Software Design Description for MEDRhythms Mobile App Version 1.0

Prepared by

Name	Date Signed
Ifeanyi Ineh	28/03/2025
Yiran Zhao	28/03/2025
Yoga Srinivas Kassireddy	28/03/2025
Chaoyi Jiang	28/03/2025

Group Name: Fantastic Four

Instructor: Dr. Gary Cantrell

Course: Foundations of Software Engineering

Teaching Assistant: Sam Morris

March 28, 2025

Revision History

Version	Date	Description	Primary Author
1.0	March 28, 2025	Initial Draft	Chaoyi Jiang

Contents

1 Introduction

MedRhythms is a company that develops treatment solutions for patients with neurological diseases and stroke survivors. We built an application for patients with neurological diseases, Parkinson's disease, stroke survivors and their caregivers to monitor and manage their walking performance through step tracking. This includes the ability to play music in line with their walking rhythm, thereby using the power of technology and music to have a positive impact on the lives of patients with neurological injuries and diseases.

The stakeholders involved include the development team, project managers, testers, and users (patients with neurological diseases, Parkinson's disease, stroke survivors and their caregivers).

2 Requirements

2.1 User Authentication

The ability to allow the user to log into the application via IMEI (International Mobile Equipment Identity) number. The Authentication data shall be stored in Firebase Firestore.The application shall automatically retrieve IMEI for login after initial setup User sessions shall timeout after 30 minutes of inactivity (based on SRS 5.3).

Priority: Essential

2.2 Step Tracking

Users can create walking sessions where stepscaloriesdistanceaverage speed are tracked via gyroscope(phone) sensors.The data shall be stored locally when offline and synchronized with Firebase when internet connection is available.Users shall receive notifications after completing 30 minutes of walking.The system shall support pausing and resuming tracking sessions.

Priority: Essential

2.3 Music Integration

Allows the music playback ability, through 3rd party services like Spotify.Music playback shall synchronize with walking rhythm when possible.

Users shall be able to:

- Connect their existing music accounts
- Select custom playlists
- Control music playback during workout sessions

The system shall accommodate user music preferences within the application

Priority: Medium (Bonus feature)

3 Data Design

MEDRhythms' data storage is based on two parts: local storage and Firebase cloud storage.

MEDRhythms' step tracking data flow is as follows:

User logs in using IMEI - User starts walking to trigger data collection - Running is detected through sensors - Step is been tracked - The data will be synchronized in Firebase - Local storage data backup.

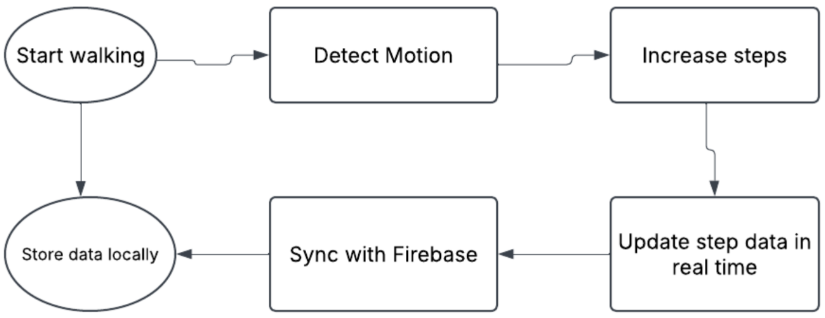

Figure 1: MEDRhythms' Step Tracking Data Flow

4 Architecture

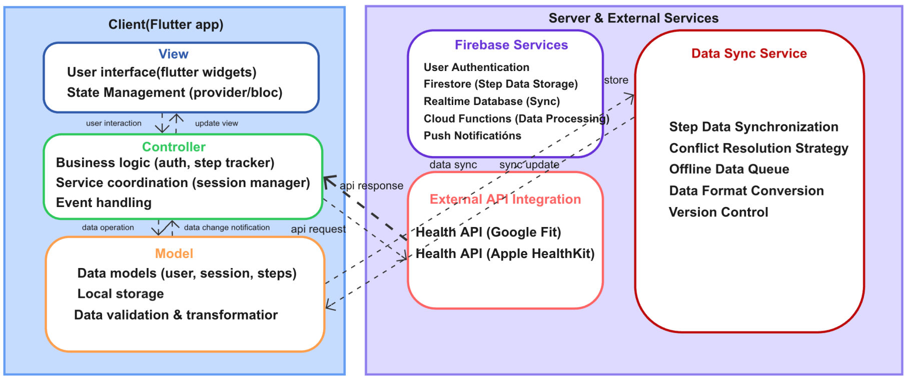

Figure 2: MEDRhythms Applicaiton Architecture

Key Data Flows:

- User Interaction Path:

  User Actions → View → Controller → Model → Local Storage

- Data Synchronization Path:

  Model Data Changes → Sync Service → Firebase → Sync to Other Devices

5 Component Design

This design is just an initial framework, and there may be adjustments in the later implementation. A comparative analysis will be conducted at the end of the semester to evaluate how well the design matches the final product.

5.1 System component architecture

The MEDRhythms application has a modular architecture consisting of six main components.

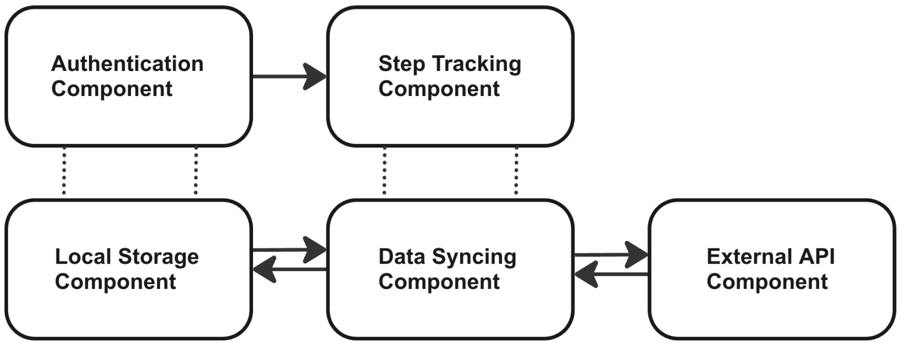

Figure 3: MEDRhythms Applicaiton Architecture

- Authentication Component(according to SRS 4.1)
- Step Tracking Component((according to SRS `4.2\5.1\5.2\5.3`)
- Local Storage Component
- Data Syncing Component(according to SRS 2.5.1)
- External API Component(according to SRS 3.5.2)

5.2 Interactions between MEDRhythms components

The process: the user logs in through the Authentication Component, and then the Authentication Component interacts with Local Storage to obtain/save credentials.

Functional interaction (one-way):

- Authentication Component → Step Tracking Component, Authentication Component provides user identity to Step Tracking Component
- Step Tracking Component((according to SRS `4.2\5.1\5.2\5.3`)
- Step Tracking Component → Data Syncing Component, Step Tracking Component passes step data to Data Syncing Component
- Data Syncing Component → External API Component, Data Syncing Component synchronizes data to the cloud with External API Component

Functional interaction (two-way):

- Step Tracking Music Integration, mutual adjustment of step frequency and music rhythm
- Local Storage Data Syncing, data caching and updating

5.3 Data flow between MEDRhythms components

user authentication $\rightarrow$ step record $\rightarrow$ local storage $\rightarrow$ cloud synchronization.

6 Object-Oriented Design

6.1 OOD Diagram

6.2 Class Description

User (according to SRS 4.1)

- Responsibilities: Represents the user of the application
- Attributes:
    1. IMEI: String - unique identifier of the user
    2. sessions: List<Session> - all sessions of the user
- Session (according to SRS 4.2/5.1)
    1. Responsibilities: Represents a walking session
    2. Attributes:
        - startTime: Timestamp - Session start time
        - endTime: Timestamp - Session end time
        - totalSteps: Integer - Total number of steps in the session
        - isActive: Boolean - Whether the session is active
        - stepRecords: List<StepDataRecord> - List of step records
- StepDataRecord (according to SRS 4.2)
    1. Responsibilities: Stores a single step count record
    2. Attributes:
        (a) timestamp(date object): Timestamp - Recording time
        (b) stepCount: Integer - Current step count
        (c) distance: Float - Walking distance (meters)
        (d) speed: Float - Walking speed (meters/second)
- StepTracker (according to SRS 4.2)
    1. Responsibilities: Manage step tracking logic
    2. Methods:
        (a) startTracking(): Boolean - Start tracking
        (b) stopTracking(): Boolean - Stop tracking
        (c) detectMotion(): Boolean - Detect motion
        (d) countSteps(): Integer - Count steps
- SensorManager (according to SRS 2.5.1)
    1. Responsibilities: Define sensor interaction interface
    2. Method:
        - getSensorData(): SensorData - Get sensor data(get walking data)

- AndroidSensors (cell phone sensors)
    1. Responsibilities: Sensor implementation for Android platform
    2. Method:
        (a) getAccelerometerData(): AccelerometerData - Get accelerometer data
        (b) getGyroscopeData(): GyroscopeData - Get gyroscope data
- MusicService (according to SRS 4.3)
    1. Responsibilities: Define music service interface
    2. Method:
        (a) connectAPI(): Boolean - connect to API
        (b) playMusic(playlist: String): Boolean - play music
        (c) syncWithSteps(frequency: Float): Boolean - sync music with step frequency
- SpotifyService (according to SRS 4.3 Music Integration, 5.4 UC4-Music Integration)
    1. Responsibilities: Spotify service implementation
    2. Methods:
        (a) authenticate(): AuthToken - Authentication
        (b) getPlaylists(): List<Playlist> - Get playlists
        (c) playSong(songId: String): Boolean - Play songs

7 Interface Design

The user interface for the MEDRhythms Mobile App is designed with simplicity and accessibility in mind, targeting users with neurological conditions.

The UI flow follows a logical progression as depicted in Figure 1, which includes:

- Authentication screen (IMEI-based login)
- Walking session initiation and monitoring screens
- Session pause/resume functionality
- Session completion and analysis screens
- Historical data visualization and analytics
- Music playback integration interface

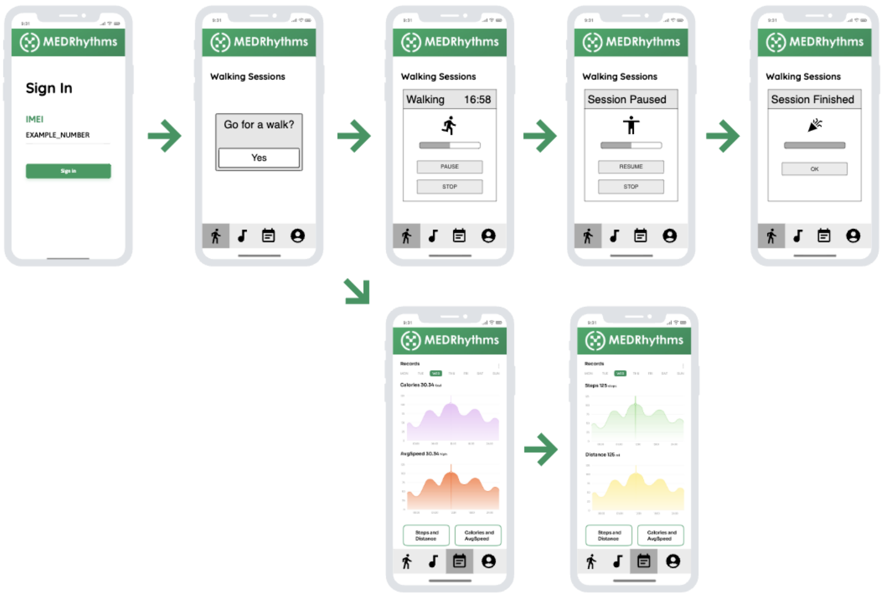

Figure 4: UI wireframes

8 Pattern(s) Used

We used the following patterns:

- Singleton pattern - used to store user details temporarily
- Factory pattern - used to create different music service implementations (such as SpotifyService)
- Observer pattern - used for step update notification UI and other components
- Strategy pattern - used to implement different step detection algorithms
- Facade pattern - provides a simplified interface to the complex subsystems, making business functionality easily accessible to the user

9 Design Concept Review

9.1 Abstraction

The MedRhythms application uses abstraction by simplifying the sensor data. In our case the users will see metrics like step count, distance and speed without needing to know how the sensors(gyroscopes and accelerometers) work.

9.2 Architecture

The MVC design architecture used for this app provides a clear structural foundation, separating the key features into three layers namely:

- Data storage and management(Model)
- User interface(View)
- App logic and interactions(Controller)

We made use of MVC architecture since it makes the code maintainable by defining each component's role clearly.

9.3 Patterns

The application incorporates the following design patterns:

- Observer
- Factory
- Strategy
- Singleton

9.4 Separation of Concerns

The application has different functions(User Authentication, Step Tracking, Music Integration) are all clearly separated into their own components each with its own specific responsibilities.

9.5 Modularity

The application is modular with separate components for User Authentication, Step Tracking and Music Integration, these parts all work independently but communicate through clear interfaces.

9.6 Information Hiding

The application hides complex implementation details behind simple interfaces. For instance the algorithms for step detection are hidden in the step tracking component.

9.7 Functional Independence

The components in the application do specific tasks clearly which means , step tracking only counts steps and the music integration would only manage music. The components interact very little which means they have a high cohesion and low coupling.

9.8 Stepwise Refinement

The app started with a general requirement of step tracking , then we gradually refined the design into specific implementations for speed, calories burned, distance calculation, music integration and synchronization.

9.9 Refactoring

MEDRhythms' architecture supports refactoring by isolating changes to specific components. For instance, changing the step detection algorithm will only affect the StepTracker class and not the entire application.

9.10 Design Classes

The following classes represent some of the core functionalities, they show some of the functional requirements of the application:

- User Login: Manages users details and authentication with their device IMEI number
- Session: This represents user sessions and interactions.

- RecordPage: This class stores and manages data related to steps like step count, speed, calories burned and distance.

10 Architectural Design Considerations

10.1 Economy

Using abstractions like MVC (model view controller) and making a data flow straight forward. This approach reduces complexity by separating data management, app logic and user interface functionality clearly. The Model manages all operations that are data related hence simplifying storage and retrieval, the View streamlines our user interface development by focusing only on presenting data to the user, the Controller reduces complexity in the app logic by acting as a mediator between the Model and View.

Refer to Section 4 and 5.3 for detailed examples

10.2 Visibility

Our Object Oriented Design and component design diagrams help show the overall structure of the application. These components make up our core application.

- The OOD diagram illustrates the relationships between the major classes, their attributes, methods and interactions.
- The Component diagram show how the application key components like the User Authentication, Step Tracking, Music Integration, Database interact and depend on one another.

Refer to section diagrams 5.1 and 5.4.1.

10.3 Spacing

We separate our UI logic from our backend logic while also having a separate service to handle our Database transactions.

- The UI layer is responsible for displaying information and user interaction. It does not handle the processing of data directly.
- The backend logic layer handles logic operations like step counting and spotify integration and communicates between the UI and database.
- The database layer manages storage of data, retrieval independently not involving data operations with the backend logic.

10.4 Symmetry

Since we plan to implement patterns like singleton for DB transactions and factory patterns for music integration, this should allow us to be consistent across all app screens. The idea is to use the same classes and functions that ensure we are implementing consistent behavior and uniform usage across all screens and modules.

10.5 Emergence

Due to client interactions involving tracking of steps or playing music, this leads to MVC-like (model view controller) architecture for our mobile application. Model being our local storage DB, view is our client, and controller being the business logic enabling the view.

11 Component Design Principles

11.1 The Open-Closed Principle

Software entities should be open for extension but closed for modification. Add new functionality without changing existing code.We want to design our analytics page to work in a way where we can support additional metrics or data visualizations. Designing a common interface can help us with this goal, we want to create new modules if needed without hurting what we currently implemented.

11.2 The Liskov Substitution Principle

Subtypes should be able to replace their base classes without changing the program. For our music player feature we want to design the module where different implementations (Spotify) can be interchanged without breaking the code while still supporting common functionality (play or stop music, choose a playlist). This allows us to extend the module possible in the future too since we can update music services independently from each other, meaning that code changes or modifications can be independent from each other without impacting code.

11.3 The Dependency Inversion Principle

High level modules should not rely on low level ones. For our app, the music playing feature will not depend on a specific music player implementation like Spotify but built off a common interface with specific functions (play, pause, stop). By using a common interface we can abstract to other music players with specific implementations like Spotify with the same common functions (play, pause, stop).

11.4 The Interface Segregation Principle

Keep interfaces focused and specific. For our app, interfaces and functions will be designed to do one thing well. For example our sessions interface is different than our login interface, while it is possible to combine the two to handle all user interactions it is better to keep the functionality focused so if in the future we need to extend the capability of sessions or allow other forms of login we can do so without possibly introducing bugs.

11.5 The Reuse/Release Equivalency Principle

Classes/modules used together should be released together.Our application will have several modules each with their own functionality. Interactions with the DB is one module (writeSession, readSession, editSession) where we plan to use it extensively across the app. Having a common set of functionality allows us to update and release new versions of it consistently.

11.6 The Common Closure Principle

Classes that change together should be packaged together. We plan to structure and organize the features of our code in a manner where common functionality will exist in the same package. For example, the DB will be structured so all DB interactions are together. The music player integration will be designed so that all implementations reside in the same location. We want to organize our code so similar things are near each other.

11.7 The Common Reuse Principle

Classes that aren't reused together should not be grouped together. We plan to keep code like the login, session management, music player integration, and analytics separate. This allows us to independently work on each section without causing confusion among developers.

12 Deployment Model

We will use direct APK(Android Package Kit) distribution methods. We can provide the assets online and include instructions for users on how to sideload the app on their phone. We can also create a changelog on our product's homepage where the latest changes or bug fixes can be listed. We can try to provide push notifications to users as new updates are pushed periodically prompting them to install newer versions (through the app store or as an APK file).