Assignment Specifications

Keepfit

Table of Contents

Section I

Group Members

1. Zhenyi Yang zhenyiya@usc.edu 2239204516
2. Yi Xu yxu70832@usc.edu 2969873928
3. Gloria Liu liuglori@usc.edu 5512683227
4. Peidi Xie cindyxie@usc.edu 4927833566
5. Kevin Lee klee3663@usc.edu 8101223380

Section II

Preface

Keepfit will be an iOS mobile application where customers will be taught to exercise at home and can share their own experience and help others to exercise as well.

Customers of Keepfit can register as users to log in to the mobile application. Once registered, Keepfit will set up a personal file for the user and user's information such as nickname, photos, exercise history can be publicly available to other users in the user's personal file page. Users can choose to join a live broadcast or watch exercise tutorial videos by their own. Users can also start a live stream themselves so that other users can join, and users can choose to follow or unfollow each other. Keepfit can also record the time of the user's exercises and the calorie user burnt during the exercise.

This document is intended for the developers of the Keepfit Project.
Current Version of This Document: 1.0. In this version, we specify changes we made in terms of our architectural design as well as the detailed design in our design documentation. We will also discuss how the requirement changes will potentially affect our design.

Section III

Introduction

Keepfit responds to the growing need of at-home exercises during the COVID-19 pandemic. It will build a community for people who wish to keep fit while stuck at home. It will have help users with tracking and recording exercises as well as connecting to a greater community of "keepfitters".

Users can register an account and set up his or her individual personal profiles which contain information about the user's nickname, photos, exercise history and so on. Users will be able to see exercises as well as post their own. They can join exercises and sort the exercises in both forms of live broadcast and exercise tutorial videos. They can also see streams and start their own stream, follow and unfollow each other, and record the time of their exercises and the calories they burn.

Keepfit is similar to other exercise applications. Taking the “Keep” application as an example, both applications have functions such as user registration and authentication systems, personal profiles, customized exercise plan, exercise tutorial videos and live broadcast to join, recording of time of exercise and calorie being burnt, etc. Keep also enables users to post content and pictures and reply to each other’s post which facilitates user interaction, whereas Keepfit enables users to start their own stream, which is a function absent in Keep.

As we discussed in the design documentation in the Assignment2, the major components of our iOS app includes a front-end UI that users of Keepfit can interact and communicate with and a back-end service which is responsible for processing the user data. As we were implementing our project according to our design documentation, we encountered different difficulties preventing us from fully following the detailed design documentation. Thus, we would have to come up with decent alternatives to fulfill our customers’ requirements. That's what we are presenting in this documentation. Notice, this implementation of the entire Keepfit project is not the final version because we are only given a limited time. In the future we will improve the app in terms of both aesthetic and functionality.

Section IV

Architectural Change

Change 1: We did not implement a remote Amazon Web Server. Instead, we are utilizing MongoDB and Firebase as our remote backend server provider. We initially chose MongoDB Realm and Atlas as our backend solution, but we decided to move on to Firebase during our implementation period. As of now, the user registration/profile mechanism as well as the streaming/video hosting is implemented using Firebase, whereas the backend support for updating zoom link and viewing user detail is still in the process of moving from MongoDB to Firebase. The reason we made these platform change decisions is mainly due to the difficulty of teamwork. Our team has people from across different time zones, with diverse and different technical backgrounds. It has proved very difficult to use AWS and MongoDB because of the lack of available documentations online, and some of our team members feel uncomfortable using these frameworks. We spent a lot of time redoing the backend logic and configuration to implement the same functionalities as a result. We feel that due to the asynchronous nature of our teamwork, these objective difficulties prove to be an inevitable setback.

Section V

Detailed Design Change

Change 1: We didn't implement separate functions for resetting each individual account info such as nickname and birthday, instead we implement a single function that changes and updates all sorts of account info at the same time. The text fields for account info in the frontend will be populated with the original user data fetched from the database so that if the user doesn't want to change a certain field, he or she can leave the text field with the existing data and that field of the user object in the database will just be set to the same data.
We find it unnecessary to implement multiple functions for changing the account info. The UI for editing all fields of account info on the same page is more intuitive for users, and the data size for each field of account info is relatively small so that rewriting fields with existing data does not cost running time.

Change 2: We implemented email-password authentication method instead of username-password method because firebase doesn't support username authentication. By using firebase authentication service, we can save a lot of work coding authentication logic and the users' accounts are more secure. Since we adopted email authentication, we deleted the username variable from the user object.

Change 3: For the backend-side,w e use Firebase as our backend service rather than AWS database or mongoDB, which were specified in the detail design document. The reason that we changed our backend service to Firebase is because we appreciate the idea of backend-as-a-service and Firebase does this job very well. We don't have to rewrite the entire user system and authentication system from scratch. This is a very good example of software reuse. It's very powerful so that we can focus on our core functionality rather than re-implement the wheel. Notice here, we don't mean that AWS backend service as well as mangoDB realm backend are not capable to achieve our requirement. But our development team found that Firebase does a better job and there are more high-quality tutorials for our coding crew to learn with.

Change 4: According to the class diagram in our detail design document we are supposed to have LiveExercise and NonLiveExercise classes which inherit the Exercise class. However, because of the technical difficulty the NonLiveExercise class is temporarily replaced with zoom link for simplicity in its early stage of development. In the future we would change the models to fit our detailed design document.

Change 5: Because in this assignment we are not required to implement all the functionalities of the app, rather we focus on core functionalities. So our models or classes do not look exactly the same as they are in our class design diagram. Because some of the data and methods are not necessary to achieve the requirement. But we are working towards our class design diagram as we are required to implement more and more functionalities. Take User as an example, we don't have username data as well as history. Another example is that we don't have the History class right now. One last example, is that for Exercise we only have the title and the link while we don't have the creator or type.

Change 6: We separated the search interfaces for workout videos and streaming for now instead of merging the two into a single exercise interface. The schema of our database has two parallel collections of users and videos that are not connected to each other yet, therefore we dispatch the work to different teammates to implement streaming search and video search separately. It takes extra time for teammates to cooperate on a single view and new bugs can be generated during the process, so currently we just maintain two separate interfaces for streaming and video search for cleaner coding and testing practices.

Section VI

Requirement Change

Scenario 1

We need to make changes to our design to achieve this scenario. Currently our intended design can fulfill the requirement of displaying a message to the user when the stream has ended, but our design cannot enable the user to choose to resume watching the stream when it goes live again. The product only fetches the appropriate zoom link data of the host from the database and presents the link to the user, but doesn't automatically return the zoom link to the user when the zoom room goes live again.

To fulfill the new requirement, we need to implement a new function that holds the zoom link data in a static variable. This function will be triggered if the user pressed the button to resume watching if the zoom room goes live again, and it will call another function to actively listen to the status of the zoom room by using functionality provided by the zoom API. Once the function catches the signal of the reopening of the zoom room, the function will send a message to the user including a button to enter the zoom room. All the functions and data for this functionality will be written in the ViewController for the streaming interface.

Scenario 2

We need to make changes to both our frontend design and database schema to achieve this scenario. Currently in our intended design exercise videos query and timer recorder are achieved in two separate interfaces that don't have any communication of data and function calls, therefore the user has to quit the time recorder and enter another interface to search for workout videos. Since the user quit the time recorder interface, the chunk of time the user watching video is not counted to the user's exercise time.

To fulfill the new requirement, we decide to implement a table view of recommended videos directly below the time recorder so that the user doesn't need to change to a new interface and the time for watching videos is also counted by the recorder. Once the exercise page is loaded, the table view will be populated with video data of the relevant exercise category directly from the database. To facilitate the fetching process, we also decide to restructure the data

collections in terms of how the videos are stored. Currently in our database we just stored all videos in a single collection without any structure or hierarchy, therefore this time we need to group the videos based on their exercise categories so that videos of a single category can be easily fetched and populated. The user will not quit the exercise time recorder interface when watching videos and shutting them down.