**Software Requirements**

**Specification**

**for**

OurWorkout

**Version 1.0 approved**

**Prepared by SZ2440 (Group 1)**

**Members: Pravind Kummar, Ryan Ong, Tony Tan, Tan Li Ying, Subeen, Adil Hasan**

**NTU SC2006**

**Last Updated: 5th November 2022**

# Content Page

[**Content Page**](#_i66b3p5quo93) **2**

[**1. Introduction**](#_bjdnprvs8w7a) **3**

[a. Project Mission Statement](#_wlk2ct8nuwpg) 3

[b. Product Scope](#_ifh86xsqvs2x) 4

[c. Assumptions](#_kowwqgefeopn) 4

[d. Constraints](#_fw9qxd36ijps) 4

[e. Initial UI Mockups](#_xgljshjss5b) 4

[**2. Functional Requirements Elicitation**](#_e4zc1wc3cpek) **9**

[a. Functional Requirements](#_f9xpz4ihdfdu) 9

[b. Use Case Diagrams](#_d41bz2w1rswa) 17

[c. Use Case Descriptions](#_j03lrrdv4kee) 20

[UA - User Access Functions](#_t1znvbao9h5p) 21

[SO - Social Interaction Functions](#_j29fppq5ccnn) 32

[GP - Gym Page Functions](#_rxsc5rn44ct4) 39

[WL - Workout log Functions](#_xuxvcia39u0w) 44

[e. Sequence Diagrams](#_1o6rliwgks5u) 50

[Use Case 1: Login via Email](#_aoh8cyxlf1hv) 50

[Use Case 2: Register new Account via Email](#_ozfiqt8dseub) 51

[Use Case 3: Change Password](#_lk2cdyekkrpl) 51

[Use Case 4: Edit Account](#_jqcqx71vfumb) 52

[Use Case 5: Searching for Gyms / Users / Workouts](#_j285hligt4tz) 53

[Use Case 6: Show all chats](#_eandm75k7p3v) 54

[Use Case 7: Create Chat](#_bzm8q06trwj) 55

[Use Case 8: Open Gym Page](#_7e1y2xcyd14a) 56

[f. Dialog Map](#_k49hqryw250v) 57

[**3. Non-Functional Requirements Elicitation**](#_kkmf77bcso88) **58**

[**4. Interface Requirements**](#_xeeyrb9ru5b) **59**

[a. User Interfaces](#_lygfmkatiuv0) 59

[b. Hardware Interfaces](#_olzykbrfm73) 59

[c. Software Interfaces](#_4b0vz1ypfxm4) 59

[d. Communication Interfaces](#_r7vm5g3ej7se) 59

[**5. Data Dictionary**](#_l7tf1euc29v5) **60**

[**6. System Architecture**](#_nyr09c4od3i2) **69**

[**7. Key design issues and patterns**](#_iwe11mwtneak) **70**

[Design issues](#_3dc06ije6w4s) 70

[Identifying and storing persistent data](#_rd7dp9s4oonl) 70

[Providing access control](#_dj6austkvq1k) 70

[Design patterns used](#_ezl2k8wej6gv) 70

[**8. Testing**](#_9hgerzgqnxq) **75**

[**9. Good Software Engineering Practices**](#_91hcv4bcetgr) **88**

# 

# 1. Introduction

## a. Project Mission Statement

Nearly 4 in 10 Singaporeans have gained weight since the pandemic, in a country which already suffers from high and rising rates of Type 2 diabetes. According to a study done, 43% of people expect to give up their new year’s fitness resolutions by February. The main reason that they give is because they go to the gym alone and hence lose motivation over time. We believe there is a strong need to not only encourage Singaporeans in leading a physically active lifestyle but also to create a community to motivate other Singaporeans to exercise.

Our software product, OurWorkout, is specially designed for Singapore’s context and, among other functionalities, will seek to facilitate community-building amongst gym-goers. It will host information on a wide variety of exercise routines and the many gyms found in Singapore and it will allow users to track and share their own workout experiences over time. In this way, it will help with the common issues of people lacking guidance or gym mates – users can chat with each other, publicly discuss matters regarding their gym, and so on, as well as writing up and sharing workout logs of their workout activities in a manner which integrates the data from our datasets.

## b. Product Scope

OurWorkout allows user to search for nearby gyms. Information about the gym such as gym location and opening times is also provided upon choosing a gym. Additionally, users will be able to create workout logs to track their past workout so as to motivate themselves to achieve better workouts. Moreover, users are able search for new workouts that they might want to try out.

In addition, users can create an account and will be able chat with other users. With an account, users are able to collaborate in gym discussion threads to ask for advice or for motivation to improve.

## c. Assumptions

· Users should have access to the internet and have a working phone.

· Users should have a valid email address.

· The portions of the Android and iOS system APIs involved in development of our application will not change substantially

## d. Constraints

The API used from data.gov.sg regarding Gyms@SG does not provide information for all the gyms in Singapore, hence limiting the extensiveness of the gym searching capabilities.

## e. Initial UI Mockups

| 1. Login Page | 2. Registration Page |
| --- | --- |

| 3. Main Page | 4. All Chat Page |
| --- | --- |

| 5. Chat Page | 6. Profile Page |
| --- | --- |

| 7. Search Screen | 8. Gym Discussion Screen |
| --- | --- |

# 2. Functional Requirements Elicitation

## a. Functional Requirements

1. In order to **make use** of the **app, users** must **authenticate** themselves.
   1. The act of authenticating is also referred to as **logging in.**
   2. The **unauthenticated user** will be able to **register** an **account**.
   3. The **unauthenticated user** will be able to **authenticate**.
2. Users must be able to authenticate themselves via **email authentication** (elaborated below).
   1. Upon successful email authentication, a visual indication must be provided in the UI to communicate that the user is now an authenticated user.
   2. Upon successful email authentication, a visual indication must be provided in the UI to communicate that the user is now able to access those functionalities indicated in the requirements as requiring a user to be an authenticated user.
   3. An **email address** must be entered in order to authenticate via email authentication.
      1. Only email addresses fitting the email address criteria for registration in Section 3.a.X is valid for the user to enter.
      2. The email address associated with the user’s account must be a **verified email address** for the user to successfully authenticate**.**
   4. A **password** must be entered in order to authenticate via email authentication.
      1. Only less than or equal to 30 characters in length and longer than 0 characters is valid for the user to enter herein.
   5. All emails and passwords fitting these criteria are considered “valid”.
   6. To successfully authenticate, the email address provided and the password provided must exactly match the email address and the password which have been previously associated with the same user account.
   7. If and only if the email and password are valid and 2.f is successful does successful authentication occur.
   8. Upon successful authentication, the user is authenticated as the user owning the user account to which the email address and the password provided were previously associated.
   9. If a valid email is provided but the user is unable to authenticate/log in, an error message is displayed.
      1. This error message must state the reason why authentication was unsuccessful, i.e. an email address which does not match records, the account lacking a verified email address, or an incorrect password.
   10. If a valid password is provided but the user is unable to authenticate/log in, an error message is displayed.
       1. This error message must state the reason why authentication was unsuccessful, i.e. a password which does not match records.
   11. If an invalid password is provided, an error message is displayed.
       1. This error message must state the case of invalidity, i.e. that it is too short or two long.
       2. This error message must state the conditions for validity, i.e. that it must be at least 1 character long and at most 30 characters long.
   12. If an invalid email address is provided, an error message is displayed.
       1. This error message must state the cause of invalidity, i.e. that it is too short or two long
       2. This error message must state the conditions for validity, i.e. that it must be at least 7 characters long and at most 100 characters long.
3. Unauthenticated users must be able to **register accounts**.
   1. In order to register an account, a valid **email address** must be provided.
      1. The email address must not be exactly equivalent to an email address already **associated with an account**.
      2. The email address must be at least 7 characters in length.
      3. The email address must be at most 100 characters long.
      4. The email address must contain exactly 1 ‘@’ symbol.
         1. Not 0 ‘@’ symbols
         2. Not more than 1 ‘@’ symbol
      5. The ‘@’ must not be the first or the last character in the email address.
      6. The portion of the email address after the first (and only) ‘@’ symbol is termed the **domain name**
      7. The email address must contain at least 1 ‘.’ (period) symbol **in the domain name**.
         1. Not 0 ‘.’ (period) symbols.
      8. The period cannot be the first symbol in the domain name.
      9. The period cannot be the last symbol in the domain name.
   2. In order to register an account, a **first name** must be provided.
      1. This first name must match the following format
         1. The first name must be able to contain at most up to 300 characters.
      2. All first names exactly matching the above specification must be permitted.
   3. In order to register an account, a **last name** must be provided.
      1. This last name must match the following format
         1. It must contain at least one character.
         2. The last name must be able to contain at most up to 300 characters.
      2. All last names exactly matching the above specification must be permitted.
   4. In order to register an account, a **password** must be provided.
      1. The user must input a password with at least 1 non-digit character.
         1. Here, digit character is defined as the digits 0-9 under UTF-8 encoded ASCII.
      2. The user must input a password with at least 1 digit character.
      3. The user must input a password which is at least 8 characters long.
   5. If any of the above conditions in 3.X are not satisfied
      1. The app must indicate that the values entered are invalid.
      2. The app must inform the user of at least one of the unmet requirements.
         1. The app must inform the user of at least one of the unmet requirements by displaying suitable text.
      3. The app must not register the user.
   6. During account registration, a **profile picture** may be provided.
      1. The profile picture must be of a supported format.
      2. If any requirements the app imposes on profile pictures are not satisfied
         1. The app must indicate that the values entered are invalid.
         2. The app must inform the user of at least one of the unmet requirements.
         3. The app must not register the user.
      3. Upon registration, any desired image encoding may be used to store the profile photograph by the app.
      4. If no profile photo is provided, the app will instead associate a **placeholder image** with the user’s account.
         1. This placeholder image is then displayed in all instances where the user’s profile photograph would normally be displayed.
   7. Upon registration of the account, the email address is considered to be unverified.
      1. A **verification email** is sent to the email address, containing a **verification URL**
      2. Upon the user accessing the verification URL in a web browser, the email address is considered to be a verified email address
4. An authenticated user must be able to view any other **user’s details** through the app UI.
   1. Only an authenticated user will be able to view any other user’s details through the app UI.
   2. It must not be possible for any user to view the email address of another user through the app UI.
   3. The profile must show:
      1. The first name of the other user.
      2. The last name of the other user/
      3. The profile photograph of the other user.
5. An authenticated user must be able to create **chats** with any other authenticated user they are permitted to chat with.
   1. It is sufficient for all authenticated users to be permitted to create chats with any other user.
   2. In order to create a chat with another user, the user creating the chat must send one initial chat message which will become the first message in the chat.
      1. This chat message must comply with Subsection 5.d.i and all parts thereof.
   3. If a user has created a chat with another user, they and the other user are considered to be **chat members** of the chat created.
   4. If an authenticated user is a member of a chat, they must be able to send **chat messages** in the chat, which means with compliance with Subsection 5.d.ii and the points under it..
      1. To send a chat message, the user must enter a **message text**.
         1. To enter message text.
            1. The message text must be at least 1 character long.
            2. The message text must be at least as many characters as the **maximum message text length**.
            3. If any of the above requirements from 5.c.i.X are not met and the user attempts to send a message, the app must inform the user of at least one of the unmet requirements via displaying text to that effect.
            4. If any of the above requirements from 5.c.i.X are not met and the user attempts to send a message, the app must not allow the user to send the message.
      2. The chat messages must be viewable by any other authenticated user who is a chat member of the chat.
         1. To be judged viewable, the following information regarding the chat message must be seen and visible to by the authenticated user who is “viewing”.
            1. The name of the user who sent the chat message.
            2. The message text.
         2. All chat messages in a chat must be sorted in ascending chronological order of sending.
            1. This means that when displayed, the chat message sent earlier must appear before the chat message sent later.
   5. While a user is viewing a chat, if a user sends a message in the chat, the user viewing the chat must eventually see the new message without having to engage in any UI interaction.
   6. While a user is viewing a chat, if a user sends a message in the chat, the user viewing the chat must eventually see the new message without having to engage in any UI interaction.
   7. An authenticated user must be able to view a list of all chats they are members of.
   8. An unauthenticated user must not be able to perform any of the below actions.
      1. An unauthenticated user must not be able to create chats.
      2. An unauthenticated user must not be able to view chats.
      3. An unauthenticated user must not be able to send chat messages.
      4. An unauthenticated user must not be able to view chat messages.
6. An authenticated user must be able to view the **locations** of **gyms** near him or her. What this means is elaborated in the below points unambiguously.
   1. There must be a sequence of UI interactions, i.e. taps, swipes, or button presses, which results in the “**gyms map display**” being displayed to the user.
   2. This gyms map display must display the locations of all gyms known to the app.
   3. The gyms shown must include all of the gym locations found in the **Data.gov.sg dataset.**
   4. This gyms map display must be pannable with one finger.
      1. User must be able to pan vertically.
      2. User must be able to pan horizontally.
      3. User must be able to pan both vertically and horizontally in one straight-line movement of the finger with which they are panning.
   5. This gyms map display must be zoomable with two fingers.
      1. User must be able to zoom in.
      2. User must be able to zoom out.
   6. Interacting with elements of this gyms map display must bring up the relevant **gym page**, as described in the next top-level section.
   7. An unauthenticated user must not be able to conduct any activities in this above section (Section 6).
7. Each gym in the **Data.gov.sg dataset** must have a “gym page” accessible through some sequence of UI interactions
   1. This gym page must display the name of the gym, as recorded in the **Data.gov.sg dataset**.
   2. This gym page must display the location of the gym, as recorded in the **Data.gov.sg dataset**.
   3. This gym page must allow any authenticated users to create “**posts**”.
      1. Creating a post must require that the user enter either **post text** or a **post image.**
         1. The post text entered must satisfy the following conditions.
            1. The post text must not be empty.
            2. The post text must have fewer characters than the **maximum post text length**.
         2. The post image is to be chosen from the user’s device storage.
         3. The post image must be of a supported format.
      2. If any of the above conditions from section 7.c.i.X are not satisfied
         1. The app must indicate that the post content is invalid.
         2. The app must inform the user of at least one of the unmet requirements.
            1. The app must inform the user of at least one of the unmet requirements by displaying suitable text.
         3. The app must not create the post
      3. If the conditions from section 7.c.i.X are all satisfied, the app must permit creation of the post.
      4. The maximum post text length permitted by the app must be at least 300 characters.
   4. Any authenticated users must be able to view any posts created on any given gym page.
      1. When viewed, a post must display the **full name** of the **post author**.
      2. When viewed, a post must display the profile picture of the post author, if any.
         1. Otherwise, a placeholder image must be displayed.
      3. When viewed, a post must display the **post text**, if any.
      4. When viewed, a post must display the **post image**, if any.
      5. All posts on a gym page must be sorted in ascending chronological order of sending.
   5. If a user is viewing a gym page, if another user then creates a post in the gym page, the user who is already viewing the gym page must eventually see the new post without having to engage in any UI interaction with the gym page.
   6. No authenticated users can view gym pages.
   7. Only authenticated users can post on gym pages
8. Each **exercise** listed in the **WGER Exercise Database** must have an “**exercise page**” accessible through some sequence of UI interactions.
   1. Each exercise page must display the **exercise image** of the exercise.
   2. Each exercise page must display the **exercise description** of the corresponding exercise.
   3. Only authenticated users can view exercise pages.
9. Each authenticated user is associated with a **personal workout log**.
   1. An authenticated user must be able to access their own personal workout log through the app UI.
   2. An authenticated user must be able to add **workout log entries** to their own **workout log**.
      1. In this document, the workout log is considered to “contain” or “own” every workout log entry the authenticated user has written.
      2. A workout log entry must record an associated exercise.
         1. The app UI must allow the user to select the exercise from a list which is displayed to them.
      3. A workout log entry must record associated **workout log notes.**
         1. The app UI must provide a text box for entering the workout log notes.
            1. The text entered must be at least 1 character long.
            2. The text entered must be fewer characters than the **maximum workout log notes length**.
         2. The **maximum workout log notes length** must be at least 1000.
      4. A workout log entry must record an associated **intensity.**
         1. The intensity must be entered by the user.
            1. The text entered must be in number format.
      5. If any of the above requirements for **user-entered data** from section 9.b.X are not satisfied:
         1. The app must indicate that the entered data is invalid.
         2. The app must inform the user of at least one of the unmet requirements by displaying suitable text.
         3. The app must not create the workout log entry.
   3. Accessing a workout log must allow the user to view all of the workout log entries contained by the workout log through the app UI.
      1. When viewing a workout log entry, the following must be displayed.
         1. The name of the associated exercise.
         2. The corresponding workout log notes.
         3. The corresponding intensity.
         4. The date of the associated exercise
   4. No user can be allowed to add workout log entries to any workout log besides their own.
   5. A user must not be able to access any other personal workout logs besides those listed in 9.a and 9.b.
   6. No unauthenticated user may access a personal workout log.
10. The **app** must provide the below **search functionality**.
    1. It must be possible to search **searchable items** using their **searchable attribute**.
    2. The searchable attribute is considered as text for the purpose of searching.
    3. User accounts are a searchable item and their searchable attribute is their full name.
    4. Gyms as found in the Data.gov.sg dataset are a searchable item and their searchable attribute is their name as found in the Data.gov.sg dataset.
    5. Exercises as found in the WGER Exercise Database are a searchable item and their searchable attribute is their name as found in the WGER Exercise Database.
    6. In order to search, the user must be able to provide **search text**.
    7. The search text can be up to 300 characters long.
    8. The app UI must not allow the user to input more than 300 characters as the search text.
    9. **Searching** displays **search results** corresponding to those searchable items for which the search text is a prefix of the searchable attribute of the searchable item.
    10. The ordering of the displayed search results must be in alphabetical order of the searchable attribute.
    11. Appropriate search details will be displayed and user must be able to click on that search result to display the relevant page.
11. Each gym page in the app must have a button to “**favourite**” the gym.
    1. Pressing the button to favourite a gym causes the app to record the gym as a gym the user has “favourited”.
    2. The user must be able to view a list of all gyms they have favorited.
       1. The user must be able to go to the gym page of an item in this list via a UI interaction with the item in the list
    3. It must be possible for a user to **unfavourite** a gym through the app UI.
       1. This causes the app to record the gym as no longer favourited by the user.
    4. Only authenticated users are able to favourite or unfavourite gyms.
    5. No unauthenticated user can view the list of gyms they have favourited.
12. All **app data** and **user data** of the app must be persisted after any of the below listed events, meaning that it should still be viewable through the app UI by an authenticated user who is required by other clauses in this requirements document to be able to view the data.
    1. **App shutdown**
13. Unless specified otherwise, all user-entered text in the app may be arbitrary **UTF-8 characters.**

## 

## b. Use Case Diagrams

## 

## 

## c. Use Case Descriptions

| USE CASE ID | USE CASE NAME |
| --- | --- |
| **UA - User Access Functions** | |
| UA.1 | Register account |
| UA.2 | Login |
| UA.3 | Reset password |
| UA.4 | Edit personal details |
| UA.5 | Edit user settings |
| UA.6 | Logout |
| **SO - Social Interaction Functions** | |
| SI.1 | Find other user(s) |
| SI.2 | Create User chat |
| SI.3 | Open chat |
| SI.4 | Delete chat |
| **GP - Gym Page Functions** | |
| GP.1 | Open gym page |
| GP.2 | Like/Unlike gym |
| GP.3 | Open gym chat/discussion |
| **WL - Workout log Functions** | |
| WL.1 | Add workout log |
| WL.2 | Edit workout log |
| WL.3 | Delete workout log |

### **UA - User Access Functions**

| Use Case ID: | UA.1 | | |
| --- | --- | --- | --- |
| Use Case Name: | Register account | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User/Guest (Initiating Actor), Database | |
| --- | --- | --- |
| Description: | User creates an account to access system functionalities. User can choose to authenticate themselves via email, Google | |
| Preconditions: | 1. The user has a personal email | |
| Postconditions: | 1. The user successfully created an account | |
| Priority: | High | |
| Frequency of Use: | Once per user | |
| Flow of Events: | Actor steps | System steps |
| 1. The user proceeds to register by clicking on the “Register new account” option. | 1. The system directs the user to the registration page to create a new account. 2. The system requests the user to fill in the required fields such as a valid email, password, confirm password and personal information including their full name. |
| 1. The user enters their personal email, account password, and the remaining information and clicks submit.    1. Must fulfil password requirements       1. The password must be at least 20 characters long up to 100 characters | 1. The system transfers the information to validate the account. 2. The system creates the user account once confirmation is received and stores it in the database. |
| Alternative Flows: | UA.1.AC.1: If the user enters an invalid email address to register.   1. The system displays a message “Please enter a valid email address”. 2. The system returns to step 4.   UA.1.AC.2: If the password does not meet the specified requirement.   1. The system displays a message “The password must be at least 20 characters long up to 100 characters and must not contain as a subsequence the user’s name or email address” 2. The system returns to step 4.   UA.1.AC.3: If the user enters an email address that already exists in the database.   1. The system displays a message “User account already exists, please enter a different email address or login”. 2. The system returns to step 4. | |
| Exceptions: | UA.1.EX.1 User aborts registration     1. User aborts the registration using the “returnl” button 2. System does not save any user details and returns to login screen   UA.1.EX.1 User already registered   1. In step 5 system found that user already exist, system will not allow further registration of the user 2. User may return to login screen and login following use case UA.2 | |
| Includes: | Authenticate email, Check existing user | |
| Special Requirements: | 1. System does not collect information which violates the PDPA 2. Password must be stored securely in salted and hashed form | |
| Assumptions: | - | |
| Notes and Issues: | - | |

| Use Case ID: | UA.2 | | |
| --- | --- | --- | --- |
| Use Case Name: | Login | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan O |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User | |
| --- | --- | --- |
| Description: | User proceeds to log into a registered account to access system functionalities on their local device. | |
| Preconditions: | 1. User must have an existing account | |
| Postconditions: | 1. System will grant access to system functionalities 2. System will transfer user to the home page | |
| Priority: | High | |
| Frequency of Use: | Medium | |
| Flow of Events: | Actor steps | System steps |
| 1. User types in his/ her email and password and presses login | 1. System checks if both email and password fields are typed in. 2. System authenticate login details and transfer user to home page |
| Alternative Flows: | UA.2.AC.1: User types in an invalid email address that does not exist in the database.   1. System shall prompt the user to register a new account using UA.1 before logging in again.   UA.2.AC.1: User types in an invalid password   1. System shall prompt user to type in the password again or reset his/ her password. 2. User may reset password following use case UA.3 before attempting to login again | |
| Exceptions: | - | |
| Includes: | - | |
| Special Requirements: | - | |
| Assumptions: | Users must be connected to the internet. | |
| Notes and Issues: | - | |

| Use Case ID: | UA.3 | | |
| --- | --- | --- | --- |
| Use Case Name: | Reset password | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User, System |
| --- | --- |
| Description: | Users may choose to reset their password in the event they forgot their password. |
| Preconditions: | 1. User must be registered using email authentication |
| Postconditions: | 1. System auto-generates a password to be set to the user account and saved to the cloud |
| Priority: | Medium |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User access application site login page 2. User selects “Reset Password”    1. System will transfer user to password reset page 3. User enters registered email 4. System verifies that the email is registered 5. System generates a password 6. System sends the generated password to the registered email |
| Alternative Flows: | - |
| Exceptions: | UA.2.EX.1 Unidentified account   1. In step 4, system does not find an existing account 2. System will not run step 5-6   UA.2.EX.2 User aborts password reset   1. User aborts password reset using the “cancel” button 2. System returns user to the user login page |
| Includes: | - |
| Special Requirements: | - |
| Assumptions: | 1. Users do not have access to other users email accounts |
| Notes and Issues: | Not yet implemented |

| Use Case ID: | UA.4 | | |
| --- | --- | --- | --- |
| Use Case Name: | Edit personal details | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | Users may change their personal details over time. They can view their current details and make changes to it via the profile page. |
| Preconditions: | 1. User must be registered and logged in |
| Postconditions: | 1. System saves the changes made to user details in the cloud |
| Priority: | Medium |
| Frequency of Use: | Low |
| Flow of Events: | 1. User navigates the profile page 2. User selects the “Edit” button 3. User modifies the relevant details 4. User selects the “Save” button at the bottom of the page 5. System will record the changes made to the user account |
| Alternative Flows: | - |
| Exceptions: | UA.4.EX.1 Change of user email   1. User must contact app developer 2. User must provide a valid new email 3. App developer will verify the email 4. App developer will change user email upon request |
| Includes: | Login |
| Special Requirements: | 1. User email cannot be modified through the UI 2. User may leave a field empty to delete the relevant field |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | UA.5 | | |
| --- | --- | --- | --- |
| Use Case Name: | Edit user settings | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | User may wish to change their notification settings or remove users from the blocklist. Users can view their current settings and blocklist, and make changes to it via the settings page.  User may also change their registered password if they registered via email authentication. |
| Preconditions: | 1. User must be registered and logged in |
| Postconditions: | 1. System saves the changes made to the settings, blocklist, and/or password to the cloud |
| Priority: | Low |
| Frequency of Use: | Low |
| Flow of Events: | 1. User navigates to setting page 2. User selects the “Edit” button 3. User modifies the relevant settings    1. Notification preferences    2. Privacy preferences    3. Remove users from blocklist    4. Change password 4. User selects the “Save” button at the bottom of the page |
| Alternative Flows: | UA.5.AC.1 Change of password via reset   1. User may reset their password via password reset on login page following use case UA.3 2. Reset password is randomly generated |
| Exceptions: | UA.5.EX.1 Cancel changes   1. User may choose to cancel edits after step 2 by selecting the “Cancel” button at the bottom of the page 2. System will bring user back to setting page |
| Includes: | Login |
| Special Requirements: | 1. If a password change is made    1. New password must fit password requirements    2. Password must be stored securely in salted and hashed form 2. If user choose to turn off “share my private details”    1. No other users can view the user’s details |
| Assumptions: | 1. Users do not have access to other users account |
| Notes and Issues: | Not yet implemented |

| Use Case ID: | UA.6 | | |
| --- | --- | --- | --- |
| Use Case Name: | Logout | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | User may wish to logout of the application, removing access to system functionalities until next login. |
| Preconditions: | 1. User must be registered and logged in |
| Postconditions: | 1. System will clear local data 2. System will lock access to system functionalities 3. System will transfer user to login page |
| Priority: | Medium |
| Frequency of Use: | Low |
| Flow of Events: | 1. User navigates to setting page 2. User selects the “Signout” button at the end of the page |
| Alternative Flows: | UA.6.AC.1 Clear cache   1. User may clear the app cache on the device 2. User login will be cleared and user will be logged out |
| Exceptions: | - |
| Includes: | Login |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

### **SO - Social Interaction Functions**

| Use Case ID: | SI.1 | | |
| --- | --- | --- | --- |
| Use Case Name: | Find other user(s) | | |
| Created By: | Tony | Last Updated By: | Ryan Ong |
| Date Created: | 7/9/2022 | Date Last Updated: | 20/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | The “search” page will show the user a list of users with similar tags and age they could message to work out together.  A registered user will be able to access the “search” page and be able to search up other users by their profile names and tags and register them as friends.  A registered user will be able to look at other users' profile pages after searching for them. |
| Preconditions: | 1. User must be logged in to the application. |
| Postconditions: | - |
| Priority: | High |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User taps on “search” button (magnifying glass icon) 2. User is shown a list of users with similar tags and age group    1. User may tap on the search bar and enters the desired profile name    2. User is shown a list of users with similar names to what was entered, with the topmost user being the most relevant match. 3. User taps on the desired user 4. The selected user’s profile page will be shown    1. Users may:       1. Start chat with the selected user |
| Alternative Flows: | - |
| Exceptions: | SI.1.EX.1 No search result   1. If user uses the search bar and enters an input which does not match any user’s name 2. Search page will show an empty list 3. User must clear search entry to view list of users again |
| Includes: | - |
| Special Requirements: | 1. If the selected user is blocked, the blocked user cannot view the blocking user’s details    1. User cannot be added as friend    2. Chat with the user cannot be created |
| Assumptions: | - |
| Notes and Issues: | Blocking feature not implemented  Chat via user page not implemented |

| Use Case ID: | SI.2 | | |
| --- | --- | --- | --- |
| Use Case Name: | Create User Chat | | |
| Created By: | Tony | Last Updated By: | Ryan Ong |
| Date Created: | 7/9/2022 | Date Last Updated: | 20/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | The “chat” button (message bubble icon) will show a list of other users the app user had sent or received messages from. The list will be sorted according to the most recent message sent or received.  To create a private chat, tap on any of the listed users and send the very first message to them. |
| Preconditions: | 1. User must be logged in. |
| Postconditions: | 1. Chat details are saved on the system cloud |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | 1. User searches for the person that they want to chat to in the search option 2. In the person’s page, user presses the chat button 3. User is able to send messages to that other user |
| Alternative Flows: | SI.2.AC.1 Create chat via user page   1. User may find a selected user following use case SI.1 2. User may create a chat with the selected user by selecting the “Start chat” button on the user page |
| Exceptions: | - |
| Includes: | Login, Find other user(s) |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | SI.3 | | |
| --- | --- | --- | --- |
| Use Case Name: | Open Chat | | |
| Created By: | Tony | Last Updated By: | Ryan Ong |
| Date Created: | 8/9/2022 | Date Last Updated: | 20/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | A user will be able to open any chat with other users or groups by tapping on the list of active chats displayed in the “chat” page. |
| Preconditions: | 1. User must be logged in to the application 2. An active chat must have been created |
| Postconditions: | 1. Chat history page will be brought up 2. Chat history will be backed up on the system cloud |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | 1. User selects the chat button 2. A list of active chats with other users are brought up 3. User clicks on any of the listed chats 4. The chat page is brought up with previous chat history    1. Users may send a message by selecting the text box |
| Alternative Flows: | - |
| Exceptions: | - |
| Includes: | Login, Create Chat, View list of active chats |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | SI.4 | | |
| --- | --- | --- | --- |
| Use Case Name: | Delete chat | | |
| Created By: | Tony | Last Updated By: | Tony |
| Date Created: | 8/9/2022 | Date Last Updated: | 8/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | A user may delete their chat history with another user or leave a group chat by long pressing on their chat history and tapping the trash bin icon that appears in the top right corner of the screen.  The user will be able to choose to delete chat history for the other user such that neither of them will be able to see their chat history.  The user will only be able to delete chat history for himself when leaving a group chat. |
| Preconditions: | 1. User must be logged in 2. An active chat must have been created |
| Postconditions: | 1. Chat history for the deleted chat will be removed permanently for the user. |
| Priority: | Low |
| Frequency of Use: | Low |
| Flow of Events: | 1. User taps on the chat icon 2. User long press on any of the listed chats displayed 3. A trash bin icon appears in the top right corner of the interface 4. User taps on the trash bin icon, a message box pops up to confirm if the user wants to delete the chat.    1. If the selected chat is a private chat, an additional checkbox to delete chat history for the other user will be shown. 5. User taps on the confirm button and the chat history is deleted permanently |
| Alternative Flows: | - |
| Exceptions: | - |
| Includes: | Login, View list of active chat |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | Not yet implemented |

### **GP - Gym Page Functions**

| Use Case ID: | GP.1 | | |
| --- | --- | --- | --- |
| Use Case Name: | Open gym page | | |
| Created By: | Subeen | Last Updated By: | Subeen |
| Date Created: | 7/9/2022 | Date Last Updated: | 7/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | All gyms registered on the application have their own gym pages. Gym page of a specific gym acts as a hub for all information regarding the gym. |
| Preconditions: | 1. The user must be registered via UA.1 2. The user must be logged in via UA.2 |
| Postconditions: | - |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | 1. User navigates to the search page via bottom tab navigator 2. User searches for a gym by its name    1. System sorts the gym list according to the search word or the filter    2. System displays the sorted gym list 3. User scrolls up on a list until he/she finds a gym of interest 4. User clicks on the selected gym 5. System redirects the user to the gym page |
| Alternative Flows: | - |
| Exceptions: | - |
| Includes: | View list of gyms |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | GP.2 | | |
| --- | --- | --- | --- |
| Use Case Name: | Like/Unlike gym | | |
| Created By: | Subeen | Last Updated By: | Subeen |
| Date Created: | 7/9/2022 | Date Last Updated: | 7/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | Liking a gym page enables a user to have quick access to the gym chat via the gym discussion page accessible using the navigation bar. |
| Preconditions: | 1. The user must be registered via UA.1 2. The user must be logged in via UA.2 |
| Postconditions: | 1. Once followed, the gym is added to the “liked” gym list on the gym discussion page 2. Once unfollowed, the gym is removed from the “liked” gym list on the gym discussion page |
| Priority: | Low |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User enters the gym page to follow via GP.1 2. User clicks heart shaped button    1. Changes gym from liked to unliked or unliked to liked       1. If gym is liked, the gym will be added to user’s liked gym       2. If gym is unliked, the gym will be removed form the user’s liked gym |
| Alternative Flows: | - |
| Exceptions: | - |
| Includes: | Open gym page |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | GP.3 | | |
| --- | --- | --- | --- |
| Use Case Name: | Open gym chat/discussion | | |
| Created By: | Subeen | Last Updated By: | Subeen |
| Date Created: | 7/9/2022 | Date Last Updated: | 7/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | On the gym page, users can enter a group chat for the gym to interaction with other users. |
| Preconditions: | 1. The user must be registered via UA.1 2. The user must be logged in via UA.2 |
| Postconditions: | - |
| Priority: | Medium |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User enters the gym page via GP.1 2. User clicks speech bubble button 3. The gym chat/discussion is displayed on the user’s UI |
| Alternative Flows: | GP.3.AC.1 Open gym chat via gym discussion page   1. User navigates to gym discussion page using navigation bar 2. Gym discussion page will show all gyms which user liked 3. User may select the gym to open the gym chat |
| Exceptions: | - |
| Includes: | Open gym page, View gym group chat |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

### **WL - Workout log Functions**

| Use Case ID: | WL.1 | | |
| --- | --- | --- | --- |
| Use Case Name: | Add workout log | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 20/9/2022 | Date Last Updated: | 20/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | Users can create workout log by creating an order of pre-defined exercises and define the number of sets, reps, or time-taken |
| Preconditions: | 1. User must be logged in |
| Postconditions: | 1. Created workout logs is saved to the system cloud |
| Priority: | High |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User navigates to the gym workout log page on home page 2. User selects the “+” button 3. User will be directed to the workout log creation page 4. User adds exercises from a list of pre-defined exercises    1. User may filter the list by exercise name and/or target muscle group 5. User defines if exercise is conducted by duration or repetitions (reps)    1. If duration is selected, user defines the time taken    2. If reps is selected, user defines number of reps 6. Repeat step 4 & 5 until as desired by the user 7. User selects the “Create” button to complete the creation |
| Alternative Flows: | - |
| Exceptions: | WL.1.EX.1 Cancel workout log creation   1. At any point between step 3 to 6, user may select the “Cancel” button 2. All workout log details will not be saved 3. User will return to the gym workout log page |
| Includes: | - |
| Special Requirements: | 1. At least one exercise must be chosen to create the workout log |
| Assumptions: | - |
| Notes and Issues: | Current implementation only takes in one exercise  Filter by exercise name or muscle group not yet implemented |

| Use Case ID: | WL.2 | | |
| --- | --- | --- | --- |
| Use Case Name: | Edit workout | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 20/9/2022 | Date Last Updated: | 20/9/2022 |

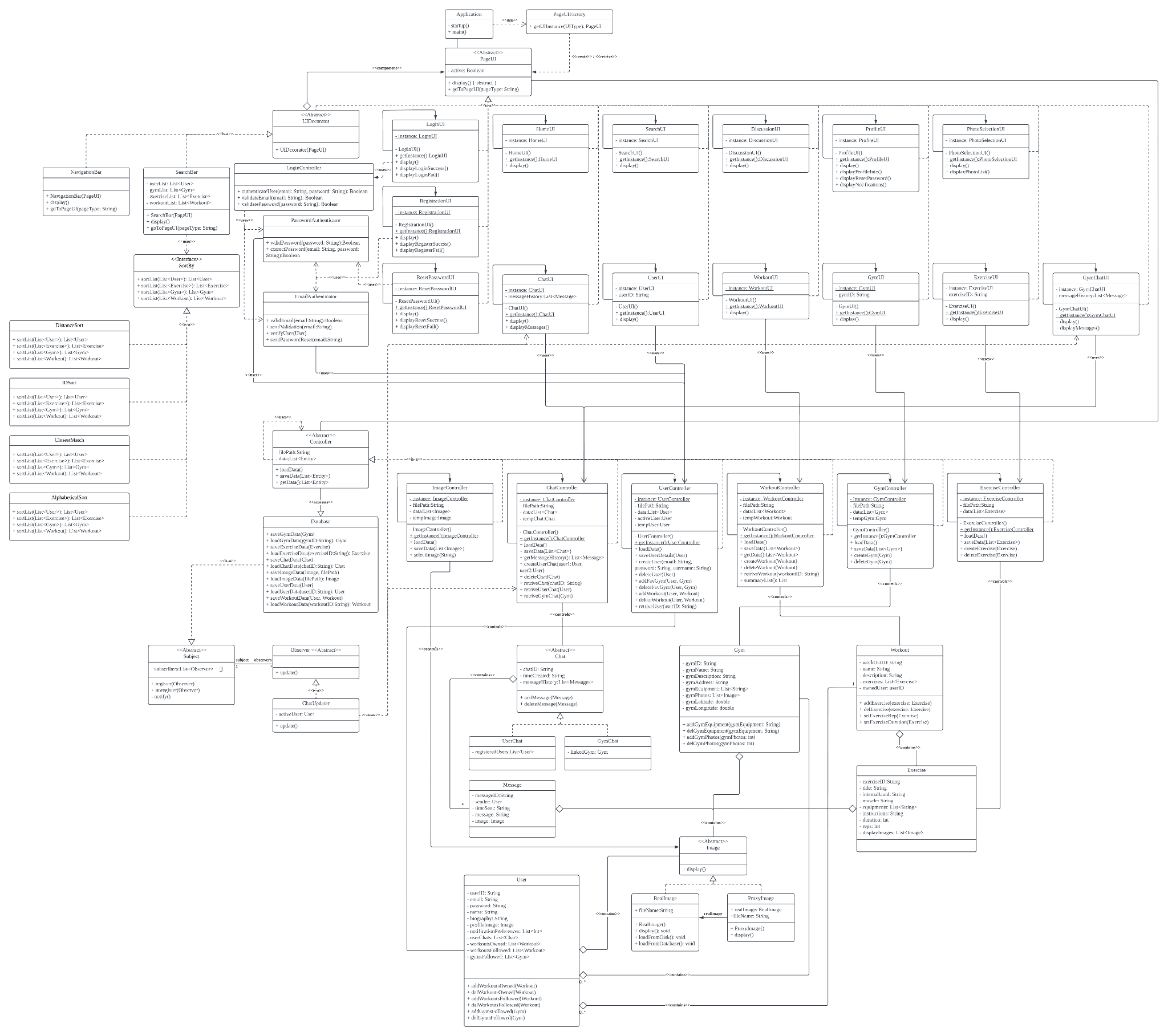
| Actor: | User |
| --- | --- |
| Description: | User may choose to edit an existing workout log which they have created. Exercises may be added or removed, and the duration or repetitions of the exercises may be changed by the user. |
| Preconditions: | 1. User must be logged in 2. Workout must exist and belong to the user |
| Postconditions: | 1. Edits to the workout must be saved to the cloud |
| Priority: | Medium |
| Frequency of Use: | Low |
| Flow of Events: | 1. User navigates to the gym workout log page 2. User selects the dropdown menu on the selected workout 3. User selects the “Edit workout” option 4. User will be directed to the edit workout log page 5. User makes edits to the workout log    1. Add/remove exercises    2. Edit exercise duration/repetitions 6. User selected the “Confirm” button |
| Alternative Flows: | - |
| Exceptions: | WL.2.AC.1 Cancel edits   1. User may choose to cancel edits by selecting the “Cancel” button 2. All edits will not be saved 3. User will be returned to the gym workout log page |
| Includes: | - |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | - |

| Use Case ID: | WL.3 | | |
| --- | --- | --- | --- |
| Use Case Name: | Delete workout log | | |
| Created By: | Ryan Ong | Last Updated By: | Ryan Ong |
| Date Created: | 20/9/2022 | Date Last Updated: | 20/9/2022 |

| Actor: | User |
| --- | --- |
| Description: | User may choose to delete any of workout log which the user have created |
| Preconditions: | 1. User must be logged in 2. Workout log must exist 3. Workout log must have been created by the user |
| Postconditions: | 1. Workout log will be deleted from the cloud |
| Priority: | High |
| Frequency of Use: | Medium |
| Flow of Events: | 1. User navigates to the gym workout log page 2. User selects the dropdown menu on the selected workout log 3. User selects the “Delete workout” option 4. User confirmed the deletion 5. Workout log will be deleted |
| Alternative Flows: | - |
| Exceptions: | - |
| Includes: | - |
| Special Requirements: | - |
| Assumptions: | - |
| Notes and Issues: | Not yet implemented |

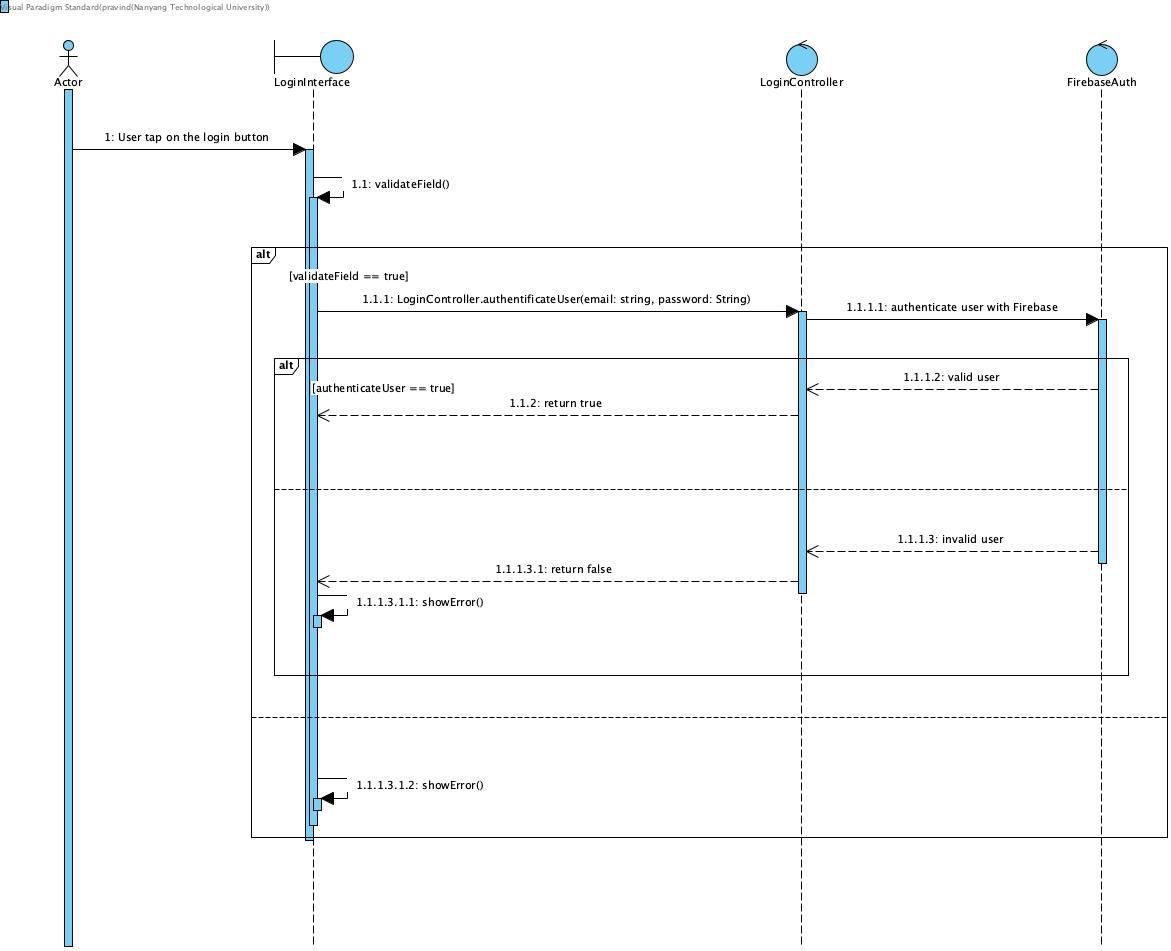
d. Class Diagram

Link to full diagram <https://lucid.app/lucidchart/77f5a862-e461-4735-9363-92589db94bb5/edit?viewport_loc=-1840%2C-2167%2C7518%2C3322%2C5a966tluIAlw&invitationId=inv_659d5a8c-0437-4308-a378-cbc4782f1784>



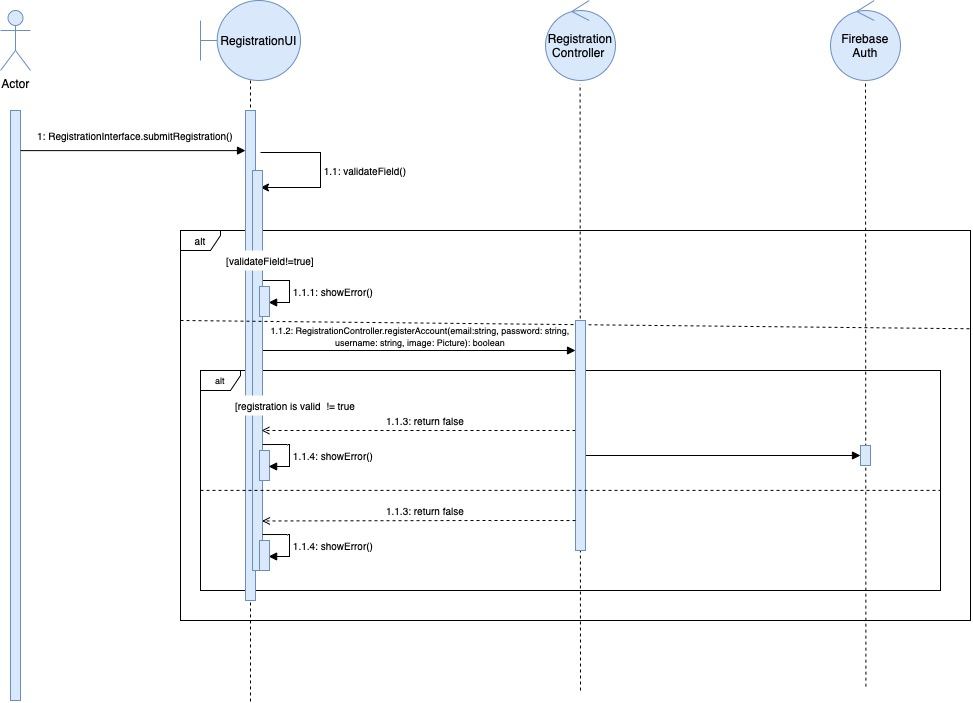
e. Sequence Diagrams

### Use Case 1: Login via Email

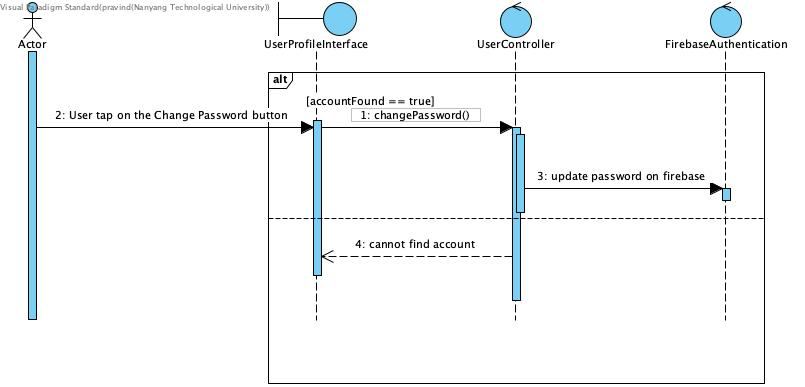


### 

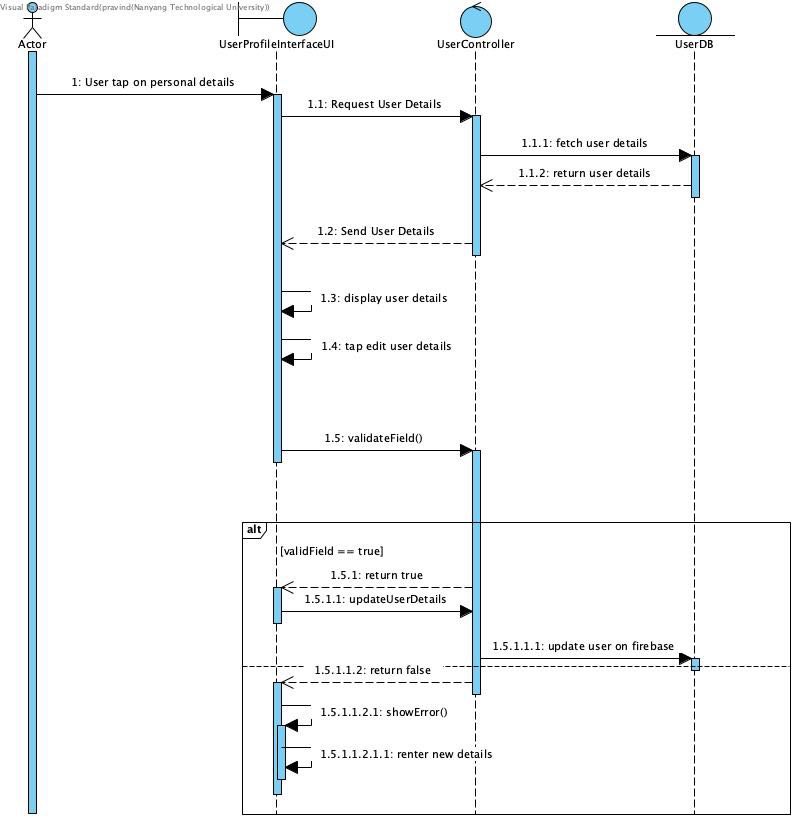
### Use Case 2: Register new Account via Email



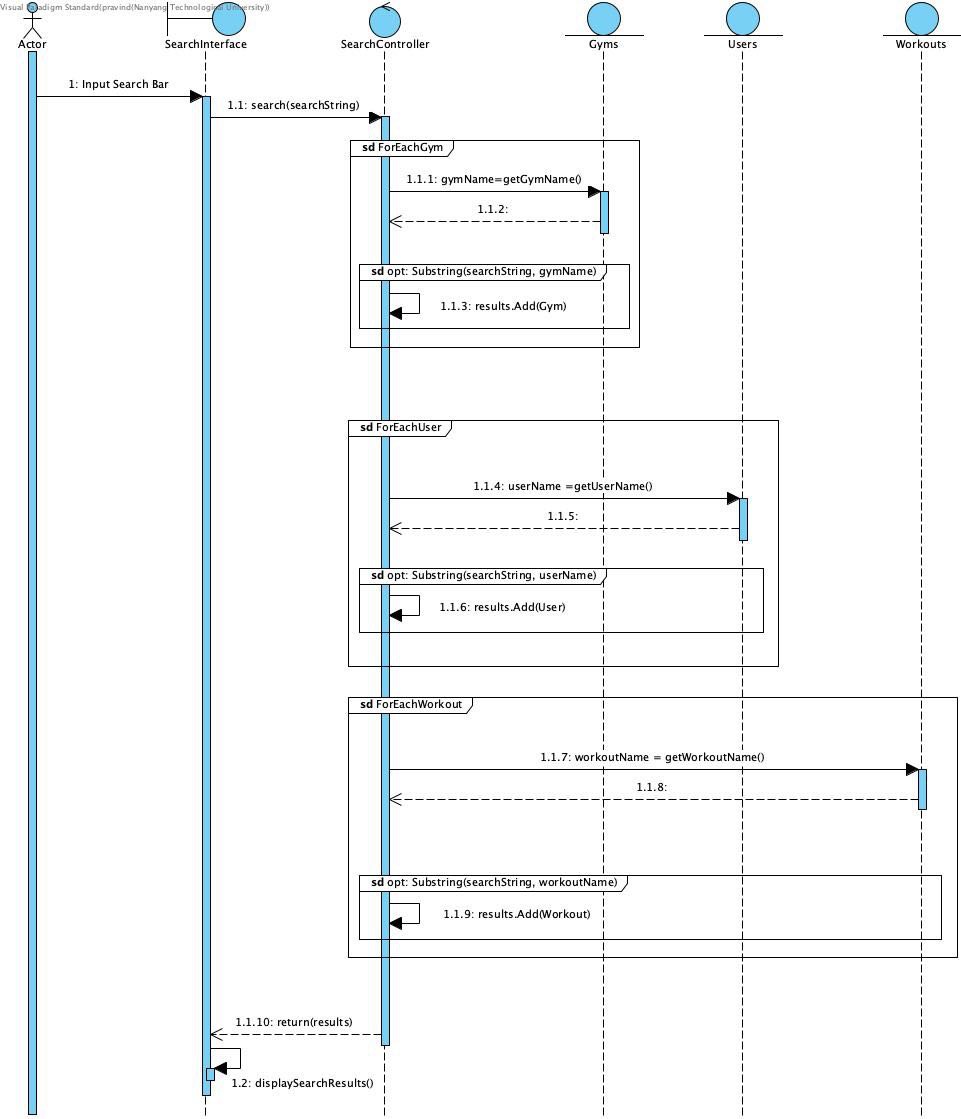
### Use Case 3: Change Password



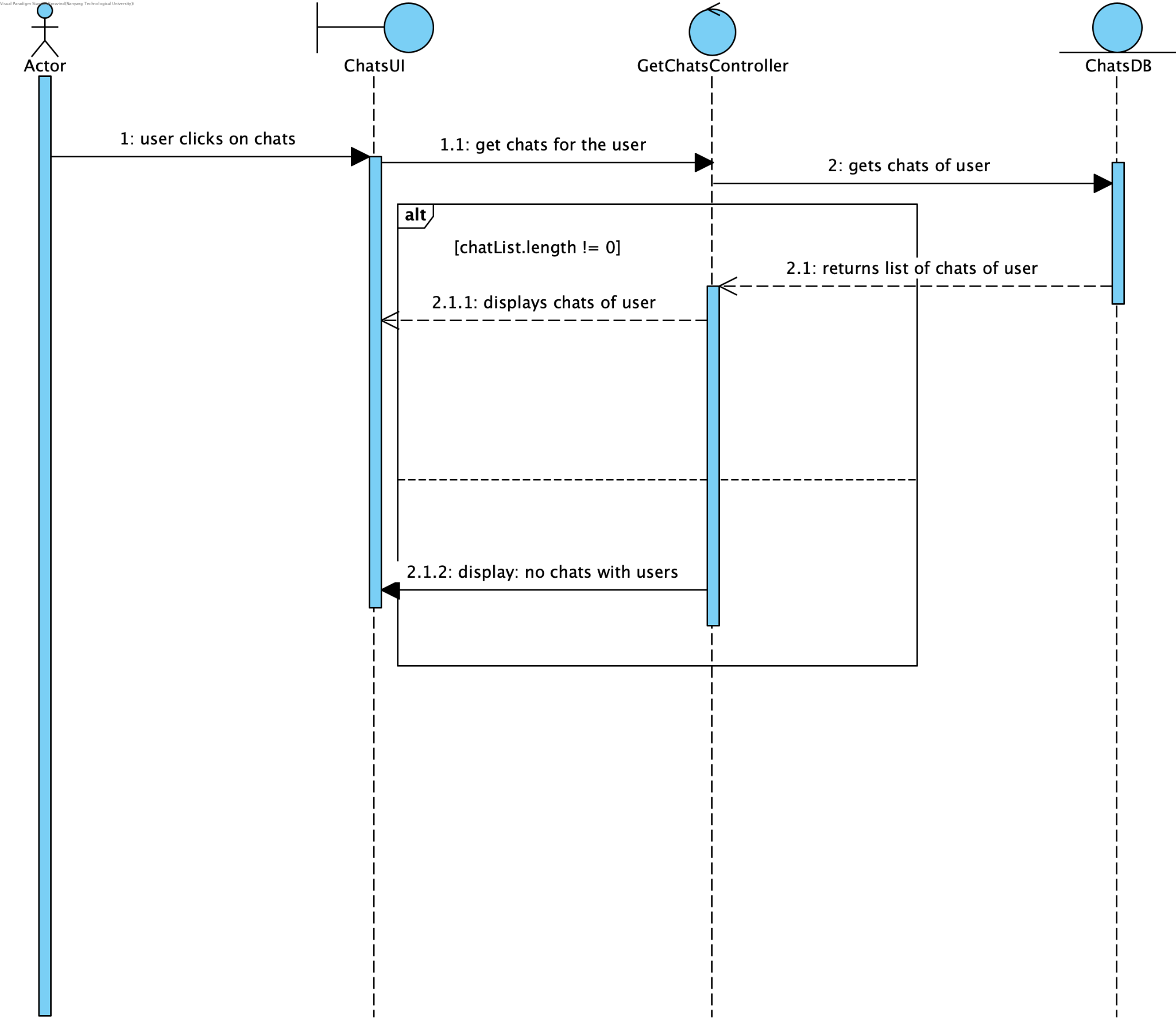
### Use Case 4: Edit Account



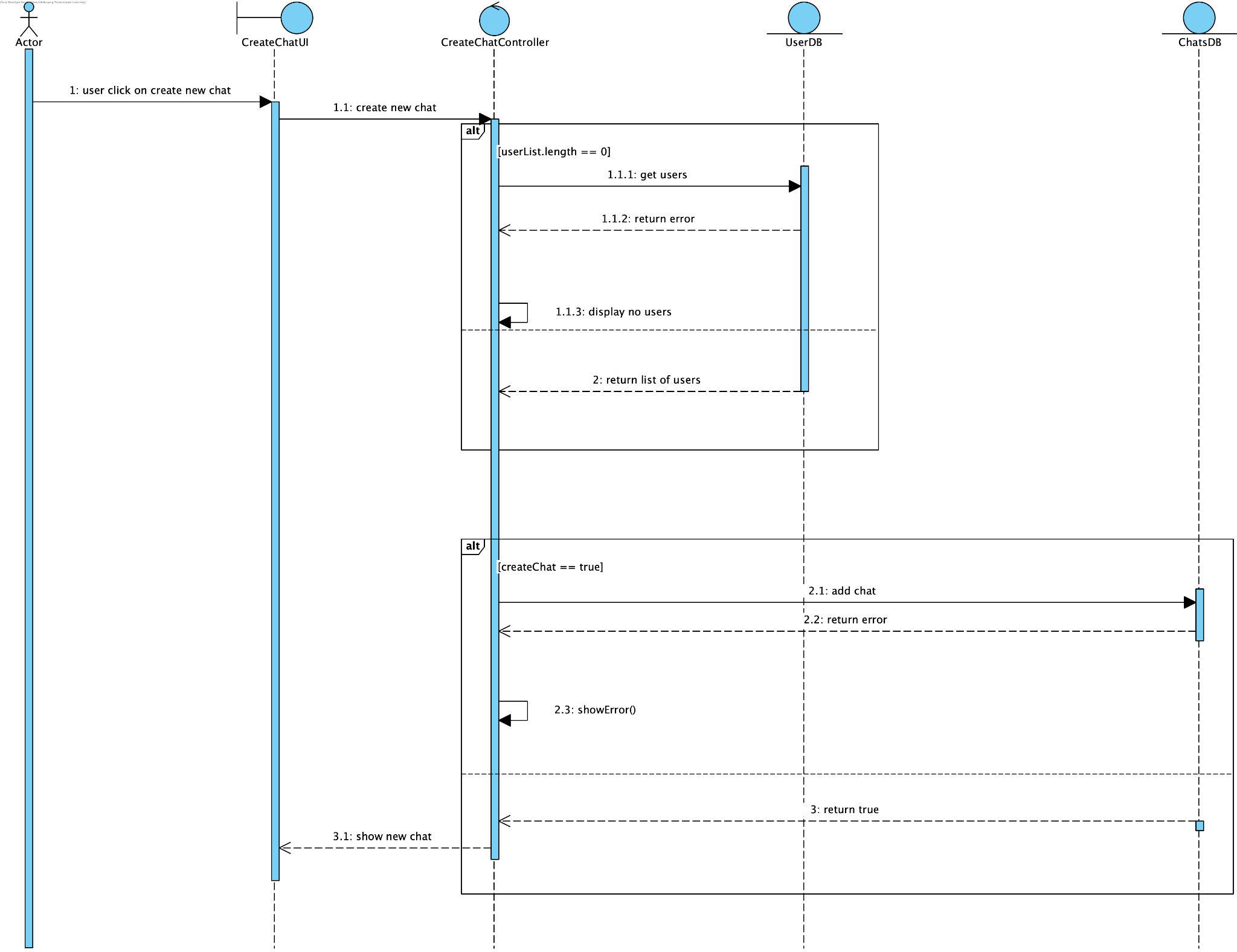
### Use Case 5: Searching for Gyms / Users / Workouts



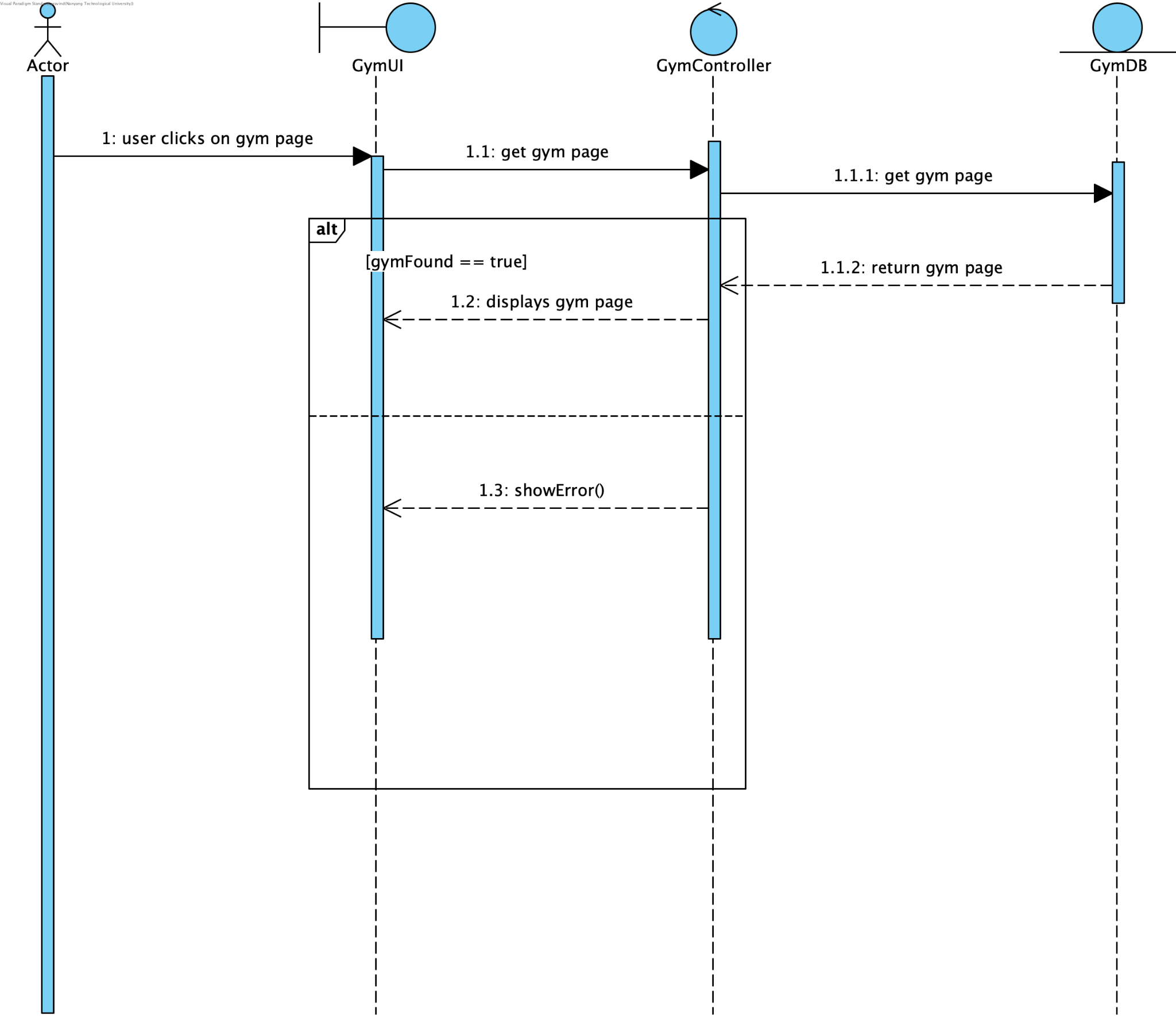
### Use Case 6: Show all chats



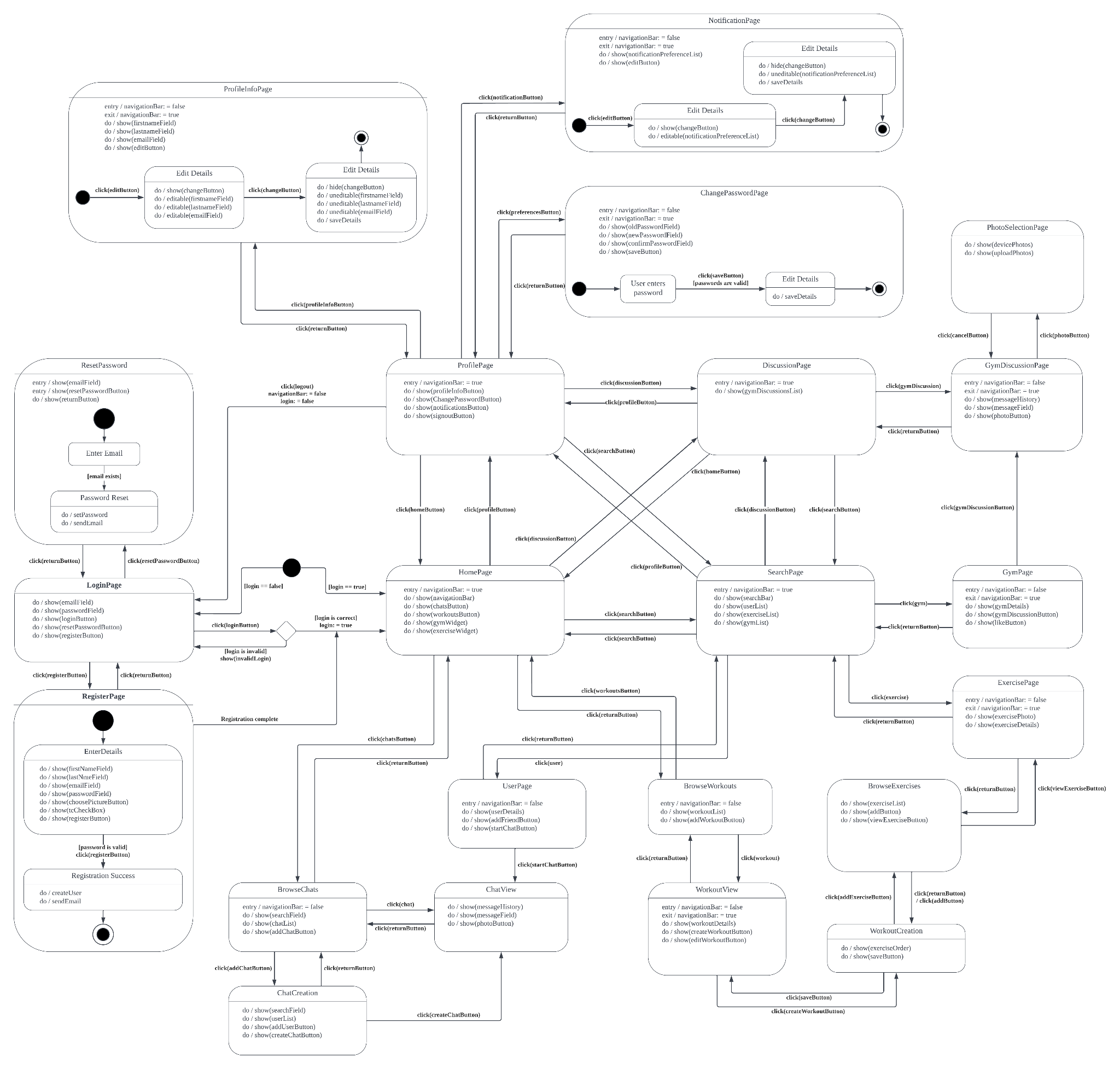
### Use Case 7: Create Chat



### Use Case 8: Open Gym Page



## f. Dialog Map



# 3. Non-Functional Requirements Elicitation

1. Under **favourable network conditions** (defined exactly and unambiguously in data dictionary):
   1. If a user is viewing the list of the chats they are a member of
      1. It must take less than 5 seconds for the chat message sent by another user to result in any required update of the unread chat visual indicator displayed to the viewing user.
   2. If a user is viewing a chat they are a member of
      1. It must take less than 5 seconds for the chat message sent by another user to result in the app displaying the newly sent message for the user viewing the chat they are a member of.
   3. If a user is viewing a gym page
      1. It must take less than 5 seconds for the post created by another user to result in the app displaying the newly created post for the user viewing the gym page.
2. All **app data** and **user data** must be stored in **remote storage.**
   1. App data and user data may additionally be stored in the storage of user’s **devices.**
3. All **remote storage** must be through the **Firebase** cloud provider.
4. All relevant **PDPA regulations** must be obeyed.
5. The app and its features must be available on both **up-to-date iOS** and **up-to-date Android devices**.
6. All **app animations** should execute without visible stuttering.
7. All app animations should execute in less than 2 seconds.
8. **First-time app startup** should take less than 10 seconds.
9. **App startup** after the first start-up should take less than 7 seconds.
10. The final app executable must be less than 50MB in size.
11. The final app executable should achieve a lower than 15% detection ratio on the file-scanning website <https://www.virustotal.com/gui/home/upload>.
12. All above nonfunctional requirements are to be considered via execution on a phone with at least a 2.0 GHz clock speed CPU and 4GB RAM (Random-Access Memory).
    1. No nonfunctional requirements need to be met on a phone not meeting this specification.

# 

# 4. Interface Requirements

## a. User Interfaces

Users who use OurWorkout are gym-goers who would like to search for nearby gyms, discover new workouts and interact with other users for motivation or advice.

## b. Hardware Interfaces

OurWorkout is designed to work on hardware devices that support WiFi or Mobile Data connection that can connect to the Internet.

## c. Software Interfaces

OurWorkout is designed to work on both iOS and Android platforms enabled devices.

## d. Communication Interfaces

OurWorkout can be accessed through the use of the Internet.

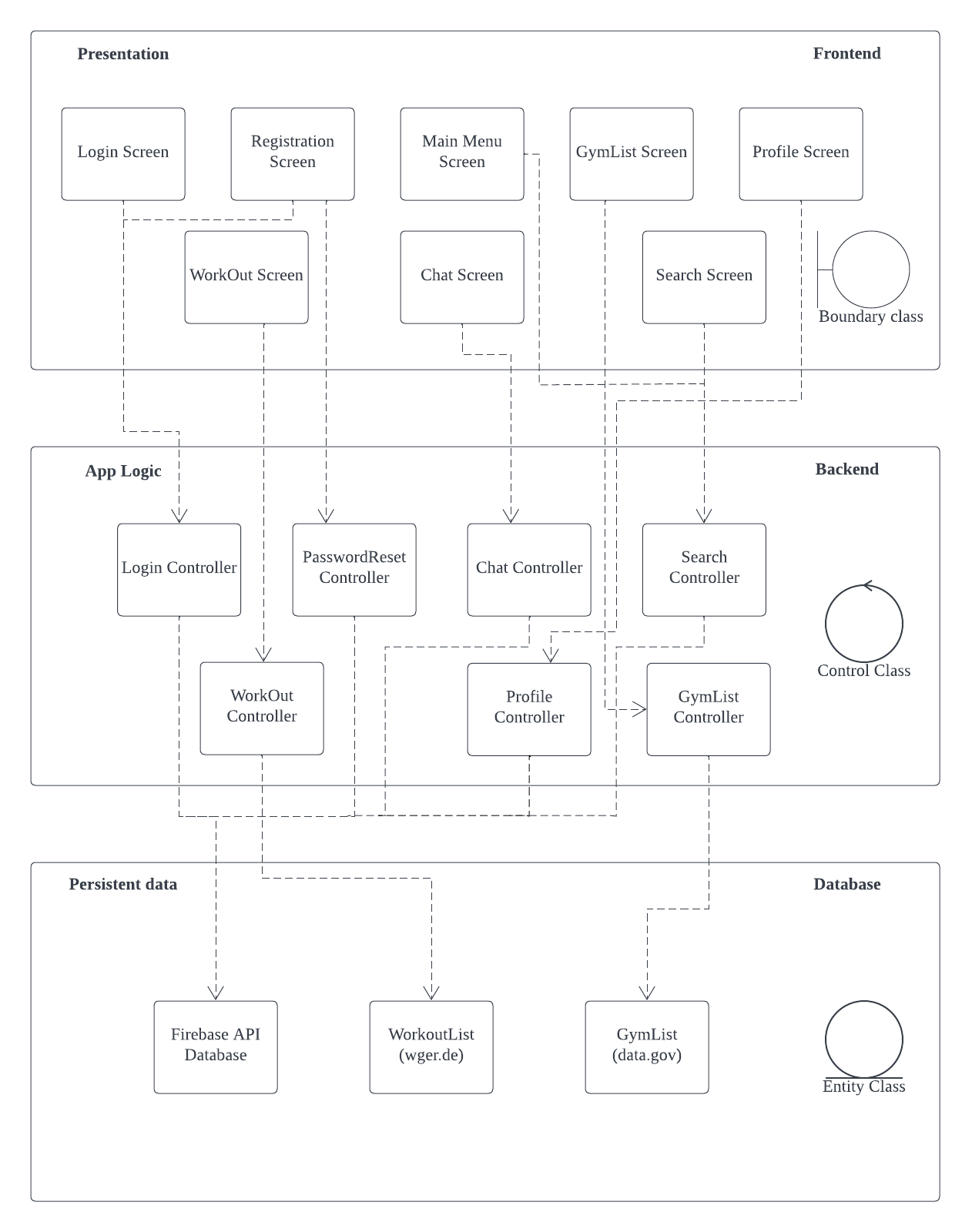
## 

## 

# 5. Data Dictionary

| **Index number** | **Term** | **Meaning** |
| --- | --- | --- |
|  | Make use [of the app]  *Verb* | To utilise any functionality of the app which is described in sections other than Sections 1, 2 and 3 of the requirements. |
|  | App | The finished mobile application which is the final deliverable product of the software development process, knownas “OurWorkout”. |
|  | User | Any individual who desires to or has already made use of the app. |
|  | Authenticate [a user]  *Verb* | The process, also known as “logging in”, by which a user provides credentials in order to prove their identity (as the user to whom a give user account belongs), and thereby gain access to functionality which they are in their capacity as an individual allowed to access. |
|  | Logging in [a user] | Synonymous with “authenticate”. |
|  | Authentication | The process by which a user is authenticated. |
|  | Unauthenticated user | A user who has from the perspective of a single running instance of the app client on their device not yet authenticated themselves as being the user they claim to be or not yet claimed to be a specific user and thereby having access to the functionality permitted to that user. |
|  | Register [a user]  *Verb* | To create a user account and thereby inform the app in its capacity as a software system with components both local to users devices and running on remote servers of the register of the newly-registered user’s existence as a user distinct from any other users who have either previously registered or shall register in the future, as well as provide credentials whereby in the future the user may authenticate themselves. |
|  | Account, User Account | The result of registration, a discrete entity existing on a server or servers used for the app to which the post-authentication activities, produced content and any other relevant data belonging to a user may be tied. |
|  | Email authentication | A form of authentication such that one of the credentials required in order to successfully authenticate via this form of authentication is an email address (understood to mean the *addr-spec* in IETF RFC 5322.3.4, although further restrictions on email addresses to be used in our application are outlined in the requirements), with the email address having been previously provided through the process of account registration. |
|  | App developer | The individual, company, or agglomeration of individuals and companies united by their shared responsibility to implement the requirements outlined for the OurWorkout software. |
|  | OurWorkout software, OurWorkout app | Synonymous with “the app”. |
|  | Password | One of the credentials utilised for logging in/authentication, under the assumption that only the user whose user account is attached to the password or any individuals he or she has authorised to impersonate him know the password and are thereby able to authenticate as the user in question to whose user account the password is tied. |
|  | Character | A single character of the chosen encoding method. In UTF-8, for our purposes a character is understood to mean a single glyph. |
|  | UTF-8 | As defined by the Unicode Consortium at <https://home.unicode.org/>. The internal text encoding of the OurWorkout app. |
|  | Hostname | The section of a valid email address as defined in the requirements preceding the ‘@’ symbol. |
|  | Domain name | The section of a valid email address as defined in the requirements proceeding the ‘@’ symbol. |
|  | Symbol | Synonymous with character, unless specified otherwise. |
|  | Permitted characters | With regards to any specific text input taken from a user by the app, the characters which are permitted to appear in the text input, as outlined by the requirements. |
|  | UTF-8 encoded ASCII non-numeric characters. | Those characters which are encoded by UTF-8 as single bytes and not corresponding to the decimal digits ranging inclusively from 0 to 9. |
|  | Associated with an account | In reference to credentials and data reasonably understood to be tied to an individual user, such as email addresses, passwords, and profile photographs, to be stored remotely by the app as being tied to that individual user. |
|  | Credentials | That data which is used compulsorily or optionally as proof of identity during authentication. |
|  | Data | As defined by the second edition of the Oxford English Dictionary. |
|  | First name | The first name of a person, expressed as a UTF-8 string |
|  | Last name | The last name of a person, expressed as a UTF-8 string |
|  | Profile photograph | An image used to accompany the first and last name of a user when displaying a representation of the user to other users, to be uploaded by the user themselves. Further specifications are given in the requirements. |
|  | Placeholder image | The image displayed in place of the profile photo wherever a profile photo is required if the user has not provided a profile photo. |
|  | Identical email address(es) | Used to describe a pair of email addresses for which under UTF-8 encoding every character is identical, that is to say, they are represented via UTF-8 strings with identical values. |
|  | Verification email | An email sent in order to verify a user’s account, so that, among other things, their email address can be confirmed to really exist in the world. This is done via the user clicking on a personalised verification link present in the email. |
|  | Verification link | A link sent in the verification email, crafted such that being accessed causes the app to verify the account of the user intended to access it. |
|  | Web browser | As defined by the Oxford English Dictionary 3rd Edition |
|  | User’s details | The first name, last name, and profile photograph (if any) of the user, taken together as one entity. If the value of this information is changed later, then the new version is considered to comprise the user details. |
|  | Chat(s) | The UI element(s) and logical entity used to represent a series of chat messages sent by two users to each other, defined at a conceptual level primarily by the idea of real-time textual communication. More concrete details are provided in the requirements. |
|  | Chat message(s) | Individual, primarily textual missives sent one-at-a-time and received in real-time (taken to mean under specific time limits outlined by the requirements) by one user to another. Displayed as the constituent parts of chats. |
|  | Real-time | Under specific time requirements outlined by the requirements. |
|  | UI interaction (with) | A swipe, click, button press, other individual conscious action by a user or the series or combination of the above taking place visually upon or inside the displayed or viewed object being interacted with. |
|  | Unread chat | A chat in which messages have been sent since the chat was last viewed by the user from whom’s perspective the chat is an unread chat. An unread chat may be unread for one chat member and not unread for another chat member – the unread status is a property of both the chat and the user who is a member of the chat and for whom it is to be considered unread or read. |
|  | Blocked (user), *noun*  Blocking, *verb* | A user, if blocked by a user, is considered to be blocked by that specific other user, never without relation to any other users. The blocked user is unable to send any messages to the user by whom they have been blocked. |
|  | Chat members | The two users who are participating in a chat, having either sent a chat message to the other user or being the recipient of the aforesaid chat message. |
|  | Message text | The UTF-8 string which is one of the possible contents of a chat message. |
|  | Gym log invitation | A chat message which instead of or (although support for this is not required) in addition to message text confers the ability for the recipient to via performing a UI interaction while viewing the chat message instead or in addition view the gym log of the sender. |
|  | Maximum message text length | The maximum number of characters the message text of a message is to be permitted to contain. |
|  | Location | The geophysical location in the real world of any object of interest. |
|  | Gym | Any of the gymnasiums in Singapore found in the Data.gov.sg dataset. |
|  | Gyms map display | A map-based display by which users may view the locations of various gyms and go to their gym pages. Further elaboration is given in the requirements. |
|  | Data.gov.sg dataset | The dataset regarding gyms in Singapore sourced from Data.gov.sg, with the URL given earlier in this document. |
|  | Gym page | A distinct conceptual and concrete data entity associated with each gym, agglomerating the gym’s name and location with a list of the “gym posts” users have chosen to make in association with the gym. |
|  | Supported format | With regards to an image to be chosen from the device storage of the user, this indicates the supported file type(s) the image may take. The supported formats may vary situationally and are further specified in the requirements. |
|  | Post | Textual or image-based missives sent one-at-a-time and viewable in real-time as they are sent (taken to mean under specific time limits outlined by the requirements) as a constituent part of a gym page. A gym page allows for viewing all posts made as part of it. |
|  | Full name | The first and last name of a user, joined together with a space in the middle. |
|  | Post text | The text provided by a user during creation of a post, if any. |
|  | Post image | The image provided by a user during creation of a post, if any. |
|  | Maximum post text length | The maximum number of characters a post is permitted to contain. |
|  | WGER Exercise Dataset | The dataset of exercises obtained from the WGER exercise database, link provided earlier in this document. |
|  | Exercise page | A UI state that exists for each exercise in the WGER Exercise Dataset, allowing for viewing the exercise image, if any, and exercise description if any, of the exercise. |
|  | Exercise image | The image assigned to the exercise by the WGER Exercise Database. |
|  | Exercise description | The description assigned to the exercise by the WGER Exercise Dataset. |
|  | Exercise | A distinct fitness routine, possibly with provided image or description, listed in the WGER Exercise Dataset. |
|  | Personal workout log | A UI state, abstract concept and concrete data entity mainly defined by its acting as a list of individual fitness training sessions (referred to as ‘workout log entries” a user has engaged in and chosen to record in the app, which may be viewed either by the user themselves or by users it has been shared with as a chat message (elaborated in the requirements pertaining to chats). Also referred to as a “workout log”. |
|  | Workout log entry | An individual user-entered record of a fitness session/workout they have engaged in, storing the exercise engaged in, a textual description of the fitness session/workout and a brief listing of the “intensity” of the workout. All of the workout log entries a user creates are listen in their personal workout log. |
|  | Workout log notes | A length of UTF-8 text entered by a user as one of the three required components of a workout log entry (as the aforementioned“textual description”). In this document, a “workout log note” (singular) has no meaning. |
|  | Intensity | With regards to a workout log entry, a short snippet of text (maximum number of characters expounded on in the requirements), used to describe how strenuous a fitness session/workout was, i.e. “4 reps, 3 sets at 100kg”. This does not need to be a “structured format” – in fact, arbitrary UTF-8 text below the character limit is required to be permissible. |
|  | Search functionality | The ability to “search” searchable items, constituted by user accounts, exercises and gym pages, allowing the user to view a list of those items which match an entered UTF-8 textual prefix and furthermore via some UI interaction with the items on this list view the corresponding chat, exercise or gym page. |
|  | Search results | The list of searchable items which is displayed upon searching. |
|  | Searching | The act of utilising the search functionality to list those searchable items which match a UTF-8 textual prefix. |
|  | Search text | The UTF-8 text matched with the searchable attribute as a prefix of the string, in order to produce search results. Further specified in the requirements. |
|  | Searchable attribute | The name of an exercise or gym (page), or the full name of a user, in all cases used as UTF-8 text. |
|  | Favourite | The act by which a user records a gym as one of their “favourite gyms” through a provided button on the corresponding gym page, elaborated on further in the relevant section of the requirement. |
|  | Unfavourite | The act by which a user records a gym as no longer one of their “favourite gyms” through a provided button on the corresponding gym page, elaborated on further in the relevant section of the requirement. |
|  | App data | That data (not including code and not including data used by software libraries the app utilises) which is displayed or utilised by the application software and was not provided by any particular authenticated user. Includes all data obtained from the earlier listed in this document public datasets. |
|  | User data | Data voluntarily provided by an authenticated user, i.e. their first name, last name, profile photograph (if any), the contents of any chat messages they have sent or posts they have made, the contents of any gym log entries they have written. |
|  | Delete | As used in the requirements, to make no longer accessible through the app UI. |
|  | App shutdown | The termination of the running app instance on an Android or iOS device. |
|  | Placeholder data | As part of data deletion, placeholder data is, if such is required, provided in order to be displayed or utilised in its place. |
|  | Favourable network conditions | “Good enough” network connectivity, defined unambiguously as follows. The running app instance experiences favourable network conditions if   1. It is able to access any and all public web servers operated by the app developer over the internet. 2. It experiences less than 400ms round-trip time thereby. 3. It is able to transfer uncompressed data at 400 KB/s thereby. |
|  | Local app client | The component of the app which is installed in the storage of a user’s device, as opposed to existing on a remote server |
|  | Device | The Android or iOS device through which the user makes use of the app. |
|  | Running app instance | The executing instance of the local app client. |
|  | Remote storage | Storage on a server to which the app developer has access rights, or storage with a cloud storage provider. |
|  | PDPA regulations | All regulations laid out in Singapore’s Personal Data Protection Act, as outlined at <https://www.pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act>. |
|  | Up-to-date iOS devices | Devices running Apple iOS of major version 14 or later. |
|  | Up-to-date Android devices | Devices running Android OS of major version 11 or later. |
|  | App features | Any distinguishable component of the operation of the app in accordance with and obeying the functional requirements. |
|  | App animations | Any fixed-length animation played by the UI of the app. |
|  | App startup | The process of the transition of the local app client from non-running to running state. |
|  | First-time app startup | The first instance of the local app client undergoing app startup after app installation. |
|  | App installation | The process of the introduction to storage of all components of the local app client and the transition to being able to run the local app client, obeying all relevant recommendations and directives of the device OS manufacturer. |
|  | Firebase | Firebase is a cloud storage service and app development platform operated by Google. <https://firebase.google.com/> |

# 6. System Architecture



# 7. Key design issues and patterns

## Design issues

### Identifying and storing persistent data

All entity classes (User, Message, Chat, Image, Workout, & Exercise) are stored on a Firebase Realtime Database as object classes. Entities containing other entities (e.g. chat class has message class components), are saved via class referencing.

Object oriented structure allow multiple entities to call on a shared class entities while reducing the required storage size of the system. For example, in the scenario of two users in an active chat the two user entities and each user UI are referencing to the a single chat instance shared by the two user entities, thus ensuring no unncessary duplication of the chat instance and allowing better data synchronization.

Image data are referenced to via filepath within entity classes to improve system performance. Upon app initialization or class creation, using a lightweight reference to the images, filepath, allows for faster loading time and response from the application. Application will only load the required image when required and images will be cleared from local cache regularly.

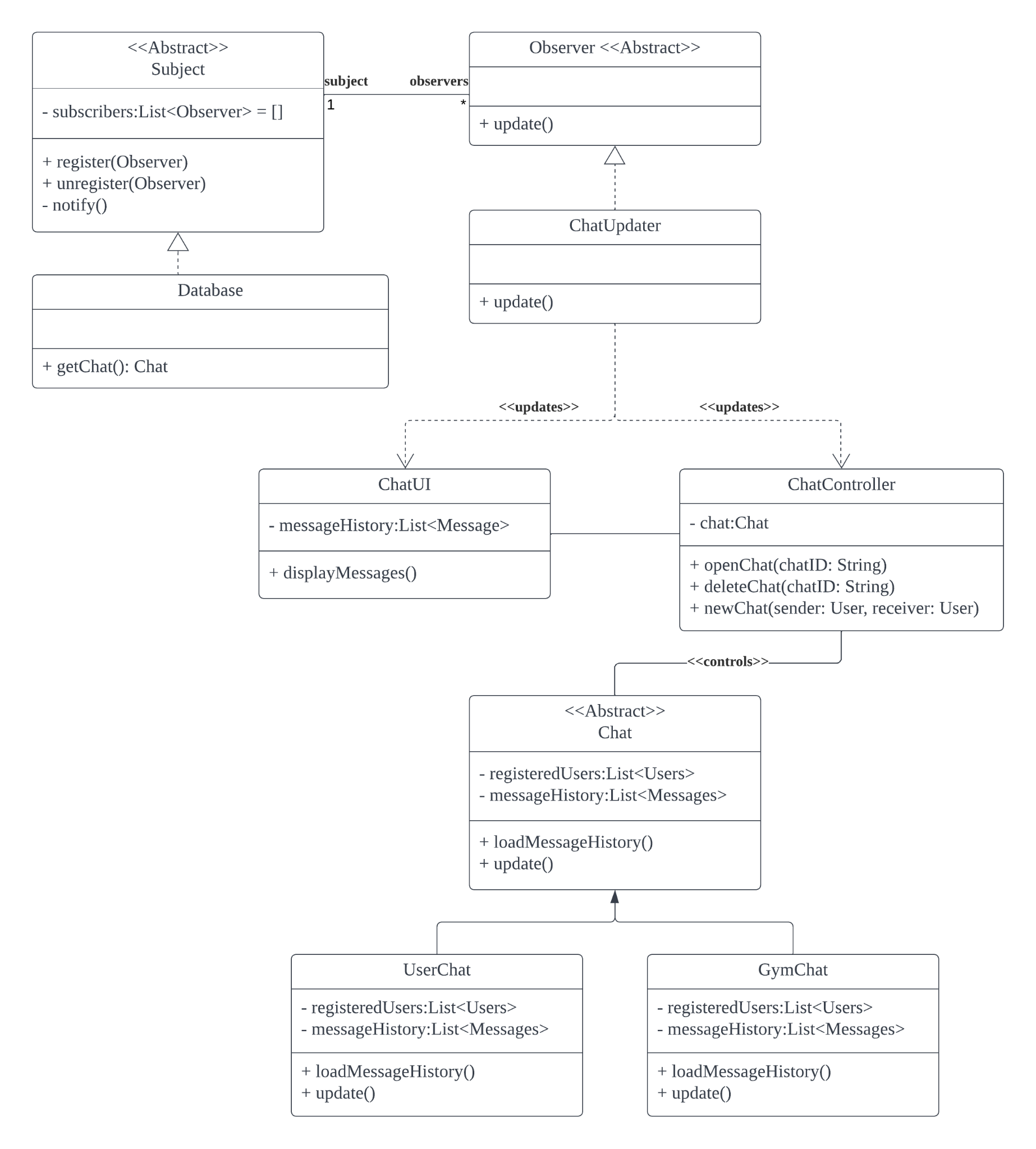
### Providing access control

As of current implementation, there is only one level of user access.

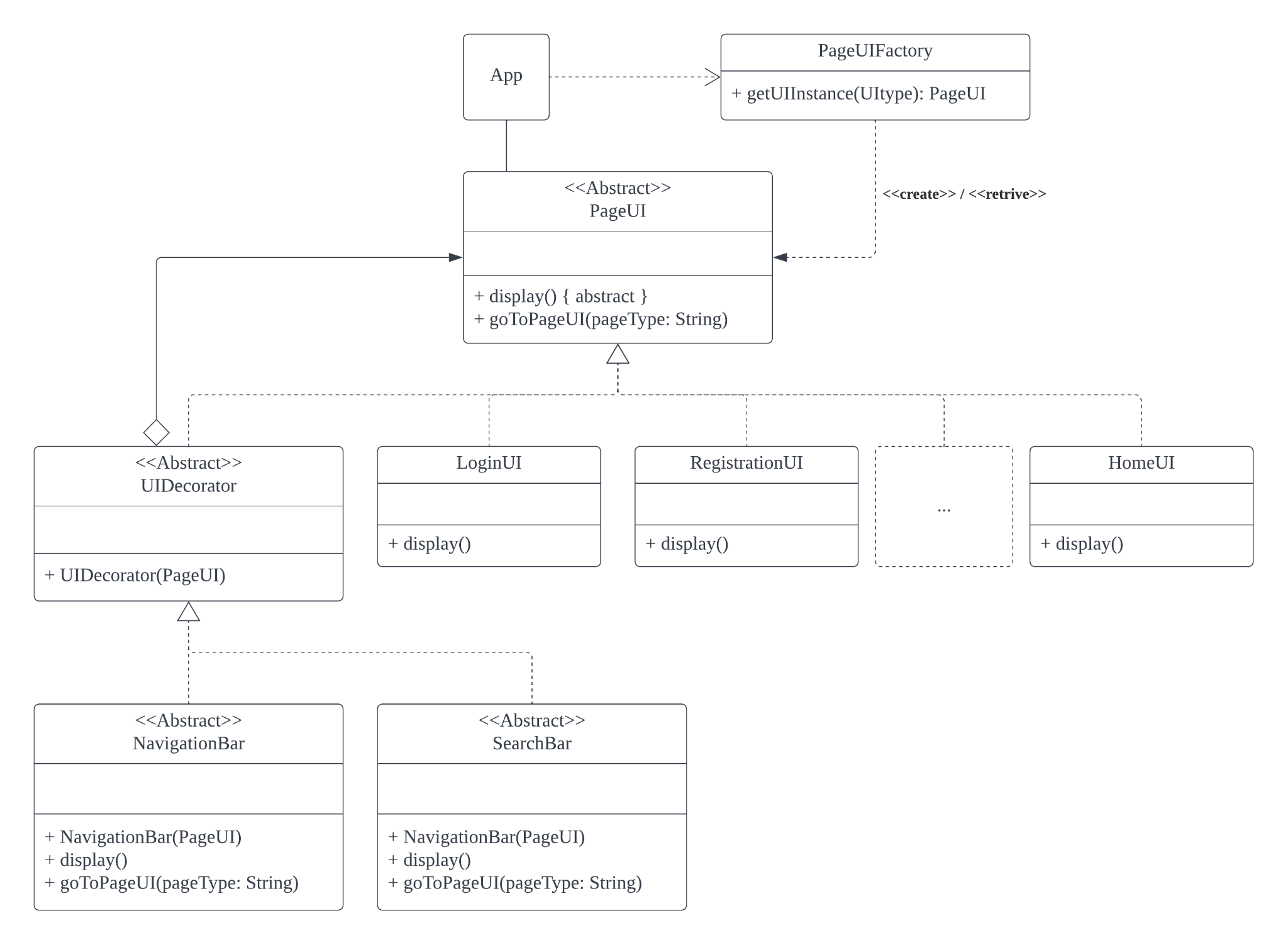
## Design patterns used

**Design Problems/Patterns**

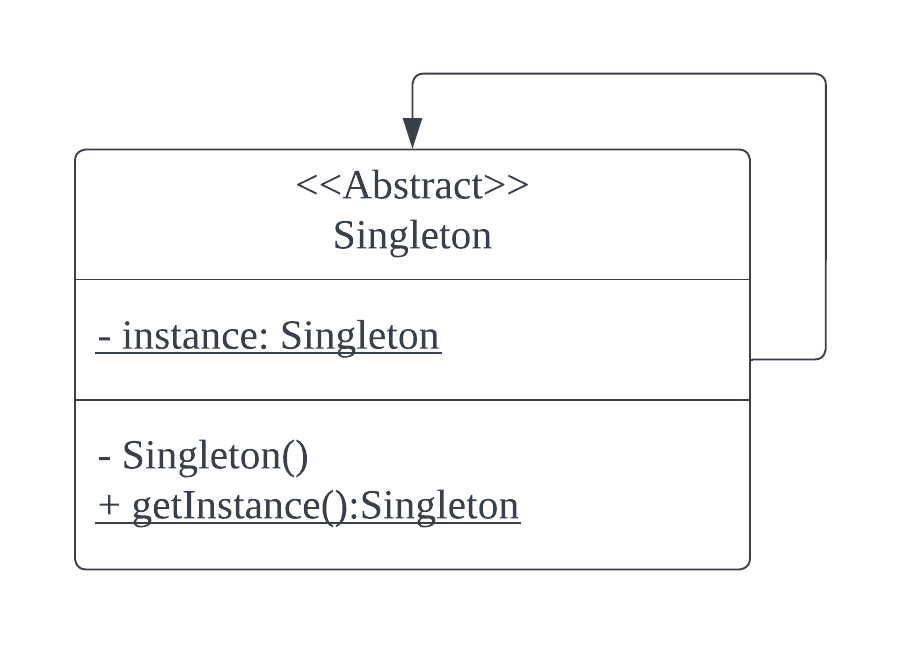
1. **Observer + Facade pattern**

* Description:
  + “Event driven” software design where observers registers to a subject and receives updates/notification when subject data is updated
  + Facade class includes the classes whose methods are required and runs the methods to complete a function
* Scenarios:
  + User chat page and gym dicussions automatically updates when any member of the chat sends a message
* Solution:
  + ChatUpdater class class is a observer and also a facade
  + Chats and gym discussions databases are subjects
  + ChatUpdater are automatically registered to the database upon creation
  + Gym discussions are registered to the database based on user’s decision to like/follow the gym
  + When gym discussion or chats are updated with new messages, database will trigger push update to ChatUpdater, and ChatUpdater will trigger methods required on ChatController and ChatUI to retrieve latest data
  + User’s UI will reflect the updated message history
* Notes:
  + Other pages (e.g. workouts page, user page, gym page, etc…) updates during user browsing does not fall under observer pattern as user is required to manually trigger update to receive latest information
  + ChatUpdater will be deregistered upon logout or deletion of app
* Class diagram:  
  

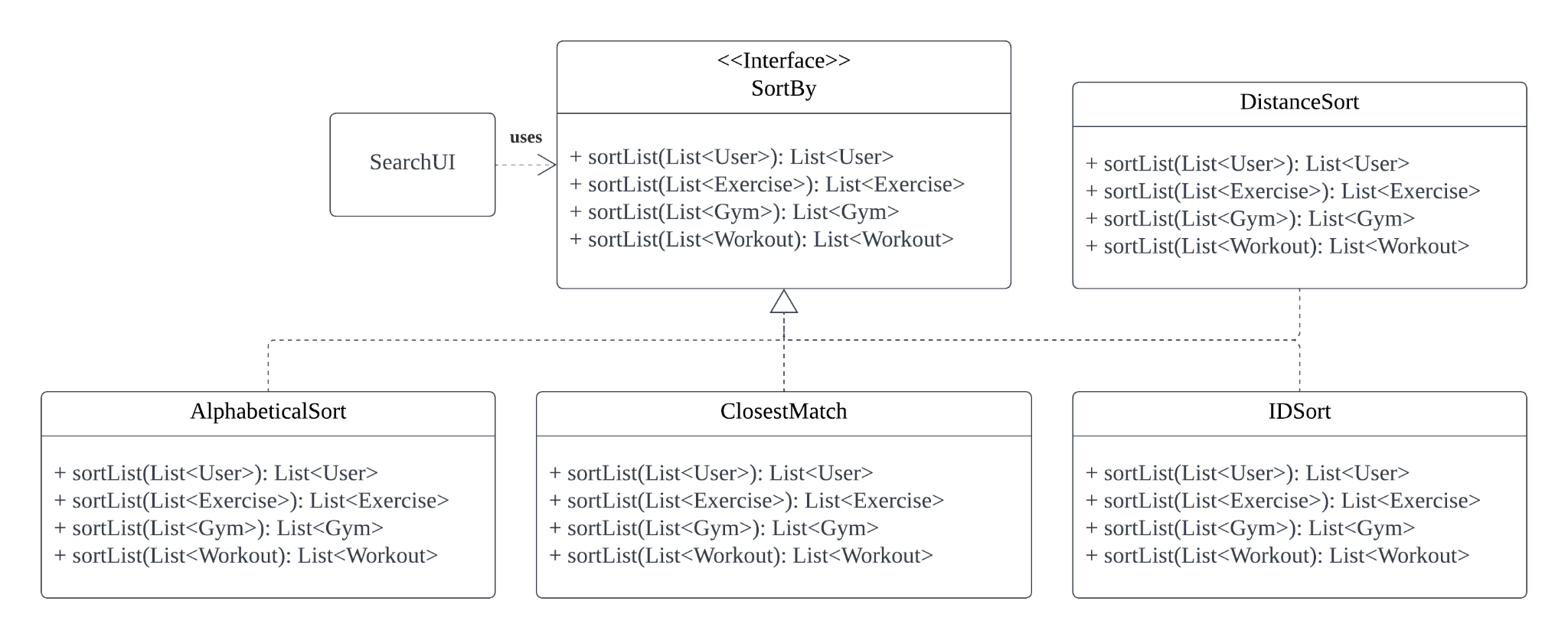
1. **Factory pattern + Decorator**

* Description:
  + Dynamic creation of objects through a factory object, creating instances of objects implementing a abstract class
  + Attach additional responsibilities to an object dynamically without affecting other objects
* Scenarios:
  + Switching between UI pages
  + Each UI page may not always need a component like navigation bar
* Solution:
  + Each UI page are implementations of a UI abstract class which can be created or called by FactoryUI object
  + Use PageUIDecorator to add the components dynamically when needed.
* Notes:
  + NIL
* Class diagram:  
  

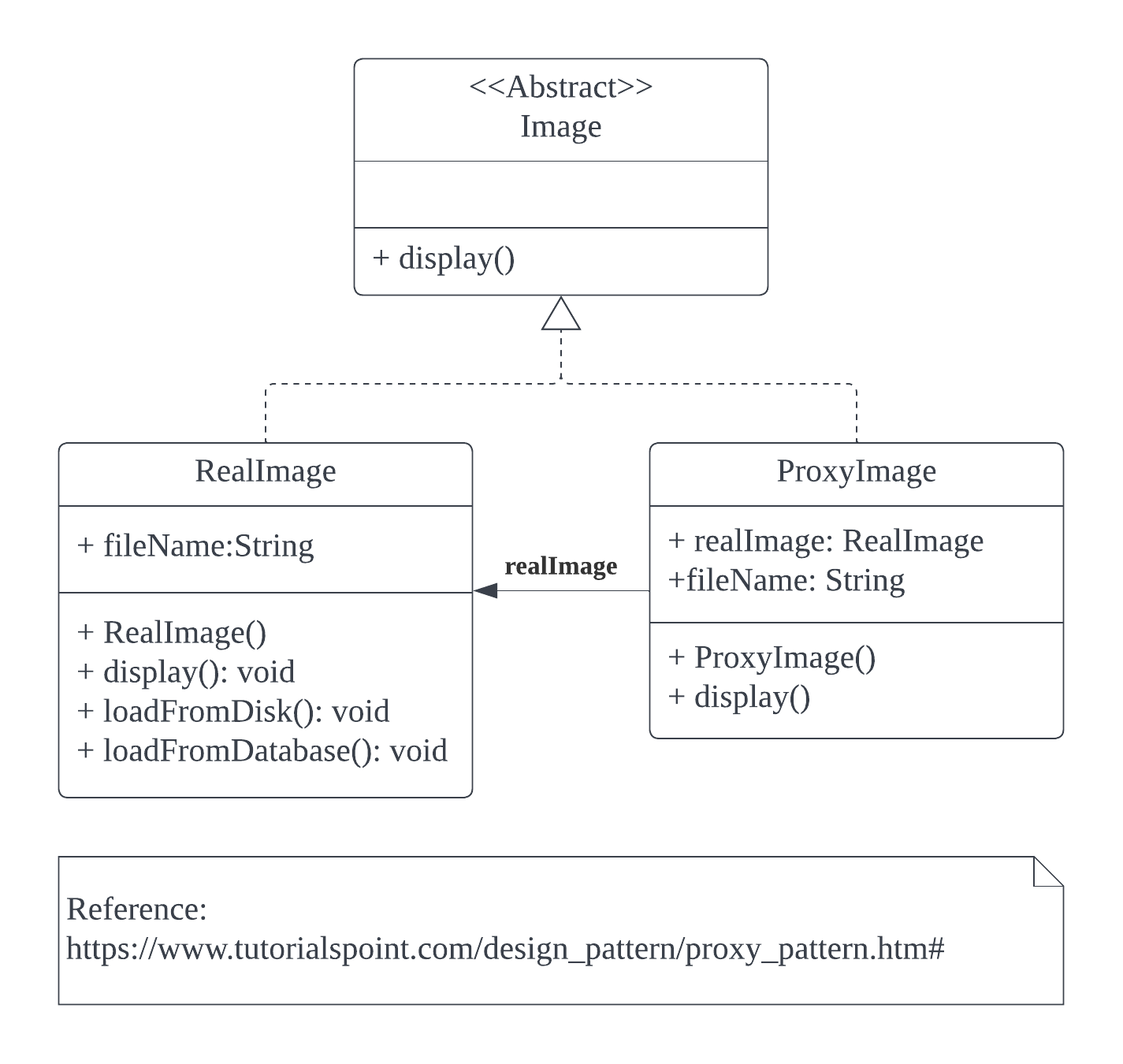
1. **Singleton pattern**

* Description:
  + Design pattern that restricts the instantiation of a class to a singular instance
* Scenarios:
  + Only one instance of each boundary and control classes required for program
* Solution:
  + Define every boundary and control classes according to singleton structure
* Notes:
  + Entity classes should not be singleton as multiple instances of the object will be created
* Class diagram:  
  

1. **Strategy pattern**

* Description:
  + Allow for interchangeability of algorithms or objects with similar functions
* Scenarios:
  + Entity lists (users, gyms, and exercises) can be sorted by a variety of manners depending on user preference
* Solution:
  + SortBy interface is referenced to to represent and call for specific sorting algorithms usable by other classes
  + Multiple methods under the same name can be created to handle the different type of lists with different objects
* Notes:
  + DistanceSort only applicable to gyms
* Class diagram:  
  

1. **Proxy pattern**

* Description:
  + Use of a lightweight proxy object to represent an expensive object, instantiating the expensive object only when required
* Scenarios:
  + Images to display during runtime may slow operations if all images loaded upon startup
* Solution:
  + Use of proxy object to indicate image location or access which will load the image during loadImage method call
* Notes:
  + NIL
* Class diagram:  
  

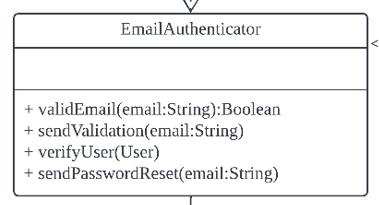
# 8. Testing

## Black Box – Equivalence Class and Boundary Value Testing

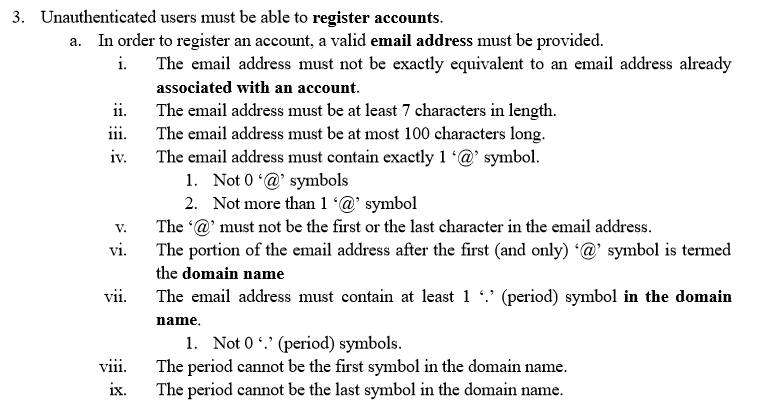
**Design test cases using equivalence class and boundary value testing techniques to test one important control class that implements some important application logic according to the requirements specification.**

What we are testing: The **EmailAuthenticator**.

How do we test it: Manual testing.



Below, we reproduce the relevant functional requirements for validity of an email.





We thus end up with equivalence classes corresponding to these attributes:

1. Length of email address.
2. Number of ‘@’ characters in email address.
3. Is there at least one ‘.’ (period) character after the ‘@’?
4. Is there a ‘.’ (period) character as the first character immediately after the ‘@’?
5. Is there a ‘.’ (period) character as the last character in the email address?

Valid and invalid ECs for these are outlined below.

| Length of email address | <7 | ≥7, ≤100 | >100 |
| --- | --- | --- | --- |
| Equivalence Class | Invalid | Valid | Invalid |

| Number of ‘@’ characters in email address | <1 | =1 | >1 |
| --- | --- | --- | --- |
| Equivalence Class | Invalid | Valid | Invalid |

| Is there at least one ‘.’ (period) character after the ‘@’? | =0 | >0 |
| --- | --- | --- |
| Equivalence Class | Invalid | Valid |

| Is there a ‘.’ (period) character as the first character immediately after the ‘@’? | Yes | No |
| --- | --- | --- |
| Equivalence Class | Invalid | Valid |

| Is there a ‘.’ (period) character as the last character in the email address? | Yes | No |
| --- | --- | --- |
| Equivalence Class | Invalid | Valid |

Let us now design the corresponding test cases, using concepts of EC and BC testing. While our inputs to the algorithm for checking validity of email addresses are multiple, we will only input one string to our app, the email address, thus that is noted in the columns. Our range-based variables are length of email address and number of ‘@’ characters. Our other three variables are discrete. Thus we have two sets of range-based ECs with three ECs each, and three sets of discrete ECs with two ECs each.

## EC-based Testing

| Length of email address | Number of ‘@’ characters in email address | Is there at least one ‘.’ (period) character after the ‘@’? | Is there a ‘.’ (period) character as the first character immediately after the ‘@’? | Is there a ‘.’ (period) character as the last character in the email address? | Email address input | Expected result | Actual result |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 1 | Yes | No | No | ac@b.com | Valid | Valid |
| 5 | 1 | Yes | No | No | a@b.c | Invalid | Invalid |
| 120 | 1 | Yes | No | No | LiMIsPSGeP7NNtLLBzfTx3NSkwgw70tfEmN6FBaFdL.6sGUcyvg  vl5.vYBA5anVYQQ9abGQCF2E6dDNDfEmN6FBaFdL16s@GUcyvg4G6zEf.yJP1jJwCmO.f04 | Invalid | Invalid |
| 9 | 0 | Yes | No | No | fhfhfh.com | Invalid | Invalid |
| 10 | 4 | Yes | No | No | A\_@@@@z.co | Invalid | Invalid |
| 12 | 1 | No | No | No | adhhd@xsfsdf | Invalid | Invalid |
| 15 | 1 | Yes | Yes | No | Ac.s\_Axs@.sfsdf | Invalid | Invalid |
| 17 | 1 | Yes | No | Yes | 9\_4f\df123\_@23sdfs. | Invalid | Invalid |

## BC-based testing

**Length of email address**: Valid boundaries are {7, 100}. Invalid boundaries are {6, 101}

**Number of ‘@’ characters in email address**:

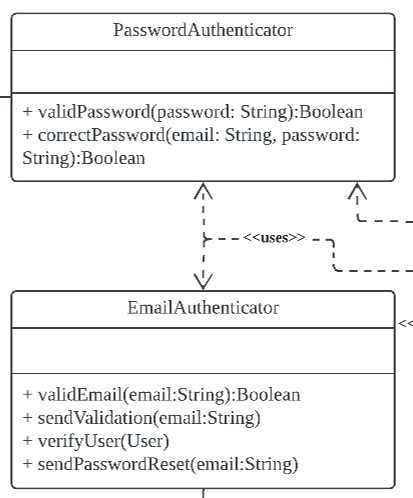
Valid boundary is { 1 }. Invalid boundaries are {0, 2}

The rest are discrete and don’t have boundary values.

| Length of email address | Number of ‘@’ characters in email address | Is there at least one ‘.’ (period) character after the ‘@’? | Is there a ‘.’ (period) character as the first character immediately after the ‘@’? | Is there a ‘.’ (period) character as the last character in the email address? | Email address input | Expected result | Actual result |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 1 | Yes | No | No | a@b.com | Valid | Valid |
| 100 | 1 | Yes | No | No | fkfgkcebezmgtdzvewdqyxbptggzhpkasttmerhvhbuysu@hclyxkzsaffthlymfoiupxrapwzquzrimelhmvhhtsjjnsw.ptwyr | Valid | Valid |
| 6 | 1 | Yes | No | No | ac@b.c | Invalid | Invalid |
| 101 | 1 | Yes | No | No | rdmawzlooxxnxwcornkpznad@kckhtbisrecvzorvekvvxvelvlrmnnndqavkbvnhzgfj.yxhbzviehnpqdf.lux.xzs.obetrcqt | Invalid | Invalid |
| 7 | 0 | Yes | No | No | ffhm.sd | Invalid | Invalid |
| 100 | 2 | Yes | No | No | trmiikcck@berbulrztwhbgad.hbjedtdqbbrgjfugrudzccgb@hmetpswjpvijaxjavuruqauvjpdvbjgpm.wgfo.ylooa.aedd | Invalid | Invalid |
| 7 | 1 | No | No | No | ad@ds1f | Invalid | Invalid |
| 100 | 1 | Yes | Yes | No | kzd11glfzbyyxyaoivxjjrfrmzge.bdgev.oqxrdrzxpktlkr@.gdzsjbwptwxewp33vm.qszdvlhcllqjvk00qj\_\_.jkwe.hzle | Invalid | Invalid |
| 7 | 1 | Yes | No | Yes | 9\_\\@f. | Invalid | Invalid |

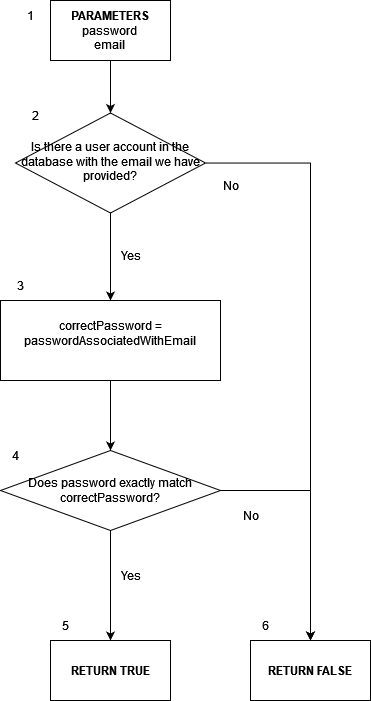
## Basis Path Testing

Again, we use exclusively manual testing.



**correctPassword** and **validEmail** are the two methods we choose to test via basis path testing.

### CFG of correctPassword



**Cyclomatic Complexity: 2 + 1 = 3.**

**Basis path #1 (baseline path):** 1 -> 2 -> 3 -> 4 -> 5.

**Basis path #2:** 1->2->**6**.

**Basis path #2:** 1->2->3->**4->6**.

Before running these test cases, ensure that user accounts with these credentials have been registered and email-verified.

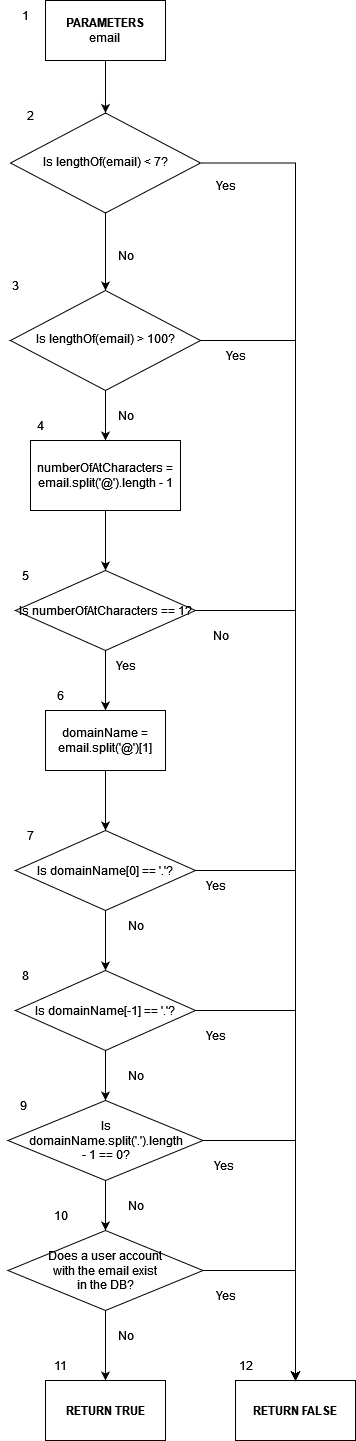
Email: [adakjsakd@example.com](mailto:adakjsakd@gmail.com). Password: 1J@jasdkasfkj

Password: [aosdlasjd@example.com](mailto:aosdlasjd@example.com). Password: M7@skhdaadsa

| Test case #  Aka, basis path # | **Input:** email | **Input:** password | Expected result | Actual result |
| --- | --- | --- | --- | --- |
| 1 | [adakjsakd@example.com](mailto:adakjsakd@gmail.com) | 1J@jasdkasfkj | True | True |
| 2 | [sjd@example.com](mailto:aosdlasjd@example.com) | M7@skhdaadsa | False | False |
| 3 | [aosdlasjd@example.com](mailto:aosdlasjd@example.com) | 9A@AKnkjasa | False | False |

### 

### CFG of validEmail



**Cyclomatic Complexity: 7 + 1 = 8.**

**Basis path #1 (baseline path):** 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11.

**Basis path #2:** 1-> 2 -> **12**.

**Basis path #3:** 1-> 2 -> 3-> **12**.

**Basis path #4:** 1-> 2 -> 3-> 4 -> 5 -> **12**.

**Basis path #5:** 1-> 2 -> 3-> 4 -> 5 -> 6 -> 7 -> **12**.

**Basis path #6:** 1-> 2 -> 3-> 4 -> 5 -> 6 -> 7 -> 8 -> **12**.

**Basis path #7:** 1-> 2 -> 3-> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> **12**.

**Basis path #7:** 1-> 2 -> 3-> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -**>** 10 **-> 12**.

Before running these test cases, ensure that user accounts with these credentials have been registered and email-verified.

Email: [adakjsakd@example.com](mailto:adakjsakd@gmail.com). Password: 1J@jasdkasfkj

Password: [aosdlasjd@example.com](mailto:aosdlasjd@example.com). Password: M7@skhdaadsa

| Test case #  Aka, basis path # | **Input:** email | Expected result | Actual result |
| --- | --- | --- | --- |
| 1 | [sjd@example.com](mailto:aosdlasjd@example.com) | True | True |
| 2 | abc | False | False |
| 3 | abcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabc | False | False |
| 4 | abc@dasaa@sdg | False | False |
| 5 | ashdjsahdj@.aass | False | False |
| 6 | asdkasdkk@aass. | False | False |
| 7 | ajhsdj23hi@djsk | False | False |
| 8 | [adakjsakd@example.com](mailto:adakjsakd@gmail.com) | False | False |

# 9. Good Software Engineering Practices

Solid Design Principles

**Single Responsibility Principle**

Description:

* + There should never be more than one reason for a class to change.
  + High cohesion

Application:

* + All classes have a single purpose following the Boundary-Contorl-Entity architecture
  + Each UI class displays its respective component
  + Each control class is only responsible for the entities under it’s control

**Open-Closed Principle**

Description:

* + Module should be open for extension but closed for modification.
  + When new subclass is added, changes to the superclass should not be needed.

Application:

* + PageUI class holds an abstract method for subclasses to implement.
  + SortBy interface is extendable by adding more sorting methods and individual methods can be edited without affecting others

**Liskov Substitution Principle**

Description:

* Subtypes must be substitutable for their base type without disrupting the behaviour of the program

Application:

* Classes are extended or implementation only from the appropriate abstract or interface classes
* Method class can be used following the interface class methods

**Interface Segregation Principle**

Description:

* Larger interfaces should be split into smaller ones, ensuring implemented classes only need to be concerned about the methods that are of interest to them

Application:

* UIDecorator class is an extension of PageUI class, implementing methods that other UI related classes do not need

**Dependency Inversion Principle**

Description:

* Decouple software modules reducing dependency of high-level modules on low-level modules

Application:

* Subclasses of abstract and interface classes can be instantiated using the super class instance, and methods can be called from instantiated super class as well

Agile Method

The Agile Development Method was chosen by our team as we felt that it was more suitable for our team. This meant that while we were developing the application, we were also writing the documentation and updating it along with our code by using an iterative approach.

Phases/Steps:

1. Identification of functional and non-functional requirements
2. Identifying how users will use the application
3. Planning of required base classes
4. Identify the applicable key design patterns
5. Planning of system structure
6. Implementation of system structure
7. Testing
8. Final product

Version control

Since being able to track changes in the development of our project is very important, GitHub was used to collaborate and track our code changes.

New features and bug fixeds are worked on different branches forked from the released application branch, allow multiple feature/bug fix developments to be worked on concurrently. This also ensure little overlap of work between developers, prevent overwriting of each other’s code. Proper merging conflict resolution and testing are done on a testin branch before it is merged back into the main app release branch.

Since GitHub tracks who uploaded which segment of code and allows us to roll back erroneous code, it saved a lot of time during prototyping. Every update pushed onto the git repository have the appropriate naming and descriptions to allow every developer to get a summary of the update. Codes are pushed onto the repository in small batches to allow for smaller rollbacks in the case of erroneous code.

Model View Controller

As our web application has a large number of user interactions, we decided to use the model view controller architecture. This allows the user interface to be separated from our application logic, allowing for high cohesion and loose coupling. Controller logic such as API calls and validation are independent of how the interface is displayed. Additionally, the interface subscribes to input errors caught to display them.

Being a web app, some validations are performed on the client side, while others are done on the server. Non-critical checks, such as name length and valid email format are checked by the client controllers to ensure speed and responsiveness. Others such as SMS sending are performed server side to protect other users' data.