CPEN 321 Software Engineering

M6: Test Setup

UBC Explore

Group members:

Jane Shi 37998283

Xiaoqin Mei 71315980

Dylan Pither 64661267

Akshat Hari 28768299

Description:

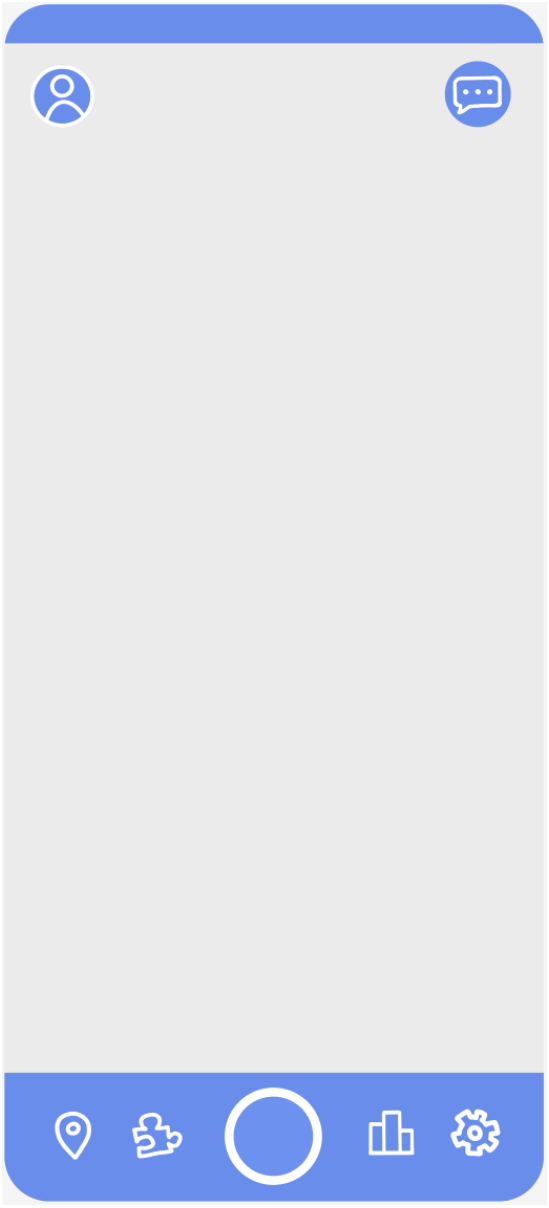
The UBC campus is large and it takes effort to explore and understand every part of it. Our app proposes to decrease the difficulty by showing the history and fun facts of UBC at the tip of your fingers. If a student or a tourist wants to know more about a part of the campus, they can select it as a destination and follow the directions on our app to get there. Once they arrive, they can direct their phone camera to the location and our app would give them a brief summary about it and direct them to related resources. To encourage people to explore, we will hide away AR cashes and custom Live2D creatures that can be found around the campus and collected for leaderboards and prizes.

- Google Maps API will be used to provide directions in our app.
- Leaderboards will be updated in real-time.

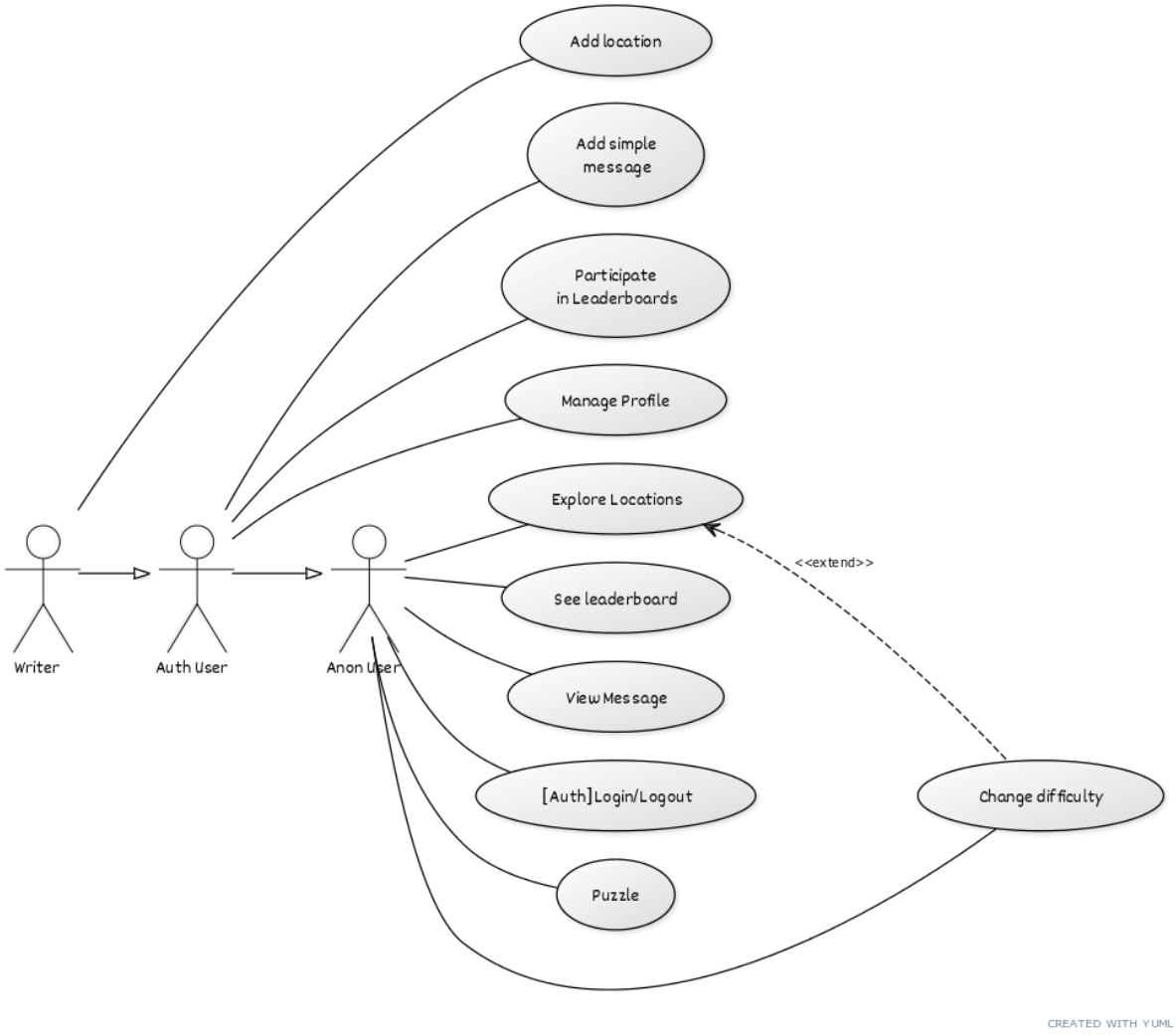
Interesting Features:

- Our app will have Artificial Reality items and custom Live2D creatures for the user to collect when they get to the location.
- Our app will also have Slick UI designs.

Main Screen Sketch:



Use Case Diagram:



Formal Use Case Specifications:

Title: Explore locations

Description: The user browses explorable locations and selects one they wish to explore. After following the provided directions to the location, information about the location is displayed.

Primary Actor: Anon User

Preconditions: None

Postconditions: The user is given information about their selected location

Main Success Scenario:

1. The user selects the location list button on the main page.
2. The app displays a list of explorable locations.
3. The user selects a location.
4. The app displays directions to the selected location using Google Maps API.
5. The user follows the directions to the location.
6. When the user arrives at the location the app displays information about the location.

Extensions:

7b. The Maps API fails

- 7b1. The application requests the user to try again.

Title: Puzzle

Description: This is an always-on feature. In the main screen with the camera, the user would find nooks and crannies to find either AR buttons or AR puzzle piece collectibles. After solving either of the puzzles they will get points in achievement as well as unlock secret locations from the location list.

Primary Actor: Anon User

Preconditions: None

Postconditions: The user solves the puzzle and gets achievements, a score and unlocks a secret location.

Main Success Scenario:

1. The user explores locations using their camera.
2. The user finds a set of AR interactable buttons
    2.1 The user presses buttons in sequence.
    2.2 The user gets achievements, a score and unlocks a secret location.
3. The user finds a puzzle piece
    3.1 The app stores the puzzle piece as user information.
    3.2 The user collects x puzzle pieces.
    3.3 The user solves the puzzle and gets achievements, a score and unlocks a secret location.

Extensions:

2a. The user does not press buttons in the correct order

- 2a1. The app displays a message saying “wrong order, please try again”.

3b. The user does not collect x puzzle pieces.

- 3b1. User puzzle piece collection will be stored in the user profile for later use.
- 3b2. Nothing happens

Title: Login/logout

Description: The user uses Google authentication to log into or out of the app.

Primary Actor: Anon User

Preconditions: To login, the user must be logged out of the app; to logout, the user must be logged into the app.

Postconditions: The user is logged into/out of the app.

Main Success Scenario:

1. The user selects the login button from the main page.
2. The app displays a dialog showing Google authentication.
3. The user can add their Google account to login.
4. The app shows the message “logged in successfully” and returns to the main page.

Extensions:

3a. Wrong credentials are entered.

- 3a1. The app displays an error message.

Title: Change difficulty

Description: The user can change their app’s difficulty by choosing between easy and medium.

Primary Actor: Anon User

Preconditions: None

Postconditions: The user has their difficulty set to the desired level.

Main Success Scenario:

1. The user selects the settings button on the main page.
2. The app displays the settings page.
3. The user selects the “change difficulty” button.
4. The user chooses between one of the two difficulty levels.
5. The user clicks “Ok”.
6. The app alerts the user of the new difficulty level that has been selected.

Extensions:

5a. The user clicks submit without clicking one of the two buttons.

- 5a1. The app requests the user to pick a difficulty level.

Title: See leaderboard

Description: The user views a leaderboard that displays authenticated users ranked by their achievements and collection score.

Primary Actor: Anon User

Preconditions: None

Postconditions: The leaderboard is presented.

Main Success Scenario:

1. The user selects the leaderboards button on the main page.
2. The app displays the global leaderboard to the user.
    2.1 If the user is an authenticated user they can select a friends tab on the leaderboards page.
    2.2 The app then displays a leaderboard where only the friends of the user are ranked.

Extensions:

2a.The app cannot retrieve the leaderboard.

- 2a1. The app displays a message stating that there was an error retrieving the leaderboard and to try again later.

2b. An anonymous user selects the friend's tab.

- 2b1. The app displays a message stating that you need to be logged in to access this feature. This message disappears after 5 seconds.

Title: View message

Description: The user can read the message left by another user at a certain location.

Primary Actor: Anon User

Preconditions: The user must be at a location where a user has left a message.

Postconditions: The message at the marked location is displayed to the user.

Main Success Scenario:

1. While walking around, the user can press on the get a message button to get notifications of any messages that are nearby.
2. The user clicks the button.
3. The app displays the message as a notification to the user.

Extensions:

2a. Internet connection error.

- 2a1. The app displays the error and asks the user to retry.

Title: Participate in leaderboards

Description: Authenticated users can share their achievements to compete with other users.

Primary Actor: Auth user

Preconditions: None.

Postconditions: The user's number of achievements is collected and ranked with other users.

Main Success Scenario:

1. The user presses the leaderboard button on the main screen.
2. The app displays the leaderboards.
3. The user presses the "Participate in Leaderboards" button to join the ranks.
4. The app collects the user's achievement information and updates the leaderboard.

Extensions:

2a. The app cannot retrieve the leaderboard.

- 2a1. The app displays a message stating that there was an error retrieving the leaderboard and to try again later.

Title: Add a simple message

Description: The user is able to choose and add a simple 240-character message that can only be viewed by going to the same location.

Primary Actor: Authenticated User

Preconditions: None

Postconditions: A message is added to the location successfully.

Main Success Scenario:

1. The user clicks the “add message” button.
2. The app shows a form to add a message.
3. The user adds the message.
4. The user presses submit.
5. The app shows a message saying that the submission is successful.

Extensions:

1a. The user is an anonymous user.

- 1a1. The user gets an error message saying that they have to login to use this feature.
- 1a2. The user goes back to the main page.

4a. The user submits an empty message.

- 4a1. The app shows a warning message: “please add something before submitting.”
- 4a2. The user rectifies and submits.

5b. The app fails to submit.

- 5b1. Asks the user to wait a while and retry later.
- 5b2. The user waits and retries the submission.

Title: Add a location

Description: The user adds a location to the list of explorable locations.

Primary Actor: Writer

Preconditions: None

Postconditions: The provided location is added to the location list.

Main Success Scenario:

1. The user selects the add location button on the location list.
2. The app displays a form to add a location.
3. The user enters a location name, coordinates, a short description, and provides an image. Optionally they can add a creature they want to appear at the location.
4. The user clicks “Add location”.
5. The app alerts the user that the location has been successfully added.

Extensions:

4a. The user leaves any field empty

- 4a1. The app alerts that all fields must be filled out, displaying which fields are empty.

4b. The user provides invalid coordinates.

- 4b1. The system alerts the user that the provided coordinates are invalid and a range that they should be in.

4c. The user enters illegal characters in the location name or description.

- 4c1. The system alerts the user of the illegal characters that the fields contain.

4f. The user does not provide a creature.

- 4f1. The app alerts the user that a default creature will be used asking the user if this is ok.
- 4f2. If the user clicks no they return to the form where they can upload a creature.

5a. The app fails to add the location

- 5a1. The app tells the user to wait to try again.

Title: Manage profile

Description: The user will be able to edit their own profile, view a progression tab for their collectible and puzzle collection, and add authenticated users as friends.

Primary Actor: Authenticated User

Preconditions: None

Postconditions: The user successfully managed/viewed desired information on their profile.

Main Success Scenarios:

1. The user clicks on the profile button on the main page.
2. The app displays the user's collection of creatures and any puzzle pieces they have obtained. The completeness of their collection is also displayed.
3. The user can click the edit display name button on their profile page to change their display name.
    3.1 The app displays a text box.
    3.2 The user types in their desired name.
    3.3 The user clicks “confirm”.
    3.4 The app displays their profile with the updated name.
4. The user can click the friends tab on the profile page to view their friends list.
    4.1 The app displays the user's friends list.
    4.2 The user clicks the “+” button on the friends list page.
        4.2.1 The app displays a text box prompting the user to enter a display name.
        4.2.2 The user enters their friend's display name.
        4.2.3 The user clicks “Send request”.
        4.2.4 The app adds the request to the user's outgoing requests and to their friend’s incoming requests.
    4.3 The user clicks the “outgoing requests” tab on the friends list page.
        4.3.1 The app shows a list of display names corresponding to unanswered friend requests.

4.4 The user clicks the “incoming requests” tab on the friends list page.

4.4.1 The app shows a list of display names corresponding to incoming friend requests from other users.

4.4.2 The user can accept or decline these requests by clicking on the display name and choose accept or decline friend request.

4.5 The user clicks “X” on the friends list page.

4.5.1 The app displays a text box prompting the user to enter a display name.

4.5.2 The user enters their friend's display name.

4.5.3 The user clicks “Remove”.

4.5.4 The app removes the friend.

Extensions:

2a. There are no creatures or puzzle pieces collected.

- 2a1. The app displays an empty page with 0/x creatures/puzzle pieces found.

3a. Network error

- 3a1. The app displays a message indicating there was a network error when trying to update the display name.

3.3a. The desired name is taken.

- 3.3a1. The app displays a message stating the desired name has already been taken and to try a different name.

3.3b. The desired name is too long or too short.

- 3.3b1. The app displays a message stating the name must be 3-20 characters.

4a. Network error

- 4a1. The app displays a message indicating failure to add/remove friend

4.1a. There are no friends of the user.

- 4.1a1. The app displays that the user has no friends and shows a button that leads to the add friends screen.

4.2.3a. The entered user does not exist.

- 4.2.3a1. The app displays that there is no user named x and to try again.

4.3a. There are no outgoing requests.

- 4.3a1. The app displays text stating that there are no outgoing friend requests.

4.4a. There are no incoming requests.

- 4.4a1. The app displays text stating that there are no incoming friend requests.

Non-Functional Requirements:

[Usability] The user should not need more than 5 clicks to perform any action.

- This requirement is relevant because it ensures that the user has fast access to all of the functionality of the app which is important for the user experience.
- We plan to test this requirement by going through each of the use cases and make sure they all need no more than 5 clicks to perform.

[Performance] AR/relevant information should show up within 1 second after the user points their phone in a certain direction at a certain location.

- This requirement is relevant because showing AR/relevant information about the location is a major functionality of our app and we don't want there to be a significant delay when showing this information.
- We plan to test this requirement by going to each of the locations and measuring the time it takes for the AR/information to show up.

[Energy efficiency] The battery life should not drop by more than 1% after continuous usage of the app for 5 minutes.

- This requirement is relevant since our app is an outdoor-based app requiring our app to be constantly active, while requiring location and camera services at the same time, which both have high energy consumption.
- We plan to test the drainage using Battery Historian for virtual deployment and battery manager apps like AccuBattery for live deployment. We will test the app under heavy use (Constantly display AR image).

[User-friendly Interface] The UI should be simple and easy to use. The icons of the buttons should be able to suggest what they are used for.

- This requirement is relevant since our app's target users are students and tourists on UBC campus. The app's interface should be easy for them to understand.
- We plan to find several students who have never used the app before and introduce the general functionality of the app. After that, we will ask them to match the usages to each button.

Android Device:

Our group has three Android devices running Android versions 9,10 and 12 which can run the front-end app.

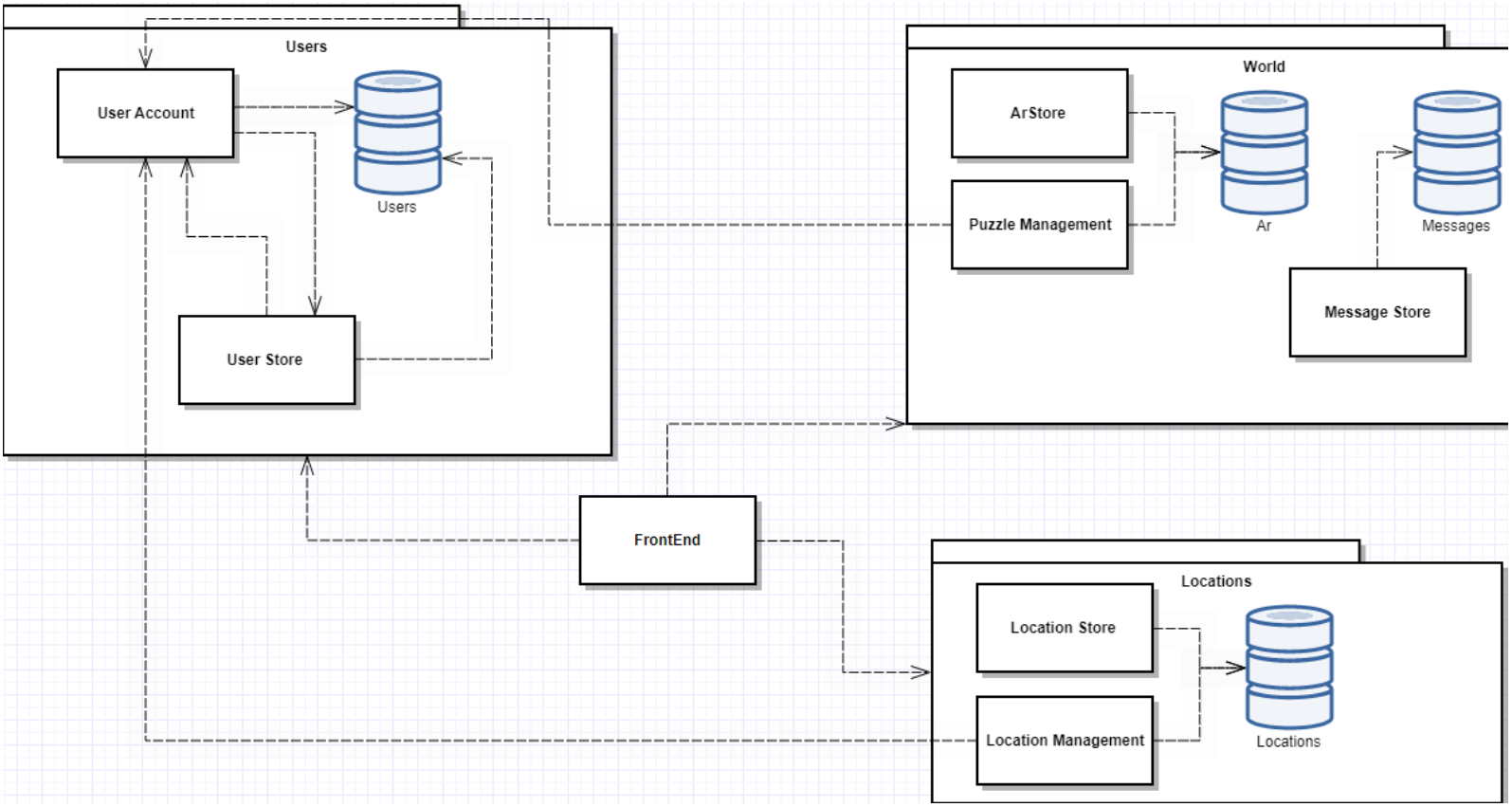
Main Modules and Sub-modules:

- Main modules:
    - Users
        - Purpose: Management of everything linked to user accounts including their collected creatures, puzzle pieces and friends. Tracks user preferences such as difficulty and if they are participating in leaderboards. Also manages global and friend leaderboards.
        - Sub-modules:
            - UserAccount
            - UserStore
    - World
        - Purpose: Management/Display of Artificial reality models and messages. Management of detailed location information as well as puzzles
        - Sub-modules:
            - ARStore
            - MessageStore
            - LocationStore

- Databases:
    - Users
        - Table to store general authenticated user information:
            - Display name, id, leaderboard participant, difficulty, score
        - Table to store friendships/requests:
            - Link two user ids with a status to represent a request or friendship
        - Table to store user collections:
            - Link an item id to a user id
        - Table to store user unlocked locations:
            - Link a location name to a user id
        - Table to store user achievements:
            - Link an achievement to a user id.
    - Messages
        - Messages sent by authenticated users into the world
        - Links one way into accounts
    - AR
        - Holds all AR components
            - Puzzles
            - Live2d models, motions and expressions
    - Locations
        - Holds information about locations
            - Location histories, fun facts, geolocation

- External components:
    - Google Authentication
        - Purpose: Get authentication for google service API
    - Google Maps
        - Purpose: Built-in Google Maps to set puzzle locations and guide users to the location

Module Dependency Diagram:



List of Interfaces:

- UserStore:
  - UserAccount findByName(String displayName)
    - Required when wanting to find another user to add as a friend or to check for a unique display name. Returns the account.
  - UserAccount createAccount(TokenInfo credentials)
    - Required if there is no existing user for the provided token. Returns the created account
  - UserAccount findById(String user_id)
    - Required to find UserAccounts from given ids.
  - UserAccount login(TokenInfo credentials)
    - Required after user validates their login token. Returns the account of the user (creating it if it does not exist).

- UserAccount login(String token)
    - Required when a user wants to retrieve their account after logging into their Google account and receiving a token. Returns their account provided the token is valid.
- UserAccount changeDifficulty(String user_id, String difficulty)
    - Required when a user wants to change their difficulty. Returns the updated account.
- UserAccount setDisplayName(String user_id, String name)
    - Required when a user wants to change their name. Returns the updated account.
- UserAccount addFriend(String user_id, String friendName)
    - Required when a user wants to send a friend request to another user. Returns the updated account, friendName is unique.
- UserAccount removeFriend(String user_id, String friendName)
    - Required when a user wants to remove another user from their friend's list. Returns the updated account, friendName is unique.
- UserAccount acceptRequest(String user_id, String friendName)
    - Required when a user wants to accept another user's friend request. Returns the updated account, friendName is unique.
- UserAccount denyRequest(String user_id, String friendName)
    - Required when a user wants to deny another user's friend request. Returns the updated account, friendName is unique.
- UserAccount participateInLeaderboard(String user_id)
    - Required when a user wants to participate in leaderboards (friend or global) which by default they are not. Returns the updated account.
- UserAccount unlockLocation(String user_id, LocationInfo location)
    - Required when a user unlocks a location from solving a puzzle. Returns the updated account.
- UserAccount unlockItem(String user_id, ARModel item)

        - Required when a user gets a new AR item (creature or puzzle piece). Returns the updated account.
    - UserAccount updateAchievements(String user_id, AchievementInfo achievement)
        - Required when a user gets a new achievement or makes progress on an achievement. Returns the updated account.
    - List getFriendsLeaderboard(String user_id)
        - Required when a user wants to view their friends leaderboard. Returns the list of only friends ordered by their score.
    - List getGlobalLeaderboard()
        - Required to retrieve the global leaderboard. Returns the top 100 account names ordered by their score.
- UserAccount:
    - UserAccount changeDifficulty(String difficulty)
        - Required to change the difficulty level of an account. Returns the updated account.
    - UserAccount setDisplayName(String name)
        - Required to change an account's display name. Returns the updated account.
    - List getFriends()
        - Required when retrieving a user’s friends list so it can be viewed in their profile or for use in the leaderboard.
    - UserAccount addFriend(String friendName)
        - Required for an account to send a request to another account. Returns the updated account, friendName is unique.
    - UserAccount removeFriend(String friendName)
        - Required for an account to remove a friend from its friends list. Returns the updated account, friendName is unique.
    - UserAccount acceptRequest(String friendName)
        - Required for an account to accept a friend request from another account. Returns the updated account, friendName is unique.

    - UserAccount denyRequest(String friendName)
        - Required for an account to deny a friend request from another account. Returns the updated account, friendName is unique.
    - UserAccount participateInLeaderboard()
        - Required for an account to participate in leaderboards (friend or global) by default accounts are not participating in leaderboards. Returns the updated account.
    - UserAccount unlockLocation(LocationInfo location)
        - Required for an account to add a location to its unlockedLocation list. Returns the updated account.
    - UserAccount unlockItem(ARModel item)
        - Required for an account to add AR item ids to its item list. Returns the updated account.
    - UserAccount updateAchievements(AchievementInfo achievement)
        - Required for an account to update its achievements list upon new acquiring new achievements or making progress. Returns the updated account.
    - List getFriendsLeaderboard()
        - Required for an account to retrieve the leaderboard of a users friends. Returns the list of friends ordered by their score.
- Users Database:
    - Int updateAchievements(AchievementInfo)
        - Required when a user gets to a new location and their achievements need to be updated
    - Int updateCollection(Collection)
        - Required when a user gets new AR collections to be updated in their profile.
    - Int updateUserLocation(Location,UserAccount)
        - Required when the user unlocks a location from solving puzzles
    - result createFriendRequest(String user_id, String friend_id)

        - Required when a user sends a friend request to another user. Returns an object describing the result of the database insertion.
    - result removeFriendship(String user_id, String friend_id)
        - Required when a user wants to remove another user from their friends list. Returns an object describing the result of the database deletion.
    - result acceptFriendRequest(String user_id, String friend_id)
        - Required when a user wants to accept a friend request, changing the status from pending to friends. Returns an object describing the result of the database update.
    - result removeFriendRequest(String user_id, String friend_id)
        - Required when a user wants to deny a friend request. Returns an object describing the result of the database deletion.
    - result updateDifficulty(String user_id, String difficulty)
        - Required when a user wants to change the difficulty. Returns an object describing the result of the database update.
    - result participateInLeaderboard(String user_id)
        - Required when a user requests to participate in leaderboards. Returns an object describing the result of the database update.
    - List getGlobalLeaderboard()
        - Required when a user wants to view the global leaderboard. Returns a list of up to 100 names ordered by their score.
    - List getFriendsLeaderboard(List friends)
        - Required when a user wants to view their friends leaderboard. Returns a list of only accounts in the provided list ordered by their score.
    - result updateDisplayName(String user_id, String name)
        - Required when a user wants to change their name. Returns an object describing the result of the database update.

- MessageStore:

        - Message getMessage(coordinates)
            - Required when a user gets to a certain coordinates, the app retrieves messages left at the location.
        - Int addMessage(Message)
            - Required when an authenticated user wishes to add a message at a certain location.
- ARStore:
    - ARModel getARModel(coordinates)
        - Required when a user gets to a certain coordinates, the app retrieves AR models at the location.
    - Int addARModel (ARModel)
        - Required when a writer wants to add an AR model to the app.
- LocationStore:
    - LocationInfo getLocationInfo(coordinates)
        - Required when a user gets to certain coordinates, the app retrieves location information.
    - Int addLocation(LocationInfo, ARModel)
        - Required when a writer wants to add a location to the app.
    - List getLocationList(UserAccount)
        - Required when a user wants to view the list of explorable locations
    - Void getDirections(String locationName)
        - Required when a user wants to get directions to a location, will either open google maps directions to the location or show an image depending on user difficulty level.
- Locations Database:
    - LocationInfo getLocationInfo(coordinates)
        - Required when a user gets to certain coordinates, the app retrieves location information.
    - Int addLocationInfo(LocationInfo)
        - Required when a writer wants to add a location to the app.
    - LocationInfo getLocation(String locationName)

        - Required when a user wants to get directions to a selected location.
    - List getLocations()
        - Returns the list of explorable locations
- AR Database:
    - ARModel getARModel(coordinates)
        - Returns the AR model at the location coordinates
    - Int addARModel(ARModel)
        - Required to add AR models to the database.
- Messages Database:
    - Message getMessage(coordinates)
        - Required when a user gets to a certain coordinates, the app retrieves messages left at the location.
    - Int addMessage(Message)
        - Required when an authenticated user wishes to add a message at a certain location.

DataTypes:

TokenInfo (https://developers.google.com/identity/sign-in/android/backend-auth){

String: iss,

String: sub (unique id),

String: azp,

String: aud,

String: iat,

String: exp,

String: email,

String: email_verified,

String: name,

String: picture,

String: given_name,

String: family_name,

String: locale

}

AchievementInfo{

String id,

String Type,

Int points,

String Image.Id,

}

ARModel (https://www.live2d.com/en/download/sample-data/) {

Int id,

Model data (cmo3)

Basic motions (can3)

Set of files for embedding (runtime folder)

• Model data (moc3)

• Motion data (motion3.json)

• Model settings file (model3.json)

• Physics settings file (physics3.json)

• Display auxiliary file (cdi3.json)

}

Coordinates{

int : longitude

Int: latitude

}

LocationInfo{

Int id,

Coordinates,

String location_name,

String: Fun Facts,

String: Related Links,

String: About,

String: Image,

UserAccount: creator,

String: public_access:

}

Message{

Int id,

Coordinates,

String: message,

UserAccount: creator,

}

UserAccount{

Collection{

List[AchievementInfo] achievements,

List[ARModel.id] items,

}

List[UserAccounts.displayName] friends,

List[LocationInfo.location_name] unlockedLocations,

String: difficulty,

Boolean: leaderboardParticipant,
String: displayName,
List[UserAccounts.displayName]: incomingRequests,
List[UserAccounts.displayName]: outgoingRequests,
int: score,
String: id
}

Sequence Diagrams:

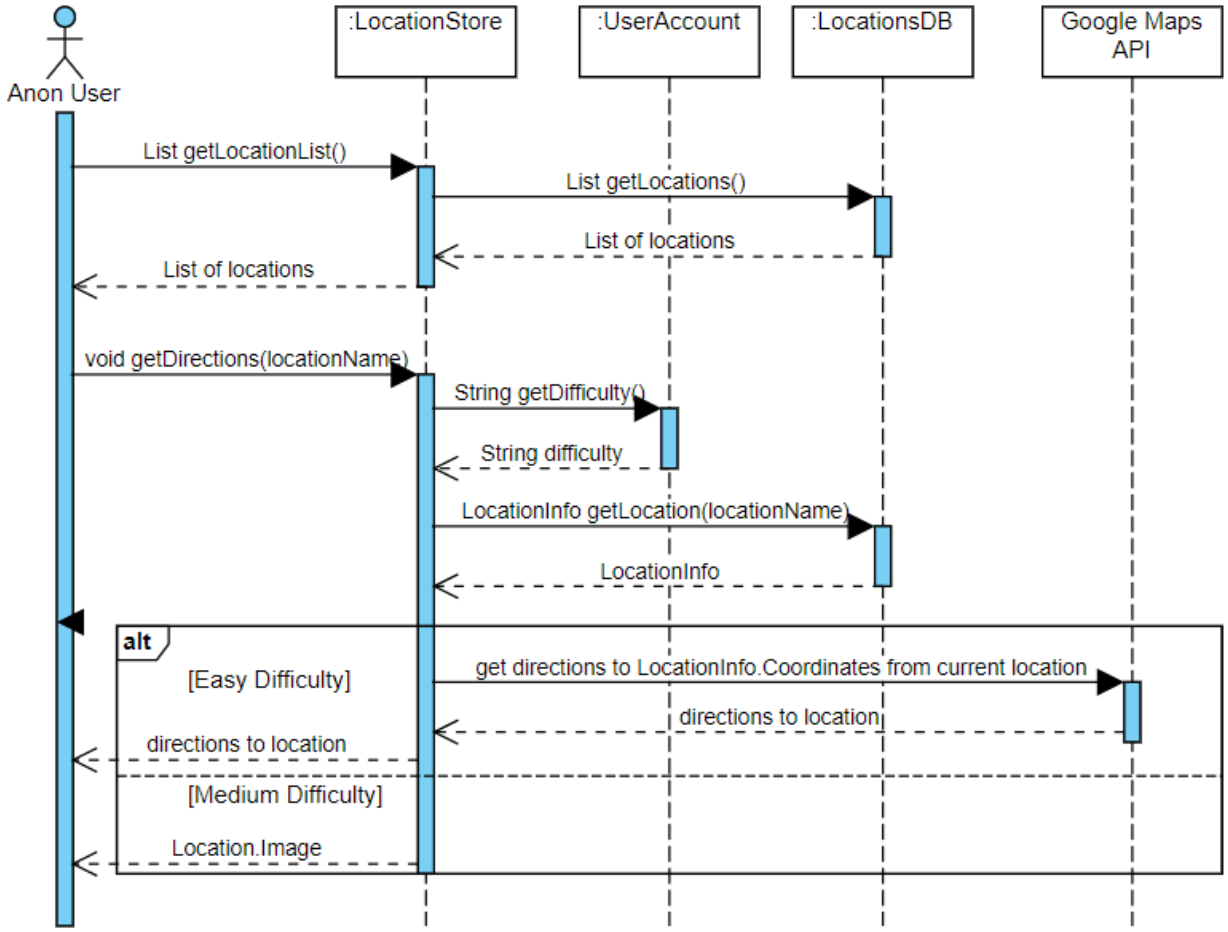
We assume the following meaning of return values in our sequence diagram:

1: Success

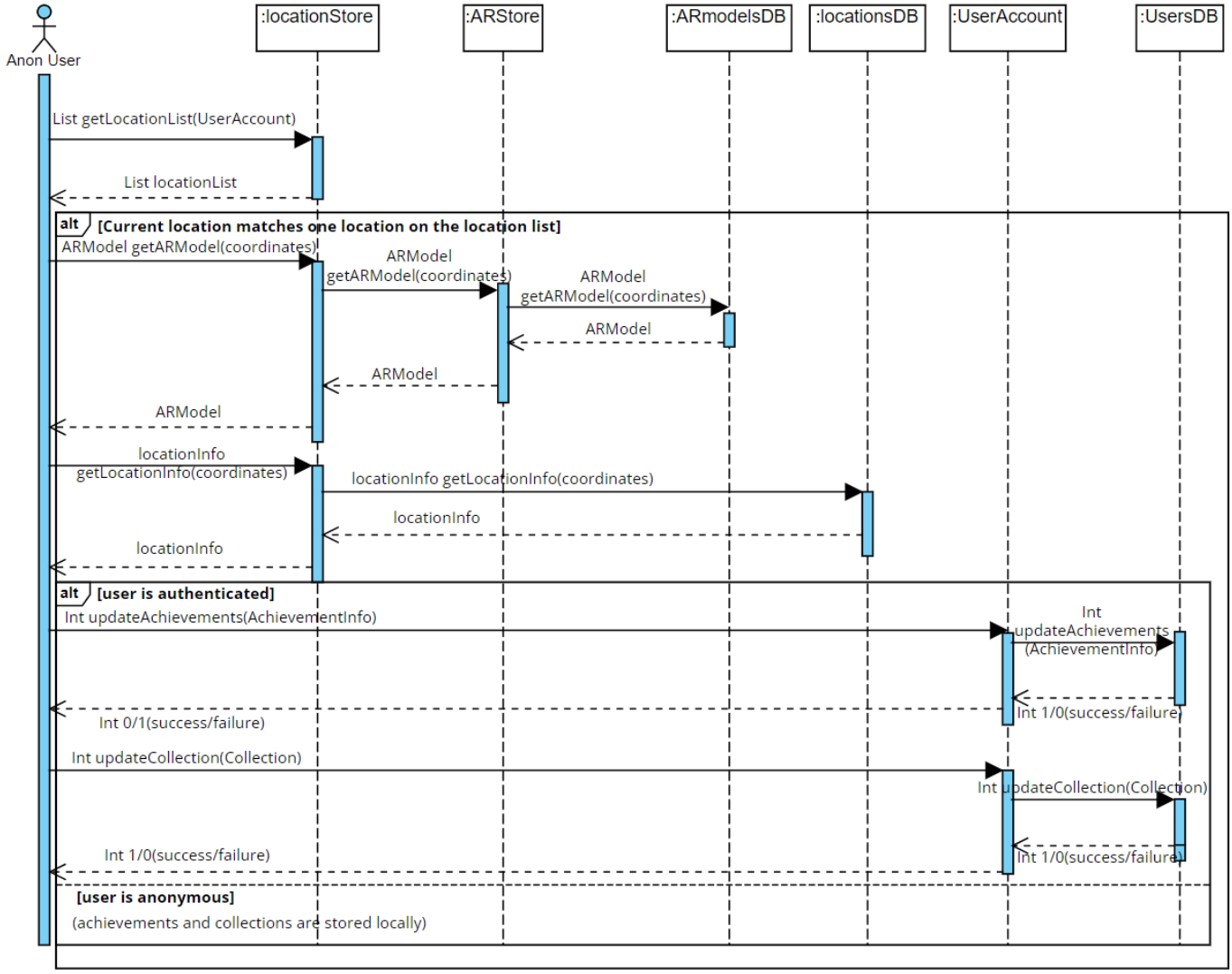
0: Failure

- **Explore locations**

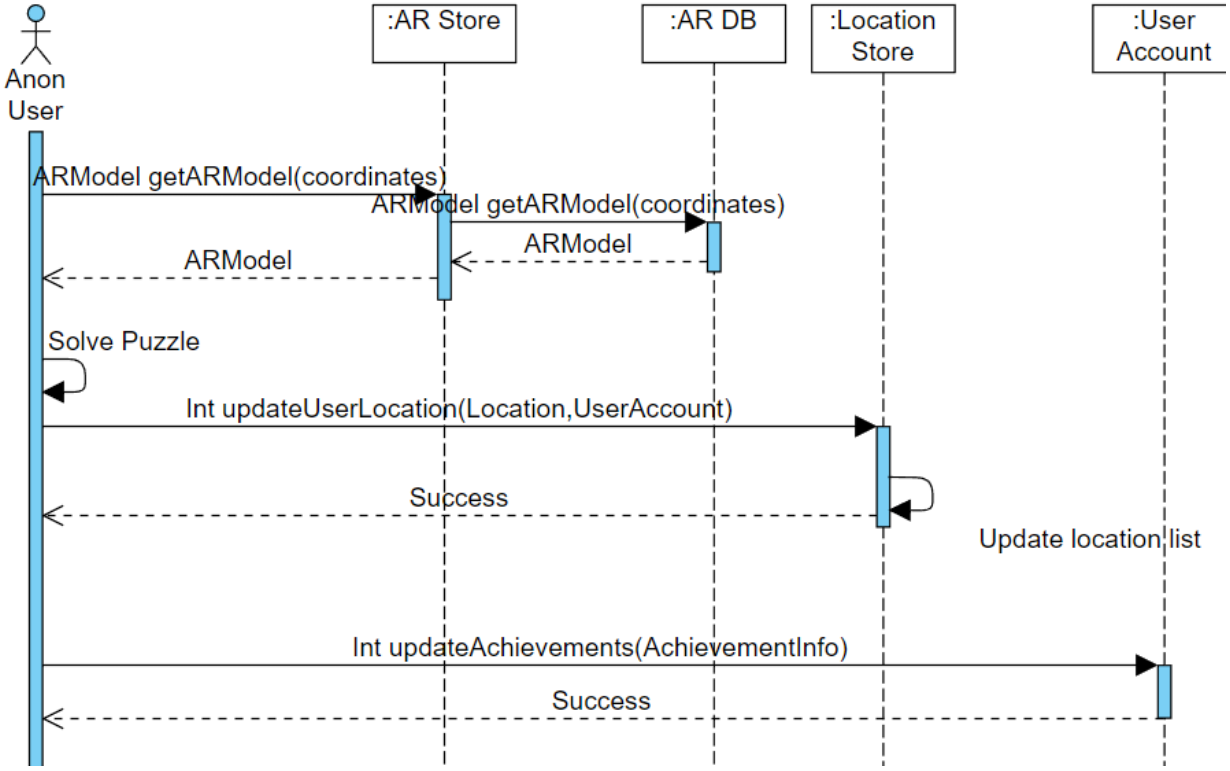
  Getting directions to the location:



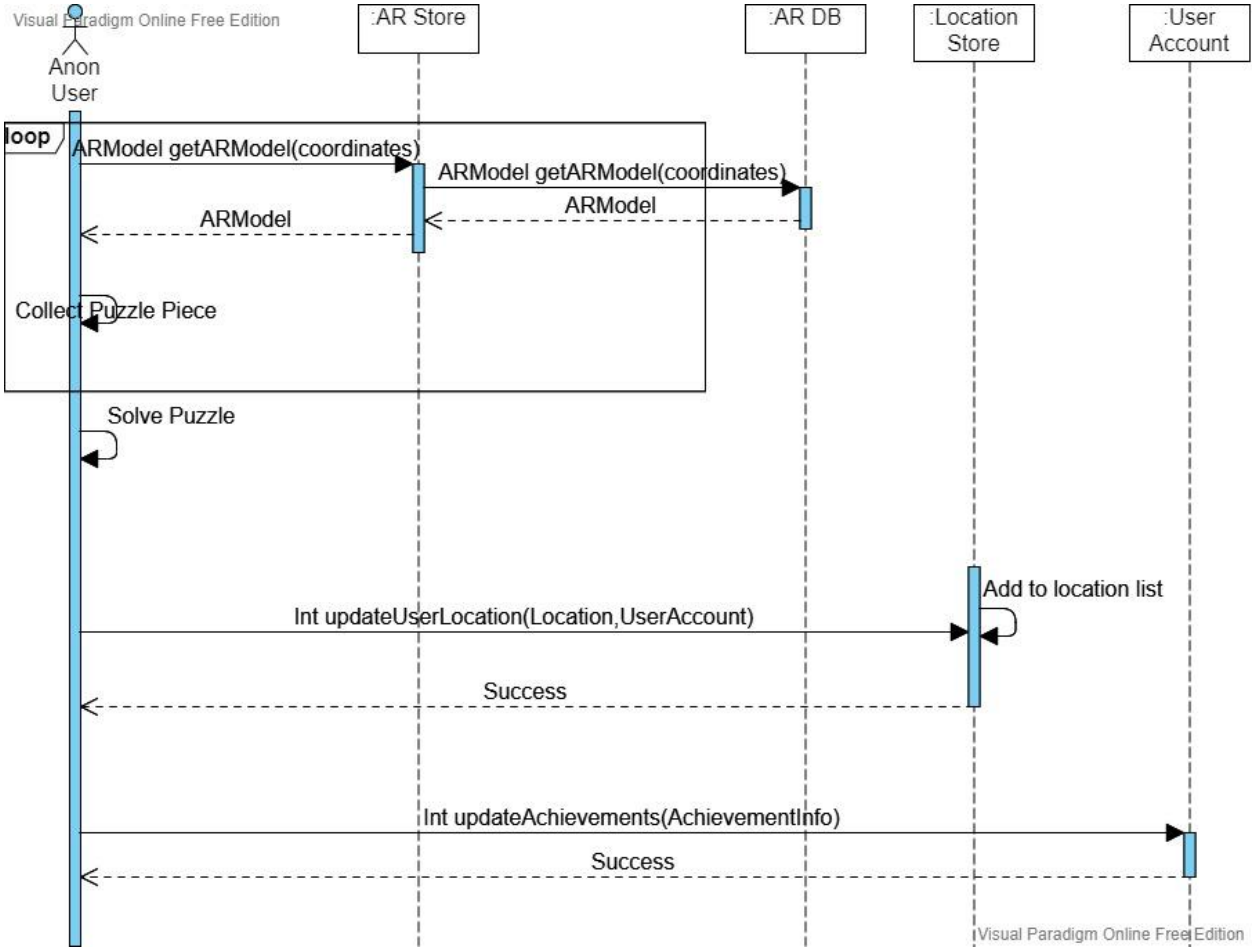
After reaching the location:



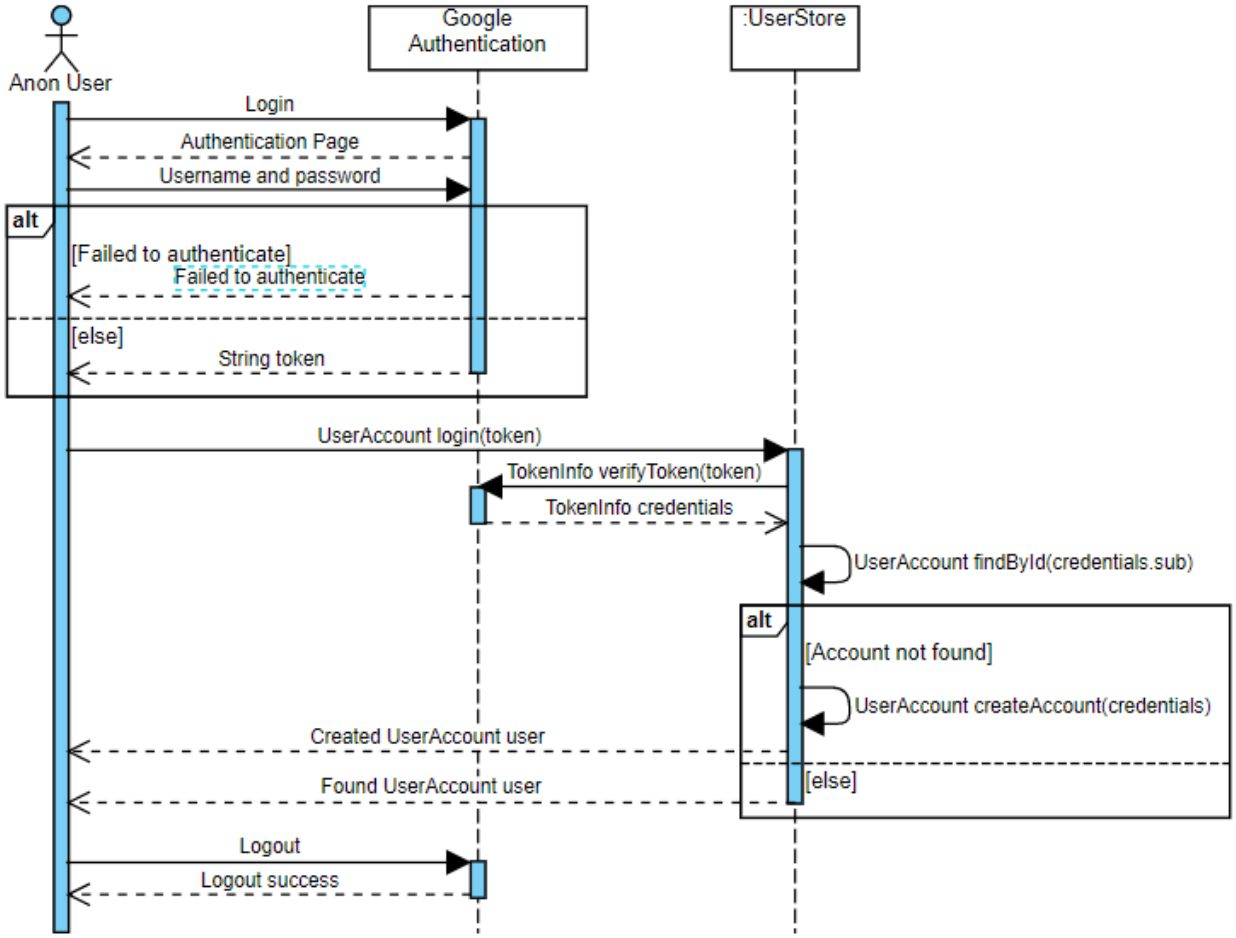
- **AR buttons**


Anon User
:AR Store
:AR DB
:Location Store
:User Account
ARModel getARModel(coordinates)
ARModel getARModel(coordinates)
ARModel
ARModel
Solve Puzzle
Int updateUserLocation(Location,UserAccount)
Success
Update location list
Int updateAchievements(AchievementInfo)
Success

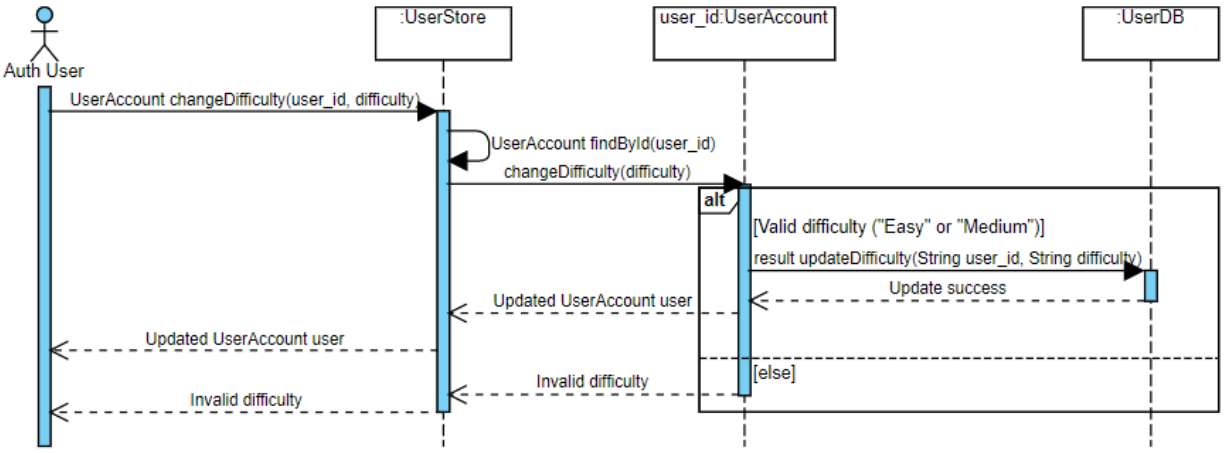
- **AR puzzles**


Visual Paradigm Online Free Edition
Anon User
:AR Store
:AR DB
:Location Store
:User Account
loop
ARModel getARModel(coordinates)
ARModel getARModel(coordinates)
ARModel
ARModel
Collect Puzzle Piece
Solve Puzzle
Add to location list
Int updateUserLocation(Location,UserAccount)
Success
Int updateAchievements(AchievementInfo)
Success
Visual Paradigm Online Free Edition

- **Login/Logout**

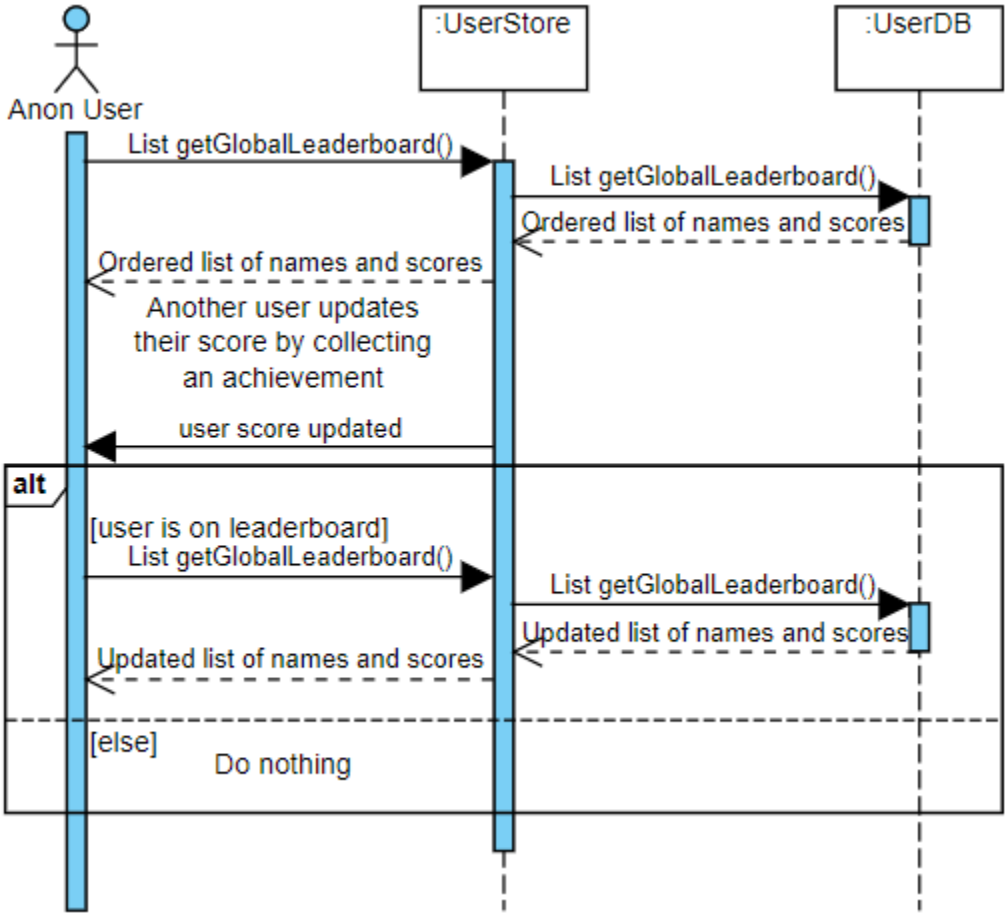


- **Change difficulty**

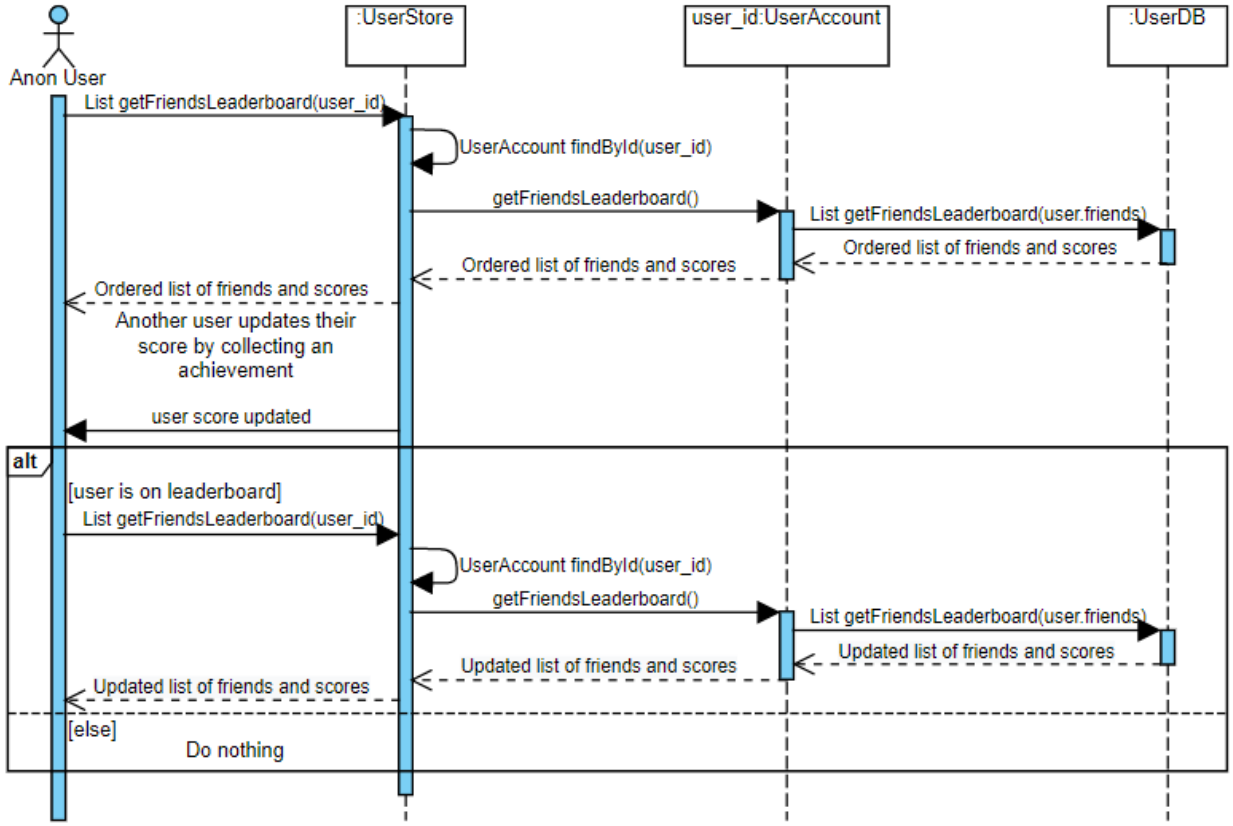


- **See leaderboard**

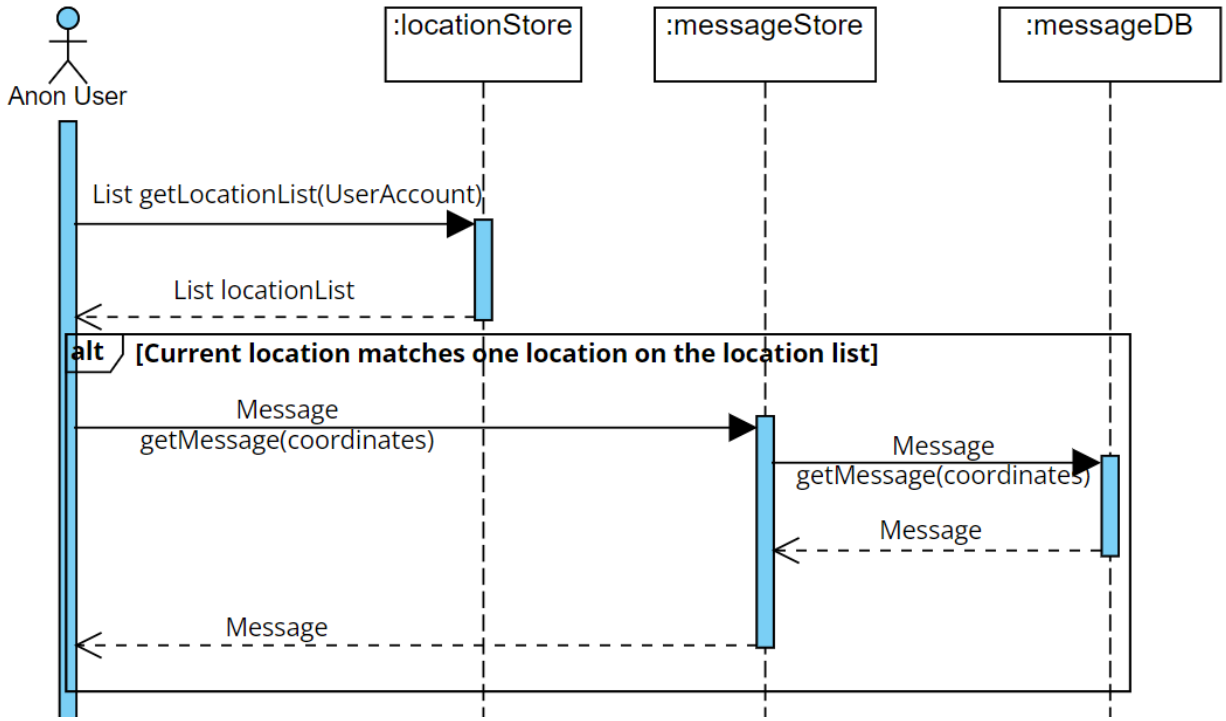
Viewing the global leaderboard as Anon user:



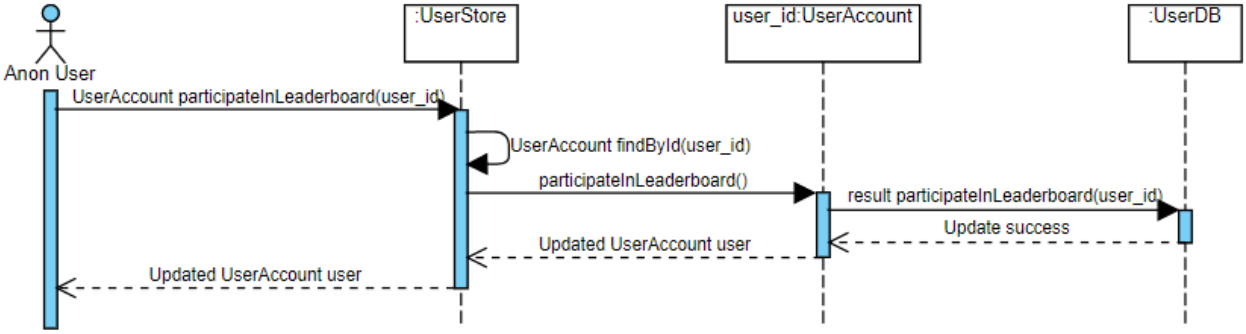
Viewing the friend leaderboard as Auth user:



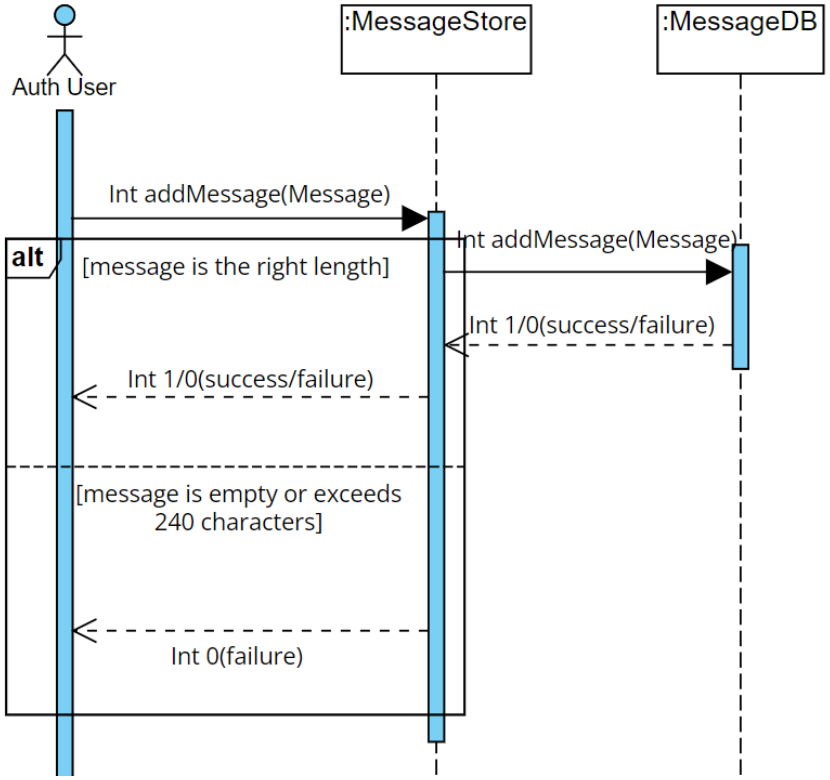
- **View message**



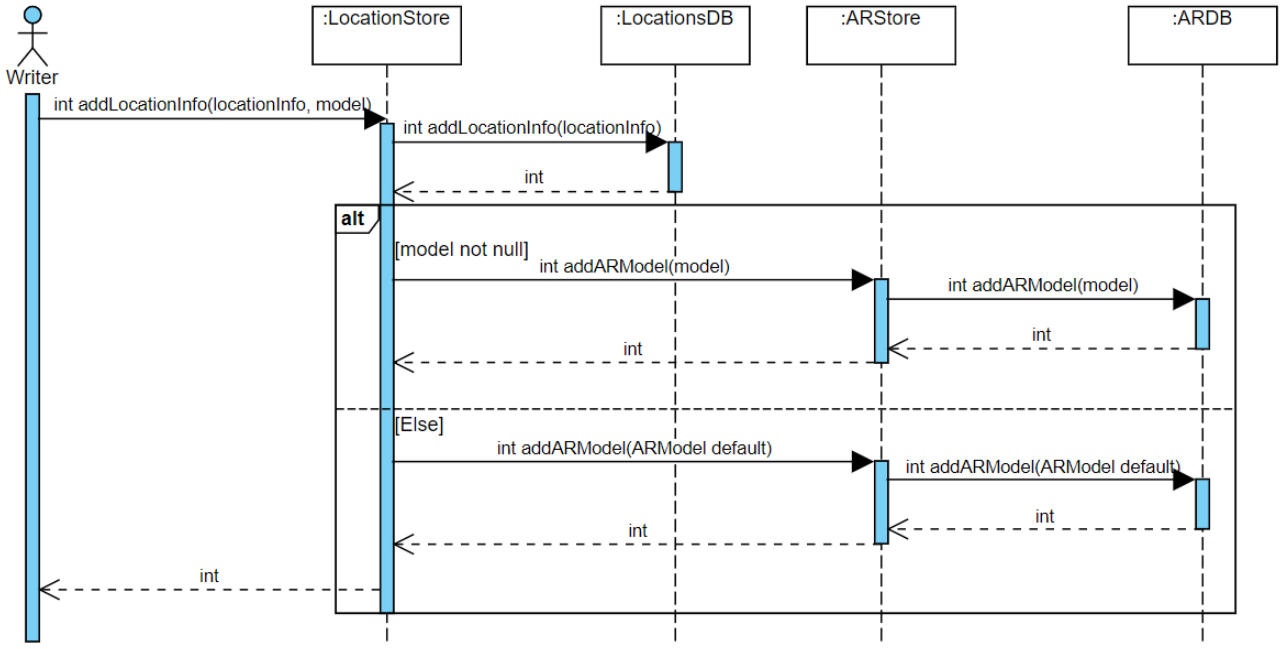
- **Participate in leaderboards**



- **Add a simple message**

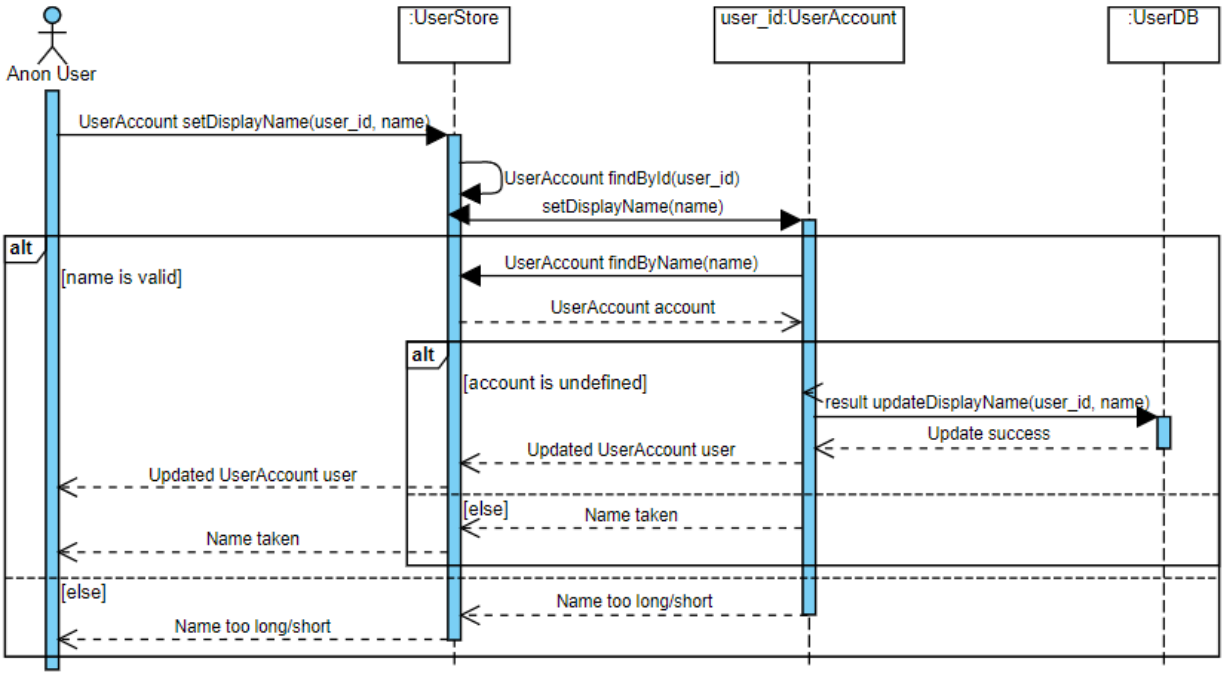


- **Add a location**

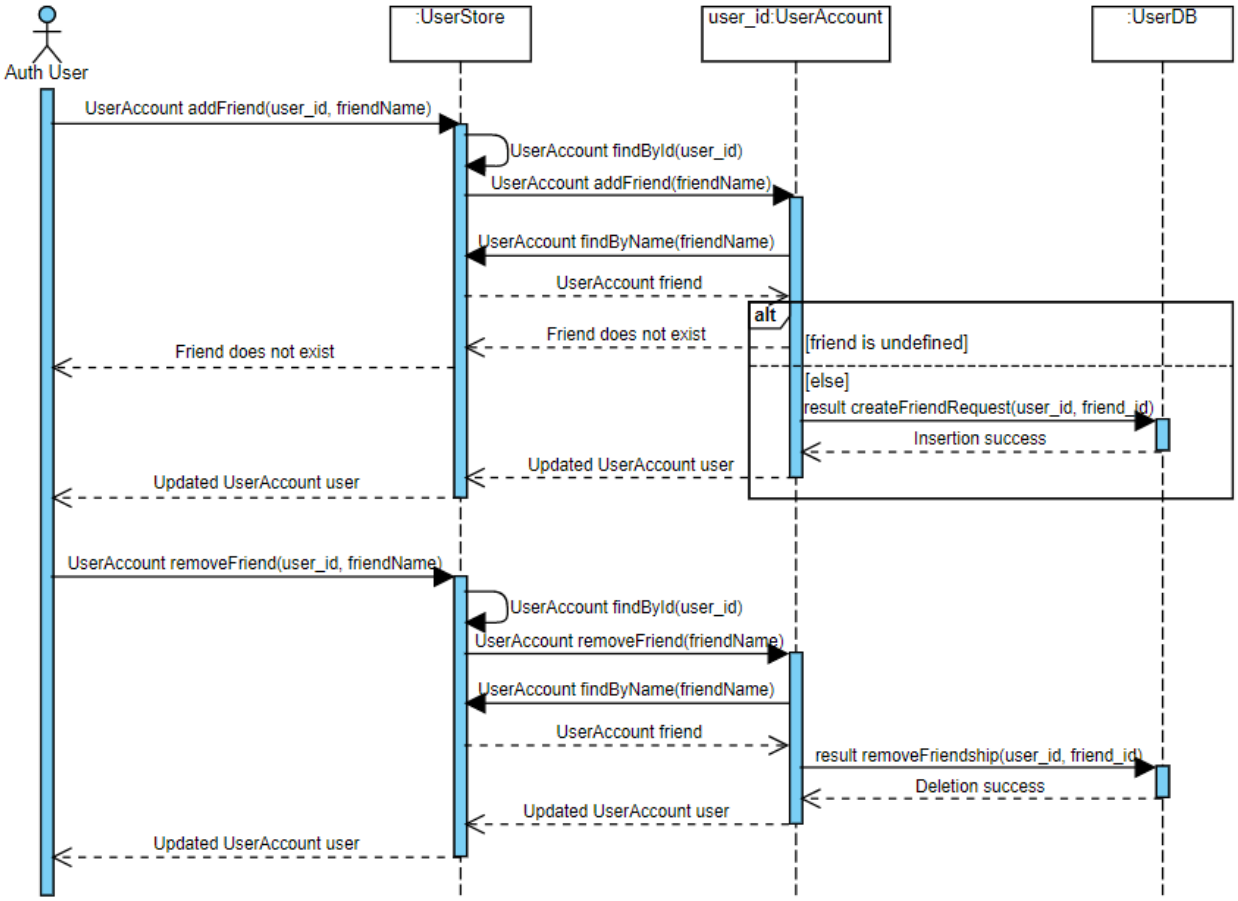


- **Manage profile**

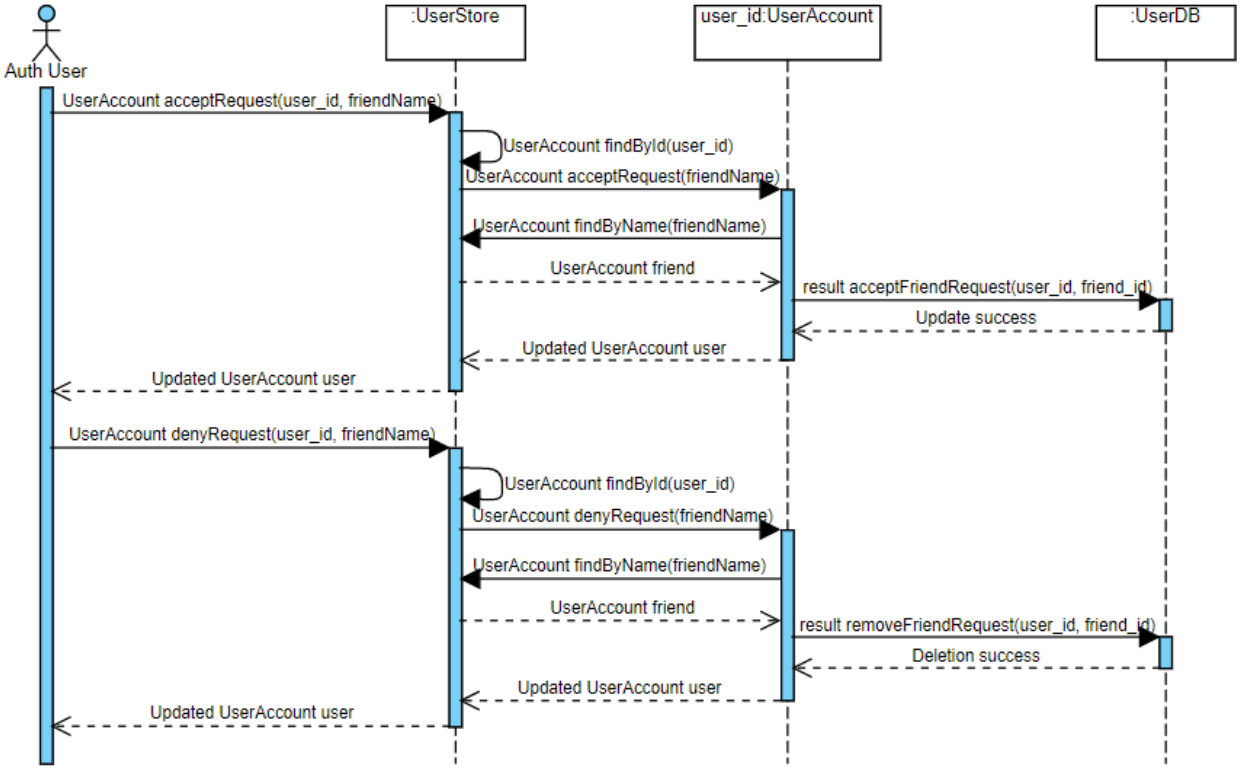
Changing name:



Sending requests and removing friends:



Managing incoming requests:



Realizing Non-Functional Requirements:

[Usability] The user should not need more than 5 clicks to perform any action.

- We will implement this requirement by making sure that features should only go to a minimum depth of 5.
- If it exceeds that we will
    - Reconsider the feature
    - Add a shortcut to the feature
- Each screen or button press will be automated into a list, our test will check if the list cascading is less than 5

[Performance] AR/relevant information should show up within 1 second after the user points their phone in a certain direction at a certain location.

- We plan to implement this feature making sure that all AR relevant features are pre-downloaded to the app for fast access
- We plan to test this requirement by simulating the camera using predefined pictures and locations and then measuring the time it takes for the AR/information to show up.

[Energy efficiency] The battery life should not drop by more than 1% after continuous usage of the app for 5 minutes.

- We plan to make sure that camera/internet only activates if our application detects that the phone is in a predestined location
- We plan to test the app before a feature is added to make sure it goes under our requirements
- We plan to test the drainage using Battery Historian for virtual deployment and battery manager apps like AccuBattery for live deployment. We will test the app under heavy use (Constantly display AR image).

[User-friendly Interface] The UI should be simple and easy to use. The icons of the buttons should be able to suggest what they are used for.

- All interactable elements should be at least 48 x 48 dp
- Neutral and Friendly Color Theming
- We will standardize the font usage to 2 font families
- Make sure that the minimum font size for our app is 12pt

- We plan to find several students who have never used the app before and introduce the general functionality of the app. After that, we will ask them to match the usages to each button.

Design For Beyond the Minimal Scope Functionality:

- AR animation:

  We plan to combine Live2d Cubism SDK for Unity and Google ARCore to create interactive AR puzzles for users. Live2d Cubism is an animation software that can create 2D models based on layers of images. Live2d animations occupy fewer resources compared to 3D models while being smoother than traditional frame-by-frame animations.

  Our team will design and build original live2d models. By changing model parameters, we can present various animations to users. We can also let users interact with the models by touching or dragging them. This will encourage users to challenge puzzles to collect these creatures as rewards, and so explore more places.

Frameworks:

Google ARCore

- In built functionality with Android devices
- Cheap
- Well documented with community support

Live2d Cubism

- Smoother and easier to make compared to frame-by-frame animation
- Occupies fewer resources than using 3D models
- Easy to control animations with automatically generated .json files

Unity

- Allows in-built integration between Cubism and ARCore
- Easy to use for modeling and implementation

MySQL vs MongoDb

- Our dataset is interlinked between user accounts, location information, AR model information, and Message information
- Our dataset is horizontally fixed, potential addition of columns are rare
- Accessing multi-table information is easier with MySQL

Node.js vs Python

- Required for Project
- Node.js is faster
- Has inbuilt functions for Multi-Threading (useful for servers)
- Scalable

Link to Git Repository:

https://github.com/jane-cz/CPEN321-final-project

Public IP of Server and APIs:

20.228.168.55

Users module:

- POST http://20.228.168.55/users/login
  - Request content: {“token”: ”value”}
    - Value is the token received after Google sign in
- PUT http://20.228.168.55/users/user_id/difficulty
  - user_id is a parameter
    - The id of the account to change the difficulty of
  - Request content: {“difficulty”: “value”}
    - Value is the difficulty the user wishes to change too
- PUT http://20.228.168.55/users/user_id/displayName
  - user_id is a parameter
    - The id of the account to change the name of
  - Request content: {“displayName”: “value”}
    - Value is the name that the user wishes to change their name too
- POST http://20.228.168.55/users/user_id/friends
  - user_id is a parameter
    - The id of the account that is sending a request
  - Request content: {“displayName”: “value”}
    - Value is the name the user wishes to send a friend request too
- DELETE http://20.228.168.55/users/user_id/friends/displayName
  - user_id is a parameter
    - The id of the account that is removing a friend
  - displayName is a parameter

        - Name of the friend the user wishes to delete off their friends list
- PUT http://20.228.168.55/users/user_id/requests
    - user_id is a parameter
        - The id of the account that is accepting a request
    - Request content: {“displayName”: “value”}
        - Value is the name the user wishes to accept a friend request from
- DELETE http://20.228.168.55/users/user_id/requests/displayName
    - user_id is a parameter
        - The id of the account that is deny a request
    - displayName is a parameter
        - Name of the friend the user wishes to deny a request from
- PUT http://20.228.168.55/users/user_id/participateInLeaderboard
    - user_id is a parameter
        - The id of the account that is going to participate in leaderboards
- POST http://20.228.168.55/users/user_id/locations
    - user_id is a parameter
        - The id of the account that is unlocking a location
    - Request content: {“location_name”: “value”}
        - Value is the name of the location the user has unlocked
- POST http://20.228.168.55/users/user_id/items
    - user_id is a parameter
        - The id of the account that its unlocking an item
    - Request content: {“id”: “value”}
        - Value is the id of the item the user has unlocked
- PUT http://20.228.168.55/users/user_id/achievements
    - user_id is a parameter
        - The id of the account that is updating achievements

        - Request content: {“id”: “value1”, “Type”: “value”, “points”: value, “image”: “value”}
            - The fields of an achievement (see AchievementInfo)
    - GET http://20.228.168.55/users/user_id/leaderboard
        - user_id is a parameter
            - The id of the account to get the friends leaderboard of
    - GET http://20.228.168.55/users/leaderboard

Locations module:

- GET http://20.228.168.55/locations/
- POST http://20.228.168.55/locations/
    - Request content: { "location_name": "value", "coordinate_latitude":value, "coordinate_longitude":value, "fun_facts":"value", "related_links":"value", "about":"value", "image_url":"value" }
- GET http://20.228.168.55/locations/location_name
    - location_name is a parameter
        - The name of the location to be retrieved
- PUT http://20.228.168.55/locations/location_name
    - location_name is a parameter
        - The name of the location to be updated
    - Request content: { "location_name": "value", "coordinate_latitude":value, "coordinate_longitude":value, "fun_facts":"value", "related_links":"value", "about":"value", "image_url":"value" }
- DELETE http://20.228.168.55/locations/location_name
    - location_name is a parameter
        - The name of the location to be deleted
- GET http://20.228.168.55/locations/user/user_account_id
    - user_account_id is a parameter
        - The id of the account to get unlocked locations of

World module:

- GET http://20.228.168.55/messages/
- POST http://20.228.168.55/messages/
    - Request content: { "coordinate_latitude":value, "coordinate_longitude":value, "message_text":"value", "user_account_id":value }
- GET http://20.228.168.55/messages/id
    - id is a parameter
        - The id of the message to be retrieved
- PUT http://20.228.168.55/messages/id
    - id is a parameter
        - The id of the message to be updated
    - Request content: { "coordinate_latitude":value, "coordinate_longitude":value, "message_text":"value", "user_account_id":value }
- DELETE http://20.228.168.55/messages/id
    - id is a parameter
        - The id of the message to be deleted

Plan for Implementing Complexity Idea:

The next step of our project is to integrate AR modules and puzzles into our application. This includes creating AR modules using unity and Live2dCubism. Also integrating those modules with google AR core. The management of the AR will be done on the backend while the camera integration with the models will be done on the frontend.

Changes in Project Scope:

None.

Tool Setup:

Mocked backend module and interfaces:

Users Module:

- login
- addFriend
- removeFriend
- acceptRequest
- denyRequest
- participateInLeaderboard
- setDisplayName
- changeDifficulty
- unlockLocation
- unlockItem
- updateAchievements
- getFriendsLeaderboard
- findById
- getGlobalLeaderboard

Location Module

- addLocation
- deleteLocation
- updateLocation
- getLocationsByParameters

World Module

- addMessage
- deleteMessage
- updateMessage
- getMessageByParameters

UI Component tested:

Tested pressing the add location button on the main screen leads to the input_name text box being displayed.

Github Actions:

- .github\workflows\jest-backend-test.yml

Test Design:

Back-end:

Users Module, UserStore: UserAccount findByName(String displayName):

Scenario	**Input**	**Expected Result**
Account with name exists	“John Doe”	Return: UserAccount of John Doe
Account with name does not exist	[assumes: Doe John is not the name of any account] “Doe John”	Return: Undefined
Empty string provided	“ “	Return: Undefined
Provided name undefined	undefined	Return: Undefined
SQL Injection	[assumes: Test is not the name of any account] “Test\’ OR \’1 = 1”	Return: Undefined

Users Module, UserStore: UserAccount findById(String user_id):

Scenario	**Input**	**Expected Result**
Account with id exists	“1”	Return: UserAccount of John Doe
Account with id does not exist	[assumes: -1 is not the id of any account] “-1”	Return: Undefined
Empty string provided	“ “	Return: Undefined
Provided id undefined	undefined	Return: Undefined
SQL Injection	[assumes: 123 is not the id of any account] “123\’ OR \’1 = 1”	Return: Undefined

Users Module, UserStore: UserAccount login(TokenInfo credentials)

(credentials.sub (id) and credentials.name only parts used in this interface)

Scenario	**Input**	**Expected Result**
Successfully login	credentials{sub: 12345, name: “John Doe”}	Return: The UserAccount of the provided credentials.
Successful login of new account	[assumes: There is no account with id 1234] credentials{sub: 1234, name: “Doe John”}	Return: The newly created UserAccount Account added to the database
Credentials has no id	credentials{name: “John Doe”}	Return: undefined
Credentials has no name and account does not already exist	[assumes: There is no account with id 1234] credentials{sub: 1234}	Return: Error (“Unable to create account, no name provided”) Database not updated

Users Module, UserStore: UserAccount createAccount(TokenInfo credentials)

(credentials.sub (id) and credentials.name only parts used in this interface)

Scenario	**Input**	**Expected Result**
Successfully create an account	credentials{sub: 1234, name: “Doe John”}	Return: The newly created UserAccount Account added to the database
Account with name exists	[assumes: There exists an account with name John Doe] credentials{sub: 12345, name: “John Doe”}	Return: The newly created UserAccount with the name John Doe1 Account added to the database
Account with id exists	[assumes: There exists an account with id 1] credentials{sub: 1, name: “Johnny Doe”}	Return: Error(“Account with id already exists”) Database not updated
Credentials has no id	credentials{name: “Doe John”}	Return: Error(“No id provided”) Database not updated
Credentials has no name	credentials{sub: 1234}	Return: Error(“No name provided”) Database not updated

Users Module, UserStore: List getGlobalLeaderboard()

Scenario	**Input**	**Expected Result**
No users exist		Return: []
No users are participating in leaderboards		Return: []
The leaderboard has participants	[assume: JohnDoe1, JohnDoe2, JohnDoe3 are all participating in leaderboards and have scores 20,10,5]	Return:[{displayName: “John Doe1”, score: 20}, {displayName: “John Doe2”, score 10},{displayName: “John Doe3”, score 5}]

Users Module, UserAccount: UserAccount participateInLeaderboard()

Scenario	**Input**	**Expected Result**
User is not currently participating in leaderboard		Return: Updated UserAccount (account.leaderboardParticipant = 1) Database sets account to be a leaderboard participant
User is already participating in leadboard		Return: Unchanged UserAccount Database not updated

Users Module, UserAccount: UserAccount changeDifficulty(String difficulty)

Scenario	**Input**	**Expected Result**
Successfully change difficulty	[assumes: The previous difficulty was Easy] “Medium”	Return: Updated UserAccount (account.difficulty = “Medium”) Database sets account difficulty to the new difficulty
Difficulty is the same as before	[assumes: The previous difficulty was Easy] “Easy”	Return: Unchanged UserAccount Database not updated
Provided difficulty is not	“Impossible”	Return: Error(“Invalid

“Easy” or “Medium”		difficulty”) Database not updated

Users Module, UserAccount: UserAccount addFriend(String friendName)

Scenario	**Input**	**Expected Result**
Successfully send request to provided name	[assumes: Johnny Doe is another user] “Johnny Doe”	Return: Updated UserAccount (account.outGoingRequests contains John Doe) Database relationship created with status pending
Account with provided name does not exist	[assumes: No account has name Doe John] “Doe John”	Return: Error(“Account does not exist”) Database not updated
Provided name undefined	undefined	Return: Error(“Account does not exist”) Database not updated
Sending a request to self	[assumes: This account is named John Doe] “John Doe”	Return: Error(“Cannot send requests to self”) Database not updated
Already friends with the provided name	[assumes: This account is already friends with Anne Smith] “Anne Smith”	Return: Error(“Already friends with this user”) Database not updated
Already sent a request to the provided name	[assumes: This account has already sent a request to Johnny Doe] “Johnny Doe”	Return: Error(“Already sent a request to this user”) Database not updated
Provided name already sent a request to this account	[assumes: Joe Shmoe already sent a request to this account] “Joe Shmoe”	Return: Error(“Already been sent a request by this user”) Database not updated

Users Module, UserAccount: UserAccount removeFriend(String friendName)

Scenario	**Input**	**Expected Result**
Successfully remove provided name	“Anne Smith”	Return: Updated UserAccount

		(account.friends no longer contains Anne Smith) Database relationship deleted.
Account with provided name does not exist	[assumes: No account has name Doe John] “Doe John”	Return: Error(“Account does not exist”) Database not updated
Provided name undefined	undefined	Return: Error(“Account does not exist”) Database not updated
Not friends with the provided name	[assumes: Joe Shmoe is not friends with this account] “Joe Shmoe”	Return: Error(“Not friends with this user”) Database not updated
Removing self	[assumes: This account is named John Doe] “John Doe”	Return: Error(“Cannot remove self”) Database not updated

Users Module, UserAccount: UserAccount acceptRequest(String friendName)

Scenario	**Input**	**Expected Result**
Successfully accept a request from provided name	“Joe Shmoe”	Return: Updated UserAccount(Joe Shmoe now in account.friends instead of account.incomingRequests) Database relationship status updated to friends
Account with provided name does not exist	[assumes: No account has name Doe John] “Doe John”	Return: Error(“Account does not exist”) Database not updated
Provided name undefined	undefined	Return: Error(“Account does not exist”) Database not updated
No request from provided name	[assumes: No request from Johnny Doe] “Johnny Doe”	Return: Error(“No request from this user”) Database not updated
Accepting a request from	[assumes: This account is	Return: Error(“Cannot

self	named John Doe] “John Doe”	accept a request from self”) Database not updated
Already friends with the provided name	[assumes: This account is already friends with Anne Smith] “Anne Smith”	Return: Error(“Already friends with this user”) Database not updated
Account already sent a request to provided name	[assumes: This account has sent a request to Joe Smith] “Joe Smith”	Return: Error(“Cannot accept a sent request”) Database not updated

Users Module, UserAccount: UserAccount denyRequest(String friendName)

Scenario	**Input**	**Expected Result**
Successfully deny a request from provided name	“Joe Shmoe”	Return Updated UserAccount (Joe Shmoe removed from account.incomingRequests) Database relationship deleted
Account with provided name does not exist	[assumes: No account has name Doe John] “Doe John”	Return: Error(“Account does not exist”) Database not updated
Provided name undefined	undefined	Return: Error(“Account does not exist”) Database not updated
No request from provided name	[assumes: No request from Johnny Doe] “Johnny Doe”	Return: Error(“No request from this user”) Database not updated
Denying a request from self	[assumes: This account is named John Doe] “John Doe”	Return: Error(“Cannot deny a request from self”) Database not updated
Already friends with the provided name	[assumes: This account is already friends with Anne Smith] “Anne Smith”	Return: Error(“Already friends with this user”) Database not updated
Account already sent a	[assumes: This account	Return: Error(“Cannot deny

request to provided name	has sent a request to Joe Smith] “Joe Smith”	a sent request”) Database not updated

Users Module, UserAccount: UserAccount setDisplayName(String displayName)

Scenario	**Input**	**Expected Result**
Successfully change name	“unique name”	Return: Updated UserAccount (account.displayName = “unique name”) Database sets account name to “unique name”
Provided name is the same as old name	[assumes: John Doe was the old account name] “John Doe”	Return: Unchanged account Database not updated
Provided name is already taken	[assumes: There is an account with name Joe Shmoe] “Joe Shmoe”	Return: Error(“Name taken”) Database not updated
Provided name undefined	undefined	Return: Error(“No name provided”) Database not updated
Provided name is too short	“A”	Return: Error (“Name is not between 3 and 45 characters”) Database not updated
Provided name is too long	“ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST” (46 characters)	Return: Error (“Name is not between 3 and 45 characters”) Database not updated

Users Module, UserAccount: UserAccount unlockLocation(LocationInfo location)

(location.location_name is the only part used in this interface)

Scenario	**Input**	**Expected Result**
Successfully unlock a location	location{location_name: “Super secret spot”}	Return: Updated UserAccount (account.unlockedLocations contains “Super secret spot”)

		Database creates connection between account and “Super secret spot”
Location has no name	location{}	Return: Error(“Location has no name”) Database not updated
Already unlocked location	[assumes: Already unlocked location called “Secret spot”] location{location_name: “Secret spot”}	Return: Unchanged UserAccount Database not updated
Unlock location that does not exist	[assumes: Location called “Fake Spot” does not exist] location{location_name: “Fake Spot”}	Return: Error(“Location does not exist”) Database not updated
Location is undefined	undefined	Return: Error(“No location provided”) Database not updated

Users Module, UserAccount: UserAccount unlockItem(ARModel item)

(item.id is the only part used in this interface)

Scenario	**Input**	**Expected Result**
Successfully unlock an item	item{id:1}	Return: Updated UserAccount (account.collection.items contains item with id 1) Database creates connection between account and item with id 1
Item has no id	item{}	Return: Error(“Item has no id”) Database not updated
Already unlocked item	[assumes: Already unlocked location with id 2] Item{id: 2}	Return: Unchanged UserAccount Database not updated
Unlock item that does not exist	[assumes: Item with id 3” does not exist]	Return: Error(“Item does not exist”)

	litem{id:3}	Database not updated
Item is undefined	undefined	Return: Error(“No item provided”) Database not updated

Users Module, UserAccount: UserAccount updateAchievements(AchievementInfo achievement)

Scenario	**Input**	**Expected Result**
Successfully collect a new achievement	achievement{id:1, type: “collection”, points:1, image: “image”}	Return: Updated UserAccount (account.collection.achievements contains achievement, account.score is incremented by achievement.points) Database creates a new connection between account and achievement and updates account score
Successfully update an achievement	[assumes: Already collect achievement with id 1] achievement{id:1, type: “collection”, points:1, image: “image”}	Return: Updated UserAccount (account.collection.achievements contains the updated achievement and account.score is incremented) Database updates the related achievement by incrementing its points and updates account score
Achievement does not exist	[assumes: given achievement is not valid] achievement{id:1230, type: “free pts”, points:1000, image: “image”}	Return: Error(“Achievement does not exist”) Database not updated
Achievement is undefined	undefined	Return: Error(“No achievement provided”) Database not updated
Achievement is missing	achievement{}	Return:

fields		Error(“Achievement is missing fields”) Database not updated

Users Module, UserAccount: List getFriendsLeaderboard()

Scenario	**Input**	**Expected Result**
Account has no friends		Return: []
Account has friends but none are participating in leaderboards		Return: []
Account has friends that are participating in leaderboards	[assume: Friend1, Friend2, and Friend3 are all participating in leaderboards, on the friends list and have scores 6,5,7]	Return: [{displayName: “Friend3”, score: 7}, {displayName: “Friend1”, score 6},{displayName: “Friend2”, score 5}]

Locations Module, LocationStore: LocationInfo addLocation(string location_name,double coordinate_latitude,double coordinate_longitude, string fun_facts,string related_links,string about,string image_url,string access_permission): Unless noted these are the default values

```
{
   "location_name": "Engineering Cairn",
   "coordinate_latitude": 49.262209793949296,
   "coordinate_longitude": -123.25068964753649,
   "fun_facts": "Did you know that Cairn is painted all the time?",
   "related_links": "https://www.instagram.com/theubce/?hl=en",
   "about": "Historical landmark in University Endowment Lands, British Columbia",
   "image_url": "http://maps.ubc.ca/PROD/images/photos/N044_a.jpg",
   "access_permission": "PUBLIC"
}
```

Scenario	Input	Expected Result
location_name is duplicate	“UBC”	Error(“Duplicate Name”)
Location_name	"Engineering Cairn"	LocationInfo of Engineering Cairn
Coordinate_latitude is invalid	Coordinate_latitude = 91	Error("Coordinate latitude is not between -90 and 90"
Coordinate_longitude is invalid	Coordinate_longitude = -182	Error("Coordinate longitude is not between -180 and 180"
Location_name is greater than 255 chars	Location_name = “Lorem ipsum dolor sit amet, nonummy ligula volutpat hac integer nonummy. Suspendisse ultricies, congue etiam tellus, erat libero, nulla eleifend, mauris pellentesque. Suspendisse integer praesent vel, integer gravida mauris, fringilla vehicula lacinia nona”	Error("Location name is not between 1 and 255 characters")
Access Permission is invalid	Access_permission = “NONE”	Error("Access permission is not either \"PUBLIC\" or \"PRIVATE\"");

Locations Module, LocationStore: LocationInfo updateLocation(string location_name,double coordinate_latitude,double coordinate_longitude, string fun_facts,string related_links,string about,string image_url,string access_permission): Unless noted these are the default values

```
{
    "location_name": "Engineering Cairn",
    "coordinate_latitude": 49.262209793949296,
    "coordinate_longitude": -123.25068964753649,
    "fun_facts": "Did you know that Cairn is painted all the time?",
    "related_links": "https://www.instagram.com/theubce/?hl=en",
    "about": "Historical landmark in University Endowment Lands, British Columbia",
```

"image_url": "http://maps.ubc.ca/PROD/images/photos/N044_a.jpg",

"access_permission": "PUBLIC"

}`

Scenario	**Input**	**Expected Result**
Location_name exists	"Engineering Cairn"	Location Info of Engineering Cairn
Location_name doesn't exist	"Walter"	[]
Coordinate_latitude is invalid	Coordinate_latitude = 91	Error("Coordinate latitude is not between -90 and 90"
Coordinate_longitude is invalid	Coordinate_longitude = -182	Error("Coordinate longitude is not between -180 and 180"
Location_name is greater than 255 chars	Location_name = "Lorem ipsum dolor sit amet, nonummy ligula volutpat hac integer nonummy. Suspendisse ultricies, congue etiam tellus, erat libero, nulla eleifend, mauris pellentesque. Suspendisse integer praesent vel, integer gravida mauris, fringilla vehicula lacinia nona"	Error("Location name is not between 1 and 255 characters")
Access Permission is invalid	Access_permission = "NONE"	Error("Access permission is not either \"PUBLIC\" or \"PRIVATE\"");

Locations Module, LocationStore: deleteLocation(String location_name)

Scenario	**Input**	**Expected Result**
Location_name exists	"Engineering Cairn"	"Location Deleted"
Location_name doesn't exist	"Walter"	"Location Deleted"

Locations Module, LocationStore: deleteAllLocations()

Scenario	Input	Expected Result
Success		"All locations deleted"

Locations Module, LocationStore:List LocationInfo [] getLocationsByParameters(string location_name,double coordinate_latitude,double coordinate_longitude,double radius, string fun_facts,string related_links,string about,string image_url,string access_permission):

Scenario	Input	Expected Result
Location_name exists	"Engineering Cairn"	Location Info of Engineering Cairn
Location_name doesn't exist	"Walter"	[]
No inputs		Returns all locations in the database

Locations Module, Location_management: List LocationInfo [] getLocationsByUserAccountId(string user_account_id)

Scenario	Input	Expected Result
No account with given id	“46578123”	"No account with given id"
Account Exists and there are unlocked locations	"123456"	[LocationInfo, LocationInfo]
Account Exists and there are no unlocked locations	"1234561"	[]

World Module, Message Store: Message addMessage(double coordinate_latitude,double coordinate_longitude, string message_text, int user_account_id)

Unless noted these are the default values

```
{
   "coordinate_latitude": 49.262209793949296,
   "coordinate_longitude": -123.25068964753649,
```

"message_text": "Treasure here",

“User_account_id”: 2

}`

Scenario	**Input**	**Expected Result**
Coordinate_latitude and coordinate_longitude exists	"coordinate_latitude": 50, "coordinate_longitude": -60,	Error(“Message cannot be placed there”)
Coordinate_latitude and coordinate_longitude doesn’t exist	default	`{ “id”:2 "coordinate_latitude": 49.262209793949296, "coordinate_longitude": -123.25068964753649, "message_text": "Treasure here", “User_account_id”: 2 }`
Coordinate_latitude is invalid	Coordinate_latitude = 91	Error("Coordinate latitude is not between -90 and 90"
Coordinate_longitude is invalid	Coordinate_longitude = -182	Error("Coordinate longitude is not between -180 and 180"
message _text is greater than 255 chars	Messsage_text = “Lorem ipsum dolor sit amet, nonummy ligula volutpat hac integer nonummy. Suspendisse ultricies, congue etiam tellus, erat libero, nulla eleifend, mauris pellentesque. Suspendisse integer	Error("Message Text name is not between 1 and 255 characters")

	praesent vel, integer gravida mauris, fringilla vehicula lacinia nona”	
User_account doesn’t exist	User_Account_id:0	Error(“This user does not exist”)

World Module, Message Store: Message updateMessage(int id, double coordinate_latitude,double coordinate_longitude, string message_text, int user_account_id)

Unless noted these are the default values

`{
“id”:2
"coordinate_latitude": 49.262209793949296,
"coordinate_longitude": -123.25068964753649,
"message_text": "Treasure here",
“User_account_id”: 2
}`

Scenario	**Input**	**Expected Result**
Coordinate_latitude and coordinate_longitude exists	default	Return the updated Message
Coordinate_latitude and coordinate_longitude doesn’t exist	"coordinate_latitude": 50, "coordinate_longitude": -60,	Error(“Message does not exist”)
Id exists	default	Return the updated Message
Id does not exist	Id: 0	Error(“Message does not exist”)
message _text is greater than 255 chars	Messsage_text = “Lorem ipsum dolor sit amet,	Error("Message Text name is not between 1 and 255

	nonummy ligula volutpat hac integer nonummy. Suspendisse ultricies, congue etiam tellus, erat libero, nulla eleifend, mauris pellentesque. Suspendisse integer praesent vel, integer gravida mauris, fringilla vehicula lacinia nona”	characters")
User_account doesn’t exist	User_Account_id:0	Error(“This user does not exist”)

World Module, MessageStore: deleteMessage(int id)

Scenario	**Input**	**Expected Result**
Id exists	"1"	“Message Deleted”
Id doesn’t exist	"0"	“Message Deleted”

World Module, MessageStore: deleteAllMessages()

Scenario	**Input**	**Expected Result**
Success		"All messages deleted"

World Module, MessageStore: List Message [] getMessagesByParameters(int id,double coordinate_latitude,double coordinate_longitude,double radius, string message_text, string user_account_id):

Scenario	**Input**	**Expected Result**
Id exists	“1”	Message of id
Id doesn’t exist	“0”	[]
X parameter exists		Message with X parameter
X parameter does not exist		[]
No inputs		Returns all messages in

		the database

Interfaces:

Manage Locations

`{

"location_name": ””,,

"coordinate_latitude":,

"coordinate_longitude":,

"fun_facts": "",

"related_links": "",

"about": "",

"image_url": "",

"access_permission": "PUBLIC"

}`

Scenario	**Action**	**Expected Behavior**
Add Location	Add location non-existing location_name	“Location Added” Status code: 201
	Add location with existing location_name	“Duplicate Location” Status code: 400
	Add location with missing parameter	“Missing x” Status code:400
	Add location with invalid coordinate_latitude	“Cooridinate_latitude is not between -90 and 90” Status code:400
	Add location with invalid coordinate_longitude	“Coordinate Longitude is not between -180 and 180” Status code:400
	Add location with invalid image url	“Image URL is invalid” Status code:400
Get Location	Get location with blank	Returns all the locations

	location name	inside the database Status code:200
	Get location with existing location name	Returns the location with the associated location name Status code:200
	Get location with non-existing location name	Returns an empty array Status code:404
	Get location with access_permission = “PUBLIC”	Returns all locations with “PUBLIC” access_permissions Status code:200
Delete Location	Delete with Location name is blank	Cannot delete Status code:404
	Delete with Location name exists	“Location deleted” Status code:200
	Delete with Location name that does not exist	“Location deleted” Status code:200
Update Location	Update with Location name is blank	Cannot update Status code:404
	Update with Location name exists	“Location updated” Status code:200
	Update with Location name that does not exist	Cannot update Status code:404

Manage Messages

```
{
   “id”:
   "coordinate_latitude":,
   "coordinate_longitude":,
   "message_text": ""
}
```

Scenario	**Action**	**Expected Behavior**

Add Message	Add message on already existing coordinates	“Cannot add a message there” Status code: 400
	Add message on new coordinates	“Message added” Status code: 200
	Add message with missing parameters	“Missing x” Status code:400
	Add message with invalid coordinate_latitude	“Cooridinate_latitude is not between -90 and 90” Status code:400
	Add message with invalid coordinate_longitude	“Coordinate Longitude is not between -180 and 180” Status code:400
	Add message with text greater than 255	“Message text is not between 1 and 255” Status code:400
Get Message	Get message with blank id	Returns all the messages inside the database Status code:200
	Get message with existing id	Returns the message with the associated id Status code:200
	Get message with non-existing id	Returns an empty array Status code:404
	Get a message with coordinates = (x,y) and radius = 10.	Returns all messages within 10 degrees of (x,y) Status code:200
	Get messages with existing user_account_id	Returns all messages with user_account_id Status code:200
	Get messages with non-existing user_account_id	“This user does not exist” Status code:400
Delete Message	Delete with id blank	Cannot delete Status code:404

	Delete with id exists	“Message deleted” Status code:200
	Delete with id that does not exist	“Message deleted” Status code:200
Update Message	Update with id blank	Cannot update Status code:404
	Update with id that exists	“Message updated” Status code:200
	Update with id that does not exist	Cannot update Status code:404

Manage Profile

Scenario	**Action**	**Expected Behavior**
Changing name	Change name to an available name	Return updated account Update account in database with new name Status code: 200
	Change name to a taken name	Name taken error Status code: 400
	Change name to a name of improper length	Name length error Status code: 400
	Change name of non-existing user	No account with id error Status code: 404
	Change name with no name	No name given error Status code: 400
Sending requests	Send a request to existing user	Return updated account Status code: 200 Add relationship to database
	Send a request to a non-existing user	User not found error Status code: 404
	Send a request to a friend	Already friends error Status code: 400

	Send a request to a user twice	Already sent request to user error Status code: 400
	Send a request to a user that has sent you a request	Already sent request by user error Status code: 400
	Send a request to self	Self request error Status code: 400
	Send a request from a non-existing user	No account with id error Status code: 404
	Send a request with no name	No name given error Status code: 400
Removing friends	Remove an existing friend	Return updated account Delete relationship from database Status code: 200
	Remove a non-existing user	User not found error Status code: 404
	Remove a existing non-friend	Not friends error Status code: 400
	Remove self	Self request error Status code: 400
	Remove a friend from a non-existing user	No account with id error Status code: 404
Accepting requests	Accept an existing incoming request	Return updated account Update relationship in database Status code: 200
	Accept a request from a non-existing user	User not found error Status code: 404
	Accept a request that does not exist	No request error Status code: 404
	Accept a request from self	Self request error Status code: 400

	Accept a request from a friend	Already friends error Status code: 400
	Accept an outgoing request	Accepting outgoing request error Status code: 400
	Accept a request to a non-existing user	No account with id error Status code: 404
	Accept a request with no name	No name given error Status code: 400
Denying requests	Deny an existing incoming request	Return updated account Delete relationship from database Status code: 200
	Deny a request from a non-existing user	User not found error Status code: 404
	Deny a request that does not exist	No request error Status code: 404
	Deny a request from self	Self request error Status code: 400
	Deny a request from a friend	Already friends error Status code: 400
	Deny an outgoing request	Denying outgoing request error Status code: 400
	Deny a request to a non-existing user	No account with id error Status code: 404
Unlocking locations	Unlocking a location that is new	Return updated account Create account-location connection in database Status code: 200
	Unlocking a location that was previously unlocked	Return unchanged account Status code: 200
	Unlocking a location that does not exist	Location does not exist error Status code: 404

	Unlocking a location with no location	No location given error Status code: 400
Unlocking items	Unlocking an item that is new	Return updated account Create account-item connection in database Status code: 200
	Unlocking an item that was previously unlocked	Return unchanged account Status code: 200
	Unlocking an item that does not exist	Item does not exist error Status code: 404
	Unlocking an item with no item	No item given error Status code: 400
Updating achievements	Updating an achievement that is new	Return updated account with new achievement and score Create new account-achievement connection in database and update account score Status code: 200
	Updating an achievement that was previously collected	Return updated account with updated achievement and score Update account-achievement connection in database and update account score Status code: 200
	Updating an achievement that does not exist	Achievement does not exist error Status code: 404
	Updating an achievement with no achievement	No achievement given error Status code: 400

Change difficulty

Scenario	**Action**	**Expected Behavior**

Change difficulty	Change difficulty to a new difficulty	Return updated account Update account in database with new difficulty Status code: 200
	Change difficulty to invalid difficulty	Invalid difficulty error Status code: 400
	Change difficulty with no difficulty	No difficulty given error Status code: 400
	Change difficulty of non-existing user	No account with id error Status code: 404

Login

Scenario	**Action**	**Expected Behavior**
Login	Login with valid token account does not exist	Return newly created account Create new account in database Status code: 200
	Login with valid token account already exists	Return account Status code: 200
	Login with no token	No token error Status code: 500
	Login with invalid token	Invalid token error Status code: 500

Puzzles

Scenario	**Action**	**Expected Behavior**
Get AR items	Get AR items at location x (location exists)	Return AR model Status code: 200
	Get AR items at location x (location does not exist)	Invalid location error Status code: 404

Manage Leaderboards

Scenario	Action	Expected Behavior
View global leaderboard	View global leaderboard with participants	Return ranked list of usernames and scores Status code: 200
	View global leaderboard with no participants	Return empty list Status code: 200
View friend leaderboard	View friend leaderboard with valid user id with friends	Return ranked list of usernames (that are friends to the user) and scores Status code: 200
	View friend leaderboard with valid user id with empty friends list	Return empty list Status code: 200
	View friend leaderboard of non-existing user	No account with id error Status code: 404
Participate in leaderboard	Participate in leaderboard	Return user info Status code: 200

Front-end:

Manage profile

Action	Expected Behavior
The user clicks on the profile button on the main page.	If the user is not logged in yet, the login page appears prompting the user to login using their Google account. If the user is logged in, the profile page appears, displaying the user's display name, score, collection of creatures and any puzzle pieces they have obtained.
The user clicks on the “edit display name” button.	A text box is displayed and two buttons: “cancel” and “confirm”.
The user clicks “cancel”.	The text box and the two buttons disappear.
The user enters a display name that’s not	A toast message appears saying “Name

between 3-20 characters.	must be between 3 and 20 characters!”
The user enters a display name that’s within the required length and clicks “confirm”.	The app displays their profile page with the updated name. The textbox and the two buttons disappear.
The user clicks on the “friends” button.	The app displays the friends page with a list of the user’s friends and buttons for adding friends, removing friends, and checking outgoing and incoming friend requests.
The user clicks on the “+” button for adding friends.	The app displays a text box and two buttons: “send request” and “cancel”.
The user clicks on the “cancel” button.	The text box and two buttons disappear.
The user enters the friend’s name and clicks “send request”.	The app displays “Friend request sent!” and the text box and the two buttons disappear. If the send fails, the corresponding error from the backend is displayed.
The user clicks on the “x” button for removing friends.	The app displays a text box and two buttons: “remove” and “cancel”.
The user enters the friend’s name and clicks “remove”.	The app displays “removed successfully!” If the remove fails, the corresponding error from the backend is displayed.
The user clicks on the “...” button.	The app displays the friend requests page with two buttons “outgoing friend requests” and “incoming friend requests”.
The user clicks on the “outgoing requests” button.	The app displays a list of the user’s outgoing friend requests.
The user clicks on the “incoming requests” button.	The app displays a list of the user’s incoming friend requests.
The user clicks on a friend on the incoming friend requests list.	The app displays two buttons: “accept” and “decline”.
The user clicks on “accept”.	The app displays “Friend request accepted!” If it fails, the corresponding error from the backend is displayed.

The user clicks on “decline”.	The app displays “Friend request denied!” If it fails, the corresponding error from the backend is displayed.

Explore locations

Action	**Expected Behavior**
User selects the “location” tab on the main page.	The app opens the location list screen displaying a list of locations
User selects a location from the list	The app confirm if the user want to explore the location by displaying a text asking if the user want to go to the location
User presses “Yes” for the selected location	The app displays a map with the location marked and a route from user’s current location to the destination
User presses “No” for the selected location	The app displays the list of locations
User presses “Cancel” while the app displaying the map	The app returns to the main screen

Puzzles

Action	**Expected Behavior**
The user points their camera in a certain direction.	A set of AR interactable buttons is shown.
The user presses buttons in sequence.	If the sequence is correct, the app displays the achievement and scores they get, and the name of the secret location. If the sequence is wrong, the app displays a message: “wrong order, please try again”.
The user points their camera in a certain direction where a puzzle piece is.	A puzzle piece is displayed.
The user clicks on the puzzle piece.	A message is displayed: “You have collected the puzzle piece!”

Non-functional requirements:

- Performance: When the user reaches the location they choose to go to, the app should display relevant information about the location within 1 second. This requirement can be tested by writing automated tests to change the location of the virtual device and record the time it takes from the location change to when the relevant info successfully shows up to check if the time is within 1 second.

- Energy efficiency: After using the app for 5 minutes, the battery life should not drop by more than 1%. We plan to test this requirement by setting random automatic interactions with the app several times while recording the battery usage by using Battery Historian and AccuBattery.

Folder for tests:

(On develop branch)

- Frontend:
  frontend/app/src/androidTest/java/com/example/ubcexplore/EspressoTest.java
- Backend:
  backend/users/users_server_mock.test.js
  backend/world/test/messages.test.js
  backend/world/test/locations.test.js

Member Contributions:

- Akshat: Wrote mocks for location and world module. Written test cases for the aforementioned back-end modules and api.
- Jane: Wrote some of the interface and frontend test designs. Wrote a simple frontend test using Espresso.
- Dylan: Wrote mocks for users module. Wrote test design for users module and most of the user related interfaces.
- Mei: Wrote non-functional requirement test designs. Wrote frontend test designs for the explore location use case.