

Info 3308

Map navigation

~DMAPS~

Participating students:

Joseph maary 50906

John-Paul Ali 53377

Project owner/supervisor:

Doctor Ralph El Khoury

Summary

* Introduction
* Brief description of the main aspects of the project.
* Brief description of the platform used for development.
* Analysis
* Thinking Map!
* Map data retrieval
* Linking data with the used platform
* Implementing dijkstra to our map
* Implementation
* Step by step description showing the building process of our platform
* Set backs
* Description of all the control and classes used.
* Conclusion
* Short paragraph explaining what we have learned through the whole process.

Introduction

One of the most asked questions in the world of technology are:

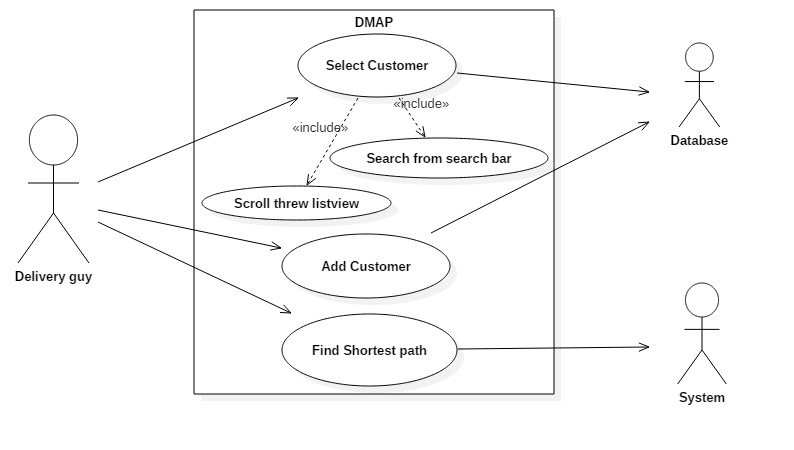
How does google Maps know my location?

How does google Maps calculate the distance between two geographical points?

How does it even provide us the shortest, fastest and the cheapest route in terms of traffic?

DMAP, the name of our application, was simply created by taking the first letter of the word **Dijkstra** and the word **MAP** since, as we stated earlier, it is an application simulating google maps work using Dijkstra.

In this project we will be implementing our own code trying to simulate in one way or another the functionalities of google maps. In specific, we will be implementing Dijkstra’s algorithm helping a supermarket’s delivery drivers to access a customer’s location at the fastest time possible.



Use Case Diagram.

This application is going to be built using ‘Android Studio’ Software for mobile applications where XML and JAVA are its primary languages.

Although, we could have picked another platform for implementation such as ASP.NET, but we have been accustomed to the ‘Android’ platform. In addition, we thought about the use of the application.

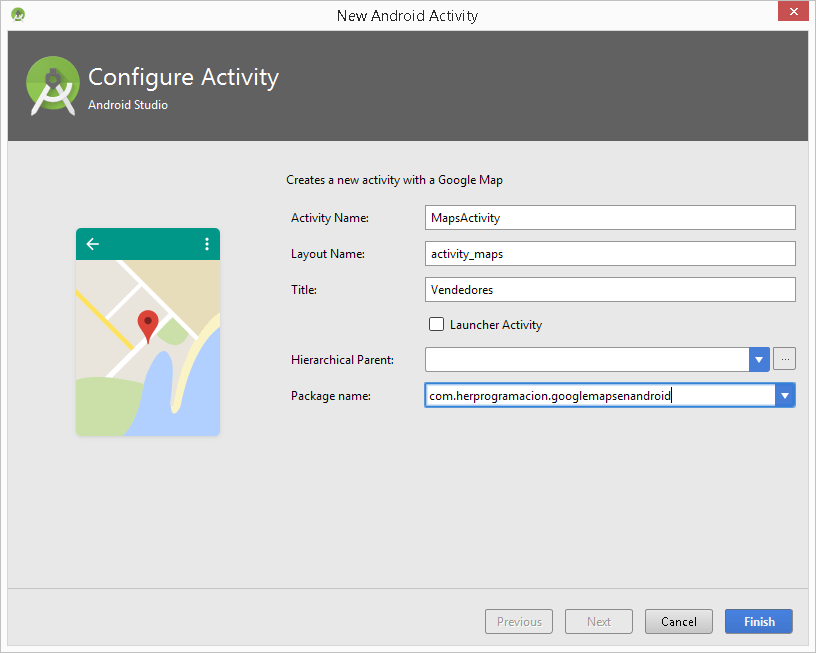
Since it is a ‘map’ application, users might want to discover certain areas unexpectedly. So having the application as a start on the phone is much easier and practical for a user than to have it built on a computer.

Analysis

* Thinking Map!

Considering ‘Android Studio’, many libraries can be included into our project such as Google Maps, Mapbox.

In fact, Android Studio has a Google Maps finished template for developers to use.

****

A corresponding API is used to import all the necessary libraries.

Reference for google maps Implementation:

<https://developers.google.com/maps/documentation/android-api/>

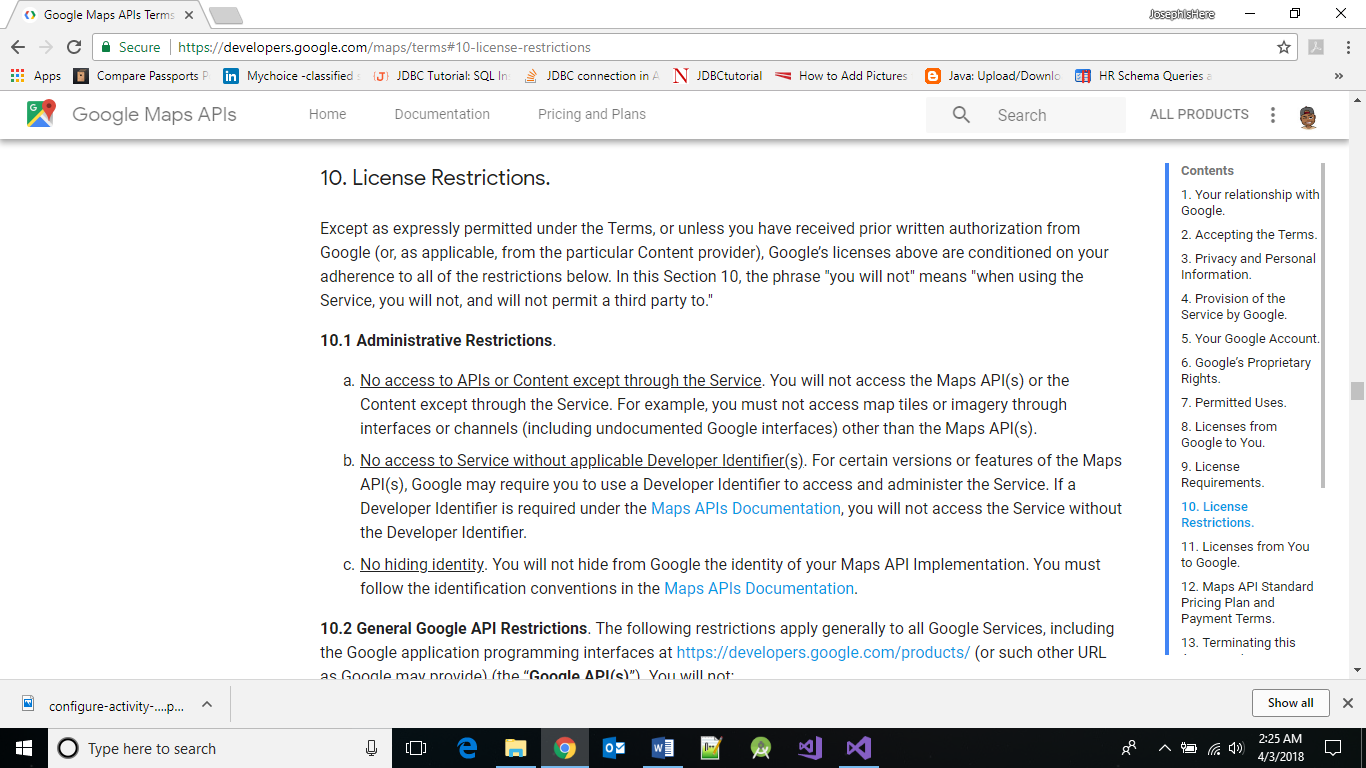
Reference for Mapbox implementation:

<https://www.mapbox.com/help/first-steps-android-sdk/>

These libraries are responsible for map manipulation in all aspects. These libraries can be really helpful providing us the right built-in functions, classes and tools.

Unfortunately, map navigation and route surfing algorithms have already been implemented in them. In addition, developers and programmers cannot overwrite the existing functions by writing their own algorithms for map manipulations.

Moreover, accessing Google Maps own functions from any other application not using Google Maps API is prohibited since it violates the ‘terms of use’ agreement of Google Maps



1. **No access to APIs or Content except through the Service. You will not access the Maps API(s) or the Content except through the Service. For example, you must not access map tiles or imagery through interfaces or channels (including undocumented Google interfaces) other than the Maps API(s).**

* Map data retrieval

Using our own customized algorithm in the application requires a proper database containing information about the area where we are willing to find the shortest path between two geographical locations.

Not able to use any APIs in our project makes the importing process a lot harder.

Its difficulty lays in finding the coordinates of roads for a certain geographical area.

Why using roads coordinates?

Dijkstra is a shortest path algorithm that, having one source node and one destination node in a graph, calculates the shortest path between the source and destination, using the weight of the edges consisting a graph. In our case, these edges are the roads in certain geographical area.

After a lot of internet search for map data information to collect for database use, we laid eyes on an Indonesian website that serves exactly what we want.

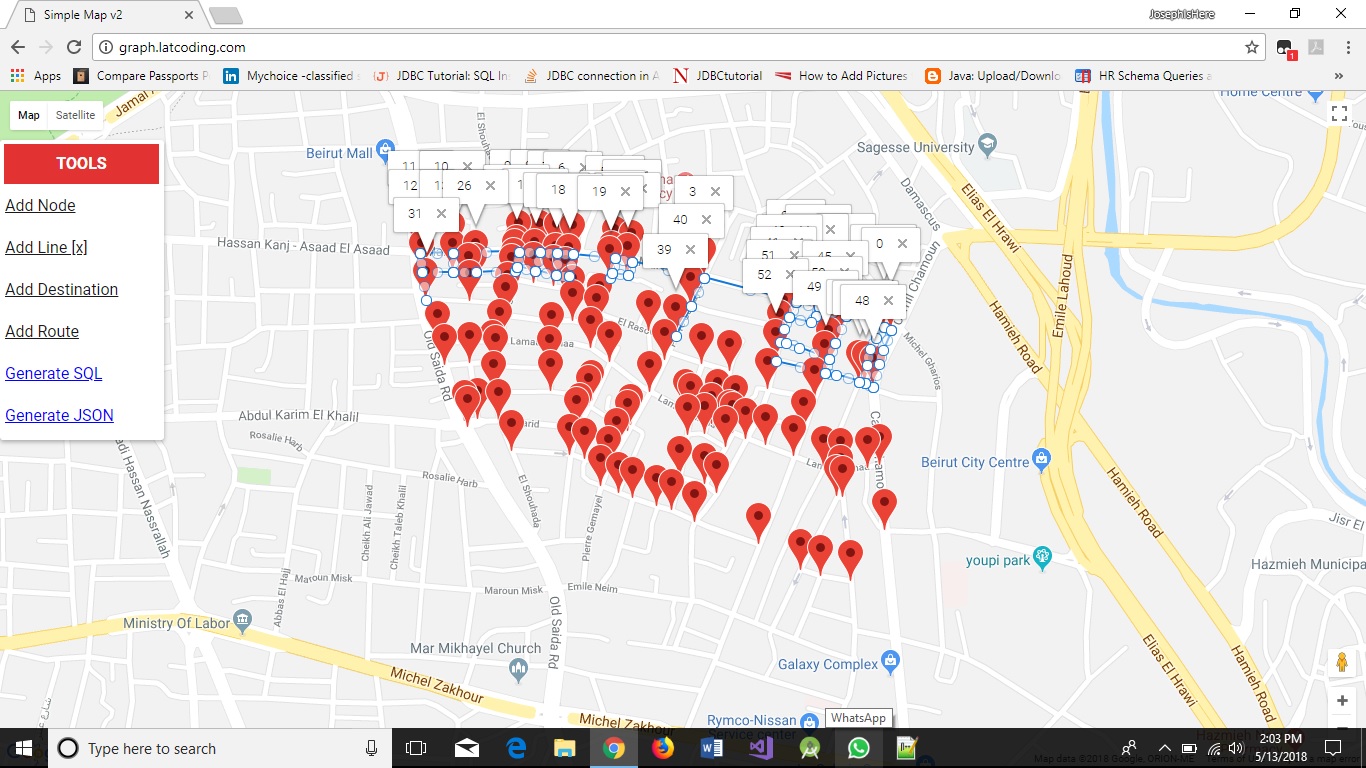
This website is called: [graph.latcoding.com](http://graph.latcoding.com/) .

It provides us with the capability of customizing our own geographical area.

A google maps similar map appears in a webpage, along with a menu table on the left, which contains the Add node, Add Line, Add Destination, Add route, Generate SQL and Generate JSON buttons.

We used the “Add node” and “Add line” features to build our small graph on this map linking one waypoint to others using the “Add Line” button.

Example of the process:

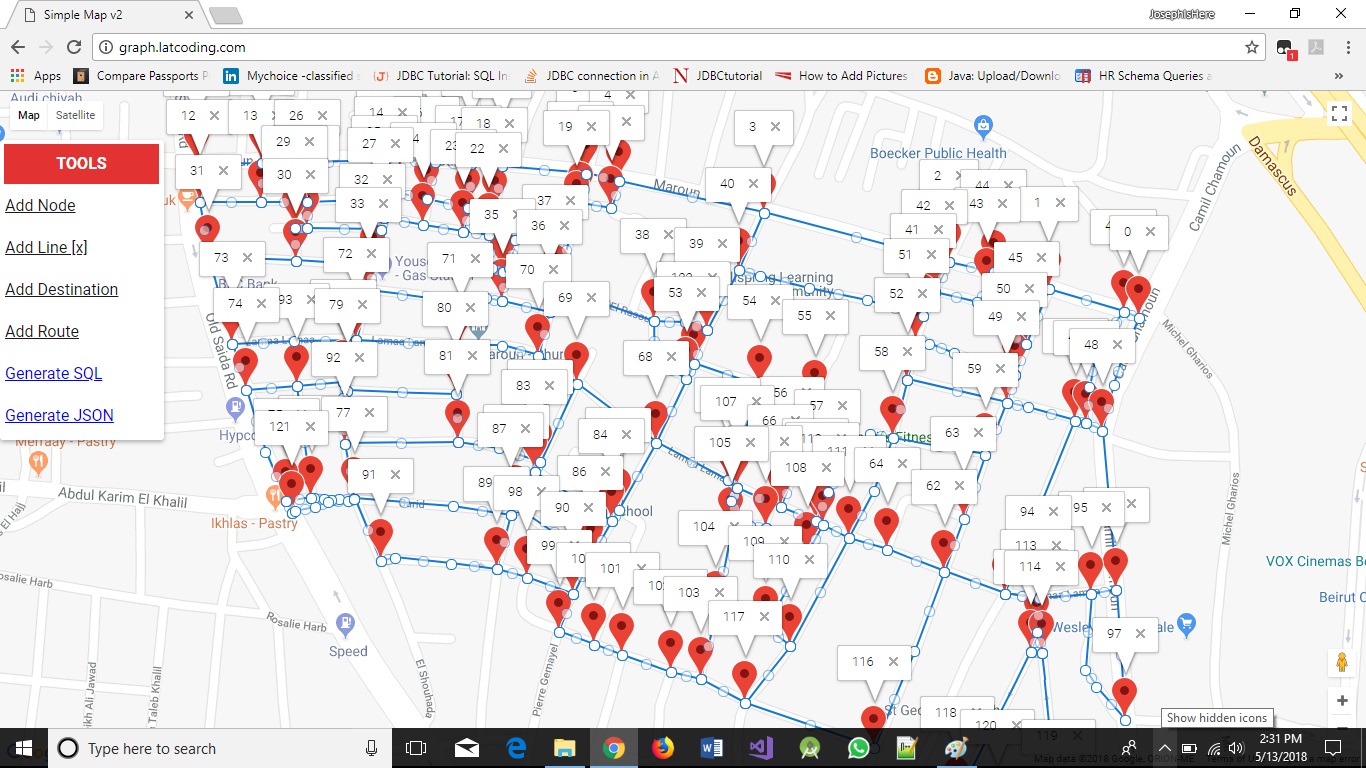


Line drawn using “Add Line”

Node clicked using “Add Node”

The choice of the nodes (waypoints) must be precise or the result we are going to have will not be efficiently accurate!

After a LOT of clicks and links, the tiring process (162 nodes clicked with intermediate lines drawn between them) results with a final output that looks like this:

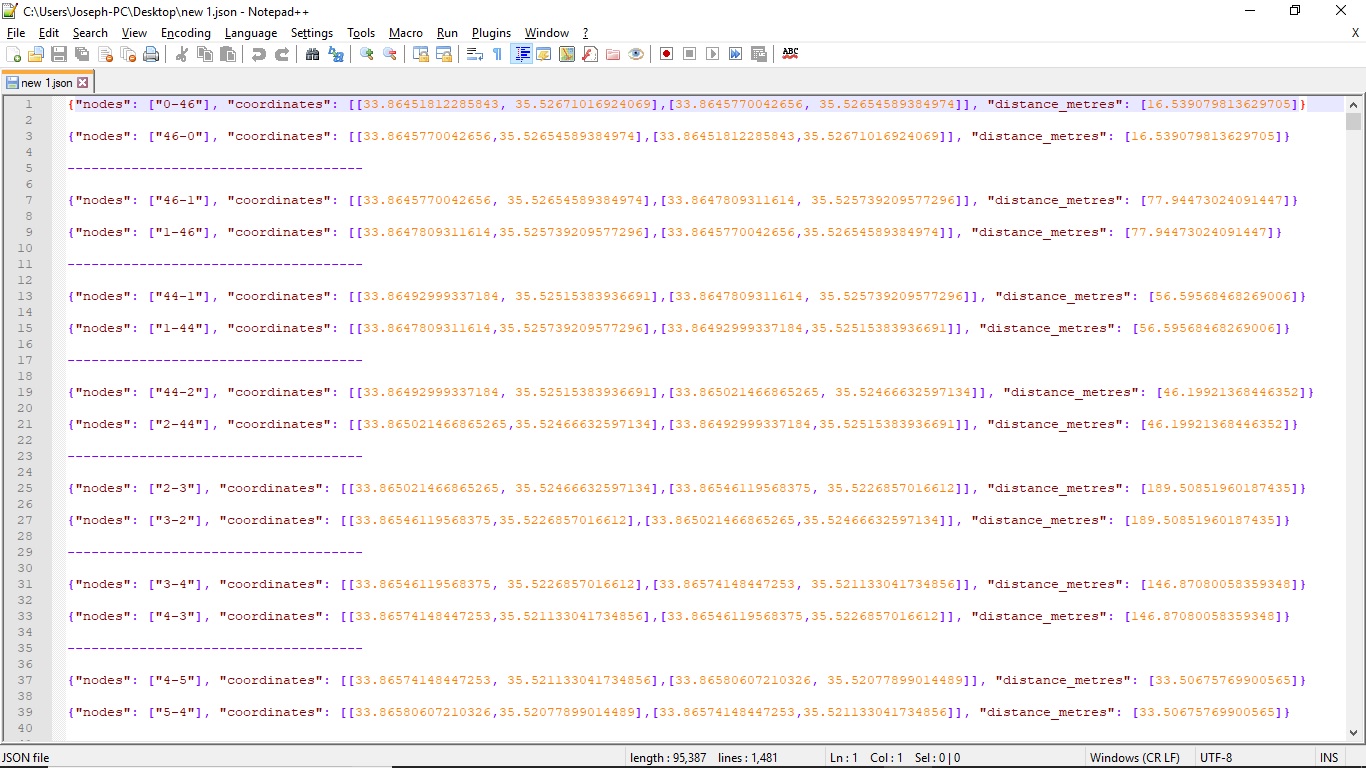


The resulting graph is a small geographical area in Ain El Remeneh – Beirut.

Extracting the data from this graph is done by clicking in the Generate JSON button on the left of the page.

A JSON code file is generated containing:

* + 1. Nodes in contribution (ex: [2-5])
    2. Coordinates of the line between the source and destination nodes including the source and destination.
    3. Distance in meters of the line joining these two nodes, in other words, the road between these 2 geographical points



494 rows in total

* Linking data with the used platform

In order to benefit from the data collected from the [website](graph.latcoding.com) saved in the JSON file, we created a SQLite database using SQLite Administrator.

SQLite Administrator is a powerful tool if you easily want to create, design or administrate SQLite database files.

Why SQLite administrator?

Android Visual Studio has a built-in SQLite database implementation.

Using SQLite administrator, we created a database that contains one table with the following columns:

* + - 1. id ( id of the row)
      2. start\_vertex (the starting node)
      3. destination\_vertex (the ending node)
      4. path (an array containing the coordinates of the road between the start vertex and the end vertex)
      5. distance (distance between the start and end vertex)

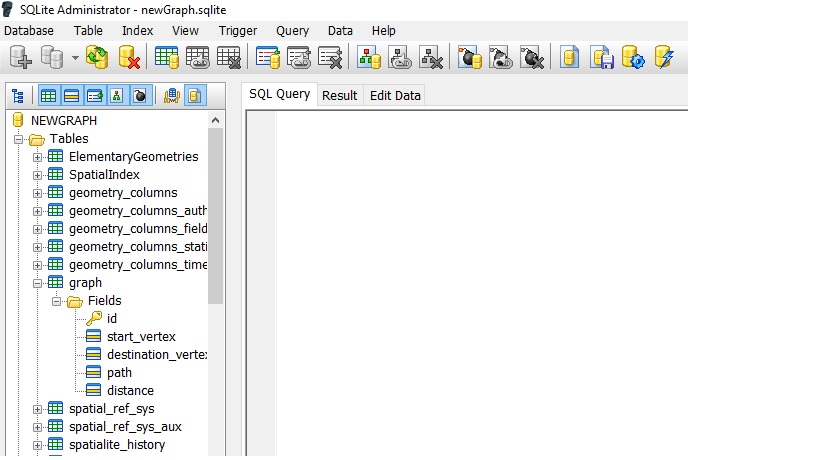
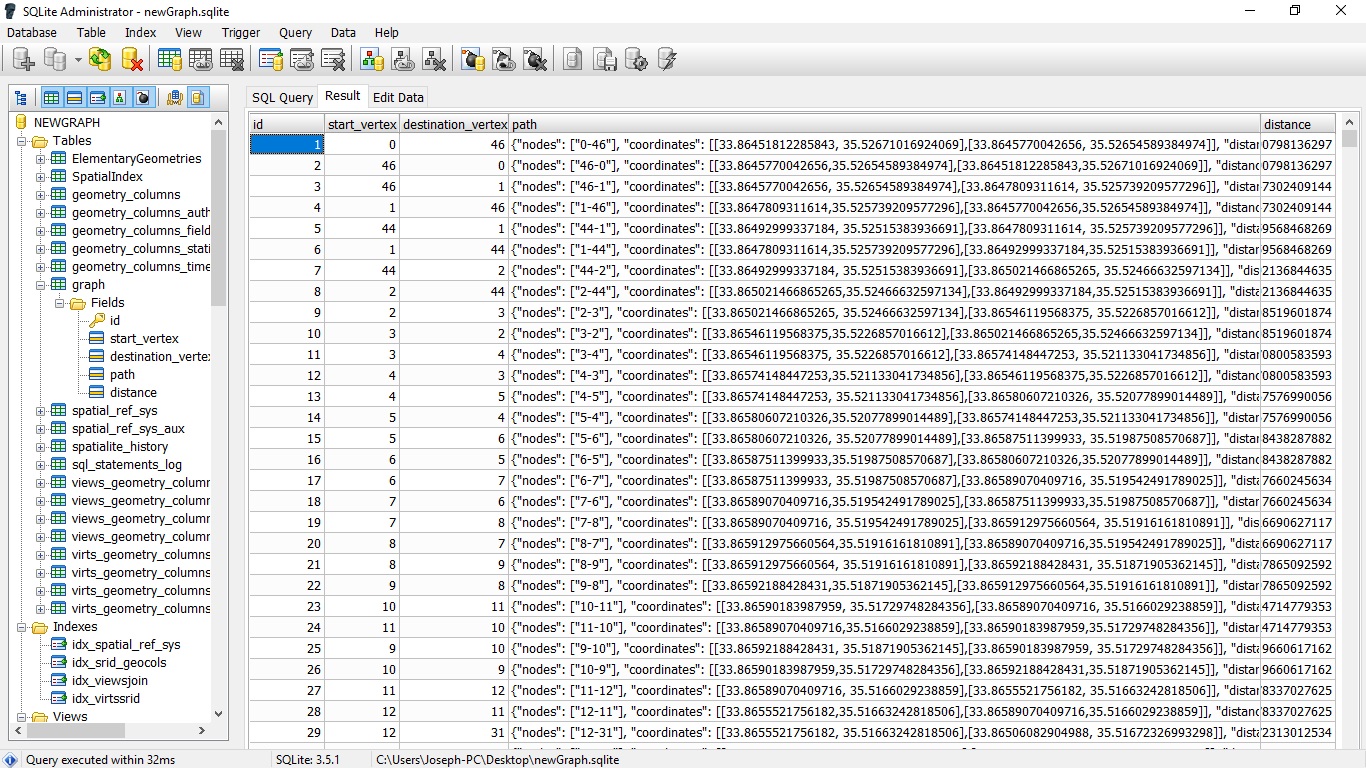


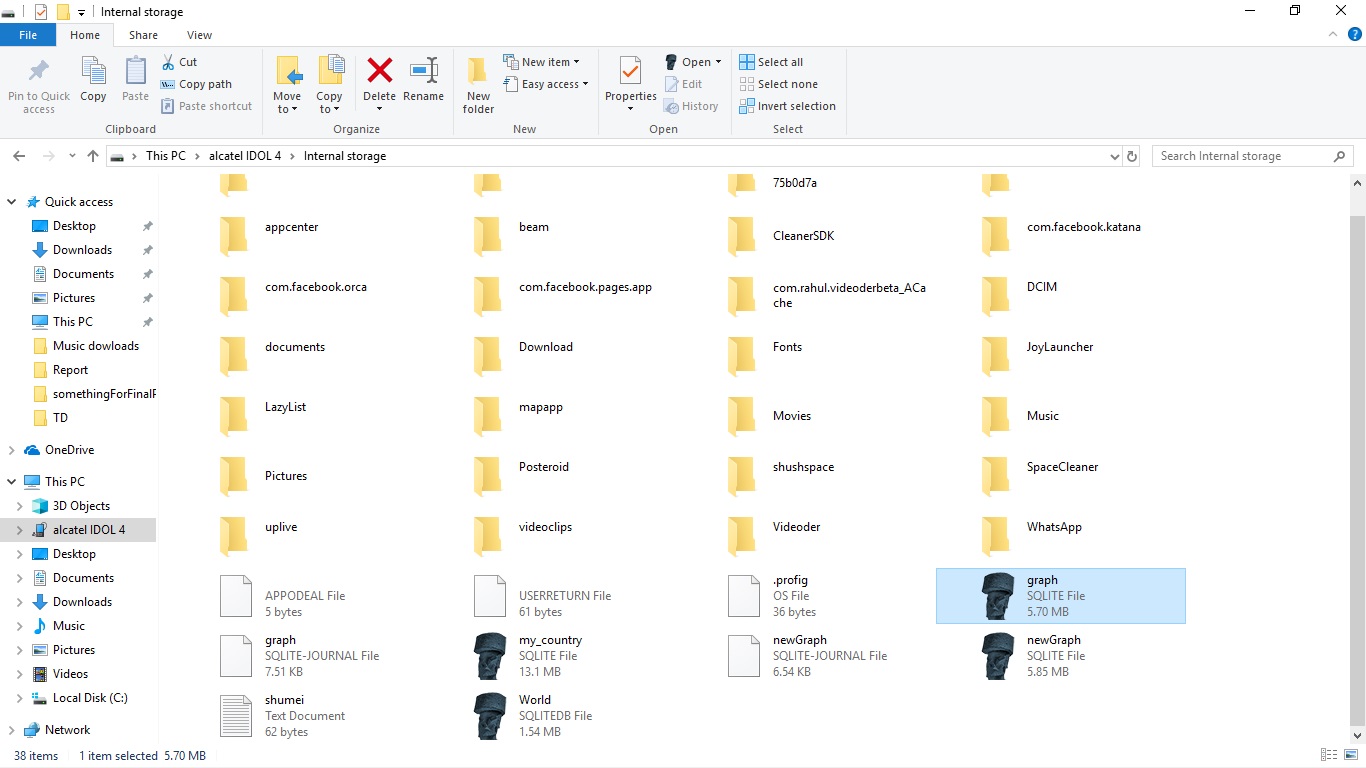
Table “graph” containing the previously mentioned columns

Moreover, this is a preview of all the inserted rows into the SQLite database:

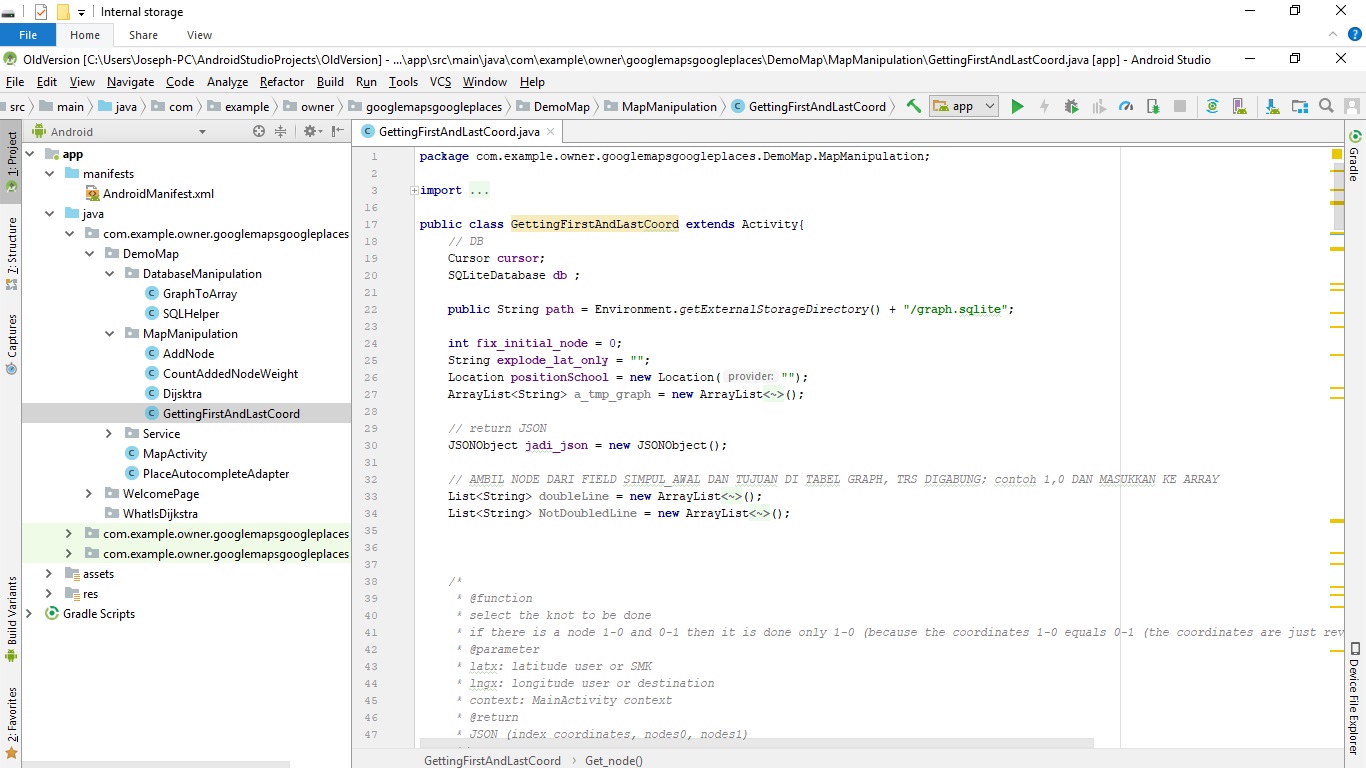


As we mentioned earlier, Android Studio has a built-in database, which allows a quick and easy database manipulation. We can use the internal methods of Android Studio to read, write, update and delete information from a SQLite Database.

After creating the database on SQLite administrator, we saved the resulting database in a file called “graph” and inserted the file in our phone storage.



This way we can reach the database created using Environement.getExternalStorage() method in Android Studio each time we want to use the data for the Dijkstra implementation.



* Implementing Dijkstra to our map

As mentioned before, **Dijkstra's algorithm** is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist [Edsger W. Dijkstra](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra) in 1956 and published three years later.

Generally, it works as follows:

.It picks the unvisited vertex with the lowest distance,

.Calculates the distance through it to each unvisited neighbor,

.And updates the neighbor's distance if smaller

This workflow is then repeated at every visited node in the graph until we end up with a path that represents the ideal path to cross from point A to point B with a minimum cost. In our case, the distance is the cost!

Therefore, our basic idea was drawing on the map a line, showing the ideal path to cross for a given location A and location B.

However, this is not as easy as it sounds since any location ‘X’ on the map has two indicators: Longitude and Latitude.

More details are to be discussed in the following sections of the report.

Implementation

* Step by step description showing the building process of our platform:

As discussed before, the main platform used for our project development is Android Studio using XML for designing and JAVA for controlling.

Our first step throughout the process was showing the map as a simple output on our phones as a base map for testing.

Next, we focused on finding the right way to implement all the classes used for dijkstra’s algorithm deployment in our map.

Linking our database with Android Studio, and giving classes related to Dijkstra access for the database for future “Shortest path discovery”.

Testing our work by assigning random point and trying to analyze the results.

Studying the main Idea of the application where we include the Dijkstra’s algorithm feature in. We should point to the difficulties we had creating a suitable idea to goes with Dijkstra.

(More details about the setbacks we faced later in this report).

Designing and styling our application in the most user-friendly way possible. Finally, testing the application as a whole!

* Description of all the control and classes used:

In this section, we are focusing on the classes contributing to the realization of the Dijkstra algorithm and the retrieval of all the coordinates that form the shortest path between point A and point B.

* SQLHelper.java:

Create a connection to the database environment that is located on mobile device storage.

* GraphToArray.java:

Converts the data of the graph table located in the database into a multidimensional [ ] [ ] array form that contains the destination vertex with the weight of the path.

* GetCoordinates.java:

Get from the database the vertices of the path and the node between these vertices that is the closest to the selected position.

* AddVertex.java

Add the node extracted from the latter class to the database as a new vertex.

* AddWeight.java

Calculates the weight of the new added vertex.

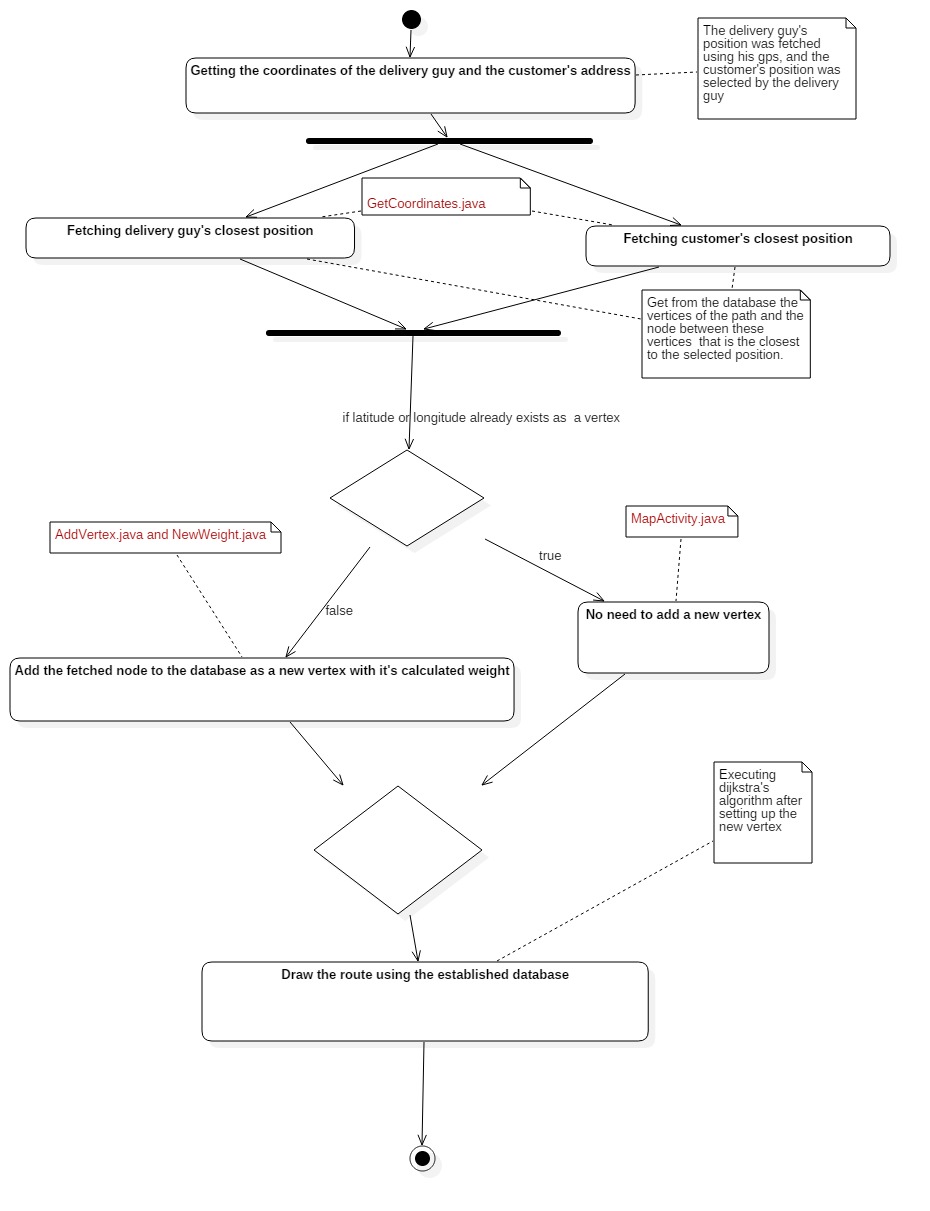
* Dijkstra.java

The “Orchestra Chief” of these whole classes. Performs the required operations for creating the shortest path between the source and destination.

* MapActvity.java

The View Class where all the functionalities are translated in a form that the user can manipulate.

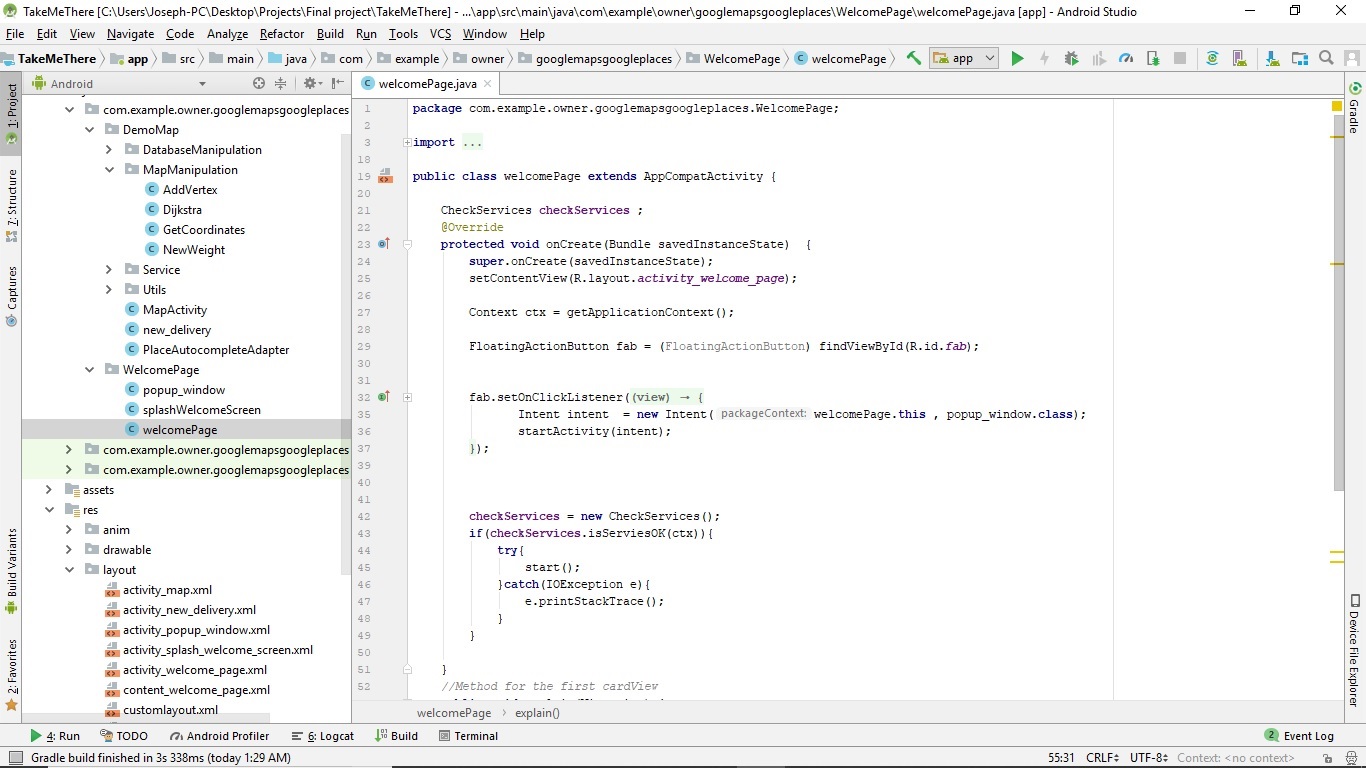
The other classes in this application are standard classes having layout and organization functionalities for this project.

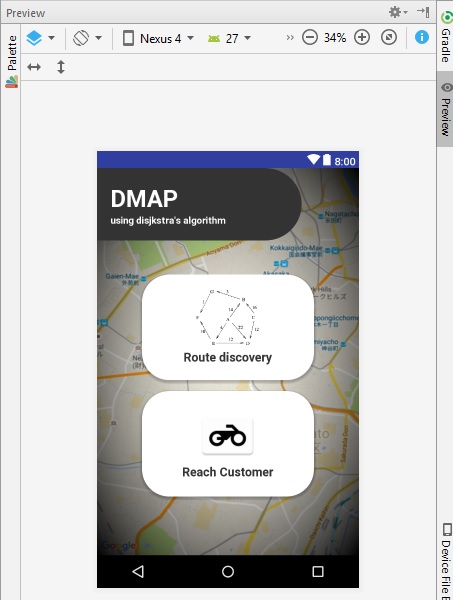


Path drawing Activity Diagram.

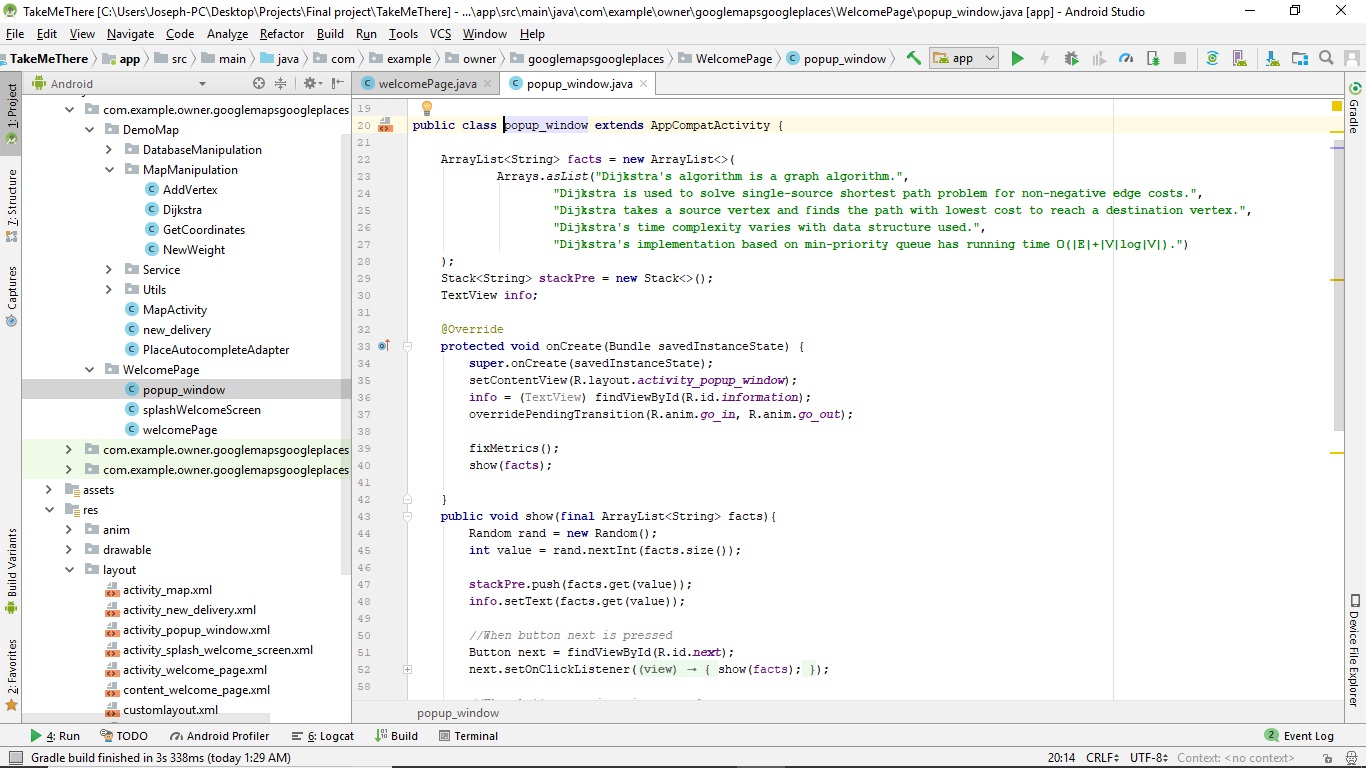
One of our main classes in this project:

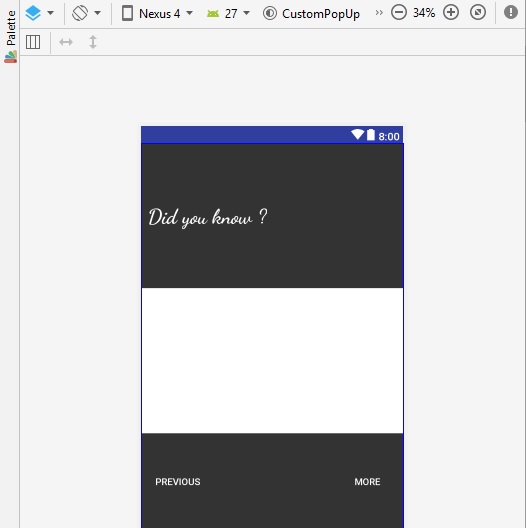
* WelcomePage.java: which is our main page. It has 2 clickable card view images that allows to navigate throughout the application.



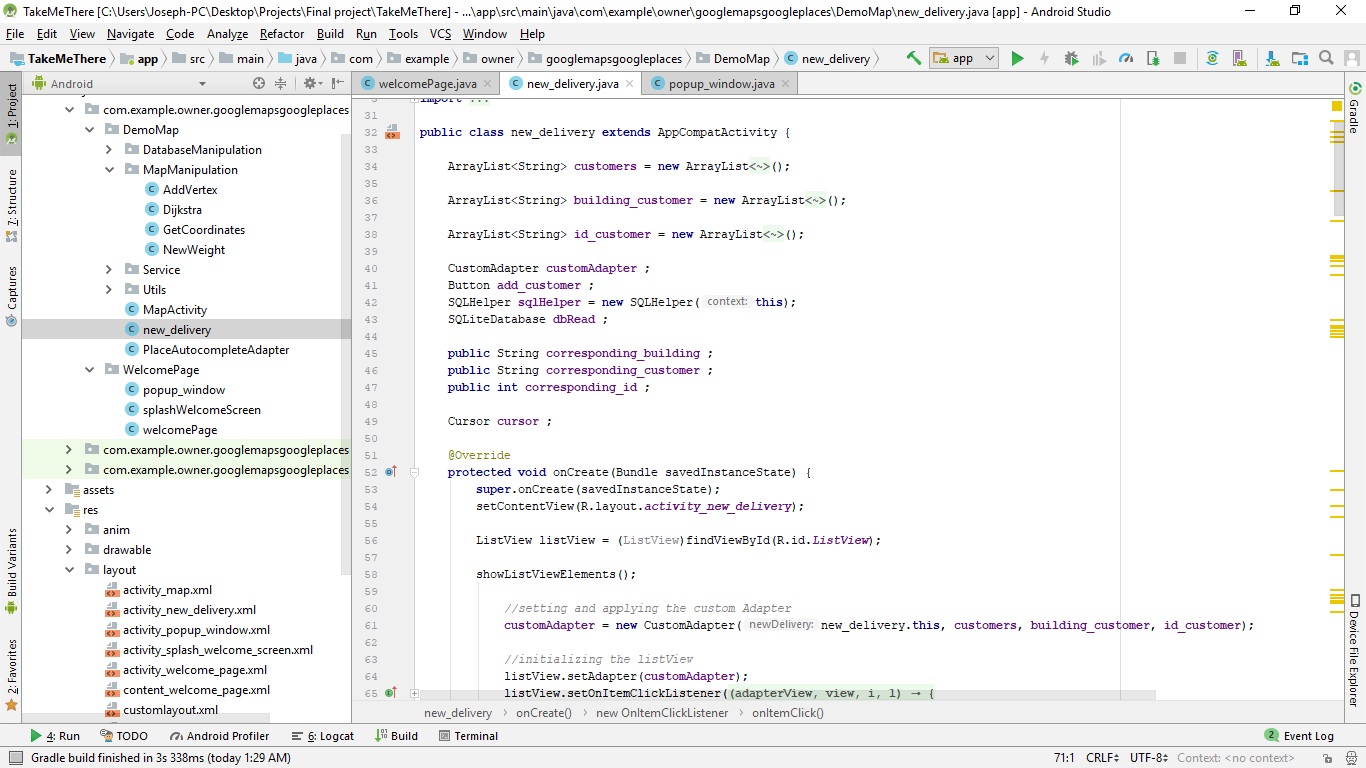


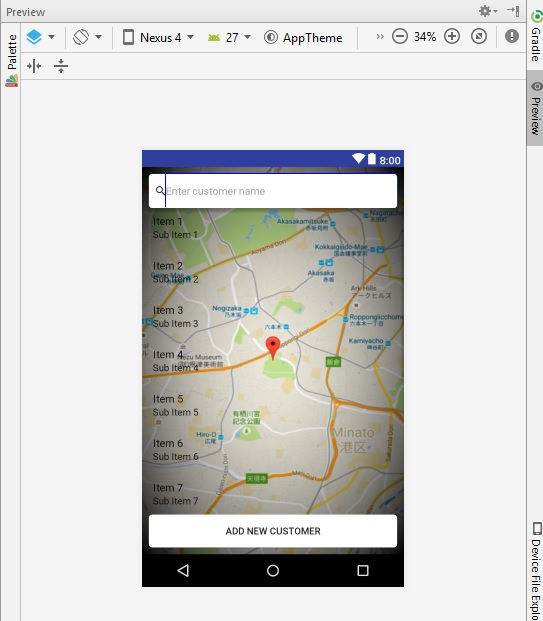
* Pop\_window.java: included inside of the main page. This is pop up activity that gives us some little more information about Dijkstra.





* new\_Delivery.java: Since our application is responsible for giving the shortest path between the supermarket and the desired customer (as mentioned in the introduction section of this report), this class is responsible showing a list view of all the available customers with an option of editing and deleting a certain customer.

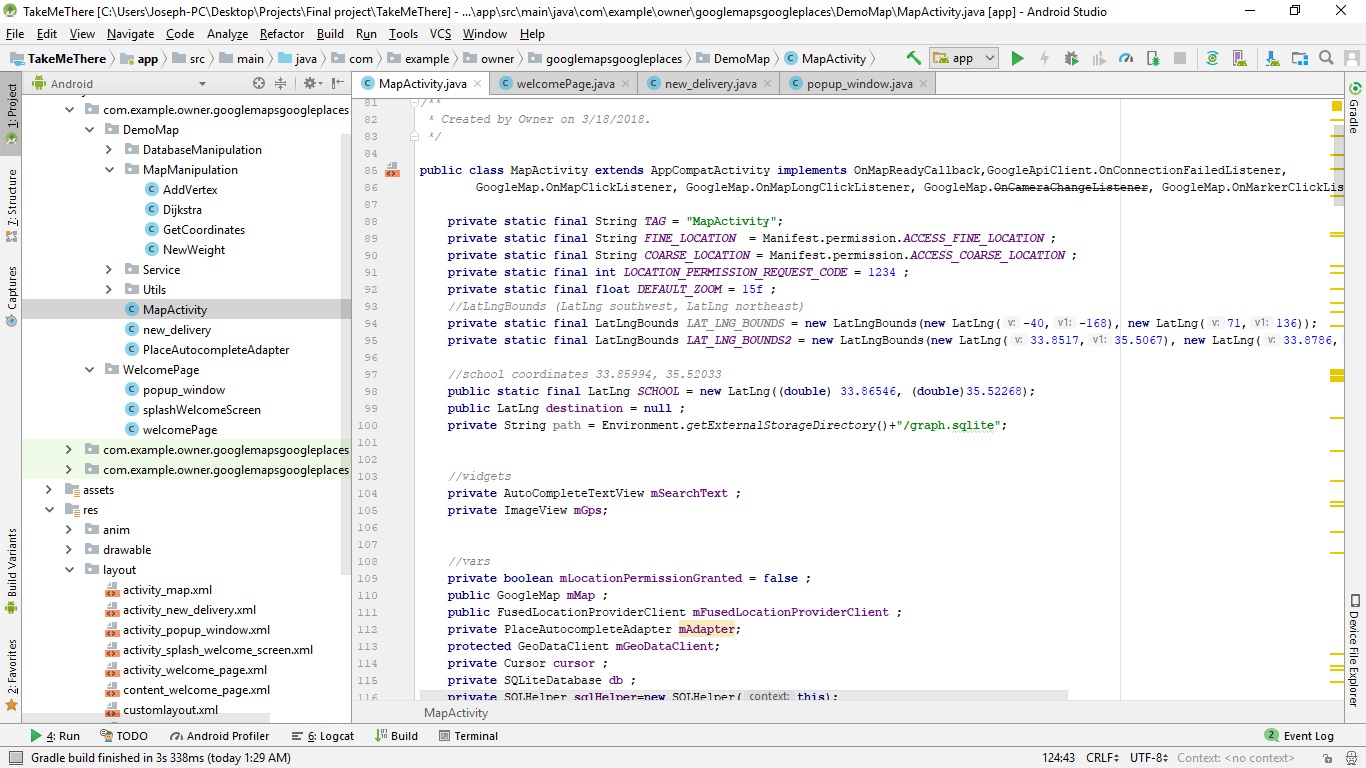




* Map\_activity.java: The kitchen of the whole project where we can see all the resulting work shown in the map.

Many gestures were handled inside this class such as

1. adding a new customer upon click on a certain location,
2. Seeing all the available customers represented by their building (small house on the map) etc…



* Setbacks:

Similar to all project assignments, their goal is to convey a theoretical and empirical knowledge. Moreover, with this kind of journey, comes many setbacks especially in learning new things never taught before in class.

In this project, we faced many setbacks and difficulties before reaching our objectives. Just to name a few:

1. **Learning about maps**

This whole project was a really new idea for us. Understanding how maps are built, understanding the relation between algorithms and coordinates, since we already are familiar with Dijkstra algorithm in simple graphs where a vertex is represented by one point and not two geographical locations using coordiantes.

Not being able to override some of the Api’s code to implement our own code for route navigation and having difficulties finding an appropriate database source for coordinates extraction, gave us no choice but to look elsewhere and start with the basics.

At first we layed on a blog website called: [fuschs maps](fuchsmaps.com) that built a map from scratch using mathematical equations to organize the tiles size and positions for display.

Ps: Tiles are rectangular images that consists a whole map.

As much as how interesting this blog was, but unfortunately, it didn’t provide us with the full freedom of route navigation and coordinates integration.

After jumping around many sites with same problem we faced with [fuschs maps](fuchsmaps.com) , we thought about trying to make our own database of roads coordinates for ease of use.

1. **Including coordinates in our application**

In order to use Dijkstra’s algorithm on an actual map, we had to have a certain database containing the coordinates of all roads in the area of work.

Not having our own database to work with, and even being limited with implementing our own line of code concerning Dijkstra, forced us to look elsewhere searching for map coordinates.

[Openstreetmap.com](http://www.openstreetmap.org) (OSM) is a collaborative project to create a free editable map of the world.

Therefore, we can choose a certain area of interest and download the related database in a “.osm” extension file.

The problem is, the data organization in the downloaded osm file is difficult.

The way the coordinates are stored in the database demands a lot of focus and time to understand and master.

In addition, Android Studio doesn’t read an .osm type file as a database. We forced to convert all the element of the .osm file format manually using special functions depending on the downloaded file architecture which also leads to the crucial time factor we wanted to avoid!

Eventually, [graph.latcodgin.com](graph.latcoding.com) (mentioned earlier in this report) gave us the opportunity to choose our certain area of interest and save the roads coordinates by clicking on a cross-road for vertex representation, and assigning one line between two clicked vertices as an edge.

1. **Many prototypes!**

The long process of learning the philosophy of maps, tiles, point of interests, polylines etc.… led us in way or another to create many useless application just to discover what maps is all about.

This time consuming process left us jumping from one source to another looking for the best map to work on, before realizing that using google maps API without accessing Google’s route navigation algorithms can be done.

Conclusion

What is the real purpose of an academic project in our final year? Why do we think this activity so important that it is worth spending all this time?

Concerning our project, we were exposed to some of the most important ideas to be earned in the field of work. Some of these ideas are:

1. **Effective team work**

Dividing up the tasks, monitoring the work of each person, and integrate these individual efforts into a single package.

In the real world, a project is rarely developed alone, so learning to be an effective part of a team is an important thing to learn about.

1. **Software discovery**

Digging deep into our project while coming up with one final realization about what software to choose concerning our Database-Android Studio provided the capability to discover many new interesting software.

Naming a few we have:

* ***SQLITE ADMINISTRATOR*** (Designing, creating and manipulating database file)
* ***GUI\_SPATIALITE***(is a spatial extension to SQLite, providing vector geodatabase functionality)
* ***QGIS***( free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data)
* ***JOSM***(free software desktop editing tool for OpenStreetMap geodata ).
* ***AtLAS***(collection of maps)

1. **Prototyping**

Building prototypes is a common task in a development project.

At first, we were unable to express our needs without implementing prototypes to visualize what we should and should not do regarding our application.

1. **Oral and writing presentation skills**

Two fundamentally important parts of the project are the written documents you produce and the oral presentations you give.

There is more in to a final year project than “simply writing a lot of code and documentation”.

1. **Packaging/MVC**

The ability to architect the organization of our classes.

Creating suitable packages inside our project holding correspondingly relatable classes**.**

This type of organization reflects the type of work we are going to put in. In addition,

Defining what classes goes along with what packages facilitates the comprehension gap in a development team.

What have we learned throughout all the projects done throughout our final year?

The answer is simple: EARLY EXPERIENCE!

This experience provides a small window to the outside working world.