

# 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	4
II.	Team Members - Bios and Project Roles	5
III.	Project Requirements	8
I.	System Requirements Specification	8
I.1.	Use Cases	8
I.2.	Functional Requirements	10
I.2.1.	Cross Platform Mobile Application	10
I.2.2.	High Performance Routing Accounting for Local Incidences	10
I.2.3	Dynamic Filtering and Citizen Voice	11
I.2.4	Safe and Accessible	11
I.3.	Non-Functional Requirements	12
II.	System Evolution	13
IV.	Solution Approach	13
I.	System Overview	13
II.	Architecture Design	13
II.1.	Overview	13
II.2.	Subsystem Decomposition	14
I.1.1.	[Navigation Request Handler]	14
I.1.2.	[Filter Request Handler]	15
I.1.3.	[Feedback Reports]	16
I.1.4.	[Live Instruction Feed]	17
I.1.5.	[Feedback Request Handler]	17
I.1.6.	[Feedback Parser]	18
III.	Data design	19
IV.	User Interface Design	20
V.	Test Plan	23
I.1.	Test Objectives and Schedule	23
I.2.	Scope	24
I.	Testing Strategy	24
II.	Test Plans	25
II.1.	Unit Testing	25
II.2.	Integration Testing	26
	Alpha Prototype Report	2

II.3.	System Testing	26
II.3.1.	Functional testing:	26
II.3.2.	Performance testing:	26
II.3.3.	User Acceptance Testing:	26
III.	Environment Requirements	27
VI.	Alpha Prototype Description	28
VI.1.	Navigation Request Handler	28
VI.1.1.	Functions and Interfaces Implemented	28
VI.1.2.	Preliminary Tests	28
VI.2.	Filter Request Handler	29
VI.2.1.	Functions and Interfaces Implemented	29
VI.2.2.	Preliminary Tests	29
VI.3.	Live Instruction Feed	29
VI.3.1.	Functions and Interfaces Implemented	29
VI.3.2.	Preliminary Tests	29
VI.4.	Feedback Request Handler	29
VI.4.1.	Functions and Interfaces Implemented	29
VI.4.2.	Preliminary Tests	30
VI.5.	Feedback Parser	30
VI.5.1.	Functions and Interfaces Implemented	30
VI.5.2.	Preliminary Tests	30
VI.6.	Feedback Database	30
VI.6.1.	Functions and Interfaces Implemented	30
VI.6.2.	Preliminary Tests	31
VI.7.	Feedback Reports	31
VI.7.1.	Functions and Interfaces Implemented	31
VI.7.2.	Preliminary Tests	31
VII.	Testy Results	31
VIII.	Project and Tools Used	35
IX.	Description of Final Prototype	35
X.	Product Delivery Status	38
XI.	Future Work	39
XII.	Glossary	39
	Alpha Prototype Report	3

## 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. Team Members - Bios and Project Roles

Project Team Lead:



**ADAM WAGENER**

COMPUTER SCIENCE | MAPLE VALLEY, WA

### EDUCATION

Tahoma High School

University of Washington  
BA Mathematics

University of Washington  
Masters in Teaching

Washington State University  
BS Computer Science  
GPA: 4.0

### SKILLS

- C/C++ from Systems Programming and advanced data structures
- Python from Interning with SEL
- Arm assembly from firmware engineering with SEL
- C# from large scale spreadsheet GUI application
- Rust from personal projects and mentored learning
- Linux laptop environment for one year
- Haskell from functional programming courses
- LabView from high school robotics
- TA for CPT\_S 355, Programming Language Design

### WORK EXPERIENCE

Software Engineering Intern  
with Schweitzer Engineering  
Laboratories (SEL) 2022-  
present

Math, Science, and Computer  
Engineering instructor with  
Washington State Public  
Schools 2016 - 2021

### OTHER INTERESTS

- Tabletop Gaming
- Game Development and Design
- Fiction Writing
- Bicycling
- White Water Rafting
- Camping
- Hiking

Additional Members:



## MITCHELL FOOTER

COMPUTER SCIENCE | SNOHOMISH, WA

### EDUCATION

Glacier Peak High School

Everett Community College

Washington State University  
GPA: 3.31

### SKILLS

- C from Systems Programming and advanced data structures
- Java from AP computer science and personal projects
- Dart from personal project Disco and other web development research
- Linux laptop environment for around a year
- TA for CPT\_S 260, Computer Architecture

### HONORS AND AWARDS

WSU Regents Scholar

WSU Top Scholar

Voiland Dean's Scholarship  
Recipient

### OTHER INTERESTS

- Backpacking/Camping
- Mountaineering
- Running
- Cooking
- Gaming
- Traveling



# JUAN IBARRA-DELGAOD

SOFTWARE ENGINEERING | OMAK, WA

## EDUCATION

Omak High School

Washington State  
University  
GPA: 3.30

## SKILLS

- C/C++ with Advance Data Structure
- C with Systems Programming
- C# with making a Spreadsheet GUI
- Python and SQL from a Introduction to Database Systems course
- Using Linux as my main OS on my laptop for 3 months
- Teachers Assistant for Program Design and Development course (CptS 121)
- Cooperated with multiple team members from the same school and from different schools on Military Training events that have taken about a whole semester to plan for over 200 cadets.

## WORK EXPERIENCE

ARMY NATIONAL GUARD  
Worked as a lower enlisted for about 2.5 years and then became a Platoon Leader for about 2.5 years as well.

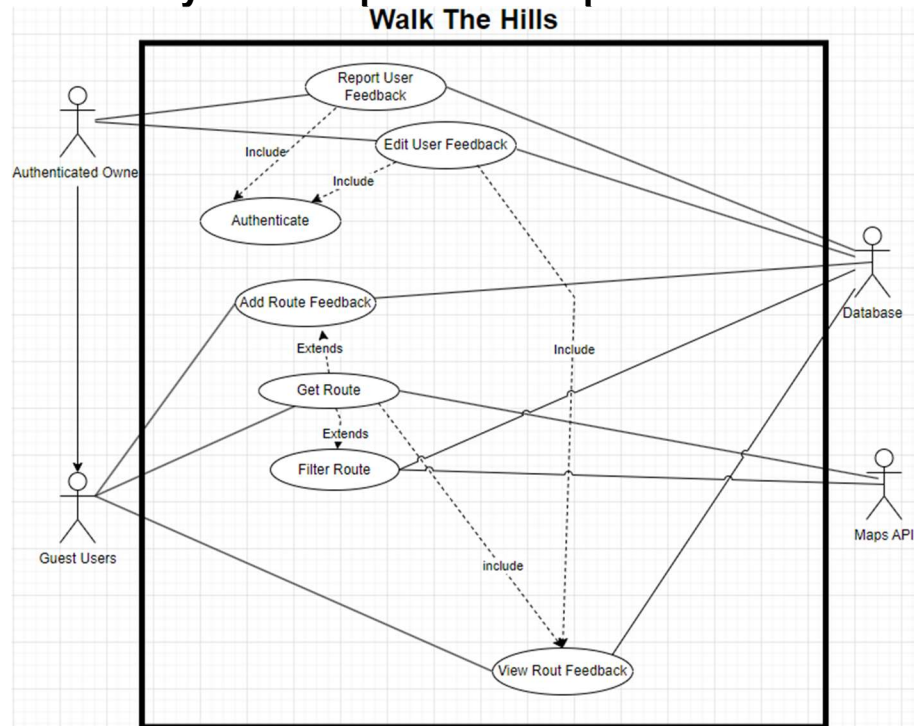
ARMY ROTC  
Worked as a lower level cadet and then was put in a position that was responsible for planning training events and ensuring cadets met regulations and standards.

## OTHER INTERESTS

- Trail running
- Spending quality time with family
- Watching Movies/TV with my family
- Playing video games with friends and family
- Doing something competitive with my wife
- Exercising

### III. Project Requirements

#### I. System Requirements Specification



This application serves two main services that are handled through the functions as shown in the above UML diagram. The first of these functions we will discuss is providing walking routes for general users. These users should be able to access the application through a mobile device and gain a quick access view of walking instructions to get from point A to point B. This process should be accessible for all general users without needing log in or verification. In addition to providing a walking route, it is essential that the application also give users the ability to filter their route using multiple factors such as safety metrics, elevation, or fastest travel time. The get route function will be the bulk of the project as it services our largest stakeholder group.

In addition to routing and filtering, the get route function must also allow users to input feedback on the route they have taken. This feedback will be stored in a database to be accessed later. The route given will come from another stakeholder, Google, who provides us with their maps API.

The database object will provide a great deal of handling for route filtering, but also serves as a critical infrastructure to accomplish the second major task of the project. Reporting user feedback to city council members and the local police in an effort to get the city of Pullman the maintenance and safety information it needs to keep its population safe and happy. Through the database a registers council member or other authenticated user can access and edit the user input feedback to adjust route filtering. Accessing this data will allow them to make city changes that can then be used as justification for removing old database items such as streetlamp outages once fixed.

##### I.1. Use Cases



## Get Route

Pre-Condition	Application launched
Post-Condition	Route information displayed. Filter dropdown available.
Basic Path	<ol style="list-style-type: none"><li>1. Locate location and destination fields.</li><li>2. Input destination.</li><li>3. input location field.</li><li>4. Select route.</li><li>5. Walk route.</li></ol>
Alternate Path	<ul style="list-style-type: none"><li>- In step 4 the user may select to add filters to their route and adjust their route before beginning to walk.</li></ul>
Related Requirements	<ul style="list-style-type: none"><li>- Maps API Integration</li><li>- Filter Route</li></ul>

## Filter Route

Pre-Condition	Route request received
Post-Condition	New Route returned, filtered based on selection
Basic Path	<ol style="list-style-type: none"><li>1. Select filter dropdown and desired filter.</li><li>2. Select route.</li><li>3. View new route.</li></ol>
Alternate Path	<ul style="list-style-type: none"><li>- NA</li></ul>
Related Requirements	<ul style="list-style-type: none"><li>- Maps API Integration</li><li>- Database integration</li></ul>

## Add Route Feedback

Pre-Condition	Route in progress or completed
Post-Condition	Feedback stored in database
Basic Path	<ol style="list-style-type: none"><li>1. Prompt for user feedback</li><li>2. Input feedback for safety rating</li><li>3. input feedback for other issues</li><li>4. submit feedback</li></ol>
Alternate Path	<ul style="list-style-type: none"><li>- Before step 1 the user may add a feedback pin to current location and input their own information to follow steps 3 and 4.</li></ul>
Related Requirements	<ul style="list-style-type: none"><li>- Maps API Integration</li><li>- Get Route</li><li>- Database Integration</li></ul>

## Report User Feedback

Pre-Condition	Registered Owner Log In
Post-Condition	Route feedback data copied
Basic Path	<ol style="list-style-type: none"><li>1. Registered Owner Login Authentication</li><li>2. Select feedback report.</li><li>3. Download file of feedback report from database</li></ol>
Alternate Path	<ul style="list-style-type: none"><li>- NA</li></ul>

Related Requirements	<ul style="list-style-type: none"> <li>- Database integration</li> <li>- Add route Feedback</li> </ul>
----------------------	--

### Edit User Feedback

Pre-Condition	Registered Owner Log In
Post-Condition	Route feedback database altered.
Basic Path	<ol style="list-style-type: none"> <li>1. Registered Owner Login Authentication</li> <li>2. Select edit feedback.</li> <li>3. Select feedback pin to edit.</li> <li>4. Adjust safety rating or other information report and change values.</li> <li>5. Select save.</li> <li>6. Select new pin or exit.</li> </ol>
Alternate Path	- NA
Related Requirements	<ul style="list-style-type: none"> <li>- Database integration</li> <li>- Add Route Feedback</li> <li>- View Route Feedback</li> </ul>

### View Route Feedback

Pre-Condition	Application Launched
Post-Condition	Feedback viewable
Basic Path	<ol style="list-style-type: none"> <li>1. Select view feedback.</li> <li>2. Select feedback pin location.</li> <li>3. Read displayed feedback.</li> <li>4. Select new pin or exit.</li> </ol>
Alternate Path	- NA
Related Requirements	<ul style="list-style-type: none"> <li>- Database integration</li> <li>- Add route Feedback</li> </ul>

## I.2. Functional Requirements

### I.2.1. Cross Platform Mobile Application

#### Cross Platform

Description	The application must work on a number of mobile devices to be accessible to the general public. This requirement leads to the decision to work in JavaScript using the React framework. This code can then be packaged and run across an exceptionally large number of devices without major requirements by system.
Source	Internal requirement among team members to address College Hill Association desire for accessibility.
Priority	Priority Level 0: Essential and required functionality

### I.2.2. High Performance Routing Accounting for Local Incidences

### Maps API

Description	The application must adapt to the ever-changing landscape of the city and provide accurate real-time mapping. This requirement leads to the need for Google Maps API integration. A mapping requirement of this size can only be handled by a team of hundreds with round the clock maintenance, so it is best to outsource this to a well-established reliable source such as Google.
Source	Internal requirement among team members.
Priority	Priority Level 0: Essential and required functionality

## II.2.3 Dynamic Filtering and Citizen Voice

### Database Integration

Description	The application must allow users to provide input on what they are seeing. That input must be used to filter future routes. The need leads to the use of an internal database to store and filter user input safety ratings.
Source	City Council members on Pullman 2040 team asked for data input and retrieval.
Priority	Priority Level 1: Desirable functionality

### Database Reporting

Description	The application must allow city council and the local police to view user input data so they can make adjustments to routes and city walkways.
Source	City Council members on Pullman 2040 team asked for data input and retrieval.
Priority	Priority Level 1: Desirable functionality

### Database Editing

Description	The application should allow council members and other entities of authority to edit user feedback once issues have been addressed to keep the database and filters dynamic.
Source	City Council members on Pullman 2040 team asked for data input and retrieval.
Priority	Priority Level 2: Extra features or stretch goals

## II.2.4 Safe and Accessible

### Get Route

Description	The application must provide pedestrian accessible public walking routes through the city of pullman with live location updates and adjustments for walkway closures. We will accomplish this through Google Maps API integration and our own database filters.
Source	Pullman 2040 desires a benefit to users in order to encourage feedback reporting.
Priority	Priority Level 0: Essential and required functionality

### **Route Filtering - Safety**

Description	The application must allow users to alter their route to gain access to the safest route according to user input metrics.
Source	College Hill Association desires a safer walking application for their community.
Priority	Priority Level 1: Desirable functionality

## **I.3. Non-Functional Requirements**

### **Dynamic:**

The system shall update output routes and filtering using a dynamic adaptation. The system must respond to local closures and updates from city council and other users to continually adjust the overall safety and accessibility of routes.

### **Collaborative Community:**

The system shall allow the local community to not only use the application, but also inform the local government of their needs and desires. The application shall report this information to better public transportation and public safety.

### **Intuitive Use:**

The map instructions and user feedback inputs should be self explanatory and easily accessed by all members of the community regardless of technical skills or experience.

### **High Performance:**

The system shall provide live walking instructions in real-time. Users should not need to wait to receive their next instruction while walking. The application will not delay travel in any way.

### **Safe:**

The application shall provide safe routes where possible and will not route people down dangerous or impassible walking paths. The application will avoid private or protected paths.

### **Private:**

The application shall not collect user data beyond that which the user provides such as location and user feedback. All other private information will not be collected unless provided voluntarily by the user.

## **II. System Evolution**

The most significant point of potential software degradation and maintenance comes from our use of the Google Maps API. This mapping software is brilliant, but it also comes with changing requirements and updates that may impact the life of our application. Google Maps API strives to be backwards compatible and provides support for old technology, but eventually updates may be required [1]. It will be the prime effort of our team to inform the technical board member in charge of software maintenance how to update the API integration when needed to ensure this process is painless and actionable. Though the Maps API will possibly cause need for software maintenance, it is still the only option moving forwards. It would not be possible to create an application as powerful as Google Maps as a three-person team in under a year with little to no funding [2]. The good news is that google maps and the react framework will continue to build cross platform designs that allow our application to transcend new generations of devices without major updates beyond the one mentioned above [1].

## **IV. Solution Approach**

### **I. 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.

## **II. Architecture Design**

### **II.1. Overview**

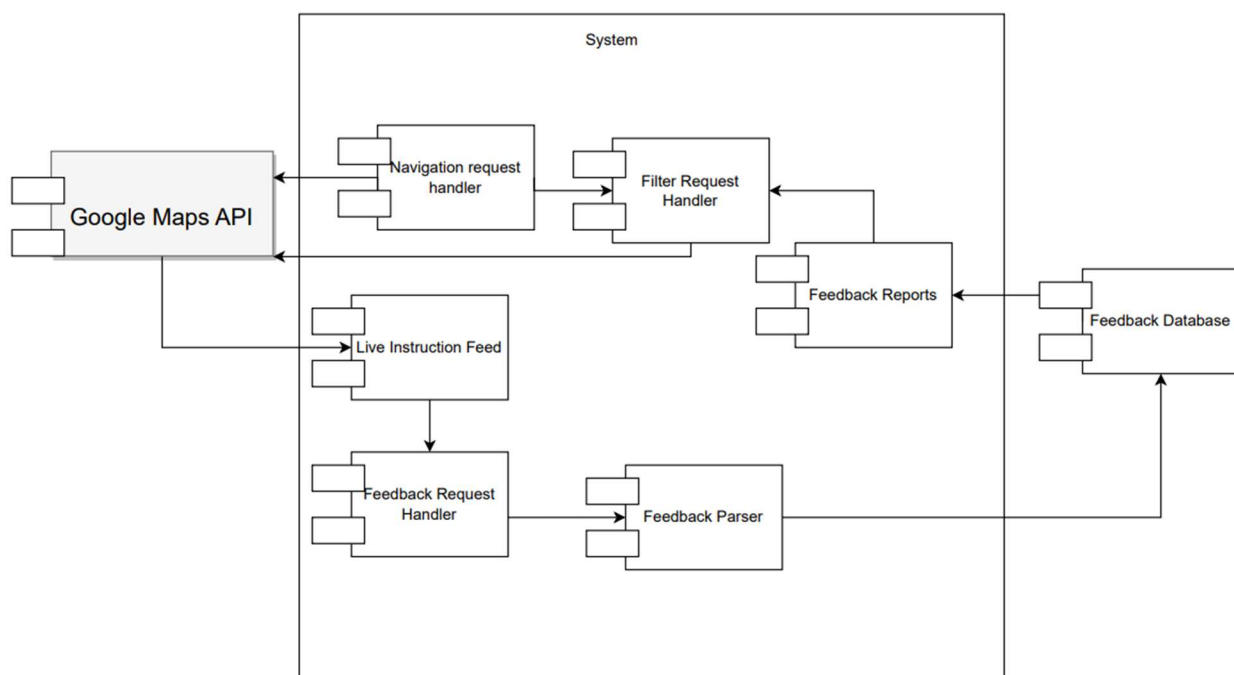
The Finite Cipher team has decided to implement a command query response segregation (CRQS) model for this project. This model will allow us to handle the in and outflows of data through our database and google maps in a very efficient manner. Since the database we will implement must be doing constant 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 reads, 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 routes 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.



## II.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*

<u>Service Name</u>	<u>Service Provided to</u>	<u>Description</u>
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:

<u>Service Name</u>	<u>Service Provided From</u>
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-route 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-route 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
Query 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 star 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

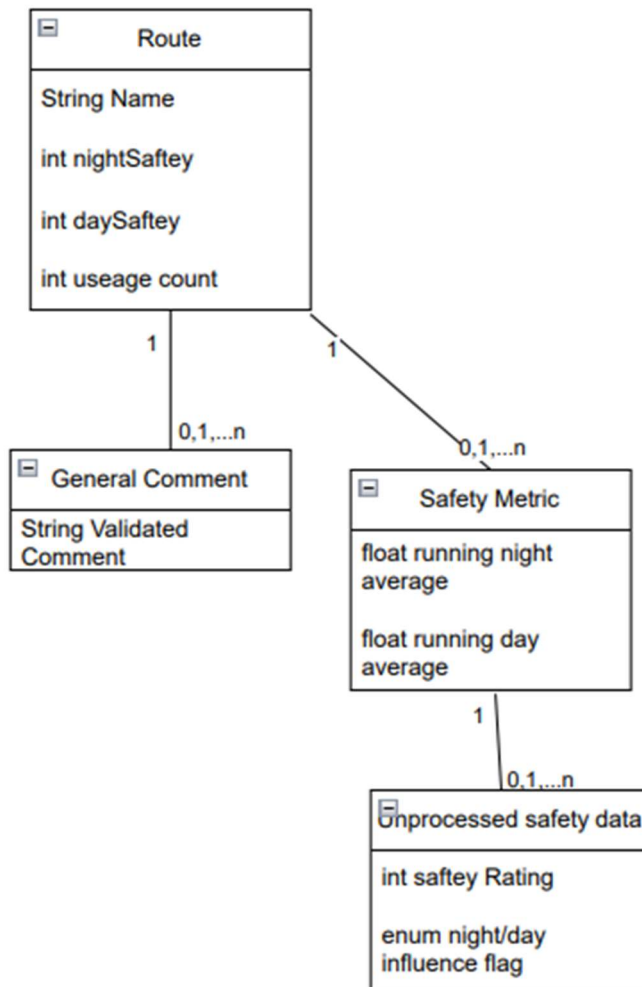
		should generate one or more reports to the database to be stored containing lists initialized with their route information followed by either the comment or safety metric received.
--	--	--

#### Services Required:

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

### III. 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.



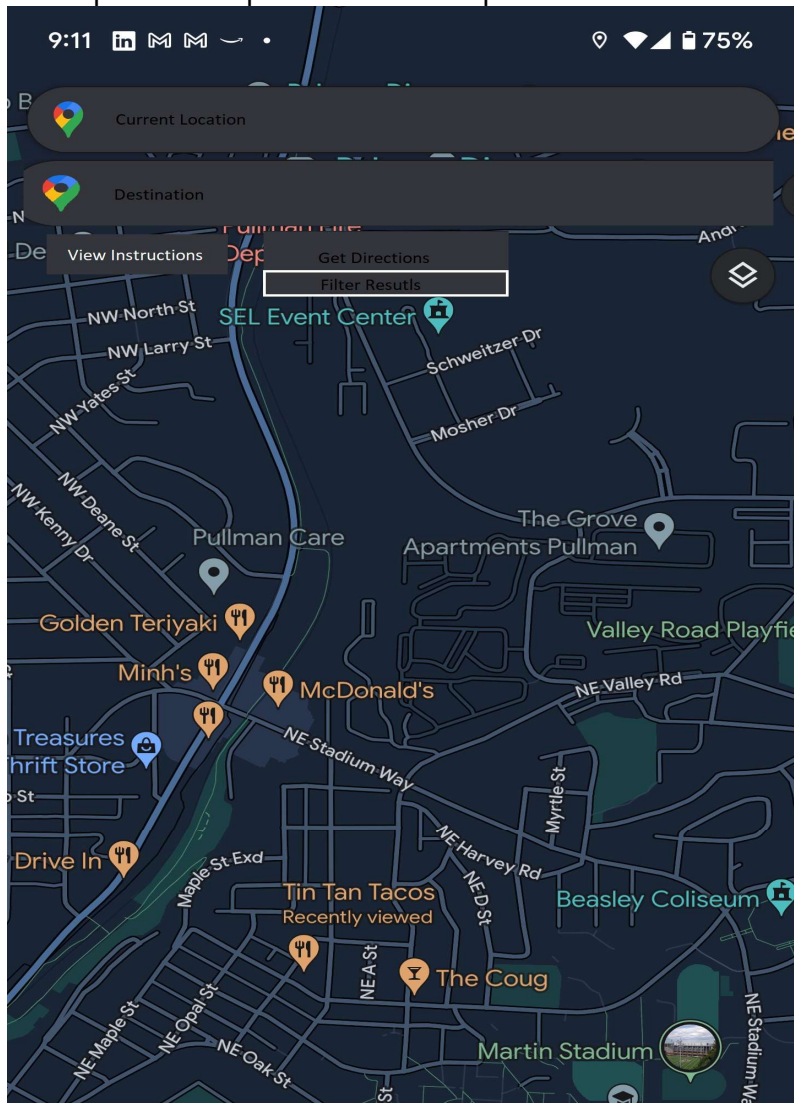
Through this storage space we can generate a safety report quickly on app startup, generating suer reports to an authenticated council member will take much longer, but that process is one that can be lengthy as the information will take days to process on their end as well. This data should have some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

## IV. User Interface Design

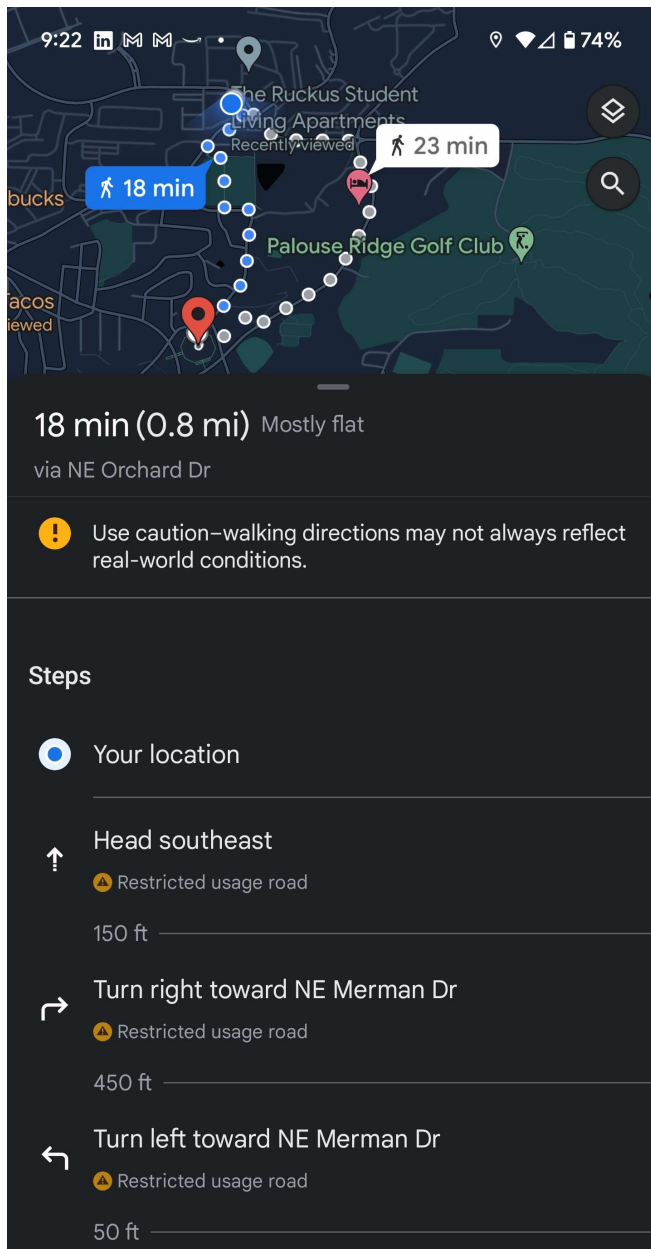
The focus of the user interface is to provide a familiar experience to users. We will mimic the google maps layout in most respects with some simple button additions to reflect our added

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.

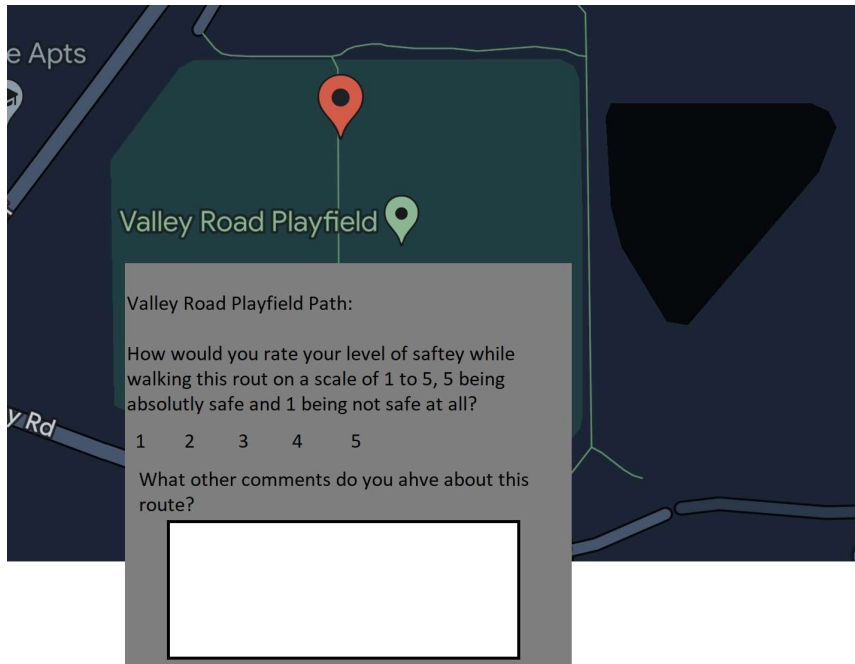


This interface is exactly the one you would see when using google maps. This should allow our users the same friendly experience with the added benefit of safety filters and feedback reporting. Google Maps is massively successful for its driving instructions, but many do not see the opportunities for 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 star rating system as seen below:



## V. Test Plan

### I.1. Test Objectives and Schedule

The primary goal with tests in this style of project is to gain confidence that the code and processes we've developed and written will result in a consistently successful execution of the desired behavior and fulfill all necessary requirements at runtime. Automated tests serve the additional purpose of serving to safeguard the project against being broken by future updates. Since we are using the Google Maps API, we know we have at least five years of support for the code we are currently running, which means updates aren't a huge concern at this time. However, we want the future of this app to remain stable for as long as possible, which is why our automated and manual tests will accomplish both consistent successful execution, and insurance of compatibility, helping to ensure that all core features are operating correctly each time that we, or a future maintainer, deploy a new update to the app. We acknowledge these kinds of tests are especially important to have established as we transition into the second half of our development process next semester. This second development phase will be customer focused. In this phase we will directly interact with users and slowly roll out beta tests for the application. Beta tests like these are important for receiving feedback, which we can use to rapidly iterate on our build. Making sure our tests are of high quality now, will ensure reliable iterations during these beta tests.

Because the project is using the react framework, we will need to adopt a very specific set of testing technologies. For unit testing we will have to use the React Testing Library, a way to "write maintainable tests for your React components" that integrates with the React framework [1]. We can use this framework to develop some isolated level of integration testing since it allows us to "test React components without relying on their implementation details", meaning we can test that certain components will work before integrating them into our code [1]. Due to the heavy use of another professionally managed API, we will not be able to incorporate full integration testing. However, Google Maps API is an exceptionally well managed highly

professional and well tested API whose functions can be relied on thanks to the efforts of the Google team.

Manual testing procedures will need to be created and documented. We predict a lot of it will be reliant on the browsers console log, allowing us to print certain things within that form of output, where we can manually view what is going on while we are creating or modifying our code. This form of testing will be conducted during our acceptance testing process and will ensure that our product continues to perform all expected services and functionalities as they were designed. Beta testing will allow us to gauge how users interact with certain systems, leading to rapid build iterations based on the feedback received from their experiences.

Our testing process will deliver 3 major deliverables. The first is a suite of unit tests covering all non-trivial non-UI scripts produced when creating the functionality of the application. The

second is a document containing the testing procedure including instructions needed to operate the tests as well as expected results for all manual functional tests and playtests. The final deliverable is a GitHub CI pipeline that will allow for more efficient deployment of future iterations. We will do this using GitHub Actions to assist us with deployment to development platforms.

## **I.2. Scope**

This document discusses our plans for how we intend to test and create test material for Walk the Hills. It covers how we plan to implement the tests, as well as the procedures for said testing including our essential product tests. While this document does not contain every test we plan to implement, along with every procedure, we will include examples within the appendices of the document to offer a sampling of those methods.

## **I. Testing Strategy**

Project testing will be designed to create full automated tests of all core functionality of the app, running these tests through a CI pipeline for continuous integration. While sprints would allow us to easily use CD by deploying changes monthly, we believe CI will be a better practice for our team as we are currently in other classes that may pick up or slow down variably. The major components of the core functionality are the Google Maps integration, and the integration of our database. Our team has worked with our client to identify future stretch goals that may be tested, but these may be run without automated tests due to the time frame and scale of the project. Because React and Google's API automatically catch and handle many errors at runtime, errors in these components will be known about before deployment, and as a result will not compromise the overall operation of the finished system, only certain components that are still in development or being implemented. Our specific testing procedure is broken down below into 2 distinct processes. The first process describes how we develop, run and push code tested through automated unit/integration testing, as well as manual FT and acceptance testing. The second process is a preliminary outline of unit tests during playtesting of our alpha and beta phases, to be finalized later with the help of the College Hill Association and the Pullman 2040 team.

Developer Testing Process:

1. Code is written by Developer: We are not planning on using TDD frequently in this project. Our team has established it will be more beneficial for us to implement white box testing since we are using so many heavily tested implements. The details of the



implementation are more important than the direct input to output relationship tested best with black box testing.

2. Team determines test cases for code: For each core functionality, we will have at least two tests. We know users may interact with our systems in ways we may not intend, so one test will cover unexpected inputs and/or interactions ensuring satisfactory user feedback for proper use, while the other will test the expected interaction for validity. For non-core functionality only one test is expected which will test intended interactions.
3. Run tests: The team will run tests on all current functionalities and features that are present within the app.
4. Inspect test data: The team will discuss the results of the tests, and determine if code needs to be changed. If changes are needed, the specific changes will be discussed and documented before final product release.
5. Fix any code if necessary: If code needs to be changed, the developer will change it to correctly adhere to quality and testing standards, ensuring the functionality functions correctly. This will then restart the testing processes for all affected files.
6. Push code to GitHub: All changes will be pushed to a separate branch on GitHub. A branch will only be eligible for a merge if its most recent commits pass all tests run in the CI.
7. Developer makes a pull request against main: The developer makes a pull request against the main branch, prompting at least one other developer to review the code being pushed. The branch should not be merged until this code review has occurred.
8. Merge branch: Assuming the branch passes all previous CI tests and code reviews, it will be merged with the main branch.

#### Playtesting Process:

1. Create playtest build: A developer will create a build of the app they are ready to playtest. This can be a fresh build, or one specifically targeting a certain feature needing feedback.
2. Deliver build with time to test: Deliver the build to testers, with at least one week for them to test thoroughly, this time frame may be expanded depending on the number of features needing to be tested.
3. Send out feedback forms: Send out a Google form that will allow for testers to provide both positive and negative feedback. This form should be tailored to what is being tested, asking specific questions and should be remade for each build to ensure the most accurate information is received each test.
4. Collect and record feedback data: Collect the feedback after one week, collect it all in one document used to record the tests, and analyze the feedback received.

Make necessary changes: Using the feedback received, make necessary changes to functionality or user experience to move towards an improved iteration for the next playtest.

## II. Test Plans

### II.1. Unit Testing

The team will generally follow traditional unit testing procedures, however, the team will diverge in areas that are React specific. The team will be using React Test Library and Jest to deploy accurate and meaningful tests to the code. In order to ensure the code has been effectively tested, the team will be required to test all core functionalities within the app. Core functionalities can be defined as anything critical to the experience of the user such that losing them would result in the app's inability to function, such as navigational services. Additionally, the team will

evaluate the relevance and impact of all non-essential functions. If the developer feels it necessary, more unit tests can be developed for additional functionalities. For this section, we will defer to individual developer discretion when considering the extent of additional unit tests.

## **II.2. Integration Testing**

In the React Testing Overview document, it is said that “the distinction between a ‘unit’ and ‘integration’ test can be blurry” [2]. For this reason, our integration testing will mirror the Unit testing section, allowing for multiple components and functionalities to be integrated and tested.

## **II.3. System Testing**

### **II.3.1. Functional testing:**

The team’s functional testing plan is mainly reliant on manual testing implemented by developers. For this section, the team will primarily focus and refer back to the previously outlined Requirements and Specifications document, which contains developer and stakeholder expectations for project functionality each functional requirement, outlined in the document mentioned above, will be associated with one functional test. This section of the document is also subject to revision if there is an update or restructuring to any of the project's functional requirements. Given the nature of working with the React framework, these tests will be manually validated by the developers. In the case that a functional test should fail, the developer who is testing that component will provide a description of the test conditions and request the original developer to reevaluate and solve the error, documenting what went wrong and how the fix was implemented.

### **II.3.2. Performance testing:**

React has a multitude of tools built in that can help with testing, including performance tools. The team has elected to use the React Performance add on in order to analyze things like wasted time, operations, and other various measurements. This is especially important since Walk the Hills will be a cross platform app, dealing with various different kinds of performance constraints, so ensuring that we make the most of the memory or CPU usage we are allotted will be crucial in later sprints. Additionally, this tool will provide the resources for the developers to monitor the performance of the application under variable loads. In regards to the non-functional requirements specified in the Requirements and Specifications document, our team will manually test the performance of these functionalities.

### **II.3.3. User Acceptance Testing:**

In collaboration with the College Hill Association and the city of Pullman, the team will employ several playtesting iterations before the final release of the app. Our testing strategy for user acceptance testing, will be outlined below with sections for provided resources, instructions for testing groups, and plans for feedback and revision. This outline seen in Appendix A mirrors the Playtesting Process detailed above, however this description focuses more on the developer view.

#### **A. Resources**

- a. A build of the application (for each tester)
- b. A google form for feedback (for all testers)

#### **B. Instructions**

- a. A small select group of testers will be chosen for this testing iteration.

- b. Each tester in the testing group will be provided with a working build of the Walk the Hills app.
- c. A to be determined timeframe will be provided for testing.
- d. After the tester reaches the end of the testing time frame, the tester will be provided an exit google form for feedback.

C. Revision

- a. The team will receive and review all feedback forms.
- b. The team will discuss group feedback and discuss any possible modifications.
  - i. Documentation will be provided based on any modifications or additional features that are added.

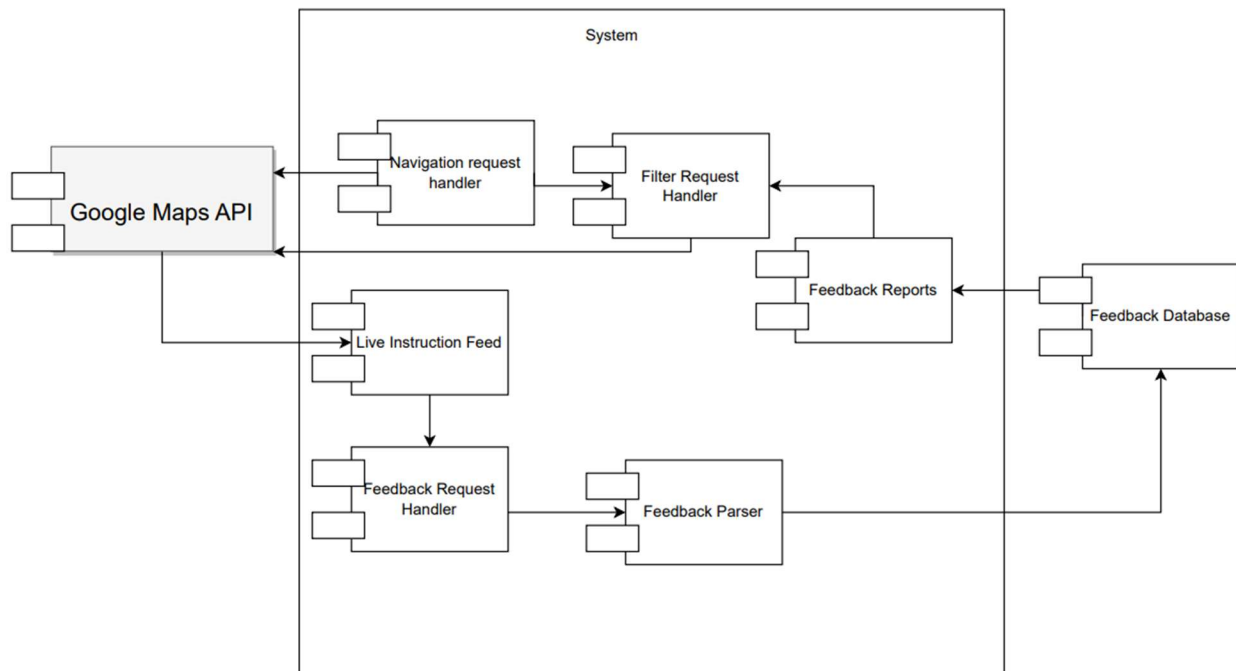
Some of the features we are hoping to include in these playtests are reporting outages and safety ratings, getting feedback on unmarked trails Google may not have, and changing routes based on situations like time of day and how well those work for our users. Our sponsors are ultimately hoping this will be an app that will be able to be involved with anything walking related. If there is a broken sidewalk or streetlamp bulb that is out, they hope the app will be able to send that information to whoever needs it. The city is also hoping to eventually expand the app to feature recommendations for things like restaurants or entertainment curated by locals, however this may be a stretch goal due to our time constraints with the app.

### **III. Environment Requirements**

Our testing environment will be contained within the React framework. While we are planning on making our app cross platform, using the tools React provides us, we will be able to stay within the React environment when testing these individual platforms both before and during deployment. As previously mentioned, React acknowledges that the difference between unit and integration tests can be blurry, as such they will be treated the same.

When it comes to hardware requirements, users will need a device that is still receiving security updates from its manufacturer. For mobile phones, this is often an indicator that a device is still being supported by it's app store. When it comes to PC's the same applies but is more based on the software manufacturer, such as Microsoft or Apple continuing to support that generation of hardware, however the same general principles apply when discusses current app updates. This will allow us to focus our efforts to keep the app updated for the correct devices.

## VI. Alpha Prototype Description



The above UML diagram is being provided again to remind us of the structure of the application. All blocks in the diagram represent major subsystems all of which have a successful prototype implementation at the application's current state. Below we will go over each portion and discuss how it is implemented and all future work that will be done to enhance its functionality.

### VI.1. Navigation Request Handler

#### VI.1.1. Functions and Interfaces Implemented

##### WALK THE HILLS

Destination:

Current Location:

Enter Destination

Enter Current Location

☐ Elevation ☐ Fastest

Open Panel

Switch to Dark Mode

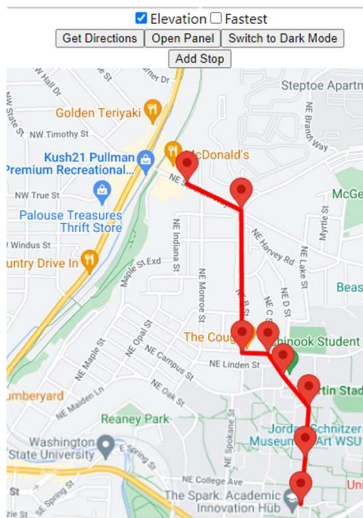
This subsystem involves the UI maintenance and logic layers that support retrieving user input direction request and returning visible poly lines on the map interface. This is the central view for our users and is in constant update and development. As of now the subsystem is fully operational but continues to change and evolve as additional features are added.

#### VI.1.2. Preliminary Tests

Given the decentralized nature of this subsystem, manual testing was performed on each component prior to integration of that component, in order to confirm that the expected behavior was consistent with the actual behavior. As specified in the testing and acceptance plans section of this report, there will be automated tests for this subsystem in the future.

## VI.2. Filter Request Handler

### VI.2.1. Functions and Interfaces Implemented



This subsystem has been, by far, the most challenging one to integrate. We have run into many issues with Google Maps legal terms of use trying to implement our own route sorting. As of now, the subsystem is functional with only elevation and shortest distance, as seen above. This two filters are a strong start, but the feature needs additional logical power to sort routes by safety rating or other factors.

### VI.2.2. Preliminary Tests

Similar to the get route handler, the filter request handler is highly UI dependent and has been undergoing manual testing. This testing can be fully automated once a concise use case has been developed. Testing both logic and UI layer together is not something that a unit test does well, but once UI changes are confirmed, scripted logic layer testing will secure all edge cases.

## VI.3. Live Instruction Feed

### VI.3.1. Functions and Interfaces Implemented

This subsystem is fully working and available on web-based application. Until mobile development begins, however, this feature will remain only preliminarily available. GPS tracking through movement does not occur on non-mobile devices and thusly cannot be worked on nor tested.

### VI.3.2. Preliminary Tests

Since web development cannot run live gps directions this has not yet been fully tested. Manual testing of GPS fetch on web development side has been functional and will pave the way for manual testing once mobile development begins.

## VI.4. Feedback Request Handler

### VI.4.1. Functions and Interfaces Implemented

The Feedback Request handler bridges the gap between the development server and the user interface. Through this system users can communicate with the city directly by entering information into the review boxes and sending it to the database. This interface and data transmission is working and fully operational. Additional changes to the UI are being made at your request in weekly meetings.

#### **VI.4.2. Preliminary Tests**

Fully automated testing is done to ensure that data entered is data returned through this subsystem. Unit tests are made that spoof a route review and verify the server received the same information. At this time, these tests are failing due to a not fully operational comment parsers (see next subsystem).

### **VI.5. Feedback Parser**

#### **VI.5.1. Functions and Interfaces Implemented**

In order to communicate effectively with the database, the feedback parser interprets user input and returns information formatted as the database expects. This subsystem also communicates with the database directly and requires web service traffic protocols to be implemented. Those protocols are still in development. As of now the subsystem only works on local host devices. Future development will open this communication to global web access.

#### **VI.5.2. Preliminary Tests**

At this time this system only undergoes manual testing as its full functionality has not yet been integrated. This manual testing involves verifying server connection and communication via terminal view interactions on a local device running both the application and server simultaneously.

### **VI.6. Feedback Database**

#### **VI.6.1. Functions and Interfaces Implemented**

The Feedback Database is a backend server which hosts the logical structure of our filtering and feedback repository reporting. Here all data collected is stored and managed. This subsystem is fully operational and can perform some sorting for route filtering, but needs a strong set of initial data to make its results more robust. Additionally, this subsystem is responsible for all server hosting responsibilities. These responsibilities are working in full on local devices only. Future development will mean opening the server hosting to web protocols.

#### **VI.6.2. Preliminary Tests**

Fully automated testing is done to ensure that data entered is data returned through this subsystem. Unit tests are made that spoof a route review and verify the server received the same information. At this time, these tests are failing due to a not fully operational comment parsers (see next subsystem).

### **VI.7. Feedback Reports**

#### **VI.7.1. Functions and Interfaces Implemented**

The Feedback Reports subsystem is the serverside integration showing all data returned by the database in a user friendly manner. This section has not yet been fully developed as we are awaiting user specified formatting materials to ensure all reports are structured in a useful and desirable method. Until then, a simple report can be generated.

#### **VI.7.2. Preliminary Tests**

Fully automated testing is done to ensure that data entered is data returned through this subsystem. Unit tests are made that report all initial test data entered into the server. These tests are fully passing as well. Additional tests will be made once formatting information is received.

## **VII. Testy Results**

Generate walking instructions.

Description:

This is a series of manual test cases for generating walking instruction in a visual and text based format.

assumptions

- The application has been launched without any prior autofill data entered.

Test ID	Expected Inputs	Expected Outputs	Actual Outputs	Pass/Fail
m1	Click get route no fill data	No action taken, no route displayed. Allows more input	No action taken, no route displayed. Allows more input	Pass
m2	The sp[tab]	Autofill generates both the spark innovation hub data in source and the cogue bar	Both addresses fill correctly and expectedly with Pullman default	Pass

	[tab] the co[tab]	address information in destination, no route is generated.	locations assumed, no route is generated as no additional information is selected.	
m3	The sp[tab] [tab] the co[tab]. Fastest filter selected. Get Route selected	A route from the spark to the cogue is generated that moves in the shortest possible distance using valid walking paths. Such a path should, unless construction exists, use the main walk only paths across campus.	The shortest path walkable is projected on the map.	Pass
m4	The sp[tab] [tab] the co[tab]. Lowest elevation filter selected. Get Route selected	A route from the spark to the cogue is generated that moves in the shortest possible elevation gain using valid walking paths. Such a path should, unless construction exists, avoid main walk only paths across campus and instead go around to avoid the main campus hill.	The lowest elevation path walkable is projected on the map.	Pass
m5	The sp[tab] [tab] the co[tab]. Fastest filter selected. Get Route selected	A route from the spark to the cogue is generated that moves in the shortest possible distance using valid walking paths. Such a path should, unless construction exists, use the main walk only paths across campus. Text instructions visible if instructions pane is selected.	The shortest path walkable is projected on the map. Instructions pane no longer displays walking instructions after latest changes	Fail

## Database management

### Description:

This is a series of manual test cases for inputting information into and accessing data from the database

### assumptions

- The application has been launched in test configuration so that all database entries are submitted in the test database model with preloaded input and outputs.

Test ID	Expected Inputs	Expected Outputs	Actual Outputs	Pass/Fail
d1-d9	Data array of test	Output file: <a href="#">see sample file in GitHub repo.</a>	<a href="#">See actual file produce in GitHub repo.</a>	Pass Pass



	case inputs. <a href="#">See code file in GitHub repo:</a>	Checking, in order for: correct averages in report, correct change in day/night flag, correct duplicity of input, comma handling in comment, multiple map key inputs, double data invalidation via comma, updating rolling average score, null string, invalid syntax input cleaning.		Pass Pass Fail Pass Fail Pass Pass pass
--	---	---	--	--

## Client communication

### Description:

This is a series of manual test cases for communicating from the phone app to the server. The Section below will detail the tests for this section once team members have developed the systems needed for this interaction, for now the table is simply a placeholder.

### assumptions

- The server is launched in live configuration, the application is launched as a fresh application on 2 devices.

Test ID	Expected Inputs	Expected Outputs	Actual Outputs	Pass/Fail
C1	Hold down on "The Cougar"  After that Latitude and Longitude are collected	New Marker dropped in the map and added to the Marker Array	New marker is added to desired location and it is also added to the array of Markers	Pass
C2	Click on the new Marker that is dropped.	Review Box appears with rating check boxes and comment section	Review box appears with the expected items as well	Pass

C3	Check Morning for time of day, Safe for rating and type "Safest bar in town" comment section then hit submit	Latitude and Longitude with numbers representing the checked boxes Morning == 3 and Safe == 4 along with the Comment	The expected outputs are returned and the review box closes afterwards	Pass
C4	Allow browser to use location	Green visible marker drops on your location using it as a start point	Green visible marker drops on your location using it as a start point	Pass
C5-C11	(5)Click on Destination and (6)type The sp [downArrow] [tab].(7) Select Lowest elevation Filter selected. (8)Then hit green button (represents get directions).(9) Click on a marker, (10)select review specific path, (11)click on another	Review box appears with C2 expected output	Review box appears with C2 expected output	Pass Pass Pass Pass Pass Pass Pass

C12	Check Afternoon for time of day, Very Safe for rating and type "Safest Path for route" comment section then hit submit	Latitude and Longitude for markers that are in the specific path with numbers representing the checked boxes Afternoon == 2 and Very Safe == 5 along with the Comment	The expected outputs are returned and the review box closes afterwards	Pass
C13-C14	(13)Same inputs of C5-C9. Select Review whole route. (14) Same input as C12	Latitude and Longitude for all Markers in route with numbers representing the checked boxes Afternoon == 2 and Very Safe == 5 along with the Comment	The expected outputs are returned and the review box closes afterwards	Pass

## VIII. Project and Tools Used

Tool/library/framework	Quick note on what it was for
React	Framework for cross platform JavaScript development.
Google Maps API and all sub integration APIs	The google maps suite of application and services in order to gain autofill, polyline, map data, and more.
WinSock	Windows socket library helper for server connection

Languages Used in Project			
C	C++	JavaScript	CSS
HTML	Make		

## IX. Description of Final Prototype

### 1. Final Prototype summery

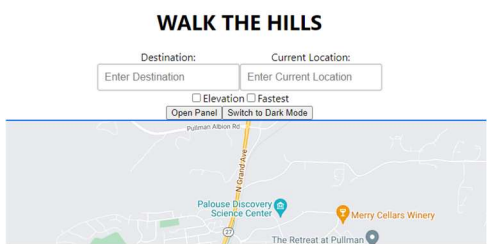
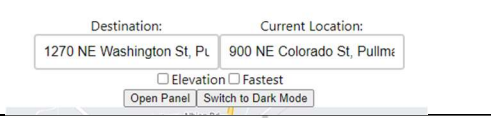
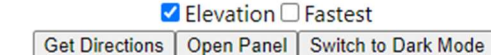
- a. Finite Cipher has built a safe walking application and data collection database for the College Hill Association and city of Pullman. This project is broken into two major pieces each with their own scope and functionalities.
  - i. A live walking application that provides filtering of walkable paths across the city of Pullman through semi-live location tracking
  - ii. A database of Pullman street usage, safety, and public opinion with a detailed reporting and monitoring output system.

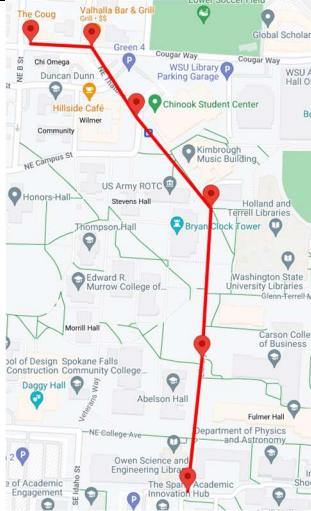
These systems communicate together to complete the final project. Users who request walking routes through the mobile application will agree to send data to the database that can then be reported on for the city to use as it monitors needs for city works projects. This interconnection will allow the application to provide two major services: walking instructions in a user-friendly environment focused on safety, and data collection for the city to help focus its funding where its citizens will get the most value.

The objective of the application is to give users a means of finding walking paths through the city and will also learn their preferences for walking about the town. These things allow Pullman to better understand the needs and wants of visitors and residents alike and make needed infrastructure changes. The primary focus of our application is to ensure that user data is collected and reported in a simple and streamlined manner.

## 2. Major Use Case

- a. Mobile Application use case:

	Step Description	UI description or image
1	The user will see the map of Pullman and be prompted for a start and end destination	
2	The user will accept auto fill address help for each prompt box	
3	The user will select a filter: Safest, lowest elevation, Most direct	

4	The user will select get directions and a route will be shown	
5	The user can optionally open text directions or begin walking the given route	The opened panel will show all text directions for the user. Currently in development.
6	The user will optionally add a stop to their destination and select update route	<div> <div>Cancel Stop</div> <div>Stop Location:</div> <div>1500 North, Glenn Terrell M</div> <div>Add</div> </div>
7	The user will complete their journey and select any pins to rate sections or the entire walk	<div> <div>Turn right at SE Nevada St Destination will be on the left</div> <div>TIME OF DAY</div> <div> <input type="checkbox"/> Morning(3) <input type="checkbox"/> Afternoon(2) <input type="checkbox"/> Evening(1) </div> <div>RATING</div> <div> <input type="checkbox"/> Very Safe(5) <input type="checkbox"/> Safe(4) <input type="checkbox"/> Somewhat Safe(3) <input type="checkbox"/> Not So Safe(2) <input type="checkbox"/> Not Safe(1) </div> <div>Leave a Review</div> <div> <div>Type Here...</div> <div>Submit</div> </div> </div>
8	The user submits their report form to the database. The database processes and logs the information (see database interaction steps next).	<div> <div>Turn right at SE Nevada St Destination will be on the left</div> <div>TIME OF DAY</div> <div> <input type="checkbox"/> Morning(3) <input checked="" type="checkbox"/> Afternoon(2) <input type="checkbox"/> Evening(1) </div> <div>RATING</div> <div> <input checked="" type="checkbox"/> Very Safe(5) <input type="checkbox"/> Safe(4) <input type="checkbox"/> Somewhat Safe(3) <input type="checkbox"/> Not So Safe(2) <input type="checkbox"/> Not Safe(1) </div> <div>Leave a Review</div> <div> <div>Fix the broken lamp post near walter hall please.</div> <div>Submit</div> </div> </div>

b. Database Application use case:

	Step Description	UI description or image
1	The database automatically collects user data from the application	No UI for this process. See above for user interactions.
2	An administrator logs into their server access through a private connection and terminal	A simple terminal application will prompt the user to provide basic credentials
3	The administrator requests a new report is generated	A report is generated as an SVG. <a href="#">See actual file produce in GitHub</a> repo.
4	The administrator is prompted if they wish to clean the comment data, reset all data, or leave all data as is.	A simple command prompt asks this question with a 1. 2. 3. Menu selection.

Most of the database applications are handled automatically with very little UI or user involvement beyond requesting a report.

The client's current request is that a web version of the map application goes live before we move to the mobile device application. As such, the final prototype will involve a mobile accessible web page as shown above instead of the original proposed mobile application.

## X. Product Delivery Status

The final product will be delivered as a web application only, not a mobile phone ported application. The first beta will go live at the end of March with a full prototype ready by April 29<sup>th</sup>. The project will be presented to Allison Munch-Rotolo of the College Hill Association to be launched on a server purchased by the College Hill Association or provided by Pullman City Hall. Allison has seen an alpha prototype and receives by weekly demonstration of the project as we continue to develop the applications.

### Project Location Information:

1. The latest release of the project is available under Releases in the team's GitHub: <https://github.com/WSUCptSCapstone-F23-S24/cha-mobilefullstackapp>

2. The main branch of the team's GitHub will contain a stable version of the application and database.
3. There is no physical product to be stored.

Setup instructions:

The application comes in two pieces and will have instructions for set-up detailed in the GitHub read-me. Both pieces will require a server to be launched so that clients can connect through their chosen web portal.

## **XI. Future Work**

The following tasks need to be developed in more detail before a beta test can be run.

- a) Server needs and setup must be arranged with city hall.
- b) The web page client connection service was not completed and must be made in full.

## **XII. 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.

Functional requirements: The requirements for Finite Cipher that can be tied to a software functionality implemented by the team.

Non-functional requirements: The requirements for Finite Cipher that are general, often qualitative, and not related to a specific functionality.

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

## **XIII. References**

- [1] Google Maps platform documentation | maps javascript API | google for developers, <https://developers.google.com/maps/documentation/javascript> (accessed Sep. 03, 2023).
- [2] S. Koundinya, "Google maps vs. Waze: Which is the Best Navigation App," Guiding Tech, <https://www.guidingtech.com/google-maps-vs-waze/#:~:text=Waze%3A%20Faster%20Routes%20and%20Traffic,on%20driving%20and%20riding%20directions> (accessed Sep. 01, 2023).
- [3] "Performance tools," React, <https://legacy.reactjs.org/docs/perf.html> (accessed Oct. 26, 2023).
- [4] "Jest," Jest RSS, <https://jestjs.io/> (accessed Oct. 26, 2023).
- [5] "Testing overview," React, <https://legacy.reactjs.org/docs/testing.html> (accessed Oct. 26, 2023).

[6] Testing-Library, "Testing-library/react-testing-library: simple and complete react DOM testing utilities that encourage good testing practices.," GitHub, <https://github.com/testing-library/react-testing-library> (accessed Oct. 26, 2023).