MIAMI

CSC 431

COR

System Architecture Specification (SAS)

Team 13

Julia Eisner	Scrum Master
Jeffrey Hudak	Developer
TC McCaffrey	Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	3/24/2022	TC McCaffrey, Julia Eisner, Jeffrey Hudak	First Draft

Table of Contents

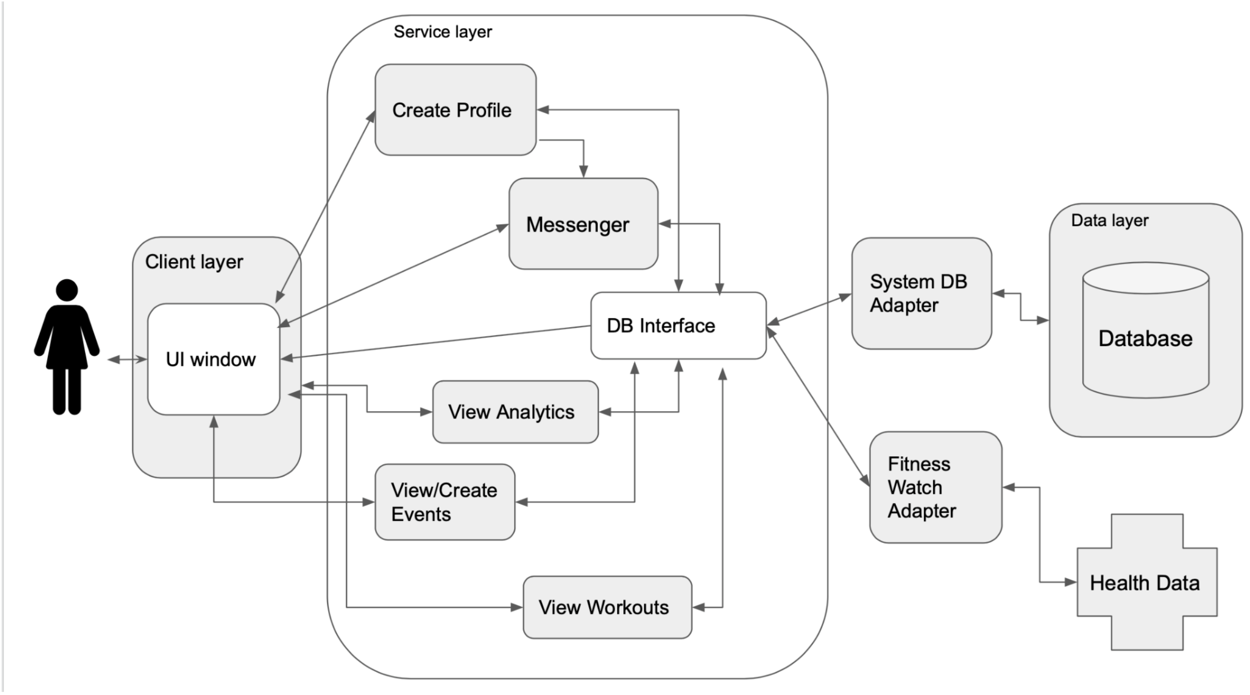
System Analysis

1.1 System Overview

This document describes the architecture design and specification of the COR Fitness Application which uses a three-tiered architecture, including a client layer, a service layer, and a database layer. The client-server is the IOS client user interface, which accesses the back-end service domain through DNS and provides the data that is reflected back to the user in the user interface. The service layer includes the application logic, including the messaging feature, the events feature, viewing graphed health analytics, and the workout finder. The database layer uses Firebase Database to read and store data which also provides backend services like authentication and security. The data from the fitness watches will be sent to the database immediately while also being incorporated into the service layer.

Diving deeper in the three layers, first there is the UI layer, which is the user interface between the user and the service. The UI Layer connects directly to the service layer to display the information it has to the user with fancy graphics and layout design. The user will almost entirely interact with the UI layer purely through buttons in the UI layer, with the exception of the fitness watch's data which has its own passthrough to the database. The service layer has components for all the actions stated above. It will receive the request from the UI, and then reach into the database through various adapters to pull and be able to read the proper information requested. The DBInterface within this layer is implemented by converting UI requests into a language readable by the DB, and it will also translate extracted data into a language usable by the components in the service later; it also communicates with Firebase to verify login credentials and authenticate users.

1.2 System Diagram



1.3 Actor Identification

Actors interacting with the system include:

- New Users: Users are individuals who have not yet created an account with COR.
- Registered Users: Users are individuals who have registered their accounts with COR.
- Platform Server: The platform, Firebase, includes a built-in cloud database and authentication services with third-party applications.
- Fitness Watch: The data from the user's watch will be incorporated into the system.

1.4 Design Rationale

1.4.1 Architectural Style

The app will utilize 3-tier architecture in the form of:

- UI Layer: This layer uses Flutter to create UI widgets.
- Service Layer: This layer includes the logic for the preferences matcher and suggested upcoming concerts.
- Database Layer: This layer uses Firebase for services like authentication through third-party applications and database storage.

1.4.2 Design Pattern(s)

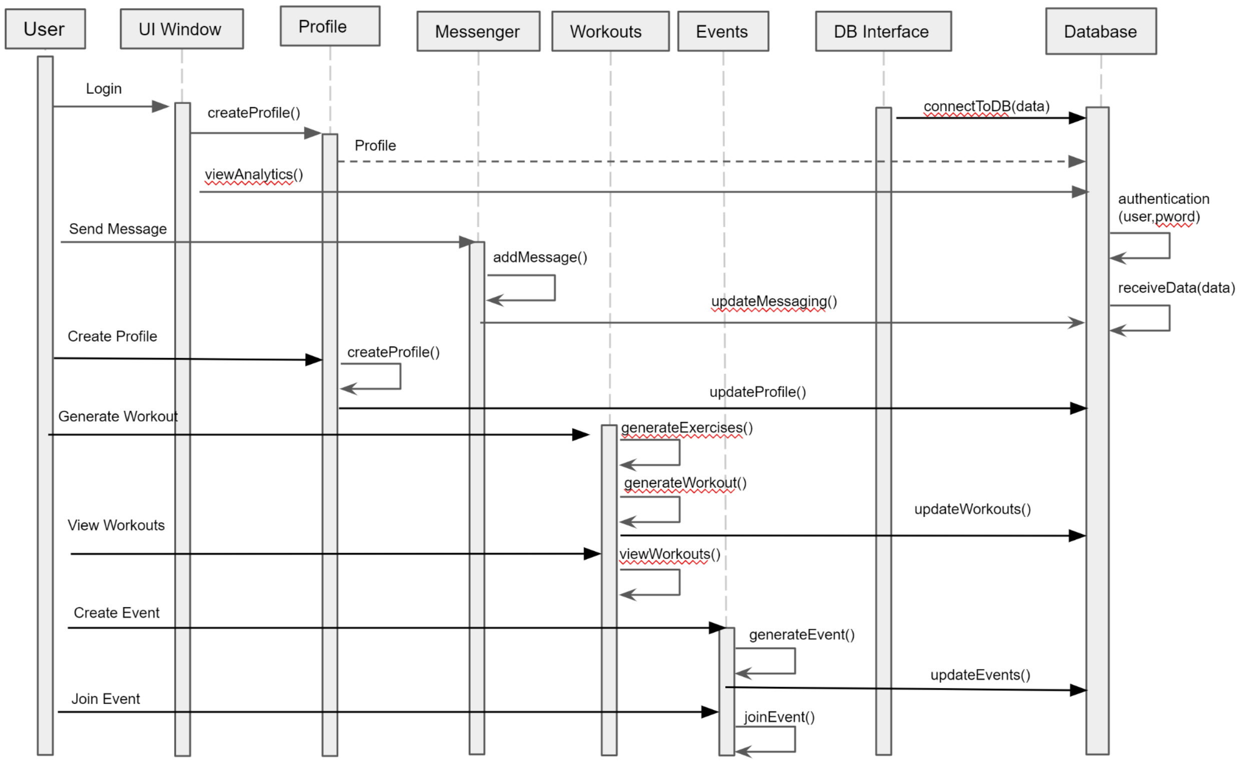
Adaptor Design will be heavily used in our project as every component in the service layer eventually leads to the DB Interface which is an adapter from the service information to the data layer. Similarly, data from the database in the service layer will be converted back into usable data by components in the service layer using an adaptor class.

1.4.3 Framework

We will be using Node.js for the UI framework which is well-documented, and there are extensive resources and learning abilities on the web. However, our app also requires additional backend services, such as authentication, database storage, and health data compatibility to implement workout generating machine learning. Firebase is a Backend-as-a-Service that can implement authentication, cloud database storage, and security functions for our app with ease.

2. Functional Design

2.1 User to Data



Description:

Login - When the user logs in it will either prompt them to make a profile, which will be authenticated by checking if the profile already exists in the database. If not, creating a profile will send a new profile to the database for future use. Otherwise, the system will allow the user to view their analytic data.

Send Message - The user can add a new message to either an existing or new conversation with another profile in the system, which either way will add a new message between the two users to the database.

Create Profile - The user can either create a new profile if they choose with different authentication or they can update their profile such as updating an email address or password.

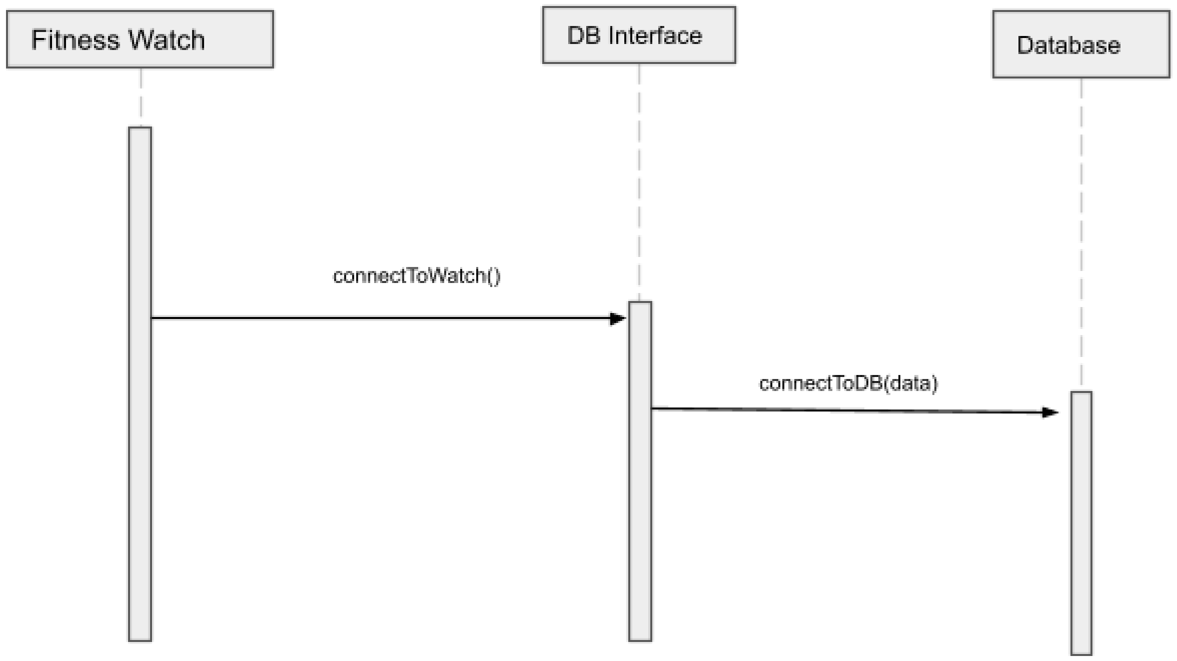
Generate Workout - The user can choose to either generate exercises or generate entire workouts. Exercises are individual pieces to workouts, such as “10 pushups”. Workouts are a curated list of exercises for the user to complete, which will then be sent to the database so the user can reference their old workouts in the future.

View Workouts - This pulls old workouts saved within the database for the user to see.

Create Event - After putting in event details, the user can officially generate an event which will save the event to the database for other users to see and join when they go to join events.

Join Event - A user can tell the system they plan on joining an event which will save the location, name, and time of the event to the user’s profile.

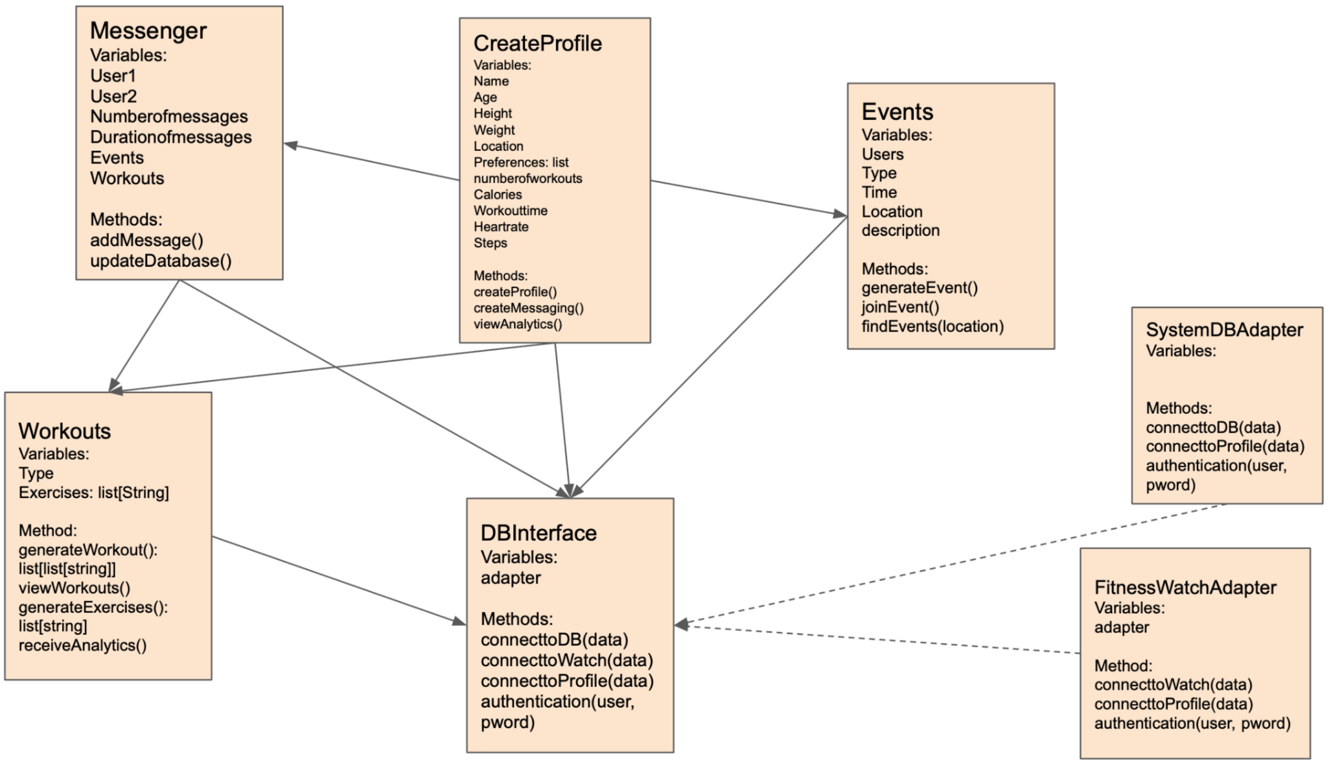
2.2 Watch to Database



Description:

Fitness Watch - After completing a workout, whether through workouts offered through COR or another fitness application a user chooses to use, the database will connect to the watch and pull new workout information to save into the database as workouts to be viewed in the app.

3. Structural Design


Messenger
Variables:
User1
User2
Numberofmessages
Durationofmessages
Events
Workouts
Methods:
addMessage()
updateDatabase()
CreateProfile
Variables:
Name
Age
Height
Weight
Location
Preferences: list
numberofworkouts
Calories
Workouttime
Heartrate
Steps
Methods:
createProfile()
createMessaging()
viewAnalytics()
Events
Variables:
Users
Type
Time
Location
description
Methods:
generateEvent()
joinEvent()
findEvents(location)
SystemDBAdapter
Variables:
Methods:
connecttoDB(data)
connecttoProfile(data)
authentication(user, pword)
Workouts
Variables:
Type
Exercises: list[String]
Method:
generateWorkout(): list[list[string]]
viewWorkouts()
generateExercises(): list[string]
receiveAnalytics()
DBInterface
Variables:
adapter
Methods:
connecttoDB(data)
connecttoWatch(data)
connecttoProfile(data)
authentication(user, pword)
FitnessWatchAdapter
Variables:
adapter
Method:
connecttoWatch(data)
connecttoProfile(data)
authentication(user, pword)