Project Team: Coin-Flippers
Mabelyn Espinoza, Priyanka Chhetri, Ariana Song, Dane Hansen

Github Repository: https://github.com/emabelyn/Scavenger
v1.0

1. Project Definition

Ex-scavenge-anza is a mobile application that allows users to create, manage, and join a scavenger hunt. The idea came to be when deciding on a good and fun way to give back to the community. Ex-scavenge-anza is primarily meant to be used on college campuses during freshmen orientation so that new students can learn their way around their campus. The hunt will be managed by an administrator with a specific username and password. They will have the ability to drop clues in any desired location within a given radius. They will also have the ability to adjust the time and date of the hunt. The students participating in the hunt will be required to download the application, they will then be prompted to enter a code that will link them to the specific scavenger hunt of their campus as well as give them the first clue. They must then follow the clues to the point of interest where they are prompted for a code to access the next code. The project will be broken up into four main components: geolocation, database development, user management and the GUI. Android Studio and Java coding will be utilized to implement the application for Android devices.

2. Project Requirements

2.1 Functional:

A user must sign in with their Google account in order for them to access host abilities. The host will be able to create a new scavenger hunt, manage existing hunts, and schedule a date and time for the hunt to be held. General users must be able to join a scavenger hunt with the correct code. Once participating in a scavenger, they should be able to see points of interest (POI) and their associated clues. At a POI they are able to type in a code. If the code is correct the user unlocks the next set of clues. This process continues until they finish the scavenger.

2.2 Usability:

2.2.1 User Stories

As a...	I want to...	So that...	Acceptance
Host	Create a new scavenger hunt.	I can host a scavenger.	Must be logged in via Google
Host	Edit an existing scavenger hunt.	I can update points of interest, clues.	Must have a previously created scavenger
Host	Schedule a scavenger hunt.	Scavengers can participate in a hunt.	Must have a scavenger made already.
Host	Create a new point of interest.	I can add clues to the scavenger hunt.	Must be in an existing/ new scavenger hunt.
Host	See clue information	I can edit and confirm information	Must tap/click on a created the point of interest.
Participant	Join an existing scavenger hunt.	I can view clues.	Must have an access code from a host
Participant	Enter a clue at a point of interest.	I can get the next clue.	Must have joined the scavenger hunt
Participant	Leave a scavenger hunt.	I can no longer participate.	Must use dropdown menu and confirm intent to leave.

2.2.2 Performance

The mobile app will need to run without major lag to maintain seamless navigation.

2.3 System:

Ex-scavenge-anza will be developed using Android 4.0.3 Ice Cream Sandwich for Android devices because using this version will allow about 99.7% of android users to use the app(Android Developers, 2018). MySQL using Amazon RDS will be used for the database.

Android Developers. (2018). *Distribution dashboard | Android Developers*. [online] Available at: https://developer.android.com/about/dashboards/ [Accessed 27 Sep. 2018].

2.4 Security:

We will be using Google Sign In so that we don't have to manage hosts' password and username. This means that we will not need a lot of security for our data because the participants username will be completely randomized. The only thing that will need to be protected will be the POI's.

3. Project Specification

3.1 Focus / Domain / Area

The targeted audience for version 1.0 will be UNCG SOAR Organizers and incoming freshmen/transfer students. This is to create a platform where SOAR can create a scavenger hunt that will allow students to become more acclimated to the campus layout.

3.2 Libraries / Frameworks / Development Environment

App development will be completed in Android studio. Google Maps API, Leaflet, and OpenStreetMap are being considered for the app. Google Maps API would be the best, however it has a cost associated with extended use.

3.3 Platform/Genre

Scavenger is a geolocation based mobile game and the intended platform for Version 1.0 is Android devices. This is to focus on one coding style and reduce time spent learning a new language.

4. System – Design Perspective

4.1 Identify subsystems – design point of view

4.1.1 Initial Design and Model

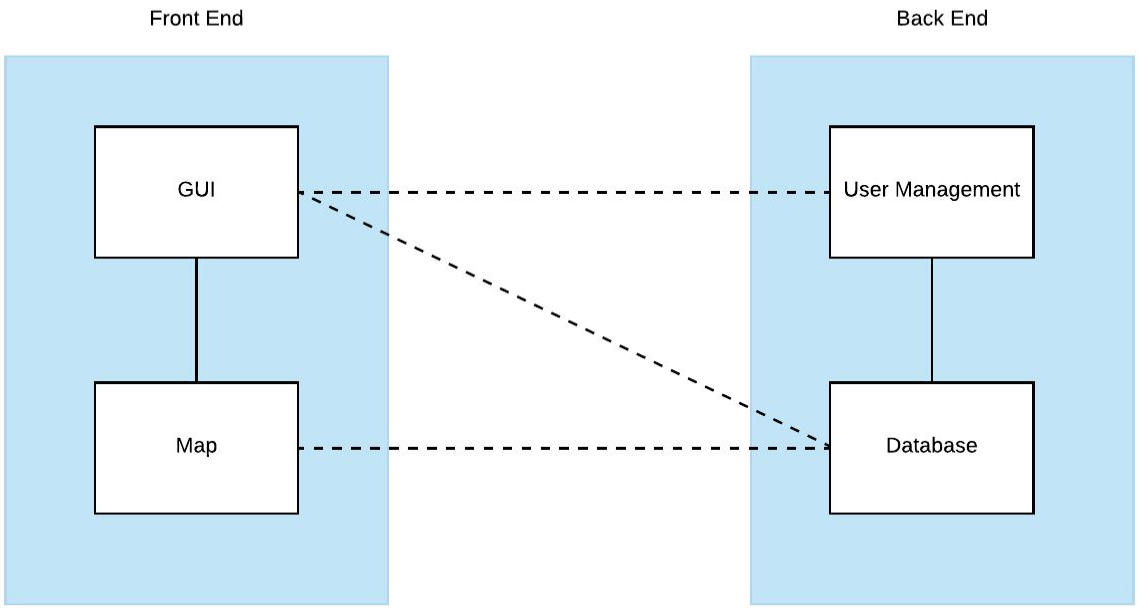


Figure 4.1.1a Subsystem Communication Diagram

The system was divided into four different subsystems: GUI, Map, User Management, and Database. The subsystems can be divided into two categories: front-end and back-end. The GUI and map will be considered as the front-end and user management and database will be the back-end. The GUI needs to interact with the whole subsystem to make everything work efficiently and to also display the map. User management will be divided into two users: Host and Participants. The host will be able to create, edit, and start the hunt. Once the host starts the hunt, the points of interest and clues will be loaded into the database. Then once the user joins a hunt after entering a correct Hunt ID, the data (POI and Clues) will be downloaded from the database into the participants' device. The database will also keep track of the number of participants in the hunt as well as which part of the scavenger hunt the participant is on. Once the host decides to end the hunt, the scavenger is terminated and removed from the database.

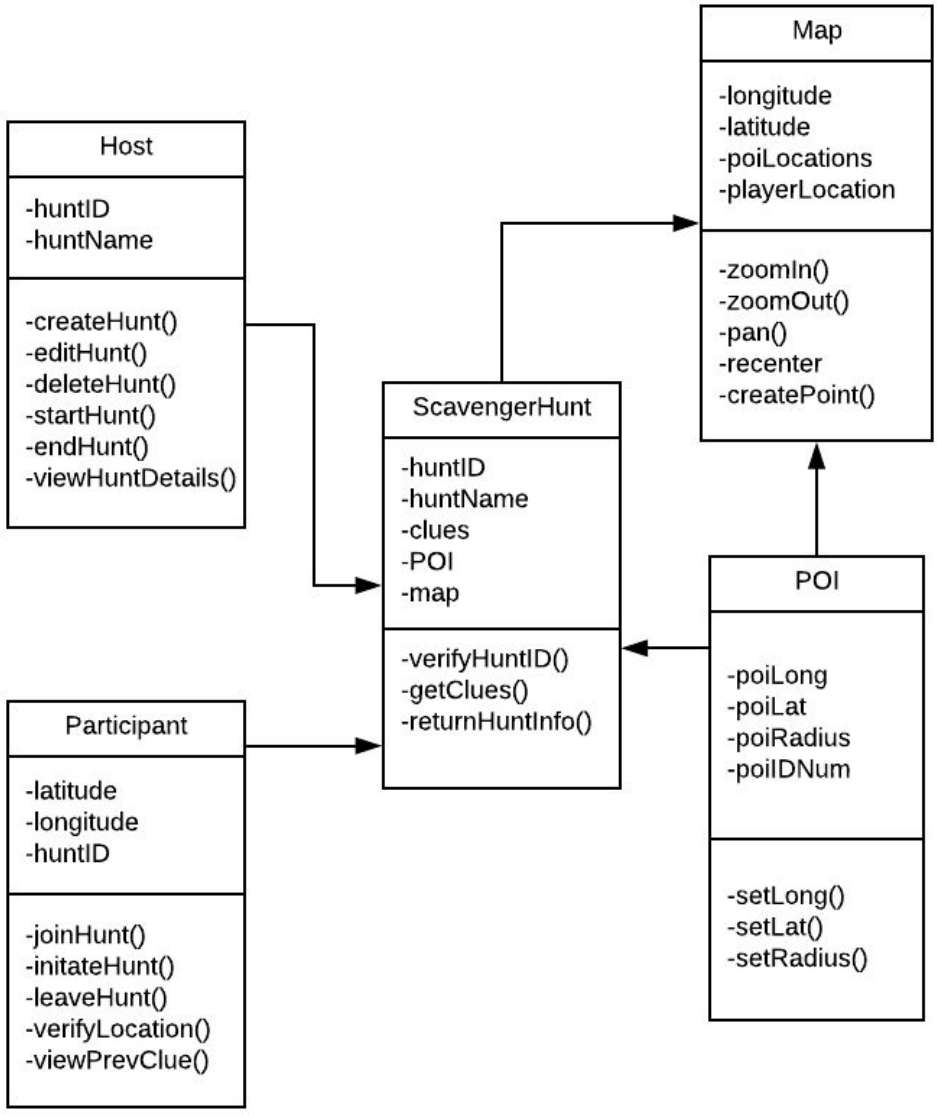


Figure 4.1.1b UML Diagram

The map is displayed through tiles hosted on a public server. Once the mobile device of a participant is able to connect to a hunt with the proper huntID, they are able to load the map along with the POI data. The participant's location is also checked against the database to see if they are within range of the next POI according to their clues.

Hosts are able to access the map upon entering scavenger creation. They have the ability to place markers on the map to create POI’s and edit clues associated with them. POI’s also have an identification number and their respective geographical coordinates linked to them, which are all sent and stored in the cloud database under their associated huntID. Once the host starts the scavenger, participants are able to retrieve this data to begin playing.

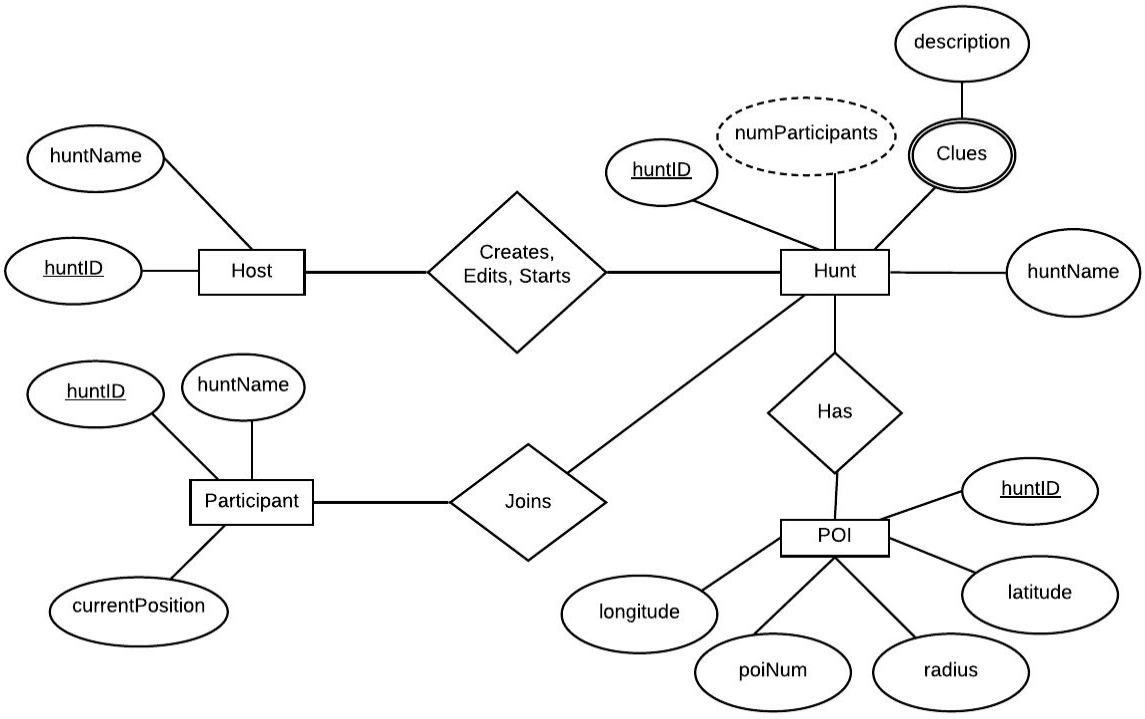


Figure 4.1.1c ER Model

The ER model is composed of entity types and specifies relationships that can exist between instances of those entity types. This ER model shows the entities which are denoted by the rectangular boxes, the attributes of the entity(denoted by the oval), and the relationship between two entities (denoted by the diamond). The relationship between the hunt and host entity is that the host can create, edit, and start the hunt. The relationship between the hunt and participant entity is that the participant joins the hunt, and lastly the relationship between the POI and hunt entity is that the hunt has POI's.

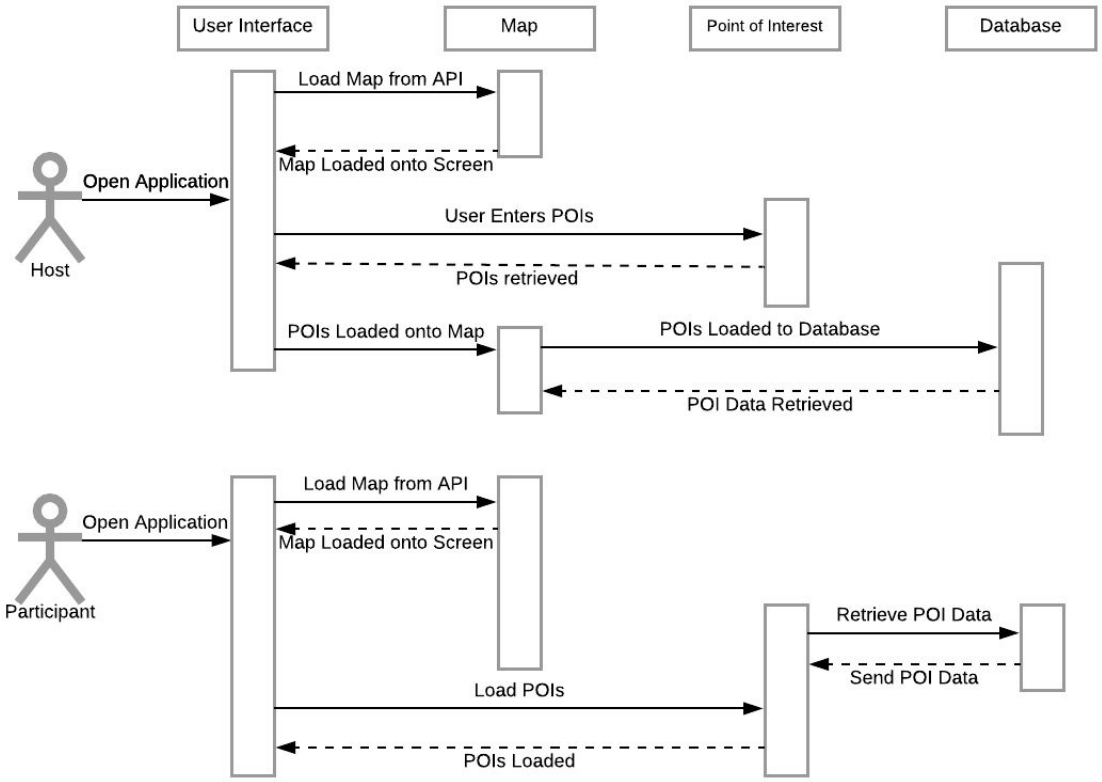


Figure 4.1.1d Sequence diagram

This is the sequence diagram shows object interactions arranged in time. The host and the participant communicates with the UI, and depending on what the user wants the system will sequentially communicate and load whatever was being requested.

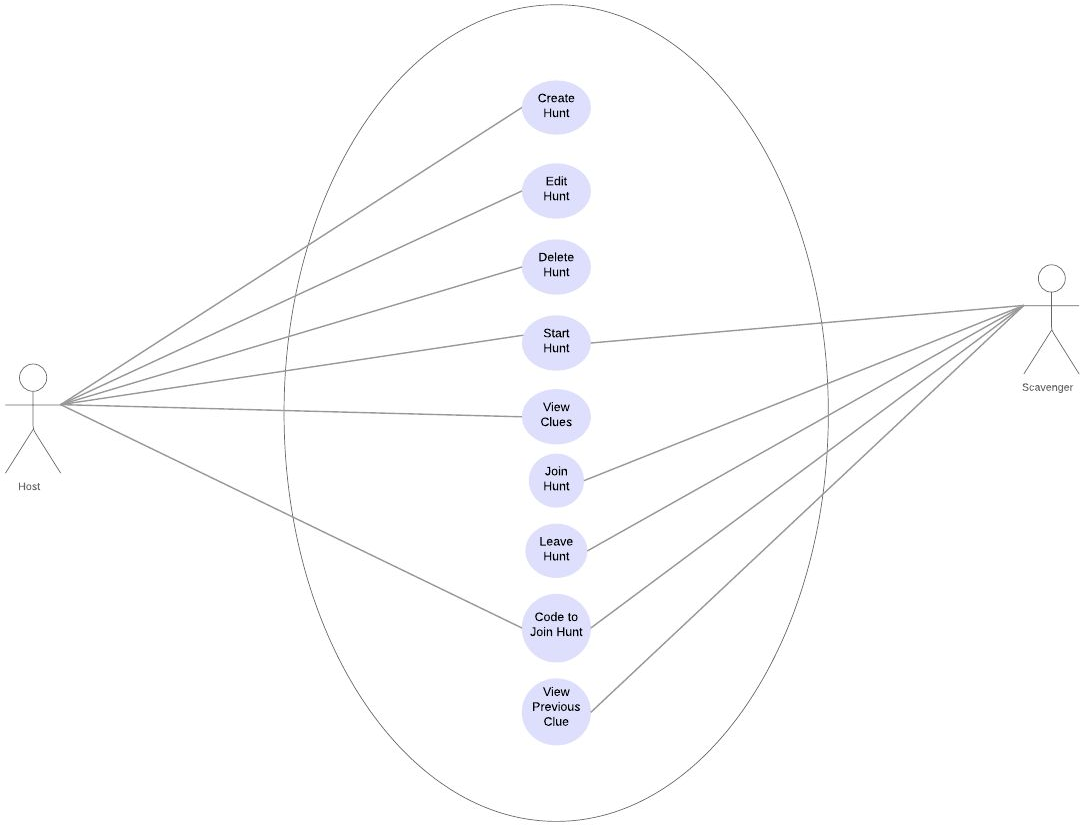


Figure 4.1.1e Use Case Diagram

This use case diagram shows the different actions that the two users (host and participant) can perform on the system.

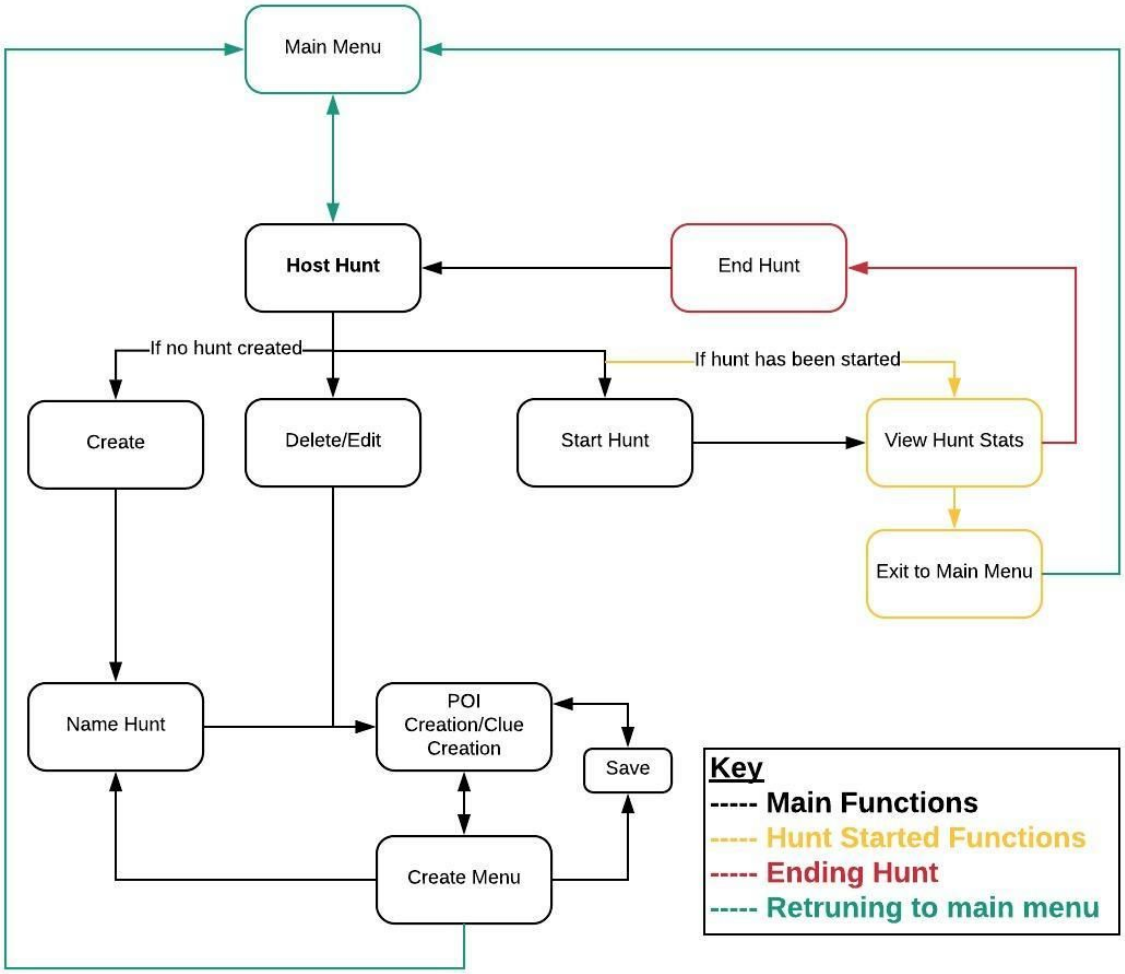


Figure 4.1.1f Flow Chart of the Host

Upon entering scavenger creation, hosts are able to begin creating a hunt, or edit and existing hunt. They have the ability to place markers for POIs and edit clue information. Once saved, they are then able to start the hunt for participants to join. Basic player stats are also available to hosts once the hunt has begun.

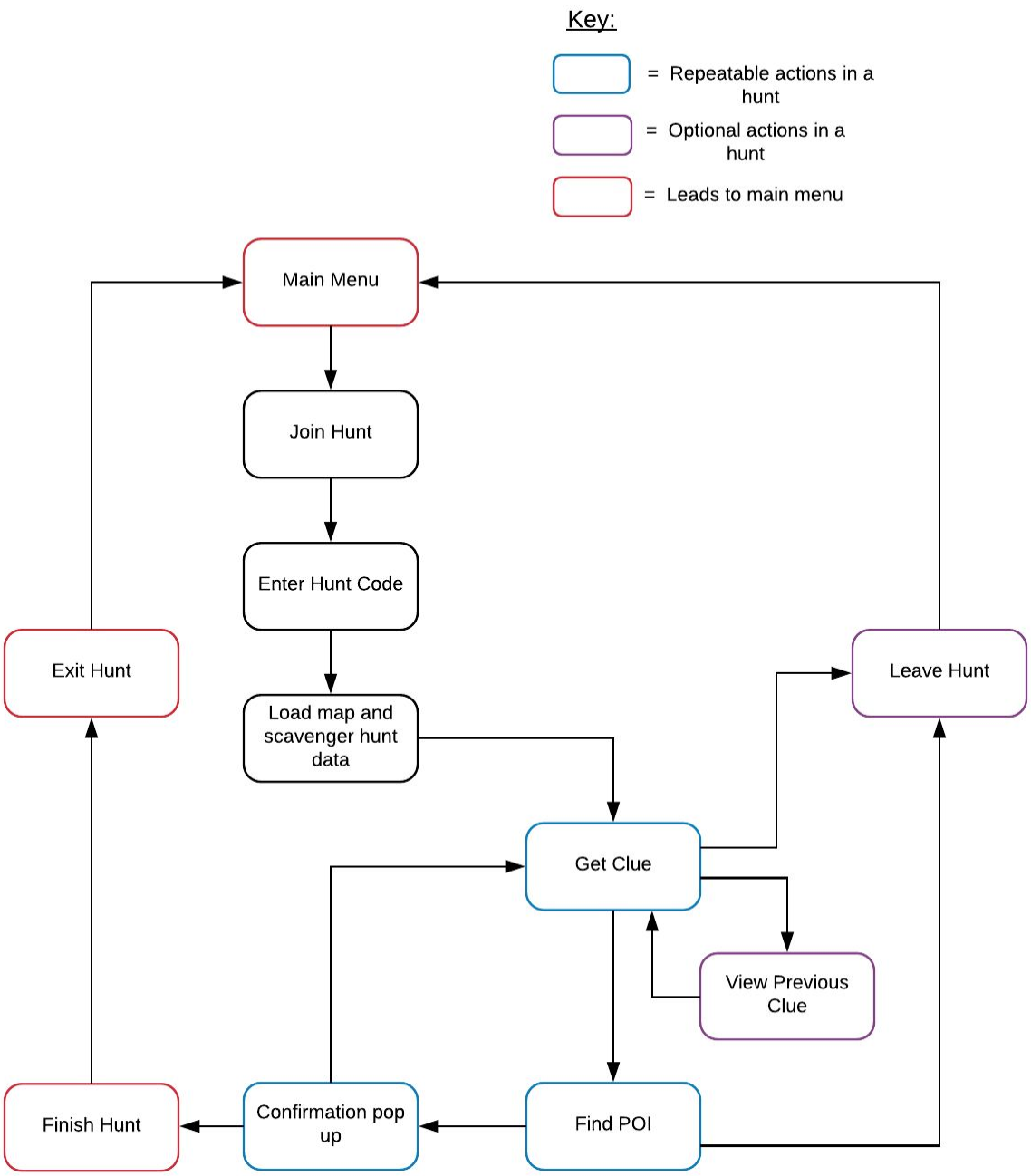


Figure 4.1.1g Flow Chart of the Participant

This is a flow chart that shows how the application will flow from the participant side. From the main menu, the participant can join a hunt and after that they will be prompted to enter a hunt code, and then the map and the data will load from the database. After that they will go to different buildings looking for clues. They can also view previous clues and leave the hunt anytime while they're in the game.

Figure 4.1.1h Design Choice #1: Data Storage in Server vs. Client

Design Decision #1: Should scavenger hunt data be stored on client side or server side?
Description and Alternatives: The system needs to be able to either store data on the client side or the server side. Two strong contenders are:

1. Server Side
2. Client Side

Initial Observations and Comparison: Storing data on the client side can be beneficial to those with poor data coverage since all the POI's would already be downloaded onto their phone, while storing the data on the server side means that the participant would need a reliable connection to be able to use the app.

Decision	Advantages	Disadvantages
Store scavenger hunt data on client side	A. Less communication between device and data; only needs to update user location B. App would run faster since less data is being updated while playing	A. No real time data update B. Files are easily accessible, which could potentially lead to cheating or other misconduct
Store scavenger hunt data server side	A. Data can be updated in real-time so stats can be tracked B. User does not have to download data to their device; saves time and space	A. Participants would need reliable internet connection to keep scavenger hunt updated while playing B. Could cause stress to the server with many players connected at the same time

Decision and Justification: Decided to store the data on the client side. The reason for this was because there would be less communication between the device and the database, and the app would run faster since it's not constantly being updated. Also, everyone should be able to enjoy playing the game so if all the POI's are already downloaded into their devices, there would be less stress on the players, because they wouldn't have to worry about internet connection.

Figure 4.1.1i Design Choice #2: Google Maps API vs. Leaflet

Design Decision #2: Which map service should Scavenger make use of?
Description and Alternatives: An accurate map of the world must be displayed with proper roads and buildings. Two strong contenders are:

1. Google Maps API
2. Leaflet

Initial Observations and Comparison: Google Maps API has the most comprehensive map data between the two, however Leaflet provides all of the essential information Scavenger requires. Both have extensive documentation available online and are mobile friendly, however Leaflet also has wide community support with a plethora of community plugins.

	Advantage	Disadvantage
Google Maps API	● Extremely well suited for work in Android Studio ● Copious amounts of documentation available ● Many useful supplemental resources offered by Google	● Credit card information required upfront ● Payment due after a certain amount of usage
Leaflet	● Completely free ● Large amount of documentation and plugins easily available online	● Additional services will be from third-parties ● Less comprehensive overall

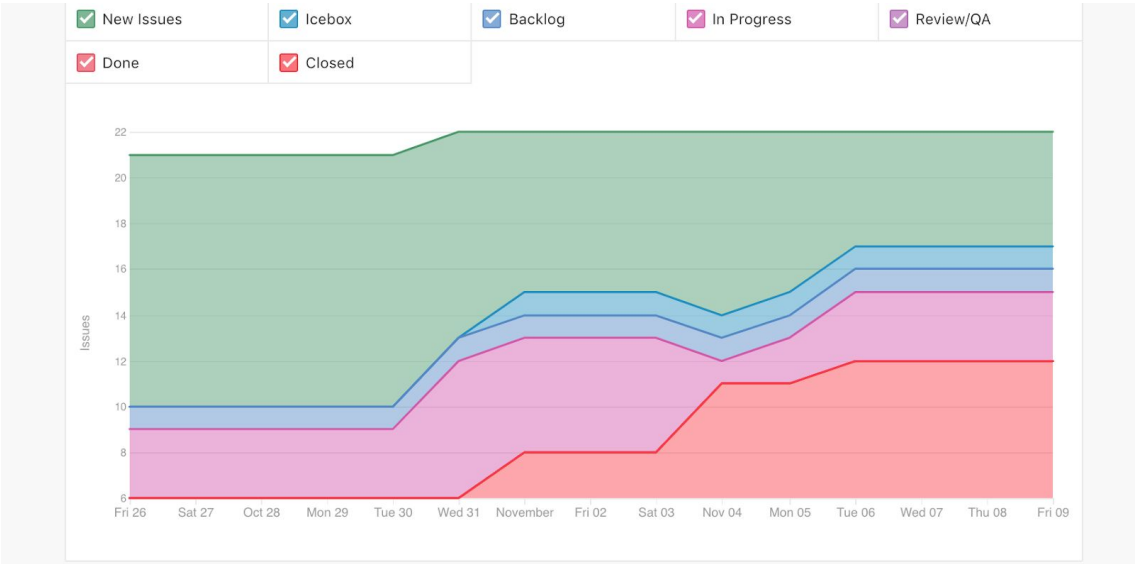
Decision and Justification: Leaflet was chosen as the best option for the application. Google obviously excels in the overall resources and data departments, however the mandatory credit card information and restricted use before being subject to payment was a major factor in deciding against it. Also, it is unlikely the Scavenger application would make use of much of the data and services Google would be able to provide. Leaflet, on top of being free, offers the functionalities Scavenger requires.

6. Project Scrum Report

6.1 Product Backlog (Table / Diagram)

6.1.1 Cumulative Flow

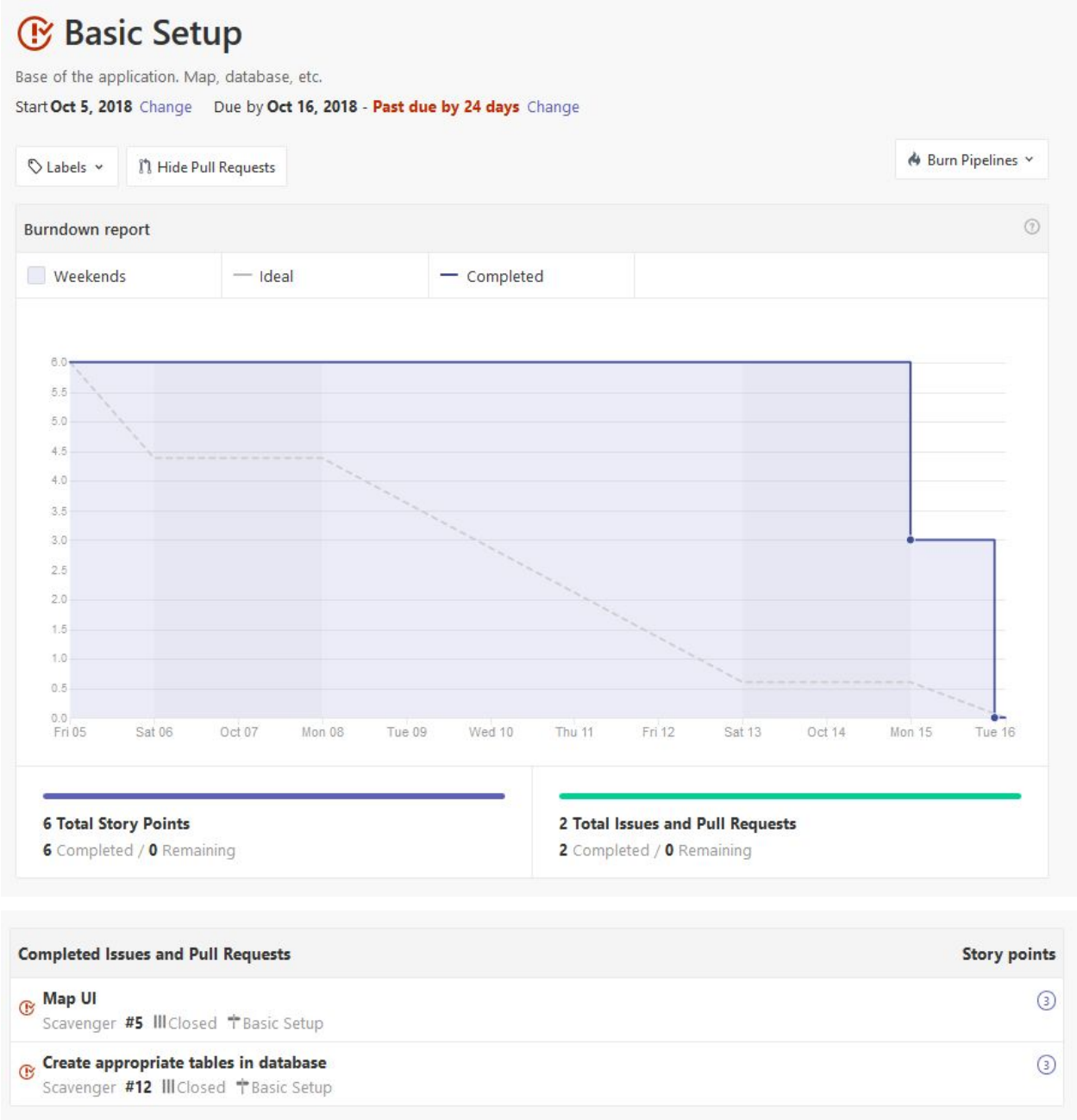
The cumulative flow diagram shows a color coded macro view of all of the issues in the project. The workflow is visible through the issues that are marked in progress and closed, and as other issues are discovered in the development process they are visible in green.



Issues in Pipeline	November 6th	November 7th	November 8th	November 9th
New Issues	5	5	5	5
Icebox	1	1	1	1
Backlog	1	1	1	1
In Progress	3	3	3	3
Review/QA	0	0	0	0
Done	0	0	0	0
Closed	12	12	12	12

6.1.2 Burndown Report

The burndown reports have been organized into four main milestones as listed below. Issues created for the project have been compartmentalized accordingly.



6.1.2a Milestone: Basic Setup

The first milestone was designed as a stepping stone to get into the project. A portion of the backend and frontend were to be implemented so momentum could be built off of those starting points. The milestone had a slow start but the issues associated with it were completed within the due date; the milestone itself was just closed late.

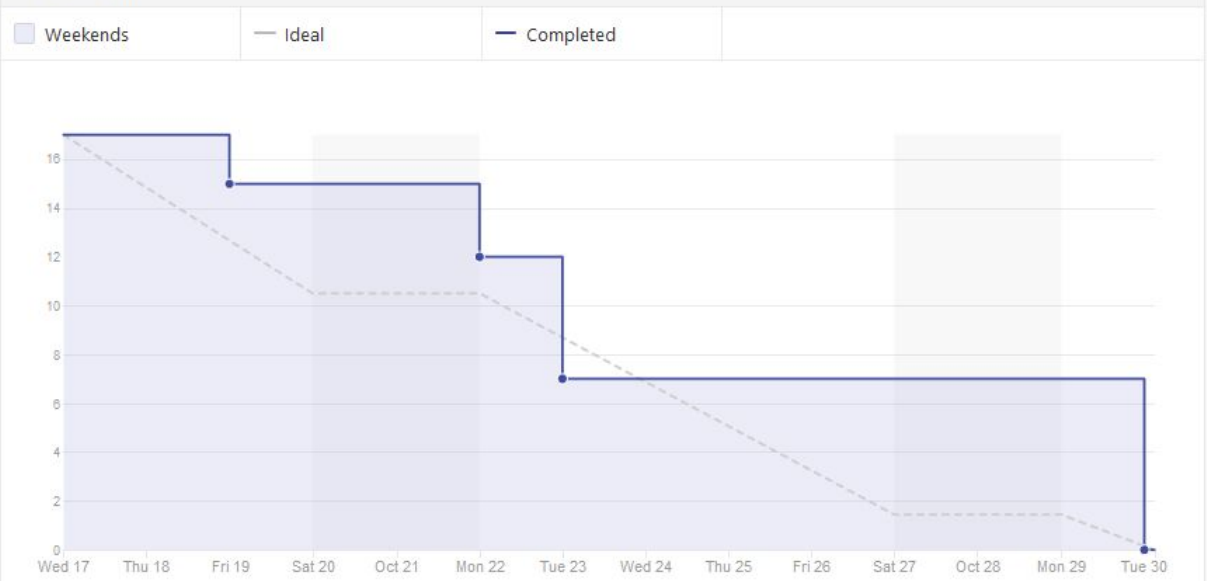
Finetune Database and Map, Base of GUI

Database: solidify tables, establish connection Map: map controls, markers for player/POIs, radius of POIs Set up GUI

Start **Oct 17, 2018** Change Due by **Oct 30, 2018** - **Past due by 10 days** Change

Labels Hide Pull Requests Burn Pipelines

Burndown report



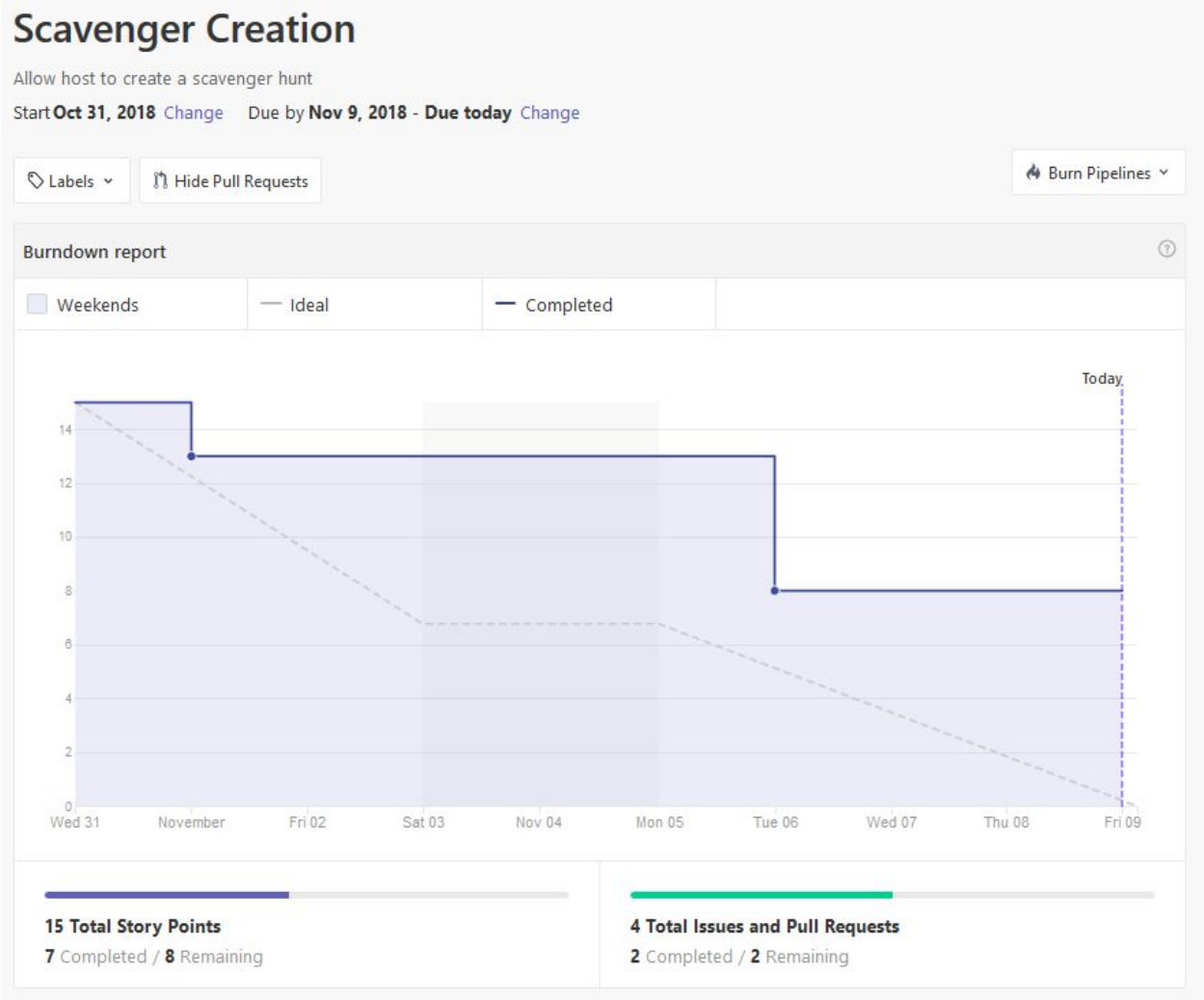
17 Total Story Points
17 Completed / **0** Remaining

7 Total Issues and Pull Requests
7 Completed / **0** Remaining

Completed Issues and Pull Requests	Story points
GUI - Main Screen Scavenger **#1** Closed Finetune Database and Map, Base of GUI	2
GUI - Participant Code Screen Scavenger **#2** Closed Finetune Database and Map, Base of GUI	3
POI Creation & Markers Scavenger **#8** Closed Finetune Database and Map, Base of GUI	3
General Map Controls Scavenger **#11** Closed Finetune Database and Map, Base of GUI	2
Link the Buttons to their respective pages Scavenger **#19** Closed Finetune Database and Map, Base of GUI	2
Add files via upload Scavenger **#20** Closed Finetune Database and Map, Base of GUI	3
Dev Scavenger **#21** Closed Finetune Database and Map, Base of GUI	2

6.1.2b Milestone: Finetune Database and Map, Base of GUI

The focal point of the issues in this milestone was to build off of the foundation created in the first milestone. With the base of the map and database implemented, further steps could be taken in the development of the app. Implementation in this stage included things like more GUI screens and map functionalities. Again, the issues were completed within the allotted time, however the milestone was closed after the due date.



6.1.2c Milestone: Scavenger Creation

Scavenger creation includes being able to communicate between the user's device and the database to be able to save hunt information. This milestone ensures all scavenger information in the creation process is sent to the database and saved, and is currently still in progress.

Play Scavenger

Allow participants to successfully connect to a created Scavenger hunt Database connection Proper data sent to and from devices of host/participants

Start **Nov 10, 2018** Change Due by **Nov 20, 2018** Change

Labels Hide Pull Requests Burn Pipelines

Burndown report

22 Total Story Points
0 Completed / **22** Remaining

7 Total Issues and Pull Requests
0 Completed / **7** Remaining

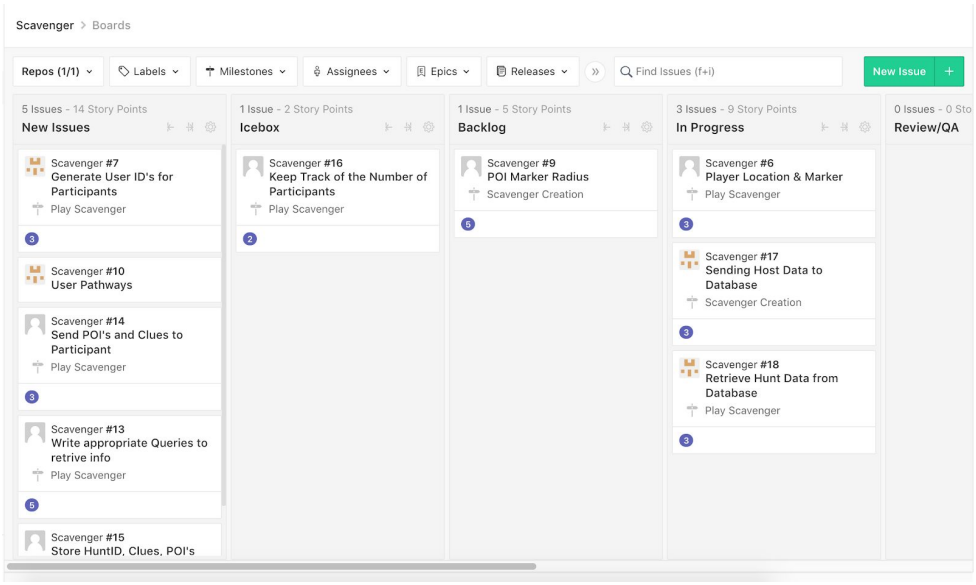
Remaining Issues and Pull Requests	Story points
Player Location & Marker Scavenger **#6** In Progress Play Scavenger	3
Generate User ID's for Participants Scavenger **#7** New Issues Play Scavenger	3
Write appropriate Queries to retrive info Scavenger **#13** New Issues Play Scavenger	5
Send POI's and Clues to Participant Scavenger **#14** New Issues Play Scavenger	3
Store HuntID, Clues, POI's Scavenger **#15** New Issues Play Scavenger	3
Keep Track of the Number of Participants Scavenger **#16** Icebox Play Scavenger	2
Retrieve Hunt Data from Database Scavenger **#18** In Progress Play Scavenger	3

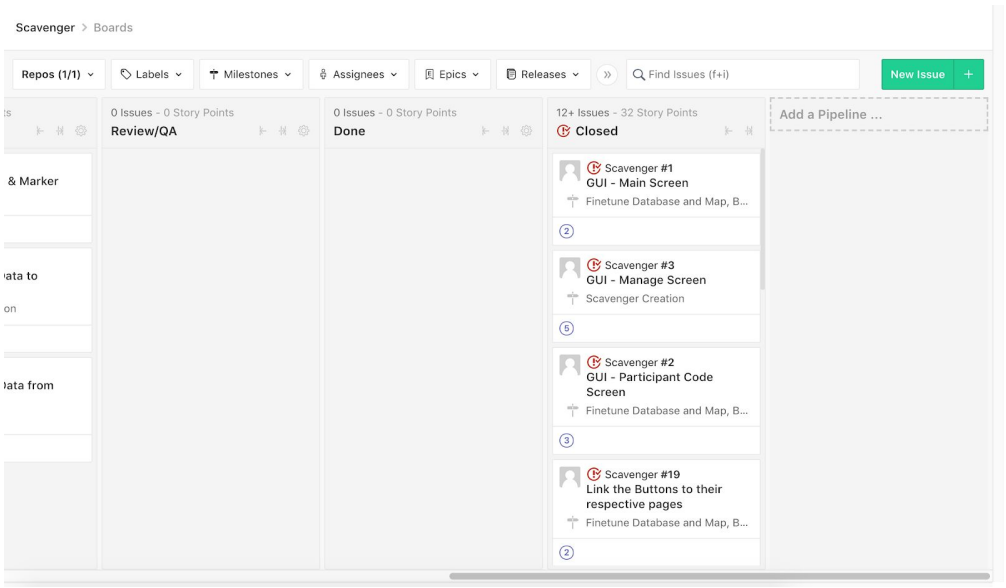
6.1.2d Milestone: Play Scavenger

Issues associated with being able to join and participate in a scavenger hunt are included in this last milestone. Connection between the database and the participants would be established so that the game can run smoothly. This is the next and final step of the project.

6.2 Sprint Backlog (Table / Diagram)

The board displays the created issues and their current status.





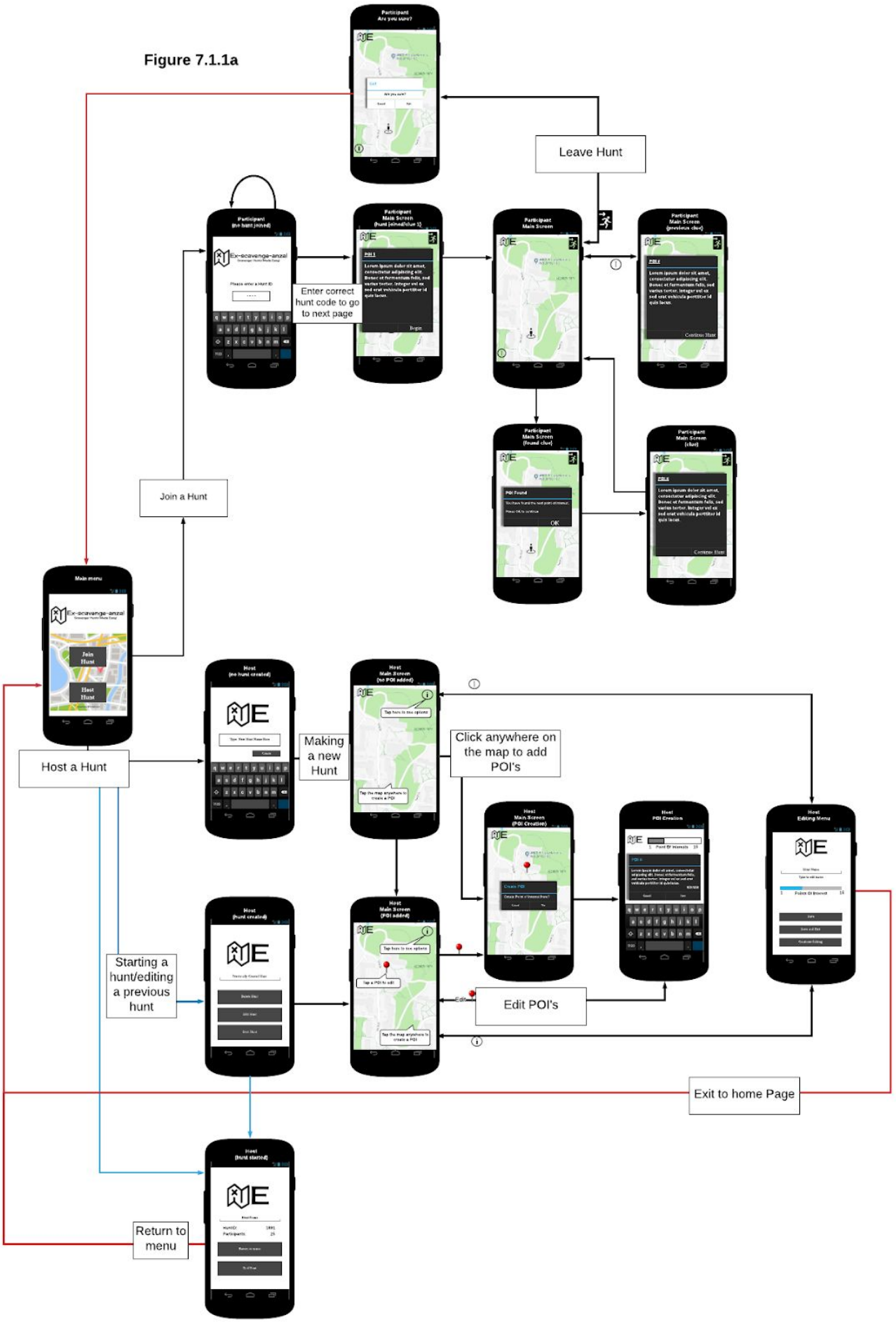
7. Subsystems

7.1 Graphical User Interface – Priyanka Chhetri

This first portion of this subsystem was done by Dane Hansen but due to scheduling conflicts the second portion of this section will be completed by Priyanka Chhetri.

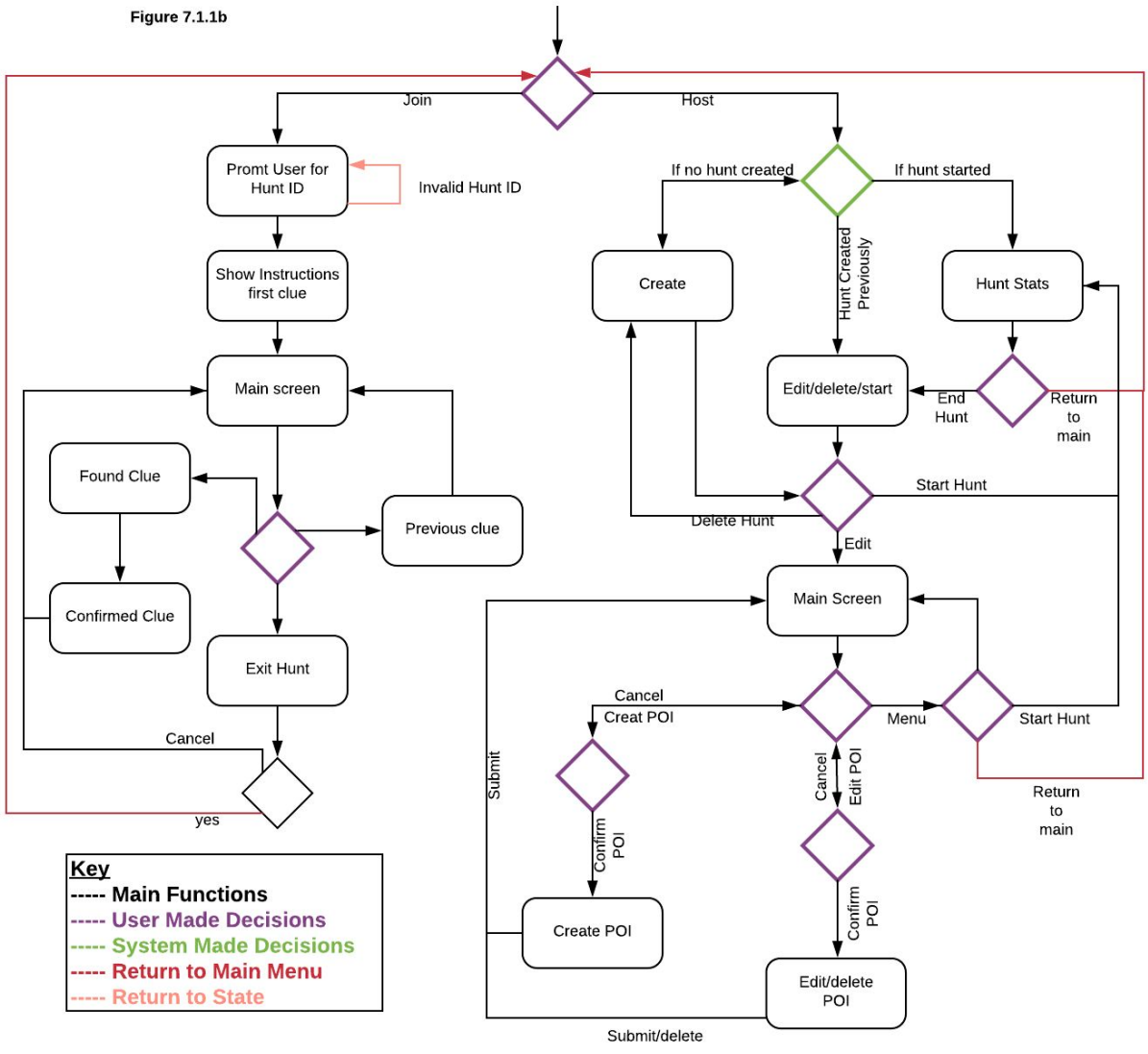
7.1.1 Initial Design and Model

Figure 7.1.1a


Participant
Are you sure?
Leave Hunt
Participant
(no hunt joined)
Ex-scavenge-anzai
Please enter a Hunt ID
Enter correct hunt code to go to next page
Participant
Main Screen
(hunt joined/clue 1)
POI 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec et fermentum felis, sed varius tortor. Integer vel ex sed erat vehicula porttitor id quis lacus.
Begin
Participant
Main Screen
Participant
Main Screen
(previous clue)
Continue Hunt
Participant
Main Screen
(found clue)
POI Found
You have found the next point of interest.
Press OK to continue
OK
Participant
Main Screen
(clue)
Continue Hunt
Join a Hunt
Main menu
Ex-scavenge-anzai
Join Hunt
Host Hunt
Host a Hunt
Host
(no hunt created)
Making a new Hunt
Host
Main Screen
(no POI added)
Tap here to see options
Tap the map anywhere to create a POI
Click anywhere on the map to add POI's
Host
Main Screen
(POI Creation)
Create POI
Create Point of Interest Here?
Host
POI Creation
Point Of Interest
POI A
Host
Editing Menu
Host
(hunt created)
Starting a hunt/editing a previous hunt
Host
Main Screen
(POI added)
Tap here to see options
Tap a POI to edit
Tap the map anywhere to create a POI
Edit
Edit POI's
Exit to home Page
Host
(hunt started)
Return to menu

7.2.1a Interface

This is roughly what the GUI will look like. This diagram shows the flow of the app and the different screens. Image Link: https://www.lucidchart.com/documents/edit/f8dec823-1e84-4d64-a930-118b81e7e42c/1?callback=close&name=docs&callback_type=back&v=20359&s=612



7.2.1b GUI State Diagram

This is a state diagram of the GUI. It shows the different decisions one can make when they are on different screens.

7.1.2 Data Dictionary

Term	Definition
Graphical User Interface (GUI)	A touch screen display that is used to trigger functions that control the app.
display_touch	The location that the user has touched the screen.
display_pinch	The location that the user has pinched the screen.
display_pinchvalue	The touch location minus the pinch location.

7.1.3 Refinements

As of now, nothing has been changed for the GUI.

7.1.4 Scrum Backlog

See section 6.

7.1.5 Coding

7.1.5a Approach/ Language

The functional approach to the UI has been Object Oriented Programming. The code for the GUI has been implemented in Java using Android Studio. Each “activity/class” represents a different screen in the GUI and those activities are linked by different buttons. Some of the activities are Homescreen.java, MainActivity.java, JoinHunt.java, and ParticipantMap.java.

7.1.5b Code

```
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;

public class HomeScreen extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home_screen);
    }

    public void joinHunt (View view){
        startActivity(new Intent(getApplicationContext(), JoinHunt.class));
    }

    public void createHunt(View view) {
        startActivity(new Intent(getApplicationContext(), MainActivity.class));
    }

}
```

This section of code links the activities together so that when you click on the button, it goes to their respective pages. There were some complications with this part because the application kept crashing but it was fixed eventually.

7.1.6 Testing

7.1.6a Introduction

Some of the testing that will be required for GUI are resolution testing, error checking, and correct showing of screen. Doing these tests will make sure that the app runs smoothly and effectively.

7.1.6b Items and Features Tested

User	**Feature to be Tested**	**Passing Criteria**
Host	Create Hunt Button	Loads the CreateHunt Page
User	Resolution	Resolution should be the same on all devices.
Participant	Join a Hunt Button	Loads the JoinHunt Page

Participant	Entering a Correct HuntID	Goes to the Participant Page
Participant	Entering an Incorrect HuntID	Shows an Error Popup message

7.1.6c Approach

The testing for the UI will be done manually.Firstly, the resolution has to be checked on different phones to make sure that the images show up the same on all devices. Error checking has to be done to make sure that the correct pop up messages show up once someone clicks something, also if one enters an incorrect huntID there should be an error message. Selecting the right button should also take the user to the right screen so that will also have to be tested.

7.1.6d Testing Procedure

Host: For testing of the host, the user must be able to click the create a hunt page, where they will be taken to the HostMap page. Once there the user should be able to add POI's/delete POI's/edit clues. After that the user should be able to save the hunt or submit the hunt and after they submit their hunt the information should be sent to the database.
Participant: For testing of the participant, the user will be sent to the huntCode page where they will first enter an incorrect huntID, which will show an error popup message. Then the user will enter a correct huntID (of a mock scavenger). After that they will hit the submit button which should take them to the participantMap page and they are in the hunt. They will go around and find POI's (which cannot be seen by the user). If they are within the range of the POI, a message will pop up saying that a POI has been found. While inside the hunt the user has a chance to exit the hunt, which will take them to the home screen. After the hunt is finished they can exit ad be taken to the home screen.

7.2.6e Initial Results

User: To be completed.
Host: The features of the host outlined in 7.2.6b functioned properly and it did meet passing criteria.
Participant: The join a hunt feature of the participant outlined in 7.2.6b functioned properly and it did meet passing criteria. The entering of incorrect/correct huntID still has to be tested.

7.2 Mapping – Ariana Song

7.2.1 Initial Design and Model

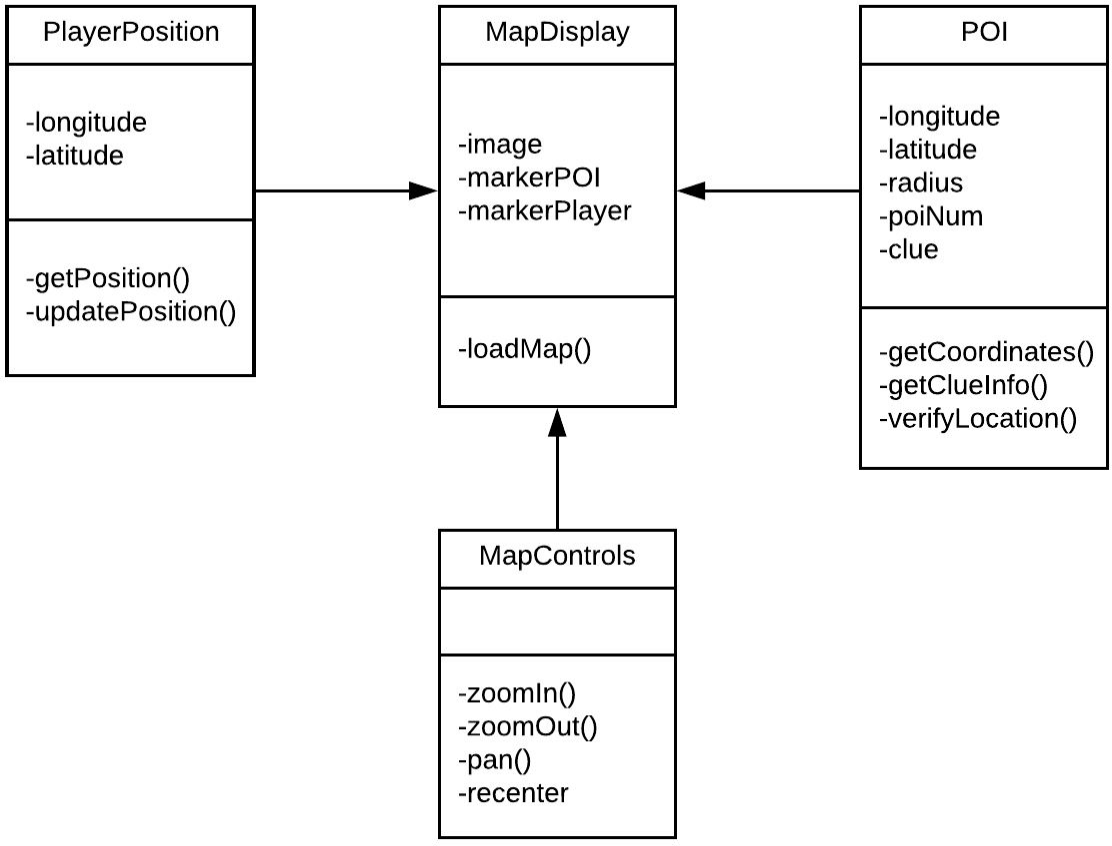


Figure 7.2.1a Map UML Diagram

Displaying the map with the proper controls is the key component. Once the map has been initialized and loaded from the public server online and the proper map tile has been layered, the player position and points of interest can be set on the map; both consist of longitude and latitude coordinates. Along with coordinates, points of interest are to be associated with the corresponding clue information, point of interest marker number, and radius. From the participant map, points of interest are not visible until they have followed the clues to the proper location.

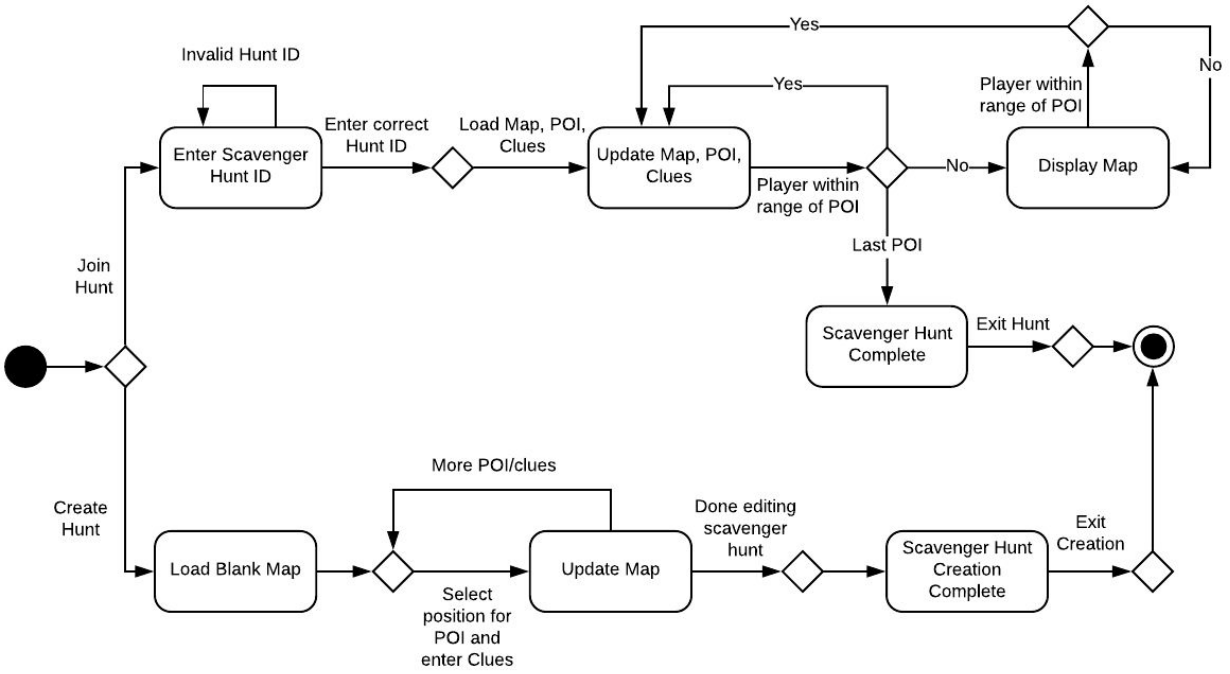


Figure 7.2.1b Map State Diagram

The state diagram shows the progression of events from the hunt creation and participation aspects. Once the map is loaded, there is a steady stream of communication between the device and database to ensure the proper information is being sent back and forth. Participants would be receiving POI information while sending their current location to check against the POI coordinates. Hosts would be sending coordinates and clues associated with POIs to save in the database under the proper huntID.

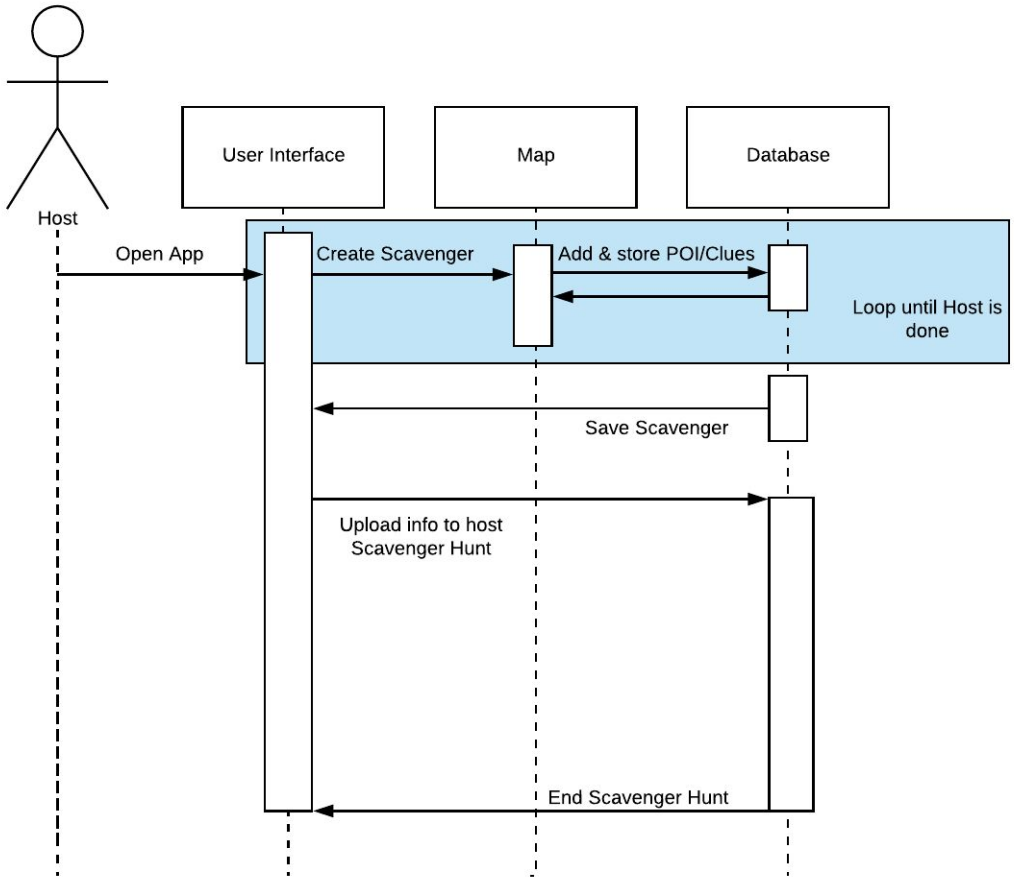


Figure 7.2.1c Map Sequence Diagram for Host

Once in scavenger creation, the system loops between the map and the database. POI and clue information are sent to the database to be saved under the associated huntID. Once the host is done creating the scavenger, the database will have received and stored all relevant information for the hunt, and the host can start the scavenger.

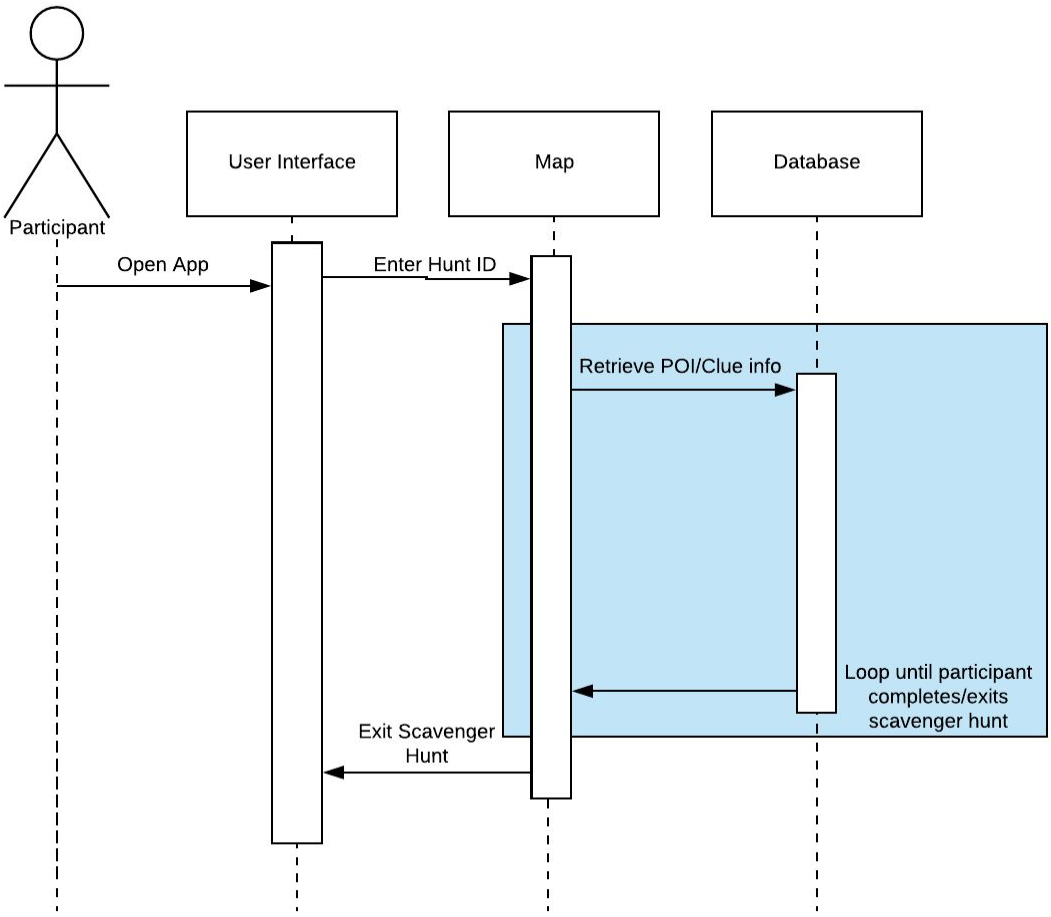


Figure 7.2.1d Map Sequence Diagram for Participant

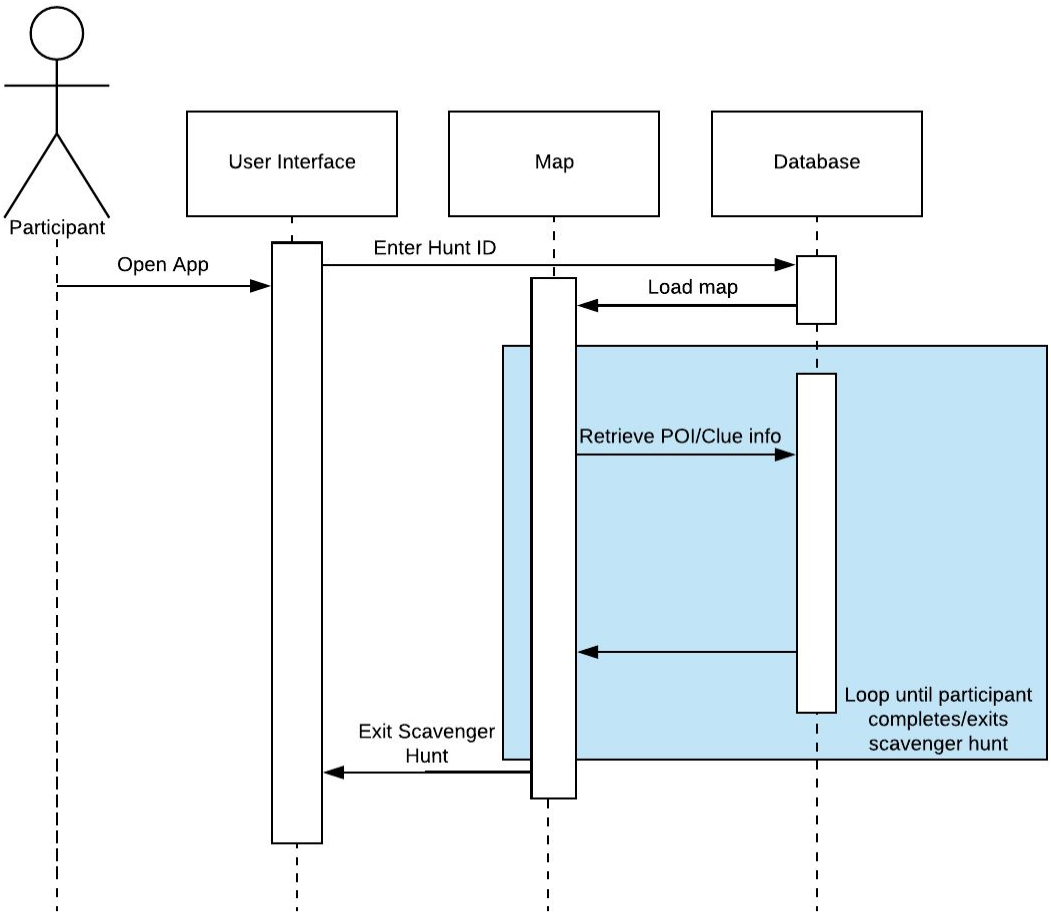
To participate in a scavenger the user will be prompted to enter a huntID. Once verified, the database is able to retrieve the associated data. The participants location is then matched against the coordinates of the POIs connected to that hunt and information is sent back and forth between the participant's device and database until the hunt is complete, or the participant leaves the scavenger hunt.

7.2.2 Data Dictionary

Term	**Definition**
Host	User that has created a scavenger hunt and is hosting it publicly for others to participate in
huntID	ID number created by the host of a scavenger so that other

	players can join the hunt
huntName	The name of the hunt.
Hunt/Scavenger Hunt	Location based game in which players follow clues to a certain destination (POI)
Latitude	Angular distance expressed in degrees and minutes measuring north/south
Longitude	Angular distance expressed in degrees and minutes measuring east/west
Participant/Player	User that has entered a hunt ID to participate in an existing scavenger hunt
Point of Interest (POI)	A specific location that the host has selected as a spot for participants to travel to
POI marker number	Number to count and identify POIs
Radius	Distance from the POI that participants are able to confirm finding the POI

7.2.3 Refinements



7.2.3a Refined Participant Sequence Diagram

Minor refinement made to reflect communication between the user's device and the database to confirm whether the inputted huntID matches up with a hunt stored in the database. Prior to making this revision, entering a huntID went straight to loading a map. Error checking against the stored information in the database did not occur in the original diagram. This ensures the participant is only able to see the map and play the game after a proper huntID is entered.

7.2.4 Scrum Backlog

See section 6.

7.2.5 Coding

7.2.5a Approach, Language

The map subsystem has been implemented primarily in html and javascript. The main portion has been organized into three files, one html file with the javascript in the same file for the host map (creation.html) and the separate html and javascript files participant map (participant.html and participantScript.js). Additionally, there is a short css and javascript file containing code for some of the map control functionality (leaflet.zoomhome.css and leaflet.zoomhome.min.js).

7.2.5b Code

```
function addPOI(e)
{
      var id;
      if(markers.length<1){
            id=0;
      }else{
            id=markers[markers.length-1]._id+1;
      }
      var markerPOI = new L.marker(e.latlng,
{draggable:false}).addTo(map);
      markerPOI._id=id;
      markerPOI.bindPopup('lat, lng at this POI: ' +
                  e.latlng.toString() + '</br>' +
                  '<poi clue ssdfkjlfdskl </br>' +
                  '<button onclick="inputClue"> Edit Clues</button> '
+
                  '<button onclick="removePOI(' + id +
                  ')"> Remove POI</button></br>').openPopup();
      map.addLayer(markerPOI);
      markers.push(markerPOI);
}
function removePOI(id){
      markers.forEach(function(marker){
      if(marker._id==id){
            map.removeLayer(marker);
            map.removeLayer(marker.circle);
      }
      })
}
```

This portion of code shows two functions in the creationScript.js file; one to add a POI and one to remove a POI. Hosts must be able to edit POIs. While the ability to drag the markers that indicate a POIs location exists, the coordinates of the marker do not update after its initial placement.

To combat this issue, the markers have been made undraggable and there is a button that will remove the POI from the map so that it can be replaced. In order for this to work, a separate function was created to handle removing POIs, and POIs had to have some identification to feed into the removePOI function. In the addPOI function, the if...else loop was implemented to assign each POI a number as the host places it. This would then allow specific POIs to be removed even if there are multiple POIs on the map.

7.2.6 Testing

7.2.6a Introduction

The testing portion of the mapping subsystem should uncover any issues that may affect the final product before it is finalized. Testing will include going through usage of the map that users are expected to perform to ensure flawless performance when the project is complete.

7.2.6b Items and Features Tested

User	**Feature to be Tested**	**Passing Criteria**
Host	Time to render map	Map must render within 5 seconds
Host	Map controls (panning, zooming in/out, recentering position)	All map controls must function with minimal lag or delay
Host	Adding and removing POIs	POI markers must be visible upon adding and removed completely upon removal
Participant	Time to render the map	Map must render within 5 seconds
Participant	Map controls (panning, zooming in/out, recentering position)	All map controls must function with minimal lag or delay
Participant	Real-time position updates (geolocation)	Player marker must update in near real-time

Participant	Proper visibility settings for POI markers as POIs are found	POI markers must remain invisible until located with the proper clues

7.2.6c Approach

Testing will be completed manually. An Android device or emulator will be used to run the application as a host and participant to analyze functionalities for each map. Scenarios will be created in which the evaluator will pose as a host and participant in two separate trials.

7.2.6d Testing Procedure

Host: The tester must go through the process of creating a scavenger with multiple POIs. POIs must be deleted and readded. All map controls must be evaluated.

Participant: A mock scavenger to join will have been created. They will join the artificial scavenger and run through the clues to find POIs. This can be done either in person or using an emulator with the ability to spoof location. Clues must be followed to within range of the corresponding POI so that the tester is prompted to confirm locating the POI. POI markers must remain invisible to the participant until they have been located. Again, all map controls must be assessed.

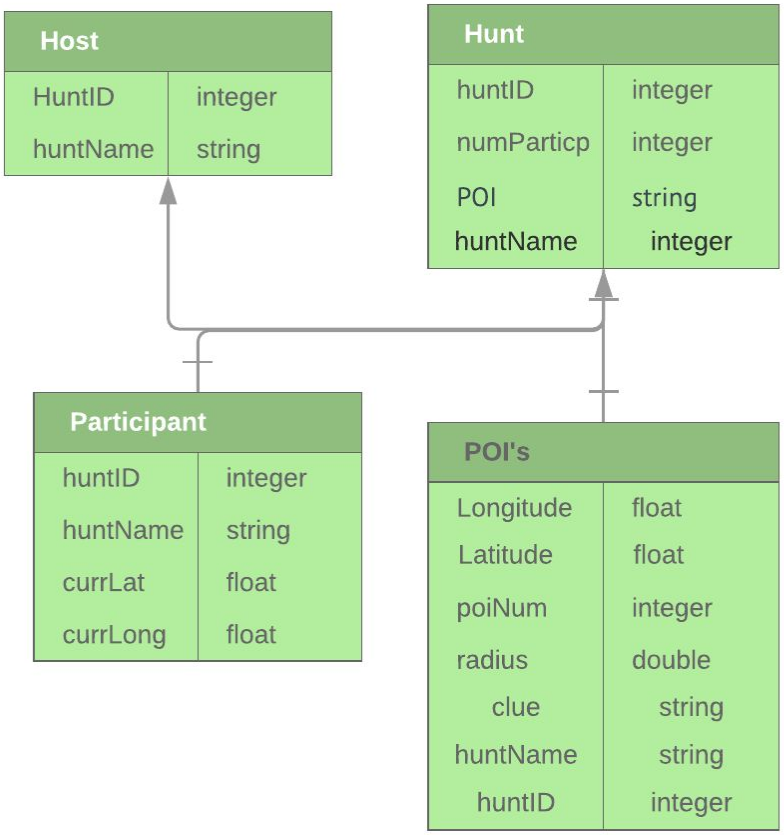
7.2.6e Initial Results:

Host: All host items and features as outlined in 7.2.6b functioned as expected and were able to meet the passing criteria.
Participant: To be completed.

7.3 Database – Mabelyn Espinoza

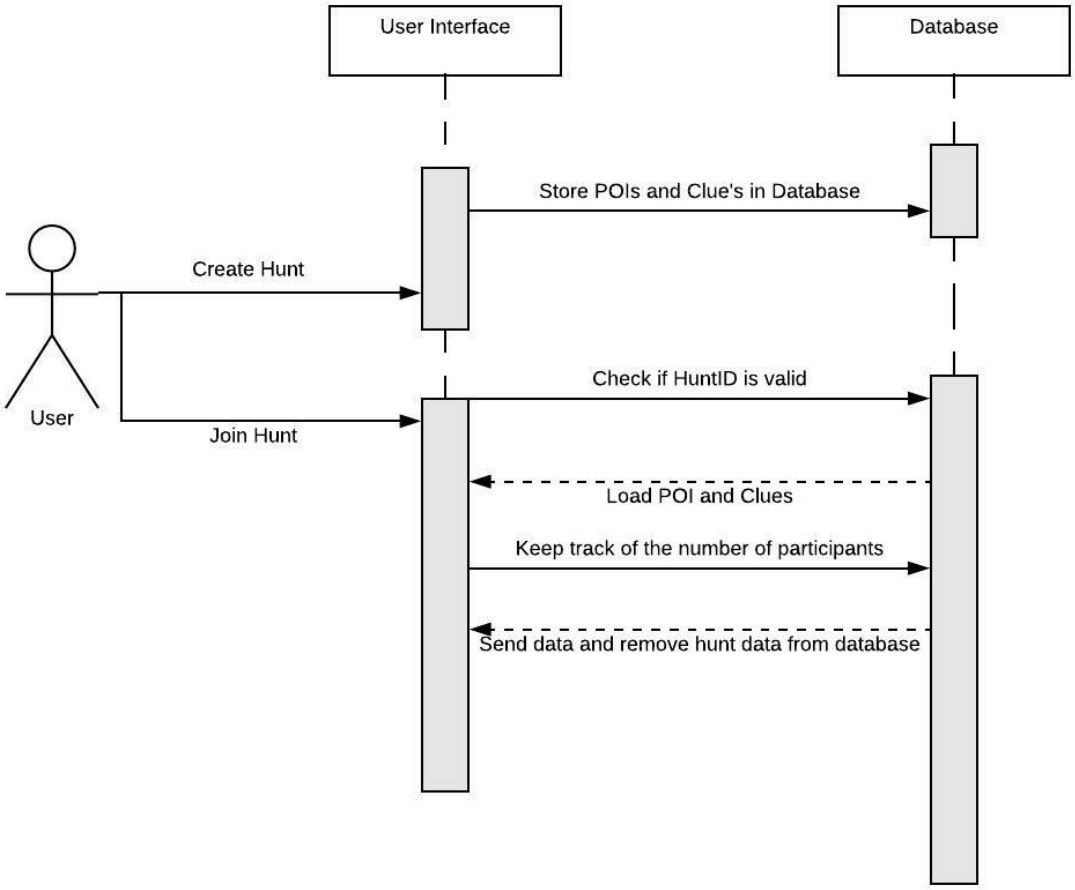
This first portion of this subsystem was completed by Priyanka Chhetri, but due to group scheduling conflicts it became easier for Dane and I to work together on the backend of the project and Priyanka and Ariana to work on the frontend together. I will be completing the second portion of the database subsystem.

7.3.1 Initial Design and Model



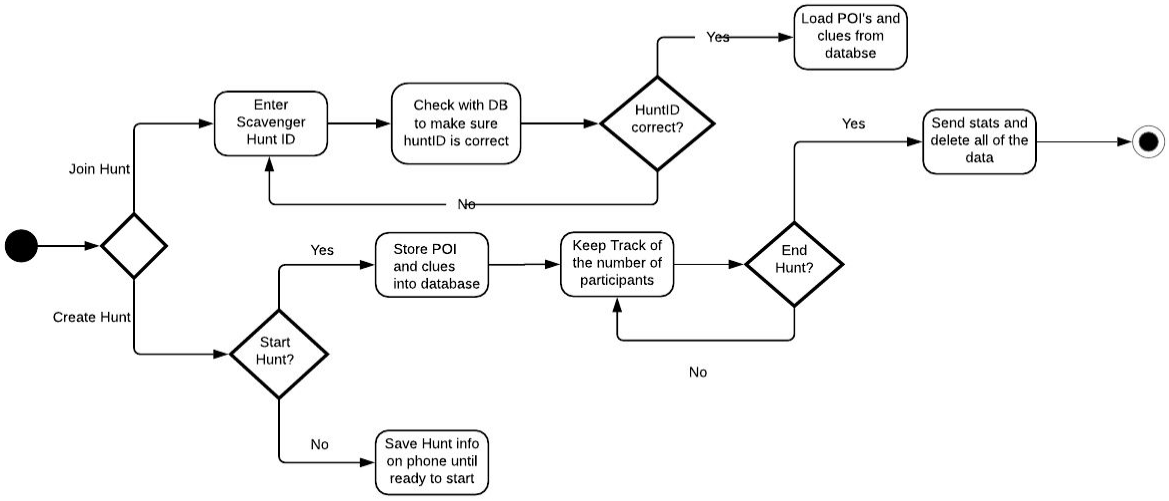
7.3.1a Database UML Diagram

Figure 7.3.1a represents the UML Diagram for the Database. The items that need storing in the database were divided into four main components: Host, Hunt, Participant, and POI.



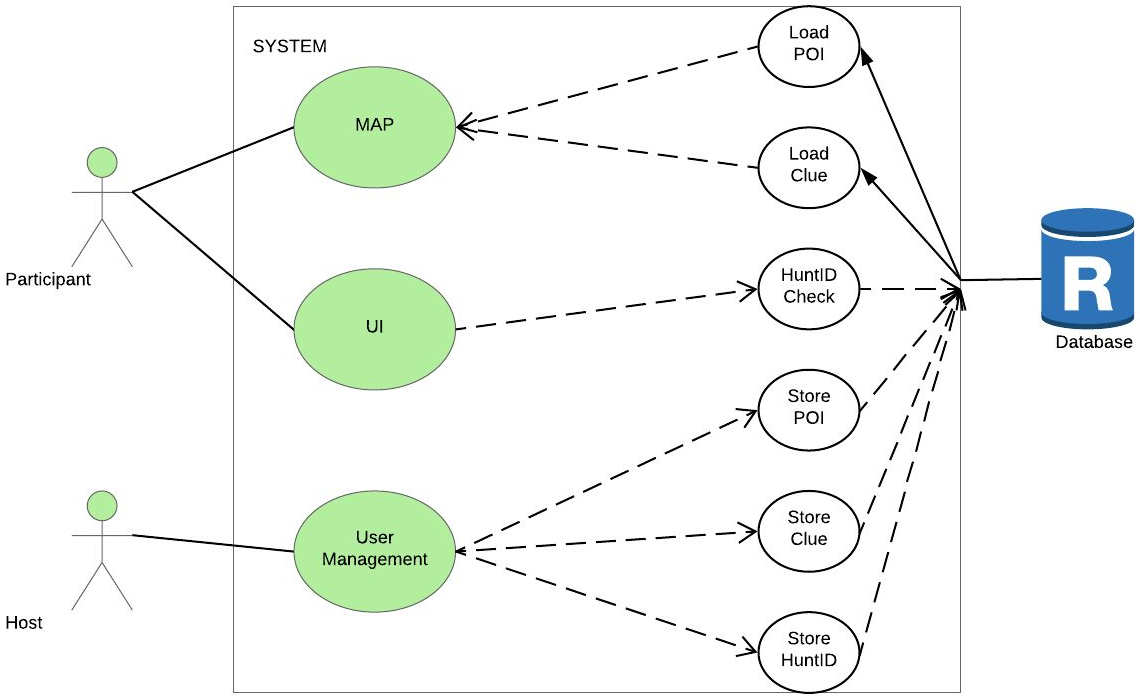
7.3.1b Database Sequence Diagram

Figure 7.3.1b demonstrates the Database's Sequence Diagram. The database comes into play once the user of application decides to save a hunt or start a hunt. A huntID will be randomly generated that connects to the data sent by the host into the database. The participant will then be required to insert the huntID into the textbox on the application in order to start receiving the hunt clues on their device.



7.3.1c Database State Diagram

The state diagram in Figure 7.3.1c shows the user pathways for either a host or participant and how the database interacts with them in the application.



7.3.1d Database Use Case

Figure 7.3.1d shows how the database connects to the host and participants.

7.3.2 Data Dictionary

Term	Definition
Database	Where the data will be stored. Collection of information that is organized so that it can be easily accessed, managed and updated. Using MySQL.
Firebase	Google's mobile platform used to develop mobile applications.
MySQL	MySQL is an open-source relational database management system.
NoSQL	A non-relational database
Relational database	A digital database based on the relational model of data.

7.3.3 Refinements

Originally, we had the idea to use Amazon Web Service's Amazon RDS for our database. An account was created and connected to the Ex-scavenge-anza application. After playing around with it, we realized that while Amazon RDS is a great option, Google's Firebase Real-Time Database was slightly better to use for our application. Firebase is a mobile platform that is used to build mobile applications. For the purpose of our application, it is crucial to use a real-time database because we constantly need to update the user where they are located as well as send out clues once they've arrived at a specific location. The database will constantly be tracking the user location and compare it to the point of interest. Firebase also utilizes cloud storage so that it is accessible anywhere and it is compatible with Android Studio. We decided to make the switch from MySQL to NoSQL because the data related to our app is associated to particular locations, therefore we must be flexible with our data.

7.3.4 Scrum Backlog

Refer to section 6.

7.3.5 Coding

Due to the switch to Firebase Database, most of the code required to connect the database and send data back and forth to the application is written in both Java and JSON.

```
"client": [
  {
    "client_info": {
      "mobilesdk_app_id": "1:122640338060:android:20ad9bef12078669",
      "android_client_info": {
        "package_name": "com.example.ariana.scavenger"
      }
    },
    "oauth_client": [
      {
        "client_id": "122640338060-f6poklu0m8936l3bjmootdn51e4ppaf3.apps.googleusercontent.com",
        "client_type": 3
      }
    ],
    "api_key": [
      {
        "current_key": "AIzaSyCaZ4moyiPrWceZKmGzQt1LPfBS-3BjlOQ"
      }
    ],
```

Figure 7.3.5a: Connection to Firebase

Figure 7.3.5a demonstrates some of the code that was used to connect Firebase to the application.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write;
    }
  }
}
```

Figure 7.3.5b: Reading and Writing to Database

Figure 7.3.5b demonstrates one of the first issues we had with connecting to Firebase which was making the database public. When it was private, we were unable to send data back to the application if it was not on my own personal computer. We needed to make it public in order to be able to retrieve data from any device. However, this could lead to many problems since anyone could access the database. It has since been set back to private and I am currently working on a way so that the database only authorizes the host to make any possible changes to the application.

7.3.6 Testing

For the testing portion of the database, it is very important to properly receive and send data to the database from the user device. Several tests need to be completed to make sure that the database is running smoothly. Part of the testing that has been completed thus far has consisted of manually inserting text into the database and having it sent back to the application to appear on the screen. The chart below will list any additional testing required for the database:

Test	Expected Outcome
Data Mapping	The user needs to be able to easily navigate through the application while having the database respond properly to their commands. Front-end needs to be properly connected to the back-end as the user is attempting to create a new scavenger hunt.
Data Handling/Integrity	The database needs to be able to handle the data properly without the possibility of losing the data or manipulating the data the user inputs. The database must check if all the data being input is valid.
Security	The database should only allow modifications by the admin.
Queries	Data, i.e. created scavenger hunts, should be able to properly appear on the screen of the participant.

7.4 Integration – Dane Hansen

The focus of this subsystem is to logically and effectively connect subsystems into a coherent system. While doing this, verifying requirements and validating system performance. Analysis of system once connected, working to have better speeds, and making sure that system runs identically on separate devices.

7.4.1 Initial design and model

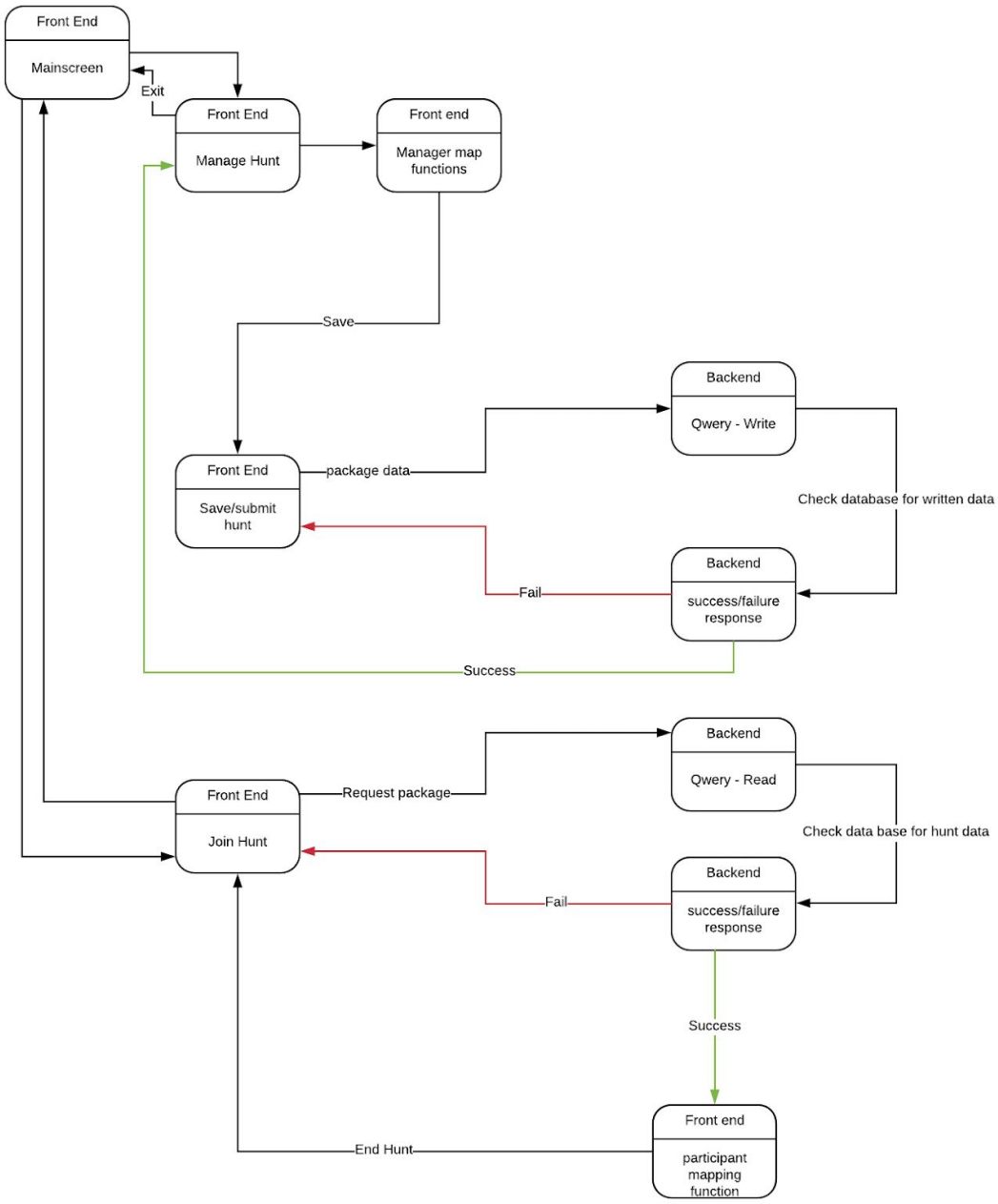


Figure 7.4.1a

Backend and frontend connection using database query and screen selection. The manager has more options using the map functions, it needs to be separated to ensure participants of hunts are not able to create points of interest. The queries to the database will return a success or failure on the write/read.

7.4.2 Scrum Backlog

Refer to section 6.

7.4.3 Coding

The frontend menu screens are using Java, the map function screens are in javascript, and the database uses a mixture of java and JSON calls. The integration subsystem will use a mixture of all three languages to connect each subsystem.

```
    setContentView(R.layout.activity_join_hunt);
}

public void joinBtn(View view) {
    startActivity(new Intent(getApplicationContext(), ParticipantMap.class));
}


    setContentView(R.layout.activity_home_screen);
}

public void joinHunt (View view){
    startActivity(new Intent(getApplicationContext(), JoinHunt.class));
}

public void createHunt(View view) {
    startActivity(new Intent(getApplicationContext(), MainActivity.class));
}
```

7.4.4 Testing

When an action is completed, make sure the correct response is given from the system. Running code on an emulator and creating/joining hunts to ensure useability across the app. Using different versions of android to ensure that graphics and usability is obtained across different devices. While testing, the team noticed a change in accuracy when connected to Wifi or a cellular network. Other functionality with the application has been tested to make sure the right popup appears and the POI tag is correct.

8. Complete System

Refer to Github Repository: https://github.com/emabelyn/Scavenger