**CS3354 Software Engineering**

**Final Project Deliverable 2**

Group 4

UTD Campus Pathfinding Application

Abdullah Akbar

William Bumpass

Jenain Khan

Vishakh Nair

Benjamin Rubarts

Michael Villordon

Adam Wajahat

1. Task delegation

* Address feedback to the project proposal: William Bumpass, Abdullah Akbar, Benjamin Rubarts
* Creating Github repository (3354-group4) and adding group members and TA to it: Vishkah Nair
* First commit to the repository (README file): Jenain Khan
* Second commit to the repository (“project\_scope” file): Adam Wajahat
* Commit to the repository (Project Proposal): Adam Wajahat
* Commit to the repository (Deliverable 1): Michael Villordon
* Commit to the repository (Deliverable 2): Benjamin Rubarts
* Commit to the repository (JUnit test code): Adam Wajahat
* Commit to the repository (Presentation slides): Benjamin Rubarts
* Add link to repository: Michael Villordon
* Software process model evaluation: Jenain Khan, William Bumpass, Vishakh Nair, Benjamin Rubarts
* Requirement evaluations:
* Functional: William Bumpass
* Non-functional: William Bumpass, Abdullah Akbar
* Use Case diagram(s): Adam Wajahat, William Bumpass
* Sequence diagrams: Michael Villordon, William Bumpass
* Class diagram: Benjamin Rubarts, Michael Villordon
* Architectural design (choosing and “implementation”): Abdullah Akbar, Michael Villordon, Jenain Khan
* Project Deliverable 1 content insertion: Jenain Khan
* Presentation delivery information: Benjamin Rubarts, Jenain Khan
* Project scheduling: Benjamin Rubarts, Adam Wajahat
* Cost methods details and price of project: Adam Wajahat, Benjamin Rubarts
* Costs of hardware: Vishakh Nair
* Costs of software: Jenain Khan
* Costs of personnel: Michael Villordon
* Effort “evaluations”: Vishakh Nair
* Test Plan for Software: Adam Wajahat, Vishakh Nair, Benjamin Rubarts
* Comparison to similar designs: Abdullah Akbar
* Conclusion: Michael Villordon
* References: Everyone
* Presentation Creation: Everyone
* Implemented Code (not currently implemented): Everyone

1. Project 1 Deliverable content

1. Final Project Draft Description:

Optimization of paths from one point to another can be tricky, especially on the UTD campus where navigation is often complicated, and the shortest path often difficult to find. Thus, in an effort to make campus navigation more friendly, we propose an application that would allow for the quickest route across campus to be decided with just a start and end point. Paths - intended to be taken by those on foot - would be similar to other pathfinding applications such as Google Maps, though limited only to the UTD Campus and surrounding amenities (i.e. testing center).

* Would be a useful tool for campus guests or those adjusting to the changing campus
* Could be expanded upon to larger areas around campus to include distant apartments
* Setting path navigation ahead of time could allow for pre-planned paths to display at certain times
* Could be integrated into the UTD Mobile app, allowing for more concise school tool usage
* Allows for color coding or categorization of recurring routes
* Sharing of routes and directions through the app and other mediums
* Accommodates routes affected by restrictions (disability, shutdown areas, construction, etc.)
* Creation of agendas for daily routes utilizing calendar application integration (such as Google Calendar)
* User path preferences may be set (prioritize inside walkways vs. outside walkways, etc.)

Proposal Feedback:

Good proposal and fair distribution of tasks to group members.

In your final project report (del﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿﻿iverable 2) please make sure to include the following:

- A thorough search to find similar application implementations. Please cite these work using IEEE citation format provided on Final Project Specifications document.

- Please make sure to differentiate your design from existing similar applications by including extra features into it.

- Please make sure to explicitly specify those differences by comparing your design with those existing similar applications.

Thank you

Feedback Response:

* We will attempt to do a thorough search on the implementations of similar applications. Our main target for observation would be Google Maps - given its popularity and similarity in nature - though it will not be the only case. We will also analyze the Starship delivery app, which sets routes for the delivery robots according to a map of the campus. As we address the work of these implementations, we will cite them using the IEEE citation format.
* Our project will be different from other similar applications through the following features:
  + In order to make our application more student-oriented, we will allow users to input routes between specific classrooms instead of just buildings. Including inter- and intra-building maps allows for users to find harder-to-access classrooms.
  + To allow for more user-friendly reporting, people can submit various blockages and obstacles found along a path. This will allow others to note the blocks on their paths. Few other applications allow for the direct and intentional submitting of accidents and other obstacles.
  + We will also integrate student life with the application; users can submit events happening on campus and they will appear on the campus map. Clicking on the events will show the closest routes to get to them.
  + Our project would be integrated into another application (UTD Mobile) so the database could autoupdate with information about events and other possible obstacles on paths.
* In addition to the distinct features, we will mention which features also appear in the application implementations we are comparing, as well as which features we are leaving out in our application.

1. Github

Link To Project Repository: <https://github.com/vishakh15n/3354-group4>

1. Task delegation

* Address feedback to the project proposal: William Bumpass, Abdullah Akbar, Benjamin Rubarts
* Creating Github repository (3354-group4) and adding group members and TA to it: Vishkah Nair
* First commit to the repository (README file): Jenain Khan
* Second commit to the repository (“project\_scope” file): Adam Wajahat
* Commit to the repository (Project Proposal): Adam Wajahat
* Commit to the repository (Deliverable 1): Michael Villordon
* Commit to the repository (Deliverable 2): Benjamin Rubarts
* Add link to repository: Michael Villordon
* Software process model evaluation: Jenain Khan, William Bumpass, Vishakh Nair, Benjamin Rubarts
* Requirement evaluations:
  + Functional: William Bumpass
  + Non-functional: William Bumpass, Abdullah Akbar
* Use Case diagram(s): Adam Wajahat, William Bumpass
* Sequence diagrams: Michael Villordon, William Bumpass
* Class diagram: Benjamin Rubarts, Michael Villordon
* Architectural design (choosing and “implementation”): Abdullah Akbar, Michael Villordon, Jenain Khan
* Project scheduling: Benjamin Rubarts, Adam Wajahat
* Cost methods details and price of project: Adam Wajahat
* Costs of hardware: Vishakh Nair
* Costs of software: Jenain Khan
* Costs of personnel: Michael Villordon
* Effort “evaluations”: Vishakh Nair
* Test Plan for Software: Adam Wajahat, Vishakh Nair
* Comparison to similar designs: Abdullah Akbar, William Bumpass
* Conclusion: Michael Villordon
* References: Everyone
* Presentation Creation: Everyone
* Implemented Code (whole code or just JUnit test code): Everyone

1. Software Process Model Evaluation:

We are using the V-model, a variation of the waterfall model, in our project. We intend not to cut corners and simply release multiple versions as in the prototype model, add separate features at different times and find ways to bring them together as in the incremental process model, or release more progressive prototypes as in the spiral model. We intend to plan everything out one step at a time and release a complete application, but we also want the flexibility of not having to start completely over when we detect a mistake.

The V-model Software process is based upon the concept of verification and validation where testing occurs at each level of development. The next phase can only start once the previous phases are complete and correct making the V-model simple, efficient, and best suited for smaller projects, such as our Pathfinding Application. Since we have clearly defined requirements for our Pathfinding Application as mentioned in our description the process of the V-model of going from broad to refined matches our application well, as we can go from requirement analysis to design phases. We also found it to be advantageous how the V-model allows for errors to be caught in earlier phases.

* + 1. Pros:
       1. Fully documents code
       2. Validates and reviews at all levels
       3. Good for smaller projects
    2. Cons:
       1. Requires long specifications
       2. A large amount of time spent planning
       3. No iterations
       4. Intensive testing after each step
       5. Poor with larger projects

1. Requirement Evaluations:

Functional:

* + 1. The user shall be able to request paths for a desired route
    2. The webserver/database shall be able to handle multiple requests in parallel
    3. The webserver/database shall be able to gather information on Comet Cab routes
    4. The webserver/database shall be updated with information on blockages through construction and accidents
    5. Users shall be able to store their recent route requests for review/reuse later.
    6. Routes should be generated within 10 seconds of query at maximum
    7. User preferences for routes will be selected through a list format, allowing for multiple selections over different categories (disabilities, inside/outside)

Nonfunctional:

I. Product Requirements

* + - 1. Usability Requirements:
         1. The app interface must be easy enough for a regular student or faculty member to use for general purposes
         2. The interaction with the map display should be similar to other apps that use map routes, like Google Maps or Uber.
      2. Efficiency Requirements
         1. Performance Requirements:

The mobile app should be able to run on both iOS and Android devices, provided they have up-to-date operating systems and are from after 2010

* + - * 1. Space Requirements:

Google Maps takes up a little less than 400 Mb of storage, so our app should take up no more than half a gigabyte of storage.

* + - 1. Dependability Requirements:
         1. As our app is nonessential, a continuous downtime of at most 24 hours is permissible. We should do monthly maintenance checks on the servers to make sure no errors cause the downtime to exceed this limit. Furthermore, the webserver/database should be able to handle 10000 concurrent users without affecting its performance.
      2. Security Requirements:
         1. In order to enhance ease-of-access and ensure only enrolled students/faculty are setting up events, we would want to integrate our app with the UTD NetID system. This means that our backend authentication system must meet the security requirements the Office of Information Technology requires in applications integrated with UTD NetID.
         2. Communication between the mobile application and the backend servers should be encrypted to ensure the privacy of the user’s input.

II. Organizational Requirements:

* + - 1. Environmental Requirements:
         1. The frontend application should work fundamentally the same on iOS and Android so that moving from one OS to another does not cause any inconveniences for the users.
      2. Operational Requirements:
         1. The routing feature of the software should allow people on campus to select two locations on campus and have the software calculate and display the optimal path between both points.
         2. The location search bar should also save past locations searched so the user can find the current route to them with just one tap (like Google Maps saves previous queries). Such locations should be saved locally on the user device to preserve privacy.
         3. The event organizing feature should allow users to add events and their locations, and then specify who can view the event details (public or limited to specific users). Then the respective users should see the event pop up on the campus map, along with its details.

3. Development Requirements:

1. The software should include a backend server system and a frontend mobile app system that should run on both iOS and Android.

III. External Requirements

1. Regulatory Requirements:
   1. Privacy of information, the export of restricted technologies, intellectual property rights, etc. with regards to the software should be audited by the University of Texas at Dallas.
   2. Any updates to the software’s privacy policy or any changes made with regards to how user data is handled will need to be approved first by the appropriate boards of the University of Texas at Dallas.

2. Ethical Requirements:

1. User privacy is a top priority and NetID’s will only be used to ensure any user organizing a campus event is actually a UTD personnel.
2. An “About” page shall provide general information regarding the implementation of the app, as well as detailed information of the processes which occur through the app.

3. Legislative Requirements

1. Accounting Requirements:

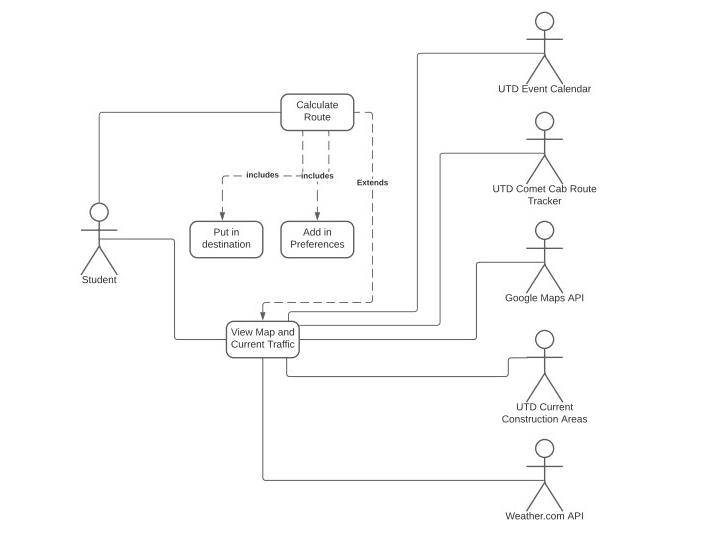
All funding regarding the app should be agreed on beforehand between us, the app developers, and the University of Texas at Dallas. Since no money is involved in the actual app, the only accounting to consider is the money required to maintain the necessary app servers.

b. Safety/Security Requirements:

i. Users should be able to report any misconduct that happens at an event displayed on the app; such reports should be immediately forwarded to the authorities.

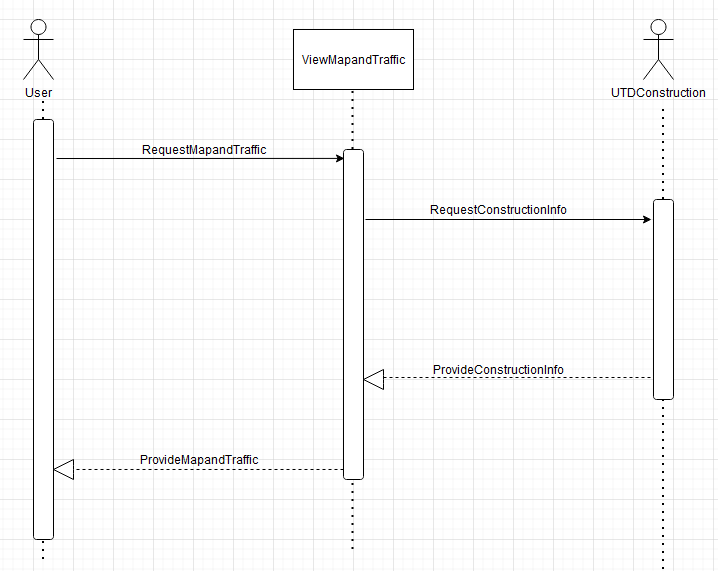
A no-leniency policy will be put in place so that organizers of illicit events on the app will be removed after one offense.

1. Use case diagram:

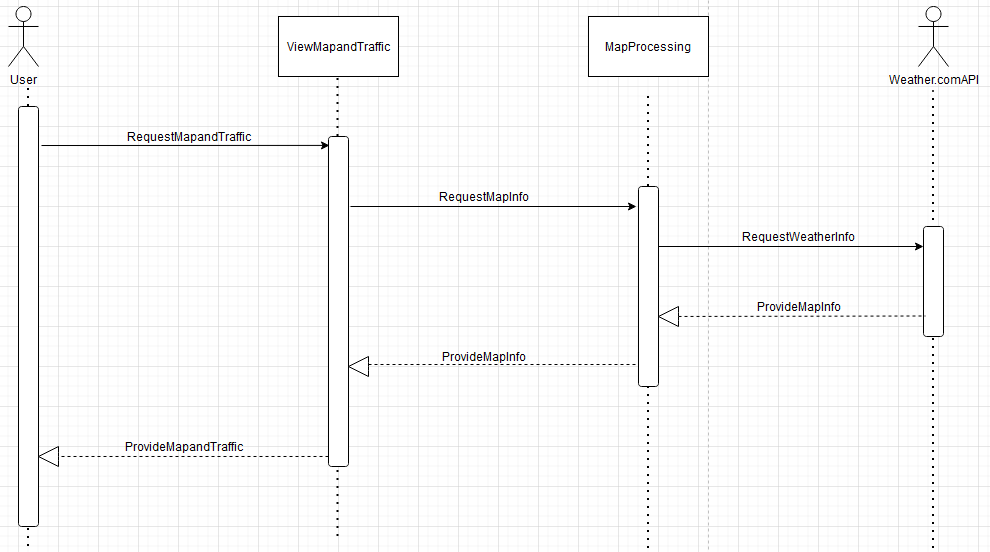


1. Sequence diagrams:

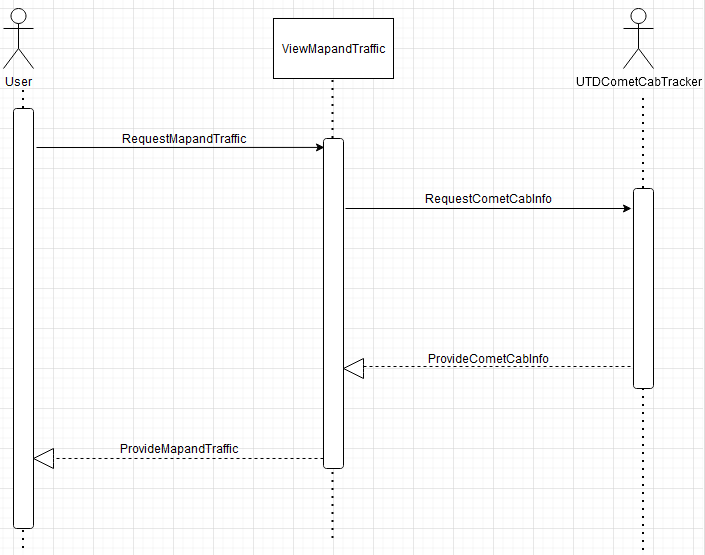
(for View Map and Current Traffic use case in connection with UTDConstruction)



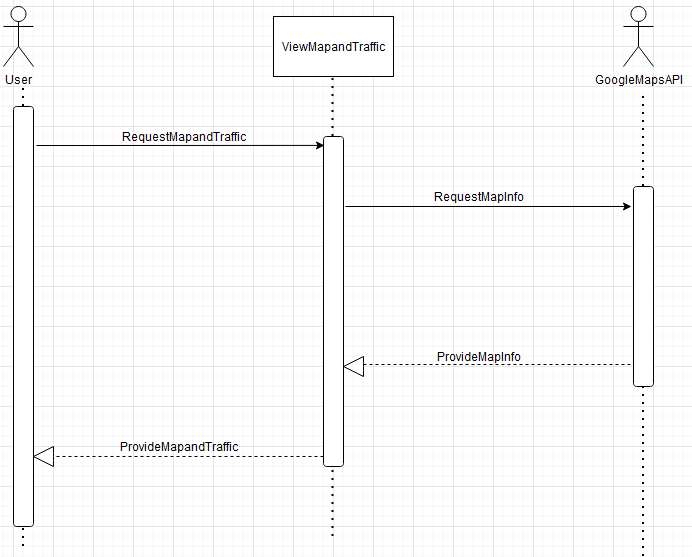
(for View Map and Current Traffic use case in connection with the Weather.comAPI)



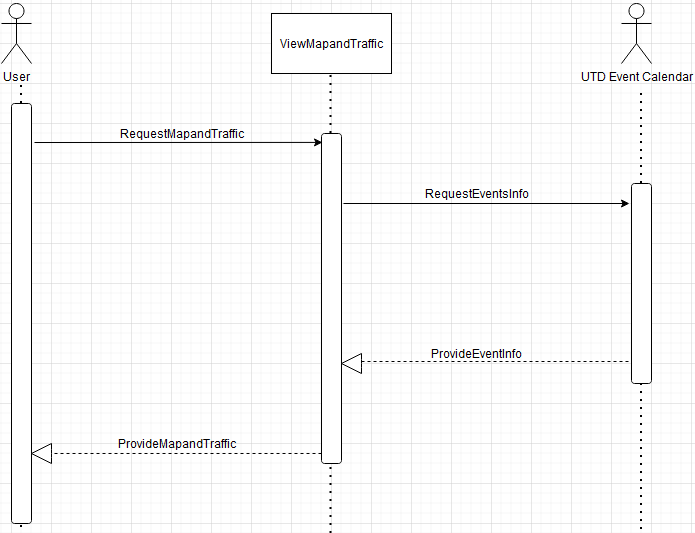
(for View Map and Current Traffic use case in connection with the UTDCometCabTracker)



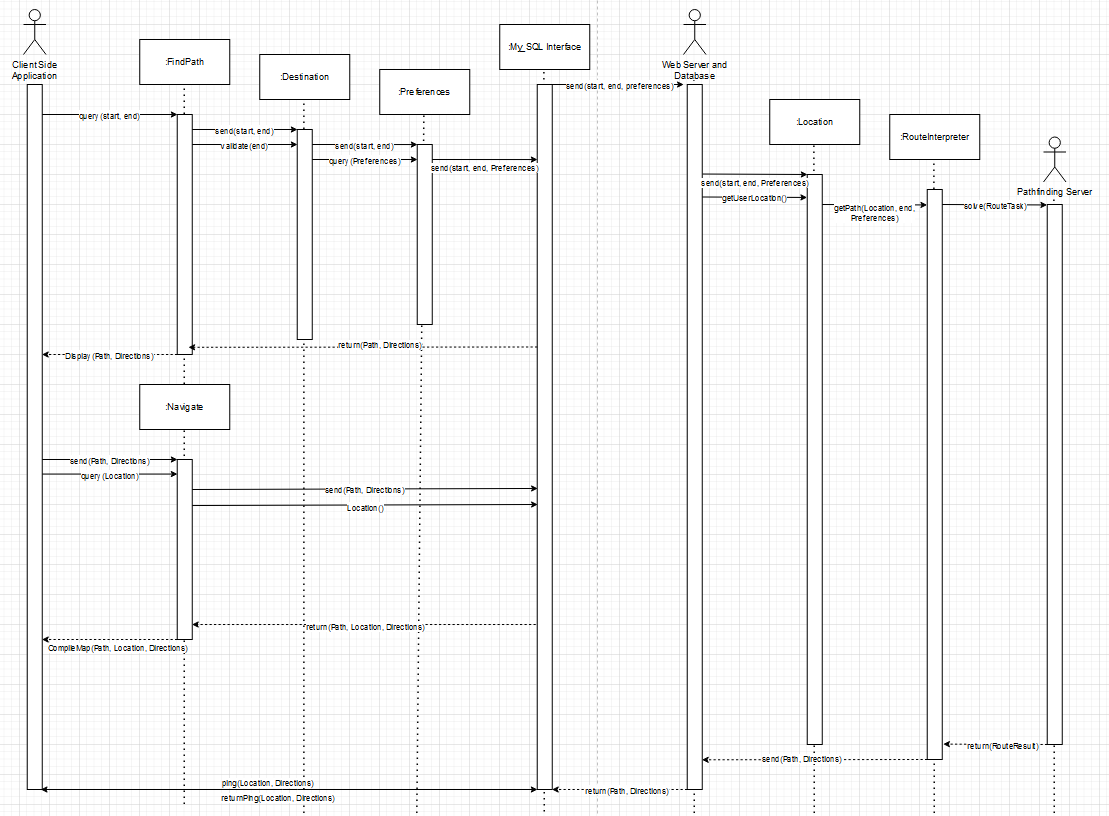
(for View Map and Current Traffic use case in connection with the GoogleMapsAPI)



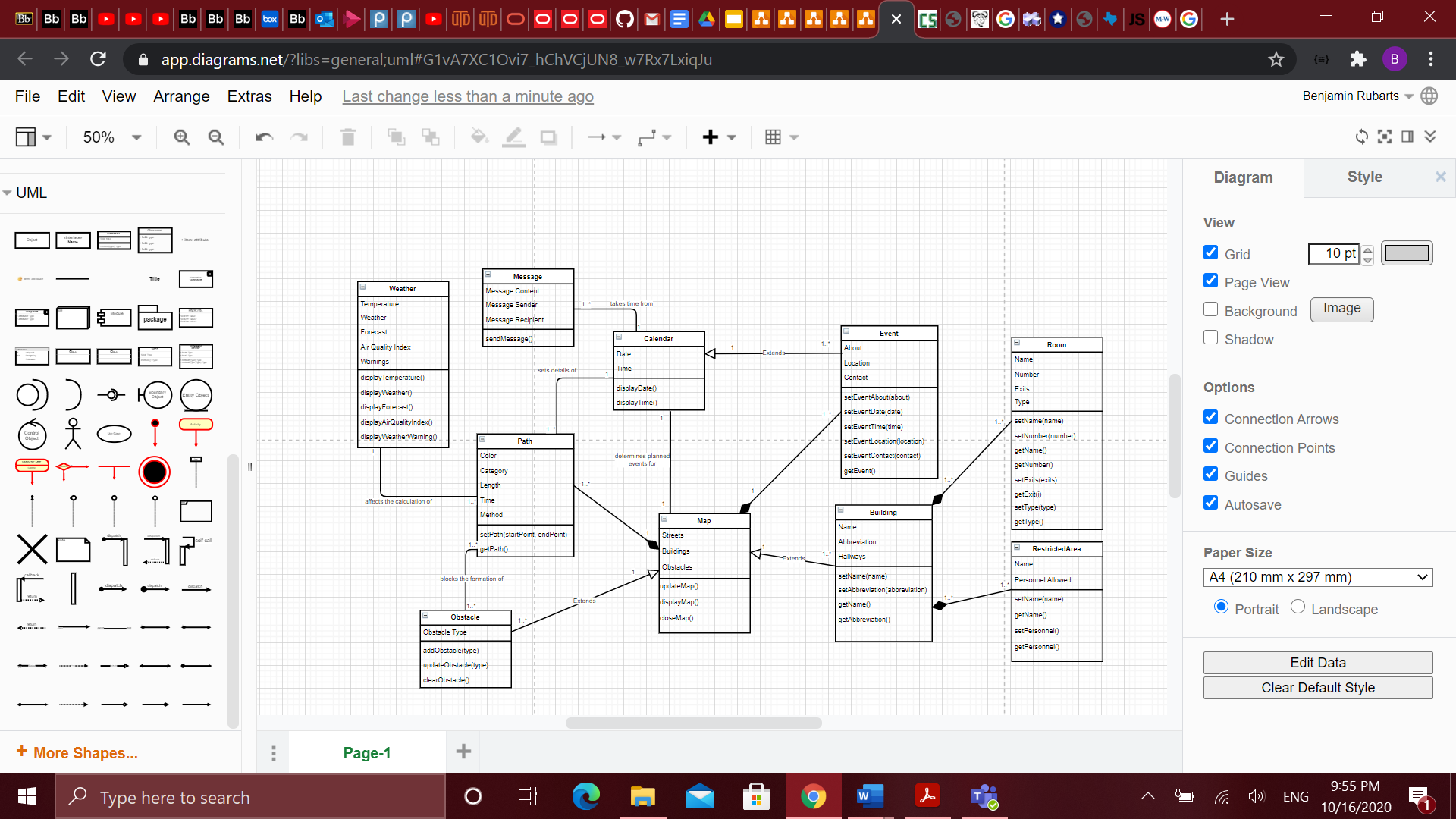
(for View Map and Current Traffic use case in connection with the UTD Event Calendar)



(for the Calculate Route use case; features included use cases Put in Destination and Add in Preferences)

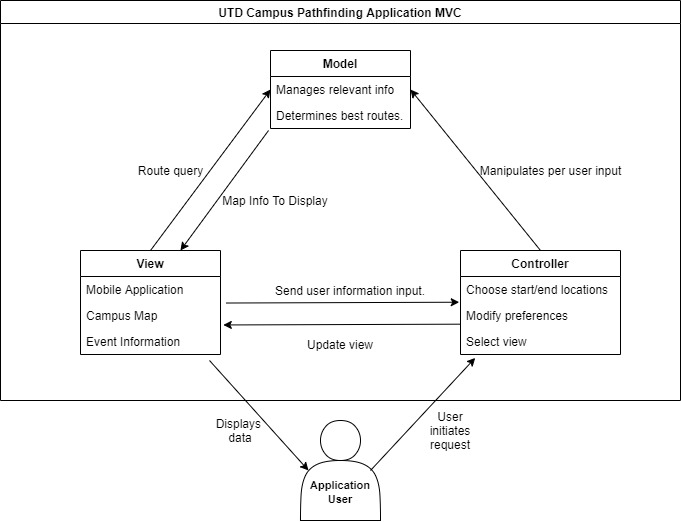


1. Class Diagram:



1. Architectural Design:

Model-View-Controller Pattern (MVC)



1. Choice Of Presentation Delivery Type

We pre-recorded our presentation, provided captions to it, and uploaded it to Microsoft Stream (Option 2). The URL is: <https://web.microsoftstream.com/video/cf448a5e-b731-4007-9f47-1d76ce48e520>

1. Project Scheduling, Cost, Effort and Pricing Estimation, Project Duration and Staffing

4.1. Project Scheduling

We are estimating that we will begin this project on January 4, 2021. This is to match up with the new quarter for the fiscal year. Based on our estimations, we are expecting a team of 7 moderately experienced programmers to be working on the project for about 2 weeks. This is assuming no weekends will be counted and there is a 30 hour work week/6 hour work day. Originally, our calculations were based on every day being a work day; however, we realized by giving an extra week, we can give more time as a cushion and put less stress on the employees. The project should be completed by January 16, 2021.

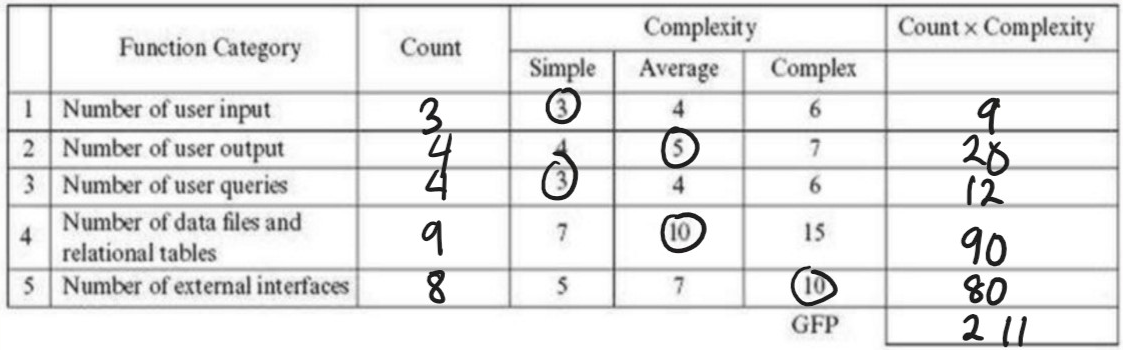
After starting, we would spend the first 3 days having half the team work on a routing algorithm. This algorithm would have to take into account different parameters for the events and weather and be able to calculate the shortest path. The other half of the team would be working on creating modules that source that data from the different services. This effort should take about 2 days and includes third-party services such as Google Maps, the Weather API, the UTD Comet Cab live feed, and the UTD event calendar. Once finished with the communication modules, the team of 3 would start working on the UI of the app. This would mean there is a day's worth of slack for the communication modules team because the routing algorithm team needs one more day to complete its task.

Once the routing algorithm team has finished in the allocated 3 days, the next 2 days would be spent working on the app UI and connecting the buttons to the different components of the routing function in which the whole team would be a part of. This would also mean that we would need additional functions to accommodate for the preferences and connecting that back to the routing algorithm. The last 3 days would be spent in testing. Each of the steps of testing are highlighted in another section.

4.2. Cost, Effort, and Pricing Estimation

We calculated the estimated cost using the Function Point method.

Step 1:



Number of User Input: 3

- Destination

- Preferences

- Saved Locations

Number of User Output: 4

- Route

- Map

- Events (Construction, Comet Cabs, Campus Events)

- Weather

Number of User Queries: 4

- Route

- Log-In

- Events

- Weather

Number of Data Files and Relational Tables: 9

- Map

- Buildings/Rooms

- Events

- Calendar

- Route

- Obstacles

- Weather

- Preferences

- Classes

External Interfaces: 8

- Google Maps

- Weather

- Construction

- Comet Cab

- Event Calendar

- Database Storage

- Networking

- App Store Fronts for Deployment

Step 2:

(1) - 2 There is little to no reliable backup and recovery needed. The most that the system needs to do is save preferences, which can easily be added each time the user opens the app.

(2) - 5 A majority of this system is reliant on data communication as that would be the main way of giving the user the best route possible. Calculating routes based on traffic, available Comet Cabs, events, and such is the main benefit this app is giving the user.

(3) - 4 Yes. Each individual app user will have its own processing built into the app. This would also put less strain on a system if we had to do a single server that would do the calculations for each user.

(4) - 4 We think that the performance of the app is not all that important but it should definitely get the user a route estimation in a reasonable time. If the user can walk to their destination before the route can load up, then there is no point of the app. It is made to save time.

(5) - 2 Most of the information is being pulled from different services. All computing is done on the local device.

(6) - 4 Very little other than user input. These are passed to the app to compute which is done locally. There is information that must be pulled from other services.

(7) - 1 There are two separate screens where the user can put in input. One is the settings screen, which is where they can save preferences, and the other is in the main app section, where they can put in their destination.

(8) - 4 All master files will be put on an online server for the apps to access and will be updated through those means; however, route processing is done locally.

(9) - 5 Inputs and outputs for the app can be complex. The system needs to take into consideration the user input as well as the input received from different services. Then, the system needs to compute all these inputs together and then output a route. The route is simple; however, the inputs taken into consideration are not.

(10) - 4 The general idea for the routing process is not complex; however, there are quite a bit of variables the system needs to take into consideration when sketching a route.

(11) - 3 Some of the routing processing code be reused from other sources but much of the code is specific to this situation. It might be able to be reused for other campuses.

(12) - 4 As it is a mobile app, the installation is simple but needs to be able to take into consideration the fact there are different OS devices that will have access to this. The app needs to accommodate that OS.

(13) - 2 This design is specific for our scenario of UTD. Other institutions could implement the app; however, it is originally not made for them.

(14) - 5 As this app is used to make the user's experience on campus better, it must be easy to use and navigate. That was the thought process when designing the app: to make it simple.

Step 3:

PCA = 0.65 + 0.01(2+5+4+4+2+4+1+4+5+4+3+4+2+5)

PCA = 1.14

Step 4:

FP = GFP X PCA

FP = 211 X 1.14

FP = 240.54

Step 5:

We are assuming 30 function points for our team of 7 moderately experienced programmers.

240.54/30 = 8.018 (or 9) person-weeks

With 7 people, 9 / 7 = ~2 weeks

Going based off of a $5000/month pay with 7 employees, this would mean for two weeks of work, we can estimate ~$1700/person for the duration. Also, this would mean that the total cost of labor would be ~$11,700. Nevertheless, we intend to make the application free at the point of installation just like UTD Mobile. We estimate UTD will compensate us as a portion of its employment costs.

4.3. Hardware costs:

In terms of hardware, we don’t have many expenses in that regard. Our main project focuses on the software side and the only hardware necessary would be at least one computer to store the data and any given smartphone or laptop. That said, any computer needed for coding the actual app could also be considered as cost for hardware. This could range anywhere from $600 to $3000 in terms of software. However, while we suspect that we won’t need a supercomputer in order to code this, we feel like a strong PC is probably required, which would estimate the cost of hardware at around $2000.

4.4. Software costs:

*Google Maps*

Our team plans on using Google Maps SDK to implement the map system on our application. Google Maps Platform charges vary based on monthly volume range (per thousands) and loads. Google Maps Platform also charges differently based on specific SKU’s.

For our UTD Pathfinding application we will be using “Mobile Native Dynamic Maps” of Google maps since it will be used within an app. With this SKU option, our $200 monthly credit equivalent free usage gives us unlimited loads of usage. As for our monthly volume range in thousands, since our application is specific to UT Dallas, we can expect only staff and students to use it (total population approximately around 30,000). Taking into account use of the application on multiple devices and outlier users, we would still fall below the 100,000 volume range. This gives us a total cost of $0.00 for our monthly volume pricing.

For further use of elements such as “Static Street View,” we will be charged a monthly cost of $7.00 per thousand people, giving us a monthly charge around $210.00.

For use of the Distance Matrix API, we can use up to 40,000 elements for free, but for volume-based charges will be charged $5.00 per thousand (total $150.00). We will not need the Routing API because we will be developing our own.

Within Google Maps, for the Places API, there is a separate Place SDK, which we will make basic use of. For elements such as Autocomplete, paces Details, and Basic Data, we will be charged $17.00 per month per thousand users. This will charge us $510.00 per month. If finances do not allow, this could be removed from the application. [1]

*Google Calendar*

We will also have a calendar component for scheduling a user’s classes and route times. The Google Calendar API offers 1,000,000 queries per day with no charge, which we will be under.[2]

*UTD Mobile App*

For integration into the UTD Mobile application, since it has been developed through the UTD Digital Transformation group, it is likely integration would be favored. This would bear no extensive cost to us, as UTD Mobile has a larger client base.

*Weather*

We will also be including a weather feature in our application. To do so, we have decided on Weather API due to its extensive features with low pricing. The developer package allows 2,000,000 calls per month with 5 day, weekly, daily, and hourly weather reports all for $4.00 a month.[3]

*Total*

Our total monthly software cost for licenses, SDK’s and API’s totals to $564.00 without the Places API, and $1074.00 with the Places API.

4.5: Estimated cost of personnel

This app primarily requires knowledge of the MVC Web Application model. Microsoft’s exam for an ASP.NET MVC Web Application certification costs $165. For 7 people, that is 7 \* 165 = $1155. Other than this, there are no other major personnel training costs. [4]

1. Test plan

Our software testing plan is made up of 4 main steps. We have Unit Testing, followed by Integration Testing, then Validation Testing and finally System Testing.

For unit testing, we decided to go with Black-box testing. This is because Black-box testing is simpler and would allow us to just test our modules based on only the domain of inputs. In Black-box testing, there are many methods of testing; however, the one we went with was Equivalence Partitioning. For our project, the inputs are very specific. For the routing part, there can only be locations inside and outside of UTD. The partition would be a lined map of the area that is considered a part of the UTD campus. For the preference module, we would test each of the preferences. This means when the indoor is selected and when it is not. Also, we would test the Comet Cab selection to see if it uses Comet Cabs on the way to the route. Finally, we would test the events module. This includes taking into consideration the weather, seeing if the route will consider if it's raining or not, and modifying the route accordingly. It will also test the calculation of routes, to determine whether or not the events are scheduled on that day, to avoid that traffic.

For our integration test, we decided to use a top-down approach. This means that we test the main module with “dummy” parts and keep testing parts till it works. For our project, our main module would be the routing (pathing not Internet) process. For our first test, we would need to check if our basic pathfinding algorithm works by running the program for 2 points and seeing if we get the correct shortest path. Once that is working, we need to add each component in one at a time. Next, we would check if we can get the shortest path from our current location to a specified point. Then, we would have to check if we can look for an event to find the shortest path. Keep in mind, this also checks our input. Then, we can add a construction zone in our current path and see if the shortest path will change. After this, we still need to check if our weather section works so we will see if we can get a path that is only indoors. Finally, we will check if our paths can be displayed on the map for the world to see. We will also need to see if our log in works, but considering that it is the same as every other log in, there shouldn’t be any problem. Our JUnit example test code determines whether destination information is successful and correct.

Validation testing is when the individual components are put together to see if everything can work together. This is to iron out any interface issues. At the beginning, we would deploy the app onto the employee’s phones. This would be considered alpha testing. We would take into consideration phones that support the Google Play store and the Apple’s App Store. This would allow us to hit the majority of phones on the market. Next, we would move onto beta testing. This would be done by deploying the app to a certain group of testers. This is where we would be testing the interface on different phone types, such as iPhones, Samsungs, Androids, and Pixels. Each of these phones has a different screen and base running OS. Beta testing would help us test with a wider range of devices the app might be deployed on.

For our system test, we have to test our product as a whole. This will require at least two tests. Our plan is to first test an event that goes through construction, and then test a classroom that will take an indoor path due to the weather.

1. Comparison

In comparing our design with other designs, we looked specifically at Google Maps, Starship Food Delivery, and Snapchat.

*Google Maps*

Google Maps is by far the most well-known real-time mapping software, which allows users to input two places and get the fastest route between them, just like our application does. This then raises the question: What differences does our application have from Google Maps? Upon our analysis of Google Maps, we realized that while it is good for navigating around cities and larger areas, particularly those connected by roads, on a smaller scale, like the UT Dallas campus (where every building is connected by footpaths rather than roads), the application does not consider real-time factors as well. That is to say, whereas Google Maps can identify construction and traffic jams on a larger scale, thereby suggesting different fastest routes at different times, it will always suggest the same route for paths between classes, even if new construction zones have popped up. This is because Google Maps uses the Department of Transportation data and collects speed and location data from users’ phones to figure out where traffic jams are [5].

In our application, we would replace the DOT data with our own database of obstacles (construction, gas leaks, event set-up locations, etc.). We could also replace traffic jams with warnings for students about “busy hours,” usually in between class periods, when a lot of people would be using the walkways to get to class. Nonetheless, despite its differences, Google Maps has several features we would like to incorporate into our application (especially the ability to display a map of campus and draw routes on it), so we would almost certainly utilize the Google Maps API in our application (similar to how Uber incorporated it). The Google Maps API is not free, so we did have to factor the pricing into our cost analysis, but we agreed that the services it offers justify paying for it instead of trying to develop our own map system/infrastructure.

*Starship Food Delivery*

We looked next at the Starship Food Delivery app, which also uses the Google Maps API but focuses on the campus scale, just like our application. In particular, we were interested in Starship, as opposed to other campus delivery apps, because it has been implemented for the UT Dallas campus, which is the first campus we would like to implement in our application. In determining the best routes for its delivery robots, Starship looks at real-time campus data, noting obstacles and other closures. Starship itself uses its delivery robots to gather this obstacle data; in our application we would partner with the campus administration to receive and input this data into our obstacle database, as well as allow users to submit obstacle reports and integrate with other campus applications from which we could pull similar data [6]. We analyzed how Starship allows users to specifically place a pin anywhere on the campus map to serve as the delivery point for their food; in our application, we would need two such points for the start and destination. Of course, we don’t need any of the menu/restaurant listing abilities of the Starship app, but we would implement a very similar system of using the Google Maps API to construct a graph of valid paths on campus and then interfacing this graph with known obstacle data to determine the best routes for reaching different places on campus.

Another important point to note with regards to differences between our route mapping and those of the applications listed above is that besides just going from building to building, we would also like to provide maps inside of buildings, so that students can easily find their classrooms, even in locations with strange classroom numbering schemes, like Green Hall on the UT Dallas campus. In order to achieve this, we would integrate our application with currently existing building maps, like UT Dallas’ locator [7].

*Snapchat*

Finally, while considering the “events” aspect of our application, we looked at the popular social media application Snapchat, and more specifically the “Snapmap” feature. “Snapmap” allows users to post pictures and text relating to an event at a specific location. Then, using the Mapbox API, it displays an icon for each post appearing on a world map, which can be clicked on and viewed by other users [8]. While we don’t need to use any code or API from Snapmap, since we can implement our own version using the Google Maps API, it serves as a good example for what to base our events map on.

1. Conclusion

Overall, this app provides a pathfinding service for the UTD campus that adapts to both user preferences and the changing UTD campus. This app is intended for mobile use, and has the user inputting classes, which gets fed to a path finding service that will reference certain databases to provide the user a path to their destination from their current location.

To develop the application, we selected the V-model, which is a variation of the waterfall model, allowing us to fully document our code and validate and review our work at all levels. The primary reason we selected the V-model is due to our belief that this should be a relatively small project.

We opted for the Model-View-Controller Pattern as the design pattern for this application, as we believed it best served our purposes in developing this app for user interaction and back-end support.

Overall, we had no major design deviations or changes to our app. There were a few minor changes we made in regards to functionality, removing minor capabilities of our app to make development and conceptualization easier. However, these changes in no way changed the overall development process or design decisions of the app, and the functionalities could simply be added to the app at a later time.

Ultimately, the development process for this application should be simple, utilizing the V-model and MVC Pattern. With a 7-person team, we believe it should be relatively quick to implement, and should not require many major design changes.

1. References

[1] *Google*. [Online]. Available:

https://developers.google.com/search/apis/indexing-api/v3/quota-pricing. [Accessed 08-Nov-2020].

[2] “Calendar API | Google Developers,” *Google*. [Online]. Available:

https://developers.google.com/calendar. [Accessed 08-Nov-2020].

[3] “Simple Plans For Every Project!,” Pricing - WeatherAPI.com. [Online]. Available: https://www.weatherapi.com/pricing.aspx. [Accessed: 09-Nov-2020].

[4] “Exam 70-486: Developing ASP.NET MVC Web Applications,” *Microsoft*. [Online].

Available: <https://docs.microsoft.com/en-us/learn/certifications/exams/70-486>. [Accessed Nov. 1, 2020].

[5] T. Stenovec “Google has gotten incredibly good at predicting traffic — here's how,”

*Business Insider*. [Online]. Available:

<https://www.businessinsider.com/how-google-maps-knows-about-traffic-2015-11>.

[Accessed Nov. 8, 2020].

[6] J. Lääne “How Starship Delivery Robots know where they are going,” *Medium*. [Online].

Available:

[https://medium.com/starshiptechnologies/how-starship-delivery-robots-know-where-they](https://medium.com/starshiptechnologies/how-starship-delivery-robots-know-where-they-are-going-c97d385a1015)

[are-going-c97d385a1015](https://medium.com/starshiptechnologies/how-starship-delivery-robots-know-where-they-are-going-c97d385a1015). [Accessed Nov. 8, 2020].

[7] University of Texas at Dallas, “Map Locator”, *UTDallas.edu*. [Online]. Available:

<https://map.utdallas.edu/>. [Accessed Nov. 8, 2020].

[8] Snapchat Support “About Snap Map,” *support.snapchat.com*. [Online]. Available:

<https://support.snapchat.com/en-US/a/snap-map-about>. [Accessed Nov. 8, 2020].

**9.** Link to presentation slides:

<https://docs.google.com/presentation/d/1NtFmH4PTJXhcgMBW-NIpTtjSWrU3w9EPxzBzc_48N9s/edit#slide=id.ga8b462f0a6_0_20>

**10.** We do not currently have fully implemented code for this project.

**11.** See our GitHub repository for our Deliverable 2 commitments.