Meera Kanani - Project Portfolio

PROJECT: Pigeons

Overview

Pigeons is a desktop level application created for delivery companies to better assign and organize deliveries for the delivery people.

Summary of contributions

- **Major enhancement**: added **the ability to create an optimal delivery route when filtering the data by delivery date**
    - What it does: allows the user to filter all the data by date of delivery and then creates an optimized route from the headquarters to all the delivery points for that day.
    - Justification: This feature improves the product significantly because the user is meant to be someone who delivers packages and this make it easy for the user to determine where he has to go that day and what the best route is from his starting location. This optimized route will be outputed as a map on the UI by another team member.
    - Highlights: This enhancement affects existing commands and commands to be added in future. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required changes to existing commands but more importantly required creating new classes and methods that could be referenced by other team members and would be usable for the UI.
    - Credits: The Google API was used to get the distance between addresses, the java code was written around using this API.
- **Minor enhancement**: added a filter command that allows the user to filter the data by date (delivery date).
- **Minor enhancement**: added a getTime function that gets the time of the entire journey for the delivery man and can be displayed on the UI.
- **Code contributed**: [Functional code] [Test code]
- **Other contributions**:
    - Project management:
        - Kept team on track on TODO’s before each tutorial
    - Enhancements to existing features:
        - Wrote additional tests for existing features to increase coverage
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: #14
        - Modified existing guides to include the premises of our project
        - Removed items of the user guide, developer guide etc that were relevant for our project]

        - Added documentation for all the features I added in the dev guide and user guide.
        - Modifed the README doc
    - Community:
        - PRs reviewed (with non-trivial review comments)

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Filtering person by date: `filter`

Filers the list of people by date (only one date) Format: `filter DATE`

- Recommended date format is YYYY-MM-DD

When you filter by date the back end of the program will create an optimized route for all the people on that date and display the route on the map.

- The search can only look for one date
- Persons with the delivery date matching the argument will be returned
- This filter can be used to obtain a delivery route for a day

User Testing

When testing and using our system for the first time, here are some commands you can try to get the full experience. Feel free to try other commands outlined above when you get the feel of our program as well.

1. add n/John Doa a/321 Clementi Ave p/67340028 e/pigeonscs2103@gmail.com d/2018-03-24

- John Doa should be added to the list of addressees.

2. filter 2018-03-24

- All addressees with 2018-03-24 in their delivery date should be listed.
- Additionally, a map routing all the shown addressees should be generated.

3. find John

- All addressees with John in their name should be listed.

4. delete 1

- John, which we added at the start, should be removed.
- 1-index numbering system (i.e. the first addressee has index 1)

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Filter Command

To add to the many commands necessary for users, filtering by date will be essential when trying to optimize the route per day. The filter command was added similar to how the find command has been implemented. When the user enters filter by a date in the UI the model with the list of data is updated using the date predicate. This updated model is then executed via the CommandResult class and output on the UI. It is stores as a readonly list in the model.

Route Optimization feature

Since this program is meant for deliveries the route for the daily delivery must be optimized for the user. The RouteOptimization class will get the distance (getDistance function in GetDistance class) between all the addresses for the day.

The addresses will then be sorted based on optimal sortest route. Example: 3 deliveries for a day at Point A, B, C

Point A -→ Point B = 14km (12min) Point A -→ Point C = 13km (13min) Point C -→ Point B = 12km (10min)

Optimal Route = Point A -→ Point C -→ Point B (based on least about of km travelled)

In the RouteOptimization class the sorting function will be responsible to determine the optimal route once the distances are obtained

To getDistance - google API is required to get the geocodes for the locations. Once the geocode for each address is obtained the distance between the two points is calculated and a matrix is returned with a distance property. The distance property gives the distance in kilometers from the origin to destination. The function operates recursively determining the shortest distances from the starting point onwards and then from the next point onwards. This recursive nature ensures that the driver is following a seamless route.

For example:

- Start at A (first address in the list of addresses)
- Find all the distances from point A to the rest of the destinations
- Choose the shortest distance from the bunch (ex. A-B is the shortest)
- Add B to the list of addresses for the optimized route
- Now repeat using B as the starting point and calculating the distance to all remaining destinations (omitting A)

The optimized route will the be fed into a mapping function to display the route and get the directions for the driver.

Sorting addresses

Once the distances have been found for all combinations of addresses on that filtered day they get stored in a Map with key: "String address1_address2" and value: "Double distance (in km)". This hashmap gets sorted in order of distances and any irrelevant or duplicate calculations are removed. For example if the distance from A to B is stored and B to A is stored, the second stored duplicate will be removed This sorted Hashmap now contains the key with the orign and destination address as well as the distance between the two. The sortByComparator function is constantly used to make sure the addresses are sorted by distances as the distances are being calculated. It is important that the distances are not only sorted by distance but also in an optimal route. If the shortest distance is from A to B the next part of the route has to be starting from B. Therefore, the distances are measured recursively from the new starting point, sorted by distance and then added to the final route.