911. [Enhanced-911](#) is a free service available through every Canadian mobile carrier, nationwide. This service uses the phone's GPS data to locate the user and automatically sends that location to EMS dispatch once a 911 call is placed.

5.5 Local Data Storage

5.5.1 Motivation

Implementing local data storage allows CodeBlue to persist app settings, store historic cardiac data (past 24 hours), and save user health data (SG5). If addressed, SG5 enables CodeBlue to send potentially lifesaving information to EMS in the event of a CA.

5.5.2 React Native MMKV

Local data storage is implemented using the [React Native MMKV](#) library and exposed through a custom hook (useLocalStorage) that serves as a wrapper for key Create-Read-Update-Delete (CRUD) methods. A comparison of considered local storage libraries is discussed in Table 5.5.3.

The MMKV library provides fast, synchronous reads and writes to local storage. This property is valuable because cardiac data is only saved in local storage. Due to privacy concerns, we are unable to store this data on the cloud. Therefore, having instant CRUD operations ensures that all cardiac information

is saved and calculations are performed on accurate data. Furthermore, MMKV provides encryption for stored entries. This is important to CodeBlue because cardiac data is sensitive in nature. By encrypting all locally stored information, we protect users from privacy breaches.

5.5.3 Comparison of Local Storage Libraries

Option	Pros	Cons
React Native MMKV	<ul style="list-style-type: none">● Fast, synchronous read and write operations● Provides encryption support for security● Includes value-change listeners● Lightweight interface and easy to use	<ul style="list-style-type: none">● Increases app build time due to compilation of C++ modules
React Native Async Storage	<ul style="list-style-type: none">● Most popular local storage library● Trusted by the developer community● Lightweight interface and easy to use	<ul style="list-style-type: none">● Total storage size capped to 6MB by default● Read/write operations are async (potential to fail silently)● Only supports strings and serialized objects● Lacks security

React Native Encrypted Storage	<ul style="list-style-type: none"> • Secure version of React Native Async Storage • Provides encryption support for security 	<ul style="list-style-type: none"> • Same as above, but security is no longer a concern
React Native Realm	<ul style="list-style-type: none"> • Built by MongoDB (reputable source) • Can define custom schemas • Can register a change listener to react to changes 	<ul style="list-style-type: none"> • High set-up overhead • CodeBlue would only need to use a small subset of all possible features

Table 5.5.3 - Comparison of local storage libraries

Of all the prospective options, React Native MMKV was chosen because it is lightweight and provides fast, synchronous operations with secure encryption.

5.6 Push Notification

5.6.1 Motivation

Push notifications are real time messages sent from outside the device, for example from a server, that triggers an application of the device. These will be used to alert CodeBlue users when a possible CA is detected. Implementing push notifications satisfies functional requirement F6.

5.6.2 Firebase Cloud Messaging

All of the push notification services compared in section 5.6.4 are free and have sufficient features that support CodeBlue's requirements. Firebase Cloud

Messaging (FCM) was selected for our purposes because it is the most widely used.

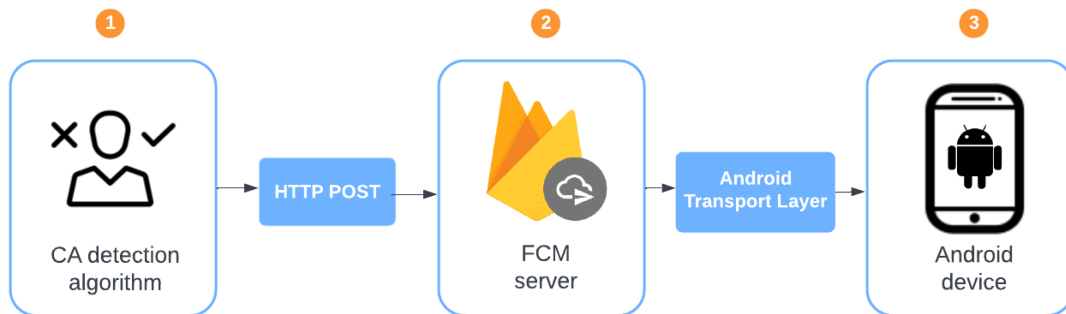


Figure 5.6.2 - High-level overview of push notification architecture

FCM supports triggering instant push notifications to Android devices through an HTTP POST. This will be added as a step in the CA monitoring algorithm when a positive CA decision is detected.

There are three types of notifications:

1. Foreground - when the application is open and in view
2. Background - when the application is minimized
3. Quit - when the device is locked or the application is not active or running

CodeBlue tracks users' cardiovascular data while the app is running in the background and will use background and quit notifications to alert users of positive CA monitoring decisions. Both types of notifications are automatically handled by the system and are displayed without waking up the app.

FireBase messages have three payloads types:

1. Notification payload - have fixed object key values (title and body)

2. Data payload - contains custom key value pairs and allows sending custom data along with notification
3. Both notification and data payload

Data messages have to be handled by the Android app [6]. Since CodeBlue only needs to alert users and does not require custom messages to be sent through the push notification, only notification payload will be implemented.

FCM messages are sent through the Android Transport Layer which uses point-to-point encryption. Added end-to-end encryption is not needed for CodeBlue, since notifications do not contain any sensitive data [5].

5.6.3 Triggering Push Notifications

FCM allows sending notifications through API calls using Firebase Admin SDK for Python. Each mobile device or simulator generates a unique device token id which FCM uses to communicate. The device token id is saved to local storage and attached to every EPI request to the monitoring algorithm server, so that when the monitoring algorithm detects a cardiac arrest, FCM pushes a notification to the device running CodeBlue.

5.6.4 Receiving FCM notifications

There are two cases for CodeBlue receiving push notifications from FCM:

1. CodeBlue is open and running in the foreground
2. CodeBlue is running in the background or phone screen is shut off

In both cases, immediately after receiving the push notification, the EP procedure starts. Namely, the 30 second delay timer starts counting down and the phone begins vibrating.

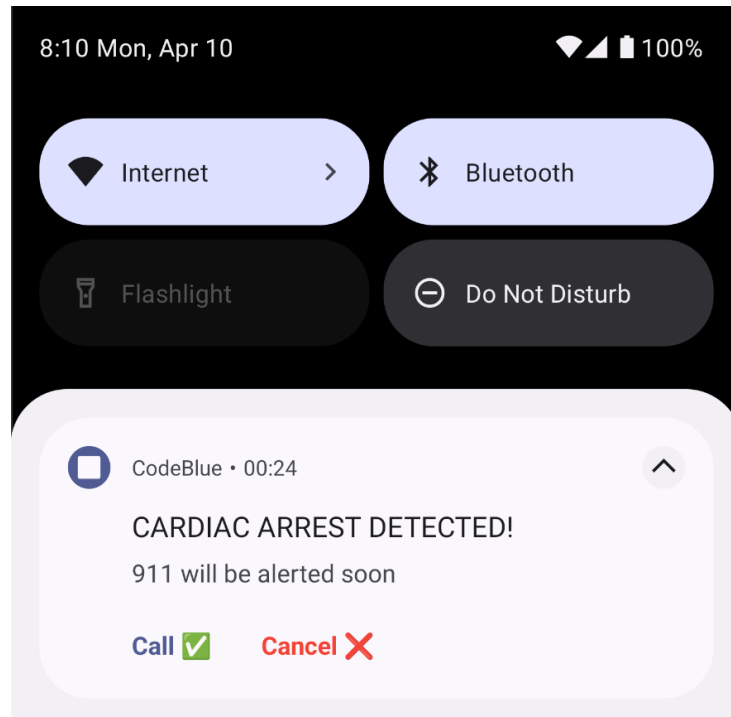


Figure 5.6.4.1 - Cardiac arrest background notification

For background notifications, during the first 30 seconds, users can click the Call button on the notification to immediately open CodeBlue and place the call, Cancel to open CodeBlue and terminate the EP procedure, or click the notification body to launch CodeBlue. Upon launching CodeBlue, the user is greeted by the EP screen with the timer displayed.

For foreground notifications, users are immediately taken to the EP screen where the 30 delay timer countdown is displayed and the user can immediately place or cancel the EMS call.

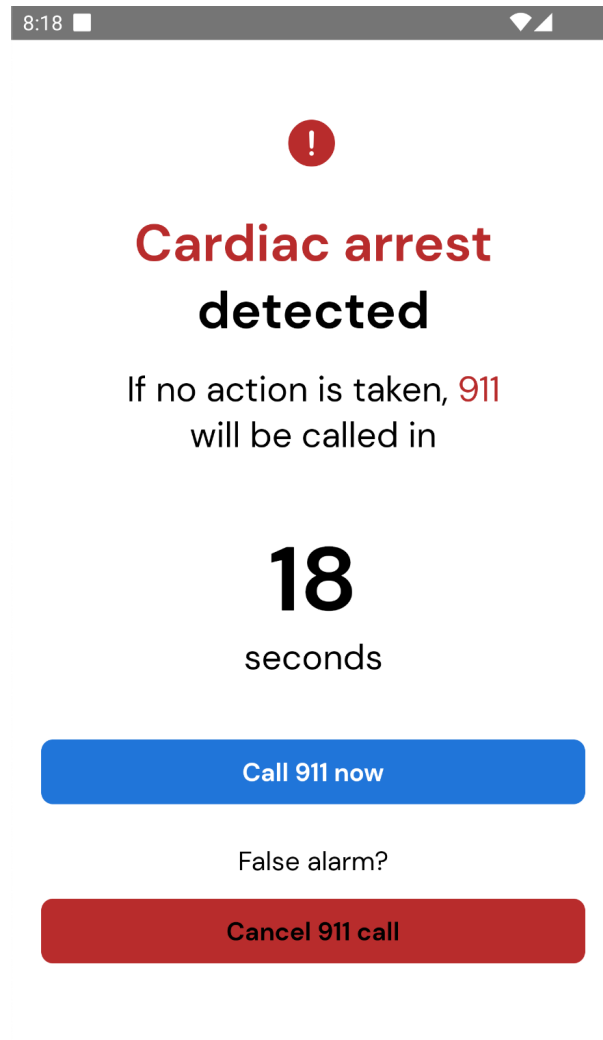


Figure 5.6.4.2 - EP screen with running countdown

5.6.5 Alternative Push Notification Services

Option	Features
FCM	<ul style="list-style-type: none">• Free tier available• Comprehensive, end-to-end development platform• Easy integration• Developer console• RESTful API

	<ul style="list-style-type: none"> ● Firebase Admin SDK allows easy integration from Python
OneSignal	<ul style="list-style-type: none"> ● Free tier available ● Instant delivery ● RESTful server API
Amazon SNS	<ul style="list-style-type: none"> ● Free tier available ● Cost-effective ● Customizable
Wonderpush	<ul style="list-style-type: none"> ● Free tier available ● Fast ● GDPR compliant ● Provides RESTful API ● Online dashboard

Table 5.6.5 - Comparison of push notification services

Of all the alternative options, FCM was chosen because it integrates most seamlessly with Python and is the most common notification library for React Native.

5.7 Background Processing

5.7.1 Motivation

The monitoring algorithm should continue running even if CodeBlue is backgrounded or the phone is locked. CodeBlue needs a background processing mechanism to monitor the user and alert EMS if a CA occurs when

the app is closed. Background processing satisfies functional requirements F5 and F6.

5.7.2 Notifee

CodeBlue uses the Notifee library to implement background processing. To keep the app “alive” while backgrounded, Notifee leverages the native Android `ForegroundService` implementation and displays a sticky, low priority notification in the phone’s notification center. All of the considered options are discussed in Table 5.7.3.

A `ForegroundService` is a long running task that executes when the app is alive and backgrounded. But, only one `ForegroundService` can be running at any given time for an app. In response to this limitation, CodeBlue implements task switching between three possible states: monitor, phone call, and idle.

In the monitor state, CodeBlue receives cardiac data and sends it to the algorithm at a constant interval. The algorithm returns a response that signals if the user is okay or in danger. If a CA is detected, the app moves to the phone call state and places the EMS call. The app is in an idle state if it has been killed, or if it is not receiving data.

5.7.3 Comparison of Background Processing Approaches

Option	Pros	Cons
Notifee	<ul style="list-style-type: none">• No minimum time delay between repeated tasks (such as sending data to the CA monitoring algorithm)• Out-of-the-box support for notification styling and handling• No native Java code required	<ul style="list-style-type: none">• Introduces a project dependency on the Notifee library
HeadlessJS: PeriodicWorkRequest	<ul style="list-style-type: none">• App performance will not be negatively affected by background processing due to long delay between tasks	<ul style="list-style-type: none">• Unable to execute a repeated task faster than every 15 minutes (not sufficient for cardiac monitoring)• Requires writing lots of native Java code, which is hard to test

HeadlessJS: custom native module	<ul style="list-style-type: none"> • No minimum time delay between repeated tasks (such as sending data to the CA monitoring algorithm) 	<ul style="list-style-type: none"> • Requires writing lots of native Java code, which is hard to test
----------------------------------	--	--

Figure 5.7.3 - Comparison of background processing implementations

Of the options above, Notifee was chosen because it does not require writing any native Java code and supports repeated tasks with no minimum time delay.

5.8 Bluetooth Low Energy (BLE)

5.8.1 Motivation

In order for CodeBlue to successfully communicate and receive the data from the wearable ECG and PPG sensors, a bluetooth library is needed. The bluetooth client allows CodeBlue to forward the data received from the bluetooth module to the monitoring algorithm.

5.8.2 React-Native-Ble-Plx

To implement the bluetooth client, CodeBlue uses the React-Native-Ble-Plx library. React-Native-Ble-Plx has an easy to use BLE manager which provides various services and methods.

To connect to a wearable device, the BLE manager first scans for nearby peripheral devices. Once a device is discovered, which is in a disconnected state by default, the manager is able to initiate a connection with the device. Upon a successful connection, the device services become available to

CodeBlue. Upon the discovery of services, CodeBlue is able to find the characteristic which notifies the cardiac data. With this information, CodeBlue is able to receive the cardiac data from the wearable device and send it to the CA Monitoring Algorithm.

5.8.3 Alternative Bluetooth Low Energy Libraries

The only other alternative to React-Native-Ble-Plx that we considered is React-Native-Ble-Manager. Although both of these libraries provide very similar features and are essentially identical in terms of functionality, CodeBlue utilizes React-Native-Ble-Plx because of its larger community support and lower number of unresolved issues/bugs.

6. Artifacts

6.1 UI Components

The [user interface design](#) for CodeBlue is shown in the figures below. The design is intended to be simple and easy to use for all potential users.

6.1.1 Onboarding

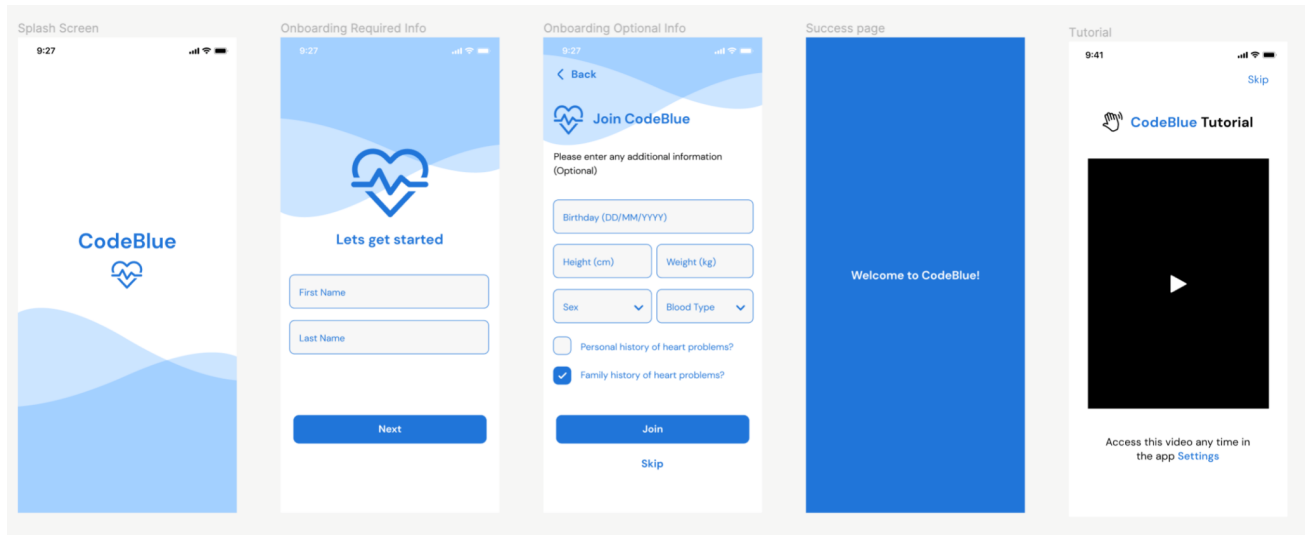


Figure 6.1.1 - Onboarding flow

6.1.2 Home screen

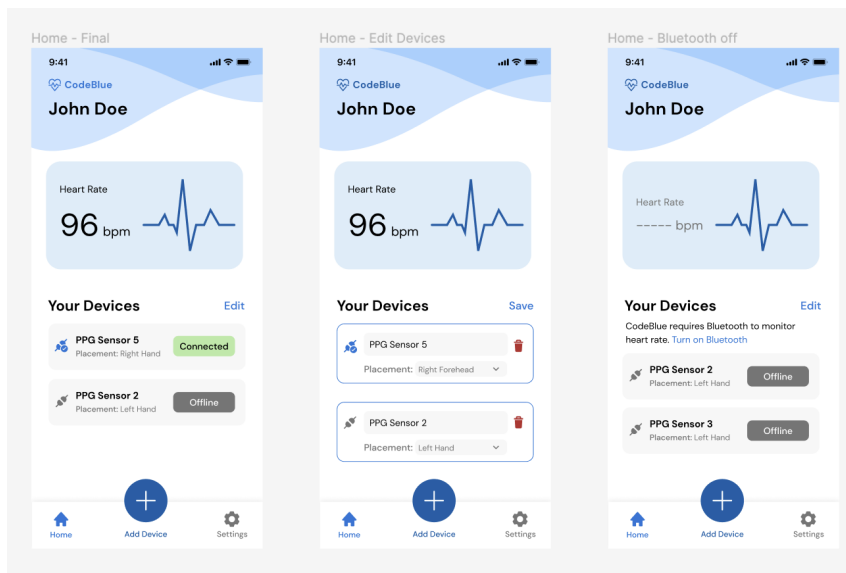


Figure 6.1.2 - Home screen

6.1.3 Add New Sensor Device

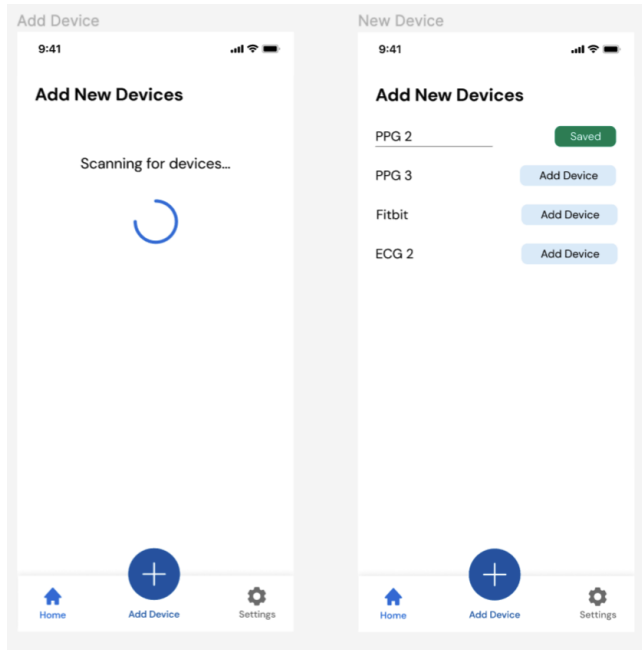


Figure 6.1.3 - Add New Device flow

6.1.4 Settings

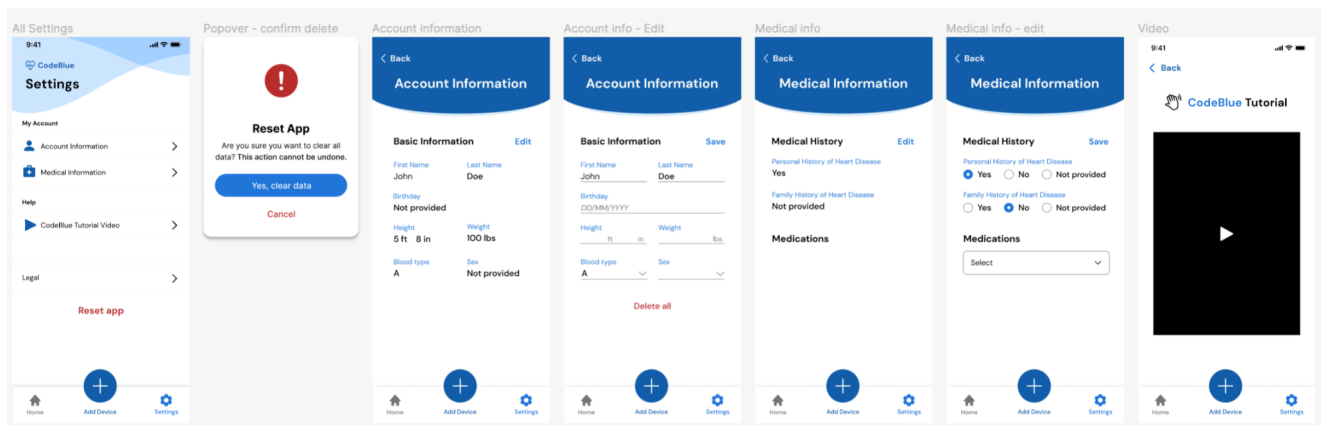


Figure 6.1.4 - Settings flow

6.1.5 Emergency Protocol

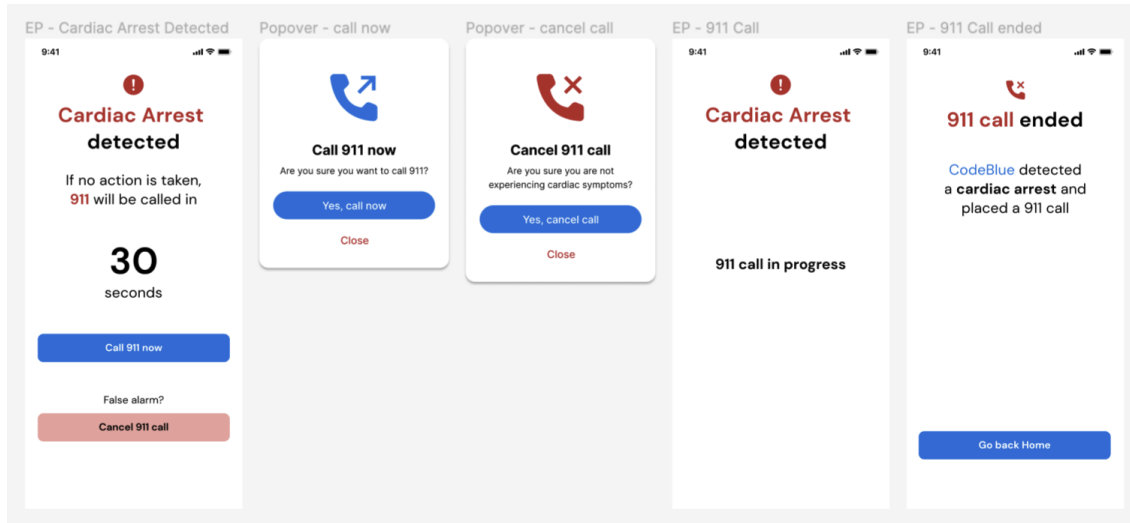


Figure 6.1.5 - Emergency Protocol flow

The figures shown above demonstrate the entire user interface of the CodeBlue app.

6.2 Repository Structure

```
CodeBlue
├── android/
├── codeblue-server/
├── src/app/
│   ├── assets/
│   ├── backgroundMode/
│   ├── ble/
│   ├── components/
│   ├── constants/
│   ├── emsCall/
│   ├── localStorage/
│   ├── navigation/
│   ├── screens/
│   └── utils/
```

A diagram showing the repository structure of CodeBlue. It is a tree view with a vertical line on the left and horizontal lines branching to the right. The items listed are: storybook/, App.tsx, index.js, README.md, and ...

```
graph LR; Root[ ] --- storybook[storybook/]; Root --- App[App.tsx]; Root --- index[index.js]; Root --- README[README.md]; Root --- dots[...];
```

Figure 6.2 - CodeBlue repository structure

6.3 Source Code

Github repository: <https://github.com/seangarveyubc/codeblue>

6.4 User Guide

Can be found at

https://github.com/seangarveyubc/codeblue/arc/app/assets/JY041_CanSAVE_Video_V1.mp4

7. References

1. "Differences Between Bluetooth and Wifi," *TechDifferences*, 13-July-2019. [Online]. Available: <https://techdifferences.com/difference-between-bluetooth-and-wifi.html> [Accessed: 26-Oct-2022].
2. "Classic Bluetooth vs. Bluetooth Low Energy (BLE)," *IoT Lab - Tertium Cloud Blog*, 19-Aug-2020. [Online]. Available: <https://iotlab.tertiumcloud.com/2020/08/19/classic-bluetooth-vs-bluetooth-low-energy-ble/> [Accessed: 26-Oct-2022].
3. A. Chugh, "Android MVVM Design Pattern," *DigitalOcean*, 03-Aug-2022. [Online]. Available: <https://www.digitalocean.com/community/tutorials/android-mvvm-design-pattern>. [Accessed: 26-Nov-2022].
4. Successive Technologies, "Firebase Push Notifications in React Native," *Medium*, 10-Mar-2022. [Online]. Available: <https://medium.com/successivetech/firebase-push-notification-in-react-native-57973ee7c11d#:~:text=Type%20of%20notifications%20Firebase%20Cloud,will%20display%20a%20specific%20message>. [Accessed: 30-Oct-2022].
5. Google, "About FCM Messages | Firebase Cloud Messaging." Google, 17-Nov-2022. [Online]. Available: <https://firebase.google.com/docs/cloud-messaging/concept-options> [Accessed: 29-Nov-2022].
6. Bhardwaj, Varun. "Sending Push Notifications by Using Firebase Cloud Messaging." *Medium, Nybles*, 18 July 2020, [Online]. Available: <https://medium.com/nybles/sending-push-notifications-by-using-firebase-cloud-messaging-249aa34f4f4c>. [Accessed: 29-Nov-2022]

8. Appendix

A. Contributions

Section	Major Content	Minor Content	Author	Reviewer
1. Design Overview		All	GT, AR	All
2. Technological Decisions		All	GT	All
3. High-Level Design		All	EL	All
4. User Interface (UI)		All	EL	All
5.1 Data Processing		All	AR	All
5.2 CA Monitoring Algorithm		All	SG	All
5.3 Cloud Hosting		All	SW	All
5.4 EMS Communication		All	GT	All
5.5 Local Data Storage		All	EL	All
5.6 Notification System		All	SW	All
5.7 Background Processing		All	EL	All
5.8 Bluetooth Low Energy		All	AR	All
6. Artifacts		All	All	All
7. References		All	All	All