Walk the Hills

College Hill Association and Pullman 2040- Mobile Application

College Hill Association

and

Pullman 2040

Finite Cipher

Adam Wagener;

Mitchell Footer;

Juan Ibarra-Delgado;

TABLE OF CONTENTS

I. Introduction

The College Hill Association and the city of Pullman recognized the need to enhance pedestrian safety and accessibility within the Pullman community. The growing concern for safe and efficient foot traffic in the area prompted these organizations to seek a solution that would empower residents and visitors to navigate the city on foot with ease. Traditional modes of transportation often proved inadequate or unsuitable pedestrian travel within the city. In response to this, the College Hill Association and the Pullman 2040 team turned to us to develop a practical, user-friendly application that could enhance and protect pedestrian travel in Pullman.

The motivation behind this project stems from a genuine desire to foster a safer, more convenient, and enjoyable environment for all people in Pullman be they residents, students, visitors, tourists, or anyone else. By providing a place to provide feedback on the local community as well as navigate from one place to another with an information source stocked with local insight, we strive to make walking in Pullman safer and better for everyone. Our vision aligns with the broader goal of creating a cohesive and vibrant community where people can explore and engage with their surroundings on foot as well as inform the city of any safety or maintenance ease with a simple and user-friendly interface.

The core objective of this project is to design and implement a robust mobile application that enables all people in Pullman to effortlessly find walking routes from point A to point B while giving them a voice in how the city shapes over time. This application will take into account various factors, such as the time of day and user evaluated safety metrics, to provide real-time recommendations for navigating the city securely. By offering a solution tailored to the specific needs of the Pullman community, we hope to enhance the overall quality of life for its residents and foster a culture of pedestrian-friendly urban planning. We also hope to gather data that will inform city council and other members of public works on how to move forward.

One major stakeholder wants to use this application as a part of his mayorship with the city if elected. Through the application he desires data collection on what people are using and what needs are unmet. The application will move to gather information on what routes are being taken and offer the public a means to report that information to the application.

This final piece of data collection was a major shift as the changing requirements of our client evolve. Though the application now has two major focuses splitting the needs and functions, the application can still thrive in a single robust unit. This has led to some major framework development setbacks, but also allows us to expand the design in a very meaningful way to better aid the city and all those that walk within it. Our new focus is on accessibility and community voice.

II. System Overview

The project has two main objects, both of which compose their own major system and design. The first objective is our original desire to provide the community with safe walking instructions to get from one location to another within the city of pullman. This functionality will largely comprise the user interface and the public facing components of the design. Here users will gain access to the power of Google Maps and the accessibility of our custom

product. They will be able to filter to a greater extent their results and walk the city in a safe and customizable way. The second portion of the project is a database for user input. Here we will store information on what routes are being walked and in what frequency. We will also store user inputs on their thoughts and feelings on routes and a dynamic safety metric that will help future users determine the safest path from one location to the next.
The two components overlap in user reporting. Our user interface for navigation will also include the ability to drop pins and make comments as well as a final evaluation request from users. That information will be stored in the database so that city council and the pullman city mayor can access the information and inform future public works projects around the city.

III. Architecture Design

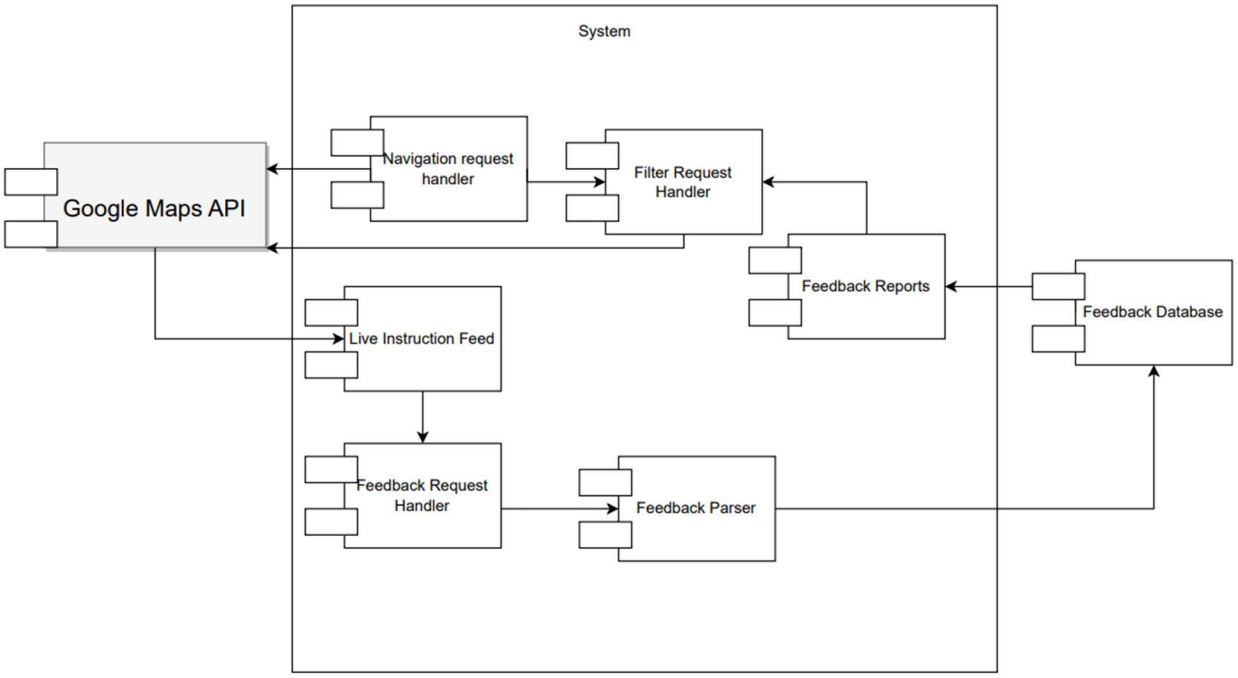
III.1. Overview

The Finite Cipher team has decided to implement a command query response segregation (CRQS) model for this project. This model will allow us to handle the in and outflows of data through our database and google maps in a very efficient manner. Since the database we will implement must be doing constant reads and writes, it is essential that we organize those reads and writes in a streamlined way so that no data gets crossed or lost. This model helps us manage our database queries into distinct read and write requests [1].

This model is perfect for systems that can be divided into two categories; one that treads, one that writes. This is exactly what our system will do. Users provide feedback which writes to the database and request routes with filtering which must read from the database. Through these simple interactions we will build our foundation.

The below component walks through how these two systems interact and their distinct read versus write pathways. The Google Maps API and our database will both follow strict rules in what allows read or write access to central information. Through this delineation we expect to build a straightforward, user-friendly, and streamlined application. The user interface will be incorporated into the read access controls while the write access controls result only from one specific user input. This delineation helps to ensure the integrity and reliability of the information accessed and stored.

The below diagram explains how navigation data all flows out of Google Maps API as well as elevation data, while safety metrics flows out of the built in database. From Google Maps navigation information, we are able to provide a route to the user. We can then filter those results using the filter request handler which needs to query google maps for new routes. The filter routs handler also receives information from feedback reports which supply the read access from our database. Below this chain we see the write access chain in which the live instruction feed received from google maps provides the user with the ability to provide feedback. This feedback must be handled in the system through the request handler. That feedback is then parsed and sent to the database which supplies the write access to the database.



III.2. Subsystem Decomposition

I.1.1. [Navigation Request Handler]

a) Description

The navigation Request Handler takes in user input to generate location information. Using Google Maps API autofill feature we can predict supported locations and help users find their desired location through the navigation handler. Once the user has their desired inputs the navigation handler sends the request to google maps to generate a route and leads the user to the next steps of rout filtering.

b) Concepts and Algorithms Generated

This subsystem is a fairly simple interface that will handle string mutations. Through user input the system will need to remain secure and verified. All information will be sent through a known secure system using google maps API autofill. This will help ensure avoidance of things such as SQL injections and other cyber attacks that come with user input strings.

c) Interface Description

Service Name	Service Provided to	Description
Get location	Google Maps API, Filter Handler	Prompt user to input location and destination fields as strings. Use autofill to assist and validate input strings to be used to get navigation controls.
Clear Location	Self	Clear existing locations and remove prior navigations to allow friendly reuse environment.

Get Route	Google Maps API, Filter Handler	Send existing autofill data to be filtered or routed via filter routes or Maps API.
Get Filter Request	Filter Request Handler	Prompt user to select from a dropdown the desired filter if moving from the default. Send this information to the internal structure to filter results.

Services Required:

Service Name	Service Provided From
Navigate	Google Maps API

I.1.2. [Filter Request Handler]

a) Description

The filter request handler will offer the user the ability to filter the route they are requesting by various criterion such as safety, elevation gain, or fastest. The filters will take into account how these variables change by time of day using the system clock on the local device. The information to filter by will be stored on google maps api data access or in our personal database.

b) Concepts and Algorithms Generated

This subsystem will be very complex as it requires direct manipulation of google maps primary feature. This feature, according to our research, will not be overridable. We will need to run a few methods to overcome this hurdle. Currently using the add stop feature we can implement route changes that a user may request, and we can do this repeatedly to alter a route entirely. This process is slow, but functional. Optimizing the add stop re-rout will be a critical late-stage issue as we look to efficiency in stretch goals.

c) Interface Description

Service Name	Service Provided to	Description
Re-Route	Google Maps API, Live Instruction Feed (secondary receiver)	Gathering data, compute best path to destination to minimize elevation gain and set stops to re-rout accordingly.

Services Required:

Service Name	Service Provided From
Get Elevation Data	Google Maps API
Get Safety Data	Feedback Reports

I.1.3. [Feedback Reports]

a) Description

Generate feedback reports to be used for safety filtering and raw data output for city council analysts to observe. This feature will generate three separate reports. One that reports the daily safety metrics to make read-access for safety filtering a safer and more reliable method. A second to report the raw strings showing user feedback reports. And a final report that shows the most common traveled paths and the usage rates for those routes.

b) Concepts and Algorithms Generated

These reports will be a simple matter of computing stored data from the database and presenting it in a human readable format. The computational complexity will be light, but modifying the reports to the specific analyst preference will take some time.

c) Interface Description

Service Name	Service Provided to	Description
Generate Reports	Filter Request Handler	Generates all three reports. Two for human consumption, one to be used for filtering by safety metrics. The safety metrics report should be stored locally for the day to make filtering a faster endeavor.
Print Report	Authenticated Council Member	This process is self-contained on the database side to be a simple report file for the authenticated developer or city council member. This function is not for public use.

Services Required:

Service Name	Service Provided From
Querry Database	Database

I.1.4. [Live Instruction Feed]

a) Description

Generate a live feed of instructions for a user to get updates and changes to their walking instructions. This service will also allow users to drop pins nearby and add comments on route details to be stored in the database. These pins will allow raw string comments only. Safety information can only be input once a route is complete or abandoned.

b) Concepts and Algorithms Generated

The actual live feed instructions and re-route gps tracking algorithms are provided by Google Maps API. However, we will need to implement custom pin drops to accommodate commenting and other interactions. These pin drops are supported by maps, but do not automatically allow users to input data to be collected. This will require some restructuring of the maps internal structure on pins and location information.

c) Interface Description

Service Name	Service Provided to	Description
Drop Pin	Feedback Request Handler	Drops a pin on a selected path to allow a user to input a comment on a routes condition such as “streetlamp out” or “requires repaving”. This is then sent to the feedback request handler to be written to the database.
Update Instructions	Self	A self-service function to continually monitor the route and update the live feed of instructions according to the map API updates on the route.

Services Required:

Service Name	Service Provided From
Update Directions	Google Maps API

I.1.5. [Feedback Request Handler]

a) Description

After a pin is dropped or a route is completed, the feedback prompt window must open and allow users to add in as much detail to their report as they desire. A safety rating metric is required to be entered if a user wants to add feedback. Here they can add comments about the routes that need to be seen by city council and help other users find the safest paths possible.

b) Concepts and Algorithms Generated

Generating successive requests for additional information will exhaust our users. It will be key to generate a simple to use and friendly feedback system that allows users a full opportunity to reach out, but not overload them with questions to prevent them from providing feedback. This method of prompting will require additional social engineering and questioning before its full implementation. Ideally the prompts will be limited, but with good suggestions to encourage full responses.

c) Interface Description

Service Name	Service Provided to	Description
Get Safety Rating	Database	Gather a simple stare rating on safety for the route. If a bad rating is received, ask if there was a particular road or street that was problematic before storing the data. Also ask if the rating would be impacted if it were day or night time.

Get Additional Comments	Feedback Parser	Generate a text window to gather user input comments on a given route. Send this raw string data to the parser.

Services Required:

Service Name	Service Provided From
Drop Pin	Live Instruction Feed

I.1.6. [Feedback Parser]

a) Description

Once a string has been received from a user to add a comment, the data needs to be verified and filtered. Getting information from users is always a security vulnerability that needs to be addressed. We need to parse the string and look for signs of malicious behavior. Once verified as safe, we will want to see if sub comments on specific routes exist and filter those into their own comment blocks. These final comment blocks can then be sent to the database for long term storage.

b) Concepts and Algorithms Generated

This information needs to be scanned for malicious behavior. The presence of special characters that might be used to attempt code generation in the string need to be filtered over. This algorithm should verify emoji's vs code injection. Verification will be case by case, and should not stop the rest of the system from running. Malicious code needs to be thrown away. For filtering route callouts, we can use Chat GPT API for advanced use if we get approval, otherwise a simple scan for road names that were routed over will suffice.

c) Interface Description

Service Name	Service Provided to	Description
Return Valid Feedback	Database	After verifying and filtering the results, this function should generate one or more reports to the database to be stored containing lists initialized with their route information followed by either the comment or safety metric received.

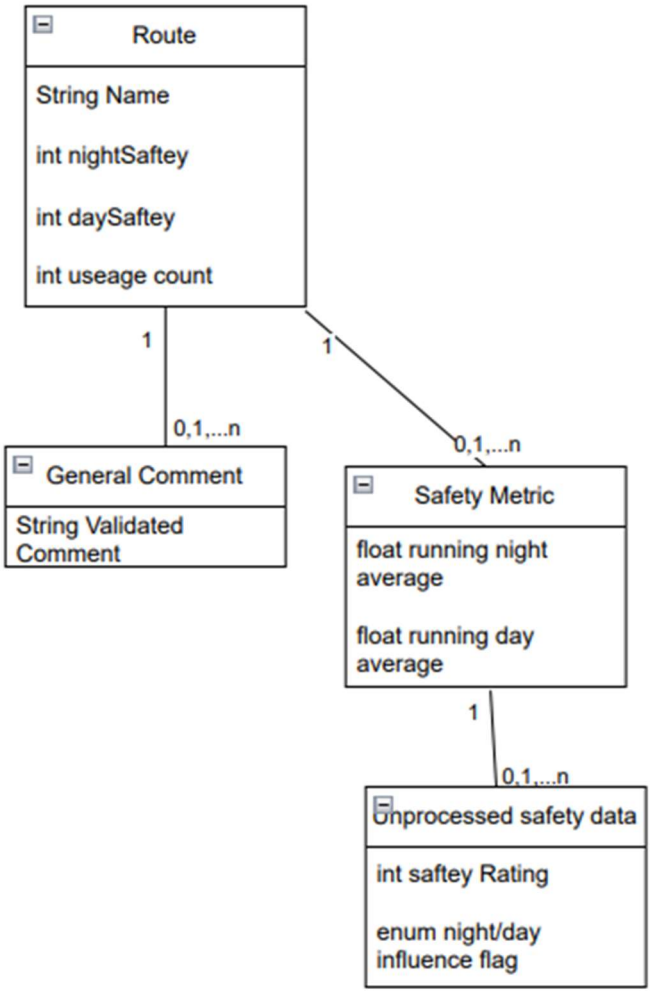
Services Required:

Service Name	Service Provided From
Get Additional Comments	Feedback Request Handler
Get Safety Rating	Feedback Request Handler

IV. Data design

This application requires a major file structure referred to as simply database. This will contain all the information available on routes provided by our users. This database will contain potentially thousands of entries over a wide range of routes and locations. Sorting through the

database will be a lengthy process we want to limit the need for. The database can be read from using a long return time as a read needs to only happen once on bootup, but the write system must be efficient as thousands of writes may happen at the same time. Additionally, these writes should never interfere with each other. Many users may enter data on a route at the same time. To handle this, we will want an indexed data type such as a dictionary or array to be used to quick access existing entries for writing. Those entries should support dynamic nodes that are associated with them to allow for writes to happen together without issue. We will implement that using a simple pointer node structure like a tree or linked list. Comments on a route will be stored in their own lists and safety data will gain more and more nodes as time goes on. To allow this to operate in a dynamic way, metric node compression will be supported to shorten list length into a single node when safety metric's follow the same pattern. This allows us to store only a running average and discard the thousands of nodes otherwise needed to store.



Through this storage space we can generate a safety report quickly on app startup, generating suer reports to an authenticated council member will take much longer, but that process is one that can be lengthy as the information will take days to process on their end as well. This data should have some edit access from an authenticated user so they can clean comments once a

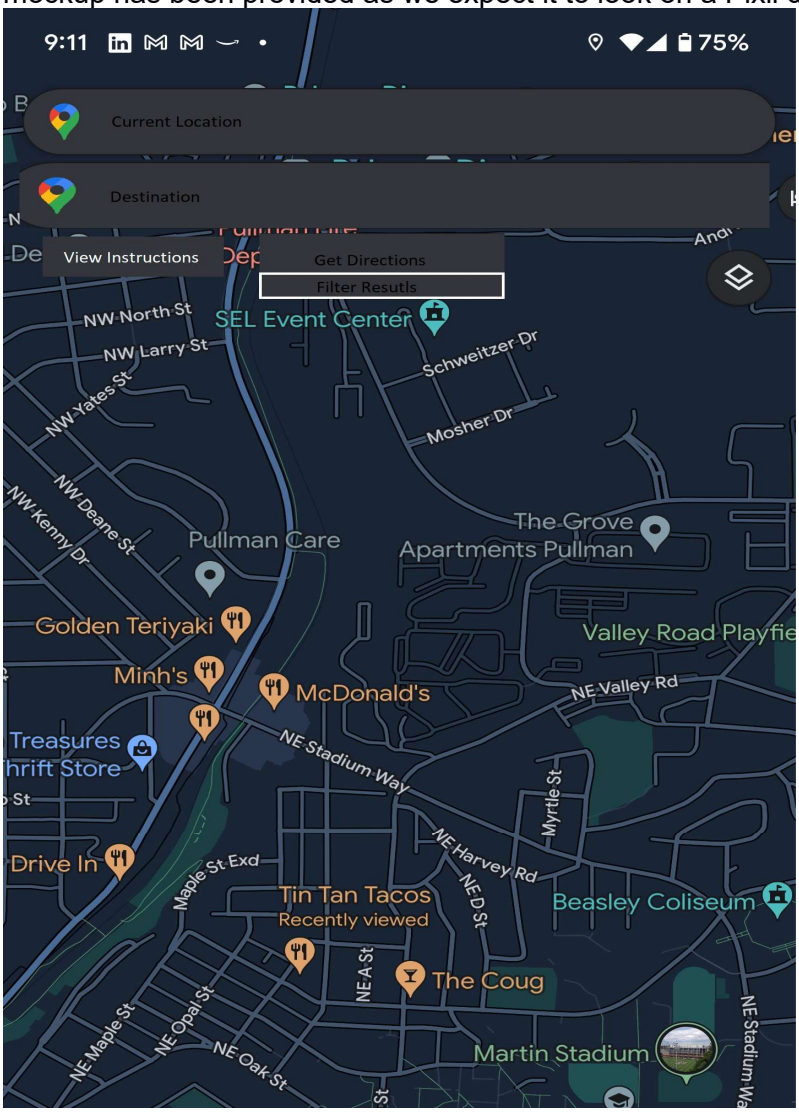
report has been generated. Cleaning comments will reduce the database size to allow the application to maintain a long lifecycle.

In order to keep both the safety metric and rout count data information intact, we will want to ensure this structure uses a queue style processing for new nodes. Once new feedback is received it should be attached as a node to be processed rather than processed immediately. This way we do not run into two nodes processing at the same time. Once in the queue the database will process them FIFO as the application has time in the background. This will aid in user experience to avoid any routing slowdowns as well.

V. User Interface Design

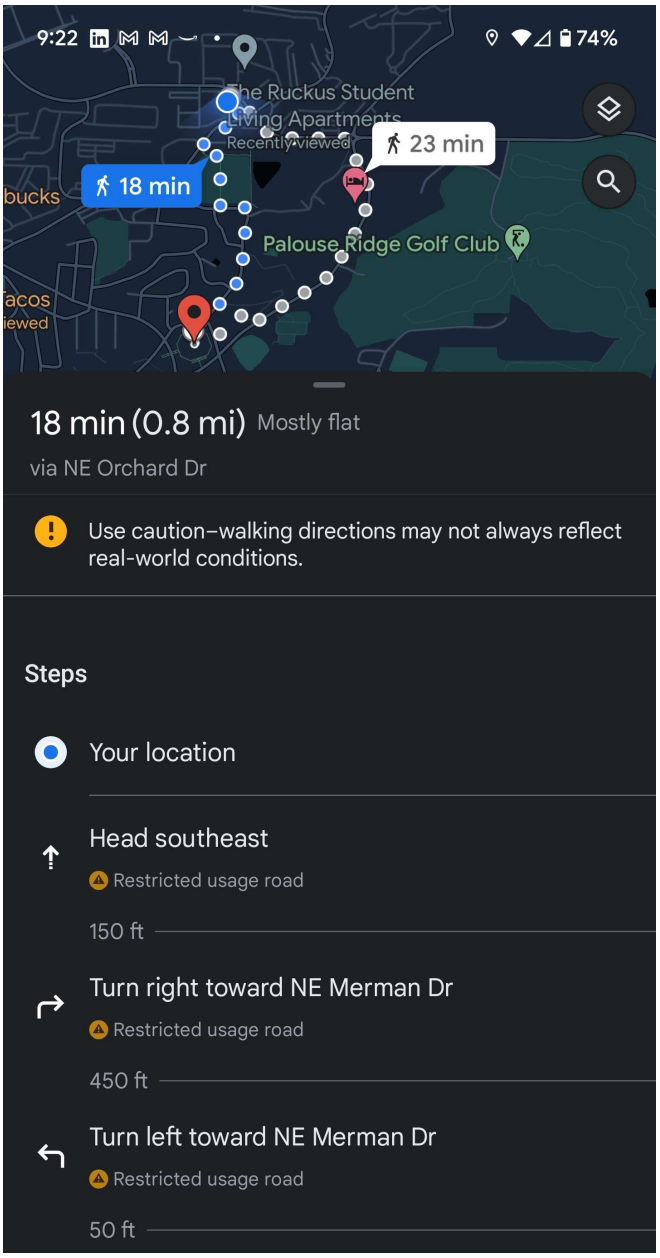
The focus of the user interface is to provide a familiar experience to users. We will mimic the google maps layout in most respects with some simple button additions to reflect our added features. We want users to feel at home with their mobile experience. The application should be user friendly and focus on giving the user the experience they installed the application for; getting safe navigation. With this in mind the application focuses on making the map and route display as large as possible with filtering and instruction explanation left minimal as optional inputs. This interface will need to be cross platform and operate on a myriad of devices. As such, each interface may look slightly different with these criterion in mind. A sample interface

mockup has been provided as we expect it to look on a Pixil device.

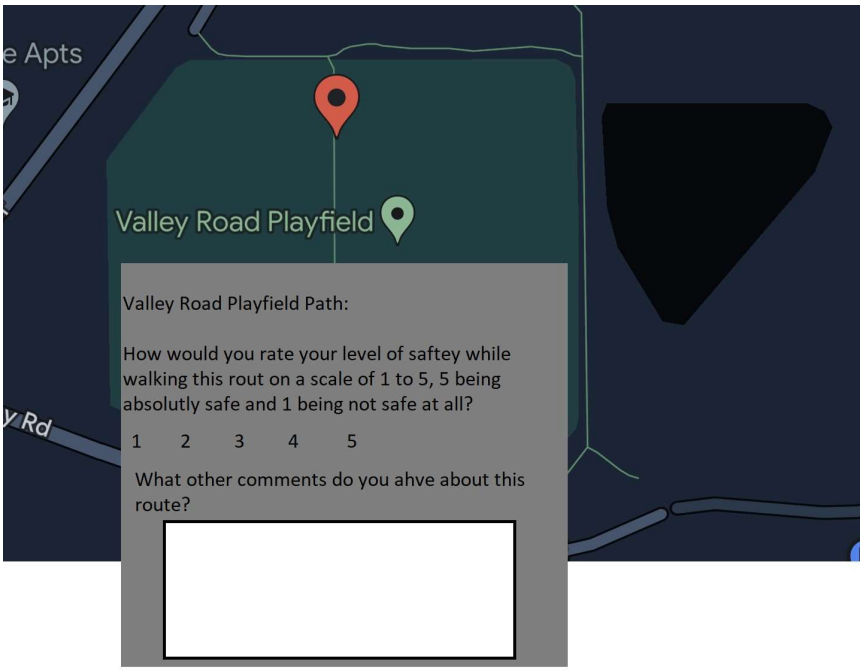


This interface is exactly the one you would see when using google maps. This should allow our users the same friendly experience with the added benefit of safety filters and feedback reporting. Google Maps is massively successful for its driving instructions, but many do not see the opportunities for walking that we will implement. By keeping the interface we expect to see the same level of quality user experiences that google has invested millions of dollars to achieve.

The first image seen here is what a user will see when they boot up the application or when they start looking for a new route. Below we see a sample of viewing the live walking instructions and the map view of route selection once a route is request:



The user should be able to pull up live instructions as needed, but otherwise they will be stored away as we saw in the first image. Finally, the user should be able to drop a pin in paths to add comments as desired. These pins will open a simple text interface and stare rating system as seen below:



VI. Glossary

API: Application Programming Interface. An interface provided by a third party to allow us to integrate their work and project into our own using our own custom functions built around theirs.

Database: A central hub of data to be collected and stored inside software.

Data: Any form of information that has been digitalized. Largely we are considering user feedback and user safety metrics when referring to data which will be stored in our database.

UML: Unified Modeling Language. A standard format for creating diagrams to explain software processes and requirements.

VII. References

[1] B. Reselman, “The Pros and cons of the CQRS architecture pattern,” Enable Architect, https://www.redhat.com/architect/pros-and-cons-cqrs (accessed Sep. 22, 2023).