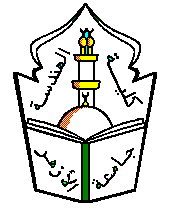
|  |
| --- |
|  |
| Carpooling API |
| Website Demo |

|  |  |
| --- | --- |
| |  | | --- | | **Team:**   1. Bassel Bakr Mohammed Sayed - (24) 2. Hossam El-Deen Faraj Doma - (25) |   **Project Supervisor:**  **Dr. Ayman El-Shenawy Elsefy** |

Computers and Systems Department

Faculty of Engineering

Al-Azhar University, Cairo.



# ABSTRACT

Carpooling refers to sharing a ride with other people in a car. By having more people share rides with each other, we can reduce the number of vehicles on the roads which in turn reduces traffic congestion and the need for more parking spaces. This reduction of number of vehicles, their fuel cost and the environmental problems they cause like air pollution and carbon emissions helps create safer environment for future generations.

# ACKNOWLEDGMENTS

First of all, I thank Allah for benefits he gave us and I would like to express our gratitude to

Project Supervisor: Dr. Ayman El-Shennawy

Professor/Lecturer at the Computer and Systems Engineering department

For his guidance, encouragement throughout our work

We are greatly indebted to our faculty and the whole administration staff.

Team :

1. Bassel Bakr Mohamed Sayed
2. Hossam El-Deen Faraj Doma

Table of Content

ABSTRACT **1**

ACKNOWLEDGMENTS **1**

TABLE OF CONTENT **2**

1. CHAPTER ONE: INTRODUCTION **5**

1.1 Project Motivation **5**

1.2 Project Aim and Objectives **5**

1.3 Types of Carpooling **5**

2. CHAPTER TWO: RELATED EXISTING SYSTEMS **6**

2.1 Introduction **6**

2.2 Existing Systems ……………………………………………………. **6**

2.3 Overall Problems of Existing Systems **7**

2.4 Overall Solution Approach **9**

3. CHAPTER THREE: SYSTEM REQUIREMENTS ENGINEERING AND ANALYSIS **11**

3.1 Introduction **11**

3.2 Feasibility Study ……………………………………………………. **11**

3.3 Targeted Users **12**

3.4 Functional Requirements **12**

3.5 Non-Functional Requirements **12**

4. CHAPTER FOUR: SYSTEM DESIGN **13**

4.1 Introduction **13**

4.2 UML Use Case Diagram **14**

4.3 System Use Case sequence **15**

4.4 Graphical User Interface (GUI) Design **16**

5. CHAPTER FIVE: SYSTEM IMPLEMENTATION **21**

5.1 Introduction **21**

5.2 Database Implementation **21**

5.3 Graphical User Interface Implementation **22**

5.4 Other Components Implementation 24

6. CHAPTER SIX: SYSTEM TESTING AND INSTALLATION **25**

6.1 Introduction **25**

6.2 Requirements Validation And Completeness **25**

6.3 Test Cases **26**

6.4 Project Software And Hardware Requirements **38**

6.5 Libraries / Dependencies Used **39**

7. CHAPTER SEVEN: CONCLUSION & FUTURE WORK …………... 40

7.1 Future Work **40**

7.2 CONCLUSION **40**

# 

This page is intentionally left blank

# **CHAPTER ONE: INTRODUCTION**

## 1.1 Project Motivation

We noticed that many companies/institutions who want private carpooling system for their own specified needs build it from scratch, though there are many common bases in each implementation. We are offering a unified model API that can respond to these queries no matter what the institutions needs are.

## 1.2 Project Aim and Objectives

Building a unified carpooling API system that responds to different client needs both fast and reliable.

## 1.3 Types of carpooling

**• Client first**

The drivers drive their cars like a cab and receive a request from a client to pick him up and drop him at his destination.

**• Driver first**

The driver has a predefined plan and time for his trips. Clients can view his plan and, the number of available seats, possibly with whom they will travel. The client then can request the driver to pick him up and drop him somewhere along his trip path.

• **Booking**

Compared to other 2 types listed above, this type is quite popular and often used when clients really need to arrive on time without delays

The client either requests someone to pick him up from source to destination or views available drivers with work schedule matching his request.

# CHAPTER TWO: RELATED EXISTING SYSTEMS

## 2.1 Introduction

There are plenty of online apps for carpooling, many of which consist of a server that connects drivers with passengers and client app, from which, the passengers can select one of the available rides for a relatively small cost compared to taking cap.

## 2.2 Existing Systems

* [Uber](https://blog.capterra.com/free-open-source-project-management-software/" \l "26)
* Careem
* Raye7
* Lyft

## 2.3 Overall Problems of Existing Systems

1. Each company has its own system, possibly built from scratch.
2. Distrust between drivers and passengers.
3. Inconsistency of service due to network fluctuation.

## 2.4 Comparison

The following table provides a comparison between the 2 most popular carpooling services in Egypt as of 2019:

|  | Uber | Careem |
| --- | --- | --- |
| Popularity | 483 cities worldwide. | 24 cities in the Middle East & Africa. |
| Registration | E-mail address + Mobile Number | E-mail, Mobile number or Facebook |
| Ride options | Economy & business cars, Bus | Economy, business cars, scooter, white taxi, water taxi, Bus |
| Rides availability | Improved availability with normal prices | Improved availability with normal prices |
| Request pickup time | Request it now or later with same charge for later rides | NOW or LATER, with extra charge for later rides |
| Pickup | Saves multiple addresses | Saves multiple addresses |
| Car request | Provides nearest rides & ETA | Provides nearest rides & ETA |
| Fare rate | Starting Fare: EGP 7, EGP 2.30 per km + EGP 1.6 per waiting minute, Minimum Fare = EGP 10 | Starting Fare: EGP 7, EGP 2.30 per km + EGP 1.6 per waiting minute, Minimum Fare = EGP 10 |
| Airport surcharge | EGP 25 | EGP 20 |
| Trip cancellation | Deducts EGP 10 after 2 minutes of ride cancelation | Deducts EGP 12 after 2 minutes of ride cancelation |
| Payment methods | Credit card or cash | Credit card, top up, cash or packages |
| Packages | No | Yes |
| Trip bill | Trip receipt is sent to client’s email with more details on the trip | Trip receipt is sent to client’s email with more details |
| Track your ride | Tracks your ride even when the app is closed | Provides tracking service but requires a lot of effort to retrieve after closing the app |
| Customer service | In app help service, with email communication | 24/7 call center plus in app help service, with email communication |
| Driver evaluation | Rating through the app | Rating through the app and SMS |
| Sharing ride Info | Shareable | Shareable |
| Splitting ride fare | Possible | Not possible |
| Trip history | Complete history of your trips, payments and driver details on the mobile app and website | Complete history of your trips, payments and driver details on the mobile app and website |

## 2.5 OVERALL Solution Approach

* Provide one universal API system that get the same job done.
* Can be used by any institution/company to their liking.
* More reliable/trusted system because it is open to developers/contributors.

## 2.6 Proposed solutions

This project tries to address possible solutions for some of the aforementioned problems:

1. Creating a carpooling service that can be used by any company or individual for an affordable price and reasonable plans.
2. With the separation of service from the usual flow, companies can focus more on hiring trusted drivers for their business or delegate recruitment process to professional agencies. While at the same time, the work on improving carpooling efficiency will be the focus of the server maintainers and reflected for all users.
3. With carpooling service being shared with multiple companies, it becomes much easier to collect useful data about most frequent places passengers often visit and make charts and heat maps for drivers to give them an idea about which roads are more profitable and how frequent some rides are.
4. Most frequent rides can be cached by the service and retrieved much faster in subsequent queries improving the overall performance and, in some cases, the roads can be stored locally on client’s device, reducing both server load and client’s need to ping the server frequently, especially when the network is poor.

# CHAPTER THREE: SYSTEM REQUIREMENTS ENGINEERING AND ANALYSIS

## 3.1 Introduction

The software requirements engineering and analysis describe features and functionalities of target system include this processes:

1. Feasibility Study
2. Requirement Gathering
3. Software Requirement Specification
4. Software Requirement Validation

## 3.2 Feasibility Study

Study if system with its requirement is applicable or not.

**System needs:**

* Server
* Interface Designer
* Sign in method
* Edit database
* API calls

\*All requirements are available

## 3.3 Targeted Users

Companies and Institutions that look for a private, fast, reliable carpooling system.

## 3.4 Functional Requirements

Requirements which are related to functional aspect of software

1. Registering
2. Signing in
3. Making API queries
4. Receiving query responses

## 3.5 Non Functional Requirements

Requirements, which are not related to functional aspect of software. They are implicit or expected characteristics of software

1. Logging
2. Storage
3. Performance
4. Cost
5. Flexibility
6. Accessibility

# 4.0 CHAPTER FOUR: SYSTEM DESIGN

## 4.1 Introduction

It describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudo code and other documentation. The most creative and challenges phase of the software development life cycle is software design. The term design describes final software and the process by which it is developed. The purpose of the design phase is to plan a solution of the problem specified by the requirements document. It also includes the construction of programs and program testing. Design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affecting the quality of the software; it has a major impact on the later phase, particularly testing and maintenance. The output of this phase is the design document

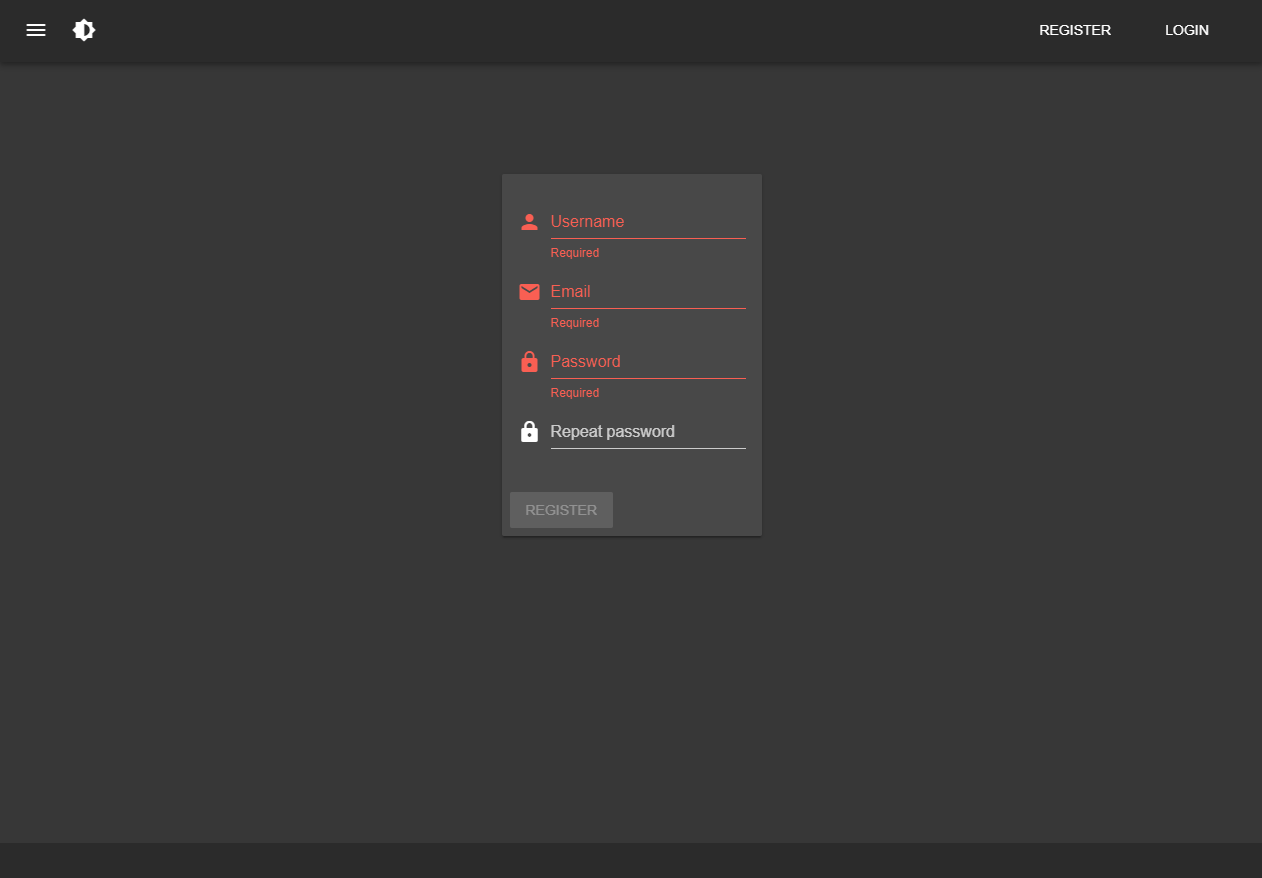
## C:\Users\Conan\Downloads\_Blank UML.pngUML Use Case DiagraM

## System Use case

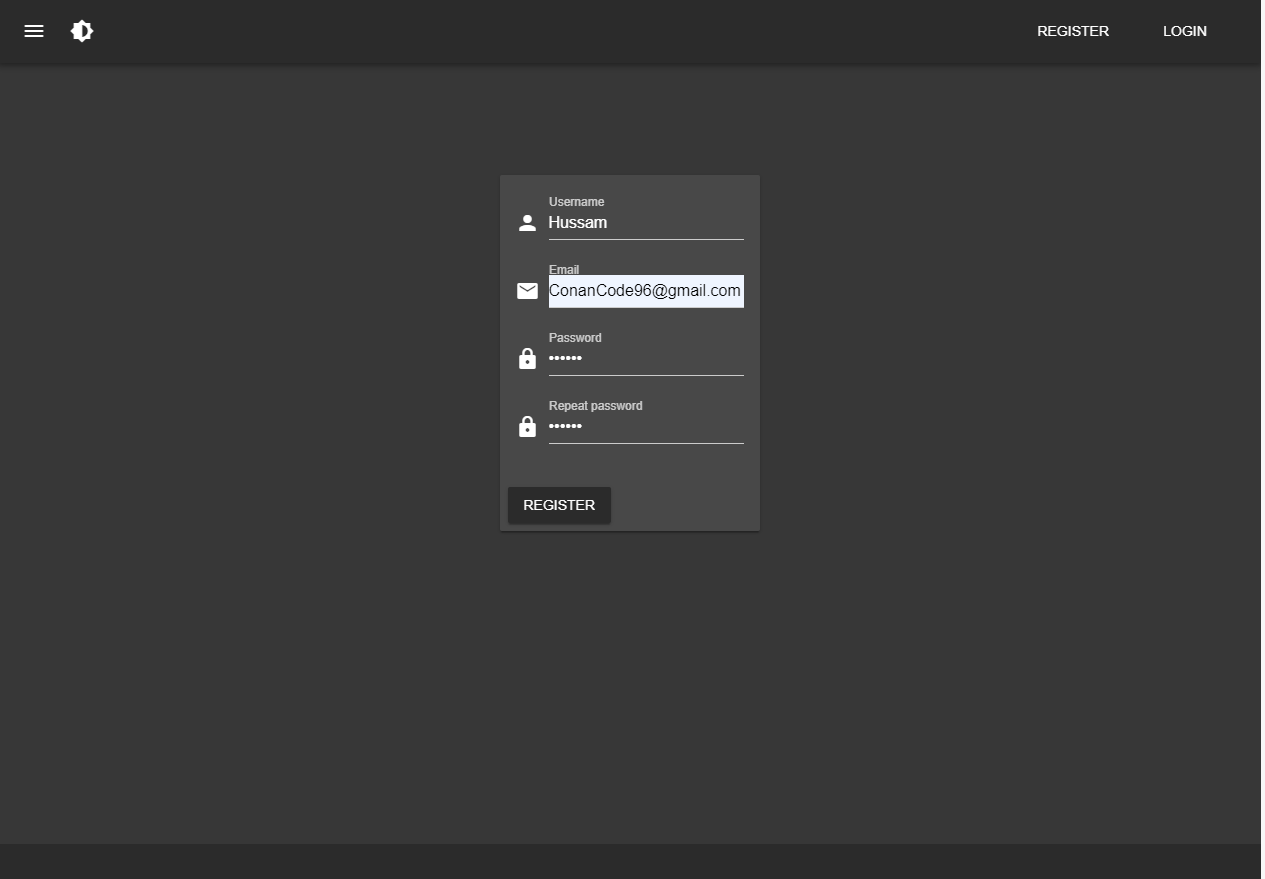
* The Institution (called user) should register on the website for an API key to use in its subsequent queries.
* The user logins to their account.
* The user can add driver and update their locations
* The user can embed their own API key in their application designed for end-users.
* The end-users (called passengers) can request to be picked up from a specific point on the map.
* The end-users would then call the API with the following:
  + valid start location on the map
  + valid destination location on the map
  + the company’s embedded API key
  + the least point in time a driver has been seen (in seconds since EPOCH)
* The API server will then respond to the request to update the driver’s locations on the map.
* The API server will search within a specific box on the map for drivers who have been seen not later than the given time parameter the end-user added to their request.
* The algorithm running of the API server will respond with the nearest driver on the shortest way from the start to the end destination.
* The driver should get notified that they are to serve this end-user(passenger).

## 4.4 Graphical User Interface (GUI) Design

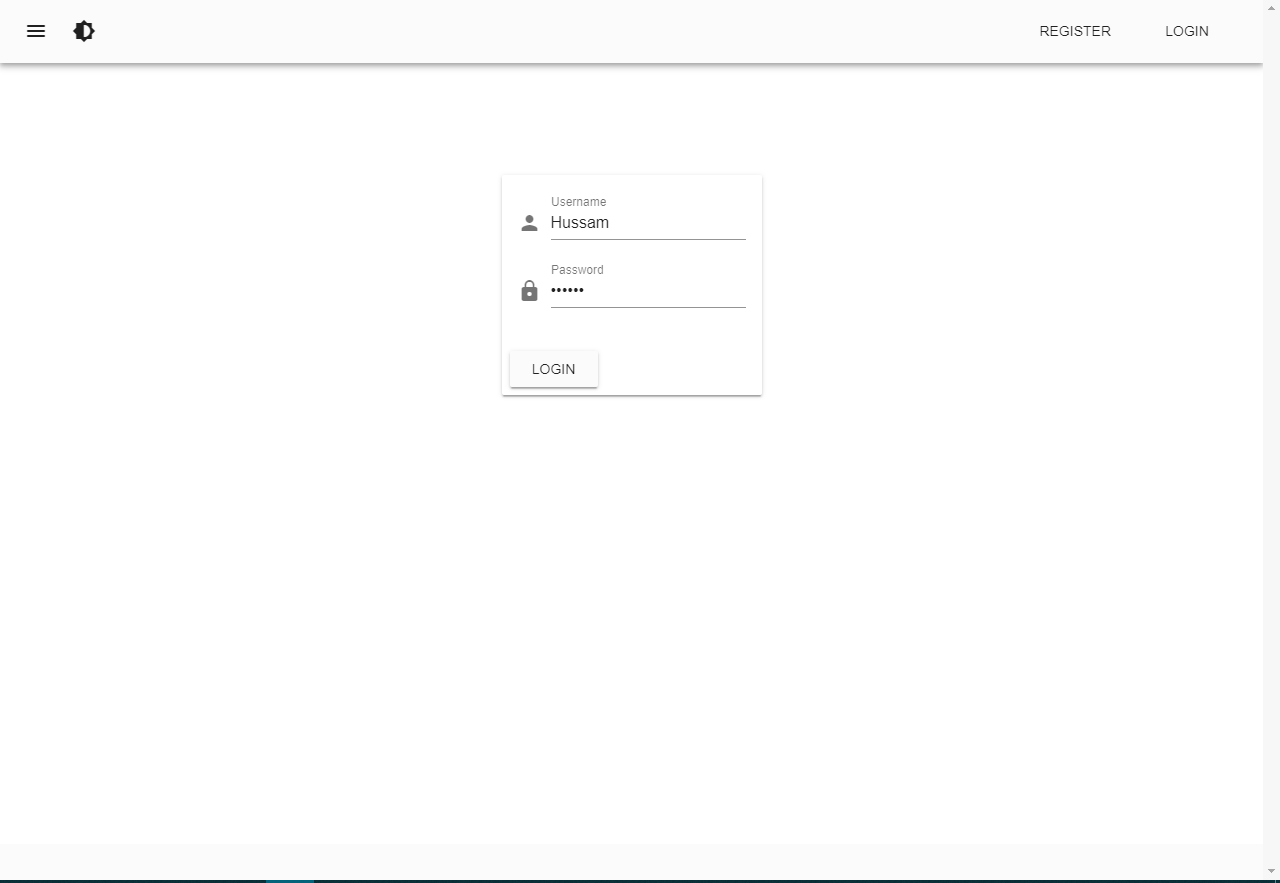
1. **Select to Sign up or login**



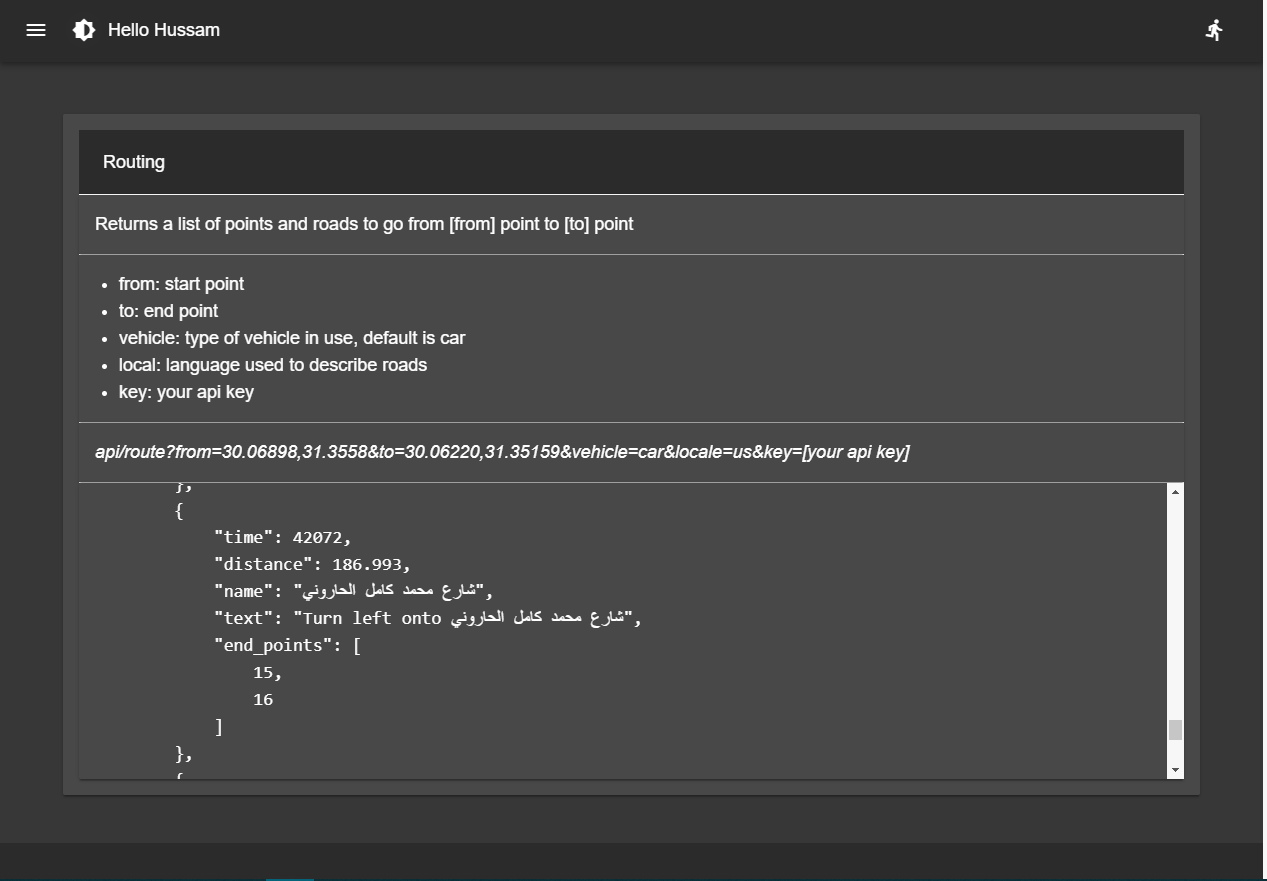
1. **Sign up**

****

1. Login



1. Querying the API



# CHAPTER FIVE: SYSTEM IMPLEMENTATION

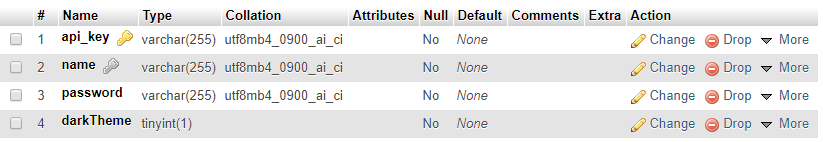
## 5.1 Introduction

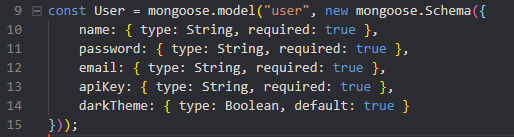
Implementation of System is divided to

1. Database
2. GUI
3. API
4. APP Development

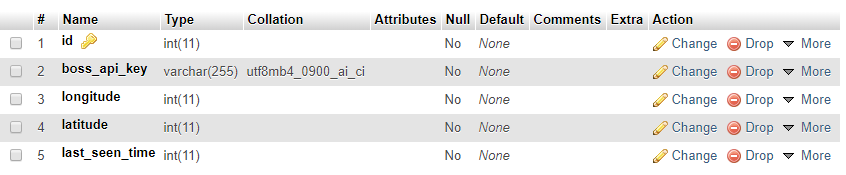
## 5.2 Database Implementation

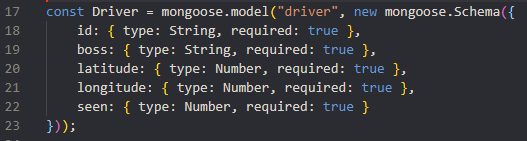
**User Data Dictionary**

****

****

**Driver Data Dictionary**





## 5.3 Implementation

The implementation can be broken into 3 main parts:

* **Server Side:**

Where the service resides and operates.

It has to support multiple concurrent requests without blocking users as much as possible. Node.JS fits in perfectly owing to its asynchronous nature and the simplicity of scaling the server to respond to heavier work loads

It’s also possible to delegate performance critical parts to C/C++ as Node.JS has bindings for both of them.

It’s also worth mentioning that using Node.JS unifies the languages used in both back-end and front-end (Javascript) reducing the problems often encountered when someone works on different code bases written in different languages and make the mistake of writing correct code yet not in correct language.

The server will offer API end points, through which the client app can interact and exchange location dat.

To support multiple companies/individuals, API access will be restricted to registered members using a token and password. When a user registers and fills the forms and billing information, he will be given a unique token id that helps the API identifies him and plans to choose from. Subscribing to any plan will allow him to use the API and send requests per the guide lines of each plan.

When subscription ends, the user can choose to auto-renew his plan or to renew it himself

The server only offers carpooling service; the API user is the one who customizes the service to his needs. For example: a user might choose to hire only female drivers and accept rides for female passengers and their young children.

* **Client side:**

The front-facing app that interacts with the server behind the scenes through web APIs.

There are many options to choose from, like Ionic, React & Vue Native and Flutter.

Flutter and React & Vue Native provide better performance compared to Ionic, especially that React uses virtual DOM and avoids editing the real DOM unnecessarily. But Ionic beats them in terms of supported platforms as they only support mobile applications while Ionic is mature enough and supports Web + Mobile applications using one code base

The performance issues addressed above regarding Ionic aren’t much of a concern because most of the work is server side and the client only needs to display the data relied by the server in a convenient way.

* **Registration website:**

The website that users can use to register to our service and acquire a token for accessing the API end points

The design will be made with Vue.JS for its component based system and styled with Bulma CSS framework as it’s based on flex box, a display mode well suited for responsive designs

# CHAPTER SIX: SYSTEM TESTING AND INSTALLATION

## 6.1 Introduction

Software testing against requirements and system specification.

**Include:**

* 1. Software validation
  2. Software verification

Database tested at MongoDB/MySQL, POSTMAN

## 6.2 Requirements Validation and Completeness

## **completed Requirement:**

1. Registering

2. logging in

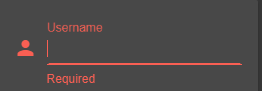
3. Calling the API

## **Uncompleted Requirement:**

* 1. Generating reports
  2. Generating Heatmap

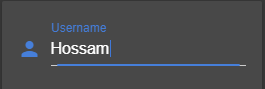
## Test CASES

1. REGISTERING TESTS:

****

**--Null Username Field--**

|  |  |
| --- | --- |
| Test Name | Username Not Null |
| Function Tested | Register() |
| Sample Input | ‘null’ or empty string |
| Expected Output | Warning that the field is null |

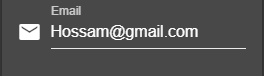
****

**--When The Field Is Non-Empty--**

|  |  |
| --- | --- |
| Test Name | Email Checking |
| Function Tested | Register() |
| Sample Input | ‘Your\_mail@Mailbox.Com’ |
| Expected Output | Ok! If Email Matches The Regex, Otherwise No. |

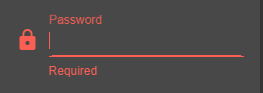
****

**--Email Not Valid--**

****

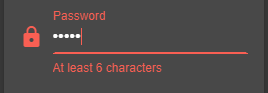
**--Email Conforms to The Email Regex--**

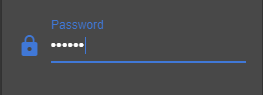
|  |  |
| --- | --- |
| Test Name | Password not null |
| Function Tested | Password field |
| Sample Input | Password not empty / null |
| Expected Output | Warning! Cannot proceed while the password is null |

****

**--Null Password Field--**

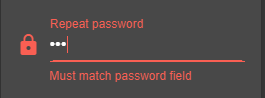
|  |  |
| --- | --- |
| Test Name | Password Not less than 6 characters |
| Function Tested | Password Field |
| Sample Input | Length of password field >= 6 characters |
| Expected Output | Ok! If password length is >= 6 characters Otherwise a warning is displayed |

****

 **--Password Length Less Than 6 Characters—**

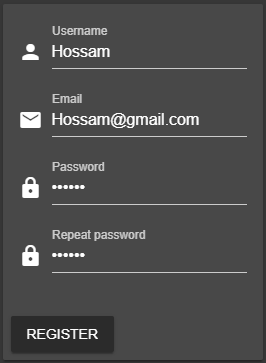
**-- Password Valid--**

|  |  |
| --- | --- |
| Test Name | Repeat Password Not different |
| Function Tested | Register(); Repeating the password should be exactly the same as the original password |
| Sample Input | Any password that matches the originally entered password while registering |
| Expected Output | Ok! If Passwords match, Otherwise a Warning is raised. |



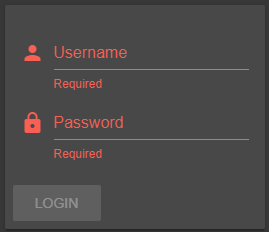
**-- Password does not match--**

|  |  |
| --- | --- |
| Test Name | Valid Registering |
| Function Tested | Register() |
| Sample Input | Valid registered username , email, and password. |
| Expected Output | ‘’User created successfully’’! If the username, email and password are valid, and haven’t existed in the database before. |

 --successful registration--

2. Login tests

|  |  |
| --- | --- |
| Test Name | Logging in |
| Function Tested | Login() |
| Sample Input | Valid registered username and password pair |
| Expected Output | Ok! If both the username and password pair Match a record in the database, Otherwise a Warning is raised. |

****

3. Database tests

|  |  |
| --- | --- |
| Test Name | User Not Found |
| Function Tested | Login() |
| Sample Input | Trying to login with invalid credentials |
| Expected Output | ‘User Not Found’ error is raised. |

|  |  |
| --- | --- |
| Test Name | User Already Exists |
| Function Tested | Register() |
| Sample Input | Trying to register with a username used by another user. |
| Expected Output | ‘User Already Exists’ error is raised. |

|  |  |
| --- | --- |
| Test Name | Email Already Exists |
| Function Tested | Register() |
| Sample Input | Trying to register with an email used by another user. |
| Expected Output | ‘Email Already Exists’ error is raised. |

|  |  |
| --- | --- |
| Test Name | Creating user error |
| Function Tested | Register() |
| Sample Input | Trying to register with an valid username, email and password, but connection to the database is lost. |
| Expected Output | ‘Failed to create user’ error is raised. |

|  |  |
| --- | --- |
| Test Name | Database connection error |
| Function Tested | db.connect() |
| Sample Input | Trying to connect with an invalid username, password to the database or a network error happened. |
| Expected Output | ‘Failed to connect’ error is raised. |

4. API Calls tests

|  |  |
| --- | --- |
| Test Name | API Call Without Authentication |
| Function Tested | getRoute() function / or any API endpoint |
| Sample Input | Making a query request on an API endpoint without being authenticated |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Invalid API Call |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without following the API call schema |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | API Key not given |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing any API Key |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Incorrect API Key |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing a valid API Key |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Invalid Location |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing a valid Location(longitude / latitude) |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Invalid Time Parameter |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing a valid Time parameter in second since EPOCH. |
| Expected Output | Query rejected! and the request is not served. “Invalid date/time, use seconds since epoch” warning is raised. |

|  |  |
| --- | --- |
| Test Name | Missing ID Parameter |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing any Diver\_ID |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Invalid ID Parameter |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint without providing a valid Diver\_ID |
| Expected Output | Query rejected! and the request is not served |

|  |  |
| --- | --- |
| Test Name | Request Limit Reached |
| Function Tested | Any API endpoint |
| Sample Input | Making a query request on an API endpoint after the user quota has reached it end |
| Expected Output | Query rejected! and the request is not served |

## 6.4 Project Software and Hardware Requirements

## **Hardware Requirement:**

* I5 Processor Based Computer
* 4GB-RAM
* 80 GB Hard Disk
* Monitor
* Internet Connection

## **Software Requirement:**

* Any Windows 7 or higher / Unix / Linux OS
* VSCode
* Web Browser
* POST MAN

**Front End:** Vue.js

**Backend:** Node.js

**Database:** MySQL/MongoDB

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6.5 Libraries used  |  |  | | --- | --- | | Library | Stable version | | Axios | 0.19.0 | | Bcrypt | 3.0.6 | | Body-parser | 1.19.0 | | Cookie-parser | 1.4.4 | | Express.js | 4.17.1 | | Graphhopper-js-api-client | 0.11.1 | | Mongoose | 5.6.1 | | MySQL | 2.17.1 | | Passport | 0.4.0 | | Passport-local | 1.0.0 | | Uuid | 3.3.2 | | Vue | 2.5.22 | | Nodejs | 10.16.0 | |

Database >> MongoDB / MySQL

Server >> NodeJS (localhost@3000)

Website >> VueJS (localhost@8000)

# 7.0 CHAPTER SEVEN: PROJECT CONCLUSION AND FUTURE WORK

7.1 Future Work:

1. Generate reports about API traffic
2. Adding extra functionality of heatmaps

7.2 CONCLUSION

App consist of TWO main stages:

1. Frontend

Implemented in Vue.JS using 3 screens (home, register, login).

Website is responsive for different Screen sizes from small mobiles to large tables and in 2 language Arabic and English depend on mobile language.

1. Backend

Implemented in Node.JS, handles authentication, encryption, decryption of credentials, calling Graph Hopper APIs, implementing the matching algorithm, fetching and putting data from and to the frontend GUI.

THE END.