Group 1: Divvy Van Navigator

Andy Hansana, Kevin Elliott, Vincent Weaver, Ryan Crowley

Testing:

When it comes to testing, the team's focus was on the core functionality of the app, the stop queue system and navigation. The tests we ran were like that of the Nearest stop test, which made sure that when a stop is provided on startup that the following stops are correctly calculated and displayed. This test was run multiple times with different inputs and lists to check to ensure that it was correct in any starting position and not just a set few. The team also tested each component of the GUI to ensure that the users do not experience any UI errors while using the app. This is particularly important because the app will be in a car and users won't have time to exit out of error messages. Those tests started with basic button presses with expected and observed results noted, and then got into more rigorous testing such as requesting a new stop after skipping others before it and changing start locations. After compiling all the test results and organizing them based on Pass or fail, the team started code review. Since all of the GUI tests passed the team's focus for review was on the API parsing and mapview. The criteria for code review was split into 4 sections, Is the code commented? Does the code have warnings? Does the code have any unused/useless code fragments? Does the code operate in a clearly inefficient manner? After inspecting the code used to parse API data, the team fixed any errors and documented changes. The largest changes made after code review was a rewriting of a loop that changed API data to usable data, and a faster way of creating URLs to push to the map view. Overall code review was a productive time for the team to virtually collaborate and go over the work we have done the past sprint.

Many tests were done in order to provide functionality for this application. Aside from the nearest stop test, there was also testing for the map functionality. The map was one of the core items of this application as it is the foreground for most of this application such as providing directions with how the driver of the van would be able to navigate around the city of Chicago. This map was tested many different ways, first originally what had happened was that the team was trying to import the map into the GUI from an FXML file, however with more testing done on that FXML file, the team realized that it would be an issue combining the JavaFX GUI and the FXML file as those two files were hard to combine, so after debugging those changes, we switched to a webview of the google maps API to have the application. In order to test the functionality of the google maps API, we ran through different directions of multiple Divvy stops throughout the city so that maximum efficiency would be achieved through being able to use this map. There were some run time errors where the map would be very slow and also on some certain compilers there were issues where if the user would zoom in on the map, the terminal would throw errors and then the map would end up crashing, so in order to debug that the group decided to compare the bugs to each other, with the conclusion that some certain IDE's that were used such as IntelliJ instead of eclipse threw these errors, thus having to change over to Eclipse. Also some packages weren't imported to the code which caused trouble to this map issue. With importing those packages, the errors were then gone and then the map was fully functional with features from google as well such as the temperature outside and all of the traffic around the area of the marker where the user would be located. Finally, to final test the map functionality, the team decided to try different locations other than divvy locations, in which the team tested directions from UIC to a restaurant, etc with it fully being able to provide directions and accurate traffic of the local time with alternate routes as well.

The first release of the project consisted mostly of creating multiple design prototypes for our GUI and deciding on the one we feel is best for the drivers. We came up with three different GUI design patterns and of the three, went with the one we felt looked the simplest for drivers and the most visually appealing. The criteria for picking was a design that had as little words on screen as possible and more space for the map and other necessary visuals.

The second release of the project was focused on establishing a system for pulling data from the Google Maps API and displaying it on our GUI. Other important parts of this process required us to link our buttons from our FXML to work properly with the GUI and data pulling process. This meant allowing the buttons to be able to add and delete stops, add/delete bikes, complete tasks, pin DIVVY bike stops, etc. To do this, we also needed to implement a system for pulling data from the DIVVY API and reading in the data through a listview in order for the buttons to be able to make necessary changes.

The third and final release involved entirely debugging and testing. Things we looked for during our debugging/testing phase were the end states of our buttons and making sure they did not extend passed their respective endstates.

Throughout each release, the development of an algorithm that solved the Traveling Knapsack problem within the scope of this project was a top priority. The first implementation of this algorithm was to solely populate a listview with each of the stops pulled from the API. This was necessary to create objects to use and manipulate using the algorithm. The second release goal was to implement a Divvy Station class, which would store all of the necessary data to use within the algorithm. The most important piece of this data would be the distance class variable, which would store the distance from the user's primary location upon loading the app, which was obtained from a separate algorithm. This would be incorporated into the final release of the project, which implemented a Greedy sorting algorithm, utilizing the distance variable in the Divvy Station class, and sorted each station in ascending order based on this variable.