# Health Application: Development Documentation

# Goal: This document will describe the details of the design, architecture and implementation of the application.

# Please refer to the repository for installation, a setup readme and the codebase: [GitHub - RamazTs/HealthApplication](https://github.com/RamazTs/HealthApplication)

# Requirements and design overview

## Requirements

### The health app needs to be cross-platform to facilitate both iOS and Android phones to collect data on Raynaud's disease outside clinical settings. In addition, the app shall be able to use voice for survey responses, automatically gather ambient data (weather, location, time, and condition keywords), and capture blood flow through images. React Native was chosen due to its user interface flexibility, performance with native components, lightweight application size, strong developer support, and data security considerations.

## Design Overview

### Our app aligns with patient needs and medical care by enabling real-time symptom documentation through voice recognition and hands-free operation, ensuring privacy with local data storage and offering cloud upload for machine learning analysis. This aids in personalized treatment and research on Raynaud's disease. Initiated by symptom flare-ups, the app captures essential data like geolocation, weather, and health statistics, which can be shared with healthcare professionals for better clinical evaluation. The insights from cloud-analyzed data help tailor treatments and advance understanding of Raynaud's disease through patient-centric research and symptom management. Our focus on modularity, scalability, and user-centered design supports the app's innovative features and the management of Raynaud's disease

# Higher level Architecture

## High level Architecture

### The tailored health application features a clutter-free, user-friendly interface designed to minimize decision-making fatigue. It includes three primary navigation options: (1) a questionnaire for health data collection, (2) a connection to wearable and Bluetooth devices for tracking health metrics in real-time, and (3) a history review for analyzing and sharing data with healthcare providers. These features facilitate efficient data collection, improve patient-clinician communication, and provide a holistic view of the patient's health, enhancing disease management and patient engagement. This design approach prioritizes the care and autonomy of individuals managing Raynaud's disease.

## Questionnaire

### A diagram of a questionthe design of the questionnaire feature, which allows patients to easily log symptoms. Users are led through clear instructions and questions provided by healthcare professionals, presented linearly for a user-focused experience. The feature includes advanced voice recognition for hands-free operation, enabling users to answer questions aloud and navigate using voice commands, enhancing accessibility. The app allows revisiting and editing answers and includes a 'Cancel Questionnaire' option for flexibility. Upon completion, a summary page combines responses with data like geolocation, weather, timestamp, and wearable device metrics for a comprehensive health overview. Users can save their data to the local history module or restart the questionnaire, emphasizing user privacy and empowerment in disease management. This intuitive flow offers a unique symptom-tracking and management experience.

## Wearable Integration

A diagram of a computer flow

Description automatically generated with medium confidence

### Wearable integration: The "Connect to Wearables" feature, integrates data from Fitbit, Apple Watch, and Samsung Watch to enhance patient data collection. This process, starts with device selection and moves through a secure authorization process, allowing users to control which health metrics are shared. Fitbit integration involves a web browser session for login and data-sharing preferences. Apple Watch users go through HealthKit for authentication and permission settings for specific health metrics, ensuring data privacy. Samsung Watches uses a consent-driven process for Android users, allowing them to specify accessible data types through device settings. Once permissions are granted, our app gains access to real-time physiological data, automating the collection and enriching patient profiles. This streamlined, privacy-focused process empowers patients in managing Raynaud's syndrome, offering an intuitive and comprehensive approach to personal health management and advancing personalized medicine.

## History viewA green screen with white rectangles Description automatically generated

### This figure highlights the history feature of our health app. This feature provides a centralized location for users to access and review their health data, displaying entries in a prioritized, timestamped list for easy navigation. Voice-activated commands enhance user interaction, enabling quick retrieval of detailed entry summaries, including data from questionnaires, geolocation, and other metrics. Additionally, the app allows users to package these records into a comprehensive file for easy sharing with healthcare providers or integration into health records systems. This streamlined process empowers patients to actively participate in their health management, bridging the gap between personal data collection and clinical analysis, and ensuring their experiences contribute to valuable clinical insights.

## Data Flow:A diagram of a computer Description automatically generated

### The data flow initiates with the patient completing the questionnaire within the application interface. After saving, the React Native asynchronous storage module, a secure facility for non-volatile data retention, is called that stores the data collected from not only the questions are responses but environmental parameters such as geolocation and weather data, and if a wearable device is connected, a pull request via its respective API fetches health metrics. These disparate data points are then conjoined into a singular Javascript object notation (JSON) file, representing an individual record within the asynchronous storage module. As seen in the figure belour application follows a flow of sensitive patient data with an emphasis on security and privacy.

# Lower Level Architecture(Found in code base in left panel)

## Views

### Our app's architectural design focuses on creating an intuitive interface for maintaining modular and maintainable code. Views are critical for creating distinct features, flexible containers for crafting and styling UI elements, reducing cognitive load by simplifying navigation and minimizing user decisions. The hierarchical structure, starting from the home view and extending to feature-specific child views, facilitates ease of use and promotes automation where possible.

### Voice navigation is enhanced by labeling UI components in React Native, improving voice command recognition accuracy. This allows users to navigate efficiently using voice commands, matching inputs with UI elements for a seamless experience.

### The modular design ensures the app is adaptable and future-proof, simplifying updates and feature additions. This approach minimizes development overhead and keeps the app responsive to user needs and technological advancements in managing Raynaud's syndrome.

## Components

### Components are key to building a user-friendly interface, enabling code reusability and encapsulation. These components ensure consistent visual and functional elements across the app and are especially crucial in the questionnaire feature to accommodate various question types and response formats, such as rating scales or selection options. This approach eliminates code redundancy, maintains uniformity, and enhances navigation ease, significantly impacting user engagement and data accuracy.

### Components streamline the development for both iOS and Android, offering platform-specific customization without compromising the user experience. Furthermore, they facilitate future enhancements, such as incorporating new question types or adapting to evolving medical insights, by allowing easy integration of new modules. This strategic use of components enhances the app's usability, maintainability, and scalability, ensuring it remains adaptable and continues to serve healthcare providers effectively.

## Services

### Services are crucial for handling data lifecycle and state, acting as an abstraction layer for tasks like fetching data, persisting information, and integrating with technologies like Text-to-Speech and Speech-to-Text. They ensure reliable data storage during the questionnaire phase and process data into actionable insights, significantly enhancing the user experience and data integrity.

### The modular design facilitated by services promotes code reusability and streamlined development, allowing for efficient logic importation across the app. This backbone of our data strategy enables a robust, user-centric tool for managing Raynaud's syndrome. Emphasizing our commitment to user needs, the app's architecture is built to quickly changing demands, with a focus on reliable speech processing for effortless navigation and interaction.

# Logic for creating more reliable natural language processing techniques

## Intial process

### Our initial strategy entailed recording a full response and parsing it into an array of words and phrases, represented as objects. For instance, the phrase "not at all" would be segmented into {not}, {at}, {all}, {not at}, {not at all}. This method aimed to match user words to anticipated responses. Despite its functional promise, this approach did not come without many limitations, including the inability to capture all permutations and a lack of real-time efficiency which were both critical for optimizing user experience.

## Iterations

### Recognizing the necessity for seamless user interaction, we revised the system so that users could respond with simple words such as "one" or "two." To achieve this, we assigned each spoken numeral to its corresponding integer, facilitating a match with the expected responses. However, this simplistic approach had trouble recognizing brief, numeric responses, which led to inaccuracies in voice-to-text translation. Conversely, the system demonstrated proficiency with longer numerical responses like "three hundred fifty-four." Almost always picking it up correctly and at the very least almost always picking up the second half of the word such as “fifty-four”.

### Investigation revealed that the root of this problem was actually not completely in our control, rather it resided in the natural language processing capabilities of the speech processing engine used by React Native. To get around these limitations, we innovated an instructional interface prompting users to articulate responses as "choice 1" or "choice 2." The integration of regular expressions (regex) allowed for pattern recognition over lingual matching, significantly enhancing the voice recognition module's precision.

### Code is found in the code base in questionnaire tab under thefunction: handle respoones

# React native, Android and iOS particulars(detailed installation instructions can be found on the repository linked here: [GitHub - RamazTs/HealthApplication](https://github.com/RamazTs/HealthApplication))

## React native: [Introduction · React Native](https://reactnative.dev/docs/getting-started)

### React Native is an open source framework for building Android and iOS applications using React and the app platform’s native capabilities. With React Native, you use JavaScript to access your platform’s APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code.

### In Android and iOS development, a **view** is the basic building block of UI: a small rectangular element on the screen which can be used to display text, images, or respond to user input. Even the smallest visual elements of an app, like a line of text or a button, are kinds of views. Some kinds of views can contain other views.

### In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. With React Native, you can invoke these views with JavaScript using React components. At runtime, React Native creates the corresponding Android and iOS views for those components. Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. We call these platform-backed components **Native Components.**

### Components: React Native has many Core Components for everything from controls to activity indicators. [Core Components and APIs · React Native](https://reactnative.dev/docs/components-and-apis)

### Paltorm Specific: [Platform-Specific Code · React Native](https://reactnative.dev/docs/platform-specific-code)

## IOS

### Pod Dependencies: The Podfile is a specification that describes the dependencies of the targets of one or more Xcode projects. The file should simply be named Podfile. the Podfile in a React Native project for iOS is typically generated automatically when you create your project. It can also be manually initialized using CocoaPods commands. The file is then maintained and updated through the lifecycle of your project as you add or update dependencies.

### Initializing the Project When you create a React Native project using the react-native init command, it creates an iOS folder within your project directory. This folder contains all the necessary iOS-specific files, including a Podfile if you are using React Native 0.60 or higher

### Podfile Creation: This is automatically created, you can navigate to the ios directory of your React Native project and run pod init. This command creates a Podfile in your project if one doesn’t already exist.

### Specifying Dependencies: Dependencies for your React Native project, such as third-party libraries or plugins, are specified in the package.json file. However, for these dependencies to be integrated into your iOS project, they must be reflected in the Podfile. Many React Native packages include podspecs (Podfile specifications) that allow them to be automatically linked to your iOS project when you run pod install.

### Automatic Linking: Since React Native 0.60, auto-linking of native modules is supported. When you install a new npm package that contains native iOS code, React Native's CLI automatically adds the necessary pods to your Podfile upon running npx react-native link or during the build process.

### Running pod install: After adding new dependencies or updating your project, you should navigate to the ios directory in your terminal and run pod install. This command analyzes your Podfile, resolves the dependencies specified in it, and updates the Podfile.lock file and your Xcode workspace accordingly.

### Podfile.lock: This file is generated after the first successful run of pod install. It locks down the versions of the pods that are being used in your project. This ensures that all team members and your CI/CD environment use the exact same versions of the dependencies.

## Android

### For android simulator: <https://reactnative.dev/docs/running-on-device>

### For Android native Modules: <https://reactnative.dev/docs/native-modules-android>

### Android Manifest/Studio – this is used to control packages that are natively used by android studio. For the simulator you have to go into android studio -> device manager(AVD)-> create device and then choose any device that is suitable.

### For assembling an apk for android:

Follow these steps:

1. Go to the terminal in your project and run:

react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/res

1. Using the terminal run:  
   cd android
2. Using terminal run:
3. ./gradlew assembleDebug
4. The APK will be in your file path as follows:
5. yourProject/android/app/build/outputs/apk/debug/app-debug.apk