Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2024

Swift Start
Sprint O' Clock

Cesar Frayre
Lauren Bryant Eyum
Kosuke Satake
Gin Sang
Shaheen Nijamudheen

Revision History

Revision	Date	Author(s)	Description
0.1	06.25.2024	CF, BE, KS, GS, SN	document creation
0.2	07.11.2024	CF, BE, KS, GS, SN	complete draft
0.3	07.19.2024	CF, BE, KS, GS, SN	revise draft
1.0	07.19.2024	CF, BE, KS, GS, SN	official release

CONTENTS

List of Figures

List of Tables

1 Introduction

Sprint O' Clock is a mobile app for runners and fitness enthusiasts, available on both iOS and Android. It offers a user-friendly interface to track, analyze, and enhance running performance for users of all levels.

A standout feature is the randomized start timer, which tests reaction times with synchronized animations and audio cues ("Ready, Set, Go") along with phone vibrations. The app accurately calculates and records run distances and times, storing them in a calendar interface.

Users can customize settings, including enabling phone vibration. The app also includes a competition feature for real-time or scheduled runs, comparing results to determine the winner. It also allows for the display of distance covered and calories burned.

Our app's back end ensures seamless performance and security, handling data tracking, API management, and user data protection. This document provides detailed design specifications, with additional insights available in the requirement specification and architectural design documents, aligning with user expectations and industry standards.

2 System Overview

The system architecture for Sprint O' Clock is structured into three primary layers: the Front End, Back End Architecture, and Database. This layered approach ensures a clean separation of concerns, enabling efficient development, maintenance, and scalability.

The Front End layer encompasses the user interface components, including the Sprint O' Clock UI Interface and the User Login. This layer is responsible for interacting with users, capturing their inputs, and displaying the necessary outputs. It provides an intuitive and engaging experience, ensuring that users can easily navigate the app and utilize its features.

The Back End Architecture acts as the intermediary between the Front End and the Database. It handles user authentication, manages API endpoints, processes event handling, and ensures security. This layer is also responsible for the business logic and database retrieval operations, ensuring that the application functions correctly and securely. By handling these crucial operations, the Back End Architecture ensures that the app can process user requests efficiently and maintain the integrity of the data.

Finally, the Database layer manages the storage and retrieval of data. This includes user profile management, security policies, data storage, and cloud storage for cross-platform synchronization. The database ensures that all user data is securely stored and readily accessible whenever needed by the Back End Architecture.

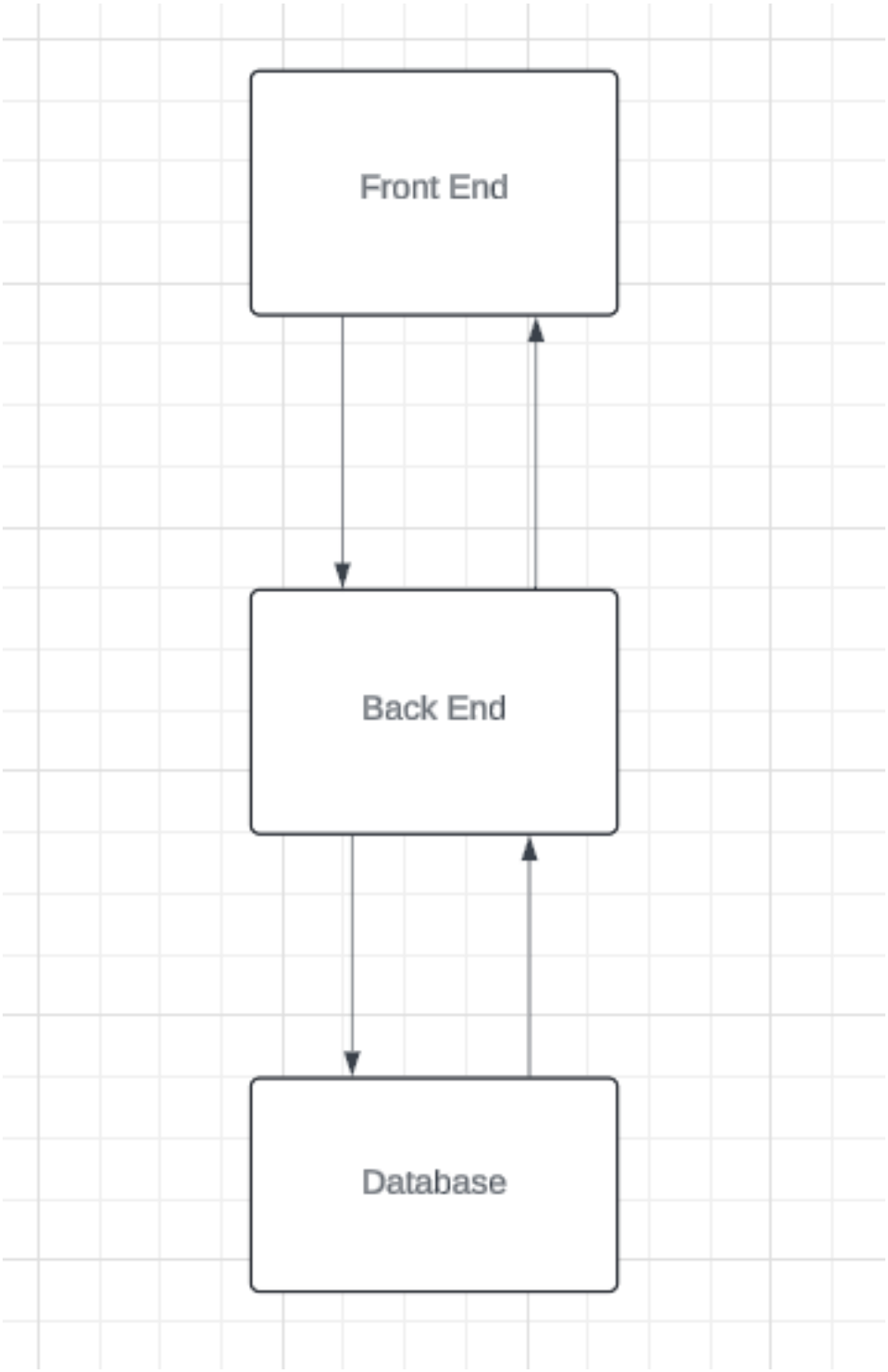

Figure 1: System Architecture

3 Front-end layer Subsystems

3.1 Layer Hardware

The front-end layer operates on standard client devices such as smartphones and tablets. These devices include both Android and iOS platforms. No specific hardware components are required beyond these standard devices. The performance and responsiveness of the application are ensured through efficient coding practices and leveraging the native capabilities of the devices.

3.2 Layer Operating System

The front-end layer is designed to be compatible with popular mobile operating systems, including Android and iOS. This compatibility ensures that the application can reach a broad audience, providing a consistent user experience across different device types and operating system versions. The application takes advantage of the native features offered by these operating systems, such as touch gestures, notifications, and camera access.

3.3 Layer Software Dependencies

The front-end layer relies on several software dependencies to function correctly. React Native is the primary framework used for building native mobile applications using React. React Navigation manages the routing and navigation within the app, providing a seamless user experience as users move between different screens. Axios is employed for making HTTP requests to communicate with backend services, ensuring data can be fetched and sent efficiently. Various UI libraries, such as Material-UI and Bootstrap, are utilized for styling and building user interface components, ensuring a consistent and visually appealing design throughout the application.

3.4 Navigation Subsystem

The Navigation subsystem is responsible for managing the routing and navigation within the application. It provides an intuitive way to move between different screens. This subsystem handles navigation actions and updates the navigation state accordingly. The key files involved are AppNavigator.js, which defines the overall navigation structure, BottomTabNavigator.js, managing bottom tab navigation, and TopTabNavigator.js, handling top tab navigation within specific sections of the app.

ID	Interface Name	Inputs	Outputs
#1	Navigation Subsystem	User actions (e.g., clicks, swipes), navigation configuration, current navigation state	Updated navigation state, screen transition, rendered screens

Figure 2: Navigation subsystem description table

3.4.1 Navigation Hardware

The Navigation subsystem does not require any specific hardware components beyond the standard client devices mentioned in the front-end layer.

3.4.2 Navigation Operating System

The Navigation subsystem operates within the web browser environment and is compatible with the operating systems mentioned in the front-end layer.

3.4.3 Navigation Software Dependencies

The Navigation subsystem relies on the front-end layer's software dependencies mentioned in section 3.3.

3.4.4 Navigation Programming Languages

The Navigation subsystem is primarily implemented using JavaScript for the front-end logic and user interaction.

3.4.5 Navigation Data Structures

The Navigation subsystem utilizes data structures such as JSON objects to represent and manipulate routes and navigation state.

3.4.6 Navigation Data Processing

The Navigation subsystem involves processing user navigation actions, updating the navigation state, and interacting with other front-end components to facilitate smooth navigation.

3.5 Component Subsystem

The Component subsystem includes reusable UI components used throughout the application. These components are designed to provide consistent and efficient UI elements across different screens. The key components include TopTab1.js and TopTab2.js, which are specific to certain tabs in the app. The subsystem relies on various UI libraries and processes component properties to render UI elements based on the provided props and state.

ID	Interface Name	Inputs	Outputs
#1	Component Subsystem	Component properties (props), application state, UI libraries	Rendered UI components, updated component state, consistent UI elements

Figure 3: Component subsystem description table

3.5.1 Component Hardware

The Component subsystem does not require any specific hardware components beyond the standard client devices mentioned in the front-end layer.

3.5.2 Component Operating System

The Component subsystem operates within the web browser environment and is compatible with the operating systems mentioned in the front-end layer.

3.5.3 Component Software Dependencies

The Component subsystem relies on the front-end layer's software dependencies mentioned in section 3.3.

3.5.4 Component Programming Languages

The Component subsystem is primarily implemented using JavaScript for the front-end logic and user interaction.

3.5.5 Component Data Structures

The Component subsystem utilizes data structures such as JSON objects to represent and manipulate component properties and state.

3.5.6 Component Data Processing

The Component subsystem involves processing component properties, updating the component state, and rendering UI elements based on the current state and props.

3.6 Screen Subsystem

The Screen subsystem includes the different screens of the application, such as Home, Login, and Profile screens. This subsystem defines the main screens and handles user interactions within each screen. The key screens include HomeContent.js, HomeScreen.js, LoginScreen.js, ProfilePage.js, RunCalendar.js, RunTimerScreen.js, and SettingsScreen.js. Each screen component is designed using React Native and integrates with React Navigation for seamless transitions. The screen properties are managed using JSON objects, and the content is rendered based on user interactions and application state.

ID	Interface Name	Inputs	Outputs
#1	Screen Subsystem	Screen properties (JSON objects), user interactions, application state, navigation events	Rendered screen content, updated screen state, navigation transitions, save state log in for perspective user

Figure 4: Screen subsystem description table

3.6.1 Screen Hardware

The Screen subsystem does not require any specific hardware components beyond the standard client devices mentioned in the front-end layer.

3.6.2 Screen Operating System

The Screen subsystem operates within the web browser environment and is compatible with the operating systems mentioned in the front-end layer.

3.6.3 Screen Software Dependencies

The Screen subsystem relies on the front-end layer's software dependencies mentioned in section 3.3.

3.6.4 Screen Programming Languages

The Screen subsystem is primarily implemented using JavaScript for the front-end logic and user interaction.

3.6.5 Screen Data Structures

The Screen subsystem utilizes data structures such as JSON objects to represent and manipulate screen properties and state.

3.6.6 Screen Data Processing

The Screen subsystem involves processing user inputs, updating the application state, and rendering screen content based on the current state and user interactions.

3.7 Service Subsystem

The Service subsystem handles communication with backend services and APIs. This subsystem is responsible for managing API calls, data fetching, and processing responses from the backend. It includes files like firebase.js, which manages Firebase services and interactions, and storage.js, which handles local storage operations. The subsystem uses Axios to make HTTP requests, processes API responses, and updates the application state accordingly.

ID	Interface Name	Inputs	Outputs
#1	Service Subsystem	API requests (Such as GPS and authentication), data parameters, local storage commands, authentication tokens	API responses, fetched data, updated application state, track location, distance

Figure 5: Service subsystem description table

3.7.1 Service Hardware

The Service subsystem does not require any specific hardware components beyond the standard client devices mentioned in the front-end layer.

3.7.2 Service Operating System

The Service subsystem operates within the web browser environment and is compatible with the operating systems mentioned in the front-end layer.

3.7.3 Service Software Dependencies

The Service subsystem relies on the front-end layer's software dependencies mentioned in section 3.3.

3.7.4 Service Programming Languages

The Service subsystem is primarily implemented using JavaScript for the front-end logic and user interaction.

3.7.5 Service Data Structures

The Service subsystem utilizes data structures such as JSON objects to represent and manipulate API request and response data.

3.7.6 Service Data Processing

The Service subsystem involves making HTTP requests to backend services, processing API responses, and updating the application state based on the received data.

3.8 Context Subsystem

The Context subsystem manages global state and context within the application. It provides a centralized state management system using React's Context API. The key file in this subsystem is SettingsData.js, which acts as a context provider for user settings data. The subsystem manages and updates the global state, ensuring that context values are available to components throughout the application.

3.8.1 Context Hardware

The Context subsystem does not require any specific hardware components beyond the standard client devices mentioned in the front-end layer.

ID	Interface Name	Inputs	Outputs
#1	Context Subsystem	Context updates, global state changes, context values, user settings data (runs, distance tracked, calories, etc)	Updated global state, context values, accessible settings data (saved data for the perspective user)

Figure 6: Context subsystem description table

3.8.2 Context Operating System

The Context subsystem operates within the web browser environment and is compatible with the operating systems mentioned in the front-end layer.

3.8.3 Context Software Dependencies

The Context subsystem relies on the front-end layer's software dependencies mentioned in section 3.3.

3.8.4 Context Programming Languages

The Context subsystem is primarily implemented using JavaScript for the front-end logic and user interaction.

3.8.5 Context Data Structures

The Context subsystem utilizes data structures such as JSON objects to represent and manipulate global state and context values.

3.8.6 Context Data Processing

The Context subsystem involves managing and updating global state, providing context values to components, and ensuring consistent state management across the application.

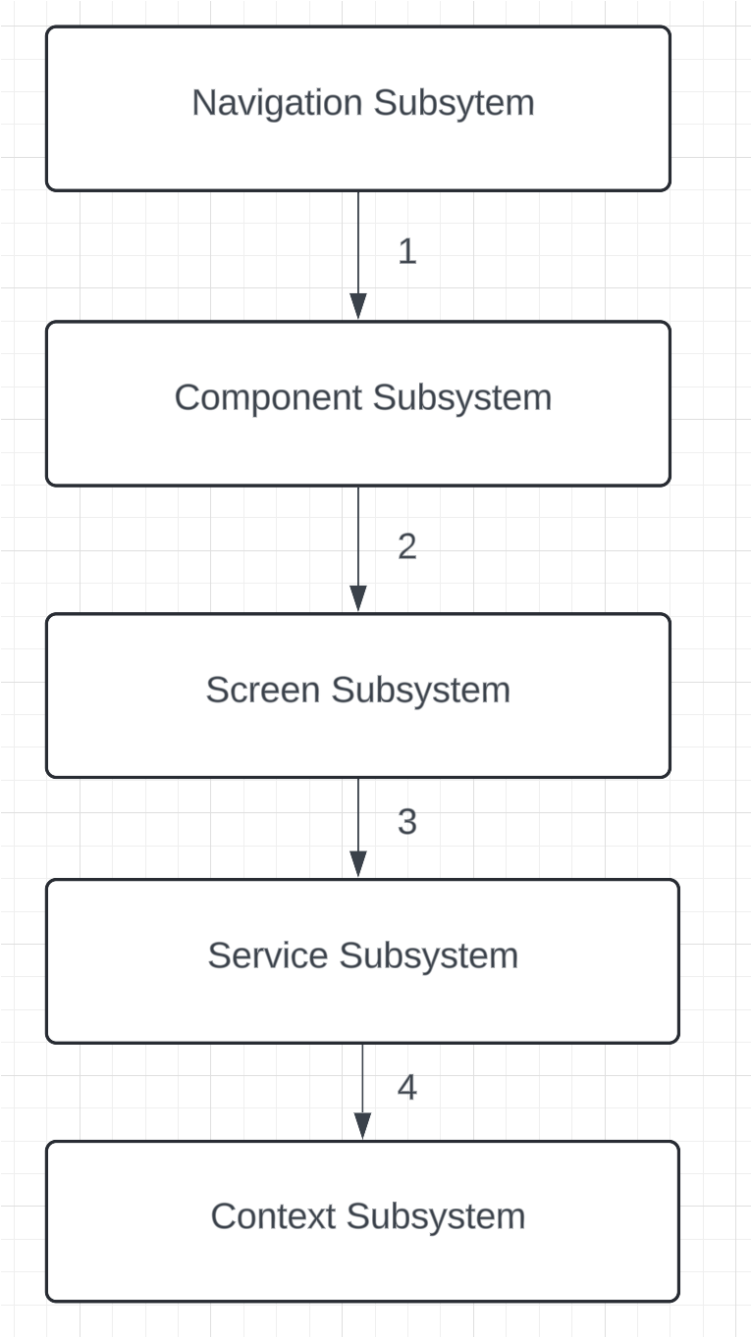

Figure 7: Front-End Layer Subsystems description diagram

4 Back-end Layer Subsystems

In this section, the layer is described in terms of the hardware and software design. Specific implementation details, such as hardware components, programming languages, software dependencies, operating systems, etc. should be discussed. Any unnecessary items can be omitted (for example, a pure software module without any specific hardware should not include a hardware subsection). The organization, titles, and content of the sections below can be modified as necessary for the project.

4.1 Layer Hardware

Mobile smartphones running Android or iOS are hardware components for the Back End layer.

4.2 Layer Operating System

Mobile operating systems Android and iOS are required by the Back End layer for the mobile application.

4.3 Layer Software Dependencies

For the framework, we use React Native (Expo), for the libraries we use React, and for the software language, we use JavaScript. These software dependencies are required by the layer.

4.4 Subsystem 1 - Security Management System

The purpose of this subsystem is to contain manages data and ensures the data is inaccessible to anyone but the user. It is also This uses the Firebase Authentication service, which secures user login data. Also, permissions for location are asked before system use in order to time track the distance of the run.

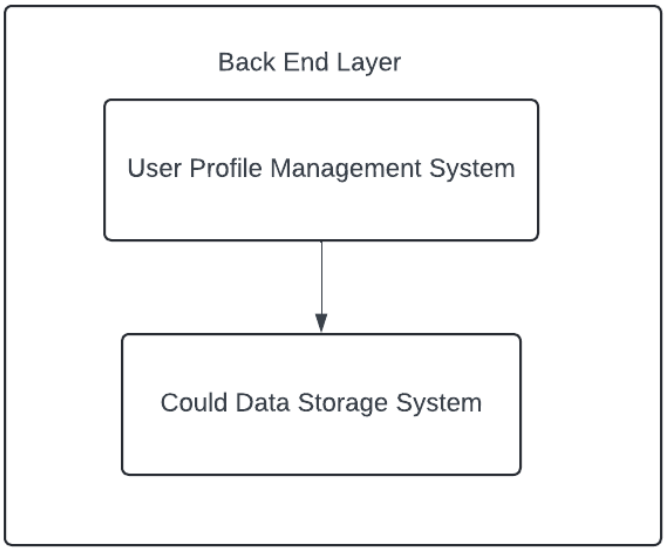

Figure 8: subsystem 1 description diagram

4.4.1 Subsystem Hardware

The involved hardware for the Security Management Subsystem is a mobile smartphone.

4.4.2 Subsystem Operating System

Android and IOS are operating systems required by the Security Management Subsystem.

4.4.3 Subsystem Software Dependencies

Software dependencies required by the system are: React Native Expo for the framework, React for the libraries, JavaScript for the code, and node.js plus npm for JavaScript development.

4.4.4 Subsystem Programming Languages

JavaScript is the programming language used by the subsystem

4.4.5 Subsystem Data Structures

The User Profile Management system utilizes Firebase services to manage user profiles efficiently. This system handles user authentication, profile data storage, and real-time data synchronization. Firebase provides a robust back-end infrastructure that allows for seamless integration and scalability.

4.4.6 Subsystem Data Processing

The User Profile Management System is processed through various forms such cloud functions, real-time database, and authentication. It collects data through registration, validates the users login, and if valid, stored into Firestore.

4.5 Subsystem 2 - Data Cloud Storage System

The purpose of this subsystem is to contain manages all user profile data and ensures the data is inaccessible to anyone but the user. This uses the Firebase Authentication service, which secures user login data, data that the user uses and so on.

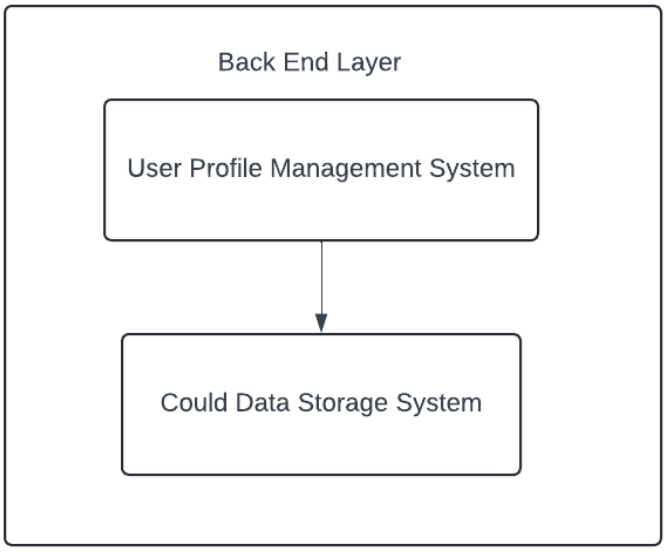

Figure 9: subsystem 2 description diagram

4.5.1 Subsystem Hardware

The involved hardware for the Data Cloud Storage System is a mobile smartphone running Android/IOS.

4.5.2 Subsystem Operating System

Android and IOS are required for the Data Cloud Storage system.

4.5.3 Subsystem Software Dependencies

Software dependencies required by the system are: React Native Expo for the framework, React for the libraries, JavaScript for the code, and node.js plus npm for JavaScript development.

4.5.4 Subsystem Programming Languages

JavaScript is the programming language used by the subsystem.

4.5.5 Subsystem Data Structures

The data structure is designed to manage running data by performing functions such as Storage, Synchronization, and Update and Retrieval.

4.5.6 Subsystem Data Processing

For Storage, it saves the running data to Firestore, ensuring it is securely stored in the cloud. Then for Synchronization it syncs running data between Firestore and local storage to maintain consistency across different devices and offline scenarios. Finally, for Update and Retrieval it facilitates the updates of running data and enable efficient retrieval for display and analysis.

5 Database Subsystems

5.1 Layer Hardware

Mobile smartphones are hardware components for the Database layer.

5.2 Layer Operating System

Mobile operating systems Android and IOS are required by the Database layer for the mobile application.

5.3 Layer Software Dependencies

For the framework we use React Native (Expo), for the libraries we use React, and for the software language we use JavaScript. Other software dependencies include Node Package Manager (npm) and Node.js.

5.4 Subsystem 1 - User Profile Management System

The purpose of this subsystem is to manage user data and ensure that it is accessible only to the respective user. Firebase Authentication is used for handling login data and profiles, with user permissions granted before any access is allowed.

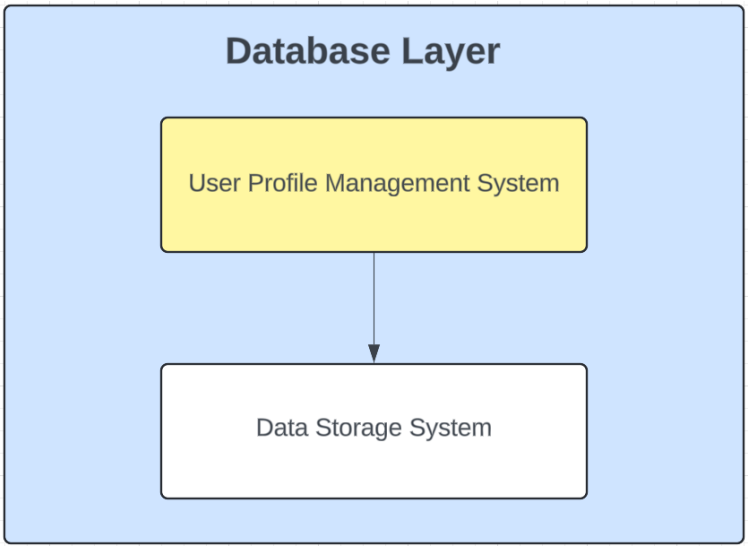

Figure 10: Example subsystem description diagram

5.4.1 Subsystem Hardware

The involved hardware for the User Profile Management Subsystem is a mobile smartphone.

5.4.2 Subsystem Operating System

Android and IOS are operating systems required by the User Profile Management Subsystem.

5.4.3 Subsystem Software Dependencies

Software dependencies required by the system are: React Native Expo for the framework, React for the libraries, JavaScript for the code, and node.js plus npm for JavaScript development and package managing.

5.4.4 Subsystem Programming Languages

JavaScript is the programming language used by the subsystem.

5.4.5 Subsystem Data Structures

The User Profile Management system utilizes Firebase services to manage user profiles efficiently. This system handles user authentication, profile data storage, and real-time data synchronization. Firebase provides a robust back-end infrastructure that allows for seamless integration and scalability.

5.4.6 Subsystem Data Processing

The User Profile Management System is processed through various forms such cloud functions, real-time database, and authentication. It collects data through registration, validates the users login, and if valid, stored into Firestore.

5.5 Subsystem 2 - Data Storage System

The Data Storage System stores tracked running data from the Sprint O' Clock app into the database. It uses Firebase Firestore for cloud storage and AsyncStorage for local storage.

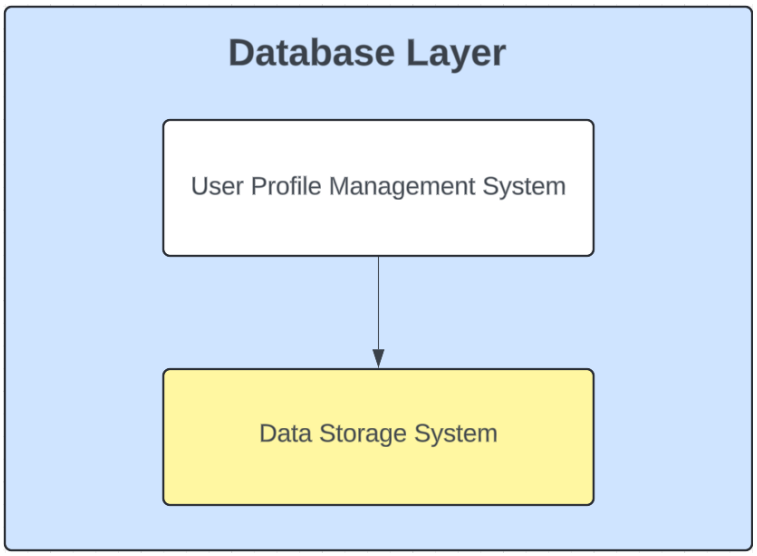

Figure 11: Example subsystem description diagram

5.5.1 Subsystem Hardware

The involved hardware for the Data Storage System is a mobile smartphone running Android/IOS.

5.5.2 Subsystem Operating System

Android and IOS are required for the Data Storage system.

5.5.3 Subsystem Software Dependencies

Software dependencies required by the system are: React Native Expo for the framework, React for the libraries, JavaScript for the code, and node.js plus npm for JavaScript development.

5.5.4 Subsystem Programming Languages

JavaScript is the programming language used by the subsystem

5.5.5 Subsystem Data Structures

The data structure is designed to manage running data by performing functions such as Storage, Synchronization, and Update and Retrieval.

5.5.6 Subsystem Data Processing

For Storage, it saves the running data to Firestore, ensuring it is securely stored in the cloud. Then for Synchronization it syncs running data between Firestore and local storage to maintain consistency across different devices and offline scenarios. Finally, for Update and Retrieval it facilitates the updates of running data and enable efficient retrieval for display and analysis.