

AR Geocache Android Application Final Report Document

Prepared by:

Yousef Bayoumi (3632485)

Sam MacPhee (3581999)

Morgan Mollins (3559709)

Vandanaa Poteeram (3567836)

Prepared for:

Yevgen Biletskiy

Prepared on:

April 13, 2022

Table of Contents

1. Project Plan	5
1.1 Project Plan overview	5
1.1.1 Purpose of the Plan	5
1.1.2 Management	5
1.1.2.1 Organization	5
1.1.2.2 Responsibilities and Tasks	5
1.2 Risk Monitoring, Mitigation, and Management	6
1.2.1 Implementation Challenges	6
1.2.2 Major Changes in System Requirements (Change in project scope/direction)	6
1.2.3 Team Member Unavailable	6
1.2.4 Loss of Data	7
1.2.5 Conceptual Difficulties	7
1.2.6 Underestimating Time	7
1.2.7 Testing Issues	7
1.3 Quality Assurance plan	8
1.3.1 Formal Technical Reviews	8
1.3.1.1 Purpose	8
1.3.1.2 Format	8
1.3.1.3 Review requirements	8
1.3.2 Test Plan	8
1.3.2.1 Unit testing	8
1.3.2.2 Integration Testing	8
1.3.2.3 Validation and Verification Testing	9
1.4 Project Schedule	9
1.5 Project Cost Estimation	9
1.5.1 Number of User Inputs	9
1.5.2 Number of Outputs	10
1.5.3 User Inquiries	10
1.5.4 Number of Files	10
1.5.5 External Interfaces	10
1.6 Function Point Analysis	11
2. Requirements Analysis	13
2.1 Introduction	13
2.1.1 Purpose of the system	13
2.1.2 Scope of the system	13

2.1.3 Objectives of the project	13
2.1.4 Definitions, acronyms, and abbreviations	13
2.2 Current System	13
2.3 Proposed system	14
2.3.1 Overview	14
2.3.2 Functional requirements	14
2.3.3 Non-functional requirements	15
2.3.3.1 Usability	15
2.3.3.2 Reliability	15
2.3.3.3 Implementation	15
2.3.3.4 Maintainability	15
2.3.3.5 Legal	15
2.3.4 System Models	16
2.3.4.1 Use Case Model	16
2.3.4.2 Object Model	23
2.3.4.3 Dynamic Model	24
2.3.4.4 User Interface	26
3. Software Design	33
3.1 Introduction	33
3.1.1 Purpose of the system	33
3.1.2 Design Goals	33
3.1.3 Definitions, acronyms and abbreviations	33
3.2 Current software architecture	33
3.3 Proposed software architecture	34
3.3.1 Overview	34
3.3.2 Subsystem decomposition	34
3.3.3 Hardware/Software mapping	34
3.3.4 Persistent Data Management	35
3.3.5 Access Control and Security	35
3.3.6 Global Software Control	35
3.3.7 Boundary conditions	36
3.3.7.1 Initialization	36
3.3.7.2 Termination	36
3.3.7.3 Failure	36
4. Component Design	37
4.1 Introduction	37
4.1.1 Object Design Trade-Offs	37

4.1.1.1 BUY vs BUILD	37
4.1.1.2 RUNTIME vs STORAGE SPACE	37
4.1.2 Interface Documentation Guidelines	37
4.1.2.1 Naming conventions	37
4.1.2.2 Layout conventions	37
4.1.2.3 Commenting conventions	38
4.1.2.4 Language conventions	38
4.1.3 Glossary	38
4.1.4 References	38
4.2 Packages	39
4.2.1 Model Package	39
4.2.1.1 User Data Package	39
4.2.1.2 Item Data Package	39
4.2.2 View Package	39
4.2.2.1 Menu View Package	39
4.2.2.2 Map View Package	39
4.2.3 Controller Package	40
4.3 Class Interfaces	40
4.3.1 WelcomePage.js	40
4.3.2 LoginPage.js	40
4.3.3 Register.js	40
4.3.4 MainMenu.js	41
4.3.5 ProfilePage.js	41
4.3.6 EditProfilePage.js	41
4.3.7 Map.js	42
4.3.8 Inventory.js	42
4.3.9 PlaceGeocache.js	42
4.3.10 NavigateToGeocache.js	43
4.3.11 LocateGeocache.js	43
4.3.12 SearchUsers.js	43
Appendix A	43
A.1 Miscellaneous Javascript Classes	44
A.1.1 AuthProvider.js	44
A.1.2 Geocaching.js	45
A.2 Database Functions	49
A.2.1 pickUpGeocache	49
A.2.2 dropGeocache	49
A.2.3 updateGeocache	50

A.2.4 getListOfGeocaches	50
A.2.5 getUsername	51
A.2.6 insertUser	51
Appendix B	52
B.1 Test Plans	52
B.1.1 Unit Testing	52
B.1.2 Integration Testing	52
B.1.3 Prototype Testing	53
Appendix C	53
C.1 User Flow - User Interface Screens	54

List of Figures

Df

1. Project Plan

1.1 Project Plan Overview

1.1.1 Purpose of the Plan

The purpose of this plan is to:

- Ensure that the project is completely on time
- To detail the process which will be used to complete the project
- To define each team member's roles and responsibilities
- To ensure quality of the system
- To assess the risks and costs of the project

1.1.2 Management

1.1.2.1 Organization

Vandanaa Poteeram, Project manager/Software Engineer
Sam MacPhee, Integration manager/Software Engineer
Morgan Mollins, QA Lead/Software Engineer
Yousef Bayoumi, Dev Lead/Software Engineer

1.1.2.2 Responsibilities and Tasks

Project manager/Software Engineer

- Manage the Trello board.
- Prioritize and assign tasks.
- Develop and test features.

Integration Manager/Software Engineer

- Manage Git. (file organization, merge requests)
- Integration testing. Testing the product as a whole each sprint when new code has been merged, and informing developers when their code is not functioning properly.
- Develop and test features.

QA Lead/Software Engineer

- Main job is to ensure quality in the final product. This includes testing individual parts of the project and downloadable prototypes.
- Reporting to developers when bugs or inconsistencies are found in the code.
- Develop and test features.

Dev Lead/Software Engineer

- Responsible for the software architecture of the product.
- Responsible for managing the database.
- Develop and test features.

1.2 Risk Monitoring, Mitigation, and Management

1.2.1 Implementation Challenges	Probability: Moderate	Impact: Moderate
Mitigation	Consult with the team about methods of implementation; research implementation methods to use.	
Monitoring	Check progress regularly (compare to project timeline).	
Management	Ask teammates or professor for assistance; search for solutions to challenges encountered.	

1.2.2 Major Changes in System Requirements (Change in project scope/direction)	Probability: Low	Impact: High
Mitigation	Lay out system requirements in detail at the beginning of development; ensure feasibility.	
Monitoring	Compare project progress against project plan regularly; ensure that development meets planned requirements.	
Management	Meet as a team to determine how the project has to change to accommodate the change in requirements.	

1.2.3 Team Member Unavailable	Probability: Low	Impact: High
Mitigation	Communicate often and effectively; provide team members with general schedule/upcoming events that affect availability.	
Monitoring	Check in with team members frequently.	
Management	Re-assign unavailable person's responsibilities to other team members; report to professor about absence if situation is drastic.	

1.2.4 Loss of Data	Probability: Low	Impact: High
Mitigation	Backup data regularly; use version control (Git).	
Monitoring	Track all changes to the system; keep records of changes made.	
Management	Revert to the most recent recoverable version. Adjust project timeline and/or scope if needed.	

1.2.5 Conceptual Difficulties	Probability: Low	Impact: High
Mitigation	Perform extensive research on material the project requires.	
Monitoring	Communicate with team members about material.	
Management	Ask for assistance from team members, professor, or other sources.	

1.2.6 Underestimating Time	Probability: High	Impact: Moderate
Mitigation	Set a defined schedule and project timeline.	
Monitoring	Reference timeline; compare to project progress (Trello).	
Management	Edit scope of component; Reallocate resources to complete component on time.	

1.2.7 Testing Issues	Probability: Low	Impact: Moderate
Mitigation	Detail test cases; develop systems with unit <u>and</u> integration testing in mind.	
Monitoring	Report on each test performed on the system.	
Management	Re-work the software being tested; check test to ensure it is being performed correctly.	

1.3 Quality Assurance plan

1.3.1 Formal Technical Reviews

1.3.1.1 Purpose

A way of tracking the progress of the project throughout the development process and ensuring that it is on the right track.

1.3.1.2 Format

All formal technical reviews will follow the same format, with the material under review being discussed by the team. All issues will be recorded and addressed.

1.3.1.3 Review Requirements

Project plan review

- A reasonable project schedule must be established
- A cost estimation analysis must be completed and appropriate resources allocated.
- An appropriate quality assurance plan must be in place.

Software requirements review

- All requirements specified within the project scope must be incorporated into analysis models.
- All possible use cases must be considered.
- All external and internal interfaces must be considered.
- Requirements must be consistent with project schedule and resources.

Design review

- All requirements must be reflected in design.
- Interfaces and data structures must be considered.

Test plan review

- Test plan must include unit and system test cases for all functionality and use cases.

1.3.2 Test Plan

1.3.2.1 Unit Testing

- As each module of the system is being developed, its functionality will be tested with unit tests.

1.3.2.2 Integration Testing

- When the modules are integrated together, additional testing will be made to make sure that they work together without losing any of their functionality.

1.3.2.3 Validation and Verification Testing

- Validation and verification testing will be done periodically to make sure that the system is functioning as intended and meets what is specified within the project scope and requirements.

1.4 Project Schedule

We will have 8 - 2 week sprints:

Sprint 1: October 26th - November 9th

- RAD
- Configure database

Sprint 2: November 9th - November 23rd

- SDD
- Create Login/create profile pages

Sprint 3: November 23rd - December 7th

- Research AR functions

Sprint 4: January 4th - January 18th

- AR Functionality

Sprint 5: January 18th - February 1st

- AR Functionality

Sprint 6: February 1st - February 15th

- Online features
- Reward systems

Sprint 7: February 15th - March 1st

- Online features
- Customization options
- Integration testing

Sprint 8: March 1st - March 15th

- Testing
- Backlog

*Subject to change based on Agile development

1.5 Project Cost Estimation

To estimate the project cost, Function Points were used.

1.5.1 Number of User Inputs

Function	Count	Weight
User Login	2	3
Create Profile	4	3

Edit Profile	3	3
Search	1	3
Location	1	10
Camera	1	10
Place geocache	2	15
Touch gestures	10*	7

**There was no need to consider different touch gestures so this is not considered*

1.5.2 Number of Outputs

Function	Count	Weight
User Login	2	3
Create Profile	1	3
Edit Profile	1	3
Search	1	3
Map	1	7
Display Geocache	2	10

1.5.3 User Inquiries

Function	Count	Weight
Search	1	3

1.5.4 Number of Files

Function	Count	Weight
User Login	1	7
Profile	1	7
Geocache	1	15
Map	1	10

**subject to change*

1.5.5 External Interfaces

Function	Count	Weight
0	0	0

Total Count = 164 (Not considering touch gestures)

1.6 Function Point Analysis

To calculate the number of function points, 14 questions about the system were answered and given a score between 0 and 5, where 0 is no influence and 5 is essential.

1. Does the system require reliable backup and recovery?
4
2. Are data communications required?
5
3. Are there distributed processing functions?
0
4. Is performance critical?
4
5. Will the system run in an existing, heavily utilized operational environment?
2
6. Does the system require online data entry?
5
7. Does the online data entry require the input transaction to be multiple screens or operations?
1
8. Are the master files updated online?
5
9. Are the inputs, outputs, files or inquiries complex?
2
10. Is the internal processing complex?
5
11. Is the code designed to be reusable?
5
12. Are conversion and installation included in the design?
1
13. Is the system designed for multiple installations in different organizations?
2
14. Is the application designed to facilitate change and ease of use?
4

Function point total = 45

$$\begin{aligned}\text{FP} &= \text{Count total} * [0.65 + 0.01 * \text{Function Point total}] \\ &= 234 * [0.65 + 0.45] = 257.4\end{aligned}$$

We can then calculate our efficiency as such:

$$\text{Efficiency} = 18^*/\text{FP} = 18/257.4 = 0.06993 \text{ pm/FP}$$

**estimated human effort in a month*

2. Requirements Analysis

2.1 Introduction

2.1.1 Purpose of the System

The proposed system is a game application designed for mobile devices that is a self-contained AR version of geocaching. The system will allow users to place and find geocache items at a specific location using their device's camera. Players will be able to see the general area of a placed geocache on a map. Digital items that users can collect will be hidden in each geocache and are awarded to the player who discovers where they were hidden.

2.1.2 Scope of the System

The system must contain certain functions for the purpose of this project. The system must allow the user to create an account and log into the game, as well as update their profile in-game. The system must also make use of an accurate map system that uses location services, as well as databases for the geocaches and other items to be stored. The system must allow the user to use their camera to place and capture geocaches, which will also require specific GPS location data to be used to accurately display the geocaches in the AR view on the user's mobile device.

2.1.3 Objectives of the Project

The objective of the project is to produce a working prototype of the geocaching app that focuses on user experience and a clean user interface. The data for the app will be hosted on an independent server that can be accessed by user mobile devices through the cloud. **To ensure security, sensitive user information (like passwords) will be encrypted*** and the database will be secured from being accessed directly. The app will be developed for Android platforms for the purpose of this project, with a possible iOS version being developed later in the future.

***The database already handles security and encryption**

2.1.4 Definitions, Acronyms, and Abbreviations

- AR (Augmented Reality): a form of virtual reality where digital objects are made to appear in real space in the view of a camera feed.
- Geocaching/Geocaches: an activity where people hide small containers with rewards in them in locations throughout a community. The containers, known as geocaches, may have GPS trackers attached to them so that others can track the general area that they have been hidden in through a mobile device.

2.2 Current System

Currently the activity of geocaching is partially (if not entirely) physical, as the geocaches themselves are physical objects that have to be placed by players in locations that they can easily access. In addition, the geocaches are exposed to the elements around its hiding place, possibly

worsening the condition of the container and contents and exposing it to bacteria and other organisms. While some applications exist that allow for tracking the placed geocaches, the player will still have to physically grab the geocache once they find it.

2.3 Proposed system

2.3.1 Overview

The proposed system allows users to partake in the hobby of geocaching virtually in a simple game-like, easy-to-use environment. Users should be able to go to a real-world location by using a GPS map and open their camera and with the assistance of Augmented Reality technology, they are able to place digital objects they have stored on their phone in that real-world location. Other users within the area are alerted if there are nearby digital items or “geocaches” that could be obtained in the area and by using their camera and AR technology can look for that specific geocache. This app provides users an easy way to enjoy the hobby of geocaching from their smartphone.

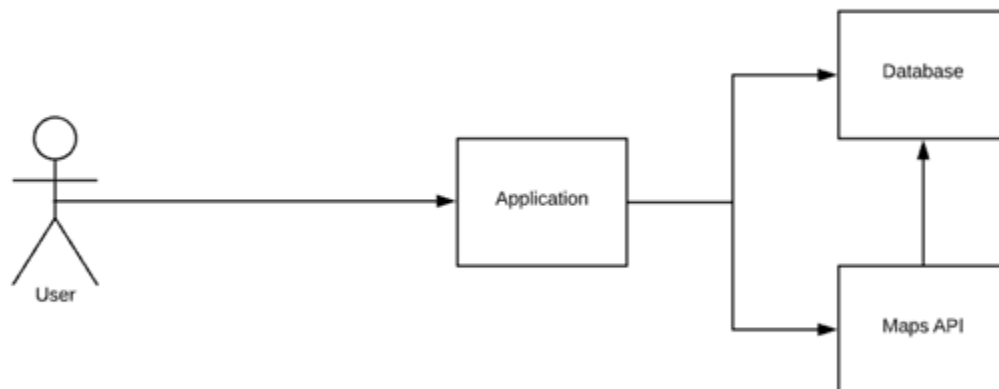


Figure 2.1 *Conceptual Model for the system*

2.3.2 Functional Requirements

The Augmented Reality Geocaching App software system works like the generic mobile application. The system allows a user to log in with a username and password, and if they are a new user, allows them to create a profile, which is then stored for future usage like auto logging in when users run the app again. Once a user owns a profile, they have access to the main interface of the app. There will be a map displayed showing the current geographical location of the user and the map will mark any areas close to the user which contain geocaches. Users can then walk to an area and press a button in order to bring up the phone’s camera which uses AR technology in order to display a geocache if the user is properly looking at it with their camera.

A user can capture geocaches with this function while also being able to hide geocaches by opening the camera and tapping on the location that the camera has a view of. In the main screen, there will also be a rewards button which when tapped brings up the rewards tab that shows the user current level and rewards they have acquired/ yet to acquire.

2.3.3 Non-Functional Requirements

2.3.3.1 Usability

There are a few operations that the user is going to make use of. These operations are also not very complex which makes the system very simple and easy to use. The system is made with the intention of making the hobby of geocaching as straightforward and enjoyable as possible for the user. **There is also a tutorial that pops up when users run the app for the first time which informs them how the app works and what each button does.** Moreover, the system should perform its tasks smoothly and efficiently so that the user finds the experience satisfactory, and willing to run the app again.

***The tutorial will not be implemented in the app.**

2.3.3.2 Reliability

The reliability of the system is very important. The system must react as the user expects, and perform its tasks to a reasonable degree of satisfaction from the user. Furthermore, the system should create backups to save the user's data in case of any system failures or the user deciding to install the app again. Thus, reliability will be focused heavily during development.

2.3.3.3 Implementation

There are few implementation requirements for the AR Geocache application system. The application will be mainly used on Android devices so the development will be made with that in mind. The majority of the data will be stored in an external database to ensure backup and safety.

2.3.3.4 Maintainability

To ensure maintainability, the system will be consistent in development style. The app itself should be simple in design and as such allows for good readability and understandability. Therefore, this simplicity of the system minimizes the work that should be done in order to update and maintain different parts of the system. The implementation of the system should be documented and organized properly. The system should also be modular in design which makes the system very adaptable and easy to manage.

2.3.3.5 Legal

The system will be licensed under as many licences which best fit with the system's requirements – both functional and non-functional. If the system makes use of existing code or modules which require to be licensed within the system, then they will be made.

2.3.4 System Models

2.3.4.1 Use Case Model

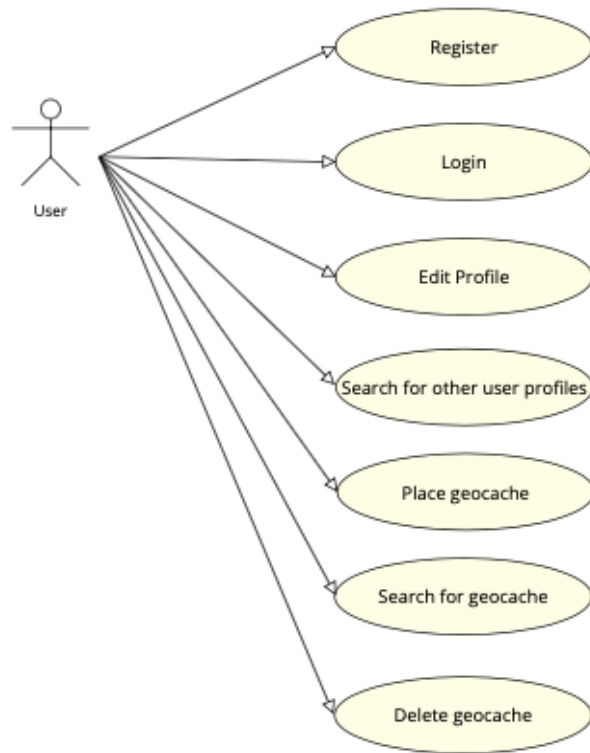


Figure 2.2 Use Case Diagram

USE CASE NAME	USER REGISTRATION
PARTICIPATING ACTOR(S)	User

FLOW OF EVENTS	<ul style="list-style-type: none"> - User opens Application on their mobile phone. - Application directs User to the main screen. - User selects the Register button on the main screen. - Application displays the Registration page. - User enters their username and password. - User adds a profile photo (optional). - User clicks the Submit button. - Application checks the User information and saves the User profile.
ENTRY CONDITION(S)	Geocache Application must be installed on the User's mobile phone.
EXIT CONDITION(S)	User creates an account successfully and is directed to the Login screen or User fails to create an account and has to modify their details to do so.
QUALITY REQUIREMENTS	Registration page is quick and responsive to the User and is easy to navigate through.

USE CASE NAME	USER LOGIN
PARTICIPATING ACTOR(S)	User

FLOW OF EVENTS	<ul style="list-style-type: none"> - User opens Application. - Application directs User to the main screen. - User clicks on the Login button on the main screen. - Application displays the Login screen. - User enters their username and password. - User clicks on the Enter button. - Application validates the User Information and displays the main menu dashboard.
ENTRY CONDITION(S)	Geocache Application must be installed on the User's mobile phone and User launches Application.
EXIT CONDITION(S)	User is directed to the main menu dashboard upon successful login or User has to re-enter their username and password upon unsuccessful login.
QUALITY REQUIREMENTS	The Application displays the main screen within 10 seconds when it is launched. The Login screen loads within 10 seconds when the User clicks on the Login button. User is directed to the main menu dashboard within 15 seconds after the Application validates their credentials.

USE CASE NAME	USER PROFILE EDIT
PARTICIPATING ACTOR(S)	User
FLOW OF EVENTS	<ul style="list-style-type: none"> - User clicks on the Profile icon on the main menu dashboard. - Application directs User to their Profile screen. - User clicks on the Edit icon to make changes to their profile.

	- After making the necessary changes, User clicks on the Save button to save the changes.
ENTRY CONDITION(S)	User has to be logged into their account.
EXIT CONDITION(S)	User receives a message saying either the changes to their profile have been made or the reason why change has failed.
QUALITY REQUIREMENTS	User should receive the acknowledgement of the changes made/failed in a reasonable amount of time, depending on the internet connection of the User's phone.

USE CASE NAME	SEARCH OTHER USER PROFILES
PARTICIPATING ACTOR(S)	User
FLOW OF EVENTS	<ul style="list-style-type: none"> - User clicks on the Search icon on the main menu dashboard - User types the username of the User profile they are looking for in the search box. - User clicks on the OK icon. - Application displays the list of Users with the same username typed by the User. - User clicks on the User Profile they wish to view.
ENTRY CONDITION(S)	User should be logged into their account and the user they are looking for should be an existing user of the Application. The User's phone should be connected to the Internet to do the search.

EXIT CONDITION(S)	User sees a list of users with usernames matching the User's search or Application displays a message stating no matches have been found.
QUALITY REQUIREMENTS	The search should be completed in a reasonable amount of time, depending on the Internet connection and size of the list of users.

USE CASE NAME	USER PLACES GEOCACHE
PARTICIPATING ACTOR(S)	User
FLOW OF EVENTS	<ul style="list-style-type: none"> - User clicks on the Map icon on the main menu dashboard. - Application opens the Map and shows the location of the User and that of nearby existing geocaches. - User clicks on the + button to place a geocache on the Map. - Application displays a list of geocaches types. - User selects the type of geocache from the list and enters its description. - User clicks OK to confirm the details of the geocache. - User places geocache at their desired location. - Application generates a confirmation message to confirm location of the geocache. - User clicks on the OK icon to confirm the location of geocache.
ENTRY CONDITION(S)	User needs to be logged into their account and Application needs to have access to the phone's location. An Internet connection is needed.

EXIT CONDITION(S)	User gets a confirmation stating if the geocache has been successfully placed or not.
QUALITY REQUIREMENTS	Application displays the list of geocaches within 10 seconds after User clicks on the + button. User should get an acknowledgement within a reasonable amount of time.

USE CASE NAME	USER SEARCHES FOR GEOCACHE
PARTICIPATING ACTOR(S)	User
FLOW OF EVENTS	<ul style="list-style-type: none"> - User clicks on the Map icon on the main menu dashboard. - Application opens the Map and shows the location of the User and that of nearby existing geocaches. - The User selects the geocache. - The Application displays the details of the geocache. - User clicks on the Navigate button to display the route to the geocache. - Upon reaching the location of the geocache, User opens the Camera App to look for geocache.
ENTRY CONDITION(S)	User needs to be logged into their account and the Application needs to have access to the phone's camera and location. An Internet connection is needed.
EXIT CONDITION(S)	User locates and finds geocache.
QUALITY REQUIREMENTS	Application loads the geocache details within 10 seconds when User selects geocache. Application displays route to

	geocache within 10 seconds when User clicks on the Get button.
--	--

USE CASE NAME	USER DELETES GEOCACHE
PARTICIPATING ACTOR(S)	User
FLOW OF EVENTS	<ul style="list-style-type: none"> - User clicks on the Map icon on the main menu dashboard. - Application opens the Map and shows the location of the User and that of nearby existing geocaches. - User clicks on the List button to view a list of the geocache they have placed. - User clicks on the Delete icon to delete geocache.
ENTRY CONDITION(S)	User needs to be logged into their account.
EXIT CONDITION(S)	User gets a confirmation stating if the geocache has been successfully deleted or not.
QUALITY REQUIREMENTS	Application should display the list of the User's geocaches within 10 seconds after User clicks on the button. User should get an acknowledgement within a reasonable amount of time.

*** This feature will not be implemented.**

2.3.4.2 Object Model

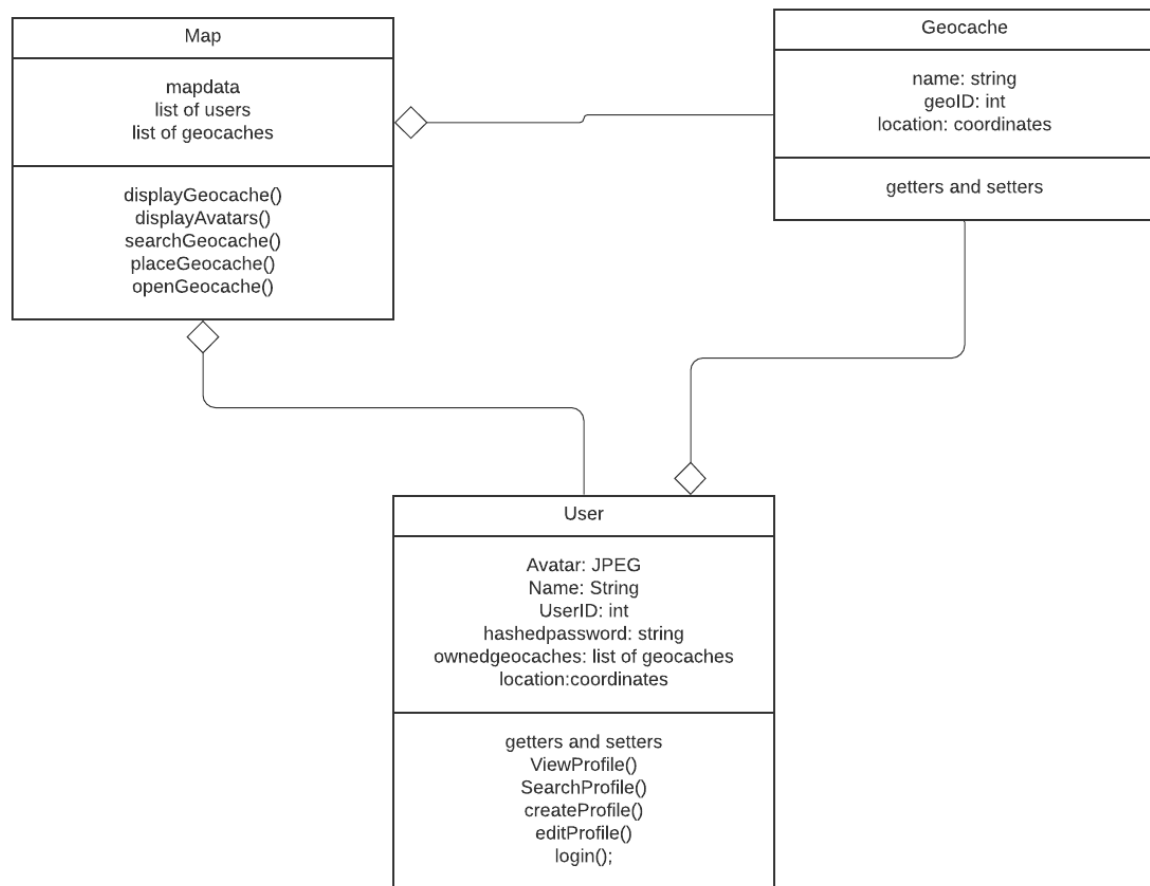


Figure 2.3 Class Diagram

2.3.4.3 Dynamic Model

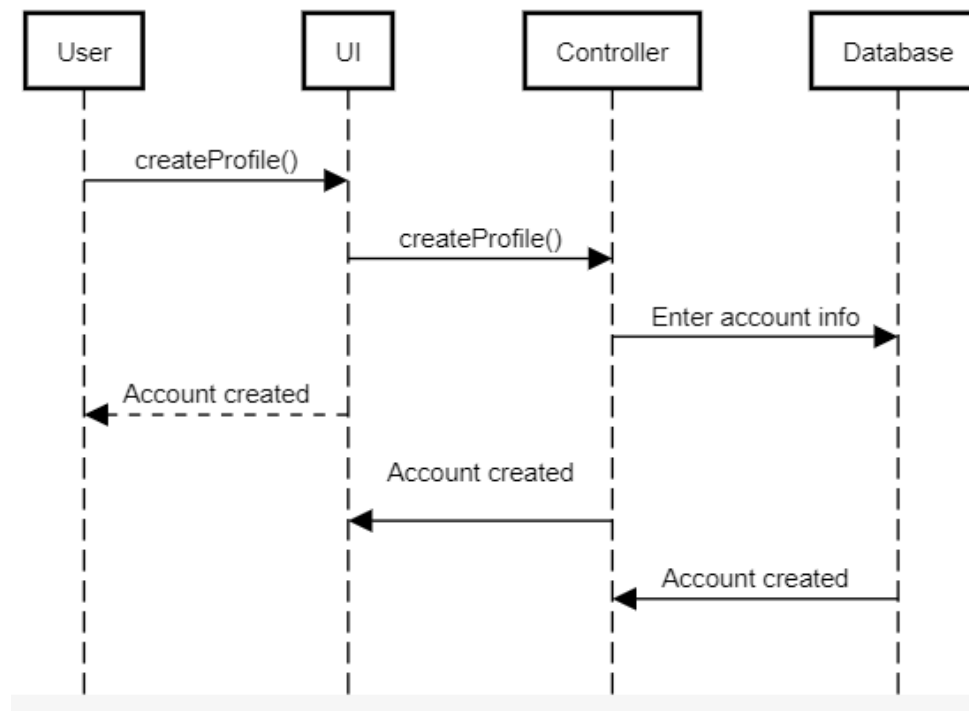


Figure 2.4 Create Profile Sequence Diagram

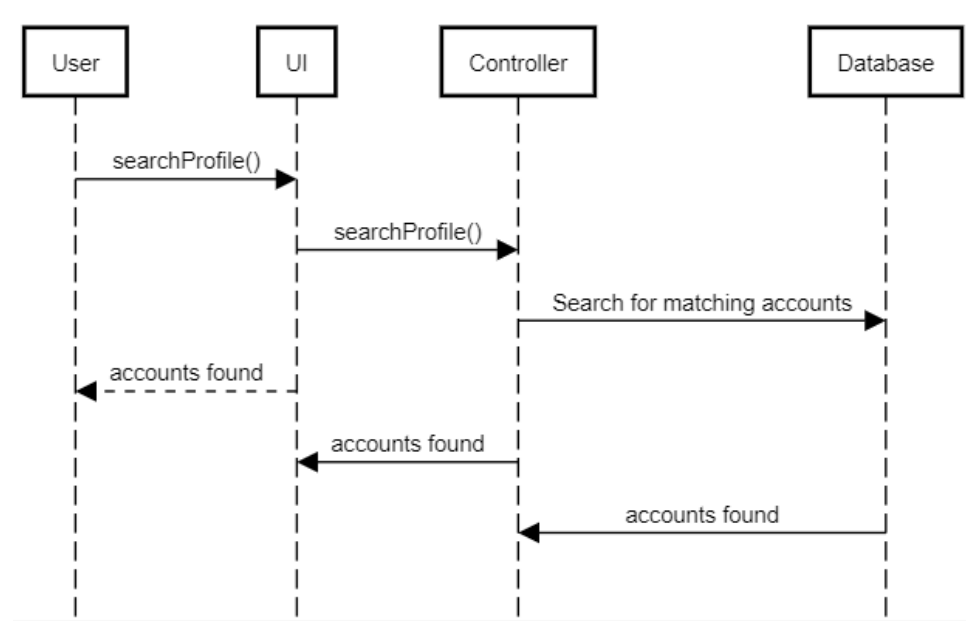


Figure 2.5 Search Profile Sequence Diagram

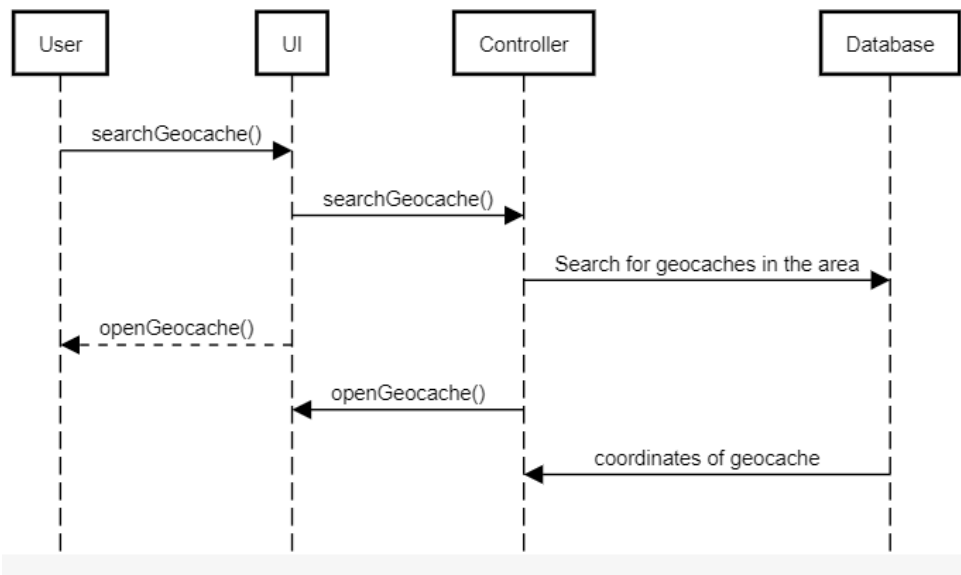


Figure 2.6 Search Geocache Sequence Diagram

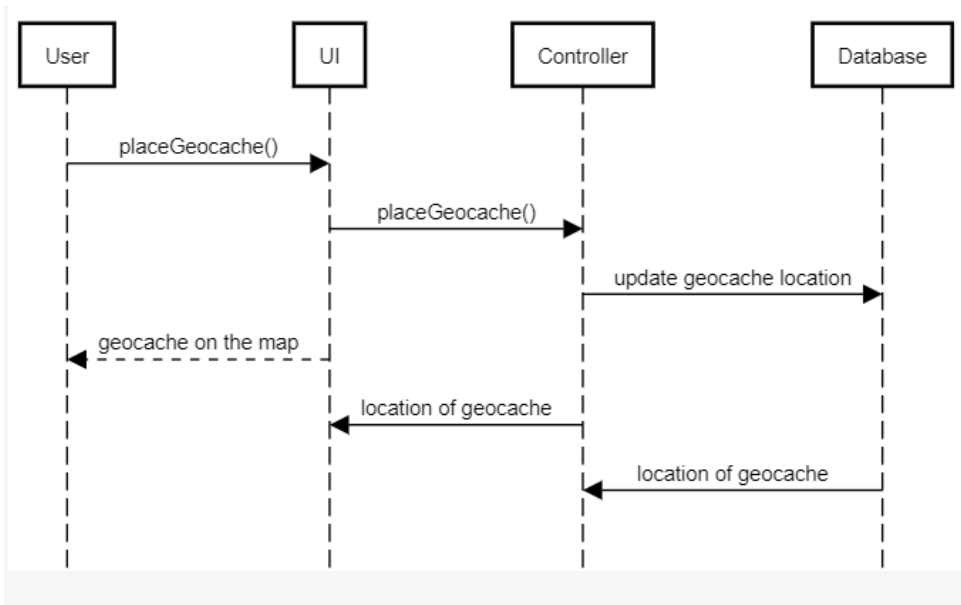
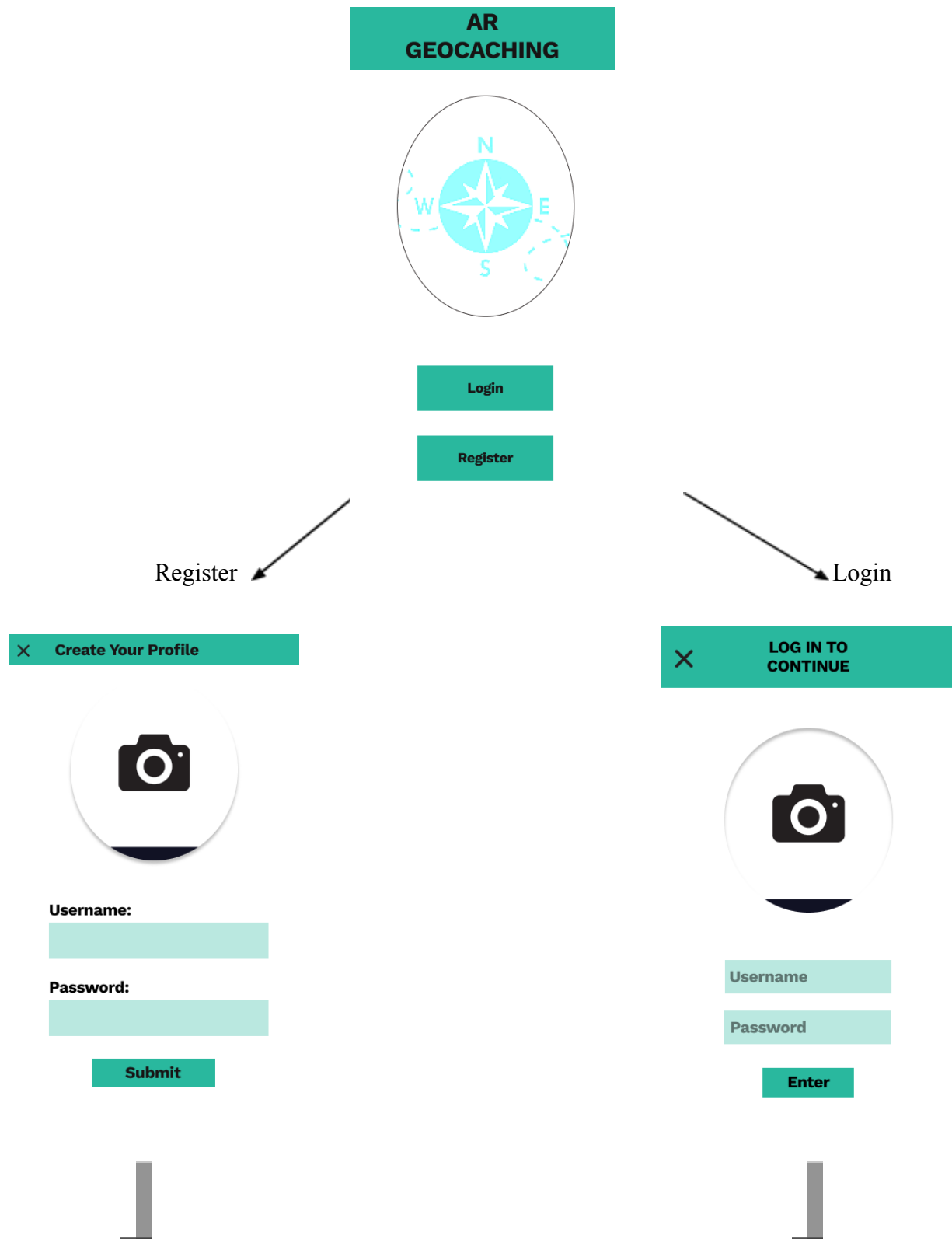
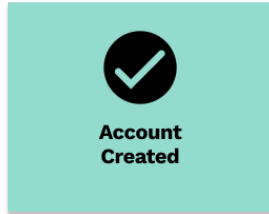


Figure 2.7 Place Geocache Sequence Diagram

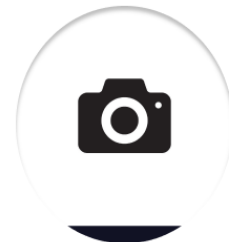
2.3.4.4 User Interface



Login screen

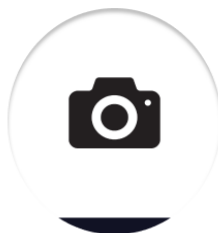


(Main menu dashboard)

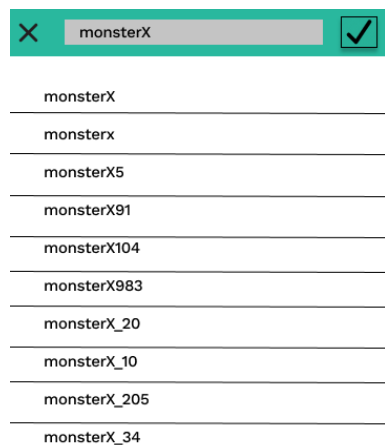
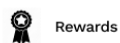


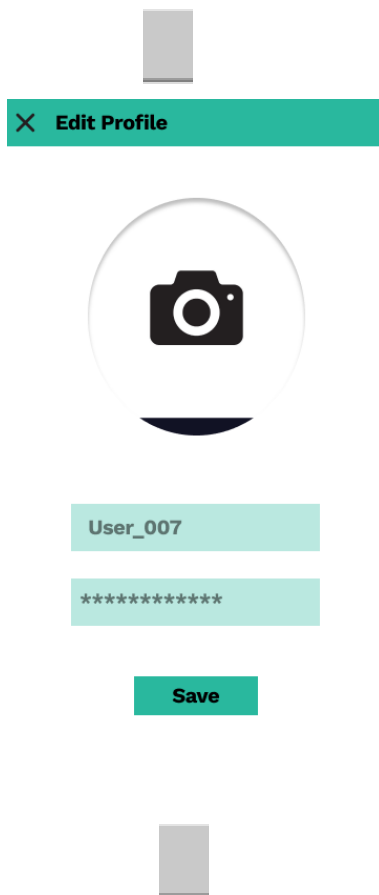
Edit Profile

Search
other
Users

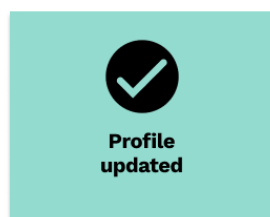


Beginner Level | Joined: 2021

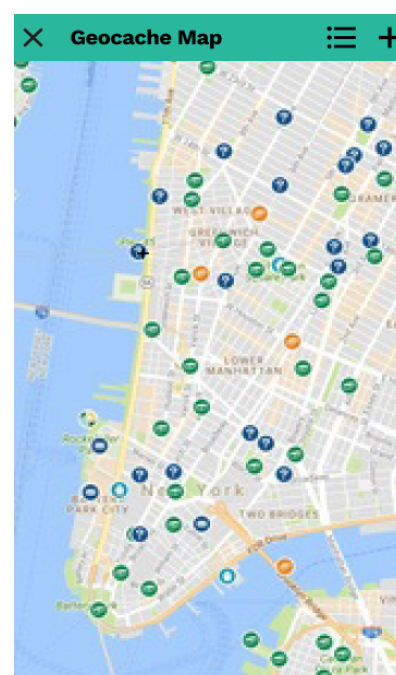
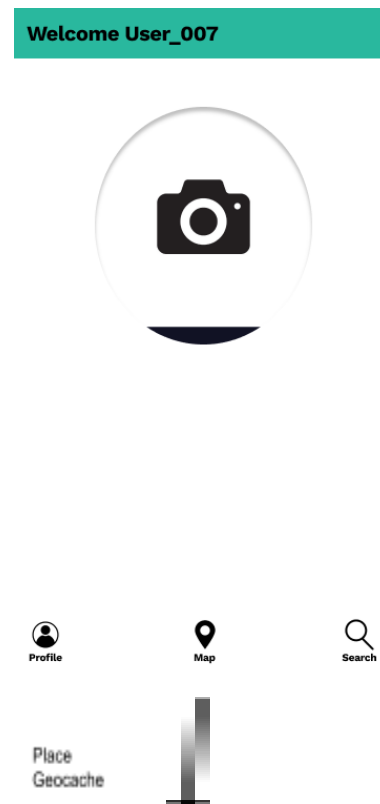




AR Geocaching



(Map use case 1)



Details

You are looking for a small hanging container.
Navigate to the geocache location and search for the hidden container. Read the details if stuck.

Difficulty: 1.5/5.0
Easy to find/solve.

Size: Small
Generally the size of a small soup container

OK

Geocache Types

- Micro
- Small
- Regular
- Large

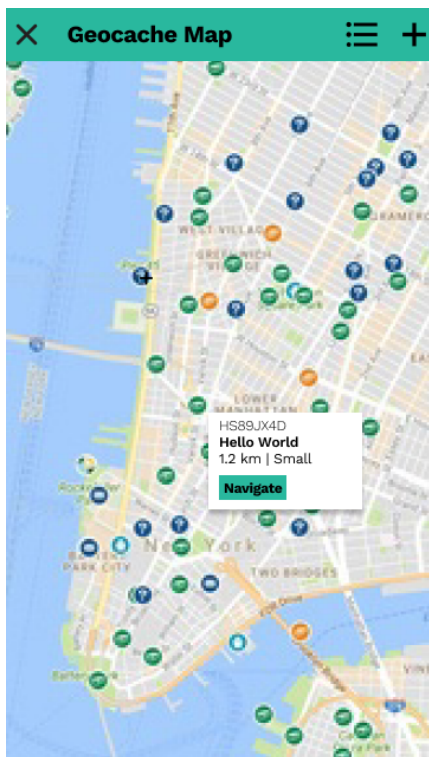


AR Geocaching



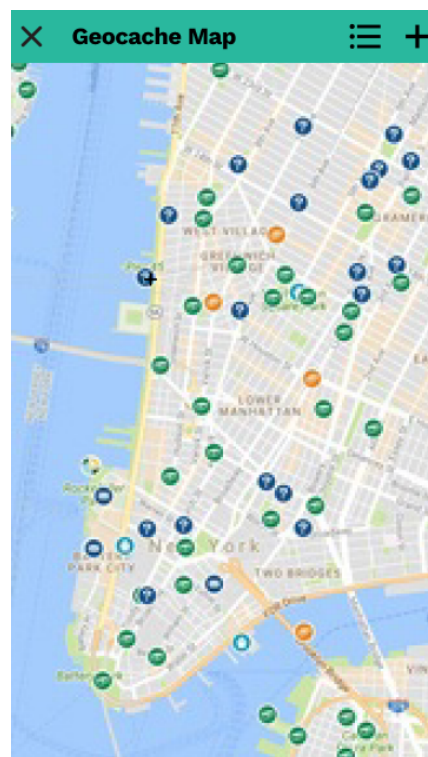
(Map use case 2)

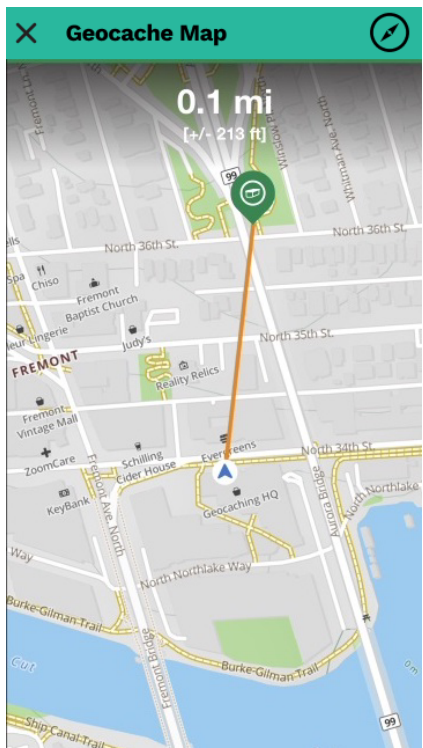
Welcome User_007



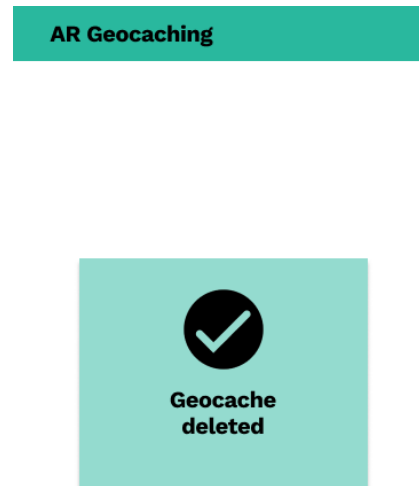
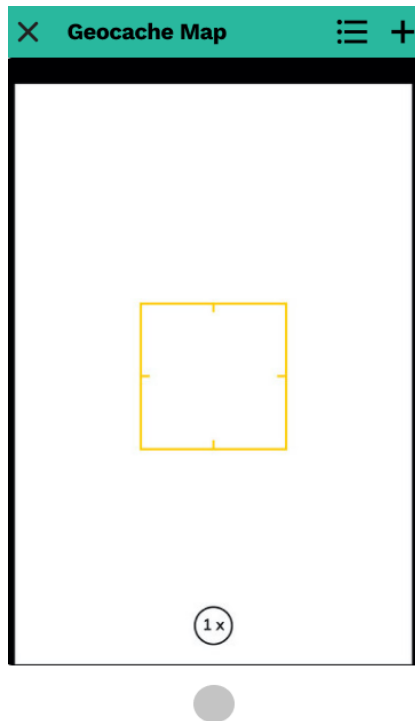
(Map use case 3)

Welcome User_007





Geocaches	
Hello World	
Christmas Surprise	
Covid-free gift	
Anime World	
K-pop Lovers	



3. Software Design

3.1 Introduction

3.1.1 Purpose of the System

The proposed system is a game application designed for mobile devices that is a self-contained AR version of geocaching. The system will allow users to place and find geocache items at a specific location using their device's camera. Players will be able to see the general area of a placed geocache on a map. Digital items that users can collect will be hidden in each geocache and are awarded to the player who discovers where they were hidden.

3.1.2 Design Goals

The main design goals of the AR Geocache Android Application are usability, reliability and maintainability.

Usability is an important factor to consider when designing the system because if the application is not simple and easy to use, the user might not have an enjoyable experience. The usability design goal is achieved through the use of simple and familiar UI components which will allow the users to navigate easily throughout the application.

The reliability of the system is very important. The system must react as the user expects, and perform its tasks to a reasonable degree of satisfaction from the user. Moreover, users will have the option to create backups of their data in case the application crashes or the application is re-installed.

The final design goal of the application is maintainability. The system will be consistent in development style. The application itself will be simple in design and as such allows for good readability and understandability. The system will also be modular in design which will make the system very adaptable and easy to manage.

3.1.3 Definitions, Acronyms and Abbreviations

- AR (Augmented Reality): a form of virtual reality where digital objects are made to appear in real space in the view of a camera feed.
- Geocaching/Geocaches: an activity where people hide small containers with rewards in them in locations throughout a community. The containers, known as geocaches, may have GPS trackers attached to them so that others can track the general area that they have been hidden in through a mobile device.

3.2 Current Software Architecture

Currently, the activity of geocaching is partially (if not entirely) physical, as the geocaches themselves are physical objects that have to be placed by players in locations that they can easily access. In addition, the geocaches are exposed to the elements around their hiding place, possibly worsening the condition of the container and contents and exposing it to bacteria and other organisms. While some applications exist that allow for tracking the placed geocaches, the player will still have to physically grab the geocache once they find it.

3.3 Proposed software architecture

3.3.1 Overview

The AR Geocache App utilises a Client-Server architecture pattern, with the application installed on the user's device acting as a client connected to a database server to retrieve the information needed for the game to function. The client application will locally store the data needed to run the UI of the system but will query the server to obtain the information for the game to function. The data stored in the database will include the following:

User credentials: The user's username and password, with the password encrypted for security purposes.

* Database already does encryption of private user information

Geocache data: The data related to each geocache placed on the game map. This data will consist of the geocache's GPS location, the user who placed it, when it was placed, and what the geocache contains.

* This information is not defined in the database

User profile data: The information provided by the user for their profile in-game.

User item data: The data relating to items that the user has obtained in-game from geocaches.

3.3.2 Subsystem Decomposition

The server-side subsystem will be a simple database query system that will take the data requests from the client-side and read/write the data in a database available to users online. The client-side subsystems will follow the Model-View-Controller architecture in terms of how they will handle the system UI and data requests.

Model

- Relay database queries to the server.
- Handle the states of the UI.

View

- Display the UI on the user's device.
- Change the displayed screen based on information obtained from the Model class.
- Update data-sensitive UI elements of a screen based on data obtained from the server.

Controller

- Handle the user inputs from the View class.
- Send user requests to the Model class.

3.3.3 Hardware/Software Mapping

The AR Geocache app will mainly be Android phone oriented but later on, IOS support could potentially be added. Most of the code will be executed on the device and with the provided user input, the system will perform the intended function. The software will make use of the phone's geographical location data to display the location of possible nearby geocaches as well as any friends nearby. The phone camera will be used to show the AR components of the system such as placing or acquiring geocaches. Initially, the system will support a few users logged in at a time, however, in the working product, the system is intended to have full multiuser support.

3.3.4 Persistent Data Management

The AR Geocache App will use MongoDB as the Database Management System. This will help to organize the data efficiently. The usernames and passwords of the users will be saved on the database and have a sha256 hashing applied to the passwords for security. Also, the real-time locations of users will be constantly updated to figure out the relative distance to the geocache coordinates.

* The database already handles encryption of passwords and sensitive data.

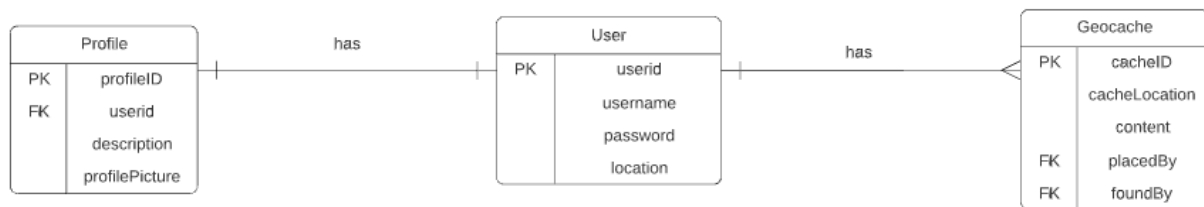


Figure 3.1 ER diagram for the system

3.3.5 Access Control and Security

Gaining access to the system will require the user to log into their account. The authentication method will be a username and a password. Users will be required to create an account on first opening the app. Usernames will have to be unique and passwords will be required to be a minimum of 8 characters long.

- **Login**
Users will have to log in by providing their unique username and password in order to enter the system.
- **Register**
Users will have to register on first opening the app. They will have to provide a unique username, and a password with a minimum of 8 characters long.
- **Profile Creation**
A user can create a profile, with a personal description, avatar, privacy settings and profile photo.
- **Profile editing**
A user can edit their own profile, but no one else's.
- **Search profiles**
Any user can search for any other user's profile unless that profile is set to private.

A user who is logged in will have full permission to all of the app's features.

3.3.6 Global Software Control

The AR Geocache app will be an online application. The user will require access to the internet in order to use the app. This means that all processes will be done through an online database service MongoDB. Users will be able to interact with other users on a map and look for the same geocaches.

System synchronization will be done through MongoDB's live objects. This means that geocache objects will be updated live so that users will always be able to access the most recent data

concerning them. Any time a user adds or removes a geocache, other users will receive that data update live on their device.

Since MongoDB allows for multiple users to modify the same data, it uses concurrency control and multi-granularity locking, which means that data will be able to be locked at a global level, a database level, or a collection level. Our system will use this to lock geocaches when a user interacts with them, so that no other user can interact with that geocache at the same time.

3.3.7 Boundary Conditions

3.3.7.1 Initialization

Upon initialization, the user will be directed to the main page of the application. The UI will present them with two options; to login if they are a returning user or to register if they are a new user.

If the user chooses to login, they enter their username and password which will be verified by the application to check if they are an existing user and upon successful login, the main menu dashboard will be displayed and the user can then decide how to proceed.

If the user chooses to register, they enter the username and password which will be verified by the application to check if they match the criteria and upon successful registration, the user will be redirected to the main menu dashboard where they can then decide how to proceed.

3.3.7.2 Termination

Termination for the user will occur when the user simply closes the application. The system will save any data when the user closes the application or if the application is accidentally closed.

3.3.7.3 Failure

Due to the nature of the data involved, which is by no means safety-or mission-critical, there is little need for extensive error or failure handling.

Software failures will be avoided for the most part due to extensive code testing completed before any new code is merged to the main branch. In case of any software failure, there will be no handling implemented and the user will be required to restart the application which will be equivalent to initialization.

4. Component Design

4.1 Introduction

4.1.1 Object Design Trade-Offs

4.1.1.1 BUY vs BUILD

We will be building most of the systems with the exception of the imported ARKit and ARCore that will be reused from Viro in order to develop the AR functionality of our app, as well as using the free version of MongoDB for our data solution.

4.1.1.2 RUNTIME vs STORAGE SPACE

Runtime will not be a concern in the development of our app since it will be largely dependent on internet connection, which is out of our control. Storage space will be prioritised to keep the filesize of the app small, this will be done using an efficient data solution using MongoDB. Overall, functionality of the app will be given the highest priority over runtime and storage space.

4.1.2 Interface Documentation Guidelines

4.1.2.1 Naming Conventions

- Folder and sub folder names should always start with a lowercase letter, and everything inside should follow a pascal case.
- All components should follow a pascal case and have a matching name to the corresponding path. For example components/common/Button.js would be named as Button.js.
- When a file name is the same as the folder name, there is no need to repeat the name. For example, components/common/Buttons/Buttons.js is correct and not component/common/Buttons/buttonButtons.js.
- There should be no spaces in between import statements. All imports that belong to the same module should be grouped together in braces and end with a semicolon.
- Class names must match file names.
- Object declaration should always be in a camel case, and there should always be a semicolon following.

4.1.2.2 Layout Conventions

- Always end each statement with a semicolon.
- Always use functional components unless using state is needed, then a class component should be used.
- All continuation lines should be indented 4 spaces.
- Leave one blank line between methods and property definitions.
- There should be no spaces between similar looking lines of code, and similar blocks of code that share functionality.

- Always use `shouldComponentUpdate()` before updating to render on demand instead of the application automatically re-rendering every time it's updated.

4.1.2.3 Commenting Conventions

- All comments should be placed on separate lines, not sharing a line with code.
- Comments should follow Canadian English language standards including proper capitalization, and punctuation.
- Include one space between the comment delimiter and the first comment character.

4.1.2.4 Language Conventions

- Variables can hold any type of data.
- Numbers can be floating point or integers, they do not use quotations
- Strings will always use double quotations.
- Backticks will be used when adding functionality and embedding other variables into a string.
- Boolean types will be either true or false with no capitalization or quotations.
- Arrays will be initialised by square brackets.
- Array elements are numbered starting from 0.
- Props (short for properties) are static variables that will never change.
- States will remain local to their component.

4.1.3 Glossary

- MongoDB is the application data platform we will be using for the application.
- ARCore is Google's platform for building AR mobile application functionality.
- ARKit is the Android development kit used for developing AR functionality.
- VIRO is an open source platform for mixed reality development that includes the packages for ARCore and ARKit.
- Pascal Case is the convention when the first letter in each word in a variable name is capitalised.

4.1.4 References

1. Rajbhar, S. (2019, April 12). *Coding standards and good practice for react native apps*. Medium. Retrieved February 8, 2022, from <https://sandeeprajbhartechgeek.medium.com/coding-standards-and-good-practice-for-react-native-apps-c8401e87f2d>
2. Patel, V. (2021, December 31). *What is augmented reality? how to implement AR using react native?* Medium. Retrieved February 8, 2022, from <https://medium.com/simform-engineering/what-is-augmented-reality-how-to-implement-ar-using-react-native-2340bdba9a8d>

4.2 Packages

The application will make use of three distinct packages based on the Model-View-Controller architecture. The Model and View packages can also be broken down into sub-packages based on the information they have to process.

4.2.1 Model Package

The Model package contains all of the back-end data and the classes that handle it. The classes in this package need to have access to the database server (MongoDB) to read/write the data used in the application.

The Model package can be split into two subsystems: User Data and Item Data.

4.2.1.1 User Data Package

The User Data package contains the classes that control how individual users' data is handled. This includes information like the player's login credentials and profile details.

4.2.1.2 Item Data Package

The Item Data package contains the classes that control the data relating to the in-game items that players can place/collect. This is primarily used for the game's geocaches, which are split into two groups: Geocaches that have been placed on the map, and the geocache items contained in each user's inventory. The latter category will require a reference to the User Data Package to relate the inventory items to the corresponding user.

4.2.2 View Package

The View package contains the classes used to display the UI on the user's device. These classes are dependent on the other two primary packages to retrieve and process the data displayed to and entered by the user.

4.2.2.1 Menu View Package

The Menu View package contains the View classes that display the static menu pages on the user's device. These classes will consist of the basic menu pages, the welcome and login screens, and the user's profile and inventory screens.

4.2.2.2 Map View Package

The Map View package contains the View classes that utilise the live maps of the user's area or the user's camera for the AR functionality. The live maps will use the React Native Maps system, which will automatically format the view to use the maps platform of the user's device (i.e. Google Maps on the Android OS).

4.2.3 *Controller Package*

The Controller package acts as the processing centre for the application. Classes in this package pass information between the Model and View packages, so it has dependencies connected to each.

4.3 **Class Interfaces**

4.3.1 *WelcomePage.js*

- The Welcome Page is the first page the user sees when he/she opens the Geocache App.
- The Welcome Page has two buttons, namely “Register” and “Login”.
- The “Register” button takes a new user to the Register Page and the “Login” button takes the user to the Login Page.


4.3.2 *LoginPage.js*

- The Login Page is the page where the user enters their credentials if they already own an account on the Geocache App.
- The Login Page has two text input fields named “Username” and “Password”. There is also a button named “Enter” and a “Back” button (represented by ↩).
- The “Username” text input field takes in a text value that represents the user’s username on the app.
- The “Password” text input field takes in a text value that represents the user’s password for their account on the app. The password text is shown as asterisks for privacy and security.
- The “Enter” button checks whether the entered username and password match any of the accounts saved on the database and returns a corresponding message on whether a successful login has been made or not.
- The “Back” button takes the user back to the Welcome Page.



4.3.3 *Register.js*

- The Register Page is the page where the user enters their credentials if they are a new user on the Geocache App.
- The Register Page has two text input fields named “Username” and “Password” along with a “Submit” button and a “Back” button (represented by ↩).
- The “Username” text input field takes a text value that the user inputs as their username.
- The “Password” text input field takes a text value that the user inputs as their password. The password being encrypted is shown as asterisks for security reasons.
- The “Submit” button checks if the username and password entered match the criteria set by the App and returns a corresponding message on whether the user was able to successfully create an account.
- The “Back” button takes the user back to the Welcome Page of the App if they change their mind.



4.3.4 *MainMenu.js*

- The Main Menu page represents the main page where the user can access the different functionalities of the Geocache App.
- The page will have a “Profile” button, a “Map” button, a “Search” button and a “Sign out” button (represented by ).
- The “Profile” button takes the user to their Profile Page.
- The “Map” button takes the user to the geocache map which shows their current location on the map and nearby geocaches.
- The “Search” button takes the user to the Search Users Page.
- The “Sign out” button signs the user out of the app and takes them back to the Welcome Page.

4.3.5 *ProfilePage.js*

- The Profile Page represents the User Profile page which contains the details of the User (e.g. achievements, rewards, year joined, etc.).
- The page has a “Back” button (represented by ) and a “Edit Profile” button (represented by a .
- The “Edit Profile” button takes the user to the Edit Profile page where the user can change their profile picture.
- The “Back” button takes the user back to the Main Menu page.

4.3.6 *EditProfilePage.js*

- The Edit Profile page is the page where the user can change their account information.
- There is a “New Username” text input field, a “New Password” text input field, a “Choose Profile Picture” button (represented by ) , a “Save” button and a “Back” button (represented by .
- The “New Username” text field allows the user to edit their username on the app.
- The “New Password” text field allows the user to edit their password on the app. The password text is shown as asterisks for privacy and security.
- The “Choose Profile Picture” button opens the users’ camera gallery and allows them to choose a picture for their profile.
- The “Save” button checks if the new username does not already exist on the app using the database. It also checks whether the new password is not the user’s old password and also saves the new profile picture if the user used the “Choose Profile Picture” button. If

successful a notification that changes are saved will pop up on the screen. If not, users will be shown any errors in their new username or password.

- The “Back” button will take the user back to the Main Menu page if they do not want to update their profile.

4.3.7 Map.js

- The Map Page is the page which shows the current location of the user and the nearby geocaches.
- The Map page contains an “Add” button (represented by a +), a “List” button (represented by a ☰) and a “Back” button (represented by ↶).
- The “Add” button takes the user to a Place Geocache Page where the user can enter the type and location of the geocache they want to place.
- The “List” button takes the user to an Inventory Page which shows a list of geocaches the user has found.
- In the Map page, when the user clicks on a geocache on the map, they can see basic details of the geocache (e.g. the geocache ID, name, distance from current user location and size). It will also display a “Navigate” button which allows the user to navigate to the selected geocache.
- The “Back” button takes the user back to the Main Menu Page.

4.3.8 Inventory.js

- The Inventory Page is the page which shows the geocaches that the user currently owns. The user can only place geocaches in their inventory.
- The Inventory page has a list of items, and a “Back” button (represented by ↶).
- The list of items shows all the geocaches the user currently owns and tapping any of them shows extra details about the geocache (e.g description, size of geocache, etc). Some of the geocache details can be edited.
- The “Back” button takes the user back to the Map Page.

4.3.9 PlaceGeocache.js

- The Place Geocache page is the page where the user chooses the geocache they want to place on the map from their inventory.
- The page contains a list of the geocaches in the user’s inventory, a “Place” button and a “Back” button (represented by ↶).
- When the user selects a geocache from their inventory, they are directed back to the Map Page where they can place the geocache.
- The “Place” button confirms the location of the geocache placed on the map by the user and returns a message on whether or not the geocache has been successfully placed on the map.

- The “Back” button takes the user back to the Map Page if the user changes their mind and does not want to place a geocache.

4.3.10 *NavigateToGeocache.js*

- The Navigate To Geocache Page is the page which displays the nearest path to the geocache area that the user has chosen on the map.
- The page contains a map with a line showing directions to the geocache area and a “Back” button (represented by ↩).
- Once the user’s current location overlaps with the area where the geocache is located, the screen switches to the Locate Geocache Page.
- The “Back” button takes the user back to the Map Page, in the case that the user wants to choose another geocache to navigate to.

4.3.11 *LocateGeocache.js*

- The Locate Geocache Page represents the page where the user uses their Camera App to locate the geocache in a certain area.
- The page contains the Camera App of the phone which opens up when the user reaches the location of the geocache, a “Pick” button and a “Back” button (represented by ↩).
- The “Pick” button picks up the geocache once it has been found by the user and puts it in the user’s inventory.
- The “Back” button takes the user back to the Map Page if the user changes their mind.

4.3.12 *SearchUsers.js*

- The Search Users Page is the page the user sees when they click the “Search” button in the main menu. This page displays the names of all usernames closely related to the user they are searching for.
- The page contains a text field, a “Search User” button (represented by 🔍), and a list of results.
- The text field is used for the user to input the name of the users they want to find.
- The “Search User” button uses the data in the text field to search the database for all usernames that are close or similar as the one in the text field and displays the results. If there is nothing in the text field, no results are displayed.

Appendix A

A.1 Most Important Javascript Classes

A.1.1 AuthProvider.js

Description: This class is used to connect the application to the database by authenticating a user. This class includes signup, login, and sign out functions that query the database for authentication, if successful the user is now able to access the database and its other functions.

```
1 import React, { useContext, useState, useEffect, useRef } from "react";
2 import Realm from "realm";
3 import app from "../realmApp";
4
5 const AuthContext = React.createContext(null);
6
7 const AuthProvider = ({ children }) => {
8   const [user, setUser] = useState(app.currentUser);
9   const [username, setUsername] = useState('');
10   //const [userGeocaches, setGeocaches] = useState([]);
11   const realmRef = useRef(null);
12
13   useEffect(() => {
14     if (!user) {
15       return;
16     }
17
18     const config = {
19       sync: {
20         user,
21         partitionValue: `user=${user.id}`,
22       },
23     };
24
25     Realm.open(config).then((userRealm) => {
26       realmRef.current = userRealm;
27       const users = userRealm.objects("Users");
28
29     });
30
31     return () => {
32       const userRealm = realmRef.current;
33       if (userRealm) {
34         userRealm.close();
35         realmRef.current = null;
36         // setGeocaches([]);
37       }
38     };
39   }, [user]);
40
41   const signIn = async (email, password) => {
42     const creds = Realm.Credentials.emailPassword(email, password);
43     const newUser = await app.logIn(creds);
44     setUser(newUser);
45     setUsername(newUser.username);
```

```

48   const signUp = async (email, password) => {
49     await app.emailPasswordAuth.registerUser({ email, password });
50   };
51
52   const signOut = async (email, password) => {
53     if (user == null) {
54       console.warn("Not logged in, can't log out!");
55       return;
56     }
57     user.logout();
58     setUser(null);
59   };
60
61   //const deleteAccount = async
62
63   //const updatePrivacy = async (username, privacy) => {
64   //  realm.write(() => {
65   //    user.privacy = privacy;
66   //  })
67   //  console.log('User')
68   //}
69
70   return (
71     <AuthContext.Provider
72       value={{
73         signUp,
74         signIn,
75         signOut,
76         user,
77         //userGeocaches,
78       }}
79     >
80       {children}
81     </AuthContext.Provider>
82   );
83 };
84
85 const useAuth = () => {
86   const auth = useContext(AuthContext);
87   if (auth == null) {
88     throw new Error("useAuth() called outside of a AuthProvider");
89   }
90   return auth;
91 };
92
93 export { AuthProvider, useAuth };

```

A.1.2 Geocaching.js

Description: This class is the code that manages the main geocaching page of the app. It includes the front end styling of the app, the map data, location permissions and google services logic, and the calls to the database methods for geocaching.

```

1  import React, {useState, useEffect} from 'react';
2  import MapView from 'react-native-maps';
3  import {Marker} from 'react-native-maps';
4  import Geolocation from '@react-native-community/geolocation';
5  import RNLocation from 'react-native-location';
6  import {useNavigation} from '@react-navigation/native';
7  import Realm, {User} from 'realm';
8  import app from '../realmApp';
9  import {useAuth} from '../providers/AuthProvider';
10 import { Alert } from "react-native";
11 import {
12   SafeAreaView,
13   View,
14   Text,
15   StyleSheet,
16   TouchableOpacity,
17   Image,
18   LogBox,
19   PermissionsAndroid,
20 } from 'react-native';
21
22 export default function Geocaching({route}) {
23   LogBox.ignoreAllLogs();
24   const navigation = useNavigation();
25   const {user} = useAuth();
26   const { username } = route.params;
27   const [currentLongitude, setCurrentLongitude] = useState(-66.64159932959872);
28   const [currentLatitude, setCurrentLatitude] = useState(45.94995093187056);
29   const [locationStatus, setLocationStatus] = useState('');
30   const [geocacheList, setGeocacheList] = useState([]);
31   const [geocacheID, setGeocacheID] = useState('');
32   const [geoLong, setGeoLong] = useState(0);
33   const [geoLat, setGeoLat] = useState(0);
34   watchID: number = null;
35   useEffect(() => {
36
37     const requestLocationPermission = async () => {
38       try {
39         const granted = await PermissionsAndroid.request(
40           PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
41           {
42             title: 'Location Access Required',
43             message: 'This App needs to Access your location',
44           },
45         );

```

```

42     title: 'Location Access Required',
43     message: 'This App needs to Access your location',
44   },
45 );
46 if (granted === PermissionsAndroid.RESULTS.GRANTED) {
47   //To Check, If Permission is granted
48   getOneTimeLocation();
49   subscribeLocationLocation();
50 } else {
51   setLocationStatus('Permission Denied');
52 }
53 } catch (err) {
54   console.warn(err);
55 }
56 };
57 const getCoordinates = async () => {
58   try {
59     const geocacheList = await user.functions.getGeocacheCoordinates();
60     setGeocacheList(geocacheList);
61   } catch (err) {
62     console.warn(err);
63   }
64 };
65 requestLocationPermission();
66 getCoordinates();
67 return () => {
68   Geolocation.clearWatch(watchID);
69 };
70 }, []);
71
72 useEffect(() => {
73   const pickupGeocache = async () => {
74     //if(Math.sqrt(Math.pow(currentLatitude-geoLat ,2) +Math.pow(currentLongitude-geoLong,2) ) <= 0.001){
75     try{
76       console.log("picking up geocache");
77       console.log(username);
78       console.log(geocacheID);
79       await user.functions.pickUpGeocache(username, geocacheID);
80       await user.functions.updateGeocache(geocacheID, 0, 0);
81       Alert.alert("Geocache collected!");
82     } catch (err) {
83       console.warn(err);
84     }
85     //}
86     //else{
87     //   Alert.alert("Not in range of geocache");
88     //}

```



```

89     }
90     pickupGeocache();
91 }, [geocacheID]);
92
93
94 const getOneTimeLocation = () => {
95     setLocationStatus('Getting Location ...');
96     Geolocation.getCurrentPosition(
97         position => {
98             setLocationStatus('You are Here');
99             const currentLongitude = position.coords.longitude;
100             const currentLatitude = position.coords.latitude;
101             setCurrentLongitude(currentLongitude);
102             setCurrentLatitude(currentLatitude);
103         },
104         error => {
105             console.log('error');
106             setLocationStatus(error.message);
107         },
108         {enableHighAccuracy: true, timeout: 30000, maximumAge: 2000},
109     );
110     console.log(currentLongitude);
111 };
112
113 const subscribeLocationLocation = () => {
114     watchID = Geolocation.watchPosition(
115         position => {
116             setLocationStatus('You are Here');
117             const currentLongitude = position.coords.longitude;
118             const currentLatitude = position.coords.latitude;
119             setCurrentLongitude(currentLongitude);
120             setCurrentLatitude(currentLatitude);
121         },
122         error => {
123             setLocationStatus(error.message);
124         },
125         {enableHighAccuracy: false, maximumAge: 1000},
126     );
127 };
128
129 return (
130     <View style={styles.body}>
131         <View style={styles.viewStyle}>
132             <TouchableOpacity onPress={() => navigation.navigate('MainMenu', {username: username})}>
133                 <Image
134                     source={require('../components/back.png')}

```

A.2 Database Functions

A.2.1 *pickUpGeocache*

Description: This function was written within mongoDB as the logic to add a geocache to your inventory.

```
1 exports = function(username, geocache_id){
2   const mongodb = context.services.get("mongodb-atlas");
3   const usersCollection = mongodb.db("ARGeocacheData").collection("Users");
4   const query = { "username": username };
5   const update = {
6     "$push": {
7       "geocaches": geocache_id,
8     }
9   };
10  const options = { "upsert": false };
11  usersCollection.updateOne(query, update, options)
12  .then(result => {
13    const { matchedCount, modifiedCount } = result;
14    console.log(matchedCount);
15    if(matchedCount && modifiedCount) {
16      console.log(`Successfully updated the item.`)
17    }
18  })
19  .catch(err => console.error(`Failed to update the item: ${err}`))
20  };
```

A.2.2 *dropGeocache*

Description: This function was written within mongoDB as the logic to remove a geocache from your inventory.

```
1 exports = function(username, geocache_id){
2   const mongodb = context.services.get("mongodb-atlas");
3   const usersCollection = mongodb.db("ARGeocacheData").collection("Users");
4   const query = { "username": username };
5   const update = {
6     "$pull": {
7       "geocaches": geocache_id,
8     }
9   };
10  const options = { "upsert": false };
11  usersCollection.updateOne(query, update, options)
12  .then(result => {
13    const { matchedCount, modifiedCount } = result;
14    if(matchedCount && modifiedCount) {
15      console.log(`Successfully updated the item.`)
16    }
17  })
18  .catch(err => console.error(`Failed to update the item: ${err}`))
19  };
```

A.2.3 updateGeocache

Description: This function was written within mongoDB as the logic to change the coordinates of the geocache after it has been picked up or placed. When picking up, you would pass in null values to this method to remove it from the map, and when placing you would pass in your current location to place the geocache in your current location.

```
1 exports = function(geocacheID, latitude, longitude){
2   const mongodb = context.services.get("mongodb-atlas");
3   const geocacheCollection = mongodb.db("ARGeocacheData").collection("Geocache");
4   const query = { "geocache_id": geocacheID };
5   const update = {
6     "$set": {
7       "coordinate": {"latitude": latitude, "longitude": longitude}
8     }
9   };
10  const options = { "upsert": false };
11  geocacheCollection.updateOne(query, update, options)
12  .then(result => {
13    const { matchedCount, modifiedCount } = result;
14    console.log(matchedCount);
15    if(matchedCount && modifiedCount) {
16      console.log(`Successfully updated the item.`)
17    }
18  })
19  .catch(err => console.error(`Failed to update the item: ${err}`))
20  };
```

A.2.4 getListOfGeocaches

Description: This function was written within mongoDB as the logic to get a list of all of the geocaches currently available on the map, so that they can be displayed on the map.

```
1 exports = function(username){
2   const mongodb = context.services.get("mongodb-atlas");
3   const userCollection = mongodb.db("ARGeocacheData").collection("Users");
4   const query = {username: username};
5   const projection = {"geocaches": 1};
6   return userCollection.find({}, {_id:0, })
7   .toArray()
8   .then(items => {
9     console.log(`List of geocaches: ${items[0]}`)
10    console.log(items)
11    return items
12  })
13  .catch(err => console.error(`failed ${err}`));
14 }
15 }
```

A.2.5 getUsername

Description: This function was written within mongoDB as the logic to retrieve the username of the current player logged in.

```
1 exports = function(id){
2
3   const userCollection = context.services.get("mongodb-atlas").db("ARGeocacheData").collection("Users");
4   return userCollection.find({ "_id.$oid": id }, { username: 1 })
5     .toArray()
6     .then(items => {
7       console.log(items)
8       return items
9     })
10  }
11  .catch(err => console.error(`failed ${err}`));
12
13 };
```

A.2.6 insertUser

Description: This function was written within mongoDB as the logic to create an account.

```
1 exports = function(username, password){
2   const mongodb = context.services.get("mongodb-atlas");
3   const usersCollection = mongodb.db("ARGeocacheData").collection("Users");
4   const newUser = {
5     "username": username,
6     "password": password,
7     "privacy": false,
8     "geocaches": [],
9   }
10  usersCollection.insertOne(newUser)
11    .then(result => console.log(`Successfully inserted item with _id: ${result.insertedId}`))
12    .catch(err => console.error(`Failed to insert item: ${err}`))
13 }
```

Appendix B

B.1 Test Plans

B.1.1 Unit Testing

Before each feature was added to the prototype, it had its own set of test cases to be performed to ensure it was functional. Here is an example test plan for the authentication feature.

Test 1: Register for an account with a unique username “TestUser1” and password of 8 characters “password”

Expected output: ” Account registered successfully, you are now logged in”

Test 2: Register for an account with a taken username “TestUser1” and a password of 8 characters “password”

Expected output: “Username already in use.”

Test 3: Register for an account with a unique username “TestUser2” and a password of less than 8 characters “123”

Expected output: “Password must be at least 8 characters long.”

Test 4: Login to existing account with username “TestUser1” and password “password”

Expected output: “Successfully logged in”

Test 5: Login to existing account with username “TestUser1” and incorrect password “123”

Expected output: “Password Incorrect”

Test 6: Login to account with username not registered for “TestUser3” and password “password”

Expected Output: “There is no account with the given username”

Test 7: Logout of the system.

Expected output: “successfully logged out”

All tests passed. All other unit test plans were simple and successful and therefore not included in the report for brevity.

B.1.2 Integration Testing

After each feature was added to the prototype, we tested each other feature in the app to make sure that no bugs were introduced elsewhere in the code due to the addition of the new feature.

B.1.3 Prototype Testing

This is the test plan written for when the application has all of the intended features implemented:

Task 1: Two users will each create an account and login to the system.

Task 2: The users will search for each other's profile and verify the information is correct.

Task 3: The users will set their accounts to private, and repeat task 2 ensuring they cannot find the other user.

Task 4: The users will enter the map and each search for a geocache, and pick it up.

Task 5: The users will then place their found geocache on the map.

Task 6: The users will attempt to find the other's geocache and pick it up.

Success criteria: The task will be considered successful if no errors or bugs appeared within the system, and the user was able to complete the task.

Appendix C

C.1 User Flow - User Interface Screens