CSC 431

Apollo: A Music Dating and Social Network Application

System Architecture Specification (SAS)

Team 09

Samantha Kamath	Project Manager
Megan Page	Developer
Yunting Zhao	Developer

Version History

Version	Date	Author(s)	Change Comments
1	4/1/2021	Samantha Kamath, Megan Page, Yunting Zhao	
2	4/30/2021	Samantha Kamath, Megan Page, Yunting Zhao	Revised System overview, class diagram, sequence diagram
3	5/5/2021	Samantha Kamath, Megan Page, Yunting Zhao	Revised Sequence and class diagrams

Table of Contents

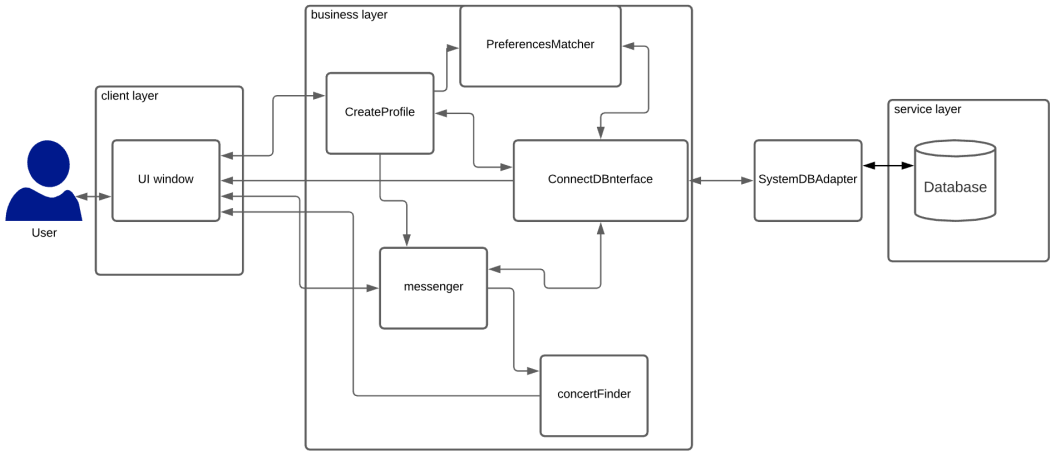
1. System Analysis

1.1 System Overview

This document describes the architecture design and specification of the Apollo Music Dating and Social Network Application. This system uses a three-tier architecture, including a client layer, a business layer, and a service layer. The client server is the IOS client, which accesses the back-end service domain through DNS and provides the user interface. The business layer includes the application logic, including the messaging feature, the upcoming concerts feature, and the compatibility matching. The service layer uses Firebase Database to read and store data; Firebase Database also provides backend services like authentication and security.

The system consists of three main parts. Firstly, there is the UI layer, which is the UI interface the user interacts with. The UI Layer connects directly to the ConnectDBInterface within the business logic layer to login the user.
The business logic layer has components for creating user profiles, matching preferences, and messaging between users. The ConnectDBInterface within this layer is implemented by SystemDBAdapter, which converts information from user Profile objects to JSON data and vice versa; it also communicates with Firebase to verify login credentials and authenticate users.
Lastly, the system uses the service layer, which includes Firebase Realtime Database; this database provides functions such as registration, login, and authentication; the database also stores user information.

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 Apollo.
- Registered Users: Users are individuals who have registered their accounts with Apollo.
- Platform Server: The platform, Firebase, includes a built-in cloud database and authentication services with third-party applications.

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.
- Business Logic Layer: This layer includes the logic for the preferences matcher and suggested upcoming concerts.
- Service Layer: This layer uses Firebase for services like authentication through third-party applications and database storage.

1.4.2 Design Pattern(s)

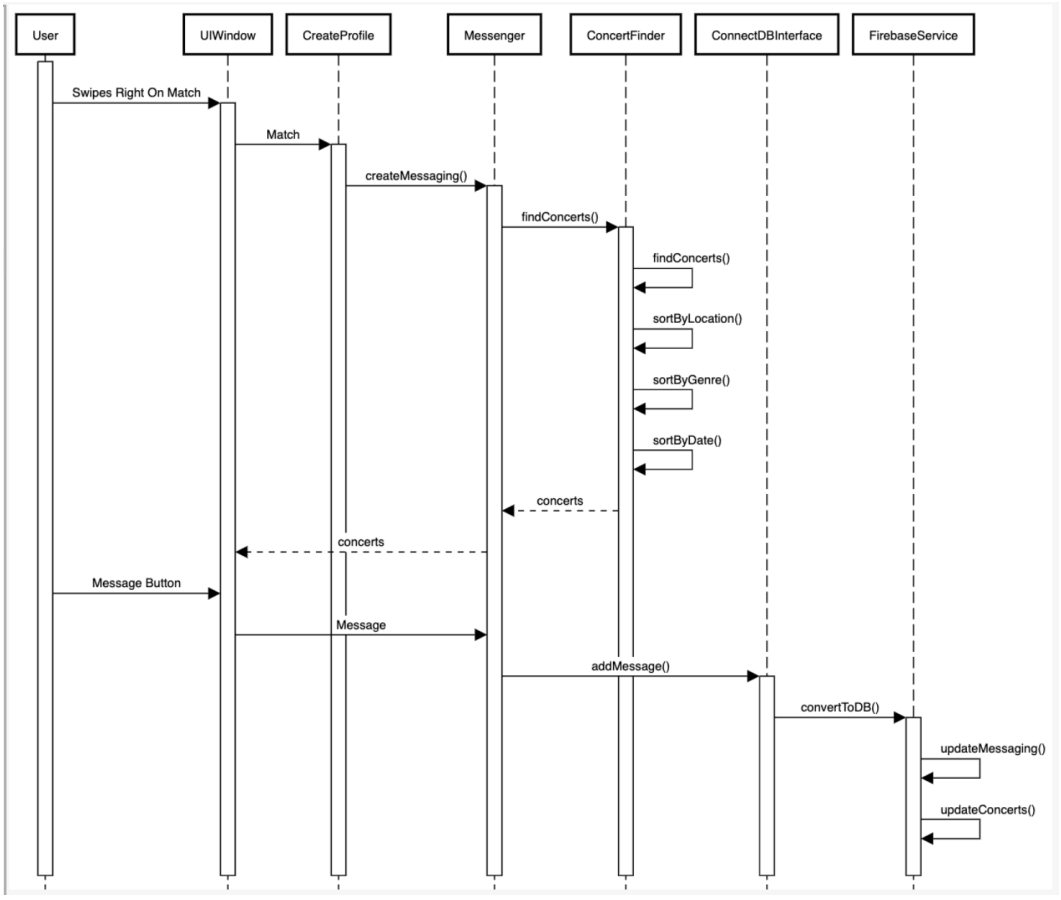
We predict that the Adaptor Design pattern will be primarily used in our project. Adaptor classes will be used to communicate between the business logic layer and service layer. Data from the database in the service layer will be converted to the Profile class in the business logic layer using an adaptor class; similarly, user profiles will be converted to JSON data to add to the database in the service layer using another adaptor class.

1.4.3 Framework

We will be using Flutter for the UI framework; Flutter is well-documented, and developers on our team have previous experience with the framework. However, our app also requires additional backend services, such as authentication, database storage, and business logic to implement compatibility algorithms and the upcoming concerts algorithm. Firebase is a Backend-as-a-Service that can implement authentication, cloud database storage, and security functions for our app with ease. For the other backend logic, such as the app's algorithms, the ExpressJS framework will be used. ExpressJS has access to all the NodeJS packages, and developers on our team have previous experience using JavaScript and ExpressJS.

2. Functional Design

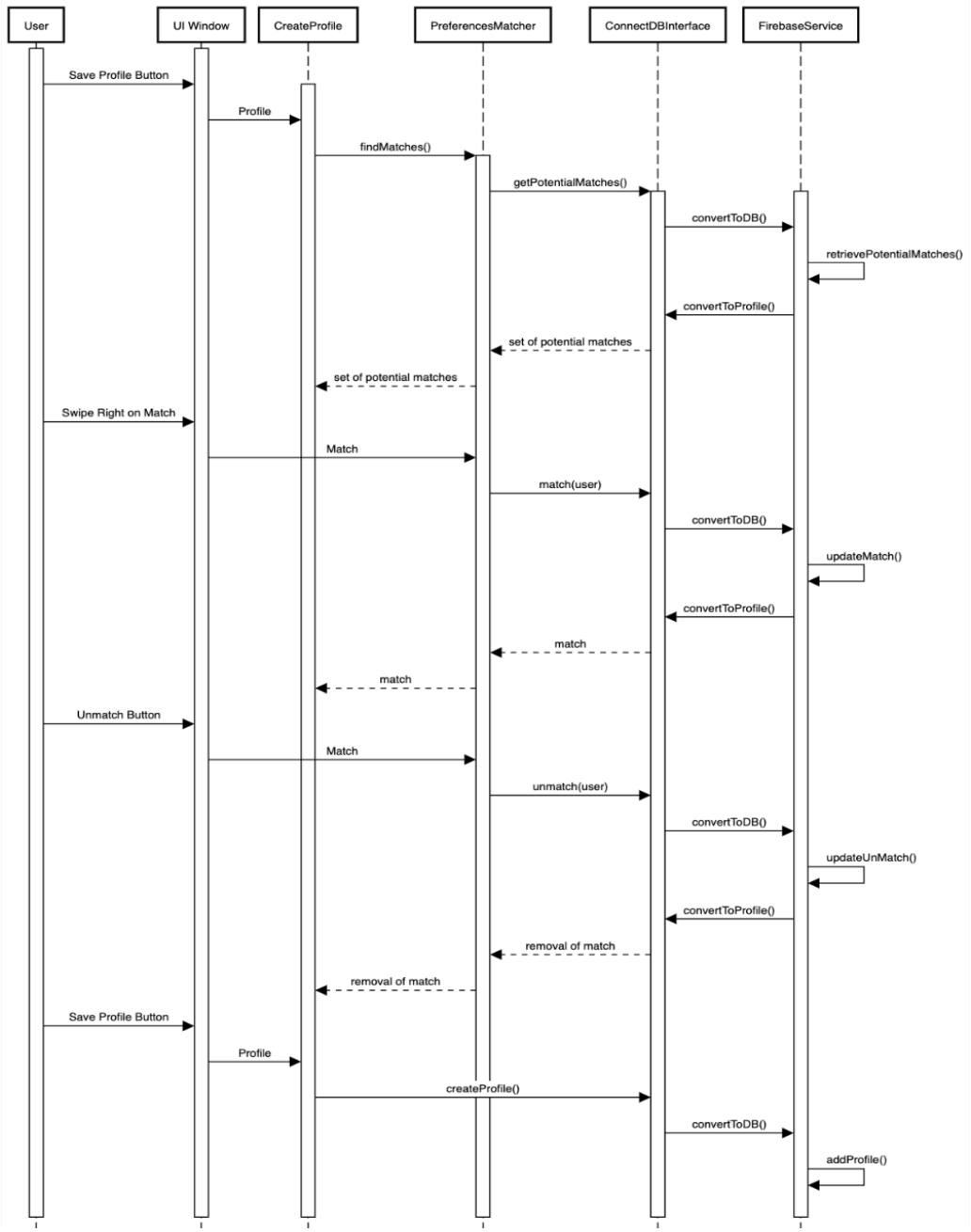
2.1 Messaging Users and Finding Concerts



- When a user matches with a person after swiping right, the UI Window sends the Match information to their CreateProfile class, which stores matches. The CreateProfile class is responsible for calling createMessaging(), and this creates a Messenger class.
- The Messenger class calls findConcerts(), which calls ConcertFinder's findConcerts() and this feature finds Suggested Upcoming Concerts for the match. From here, the user can choose to filter how they want to view the order of concerts, which can be filtered by sortByLocation(), sortByGenre(), or sortByDate(), which sort the list of concerts by location, genre, and concert date, respectively. The concerts are returned to Messenger. Messenger also returns the list of concerts to the UI window for the user to see.
- When the user messages another user via the Message Button, the button communicates with the UIWindow class, which sends a Message object to Messenger. Then, Messenger

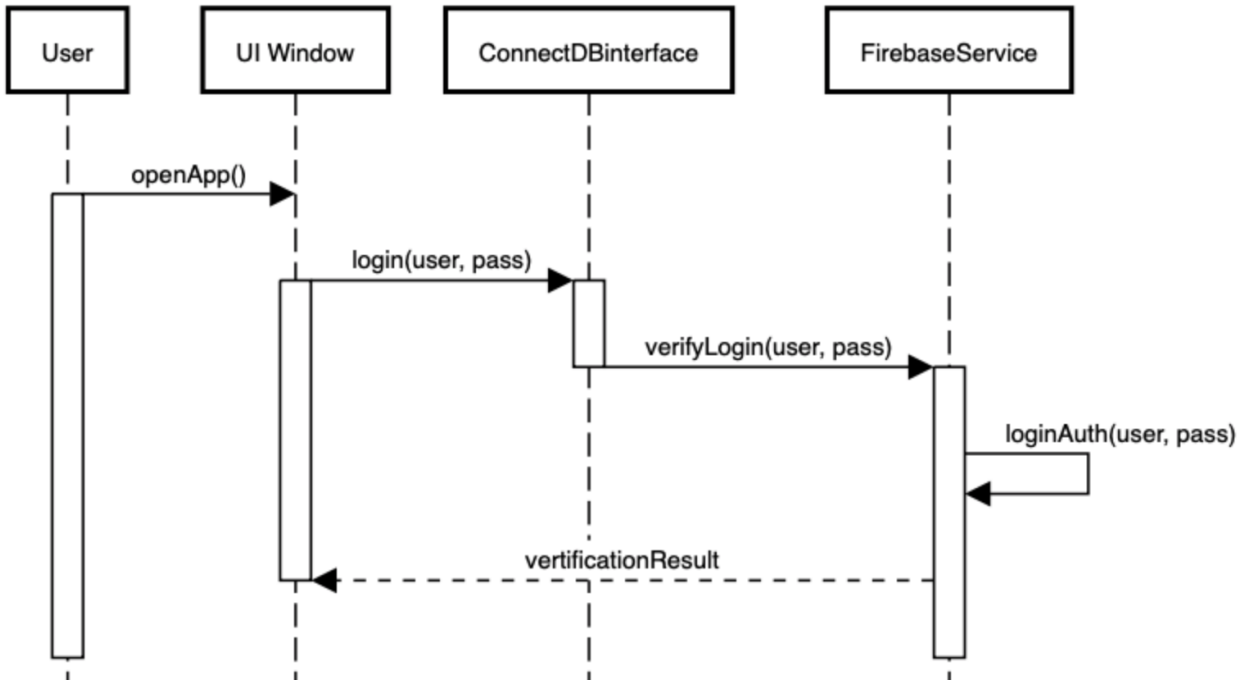
calls addMessage(), which communicates with the ConnectDBInterface class. This class calls convertToDB() to convert the concerts found in the Suggested Upcoming Concerts and the new message information to JSON data. Then, FirebaseService adds this information to the database via updateMessaging() and updateConcerts().

2.2 Finding Potential Matches and Matching Users



- When a user saves their profile, the UI Window sends the Profile information to CreateProfile. The CreateProfile class calls findMatches(), which creates a PreferencesMatcher for that user's profile. This PreferencesMatcher is responsible for all matching functionality, including finding compatible people for potential matches, actually matching users, and unmatching previously matched users.
- PreferencesMatcher calls getPotentialMatches(), which conducts the compatibility algorithm to find potential matches for the user. This function communicates with the database via ConnectDBInterface, which is implemented by our Adapter class. The ConnectDBInterface converts the user's information to JSON via convertToDB(). Then, the FirebaseService retrieves and updates the potential matches for the user through retrievePotentialMatches(), returning the set of potential matches. The potential matches are converted from JSON to a set of Profile objects via convertToProfile(), returning a set of Profiles to the PreferencesMatcher and CreateProfile classes.
- When a user matches with another by swiping right, the UI Window sends the Match object information to PreferencesMatcher. The PreferencesMatcher class calls match(user), which uses ConnectDBInterface to convert the match information to JSON via convertToDB(). Then, FirebaseService updates this match in the user's database information through updateMatch(), and the Match is converted back to a Match object via convertToProfile(). The Match is returned to the user's PreferencesMatcher and CreateProfile class to add to the user's set of matches.
- When a user unmatches with another, the Match object information to remove is sent to PreferencesMatcher. The PreferencesMatcher class calls unmatch(user), which uses ConnectDBInterface's convertToDB() function to convert the match information to JSON. FirebaseService updates this unmatch in the user's database information through updateUnMatch(). Then, the unmatch is converted back to a Match object via convertToProfile() and returned to the PreferencesMatcher and CreateProfile classes to be removed from the stored set of matches within these classes.
- Additionally, when a user saves their profile, the CreateProfile class also communicates directly with ConnectDBInterface via the function createProfile() to store user information in the database.

2.3 Logging Into with the App



- When the user opens the app and starts interacting with the UI window, it calls openApp().
- When the user enters the login page in the UI window, and logs in with the account and password, the UI window will call login(user, pass), which communicates with ConnectDBinterface.
- Then, ConnectDBinterface will call verifyLogin(user, pass), which communicates with FirebaseService.
- FirebaseService calls loginAuth(user, pass) to verify the user's account and password, and after authentication, the FirebaseService will send the verification result to the UI window.

3. Structural Design