
# Car Rental System

Data Engineering 1
Advanced Databases – Project Report (Team Diamonds)
**UNDER THE GUIDANCE OF**
Professor Frank Hefter

Prepared by:

Team Diamond

Authors:


| | |
|---|---|
| Roshan Scaria | 11018152 |
| Gaurav Khurana | 11018120 |
| Monica Suresh | 11018066 |
| Jerin Joshy | 11018153 |
| Ahmad Saud Azmi | 11018047 |

# Contents

7. API
    7.1. Foreign API description
        7.1.1Example data (show 1-2 records from the api)
8. Evaluation
9. References
    9.1. Books
    9.2. Webpages (Video, Tools, Documentation , ... )

# 1. Introduction

The car rental system is an online and offline platform where cars are rented with or without a chauffeur, serving as a market ground for customers who are looking to rent a vehicle. The Car rental system operates throughout Germany. The system comprises a variety of features such as a Master database management system, tracking of cabs, Nearest service center, etc., The Web application has tie-ups with private hotels and trains, thus allowing booking of these through the web app. It also includes a loyalty module where customers are provided discounts. The project 'Car Rental System' is based on the following three databases: MongoDB, Neo4j, and Redis.

# 2. Organization

## 2.1. Roles and responsibilities, use case

| Tasks | Team Members | | | | |
|---|---|---|---|---|---|
| | Gaurav | Jerin | Monica | Roshan | Saud |
| Project analysis | R | R | R | R | R |
| Requirement gathering | RA | RA | RA | R | R |
| Selection of database | R | R | RA | R | R |
| Data setup | R | R | R | R | R |
| Database setup | RA | R | R | RA | R |
| UML diagram | R | R | R | R | R |
| Data flow Diagram | R | RA | R | R | RA |
| Report Documentation | RA | RA | RA | RA | RA |
| User Stories | R | R | R | R | R |
| MongoDB | | D | | | D |
| Neo4j | | D | D | D | |
| Redis | D | | | | |
| Car rental Tie-Ups | | | RDO | | |
| Admin Panel in Car Rental | | | | | RDO |
| Booking Simulation | RDO | | | | |
| Shortest path between service points | | | | RDO | |
| Service Point finder, Car availability | | RDO | | | |

Legend: Responsible = R; Developing = D; Assisting = A; Documentation = O

Table 1: Roles and Responsibilities

## 2.2. Meetings (Date, Topic, Outcome, Attendees)

| Date | Discussion on | Attendees |
|---|---|---|
| 28.06.2022 | Choice of topic for the project. | Gaurav, Jerin, Monica, Roshan, Saud |
| 29.06.2022 | Decided on use cases for the project and brainstormed ideas for every use case. | Gaurav, Jerin, Monica, Roshan, Saud |
| 01.07.2022 | Distributed use cases between each other and brainstormed for using databases over the designated use cases. | Gaurav, Jerin, Monica, Roshan, Saud |
| 05.07.2022 | Distributed use cases between each other and brainstormed for using databases over the designated use cases. | Gaurav, Jerin, Monica, Roshan, Saud |
| 08.07.2022 | Reflected on the meeting with professor and worked on finding solutions for the problems mentioned in the meeting over the prepared use cases. | Gaurav, Jerin, Monica, Roshan, Saud, Professor Frank Hefter |
| 09.07.2022 | Finalised User Stories and decided which databases to use for which use case. | Gaurav, Jerin, Monica, Roshan, Saud |
| 11.07.2022 | Discussed what we found for our use cases with respect to databases | Gaurav, Jerin, Monica, Roshan, Saud |
| 13.07.2022 | Discussed how to work on the Report and finalised use cases | Gaurav, Jerin, Monica, Roshan, Saud |
| 18.07.2022 | Prepared initial draft of the document | Gaurav, Jerin, Monica, Roshan, Saud, Professor Frank Hefter |
| 21.07.2022 | Prepared final draft of Report to be cleared with professor in the next meeting | Gaurav, Jerin, Monica, Roshan, Saud |

## 2.3. Tools used

*1. Lucid Chart for UML diagrams*

*2. VS Code - Integrated development environment*

*3. PyCharm*

*4. Neo4j Desktop*

*5. Docker*
*6. MongoDB-COMPASS*
*7. MongoDB Atlas*
*8. Visual Paradigm*
*9. TkInter*

# 3. User Stories

## 3. User Stories:

- **Car Booking Simulation:**

- As a user, I want to see the list of cars that are available to be rented near me in a given radius that I have the ability to input as well as the ability to choose the destination. I can choose the type of car that I want to drive along with the option of being driven by a chauffeur to my destination. I can also choose the chauffer from a list of chauffeurs by deciding upon either the nearest chauffer from the car or the highest rated chauffer in a particular radius of the car.

- **Customer Management:**

- As an Admin, I would like to add new customers to the company's database, as well as update, modify, and delete customer information.
- In addition to that I would like to find the customers details based on the Customer ID.

- **Car Management:**

- As an Admin, I would like to update the cars details to the database, add, modify, and delete car information.
- In addition to that I would like to know which cars are available for bookings.
- As a user, I would like to know the health status of the car based on the last Maintenance date.
- I would like to get the last service details of the car and what repair works where done.
- As an Admin I would like to know the car service centre near my radius.

- **Booking Management:**

- As an Admin, I would like to search the previous booking details for the drivers, customers, or cars.
- In addition to that I would like to make reservation for new bookings and update the booking details in case of changing the details. example: drop locations.

- **Driver Management:**

- As an Admin, I would like to add, delete, update, and create new entries for the drivers.
- I would also like to have the aggregated list of drivers with multiple ride details.

- **Promo-code and Discounts Management:**

- As an Admin, I would like to give Promocodes/discounts for the next ride to the customer based on their rating.
- As an Admin, I would like to get the list of drivers who needs to bonus based on their rating.

- **Reports Generation:**

- As an Admin, I would like to generate stats and reports graphically to compare it with the predefined target and make better decision.

- **Shortest Path Between the service points:**

- The manager of the company is planning to visit the service centres to ensure the smooth functioning, he wants to ensure that the transportation between the service points is smooth and obstructions in one of the paths doesn't affect the business productivity.
- He wants to promote a service center to a main office by analysing the service centres which is having the highest number of connections.
- The manager wants to find out the shortest distance between the main office to the service center that is the having the least connections. He also wants to calculate the possible distance to the rest of the service centres from the service center with least connections.

- **Creation of promotional offer based on customer preferences**

- Based on the preferences of the customer a promotional offer is created for a product category. From the list of cars with the same type, that a customer has viewed or wish listed, a promotional offer is issued to those customers.

- **User Stories: Service Point-Car availability**

- As a service point manager, when a customer comes in for booking a car, we need to check if car is available for booking or is it in a run.
- As a service point manager, if the cars already fully booked, we need to suggest the user a nearby service point where they can rent a car.
- As a service point manager, if the cars are not available at the nearby locations, company should consider placing a new service point.

- **Tie-ups with restaurants and tourist organizations:**

- As a tourist, Jonas wants recommendations of restaurants, castles, or museums to visit near his drop-off place and recommendation of hotels booked by similar customers.

- Admin - In a scheme to attract more drivers for the car rental service, the platform has assured to cover the car insurances of drivers that will enrol with the company. The admin needs to provide a liability assessment for the cost bore by this scheme.

- User - As a promotion scheme, the rental platform has tied up with PlixTrain, so any new customer doing a booking through the platform will get additional discounts.

# 4. Details

## 4.1. Admin Panel (Ahmad Saud Azmi)

### 4.1.1. User Story

- **Customer Management:**
  - As an Admin, I would like to add new customers to the company's database, as well as update, modify, and delete customer information.
  - In addition to that I would like to find the customers details based on the Customer ID.

- **Car Management:**
  - As an Admin, I would like to update the cars details to the database, add, modify, and delete car information.
  - In addition to that I would like to know which cars are available for bookings.
  - As a user, I would like to know the health status of the car based on the last Maintenance date.
  - I would like to get the last service details of the car and what repair works where done.
  - As an Admin I would like to know the car service centre near my radius.

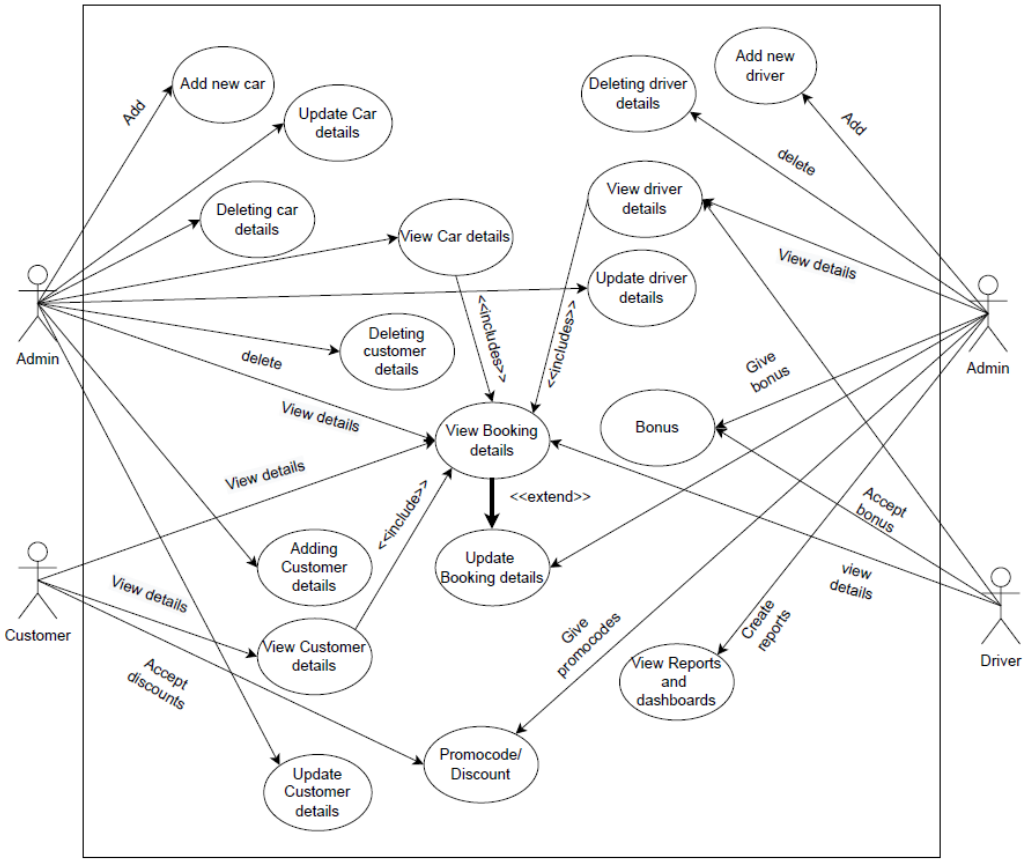
- **Booking Management:**
  - As an Admin, I would like to search the previous booking details for the drivers, customers, or cars.
  - In addition to that I would like to make reservation for new bookings and update the booking details in case of changing the details. example: drop locations.

- **Driver Management:**
  - As an Admin, I would like to add, delete, update, and create new entries for the drivers.
  - I would also like to have the aggregated list of drivers with multiple ride details.

- **Promo-code and Discounts Management:**
  - As an Admin, I would like to give Promocodes/discounts for the next ride to the customer based on their rating.
  - As an Admin, I would like to get the list of drivers who needs to bonus based on their rating.

- **Reports Generation:**
  - As an Admin, I would like to generate stats and reports graphically to compare it with the predefined target and make better decision.

### 4.1.2. Identified Use Case




### 4.1.3. Actors

Admin: Admin can add, modify, delete, and view customer details, car details, driver details and booking details. Admin can also give promo-code and discounts and can create reports as well.

Customer: Customer can only view the customer details, booking details and promo-code or discounts.

Driver: Driver can only view the customer details, driver details, booking details and bonus related details.

### 4.1.4. Descriptions of task with all interactions between actor and database.

**Customer Management:** Admin uses “Customer UI” to create, delete, update and view the customer details. At the backend of this UI, we are adding the Customer based on the last Customer ID and generating the latest customer id through the logic. In the UI suppose we want to update the details; we have used the find and update function to update the details.

Similarly we have used the delete function to delete the details based on the customer id and these all things are happening at the backend of the UI.

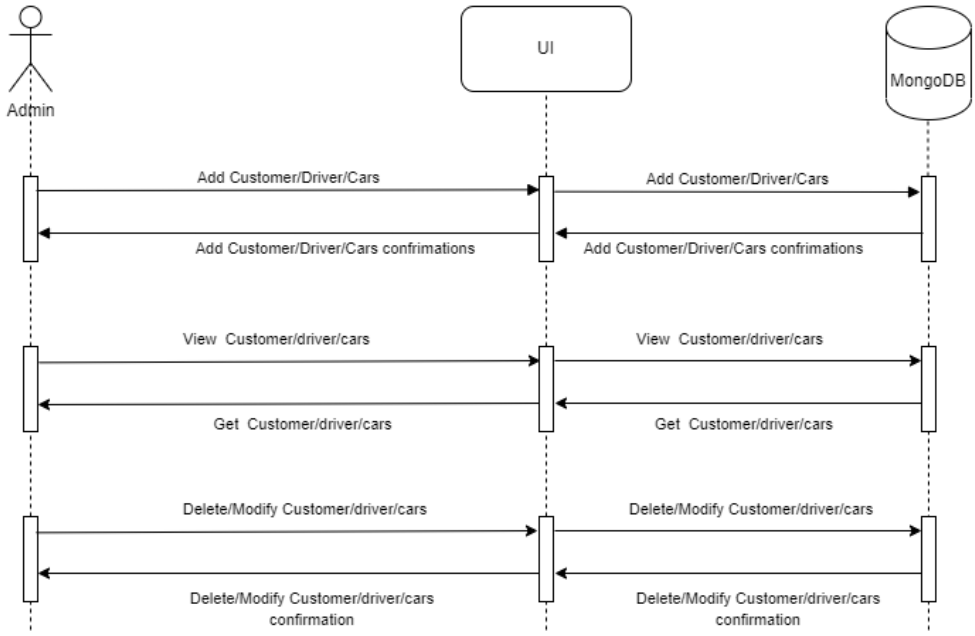
**Car Management:** For Car Management we have created another UI using the CAR collection in which we are performing the operations of adding, modifying, deleting, and viewing the Car details. Admin can use the find function to get the list of cars available from the Car collections. Using the logic and basis of the last maintenance date we can identify the status of the car using the aggregation operations and some calculation conditions. We can also get the last service details of the car while integrating the “Car” and “Service centre” collection. For finding the nearby Car service centre we have used the MongoDB's geospatial allows to efficiently execute spatial queries on a collection that contains geospatial shapes and points. We have used the concept of $geoIntersects to determine the admin neighbourhood, $geoWithin to find the number of car service centre in that area and $nearSphere to find it within a particular distance.

**Driver Management:** Using the similar UI, admin performs the basic operation of creating a new driver detail, updating the old details, and deleting the details once driver leaves the organization. These all operation are performed using the CRUD operations in mongo and list logic of python at the backend. Using the aggregated function admin can get the list of drivers with multiple rides details using the driver collection.

**Booking Management:** In booking management admin uses the booking collections to identify the booking details based on the customer id, driver id and booking id. In booking management admin can make reservation and make changes in the ride details if customer wants to update the drop off or pickup location.

**Promo-code and Discounts Management:** Using all the collections we are giving the promo-code/discounts to customer based on their rating and similarly we are giving the bonuses to the driver based on their average rating.

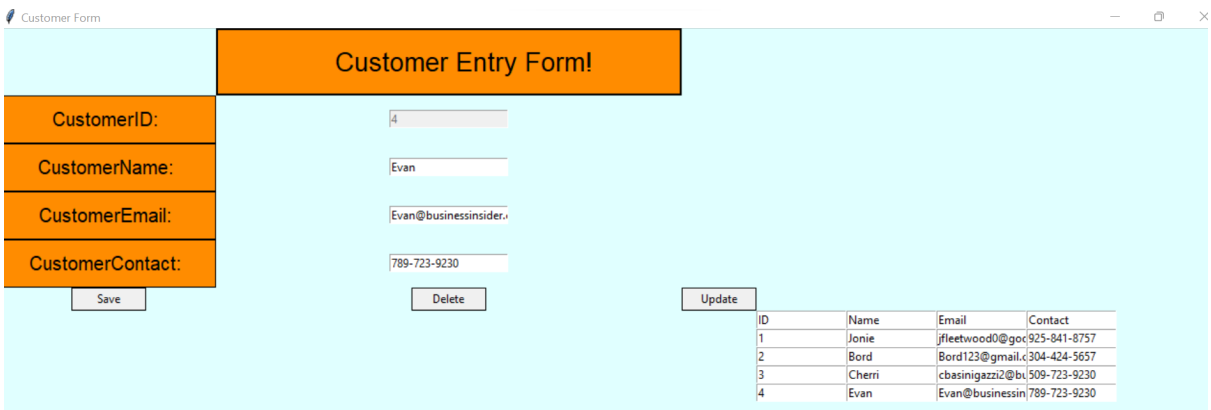
**Reports Generation:** Another task of admin is to generate the reports or view the dashboards to compare it with the predefined target and make better decision. For generating reports, we have used the sparks and its visualization tools.



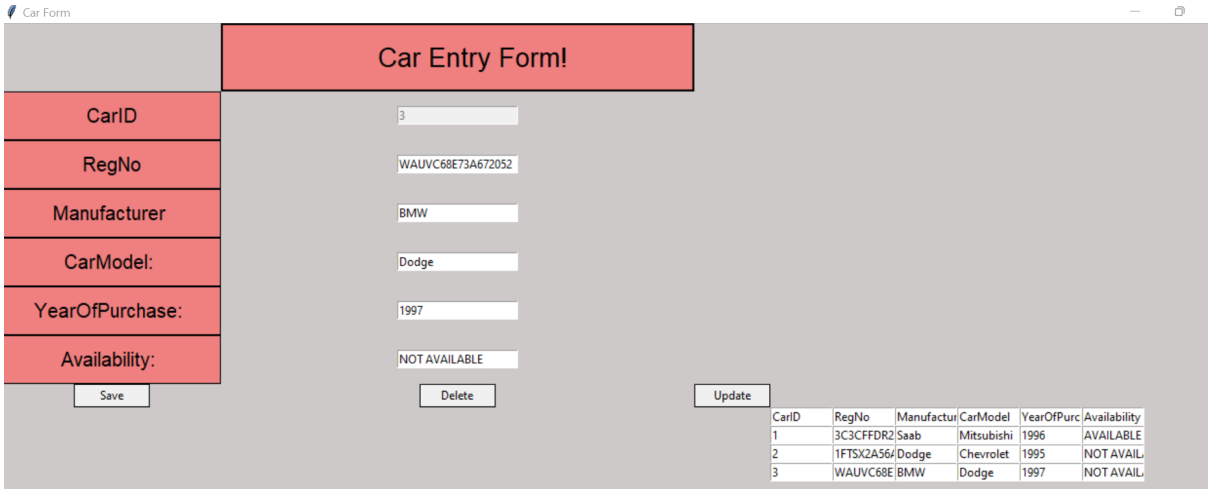

### 4.1.5. Optional: Frontend used / Command lines to reproduce execution

Frond end used is Tkinter, Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. It is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

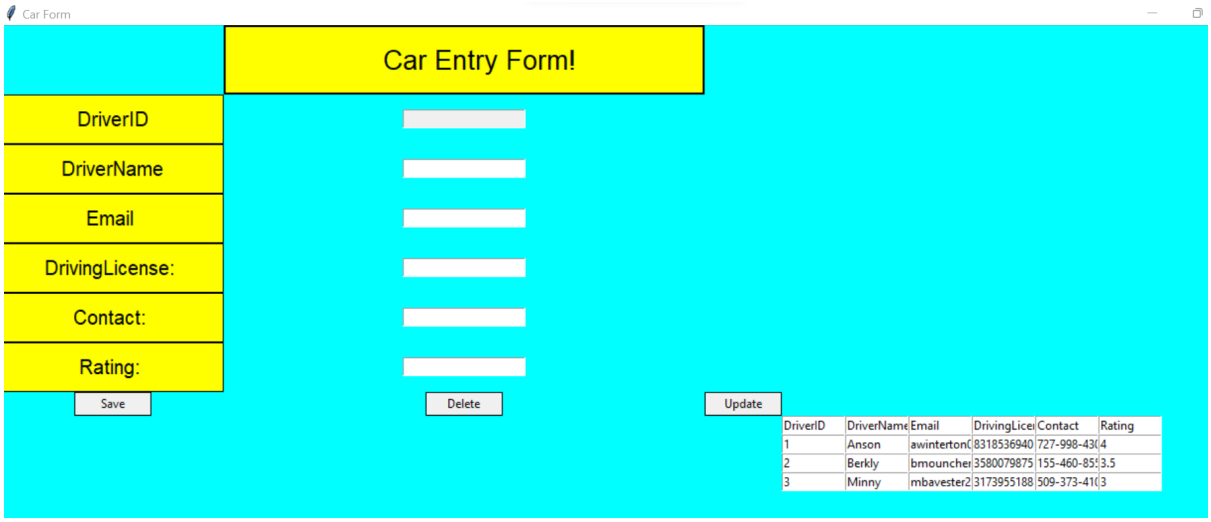
**Customer UI:**



Car UI:




Driver UI:



```
50
51  number=int(input("Enter number  : "))
52  radius=3963.2
53  Query2= collection_carservicecentre_Details.find({ "location":
54      { "$geoWithin":
55          { "$centerSphere": [ [ -73.93414657, 40.82302903 ], number / radius ] } } })
56
57  print(list(Query2))
58
59  ##########All car service within five miles of the user in sorted order from nearest to farthest#########
60
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
ive\Documents\mongodb\Use-case\Find_carservicecentre.py'
Enter number  : 5
[{'_id': ObjectId('55cba2476c522cafdb053b75'), 'location': {'coordinates': [-73.840344, 40.834541], 'type': 'Point'}, 'nam
e': 'FINAL TOUCH AUTO'}, {'_id': ObjectId('55cba2476c522cafdb053b52'), 'location': {'coordinates': [-73.841133, 40.835203]
, 'type': 'Point'}, 'name': 'CUSTOM AUTO TECH'}, {'_id': ObjectId('55cba2476c522cafdb053b2d'), 'location': {'coordinates':
 [-73.841703, 40.838511], 'type': 'Point'}, 'name': 'ANTHONY AUTOBODY'}, {'_id': ObjectId('55cba2476c522cafdb053bea'), 'lo
cation': {'coordinates': [-73.841703, 40.838511], 'type': 'Point'}, 'name': 'TERMINAL AUTO BODY'}, {'_id': ObjectId('55cba
2476c522cafdb053b5d'), 'location': {'coordinates': [-73.860505, 40.867014], 'type': 'Point'}, 'name': 'DYNAMIC COLLISION'}
, {'_id': ObjectId('55cba2476c522cafdb053bc2'), 'location': {'coordinates': [-73.865024, 40.863494], 'type': 'Point'}, 'na
me': 'PERSONAL TOUCH BODY'}, {'_id': ObjectId('55cba2476c522cafdb053bf9'), 'location': {'coordinates': [-73.873881, 40.874
144], 'type': 'Point'}, 'name': 'VICTORS AUTO BODY'}, {'_id': ObjectId('55cba2476c522cafdb053ba9'), 'location': {'coordina
tes': [-73.877934, 40.87045], 'type': 'Point'}, 'name': 'M G AUTO BODY INC'}, {'_id': ObjectId('55cba2476c522cafdb053bfd')
```

```
58
59  ##########All car service within five miles of the user in sorted order from nearest to farthest#########
60
61  query3=collection_carservicecentre_Details.find({ "location": { "$nearSphere": { "$geometry":
62      { "type": "Point", "coordinates": [ -73.93414657, 40.82302903 ] }, "$maxDistance": 5 * 1609.34 } } })
63  print(list(query3))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
'; & 'C:\Users\Saud Azmi\AppData\Local\Microsoft\WindowsApps\python3.10.exe' 'c:\Users\Saud Azmi\.vscode\extensions\ms-pyt
hon.python-2022.10.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '56781' '--' 'c:\Users\Saud Azmi\OneDr
ive\Documents\mongodb\Use-case\Find_carservicecentre.py'
[{'_id': ObjectId('55cba2476c522cafdb053b1f'), 'location': {'coordinates': [-73.941295, 40.82459], 'type': 'Point'}, 'name
': '5 STAR AUTO'}, {'_id': ObjectId('55cba2476c522cafdb053bc9'), 'location': {'coordinates': [-73.928847, 40.833885], 'typ
e': 'Point'}, 'name': 'RALPHS AUTO REPAIR'}, {'_id': ObjectId('55cba2476c522cafdb053b85'), 'location': {'coordinates': [-7
3.913495, 40.825576], 'type': 'Point'}, 'name': 'GLB TOWING &'}, {'_id': ObjectId('55cba2476c522cafdb053b40'), 'location':
 {'coordinates': [-73.911753, 40.826998], 'type': 'Point'}, 'name': 'BX AUTOMOTIVE LLC'}, {'_id': ObjectId('55cba2476c522c
afdb053b72'), 'location': {'coordinates': [-73.920382, 40.837426], 'type': 'Point'}, 'name': 'F&C BODY SHOP AND'}, {'_id':
 ObjectId('55cba2476c522cafdb053bb0'), 'location': {'coordinates': [-73.910529, 40.82604], 'type': 'Point'}, 'name': 'NEW
LAST CHANCE AUTO'}, {'_id': ObjectId('55cba2476c522cafdb053bcd'), 'location': {'coordinates': [-73.910891, 40.829178], 'ty
pe': 'Point'}, 'name': 'RECHARGE AUTO BODY'}, {'_id': ObjectId('55cba2476c522cafdb053b6c'), 'location': {'coordinates': [-
```

```
53  ################################ car health#######################################
54
55  myquery= Service_centre_Details.aggregate( [
56      {
57          "$project":
58          {
59              "Car_id":1,
60              "service_id":1,
61              "maintenance_date" :1,
62              "dtDiff2": {"$subtract": [{"$toLong":{"$round": {"$divide": [{ "$subtract": ["$$NOW", "$maintenance_date
63                                                                                  86400000] }}},{"$toLong":150}]},
64
65              "status":
66                      {
67                      "$cond":{"if":{"$gte": [{"$toLong":{"$round": { "$divide": [{ "$subtract": ["$$NOW","$maintenanc
68                                                                                  86400000] }}},{"$toLong":150}]},
69                                  "then":"Need Service","else" : "Good"}
```

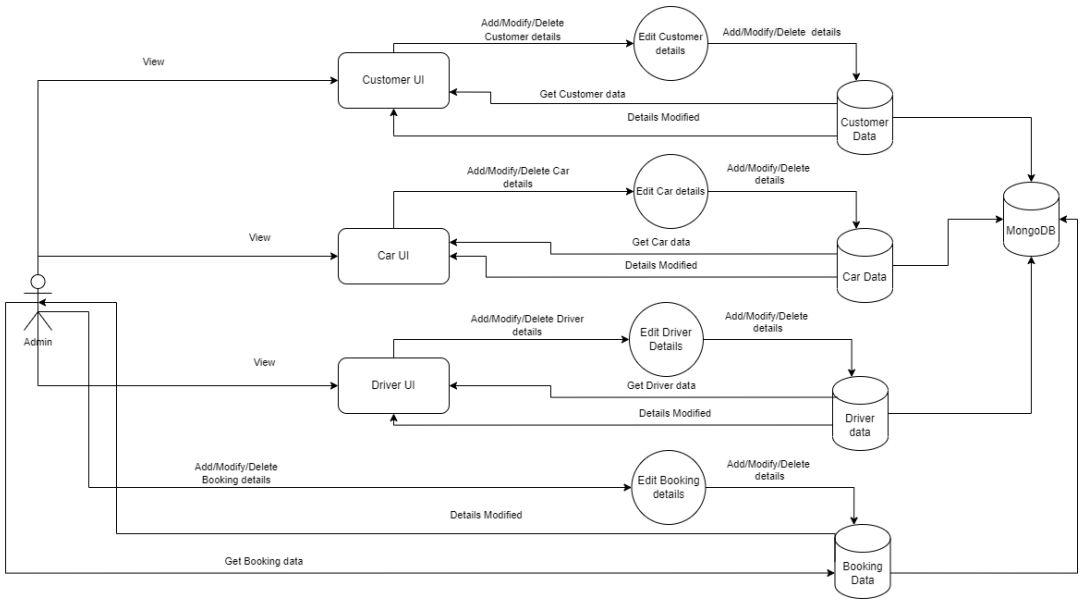
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
{'_id': ObjectId('62d8311089074f3e5b4417bc'), 'service_id': 691, 'Car_id': 'WA1DGAFE4CD194173', 'maintenance_date': dateti
me.datetime(1970, 1, 1, 0, 0), 'dtDiff2': 19045, 'status': 'Need Service'}
{'_id': ObjectId('62d8311089074f3e5b4417bd'), 'service_id': 692, 'Car_id': '3C3CFFJH2DT697543', 'maintenance_date': dateti
me.datetime(1970, 1, 1, 0, 0), 'dtDiff2': 19045, 'status': 'Need Service'}
{'_id': ObjectId('62d8311089074f3e5b4417be'), 'service_id': 693, 'Car_id': 'JN8AZ2KR1CT443878', 'maintenance_date': dateti
me.datetime(1970, 1, 1, 0, 0), 'dtDiff2': 19045, 'status': 'Need Service'}
{'_id': ObjectId('62d8311089074f3e5b4417bf'), 'service_id': 694, 'Car_id': '3GYT4LEFXCG540661', 'maintenance_date': dateti
me.datetime(2021, 6, 11, 22, 0), 'dtDiff2': 255, 'status': 'Need Service'}
{'_id': ObjectId('62d8311089074f3e5b4417c0'), 'service_id': 695, 'Car_id': 'WAUEF78E28A047403', 'maintenance_date': dateti
me.datetime(2021, 6, 6, 22, 0), 'dtDiff2': 260, 'status': 'Need Service'}
{'_id': ObjectId('62d8311089074f3e5b4417c1'), 'service_id': 696, 'Car_id': 'SCFEBBBK4DG694815', 'maintenance_date': dateti
```

```
d = int(input("Enter number  : "))
myquery2= Driver_Details.aggregate( [{
    "$project":
    {
        "DriverID": 1,
        "status":{
            '$cond':{
                "if":{"$gte":["$AverageRating",d]},
                "then" :"10 percent bonus",
                "else" : "4 percent bonus"
                }
                }
    }
}])
```

```
EMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

d': ObjectId('62d16f60a9e8cb86b4a8218c'), 'DriverID': 991, 'status': '4 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a8218d'), 'DriverID': 992, 'status': '10 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a8218e'), 'DriverID': 993, 'status': '4 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a8218f'), 'DriverID': 994, 'status': '10 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82190'), 'DriverID': 995, 'status': '10 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82191'), 'DriverID': 996, 'status': '10 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82192'), 'DriverID': 997, 'status': '4 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82193'), 'DriverID': 998, 'status': '4 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82194'), 'DriverID': 999, 'status': '10 percent bonus'}
d': ObjectId('62d16f60a9e8cb86b4a82195'), 'DriverID': 1000, 'status': '4 percent bonus'}
d': ObjectId('62d8182d9d06a8c5b4351df8'), 'DriverID': '123', 'status': '10 percent bonus'}
:\Users\Saud Azmi\OneDrive\Documents\mongodb\Use-case>
```

### 4.1.6. Data Flow



### 4.1.7 Databases

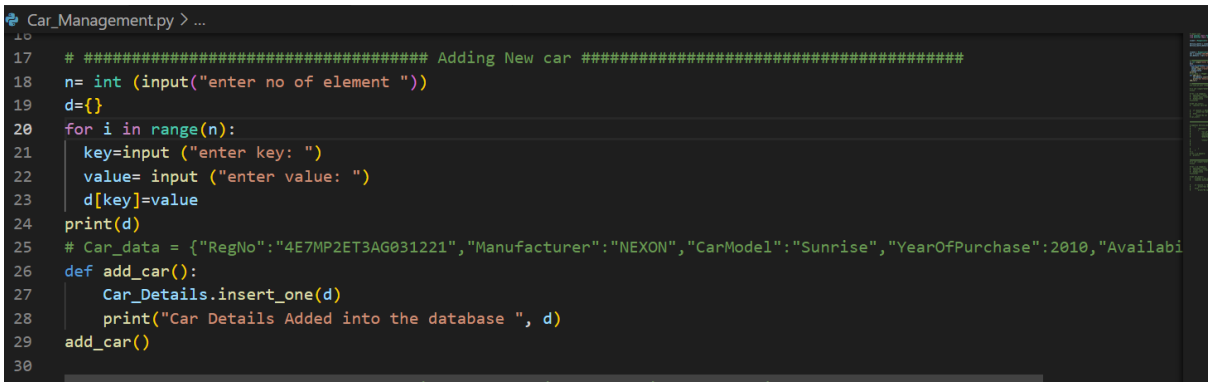
MongoDB

#### *4.1.7.1. Database used and why. 1-2 Sentences each*

MongoDB is a NoSQL database, and it supports powerful querying and analytics. In MongoDB, the fields can be made to vary within documents to enable data structure changes over a period. The document data model is a powerful way to store and retrieve data. As MongoDB stores data in the form of key-value pair, there are no needs of complex joins. Joins can be performed easily using aggregate method. It supports dynamic queries. Also, as it is schema free, no schema migrations are required. MongoDB is horizontally scalable i.e. sharding is possible and it emphasizes on the CAP theorem (Consistency, Availability, and Partition tolerance). It easy for developers to store, manage, and retrieve data when creating applications with most programming languages and it also provide great performs.

#### 4.1.7.2. Expressions used for this use case

To add car details:




To find the list of cars which are available:

To get the car health status:

```
54
55   myquery= Service_centre_Details.aggregate( [
56       {
57           "$project":
58           {
59               "Car_id":1,
60               "service_id":1,
61               "maintenance_date" :1,
62               "dtDiff2": {"$subtract": [{"$toLong":{"$round": {"$divide": [{ "$subtract": ["$$NOW", "$maintenance_date"
63                                                                                           86400000] }}},{"$toLong":150}]},
64
65               "status":
66                       {
67                       "$cond":{"if":{"$gte": [{"$toLong":{"$round": { "$divide": [{ "$subtract": ["$$NOW","$maintenance
68                                                                                           86400000] }}},{"$toLong":150}]},
69                                "then":"Need Service","else" : "Good"}
70                       }
71
72           }
73       }
74   ])
75   for x in myquery:
76     print(x)
77
```

To get the car service details based on the car id:

```
############################ Car Service History details##########################
n= int (input("enter no of element "))
d={}

for i in range(n):
  key=input ("enter key: ")
  value= input ("enter value: ")
  d[key]=value
print(d)

def Car_hist():
    # data2= {'Car_id': '3C3CFFDR2FT212065'}
    result2= Service_centre_Details.find(d)


    if result2 != "None":
        print("Car Service details found:", list(result2))
    else:
        print("No Car Service  details found")
```

To get nearby car service centre:

```
#############all Car_service_centre within five miles of the user:
######use $geoWithin with $centerSphere.
# $centerSphere is a  syntax to denote a circular region by specifying the center and the radius in radians.
####3963.2 is radius of earth in Miles.

number=int(input("Enter number  : "))
radius=3963.2
Query2= collection_carservicecentre_Details.find({ "location":
   { "$geoWithin":
     { "$centerSphere": [ [ -73.93414657, 40.82302903 ], number / radius ] } } })

print(list(Query2))
```

To get the list in sorted order from nearest to farthest:

```
query3=collection_carservicecentre_Details.find({ "location": { "$nearSphere": { "$geometry":
    { "type": "Point", "coordinates": [ -73.93414657, 40.82302903 ] }, "$maxDistance": 5 * 1609.34 } } })
print(list(query3))
```

To get customer details:

```
########################## Finding a particular Customer on the basis of Customer_ID ##########################
class_list = dict()
data = input('Enter name & score separated by ":" ')
temp = data.split(':')
class_list[temp[0]] = int(temp[1])

print (class_list)
##### function to display customer details########
def cust():
  data1= class_list
  result= str(Customer["Customer Details"].find_one(data1)
          )
  if result != "None":
    print("Customer you are looking for is :", result)
  else:
    print("Unknown CustomerID")

cust()
```

To update customer details:

```
def Cust_update():
    cust_result= str(Customer["Customer Details"].find_one(class_list))
    print(cust_result)

    if cust_result != "None":
        update_cust_details= CUstomer_Details.find_one_and_update(class_list,
                                {"$set":
                                    {"Email": "test12@gmail.com"}}
                                ,upsert=True)

    else:
        print("No customer found")
    cust_result1= str(Customer["Customer Details"].find_one(class_list))
    print(cust_result1)
```

List of Driver with multiple rides:

```
def driver_list():
    myquery2= Booking_Details.aggregate( [
        {"$group": {
            "_id": {"DriverID": "$DriverID"},
            "uniqueIds": {"$addToSet": "$_id"},
            "count": {"$sum": 1}
            }
        },
        {"$match": {
            "count": {"$gt": 1}
            }
        },
        {"$sort": {
            "count": -1
            }
        }
    ])
```

To debar a driver when the rating is below a prescribed limit:

```
def debar_driver():
    driver_rating = Driver["Driver Details"].find({"AverageRating" : data } )

    print("Driver with good rating:",list(driver_rating))

    if driver_rating != "None":
        print("Driver details :", driver_rating)
    else:
        print("No driver  details found")
    for x in driver_rating:
        print(x)
```

To update the booking details:

```
n= int (input("enter no of element "))
d={}

for i in range(n):
  key=input ("enter key: ")
  value= input ("enter value: ")
  d[key]=value
print(d)

data2=class_list

result123=Booking_Details.find_one_and_update (data2,
                                {"$set":
                                    d
                                },upsert=True)
print(result123)
```

To get customer discount:

```
Query= [{
        "$project":
        {
            "CustomerID": 1,
            "Customer_Rating":1,
            "status":{
                '$cond':{
                    "if":{"$gte":["$Customer_Rating",n]},
                    "then" :"20 percent off",
                    "else" : "10 percent off"
                    }
                    }
        }
    }]

cursor = Booking_Details.aggregate(Query)
for doc in cursor:
        print(doc)
```

To get bonus to the drivers:

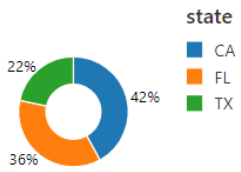
```
myquery2= Driver_Details.aggregate( [{
    "$project":
    {
        "DriverID": 1,
        "status":{
            '$cond':{
                "if":{"$gte":["$AverageRating",d]},
                "then" :"10 percent bonus",
                "else" : "4 percent bonus"
                }
                }
    }
}])
```

### 4.1.7.3. Reports and Data analysis on PySpark data frame

Checking the top 3 states that uses the most cars.

```
%sql
/* Top 3  states that uses the most cars :*/
select state,count(state) as Count from `Car_Rental` group by 1 order by Count desc
limit 3
```

▸ (2) Spark Jobs




Top 3 cities that uses most cars:

```
%sql
/* Top 3 cities that uses the most cars :*/
select city,count(city) as Count from `Car_Rental` group by 1 order by Count desc
limit 3
```

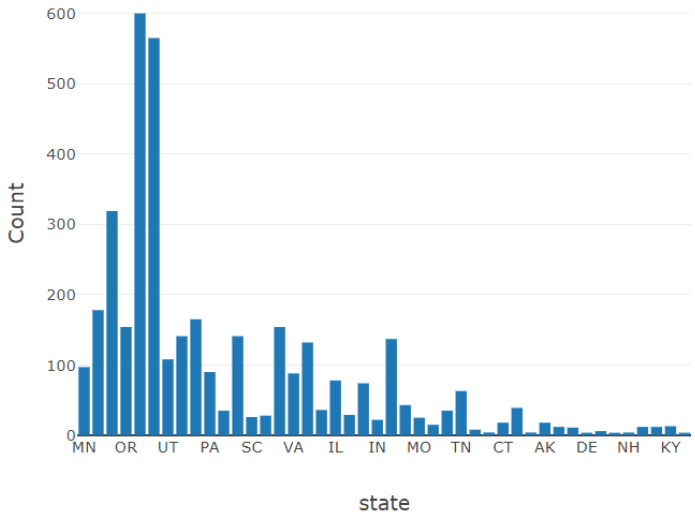
▸ (2) Spark Jobs

| | city | Count |
|---|---|---|
| 1 | Las Vegas | 186 |
| 2 | Portland | 166 |
| 3 | San Diego | 163 |

Top states with maximum owners:

```
%sql
/*Top states with max owners*/
select state,id,count(state) as Count from `Car_Rental` group by state,id   order by Count desc
```

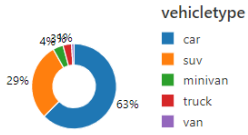
(2) Spark Jobs




Different type of vehicle used:

```
%sql
/*different vehicle:*/
select vehicletype,count(vehicletype) as Count from `Car_Rental` group by vehicletype   order by Count desc
```

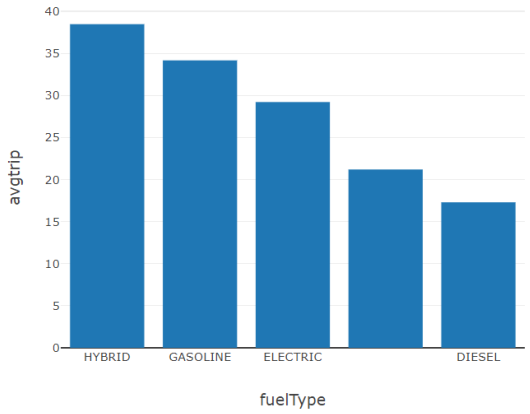
(2) Spark Jobs




Top 5 vehicle makers who helped making most avg. daily rate:

```
%sql
/* Top 5 vehicle makers who helped making most avg. daily rate*/
select fuelType,avg(renterTripsTaken) as avgtrip from `Car_Rental` group by fuelType   order by avgtrip desc
limit 5

```

(2) Spark Jobs

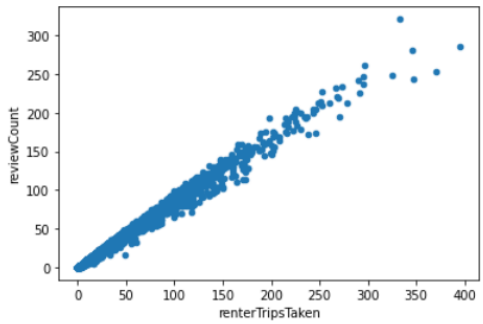



Does renter trips affect review counts?

```
##############Does renter trips affect review counts?#############

df = sqlContext.sql("Select * from `Car_Rental`")
df = df.toPandas()
# df.set_index('rating').plot()
df.plot.scatter(x='renterTripsTaken',y='reviewCount')

```

(1) Spark Jobs

```
Out[12]: <AxesSubplot:xlabel='renterTripsTaken', ylabel='reviewCount'>
```

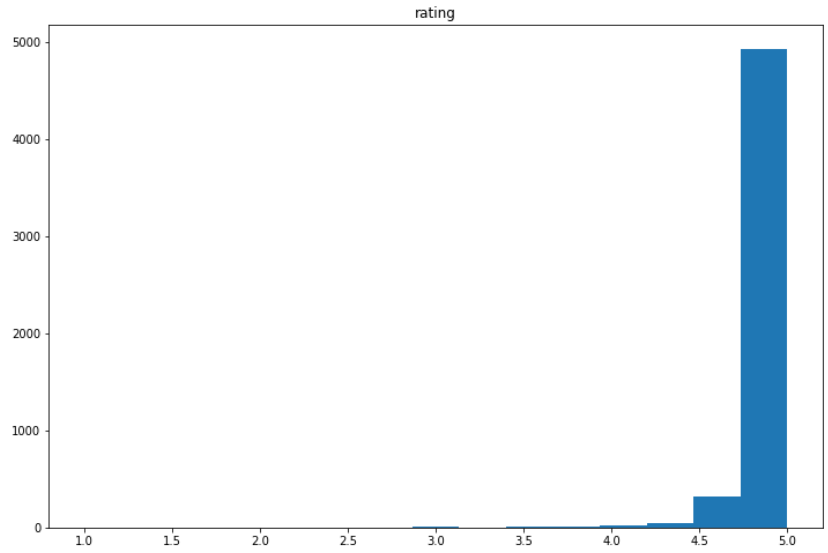



Car rating:

```
1  ############### car rating ################
2  df = sqlContext.sql("Select * from `Car_Rental`")
3  df = df.toPandas()
4  df.hist(column='rating',bins=15,grid=False, figsize=(12,8))
5
```

▸ (1) Spark Jobs

```
Out[18]: array([[<AxesSubplot:title={'center':'rating'}>]], dtype=object)
```



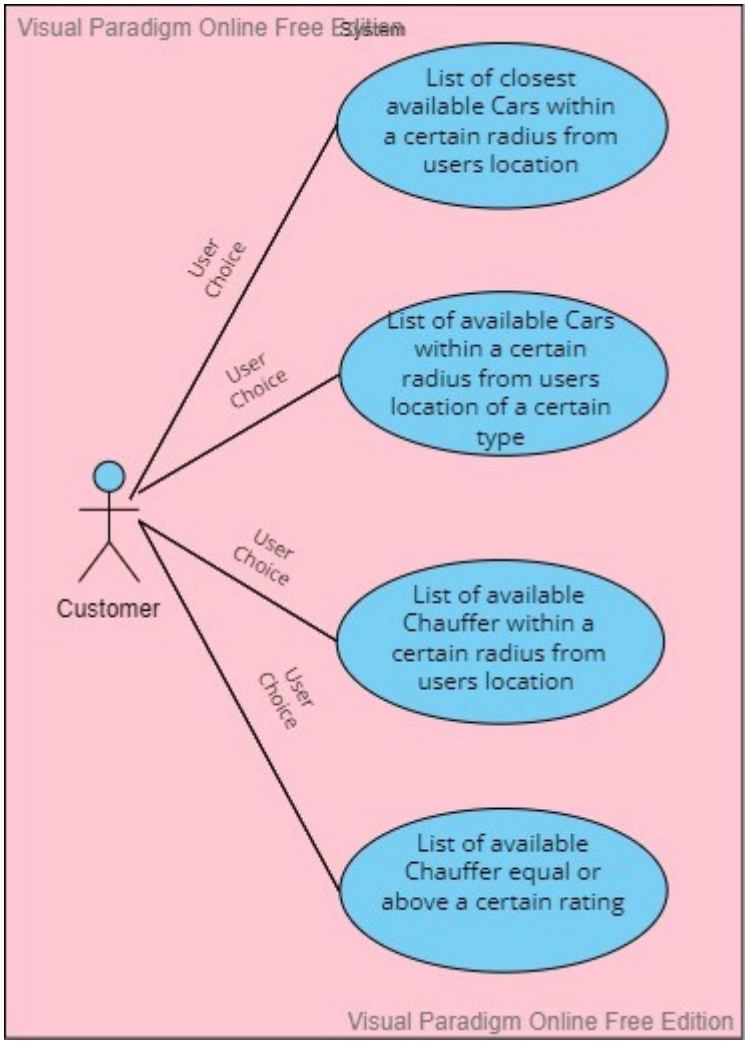

## 4.2. Subject 1 ("Car Rental Customer Booking Interface (Gaurav Khurana)")

Find the nearest car that is available for rent from a list of available cars within an input radius, the rented car could either be driven by self or you could have a chauffeur get the car to your location and drive you to the destination.

### 4.2.1. User Story (For this Subject)

As a user, I want to see the list of cars that are available to be rented near me in a given radius that I have the ability to input as well as the ability to choose the destination. I can choose the type of car that I want to drive along with the option of being driven by a chauffeur to my destination. I can also choose the chauffer from a list of chauffeurs by deciding upon either the nearest chauffer from the car or the highest rated chauffer in a particular radius of the car.

### 4.2.2. Identified Use Case (UML: Use Case)




### 4.2.3. Actors (Not just Admin)

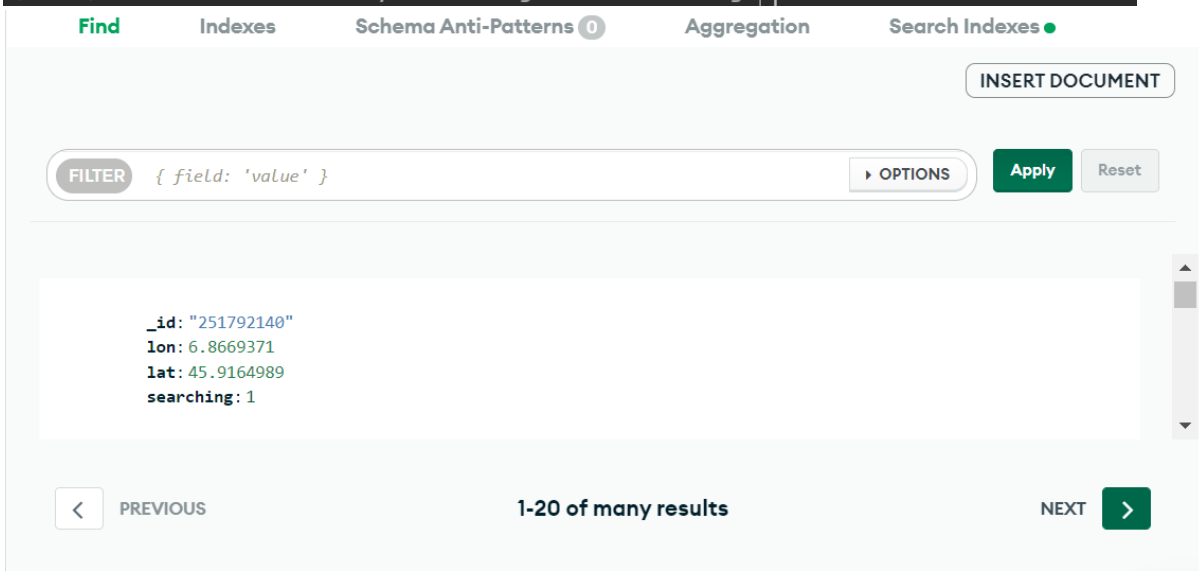
- Registered User

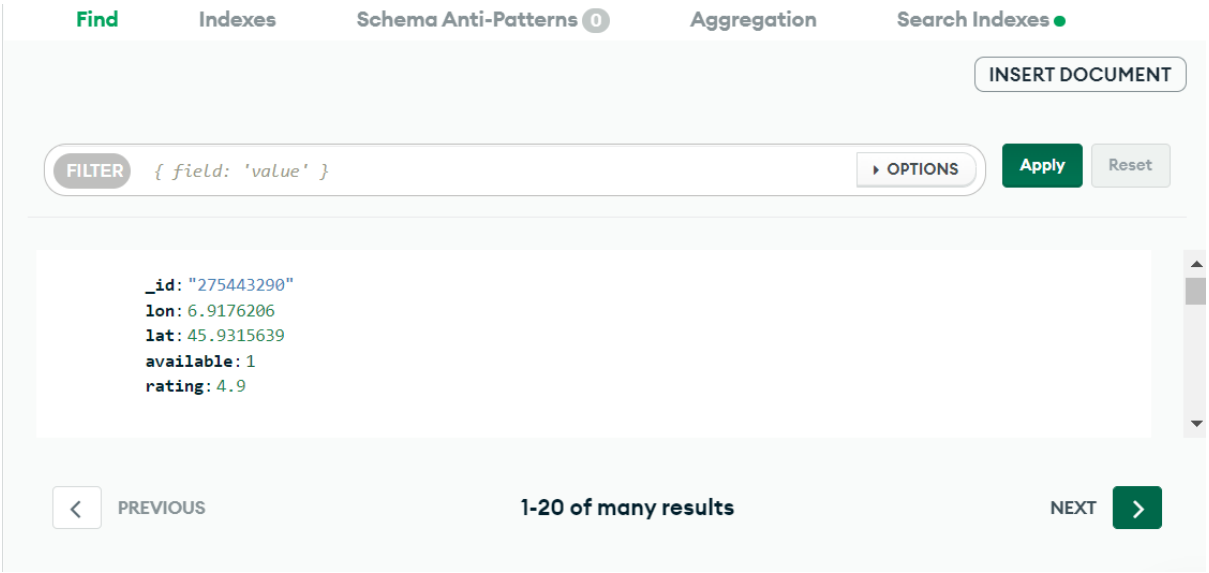
### 4.2.4. Descriptions of task with all interactions between actor and database. (Detailed)

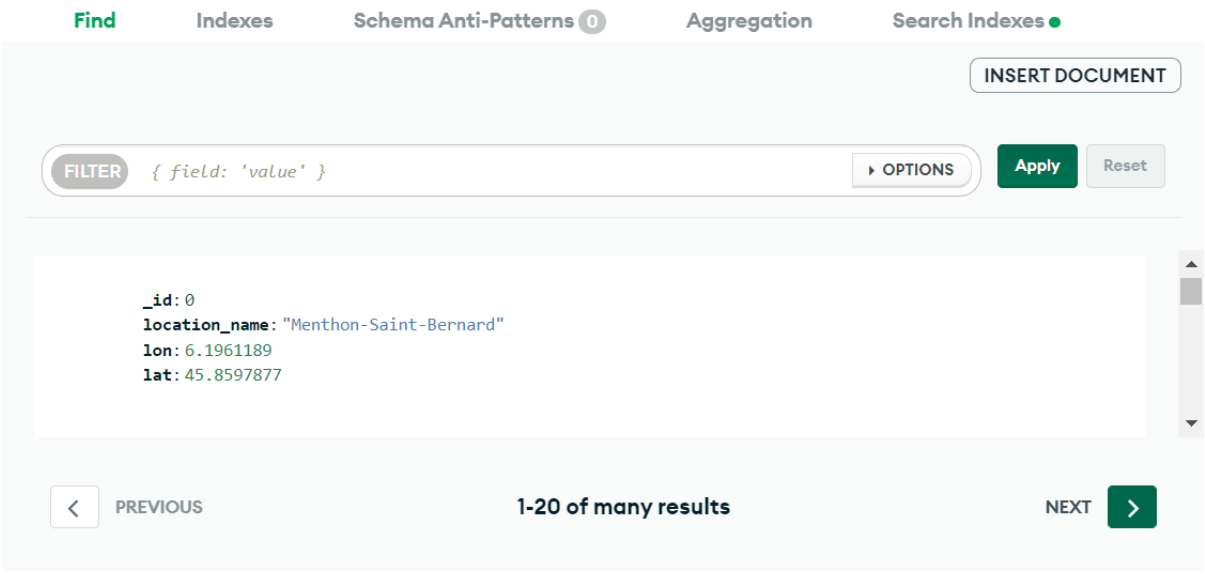
Registered users are a particular location. These locations are randomly generated and stored for a particular customerID (stored and retrieved from MongoDB Atlas). From the location coordinates user has the option to find a car available for rent in a particular radius (All available cars are then stored in Redis and list of available cars are retrieved using the geospatial functions of redis). If there are no cars available then the user will be prompted to expand his area of search. Similarly, the user will also have the option to hire a chauffeur along with the car to be driven to a particular destination. The user can select a destination and the program will generate the 1) the nearest available vehicles and the nearest vehicle 2) if they opt for a chauffeur, the list of nearest available chauffer to the vehicle is given to the user to choose from 3) Calculation of the total distance, time and the amount are calculated for the user 4) OTP is generated for the user to verify before starting the trip.

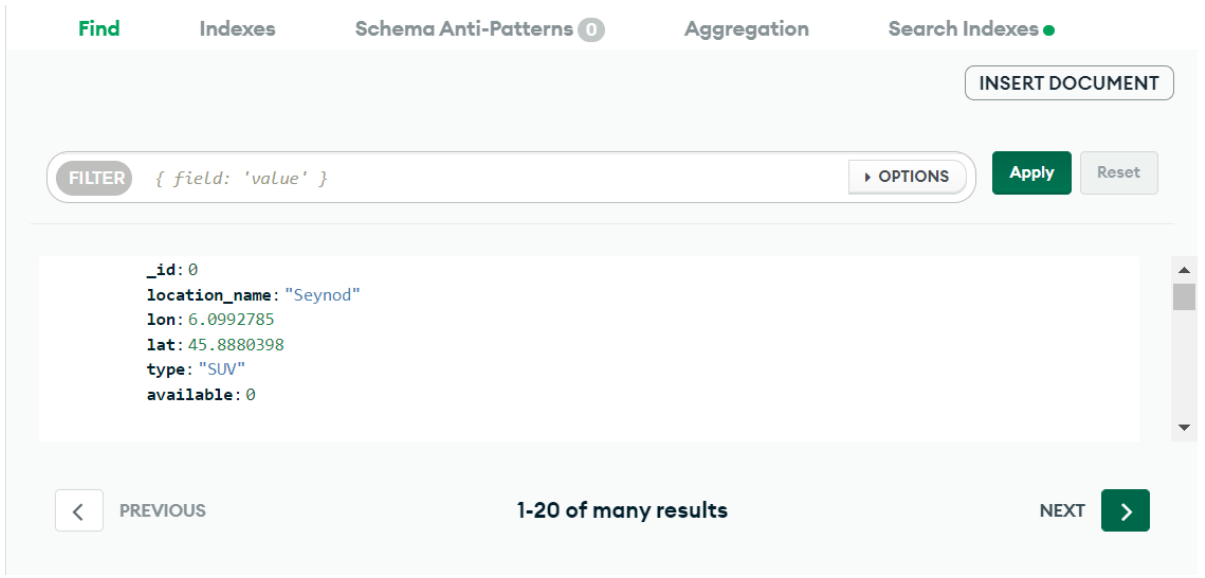
### 4.2.5. Optional: Frontend used / Command lines to reproduce execution

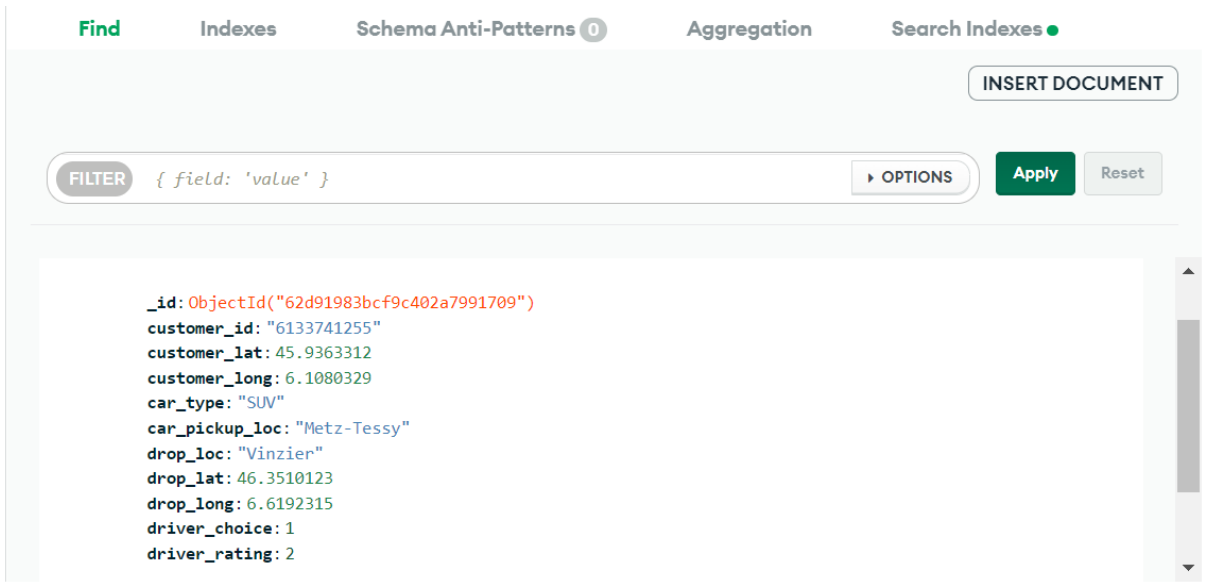
```
(venv) C:\Users\Admin\PycharmProjects\tracking>python data_to_mongo.py
Emptied the collection PickUpLocations
160 data points inserted successfully in collection PickUpLocations!
Emptied the collection DropOffLocations
161 data points inserted successfully in collection DropOffLocations!
Emptied the collection CustomerLocations
1038 data points inserted successfully in collection CustomerLocations!
Emptied the collection DriverLocation
141 data points inserted successfully in collection DriverLocation!

(venv) C:\Users\Admin\PycharmProjects\tracking>
```




Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
INSERT DOCUMENT
FILTER
{ field: 'value' }
OPTIONS
Apply
Reset
_id: 0
location_name: "Menthon-Saint-Bernard"
lon: 6.1961189
lat: 45.8597877
PREVIOUS
1-20 of many results
NEXT


Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
INSERT DOCUMENT
FILTER
{ field: 'value' }
OPTIONS
Apply
Reset
_id: 0
location_name: "Seynod"
lon: 6.0992785
lat: 45.8880398
type: "SUV"
available: 0
PREVIOUS
1-20 of many results
NEXT


Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
INSERT DOCUMENT
FILTER
{ field: 'value' }
OPTIONS
Apply
Reset
_id: ObjectId("62d91983bcf9c402a7991709")
customer_id: "6133741255"
customer_lat: 45.9363312
customer_long: 6.1080329
car_type: "SUV"
car_pickup_loc: "Metz-Tessy"
drop_loc: "Vinzier"
drop_lat: 46.3510123
drop_long: 6.6192315
driver_choice: 1
driver_rating: 2

```
(venv) C:\Users\Admin\PycharmProjects\tracking>python customer_interface.py
Successfully Ingested in drop_off
Successfully Ingested in customer
Welcome to the Car Rental Interface


Hello Customer : 6133741255

Your current latitude is: 45.9363312
 Your current longitude is: 6.1080329
```

```
Are you looking for a specific type of vehicle?
1. Yes    2. No
1
The following are the car types available:
SUV       (Base Rate: Eur 35     Per Km: Eur 1.2)
HATCHBACK (Base Rate: Eur 25     Per Km: Eur 0.8)
SEDAN     (Base Rate: Eur 30     Per Km: Eur 1)
Please enter your choice: suv
Successfully Ingested in pick_up
Please input the radius to search the available car in:
10
```

Near by Vehichle Pick-Up Locations are: {'Metz-Tessy': 0.3577, 'Épagny': 1.9192, 'Pringy': 1.9327, 'Argonay': 3.9612, 'Cuvat': 4.2929, 'Saint-Martin-Bellevue': 4.6043, 'Annecy': 4.7126, 'Poisy': 5.289, 'Seynod': 5.4288, 'Charvonnex': 7.0436, 'Allonzier-la-Caille': 7.1539, 'Dalmaz': 7.4603, 'Villy-le-Pelloux': 7.554, 'La Balme-de-Sillingy': 7.9247, 'Sillingy': 8.1032, 'Choisy': 8.3721, 'Chavanod': 8.4462, 'Vieugy': 8.4529, 'Corbier': 8.5355, 'Sevrier': 8.7647, 'Nâves-Parmelan': 9.0359, 'Lovagny': 9.1377, 'Veyrier-du-Lac': 9.2517, 'Nonglard': 9.2756, 'Villaz': 9.4356, 'Les Ollières': 9.4812}

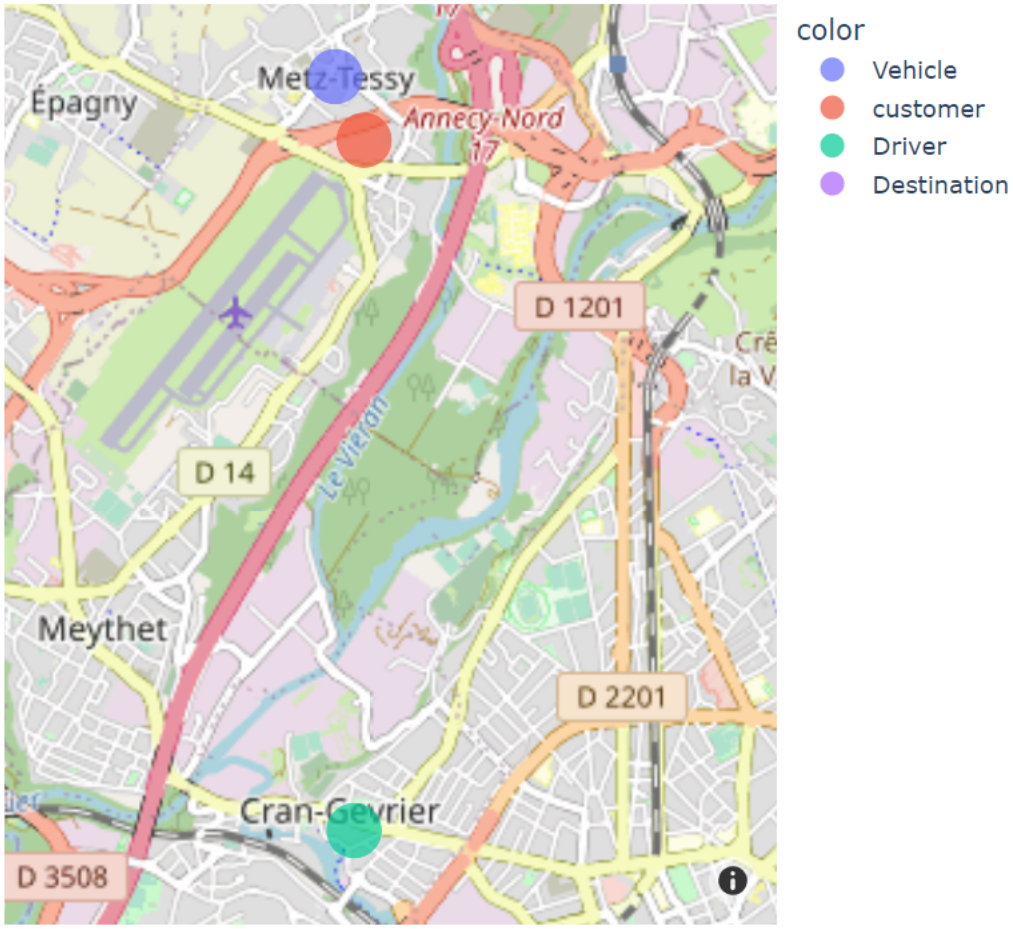
```
Nearest Pick-Up Location: Metz-Tessy
Distance to nearest vehichle :0.3577
Your drop location is: Vinzier
 Drop latitude is : 46.3510123
 Drop longitude is : 6.6192315

Distance between Pick-Up and Drop-Off: 73.0001 Km
```

```
Would you like a Chauffeur for this ride?
1. Yes   2. No
1
Are you looking for a driver of particular rating?
1. Yes    2. No
1
The following are the driver rating types available:
1. Between 1 and 3 stars (Base Rate: Eur 7     Per Km: Eur 0.5)
2. Between 3 and 4 stars (Base Rate: Eur 9     Per Km: Eur 0.7)
3. Above 4 stars         (Base Rate: Eur 11     Per Km: Eur 1)
Please enter your choice: 2
```

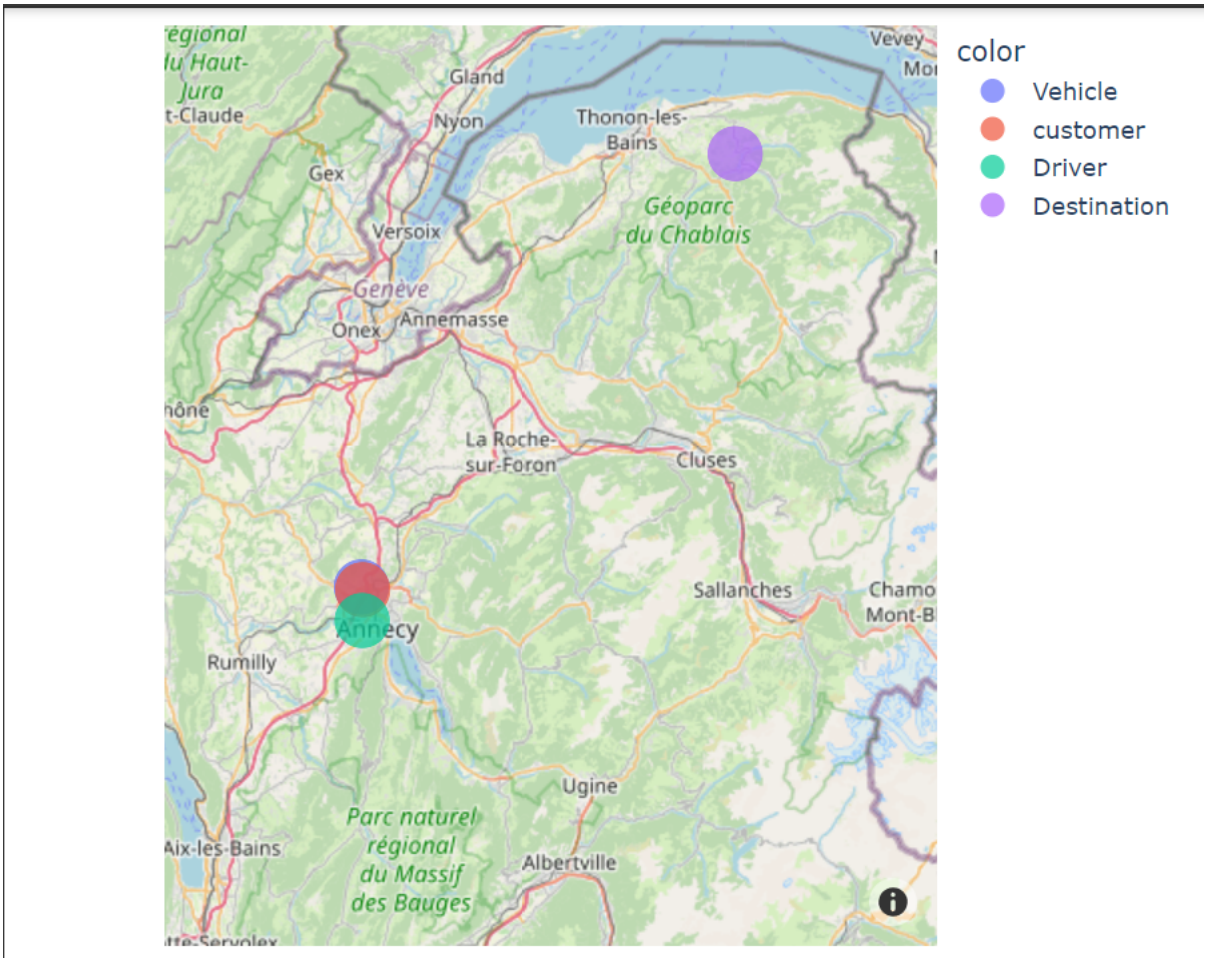
```
Successfully Ingested in drivers
Near by Driver by distance are: [('3378215477', 0.0801), ('6227049953', 3.5448), ('1746669767', 3.5537), ('341372776', 3.8594), ('2234304727', 3.9363), ('7
631856656', 4.3078), ('6467938179', 4.5181), ('4219944397', 4.532), ('7622483522', 4.6266), ('3379384774', 4.6376)]
```

```
Nearest Driver : 3378215477

Distance between Driver and Vehicle: 0.0801 Km

Are you happy with the choices and would like to confirm the booking?
1. Yes   2. No
1
Thank you for confirming the ride details! Your trip starts now.

Dear customer, please note your OTP for the service is: 106312
```

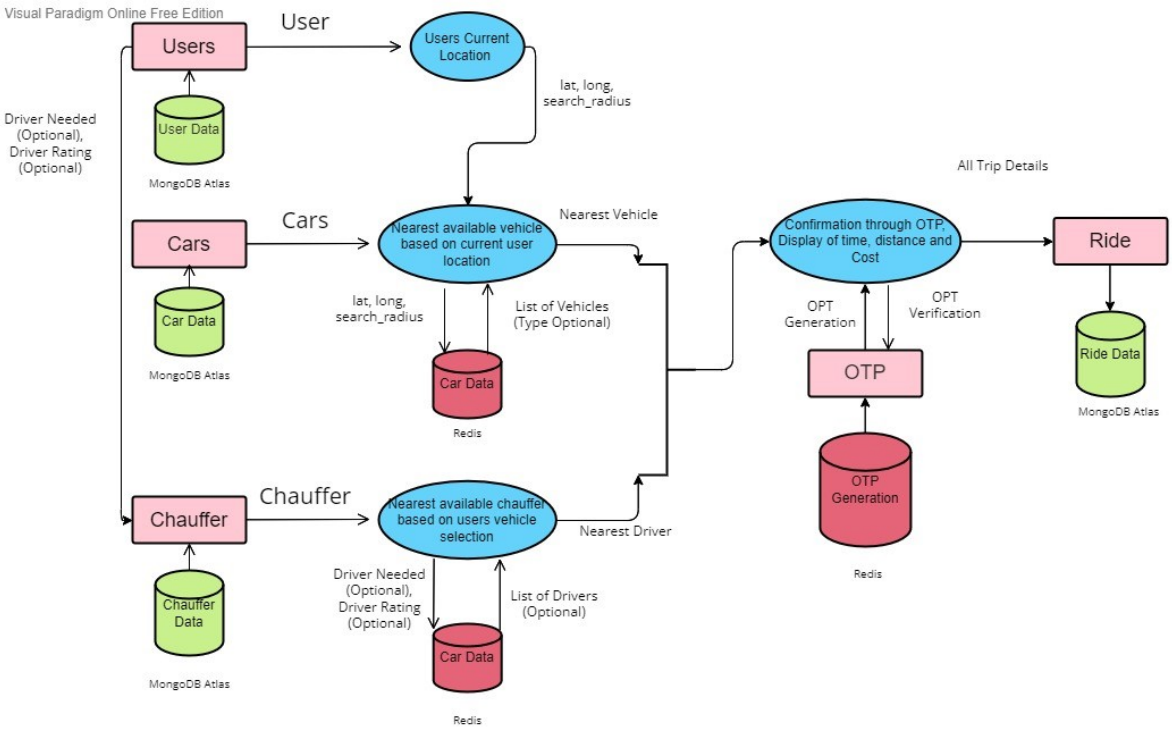
```
The Chauffeur has begun his journey to pick up the vehicle.
He's at a distance of 0Km. from the vehicle.
He will reach the vehicle in 0 mins.

The Chauffeur has picked up the vehicle, he's on his way to take you to your destination.
Please be ready with the OTP to start the ride.
The vehicle is 0Km away.
It will reach your current position in 0 mins.

The Chauffeur has arrived at your location.
Your destination is 73Km away.
You will reach your destination in 63 mins.

Dear customer please enter your OTP to start the trip:
106312
OTP verified successfully, enjoy your ride.

Dear Customer, your ride has been completed. Thank you for using our service.
```




```
Dear customer, The total cost of the trip is Eur 152.8.
This amount will be directly debited from your account within 1 Day. Please ensure you have enough balance in your account.
Thank you for using our services, hope to see you again soon!

All ride details stored in Mongo successfully! End of session...
```




### 4.2.6. Data Flow (Step by step, detailed)



### 4.2.7 Databases

#### *4.2.7.1. Database used and why. 1-2 Sentences each*

- MongoDB Atlas: The user, car, driver and ride data are stored in MongoDB. These data are generated randomly whenever the script runs, creating customer locations, car locations, drop locations, driver locations and the ride details.
- Redis: All the data depending upon the user input condition are loaded into Redis. Since we are dealing with geospatial data such as latitudes and longitudes, Redis is optimised to deal with such kind of data using its geospatial functions which helps us find out the nearest vehicle, driver, distance to destination from the customer location to the drop location.

#### 4.2.7.2. Expressions used for this use case

```
def insert_to_mongo(cluster_name, collection_name, insert_data):
    client = pymongo.MongoClient(
        "mongodb+srv://gk9713:DvZXo0a2WbphLM1r@cluster0.jufutvi.mongodb.net/?retryWrites=true&w"
        "=majority")

    db = client[cluster_name]
    collection = db[collection_name]

    collection.delete_many({})
    print(f"Emptied the collection {collection_name}")

    for data in insert_data:
        collection.insert_one(data)

    print(f"{len(insert_data)} data points inserted successfully in collection {collection_name}!")
```

```
def fetch_customer_data(cluster_name, collection_name):
    client = pymongo.MongoClient(
        "mongodb+srv://gk9713:DvZXo0a2WbphLM1r@cluster0.jufutvi.mongodb.net/?retryWrites=true&w"
        "=majority")

    db = client[cluster_name]
    collection = db[collection_name]

    customer_data = [tuple(data.values()) for data in collection.find({"searching": 1})]

    return customer_data
```

```
def vehicle_type():
    car_choice = str(input("The following are the car types available: \n"
                           "SUV       (Base Rate: Eur 35     Per Km: Eur 1.2)  \n"
                           "HATCHBACK (Base Rate: Eur 25     Per Km: Eur 0.8) \n"
                           "SEDAN     (Base Rate: Eur 30     Per Km: Eur 1)   \n"
                           "Please enter your choice: ")).upper()
    if car_choice not in ["SUV", "HATCHBACK", "SEDAN"]:
        print("You made an invalid choice, please try again! \n")
        vehicle_type()
    else:
        client = pymongo.MongoClient(
            "mongodb+srv://gk9713:DvZXo0a2WbphLM1r@cluster0.jufutvi.mongodb.net/?retryWrites=true&w"
            "=majority")

        db = client["LocationData"]
        collection = db["PickUpLocations"]

        available_cars = [tuple(data.values())[1:4] for data in collection.find({"available": 1,
                                                                                 "type": car_choice})]
        return available_cars, car_choice
```

```
def driver_rating():
    driver_rating_choice = int(input("The following are the driver rating types available: \n"
                                     "1. Between 1 and 3 stars (Base Rate: Eur 7     Per Km: Eur 0.5)  \n"
                                     "2. Between 3 and 4 stars (Base Rate: Eur 9     Per Km: Eur 0.7) \n"
                                     "3. Above 4 stars         (Base Rate: Eur 11     Per Km: Eur 1)   \n"
                                     "Please enter your choice: "))
    if driver_rating_choice not in [1, 2, 3]:
        print("You made an invalid choice, please try again! \n")
        driver_rating()
    else:
        min = 0
        max = 0
        if driver_rating_choice == 1:
            min = 1
            max = 3
        elif driver_rating_choice == 2:
            min = 3
            max = 4
        else:
            min = 4
            max = 5
```

```
        client = pymongo.MongoClient(
            "mongodb+srv://gk9713:DvZXo0a2WbphLM1r@cluster0.jufutvi.mongodb.net/?retryWrites=true&w"
            "=majority")

        db = client["LocationData"]
        collection = db["DriverLocation"]

        driver_data = [tuple(data.values())[:3] for data in
                       collection.find({"$and": [{"rating": {"$gte": min}}, {"rating": {"$lt": max}}]})]

        return driver_data, driver_rating_choice
```

```python
def redis_insertion(name, insert_data):
    for data in insert_data:
        r.geoadd(name, [data[2], data[1], data[0]])
    print(f"Successfully Ingested in {name}")


redis_insertion("pick_up", car_pick_ups)
redis_insertion("drop_off", drop_off_locations)
redis_insertion("customer", customers)

# Creating the dataset to search in
r.zunionstore("Temp", ("pick_up", "customer", "drop_off"), aggregate="MIN")
r.expire("Temp", 600)
```

```python
def generateOTP():
    digits = [i for i in range(10)]
    otp_str = ""

    # creating a 6 digit long otp string
    for i in range(6):
        index = math.floor(random.random() * 10)
        otp_str += str(digits[index])

    # storing customer OTP in redis
    r.set("OTP_Customer", str(otp_str))
    r.expire("OTP_Customer", 20)

    print(f"Dear customer, please note your OTP for the service is: {otp_str} \n")

    # storing drive OTP in redis, will be same for driver or self driven
    r.set("OTP_Drive", str(otp_str))
    r.expire("OTP_Drive", 20)
```

```
def verifyOTP():
    otp_customer = r.get("OTP_Customer")
    otp_drive = r.get("OTP_Drive")

    otp_received = str(input("Dear customer please enter your OTP to start the trip: "
                             "\n"))

    # print(otp_customer.decode(), otp_drive.decode())

    if otp_received == str(otp_customer.decode()) and str(otp_customer == otp_drive.decode()):
        print("OTP verified successfully, enjoy your ride. \n")
    else:
        print("Invalid OTP, please try again! \n")
        verifyOTP()
```

```
radius = int(input("Please input the radius to search the available car in: "))

vehicle_distances = r.georadiusbymember("Temp", random_customer, unit="km", withdist=True, radius=radius)
```

```
# Generating a random drop off location
drop_location = drop_off_locations[random.randrange(len(drop_off_locations) - 1)]
print(f"Your drop location is: {drop_location[0]} \n "
      f"Drop latitude is : {drop_location[2]} \n "
      f"Drop longitude is : {drop_location[1]} \n")

# Calculating the distance to drop off using redis
distance_to_drop_off = r.geodist("Temp", nearest_pick_up_location, drop_location[0], unit="km")
print(f"Distance between Pick-Up and Drop-Off: {distance_to_drop_off} Km \n")
```

```
redis_insertion("drivers", drivers)

r.zunionstore("TempDriver", ("pick_up", "drivers"), aggregate="MIN")
r.expire("Temp", 600)
driver_distances = r.georadiusbymember("TempDriver", nearest_pick_up_location, unit="km", withdist=True
                                       radius=radius)
# print(driver_distances)
driver_locations, nearest_driver_location = check_driver_availability(driver_distances)
```

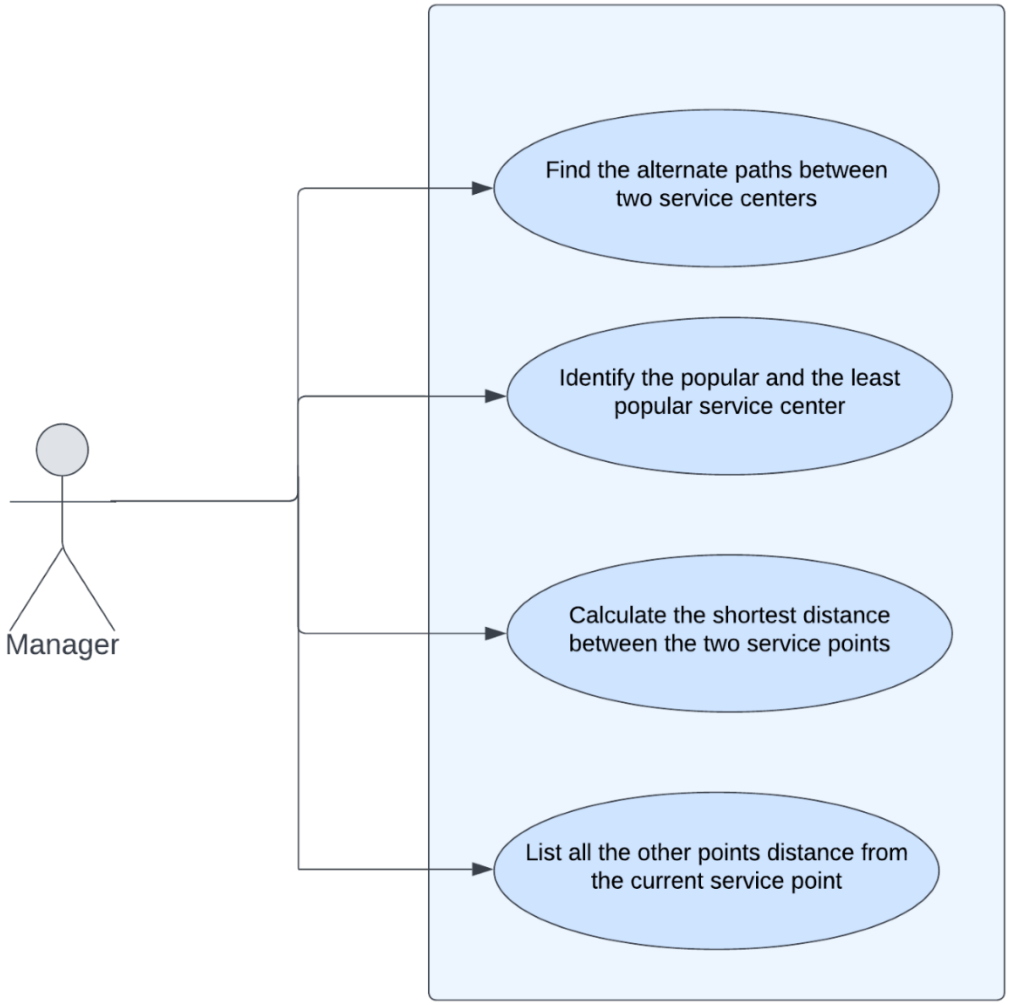
```
# Calculating the distance from driver to vehicle using redis
distance_from_driver_to_car = r.geodist("Temp", nearest_driver_location, nearest_pick_up_location, unit
print(f"Distance between Driver and Vehicle: {distance_from_driver_to_car} Km \n")

ride_confirmation(car_dist=int(dist_nearest_pick_up_location), drop_dist=int(distance_to_drop_off),
                  driver_dist=int(distance_from_driver_to_car), car_type=type_of_car,
                  rating_type=driver_rating_range)
```

## 4.3 Shortest Path Between the service points(Roshan Scaria)

### 4.3.1 User Story

The manager of the company is planning to visit the service centers to ensure the smooth functioning, he wants to ensure that the transportation between the service points is smooth and obstructions in one of the paths doesn't affect the travel and thereby business productivity. He wants to promote a service center to a main office by analyzing the service centers which is having the highest number of connections. The manager wants to find out the shortest distance between the main office to the service center that is the having the least connections. He also wants to calculate the possible distance to the rest of the service centers from the service center with least connections.

### 4.3.2 Identified Use Case



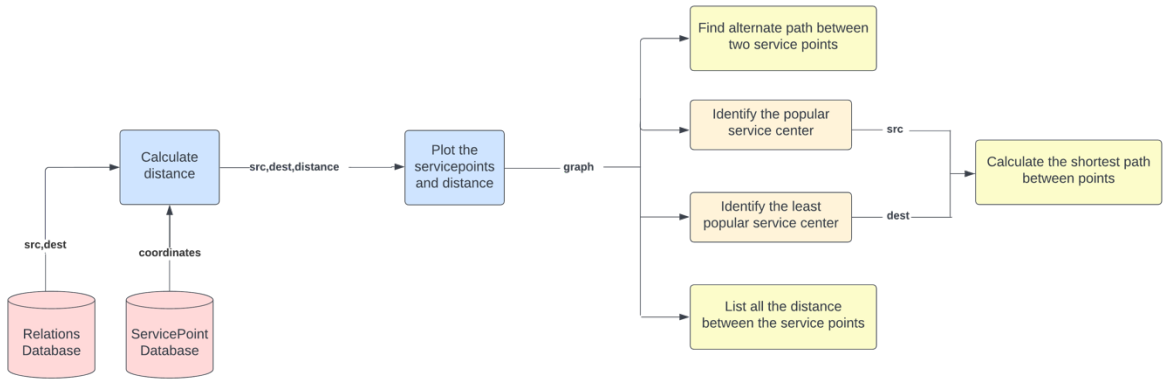

### 4.3.3 Actors

- Manager

## 4.3.4 Description

To get the different paths between two service points, to identify the most and fewest popular service centers and calculate the distance between them. To find the distance to the rest of the service centers from the service point.

## 4.3.5 Data Flow




## 4.3.6 Databases

### 4.3.6.1 Database used

In a graph approach, the query results are much faster with minimum traversals between the nodes. Hence, the time taken to provide the results back to the user is considerably less as compared to a relational model. This is mainly because of the highly connected data. Thus, Neo4j was used.

### 4.3.6.2 Expressions used

```
//To get the paths from a particular service point
MATCH (source:city {name: 'Mannheim'}), (target:city {name: 'Ludwigsburg'}) CALL
gds.shortestPath.yens.stream('road', {sourceNode: source,targetNode: target,k:
3,relationshipWeightProperty: 'distance'})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN index, gds.util.asNode(sourceNode).name AS sourceNodeName,
gds.util.asNode(targetNode).name AS targetNodeName, totalCost,[nodeId IN nodeIds |
gds.util.asNode(nodeId).name] AS nodeNames,costs,nodes(path) as path


//To get the service point with maximum connections
CALL gds.degree.stream('road')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS total_traffic ORDER BY total_traffic
DESC, name DESC limit 5


//To get the service point with minimum connections
CALL gds.degree.stream('road')
```

```
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS total_traffic
ORDER BY total_traffic ASC limit 1
```

```
//To get the paths from a particular service point
MATCH (source:city {name: 'Mannheim'}), (target:city {name: 'Ludwigsburg'}) CALL
gds.shortestPath.yens.stream('road', {sourceNode: source,targetNode: target,k:
3,relationshipWeightProperty: 'distance'})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN index, gds.util.asNode(sourceNode).name AS sourceNodeName,
gds.util.asNode(targetNode).name AS targetNodeName, totalCost,[nodeId IN nodeIds |
gds.util.asNode(nodeId).name] AS nodeNames,costs,nodes(path) as path
```

```
//Distance to other service centers from the least popular node
CALL gds.alpha.allShortestPaths.stream("road", {
relationshipWeightProperty: 'distance'
})
YIELD sourceNodeId, targetNodeId, distance
WITH sourceNodeId, targetNodeId, distance
WHERE gds.util.isFinite(distance) = true
MATCH (source:city) WHERE id(source) = sourceNodeId
MATCH (target:city) WHERE id(target) = targetNodeId
WITH source, target, distance WHERE source <> target AND source.name="Ulm"
RETURN source.name AS source, target.name AS target, distance as distance_in_kms
```

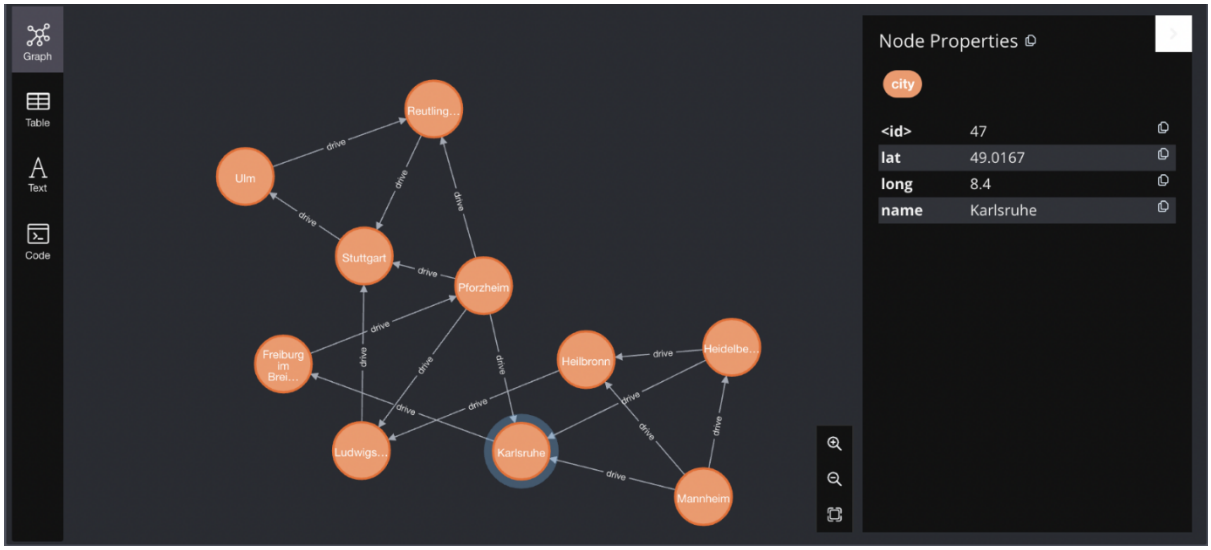
//Calculate driving distance between two points

drivingdistance.py > ...

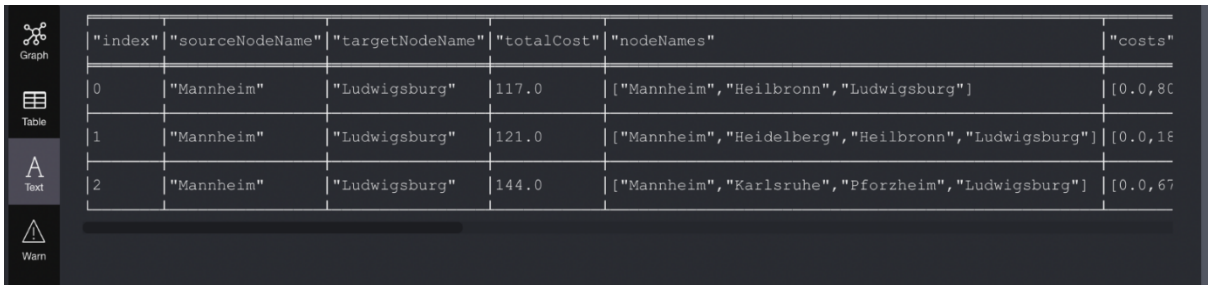
```python
import requests
import json
from geopy.geocoders import Nominatim
from geopy import distance
from numpy import place

geolocator=Nominatim(user_agent="geoapiExercises")

#place input
input_place1="Mannheim"
input_place2="Stuttgart"

#location of the place
place1=geolocator.geocode(input_place1)
place2=geolocator.geocode(input_place2)

#Get Latitude and Longitude

loc1_lat,loc1_long=(place1.latitude),(place1.longitude)
loc2_lat,loc2_long=(place2.latitude),(place2.longitude)

# call the OSMR API
long1,latt1=8.6820917,50.1106444 #frankfurt


#mannheim to heidelberg
r = requests.get(f"http://router.project-osrm.org/route/v1/car/{loc1_long},{loc1_lat};{loc2_long},{loc2_lat}?overview=false""")
# then you load the response using the json libray
# by default you get only one alternative so you access 0-th element of the `routes`
routes = json.loads(r.content)
# route_1 = routes.get("routes")[0]
route_1 = routes.get("routes")[0]
# print(route_1)

print("Distance betweeen ", input_place1,input_place2)
print(route_1.get("distance")/1000," km")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

(venv) (base) roshanscaria@Roshans-MacBook-Air Team-Diamond-main % /Users/roshanscaria/Desktop/Team-Diamond-main/venv/bin/python /Users/ros
hanscaria/Desktop/Team-Diamond-main/drivingdistance.py
Distance betweeen  Mannheim Stuttgart
133.1734  km
(venv) (base) roshanscaria@Roshans-MacBook-Air Team-Diamond-main %
```

MATCH(n) RETURN(n)




//To get the paths from a particular service point



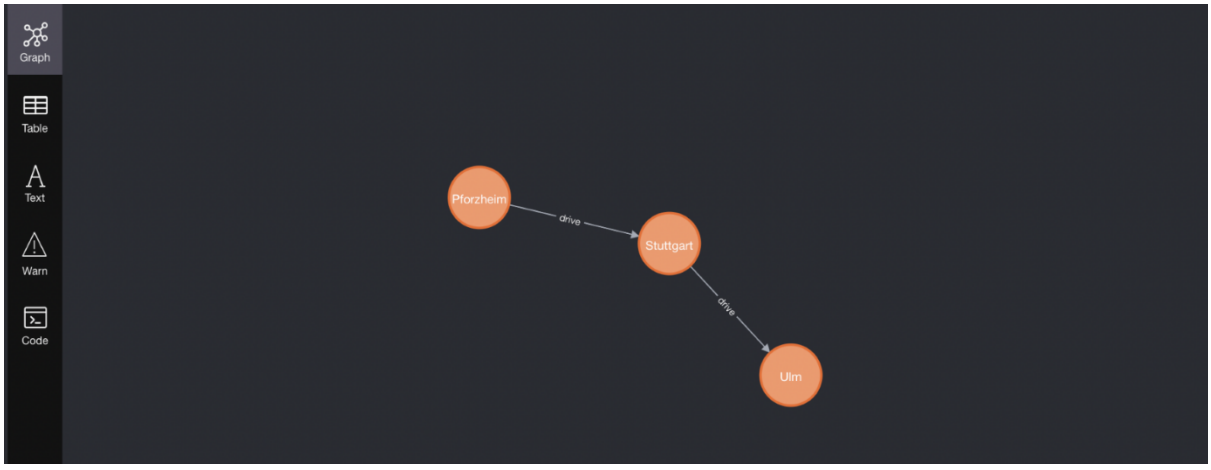
//To get the service point with maximum connections

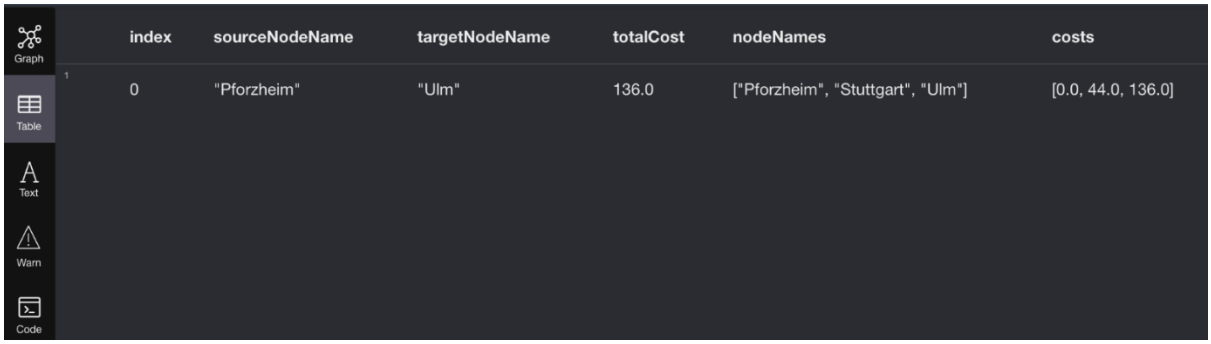
| | name | total_traffic |
|---|---|---|
| 1 | "Pforzheim" | 5.0 |
| 2 | "Stuttgart" | 4.0 |
| 3 | "Karlsruhe" | 4.0 |
| 4 | "Reutlingen" | 3.0 |
| 5 | "Mannheim" | 3.0 |

//To get the service point with least connections

| | name | total_traffic |
|---|---|---|
| 1 | "Ulm" | 2.0 |

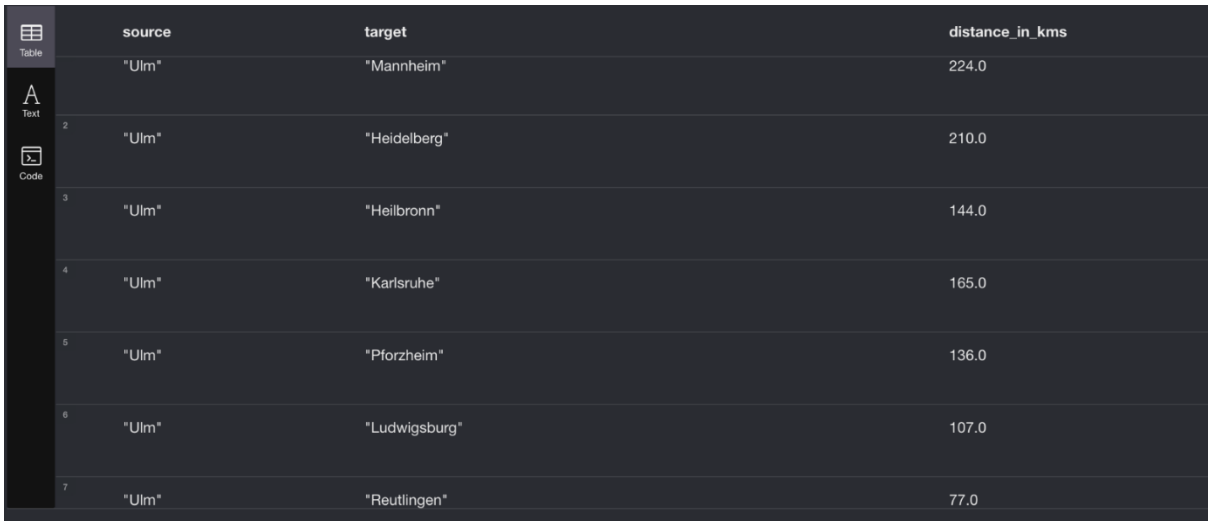
//To get the paths from a particular service point





| | index | sourceNodeName | targetNodeName | totalCost | nodeNames | costs |
|---|---|---|---|---|---|---|
| 1 | 0 | "Pforzheim" | "Ulm" | 136.0 | ["Pforzheim", "Stuttgart", "Ulm"] | [0.0, 44.0, 136.0] |

//Distance to other service centers from the least popular node



| | source | target | distance_in_kms |
|---|---|---|---|
| | "Ulm" | "Mannheim" | 224.0 |
| 2 | "Ulm" | "Heidelberg" | 210.0 |
| 3 | "Ulm" | "Heilbronn" | 144.0 |
| 4 | "Ulm" | "Karlsruhe" | 165.0 |
| 5 | "Ulm" | "Pforzheim" | 136.0 |
| 6 | "Ulm" | "Ludwigsburg" | 107.0 |
| 7 | "Ulm" | "Reutlingen" | 77.0 |

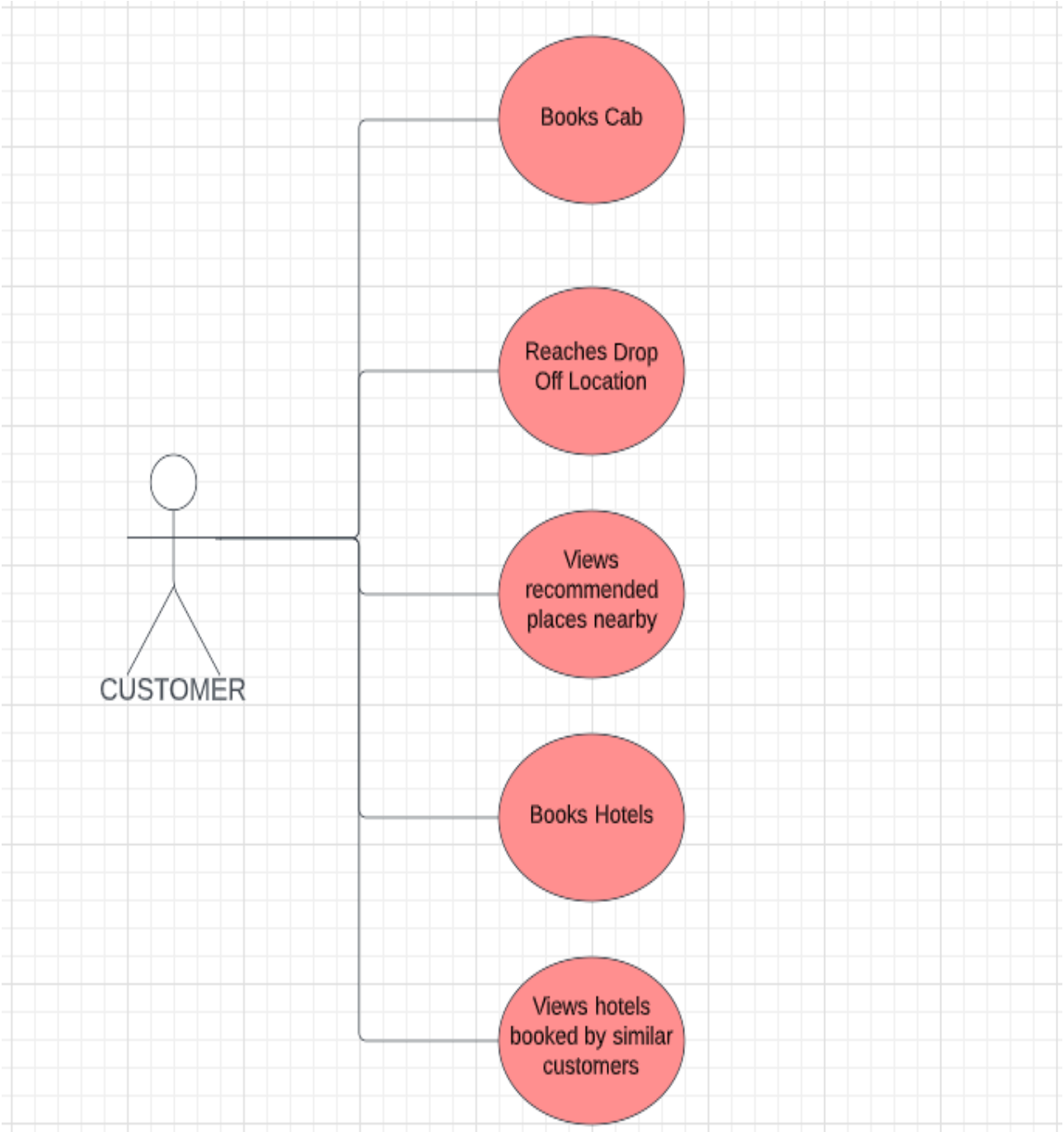
4.4 CAR RENTAL TIE-UPS (Monica):

4.4.1 USER STORY:

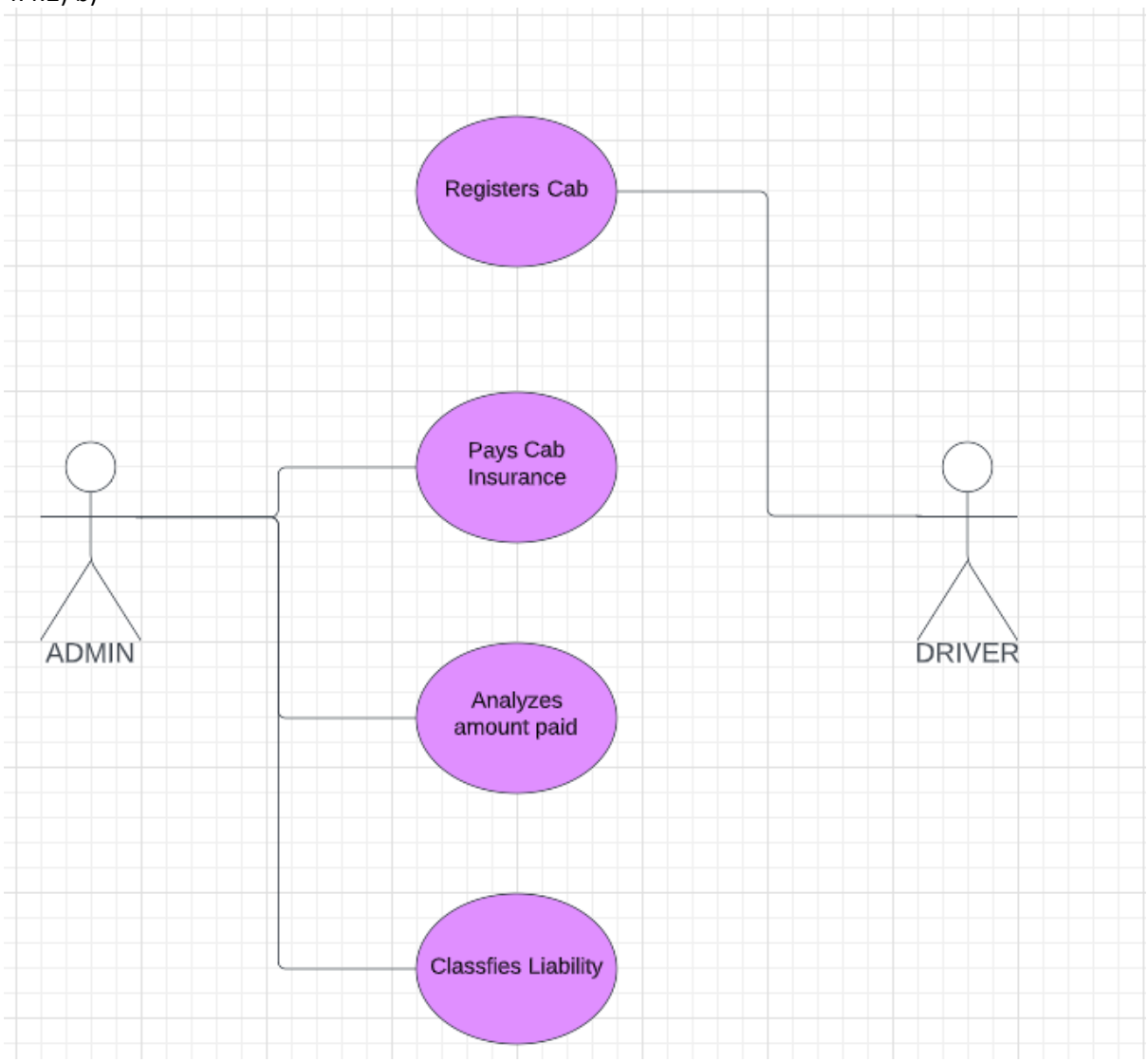
| ACTOR | USECASE |
|---|---|
| a. Customer | The car rental system has tie-ups with restaurants and tourist organizations. Whenever a customer is dropped off, we provide them with recommendations of the nearby places they can visit.<br>As a tourist, Jonas wants recommendations of restaurants, castles, or museums to visit near his drop-off place and recommendation of hotels booked by similar customers. |
| b. Admin | The Car Rental system has introduced a new platform where taxi drivers can register themselves and bookings done online through the app will be assigned to them. To attract more drivers the company has assured to take care of their car insurance. The admin now wants to analyze the cost bore by the company for this scheme and group them according to liability. |
| c. Customer | The car rental has a tie-up with PlixTrain, a private train chain. The customers of the car rental get 20 percent off on their train bookings when they book them through the rental app.<br>Amelie, as a customer wants to utilize the discount system offered by the car rental when she books PlixTrains through the Car rental app. |

4.4.2 IDENTIFIED USE CASE:

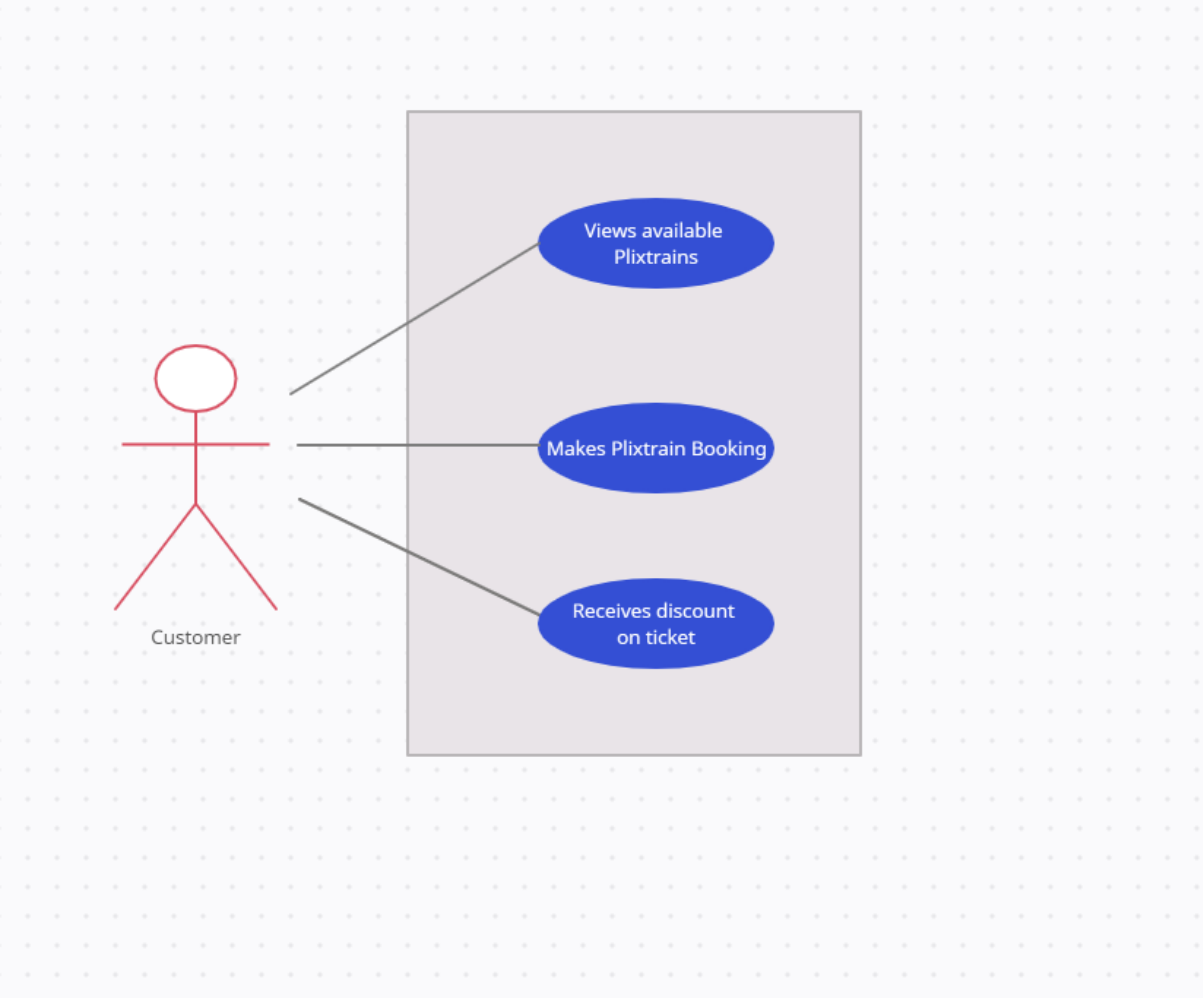
4.4.2) a):



4.4.2) b)


Registers Cab
Pays Cab Insurance
Analyzes amount paid
Classfies Liability
ADMIN
DRIVER

4.4.2) c)




4.4.3 ACTORS:

a) Customer: The customer gets recommendations of places to visit nearby his drop-off location, whenever he books a ride. He also gets recommendations based on the hotels other customers have booked.

b) Admin: Cab drivers can register their vehicles with the car-rental system. When a booking is done online, depending on their location rides will be allotted to them. The car rental pays for the car insurance of these cabs.
The admin analyses the insurance amount paid for these cabs and classifies them on a liability scale.

c) Customer: The car rental has a tie-up with a private train chain PlixTrain. If booked through the car rental app, the Plixtrain tickets can be obtained with a twenty percent discount. The customer initiates PlixTrain bookings and a booking is created.

4.4.4 DESCRIPTION:

a): Customers receive recommendations of restaurants, churches, pubs and castles based on their drop-off location. They also receive a recommendation of hotels based on what similar customers have booked.

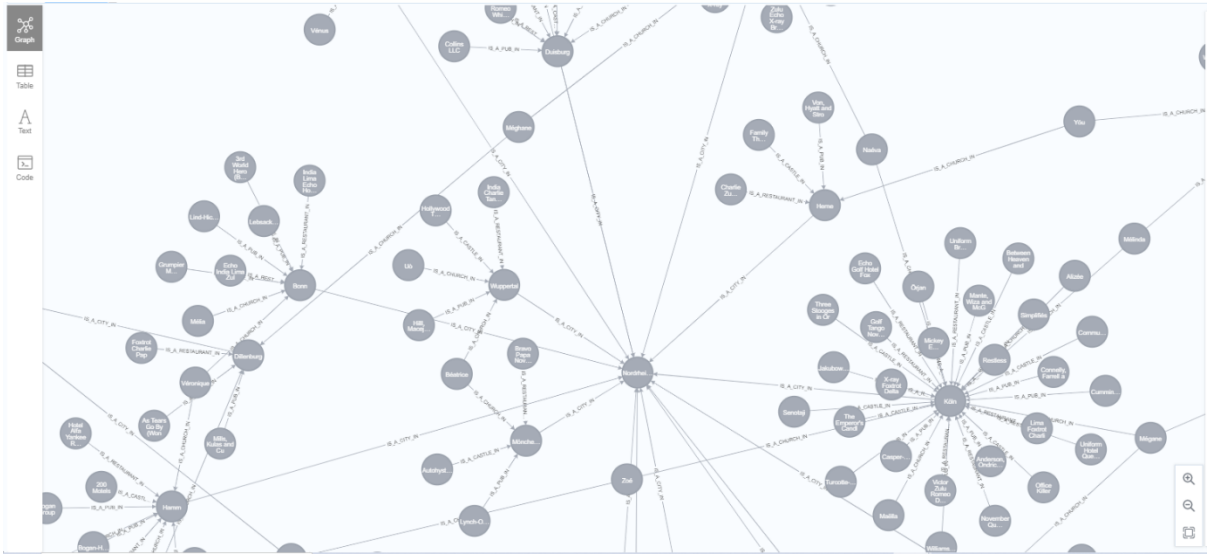
b): The car rental organization covers the car insurance for cabs registered with them. The Admin analyses the Insurance amount covered and classifies it according to liability.
c): Customers can book PlixTrains(a Private train organization) with a discount of twenty percent through the car rental app. The customer can select journey based on the departure location(From), arrival location(To) and departure time.

4.4.5 OPTIONAL: FRONTEND/COMMAND LINES USED TO REPRODUCE EXECUTION:

a)

//RECOMMENDATION BASED ON DROP-OFF LOCATION:

```
CALL apoc.load.json("file:///PlacesData.json")
YIELD value AS data
MERGE (n1:STATE {name:data.STATE})
MERGE (n2:LOCATION {name:data.CITIES})
MERGE (n3:RESTAURANTS{name:data.RESTAURANTS})
MERGE (n4:PUB{name:data.PUBS})
MERGE (n5:CASTLE{name:data.CASTLES})
MERGE (n6:CHURCH{name:data.CHURCHES})
MERGE(n2)-[r1:IS_A_CITY_IN]->(n1)
MERGE(n3)-[r2:IS_A_RESTAURANT_IN]->(n2)
MERGE(n4)-[r3:IS_A_PUB_IN]->(n2)
MERGE(n5)-[r4:IS_A_CASTLE_IN]->(n2)
MERGE(n6)-[r5:IS_A_CHURCH_IN]->(n2)
RETURN *
```



```
MATCH (n2:LOCATION{name:"Essen"})
MATCH (n2:LOCATION{name:"Essen"})-[:IS_A_CITY_IN]->(n3:STATE)<-
[:IS_A_CITY_IN]-(res:LOCATION)
MATCH (pubname:BIERGARTEN)-[:IS_A_PUB_IN]->(res:LOCATION)
MATCH (hotel:RESTAURANTS)-[:IS_A_RESTAURANT_IN]->(res:LOCATION)
MATCH (castle:CASTLE)-[:IS_A_CASTLE_IN]->(res:LOCATION)
```

```
MATCH (church:CHURCH)-[:IS_A_CHURCH_IN]->(res:LOCATION)
return  pubname.name as PubRecommendation, hotel.name as HotelRecommendation,castle.n
ame as CastleRecommendation,
church.name as ChurchRecommendation
LIMIT 3
```

```
1  MATCH (n2:LOCATION{name:"Essen"})
2  MATCH (n2:LOCATION{name:"Essen"})-[:IS_A_CITY_IN]→(n3:STATE)←[:IS_A_CITY_IN]-(res:LOCATION)
3  MATCH (pubname:PUB)-[:IS_A_PUB_IN]→(res:LOCATION)
4  MATCH (hotel:RESTAURANTS)-[:IS_A_RESTAURANT_IN]→(res:LOCATION)
5  MATCH (castle:CASTLE)-[:IS_A_CASTLE_IN]→(res:LOCATION)
6  MATCH (church:CHURCH)-[:IS_A_CHURCH_IN]→(res:LOCATION)
7  return  pubname.name as PubRecommendation, hotel.name as RestaurantRecommendation,castle.name as CastleRecommendation,
8  church.name as ChurchRecommendation
9  LIMIT 3
10
```

| | PubRecommendation | RestaurantRecommendation | CastleRecommendation | ChurchRecommendation |
|---|---|---|---|---|
| 1 | "Hudson-Ullrich" | "India Romeo Hotel Zulu Echo Delta X-ray Sierra Uniform Yankee Quebec Whiskey Juliett" | "Saint John of Las Vegas" | "Bécassine" |
| 2 | "Hudson-Ullrich" | "India Romeo Hotel Zulu Echo Delta X-ray Sierra Uniform Yankee Quebec Whiskey Juliett" | "Saint John of Las Vegas" | "Audréanne" |
| 3 | "Waelchi and Sons" | "India Romeo Hotel Zulu Echo Delta X-ray Sierra Uniform Yankee Quebec Whiskey Juliett" | "Saint John of Las Vegas" | "Bécassine" |

//RECOMMENDATION BASED ON WHAT OTHER CUSTOMERS HAVE BOOKED

```
1  LOAD CSV WITH HEADERS FROM 'file:///Tie_Up_Booking.csv' as TUHotelBooking
2  MERGE(n1:Customer{name:TUHotelBooking.CustomerName, id:TUHotelBooking.CustomerID})
3  MERGE(n2:Hotels{name:TUHotelBooking.HotelBooked})
4  MERGE(n1)-[:BOOKED]→(n2)
5  RETURN *
```

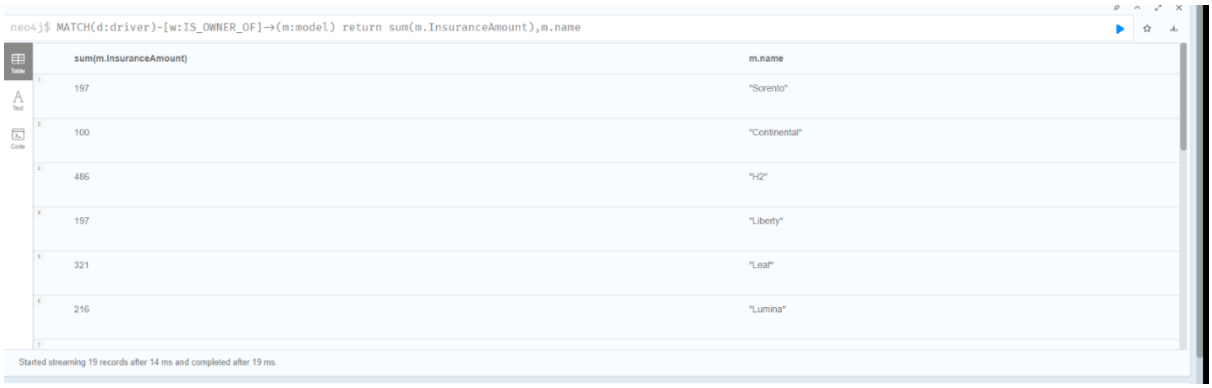
```
1  MATCH(n1:Customer{name:"Emmi"})-[:BOOKED]→(n2:Hotels)←[:BOOKED]-(u:Customer)
2  MATCH(u:Customer)-[:BOOKED]→(rec:Hotels)
3  WHERE NOT (rec)←[:BOOKED]-(rec)
4  return distinct(rec.name) as Hotelrecommendation
5
```

Table / Text / Warn

| | Hotelrecommendation |
|---|---|
| 1 | "IBIS Heidelberg" |
| 2 | "NH Heidelberg" |

b)
//Liability Assessment

```
CALL apoc.load.json("file:///cars_owners.json")
YIELD value AS data
MERGE(b:brand{name:data.Brand})
MERGE(m:model{name:data.Model,LicenseNo:data.LicenseNo,InsuranceAmount:data.CarInsurance})
MERGE(d:driver{name:data.owner,gender:data.Gender,YOJ:data.YearOfJoining})
MERGE(m)-[r1:IS_OF]→(b)
MERGE(d)-[r2:IS_OWNER_OF]→(m)
return*
```




```
MATCH(m:model)
SET m.liability = CASE WHEN m.InsuranceAmount <= 100 THEN 'LOW'
    WHEN m.InsuranceAmount>100 and m.InsuranceAmount<200 THEN 'MEDIUM'
    WHEN m.InsuranceAmount>200 THEN 'HIGH'
    ELSE 'NA'
    END
```

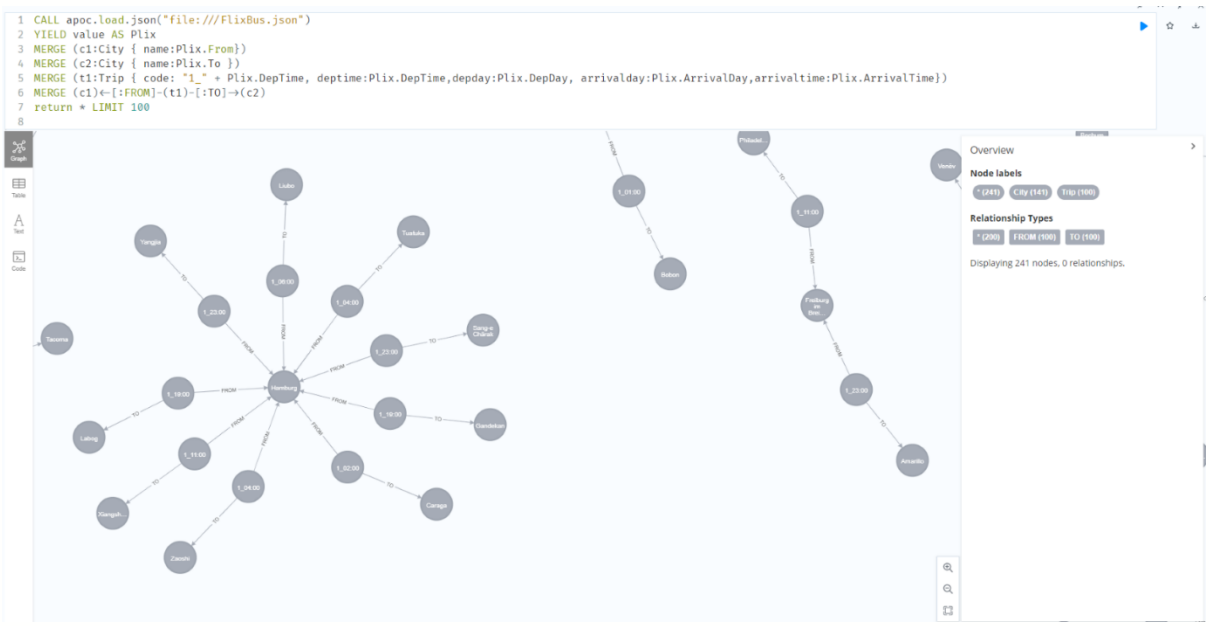
Table
Code

Set 20 properties, completed after 42 ms.

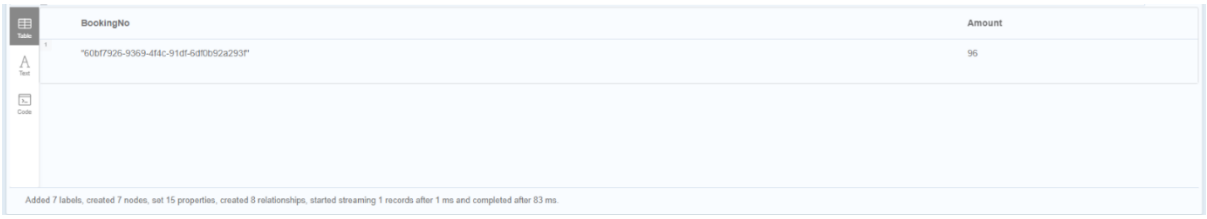
Set 20 properties, completed after 42 ms.

c)
//PlixTrain Booking through the car rental app

```
CALL apoc.load.json("file:///FlixBus.json")
YIELD value AS Plix
MERGE (c1:City { name:Plix.From})
MERGE (c2:City { name:Plix.To })
MERGE (t1:Trip { code: "1_" + Plix.DepTime, deptime:Plix.DepTime,depday:Plix.DepDay, a
rrivalday:Plix.ArrivalDay,arrivaltime:Plix.ArrivalTime})
MERGE (c1)<-[:FROM]-(t1)-[:TO]->(c2)
return * LIMIT 100
```

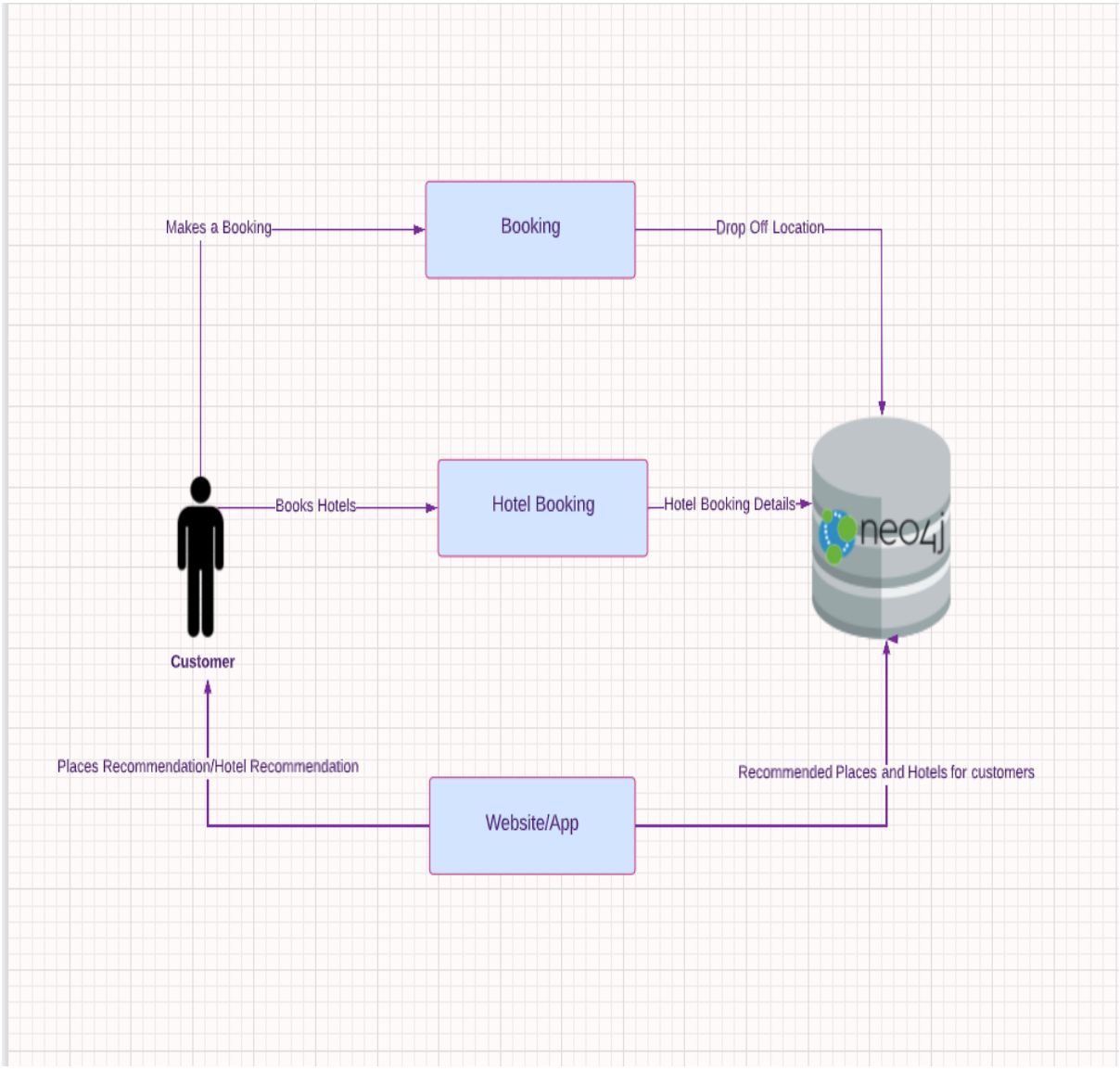



```
MATCH(c1:City{name:"Wuppertal"})<-[:FROM]-(tr1)-[:TO]->(c2:City{name:"Luwu"})
WHERE tr1.deptime="21:00"
MERGE(p1:Passenger{name:"Mina",passport:"ABC"})
MERGE(p2:Passenger{name:"Joseph",passport:"BBB"})
MERGE(t1:Ticket{seatNo:7, compartment:3})
MERGE(t2:Ticket{seatNo:8, compartment:4})
MERGE(b:Booking{createdAt: localdatetime(), BookingNo:apoc.create.uuid()})
MERGE(p:Payment{createdAt:localdatetime(), status:"Paid", amount:(120-
(120*20/100)), transactionCode:"XXX"})
MERGE (u:User { login: "min187" })
MERGE(u)-[:HAS]->(b)
MERGE(b)-[:PAID_METHOD]->(p)
MERGE(t1)-[:FOR]->(p1)
MERGE(t2)-[:FOR]->(p2)
MERGE(tr1)-[:HAS_RELEASED]->(t1)<-[:INCLUDES]-(b)-[:INCLUDES]->(t2)<-
[:HAS_RELEASED]-(tr1)
return b.BookingNo as BookingNo, p.amount as Amount
```

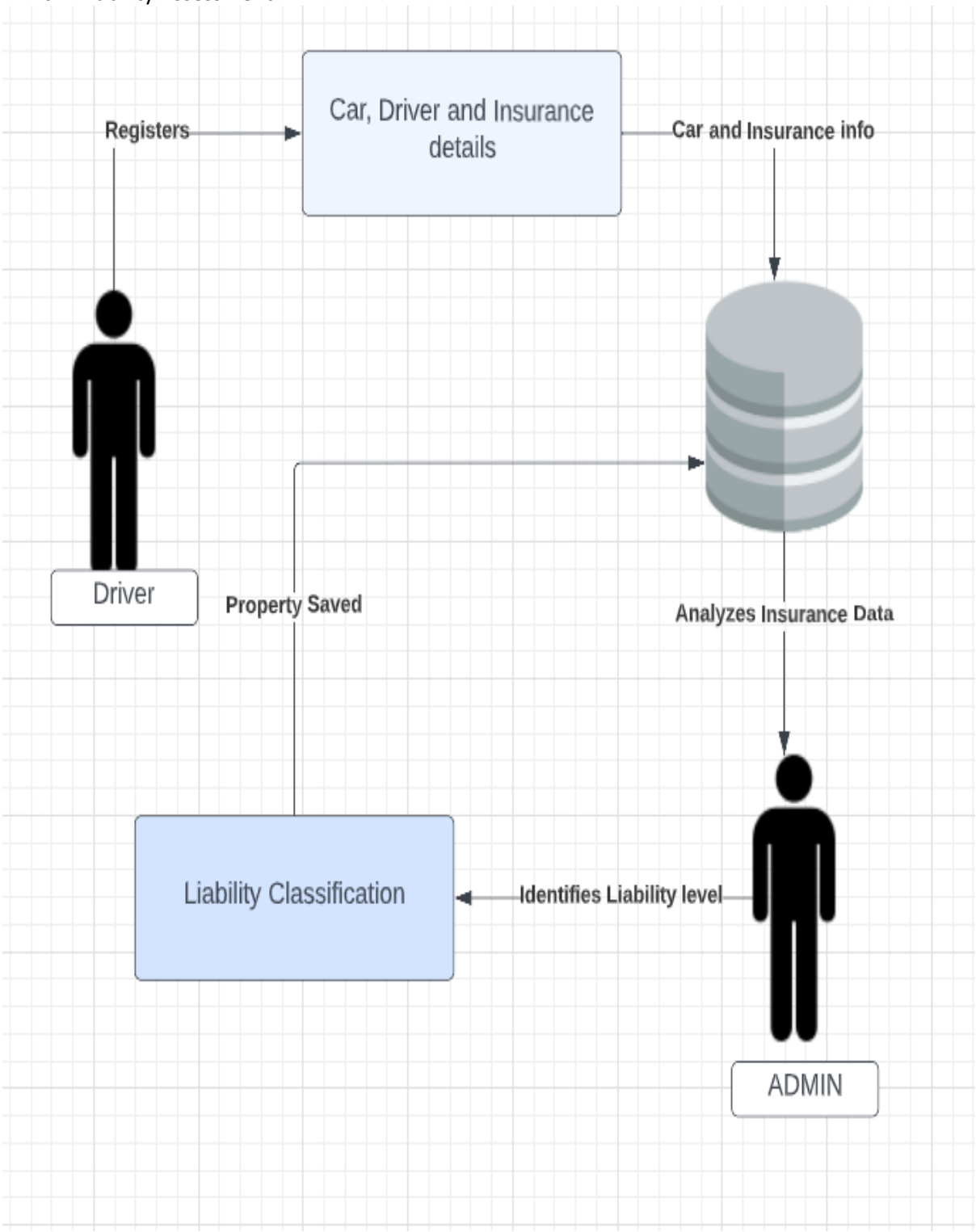



4.4.6 DATAFLOW:

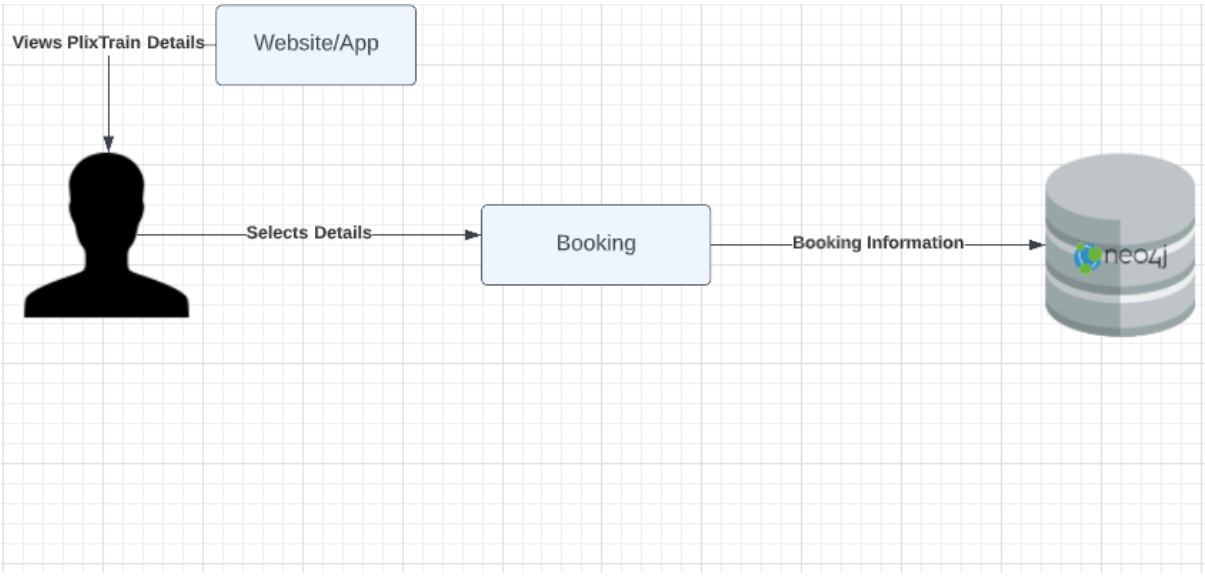
a. Recommendation:



b. Liability Assessment:



c. PlixTrain Booking:




4.4.7 DATABASE USED AND WHY:

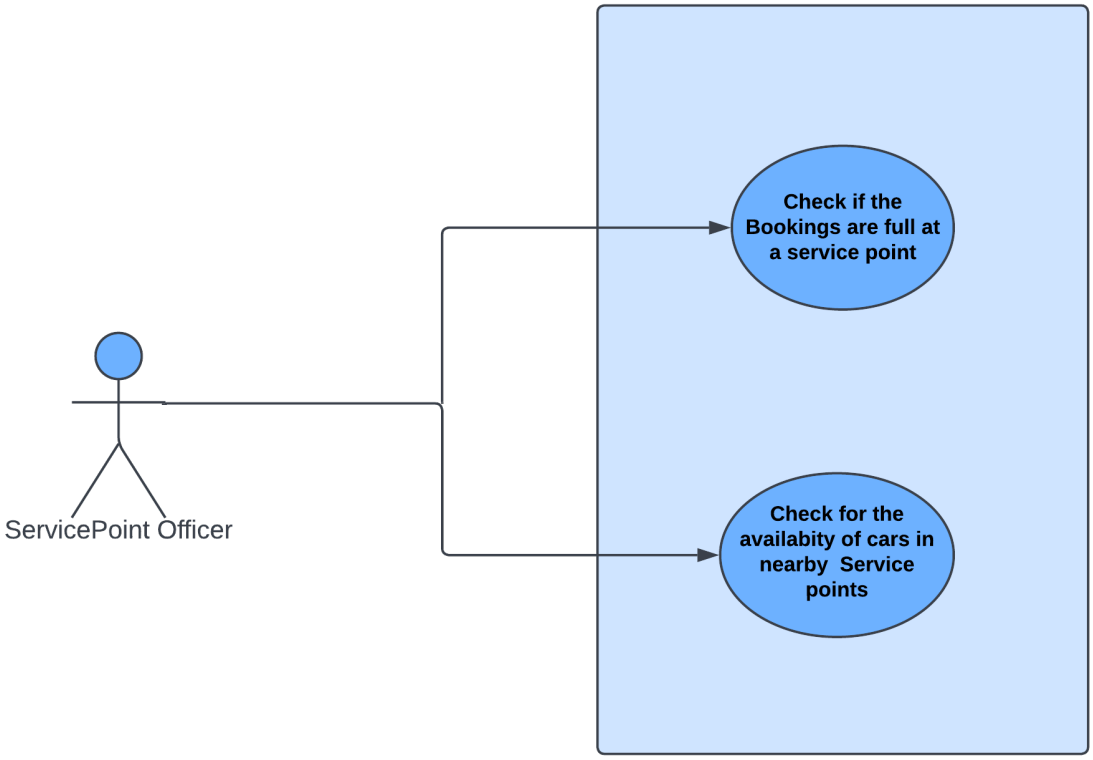
- In Neo4j we can calculate recommendations in real-time with the help of connected data.
- Neo4j's graph query language also provides the infrastructure to build recommendation algorithms.
- Graph technology is specifically designed to manage not just mountains of data but also relationships.

## 4.5 Cars available for booking at the current and nearby Service Points (Jerin Joshy)

## 4.5.1 User Story:

As a service point officer, when a customer comes to a service point to book a car, we need to check if the cars are available for booking and if not, find nearby service points where cars are available for the customer.

## 4.5.2 Identified Use case:




## 4.5.3 Actors

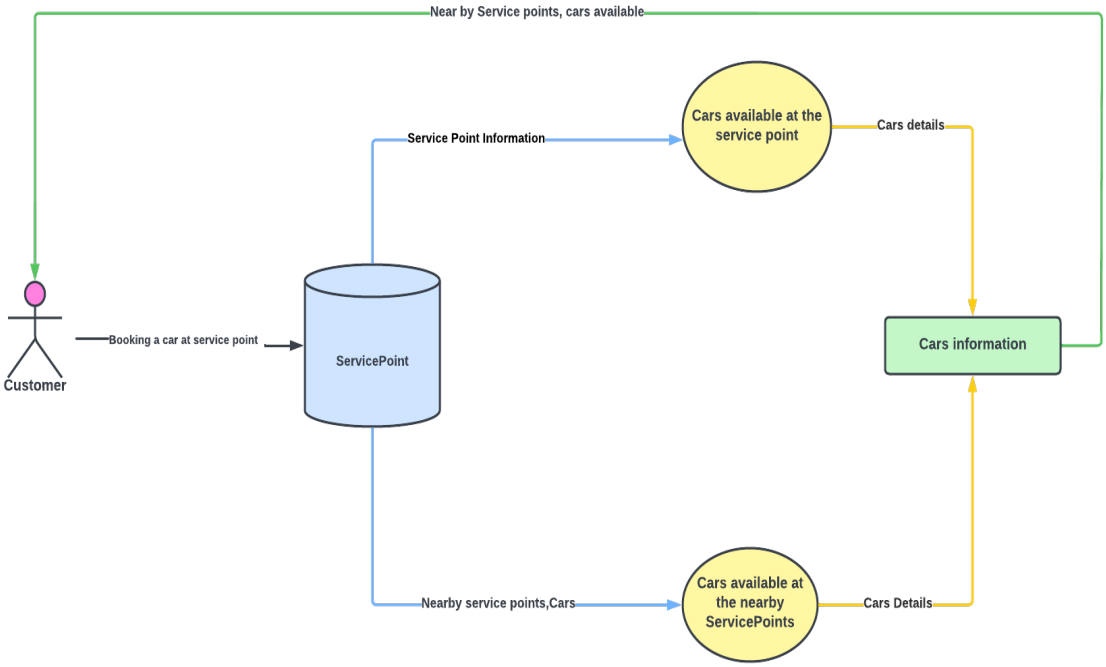
Service point Officer

Customer

## 4.5.4 Description:

The Car rental company have different service points in the country. Each Service points have a fixed number of cars available for the booking. If the bookings at a service points gets full, the company should suggest another nearby service point to the customer.

## 4.5.5 Data Flow Diagram:




## 4.5.6 Databases

### 4.5.6.1 Databases Used:

In a graph approach, the query results are much faster with minimum traversals between the nodes. Hence, the time taken to provide the results back to the user is considerably less as compared to a relational model. This is mainly because of the highly connected data. Thus, Neo4j along with MongoDB was chosen for this use case

### 4.5.6.2 Expressions used:

// pymongo expression to check if the cars booking is full at a Service point

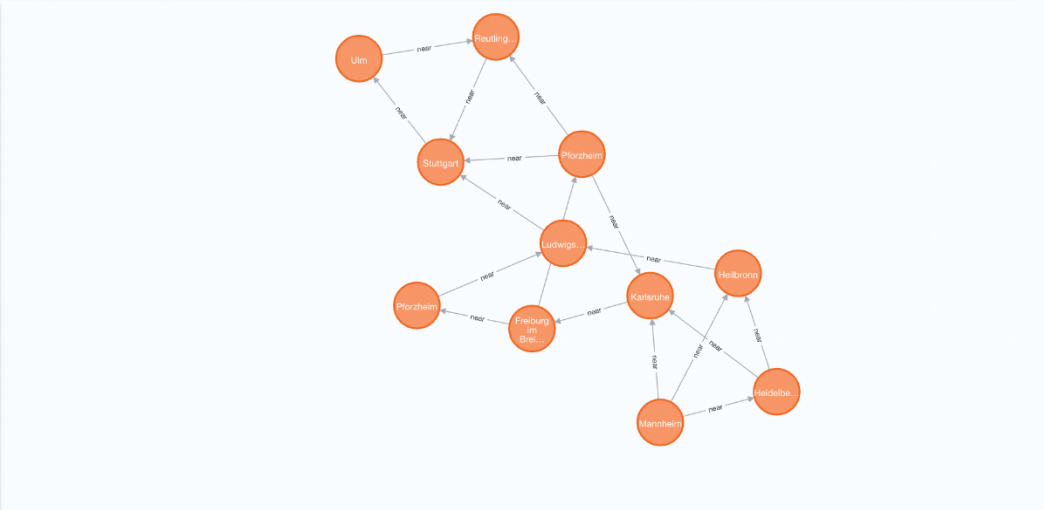
```python

import pymongo

# connect to mongodb from python using pymongo
client = pymongo.MongoClient("mongodb://localhost:27017")
# open the database
dbname = client['CarRental']
#print (dbname)
# get the collection
collection_name = dbname["ServicePoint1"]
#print(collection_name)
dbname.c
# get the data from the collection
item_details = collection_name.find()
#print(item_details)
#for row in item_details:
    #print(row)
#dbname.collection_name.find({ { “$eq”:  “Total Number of Cars” , “Number of Active Bookings” }})
#collection_name.find( [  { "CITY": 1, "Availcar": { "$subtract": [ "$TotalNumberOfCars", "$NumberOfActiveBookings" ]  } }  ] )

xy=collection_name.find( {
"CarsAvailable" : { "$eq" : 0}
            },{"CITY":1})
print(xy)
for k in xy:
    print(k)

```

//Find the nearby service points and cars available for booking at that service points

```
CALL gds.alpha.allShortestPaths.stream("road", {

relationshipWeightProperty: 'distance'

})

YIELD sourceNodeId, targetNodeId, distance

WITH sourceNodeId, targetNodeId, distance

WHERE gds.util.isFinite(distance) = true

MATCH (source:city) WHERE id(source) = sourceNodeId

MATCH (target:city) WHERE id(target) = targetNodeId

WITH source, target, distance WHERE source <> target AND source.name="Karlsruhe"

RETURN source.name AS City, target.name AS NearestCity, target.CarsAvailable as
Cars
```

Outcome:

```
jerinjoshy@Jerins-MacBook-Air test.py % /usr/local/bin/python3 /Users/jerinjoshy/sample.py
<pymongo.cursor.Cursor object at 0x104956fd0>
{'_id': ObjectId('62d8084a88be7207b34b6209'), 'CITY': 'Karlsruhe'}
```




```
neo4j$
```

```
1  CALL gds.alpha.allShortestPaths.stream("road", {
2  relationshipWeightProperty: 'distance'
3  })
4  YIELD sourceNodeId, targetNodeId, distance
5  WITH sourceNodeId, targetNodeId, distance
6  WHERE gds.util.isFinite(distance) = true
7  MATCH (source:city) WHERE id(source) = sourceNodeId
8  MATCH (target:city) WHERE id(target) = targetNodeId
9  WITH source, target, distance WHERE source <> target AND source.name="Karlsruhe"
10 RETURN source.name AS City, target.name AS NearestCity, target.CarsAvailable as
   Cars
```

Table | Text | Code

| | City | NearestCity | Cars |
|---|---|---|---|
| 1 | "Karlsruhe" | "Mannheim" | 8 |
| 2 | "Karlsruhe" | "Heidelberg" | 23 |
| 3 | "Karlsruhe" | "Heilbronn" | 39 |
| 4 | "Karlsruhe" | "Pforzheim" | 20 |
| 5 | "Karlsruhe" | "Ludwigsburg" | 39 |
| 6 | "Karlsruhe" | "Reutlingen" | 8 |
| 7 | | | |

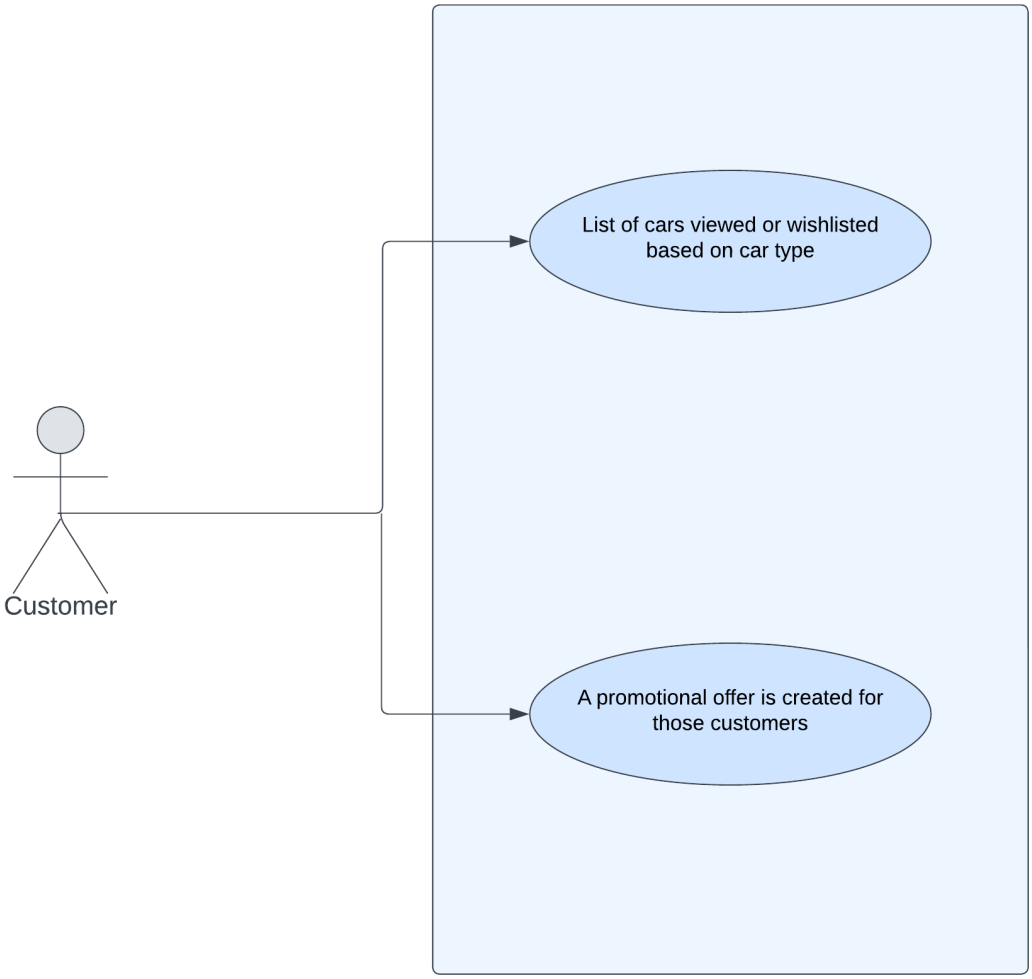
Started streaming 10 records after 182 ms and completed after 516 ms.

## 4.6 Creation of promotional offer based on customer preferences, (Roshan Scaria)

### 4.6.1 User Story

Based on the preferences of the customer a promotional offer is created for a product category. From the list of cars with the same type, that a customer has viewed or wish listed, a promotional offer is issued to those customers.

### 4.6.2 Identified Use Case



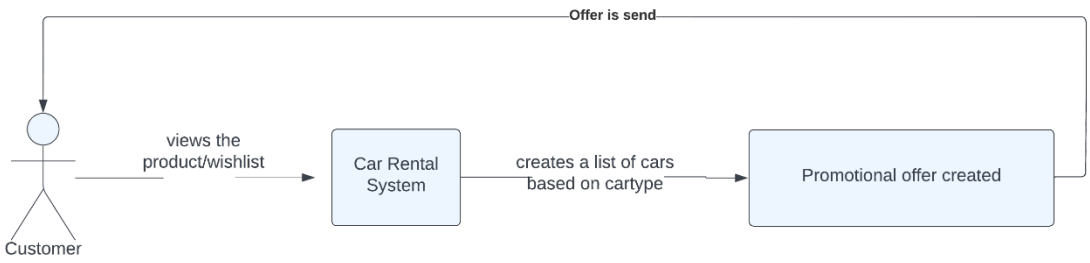
## 4.6.3 Actors

• Customers

## 4.6.4 Description

From the preferences of the customers by observing the customers wish list or cart. A discount offer is issued for the following cars based on the car category.

## 4.6.5 Data Flow

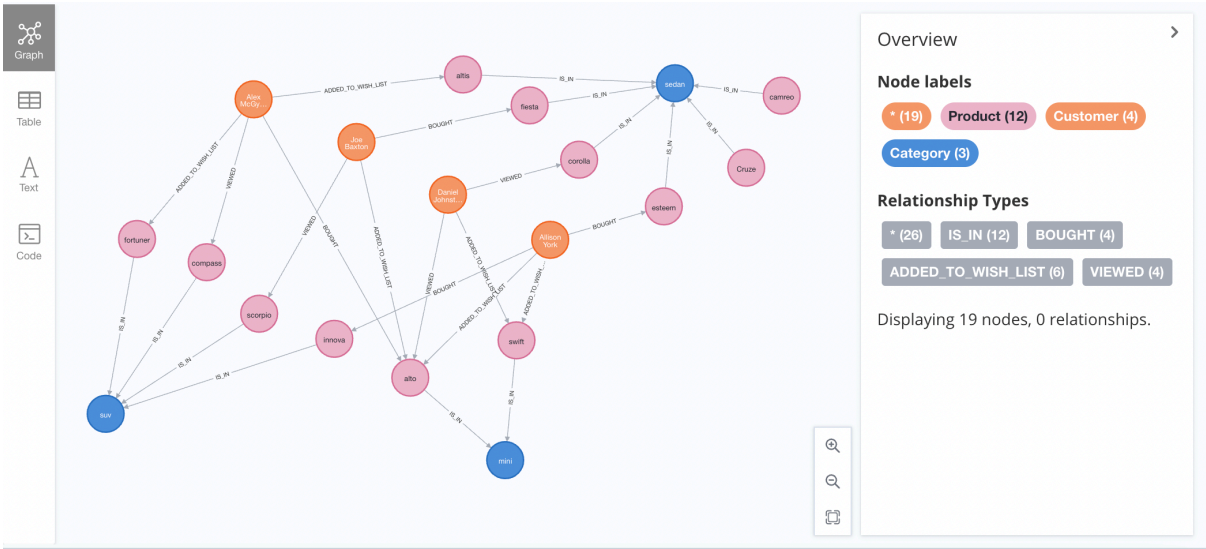



## 4.6.6 Databases

### 4.6.1 Databases Used:

Neo4j allows to create a graph but also to visualize data; this is helpful when it comes to creating an efficient email targeting campaign. Neo4j helps you model data and gain valuable insights.

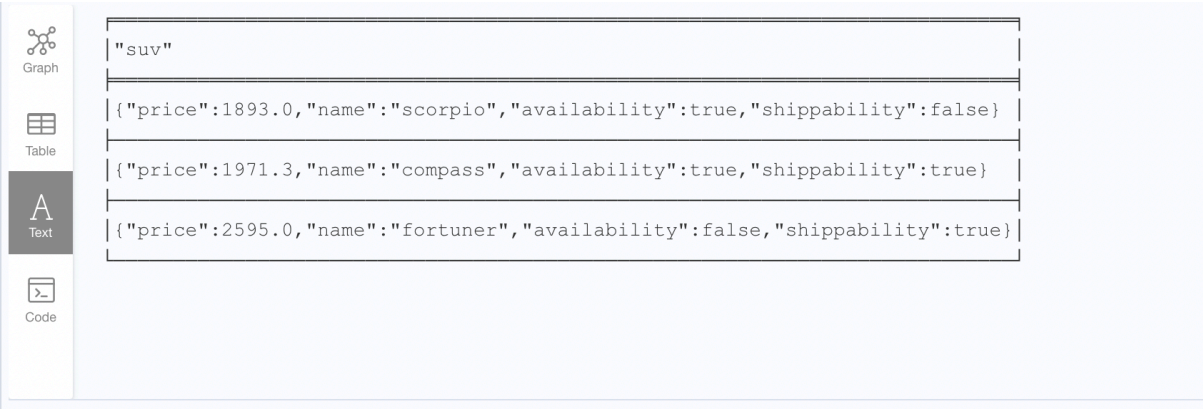
### 4.6.2 Expressions used:




//Obtain a list of suv that customers have viewed or added to their wish lists.

MATCH (:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(suv:Product)-[:IS_IN]->(:Category {name: 'suv'})

RETURN suv;



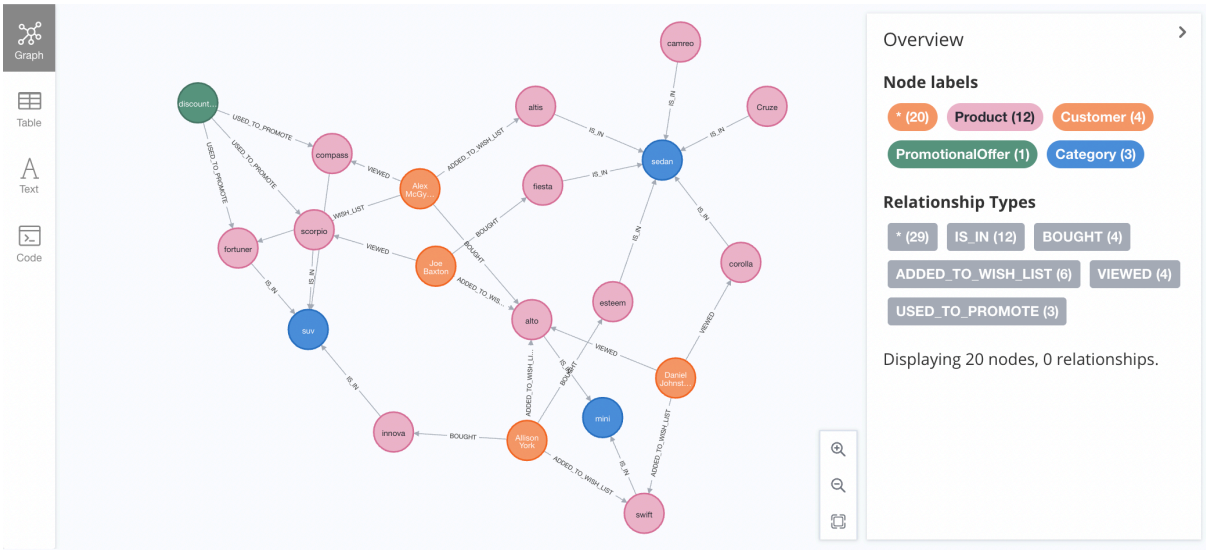

//Create promotional offer

CREATE(offer:PromotionalOffer {type: 'discount_offer', content: 'SUV discount offer...'})

WITH offer

MATCH (:Customer)-[:ADDED_TO_WISH_LIST|:VIEWED]->(suv:Product)-[:IS_IN]->(:Category {name: 'suv'})

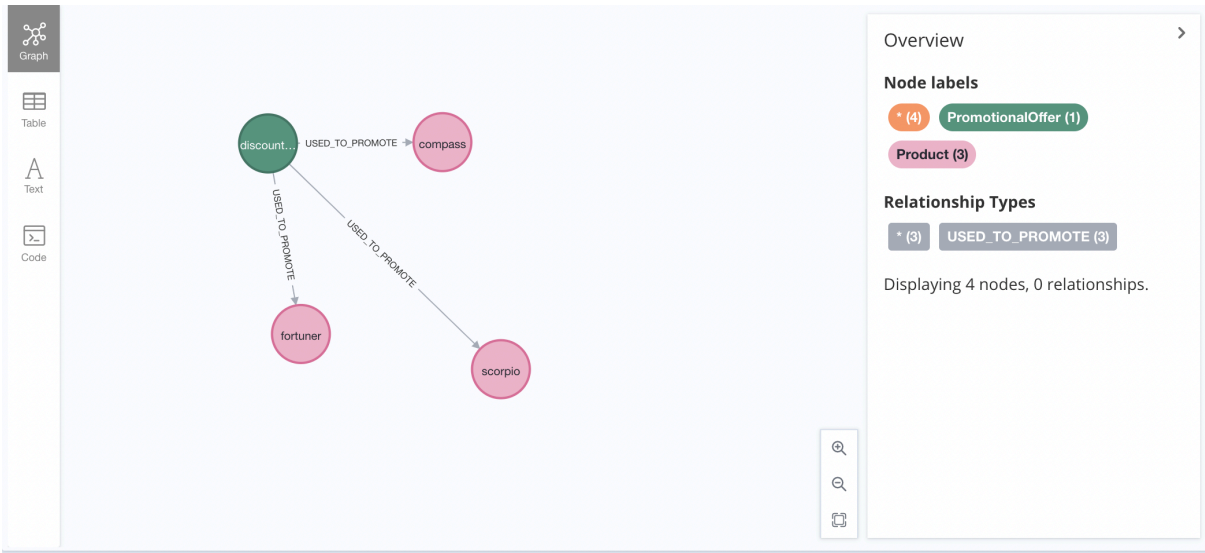
MERGE(offer)-[:USED_TO_PROMOTE]->(suv);




//See the cars in offer

MATCH (offer:PromotionalOffer)-[:USED_TO_PROMOTE]->(product:Product)

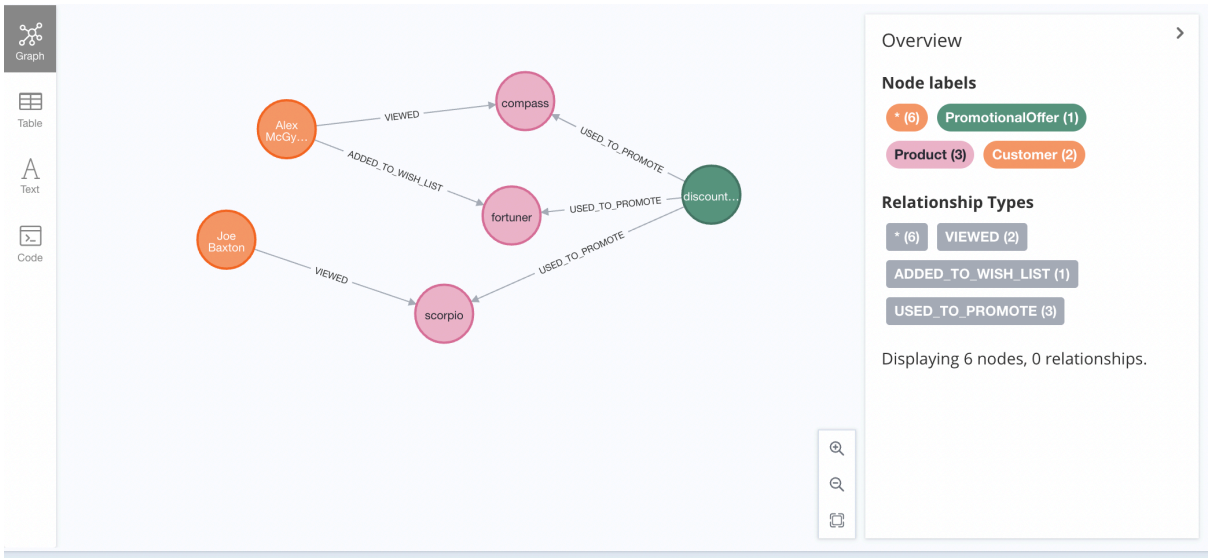
RETURN offer, product;



//View of customers who viewed the cars and has the promotional offer

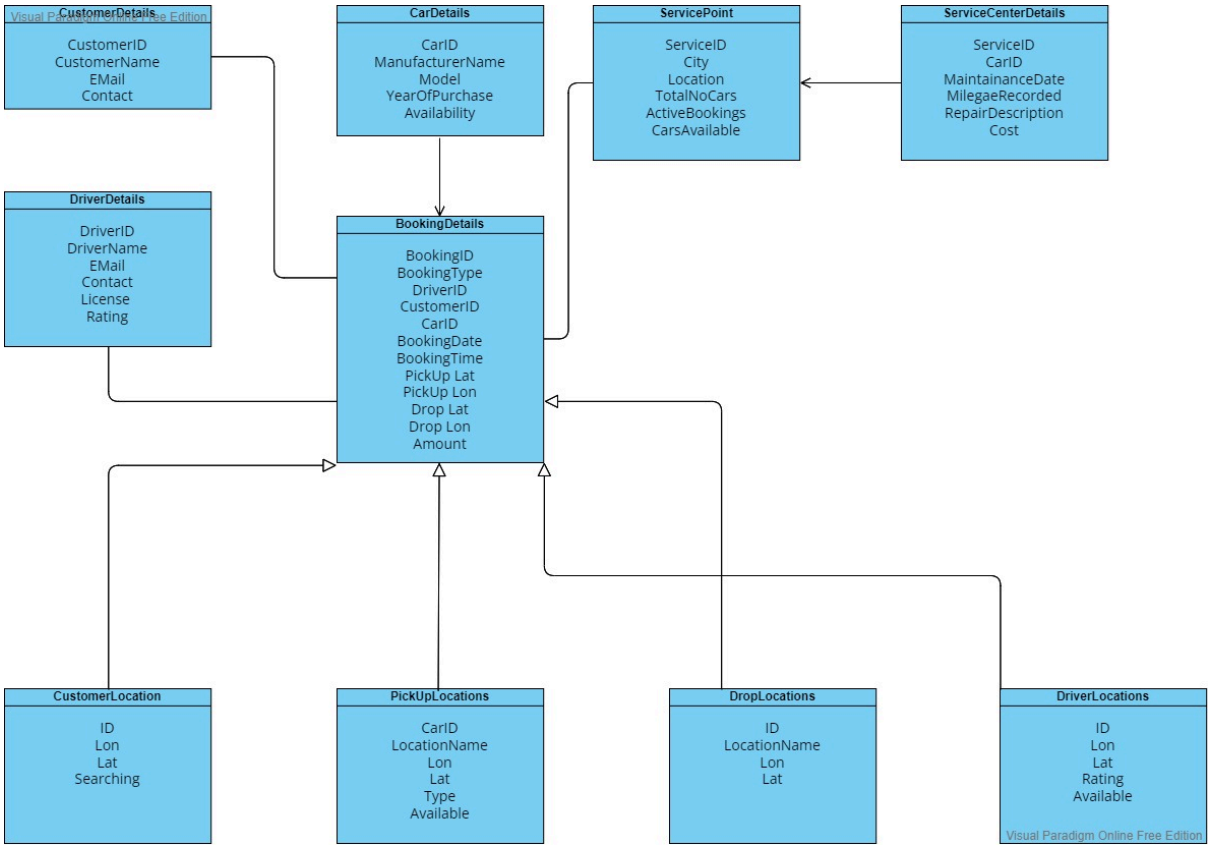
MATCH (offer:PromotionalOffer {type: 'discount_offer'})-[:USED_TO_PROMOTE]->(product:Product)<-[:ADDED_TO_WISH_LIST|:VIEWED]-(customer:Customer)

RETURN offer, product, customer;



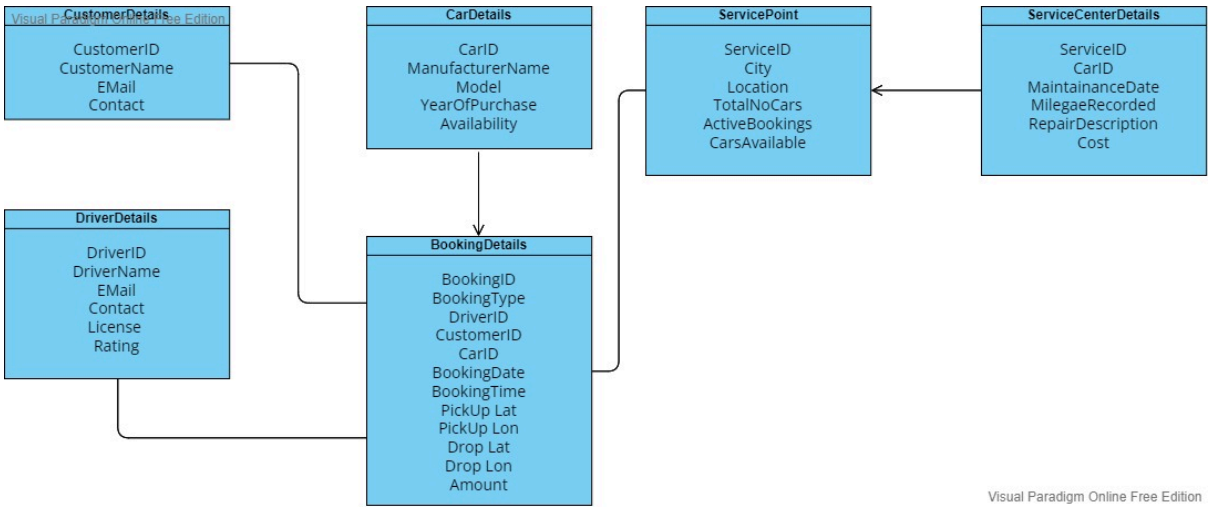
# 5. Database

## 5.1. Overall structure

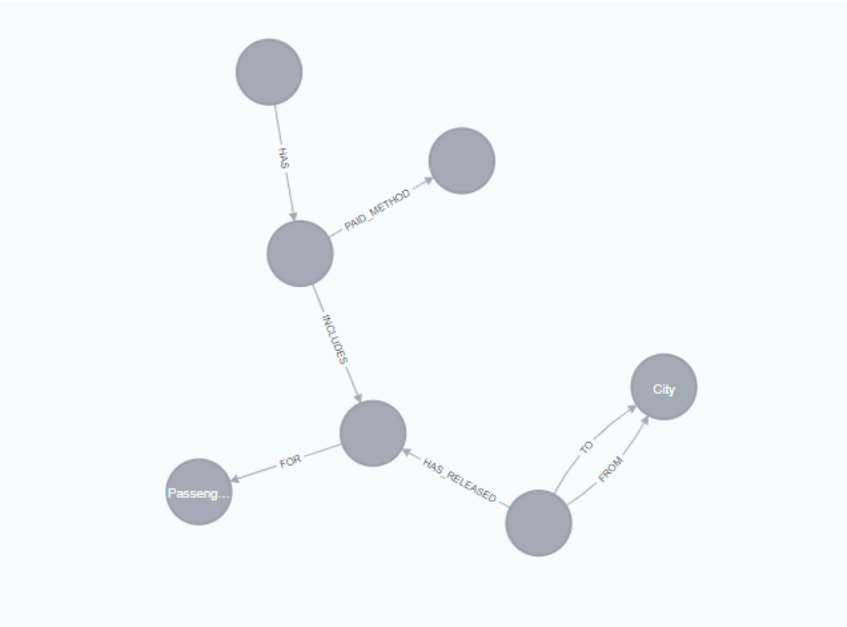


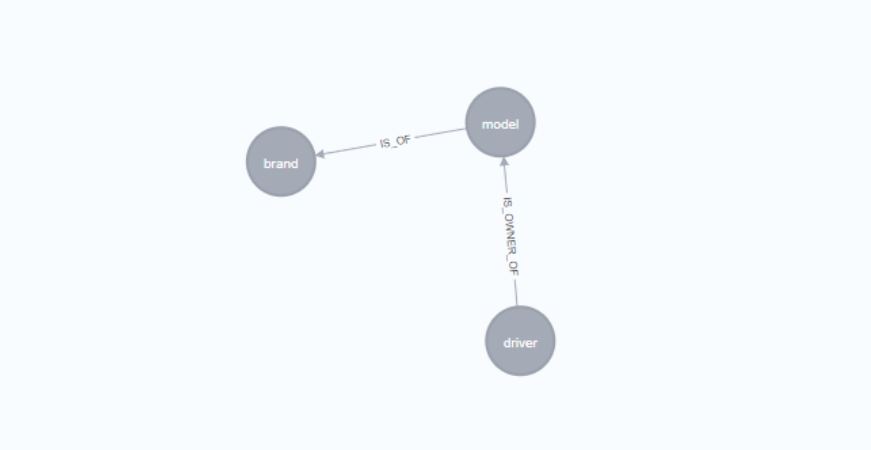

## 5.2. Data Model Overview all Databases

### 5.2.1. MongoDB

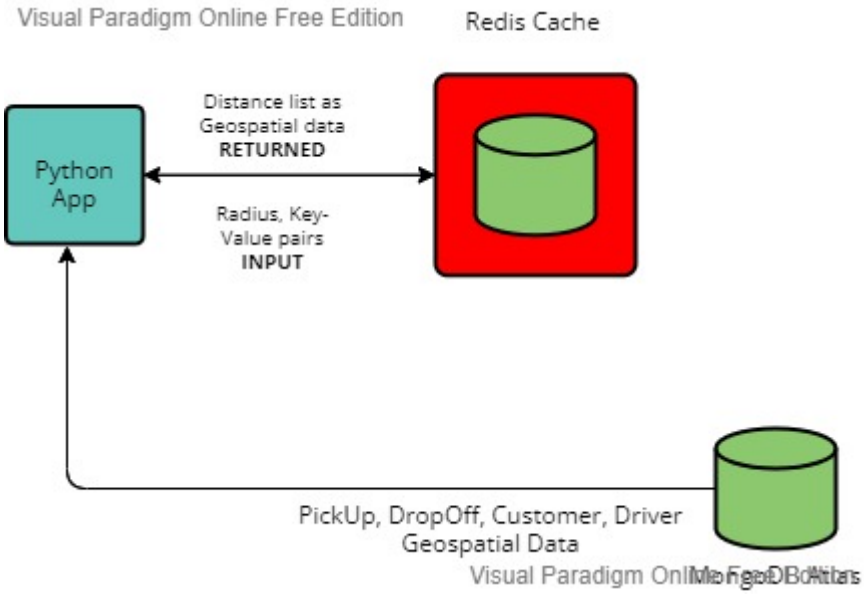


### 5.2.2. Neo4j With "CALL db.schema.visualization" command



### 5.2.3 Redis Key Value




# 6. Application

## 6.1 Language used

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. Python is used in our project for connecting to MongoDB and Neo4j and perform CRUD operations. It has Extensive support libraries like NumPy for numerical calculations, Pandas for data analysis.

## 6.2 GitHub path (Access information, NO source code in the documentation)

https://github.com/MonForever/Team-Diamond

# 7. API

## 7.1. Foreign API description

The Overpass API (formerly known as *OSM Server Side Scripting*, or *OSM3S* before 2011) is a read-only API that serves up custom selected parts of the OSM map data. It acts as a database over the web: the client sends a query to the API and gets back the data set that corresponds to the query. Unlike the main API, which is optimized for editing, Overpass API is optimized for

data consumers that need a few elements within a glimpse or up to roughly 10 million elements in some minutes, both selected by search criteria like e.g. location, type of objects, tag properties, proximity, or combinations of them. It acts as a database backend for various services.

### 7.1.1 Example data

```
(venv) C:\Users\Admin\PycharmProjects\tracking>python data_generation.py
No of Data Points Generated : 1038

API Data Example: [('251792140', 6.8669371, 45.9164989), ('251833375', 6.8731214, 45.9228079), ('255304318', 6.7395227, 45.8885766), ('281692631', 6.079482
2, 46.1414971), ('288541038', 6.8377974, 45.9333751), ('291332641', 6.6549092, 46.1481834), ('333196791', 6.4304781, 45.8752764), ('338473911', 6.7758786,
46.191848), ('339815098', 6.1319419, 45.9177072), ('357439231', 6.1300426, 45.9184003)]
```

# 8. Evaluation (Outlook, Lessons Learned, Possible Extensions)

We learned about different types of NoSQL databases and their advantages and disadvantages based on their features and user requirements in the Data Engineering module. We learned how to model a database by first developing a user story and identifying the actors within it. The story was then depicted in a diagram with users and the action performed on the database. To make it, we used a variety of tools. Then, to fulfil our user story, we created a database schema in a NoSQL database. We learned how to do data analysis and visualization using Databricks. Redis database provided fast processing and retrieval.

# 9. References

## 9.1. Books


- Graph Algorithms- Practical Examples in Apache Spark and Neo4j.


## 9.2. Webpages (Video, Tools, Documentation , ... )


- https://github.com/douglasmakey/tracking
- https://www.youtube.com/watch?v=IqBXKRrgy38&t=50s&ab_channel=Coco-XAnalytics
- https://www.youtube.com/watch?v=qftiVQraxmI&t=99s&ab_channel=Redis
- https://instanteduhelp.com/car-rental-management-system/
- https://community.neo4j.com
- https://neo4j.com/developer/python/
- https://docs.databricks.com/spark/latest/dataframes-datasets/introduction-to-dataframes-python.html
- https://www.mongodb.com/docs/manual/tutorial/geospatial-tutorial/

- https://medium.com/data-science-in-your-pocket/graph-analytics-pathfinding-algorithms-using-neo4j-c9bda1915077
- https://rubygarage.org/blog/neo4j-database-guide-with-use-cases#article_title_17
- https://redis.io/topics/quickstart
- https://redis.io/documentation
- https://redis-py.readthedocs.io/en/stable
- https://www.mockaroo.com/
- https://docs.mongodb.com/
- https://pymongo.readthedocs
- https://janakiev.com/blog/openstreetmap-with-python-and-overpass-api/#using-the-overpass-api
- https://wiki.openstreetmap.org/wiki/Overpass_API
- https://www.kaggle.com
- www.w3schools.com