CPEN 321 Software Engineering

M4: Design

UBC Explore

Group members:

Jane Shi 37998283

Xiaoqin Mei 71315980

Dylan Pither 64661267

Akshat Hari 28768299

Description:

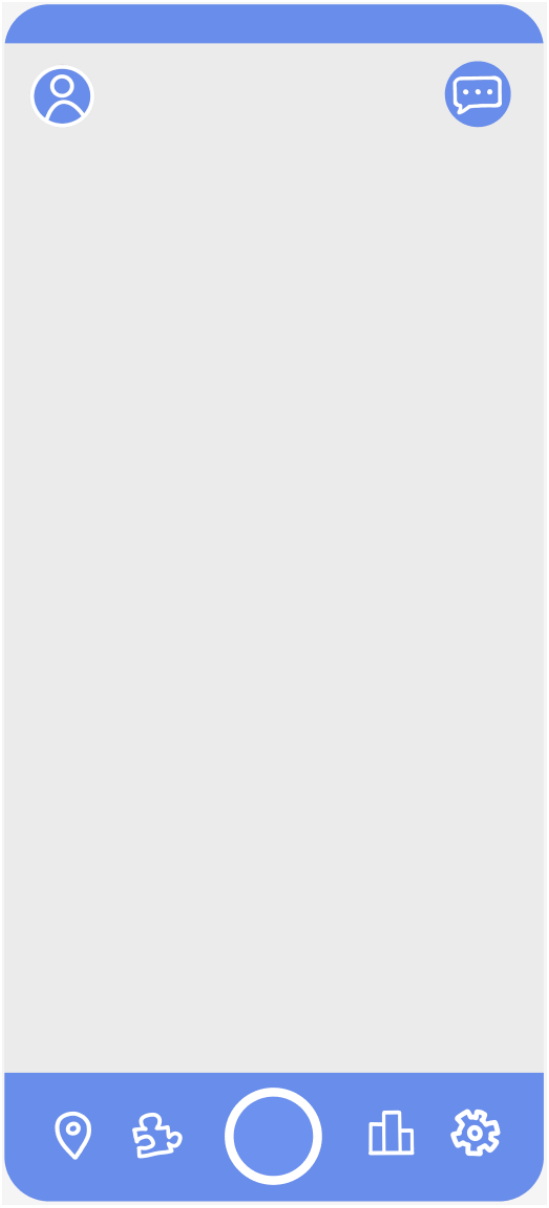
The UBC campus is large and it takes effort to explore and understand every part of it. Our app proposes to decrease the difficulty by showing the history and fun facts of UBC at the tip of your fingers. If a student or a tourist wants to know more about a part of the campus, they can select it as a 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.
- Messages will be displayed in real-time when a user arrives at a certain location.
- Leaderboards will also be updated in real-time.

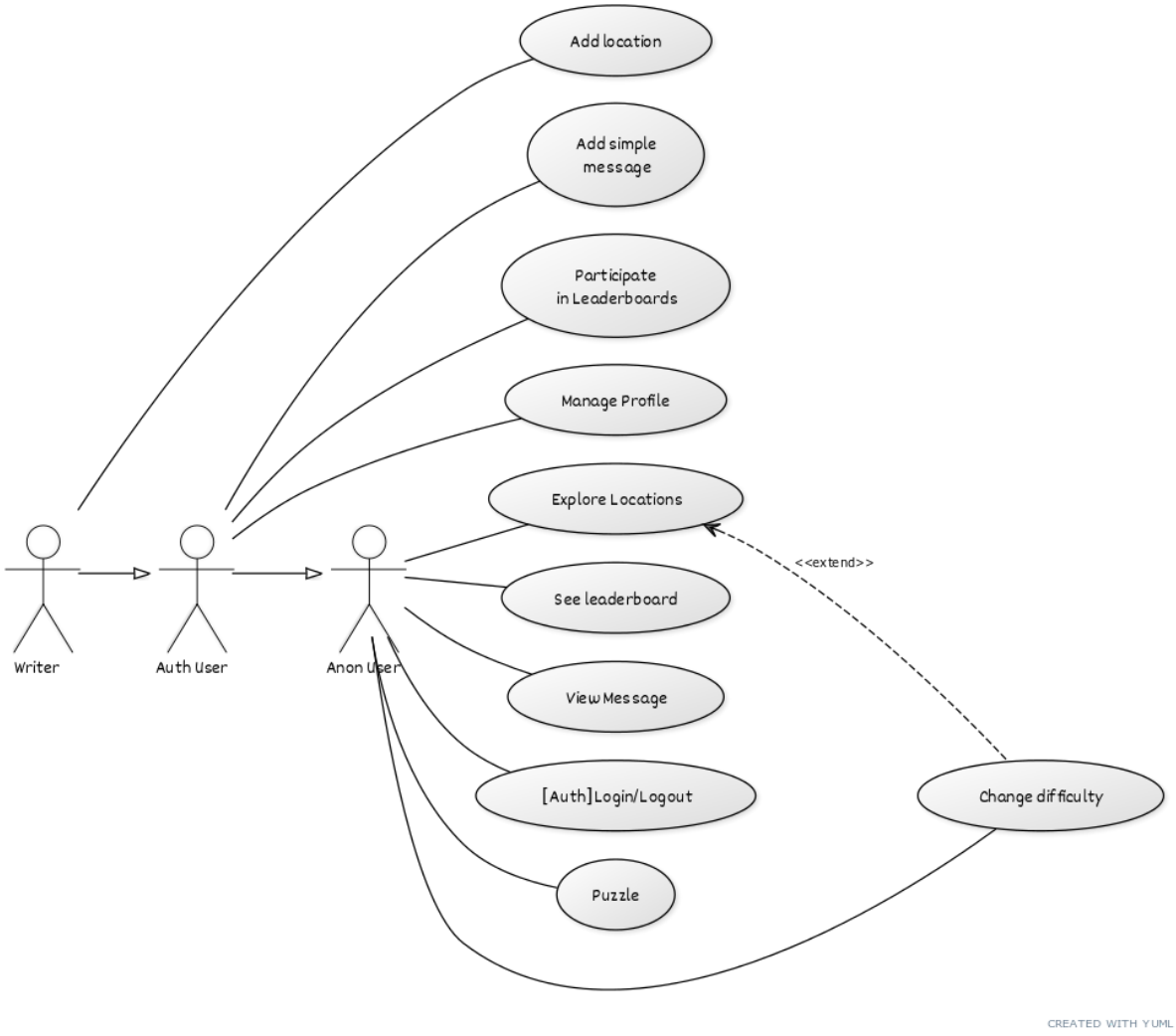
Interesting Features:

- 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 asks the user if they would like to explore the selected location.
5. The user clicks “Yes”.
6. The app checks for user difficulty.
7. Easy:
    7.1. The app displays directions to the selected location using Google Maps API.
    7.2. The user follows the directions to the location.
8. Medium:
    8.1. The app displays pictures to help guide users to the location.
    8.2. The app shows the general location.
    8.3. The user parses clues and arrives at the location.
9. When the user arrives at the location the app displays information about the location.

Extensions:

5a. User clicks “No”.

- 5a1. The app returns to the list of explorable locations.

7a. User does not follow the directions or cannot find the location.

- 7a1. The user clicks cancel on the directions and returns to the main screen.

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 app will notify the user if any messages are nearby.
2. The user clicks on the notification.
3. The app displays the message to the user.

Extensions:

2a. The user does not click anything.

- 2a1. The app does nothing, the notification will disappear after 5 seconds.

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.

4d. The user provides an image that is too large.

- 4d1. The system alerts the user that the provided image is too large and how large the image is allowed to be.

4e. The user provides a location that already exists.

- 4e1. The app alerts the user that the location already exists.

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 “Ok”.
    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 and each friend's collection score.
    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 “✓” or “X”.

4.5 The user clicks “X” on a friend in the list.

4.5.1 The app asks the user if they are sure they want to remove this friend.

4.5.2 The user clicks “Yes”.

4.5.3 The app removes the friend from the user's friends list.

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.

Project Description:

Please see the first page for our project description.

Changes In Project Scope:

- Puzzles were redesigned to have two types related to AR.
    - Reason for change: Better explanation of the idea of puzzles and how they work.
    - New artifact: Revised “Puzzle” use case.
- Updated main screen sketch
    - Reason for change: Improve the look of the UI.
    - New artifact: New main screen sketch.
- Added “Manage profile” use case.
    - Reason for change: Increase project scope
    - New artifact: Added to use case diagram, added a formal use case specification for this use case.
- Added “Friends” idea which is involved in “Manage profile” and “See leaderboard” use cases.
    - Reason for change: Increase project scope.
    - New artifact: Revised “See leaderboard” use case and friend management incorporated in “Manage profile” use case.

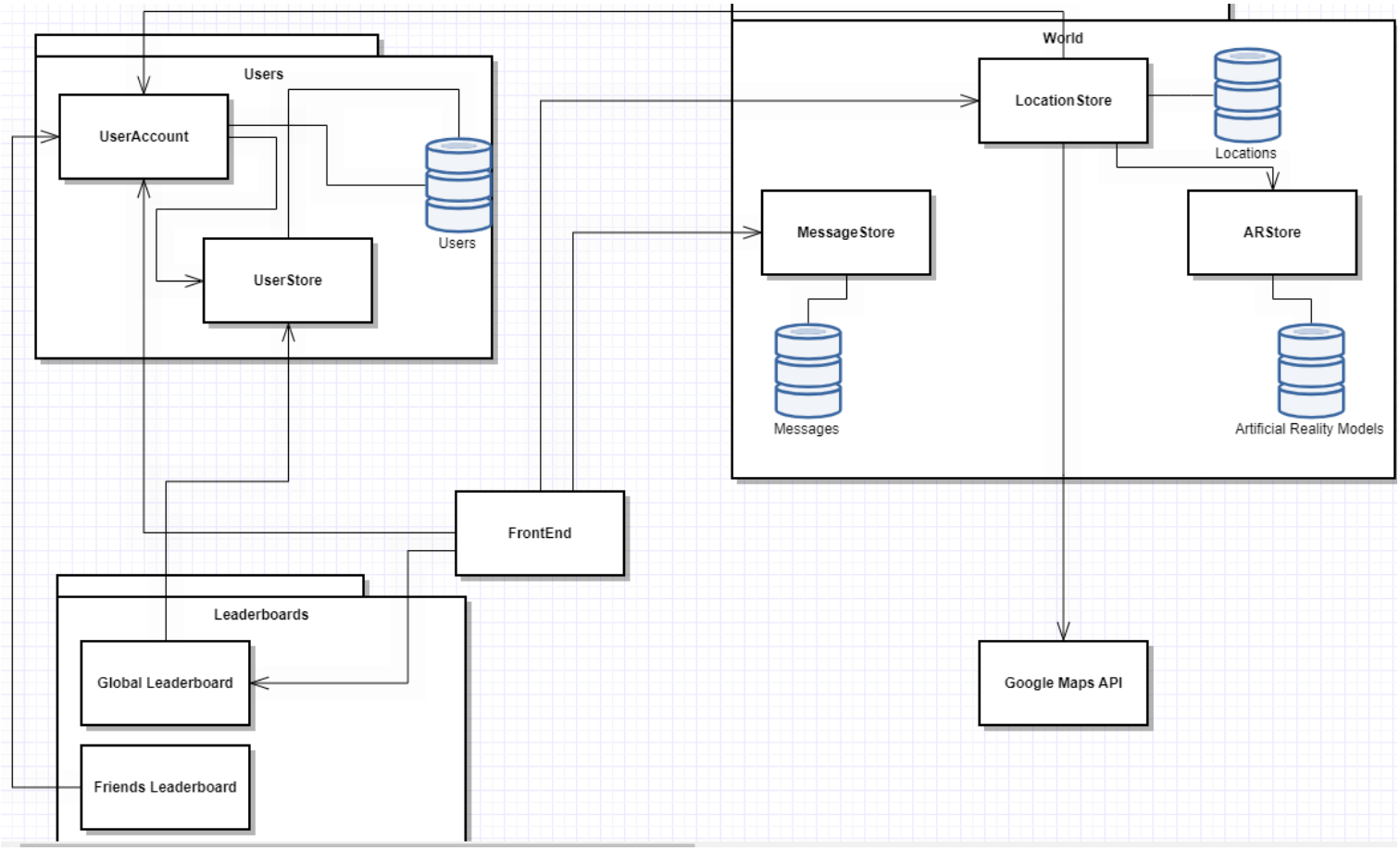
Main Modules and Sub-modules:

- Main modules:
    - Users
        - Purpose: Management of everything linked to user accounts including their collected creatures, puzzle pieces and friends. Also tracks user preferences such as difficulty and if they are a writer.
        - Sub-modules:
            - UserAccount
            - UserStore
    - Leaderboards
        - Purpose: Ranking of authenticated users' achievements and scores.
        - Sub-modules:
            - GlobalLeaderboard
            - FriendsLeaderboard
    - 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
        - This database will hold list of Authenticated users
        - It will also hold their collection of AR items and puzzles
        - It will also hold the list of friends of the Accounts
        - It will also hold the list of incoming and outgoing friends requests of the users
    - 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:

- UserAccount:
    - Int participateInLeaderboard()
        - Required when user requests to participate in leaderboards.
    - String getDifficulty()
        - Required when retrieving directions to a location because there are different types depending on difficulty.
    - Int changeDifficulty(String difficulty)
        - Required when the user wishes to change their difficulty level.
    - List getCollection()
        - Required when retrieving a user's collection so it can be viewed in their profile.

    - Int updateCollection(Collection)
        - Required when a user gets new AR collections to be updated in their profile.
    - Int setDisplayName(String name)
        - Required when the user wishes to change their display name.
    - List getFriends()
        - Required when retrieving a user's friends list so it can be viewed in their profile or for use in the leaderboard.
    - Int removeFriend(String friendName)
        - Required when deleting a user wishes to remove a friend from their friends list.
    - Int sendRequest(String friendName)
        - Required when a user wants to send a friend request to another user
    - Int addOutgoing(String friendName)
        - Required when a user sends a friend request to another user.
    - Int addIncoming(String friendName)
        - Required when a user receives a friend request from another user.
    - Int acceptRequest(String friendName)
        - Required when a user accepts a friend request from another user.
    - Int denyRequest(String friend Name)
        - Required when a user denies a friend request from another user.
    - AchievementInfo getAchievements(UserAccount)
        - Required when retrieving a user's achievements information.
    - Int updateAchievements(AchievementInfo)
        - Required when a user gets to a new location and their achievements need to be updated
- UserStore:
    - UserAccount find(String displayName)
        - Required when wanting to find another user to add as a friend or to check for a unique display name.

    - UserAccount login(credentials)
        - Required when client wants to login using google API
    - UserAccount createAccount(credentials)
        - Required if there is no existing user for credentials
    - UserAccount findAccount(credentials)
        - Find UserAccount from given credentials
- 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.

- Leaderboard:
    - List getFriendsLeaderboard(UserAccount)
        - Required when a user wishes to view the leaderboard of only their friends.
    - List getGlobalLeaderboard()
        - Required when a user wishes to view the global leaderboard.
- Users Database:
    - Int participateInLeaderboard()
        - Required when user requests to participate in leaderboards.
    - List getFriends()
        - Required when retrieving a user's friends list so it can be viewed in their profile or for use in the leaderboard.
    - AchievementInfo getAchievements(UserAccount)
        - Required when retrieving a user's achievements information.
    - 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 .Location(Location,UserAccount)
        - Required when the user unlocks a location from solving puzzles
    - int updateAccount(int id, UserAccount)
        - Updates a user account with the provided id to the contents of the provided UserAccount
    - UserAccount findUser(String displayName)
        - Required to find a user by display name in the database.
- 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:

Credentials{
String: id_token
}
Collection{
List[AchievementInfo.id],
List[ARModel.id],
}
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: Name,
String: Fun Facts,
String: Related Links,
String: About,
String: Image
}
UserAccount{
List[collection.id] collection,
List[UserAccounts.id]: friends,
List[LocationInfo.id],
String: difficulty,
Boolean: leaderboardParticipant,
String: displayName,
List[UserAccounts.displayName]: incomingRequests,
List[UserAccounts.displayName]: outgoingRequests,
int: score,
int: id
}

Sequence Diagrams:

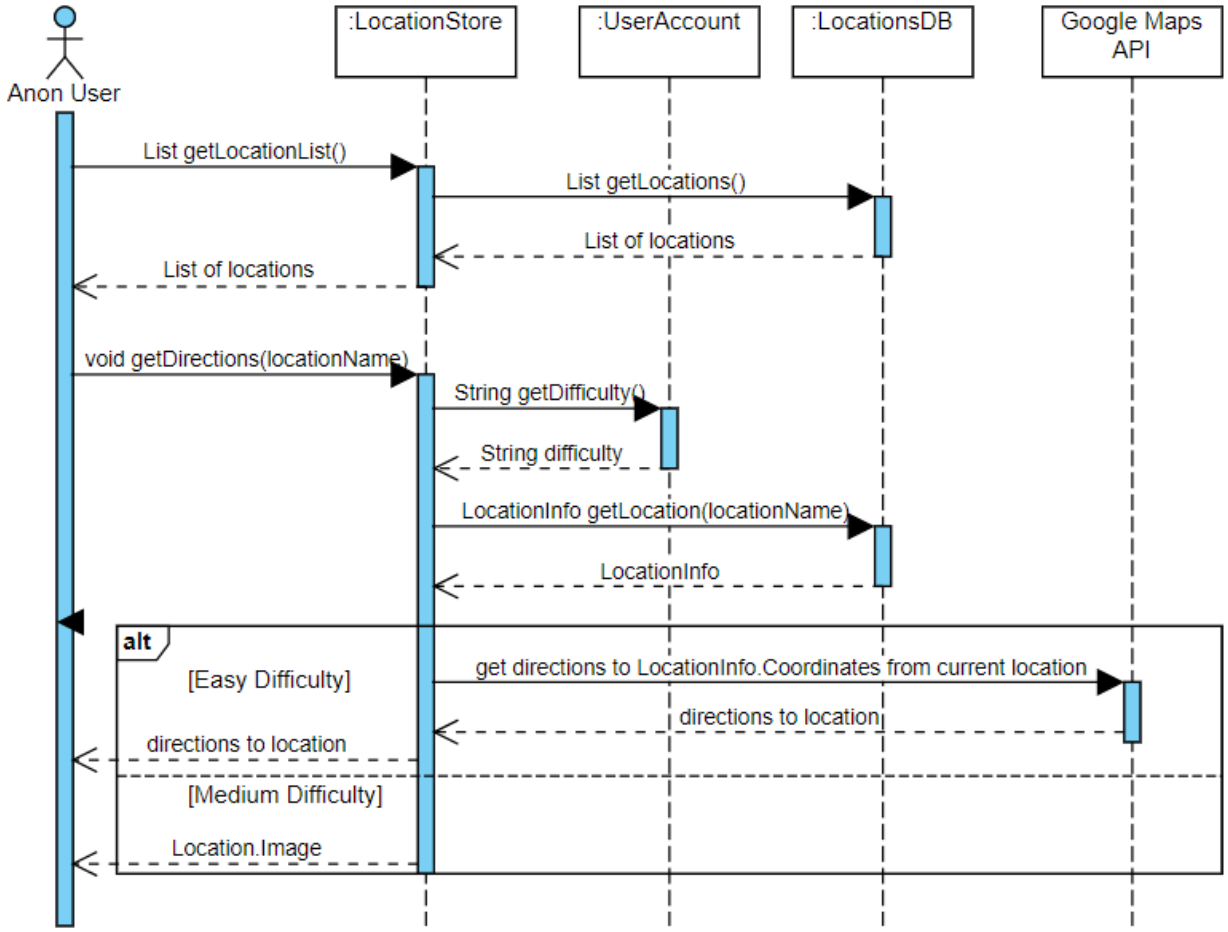
We assume the following meaning of return values in our sequence diagram:

1: Success

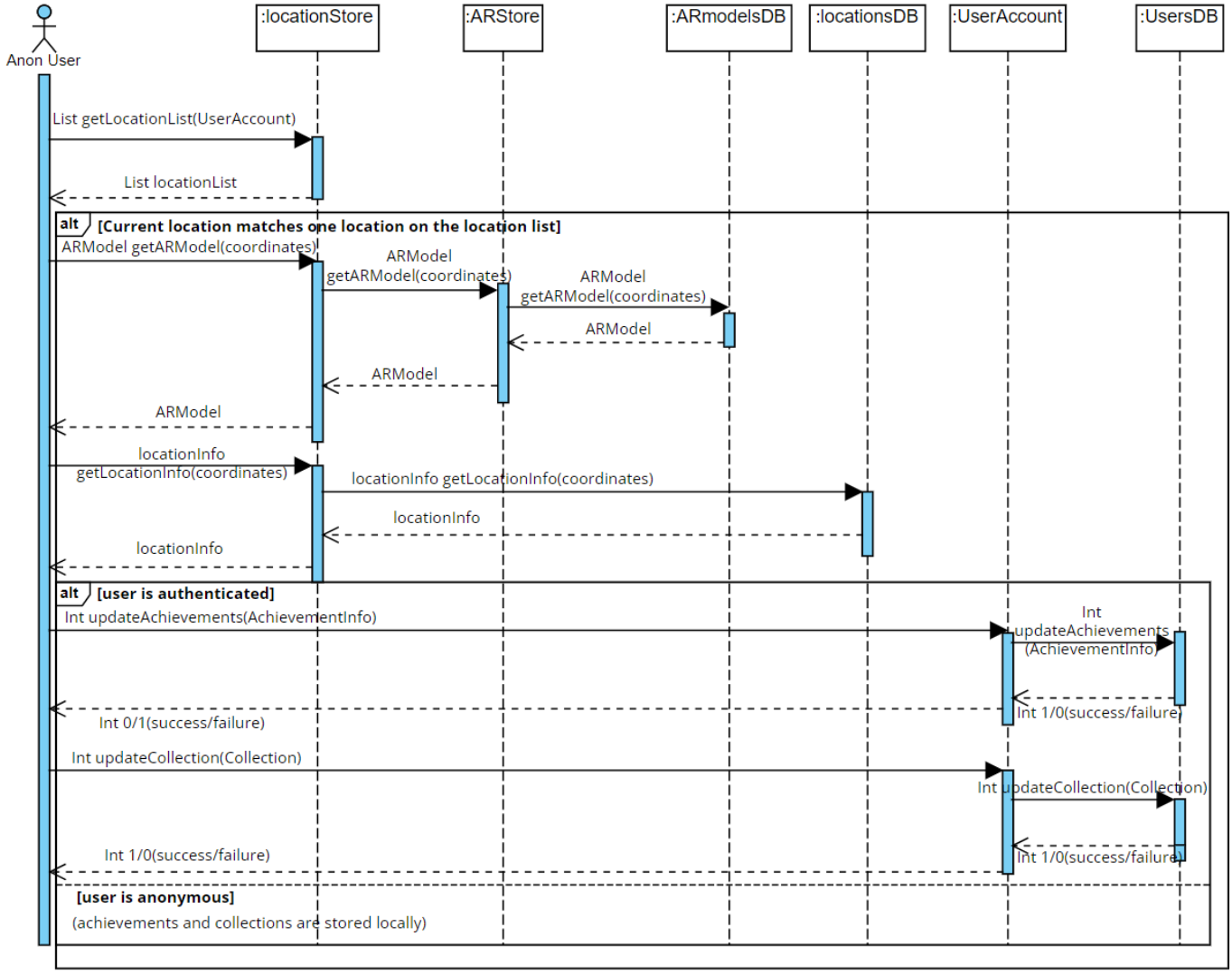
0: Failure

- **Explore locations**

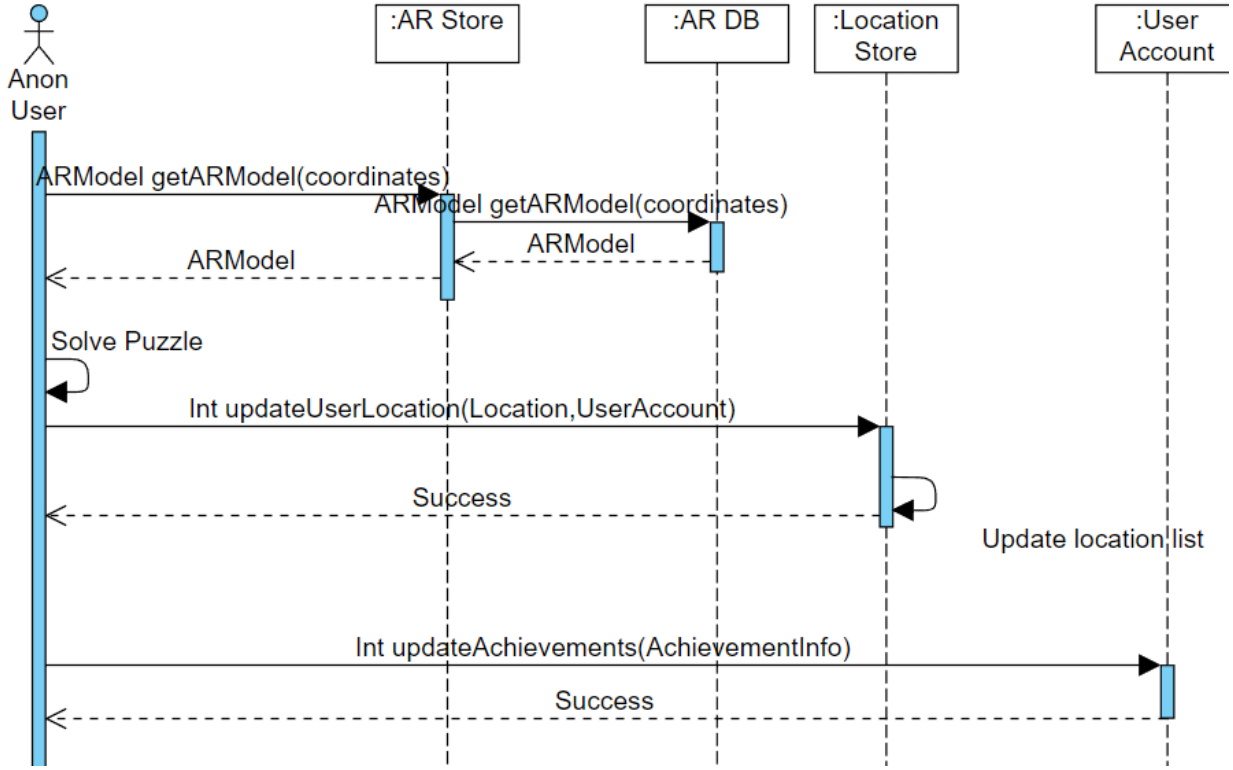
  Getting directions to the location:



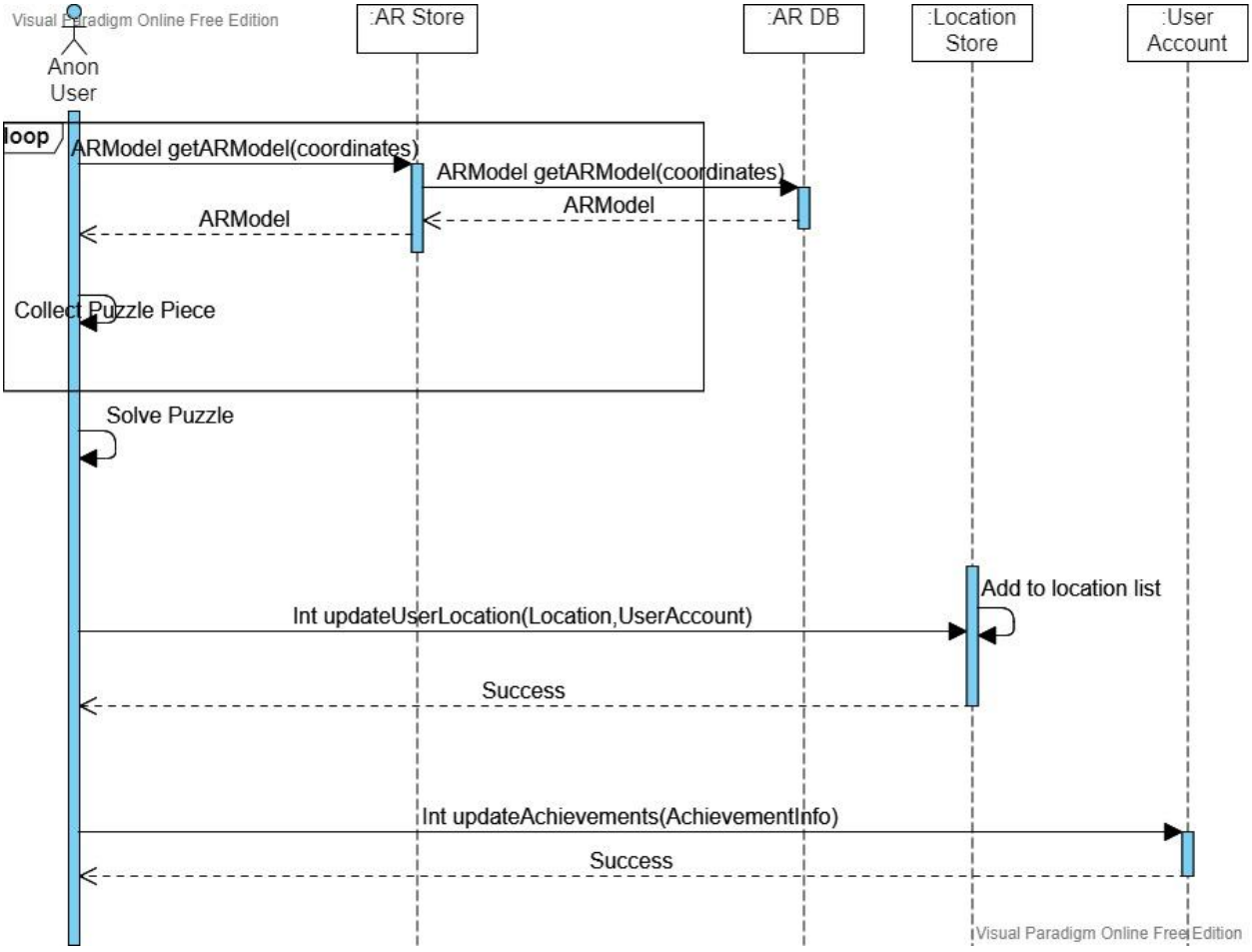
- After reaching the location:



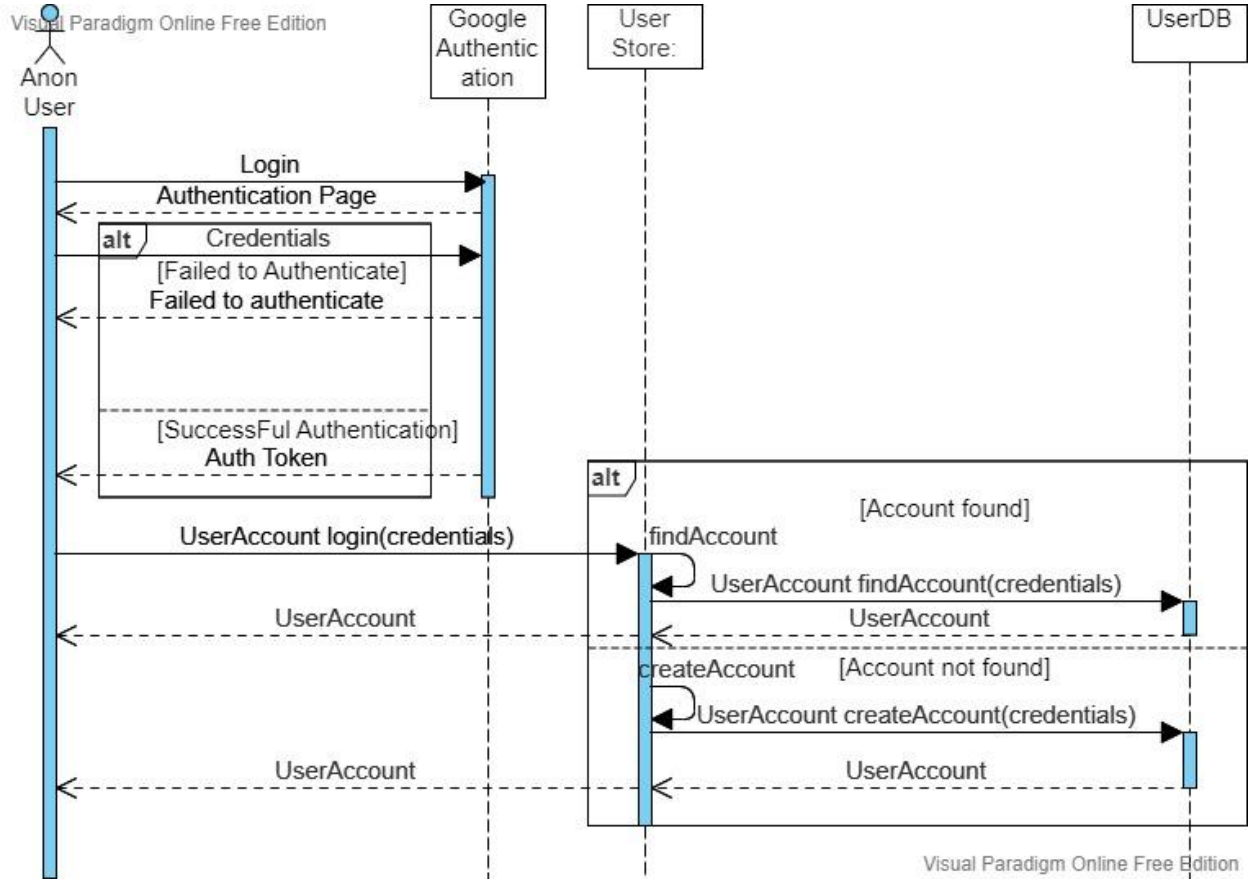
- AR buttons


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

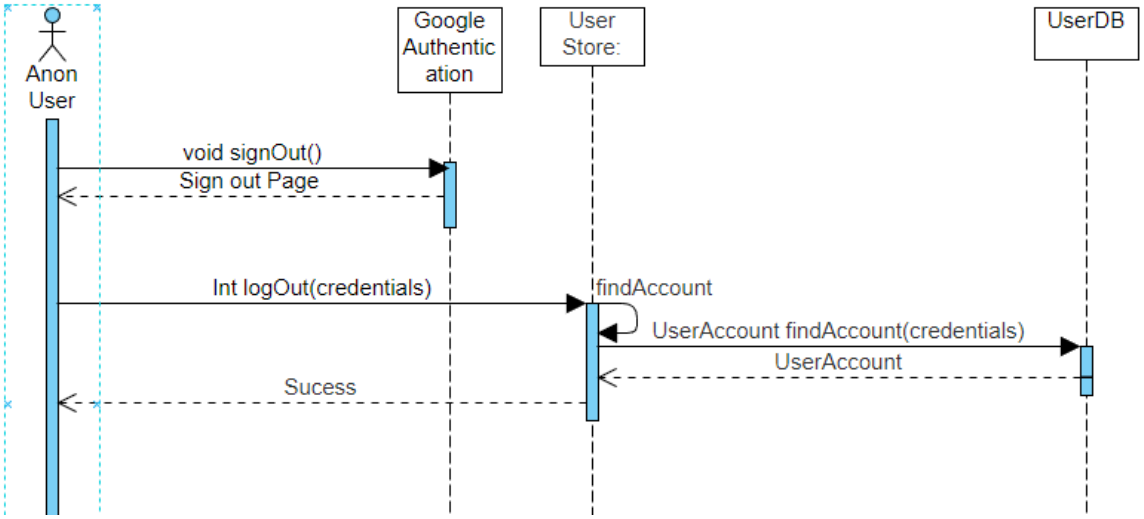
- AR puzzles


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**

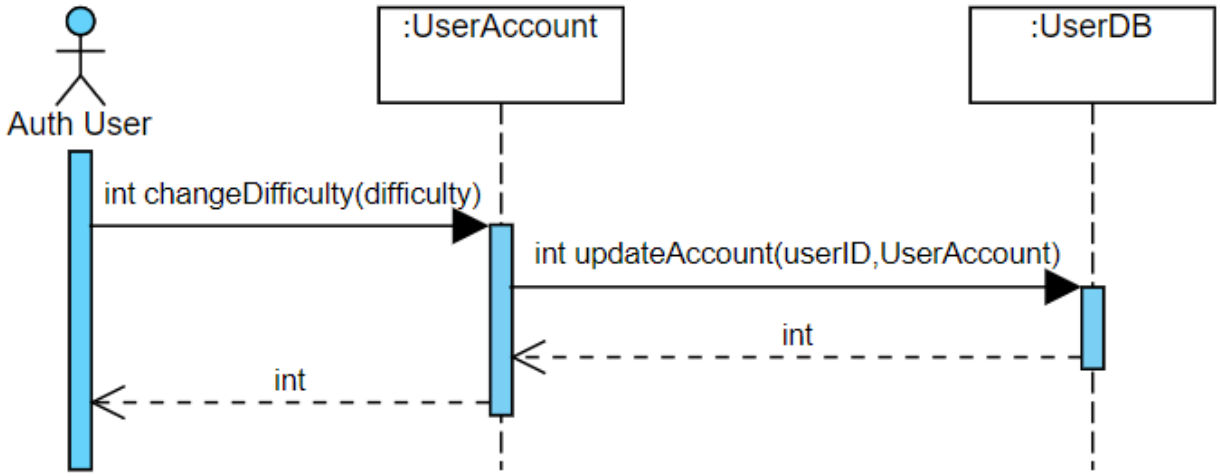


- **Logout**



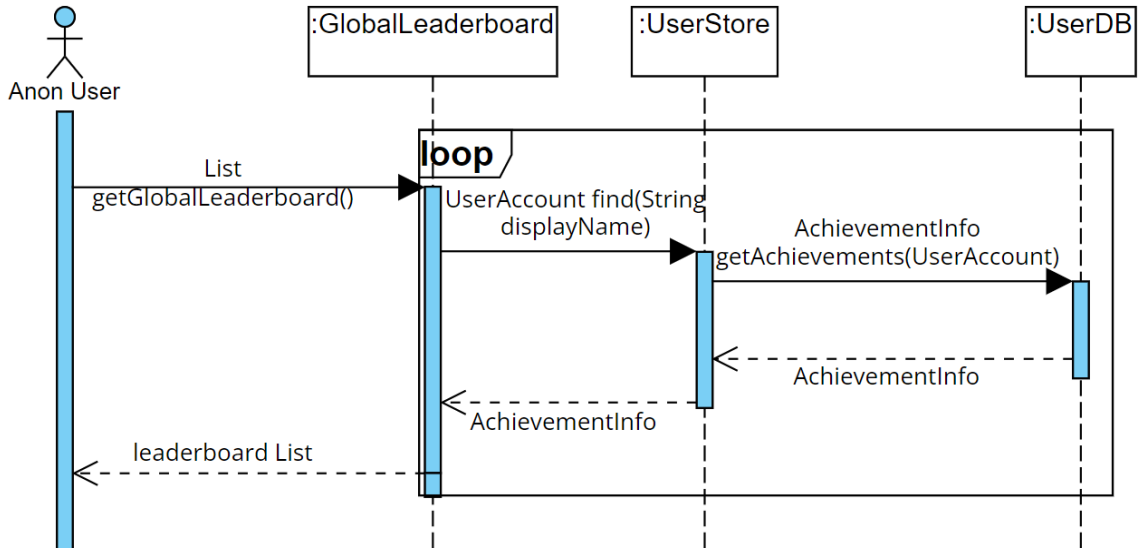
-

- **Change difficulty**

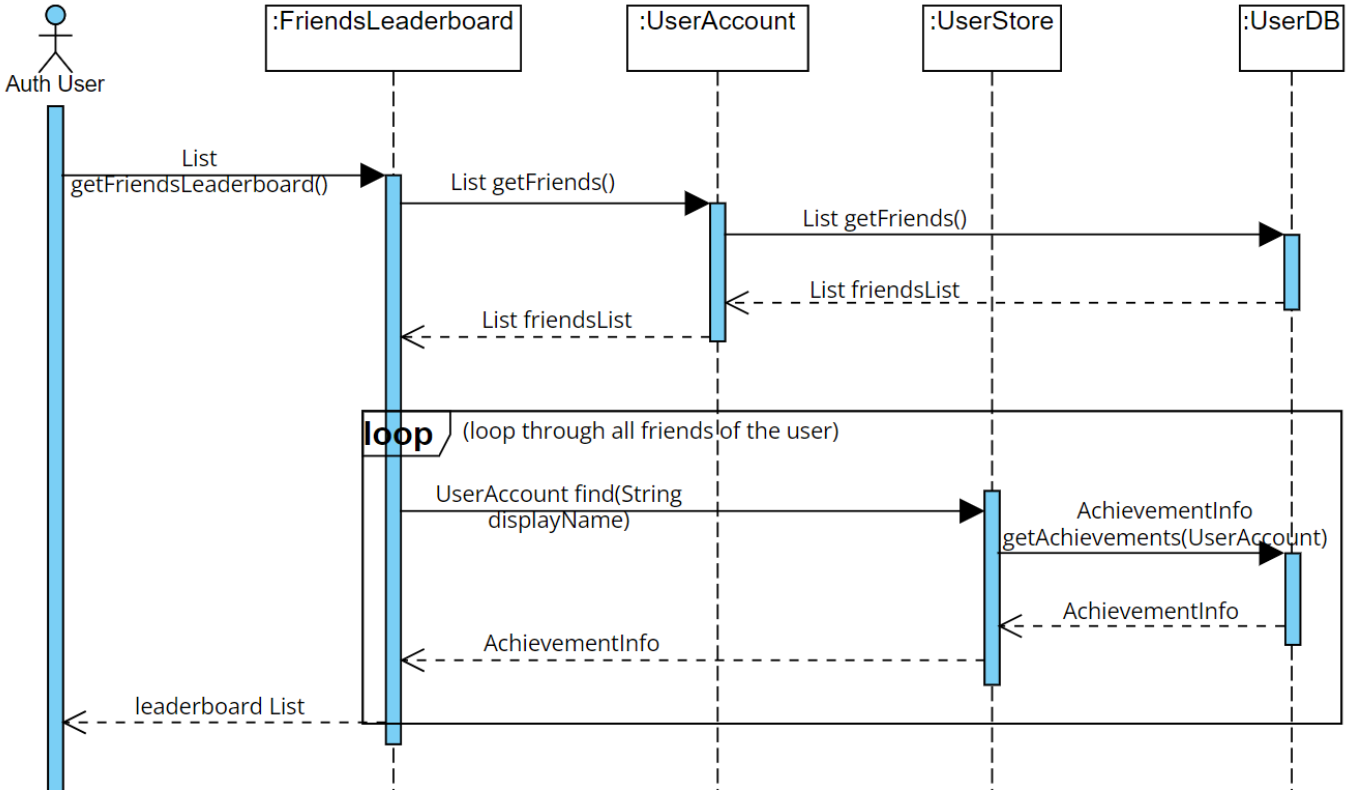


- **See leaderboard**

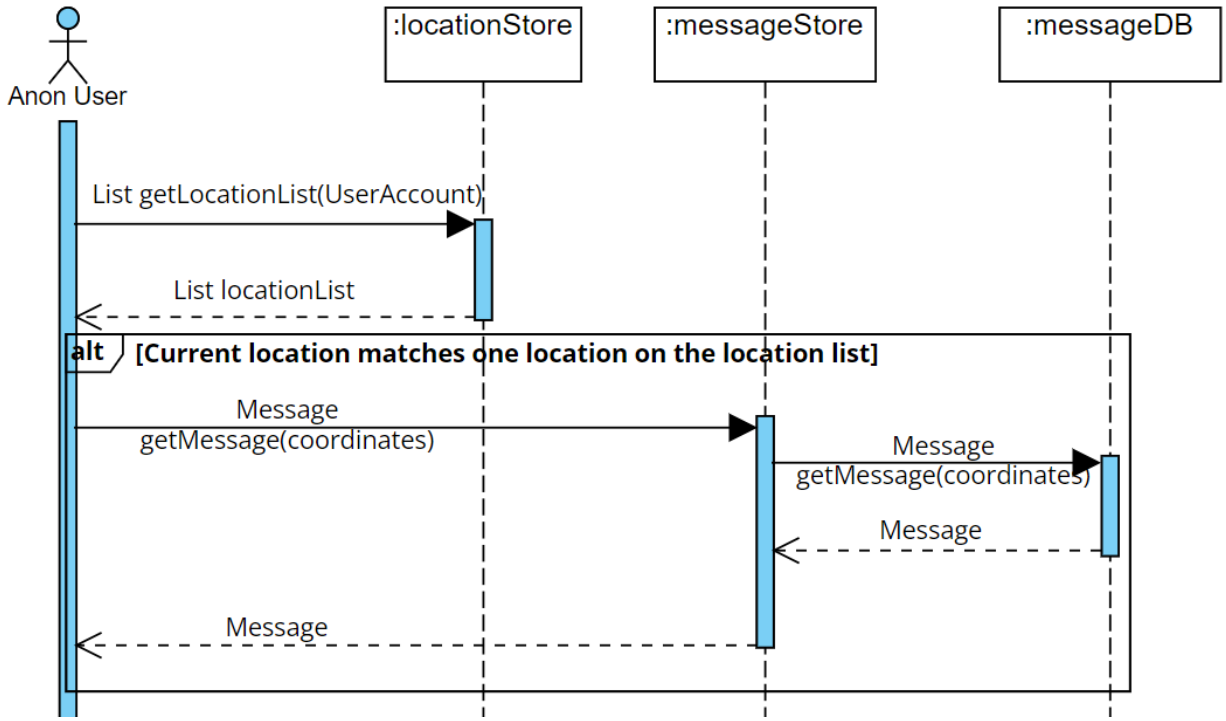
Viewing the global leaderboard as Anon user:



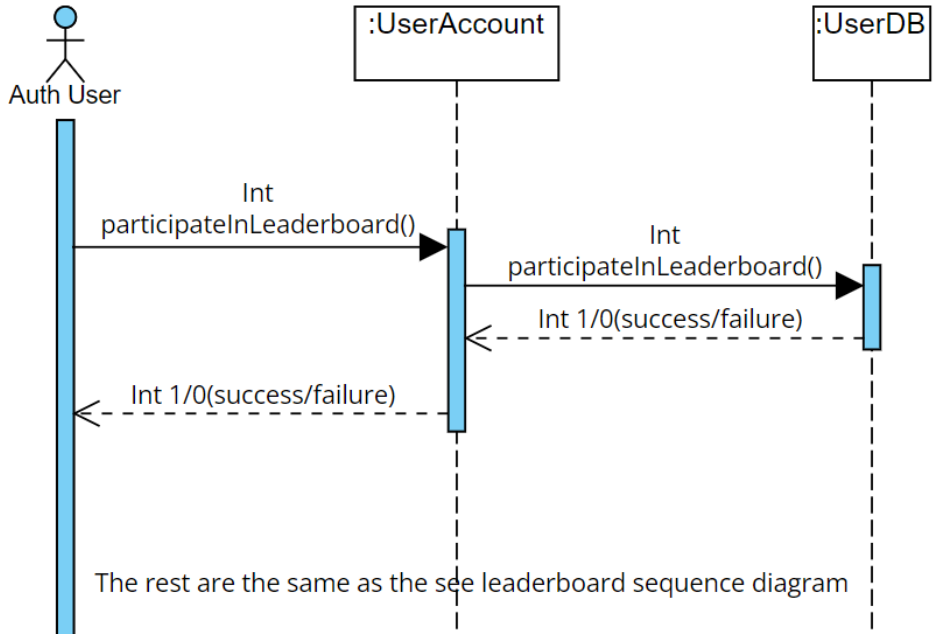
- **Viewing the friends leaderboard as Auth user:**



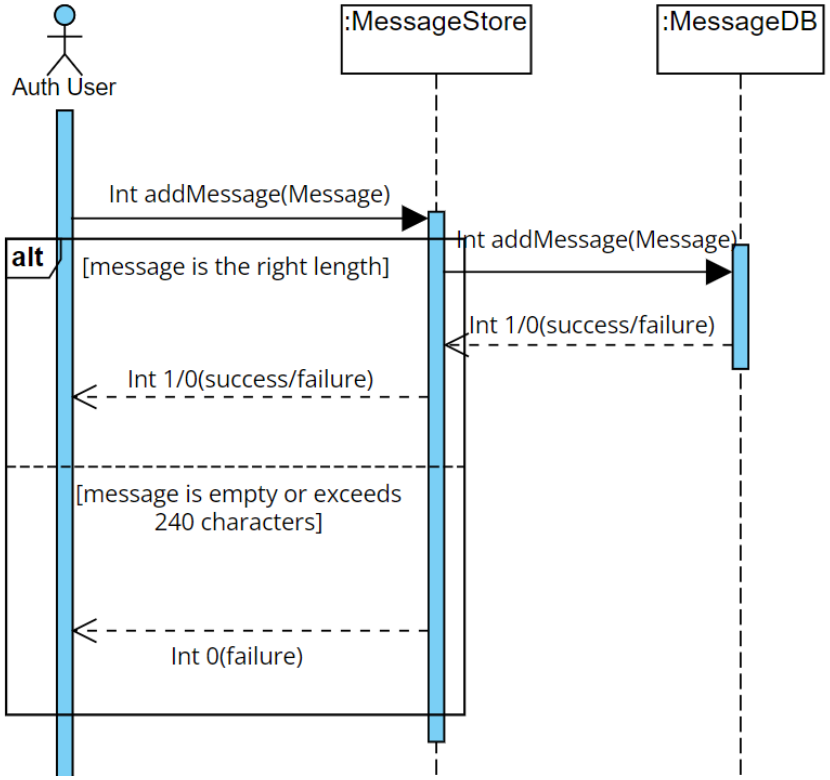
- **View message**



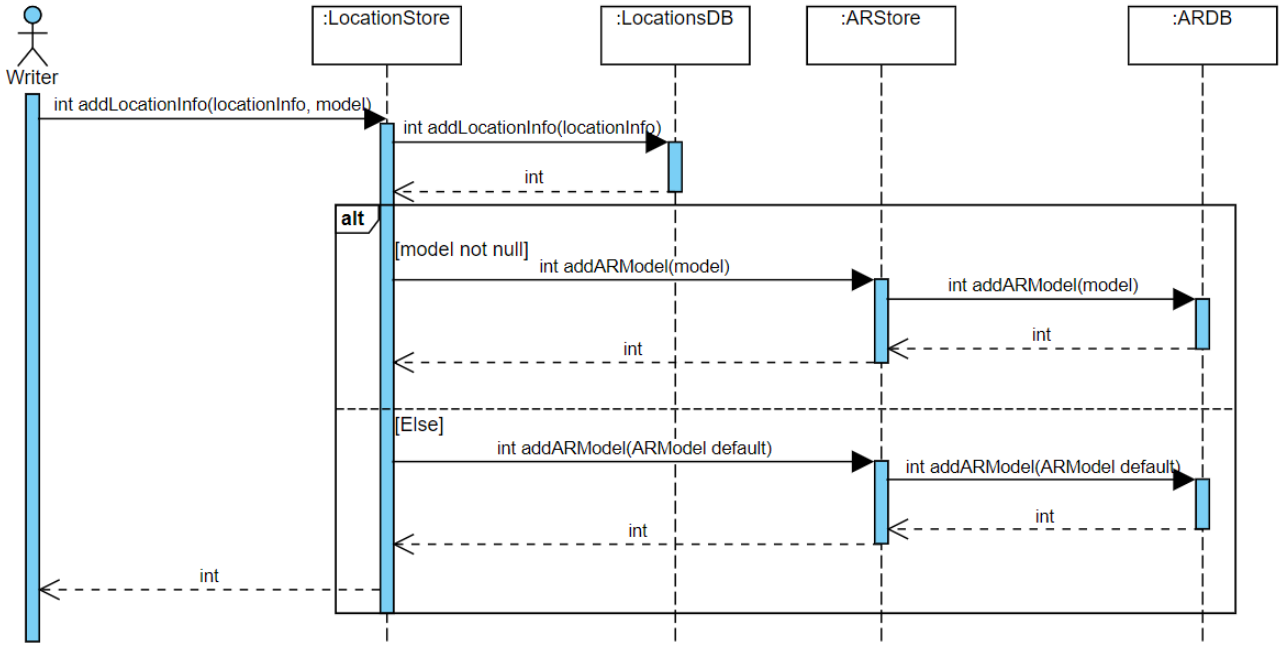
- **Participate in leaderboards**



- **Add a simple message**

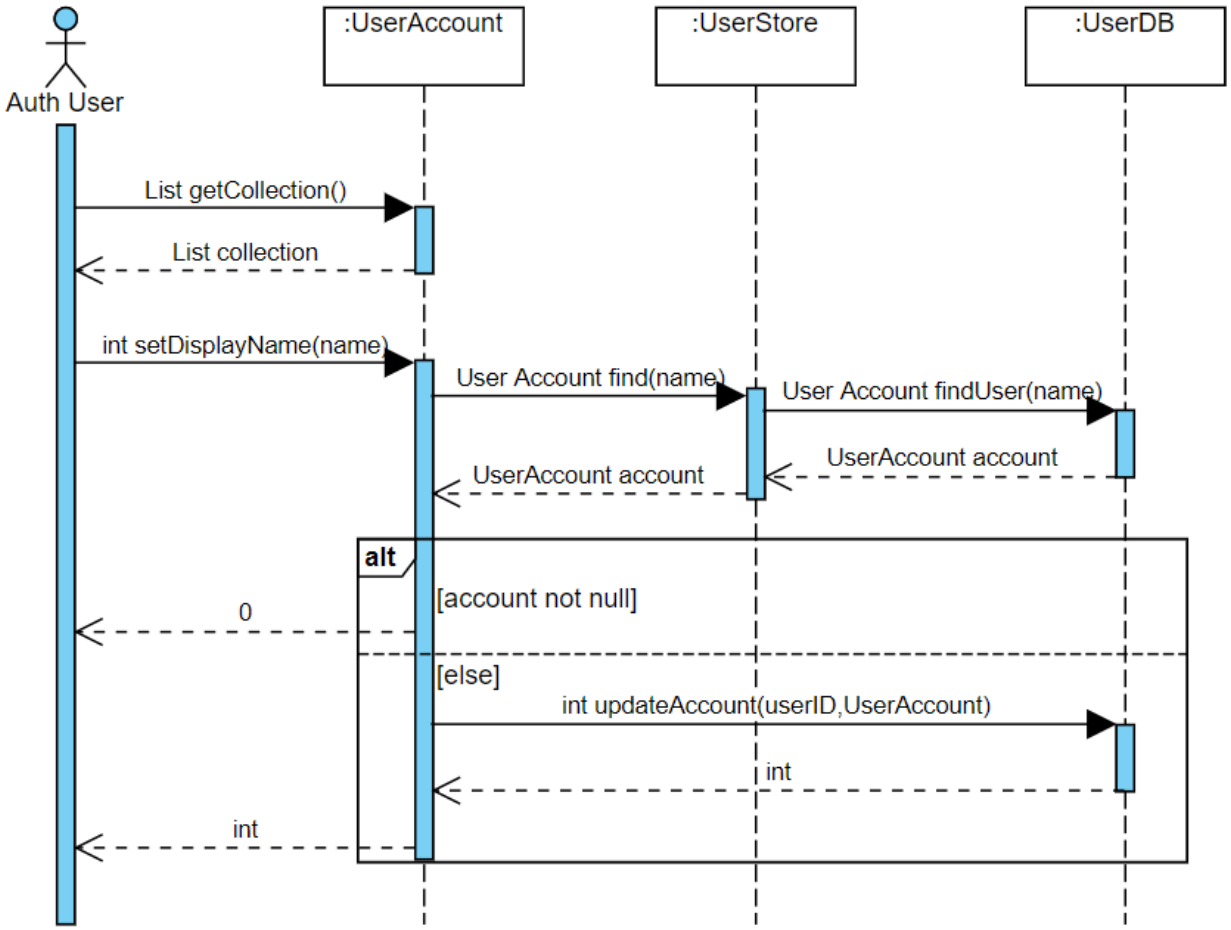


- **Add a location**

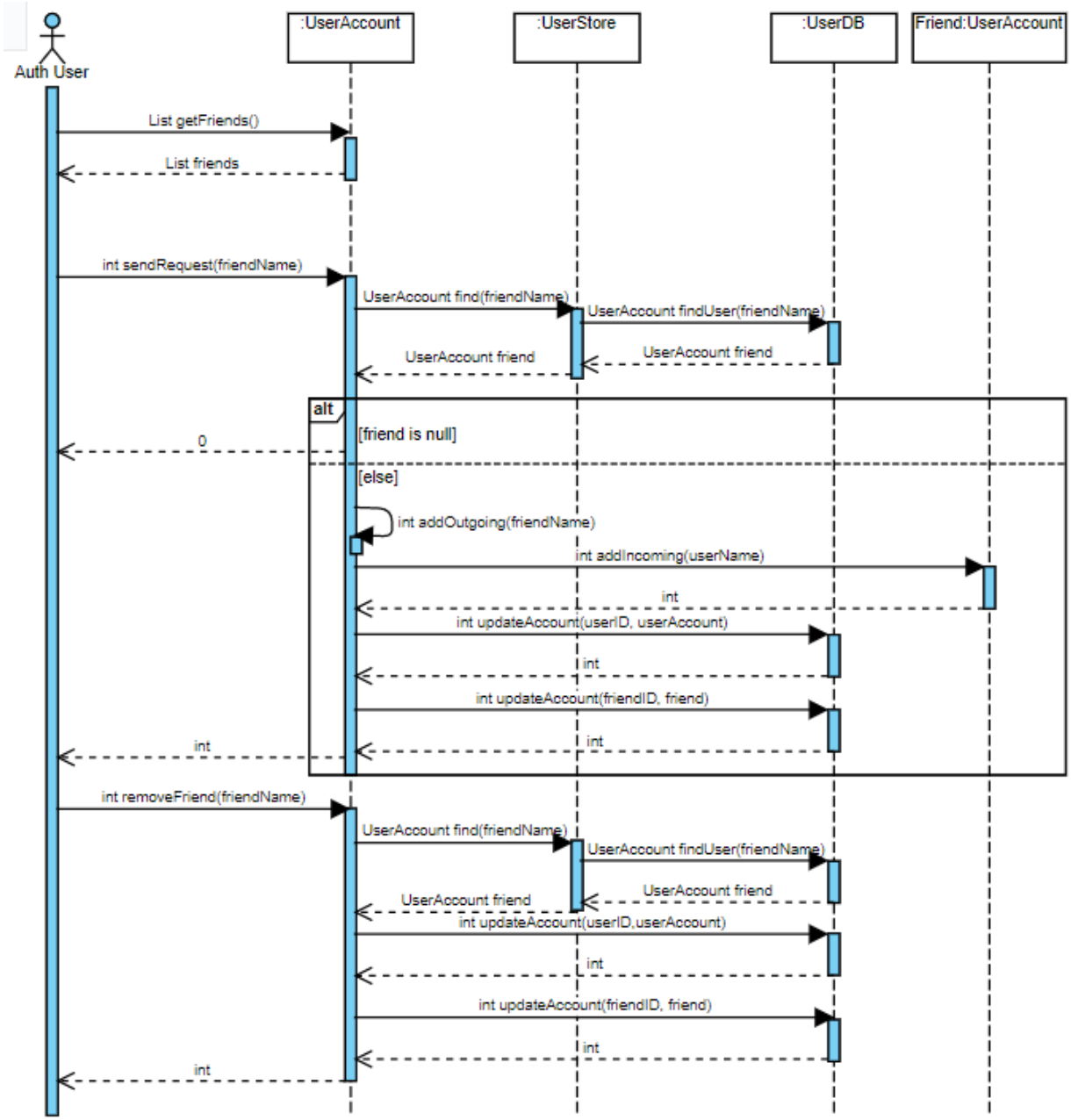


- **Manage profile**

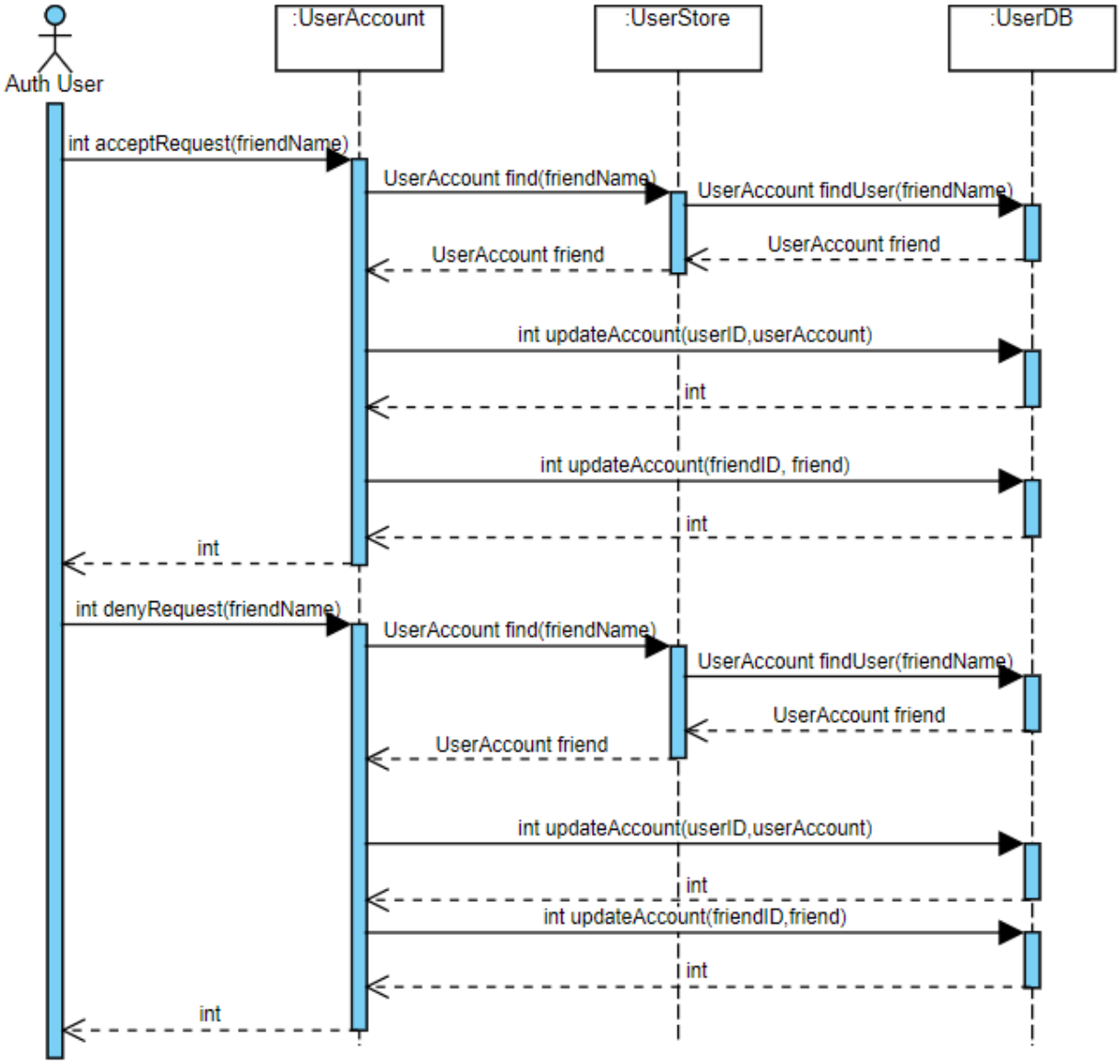
Opening the profile and changing name:



- **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

Member Contributions:

- Akshat: Realizing Non-Functional Requirements, Puzzle and login/logout sequence diagrams. Some parts of framework justification. Module Dependency Diagram, Data Types, Redo Puzzle UseCase.
- Jane: Created sequence diagrams for view global and friends leaderboards, participate in leaderboards, view messages, add messages, and explore locations after reaching the location. Added to main-modules and submodules and list of interfaces.
- Dylan: Created sequence diagrams for add location, change difficulty, manage profile and getting directions to location. Helped with project revisions, such as revising use cases, and creating managing profile use case. Helped with identifying main modules/sub-modules and interfaces.
- Mei: Explained Design for beyond the minimal scope functionality. Explained the advantages of the framework used. Revised use cases and non-functional requirements.