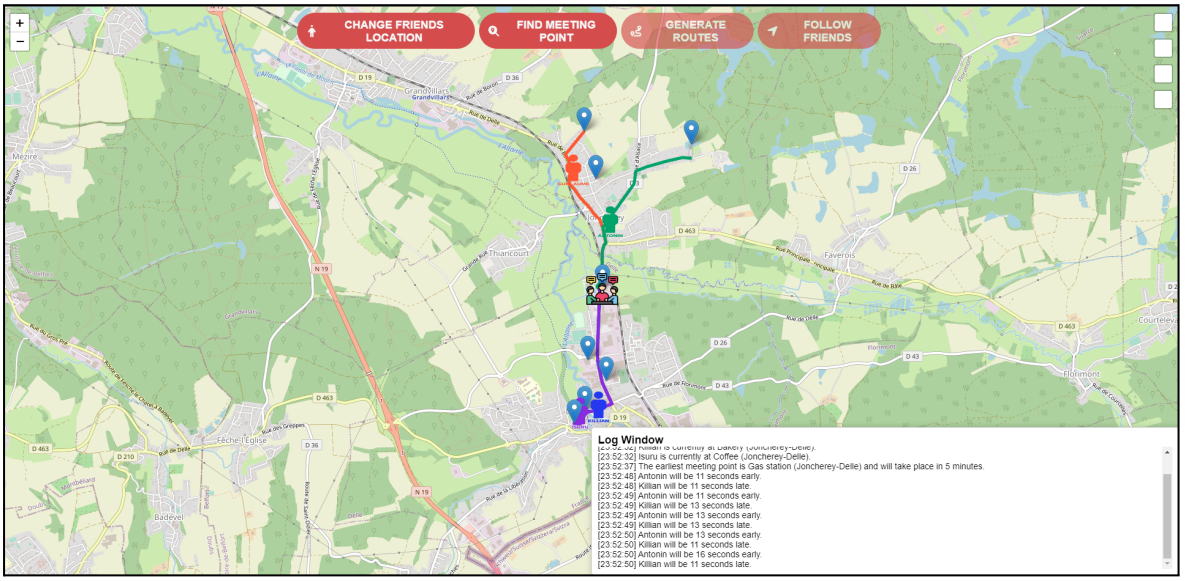
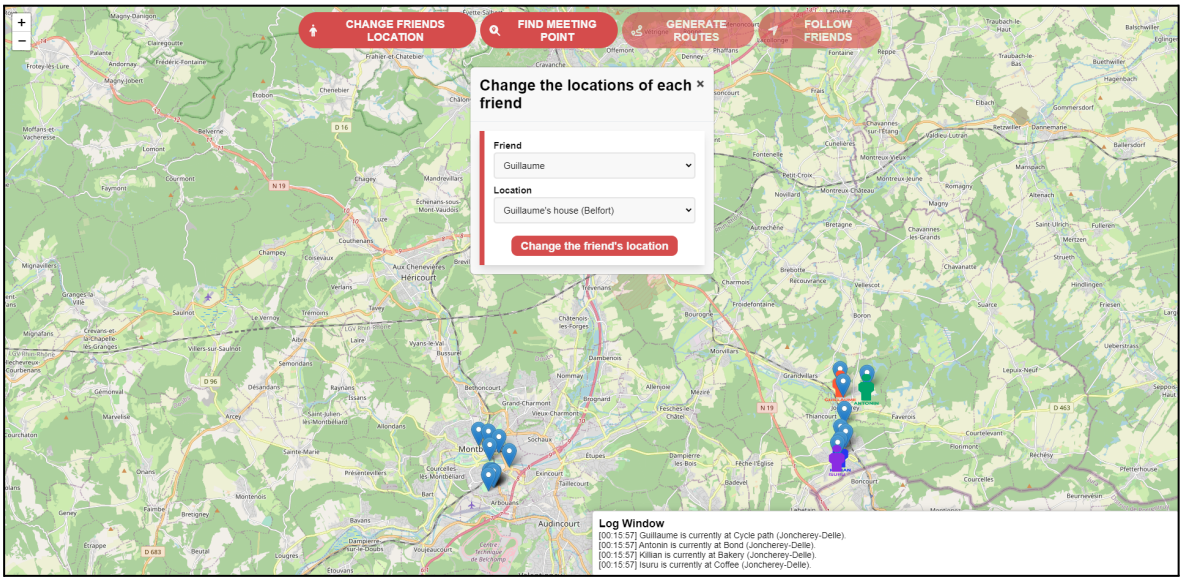
MINI-PROJECT MSC REPORT





Antonin WINTERSTEIN
Killian MAXEL

Teacher: Mr. Philippe CANALDA

CONTENT

PROJECT PRESENTATION

Introduction

The goal of the project is either to create a mobile application or a web-based application to map students' main points of interest (home, parking, university...), using an Origin-Destination (OD) matrix for these spaces, and facilitating group meetups among registered friends.

This application is designed to efficiently determine optimal meeting places and time windows for groups of friends. Additionally, it aims to generate individual routes for each friend and offer real-time tracking to signal any delays or advancements.

Specification of the subject

As specified in the subject, we must create an application that should have the capability to map various student locations. These locations can include homes, classrooms, places of daily activity, meeting points, car parks, and more.

We will also need to use an Origin-Destination matrix (with a size of 30 locations) for these locations. This matrix should provide details such as distances, time estimates, and route options between different places. This feature is vital for determining optimal meetup locations.

Three functions are to do using the map and the Origin-Destination matrix:

1. The main function is to determine a place X and a time-window where a group of friends can meet at the earliest taking into account each friend's location and schedule.
2. The second major function is to generate individual routes for each friend to reach the meetup location.
3. The third and last function is to follow friends as they make their way to the meetup location to provide alerts for delays or advancements.

Environment to consider

We should consider creating a map where we need to show some places frequented by students (in our case at Belfort-Montbéliard).

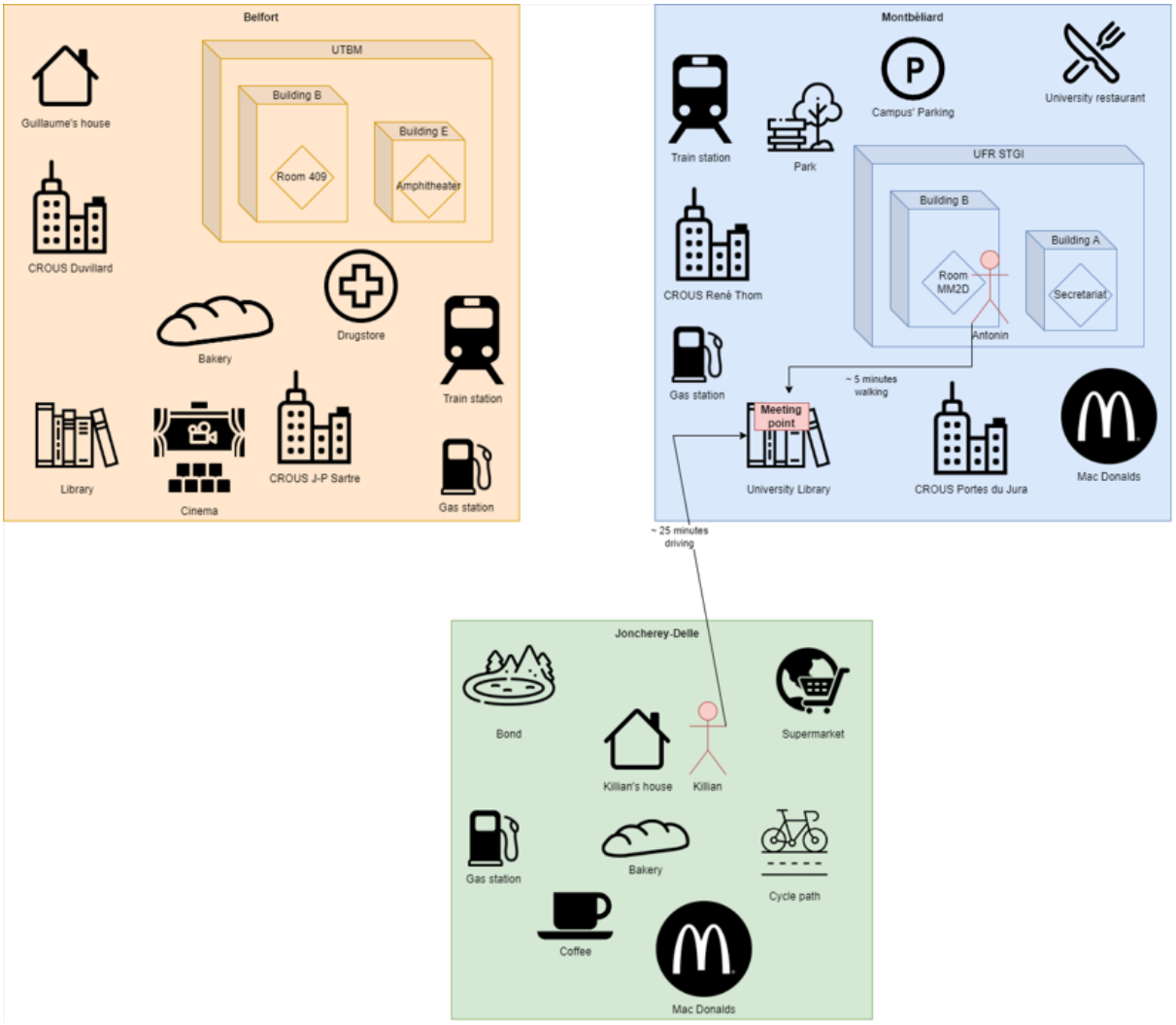
We will also need to register friends in order to place them on the points of interest depending on their actual position and also to allow the use of the Origin-Destination matrix where we need to determine the meeting point.

Next we need to include roads on the map in order to allow the generation of customized routes for each friend and also to determine, by tracking the friend going to the meeting point, if he is late or early.

Graphical resolution

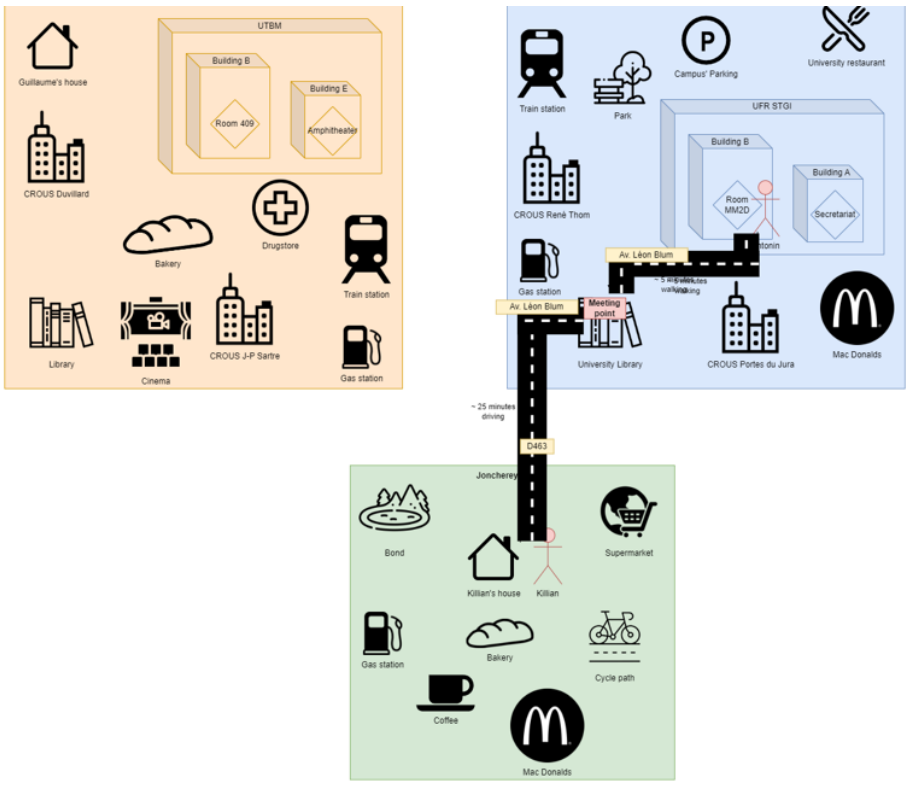
To try resolving the initial problem. We started by creating graphics for each part of the subject.

We first represented our 30 locations in three zones with two friends located at two of these locations (in this graphic, Antonin is in the MM2D room and Killian in Killian's house). From the positions of these friends, a meeting point must be determined in order to allow them to meet at the earliest. In this case it seems like the university library is the optimal candidate.



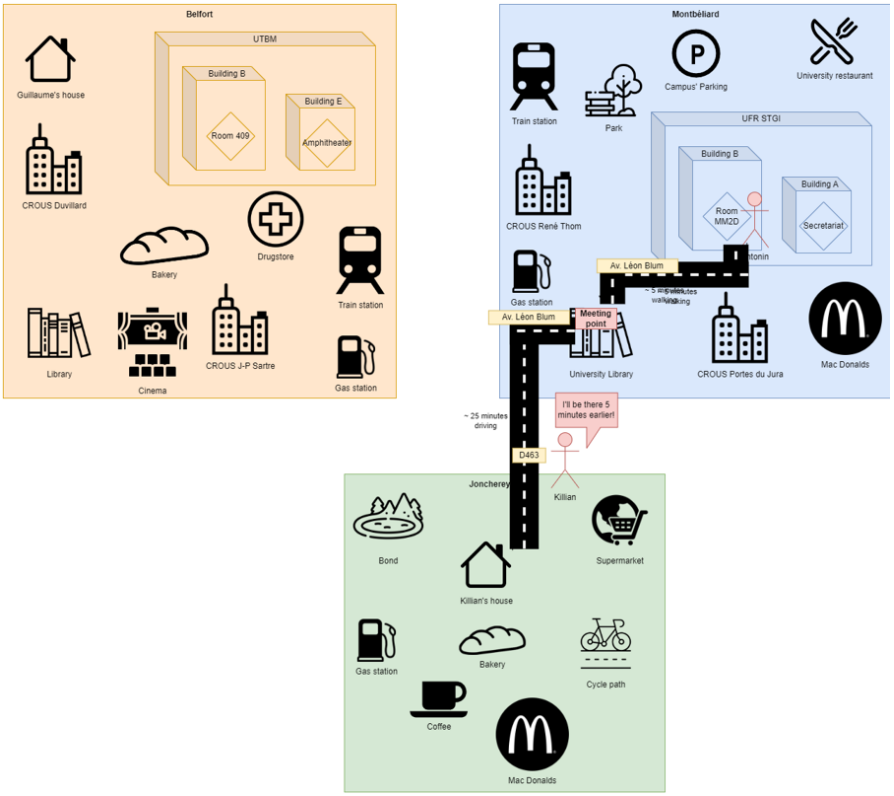
First step/function of the project representation with the election of the meeting point

Next, using the meeting point location and the locations of each friend, we must generate individual routes for each friend to reach the meetup location with which road to take, the distance, time of travel and so on.



Second step/function of the project representation with the route of each friend towards the meeting point

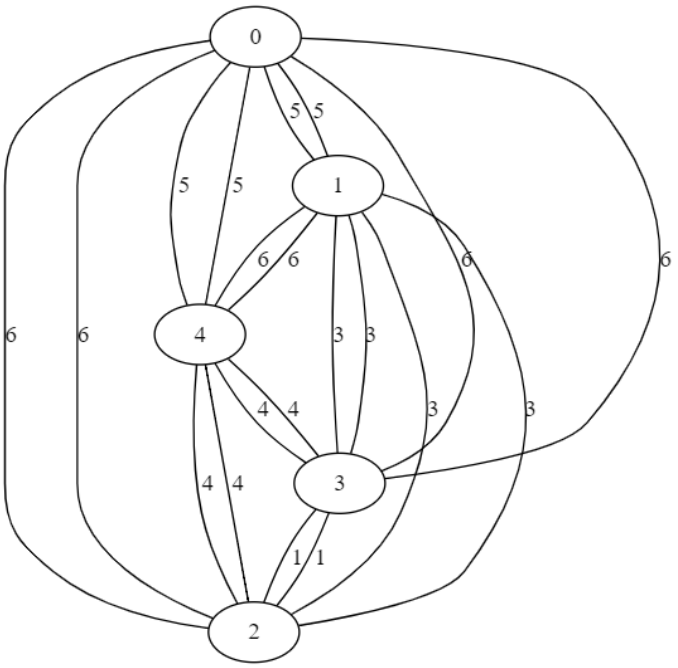
Finally, we should provide for the last function a way to alert if the friend will have delay or advance regarding its way to the meeting point.



Third step/function of the project representation with Killian alerting that he will be there earlier

Graph representation of the OD Matrix

In order to validate the values assigned to the OD matrix and ensure that the properties of reflexivity, symmetry, and transitivity are correctly followed, we opted to create a second graphical representation using the dot format and the graphviz library.

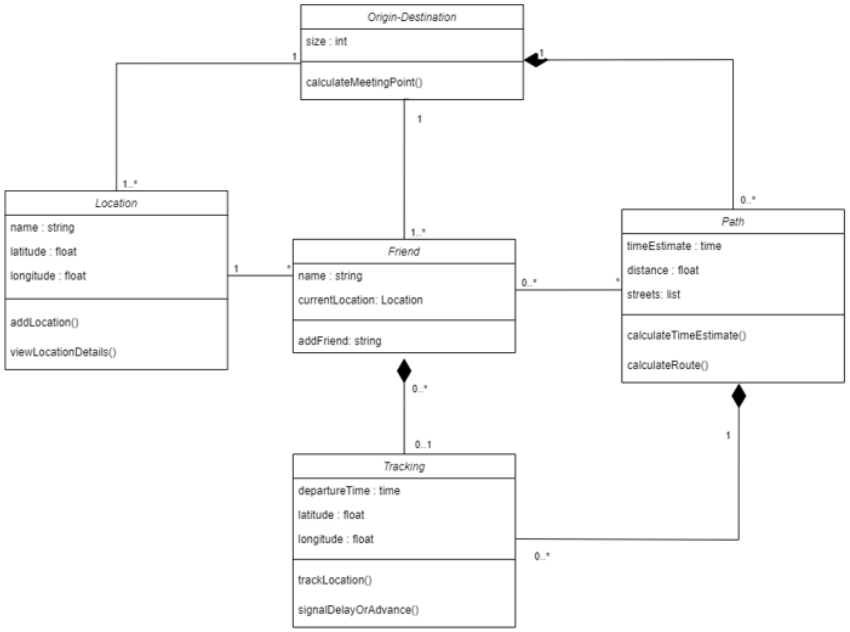


Simplified Graph representation of our matrix
(only 5 locations for visibility)
(complete version here)

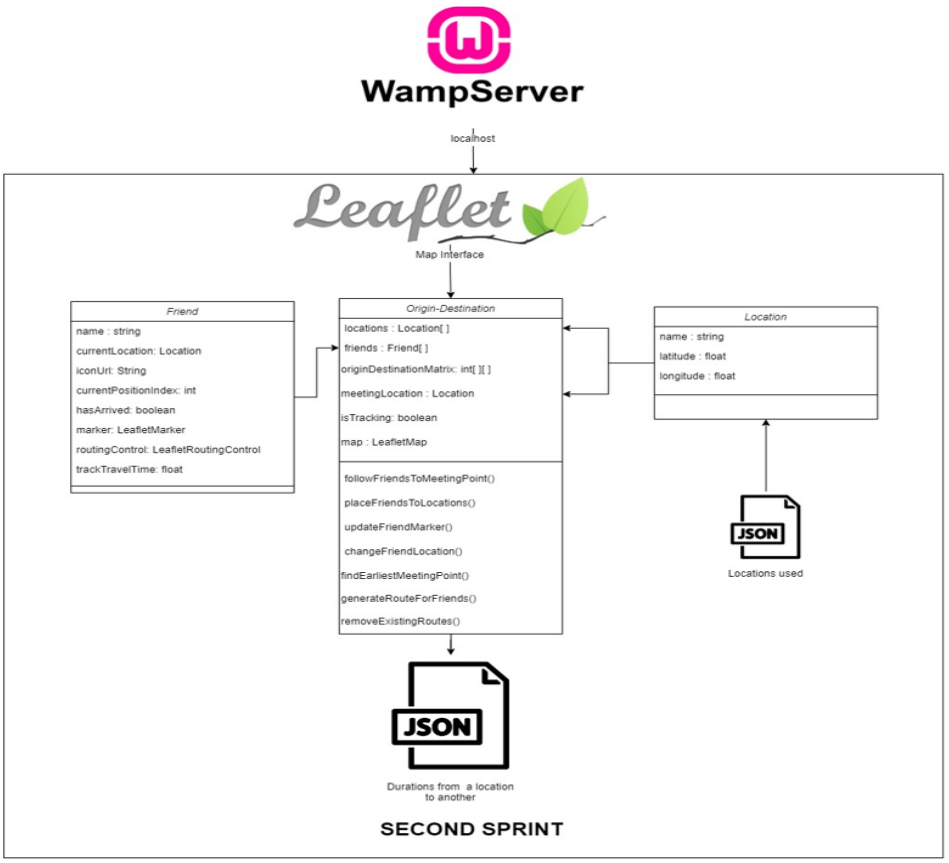
But because we have a 30x30 matrix, it is difficult to notice anything on it.

Class diagrams

After having represented the problem, we decided to work on a class diagram. For that, we first had a global representation but in practice, we changed it while developing to ease our implementation.



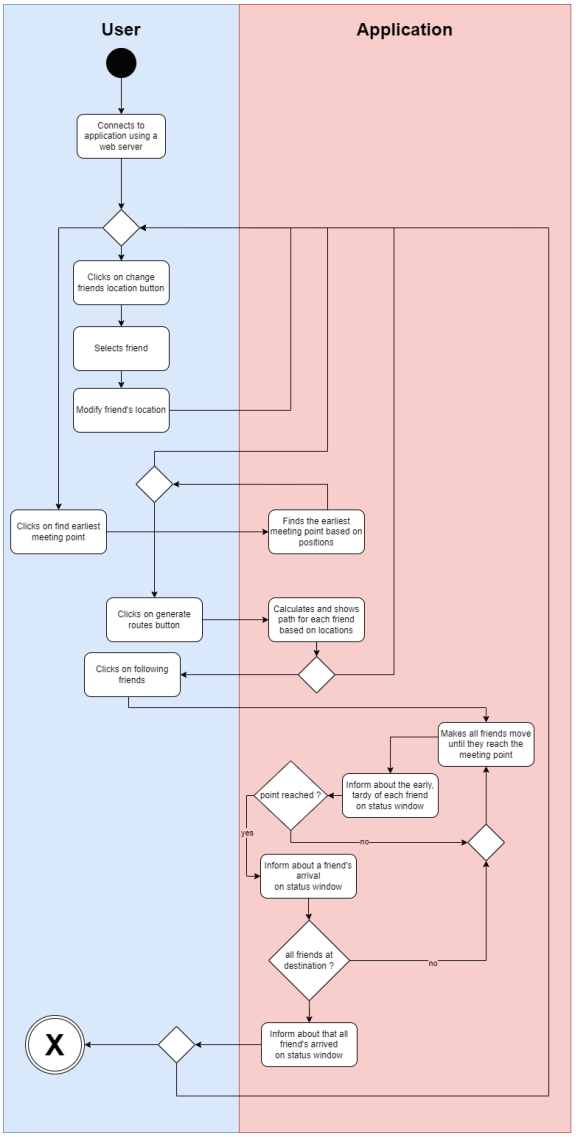
First class diagram

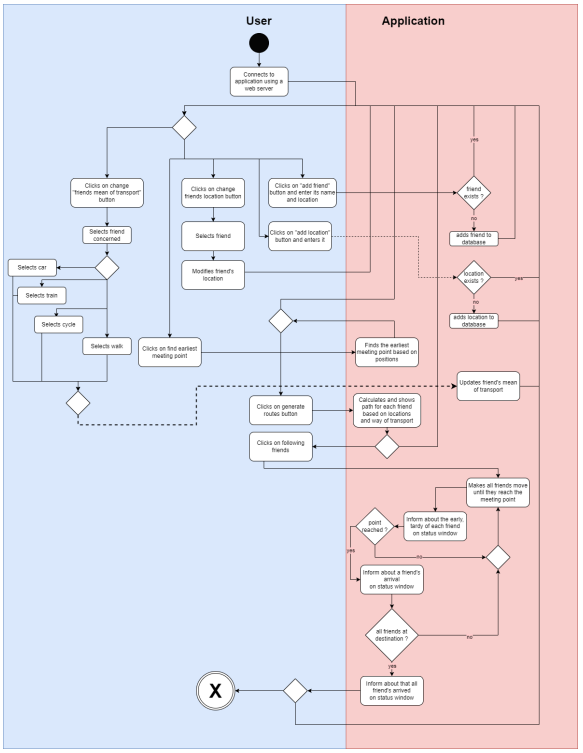


Final class diagram

Activity diagrams

After making a representation of our classes we created activity diagrams in order to have a clear view of how our application will work.





Activity diagram showing basics features	Updated activity diagram for final application's features
Link to full sized image	Link to full sized image

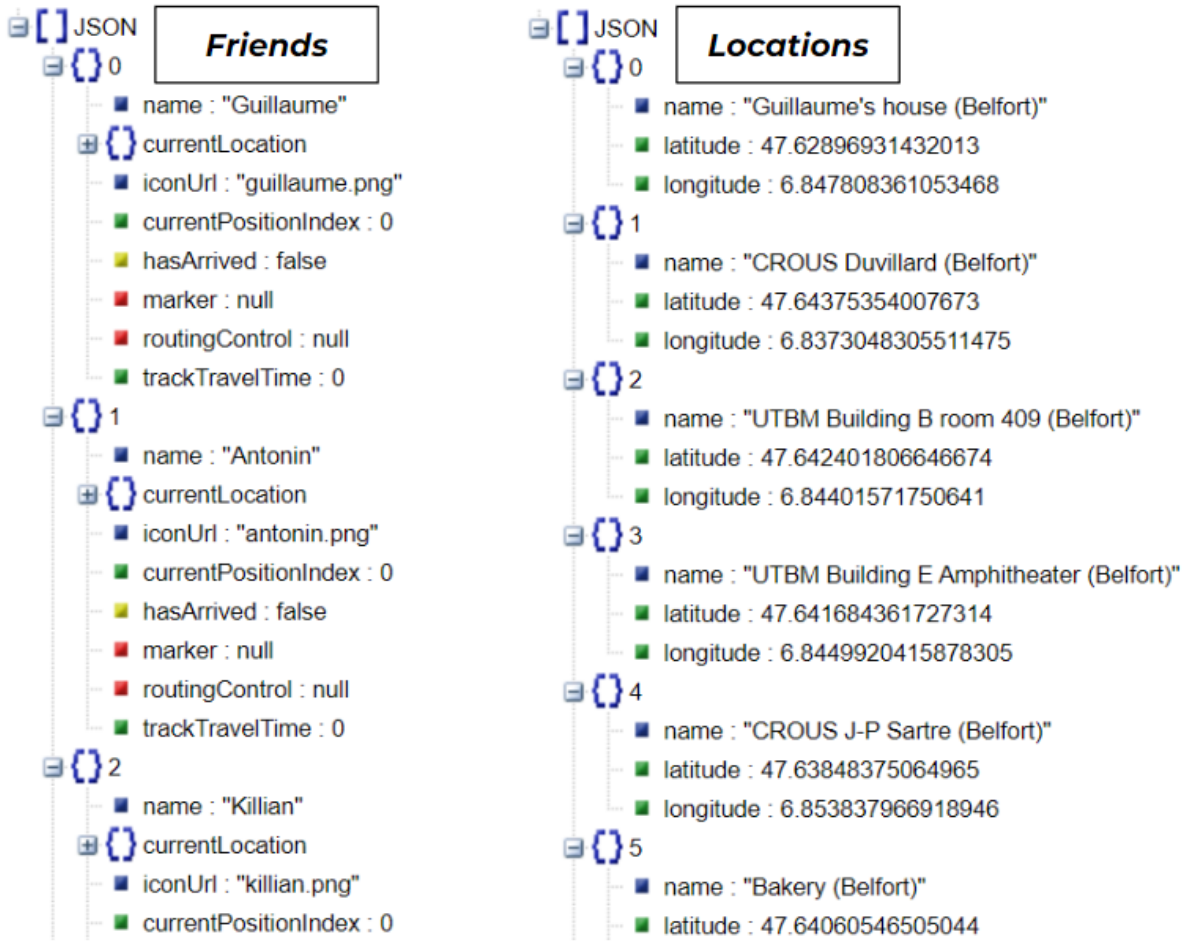
Dataset

In order to be able to develop the application with its different constraints, we created a static dataset manually.

This dataset must have information about locations, friends and time of travel between locations (i.e the Origin-Destination matrix).

To tackle this problem, we created three JSON files which are:

- ***locationsData***: contains information of 30 locations, each holding three attributes (the name of the location, its latitude and its longitude).
- ***originDestinationData***: contains the time of travel in minutes between each location in a 2D array.
- ***friendsData***: contains information of 4 friends, each holding several attributes (the name of the friend, the current location, the custom iconUrl, the current position index of the route, a boolean to know if he arrived at the destination, the marker needed to show him, the information of its route and the travel time to know if he's in advance or has delay).



Preview of our friendsData and locationsData

```
[
    [
        0, 5, 6, 6, 5, 5, 5, 3, 7, 6, 8, 3, 18, 21, 17, 17, 18, 18, 19, 19, 19, 18,
        16, 23, 26, 24, 23, 22, 24, 24
    ],
    [
        5, 0, 3, 3, 6, 2, 6, 7, 9, 11, 6, 21, 24, 20, 20, 21, 21, 22, 22, 22, 21,
        19, 26, 29, 27, 28, 27, 26, 26, 27
    ],
    [
        6, 3, 0, 1, 4, 2, 5, 6, 8, 10, 6, 21, 24, 20, 20, 21, 21, 22, 22, 22, 21,
        19, 26, 29, 27, 28, 27, 26, 26, 27
    ],
    [
        6, 3, 1, 0, 4, 2, 5, 6, 8, 10, 6, 21, 24, 20, 20, 21, 21, 22, 22, 22, 21,
        19, 26, 29, 27, 28, 27, 26, 26, 27
    ],
    [
        5, 6, 4, 4, 0, 4, 2, 1, 5, 6, 2, 20, 22, 19, 18, 19, 19, 20, 20, 20, 20, 17,
        24, 27, 26, 26, 25, 24, 24, 25
    ],
    [
```

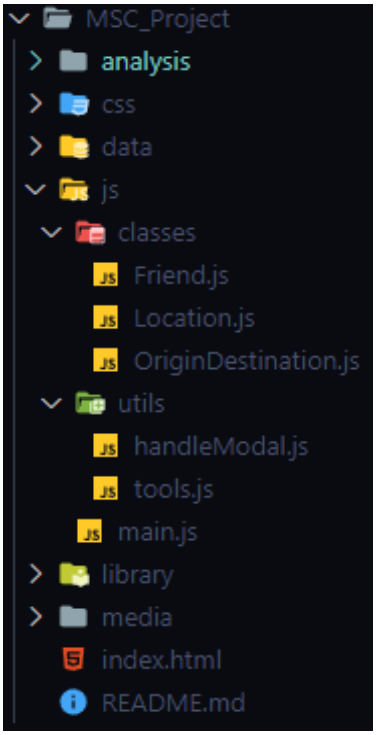

Preview of our originDestinationData

Technologies considered

Since we are storing our dataset in JSON files, we decided to create a web-based application using *HTML-CSS-JS*. Because we want to define specific locations with the possibility to determine a meeting point and a route for each location, we thought about displaying a map using the Leaflet API. It also allows us to easily add markers for these locations but also for the friends located on the map. For the routing part, we also used the Leaflet Routing Machine extension.
Note that you should use a server in order to be able to run the application.

IMPLEMENTATION

Project structure



Screenshot of the main structure of our project

- `**analysis/**` folder: contains documents about the project.
- `**css/**` folder: contains the CSS file for the design of the application.
- `**data/**` folder: contains the JSON files with the necessary data.
- `**js/classes**` folder: contains the classes used in the project.
- `**js/utils**` folder: contains additional files with helpful functions.
- `**js/main.js**` file: the main JS file of the project.
- `**library/**` folder: contains some libraries used for the project.
- `**media/icons**` folder: contains icons used for markers on the map.
- `**index.html**` file: the HTML file to open to launch the application.
- `**README.md**`: the description of the project.

How to launch the program

In order to be able to launch the program, you must launch the `*index.html*` file on a server (localhost with wamp or xampp for example). You can also use the extension "Live Server" of VSCode.

⚠️As an important note, please don't launch the program on Firefox otherwise it won't work. It's because assertions of our JSON files are not supported by the browser. Use Google Chrome instead for example. ⚠️

During the launch of the program

At the application's startup, the validation of the JSON files is done, assessing their format, field completeness, and properties such as transitivity, symmetry, and reflexivity and the size (n*n) of the Origin-Destination matrix. This process is important to ensure the application's functionality, data integrity, reliability and accuracy, this allows for the customization of the dataset provided to it.

- **Symmetry** ensures that if there is a relationship between locations i and j (i,j), then the same relationship should exist in the reverse order (j, i). It guarantees that the relationships are bidirectional and consistent.
- **Reflexivity** ensures that when a location relates to itself (when i = j), the value for this relationship should be 0. By having reflexivity, we ensure consistency in our calculations.
- **Transitivity** means that if (i, j) and (j, k) exist, then (i, k) should also exist (value different from 0 if i!=k) even if the value at (i, k) is not the sum of the values at (i, j) and (j, k), because in our context we are dealing with non-binary values in our 30x30 OD matrix, so applying strict transitive closure could lead to an unending process. In other words, transitivity ensures that indirect relationships are established correctly. , even if it doesn't have a straightforward sum relationship.

By incorporating these JSON files validation process, we ensure the integrity and consistency of the data relationships within the application to guarantee the reliability of the dataset but also its accuracy of calculations and inferences when customizing it.

Defining the meeting point (first objective)

We imagine having a group of friends, each at a different location, and we want to find the ideal meeting point where everyone can gather in the shortest time possible. To achieve this, we have a few key ingredients:

1. The current locations of all our friends.
2. A list of possible meeting locations.
3. A matrix that provides travel times between all pairs of locations.

We first begin by collecting the current locations of all friends in the group and store them in an array.

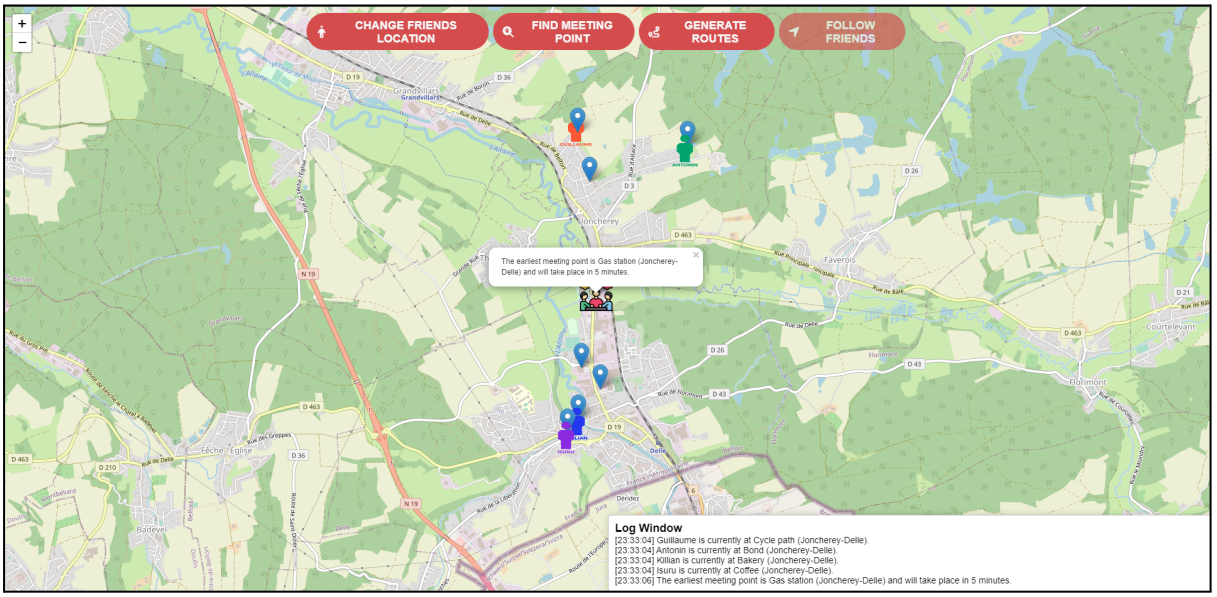
Two important variables are initialized: earliestMeetingLocation and earliestMeetingTime. These will be used to keep track of the meeting point and the time it takes to reach there.

The algorithm then considers various potential meeting locations by iterating through them one by one. For each meeting location under consideration, the algorithm calculates the maximum travel time for all friends to reach that location. This is done by finding the friend whose travel time to the location is the longest among all friends. If the maximum travel time to the current meeting

location is less than the previously recorded earliest meeting time, the algorithm updates the earliest meeting time with this new value. It also updates the earliest meeting location with the current location being considered.

When the earliest meeting point is found, we can then display it on the map and be sure that this is the fastest way to all gather together.

Note that it is possible to change the location of friends on the map in order to calculate another meeting point.

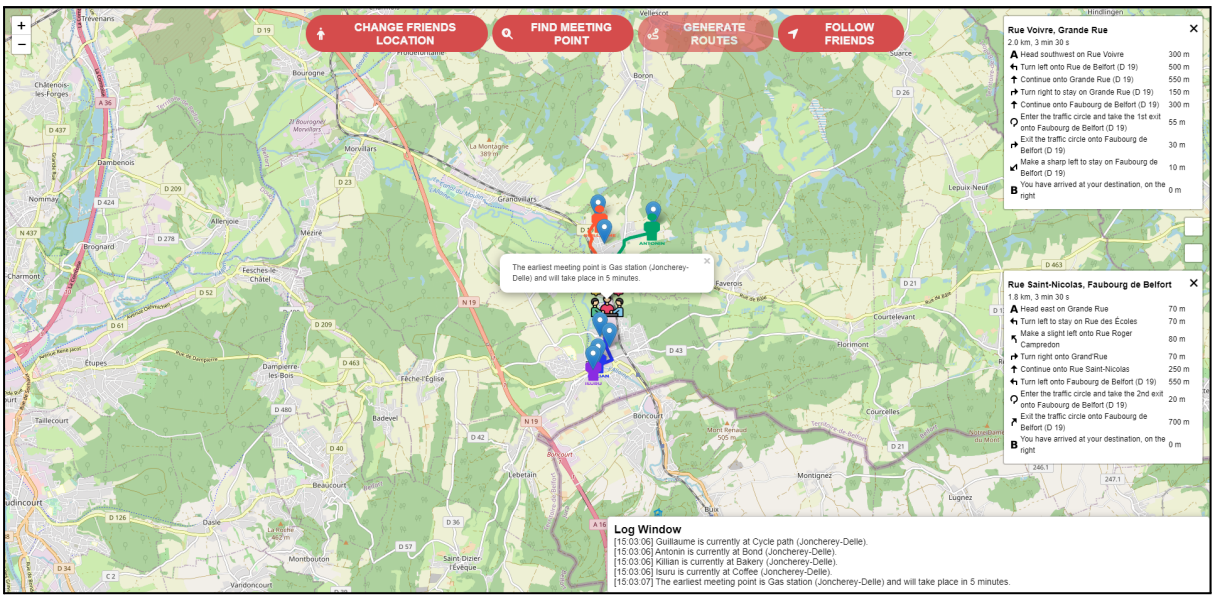


Screenshot of the application when searching for the meeting point

Defining the route for each friend (second objective)

After electing the meeting point, we can then draw the routes for each friend to it. For that, we create and store each route from the actual location of the user to the meeting point. It uses Leaflet's Routing control to create routes on the map. Each friend is assigned a route color, and their routing information is stored for future removal.

When clicking on the top right squares, we can show or hide the four itineraries to have further information.



Screenshot of the application with custom routes for each friend to the meeting point

Defining the follow of the friends (third objective)

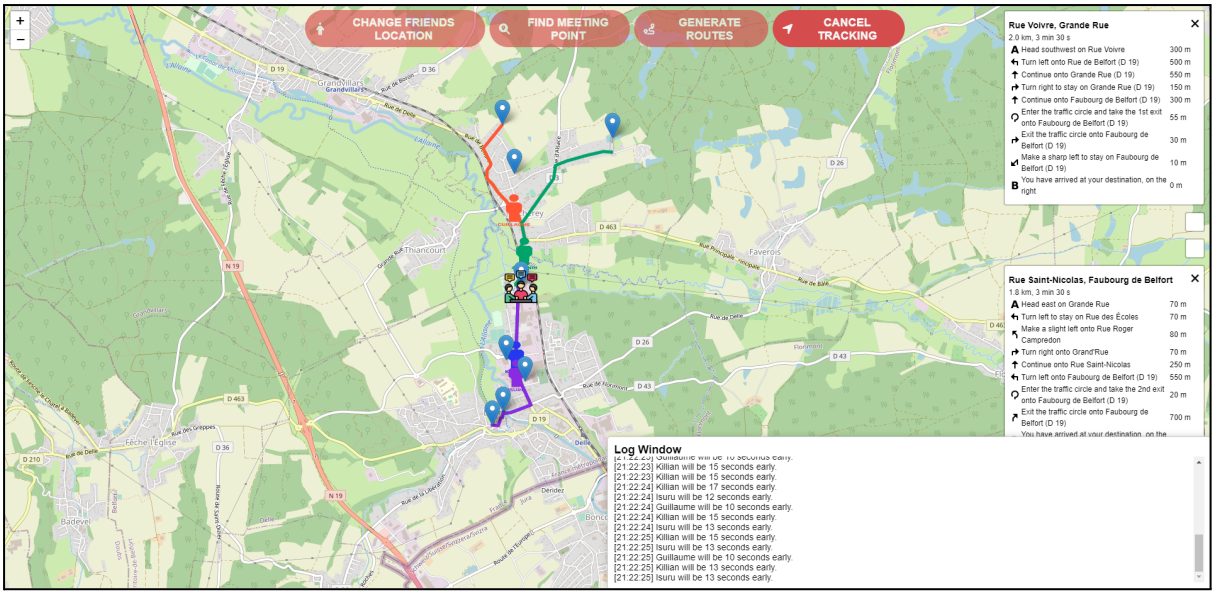
To finish, friends can be followed as they make their way to the meeting point.

Indeed, thanks to Leaflet Routing Machine, it is possible to retrieve the coordinates of the itinerary and also the complete travel time. With these coordinates, we can periodically update friend positions based on their routes. Moreover, with the travel time, it is possible to calculate the travel time per coordinate in seconds by dividing the travel time with the number of coordinates which will be useful to track delays or advances.

In order to have a kind of tracking to generate delays or advances, we randomly choose to either stay on the current coordinate (meaning that the friend loses time), go to the next one (nothing happens, it is the normal behavior) or skip a coordinate (meaning that the friend earns time).

This allows us to create alerts in case the delay or advance is high which will be shown in the log window. Finally, when a friend arrives at the meeting point, they are marked as arrived and wait for the others to come.

Note that you can at any time cancel or resume the tracking of the friends by clicking again on the same button.



Screenshot of the application with following of friends during their route

Interface

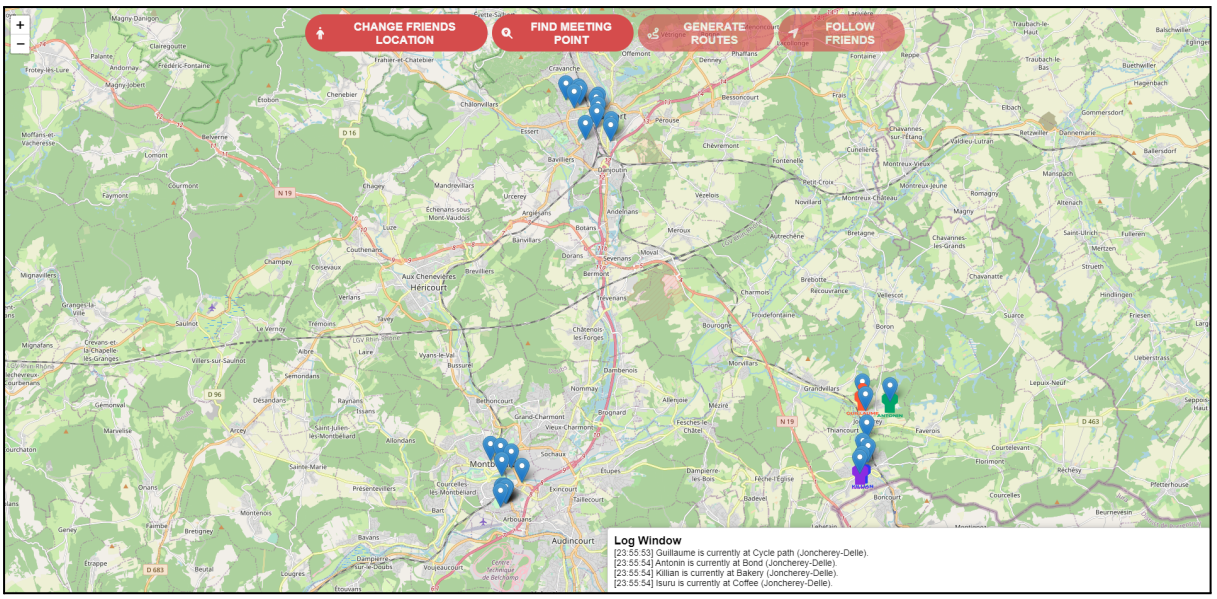
In order to better interact with the application, we created markers for each location. When clicking on them we can know to which real location it refers. Moreover, we created custom images for the four friends to differentiate them.

At the top, four buttons are available:

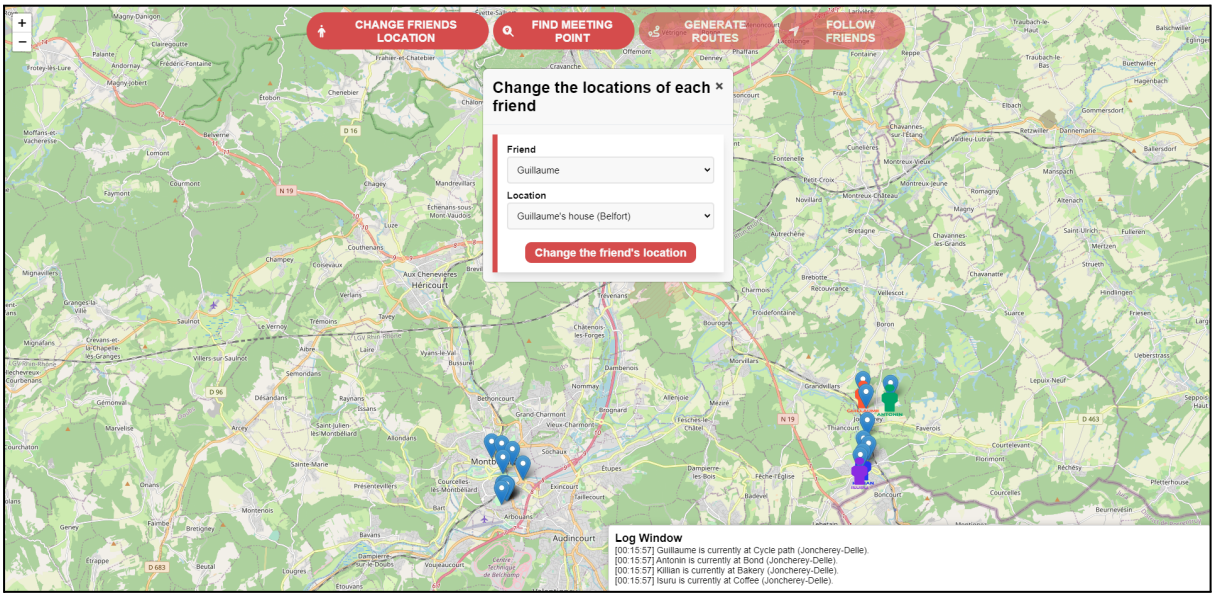
- ***Change friends location***: Opens a modal with a form with two mandatory select, one to select the friend and one to select the location of the friend. After completing it, the selected friend will be moved to the selected location.
- ***Find meeting point***: Launches the function that calculates which location is the meeting point.
- ***Generate routes***: Launches the function that generates routes for each friend to the meeting point.
- ***Follow friends***: Launches the function that follows the friends on their respective routes launching alerts in case of delay or advance.

When generating routes, the itineraries will be shown on the right. You can if you want, close each window by clicking on the cross. You can always open them again by clicking on the corresponding square of the itinerary.

Finally, at the bottom right corner, we have a window for logs about what's going on.



Screenshot of the application when launching it



Screenshot of the application when changing a friend's location

CONCLUSION

In this project, we set out to create an application that efficiently facilitates group meetups among friends by determining optimal meeting places and time windows. The application leverages an Origin-Destination (OD) matrix for various student locations and provides real-time tracking to signal any delays or advancements for friends making their way to the meetup point.

The first objective of the project was to find the ideal meeting point where a group of friends can gather in the shortest time possible. This involved considering various potential meeting locations, calculating the maximum travel time for all friends to reach each location, and selecting the location with the shortest travel time.
After determining the meeting point, the second objective was to generate individual routes for each friend to reach the meetup location. This was achieved by creating and storing routes from each friend's current location to the meeting point.

The final objective involved real-time tracking of friends as they made their way to the meeting point. This tracking included periodic updates of friend positions based on their routes and alerts for delays or advancements.

The application offers a user-friendly interface with interactive markers for locations, custom icons for friends, and intuitive buttons for actions such as changing friend locations, finding the meeting point, generating routes, and following friends. It also includes informative logs for tracking events.

However, there are opportunities for future enhancements. Instead of a static matrix, we could consider integrating a dynamic source for travel time data, such as real-time traffic information. We could implement a feature to add new friends and locations to enrich our dataset. Finally, we based our implementation only on time travel with a car but it could be with several means of transport such as bus, bicycle or walking.

APPENDICES AND REFERENCES

APIs used:

- https://leafletjs.com/
- https://www.liedman.net/leaflet-routing-machine/
- https://graphviz.org

GitHub:
https://github.com/Nexusprime22/positioning-n-lateration

Demo video:
https://youtu.be/M6srmPKkkQU