**Front-End Documentation**

Frameworks: Angular, Ionic

Notes:

* Creating a new user sends a verification email. The user can’t login until he/she clicks on the link in the email.
* There are two temporary buttons on the login page that are only meant for development. One button shows the intro page, and the other button bypasses the login page to get to the tabs pages.
* The app is currently using hard-coded data. Only signup and login functions are calling the backend.
* The Profile interface in types.ts may have some unnecessary properties. It is a mix of the data structure in the frontend and backend.
* There are custom theme colors stored in theme/variables.scss with the prefix “--msc”

**Getting Started:**

**Prerequisites:**

* Install Angular
  + > npm install -g @angular/cli
* Install Ionic
  + > npm install -g @ionic/cli

**Run the App:**

* > npm install
* > ionic serve

**Guards:**

**Auth Guard:**

* Restricts access to some pages that require authentication.

**Auto Login Guard:**

* Skips the login page if the user is already authenticated.

**Intro Guard:**

* Skips the intro page if the user has already seen it.

**Pages:**

**Confirm Registration page:**

* The user gets directed to this page when clicking on the link in the verification email.
* On load, a call is made to the backend to confirm the registration. It uses the token passed through the URL params.

**Conversation:**

* This is the page where users can message each other.
* The messaging functionality is not currently working.
* The CometChat API is meant to handle the messaging, but it has not been implemented into this page.

**Create Profile:**

* Page that takes the user through the steps of creating their profile asking for a profile image, first name, last name, gender(male, female, or other, this could be changed to “please specify” rather than “other”), birthday, a biography, and meeting preference(remote, in person, or either).
* This page is not currently being used yet.

**Edit Profile:**

* Page that is responsible for going back and potentially changing the user's current profile that they have. It has the same fields as the create profile page.

**Intro:**

* Page that cues the user to start using the app.

**Login:**

* Page that allows the user to attempt to login to their profile by entering an email address and password. Once the credentials are verified, they are then routed to the home page of the app with their profile active.
* The user can also navigate to the Signup Page from this page.

**Signup:**

* Page that prompts the user to enter a valid email address and password, then specify what type of account you are trying to sign up for: a client or a coach.

**Tabs:**

* Tab 1: This tab shows a queue of other users that the current user can potentially match with by swiping left for Dislike and swiping right for Like. This is what the coach user will see for Tab 1.
* Avatar Map: This tab shows a map with other users that the current user can potentially match with. The user can select their avatar and swipe their card left for Dislike and swipe right for Like. This is what the client user will see for Tab 1.
* Tab 2: This tab is the page where the user can view their current profile. This contains the edit profile button that will route the user to the edit profile page when pressed.
* Tab 3: This tab is the page where the user can see the people that they have matched with. The user can select one of their matches to start messaging them, which will navigate to the Conversation Page.
* Tab 4: This page contains the users account information and allows them to change their email and password along with handling their billing and subscriptions. Includes a logout button.
* Tabs: This is a static bar with the icons of the tabs at the bottom of the screen. This is how the user will navigate throughout the app. Each icon will route the user to that respected tabs page.

**Components:**

**Profile View Component:**

* The profile card that opens when the user selects the profile bubble from the swipe page.

**Services:**

**Api Service:**

* Makes http calls for user profiles.
* Stores current user profile.
* Has our own custom messaging functionalities, but it is no longer being used because we later decided to use a third party API called CometChat.

Methods:

* **getAllProfiles()**
  + Return: Observable<Profile[]>
  + Fetches all the profiles from the database(both coach and client)
* **getMatchedProfiles(email: string)**
  + Return: Observable<Profile[]>
  + Fetches all the profiles that have been matched with the profile with the given email
* **getClientProfile(email: string)**
  + Return: Observable<Profile>
  + Fetches a client profile with the given email
* **getCoachProfile(email: string)**
  + Return: Observable<Profile>
  + Fetches a coach profile with the given email
* **createProfile(profile: Profile)**
  + Return: Observable<Profile>
  + Creates a profile with the given profile object and adds it to the database
* **updateProfile(profile: Profile)**
  + Return: Observable<Profile>
  + Updates the given profile to the database
* **getAllConversations()**
  + Return: Observable<Conversation[]>
  + Fetches all conversations from the database
* **getConversationsForProfile(profileId: number)**
  + Return: Observable<Conversation[]>
  + Fetches all conversations for the given profile id
* **getConversation(id: number)**
  + Return: Observable<Conversation>
  + Fetches a conversation with the given conversation id
* **createConversation(conversation: Conversation)**
  + Return: Observable<Conversation>
  + Creates a conversation with the given conversation object and adds it to the database
* **updateConversation(conversation: Conversation)**
  + Return: Observable<Conversation>
  + Updates the given conversation to the database
* **deleteConversation(id: number)**
  + Return: Observable<Conversation>
  + Deletes a conversation with the given id
* **getMessagesInConversation(conversationId: number)**
  + Return: Observable<Message[]>
  + Fetches the messages for a conversation with the given conversation id
* **createMessage(message: Message)**
  + Return: Observable<Message>
  + Creates a message with the given message object and adds it to the database
* **updateMessage(message: Message)**
  + Return: Observable<Message>
  + Updates the given message to the database
* **deleteMessage(id: number)**
  + Return: Observable<Message>
  + Deletes a message with the given message id

**Auth Service:**

* Handles signup/login functionalities.
* Stores user email and authentication token in local storage.

Methods:

* **loadToken()**
  + Return: Promise<{email: string, authToken: string}>
  + Fetches the email and auth tokens from Storage
* **setEmailToken(email: string)**
  + Return: Promise<void>
  + Sets the email token to Storage
* **setAuthToken(authToken: string)**
  + Return: Promise<void>
  + Sets the auth token to Storage
* **authenticate(credentials: {email: string, password: string})**
  + Return: Observable<any>
  + Fetches the auth token from the backend
* **signUp(credentials: {email: string, password: string, userType: string})**
  + Return: Observable<any>
  + Creates a new user with the given credentials and user type (client or coach). A confirmation link is sent to the given email.
* **login(credentials: {email: string, password: string, userType: string})**
  + Return: Observable<ClientDto>
  + Get the profile for the given credentials
* **confirmRegistration(token: string)**
  + Return: Observable<any>
  + Confirms user’s email to the backend. This method is called when the user clicks on the confirmation link in their email

**CometChat Service:**

* Contains functions for calling the CometChat API.
* Used for messaging between users.
* Not implemented into the app yet.

Methods:

* **constructor()**
  + Initialize CometChat
* **registerUser(uid: string | number, name: string)**
  + Return: Promise<CometChat.User>
  + Registers the user to CometChat
* **loginUser(uid: string | number)**
  + Return: Promise<CometChat.User>
  + Login the user to CometChat
* **logoutUser()**
  + Return: Promise<Object>
  + Logout the user from CometChat
* **getCurrentUser()**
  + Return: Promise<CometChat.User>
  + Fetches the current logged-in CometChat user
* **getUser(uid: string | number)**
  + Return: Promise<CometChat.User>
  + Fetches the CometChat user with the given uid
* **sendMessage(receiverId: string | number, message: string)**
  + Return: Promise<CometChat.BaseMessage>
  + Sends a message to the given user id
* **addMesssageListener(uid: string | number)**
  + Return: void
  + Adds a message listener for the given uid. It can run a function when it receives a message
* **removeMessageListener(uid: string | number)**
  + Return: void
  + Removes the message listener for the given uid
* **getMissedMessages(uid: string | number)**
  + Return: Promise<CometChat.BaseMessage[]>
  + Fetches the last 30 missed messages for the given uid
* **getMessageHistory(uid: string | number)**
  + Return: Promise<CometChat.BaseMessage[]>
  + Fetches the last 30 messages for the given uid
* **getConversations()**
  + Return: Promise<CometChat.Converstaion[]>
  + Fetches the last 30 conversations for the current user
* **getConversation(uid: string | number)**
  + Return: Promise<CometChat.Conversation>
  + Fetches the conversation with the given uid

**Potential Match Service:**

* Fetch profiles for the user to swipe.
* Handles swipe functionality.

Methods:

* **getClientMatches(email: string)**
  + Return: void
  + Fetches the potential matches for the given client email
* **getCoachMatches(id: number)**
  + Return: void
  + Fetches the potential matches for the given coach id
* **swipeLike(swipedProfile: Profile)**
  + Return: void
  + Updates the status to “Like” for the given profile
* **swipeDislike(swipedProfile: Profile)**
  + Return: void
  + Updates the status to “Dislike” for the given profile