Chad William Young Foundation: Bicycle Hazard Identification Project

Final Report
Colorado School of Mines
Computer Science - Field Session
Safety Cyclers

Team Members
Russell Berkley,
Ankit Dhingra,
Jared Schneider,
Ben Ferrick

Introduction:

The Chad William Young Foundation was established by Chad's father, mother, and brother, Kevin, Lois, and Evan Young. The Foundation was created after Chad William Young passed due to a traumatic brain injury sustained during the Tour of the Gila professional cycling race. The Foundation was formed in part to help provide support, direction, and funding for developments centering on preventing injuries to cyclists, especially traumatic brain injuries. In particular, the Foundation is interested in developing a technological solution to cyclist safety that aims to prevent crashes, rather than lessen their effects. This is important, because many of the most severe cyclist injuries, including Chad's, are not preventable with physical safety devices like helmets.

This project was previously worked on by a team of Mechanical Engineering students from the Colorado School of Mines as a Senior Design project. They did much of the groundwork for the project primarily including collecting information from cyclists and the development of an system of equations to calculate a hazard plot for a given route. The state of the previous project's code was not user friendly, so it was determined that it must be modified for more common use. The project was turned into a computer science field session project in order for the algorithm to be more accessible for users. Therefore the goal for our team was to take their equations and hazard plots, and automate them. We took steps to make a user interface in the form of an app, to make the user experience as friendly as possible.

Our data comes from the trail app Strava. Strava is primarily used by cyclists and runners. It collects data like speed, elevation, location, time, heart rate, and power. Fortunately Strava also has an API that allows us to get location, elevation, and time data for a given route. This is currently the only source of data used for this project at this time. In the future we hope to include other data to supplement this.

Project Requirements:

The given algorithm had hard coded values for input data and data collected manually by the last team. For this reason the algorithm was not friendly to new users. Our job was to take that algorithm and make it more user friendly by cleaning up the code and bringing it to computer science standards, removing hardcoded values and manual data, and automating the collection of data for the algorithm through the Strava API. We were also looking to improve on the algorithm where possible. It relies on the best recorded rider for that trail, not an average of all riders. A secondary goal for our project was to take the starting algorithm and improve it by using averages and reasonably accurate data, and putting it into some kind of user interface that will make the algorithm more useful.

Functional Requirements:

- Demonstrative Android application with accessible UI.
- Collect Strava data, average it, and feed it to the hazard index algorithm.
- Translate algorithm to Java for user accessibility.
- Create plotting methods for Java, to provide users with an easy to understand plot of hazards.

Non-Functional Requirements:

- The UI should be an Android App coded to work with Java.
- Make use of Strava API.
- Clean up current algorithm including removing unnecessary data, combining loops, and general refactoring.

- **Extended Goals:**
    - **Features:**
        - Push alert notifications as cyclists approach hazards.
        - Complete integration of app and algorithm.
        - Allow users to generate the manual data from the old algorithm using the app.
        - Difficulty Ratings for Trails Based on Hazards.
        - Non-local database for the app.
        - Map data onto google maps to show actual locations of hazards.
    - **Additional Work:**
        - Gain access to Strava Metro, to get additional data to improve the algorithm.
        - Communicate with Strava for potential integration, this would allow for crowd sourced data for further improvement and ease of access of the algorithm.

System Architecture:

The algorithm given to us by the client was written in Matlab, uses hard coded values, makes use of Strava API but no longer works as the API has been updated for security, and only works for a single route being Lookout Mountain. The Strava API will be explained in more detail later. The hard coded values included things such as longitude, latitude, altitude, time, and distance. These are all things that need to be automated. We were able to salvage the equations and general structure of the original algorithm and created a new algorithm that didn't use the hard coded values. We got the API working and have access to data and we created an app to display the data. The flow of the project is detailed in Figure 2.

Components:

- The app - The flow of the app is seen in Figure 1. The main page of the app holds all the routes that have been plotted. Users are able to select a route and be taken to the corresponding route page. Additional pages include a page to add routes and a page to mark specific hazard locations on a specific route, this page is used to reincorporate the manual data removed at this stage of the project.
- The API - This component of the app was to automate the collection of data from the biking/trail app Strava. This data is then fed into the algorithm. Due to how the API works, automating this data collection is complex.
    - The Strava API is a closed off HTTPs request system that uses OAuth2, a security authentication system that protects a user's private data. This means that the only data that can be publicly accessed is data not tied to individual users, and accessing this data requires an access token from a consenting user.
- The hazard algorithm - This component is a translated and cleaned up version of the code provided by the last team to work on this project. Restructured to be more efficient and accessible to future developers.
- The plotting code - This component is the part of the algorithm that creates a visual representation of the hazard plot generated by the algorithm. This part was separated into its own component due to Java's lack of support for plotting. It uses a colormap of hazard values by longitude and latitude.

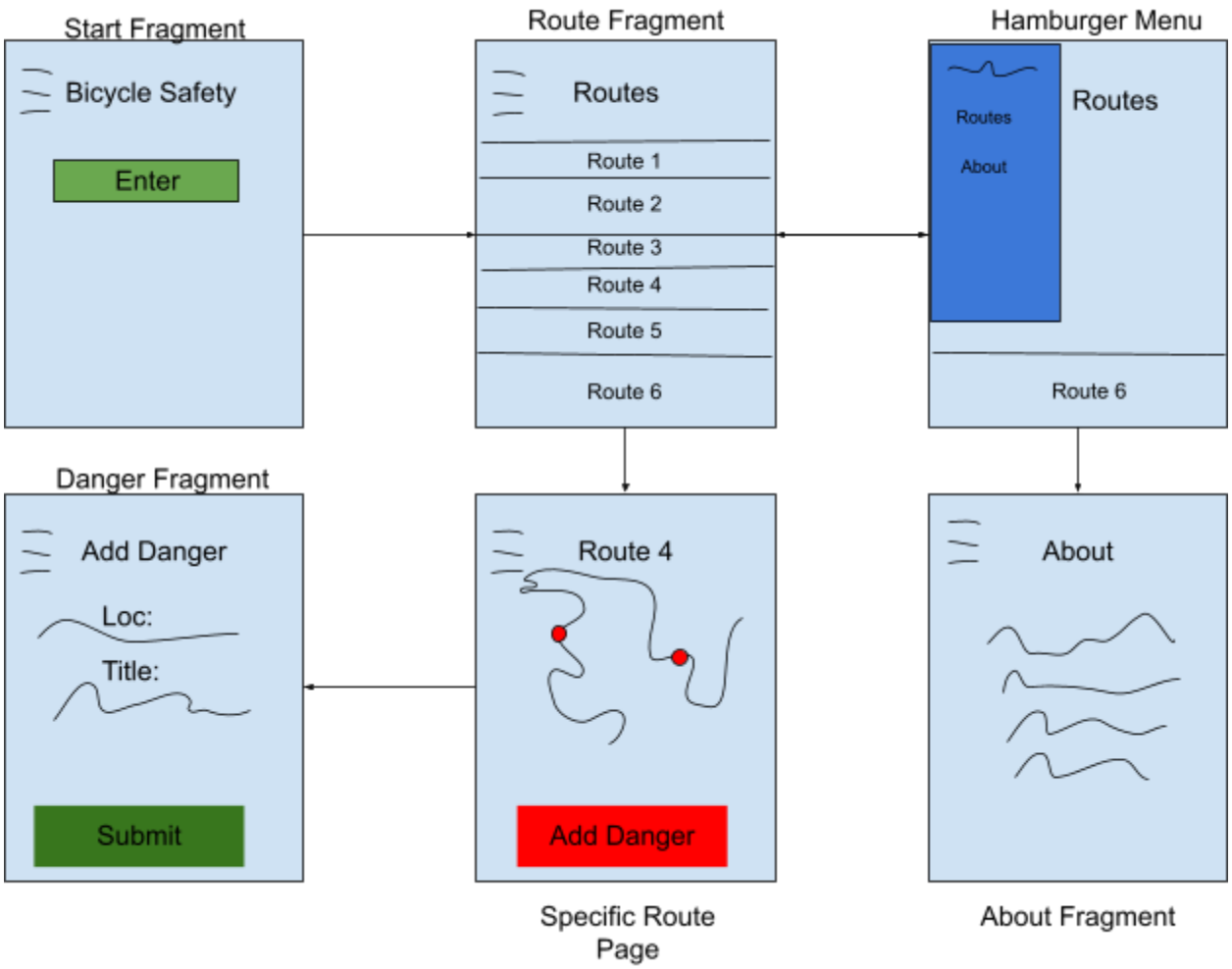

Figure 1. *Mobile Application StoryBoard*

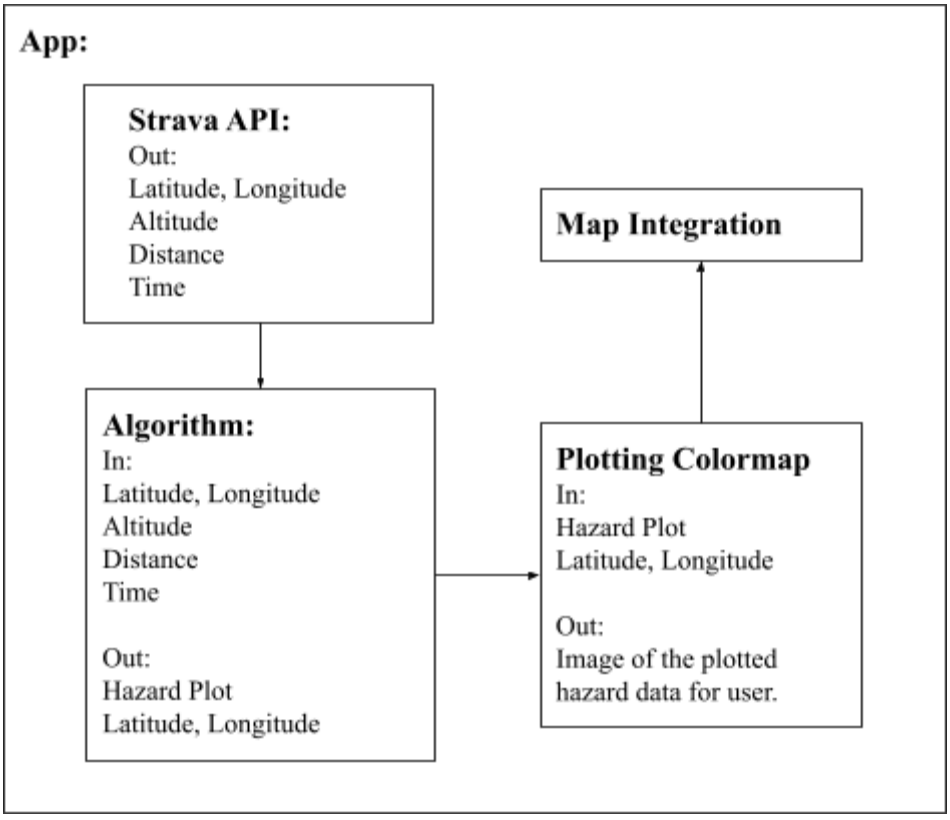

Figure 2. *Project Layout*

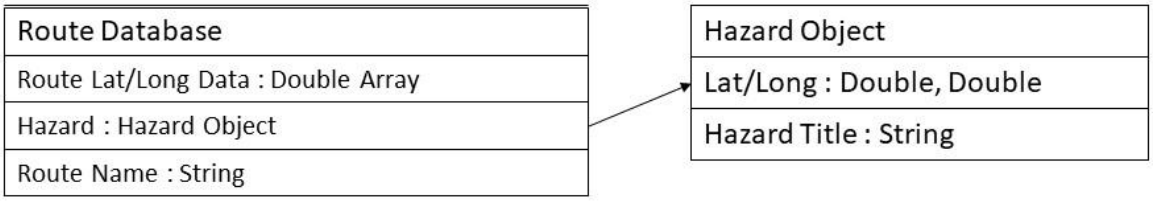

Figure 3. *Database Schema*

Technical Design:

The mobile application was one of the more interesting aspects of our project. The mobile application was thought of as a way to put our work together in a tangible form for our client. We decided the best way to do this was a mobile application that bikers could use as they're riding. Currently, there is a list of routes the user could choose from, and that will lead them to map with the route and all the dangers plotted with markers on the map. A lot of the times, terrain and routes change however. Because of this, we wanted more features to add flexibility to our application. The application was a good lesson for us because it was something we decided to do as part of our definition of done without realizing the scope of that story. There were already many obstacles with data collection and algorithm translation. Along with the fact that not many of us knew about android development, this story became more and more impossible. In the end, we decided to at least provide a framework for future teams to develop upon. *(See Figure 4)*.

Mobile application development turned out to be much more difficult than first expected. There is an Activity that runs the application, where all the data necessary to start the app is gathered. Next, the different screens are called fragments and they all must be connected to each other in some way. This involves the override methods like onCreate() and onCreateView() that set up a ViewModel and binding so that the fragments can be connected to the MainActivity. This is all simply just to make the application navigable without any functionality. The next step would be to merge the algorithm into the application, but in order for that to work, we needed the data collection and algorithm to be fully working and translated. Our priority lay there and once we got it working, due to time constraints, we were unable to work on merging the two together.

A google maps view was also implemented into the home screen using the google maps API. Our idea was to show the routes on this map when selected. The map will show the route as well as markers along the route that describe the type of danger and specific location. The routes would be stored in a database. The database would store Long/Lat data obtained from the data collection to plot out the route, a list of hazard objects that corresponds to the markers that show dangers, and the name of the route for lookup when the route is selected. The hazard object would store the Longitude and Latitude position of where it should be placed on the map as well as a string that tells the user what kind of danger it is.

There is also a FAB button *(see Figure 4 in the bottom right corner)* that would lead to a new screen to add a route to the existing list of routes if it didn't exist already. Since routes always change and new ones could be discovered, it made sense to give the user the ability to add a new one. Lastly, we have the hamburger menu that shows the application name, the foundation, a possible logo, and buttons that are used to move between the fragments. The about page gives information about the Foundation and the application and the Routes page gives a list of routes the user can choose from. If a user selects a route, the fragment switches to the home screen to then show the route. The best part of the application and why it's so interesting, is that

we enabled future groups to continuously work on this and add new features as time comes. Some new features could be adding and deleting hazards and push notifications to let the user know they are near a hazard and should be more aware of their surroundings. It'd also be great if the application gave tips on how to be a safe biker. This application can really become something useful that bikers use daily to plan out their day and be safe.

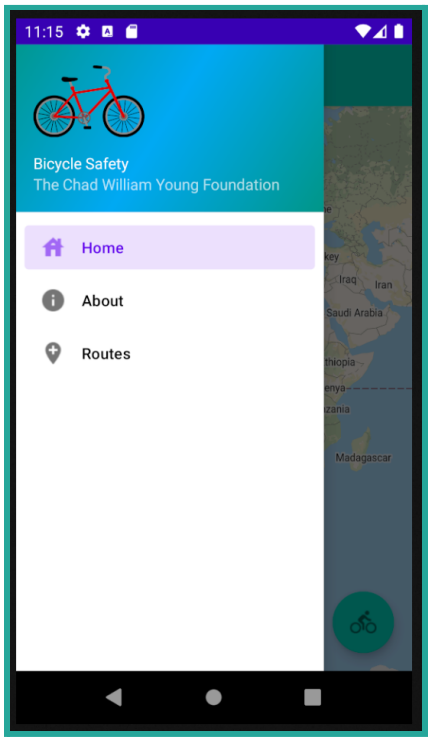

Figure 4. *Mobile Application Layout*

The Mobile App interfaces with a background data search and collection system. Utilizing the Strava API, the data search utilizes HTTP requests in order to get the proper information from Strava. This includes an access token to create proper requests from the API, a search request which returns data about routes within a nearby latitude and longitude, and the specific latitude, longitude, distance, and elevation of those routes. All data received from the Strava API is in JSON format, so once it is retrieved, it needs to be parsed into a more readable format for other programs.

First, Strava requires its API users to input an access token in order to tell Strava that they are allowed to view data (this is to ensure that no one can view personal information without clear consent of the user). The access token is reset every 24 hours, so we need to use a constant refresh token in order to get the current access token for all data collection.

It is worth noting that at this time, we use the access and refresh token of one of our team members on Strava. For the final product, it would be recommended to get the refresh token and

access token of the current user, or a dummy account that no one would wish to store personal information on.

We use an HTTPs POST request with a specific link in order to get the access token. From there, Strava has a built in search function where you can find the nearest 10 cycling routes to a specific latitude and longitude that meet a specified grade criteria. With those values inputted (we used the latitude and longitude of Golden to test, but it is possible to use any latitude, longitude, and difficulty criteria) we can collect the information of 10 routes. All information is given to us in JSON format, meaning that once the information is collected, it needs to be sorted out of JSON parsing and into useful Java Data Structures. The last bit of the data collection algorithm converts the given JSON data into a series of Array Lists containing distance, latitude and longitude, and elevation data on the given route.

From there, the data is plugged into the algorithm, which has been refactored into Java and made more efficient by our team (details to come), and we can obtain a heat map of potential dangerous areas on the trail from that data (also refactored from the original algorithm). The original algorithm received heavy modifications due to the difference in language, and also to improve the computational efficiency of the algorithm.

The algorithm given to us by the old team was not written to common computer science standards. There were many unexplained hard-coded arrays of values and many loops that were not computationally efficient, but the underlying math that the mechanical engineering team came up with was solid. As we translated their code from Matlab to Java we were able to determine which of the 30 some hard-coded arrays were actually used and which were not. We were also able to increase the spatial efficiency of the algorithm by properly pre-allocating space for the arrays used. A large amount of cleanup work was able to be done, and the algorithm is in a much better place.

The old algorithm also had access to extra data that our team was unable to automate. It was data collected manually by their team. It corresponded to the locations of things like potholes, blind turns, and heavy traffic. Unfortunately this type of data is not something that is widely available, so we were not able to incorporate it. To make up for that data, in the future we planned to have the algorithm integrated into an app. Using the app's sensors we could have users enter in the same data collected manually by the previous team.

The algorithm gives a hazard level for each Latitude/Longitude coordinate given to it. It does so using 4 different factors; velocity, acceleration, grade, and radius of turns. These must all be calculated using latitude and longitude, distance, time, and altitude.

Velocity Calculations

$$V_i = \frac{dist_i - dist_{i-1}}{time_i - time_{i-1}}$$

Where: V_i = Velocity at time index i [m/s]

$dist_i$ = distance value at time index i [m]

$dist_{i-1}$ = distance value immediately prior to time index i [m]

$time_i$ = time value at time index i [s]

$time_{i-1}$ = time value immediately prior to time index i [s]

Acceleration Calculations - (V_{avg} is simply an average of the 3 surrounding velocity values)

$$Acc_i = \frac{V_{avg,\,i} - V_{avg,\,i-1}}{time_i - time_{i-1}}$$

Where: Acc_i = Acceleration at time index i [m/s^2]

$V_{avg,\,i}$ = Average velocity value at time index i [m/s]

$V_{avg,\,i-1}$ = Average velocity value immediately prior to time index i [m/s]

$time_i$ = time value at time index i [s]

$time_{i-1}$ = time value immediately prior to time index [s]

Grade Calculations

$$G_i = \frac{alt_{i+2} - alt_{i-2}}{dist_{i+2} - dist_{i-2}} \cdot 100\%$$

Where: G_i = Percent grade at time index i

alt_{i+2} = Altitude value at two time periods after time index i

alt_{i-2} = Altitude value at two time periods before time index i

$dist_{i+2}$ = Distance value at two time periods after time index i

$dist_{i-2}$ = Distance value at two time periods before time index i

Radius Calculations

$$r = \frac{L_{1-2} \cdot L_{2-3} \cdot L_{1-3}}{4 * A}$$

Where: r = Radius of curvature [m]

L_{1-2} = Linear distance between the first and second points [m]

L_{2-3} = Linear distance between second and third points [m]

L_{1-3} = Linear distance between first and third points [m]

A = Area of the triangle formed by all three points [m^2]

Linear distance between points was calculated using the Haversine function

```
// distance between latitudes and longitudes
double dLat = Math.toRadians(lat2 - lat1);
double dLon = Math.toRadians(lon2 - lon1);

// convert to radians
lat1 = Math.toRadians(lat1);
lat2 = Math.toRadians(lat2);

// apply formula
double a = Math.pow(Math.sin(dLat / 2), 2) + Math.pow(Math.sin(dLon / 2), 2) * Math.cos(lat1) * Math.cos(lat2);
double rad = 6371;
double c = 2 * Math.asin(Math.sqrt(a));
return Math.abs(rad * c) * 1000;
```

Where: rad = the radius of the earth in kilometers

dlat / dlong = longitudes and latitudes converted to radians

Quality Assurance Plan:

To ensure a satisfactory product is given to our client we developed a quality assurance plan. This plan included testing for the algorithm, data collection, code review, and user acceptance testing.

Code Reviews - Code reviews have been conducted for both the main algorithm and data collection software. These reviews help our product to be accessible as possible for future groups working on this project. This is vital if our code is going to be taken to Strava to be integrated. If our code is well documented and easy to read it is more likely to be accepted by a company like Strava. In addition it is likely that another group from Mines will work on this project in the future, and we want them to have a better experience testing and modifying the algorithm than we have.

Unit Testing - We used unit testing whenever possible in order to test if our software was working or not just for regular cases but also edge cases. Unit testing helped guarantee that individual components of code are fully working and without errors before pushing them to the larger codebase.

Security Testing - Since we pulled data from external sources we ensured that we aren't receiving any personal user data, using that data, and giving out that data. This was done within JUnit Testing to make sure the data received was the data we needed and nothing more.

Deployment Testing - We ensured that the product is automated and correctly working and that the application is fully navigable. We also ensured that these pieces can work together once merged by a future team and can be effectively used by our users.

User Acceptance Testing - Since the original purpose of this project was to make the pre-developed algorithm more accessible for users, user acceptance testing will be quite important. Due to limitations in sources for said testing we mainly relied on the client's notes to determine whether our user interface is up to standard.

Results:

The goal of this project was to take a previously implemented algorithm that manually plotted a trail and the hazard levels along with the trail, and to automate it and make it have a lower space and time complexity. This was done by using the Strava API and data collection algorithms as well as translating the algorithm into a more usable language for future groups. Along with this, a framework application was developed so that the algorithm could be implemented in a GUI. The framework is a mobile application that implements a google maps API as well as an about page, and page for routes. The idea is that a user could select a route, see the route on a map, and all the dangers currently ongoing along that route. Seeing a route on the

map and the dangers would all be functionality provided by the algorithm. Combining the application and the algorithm did not make it into the final product due to time constraints.

List of features not implemented

There are multiple ideas and features that were not implemented due to time constraints.

- Merge Algorithm with Application
- Add danger features
- Database to store routes
- Improve algorithm even further with Strava Metro
- Add push notifications for user when in or near danger

Performance testing results

Multiple tests were performed with the data collection and mobile application. The mobile application was tested with an Android Phone and Emulator. The results showed that the app was fully navigable and the google maps API worked correctly. The UI fully works but will still need to be constantly tested upon when more functionality is added. The data collection tests were performed and successful in that it was able to get the 10 most relative routes to a search and correctly parses the data into formats us and other teams can use. The Algorithm also has been tested and works correctly, along with the fact that the algorithm was previously worked on and tested by a previous group, we can ensure that this works correctly.

User acceptance testing

Since the work has not fully been completed by the end of the Field Session, the team was unable to utilize User Acceptance Testing. While we are confident that everything currently works, more testing here would be required when more functionality is added and the project is in a more ready to go state. In a way, our team really cleaned up this project so that future teams can get a lot of work done. In terms of what our testing has proven: We have successfully added fully application navigation. The algorithm is fully automated and requires no manual data. (Unless of course more information is to be added later), no private user data has been received or leaked out, The data collection from Strava was successful, and we were able to improve the space and time complexity.

Lessons learned

This group learned a lot, not only of technical skills, but of social skills as well. We really improved our communication skills and ability to adapt to each other's time availability and other issues. We used pair programming so that everyone has an eye on what each other is doing so we can all provide insight on all parts of the project. We also learned proper email etiquettes. We really learned to work as a well oiled machine. Agile ended up working really well as there was no waiting around/waiting for others to complete their task to move on as seen in sequential methods of work. Everyone had something they worked on all the time. It kept our client and

teammates constantly involved and in the know of what's going on. In terms of technical skills, there are a lot of things all of us learned.This includes, MatLab language and to translate code into other languages efficiently, what an API is and how to use one effectively, how to find alternatives to problems we come across, plotting systems with Java, mobile application development, usage of postman and other data collection techniques, and much more.

Future Work:

There are many things to add on to this project. Some stories add proper functionality to the application while others are extras. The biggest thing to do next is to merge the algorithm and all other code into the mobile application. The algorithm and other code was translated into Java. Android Studio (the program used to make the mobile application) codes in both Kotlin and Java so no more code translation should be necessary. Since both are in the same language, with some knowledge of how mobile applications are created, this shouldn't be too hard to do, especially with the setup given.

Next, if Strava Metro is available to you, we recommend looking at the data they are able to provide to better enhance the algorithm. Strava Metro is a partnership with Strava where if applied to and accepted, Strava will allow you access to the data of transport information of other users without any user privacy conflicts. With our automated implementation, it is clear that Strava's API was meant to only work with your own data and not other users' data. This can be seen with the 2019 OAuth update Strava did. Unfortunately, because of this, the data we receive from Strava is the segment data other users can put out publicly. This might not always be effective as the data we can get from segments is limited and may not even cover an entire route. With Strava Metro, these issues can go away as the data will be readily available to us. In terms of difficulty, we cannot accurately state how hard this will be. This all depends on what Strava Metro gives and how it is formatted. Depending on how Strava gives out their data, the algorithm might have to change drastically or not very much at all.

The next few things are general improvements for the project. The mobile application can have a variety of features to help the user stay safe while biking. Things like including an add danger button that allows the user to add a danger marker to the route at their current location as something new can come up. Likewise, we'd like the user to do the opposite if a danger on a route disappears. We'd also like push notifications to be implemented so that if a user is near a danger on a route, they can hear the ring and know to be more aware of their surroundings and be on the lookout for any dangers. All these additions vary in difficulty depending on how the merging of the algorithm and application go. Hopefully, it shouldn't be too bad at all.

There are many other quality of life updates that can be added to every part of the project. Once this has all concluded, all components of the initial project can be met. One of the last things that could be added to this project is combining the algorithm and data collection with an Apple mobile application. Currently, the framework was provided in Android using Android Studio as it was more readily available and easier to work with than IOS. The data collection has been automated by us and we have done some Real Time Feedback as well as set up future groups with a plethora of things to do so that Real Time Feedback and Additional Data Incorporation can be improved and added onto later.

Lastly, while this may go above and beyond Field Session, getting contact with Strava as well as the Department of Safety and Transportation could prove to be very resourceful. Strava could help implement this project into their own systems if this project was shown as a proof of concept. The Department of Safety and Transportation already knows about this project and we

are sure they would love to help in any way they can especially if the mobile application becomes easily functional.

APPENDICES

GitHub of All Code and Documentation: https://github.com/adhingr/BicycleSafety

Step by Step Installation Guide:

1. Click the GitHub Link, and find the green "Code" button. Click on it and then click on the clipboard icon to copy the HTTPS
2. On your desktop or wherever you would like this project to be, create a folder with a relevant name.
3. Either using command prompt or Git Bash, locate and go to the location of the folder you have made
4. Type "git clone 'INSERT_HTTPS_HERE' and paste what was copied in step 1 into 'INSERT_HTTPS_HERE'
5. After running this command, the project should be stored in this folder to be viewed. Bicycle_Safety is the code for the mobile application framework and Field-Session-2021 is the rest of the code for data collection, algorithm, and mapping
    - You can open Field-Session-2021 by using an integrated development environment like Eclipse.
    - You can open Bicycle_Safety by using Android Studio

Important Information about Strava and OAuth2 Access Tokens:
Strava's API uses OAuth2 to protect client/user information. Currently, the above project uses the refresh token and access token of one of the developers, since no personal information is currently used by the project. In the future, it would be better to adapt this work to use the app user's actual Strava account information, so that then the app could make requests for personal data.

In the java project, the class "TokenGetter" is responsible for making the HTTPs requests that get the token. In order to get an access token from Strava, the request URL should look something like this format:

```
<https://www.strava.com/api/v3/oauth/token?client_id=66572&client_secret=afc6f0
19f7c3eb4e5e0233471b2c8c4c0ae59a58&grant_type=refresh_token&refresh_token=1eb01
26800cea2fe60e83b769ab3c9ac87259a53>
```

Where:
client_id: The client (aka user's) id number
client_secret: a hex code generated for the user
refresh_token: a hex code that is the user's refresh token

This will return a JSON array containing the current access token, refresh token, and other client data. It is recommended that future projects use updated client ids and refresh tokens for the individual user, or while in development, developers use their own Strava accounts. In future use, the existing developer's account may not be available for data collection. Additionally, the login information of this existing account is not being provided because it is a student's Strava account.

You can use this link while logged in with a Strava Account to create your own API app to access the data: https://www.strava.com/settings/api