Klement Tan(A0183726X) - Project Portfolio

PROJECT: TimeBook

Overview

My team was tasked to worked on a basic command line interface desktop AddressBook application for our Software Engineering project. We chose to morph AddressBook to an application, **TimeBook** that makes scheduling team meetings a breeze for students. Students can keep track of their various on going projects and sync the schedule of their project members onto a single platform. They can easily find common free time slot within a project group and we will suggest the closet location for everyone to meet. With TimeBook, scheduling Project meetings will no longer be a hassle.

This is what our project looks like

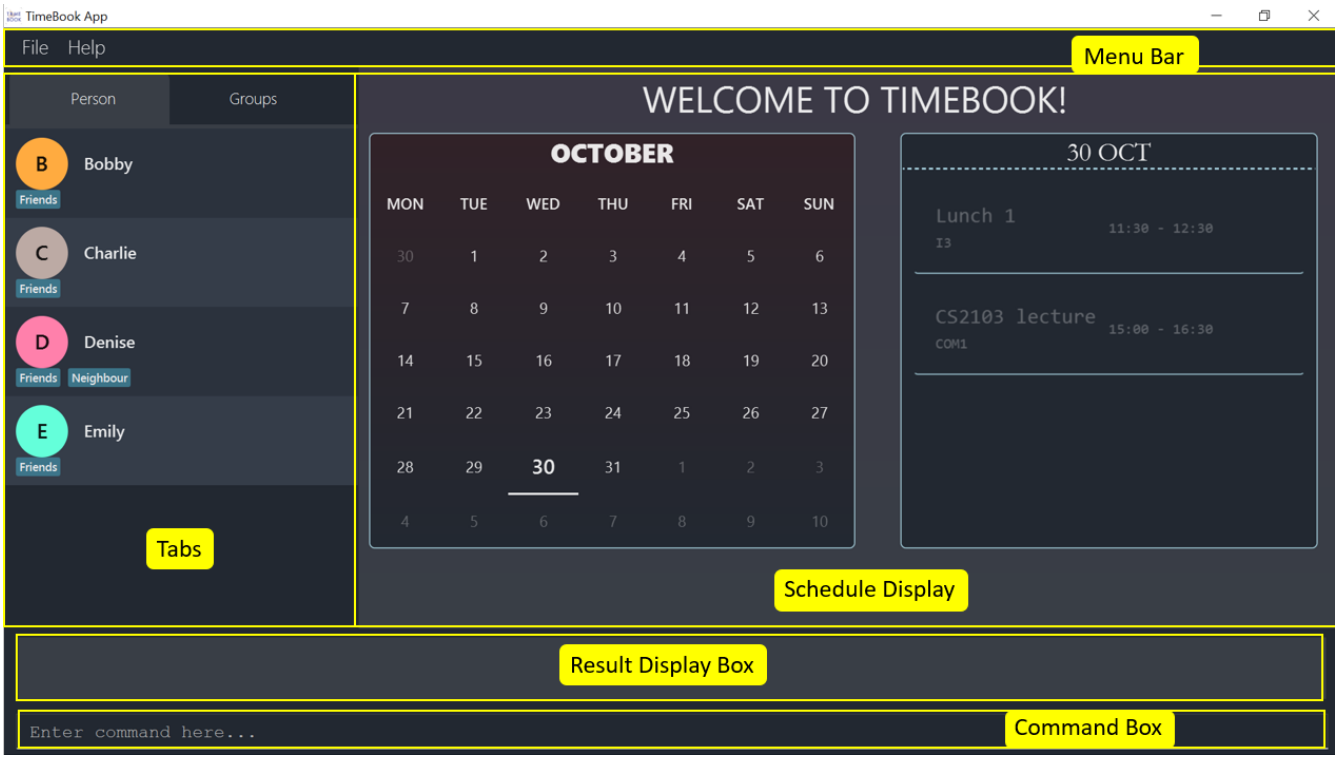

Figure 1. Home window of TimeBook

My role was to be the project manager and create an algorithm to compute the best location to meet for a meeting based on the project members initial location. This resolves the issue of students having to pick from the vast variety of options to choose the venues for project team meetings.The following sections illustrates this feature in detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Summary of contributions

This section shows a summary of my contribution to TimeBook. It includes the contribution to code base, documentation and other helpful contributions.

- **Major Feature added**: I added a feature that helps project teams choose the closest location for everyone to meet. Below is visual representation of the feature:

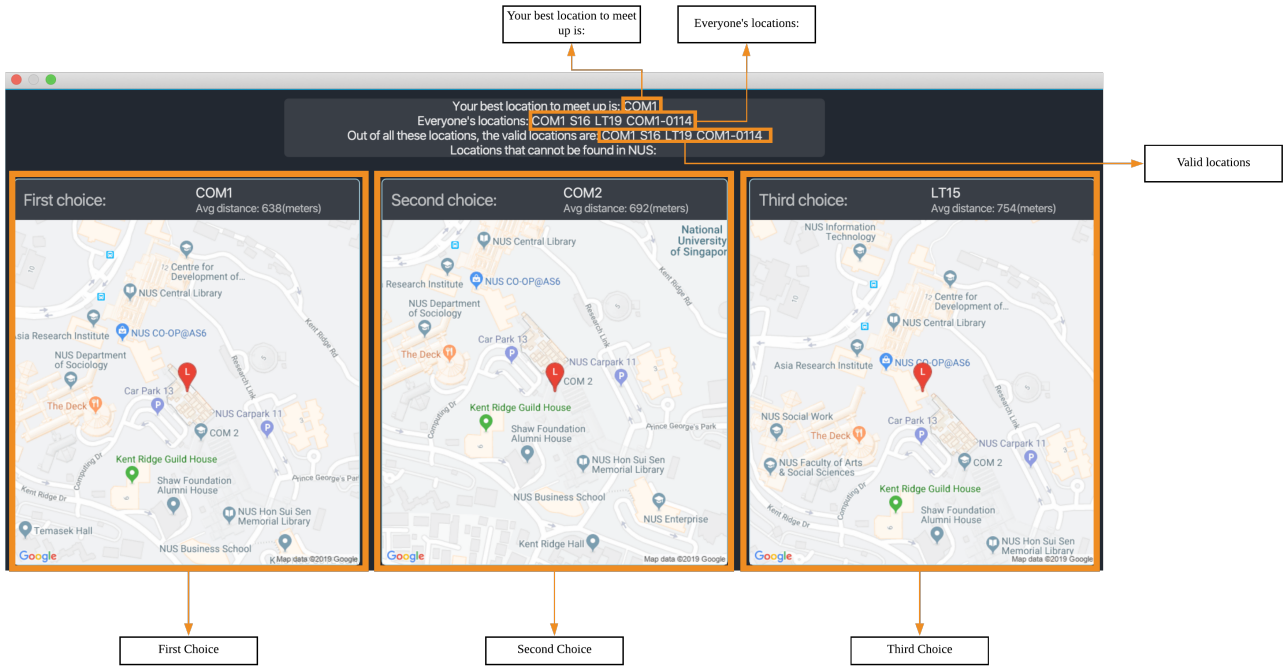

Figure 2. Home window of TimeBook

a. **What it does**: The popup command will display a popup(Figure 2 above) with information on where are the best places for everyone to meet during the school day. We define the best place to meet as the location that requires the least average travelling distance. As seen above, the users will be given the top 3 location with the least average travelling distance. They will know what the average travelling distance for each location and a picture of the location in Google Maps.

b. **Justification**: NUS is a large campus with over five-hundred different venues to conduct the project meeting. Finding the best place to meet for a project can be a hassle as project teams the best location. This feature tackles this issue by using Google Maps to compute an unbiased closest common location that will benefit the team as a whole.

c. **Highlights**: This feature works closely with external data sources such as NUSMods API(Application Programming Interface) to determine the initial location of the project members and Google Maps to compute the average traveling distance. Additionally I had to apply graph theory concepts to develop an algorithm to compute the closest common location. This feature is closely dependent on `Ui`, `Scheduling` and `NUSMods` component of TimeBook. I had to work closely with the other group members to ensure the seamless integration of this feature into TimeBook.

d. **Challenges**:

   i. **Google Maps**:

A. Google Maps APIs calls cost USD$10-USD$20/1000 call. I had to minimise API calls by having a deep understanding of Google Maps APIs and carefully designing this feature to optimise for the our resources constraints.

B. There were bugs in Google Maps API and I had to meticulously analyze the API to identify potential break points caused by it.

C. Google Maps Distance Matrix Api only allows for return 100 elements per call but the complete graph I had to construct requires 10, 000 elements. I had to carefully split the elements to blocks less than 100 and combine them together after the Api was called.

ii. **NUS Mods**:

A. Not all locations on NUS Mods were identifiable on Google Maps, I had to create a systematical approach to convert the locations on NUS Mods to locations that are identifiable on Google Maps.

iii. **Credits**: Utilized NUSMods and Google Maps API to get the necessary data for the feature.

- **Code contributed**: [Repo Sense]
- **Other contributions**:
  - Project management:
    - In charge of making sure the team meets the weekly deadline and delegated issues among the team.
    - Managed releases `v1.1` - `v1.4` (4 releases) on GitHub
    - Created `Issue` and `Pull Request` templates for more standardization in the repository.
    - Introduced an agile planning board here
    - Designed TimeBook logo for additional branding.
  - Enhancements to existing features:
    - Wrote test cases for storage component and increased coverage by 4% (Pull requests #261)
  - Documentation:
    - Adopted Open Api Specification style of documentation into the UG for a clearer, more standardized and easier to read user guide.
      - Group each command into `tag` so that we can split explanation of the User logic into the tags explanation and the detailed command logic into the command explanation.
  - Tools:
    - Integrated a third party API(Google Maps) to the project
    - Introduced Lucid Chart to the team for easy design of UML diagrams.

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.

Closest Common Location

TimeBook will also help you to decide where to meet for your project meetings by suggesting to you which location is the center for everyone. The center location is the closest location for everyone to meet in NUS based on their schedule. In time book we define the center location for everyone as the *closest common location*.

Steps

1. Setup your group by:
    a. Adding your group members to your TimeBook with `addperson` command
    b. Adding your group to TimeBook with `addgroup` command
    c. Add your group members to your group with `addtogroup` command
2. Find the free common time when you want to meet with `show` command
3. With the free time slot `id` get the details for the closest common location with `popup` command
4. Press `esc` button to close the popup

Constraints

Due to the lack of internet connectivity, TimeBook will only support locations that are in NUSMods and identifiable by Google Maps. The full list of the supported locations is below.

Definitions

1. We define closest common location as the location that requires the least average distance to travel to from different sources by car.
2. Locations that are not supported are voided and are not used for the calculations of the closest common location and average travelling distance.
3. Refer to the `Developer Guide` on how we compute the closest common location.

Closest Common Location Popup `popup`

command: `popup g/ GROUP_NAME i/ID`

Examples:

- `popup g/CS2103T i/2`

Below is an example of a popup.

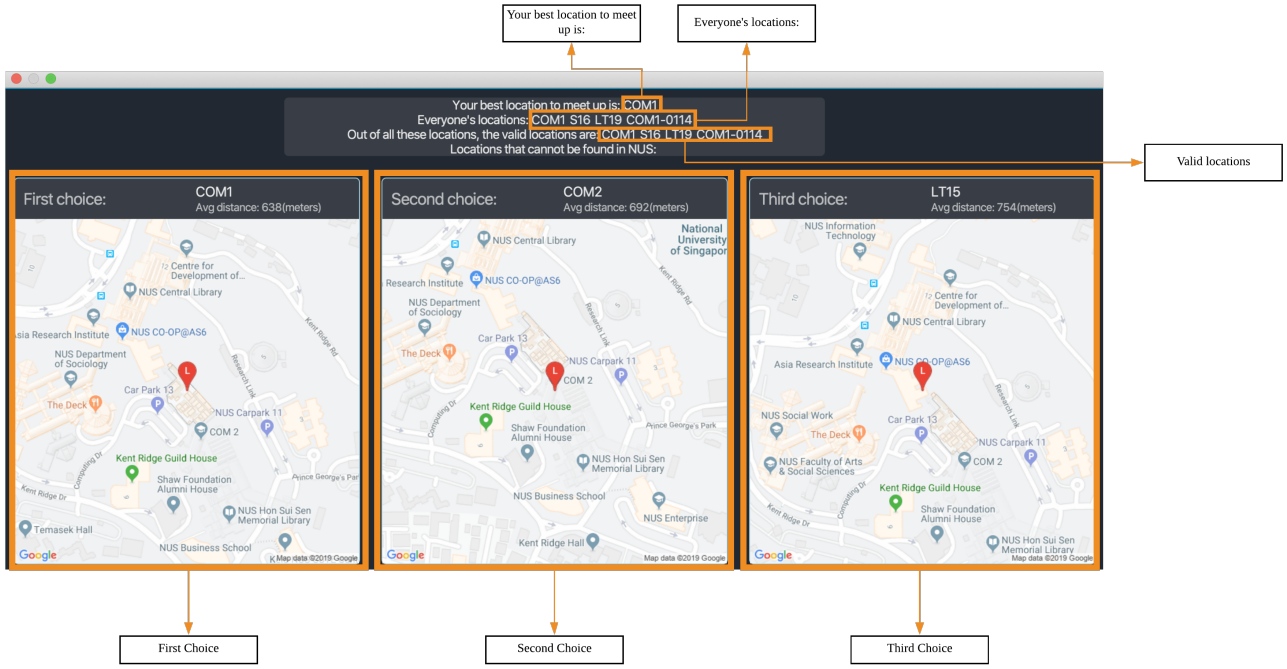

Figure 3. An example of a popup when `popup` is executed

Description

When the command is executed a popup will appear that shows the all the information of the closest common location for a particular free time slot in a group.

Parameters

1. `GROUP_NAME`
    - Description: This is the name of the Group you want to find the free time for. The `groupname` can be found on the `groups` tab in the user interface.(Use `switch-tab` if your tab is on persons)
    - Type: `String`
2. `ID`
    - Description: This is the `id` of the free time slot of a group. You can get this `id` from the user interface when you use `show g/[groupname]` command. You can only enter the `id` that is shown on the screen.
    - Type: `String`

Success Response

If you follow the steps above, a popup will appear as seen in the Figure above.

There are two main information that you will see in the popup when you execute ththe command.

1. The summary of the results will be displayed on the top of the popup.
    a. **Your best location to meet up is:** The first choice location to meet(closest for everyone).
    b. **Everyone's locations:** List of locations that the group members will be at before the free time slot.

c. **Out of all these locations, the valid locations are:** List of TimeBook supported locations that the group members will be at before the free time slot that.

   i. Subset of the **Everyone's locations:** list

d. **Locations that cannot be found in NUS:** List of locations that the group members will be at before the free time slot but not supported by TimeBook.

   i. Subset of the **Everyone's locations:** list

   ii. If all the locations are valid, this attribute will be empty as seen in the Figure above

2. The full details of the first, second and third closest common location will be shown on the bottom of the popup.

   a. First choice represents the first closest and so on and so forth.

   b. The average distance(m) to reach the various location will be displayed below the respective choices.

   c. The picture of the location of the venue on Google Maps will be displayed below each choice.

      i. For locations that are not supported on Google Maps, a picture of the center of NUS will be displayed.

Failure Response

- `Error` when all the source locations are not recognised by TimeBook, a popup will not appear.

Feedback box will show:

```
 We could not find a common location because all places cannot be found in NUS. The
locations are:
```

- `Error` when all the group members are not in school before the free time slot, a popup will not appear.

Feedback box will show:

```
We could not find a common location because:
Everyone has not started their schedule yet. Feel free to meet up any time.
```

- `Error` when invalid free time slot `ID` entered.

Feedback box will show:

```
Invalid time slot ID: 0. Please enter a valid id as shown in the GUI.
```

- `Error` when invalid `GROUP_NAME` entered.

Feedback box will show:

```
Cannot recognise GROUP_NAME. Make sure you entered the correct value.
```

Close popup `esc`

After you are done with viewing the information on the popup press `esc` button on your keyboard to close the popup and and return to the main screen.

Format: kbd:[esc] button on keyboard.

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.

Closest Common Location

Closest common location utilises Google Maps Api to get the best center location to meet for a group project meeting. We define this location as Closest Common Location. Below is an example of this feature.

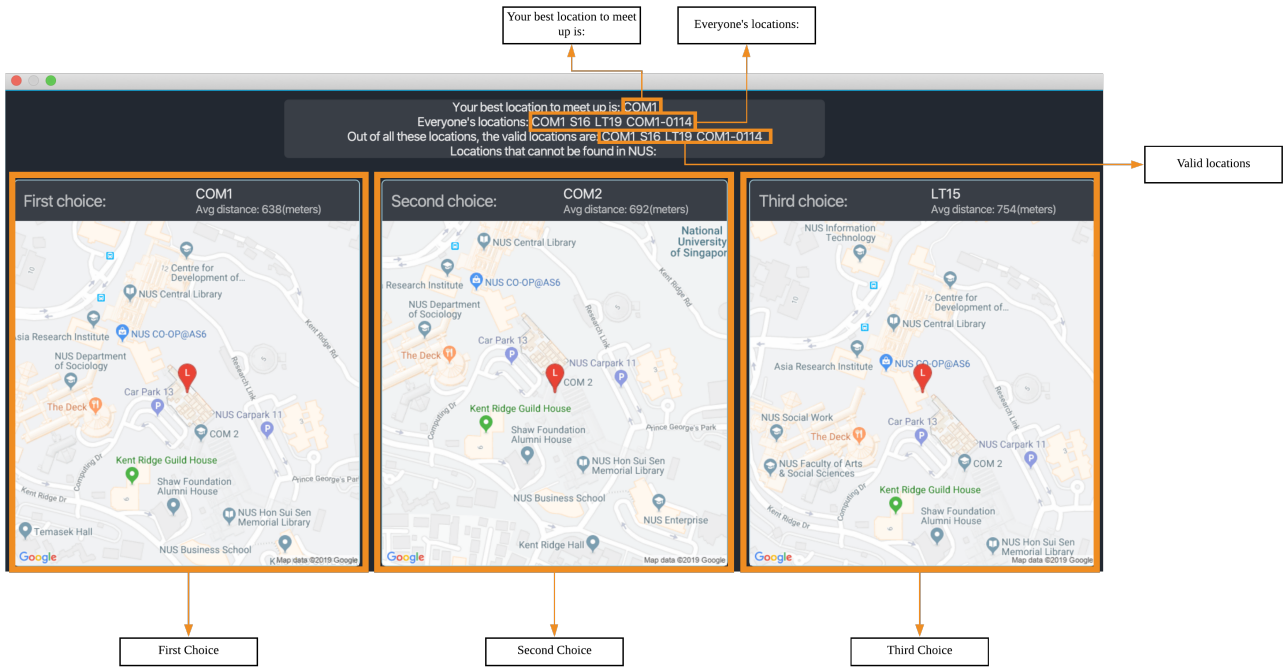

Figure 4. Popup for the closest common location.

Definition

- Due to connectivity constraints, we cannot support location outside of NUS. View User Guide for the full list of location we support.
- The closest location is the location that has the least average travelling distance by car from the various sources.

- All invalid location are omitted during as the source location will not be considered.

Algorithm

1. Create a complete graph where the vertices are the different locations in NUS and edges are the respective travelling distance by car from location `u` to `v`
2. Represent this graph in a v x v matrix where `i` represents the source location and `j` represent destination location and `distanceMatrix[i][j]` represent the time needed to travel from `i` to `j`
3. To get the closest common location of S1 ... Sn:
    a. Get the rows i = l1 ... ln
    b. Sum the values of the rows to a new row `totalDistance`
    c. The smallest value in the row is the closest common location

Below is an example of how the algorithm is applied on arbitrary locations `l1⋯ln` with arbitrary travelling distance to compute the closest common location for `l2,ln-2 and ln1`.

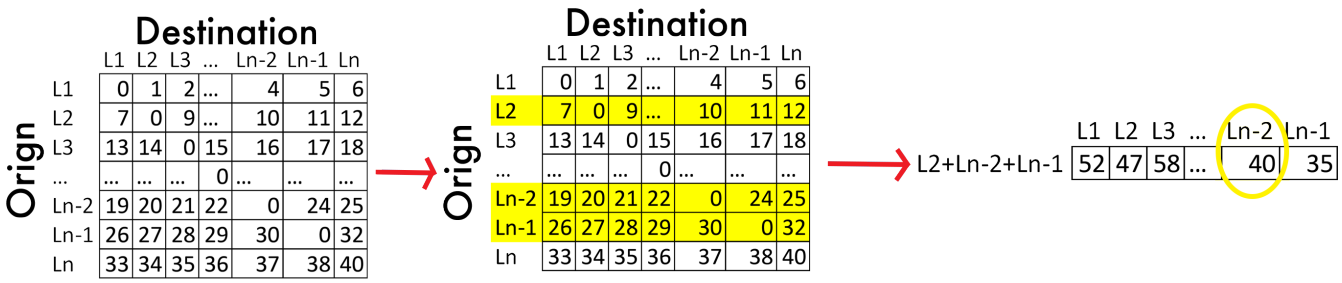

Figure 5. Example of how the algorithm is used. The closest common location for this instance is `ln-2`.

Implementation

Consideration

1. Google Maps Api only provides free $400 credit per account and every 1000 API call cost USD$10-USD$20.
2. Google Maps Distance Matrix Api has a limit of 100 elements for every API call.
3. Google Maps Api has bug
    a. Inconsistency in the identifying locations. Example
        i. `NUS_LT17` identified as the correct location and `LT17` is not.
        ii. `NUS_AS6` is not identified as the correct location but `AS6` is identified as the correct location.
    b. Certain locations are not supported by Google Maps
        i. `S4` and `S6` is identifiable but `S5` is not.
    c. Some locations are valid on Google Maps Places Api but not on Google Maps Distance Matrix Api.
4. Not all venues on NUSMods are identifiable on Google Maps Api.
5. Some venues on NUSMods are in the same building(ie AS6-0213 and AS6-0214).

Implementation

The image below represents the Class Diagram for Closest Common Location component of TimeBook

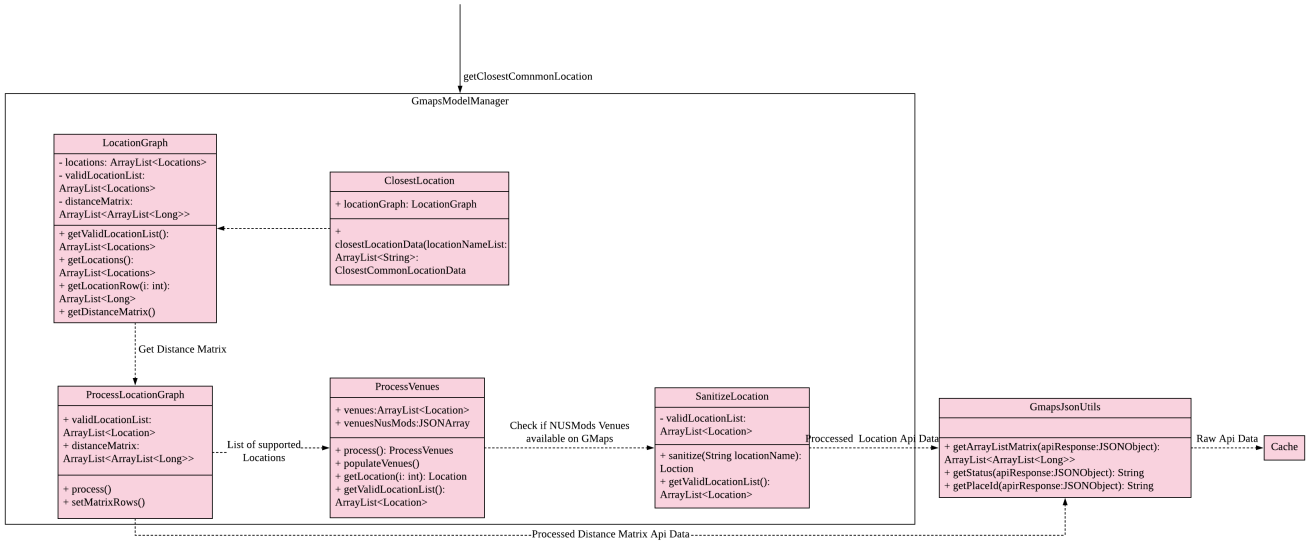

Figure 6. Class Diagram for Closest Common Location Component

There are 3 main aspects to the implementation of this component.

1. External API
2. Creating the matrix
3. Getting the closest location

External API

To support the limited internet connection, we preprocess the relevant data and save it into the resources directory (See External APIs).

Constructing the graph matrix

Below is the sequence diagram for the creation of the matrix.

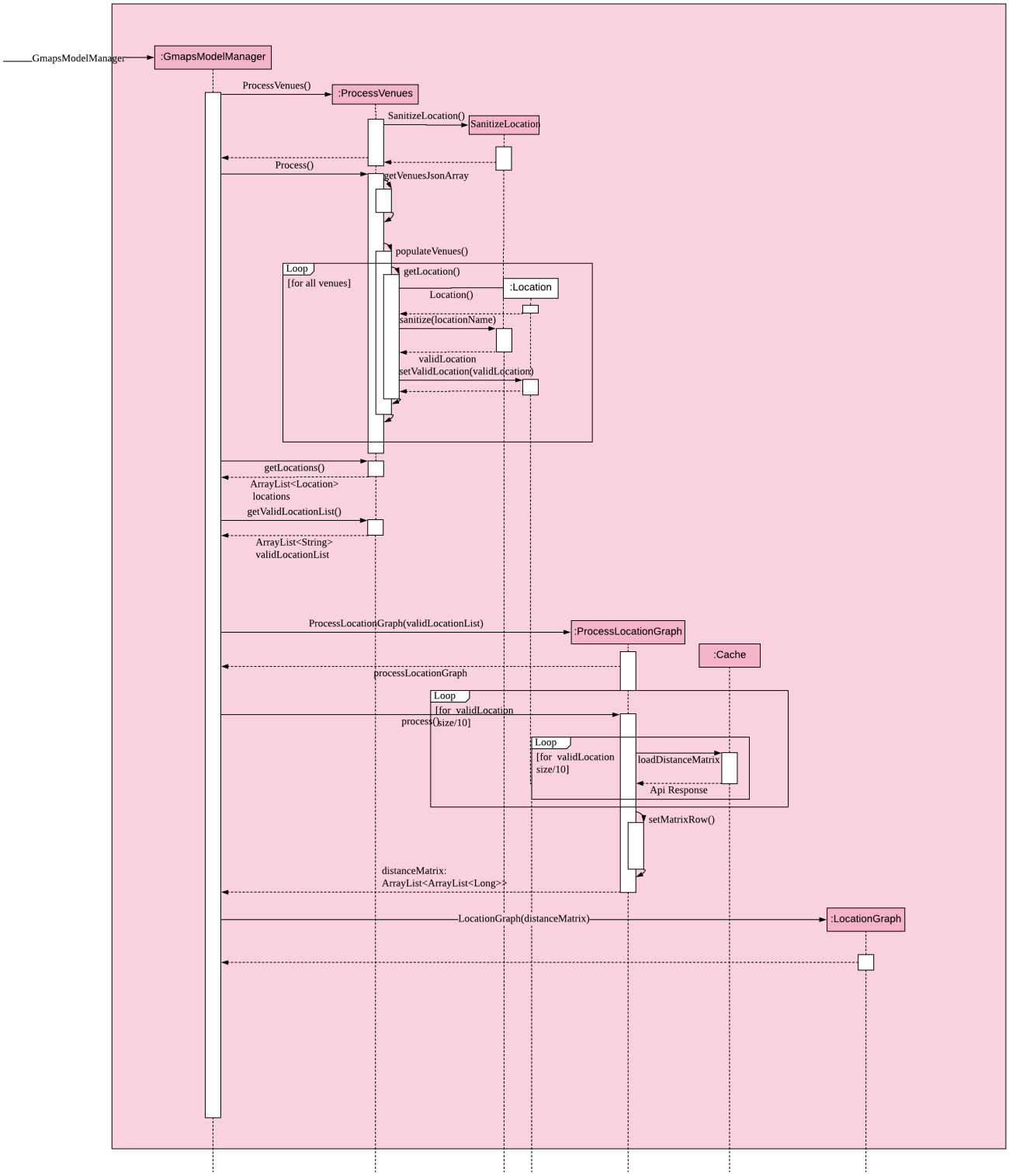

Figure 7. Sequence diagram for the construction of the graph matrix

Brief overview The initialising of the matrix is broken into 2 steps. The first step is to get the list of locations in NUSMods and checking against google maps API if that location is identifiable by google. The second step is to use the identifiable location to construct the matrix.

Steps

1. Check if the name of the location in NUSMods is identifiable on google maps. `ProcessVenues#process` is the driver for this step.
    a. Call NUSMods api with `Cache#loadVenues` to get an array of Venues(`String`) in NUS,

b. Iterate through each venue and sanitize it to Google Maps Identifiable location.

   i. Sanitizes the location name given by NUSMods by appending `NUS_` to the front and removing any characters after `-` or `/` as the room in the building does not matter. This will help to reduce the cost of Google Maps API calls.

   ii. `UrlUtil#conditionalLocationName` maps the location name that are not supported on Google Maps to a valid location name.

   iii. Each venue in the array will have a `validLocationName` and `placeId` mapped to it in the `Location` class. This will help with the generation of Google Maps Distance Matrix Api and retrieving of the location image from Google Maps Maps Static API

2. Construct matrix. `ProcessLocationGraph#process` is the driver for this step.

   a. Get the list of valid location with the relevant data(`placeId` and `validLocationName`)

   b. Divide this list into blocks of 10 to keep under the 100 element limit of Google Maps.

   c. Call Google Maps Distance Matrix Api for all the blocks in the list.

   d. Combine the API response into a single 2-Dimensional array where `distanceMatrix: ArrayList<ArrayList<Long>>`.

   e. Use the constructed 2-Dimensional to instantiate `LocationGraph` which would be utilised to compute all the closest common location.

Getting closest location

`ClosestLocation#closestLocationData` executes algorithm above to compute the closest common location. Similar to how `JSON` is used to transfer data in `HTTP APIs`, `ClosestCommonLocationData` is used to transfer the relevant data to the `UI` to display the popup.