CSC 431

UFIT

System Architecture Specification (SAS)

Team 02

Maansi Patel	Software Developer
Natalia Jimenez	Prototyper
Diep Vu	Scrum Master
Isabel Ogilvie	Software Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	3/28/22	Maansi Patel, Natalia Jimenez, Isabel Ogilvie, Diep Vu	First draft

Table of Contents

Table of Figures

1. System Analysis

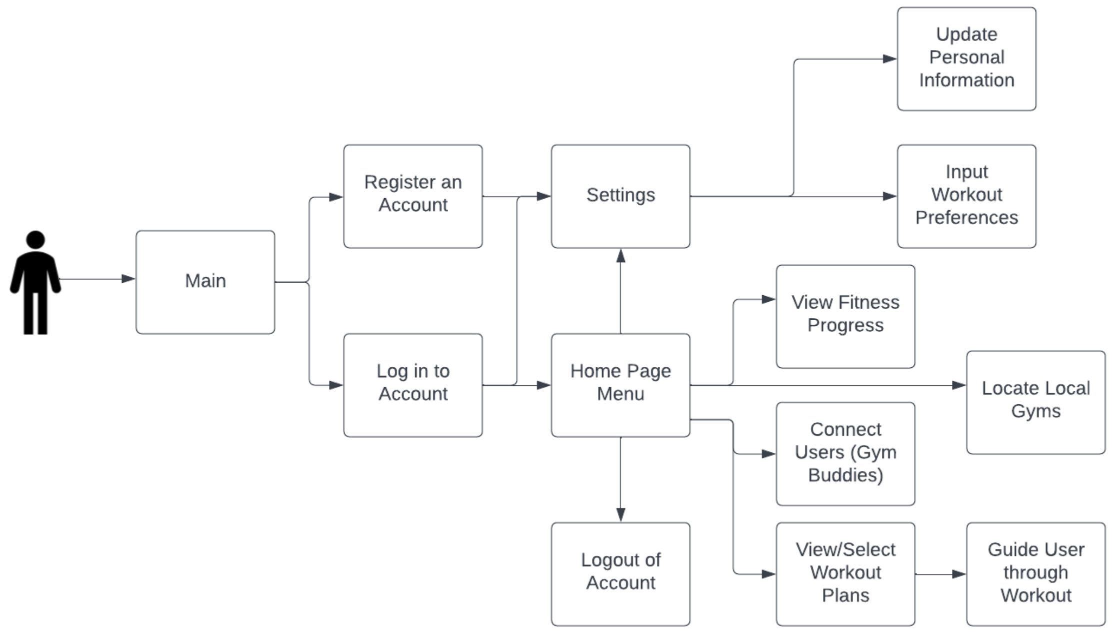
1.1 System Overview

Physical activity or exercising plays a crucial role in maintaining a healthy lifestyle and improving mental health. UFit is a fitness and social networking mobile application that will connect individuals who are looking for ways to stay active and meet their fitness goals. The composition of the system is focused around the following main functions:

- Creating customized workout plans (based on desired intensity, target muscles, existing injuries, etc.)
- Designing diet plans based on fitness goals (losing/gaining/maintaining weight/building muscles/increasing stamina, etc.)
- Connecting persons on the app to each other to help them find gym buddies to work out with based on fitness goals and location.
- Recommending suitable gym locations with the necessary equipment to achieve their goals nearby

The system will be composed of the user, the app, hosted through github, and the database which will store aloof the necessary system and user data. There will be interaction amongst all the components, and thus there will be a general flow from the main page through to the database component. The database will be hosted on the cloud, and within the database, each user's login, personal information, and workout profile/information will be stored and encrypted. The workout plans and instructions will be hosted offline, so our neural network will be stored on the device the app is hosted on for feed-forward purposes only. Lastly, because our app is being hosted on a remote server before delivering the app's capabilities to the users via the Internet, we will be using a SaaS (Software as a Service) Architecture.

1.2 System Diagram


Main
Register an Account
Log in to Account
Settings
Home Page Menu
Update Personal Information
Input Workout Preferences
View Fitness Progress
Locate Local Gyms
Connect Users (Gym Buddies)
Logout of Account
View/Select Workout Plans
Guide User through Workout

- The above diagram represents a comprehensive overview of the UFit system architecture based on the functional requirements identified previously in the development process.
- Each node correlates to a functional requirement with the exception of *Main* and *Home Page Menu*. *Main* serves as the root of the application, and allows the user to access app functionality without a pre-existing account (i.e. registering an account). The *home page menu* serves as a landing page from which all other functionalities can be accessed.
- *View Fitness Progress, Connect Gym Buddies, View/Select Workout Plans* and so on are features only available to registered users, which is why they are only accessible through the *Log in to Account* node.

1.3 Actor Identification

- Gym-goer: It is a user (human) actor. The user will be able to add workouts to their list and modify their preferences based on their physical abilities. Users will be able to receive alerts on their planned workouts and connect with other users to do workouts

together. The gym-goer is the primary actor because he is the one initiating the interaction with the system.
- Time: The system clock will help users keep track of their workout schedules by helping them stay organized through a shared calendar function with their Buddies.
- Database Administrator: It is an external system that serves as a secondary actor in a use case. It provides a workout database as well as tutorial videos. The content is provided by professional trainers.

1.4 Design Rationale

1.4.1 Architectural Style

The app will use 3-tier architecture in the form of:
1. The first layer is the UI. Adobe XD was used to create and prototype the screens. The actual widgets were created using Flutter.
2. Business Logic Layer: Includes logic for progress tracking and finding Gym-Buddy matches based on preferences shared by users.
3. Service layer: User information will be saved in the database

1.4.2 Design Pattern(s)

The app will likely use the Adaptor Design pattern, so that components can communicate between layers. This structural design pattern will allow components with incompatible interfaces to talk to each other without having to modify their source code.

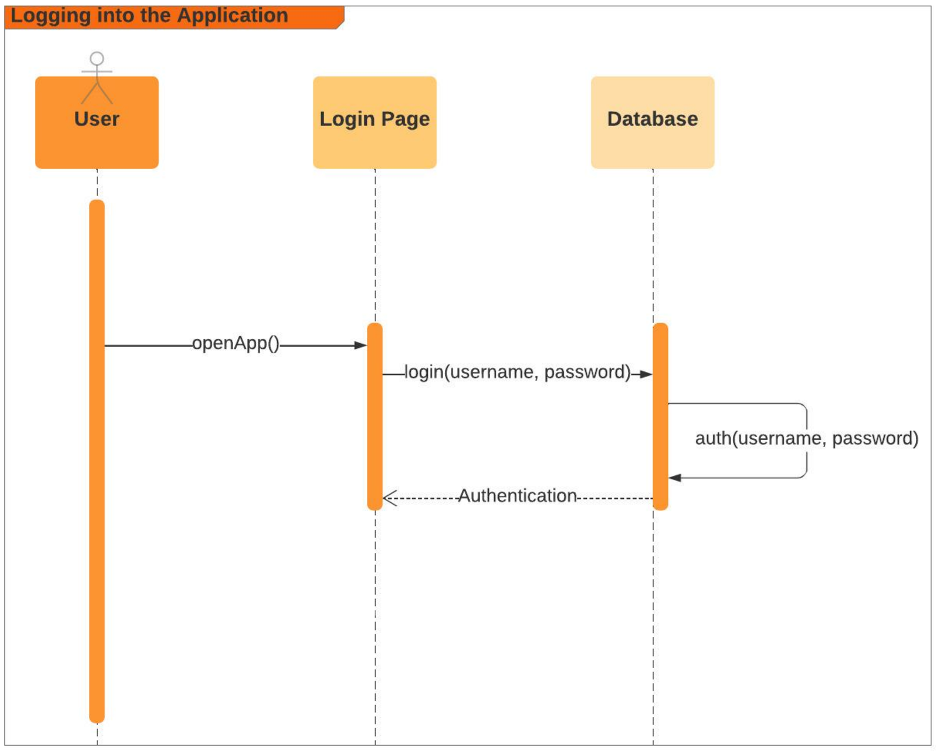
1.4.3 Framework

We will be using HTML, CSS, and Javascript for the frameworks. There is a lot of information on them and our team members are familiar with them. Amazon lightsail will be used to deploy the application. This will store user information and will be good enough for the early stages of the project. Our application will use Node.js as the framework, which will allow us to use HTML, CSS, and Javascript. Javascript will be used to write the backend and React Natieve Js will be used as the frontend framework.

2. Functional Design

2.1 Sequence Diagram

2.1.1 Logging into the Application

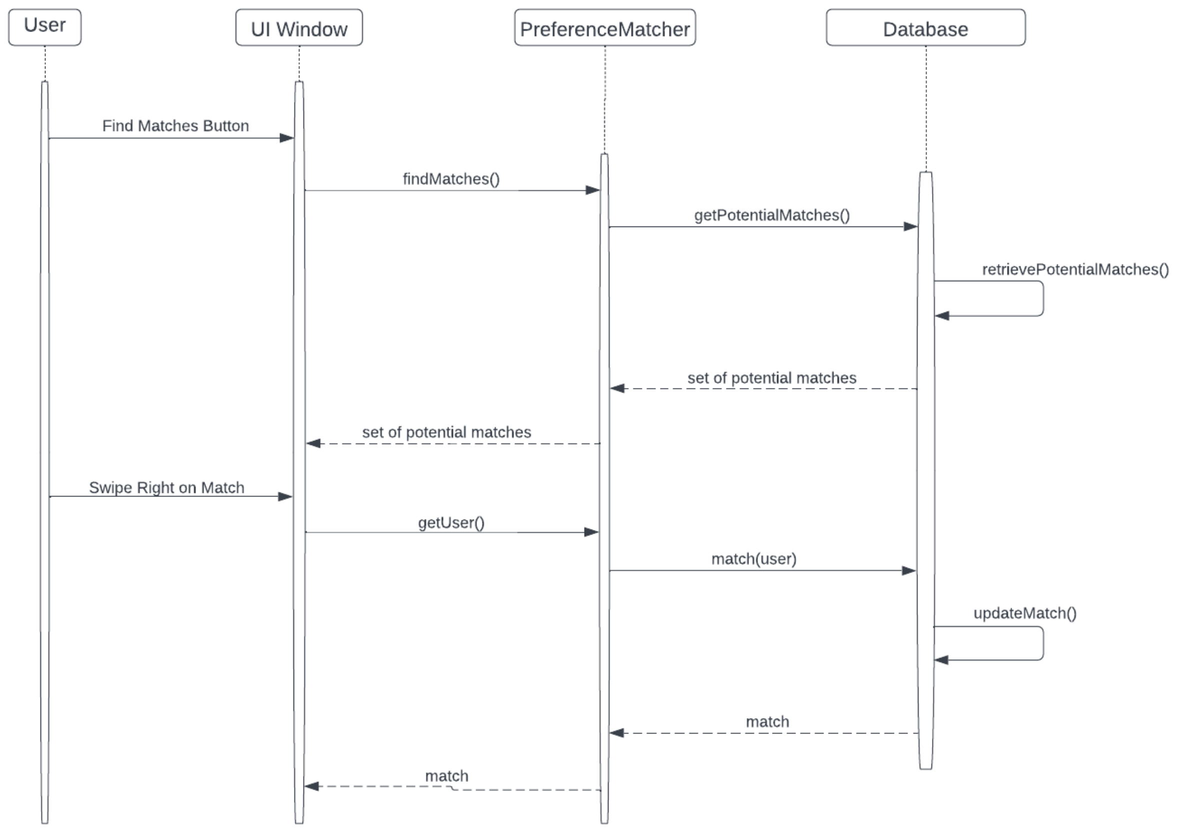


• openApp() is called when the user opens the application on their mobile device, which then takes them to the login page

• The user will then enter their username and password, which the login page will call login() with to connect to the database

• Then, the database will verify the given username and password with already registered users and send verification back to the login page

2.1.2 Finding Potential Matches and Matching Users

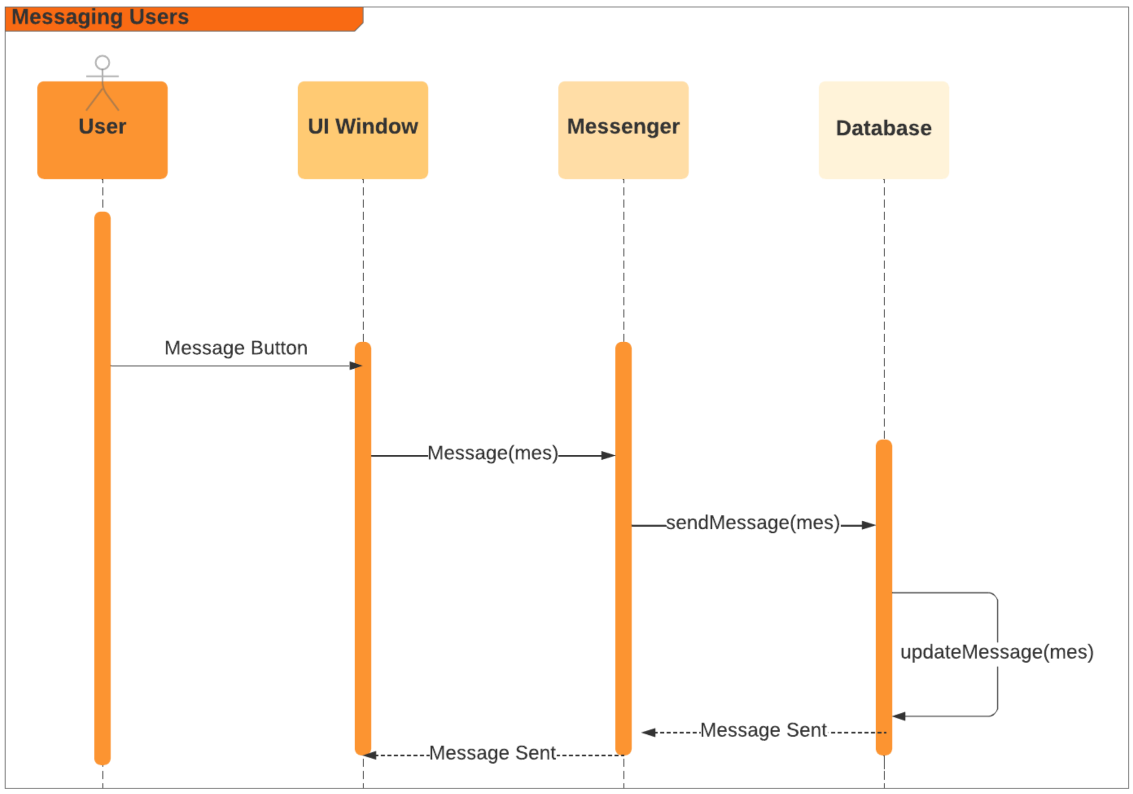


• When the user presses the Find Matches button, the UI Window will call findMatches(), which creates a PreferenceMatcher for their profile. PreferenceMatcher is responsible for all matching functionality, including finding compatible people for potential matches and matching users.

• After getPotentialMatches() conducts the compatibility algorithm to find potential matches, this information will be retrieved and updated by Database through retrievePotentialMatches(). If the process is successful, a set of potential matches is returned and passed all the way back to the user.

• When the user matches with another by swiping right, the UI Window calls getUser(), which sends their information to a PreferenceMatcher. The PreferenceMatchers class calls match(user), which then updates this match in the user's database information through updateMatch(). If the match is successful, a match is returned, added to the user's set of matches, and visible to the user.

2.1.3 Messaging Users



• Users can message other matched users by pressing the message button, which then communicates with the UI Window where the user will enter the message they want to send

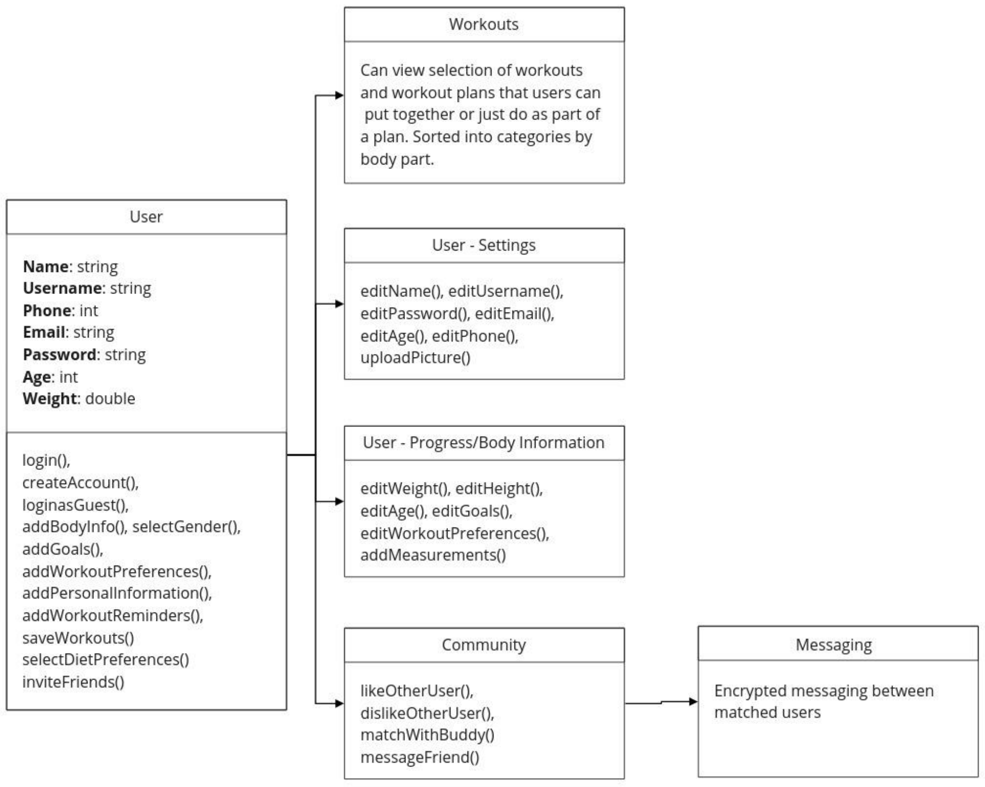
• Then the UI Window sends the message to Messenger

• Messenger then sends the message to the database where the input message is updated in the database by updateMessage()

• After it is updated, a Message Sent verification note is returned to the user

3. Structural Design

3.1 UML Class Diagram


Workouts
Can view selection of workouts and workout plans that users can put together or just do as part of a plan. Sorted into categories by body part.
User
Name: string
Username: string
Phone: int
Email: string
Password: string
Age: int
Weight: double
login(),
createAccount(),
loginasGuest(),
addBodyInfo(), selectGender(),
addGoals(),
addWorkoutPreferences(),
addPersonalInformation(),
addWorkoutReminders(),
saveWorkouts()
selectDietPreferences()
inviteFriends()
User - Settings
editName(), editUsername(),
editPassword(), editEmail(),
editAge(), editPhone(),
uploadPicture()
User - Progress/Body Information
editWeight(), editHeight(),
editAge(), editGoals(),
editWorkoutPreferences(),
addMeasurements()
Community
likeOtherUser(),
dislikeOtherUser(),
matchWithBuddy()
messageFriend()
Messaging
Encrypted messaging between matched users