COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report:
Chad William Young Foundation

Danger Dodgers

Zoe Baker
Ethan Perry
Cole Wapelhorst

Revised December 5, 2021

DANGER
DODGERS

CSCI 370 Fall 2021

Prof. Bodeau

Table 1: Revision history

Revision	Date	Comments
New	9/12/2021	Created document, added in requirements and team profile.
Rev – 2	9/14/2021	Added in design images.
Rev – 3	10/24/2021	Added ethics & quality assurance section.
Rev - 4	11/14/2021	Added in results working doc (project completion, future work, lessons learned).
Rev - 5	11/29/2021	Revised report to be sent for evaluation.
Rev - 6	12/2/2021	Placed logo on the front page.
Rev - 7	12/5/2021	Revised report according to feedback.
Rev - 8	12/7/2021	Finalized report before submission.

Table of Contents

I. Introduction

Chad Young was a man of many talents. Not only was he devoted to his Mechanical Engineering degree at Mines with a near perfect GPA, he was a powerful and accomplished cyclist, receiving numerous accolades at the professional level. On top of all that, Chad was a compassionate and humble leader. On April 28th, 2017, Chad unfortunately had his life cut too short as a brain injury sustained during a race ended up being fatal. Shortly after the accident, Chad's parents, Kevin and Lois Young, began the Chad William Young Foundation in an effort to minimize cycling incidents and continue Chad's legacy.

The unfortunate fact of the matter is that cycling is inherently dangerous; there is no way to guarantee a completely safe ride. Even with a helmet, a state-of-the-art bike, and any other gadgets a cyclist may have, accidents will still happen. As such, the foundation wants to create a product that will help mitigate this problem using technology. The product is an android application that identifies hazards on cycling routes in two ways. First, the app allows riders to pinpoint potentially dangerous portions of routes. Other users can view those hazards and upvote/ downvote them. The application also includes a physics-based risk algorithm that will determine the risk level of certain areas given physical attributes of the route itself (grade, turn radius, speed of cyclist, etc.).

II. Functional Requirements

The application must:

- Include a front-end heatmap which displays hazards: A heatmap overlayed over a map view of a certain route or area that highlights hazardous sections. The heatmap tunes to the current GPS location of the user, or be set to a certain radius around a user-entered address.
- Have a form that allows the user to report hazardous areas.
- Feature algorithmic hazard creation. An algorithm that can post-process cyclist routes and assign hazard ratings to points along the route.
- Ingest database to a database on the back end: User feedback is sent to a back-end database and used to update the hazard heatmap.
- Host data on a live hosted database, i.e. our data is hosted on Heroku.

III. Non-Functional Requirements

The application must:

- Build off the foundations of previous groups
- Be a complete mobile application. The front end is written in React Native, which compiles to native Android Studio.
- Use Python for the back end: The app uses FastAPI in Python for all the data processing.
- Use PostgreSQL for the database: The app uses PostgreSQL for durable storage.
- Runs smoothly and effectively on any Android (API level 25 or higher) cell phone.
- Stays under the $1000 budget allowed by the founders of the foundation.
- Setup to be released on Google Play for the general public.
- Back-end released on a public cloud for compatibility with the public app.
- Has user authentication functionality to protect user information.

IV. Risks

Technical risks:

- Location data is sensitive and privacy will be a concern. Users may not want to share their location data.
- Geographical/GIS data is very memory intensive. Creating an app that can scale to handle displaying GIS data and algorithmically processing GIS data could present issues.
- Live feedback requires location tracking, and potentially memory-intensive loads of local GIS data. Live feedback without an internet connection could also be challenging.

V. System Architecture

The backend of our application is created in Python, using the FastAPI library. We use a PostgreSQL database for durable storage of users and reported/ algorithmically detected hazards. Our algorithm and data analysis are also implemented in Python, using libraries such as NumPy and Pandas. The front end of our application is implemented in react native, which is compiled into native android.

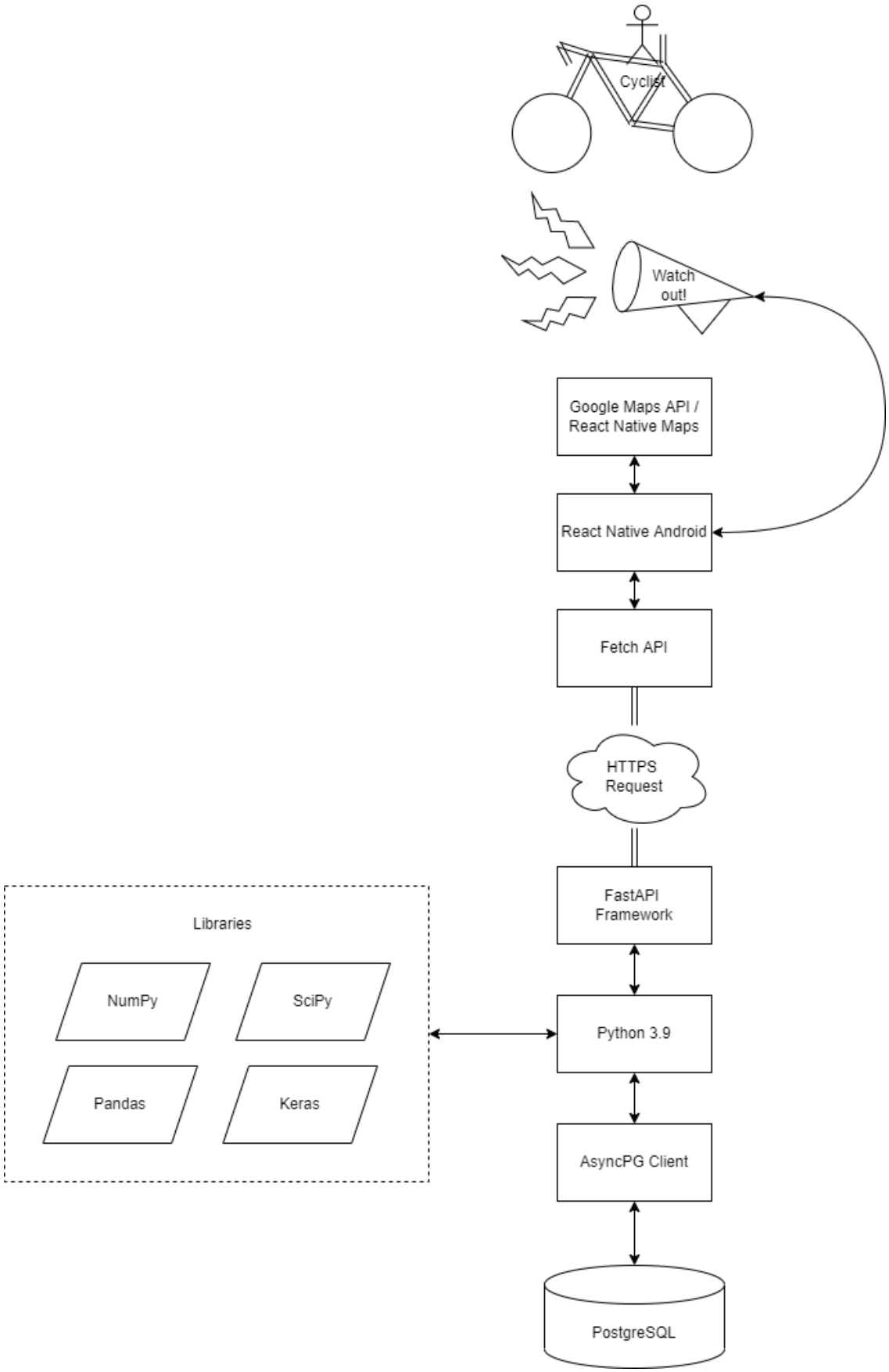


Figure 1: System architecture diagram, which shows the overall flow of our application from database to the front-end.

Our SQL database has entities for user data, hazard area data, and hazard. The relationships between these entities is explained in the Entity-Relationship Diagram (ERD) below.

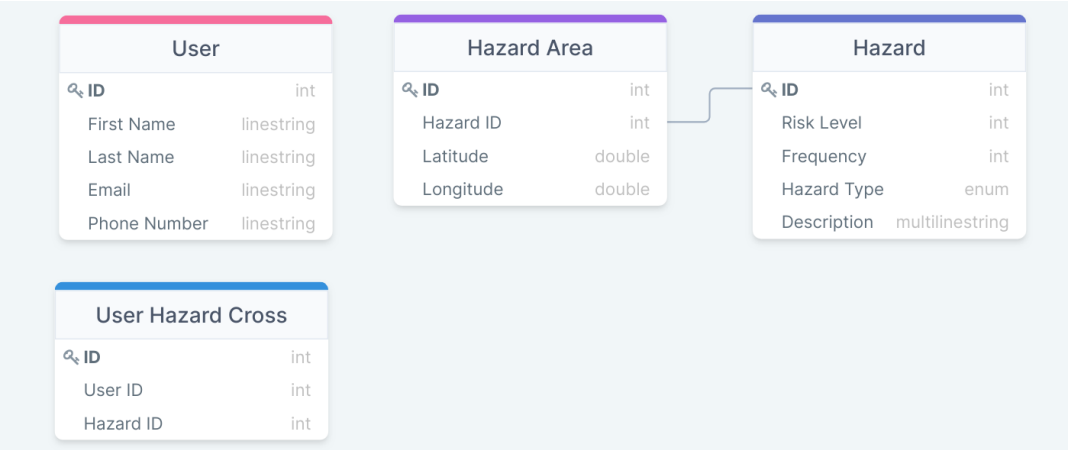


Figure 2: Entity- Relationship Diagram of our SQL database.

Finally, we created a wire-frame mockup of our mobile app user interface. These wireframes describe the central map view with different heatmap overlays, our alerting functionality, and the main menu that allows users to navigate to an about us/contribution page.

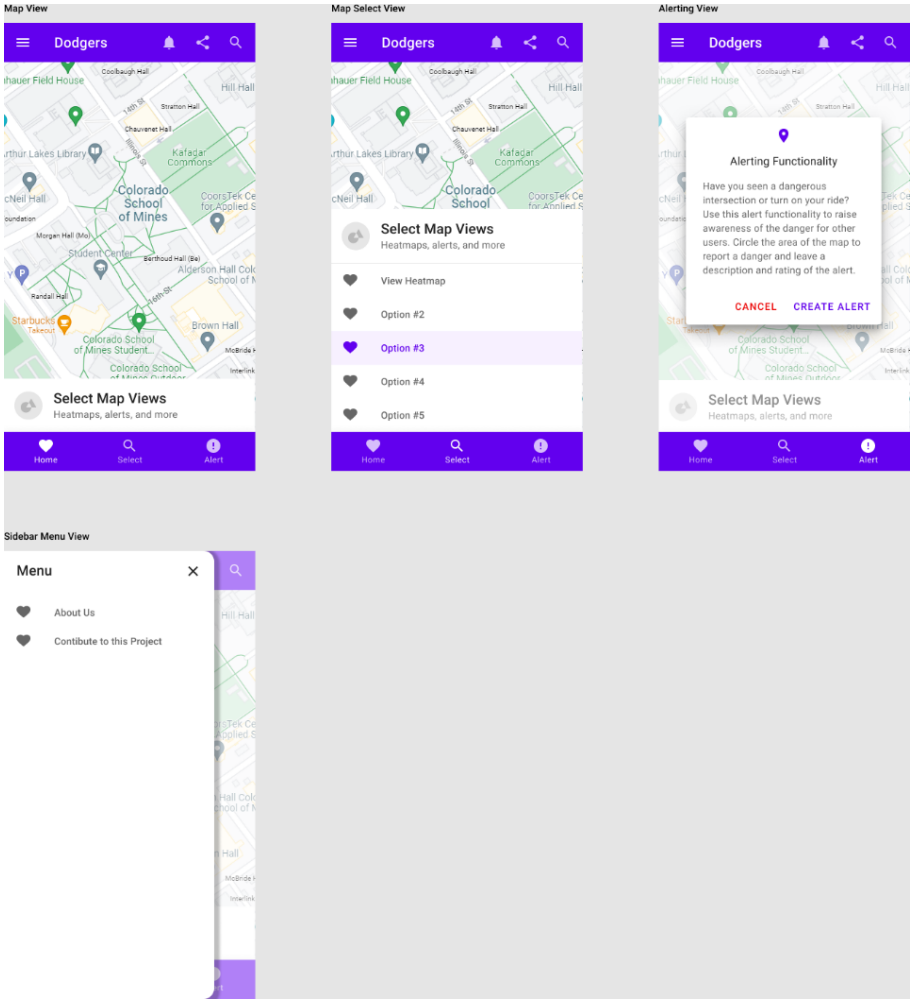


Figure 3: Initial wireframe design of the GUI of the app.

VI. Technical Design

The four main platforms used in app development are FastAPI, PostgreSQL, Docker and React.

- **Backend Application:**
  - FastAPI web framework for Python
  - Uvicorn ASGI (Asynchronous Server Gateway Interface)
    - Successor to traditional WSGI specifications.
  - AsyncPG database client for asynchronous queries
  - PostgreSQL database

We chose the asynchronous option because we observed the performance benefits over the traditional Web Server Gateway Interface (WSGI). There is a greater SQL query and server request throughput.

- **Frontend Application:**
  - React Native Android
  - React Native Maps
    - Provides a wrapper to leverage google maps API

In addition, for the client server interaction, the server implements the stateless RESTful API paradigm. At a high level, the client requests the information using a provided JWT token, and from there, the server decodes the token and provides the resource if valid.

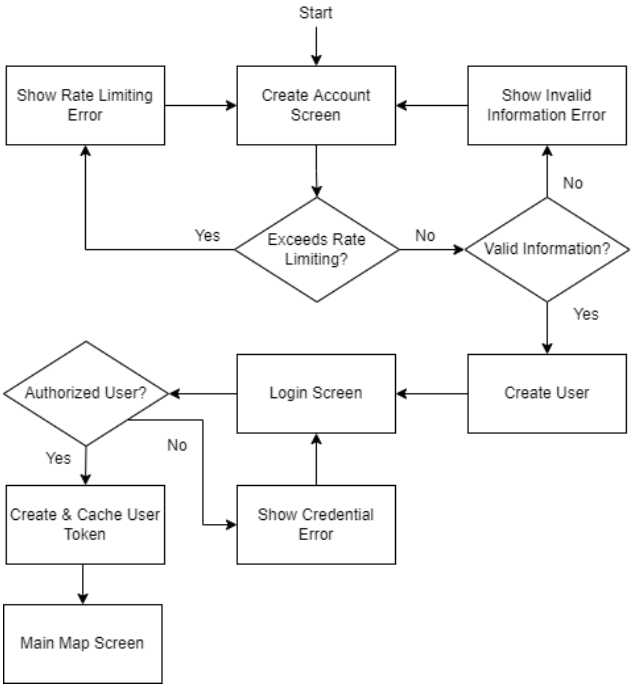


Figure 4: User authentication flow.

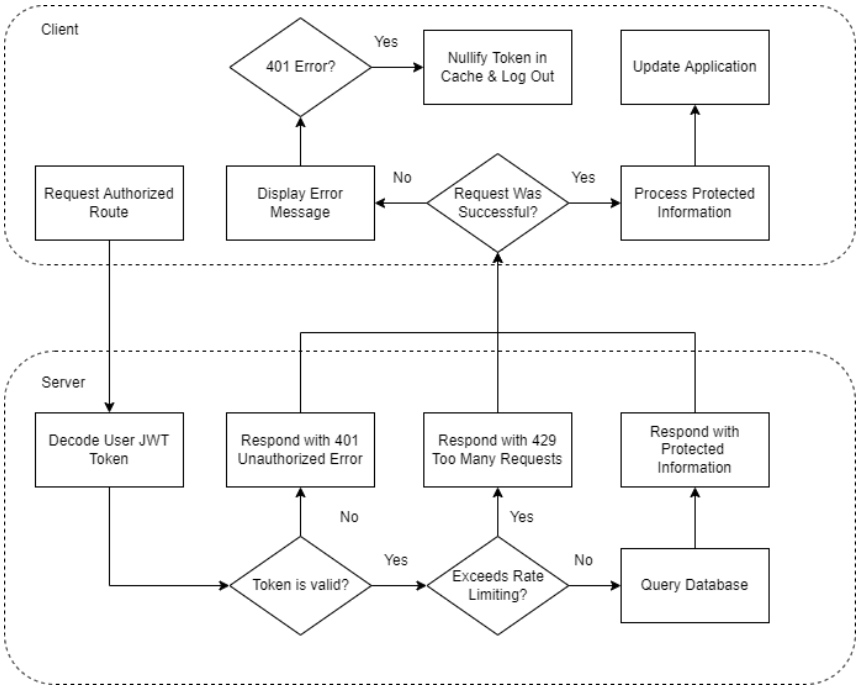


Figure 5: User authorization flow.

- **Infrastructure:**
    - Docker & Docker-Compose

    - GitHub as a repository
- **Development Environment:**
    - WatchGod Hot-Reloading for FastAPI
    - React Native Dev Server
    - Android device bridge for local device development
- **Deployment:**
    - Heroku CLI for automated deployments
    - Containerization for consistency between dev environment and production environment.

VII. Software Test and Quality

To ensure our final product is of high quality, we performed the following checks for software quality as our app is developed:

- **Code reviews:**

  We used a central Github repository for our project. Our workflow was as follows: Start a branch with individual work on a feature on a Github repository. When the feature is completed, begin a pull request to merge the feature into the repository. Teammates can review/ comment on the feature before it is merged in, to ensure that the feature is implemented in a high-quality fashion.

- **Data-science testing**:

  Data science work is often tricky to test. Since our algorithm is physics-based, we made unit tests that ensured that all of our physical formulas output accurate results. We also did some “common-sense” testing, where we ran the hazard algorithm on a route that we know and ensured that the reported hazardous areas aligned with our own understanding of the route. In this case, we used the Lookout mountain climb route to check if hazards appeared in logical places, such as steep descents and tight turns.

- **Back-end testing:**

  Our backend is built with a FastAPI server framework, which includes libraries that include a suite of tests for server functionality. These include tests for user account creation, bicycling route hazard reporting, and other functionality. Unit testing includes both testing of HTTP routes and also utility and helper functions we have written to keep our code clean. We performed integration tests using a technology called Postman. This technology performs a suite of requests to accomplish the desired task, for example creating a user, refreshing an authorization key, or altering user credentials. Postman was also heavily used to test our server routes and verify security concerns:

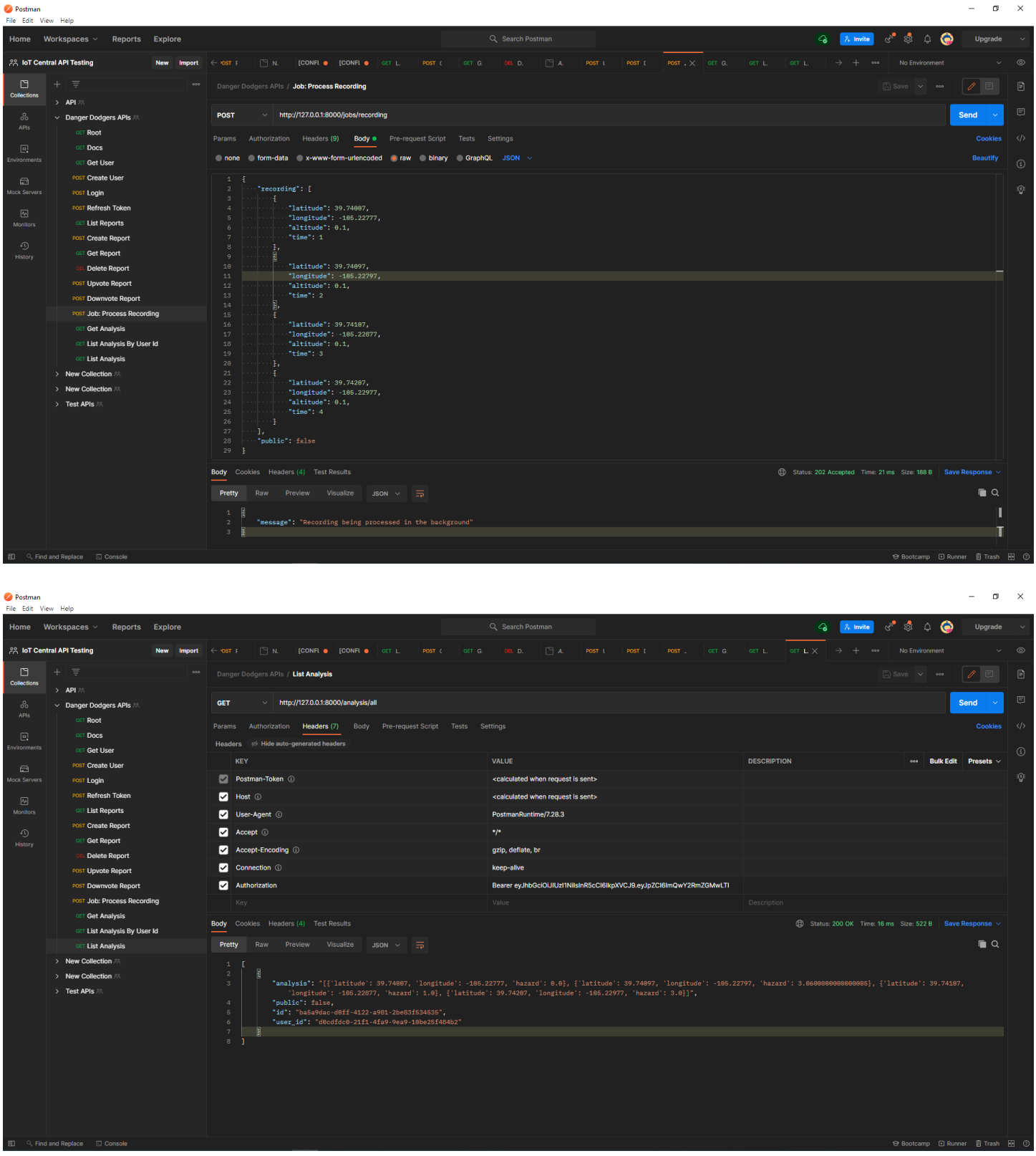


Figure 6: A Postman request/test suite designed to verify our protected server routes.

- **Front-end/ UI testing:**

  We implemented unit tests to ensure that detailed functionality such as helper functions and objects behave the way we expect. We also implemented component testing. Component testing verifies that

the UI looks as it should. An example of this in our application would be clicking and dragging a report marker. We needed to test the listener for the clicking, the dragging, and the release in this case.

- **Optimization and efficiency:**

  The original algorithms designed by the Senior Design team involved constructing n x n matrices, where n is the the number of location (latitude, longitude) points. Operations on very large matrices tend to be computationally inefficient. Thus, to optimize our algorithm's runtime, we implemented all physical calculations as rolling calculations, which only require a few data points at a time.

- **Client/user acceptance testing:**

  We recruited race organizers and cyclists that had connections to the Chad William Young Foundation and the Colorado State Patrol. To gain feedback on our app, we sent these test users a feedback form and instructions for using the app. To ensure client acceptance, we met with the foundation weekly and showed them deliverables, wireframes, and live demos.

- **Security:**

  Throughout the design of our project, we followed three core security principles: (1) Strong authentication/authorization scheme, (2) minimizing collected user information, and (3) presenting any crowdsourced data in a fully anonymous way. Our auth scheme includes MD5 hashing/salting of passwords in the PostgreSQL database and a Dual-JWT authentication scheme, with one short-term and one long-term key. We also implemented rate limiting for new account creation

VIII. Project Ethical Considerations

We have the following ethical considerations for our project:

- **Principles:**

From the ACM code of ethics, we have identified the following Pertinent Principles as the ethical principles that we are most concerned about for this project.

1. PUBLIC - Act consistently with the public interest.

In this case, public interest means reducing cycling related injuries for both recreational and professional cyclists. We have to ensure that our product does in fact help reduce accidents.

2. PRODUCT - Ensure the product meets the highest professional standards possible.

It would not be beneficial to have an application that doesn't really solve the goal of injury minimization, and especially one that gives inaccurate information and results in more injuries. We must ensure our application provides as accurate of information as we are able to produce.

3. PROFESSION - Advance the integrity and reputation of the profession.

In this case, the profession includes cycling race organizers. One of our main goals from our client is to create a product that allows race organizers to provide meaningful information to cyclists before they race about potentially risky areas on the track. It is important that we create a product that can satisfy this.

- **Ethics tests:**

To ensure that our product is ethical, we conduct the following ethics tests.

Harm Test: Does this option do less harm than any alternative? Do the benefits outweigh the harms?

Yes. The alternative to this application is not having any form of hazard detection. Though not all hazards might be noted on the app, having some information on hazards is always beneficial to the user so they have heightened awareness when embarking on a route. The potential harm is that a route with no hazards reported might give users a false sense of security. That is why we need to have strong disclaimers that a route without reported hazards is not necessarily safe.

Common Practice Test: What if everyone behaved this way?

If all cyclists behaved this way (in that they all used the app), this would be the best-case scenario. Large amounts of hazards will be detected as cyclists would upload a large amount of data through each route for the algorithm, and they would manually report hazards. Thus, there would be more data points to improve accuracy and information about specific hazards.

- **Ethical solutions:**

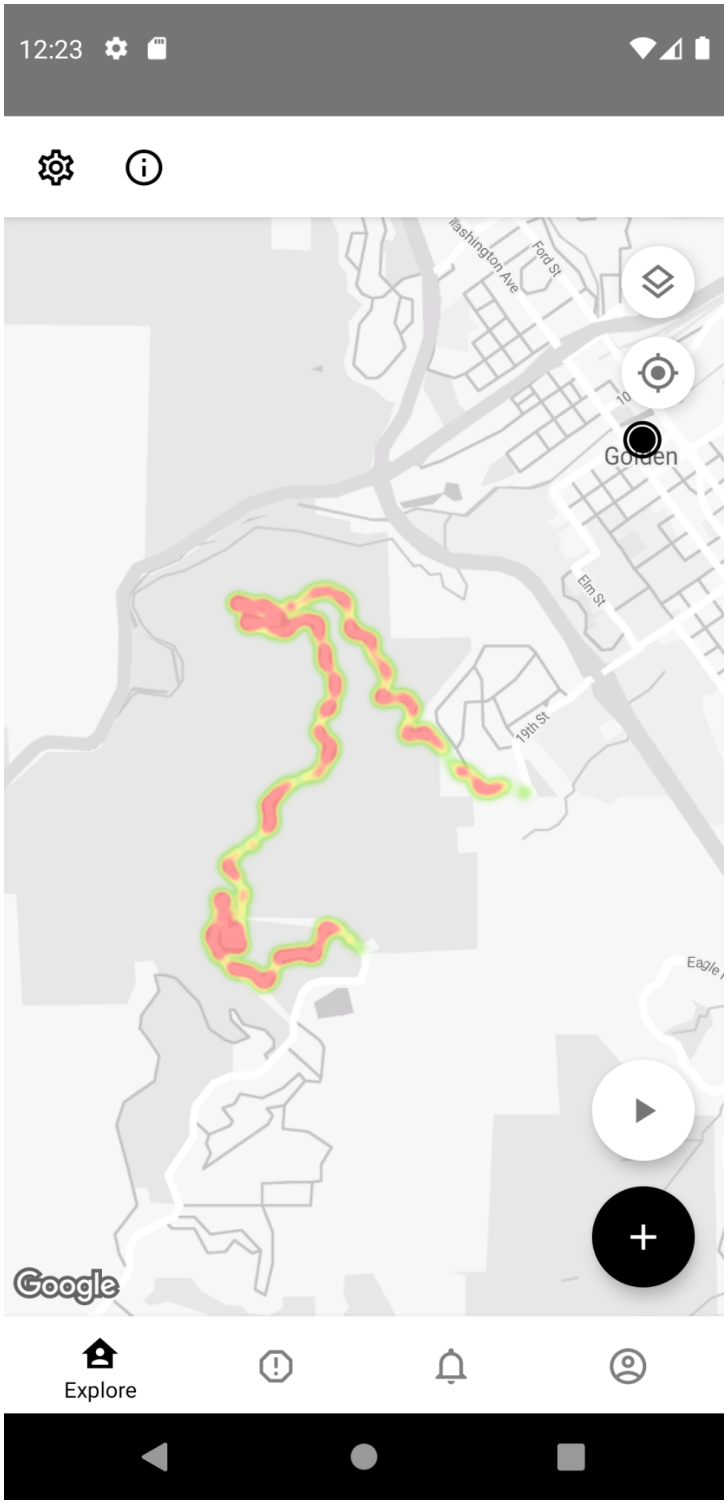
The most important solution is that we need to not oversell the capabilities of the application. Cycling involves inherent risks, and our application cannot possibly detect all hazards ahead of time. This disclaimer must be clearly broadcast when opening our app. We also need to be ethical when it comes to user data. First, we need to gather user consent for access to their data, and abide by the consent that they give. We need to protect user data through strong security, and data analysis must be done on anonymized data.

IX. Project Completion Status

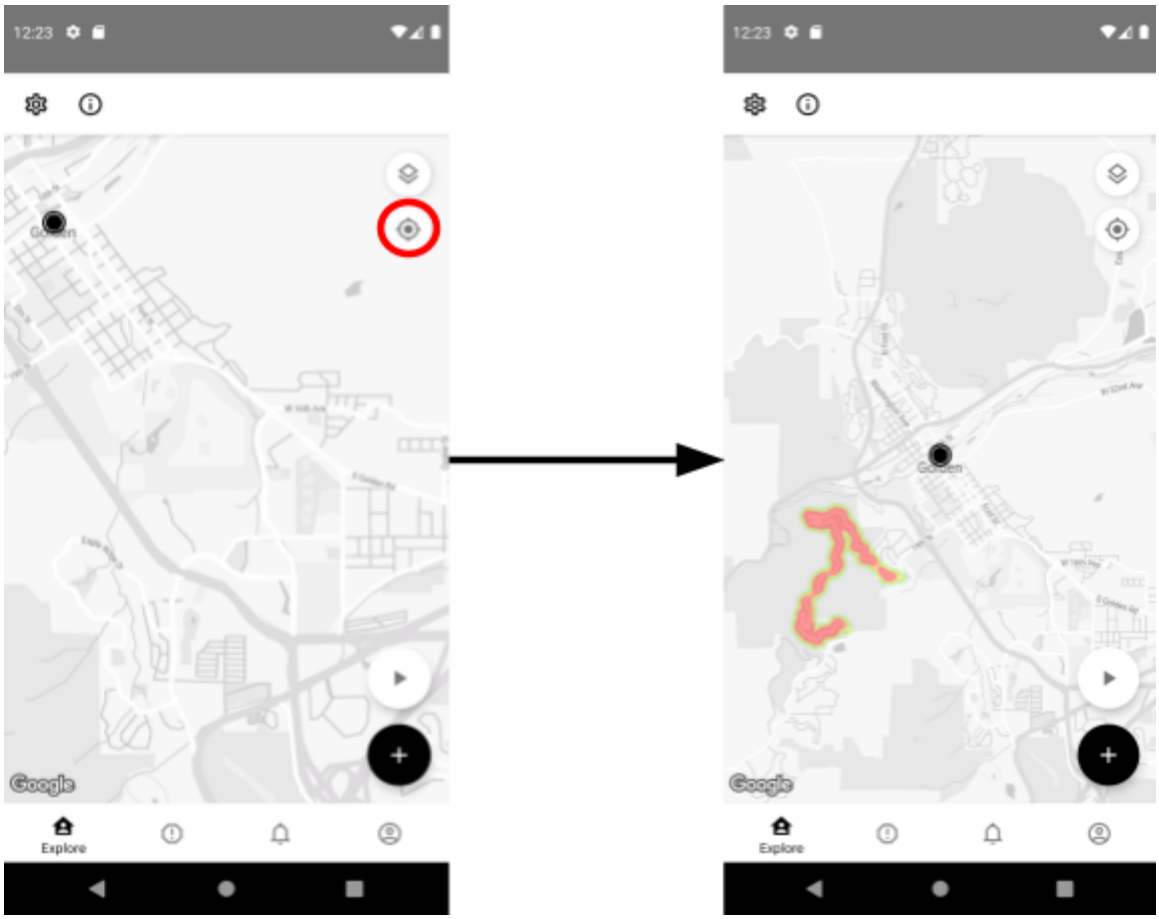
We have a working full-stack application that, at its core, shows a map of hazards to cyclists. These hazards can be generated in two ways. First, the hazard can be reported manually by a user. This hazard can be upvoted/downvoted by other users, to create a crowdsource of reported hazards. Second, hazards are generated algorithmically by a physics-based method. These hazards are generated as follows: A user first records a ride on the application. Then, the algorithm processes the information from this ride to generate hazards. These hazards are then aggregated with other previously generated hazards from the algorithm.

This application also includes the necessary infrastructure to support this core functionality, like user authentication, account settings and customization, ride recording, heatmap views, and hazard update lists.

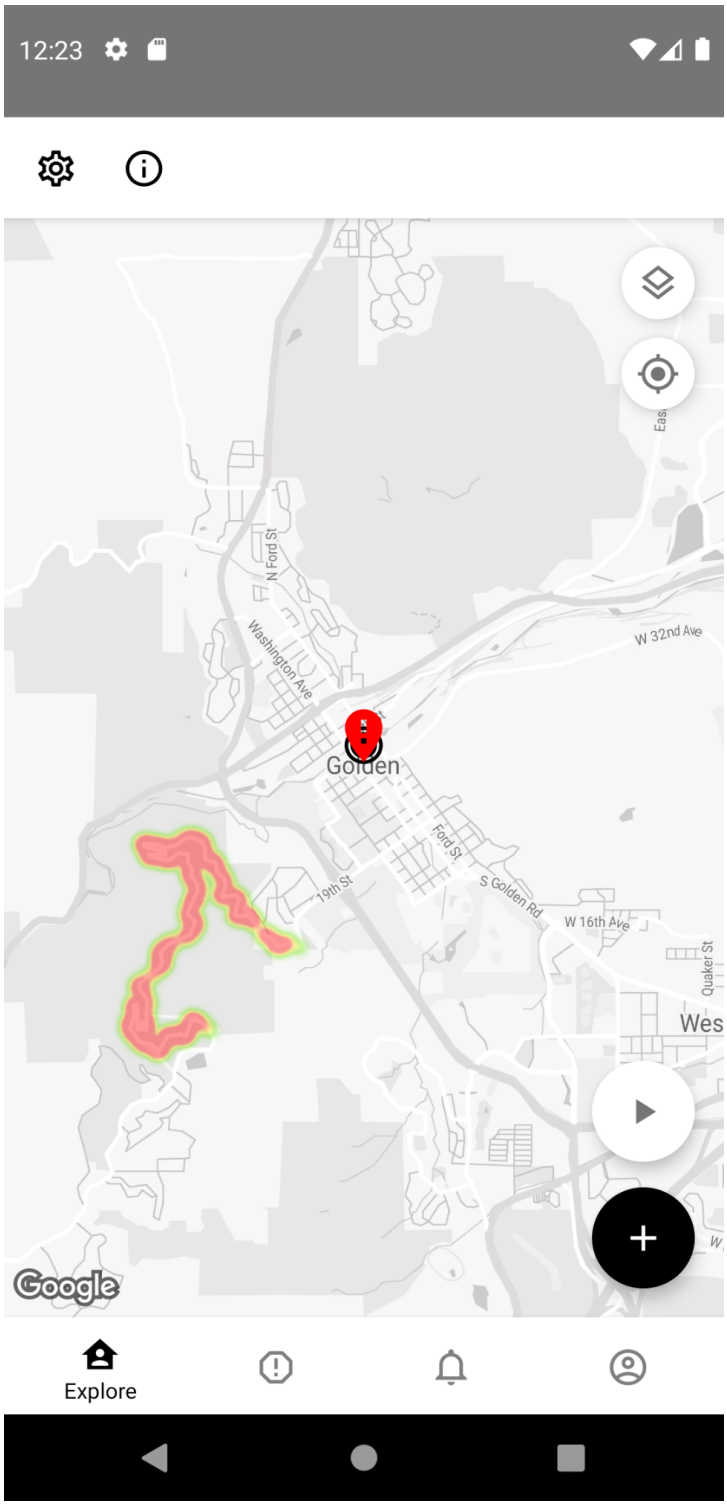
In the following section, we will delve into screenshots of specific app functionality and explain each piece in more depth.



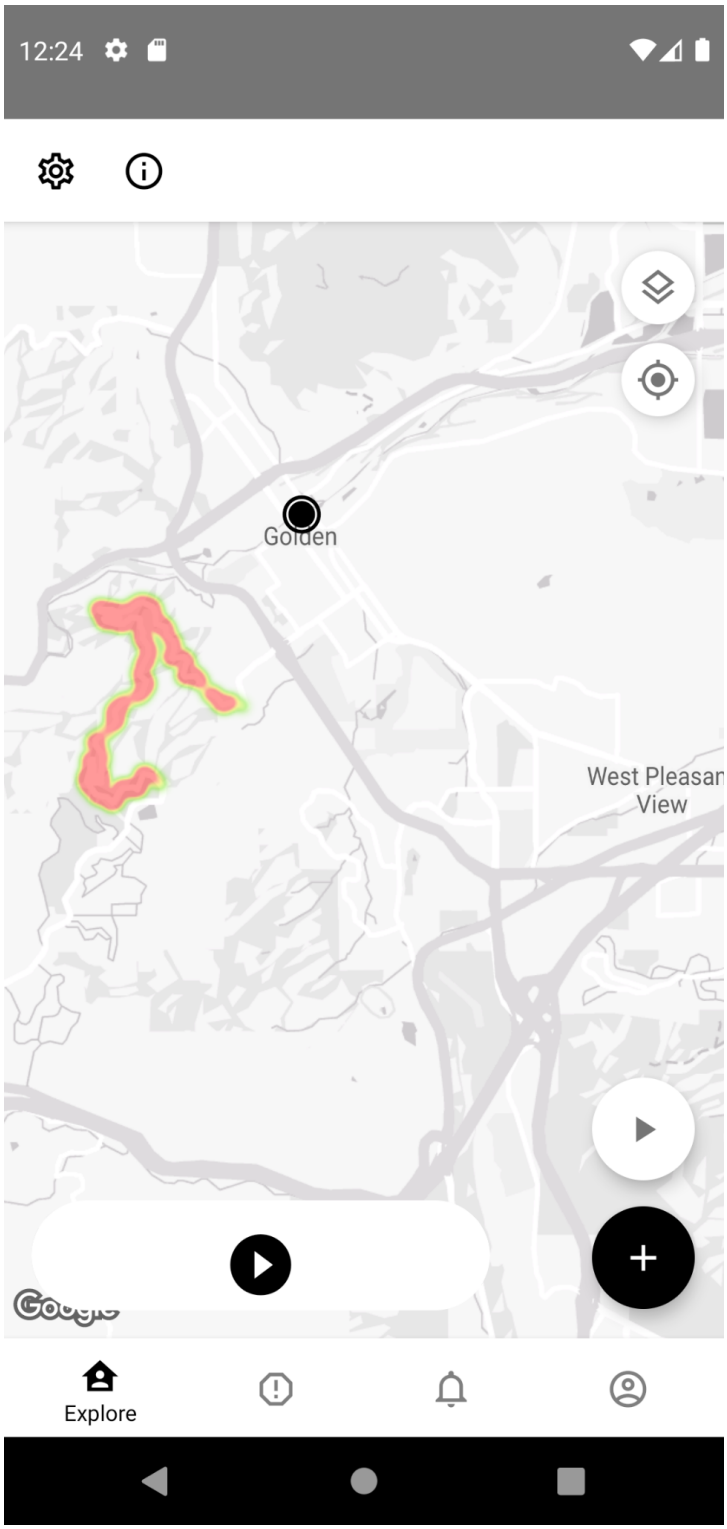
Screenshot 1. Heatmap View: The programmatically generated danger profile based on physics captured during a ride.



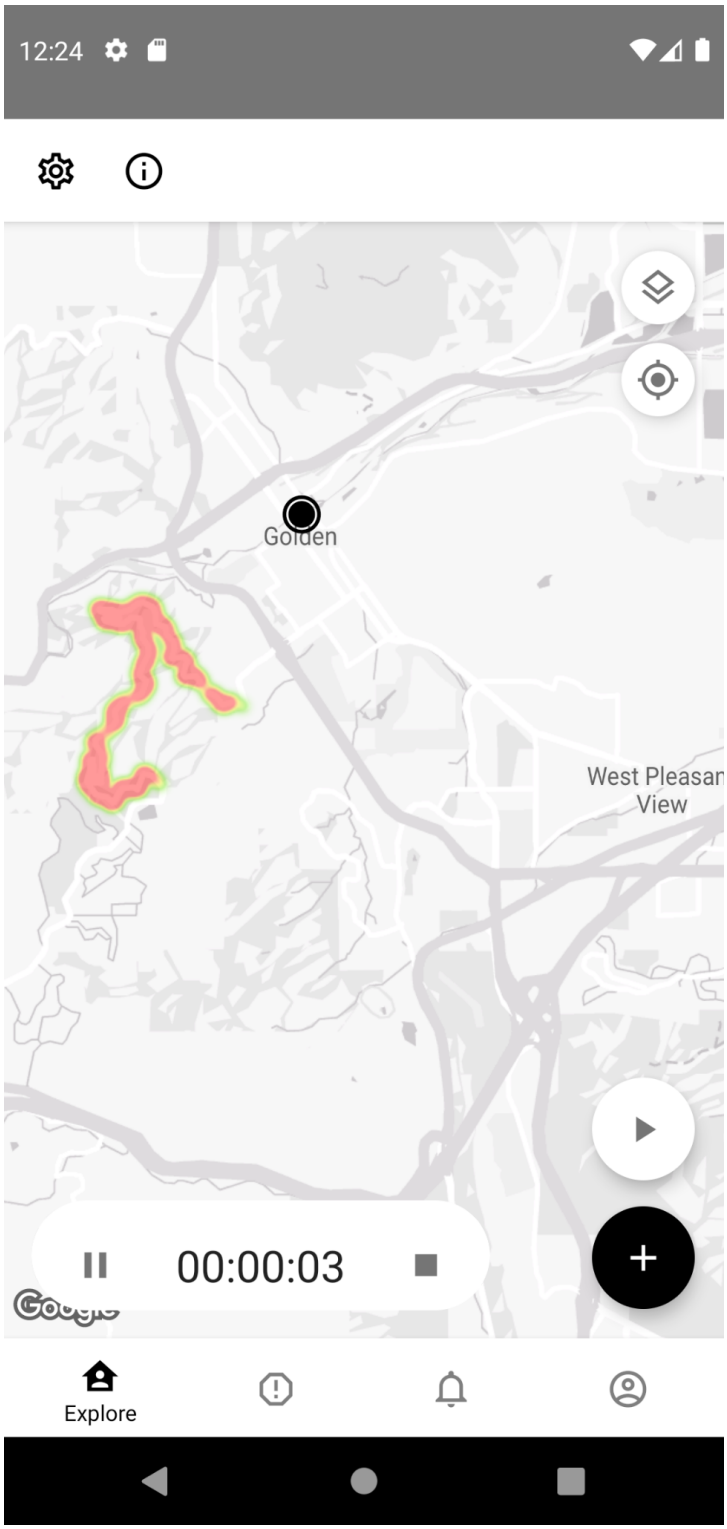
Screenshot 2. Quick Centering: A click to the centering button will move the current position to the center of the screen and move to the default zoom over 0.5 seconds.



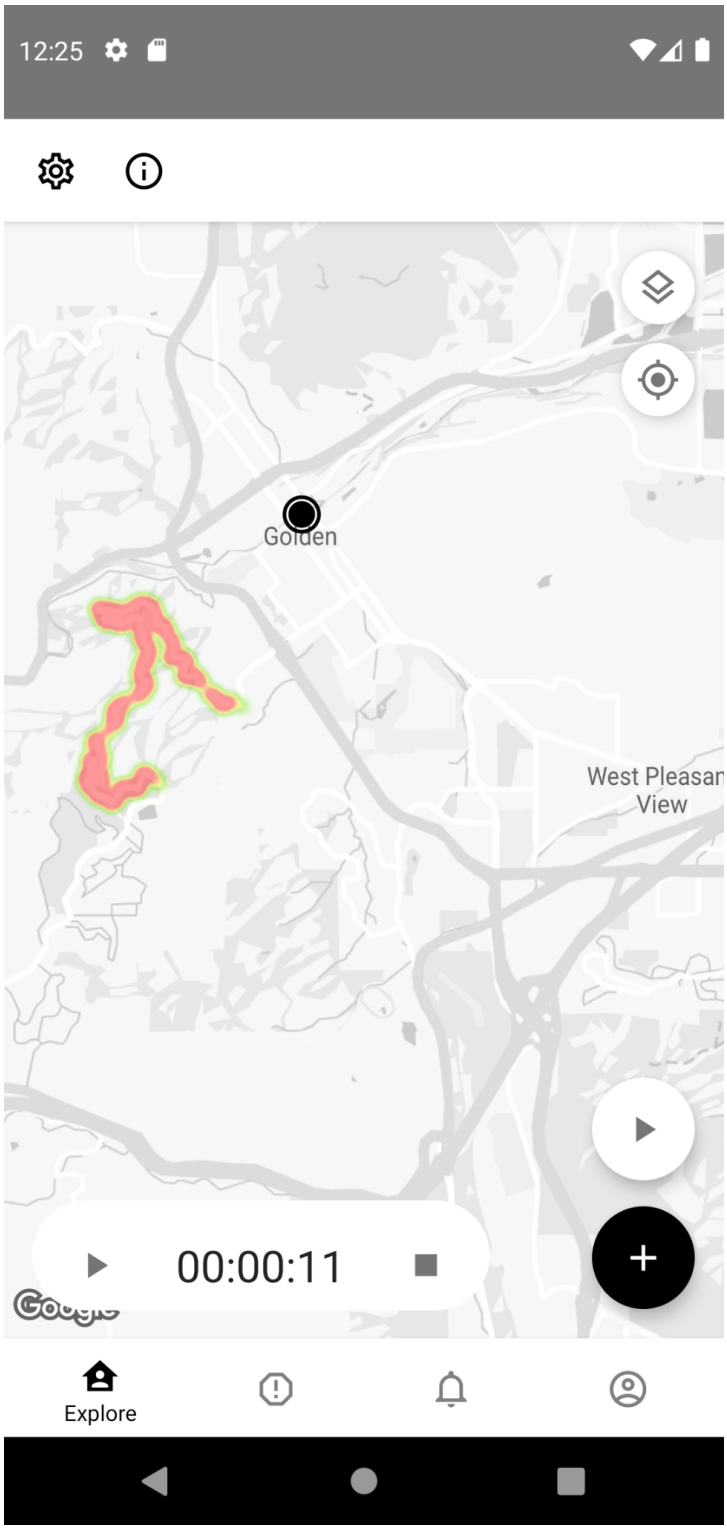
Screenshot 3. Quick Add Functionality: A click on the large black “plus icon” in the lower right corner will add a report marker to the current GPS location. This report marker can be edited at a later time.



Screenshot 4. Ride Recording Functionality: A large play button inside of a pill view at the bottom of the map view. Hitting the play button will start the recording.



Screenshot 5. Recording: A view of the recording functionality as it is recording.



Screenshot 6. Pausing the Recording.

Splash Screen

The splash screen is the least functionality component of the application but serves two primary purposes. The splash screen helps give our application some degree of brand identifiability which will support the Chad William Young Foundation in expanding the user base of their app in the future. It more importantly gives our application time to initialize map content and refresh any user authentication tokens which have been stored into persistent memory. The splash screen animation can also be easily extended to allow for greater loading capabilities.

Screenshot 7: Splash Screen

Disclaimer

A disclaimer is shown after the splash screen. In order to use the application, users must accept the agreement stating that all the information in the application should not be regarded as the absolute truth. The application requires users to report hazards to the best of their ability for it to function at maximum efficiency. It is also mentioned that cycling is inherently dangerous, and the cyclist must still be constantly attentive to their surroundings.

Account Creation/Sign-in Flow

The account creation and sign-in screen appears immediately after the splash screen has completed it's animation. The default view of the screen is the sign-in page, where a user must fill in their username and password in order to access the app.

About Page

The about page helps to tie the application back to its founders and contributors, allowing users to learn about the project. The page contains information about Chad and how the foundation came to be, as well as information about Colorado School of Mines, as many contributors to the project as of late have been Mines

students. The links to each the Chad William Young Foundation and Mines websites are included in the page as well.

App Settings

The app settings allows for personalization of the product. In the settings, the precision of the app can be changed; it can be decreased to save energy, or increased for precise location detection. The radius of the map on the main view can also be changed, along with the coloring of the application.

Main Map View

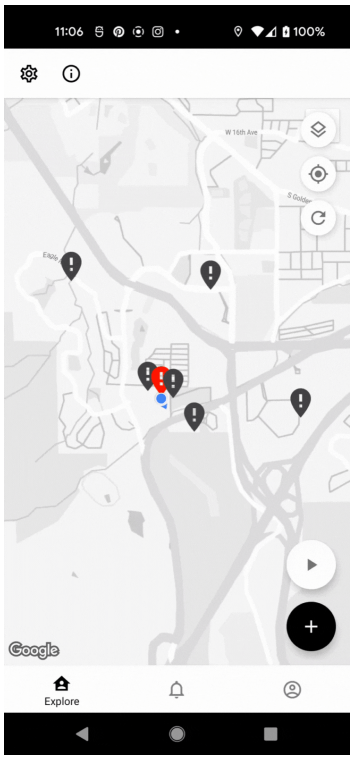
As soon as the application is opened, the map will center to the user's current location. If the user would ever like to recenter the map view, a quick centering button in the top right of the screen will complete this. From there, some heatmaps will be available to see in the main screen, and the user will have the ability to report hazards, view hazards, and much more from the main view.

Account Page

The account page is the page on the application where basic user account information is displayed, including name, phone number, and email.

Account Settings

Account settings are useful for any application like this. A good application has basic information about the user with the ability to change information if need be (phone number, email, etc.) It is important to make sure the user has a secure account with potential to get back into their account if such problems arise.



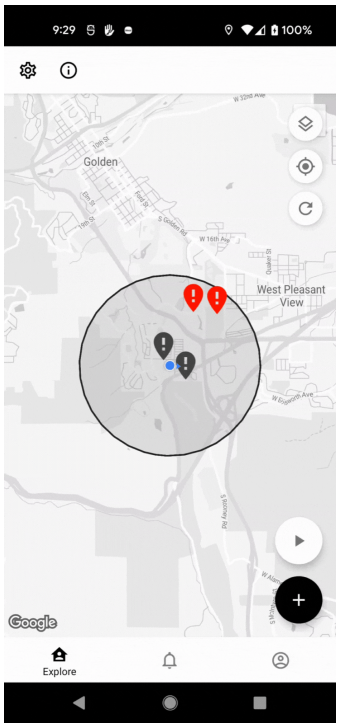
Screenshot 8: Account Settings

Alert Page

The alert page displays reports created by the users in a list-like format, in which the user can scroll through their past reports and view details of such reports. A user can also edit report information from this view, such as the report type, description, and other information related to the report. The report can also be deleted in the case that it is no longer relevant.

Quick Alert

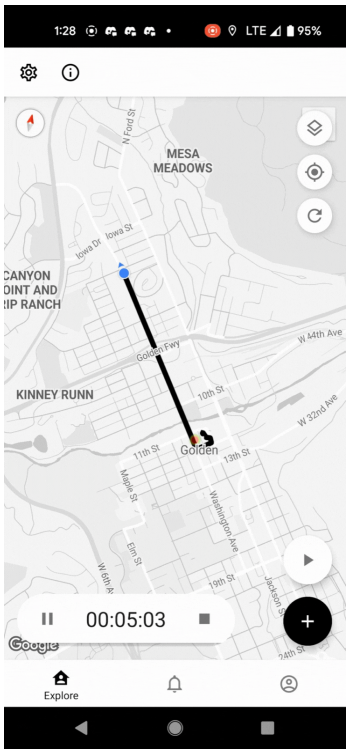
The quick alert functionality allows for users to quickly create a hazard report. The marker will drop down to the cyclist's current location, with the ability for the hazard location to be moved later. The idea behind the quick alert function is that, if a cyclist sees a potential hazard, they will quickly create an alert using the quick alert functionality. After their ride, they are then able to explain the hazard in more detail and finalize the report.



Screenshot 9: Quick Add Functionality

Ride Record

A large play button inside of a pill view at the bottom of the map view will allow for ride recording functionality (recording the current “snapshot” of the ride every few seconds). This allows for capturing of user “physics” and programmatic processing in order to generate a hazard profile.



Screenshot 10: Finalized Ride Recording Functionality

Automated Danger Report

To create an automated hazard rating, we used an algorithm that was adapted from the original algorithm designed by a past group. This algorithm works by rolling through points collected by a cyclist going through a route in time. It calculates rolling averages of physical attributes , such as velocity, acceleration, grade, turn radius, etc. It then compares the state of the physical attribute to the rolling average, and weights the deviation from that attribute. These weighted deviations are combined into an hazard rating. These hazard ratings are from individual cyclist routes and are aggregated and displayed into an automated danger report.

To calculate these physical attributes, time and latitude and longitude points are tracked along a cyclist's trajectory. The GeoPy library is used to calculate the distance between successive points. Distance and time is then used to calculate velocity and acceleration. Turn radius is calculated by first calculating triangle area using Heron's formula and then using that to calculate the height/ turn radius of the triangle . All this information is put into a Pandas dataframe and rolling_apply numpy ext function is used to calculate rolling averages. All of this is compiled into a Python class, which can be run as follows and converted into a json/ csv file.

```
"""
# Example run:
hazards = HazardWeights("my_route.csv") # my_route.csv is a csv w/ headers: Lat,Long,altitude,time
hazards.to_json("my_weights.json")
hazards.to_csv("my_weights.csv")
"""
```

Heatmap View & Toggle

The heatmap view is an overlay which exists over top of the main map view. This heat map displays danger levels of specific turns and geometries along a user's bike ride. More dangerous areas are displayed in red, while safer areas are displayed in green. A gradient exists between these colors to indicate intermediate levels of danger.

Navigation

There are two main forms of navigation in the application: stack, and tab navigation. Tab navigation is present at the bottom of the application, where a user can select from the "main map", "report", "my alerts", and "my account" page. Within each of these pages exists a form of stack navigation, where a user can navigate to any of the other pages.

Theming

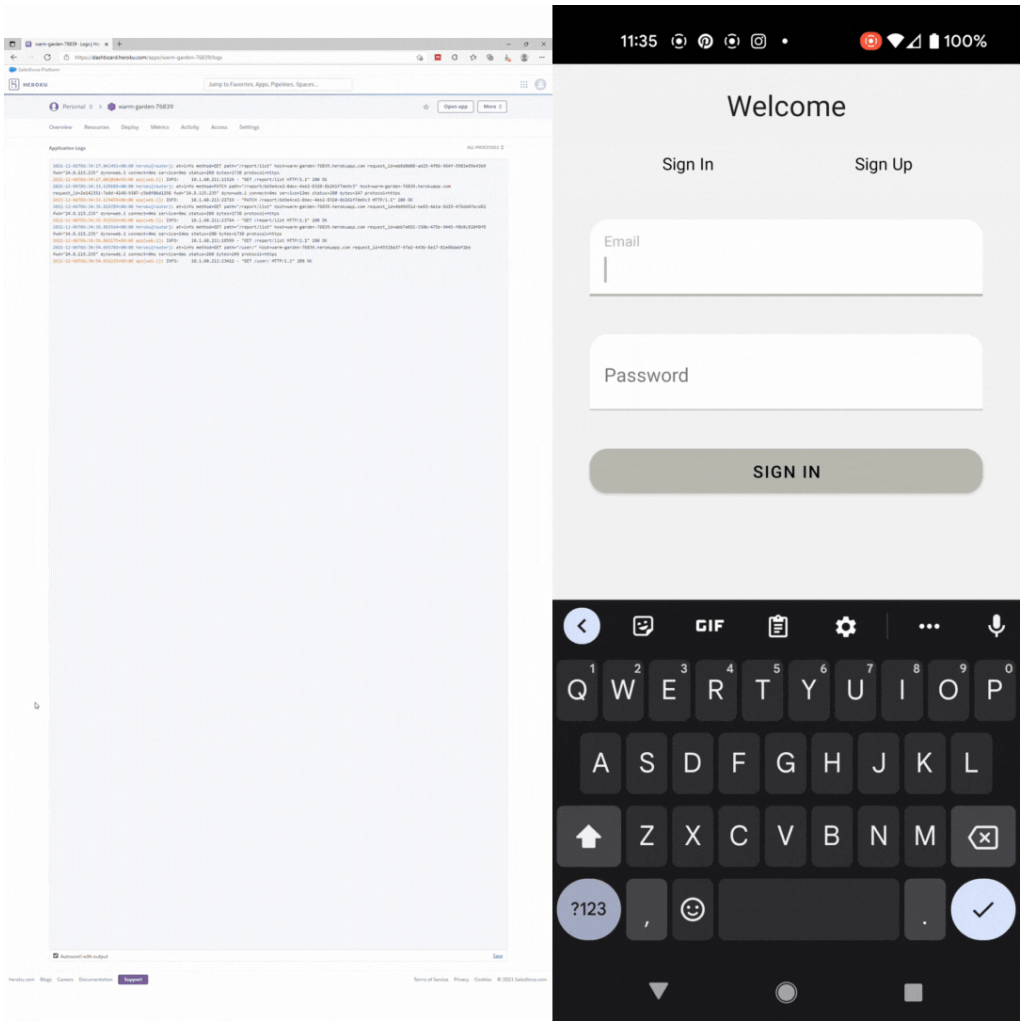
Theming and customization is an important concept in user experience design, and we wanted to give the user different options as to how they want to customize the application .

Server Documentation

Our server automatically generates OpenAPI v3.0 documentation of all routes and schemas in the back-end application. This documentation was not only useful for development on the front-end, but also in creating a robust collection of documents to allow for future extensibility with the application. Server documentation can be seen in the live environment here: FastAPI - Swagger UI.

RESTful API

We designed our server to follow the RESTful and stateless paradigms. What this implies is that the server is not tightly coupled or designed around our android application. Rather, any client, beit a web application, desktop application, android app, or iOS app, should be able to interact with the server and implement some form of user interface in the same way.



Screenshot 11. Client server interaction

Cron-Job

Cron-jobs (or long-lasting tasks) were integrated into our server, allowing for complicated and lengthy processing of user recorded bike rides. This feature boosts the performance of our server since there are some pieces of functionality which cannot be executed quickly. These recordings may need several minutes to process before returning information to the client if the recordings are lengthy. This also prevents our client from hanging when waiting for the cron-job to complete, something which may occur if the functionality was implemented under an ordinary route.

Security

Security principles were inspired by the OWASP Secure By Design Development Guidelines. The presented security principles are highly extensive, and our app adopts an abridged version that meets security needs. We focus on seven main security principles throughout our application:

The first security principle is the minimization and anonymization of sensitive user data collection. Sensitive user data is information which may reveal an individual's identity, location, personal beliefs. Since the Danger Dodger's application does take advantage of crowdsourcing, specifically in the form of locational data, it was of

utmost importance to keep such data anonymous to users. Accordingly, crowdsourced user hazard reports are displayed in a purely anonymous way on the client side, and such data is also kept anonymous when sent in a response to the client.

The second security principle is a strong upfront authentication and authorization scheme. Our application utilizes JSON web tokens to authenticate users, sending this special token to the client app upon a successful login. This token is periodically refreshed by the client in order to keep the user signed in while preventing a web token from becoming stale by exceeding its expiration date. Password encryption is implemented in our database, leveraging a SHA-256 scheme to hash and salt such passwords before they are stored into the database. The primary advantage of using JSON web tokens is keeping our server stateless, as it does not need to use server sessions to manage user sign in. JSON web tokens are also considered safe to use with mobile applications, as attackers cannot leverage the cookies of the mobile application like they may be able to with a web application.

The third security principle is encapsulating sanitization. Every query in our back-end application utilizes modern SQL sanitization to prevent SQL injection attacks. This functionality is provided by the asyncpg python library, and will prevent malicious users from taking advantage of SQL queries which are found in most routes of the server.

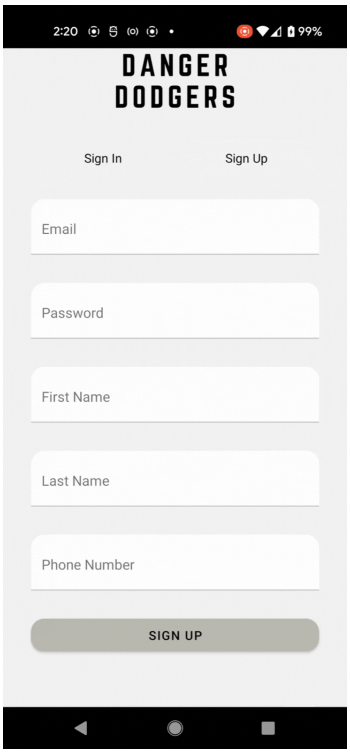
The fourth security principle is rate-limiting. Rate-limiting is implemented on specific server resources, such as account creation, hazard reporting, or changing account information. Rate limiting is the only stateful component of our server, where IP addresses are temporarily tracked and denied access to certain server resources if the rate limit has been exceeded. The 429 “too many requests” error code will be returned in the case that this rate limit is exceeded. The rate limit is especially important for our application to prevent a special kind of crowdsourcing attack referred to as “Sybil,” where a malicious user creates many false or malicious reports to trick other users into changing their behavior. While the Sybil problem is impossible to fully prevent, rate limiting can aid in preventing excessive account and report creation which will minimize the impact of the problem.

The fifth security principle is detailed error logging in responses. Rather than returning a general error to the client, we return a detailed description and appropriate error code so that a user may have more information on what kind of error occurred. This prevents frustration and also aids developers in correcting server errors.

The sixth security principle is data validation. This validation occurs before the creation or mutation of any database objects so that a malicious user cannot take advantage of or crash our server by providing corrupted data which will cause processing errors in the server. This includes filtering for special characters, null characters, enter characters, empty strings, and other possible malicious data.

The seventh security principle is stateless architecture. The only exception to the stateless nature of the server is to support the fourth security principle (rate-limiting), in which IP addresses are tracked in order to support rate limiting functionality. It would take an overwhelmingly large number of requests to take advantage of such a state, and the rate limiting would functionally prevent this.

Below demonstrates an example of the results of rate-limiting when a user tries to create an account too many times in one minute:



Screenshot 12: Rate limiting

X. Testing Results

The application was sent over to the testers for testing in November. The testers include cyclists affiliated with Bicycle Colorado & the Colorado State Patrol as well as some of Chad's high school and college friends. Each tester was given a set of 10 questions and were told to rate each statement on a scale from 1-10 (1 being strongly disagree, 10 being strongly agree). The questions are below:

The application was easy to install and create an account:
The application looks aesthetically pleasing:
I was able to customize the application to my liking:
The layout of the application makes sense:
The home map displays information about hazardous areas in an easy to understand manner:
The heatmap is easy to make sense of:
Creating and editing reports are simple tasks:
When I made a report, the location was reported accurately:
The ride recording functionality accurately records the rides that I take:
I can see the benefit that this app provides:

The testers were unable to provide their feedback in time for this project. However, we will continue to work with the foundation in order to ensure they receive the feedback from the testers as the app continues in development.

XI. Future Work

One of the original design stretch goals was to implement real-time auditory warnings to cyclists when they approached a hazard. The challenges of this feature are both technical and ethical. For our technical concerns, we would need to have a way to continually monitor the cyclist's location and direction of movement. We would also need to decide on some sort of distance radius cutoff between a hazard and a cyclist that would sound an alert. For our ethical concerns, we would need to decide which hazards are important enough to alert. If we choose to alert too often, we could cause users to tune out the alerts. If we don't alert enough, we could fail to alert users to important hazards. These real-time alerts could cause a false sense of security, or could even be enough of a distraction to be dangerous. That is why we have decided to leave the app as more of a "preview-route" feature for hazards instead of a real time warning for hazards.

Another possible direction of future work could be route pre-entry, such that a user could enter in a specific route into our application before going on a ride in order to get direct feedback on which hazards exist along their ride. Currently, users must look along their route manually and observed what hazards may be relevant; however, in a future version of our app, perhaps users would be able to draw a route and receive a corresponding hazard report, or enter in a final destination and allow the app to automatically generate a route with the fewest number of hazards. Although difficult from a technical perspective, both of these ideas could be useful additions.

One important future feature is the aggregation of automatically generated hazard reports. Currently, hazard reports are generated post route from one user's ride. The numerical hazard ratings for latitude and longitude points on the route are stored in the database. There should be an algorithm that groups together hazard ratings by latitude and longitude points that are in close radius to each other. These groups will form new points, and the aggregated hazard rating will be the average of the original hazards. Then, the displayed heatmap should display the aggregated ratings.

Strava integration could be a useful future addition to our application, where users could import a route they have recently created or ridden into our application and generate a corresponding hazard report for their ride. Strava offers a REST API which could possibly be leveraged to add this functionality, and could also help popularize our app because of Strava's current popularity amongst cyclists.

Strava Metro dataset is a dataset that aggregates the volume of cyclists on a certain edge. We obtained access to this dataset through partnership with Colorado State Patrol. In the future, a map view could be added to the application that displays the traffic on certain routes.

In the future, to ensure that our application runs efficiently, more metrics need to be collected on server response time, especially when there are concurrent users. Tests should be run to see which HTTP routes are our bottlenecks in terms of runtime and CPU usage. We also want to see how our algorithm scales with the number of points in the routes.

XII. Lessons Learned

We learned that having some flexibility in your development plan is important. After initial client meetings, we decided to create the application in Android, and thus decided to program the app using Android Studio. Through discussions with the client, the decision was ultimately made to use React Native instead. React Native was easier to develop in, and also compiled to native Android Studio, making it so that decision had no negative tradeoffs except for some lost development time that was used in our initial time programming with Android Studio. This seemed to be a good decision in hindsight, as the team was more comfortable using React Native compared to Android and we ended up saving more time. Using React Native as the framework still

allowed for a final application delivered on the Google Play Store through Android, and the development process seemed to go more smoothly after the transition took effect.

One problem our team encountered during the early stages of our project was planning. It was our intention to create an app centered around data provided to us by Strava Metro. However, when we received the data and it ultimately looked very different from what we were originally expecting, we were not able to go through with our initial plans and had to pursue an alternative. One important point that we learned is that it is always important to have a backup plan. Since we had discussed some alternatives with our client early in the project, we were able to prioritize those after receiving the data and still create a substantial final product. It was a difficult transition and left us with less time to complete the final deliverable, but we were nonetheless able to follow a backup plan and still deliver features that our client was happy with.

Another lesson that our team learned was the importance of beginning with small features, ideas, and deliverables before building upon those to create more substantial features. In particular, building the ride “recording” functionality appeared to be very daunting at first, but focusing on the subcomponents of the functionality allowed us to slowly build up the feature to completion. We first implemented the GPS and altitude tracking, then integrated persistent storage so as to not overwhelm a smart phone’s limited RAM, and lastly we integrated the recording with our algorithm. Other functionality of the app shared a similar story, being subdivided into small pieces as opposed to committing monolithic features.

XIII. Team Profile

Zoe Baker
Senior
Computer Science and Applied Math/Stats
Hometown: Longmont, CO
Work experience: Software Engineering Intern, Google. Research Engineer Intern, Lockheed Martin.
Activities: Varsity Cross Country and Track, Undergraduate Research Ambassador

Ethan Perry
Senior
Computer Science
Hometown: Littleton, CO
Work experience: Software Engineering Intern, Microsoft. Software Engineering Intern, Lockheed Martin. Program Management Intern, Tesla. Product Engineer, Spore.
Activities: Club Triathlon, Undergraduate Research

Cole Wapelhorst
Junior
Data Science and Biomedical Engineering
Hometown: Littleton, CO
Work experience: Data Science Intern, US Geological Survey.
Activities: Club Golf, Rock Climbing Club, Undergraduate Research

References

Chad William Young Foundation - Site: https://cwyf.org/

Computer Science Field Session - Site: https://cs-courses.mines.edu/csci370/

Appendix A – Key Terms

Term	Definition
Cron-Job	*Job scheduler on an operating system, used to run long term/ scheduled commands/ scripts.*
Entity- Relationship Diagram	*Graphical depiction of the relationship between entities in a database, and their properties*
Dual JWT	*An authorization scheme wherein an authenticated user gets a short term access token and a longer-living refresh token to get more short term access.*
Geographic Information System (GIS)	*A framework for the processing and analysis of spatial/ geographic data.*
Heatmap	*A type of data visualization of where the level of an attribute is displayed as a relative color/ light intensity on a map.*
Heron's formula	*Gives the area of a triangle given all three sides.* *A = sqrt(s*(s-a)*(s-b)*(s-c)), where s is the semi-perimeter*
MD5 hashing/salting	*An encryption scheme used for user password storage in our database.*
RESTful API	*An API that conforms to Representational state transfer (REST) standards. REST is a software architectural style where the client and server are interlinked as loosely as possible. Implementation details in the server are abstracted away and hidden from the client.*

Appendix B – Code

Root repository: Ethanperry247/danger-dodgers-root (github.com)

Front-end repository: Ethanperry247/danger-dodgers: (github.com)

Back-end repository: https://github.com/Ethanperry247/danger-dodgers-server

Link to presentation slides (to see animated gif demos): Final Presentation

Live Server Documentation: FastAPI - Swagger UI (warm-garden-76839.herokuapp.com)