System Implementation Document

for

KeepFit

University of Southern California
Spring 2021

Team 12

Aaron Ly
5110281976

Sajan Gutta
2725022497

Roddur Dasgupta
2659572597

Max Friedman
9342195947

Victor Udobong
9031466326

Devin Mui
2587725548

TABLE OF CONTENTS

1. Preface

This document is intended to provide a detailed description of implementing the first release of the KeepFit application. This application is developed for the customer's benefit, Tian Xie, who has requested the product's implementation in this document. This document is oriented explicitly for readership by the customer and developers of the system. It details the changes made in the implementation from the original design doc. Additionally, any other customers interested in the KeepFit application and working with the primary customer will read this document to gain a greater understanding of the application's implementation.

Following this preface, readers will be able to find an introduction to KeepFit and why it is needed. Please note that implementation of the KeepFit project has been conducted according to the issued design and requirement documents. Any critical changes made to the application's design are detailed in the sections below. The GitHub Repository's README file contains technical instructions for running the project in a client-simulated environment (submitted to the customer as a zip file). Changes to the design documentation are indicated accordingly, and this document will be accompanied by a slightly modified version/redlined version of the design document.

We encourage readers to pay close attention to this document's aspects and contact the authors with any concerns or questions.

2. Introduction

KeepFit exists to solve the problem caused by the COVID-19 pandemic, mainly an inability to access facilities for maintaining fitness, such as the gym. The application offers an extensive set of features allowing users to learn about working out, share their progress, track their progress, and more. Our application's end goal is to provide a convenient way for users to "keep fit" during these challenging times.

Account creation on KeepFit has been implemented using Google One Tap, React Native, and Firebase. Google One Tap provides a convenient way for users to sign-in to our app. At the same time, React Native has been used to design an account setup form where users will be able to enter their personal fitness information – including weight, height, and fitness level – to be stored on Firebase, where it is added to and retrieved from.

Uploading a pre-recorded workout on KeepFit has been implemented using Firebase and React Native. Users will create videos on their native device (iOS or Android) in .mp4 format and upload them through our app to be stored on Firebase via Firestore's CDN hosting. React Native manages pushing and pulling videos from Firestore for users to access.

Starting a live workout on KeeptFit has been implemented using React Native, Firebase, and the Zoom API. We have used React Native to make API calls to Zoom to create a meeting link. This meeting link is stored on Firebase and displayed to users where they can workout live. After the live exercise, our garbage collector removes the link from Firebase.

Searching and viewing users and pre-recorded videos have been implemented using React Native and Firebase. React Native is used to design the UI, specified in the Software Requirements Specification (SRS), while Firebase is used to access pre-recorded videos and users, respectively.

A tracking view of workout information has been implemented with React Native and allows users to select a workout type, muscle groups and start a timer. At the end of the workout, users can see their time and the calories they burned. These workouts are saved on Firebase as their workout history when completed. The profile page allows users to view and edit their personal information, view liked videos, and view their workout history.

The complete set of features provided in KeepFit align with the customer's objective of helping others keep fit during the current world situation. While people are stuck home, this application will make it easy to learn exercises, track exercises, and maintain a connection with others on the same journey. The changes made from design to actual implementation now follow.

3. Architectural Change

3.1 What architectural changes did you make during implementation? What's the rationale behind the decision change?

The architecture of this project has remained the same throughout its implementation. It still maintains a 3-tier architecture with storage, logic, and display/front-end layers. The implementation of the project also supports the Publisher-Subscribe and Client-Server styles as indicated in the design documentation.

However, in switching from S3 to Firebase for hosting, we altered the composition of the data storage layer of the 3-Tier architecture for timely implementation. The specific design alterations under this layer are detailed in section 4 below.

4. Detailed Design Change

What detailed design changes did you make during implementation? What's the rationale behind the decision change?

Below are specific design changes we incorporated into our implementation of the KeepFit application:

1. **Video Data Storage from Amazon S3 to Firebase Firestore**
   The reason for this change was to reduce the number of external dependencies utilized by the codebase. We decided to use Firesbases's Firestore because it provides Content Delivery Network hosting at a minimal cost. Using Firebase to store the video data allowed us to write less code, more efficiently reuse code snippets and simplify the delegation of duty among team members, especially for team members with less experience with React Native.

2. **Addition of Extra Sub-Packages**

We utilized some extra packages in our implementation. These packages are sub-packages to packages already included in our design documentation, requiring no adjustment. They are listed below with the reasons for adding each:

1. **React Native Picker**
   This off-the-shelf Picker component expedited the implementation of our user input handling. It is an easy-to-use and aesthetic list selection interface component. We used it to allow users to select skill level, muscle group, category, etc. in many different contexts including sign up and exercise filtering.
2. **React Native Stopwatch Timer**
   This package enabled us to implement the exercise timing functionality much more efficiently than creating our timer tool. Using this package saved our team a lot of time in the implementation. The timer maintains state, continuing timing in the background regardless of where the user navigates.
3. **Expo Auth Session**
   This package enables the user to login to Zoom and creates a live stream event. This package provided a straightforward way to implement the Zoom token exchange by utilizing Expo's prebuilt auth endpoint.
4. **React-Navigation**
   Using this package enabled us to wrap the entire application's component tree with a core navigation component. Under this implementation, RootStackNavigator handles navigation for the application and a Tab navigator is used for the 5 major UI screens (e.g. profile, track, etc.).

3. **<u>Adjustments to Class Diagrams</u>**
   In our implementation process, we made a few adjustments to the composition of our project classes. These changes are detailed below:

   1. **Addition of birthday field to the User class**
      We added this member item to address a client requirement that was not addressed in earlier versions of our documentation. The requirements for this deliverable specifically ask us to collect new users' birthday.
   2. **Addition of Secondary Muscle Group**
      Rather than storing a list of muscle groups for live stream and video, we chose to have a secondary muscle group. This makes user input easier.
   3. **Changed Derivation of Firebase Class**
      We adjusted the firebase class to be derived directly from an initialized firestore object built into the Expo firebase SDK. This allowed us to simplify the process of making database queries in various parts of the project and use a single database connection for the client.

4. **Implementation of Instantaneous User Search**
   Our previous design outlined a user search function that would allow users to enter a search phrase, select certain filters, and press the search button to execute their search. During our implementation, we decided to take a much simpler approach. Instead of filters and a search button, the user interface has been simplified into a simple text box. As the user types in their search phrase character-by-character, each change to the search phrase triggers an update to the search results. The search returns all users whose name contains the search phrase as a substring.

5. **Using Firebase to Handle Authentication Tokens**
   Instead of using Google Auth, we decided to use Firebase to handle authentication for existing users logging in. We decided to do this because it achieved the same goal with less complexity. This allowed us to make fewer API calls and be more reliant on our database.

6. **Following Users (Extra Feature)**
   Users have the option to follow other registered users on the platform. They can follow and unfollow them from the user search page. This serves as a foundation for future features such as viewing follower counts and viewing other users' following and being followed. There can also potentially be a feed in the future, similar to what Instagram offers.

5. Requirement Change

5.1 Scenario 1

Assume the user is looking through several live streams. Some live streams expire due to out of time or the hosts canceling the streams. Users should see a message that these streams have been ended, including an option to resume watching if the streams go live again.

This scenario does not change our design, but it does require a small addition to our implementation. Currently, all live streams created are displayed on the “Search Livestream” view within the “Explore” page for users to click on and join. Should the live stream expire due to running out of time or cancellation, the link will remain on the page. Users will realize that the stream has ended if they attempt to click on it, but there is no warning beforehand that it has ended. To rectify this, we would create a garbage collector (via an asynchronous background task or cron job) that deletes entries of defunct live streams. We would also consider adjusting the “Search Livestream” view to show live streams by the user. If the user starts another live stream, we can update the link and make sure only active links are displayed. Firebase provides tools such as listeners, which allows our data to update in real-time automatically.

5.2 Scenario 2

Assume the user clicks the Yoga button in the exercise interface to record his exercise. However, he does not remember how to perform the Yoga moves accurately and decides to watch a video first. We need to allow the user to directly jump to the Yoga videos without searching in the exercise demo interface. If the user exits the training to watch videos, he should be accounted for one exercise period and be recorded by the app.

This scenario slightly changes our design. Currently, the user can click on a button within the “Track” page to record their workout (not specifically Yoga, but we do have bodyweight, HIIT, and other exercise options available). If the user does not remember how to perform the exercise’s movements, they would have to click on the “Explore” tab and search for the specific exercise to see a video demo. To account for this feature, we would display all videos about a particular activity, such as Yoga, on the same “Track” page that they are currently on. We would do this by filtering all of our demo videos (which are present on the explore page) by activity and display the videos relevant to that activity on the “Track” page. Our app does already take care of

the second case, which is if the user exits the training to watch videos, it will still be recording the time elapsed and calories burned for the user until they stop the timer and click save.