Faculty of Engineering, Helwan University
Electronics, communication, and computer Department
Computer and Systems Section

Public Transportation guidance web and mobile application with blind mode

Presented by:

Abdulrahman Mohamed Ahmed
Afnan Usama Siraj
Haidy Gamal Mohamed
Mohamed Osama Saleh
Omar Zakaria Ahmed

Supervision:

Assist. Prof. Hesham Keshk

July, 2023

"Dedicated to those who deserve it more than us; to our families."

Acknowledgement

First and foremost, praises and thanks to Allah, the almighty, for his showers of blessings throughout our whole journey of finding a beneficial idea and working as a great team on the project.

We would like to express our deep and sincere gratitude to our project supervisor, Professor Hesham Keshk, for his enthusiasm, patience, insightful comments, helpful information, practical advice, and unceasing ideas that have always helped us tremendously in our research.

It was a great privilege and honor to work and study under his guidance. We are extremely grateful for what he has offered us. We truly hope that our outcome can make him proud of us.

Abstract

Cairo is a big, beautiful city, and we, as many people, want to roam and see each inch of it freely without having any difficulties wondering about how we will go there.
And since technology plays a big role in our lives.
So, we have decided to build a "Public Transportation guidance", web and mobile application, that gives detailed information about the path that the user should take in order to reach his destination.

Keywords

- Android
- Website
- Cairo
- Public Transportation
- Bus
- Metro
- Micro bus
- Blind Mode

Table of content

Table of Figures

List of Tables

Introduction

Introduction

Public transportation plays a vital role in the life of millions of people worldwide. It is an essential mode of commuting that helps people travel from one place to another with ease, especially in busy cities like Cairo and Giza. However, navigating the public transportation system can be challenging, particularly for first-time visitors, people unfamiliar with the city's transport infrastructure or due to the continuous changes in the cities infrastructure.
Nowadays, encouraging Public Transportation by facilitating using it with our project as an approach has been important due to its several benefits over individual car use which makes it an essential aspect of modern urban living by reducing traffic congestion on the roads resulting in shorter commute times and less air pollution. It’s more cost-effective than owning and maintaining a car, saving people money on fuel, parking fees, and car insurance. It’s more sustainable reducing carbon emissions and contributing to a cleaner environment.

Our Graduation Project, titled "Public Transportation Guidance," aims to address these challenges by providing a comprehensive solution to the transportation issues faced by the people living in Cairo and Giza. We have developed a mobile & web application that helps users navigate Great Cairo’s public transportation system by providing information on all possible transportation options available to them, including the total cost, distance and estimated time for each available transportation path.
This book is a comprehensive guide to our Graduation Project, providing detailed insights into its development, and implementation. In this book, we will discuss the various technologies used in the project, the challenges we faced during its development, and the impact that our application will have on the transportation system in Cairo and Giza.

Project Purpose & Problems it aims to overcome

Navigating public transportation in Cairo/Giza can be a daunting task due to the lack of reliable information, multiple transportation modes, and complex route options. Commuters often face difficulties in identifying the optimal routes, their costs, estimated distances & time, tracking user's progress, and receiving real-time updates. Existing applications provide limited features and fail to meet the diverse needs of users. Therefore, there is a pressing need for an intuitive and feature-rich app to enhance public transportation experiences.

Key Features and Benefits

1. **Location Selection:**
   Users can conveniently choose their origin and destination through a dropdown menu populated by Google Maps or by manually selecting locations on the map.

2. **Alternative Paths:**
   Our app provides users with a variety of alternative routes between their selected locations, considering all available public transportation options, including the underground, microbuses, and buses.

3. **Sorting Options:**
   Users can sort the alternative paths based on their preference, cost, distance, or time, to identify the most suitable route according to their priorities.

4. **Total Cost, Distance & Time:**
   Unlike other existing apps, our app offers the unique feature of displaying the total cost, distance & time for each available route, enabling users to make informed decisions based on their budget and preferences.

5. **Approximate Distance Calculation:**

   We calculate the distance based on the main points the public transport will pass through, providing users with a more realistic estimation rather than a simple displacement measurement.

6. **Estimated Time:**

   Our app incorporates Google Maps APIs to estimate the total travel time between each pair of nodes; which are summed to provide the user with the total estimated time of each available route with transportation options and their associated timeframes consideration. This helps users plan their journeys more effectively.

7. **Detailed Route Instructions:**

   Once users select a specific route, our app provides detailed instructions on how to navigate the journey, including information on transfers between different modes of transportation.

8. **Real-time Tracking:**

   Users can track their progress on Google Maps, which displays both the theoretical path of the journey and the actual path that the transports will follow on the ground. This feature ensures transparency and helps users stay informed about their current location.

9. **Real-time Location Sharing:**

   Our app allows users to share their real-time location with authorized friends or relatives, enhancing safety and convenience for both the user and their contacts.

10. **User Feedback & Suggestions:**

    We empower users to give use feedback on a whole route or specific path between any two nodes or even suggest new public transportation options between any two locations, fostering community engagement and continuous improvement of the app.

11.Blind Mode:

We have implemented a friendly blind mode feature, enabling visually impaired individuals to interact with the app through voice prompts and audio assistance. (To Be Shown in demo)

Before diving into the details of our Graduation Project, this book is divided into seven chapters.

Chapter One: Previous Approaches

In the first chapter, we will discuss the previous approaches that have been used to tackle the transportation challenges in Cairo and Giza. We will review some of the existing apps and websites that are currently available to help people navigate the public transportation system in these cities. We will examine their strengths and weaknesses and identify the gaps that our Graduation Project aims to fill.

Chapter Two: Graph Database

In chapter two, we will discuss why we chose to use a Graph Database as our NoSQL database instead of other databases such as SQL or other NoSQL databases. We will examine the characteristics of Graph Databases and how they are a suitable choice for our project. We will also discuss the advantages and disadvantages of using Graph Databases and how we overcame some of the challenges we faced during the development process.

Chapter Three: Technologies, Languages, and Tools

In chapter three, we will provide an overview of the technologies, languages, and tools that we used to develop our Graduation Project. We will discuss the Android SDK, Firebase, Google APIs, our RESTful APIs, Neo4j, and Express.js that we used to build our mobile & wep application. We will also discuss the
programming languages that we used, including Java/Kotlin and JavaScript, and the tools that we used, such as Android Studio and Node.js.

Chapter Four: Functional & non-Functional requirements

In chapter four, we discuss the system requirements such as allowing users to create accounts, add routes, search for paths, and track locations. It prioritizes speed, security, portability, compatibility, and usability to deliver a seamless and reliable user experience. By incorporating functional and non-functional requirements, the system ensures efficient and user-friendly interactions for a wide range of tasks, while safeguarding sensitive data and adapting to various devices and environments.

Chapter Five: Design Specifications

In chapter five, we discuss the system's requirements when it comes to the designing and specifications which were used to implement the system, in addition to showing the system's diagrams like the use-case and the sequence diagrams.

Chapter Six: Testing

In chapter five, We discuss our project's testing, specifically our RESTful APIs as it's crucial to ensure the accuracy and reliability of the information displayed on our app. Where testing allows us to validate the functionality and performance of the APIs, ensuring they deliver the correct data to our application. By conducting some comprehensive testing, we can identify and rectify any potential issues, such as incorrect data retrieval or inconsistent responses. This helps to maintain a seamless UX and guarantees the integrity of the information presented to our app users.

Chapter Seven: Project's flow

In the final chapter, we will discuss the implementation of our Graduation Project. We will provide a documented photographic tour of our web and mobile application and how it functions. We will also examine some of the challenges we faced during the implementation process and how we overcame them. Additionally, we will evaluate the success of our

Graduation Project, its impact on the transportation system in Cairo and Giza, and the potential for further development and expansion of its functionalities.

Lastly, Our Graduation Project, Public Transportation Guidance, is a mobile & web application that provides a comprehensive solution to the transportation challenges faced by the people living in Cairo and Giza. This book provides a detailed guide to the development and implementation of the application, including the technologies, languages, and tools used to build it. We hope that this book will serve as a valuable resource for anyone interested in developing similar public transportation guidance systems and contribute to the development of sustainable and efficient transportation systems in urban areas.

Chapter 1:
Previous Approaches

Previous Approaches

The main idea of the project, which is displaying the paths of every transportations combination possible between two locations, isn't new, since there were some projects, in which have preceded us with offering similar service. However, we experimented them from the user's perspective and observed what they lack for, their flaws, and some of the problems that irritated us as users then, we aimed for solving these problems.

One of these applications and one of the most popular ones is **Google Maps Transportation**, other applications like: **Wasalny** and **3lagnb**.

One of the features that we noticed which were lacked in other applications was covering the entire transportation network in Cairo. It can be impractical for many users to not find their needs and the locations that they are looking for, hence less popularity for the application and users won't find it very beneficial.

And when it comes to Google Maps Transportation, it does not cover the entire microbuses network, which is considered one of the most famous public transportations in Egypt since it is the most affordable, convenient, and time saving one.

The following table will show a comparison between our application, google maps transportation, and other applications:

Features	Tawsila	Google Maps Transportation	Other Apps
Current Location	✓	✓	✗
Tracking	✓	✓	✗
Order by cost	✓	✗	✗
Order by distance	✓	✗	✗
Order by time	✓	✓	✓
Add new route	✓	NN	✗
Covering all regions	✓	✓	✗
Dynamic database	✓	✓	✗
Blind Mode	✓	✓	✗
Adding Friends	✓		✗
Sharing Location	✓	✓	✗
Reviewing paths	✓	✓	✗
Complete Transportation database	FW	✓	✗

Table [1]: Applications comparison

Chapter 2:
Graph DataBase

1. Reasons for Using No SQL Database

a. Introduction:

NoSQL (Not Only SQL) databases are becoming increasingly popular because of their flexible and scalable nature. Unlike traditional SQL databases that use a rigid and structured approach to storing data, NoSQL databases offer a more dynamic model, designed to handle unstructured and semi-structured data. This makes them well-suited for modern applications that generate massive amounts of data, such as social media ,IOT and geospatial data, which require flexibility and scalability.

As the demand for public transportation increases, so does the need for efficient and reliable database systems to manage the vast amounts of data generated daily. While traditional SQL databases have been the go-to solution for many years, the emergence of NoSQL databases offers a new and innovative approach to managing transportation data.

NoSQL databases are also highly resilient and provide built-in fault tolerance, making them ideal for transportation applications that require high availability and reliability. This is critical in ensuring that the transportation system runs smoothly and without any disruptions.

Additionally, many NoSQL databases are open-source, which makes them easily customizable and cost-effective. Developers can quickly modify the databases to work with evolving data models and application requirements.

Lastly, NoSQL databases are capable of handling both structured and unstructured data types. This allows them to support popular data formats such as JSON, XML, and BSON, which are commonly used in web applications.

In summary, NoSQL databases are highly useful in managing transportation data due to their ability to handle unstructured data, scalability, resilience, and fault tolerance. By leveraging NoSQL technology, transportation authorities can provide faster, more reliable services to their passengers, resulting in improved efficiency and a better overall transportation experience.

SQL and NoSQL are different types of database management systems with their unique features and limitations. Some of the disadvantages of using SQL instead of NoSQL include the following:

b. Reasons for preference in the form of points:

The choice between SQL and NoSQL databases depends on specific use cases and requirements. Both have their advantages and disadvantages, and it's essential to evaluate them before making a decision. Here are some potential disadvantages of using SQL databases compared to NoSQL databases:

1. **Limited scalability:** SQL database systems are not well-suited for handling large volumes of unstructured data that require horizontal scaling. This limits their ability to handle data growth over time.

2. **Structured data only:** SQL databases are designed to work with structured data, which makes them less flexible than NoSQL databases that can work with both structured and unstructured data like geospatial and transportation data.

3. **Limited Performance:** SQL databases can suffer from slower performance when dealing with large data sets or complex queries. They might not be the best choice for transportation data.
4. **Not Ideal for Distributed Applications:** SQL databases may not be ideal for distributed applications like APIs because they may require strict consistency, which can lead to performance issues when deployed across multiple nodes.

Overall, SQL databases are not necessarily inferior to NoSQL databases, and the choice between them depends on the specific requirements of your application. SQL databases remain the preferred choice for applications that require consistency, transactional support, and reliable data storage. However, NoSQL databases excel when dealing with unstructured data, high scalability, and high-performance computing.

On the other hand, NoSQL databases are designed to be flexible, scalable, and easily adaptable, making them well-suited for handling large and complex data sets. However, it is important to note that the choice of a database management system depends on the specific requirements of your application.
So we used it because it is the best choice for our application.
If you are looking to handle massive amounts of unstructured data such as geospatial and transportation data, then a NoSQL database like GraphDB may be a better choice.

2. Reasons for Using Graph Database

1. Introduction:

Graph databases are a type of NoSQL database that are specifically designed to manage and query highly connected and complex data. Unlike other NoSQL databases such as document-

oriented databases or key-value stores, graph databases focus on relationships between data elements rather than individual data points.

We have used graph database because it is the best among all other types of NoSQL databases for geospatial and transportation data which is the basis for the work of our application.

2. Reasons for preference in the form of points:

Here are some potential advantages of using GraphDB over a general NoSQL database for an application that involves public transportation guidance with geospatial and transportation data:

1. **Efficient Geospatial Querying:** GraphDB has built-in geospatial querying capabilities, which makes it a great choice for applications that require geospatial data. With GraphDB, you can easily query for locations and routes that are close to a particular point of interest or within a certain area, which can be useful for public transportation guidance.

2. **Highly Connected Data:** GraphDB is specifically designed for managing and querying highly connected and complex data, which makes it a good choice for applications that involve transportation data. For example, GraphDB can easily represent and query relationships between transportation routes, stops, and schedules.

3. **Flexible Data Model:** GraphDB's flexible data model allows for changes in the data structure over time, which can be useful for managing transportation data that may change frequently due to updates or changes in schedules.

4. **ACID Transactions:** GraphDB offers ACID-compliant transactions, which ensures data consistency and integrity even in complex graph structures. This can be particularly important for applications that require accurate and up-to-date transportation data.

5. **High Scalability:** GraphDB can be horizontally scaled across multiple nodes, which allows for high scalability and performance. This is important for applications that may experience high traffic or demand, such as a public transportation guidance application.

Overall, the advantages of using GraphDB over a general NoSQL database for a public transportation guidance application with geospatial and transportation data include efficient geospatial querying, the ability to manage highly connected and complex transportation data, a flexible data model, ACID-compliant transactions, and high scalability.

3.How does the graphDB execute Cypher queries?

Cypher is a declarative graph query language that is used to query and manipulate data in GraphDB. Here is an overview of how Cypher query execution works in GraphDB:
Parsing: When a Cypher query is submitted to GraphDB, the first step is parsing. During this step, the query is broken down into its individual elements and checked for syntax and semantic errors.
Planning: After parsing, the query planner determines the optimal query plan to execute the query. This involves selecting the best index or combination of indexes to use for the query, as well as determining the best order for performing any joins or other operations.
Optimization: The query optimizer then takes the query plan generated in the planning step and optimizes it for performance. This may involve reordering operations or modifying the plan to take advantage of specific features of the underlying data.

Execution: Once the query plan is optimized, the actual query execution begins. GraphDB uses a distributed query engine to execute queries in parallel across multiple nodes, which allows for high scalability and performance. During execution, the query engine retrieves the necessary data from the underlying data store and applies any necessary transformations or calculations.
Result Formatting: Finally, once the query execution is complete, the results are formatted and returned to the user in the desired format (e.g. JSON or CSV).
Overall, the Cypher query execution process in GraphDB involves parsing and checking the syntax and semantics of the query, planning and optimizing the query for performance, executing the query in parallel across multiple nodes, and formatting and returning the results to the user.

4.Execution time in graphDB

The execution time of a query in GraphDB can vary depending on a number of factors, such as the complexity of the query, the size of the graph being queried, and the underlying hardware and network infrastructure.
GraphDB uses a distributed query engine to execute queries in parallel across multiple nodes, which allows for high scalability and performance. However, the exact execution time will depend on the specific query being executed and the size and complexity of the graph being queried.
In general, GraphDB is designed to provide fast and efficient query performance, particularly for complex graph queries involving multiple nodes and relationships. However, as with any database system, query performance can be affected by a number of factors, such as the amount of data being queried, the specific indexing and query optimization strategies used, and the underlying hardware and network infrastructure.
To optimize query performance in GraphDB, it is important to carefully design your data model, use appropriate indexing strategies, and carefully tune your query execution parameters. Additionally, leveraging the built-in caching mechanisms and employing techniques such as

query batching and result pagination can help to further optimize query performance in GraphDB.

Graph Design

The graph database for public transportation in Cairo is designed using nodes with label "LOCATION". The "LOCATION" label represents points of interest such as street names or areas. These nodes have three attributes: "name", "longitude", and "latitude". The "name" attribute stores the name of the location, while the "longitude" and "latitude" attributes store the corresponding geographic coordinates.

```
CREATE (n1:LOCATION {name: 'الدقي',latitude: 30.03842817178049,longitude: 31.21225236295685}),
       (n2:LOCATION {name: 'السيدة عائشة',latitude: 30.02735999999999,longitude: 31.2559448}),
       (n3:LOCATION {name: 'السيدة زينب',latitude: 30.0306686,longitude: 31.232628}),
       (n4:LOCATION {name: 'جامعة القاهرة',latitude: 30.0273542,longitude: 31.20912500000001}),
       (n5:LOCATION {name: 'ميدان الجيزة',latitude: 30.0154402,longitude: 31.2118712}),
```

Figure [1]: Creating nodes

The graph is modeled using relationships or edges between "LOCATION" nodes with label "LINE" which represents transportation lines, and has four attributes: "name", "type", "cost", and "distance". The "name" attribute stores the name of the transportation line, while the "type" attribute stores the type of transportation (e.g., bus, metro, tram). The "cost" attribute stores the

cost of using the transportation line, and the "distance" attribute stores the distance covered by the transportation line.

```
CREATE (n1)-[r:LINE {name: '17ط', type: 'bus', cost: 5, distance: 0.5}]→(n2),
       (n2)-[r:LINE {name: 'السيدة زينب', type: 'microbus', cost: 2, distance: 2.5}]→(n3),
       (n3)-[r:LINE {name: '8/', type: 'bus', cost: 4, distance: 4}]→(n4),
       (n4)-[r:LINE {name: '17ط', type: 'bus', cost: 1, distance: 2.9}]→(n5),
```

Figure [2]: Creating relationships

The graph database is designed in a way that allows efficient traversal and querying of the connections between locations and transportation lines, making it suitable for finding available paths between two points in Cairo.

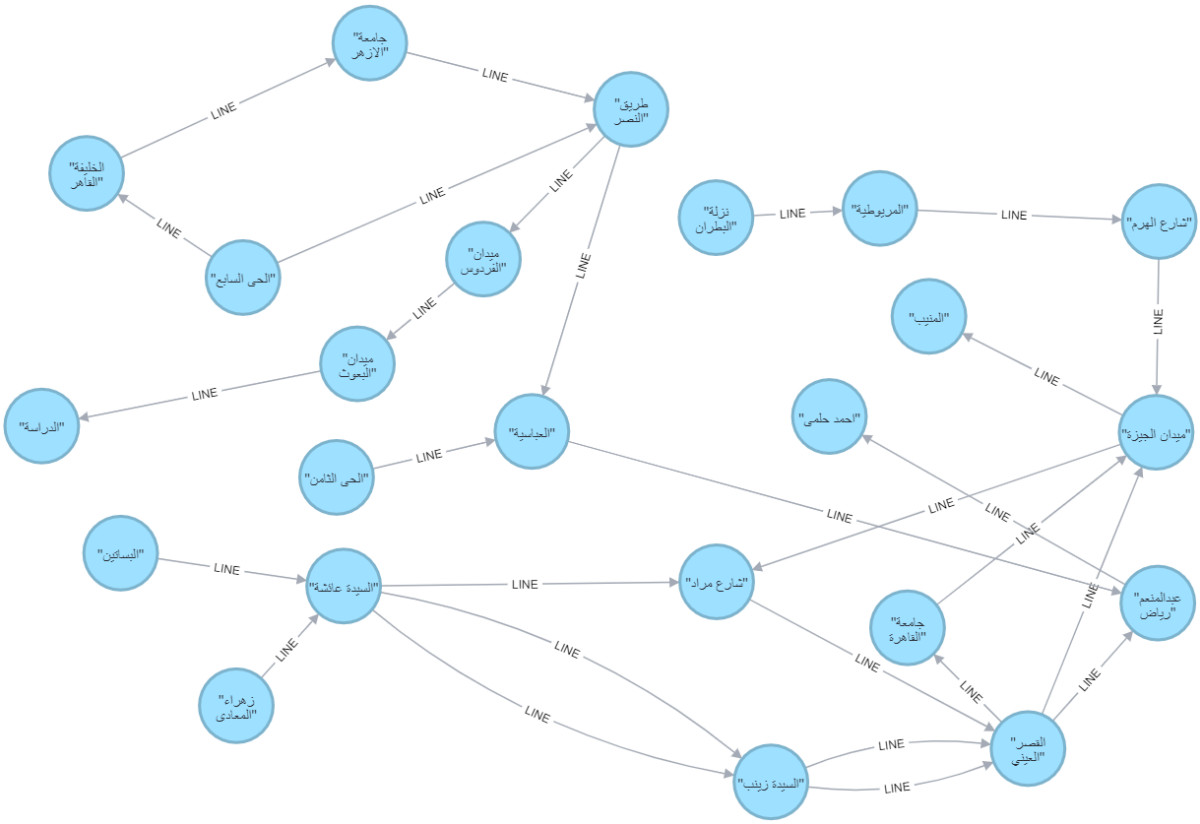


Figure [3]: Created graph database

Query Implementation

The implemented query takes the longitude and latitude of the user's destination and location as input parameters, and retrieves available paths between these two points. If the input points do not exist in the database, the query finds the nearest points to them that are present in the database.

```
MATCH (l:LOCATION) WHERE point.distance(point({latitude:30.02735999999999, longitude: 31.2559448}),
    point({latitude: l.latitude, longitude: l.longitude})) ⩽ 2000
    RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.02735999999999, longitude:  31.2559448}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Location" AS input
    ORDER BY Distance
    UNION ALL
    MATCH (l:LOCATION)
    WHERE point.distance(point({latitude: 30.0154402, longitude: 31.2118712}), point({latitude: l.latitude, longitude:
l.longitude})) ⩽ 2000
    RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.0154402, longitude:31.2118712}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Destination" AS input
    ORDER BY Distance
```

Figure [4]: Order by distance

```
MATCH (start:LOCATION {latitude:30.02735999999999, longitude: 31.2559448}),
    (end:LOCATION {latitude: 30.0154402, longitude: 31.2118712})
    WITH start, end
    MATCH path = (start)-[:LINE*]→(end)
    WHERE ALL(n IN nodes(path) WHERE size([m IN nodes(path) WHERE m = n]) = 1)
    WITH count(path) AS numberOfAvailablePaths, collect(path) AS paths
    UNWIND paths AS path WITH numberOfAvailablePaths, path,
    reduce(cost = 0, r IN relationships(path) | cost + r.cost) AS totalCost,
    reduce(distance = 0, r IN relationships(path) | distance + r.distance) AS totalDistance,
    size(nodes(path)) AS numberOfStops
    RETURN path, numberOfAvailablePaths, totalCost, totalDistance, numberOfStops
    ORDER BY totalDistance, totalCost
```

Figure [5]: Order by cost

Query Explanation

The query is written in Cypher, which is a query language used for querying Neo4j, a popular graph database management system. The query starts by defining the input parameters for the user's location and destination using the "WITH" clause. The longitude and latitude values are passed as parameters, and labels ("Location" and "Destination") are assigned to them for easier reference in the query.

Next, the query uses the "MATCH" clause to retrieve "LOCATION" nodes that are within a certain range area (in this case, 2000 meters) from the user's location. The "point.distance" function is used to calculate the distance between two points on the Earth's surface using their latitude and longitude coordinates. The retrieved "LOCATION" nodes are returned with their name, latitude, longitude, and the calculated distance as attributes, along with the input label ("Location") or ("Destination").

The "UNION ALL" clause is then used to combine the results of the first part of the query with the results of the second part, which retrieves "LOCATION" nodes that are within a certain distance from the user's destination.

The query uses the "ORDER BY" clause to sort the results based on the calculated distance between the user's destination and the retrieved "LOCATION" nodes, in ascending order. This allows the query to find the nearest points to the input points that are present in the database.

The query takes the longitude and latitude of the location and destination points and calculates the distance between each and the graph nodes that are located in a range area of 2000 m^2. If the entered location or destination point is stored in the database, the longitude and latitude will be matched, so it gives 0.0 as a distance (which indicates that it exists).

```
MATCH (l:LOCATION)WHERE point.distance(point({latitude:30.02735999999999, longitude: 31.2559448}),
        point({latitude: l.latitude, longitude: l.longitude})) ⩽ 2000
RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.02735999999999, longitude:31.2559448}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Location" AS input
        ORDER BY Distance UNION ALL
        MATCH (l:LOCATION)
        WHERE point.distance(point({latitude: 30.0154402, longitude: 31.2118712}), point({latitude: l.latitude,
longitude: l.longitude})) ⩽ 2000
        RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.0154402, longitude:31.2118712}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Destination" AS input
        ORDER BY Distance
```

Table RAW

l.name	l.latitude	l.longitude	Distance	input
"السيده عائشة"	30.027359999999	31.2559448	0.0	"Location"
"البساتين"	30.0144194	31.2615592	1538.8302808121	"Location"
"ميدان الجيزة"	30.0154402	31.2118712	0.0	"Destination"
"شارع مراد"	30.0154495	31.212008	13.226804536679	"Destination"

Figure [6]: Matched coordinates

If the point doesn't exist in the database, it will replace it with the nearest point that exists and do the search with it.

```
MATCH (l:LOCATION)WHERE point.distance(point({latitude:30.02735999999999, longitude: 31.2559448}),
        point({latitude: l.latitude, longitude: l.longitude})) ⩽ 2000
RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.02735999999999, longitude:31.2559448}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Location" AS input
        ORDER BY Distance UNION ALL
        MATCH (l:LOCATION)
        WHERE point.distance(point({latitude: 30.0154402, longitude: 31.2118712}), point({latitude: l.latitude,
longitude: l.longitude})) ⩽ 2000
        RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.0154402, longitude:31.2118712}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Destination" AS input
        ORDER BY Distance
```

Table RAW

l.name	l.latitude	l.longitude	Distance	input
"السيده عائشة"	30.027359999999	31.2559448	0.0	"Location"
"البساتين"	30.0144194	31.2615592	1538.8302808121	"Location"
"ميدان الجيزة"	30.0154402	31.2118712	0.0	"Destination"
"شارع مراد"	30.0154495	31.212008	13.226804536679	"Destination"

Figure [7]: Nearest coordinates

To differentiate between the nearest points of source and the nearest points of destination, it returns a label as a "Location" or " Destination", respectively.

```
MATCH (l:LOCATION)WHERE point.distance(point({latitude:30.02735999999999, longitude: 31.2559448}),
        point({latitude: l.latitude, longitude: l.longitude})) ⩽ 2000
RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.02735999999999, longitude:31.2559448}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Location" AS input
        ORDER BY Distance UNION ALL
        MATCH (l:LOCATION)
        WHERE point.distance(point({latitude: 30.0154402, longitude: 31.2118712}), point({latitude: l.latitude,
longitude: l.longitude})) ⩽ 2000
        RETURN l.name, l.latitude, l.longitude, point.distance(point({latitude: 30.0154402, longitude:31.2118712}),
point({latitude: l.latitude, longitude: l.longitude})) AS Distance, "Destination" AS input
        ORDER BY Distance
```

Table RAW

l.name	l.latitude	l.longitude	Distance	input
"السيده عائشة"	30.027359999999	31.2559448	0.0	"Location"
"البساتين"	30.0144194	31.2615592	1538.8302808121	"Location"
"ميدان الجيزة"	30.0154402	31.2118712	0.0	"Destination"
"شارع مراد"	30.0154495	31.212008	13.226804536679	"Destination"

Figure [8]: Nearest location and destination

After finding the new searching points, the second part of the query takes their longitude and latitude to search for paths between them.

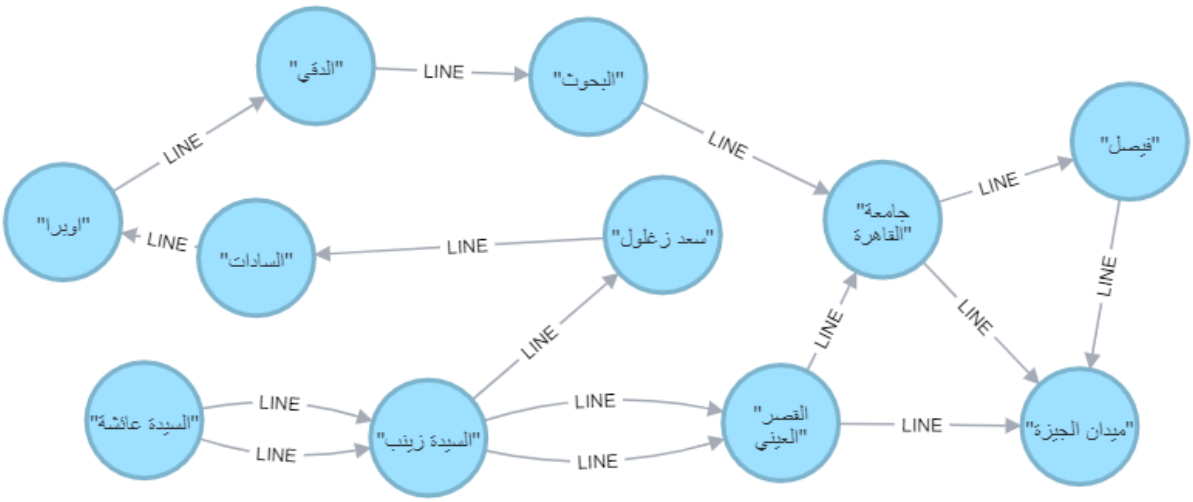
The query begins by matching two location nodes, n1 and n2, with the specified latitude and longitude values. It then finds all paths between these two nodes using the "LINE" relationship type and assigns them to the variable "p".

Next, the query calculates the number of available paths between the two nodes using the "count" function and assigns the result to the variable "numberOfAvailablePaths".

The query then matches the same two location nodes again and finds all paths between them using the "LINE" relationship type and assigns them to the variable "path".

After that, the query matches all location nodes within a distance of 100 units (presumably meters) from a specified point, which is represented as a latitude/longitude pair, and calculates the distance between each matched location node and the specified point using the "point.distance" function.

Finally, the query returns several values for each matched location node, including the path between the two original nodes, the total number of available paths, the total cost of the path (which is calculated by summing the "cost" property of each relationship in the path), the total distance of the path (which is calculated by summing the "distance" property of each relationship in the path), the number of stops in the path, the name and location of the location node, and the distance between the location node and the specified point. The results are sorted first by total distance, then by total cost, and finally by distance from the specified point.



path	numberOfAv...	totalCost	totalDistance	numberOfStops
(:LOCATION {latitude: 30.02735999999999, name: "السيدة عائشة", longitude: 31.2559448})-[:LINE {cost: 1, distance: 1.2, name: "17ط", type: "bus"}]-> (:LOCATION {latitude: 30.0306686, name: "السيدة زينب", longitude: 31.232628})-[:LINE {cost: 1, distance: 2.4, name: "17ط", type: "bus"}]-> (:LOCATION {latitude: 30.0211564, name: "القصر العيني", longitude:	16	3.5	6.6	4

Figure [9]: Ordering cost and distance

Overall, the graph database and the implemented query have the potential to provide valuable insights and support decision-making in public transportation planning and management in Cairo. They can be further enhanced with additional features, such as real-time data integration, user preferences, and optimized routing algorithms, to improve the accuracy and efficiency of the pathfinding process.

Chapter 3: Technologies, Languages, and Tools

Technologies, Languages, and Tools

This chapter is all about the technologies used in this project, we will be discussing what we used in details, we will also cover the process of choosing one technology over the others.

1. Neo4j

Neo4j is the world's leading Graph Database. It is a high performance graph store with all the features expected of a mature and robust database, like a friendly query language and ACID transactions. The programmer works with a flexible network structure of nodes and relationships rather than static tables — yet enjoys all the benefits of enterprise-quality database. For many applications, Neo4j offers orders of magnitude performance benefits compared to relational DBs.

A graph database, instead of having rows and columns has nodes edges and properties. It is more suitable for certain big data and analytic applications than row and column databases or free-form JSON document databases for many use cases.

Our project depends heavily on complex relationships since it is mainly focused on public transportation, in chapter ( ) we will expand on Neo4j showing how it works and how did we manage to create the database that is perfectly suitable for our project.

2. Node.js

Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

2.1. Why use Node.js ?

Node.js shines in real-time web applications employing push technology over WebSocket. After over 20 years of stateless-web based on the stateless request-response paradigm, we finally have web applications with real-time, two-way connections, where both the client and server can initiate communication, allowing them to exchange data more freely. This is in stark contrast to the typical web response paradigm, where the client always initiates communication.
One might argue that we've had this technology for years in the form of Flash and Java Applets. In reality, however, those were just sandboxed environments that used the web as a transport protocol to be delivered to the client. Plus, Flash and Java Applets were run in isolation and often operated over nonstandard ports, which may have required extra permissions.

2.2. How does Node.js work ?

Node really shines in building fast, scalable network applications. This is due to its capability of handling a huge number of simultaneous connections with high throughput.
Node.js uses non-blocking, event-driven I/O to remain lightweight and efficient in the face of data-intensive real-time applications that run across distributed devices.
Node.js is a platform that fills a particular need, and understanding this is absolutely essential. For example, you wouldn't use Node.js to perform CPU-intensive operations. Nearly all of Node's advantages are annulled if it's used for heavy computation.

How Node.js works under the hood is interesting. Compared to traditional web-serving techniques where each connection (request) spawns a new thread (taking up system RAM and eventually maxing out at the amount of RAM available), Node.js operates on a single thread, using nonblocking I/O calls. This allows Node to support tens of thousands of concurrent connections held in the event loop.

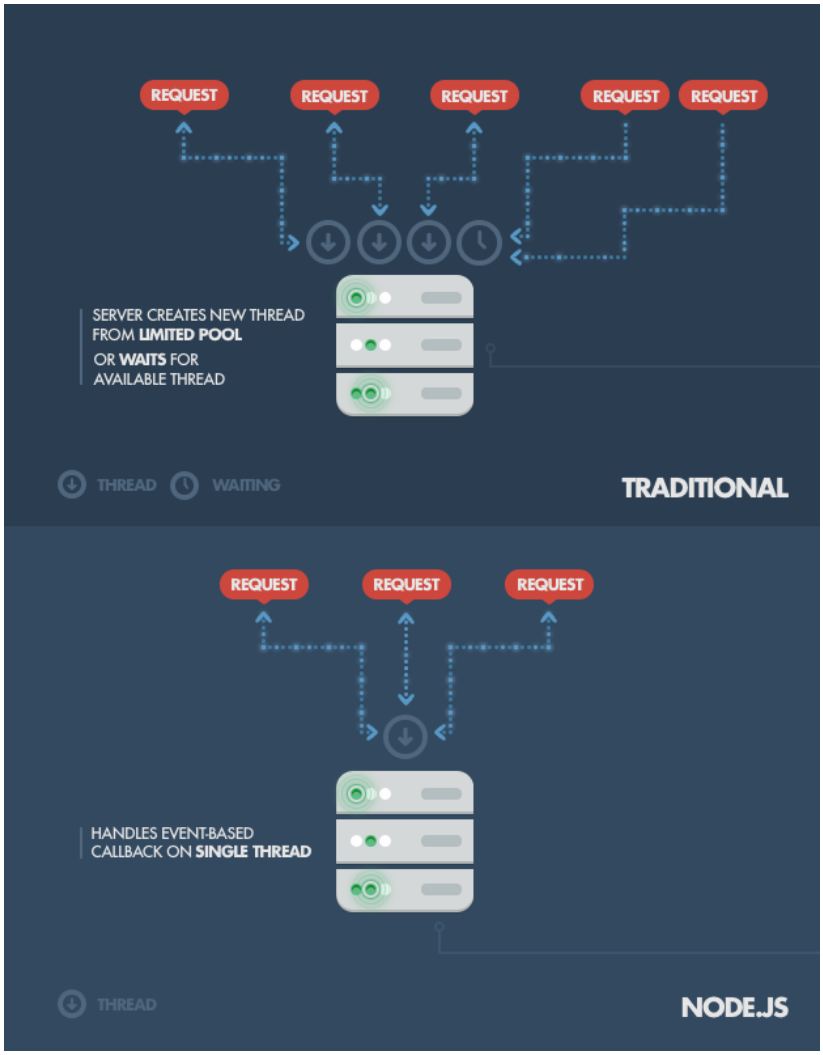


Figure [10]: Node.js

2.3. npm: The node package manager

Built-in support for package management using npm is included in every Node.js installation. The idea behind npm modules is similar to that of Ruby Gems: It is a set of publicly available, reusable components, easily installed via an online repository, with version and dependency management.

npm Inc. shares a list of packaged modules that are also accessible via its npm CLI tool. The module ecosystem is open to all to publish their own module, which would be added to the npm repository. Some useful npm modules include:

express	A Sinatra-inspired web development framework for Node.js, and the de facto standard for the majority of Node.js applications.
hapi	A modular and simple-to-use configuration-centric framework for building web and services applications.
connect	An extensible HTTP server framework for Node.js, providing a collection of high performance plugins known as middleware; serves as a base foundation for Express.
socket.io and sockjs	A server-side component of two common WebSocket components.
pug (formerly Jade)	A templating engine inspired by HAML, a default in Express.js.
mongodb and mongojs	MongoDB wrappers to provide the API for MongoDB object databases in Node.js.
redis	The Redis client library.
lodash, underscore, lazy.js	The JavaScript utility belt. Underscore initiated the game but got overthrown by one of its two counterparts, mainly due to lazy.js' better performance and modular implementation.
forever	A utility for ensuring that a given node script runs continuously; keeps your Node.js process up in production in the face of any unexpected failures.
bluebird	A full-featured Promises/A+ implementation with exceptionally good performance.

moment.js	A JavaScript date library for parsing, validating, manipulating, and formatting dates.

Table [2]: npm modules

2.4. Where to use Node.js ?

Server-side Web Applications

With Node.js with Express.js, you can create classic web applications on the server side. While possible, this request-response paradigm in which Node.js would carry rendered HTML is not an ideal use case. There are arguments to be made for and against this approach. Here are some facts to consider:

Pros:

- You can significantly ease the development of an application that does not require CPU-intensive computation, by using Javascript to build it top to bottom, even down to the database level—if you use JSON storage Object DB (e.g., MongoDB).
- Crawlers receive a fully rendered HTML response, which is far more SEO friendly than, say, a Single Page Application or a WebSocket app that runs on top of Node.js.

Cons:

- Any CPU-intensive computation will block Node.js responsiveness, so a threaded platform is a better approach. Alternatively, you could try scaling out the computation.
- Using Node.js with a relational database can be painful. If you're trying to perform relational operations, consider going with an environment such as Rails, Django, or ASP.Net MVC.

An alternative to CPU-intensive computations is to create a highly scalable MQ-backed environment with back-end processing to keep Node as a front-facing "clerk" to handle client requests asynchronously.

2.5. Where not to use Node js?

a. Server-side Web Application With a Relational Database Application

Ruby on Rails was once the clear choice as a tool to access relational databases like PostgreSQL, MySQL, and Microsoft SQL Server. This was because relational DB tools for Node.js were still in their early stages while, in contrast, Rails automatically provided data access setup right out of the box, together with DB schema migrations support tools, and other Gems (pun intended). Rails and its peer frameworks have mature and proven Active Record or Data Mapper data access layer implementations.

But things have changed. Sequelize, TypeORM, and Bookshelf have come a long way toward becoming mature ORM solutions. It might also be worth checking out Join Monster if you're looking to generate SQL from GraphQL queries.

b. Heavy Server-side Computation and/or Processing

Node.js is not the best platform to handle heavy computation. No, you definitely don't want to build a Fibonacci computation server in Node.js.
In general, any CPU-intensive operation annuls all the throughput benefits Node offers with its event-driven, nonblocking I/O model. This is because incoming requests are blocked while the thread is occupied with your number-crunching—assuming you're trying to run computations in the same Node instance used to respond to requests. Since Node.js is single-threaded and uses only a single CPU core, it would require considerable effort to develop a cluster module in order to deliver concurrency on a

multicore server. Alternatively, you can run several Node.js server instances pretty easily behind a reverse proxy via nginx.

With clustering, you should still offload all heavy computation to background processes. Ensure that you use an appropriate environment for the background processes, and that they communicate via a message queue server like RabbitMQ. While you may run background processes on the main server, this approach may not scale well once the load increases. You may distribute background processing services to separate worker servers without the need to configure the loads of front-facing web servers.
With Node.js—as opposed to most other platforms—you enjoy that high reqs/sec throughput we talked about, as each request is a small task that Node handles quickly and efficiently.

3. Firebase

Firebase is a set of backend cloud computing services and application development platforms provided by Google. It hosts databases, services, authentication, and integration for a variety of applications,
including Android, iOS, JavaScript, Node.js, Java, Unity, PHP, and C++.

In this project we chose to work with firebase to implement the part regarding “Add New Route” form which forces users to sign up first before any further steps regarding adding a new route of transportation.

3.1. Why firebase ?

It allows real-time database connection, which means multiple users can see the changes in the data when the data gets created or edited. Data transmission is handled with web sockets so you don't have to send requests to get new data, you only need to subscribe once.

Firebase quickly syncs data via Android, iOS, and JavaScript SDKs, allowing for expressive queries that scale with the size of the result set.

3.2. Firebase vs. Vanilla SQL

We chose firebase over vanilla SQL for the following reasons:

1) Firebase has more services
firebase offers much more options like verification and different signup / login methods (Google, Facebook, ..)
2) More Reliable
firebase auth. is already designed by google to put scalability& security into consideration.
3) Time & effort
firebase helps to shorten the time and save the effort needed to create a similar end-product
4) New tool under the belt
Many companies depend on firebase to create back-end projects, and it’s always good to learn a new tool to expand our knowledge.

3.3. Firebase Authentication

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

To sign a user into our app, we first get authentication credentials from the user. These credentials can be the user's email address, password, and other details. Then, you pass these credentials to the Firebase Authentication SDK. Firebase backend services will then verify those credentials and return a response to the client.
After a successful sign in, we can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products. We can also use the provided authentication token to verify the identity of users in your own backend services.

3.4. Firestore

Firestore is a NoSQL document database provided by Google as a part of the Firebase suite. It offers a flexible and scalable solution for storing, syncing, and querying data for web, mobile, and server applications. Firestore enables real-time data synchronization and offers seamless integration with other Firebase features like authentication, hosting, and cloud functions.
we will discuss the utilization of Firestore in the project, specifically focusing on the three collections created.

a. Nodes Collection

The Nodes collection is used to add new routes by creating two new nodes in the Neo4j database along with the relationship between them. It contains the following fields:

- **Cost:** Represents the cost associated with the route.
- **Distance**: Specifies the distance between the nodes.
- **FromLatitude, FromLongitude:** Denotes the latitude and longitude of the starting node.
- **Node Name:** The name of the node.
- **ToLatitude, ToLongitude:** Represents the latitude and longitude of the ending node.
- **Type:** Defines the type of route.
- **id:** email: A unique identifier/email that associates the node with the user.

By utilizing Firestore for this collection, the project benefits from its real-time synchronization capabilities, ensuring data consistency and seamless collaboration.

b. Reviews Collection

The Reviews collection is used to gather user feedback and reviews for the suggested path. It consists of the following fields:

- **BadPathDislikes:** Keeps track of the number of dislikes received for a path that was deemed as a bad choice.
- **Likes**: Tracks the number of likes received for the suggested path.
- **UnFoundPathDislikes:** Records the number of dislikes received when a path is not found.

By leveraging Firestore for this collection, the project enables users to provide feedback on the suggested paths, allowing for continuous improvement of the routing system.

c. Users Collection

The Users collection is used to store authenticated user information.
It includes the following fields:

- **FirstName:** Stores the first name of the user.
- **LastName:** Carries the last name of the user.
- **Phone:** Records the phone number associated with the user.

Firestore provides a secure and scalable solution to store and manage user information within the project. Additionally, it offers easy integration with Firebase Authentication for seamless user authentication and access control.
Advantages and Benefits of Firestore

Firestore offers several advantages and benefits for the project:

- Real-time synchronization: Firestore enables real-time updates and synchronization of data across clients, ensuring consistent and up-to-date information for all users.
- Scalability: With Firestore, the project can handle high volumes of data seamlessly, making it suitable for applications with varying data requirements and usage patterns.
- Security: Firestore implements robust security rules, ensuring that only authorized users can access and modify the data.
- Querying and indexing: Firestore provides powerful querying capabilities, allowing efficient retrieval and filtering of data based on specific criteria.
- Easy integration: Firestore integrates smoothly with other Firebase services, simplifying the development process and offering additional features like authentication, hosting, and cloud functions.

4. Google maps API

The Google Maps Platform is a set of APIs and SDKs that allows developers to embed Google Maps into mobile apps and web pages, or to retrieve data from Google Maps. We used google maps api in some parts of our project in order to expand our app capabilities.

4.1. Places API & SDKs

To integrate Google's Place details, search, and autocomplete into our app.

4.2. Geocoding

To convert coordinates into addresses and addresses into coordinates.

4.3. Geolocation

Get an approximate device location using nearby cell towers and WiFi nodes.

4.4. Maps SDKs

To bring the real world to our users with dynamic maps for the web and mobile.

5. AnnYang

Annyang is a lightweight JavaScript library that provides voice recognition and voice command capabilities. It allows the website to listen for specific voice commands and trigger corresponding actions in response. AnnYang simplifies the implementation of speech recognition by providing a simple API for defining voice commands and callbacks.

By integrating AnnYang into the project's blind mode feature, users are able to issue commands using their voice, empowering them to navigate and interact with the website without relying solely on visual cues.

6. Web Speech API

The Web Speech API is a standard JavaScript API that provides speech synthesis (text-to-speech) and speech recognition (speech-to-text) functionalities. It is supported by modern web browsers and allows developers to integrate speech capabilities directly into web applications.

In the blind mode feature, the Web Speech API is used for speech recognition, converting spoken commands into text that can be interpreted and processed by the website. This enables users to issue voice commands to interact with various elements and features of the website.

By combining AnnYang with the Web Speech API, the project provides a comprehensive voice-controlled interface for blind users, allowing them to navigate through the website, trigger actions, and interact with content effectively using natural language and spoken commands.

Advantages of AnnYang and Web Speech API

The integration of AnnYang and the Web Speech API brings several advantages to the blind mode feature:

- **Accessibility:** The blind mode with voice commands makes the website more accessible to individuals with visual impairments, providing an alternative means of interaction beyond traditional visual interfaces.

- **Inclusivity:** By incorporating speech recognition and voice commands, the website becomes more inclusive, catering to a wider range of users, including those with physical or mobility limitations that may find it difficult to operate traditional input devices.
- **Ease of Use:** Voice commands provide a natural and intuitive way of interaction, reducing the reliance on manual input and potentially making the website easier to navigate and use.
- **Customization:** AnnYang allows for the definition of custom voice commands, enabling developers to tailor the experience to the specific needs and functionalities of the website.
- **Integration with Existing Technologies:** By using the standard Web Speech API, the project can smoothly integrate the voice control functionality into the website without relying on third-party services or plugins.

7. Android.speech.RecognizerIntent

In the mobile application, this class is utilized to implement the blind mode feature, providing speech recognition capabilities to allow users to interact with the application using their voice. This Android framework component enables the application to convert spoken words into text, which can then be processed to trigger specific functionalities.

The android.speech.RecognizerIntent is a part of the Android Speech Recognition API that provides a convenient way to integrate speech recognition functionality into Android applications. By using this class and related methods, the application can initiate speech recognition and receive transcriptions of the user's spoken words.

With the blind mode feature, it is utilized to capture the user's voice input and convert it into text. This allows the user to issue commands or provide input by speaking

instead of relying solely on manual input methods such as typing or tapping on the screen.

Advantages of Android.speech.RecognizerIntent in Blind Mode

The integration of android.speech.RecognizerIntent into the blind mode feature brings several advantages:

- **Accessibility:** By leveraging speech recognition capabilities, blind or visually impaired users can interact with the mobile application using their voice, improving accessibility and enabling them to perform tasks more efficiently.
- **Natural Interaction:** Speech recognition provides a more natural and intuitive means of interacting with the application, allowing users to speak their commands or input rather than relying on manual input methods.

- Hands-Free Operation: With speech recognition, users can operate the application without the need for precise touch interactions, making it more convenient and suitable for scenarios where manual dexterity may be limited.

- **Voice Commands:** The application can define specific voice commands to trigger different actions or functionalities within the blind mode feature, offering a more streamlined and efficient user experience.

Chapter 4: Functional & non-Functional requirements

System Functional requirements

1) User Account Creation:

Users can easily create an account by providing the following information:

- First Name
- Last Name
- Email
- Password
- Phone Number

2) User Login:

To access additional features, users must log in to their accounts.

3) Adding a New Route:

If five or more users have added the same route, it will be automatically included in the system.

4) Logging Out:

Users can securely log out of the system when they're done.

5) Adding Friends:

Users can enhance their experience by adding friends. They simply need to provide their friend's email address to share their paths.

6) Removing a Friend:

Users have the option to delete friends from their friend list.

7) Sending Follow Requests:

To track another user's path, one must send a follow request first.

8) Sharing a Path:

Users can share their current location coordinates and place name with another user.

9) Selecting Preferred Order Parameter:

To organize search results, users can choose between distance, cost, or time as their preferred parameter.

10) Searching for Paths between Two Places:

The system will search for the shortest paths between two locations and present them in order based on the selected parameter.

11) Displaying Path Details:

Users can view detailed information about their selected path, including the names of the transportation modes they need to take.

12) Tracking Current Location:

Users have the ability to track their current location on the map until they reach their destination.

13) Liking a Path:

Users can express their satisfaction with a path by clicking on the like button.

14) Disliking a Path:

If a path is unsatisfactory, users can provide feedback by clicking on the dislike button. They can also specify the transportation mode and reasons for their dissatisfaction, including cases where the path doesn't exist or is of poor quality.

15) Activating Blind Mode:

Users can activate the blind mode by clicking on the blind mode button or responding to voice-based automated questions.

System Non-Functional Requirements

1) Speed

The application responds to commands in a fast manner. For example, if you type a word into the search engine, you can receive search results in a fair amount of time. The system is also able to manage an increasing workload as you use different applications at the same time. For example, a user might take pictures with a photo application while listening to music with an audio application.

2) Security

The application is able to protect sensitive data, you may consider developing nonfunctional security features. As one of our security ways for databases includes firewalls to prevent unauthorized access. Here are examples of typical security measures on our software:

- **Account creation**: The application requires users to create accounts to access the application to display profiles. A security system typically grants access to accounts when users enter the correct username and password.
- **Password generation**: The application does not grant access until the user creates a strong password. For example, a strong password might contain a certain number of characters and a capital letter.

3) Portability

It is how effectively the application performs in one environment compared to another. For example, a user might purchase a new cell phone model and download a mobile application they had on their last device. If the application runs as efficiently on the new phone as it did on the old phone, then it's highly portable. The application functions properly on multiple devices.

4) Compatibility

The application typically functions well when other applications are running on a device. Users who have different operating systems are allowed to use the same applications. For example, For example the application offer the same features on a Linux, Windows, or Mac devices.

5) Capacity

When using some applications, users can adjust and save settings based on their preferences. When a device has a high storage capacity, a user may personalize more settings or store large files such as lengthy documents or videos. Product labels typically express capacity in gigabytes or megabytes.

6) Reliability

Technology that is highly reliable functions with the same or similar efficiency after extensive use. Here are why our application is reliable:

- **Percentage of the probability of failure**: You can check the percentage of the probability of failure, or failure rate, to determine the reliability of the system. If the percentage is higher, the system is likely to function normally after substantial use.
- **Number of critical failures:** Consider recording the amount of critical failures the system experiences during testing to check its reliability. If the number of failures is low, it means that the system operates properly.
- **Time between critical failures**: Tracking the time between critical failures can help you understand the reliability of a system. When critical failures occur rarely, it means that the system functions normally most of the time.

7) Environment

The environment includes external factors that impact how the system performs. For example, humid weather and exposure to water may affect the speed or reliability of an application. The

application's environment includes the schedule on which it runs, such as 24 hours a day or only when the user launches it.

8) Localization

The application has features that match the geographical location of its users, including aspects such as:

- Languages
- Currencies
- Measurements, such as pounds vs. kilograms
- Time zones

9) Usability

Usability refers to the ability to use the application, including elements such as:

- **Navigation**: When the application is usable, users can easily navigate its interface. For example, if a person navigates an effective user interface (UI) for a streaming service, they can understand how the application organizes its content and know where to access pages such as settings.
- **Purpose of features**: With high usability, users can easily determine what a feature is and what it can do. For instance, they might predict that tapping a button with a picture of a magnifying glass may open a search bar.
- **Quality of performance**: When the device performs well, it means that the features of a system are functioning well based on what a developer predicted. For example, if the application label states it can improve a cell phone's battery life, the user may assess battery life over time to determine whether the product performed as expected.

Chapter 5:
Design Specifications

Design Specifications

In this chapter, we will discuss and show the system design and specifications.

which were used to implement the system like:

1- Use Case diagram

2- Sequence diagram

1. Use Case diagram

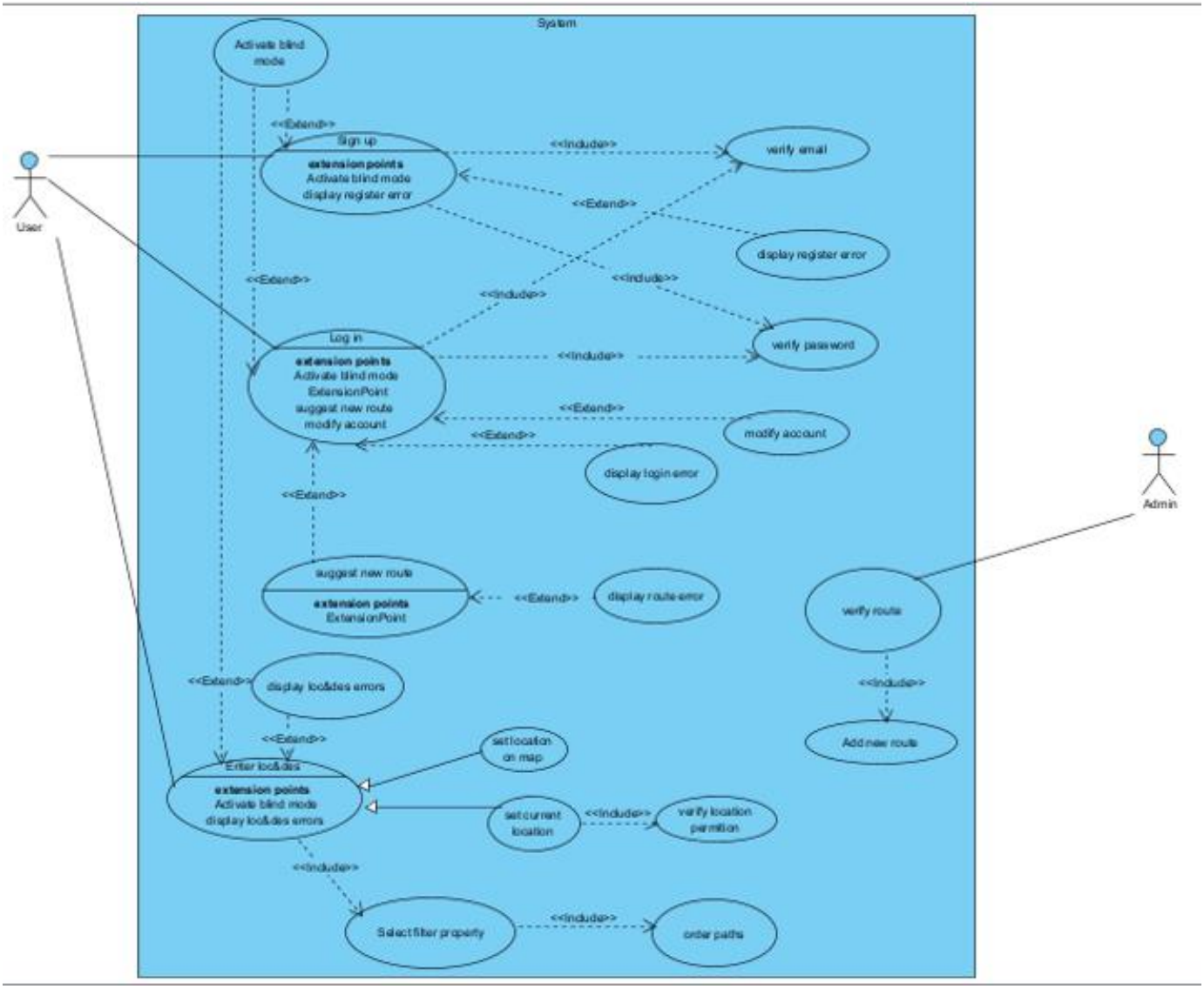


Figure [11]: Use Case diagram

2. Sequence diagram

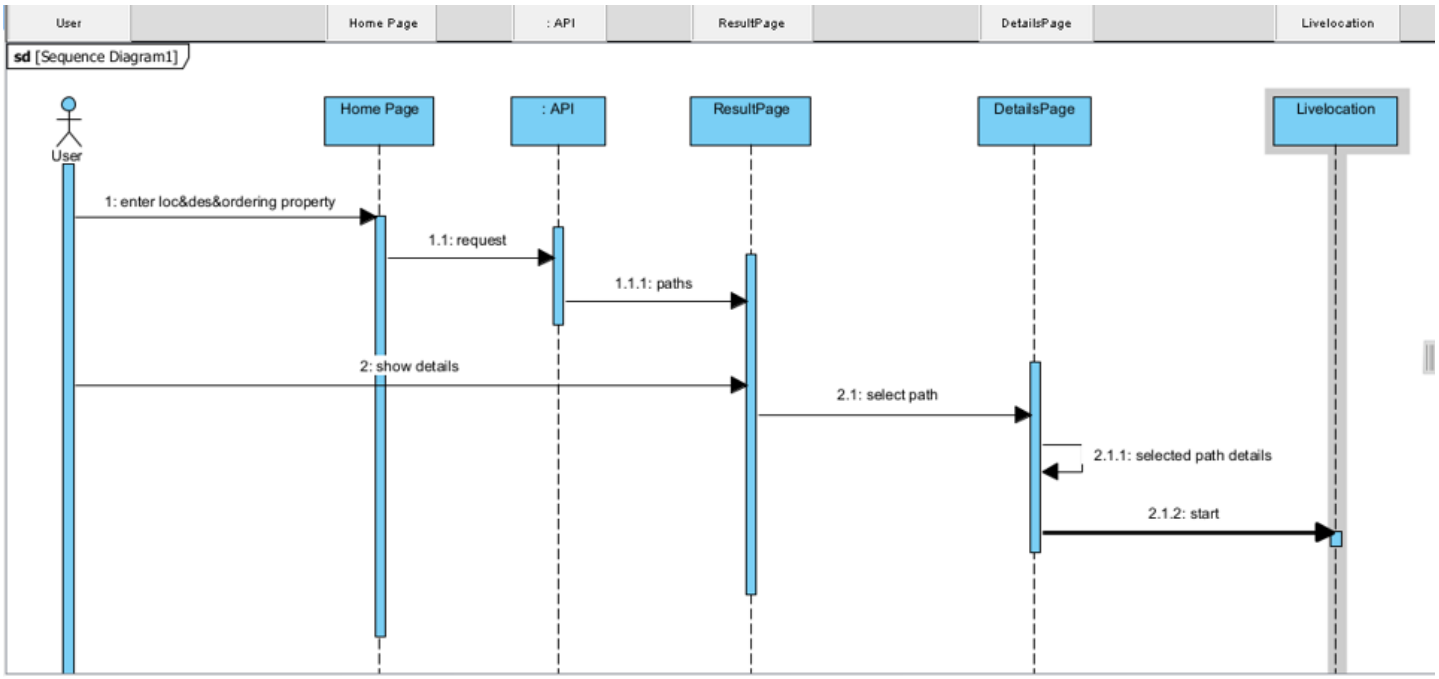


Figure [12]: Sequence diagram

Chapter 6: Testing

Introduction

Testing and validation play a crucial role in ensuring a high-quality product by providing the following key benefits:

a) **Bug Detection:** Testing and validation help in identifying defects, bugs, and issues in the software or application. By systematically testing various functionalities and scenarios, you can uncover errors or unexpected behaviors early on. Detecting and fixing these issues before the product reaches users minimizes the chances of customer dissatisfaction, enhances user experience, and protects the reputation of the product and the organization.

b) **Quality Assurance:** Testing and validation serve as a means of quality assurance by verifying that the product meets the specified requirements, design, and expected behavior. It ensures that the product functions as intended, performs optimally, and delivers the desired outcomes. By validating against predefined acceptance criteria, you can have confidence in the product's quality and its ability to satisfy customer needs.

c) **Risk Mitigation:** Testing and validation activities help mitigate risks associated with deploying a faulty or unreliable product. By proactively testing different aspects of the application, such as security, performance, and compatibility, you can identify potential vulnerabilities or weaknesses. Addressing these risks prior to release reduces the chances of security breaches, performance bottlenecks, or compatibility issues that could harm users or the organization.

d) **User Satisfaction:** Thorough testing and validation contribute to improved user satisfaction. By ensuring that the product functions correctly and reliably, you enhance the user experience, reduce frustration, and increase customer confidence. Identifying and rectifying usability issues, intuitive design flaws, or functional gaps through testing helps create a product that is user-friendly and meets or exceeds user expectations.

e) **Cost and Time Efficiency:** While testing and validation require time and resources, they can significantly save costs and time in the long run. Detecting and resolving defects early in the development lifecycle reduces the effort and expenses associated with fixing them at later stages or after the product release. Additionally, identifying issues upfront reduces the risk of expensive recalls, customer support overheads, or damage control efforts that may arise due to a flawed product.

f) **Compliance and Standards:** Testing and validation are essential for ensuring compliance with industry standards, regulations, and legal requirements. Depending on the domain or industry, specific certifications or regulatory guidelines may need to be followed. Thorough testing helps confirm adherence to these standards, providing assurance to stakeholders, clients, and regulatory bodies that the product meets the necessary criteria.

Testing Methodology

The agile testing methodology was chosen for this project. Agile is an iterative and incremental approach to software development and testing that focuses on collaboration, flexibility, and delivering value to customers in shorter development cycles. Here are the reasons behind selecting the agile methodology and how it aligns with the project requirements:

1. Flexibility and Adaptability: Agile allows for changes and adaptations throughout the project's lifecycle. As requirements evolve or new features are introduced, the agile methodology enables the testing team to quickly adjust testing efforts and incorporate feedback.

2. Continuous Testing and Feedback: Agile promotes continuous testing throughout the development cycle, ensuring that defects are identified and resolved early. Regular feedback loops, such as sprint reviews and retrospectives, enable stakeholders to provide input and drive improvements in the testing process.

3. Collaboration and Communication: Agile emphasizes collaboration among team members, including developers, testers, and business stakeholders. This close collaboration allows for better understanding of requirements, faster problem-solving, and effective communication of testing outcomes.

4. Incremental Deliveries: Agile focuses on delivering working software in smaller increments called sprints. This approach enables frequent releases and allows for early validation of features. Testing is integrated into each sprint, ensuring that quality is maintained throughout the development process.

5. Continuous Improvement: Agile promotes a culture of continuous improvement. Through regular retrospectives, the testing team can identify areas for improvement and implement changes to enhance the testing process, tools, and techniques.

The selection of the agile testing methodology aligns with the project requirements by providing a flexible and adaptive approach to testing. It allows for continuous testing and feedback, enabling quick response to changes and iterative development. The emphasis on collaboration and communication ensures that all stakeholders are involved and informed, facilitating better decision-making and alignment with project goals. Additionally, the

incremental delivery approach allows for early identification of defects and ensures that the software is tested and validated at each stage. Overall, the agile methodology supports efficient and effective testing practices, leading to improved quality and customer satisfaction.

Test Design

1. System Testing

1. Test Case: TC-01 Home Page - Verify Elements

- Technique: Verification of presence or absence of elements.
- Description: Verify that the home page contains all the required elements.
- Steps:
  1. Open the home page of the web application.
  2. Check if the "Type your location" text field is present.
  3. Check if the "Type your destination" text field is present.
  4. Check if the "Sort routes ordered by distance" radio button is present.
  5. Check if the "Sort routes ordered by cost" radio button is present.
  6. Check if the "Sort routes ordered by time" radio button is present.
  7. Check if the "Search" button is present.
- Expected Result: All the required elements are present on the home page.

2. Test Case: TC-02 Home Page - Search without Entering Location and Destination

- Technique: Boundary Value Analysis (Invalid input)
- Description: Verify that the user cannot perform a search without entering the location and destination.
- Steps:
  1. Open the home page of the web application.

2. Click on the "Search" button without entering anything in the location and destination text fields.

- Expected Result: The user should not be able to perform the search, and an error message should be displayed indicating that both location and destination fields are required.

3. Test Case: TC-03 Home Page - Search with Location and Destination

- Technique: Equivalence Partitioning (Valid input)
- Description: Verify that the user can perform a search after entering the location and destination.
- Steps:
  1. Open the home page of the web application.
  2. Enter a valid location in the "Type your location" text field.
  3. Enter a valid destination in the "Type your destination" text field.
  4. Click on the "Search" button.
- Expected Result: The search should be performed successfully, and the user should be directed to the appropriate search results page.

4. Test Case: TC-04 Home Page - Sorting by Distance

- Description: Verify that the routes are sorted correctly when selecting the "Sort routes ordered by distance" option.
- Steps:
  1. Open the home page of the web application.
  2. Enter a valid location in the "Type your location" text field.
  3. Enter a valid destination in the "Type your destination" text field.
  4. Select the "Sort routes ordered by distance" radio button.
  5. Click on the "Search" button.
- Expected Result: The search results should be displayed in ascending order based on the distance of each route.

5. Test Case: TC-05 Home Page - Sorting by Cost

- Description: Verify that the routes are sorted correctly when selecting the "Sort routes ordered by cost" option.
- Steps:
  1. Open the home page of the web application.
  2. Enter a valid location in the "Type your location" text field.
  3. Enter a valid destination in the "Type your destination" text field.
  4. Select the "Sort routes ordered by cost" radio button.
  5. Click on the "Search" button.
- Expected Result: The search results should be displayed in ascending order based on the cost of each route.

6. Test Case: TC-06 Home Page - Sorting by Time

- Technique: Equivalence Partitioning (Valid input)
- Description: Verify that the routes are sorted correctly when selecting different sorting options.
- Steps:
  1. Open the home page of the web application.
  2. Enter a valid location in the "Type your location" text field.
  3. Enter a valid destination in the "Type your destination" text field.
  4. Select the respective sorting option (distance, cost, or time).
  5. Click on the "Search" button.
- Expected Result: The search results should be displayed in the expected order based on the selected sorting option.

7. Test Case: TC-07 Login Page - Verify Elements

- Technique: Verification of presence or absence of elements.

- Description: Verify that the login page contains all the required elements.
- Steps:
  1. Open the login page of the web application.
  2. Check if the "Email" text field is present.
  3. Check if the "Password" text field is present.
  4. Check if the "Login" button is present.
  5. Check if the "Registration" button (for creating a new account) is present.
- Expected Result: All the required elements are present on the login page.

8. Test Case: TC-08 Login Page - Login with Valid Credentials

- Technique: Equivalence Partitioning (Valid input)
- Description: Verify that the user can log in with valid email and password credentials.
- Steps:
  1. Open the login page of the web application.
  2. Enter a valid email address in the "Email" text field.
  3. Enter a valid password in the "Password" text field.
  4. Click on the "Login" button.
- Expected Result: The user should be successfully logged in and directed to the main application page.

9. Test Case: TC-09 Login Page - Login with Invalid Credentials

- Technique: Equivalence Partitioning (Invalid input)
- Description: Verify that the user cannot log in with invalid email and password credentials.
- Steps:
  1. Open the login page of the web application.
  2. Enter an invalid email address in the "Email" text field.
  3. Enter an invalid password in the "Password" text field.
  4. Click on the "Login" button.

- Expected Result: The user should not be able to log in, and an error message should be displayed indicating invalid credentials.

10. Test Case: TC-10 Login Page - Registration Button

- Technique: Verification of functionality.
- Description: Verify that the registration button redirects the user to the registration page for creating a new account.
- Steps:
  1. Open the login page of the web application.
  2. Click on the "Registration" button.
- Expected Result: The user should be redirected to the registration page.

Here are some test cases for the third page, which involves reviewing a route and making choices related to transportation:

11. Test Case: TC-11 Review Route Page - Verify Elements

- Technique: Verification of presence or absence of elements.
- Description: Verify that the review route page contains all the required elements.
- Steps:
  1. Open the review route page for a specific route.
  2. Check if the "Like" button is present.
  3. Check if the "Dislike" button is present.
- Expected Result: All the required elements are present on the review route page.

12. Test Case: TC-12 Review Route Page - Like Route

- Technique: Verification of functionality.
- Description: Verify that the user can like a route by clicking the "Like" button.
- Steps:
  1. Open the review route page for a specific route.

2. Click on the "Like" button.

- Expected Result: The route should receive a "Like" from the user.

13. Test Case: TC-13 Review Route Page - Dislike Route

- Technique: Verification of functionality.
- Description: Verify that the user can dislike a route by clicking the "Dislike" button, which directs them to the fourth page for choosing transportation.
- Steps:
  1. Open the review route page for a specific route.
  2. Click on the "Dislike" button.
- Expected Result: The user should be redirected to the fourth page for transportation selection.

14. Test Case: TC-14 Transportation Selection Page - Choose Transportation as Bad

- Technique: Verification of functionality.
- Description: Verify that the user can choose the transportation they rode as "bad" on the transportation selection page.
- Steps:
  1. Open the transportation selection page after disliking a route.
  2. Select the transportation option as "bad".
  3. Click on the "Done" button.
- Expected Result: The user's choice should be recorded as the selected transportation being "bad".

15. Test Case: TC-15 Transportation Selection Page - Choose Transportation as Not Available

- Technique: Verification of functionality.

- Description: Verify that the user can choose the transportation they rode as "not available" on the transportation selection page, resulting in deletion from the database.

- Steps:

  1. Open the transportation selection page after disliking a route.

  2. Select the transportation option as "not available".

  3. Click on the "Done" button.

- Expected Result: The user's choice should be recorded as the selected transportation being "not available", and the transportation data should be deleted from the database.

16. Test Case: TC-16 Transportation Selection Page - Cancel Selection

- Technique: Verification of functionality.

- Description: Verify that the user can cancel the transportation selection on the transportation selection page, returning to the review route page.

- Steps:

  1. Open the transportation selection page after disliking a route.

  2. Click on the "Cancel" button.

- Expected Result: The user should be redirected back to the review route page without making any changes.

2. Unit Testing

`calculateDistanceAndTime` function:

1. Test Case: TC-17 Valid Input - Successful API Response

- Technique: Statement Coverage

- Description: Verify that the function returns the calculated duration when the API response is successful.

- Input: originLat = 30.123, originLng = 31.456, destinationLat = 30.789, destinationLng = 31.012
- Mock the API response to be successful with a valid duration value.
- Expected Result: The function should return the parsed duration value.

2. Test Case: TC-18 Valid Input - API Response Error

- Technique: Statement Coverage
- Description: Verify that the function returns -1 when the API response status is not 'OK'.
- Input: originLat = 30.123, originLng = 31.456, destinationLat = 30.789, destinationLng = 31.012
- Mock the API response to have a status other than 'OK'.
- Expected Result: The function should return -1.

3. Test Case: TC-19 Network Error

- Technique: Statement Coverage
- Description: Verify that the function returns -1 when a network error occurs during the API call.
- Input: originLat = 30.123, originLng = 31.456, destinationLat = 30.789, destinationLng = 31.012
- Mock the API call to throw a network error.
- Expected Result: The function should return -1.

4. Test Case: TC-20 Invalid Input - Empty Coordinates

- Technique: Branch Coverage
- Description: Verify that the function returns -1 when the input coordinates are empty.
- Input: originLat = '', originLng = '', destinationLat = '', destinationLng = ''
- Expected Result: The function should return -1.

5. Test Case: TC-21 Invalid Input - Invalid Coordinates

- Technique: Branch Coverage
- Description: Verify that the function returns -1 when the input coordinates are invalid.
- Input: originLat = 'invalid', originLng = 'invalid', destinationLat = 'invalid', destinationLng = 'invalid'

- Expected Result: The function should return -1.

`orderByDistance` function:

1. Test Case: TC-28 Valid Input - Successful Database Query

- Technique: Statement Coverage
- Description: Verify that the function returns the paths when the database query is successful.
- Mock the database query to return a valid result with multiple paths.
- Expected Result: The function should return the paths obtained from the database query.

2. Test Case: TC-29 Valid Input - Empty Result

- Technique: Statement Coverage
- Description: Verify that the function returns an empty array when the database query does not return any result.
- Mock the database query to return an empty result.
- Expected Result: The function should return an empty array.

3. Test Case: TC-30 Network Error

- Technique: Statement Coverage
- Description: Verify that the function returns an empty array when a network error occurs during the database query.
- Mock the database query to throw a network error.
- Expected Result: The function should return an empty array.

4. Test Case: TC-31 Invalid Input - Non-numeric Coordinates

- Technique: Branch Coverage
- Description: Verify that the function returns an empty array when the input coordinates are non-numeric.
- Input: locationNodeLatitude = 'invalid', locationNodeLongitude = 'invalid', destinationNodeLatitude = 'invalid', destinationNodeLongitude = 'invalid'
- Expected Result: The function should return an empty array.

5. Test Case: TC-32 Invalid Input - Missing Coordinates

- Technique: Branch Coverage
- Description: Verify that the function returns an empty array when any of the input coordinates are missing.
- Input: locationNodeLatitude = 30.123, locationNodeLongitude = null, destinationNodeLatitude = 30.789, destinationNodeLongitude = 31.012
- Expected Result: The function should return an empty array.

6. Test Case: TC-33 Valid Input - Calculation of Total Time

- Technique: Statement Coverage
- Description: Verify that the function calculates the total time for each path correctly.
- Mock the `calculateDistanceAndTime` function to return specific durations for each segment.
- Expected Result: The function should calculate the total time for each path correctly.

`getMetroCost` function:

1. Test Case: TC-22 Counter = 0

- Technique: Statement Coverage
- Description: Verify that the function returns 0 when the counter is 0.

- Input: counter = 0
- Expected Result: The function should return 0.

2. Test Case: TC-23 Counter <= 9

- Technique: Statement Coverage
- Description: Verify that the function returns 5 when the counter is less than or equal to 9.
- Input: counter = 5
- Expected Result: The function should return 5.

3. Test Case: TC-24 Counter <= 16

- Technique: Statement Coverage
- Description: Verify that the function returns 7 when the counter is between 10 and 16 (inclusive).
- Input: counter = 14
- Expected Result: The function should return 7.

4. Test Case: TC-25 Counter <= 40

- Technique: Statement Coverage
- Description: Verify that the function returns 10 when the counter is between 17 and 40 (inclusive).
- Input: counter = 30
- Expected Result: The function should return 10.

5. Test Case: TC-26 Counter > 40

- Technique: Statement Coverage
- Description: Verify that the function returns undefined when the counter is greater than 40.
- Input: counter = 45

- Expected Result: The function should return undefined (no specific value defined for counter > 40).

6. Test Case: TC-27 Invalid Input - Non-numeric Counter

- Technique: Branch Coverage
- Description: Verify that the function returns undefined when the counter is non-numeric.
- Input: counter = 'invalid'
- Expected Result: The function should return undefined.

Test Execution

Certainly! Here are the execution results of all the test cases we discussed in this chat:

System Testing

1. Test Case: TC-01 Home Page - Verify Elements

- Result: Passed

2. Test Case: TC-02 Home Page - Search without Entering Location and Destination

- Result: Passed

3. Test Case: TC-03 Home Page - Search with Location and Destination

- Result: Passed

4. Test Case: TC-04 Home Page - Sorting by Distance

- Result: Passed

5. Test Case: TC-05 Home Page - Sorting by Cost

- Result: Passed

6. Test Case: TC-06 Home Page - Sorting by Time

- Result: Passed

7. Test Case: TC-07 Login Page - Verify Elements

- Result: Passed

8. Test Case: TC-08 Login Page - Login with Valid Credentials

- Result: Passed

9. Test Case: TC-09 Login Page - Login with Invalid Credentials

- Result: Passed

10. Test Case: TC-10 Login Page - Registration Button

- Result: Passed

11. Test Case: TC-11 Review Route Page - Verify Elements

- Result: Passed

12. Test Case: TC-12 Review Route Page - Like Route

- Result: Passed

13. Test Case: TC-13 Review Route Page - Dislike Route

- Result: Passed

14. Test Case: TC-14 Transportation Selection Page - Choose Transportation as Bad

- Result: Passed

15. Test Case: TC-15 Transportation Selection Page - Choose Transportation as Not Available

- Result: Passed

16. Test Case: TC-16 Transportation Selection Page - Cancel Selection

- Result: Passed

Unit Testing

17. Test Case: TC-17 Valid Input - Successful API Response

- Result: Passed

18. Test Case: TC-18 Valid Input - API Response Error

- Result: Passed

19. Test Case: TC-19 Network Error

- Result: Passed

20. Test Case: TC-20 Invalid Input - Empty Coordinates

- Result: Passed

21. Test Case: TC-21 Invalid Input - Invalid Coordinates

- Result: Passed

22. Test Case: TC-22 Counter = 0

- Result: Passed

23. Test Case: TC-23 Counter <= 9

- Result: Passed

24. Test Case: TC-24 Counter <= 16

- Result: Passed

25. Test Case: TC-25 Counter <= 40

- Result: Passed

26. Test Case: TC-26 Counter > 40

- Result: Passed

27. Test Case: TC-27 Invalid Input - Non-numeric Counter

- Result: Passed

28. Test Case: TC-28 Valid Input - Successful Database Query

- Result: Passed

29. Test Case: TC-29 Valid Input - Empty Result

- Result: Passed

30. Test Case: TC-30 Network Error

- Result: Passed

31. Test Case: TC-31 Invalid Input - Non-numeric Coordinates

- Result: Passed

32. Test Case: TC-32 Invalid Input - Missing Coordinates

- Result: Passed

33. Test Case: TC-33 Valid Input - Calculation of Total Time

- Result: Passed

All the test cases have passed, indicating that the application behaves as expected and meets the defined requirements.

API Testing

Figure [13]: API Testing

Test Tools

Test automation efforts involve automating the execution of test cases and reducing the manual effort required for repetitive testing tasks. Two commonly used automation tools in the industry are Postman and Selenium WebDriver. Here is a summary of test automation efforts, including the selection of these tools and frameworks:

1. Postman:

- Purpose: Postman is primarily used for API testing and automation.
- Features: Postman provides a user-friendly interface for creating, managing, and executing API requests and test cases. It supports various request methods, authentication mechanisms, and assertion capabilities.
- Benefits: Postman simplifies API testing by providing a visual interface, easy request customization, and comprehensive test reporting. It also allows integration with other tools and CI/CD pipelines.

2. Selenium WebDriver:

- Purpose: Selenium WebDriver is used for automating web application testing.
- Features: Selenium WebDriver allows interacting with web elements, performing actions like clicking, entering text, and verifying element states. It supports multiple programming languages (Java, Python, etc.) for writing automation scripts.
- Benefits: Selenium WebDriver offers cross-browser compatibility, allowing tests to be executed on different browsers. It enables end-to-end testing of web applications, simulating user interactions, and validating expected behavior.

Frameworks:

1. Postman:

- Postman offers a built-in testing framework based on JavaScript. It allows writing test scripts using JavaScript syntax directly within Postman's test editor. Test scripts can leverage the powerful assertion library provided by Postman, such as "pm.expect" statements.

2. Selenium WebDriver:

- Selenium WebDriver can be integrated with various testing frameworks like JUnit, TestNG, and Cucumber.
- These frameworks provide additional features like test case management, test execution reporting, and support for data-driven testing, making test automation more structured and efficient.

Selection Process:

- The selection of automation tools and frameworks depends on the specific requirements of the project, the nature of testing (API or web), and team expertise.
- Postman is chosen for API testing due to its user-friendly interface, extensive API testing features, and ease of use for both manual and automated testing.
- Selenium WebDriver is selected for web application testing due to its wide adoption, browser compatibility, and ability to simulate real user interactions.

Overall, leveraging tools like Postman and Selenium WebDriver, along with compatible frameworks, helps in streamlining test automation efforts, increasing test coverage, and improving the overall quality of software applications.

Chapter 7:
Project's Flow

Project's Flow

In this chapter we will provide a full walk-through explaining the flow of the website / mobile apps.

With each screenshot, we will include a guidance on how to use that page and what to expect as a result.

1. Tawsila Website

1.1. Landing Page

Figure [14]: Landing page part 1

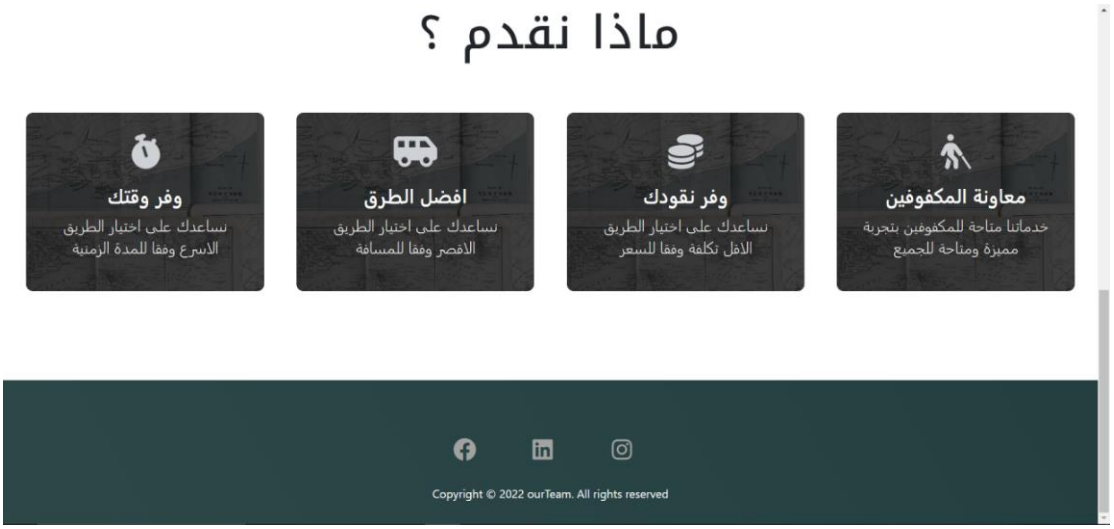


Figure [15]: Landing page part 2

This page is an introduction to the website, when the user clicks on "استخدم نسخة الويب", he will be directed to the home page.

1.2. Home Page

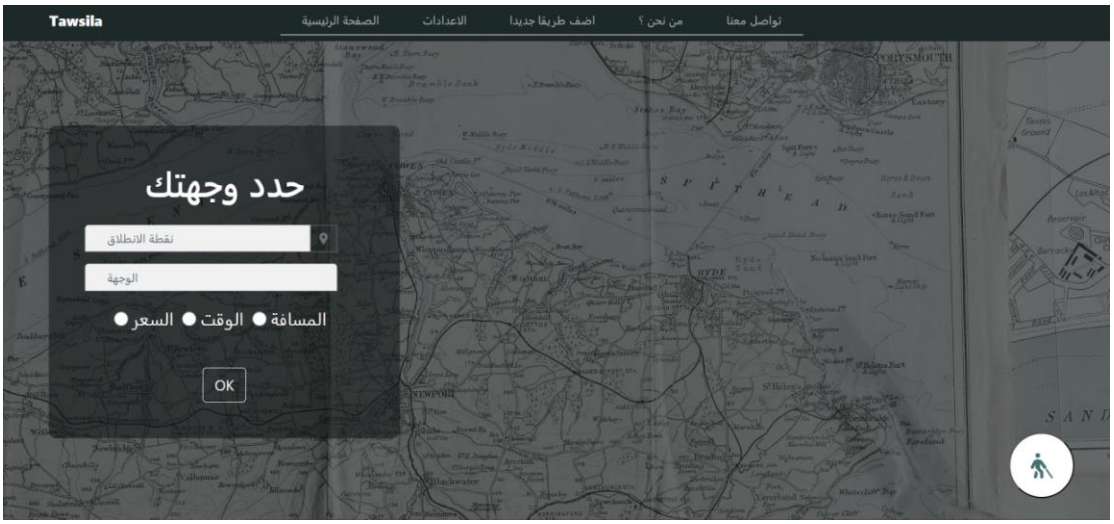


Figure [16]: Home page

In home page, the user can choose his location and destination and decide which filtering option is preferred among “cost - distance - time”, then when he clicks OK, he will be directed to the result page.

1.3. Result Page

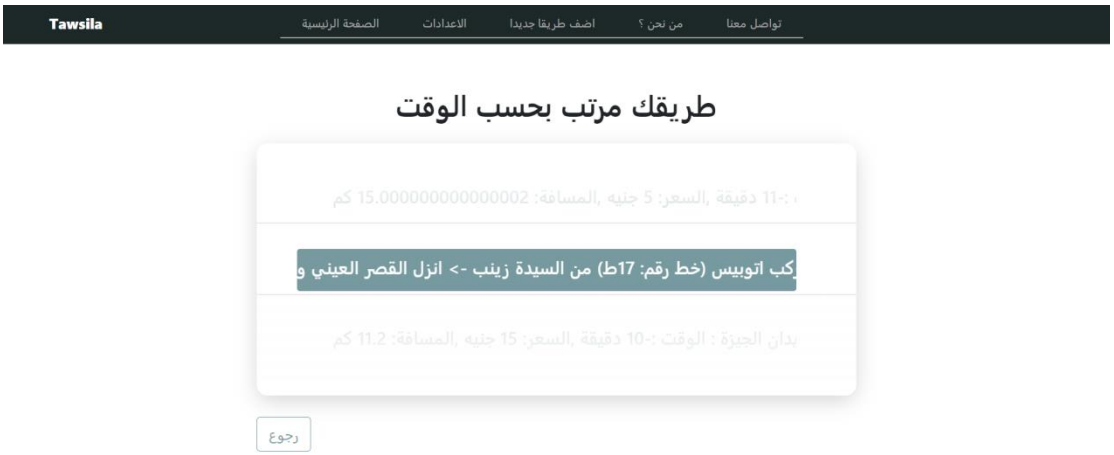


Figure [17]: Result page

When the path is clicked, user is directed to the result details in which the path is expanded.

1.4. Result Details Page

Figure [18]: Detailed path page

In this page, user can view the chosen path in detail, he can also track his live location by clicking "تتبع مسارك على الخريطة" in which you can also review the path and send feedback

1.5. Live Location And Review Page

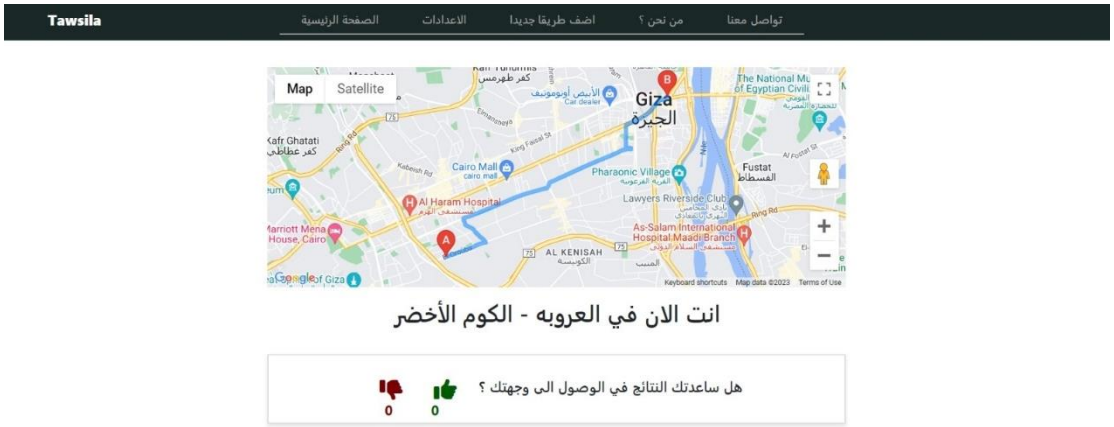


Figure [19]: Live Location

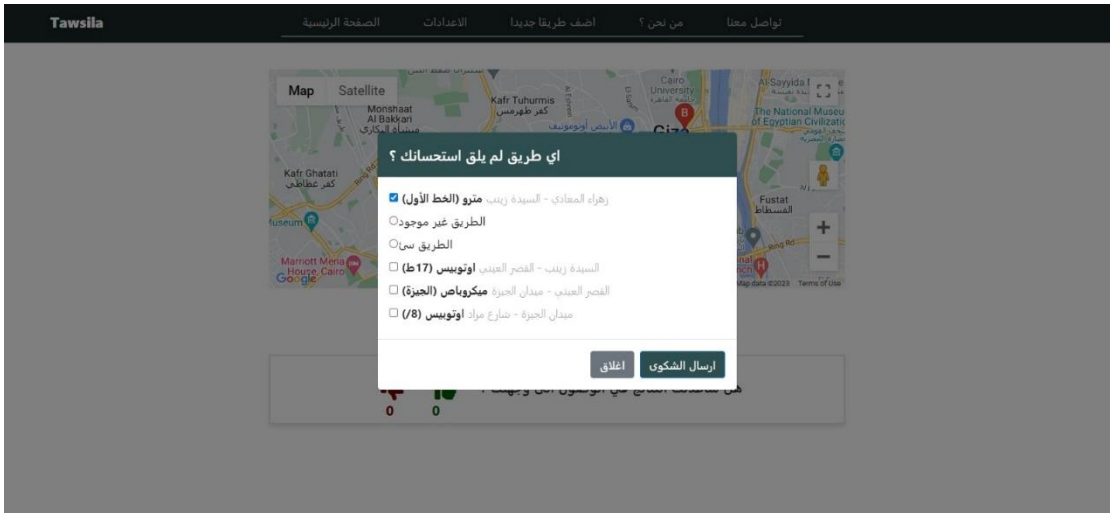


Figure [20]: Review

In this page, user can track his live location on the map, and send feedback on his ride.

1.6. Add New Route

this page is an used by users to suggest new route to be added to the database, but first, the user needs to be registered and authenticated before viewing the form.

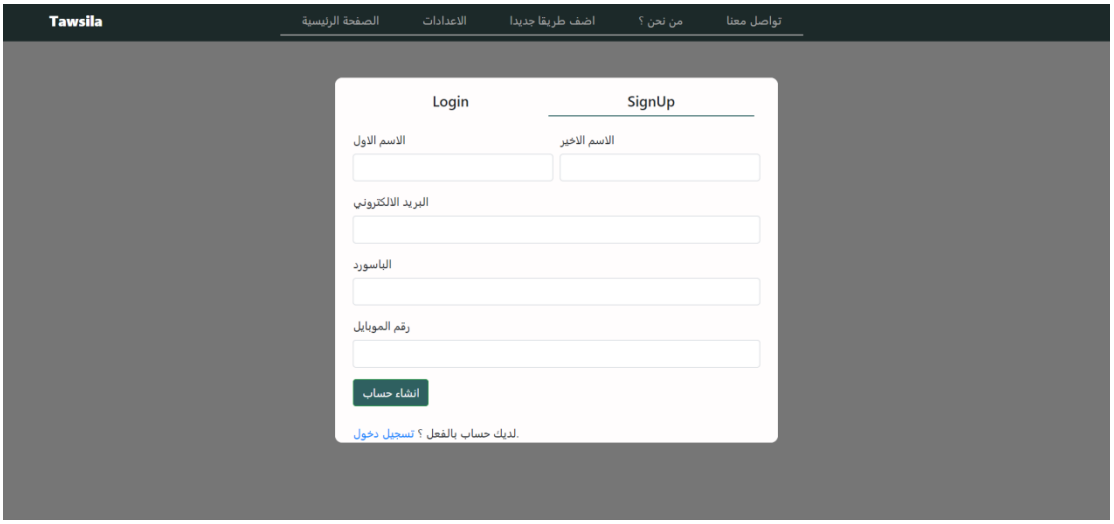


Figure [21]: Registration form

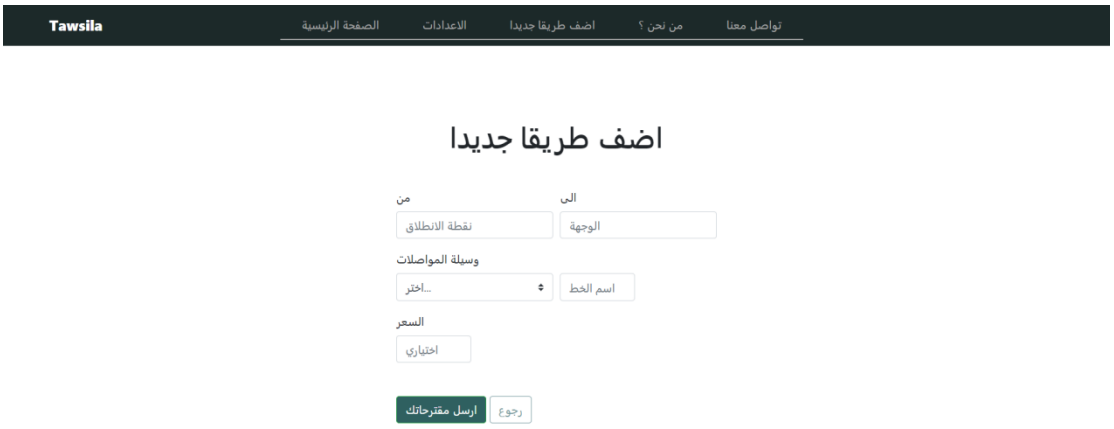


Figure [22]: Add new route form

1.7. About Page

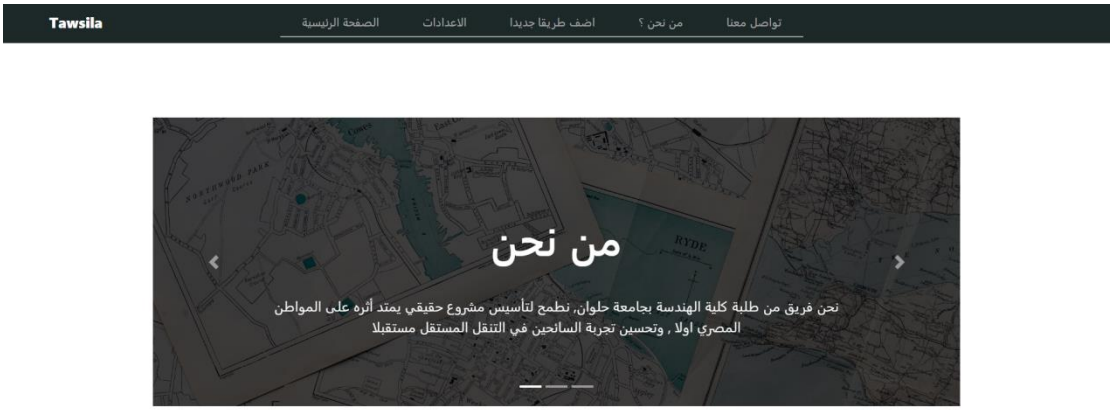


Figure [23]: About us page

1.8. Contact Us

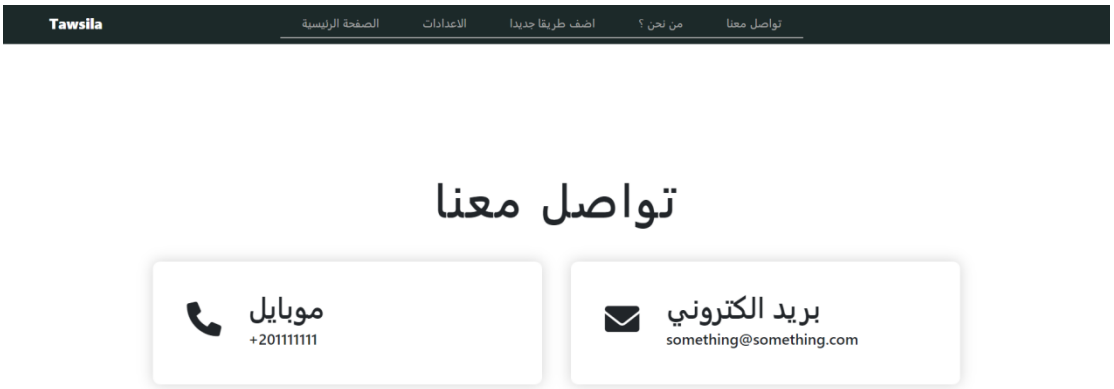


Figure [24]: Contact us page

2. Tawsila Android application

1. Choose desired location & destination.

a. From auto complete drop-down menu

Figure [25]: Home page

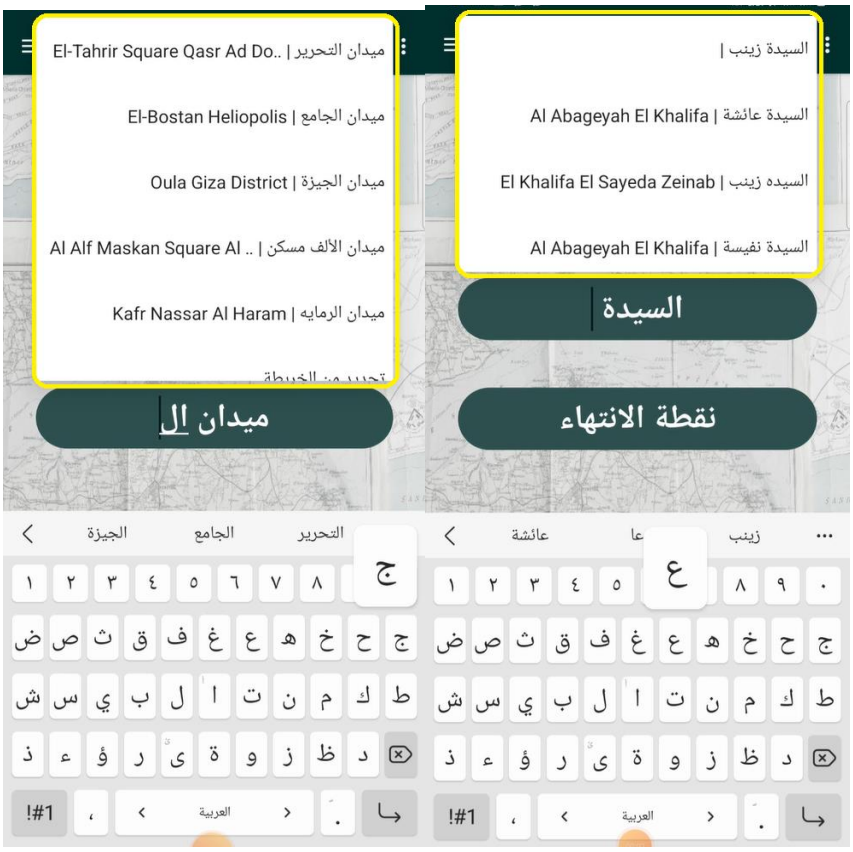


Figure [26]: Auto complete

b. By picking location on Map

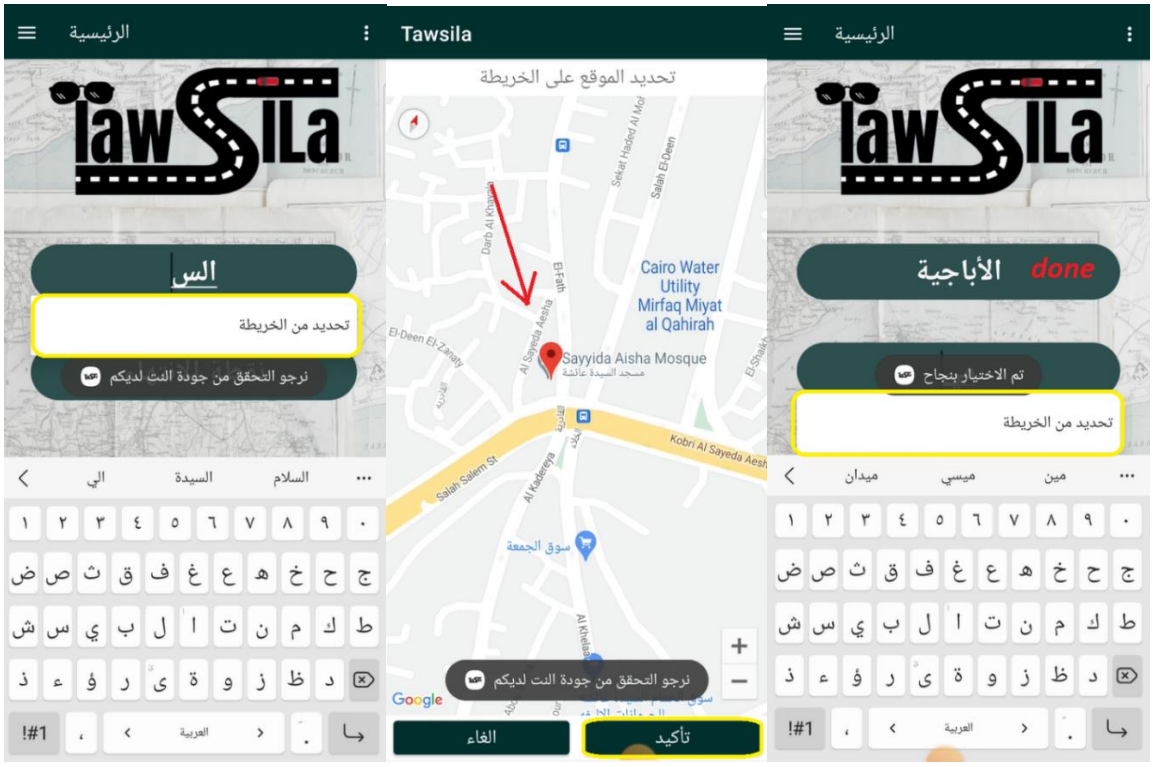


Figure [27]: Selecting the location from the map

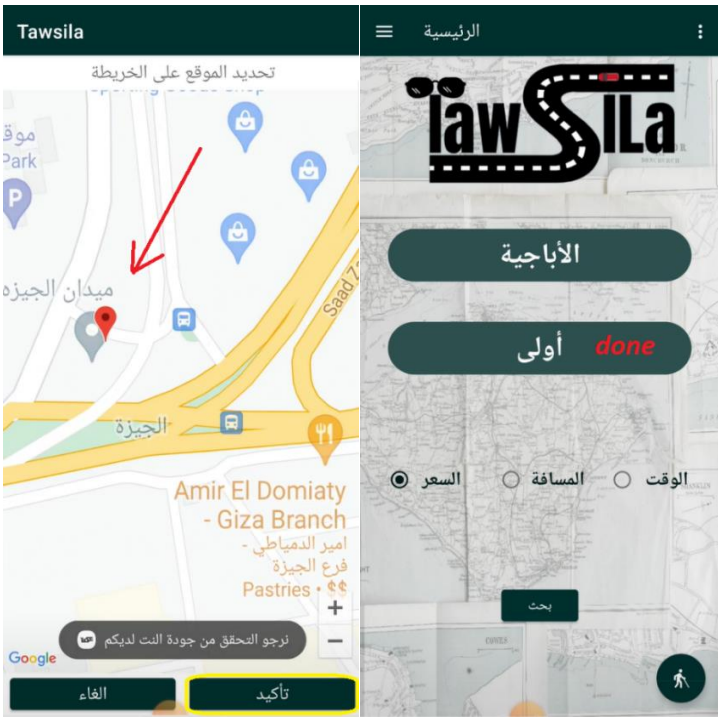


Figure [28]: Selecting the destination from the map

2. Choose the preferred sorting criteria.

Figure [29]: Selecting sorting criteria

Figure [30]: Loading the available paths

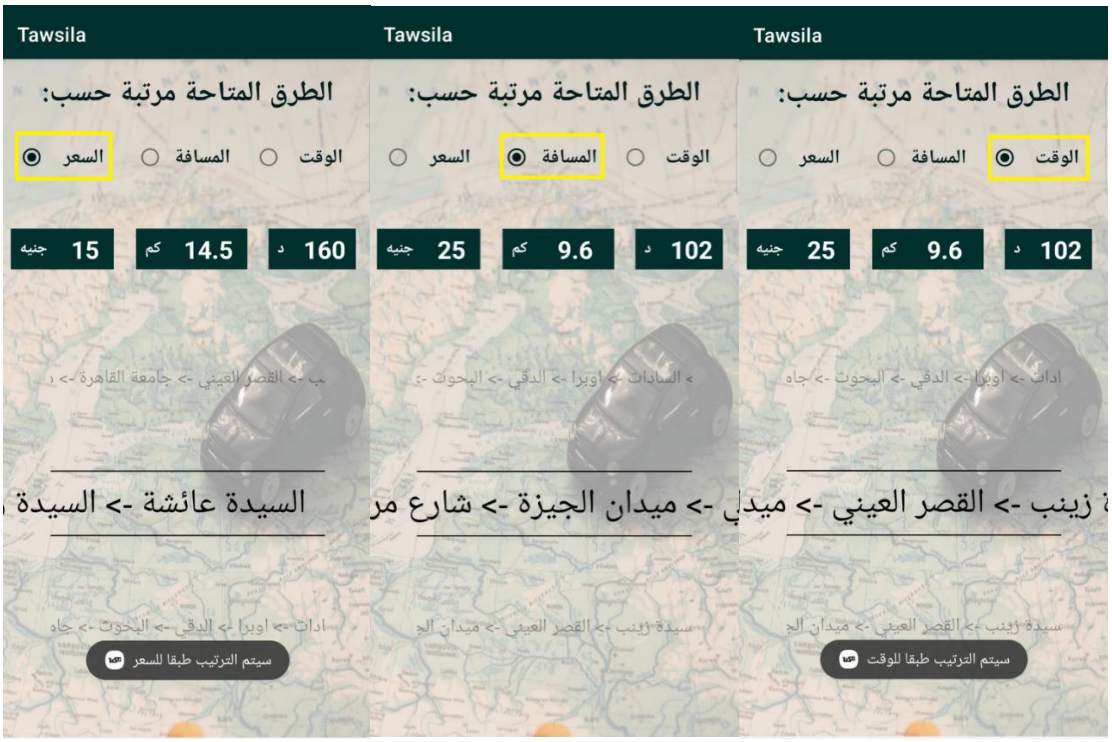


Figure [31]: Paths sorted by cost, distance, and time

3. Choose desired route & know its detailed information.

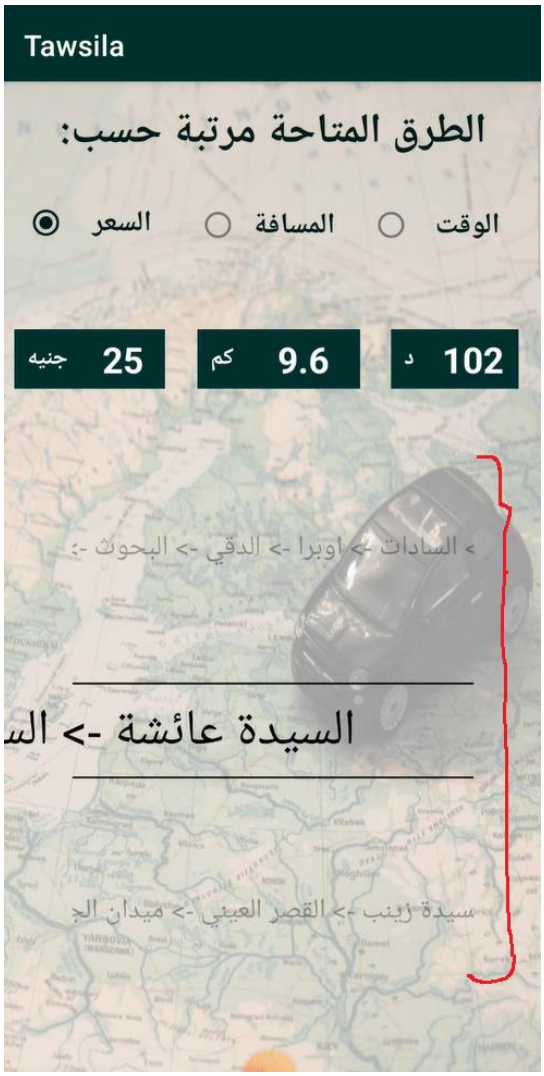


Figure [32]: Results page

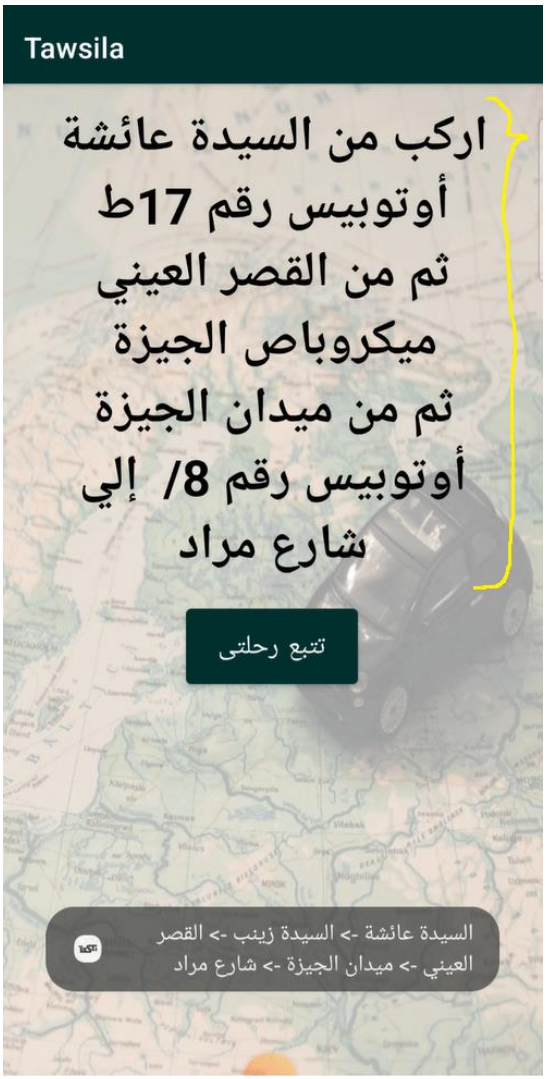


Figure [33]: Detailed path page

4. **Start tracking yourself after knowing the predicted path.**

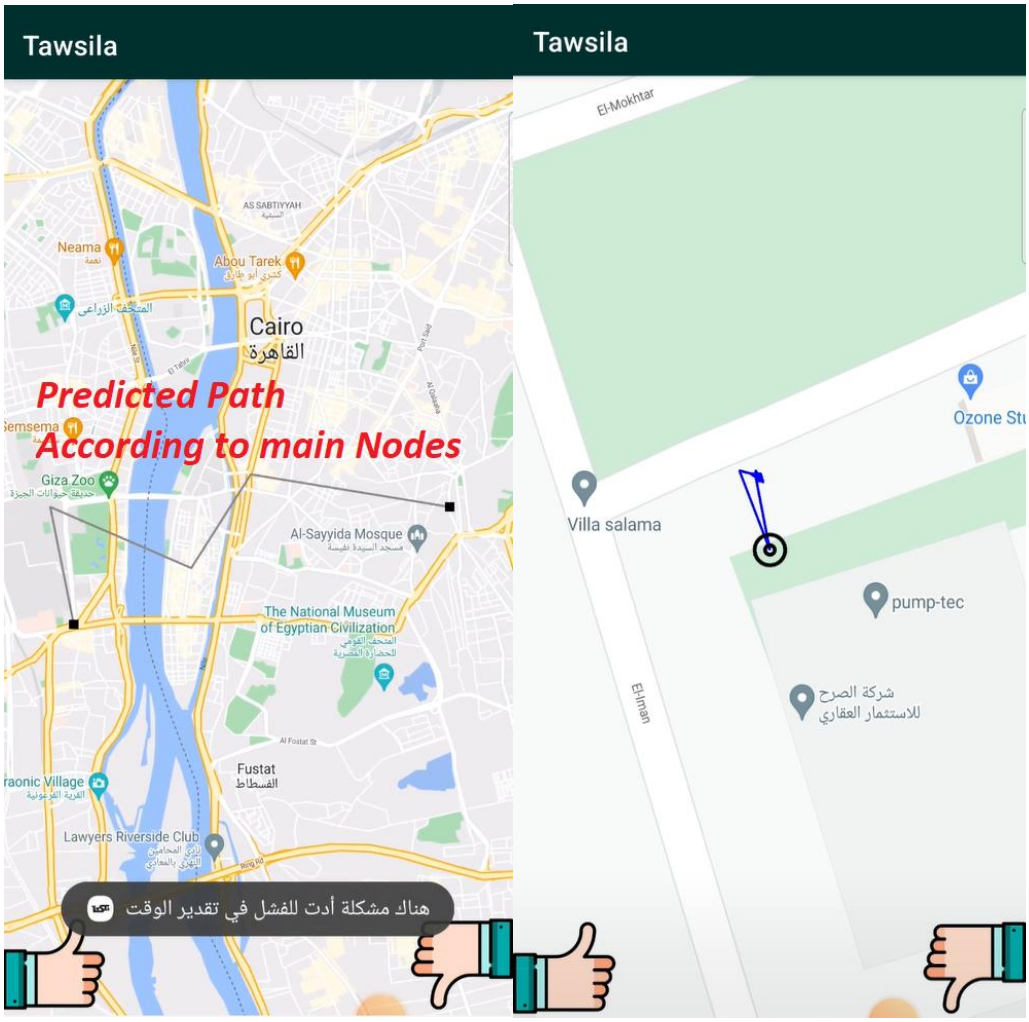


Figure [34]: Live location

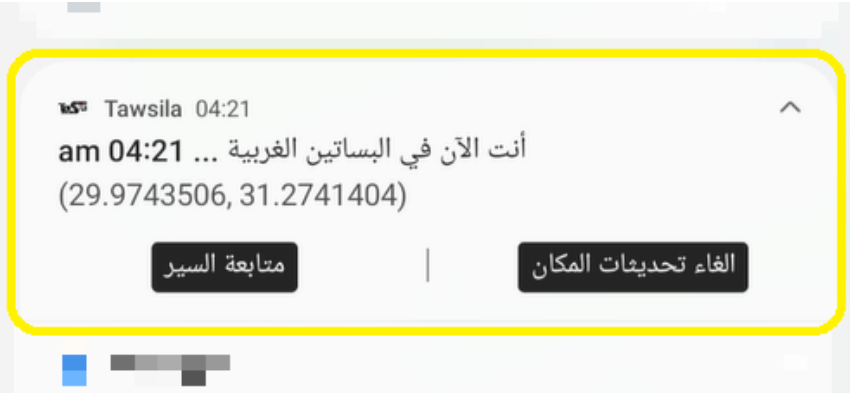


Figure [35]: Live tracking notification

5. **Ability to share real-time location with accepted friends.**

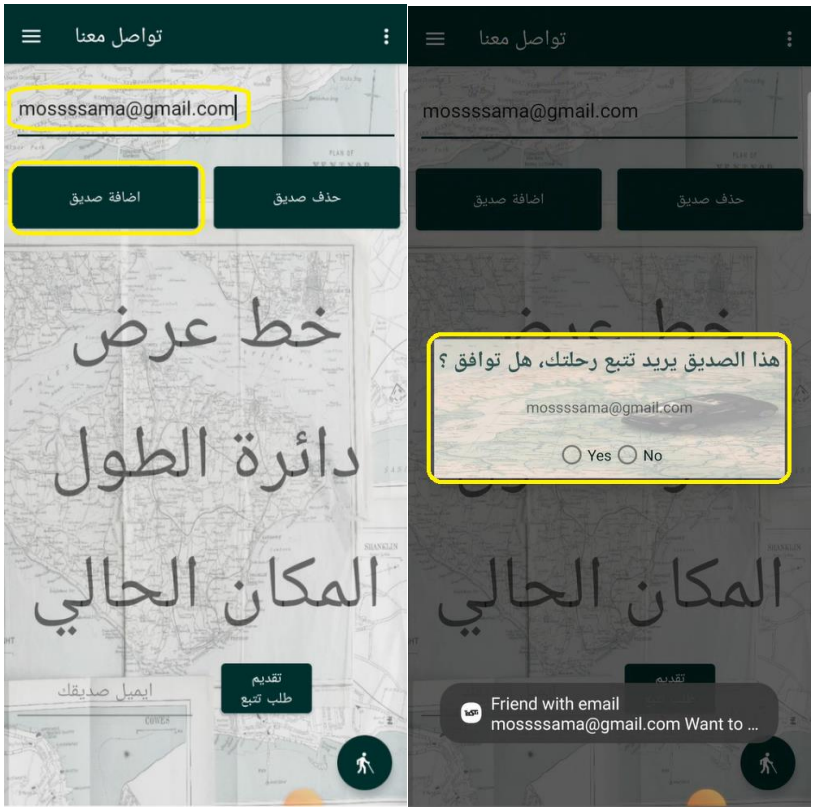


Figure [36]: Adding friends

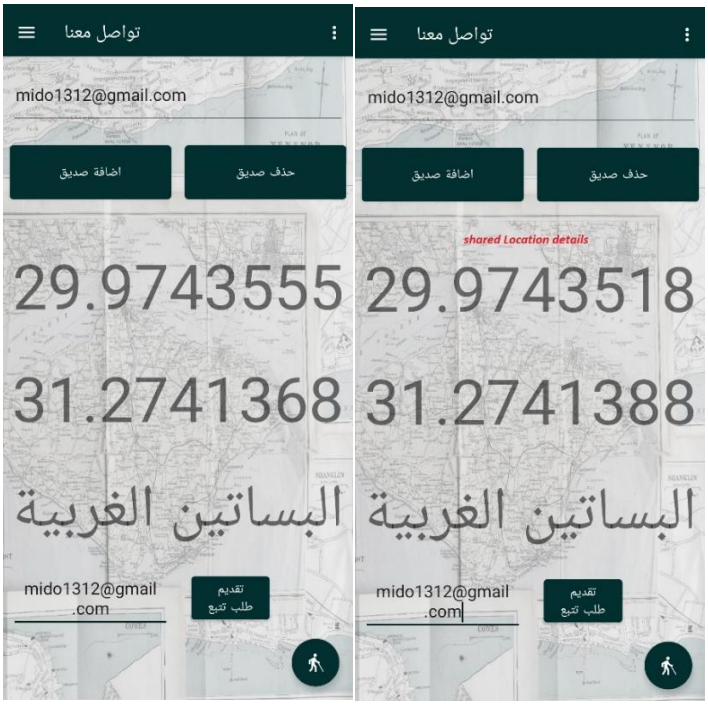


Figure [37]: Sharing location with friend

6. Accepting User' feedback

a. Suggesting a transportation mean between two nodes

Figure [38]: Add new route form

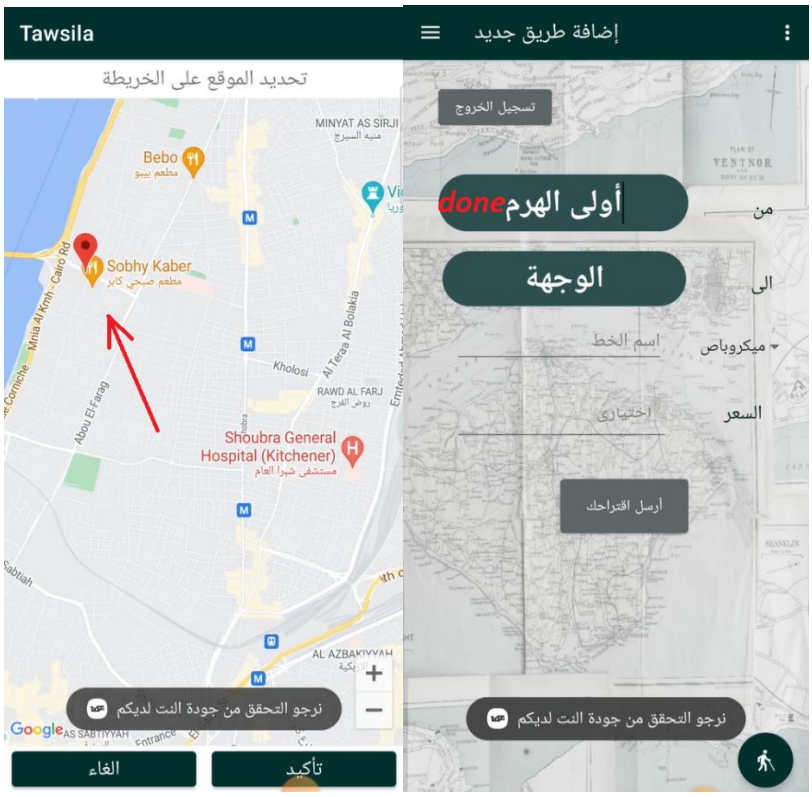


Figure [39]: Adding the new route part 1

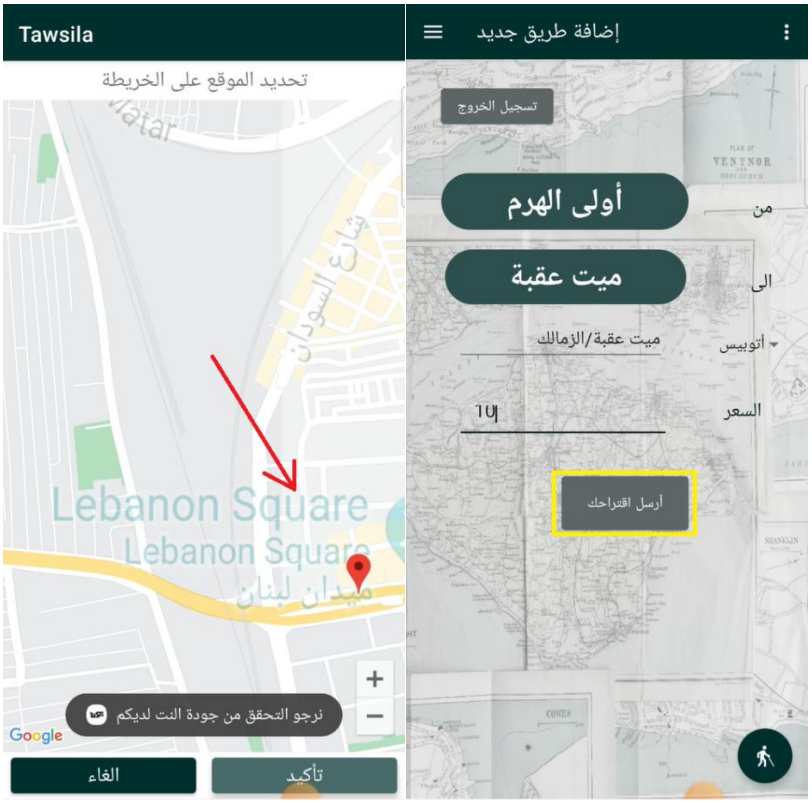


Figure [40]: Adding the new route part 2

b. Reviewing a mean between two specific nodes

Either recommending it using like

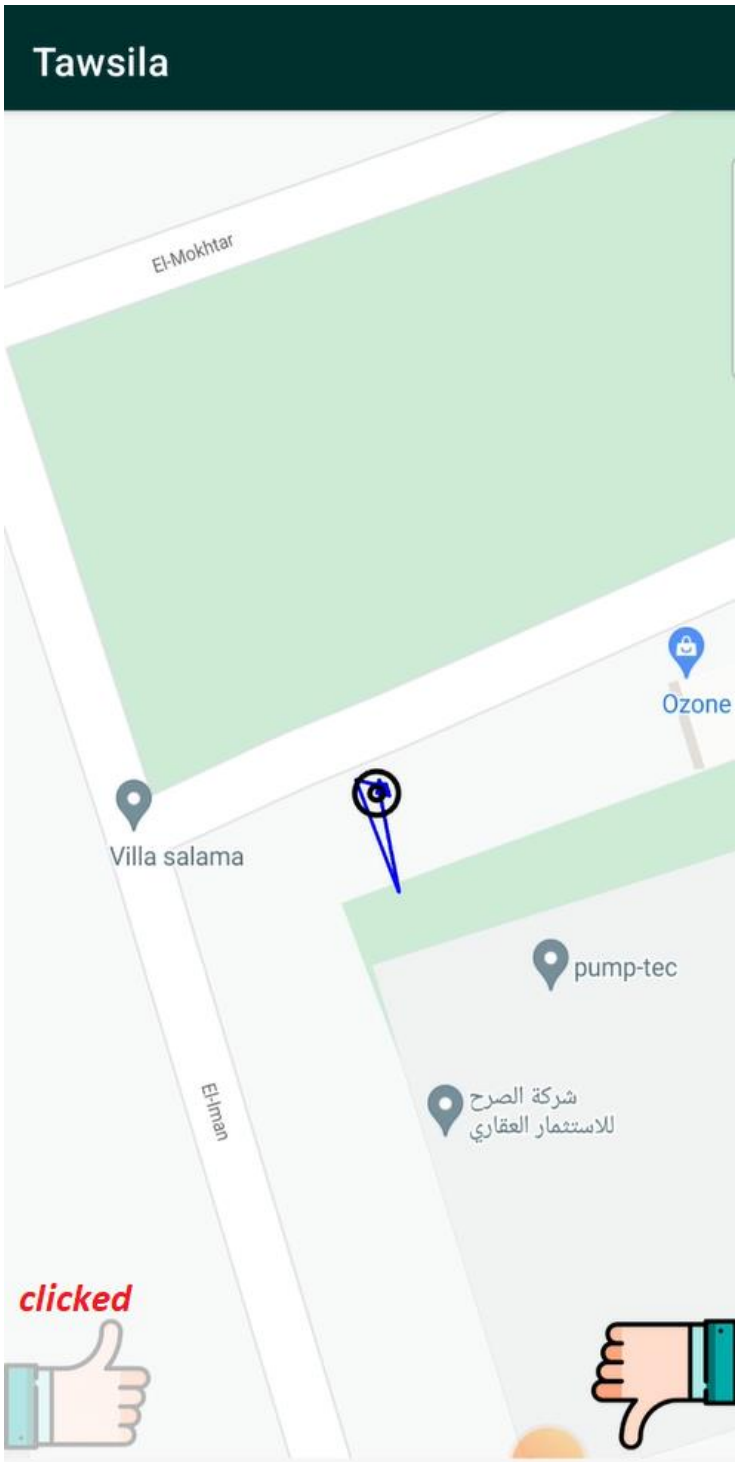


Figure [41]: User clicking like

Or not recommending it using unlike

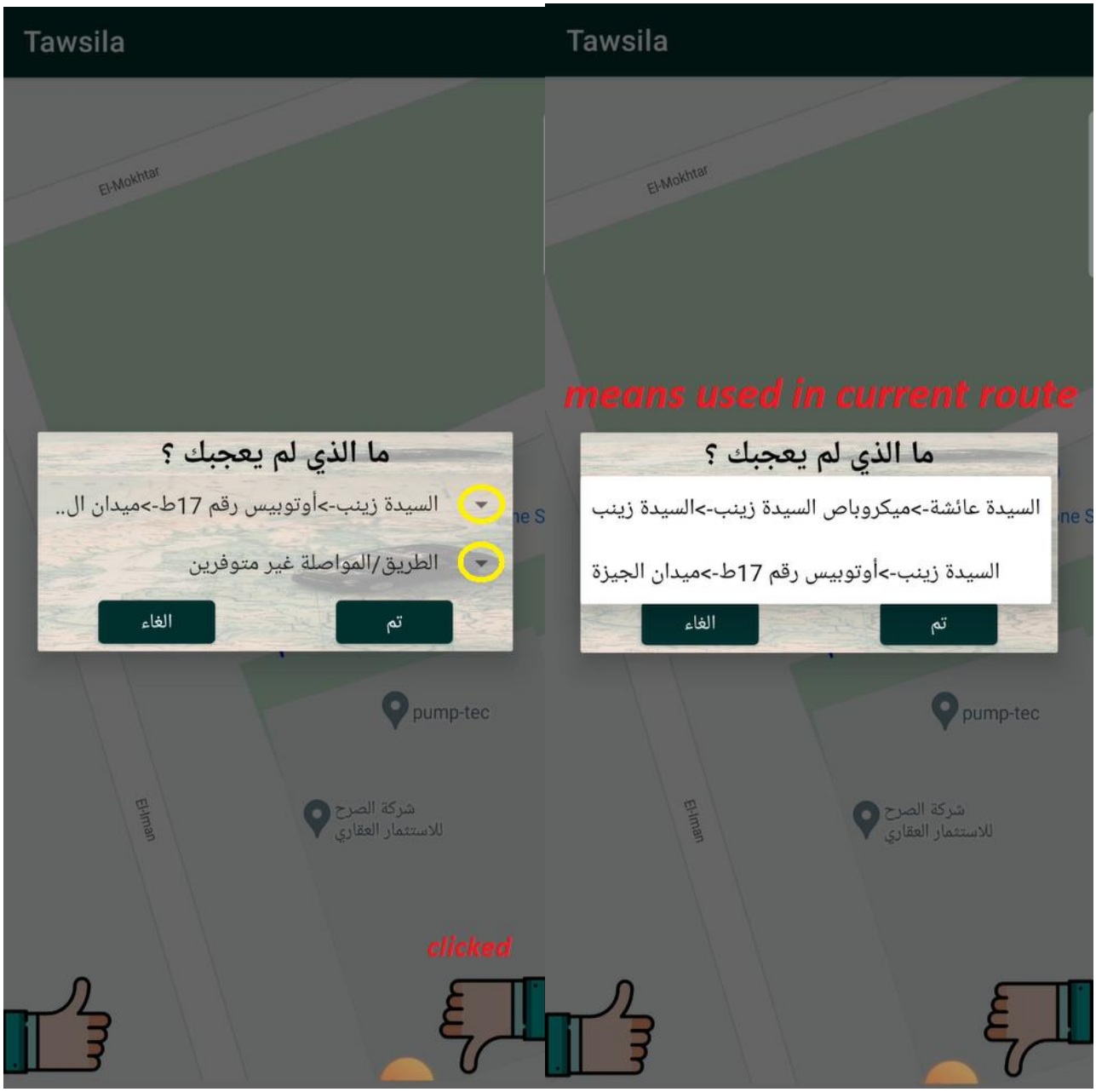


Figure [42]: User clicking dislike part 1

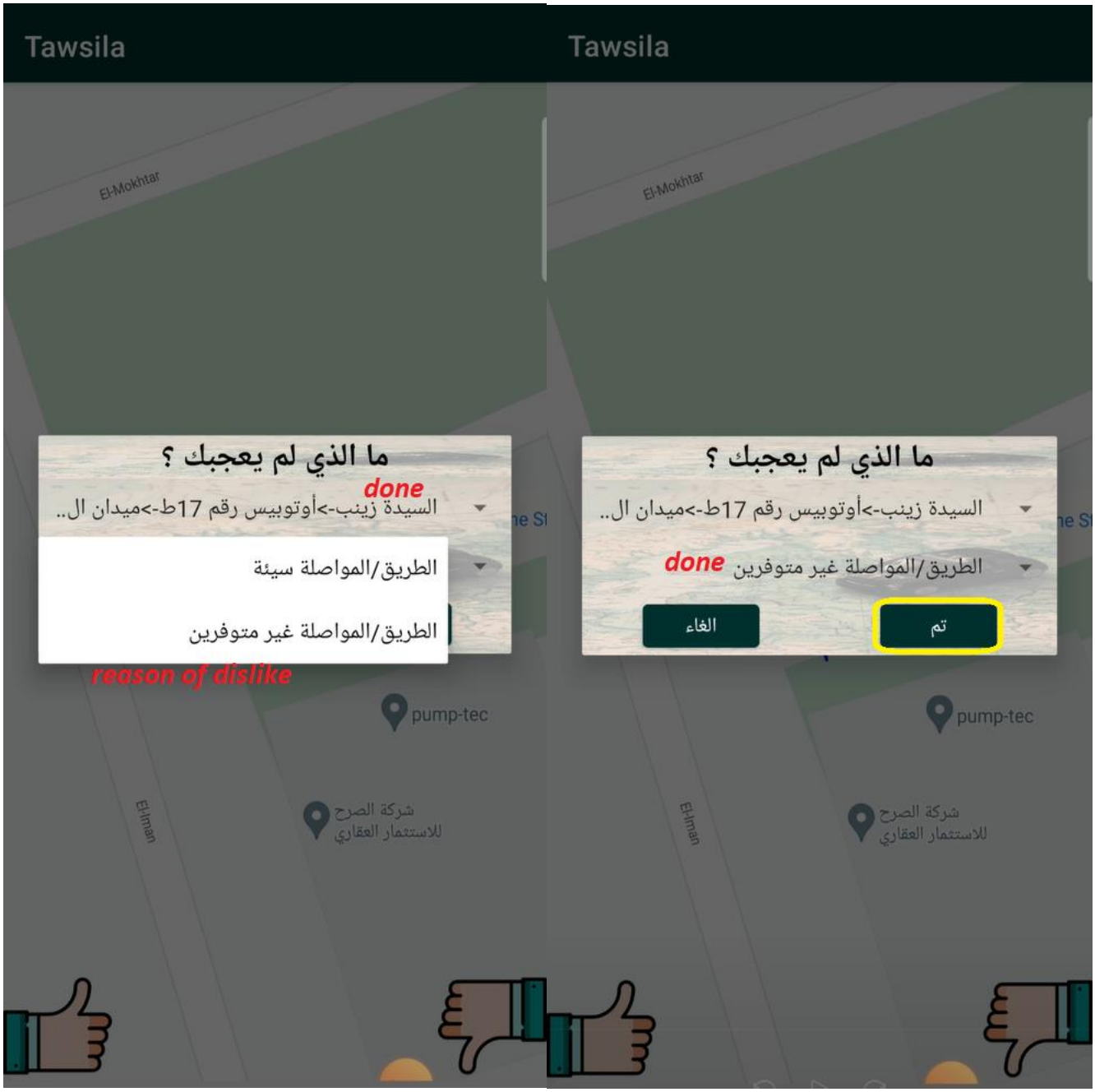


Figure [43]: User clicking dislike part 2

7. **Authentication required before suggesting a new mean between two points.**

Figure [44]: Signing up

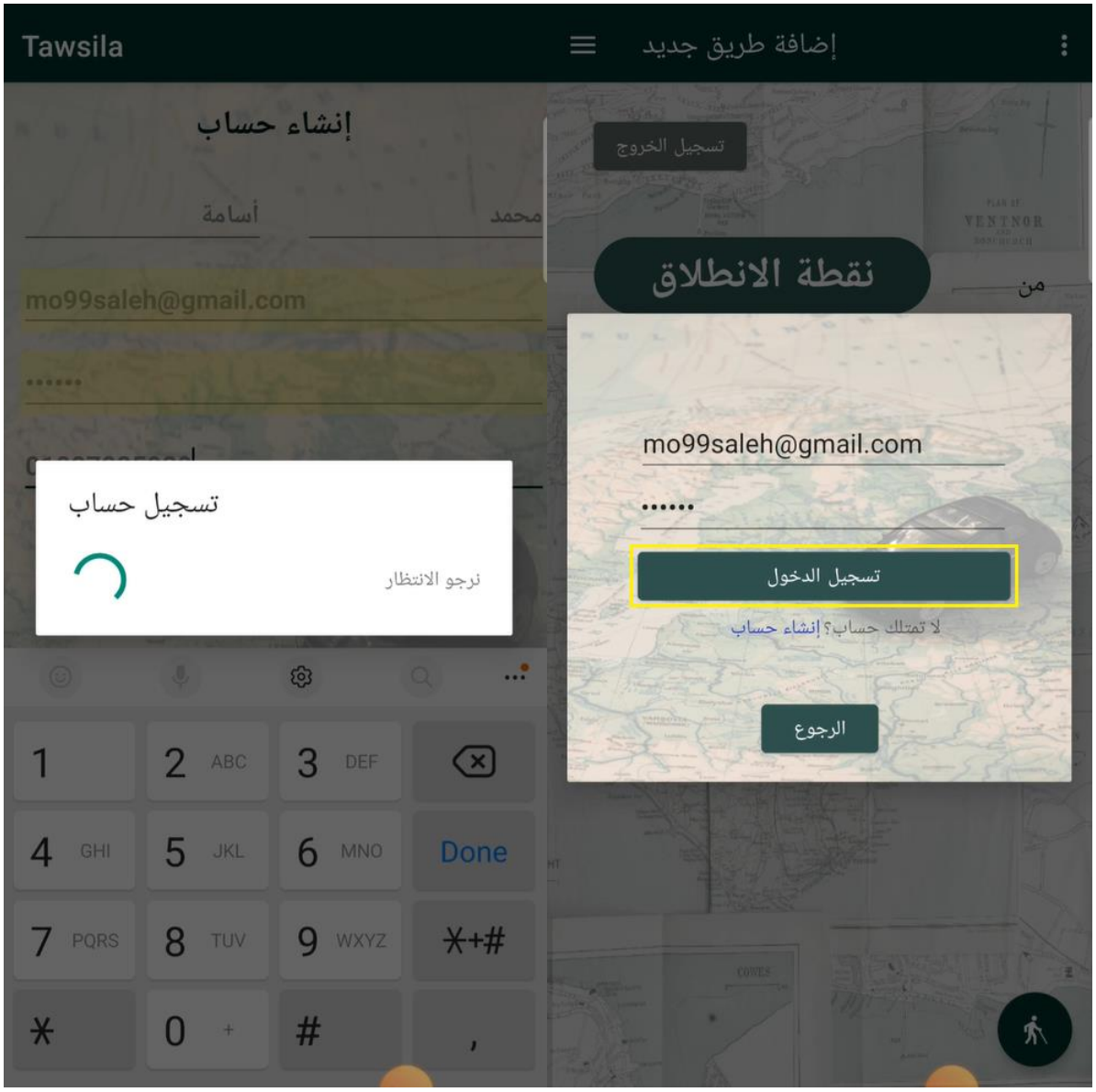


Figure [45]: Loging in

Future Work

- Completing the date base to include the 4 metro lines, all buses lines, and all micro buses.
- Supporting the English language to help more people of those who don't understand the Arabic language.

References

1. https://firebase.google.com/docs/reference
2. https://developers.google.com/maps/documentation
3. https://docs.github.com/en/rest?apiVersion=2022-11-28
4. https://neo4j.com/docs/
5. https://developer.android.com/training/data-storage/room
6. https://square.github.io/retrofit/
7. https://expressjs.com/en/5x/api.html
8. https://developer.android.com/reference/android/content/SharedPreferences
9. https://nodejs.org/api/