Pedway Navigation

Final Documentation

CS 429

University of Illinois

at Urbana-Champaign

Alex Vanyo (alexv2@illinois.edu)

Curtis Lowder (clowder2@illinois.edu)

Ian Howe (ianhowe2@illinois.edu)

Mia Schoening (miajs2@illinois.edu)

Kexiang Wang (kwang66@illinois.edu)

Yilin Ma (yilinma3@illinois.edu)

David Zhang (yzhng223@illinois.edu)

Table of Contents

Description of Project

The city of Chicago contains a Pedestrian Walkway System, commonly known as the Pedway. The Pedway serves as a network of underground and above-ground pedestrian tunnels connecting 40 city blocks and almost 50 buildings within the Loop. As a result, this system is an excellent option for pedestrians seeking a shorter commute or shelter from extreme winters and unpredictable weather.

Unfortunately, a lack of adequate funding in the Pedway over the last few years has resulted in limited awareness and poor signage throughout the system. In addition, many sections of the Pedway have seperate open and closing times, causing navigation to be unpredictable and frustrating. Chicago's leading newspaper, The Chicago Tribune, has published multiple articles about the Pedway in recent years, noting these concerns in addition to the fact that no app-based navigation exists for the Pedway. Popular navigation apps such as Google Maps aren't built to handle the unique challenges that the Pedway brings.

Pedway Navigation is the app-based solution to all of these problems, powered by community-driven resources that ensure users receive an updated and accurate navigation experience. Our app allows a pedestrian in Chicago to travel to any destination they would like and provides unique navigation options including an underground viewing mode and multiple ways to receive directions and maps even when GPS or mobile data connections are unreliable. Pedway Entrances and popular attractions including restaurants, shopping malls, and subway stations are also marked on the map. In addition, users can view information on any Pedway entrance and provide feedback in the event that updates on closure or cleanliness need to be made. This feedback is then collected on our backend server, which can be reviewed and approved by an administrator to update a particular entrance status or contact the owner of a section in need of cleaning or repair.

End User Manual

Welcome to Pedway Navigation! This tutorial will assist you in finding, installing, and using our app. If you have any questions, comments, or concerns, feel free to leave a review on our page in the app store or send us an email at pedwaynavigation@gmail.com. Let's begin:

1. Installation
    a. Frontend installation via the App Store (android / iOS)
    Search for "Pedway Navigation" in the app store on your device, and our app will show up. From there press "Install" and follow the steps shown on-screen. If Pedway Navigation has not been published to the app store that corresponds to your platform, please follow the instructions for part b instead.
    b. Development setup (Webstorm)
    Please follow the steps listed in our readme file to ensure a smooth installation: https://github.com/cwlowder/pedway/blob/master/pedwayApp/README.md
    c. Backend server setup (optional)
    If you would like to setup and access the backend server for Pedway Navigation, please follow the readme file listed on the github page: https://github.com/cwlowder/pedway/blob/master/backend/readme.md
    d. Backend server access
    Accessing the backend server can be done by going to the following link and logging in with a valid gmail account:
    https://pedway.azurewebsites.net/login.html
    Upon login, a system administrator will be able to approve you for admin privileges, allowing you to update the status of Pedway entrances as need be.
2. App usage
    a. Buttons
    Once you arrive onto the main screen of the app, there will be a number of items you can select. The app will open while centered on your current location. At the top of the screen lies our search bar, which can be used to find a particular location or a few coffee shops in the area depending on your inquiry. Below this search bar, to the right, lies a button with a bent arrow on it. This button allows you to switch between the above-ground and

underground views of the app. Below this button we have the recenter button, which will swiftly move your screen to your current location in case you have moved the map too far away. At the top left of the screen, a button with three horizontal bars can be selected, which will open a side menu. This menu provides two features: the first is a directory of popular destinations within the pedway, and the second is an offline pdf that can be access in the event that the main navigation system is unable to assist you due to GPS or mobile data connectivity issues.

b. Selecting an entrance

The map is filled with two types of special markers. The green markers denote pedway attractions, and can be tapped for more information. This will bring up a menu on the bottom of the screen with more information about the location and a few more buttons to choose from. The first button is the main routing button, which will show you a way to travel from your location to the attraction when pressed. The button can be selected once again in order to exit the routing mode. Pedway entrances operate similar to attractions, but with a few additions. The entrances are shown on the map as a round button with stairs, and when selected the menu on the bottom screen will display information about the entrance that includes whether or not the entrance is open. There is also an additional button on this menu that can allow you to submit a feedback form in the event that you find the entrance status to be inaccurate or if that particular section of the Pedway is unclean.

c. Navigating

After selecting a marker on the map and pressing the navigate button, a route will appear on your screen with instructions at the top. If the recenter button is pressed, the screen will automatically follow you as you traverse the Pedway. The instructions at the top can be swiped to the left or right in order to see what your next steps will be. This is also a useful feature to take advantage of in the event that your device cannot accurately find your location with underground, allowing you to manually move onto the next step if need be. If you stray too far from the original route, our app with recalculate a new route for you to take automatically. Upon reaching your destination, the app will congratulate you on your arrival.

Software Development Process

The process followed by our team was a modified version of Extreme Programming (XP). As a type of agile software development process, it focused on iterative development with frequent “releases” after each development cycle. During each cycle, a new set of user stories to be implemented was adopted and developed. Having a narrow set of requirements to implement during a specific iteration enabled us to have completed features and functionalities to release for our bi-weekly iteration meetings, which served as the end of each cycle. This also kept us from avoiding implementing features until they were needed, which was particularly useful as we were building the project from scratch and designing our own requirements and specifications which inevitably were subject to change as we grew to better understand the problem. By having short development cycles, we could reevaluate our requirements at the end of each cycle and update them as needed. In doing so, we were able to avoid having our changing requirements negatively impact our productivity.

We chose as a team not to pair program, instead opting for an extensive code review process. Any pull request into our ‘dev’ branch required approval from another team member before it could be merged. This helped us to maintain our standards of high-quality, simple code. Every team member participated regularly in the code review process, both providing constructive feedback to others and also receiving feedback on their own code. When certain sections of code were recognized as being subject to improvement, we would work to refactor that code. This process of restructuring existing code without changing its external behavior was another important facet of delivering high-quality code. It helped us to improve code readability and reduce the overall code complexity. Opportunities for refactoring were often identified during code reviews or iteration meetings, but were also identified individually by the specific contributor.

Our main focus for testing was to implement unit tests for all code. We then also ensured that all code had to pass all unit tests before it could be released by utilizing the Azure continuous integration and continuous deployment (CI/CD) pipeline. The coding standard that we established for our project was the Google JavaScript Style Guide, enforced with a linter.

The nature of the XP process necessitated collaborative development. We met twice weekly, with meetings typically lasting around two hours. These meetings were used to identify the requirements to be implemented during the current iteration and to divide the relevant work among the group. We also were constantly discussing the overall requirements and specifications and changing them when needed. Meeting so frequently helped us to address any problems that arose quickly and efficiently and kept us on

track to complete user stories in a timely manner. It also contributed to our sense of collective code ownership, where all members were kept updated about all sections of code and had the opportunity to offer feedback or change any part of the code.

Finally, we also used several tools to aid us in our software process. Our team Wiki was used to document for each iteration our MARS roles, user stories and story point estimates, updated use cases, meeting notes, and iteration meeting feedback. Having all of this information in a central location was useful for keeping all team members on the same page. We also used two separate Trello boards, one for software tasks and one for non-software tasks. This allowed people to claim responsibility for a task and show what their progress on it was, up until it was finally released. For software tasks, we also labeled each task with the estimated story points and then final points, and included a link to the relevant PR. In addition to our weekly meetings, we then also made frequent use of our team Slack channel, which included various sub-channels as well.

Requirements and Specifications

User Stories

- When a user chooses to navigate to a destination, they are provided with the fastest walking route from their current destination.
- The provided route will avoid any Pedway entrance closures.
- Users can select an entrance or attraction and navigate directly there.
- Users can search for a desired location in Chicago, and then choose to navigate there from their current location.
- When a user gets within 5 meters of a Pedway entrance, they are prompted to switch to the underground mapview. When they get within 5 meters of a Pedway exit, they are prompted to switch to the aboveground mapview.
- If a user strays more than 100 meters from the provided path, they will be rerouted with a new optimal path to their destination.
- As a user is navigating on a path, they can see their updated location continuously reflected on the map.
- When a user is underground and does not have access to location services or the Internet, they can manually swipe through the directions to view the next steps they need to take.
- When the user arrives at their destination, navigation automatically terminates.
- Users are notified of the estimated travel time of the provided route.
- Users can select an attraction displayed on the map to view its name and the hours that it is open.
- Users can select a Pedway entrance on the map to view its current status (closed or open).
- An error message is displayed when a user does not have access to location services or Internet, but they will still be able to view the previously loaded route (if applicable).
- Users can access a static map and directory of the Pedway system that does not require any Internet services.
- Users can report feedback for a specific Pedway entrance (closed, open, dirty, clean).

Use Case 1

Use Case Brief: The user finds the optimal path to navigate between two locations.

Casual Use Case: The user opens the app on the ground and selects their desired destination. The user is shown the recommended path to take to get to their destination.

Fully Dressed Use Case:

Primary Actor: User

Goal: Find the optimal path from the user's current location to their desired destination.

Stakeholders and interests:

Users: Figure out how to get to a certain location.

Scope: Navigation

Level: User Goal

Guarantee: If the Pedway system can be used to save time on the route, it will be included in the recommended path.

1.1 Precondition:

The user is on the ground and the user has access to the internet and GPS service.

1.2 Main Flow:

Trigger: The user opens the app with the intention of navigating to a specific location.

The user selects their desired destination [S1][S2]. The suggested route is displayed on the map, showing the complete path to be taken [S3][E1]. The user accepts the provided route and begins navigating [UC2].

1.3 Subflows:

[S1] The user's current location is used as the starting point.

[S2] The user can choose their desired destination by either selecting an entrance or attraction directly on the map, or by searching for their desired address.

[S3] The provided path will avoid any current Pedway entrance closures.

1.4 Alternate Flows:

[E1] If there is no reasonable route to the desired destination, a note indicating so will be displayed.

Use Case 2

Use Case Brief: The user navigates to the desired destination.

Casual Use Case: The user is given step-by-step directions to navigate to their destination.

Fully Dressed Use Case:

Primary Actor: User

Goal: Navigate between two locations.

Stakeholders and interests:

Users: Navigate to desired destination efficiently.

Scope: Navigation

Level: User Goal

Guarantee: If the user follows the path that the app suggested, the user will arrive at their desired destination.

2.1 Precondition:

The user is on the ground and the user has access to the internet and GPS service.

2.2 Main Flow:

Trigger: The user has chosen to navigate to a specific location using the provided route.

Step-by-step instructions are provided to the user, detailing the direction of their current step [E1]. The user can swipe between the instructions to access the previous and next steps [S1][S2][E2]. When the user arrives at their destination, the map automatically completes navigation.

2.3 Subflows:

[S1] As the user approaches the destination, the user can see his/her location change.

[S2] When the current step is within the Pedway system, and the user is not using the underground map view, the app will suggest user to switch to the underground map view.

2.4 Alternate Flows:

[E1] If there is no GPS signal or network connection, a screen will be shown to indicate the error.

[E2] If the user deviates more than 100m from the path, the app will automatically recalculate the route and display the new path to the user.

Use Case 3

Use Case Brief: The user accesses provided information about Pedway entrances and attractions.

Casual Use Case: The user selects a Pedway entrance or attraction on the map view and is provided with more information about that specific entrance/attraction of the Pedway. The user can also access an offline map of the Pedway system, as well as a directory of Pedway attractions.

Fully Dressed Use Case:

Primary Actor: User

Goal: Discover up-to-date information on the Pedway system and its attractions.

Stakeholders and interests:

Users: Find information about the attractions that each section of the Pedway can offer.

Scope: Information Feature (component)

Level: User Goal

Guarantee: The Pedway App will provide information regarding all entrances to the Pedway and the attractions that are provided within.

3.1 Precondition:

The user has the app open. The user wants to explore the Pedway.

3.2 Main Flow:

Trigger: The user chooses to learn more information about a section of the Pedway.

The app provides a map of Chicago overlaid with pins that correspond with entrances and attractions to the Pedway system [S1] [S2]. The user selects a Pedway entrance/attraction that they would like to know more about. For an entrance, the user is given its name and its current status. For an attraction, the user is given its name and hours open [A1]. From the side menu, the user can also access the directory, which lists all of the restaurants located within the Pedway and their respective address. The user can also view the static map of the Pedway system in image-form.

3.3 Subflows:

[S1] When initially opened, the map displays the Pedway system in the downtown Chicago area.

[S2] Entrance pins are not visible while navigation is in progress.

3.4 Alternate Flows:

[E1] The user can choose to navigate to this entrance or attraction [UC1].

Use Case 4

Use Case Brief: Update/View Pedway statuses for entrances

Casual Use Case: Based on feedback from users and other sources, administrators update the status for different entrances into the Pedway. Administrators can see the current status, and modify new statuses to be sent out to users

Fully Dressed Use Case:

Primary Actor: Admin

Goal: Keep Pedway information up to date.

Stakeholders and interests:

Admin: Update Pedway information as necessary

Commuters/Tourists: want up-to-date information for navigation and exploration

Scope: Pedway Status

Level: User Goal

Guarantee: Pedway status should be updated to be in the specified state.

4.1 Precondition:

User feedback has been submitted about a change in the Pedway system.

4.2 Main Flow:

Trigger: The administrator collects submitted user feedback.

The administrator can view all user submitted feedback [S1] and choose to update the database to reflect relevant information [E1]. Once the Pedway section has been updated to reflect its current status/conditions, the user will be able to view the relevant changes on their end.

4.3 Subflows:

[S1] The feedback can be filtered by section or type of feedback.

4.4 Alternate Flows:

[E1] The admin chooses to change the status of a Pedway entrance based on information gathered from somewhere other than the submitted user feedback.

Use Case 5

Use Case Brief: Report feedback about Pedway entrances.

Casual Use Case: Users of the Pedway can provide feedback about entrances of the Pedway, such as closures.

Fully Dressed Use Case:

Primary Actor: Admin

Goal: Report and share Pedway conditions with other users and administration.

Stakeholders and interests:

Admin: Get up-to-date information regarding Pedway conditions.

Users: Improve the navigation experience.

Scope: Navigation

Level: User Goal

Guarantee: Feedback is recorded and saved.

5.1 Precondition:

The user has feedback to report about a specific Pedway entrance.

5.2 Main Flow:

Trigger: The user selects the option to submit feedback.

The user selects a specific Pedway entrance and chooses to provide feedback. They then choose the status of the Pedway to report (closed, open). There is a notes section for the user to write out a detailed description of their feedback. Once the form is completed, the user can submit the form, which will be received by the admins [S1][E1].

5.3 Subflows:

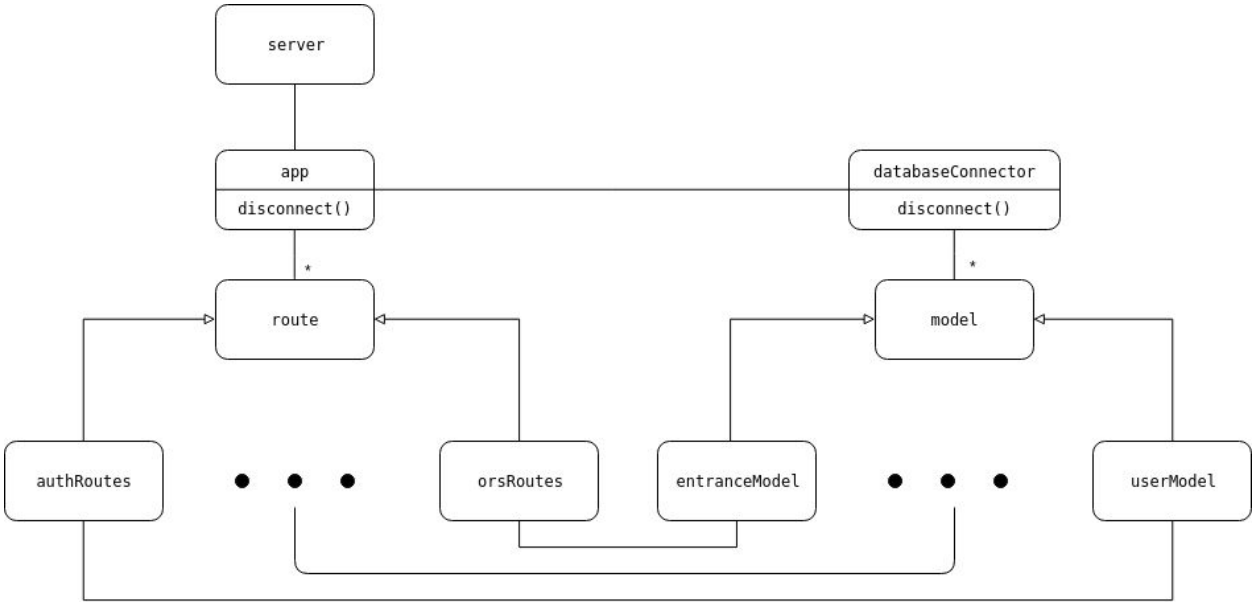
[S1] The admins can choose to update the Pedway information according to the feedback received [UC4].

5.4 Alternate Flows:

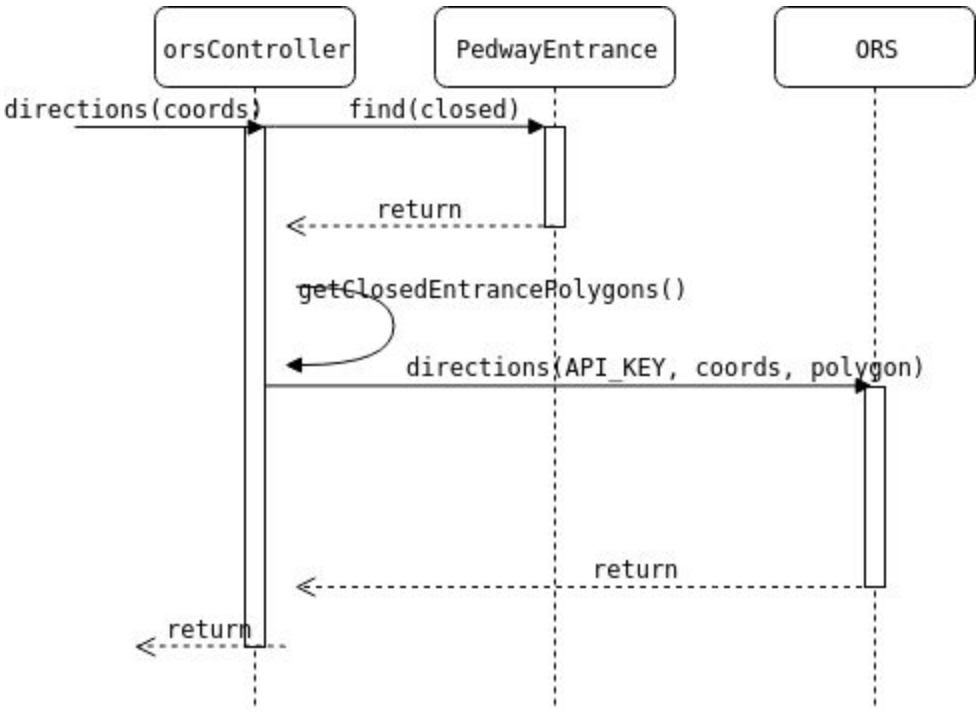
[E1] The user chooses to cancel their form.

Architecture & Design

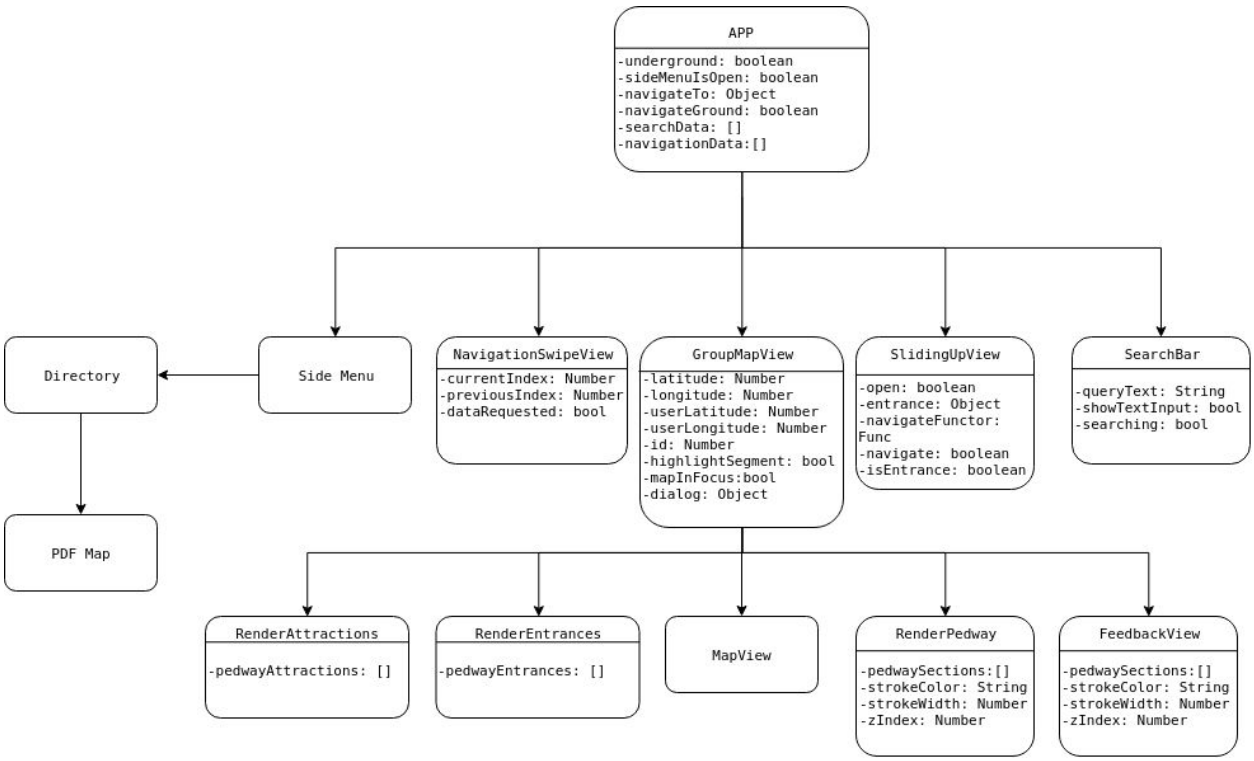
Backend Architecture Class Diagram Overview:



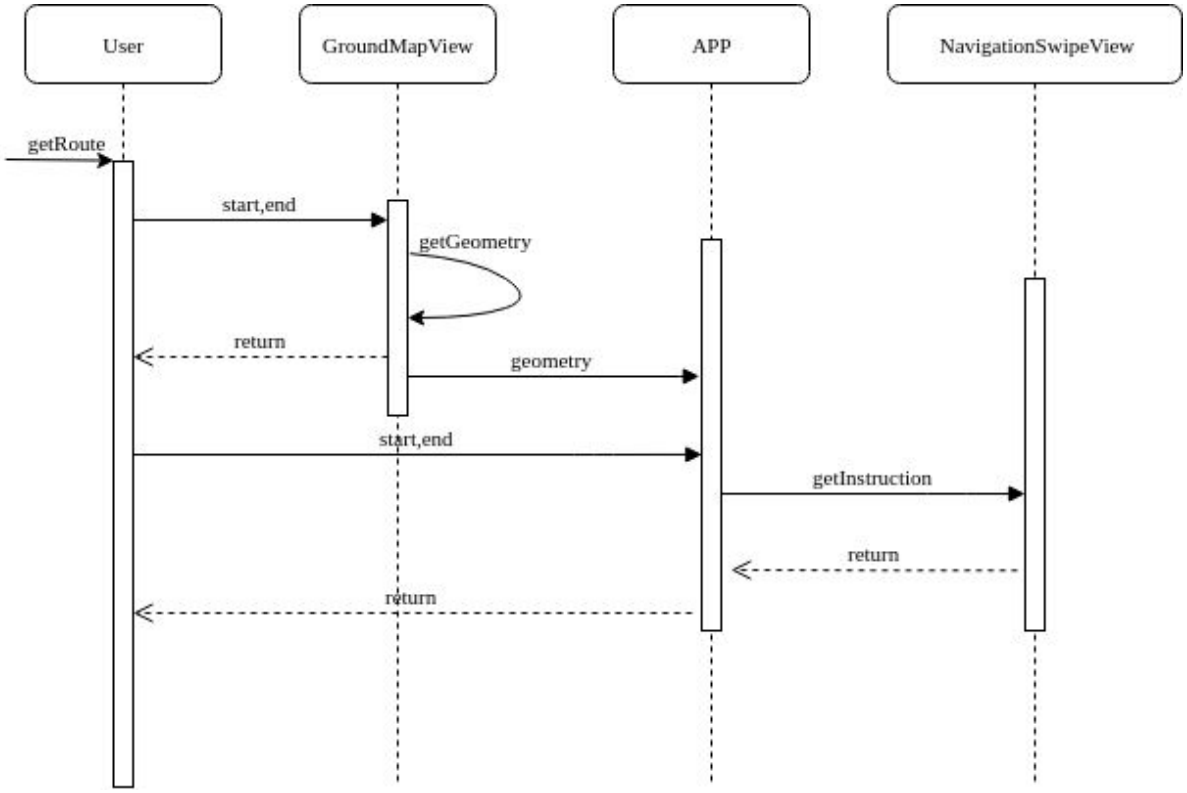
Backend Sequence Diagram for the “directions” Endpoint:



Frontend Architecture Class Diagram Overview:



Frontend getRoute Sequence Diagram



General Discussion

To meet our requirements, our application consists of two major parts: a frontend, consisting of a React Native mobile application, and a backend, consisting of a JavaScript web server. The backend, while visible publicly, is not intended to be visited by the end user; rather, the backend exposes APIs that the mobile app uses for queries about the pedway.

Backend Discussion

The main goal of the backend is to provide a central access point for any information the mobile app may need. To that end, the backend uses two core libraries: Express and Mongoose. Express is a simple web framework, which allows us to write the server entirely in JavaScript. Mongoose is a JavaScript library that connects to a MongoDB database, which was our permanent data storage system of choice.

`server.js` is the main entry point for the backend, which only sets up the ports required to actually run the server. The rest of the setup is done in `app.js`, which sets up the dynamic and static routing for the server. `app.js` also initializes `databaseConnector.js`, which connects to the configured MongoDB database.

All of the remaining components fall into one of three categories: models, routes, and controllers. Models represent the data objects being stored in the database, and also can be used to interface with the database via the aforementioned `databaseConnector.js`. Routes define the API endpoints for the app (see the API documentation in the appendix), along with the HTTP methods supported on each, and route requests to the controllers, which are the final parts of the server and where most logic goes. The controllers take requests from the app, parse them and reply appropriately, after potentially making database calls or making external API calls.

The choice of our framework makes a ‘middleware’ functional style appropriate for the backend, where requests are intercepted, modified, extended, or handled directly. Additionally, because they are rather basic, they work well with a monolithic server, but scaling and flexibility was not that important for the scope of our project.

Frontend Discussion

The front end is built using react native. During the design phase, we need to think in terms of component rather than classes. The root of the project is the `App.js`. Under `App.js` there are 5 major

components: the sidebar, the search bar, the sliding up view that display the information of the selected marker, the navigation swipe view which displays instructions during navigation and the major map view. To be honest, there are flaws in this design because most of the components inside `App.js` need to communicate with the map view. When we try to send data around, we used a lot of call backs in `App.js` as bridges. It would be more efficient and clean if we make some of them to be the children of the map view.

Inside the map view, we have components to render all the entrances, attractions, and pedway itself. We also have a feedback view which will be triggered if the user interact with the sliding up view.

We have two major states to pay attention to. The first is the ground/underground state. This state is maintained both in `App.js` and `GroundMapView`. In underground mode, the user will see a map with dark theme and the glowing pedway sections. Another important state is the navigation state. When the navigation state is set to true, the user will only see the destination and a path to the destination.

In order to achieve modularity in react native, we try to componentize each functional units in the code and reuse them as much as possible. Also, because react native updates the view based on the change of states, we need to also ensure that the right data is fed into the right component precisely. That is why we have a designed centered with our major `App.js` and `GroundMapView`.

Reflections and Lessons Learned

Alex Vanyo (alexv2)

In almost every way, team projects are different than individual projects. When you set out to do something by yourself, by default you control the entire process and make every decision. That absolute power has its advantages, but working alone limits the scope of what you can accomplish. For that reason, working as part of a team is inevitable.

Going into this project, I knew that a team of 7 was going to be challenging, especially due to the nature of our project. We individually had little or zero knowledge with our chosen technology stack, so we had very little familiar ground to work with. Furthermore, since we were building a project completely from scratch, we were all going to be responsible for all decisions made. By trusting in the process, we were able to learn as we went, and were able to create an end product.

Besides the technology, we also learned some tricks to work effectively in a larger team. We quickly realized that we needed to split up tasks and each specialize in some part of the system, so that we didn't step on each other's toes too much. We also learned about the importance of constant communication, as small misunderstandings or unclear requests could quickly result in code that doesn't integrate or other tasks slipping through the cracks.

Curtis Lowder (clowder2)

This project has taught me the importance of proper finance and budgeting. The biggest "mistake" I made had to do with the way our team seemed to burn through our budget. We had to seek 3 or 4 budget increases for our Azure hosting throughout this semester. Even after emailing, reading documentation, dealing with Azure's inadequate webportal, I'm still not sure exactly what we are spending so much money on. Next time, I would spend a week or two just trying to understand the costs associated with each service we utilize.

Besides simple budgeting, I would say this project has showed me that building a full stack application from scratch is a bigger challenge than I initially thought. Writing a few lines of code is pretty easy, but starting from a clean slate and writing code in a way that can lead to a beautiful architecture is much harder. Although I feel we ended up with a decent architecture, I can definitely see ways we can improve our backend architecture. For example, in a real production environment it would be nice to have a microservice architecture that would better deal with redundancy, reliability, and scalability. If I could

change one thing, I would change the APIs such that they require less logic and lines of code from the developers. Right now, there's a lot of similar structure and logic between many of the APIs. Although some of the shared code has been abstracted away, I could definitely abstract much more of the logic into helper functions with a major restructuring of the code.

Ian Howe (ianhowe2)

This course and the project built within it have provided me with invaluable experience in the area of software engineering. Building a fresh project from scratch has given me a much better understanding of how full stack development is done, and how each part of the system interacts with one another.

Most of this learning experience has been gathered from working alongside my teammates, who all brought a unique set of skills to the table. Over the course of the project, I've been able to learn from their techniques and experiences while also ensuring that we remained productive and efficient. This success was a direct result of our decision at the start of the semester to follow the extreme programming (XP) process, but remove any enforced pair programming. This allowed our team to work independently as needed, with the option to work in pairs if it made sense. A notable example would be the occasions where I would meet with Mia to ensure a feature worked on both an emulator and a physical phone, while also relaying any concerns about a feature or possible solutions to troubleshoot a difficult bug. The use of enforced code reviews for pull requests was also very useful towards ensuring that members on the frontend and backend had an understanding and valuable input on all parts of the project. On occasion some pull requests took awhile to be reviewed, however our team did a good job of encouraging reviews to be done at each meeting and ensuring that critical features were reviewed and integrated on time.

The primary concern that he had with our software process was that we had planned an initial set of use-cases for the project, however at that point in time we didn't have a strong understanding of how we would implement our routing and navigation services. Partway through the semester, we had decided on implementing Open Route Service in conjunction with OpenStreetMaps, which caused the nature of our navigation use-cases to change quite a bit. Other than that, I feel that our process for this project was successful, and certainly provided a strong backbone for our team when constructing this project from scratch.

Mia Schoening (miajs2)

I believe our project overall turned out to be successful, as we largely achieved the goals we had established for ourselves during the project ideation process. Because we were able to create a project completely from scratch, we were responsible for also creating the user stories and use cases to be implemented. Thus, this project showed me the importance of creating detailed and thorough requirements and specifications before even beginning the development process, but then also being able to adapt those requirements as needed. Setting up our initial use cases was essential for collectively deciding on the purpose our app would serve to its users, as well as breaking down tasks among our seven team members and constructing an initial timeline. However, it was to be expected that during the development process we would sometimes recognize requirements that needed to be adjusted, whether because we wanted to include additional functionality or instead remove some functionality that was no longer necessary or was not able to be completed within the scope of this course. For this reason, it was essential that we were in constant communication as a team and made these changes to our user stories and use cases as soon as possible.

With regard to the XP process our team followed, I really learned the value of a strong code review process through this project experience. Because we opted to not pair program, having extensive code reviews allowed us to look closely at one another's code and only push changes through to master that met our team's coding standards. In addition, it gave us the opportunity both to practice giving other people constructive feedback on their code, as well as the opportunity to gain valuable insight from receiving feedback from others.

Kexiang Wang (kwang66)

Clearly react native is the most important thing I learned in this project. Before this semester, I knew nothing about react native. It was a hard time to pick it up at the early state of the project. There were lots of struggles and frustration going on, since I basically have to google everything as I was coding. At the later stage, I realized that the early design decision we made is not effective enough to handle the growing code base. If I was to start a new react native project, I probably would spend more time on the structure before the actual coding.

As for the process, I learned the importance of continuous refactoring. I made two major refactor in the semester that brings huge changes to the code base, making it nicer and less confusing. Even

though we had the nice code review process, sometimes we still need to look at the code base from a broader view to maximize modularization and code reuse.

From the experience in iteration meetings, I also learned to let go of the trivial features which may look cool and fancy, but focus on the core feature. It is always important to keep in mind what made the app special.

Yilin Ma (yilinma3)

Unlike the final project in CS427, CS428 requires us to design and implement the project from the foundation. From this project, we learned many new aspects related to the software development process such as project ideation, how to design the architecture and how to conduct code reviews. This is also the first time I worked with React Native. Through working on this project, I learned many details related to React Native and JavaScript itself.

Furthermore, I learned that when working on projects, it is always better to plan and prepare ahead of time. This is because during the actual implementation process, not everything is ideal and there might be many unexpected scenarios. Preparing ahead of time can allow us to have more error margins and be more prepared to solve the barriers in our actual implementation.

David Zhang (yzhng223)

First of all, I would love to give huge thanks to the whole team. I believe that all of us have achieved a lot and learned a lot about the process of software engineering throughout this semester.

Personally, I joined the Pedway team with almost no experience in Javascript, NodeJs, or Express. However, I gradually got familiar with the framework as I was implementing our backend and getting feedback from my awesome code reviewers. Moreover, I have also got more technical experience in Webstorm (the Javascript IDE), MongoDB, Jest (test framework for Javascript), Git (especially in the context of a team project) and CI/CD (Azure pipeline, to be specific).

But what I have learned goes far beyond new technologies. This course is really a great opportunity for teamwork and the best practice for formal, industry-level, software development process. Topics covered both in lecture and actual use throughout the project include eXtreme Programming (XP), code review, iteration meetings, stand-up meetings, and so much more. And finally, we have a really cool real-world application. It is absolutely amazing that we can truly realize Ian’s idea at the beginning of this semester.

With the help from all group members, I am very confident that I would apply what I have learned from this experience to my further careers. Thanks again!

Appendix

API Documentation

General Notes:

- All payloads are passed as JSON strings
- If api fails on authentication, return code 401

`/api/auth`

Handles authenticated sessions using google's API

- GET
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Returns:
    - List of all current sessions on the backend
- POST
  - Creates a new session
  - Prerequisite: None
  - Body:
    - `idToken`=<Google token>
      - Required
  - Error:
    - 400
      - No user exists yet for this userId taken from google
  - Returns:
    - `userId`=<userId of user>
    - `sessionId`=<new session token>
    - `expiration_date`=<date and time>
- DELETE
  - Handles the logout case for a user's current session via the `sessionId` cookie
  - Prerequisite: Must be a session via the `sessionId` cookie
  - Returns:
    - `message`="Successfully logged out"

`/api/auth/:sessionId`

Handles a specific session with `id=sessionId`

- GET
  - Retrieves information regarding this `sessionId`
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Returns:
    - `userId`=<string representing specific user>
    - `sessionId`=<unique id for this session>
    - `expiration_date`=<date and time>
- DELETE
  - Drops session `sessionId` from the database
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Returns:
    - `message`="Successfully dropped session"

`/api/pedway/entrance`

Handles general entrance logic

- GET
  - Returns:
    - a list of all the entrances in the database
- POST
  - Creates a new entrance in the database
  - Prerequisite: Must be a valid ADMIN session via the sessionId cookie
  - Body:
    - `id`=<OSM id>
      - Required
    - `geometry`=<`mongoose.Schema.Types.Geometry`>
    - Created_date=<date>
    - `status`=[`'closed'`, `'dirty'`, `'closing'`, `'open'`]
      - `default`=`'open'`

`/api/pedway/entrance/node/:entranceId`

Handles specific entrance logic

- GET
  - Retrieves a specific entrance's information

- Returns:
  - `id`=<OSM id>
    - Required
  - `geometry`=<`mongoose.Schema.Types.Geometry`>
  - `Created_date`=<date>
  - `status`=[`'closed'`, `'dirty'`, `'closing'`, `'open'`]
    - `default`=`'open'`
- POST
  - Updates a specific entrance's information
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Body:
    - `id`=<OSM id>
      - Required
    - `geometry`=<`mongoose.Schema.Types.Geometry`>
    - `Created_date`=<date>
    - `status`=[`'closed'`, `'dirty'`, `'closing'`, `'open'`]
      - `default`=`'open'`

`/api/feedback`

Handles retrieving general feedback information

- GET
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Returns:
    - a list of all feedback in the database
- POST
  - Creates a new feedback entry
  - Body:
    - `entranceId`=<string>
    - `message`=<string>
    - `reported_status`=[`'closed'`, `'dirty'`, `'closing'`, `'open'`]
    - `type`=[`'status'`, `'bug'`, `'feedback'`]
      - required
  - Returns:

        - `_id`=<id of feedback entry>
        - `entranceId`=<string>
        - `message`=<string>
        - `reported_status`=`['closed', 'dirty', 'closing', 'open']`
        - `type`=`['status', 'bug', 'feedback']`

`/api/feedback/:id`

Handles retrieving specific feedback information

- GET
    - Retrieves information regarding specific feedback entry
    - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
    - Returns:
        - `_id`=<id of feedback entry>
        - `entranceId`=<string>
        - `message`=<string>
        - `reported_status`=`['closed', 'dirty', 'closing', 'open']`
        - `type`=`['status', 'bug', 'feedback']`
- POST
    - Updates information regarding specific feedback entry
    - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
    - Body:
        - `entranceId`=<string>
        - `message`=<string>
        - `reported_status`=`['closed', 'dirty', 'closing', 'open']`
        - `type`=`['status', 'bug', 'feedback']`
- DELETE
    - Deletes a feedback entry from the database
    - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie

`/api/ors/directions`

ORS redirect API

- GET
    - https://openrouteservice.org/dev/#/api-docs/directions/get

`/api/ors/mapsurfer/:zoom/:x/:y.png`

ORS redirect API

- GET
  - https://openrouteservice.org/dev/#/api-docs/mapsurfer

`/api/ors/pois`

ORS redirect API

- POST
  - https://openrouteservice.org/dev/#/api-docs/pois/post

`/api/ors/geocode/autocomplete`

ORS redirect API

- GET
  - https://openrouteservice.org/dev/#/api-docs/geocode

`/api/pedway/section`

Handles general pedway sections

- GET
  - Returns:
    - A list of all pedway sections
- POST
  - Updates information regarding specific feedback entry
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Body:
    - properties
      - `OBJECTID`=<integer>
        - Required
        - Unique
      - `PED_ROUTE`=<string>
        - Required
      - `SHAPE_LEN`=<number>
        - Required
    - `geometry`=<`mongoose.Schema.Types.Geometry`>

`/api/pedway/section/:sectionId`

Handles a specific pedway section

- GET

- Returns information regarding a specific pedway section
- Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
- Body
  - properties
    - `OBJECTID`=<integer>
    - `PED_ROUTE`=<string>
    - `SHAPE_LEN`=<number>
  - `geometry`=<`mongoose.Schema.Types.Geometry`>
  - `Created_date`=<date and time>

- POST
  - Updates an existing pedway section
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Body:
    - properties
      - `OBJECTID`=<integer>
        - Required
        - Unique
      - `PED_ROUTE`=<string>
        - Required
      - `SHAPE_LEN`=<number>
        - Required
    - `geometry`=<`mongoose.Schema.Types.Geometry`>

`/api/user`

Handles generic user logic

- GET
  - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
  - Returns:
    - A list of all users in the database
- POST
  - Handles signup case
  - Body:
    - `userId`=<string>

                - Required
                - Unique
            - `email`=<string>
                - Required
            - `name`=<string>
                - Required

`/api/user/:userId`

Handles specific user logic

- GET
    - Retrieves information regarding a specific user based on `userId`
    - Prerequisite: Must be a valid session via the `sessionId` cookie
    - Body:
        - `userId`=<string>
            - Required
    - Returns:
        - `userId`=<string>
        - `email`=<string>
        - `name`=<string>
        - `permission`=<role>
- POST
    - Updates a user's information
    - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
    - Body:
        - `userId`=<string>
- DELETE
    - Deletes a user from the database
    - Prerequisite: Must be a valid ADMIN session via the `sessionId` cookie
    - Returns:
        - `message`="Successfully deleted user"

Dependencies for the Backend

In addition to the following Node.js dependencies, the backend uses an API key from https://openrouteservice.org/ to make navigation and other geolocation requests.

- @turf/turf: 5.1.6
- body-parser: 1.18.3
- cookie-parser: 1.4.4
- dotenv: 6.2.0
- es6-dynamic-template: 1.0.5
- express: 4.16.4
- google-auth-library: 3.1.0
- mongoose: 5.4.11
- mongoose-geojson-schema: 2.1.3
- request: 2.88.0

Development Dependencies:

- eslint: 5.14.1
- eslint-config-google: 0.12.0
- file-system: 2.2.2
- gulp: 4.0.0
- gulp-eslint: 5.0.0
- jest: 24.1.0
- jest-environment-node: 24.6.0
- jest-image-snapshot: 2.8.1
- jest-junit: 6.3.0
- jest-puppeteer: 4.1.1
- mongodb-memory-server: 4.0.1
- nodemon: 1.18.10
- puppeteer: 1.14.0
- superagent: 4.1.0
- supertest: 3.4.2

Dependencies for the Frontend

All the packages are installed using npm and the corresponding link/documentation can be found on npmjs.com. Here are the dependencies:

- @turf/distance: 6.0.1,
- @turf/helpers: 6.1.4,
- @turf/point-to-line-distance: 6.0.0,
- @turf/turf: 5.1.6,
- appcenter: 1.12.0,
- appcenter-analytics: 1.12.0,
- appcenter-crashes: 1.12.0,
- axios: 0.18.0,
- babel-eslint: 10.0.1,
- google-polyline: 1.0.3,
- isomorphic-fetch: 2.2.1,
- jest-matcher-deep-close-to: 1.3.0,
- lint: 1.1.2,
- react: 16.6.3,
- react-native: 0.58.5,
- react-native-action-button: 2.8.5,
- react-native-gesture-handler: 1.0.16,
- react-native-maps: 0.23.0,
- react-native-popup-dialog: 0.18.2,
- react-native-side-menu: 1.1.3,
- react-native-swiper: 1.5.14,
- react-native-vector-icons: 6.3.0,
- react-navigation: 3.3.2,
- rn-sliding-up-panel: 2.0.2

Development Dependencies:

- babel-core: 7.0.0-bridge.0,
- babel-jest: 24.1.0,
- babel-preset-react-native: 5.0.0,

- eslint: 5.14.1,
- eslint-config-google: 0.12.0,
- file-system: 2.2.2,
- gulp: 4.0.0,
- gulp-eslint: 5.0.0,
- jest: 24.1.0,
- jest-junit: 6.2.1,
- metro-react-native-babel-preset: 0.51.1,
- react-native-jest-mocks: 1.4.0,
- react-test-renderer: 16.6.3

Auto-Generated Interface Documentation for the Frontend

The auto generated documentation for the frontend can be found in pedwayApp/docs.zip

Auto-Generated Interface Documentation for the Backend

The auto generated documentation for the frontend can be found in backend/backend-doc.zip