Car Pooling Service

Data Engineering
2021-2022

Prepared by: Team Yellow

Authors:

Nissy	11017563
Monica Shashidhar	11017560
Omer	11016632
Sasha	11017163

SRH Hochschule Heidelberg Fakultät für Information, Medien und Design

Contents

1. Introduction

Carpooling services is sharing of car journeys so that more than one person travels in a car, and prevents the need for others to have to drive to a location themselves. By having more people using one vehicle, carpooling reduces each person's travel costs such as: fuel costs, tolls, and the stress of driving. Carpooling is also a more environmentally friendly and sustainable way to travel as sharing journeys reduces air pollution, carbon emissions, traffic congestion on the roads, and the need for parking spaces. Also travelling alone may be stressful, so having other persons with you on a trip reduces the stress and is also the occasion to socialize and make the trip funnier.

Finding people to share a ride with is the challenge of carpooling as it is difficult to find a person going to the same place as you at a given time. Many websites and applications has been developed to help people meet to share rides. Those applications enable users to create and share their trip and find passengers. As part of this project, we are trying to recreated some of the real time scenario faced by a carpooling service and provide solutions mainly focusing on the non-sql databases like Redis, Mongo DB and Neo4j.

2. Organization

2.1. Roles and responsibilities

Student	Omer	Monica	Sasha	Nissy
Traveller	RDO			
Vehicle Owner		RDO		
Help Desk			RDO	
Financial and Operation Services				RDO
Neo4j				D
MongoDB	D	D	D	
Redis	D	D	D	

Legend:

Responsible = R Developing = D Test = T Assisting = A Documentation = O

2.2. Meetings log:

Date	Topic	Attendees
Saturday, January 22, 2022	Project kick off meeting-Deciding on the project subject	Nissy,Omer,Monica,Sasha
Sunday, January 23, 2022	Discussing use cases and decing the rolles and responsibilitie, Decided to use lablah carpool site API	Nissy,Omer,Monica,Sasha
Monday, January 24, 2022	Deciding on the use cases and senarios	Nissy,Omer,Monica,Sasha
Tuesday, January 25, 2022	discussion on database types for choosing suitable database for each usecase	Nissy,Omer,Monica,Sasha
Wednesday, January 26, 2022	chacking diagrams	Nissy,Omer,Monica,Sasha
Wednesday, January 26, 2022	Finalised the project topic and asked queries related to the project use cases to the professor	Nissy,Omer,Monica,Sasha,Prof. Frank
Thursday, January 27, 2022	building dataflow diagrams with visio, discussion on use cases	Nissy,Omer,Monica,Sasha
Thursday, January 27, 2022	Wrote an initial draft of user stories	Nissy,Omer,Monica,Sasha,Prof. Frank
Saturday, January 29, 2022	Discussed about data bases to be created by Mockaroo site	Nissy,Omer,Monica,Sasha
Tuesday, February 1, 2022	Checking created data. Discussing Queries and decission made on using MongoDB Atlas	Nissy,Omer,Monica,Sasha
Wednesday, February 2, 2022	Checking python code. Geospatial function check. MongoDB queries overview	Nissy,Omer,Monica,Sasha
Wednesday, February 2, 2022	Based on the suggestion shared we shared the UML diagram and DB to be used for the given userstories	Nissy,Omer,Monica,Sasha,Prof. Frank
Friday, February 4, 2022	Redis library and Redis python code review session	Nissy,Omer,Monica,Sasha
Saturday, February 5, 2022	Working on Spark Databricks	Nissy,Omer,Monica,Sasha
Wednesday, February 9, 2022	Working on Spark Databricks. Analysis with python and visualization with matplotlib	Nissy,Omer,Monica,Sasha
Thursday, February 10, 2022	Working on Spark Databricks. Analysis with python and visualization with matplotlib	Nissy,Omer,Monica,Sasha
Thursday, February 10, 2022	Q&A with Prof. Frank	Nissy,Omer,Monica,Sasha,Prof. Frank
Thursday, February 10, 2022	Final preparations	Nissy,Omer,Monica,Sasha

2.3. Tools used

- Lucid Chart - Use Case and Interaction Diagram
- Microsoft Vissio- Interaction and Data Flow diagram
- VS Code - Integrated development environment
- Wondershare EdrawMax- Data Flow Diagram
- PyCharm- Integrated development environment
- Neo4j Desktop
- Redis-CLI
- MongoDB Atlas
- MongoDB compass
- Studio 3T
- Docker
- Python Packages:
    - pyMongo
    - redis
    - follium
    - flask
    - geodesic

3. User Stories

Use Case #	Actor	User Story
Use Case -1	**Traveller** **Proper planning**	1.1.Want to see the list of available cars before travelling
		1.2.Want to see the estimated arrival time before travelling
		1.3.Want to see the nearest pickup points for proper planning
		1.3.Want to see the nearest pickup points for proper planning
Use Case -2	**Vehichle Owner** **Flexible and Safe Travel**	2.1. Nick wants to offer ride shares for travellers to avoide traffic and pollution.Nick wants to know the travellers identity, destiation and the pick-up location, date and time before accepting the ride request. Jewel wants to offer ride for only female travellers according to her comfort. Hence jewel wants to choose for a women only ride option to target only female travellers.
		Communication - As Nick got stuck in his meeting, he wants to let the travellers know that he will be late by half an hour to pick them up from their pick-up location. This means that if there are any slight changes in the plan, he wants to keep his travellers updated on the same.
Use Case -3	**Help Desk**	3.1.Receive Car crash notifications from Traveller
		3.2.Access a list of cars and their location- Help Desk user would like to have access to list of all cars withithn custom radius from the user who request for help
		3.3.Dispatch new car to User location
		3.4.send update message, providing arival time estimation to user and provide arrival
		3.5.Send message to available cars
Use Case -4	**Financial and operation services**	4.1.Sam who works in the financial & Operation services department needs to send pooling recommendations based on the location , trip type, Num of co-passengers, car type and expected expense for all the users
		4.2.Sam who works in the financial & Operation services department need to sort routing problems in faced in car pooling
		4.3.Sam wants to resolve a problem where the travellor completes the payment but missed the pooling. He needs to get assigned with new car pooling instead of refund.

4. Use Case-1 Traveller (Omer)

4.1. User Story

Traveller is a person who wants a car and who is willing to travel looking for car-sharing to reach him/her/them destination pick-up points. Travellers basically will be travelling from X to Y, who will be willing to join available rides for car´s owners requests .

-Hakan want to see list all available car from X city to Y city

-Hakan wants to see estimated time before travel for proper travelling

-Hakan wants to see total distance before travelling.

-Hakan wants to see nearest pick up point before travel.

4.2. Actors

4.2.1. User Traveller:

The person who willing to travel with offering shared rides with cars, who are looking for rides in the same route and time as his.

4.2.2. User Vehicle Owner:

The person who will be offering shared rides to travellers, who are looking for rides in the same route and time as his.

4.3. Use Case Diagram for Use Cases-1

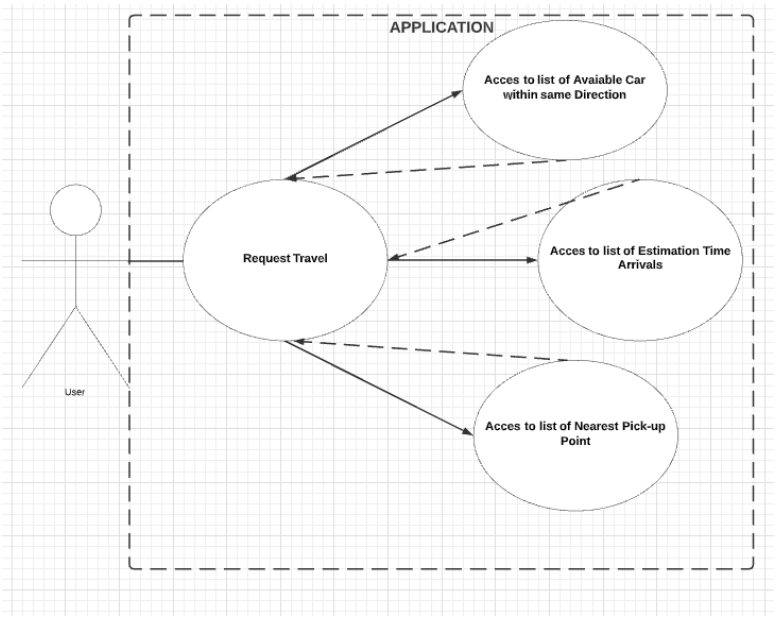

Figure 4.3-1

4.4. Interaction Diagram for Use Cases-1

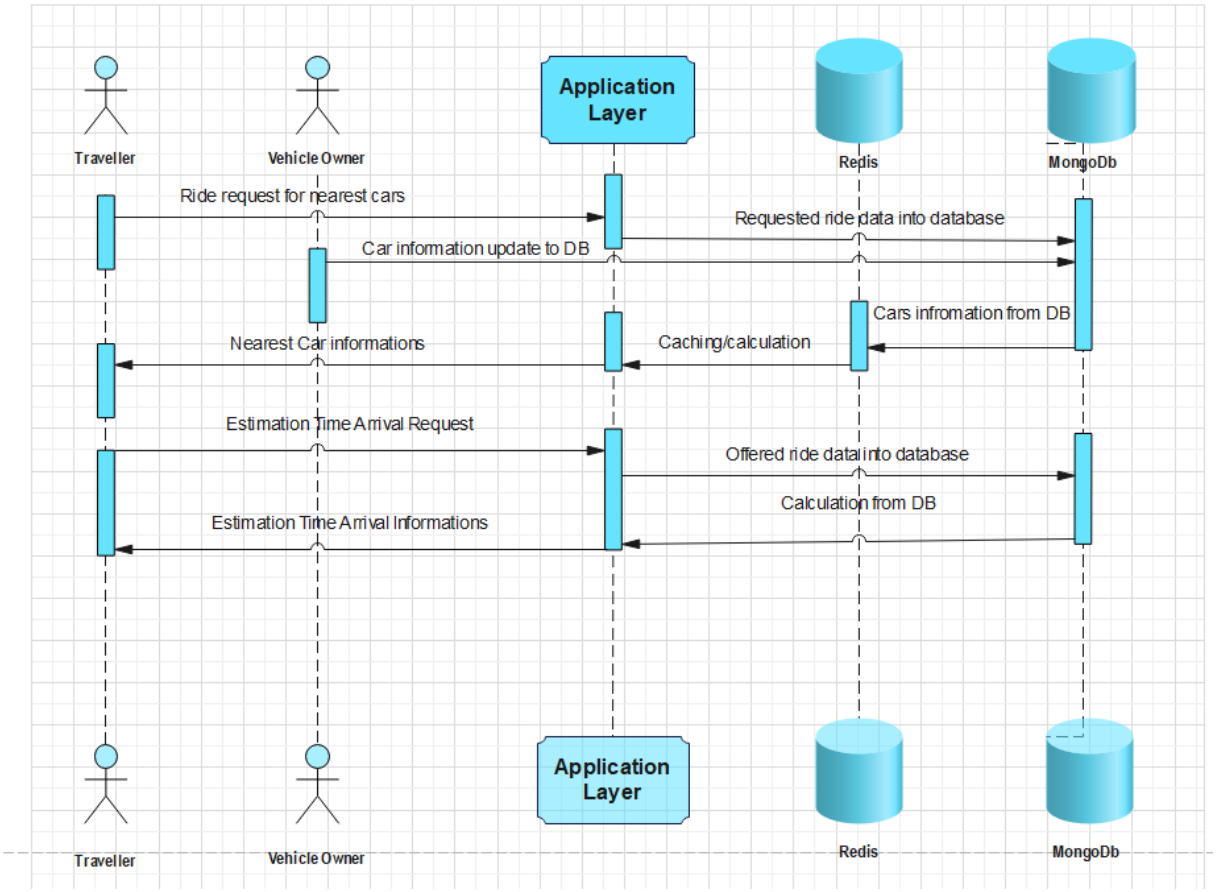

Figure 4.4-1 Descriptions of task with all interactions between actor and database

4.5. Data Flow Diagram for Use Case-1

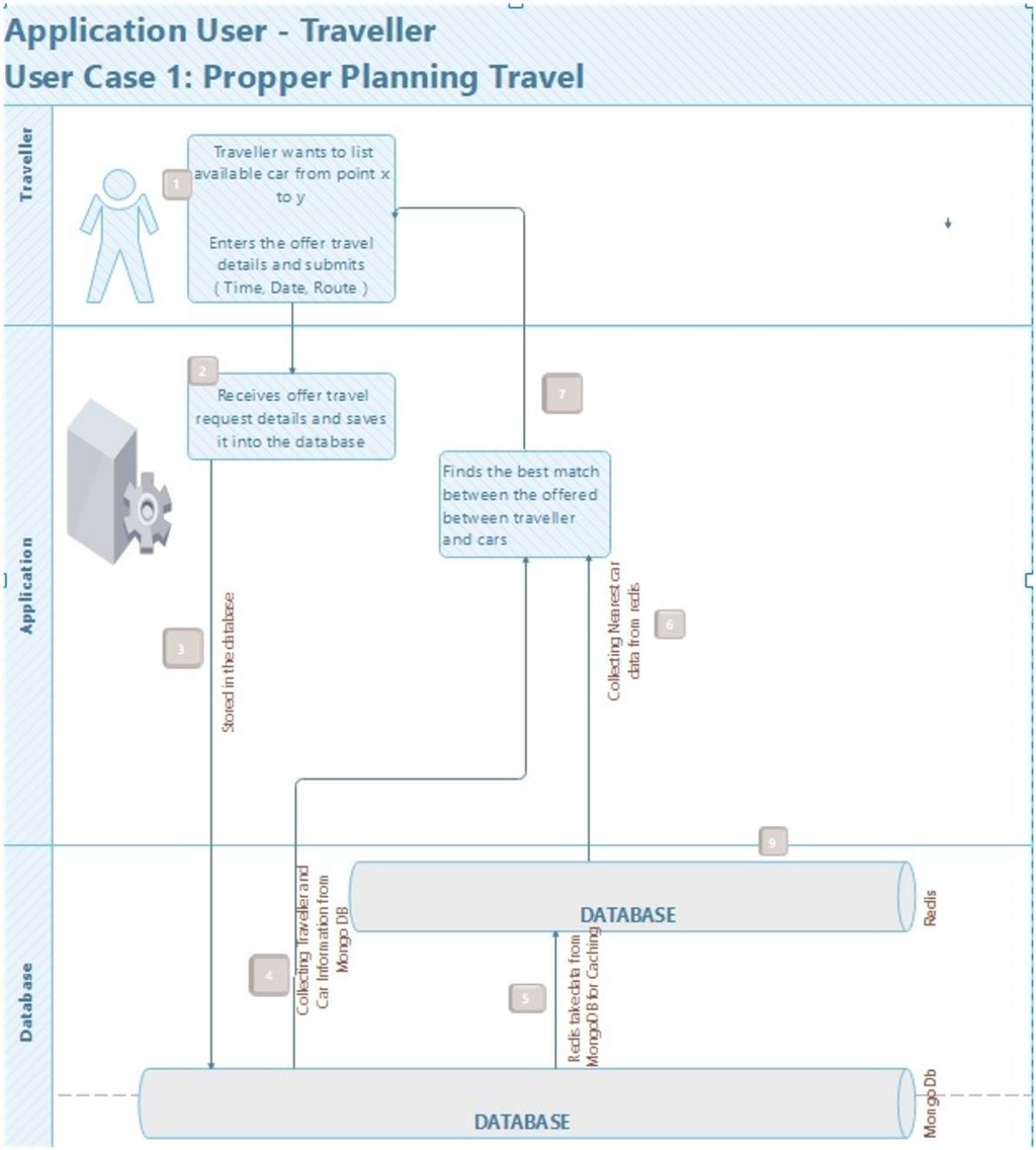

Figure 4.5-1

4.6. DataBases for Use Case -1

MongoDb
Redis

MongoDB is used to store Traveller, Car and Trip data in two separate collections. All the data is stored in MongoDB cloud for easy access of team in development phase.MongoDb has big code-native data Access.

Redis is used to cache mongoDB data because of it's efficiency in Geospatial calculations .

4.6.1. Expressions used for this use case

```
#DataBases connection with MongoDbAtlas and Redis
client = MongoClient("mongodb+srv://mongo:omer123@cluster0.kmfv3.mongodb.net/myFirstDatabase?retryWrites=true&w=majority")
r=rd.Redis('127.0.0.1')


#Collecting data from MongoDb Traveller Colllection
def get_traveller_info(user_name_traveller)-> tuple:
    source_lat,source_lng,dest_lat,dest_lng = 0,0,0,0
    db = client.yellow
    collection_traveller= db['traveller']
    quary = {'username': f'{user_name_traveller}'}
    doc = collection_traveller.find(quary)

    source_lat=float(doc[0]['latitude'])
    source_lng=float(doc[0]['longtitude'])
    dest_lat=float(doc[0]['seclatitude'])
    dest_lng=float(doc[0]['seclongtitude'])
    traveller_dest_city=doc[0]['dest_city']
    traveller_starting_city=doc[0]["city"]

    return traveller_starting_city,source_lat,source_lng,traveller_dest_city
```

- Geodist() function from redis library was used for finding distance and between Traveller and Cars.
- Georadiusbymember() function from redis library was used for calculating nearest cars.

```python
#Get Nearest car from MongoDb Atlas Collection and collecting that data in dictionary
def get_nearest_cars(user_name_traveller,traveller_dest_city):

    db = client.yellow
    collection_cars= db['car']
    quary = {'dest_city': f'{traveller_dest_city}'}
    doc = collection_cars.find(quary)

    pipe=r.pipeline()
    for i in doc:
        lat=float(i['latitude'])
        lng=float(i['longtitude'])
        pipe.geoadd('points',[lng,lat,i['username']])
    pipe.execute()
    nearest_cars = dict()
    closest_cars = r.georadiusbymember(name="points", member=user_name_traveller, radius=20, unit='km', withcoord=True)

    for i in closest_cars:
        dist=r.geodist(name="points",place1=user_name_traveller,place2=i[0].decode("utf-8"),unit="km")
        nearest_cars[i[0].decode("utf-8")] = f"{dist} km"

    return nearest_cars
```

- For loop used for finding nearest car in dictionary

```python
#Chosing nearest car in our nearest_car dictionary
user_name_car = None
min_distance = None

for name, dist in nearest_cars.items():

    if isinstance(dist, str):
        dist=float(dist.split(' ')[0])

    if name != user_name_traveller:

        if user_name_car is None:
            user_name_car = name
            min_distance = dist

        if dist < min_distance:
            min_distance = dist
            user_name_car = name

print(user_name_car, min_distance)
```

- Geodesic() function used for calculating distance between two city.

```
#Collecting nearest car information
    speed=float(doc[0]["speed"])
    departure_time=datetime.strptime(doc[0]["time"],"%H:%M")
    lat = float(doc[0]['latitude'])
    lng = float(doc[0]['longtitude'])
    dest_lat = float(doc[0]['seclatitude'])
    dest_lng = float(doc[0]['seclongtitude'])


    return speed,departure_time,lat,lng,dest_lat,dest_lng

speed,departure_time,lat,lng,dest_lat,dest_lng=get_car_info(user_name_car)
print(user_name_car,speed,departure_time.time(),lat,lng,dest_lat,dest_lng)

#Calculating distance from starting point and last point
coordinates=(lat, lng,dest_lat,dest_lng)
def get_distance(coordinates):
    lat, lng, dest_lat, dest_lng=coordinates

    distance=geodesic((lat, lng) ,(dest_lat,dest_lng))
    return distance
```

4.7. Frontend Used

For frontend Python-Folium package used to visualisation in map Traveller, nearest car , estimation time arrival and speed of car.

```
#Collecting coordinates for traveller ,cars starting point and destination city
traveller=(source_lat),(source_lng)
location_car_first=(lat),(lng)
departure_point=(dest_lat),(dest_lng)

#Creating map and opening in browser above information
m = folium.Map(location=traveller,width=800,height=400)
folium.Marker(traveller,popup="TRAVELLER").add_to(m)
folium.Marker(location_car_first,popup="CAR STARTING POINT",tooltip=speed).add_to(m)
folium.Marker(departure_point,popup=arrival_time.time(),tooltip=distance).add_to(m)
folium.PolyLine((location_car_first,departure_point)).add_to(m)
m
m.save("map.html")
webbrowser.open("map.html")
```

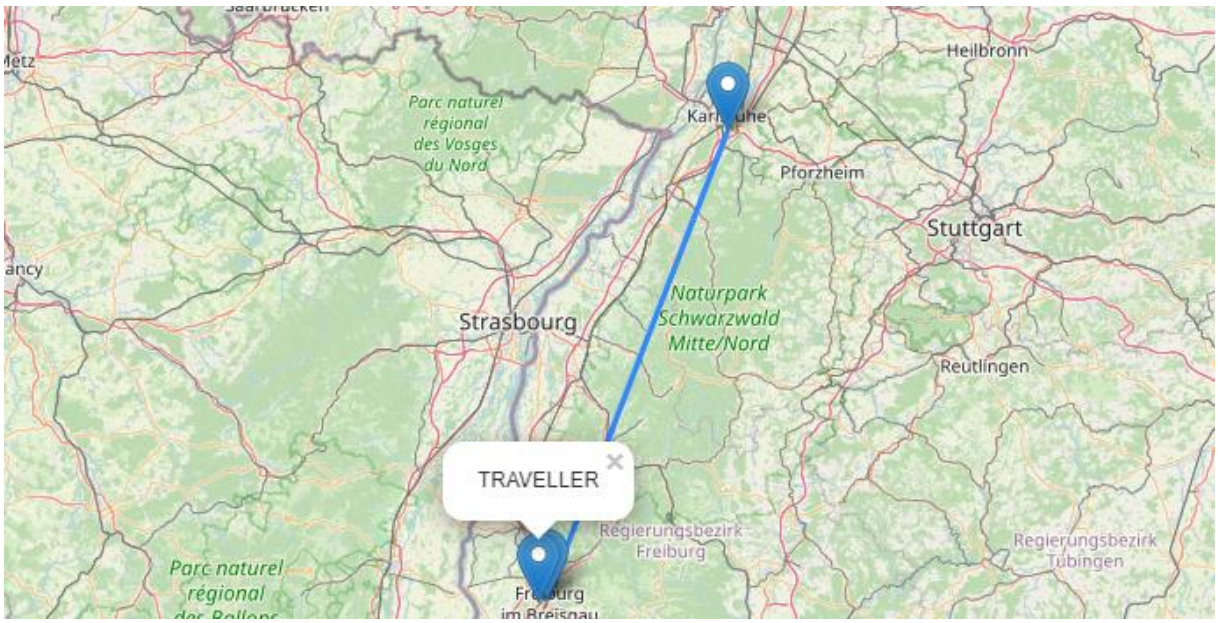

Figure 4.7-1

5. Use Case-2 Vehicle owner (Monica)

5.1. User Story

Vehicle owner is a person who owns a car and who is willing to offer rides for travellers who are looking for car-pooling rides to reach their destination point from their pick-up points. These Vehicle owners basically will be travelling from Point A to Point B, who will be willing to offer rides for traveller's whose ride requests are similar in certain criteria with the owner.

4.1.1 a)
Nick wants to offer ride share for other travellers who are willing to travel in shared cars to reach their destination. Nick wants to make sure that the time, date, destination and pickup locations are all similar to that of his, before accepting a ride request.

4.1.1 b)
Jewel is a female driver and she is willing to offer women only ride share as she is very much comfortable with that.
(make the above two as one use case and the below one as separate use case)

4.12

As Nick got stuck in his meeting, he wants to let the travellers know that he will be late by half an hour to pick them up from their pick-up location. This means that if there are any slight changes in the plan, he wants to keep his travellers updated on the same.

5.2. Actors

5.2.1. User Vehicle Owner:

The person who will be offering shared rides to travellers, who are looking for rides in the same date and time as his also the destination being similar to that of his

5.2.2. User Traveller:

The person who is requesting for shared drives to reach his destination point.

5.3. Use Case Diagram for Use Case-2

5.3.1. Use Case-2.11 and 2.12

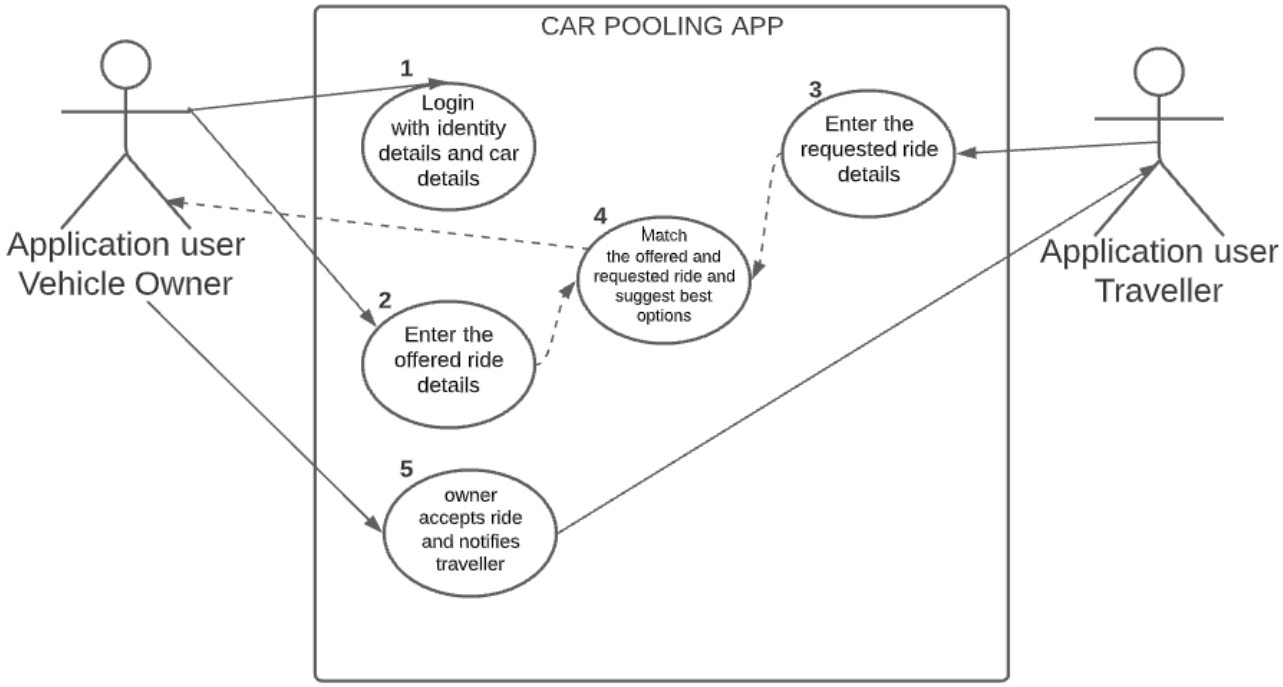

Figure 5.3-1

5.3.2. Use Case-2.2

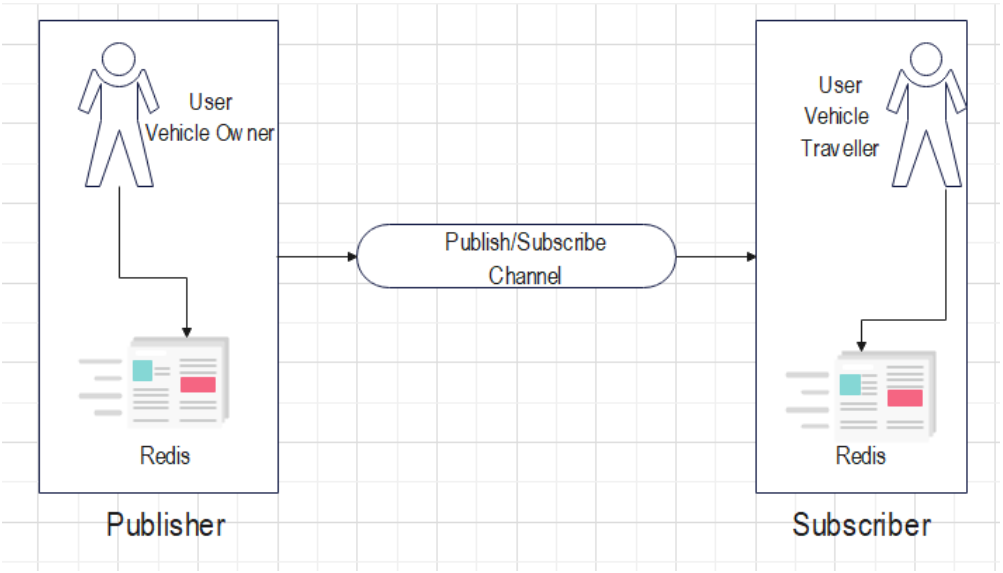

Figure 5.3-2

5.4. Interaction Diagram for Use Case-2

5.4.1. Use Case-2.11 and 2.12

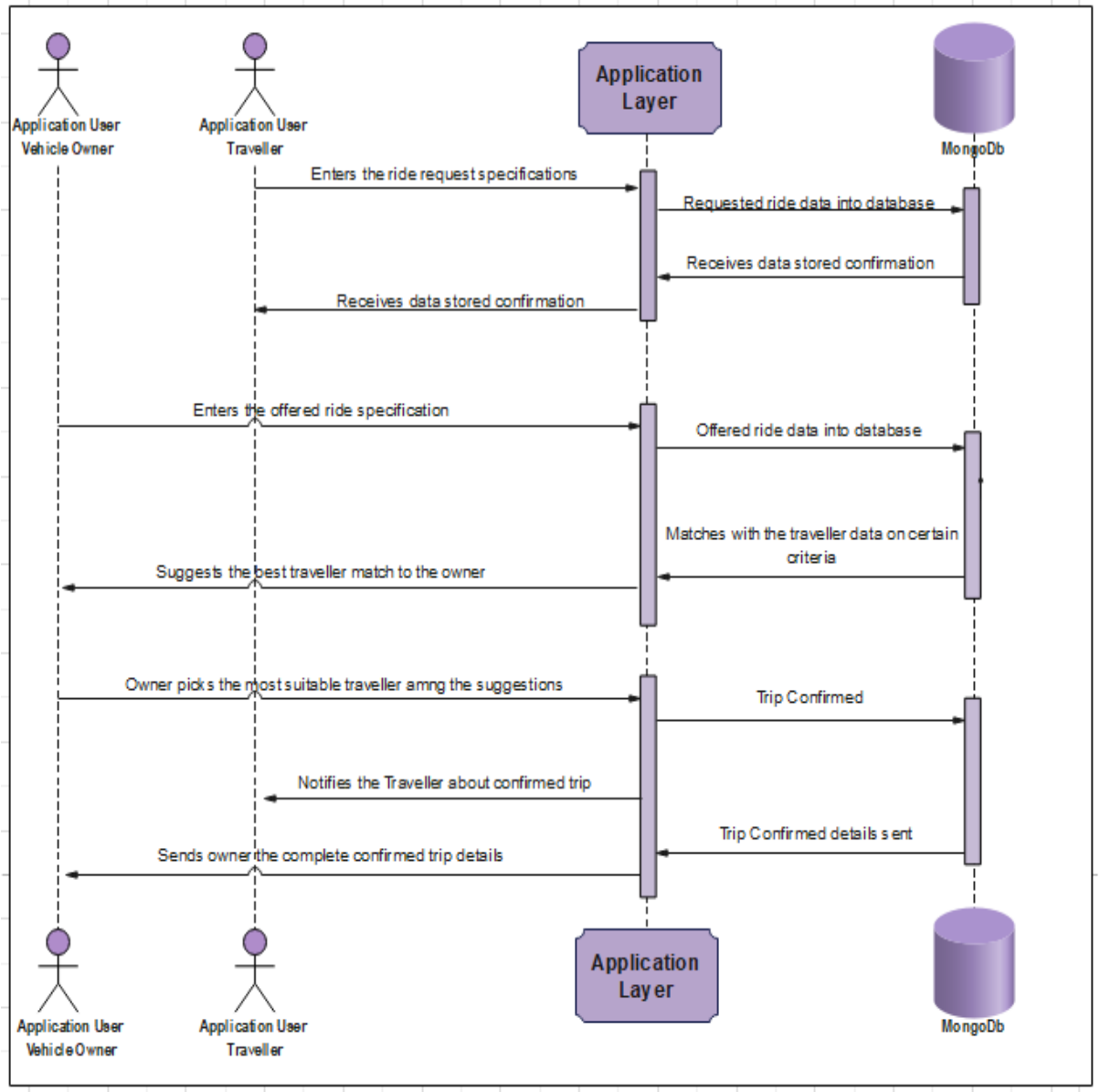

Figure 5.4-1 Descriptions of task with all interactions between actor and database

5.4.2. Use Case-2.2

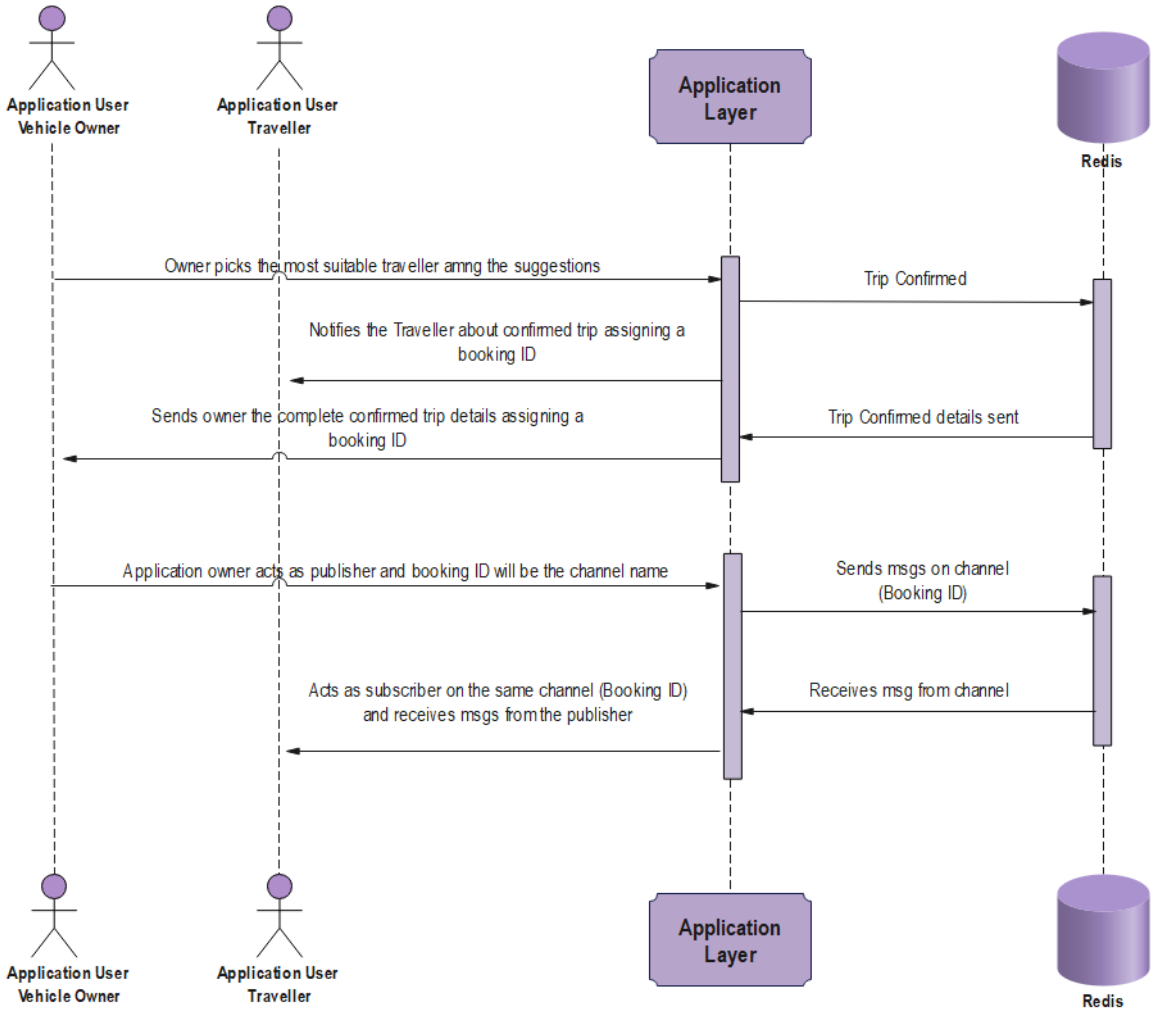

Figure 5.4-2

5.5. Data Flow Diagram for Use Case -2

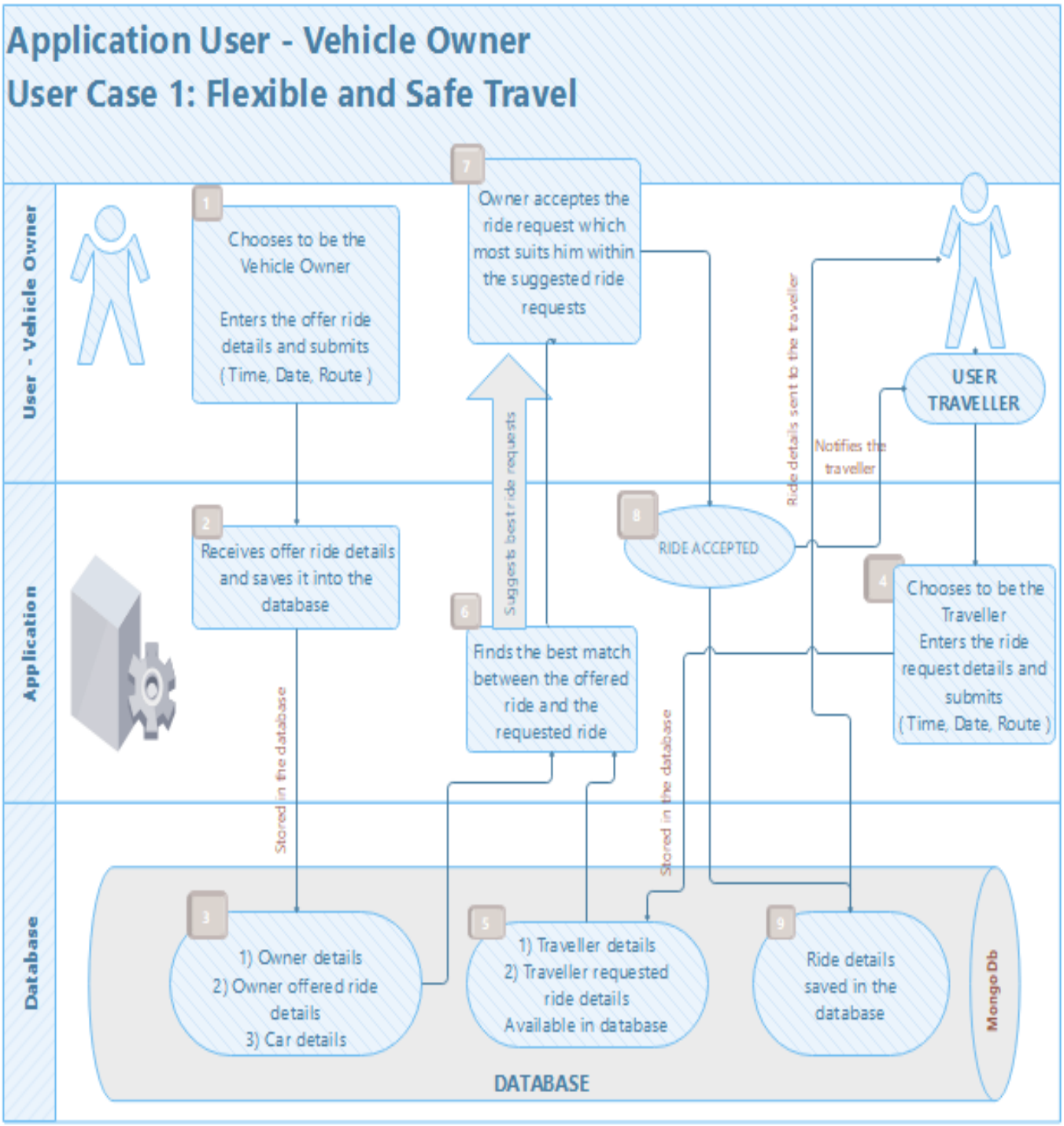

Figure 5.5-1

5.6. Databases for Use Case -2

5.6.1. Database used and why MongoDB

MongoDB: In MongoDB records can be stored with a wide variety of data structures and number of fields. MongoDB is also very popular for its Aggregation operation. This helps to process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. Pipelines can be created with multiple stages and multiple criteria's and saved, which can be used whenever needed. This is very helpful for my user case to match the requirement with database and return the expected results.

5.6.2. Expressions used for this use case

Flexible and safe travel

```
//Vehicle Owner - flexible and safe travel

//User story 1

//USE CASE 2.1 a (ride booking based on owner and travellers
preference)


// Select the database to use.


from pymongo import MongoClient


//from OWNER database


//To check cars available


use("carpool")

db.owner.find({availability:"true"})


//To offer ride as per owners specifications.


use("carpool")


//to offer ride at owners preffered time and date etc
```

```
//the owner here has also mentioned that it is female only ride

//creating another collection, inserting and saving the entries
there


db.owner_booking.insertOne({booking_id:"1011",owner_id:"91725",car_i
d:"89",date:("2021-07-
23"),time:"3:30pm",city_starting_point:"Mannheim",

start_lat: "48.5373555",start_lon: "9.2004305", city_desti_point:
"Karlsruhe", desti_lat: "-6.6038889", desti_lon: "-69.9661586",

female_only: "true", seats: "2", date: "22/01/2022", time: "2:35
PM"})


db.owner_booking.find()


//To check the Booking Details:


use("carpool");

db.owner_booking.find({booking_id:"1011"})


//from TRAVELLER database

//Created pipeline


//project only the necessary fields


use("carpool");

db.traveller.aggregate(

[

    {

        '$project': {

            'traveller_id': 1,

            'first_name': 1,

            'last_name': 1,

            'age': 1,

            'gender': 1,

            'country': 1,
```

```
            'state': 1,
            'female_only': 1,
            'phone': 1,
            'city_starting_point': 1,
            'city_desti_point': 1,
            'date': 1,
            'time': 1
        }

// match the owner_booking entries with the traveller database on
certain entries

    }, {
        '$match': {
            '$and': [
                {
                    'city_starting_point': 'Mannheim'
                }, {
                    'city_desti_point': 'Karlsruhe'
                }, {
                    'date': '22/01/2022'
                }
            ]
        }


//Vehicle Owner - flexible and safe travel
//User story 2
//USE CASE 2.1 b
//this is an example for one user case which says female only rides.

    }, {
        '$match': {
            'female_only': 'true'
```

```
        }


//after the search is complete save it in another data base and name
it 'matched_and_suggested'


    }, {
        '$out': 'matched_and_suggested'
    }
])


//This complete set of results will be sent to the owner.
//Among all the suggested travellers, owner will pick the one most
suitable for his or her ride
//this particular entry will then be added into a new database
called 'accepted_travelled_list' with the booking id same as the
owner_booking and status as 'accepted'
// created pipeline


use("carpool");
db.traveller.aggregate(
[
    {
        '$match': {
            'traveller_id': '491'
        }
    }, {
        '$addFields': {
            'status': 'accepted'
        }
    }, {
        '$addFields': {
            'booking_id': '1011'
        }
    }, {
        '$out': 'accepted_traveller_list'
```

```
    }
]

//once the booking is confirmed change availability of owner to
"booked"

use("carpool")
db.owner.updateMany({owner_id:91725},
{
    /**
     * field: The field name
     * expression: The expression.
     */
    $set: {
      availability:"booked",
      booking_id:"1011"

    }
})

// Getting final booking details

use('Carpool');
db.owner_booking.aggregate([
      {  $lookup: {
      from: 'owner',
      localField: 'owner_id',
      foreignField: 'owner_id',
      as: 'checkSer'
    }},
    {
        /**
         * query: The query in MQL.
         */
```

```
        $match: {
          Car_ID: {$in: [89]},
          Booking_ID: {$in: [1011]}
        }
    }
])

//getting complete owner details

use('Carpool');
db.owner.aggregate([
      {  $lookup: {
      from: 'accepted_traveller_list',
      localField: 'booking_id',
      foreignField: 'booking_id',
      as: 'trip_owner_details'
    }},
])
```

Use case 2 – Communication

Redis is used for this case – As Redis has a Pub/Sub package the communication between travellers and owner can be made possible and easy.

Here the Owner acts as the Publisher and the Traveller will act as the Subscriber.

The channel used here would be the “booking_id”

Once the owner offers a ride a booking id will be assigned to him and once he accepts a ride request the same booking_id will be assigned to the traveller.

Hence this booking_id will act as the channel through with the owner and the travellers will communicate.

```
Redis-cli
Traveller subscribes to the channel
SUBSCRIBE 1011
Owner publishes to the channel
PUBLISH 1011 “I am going to be late by half an hour please wait”
Publishes this onto the receivers
```

6. Use Case-3 Help Desk (Sasha)

6.1. User Story

Help desk is a department or person that provides assistance to the registered travellers in application. Trigger point for activation of a helpdesk actor is a notification from the users (inside Application) below is summary table of Help desk actor tasks:

- Help Desk employee is an actor who receives message from a user who needs car replacement due to different reasons.
- Help Desk Employee task is to check for the availability of cars in the database and receive general data about the cars.
- Also, he would like to check number of available cars who have same destination as user, so he can dispatch the car to pick them up.
- Help Desk Employee can broadcast a message through application to number of cars and he will wait for response
- He has the ability to send and receive message to user to transfer information.

6.2. Actors

6.2.1. User(Traveller):

Traveller (in this use case mentioned as User) is a traveller who had an incident during a travel time and thei car needs to be replaces.

6.2.2. Help Desk:

Help Desk is car pooling company employee who has access to the application through a helpdesk dashboard to get notification from users and help them

6.2.3. Cars:

Cars are the vehicles who are either on a trip(these cars has active travellers) or they are not inside a trip but they are registered user of app and has valid user name.

6.3. Use Case Diagram for Use Case-3

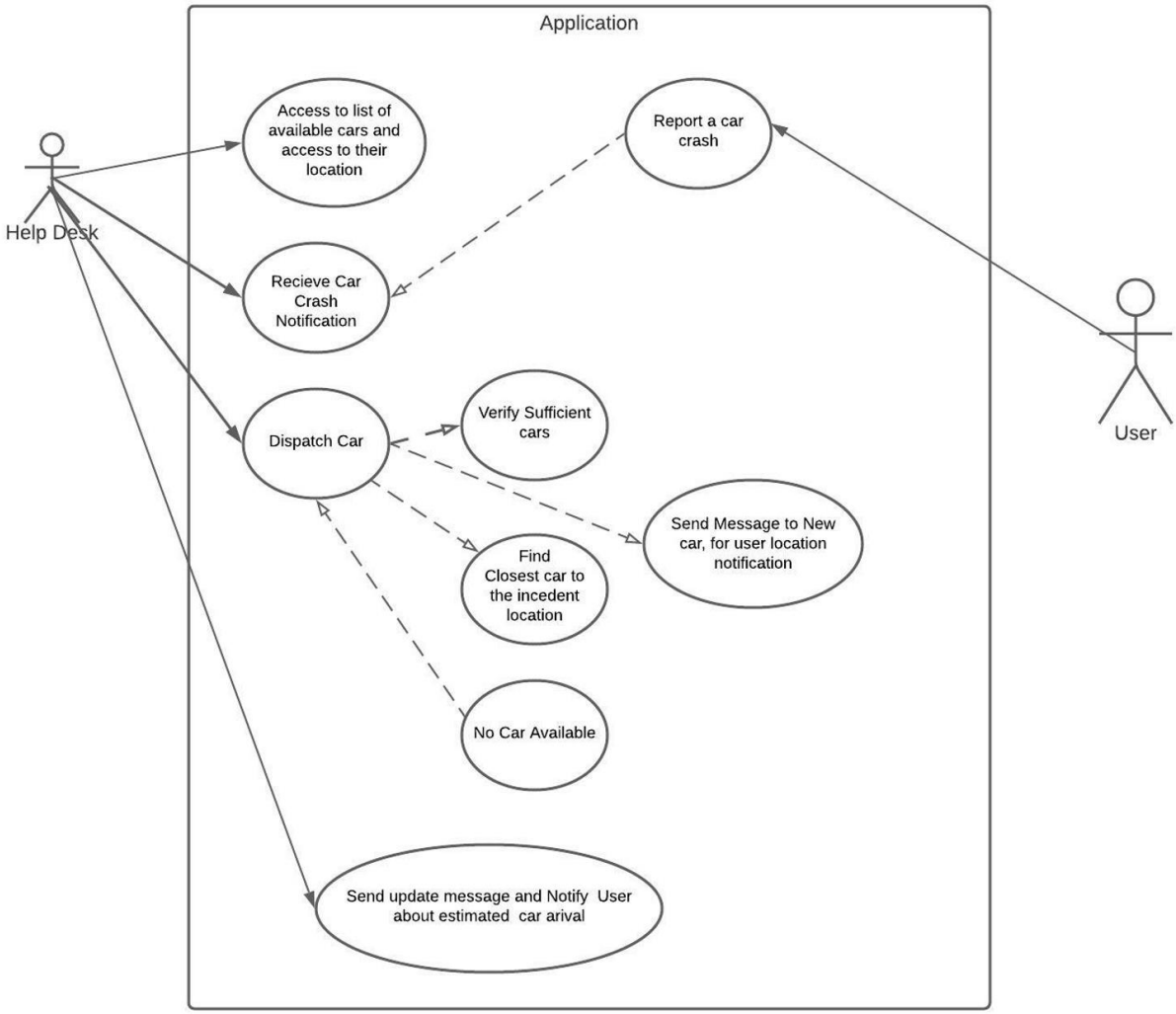

Figure 6.3-1

6.4. Interaction Diagram for Use Case-3

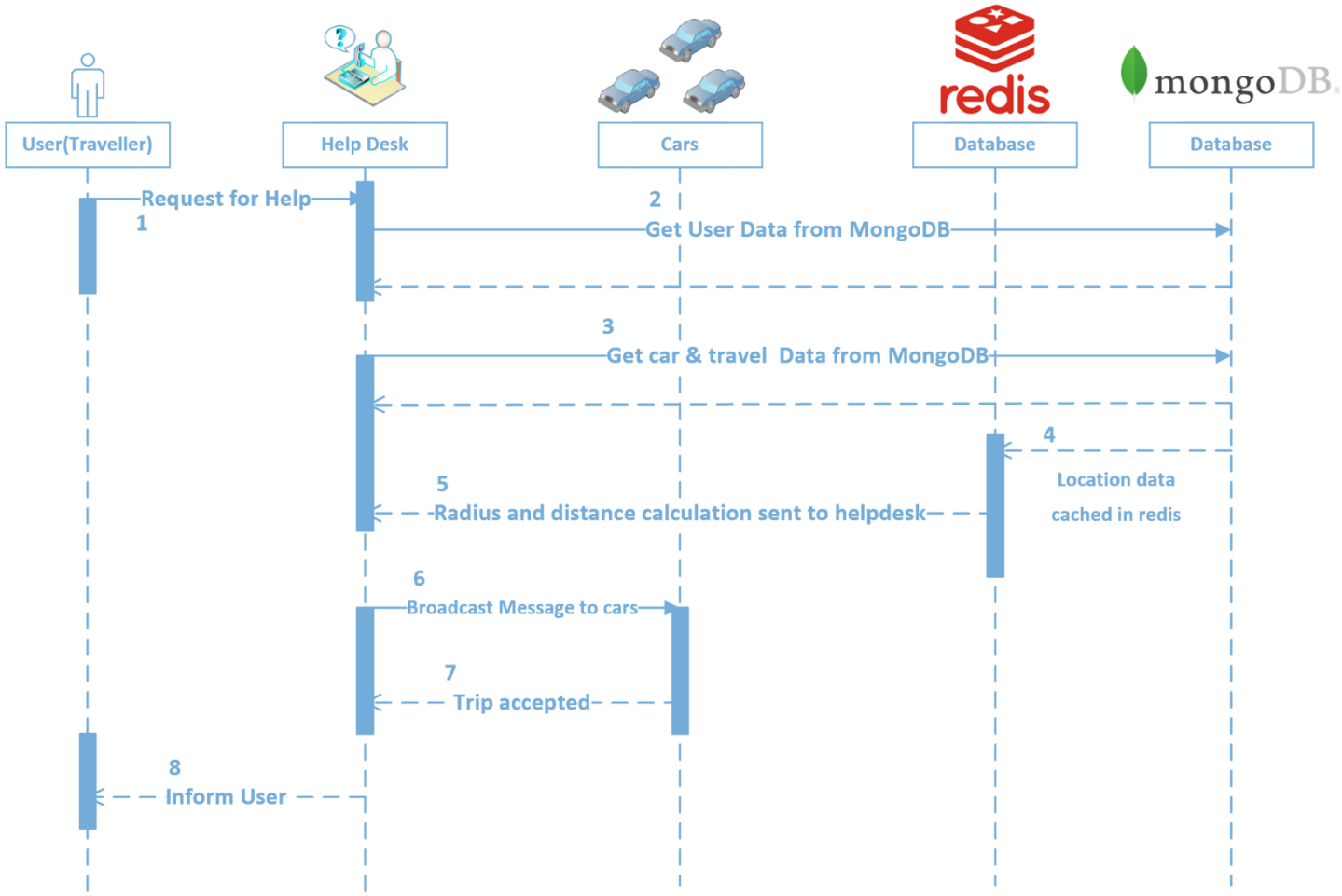

Figure 6.4-1

6.5. Data Flow Diagram for Use Case-3

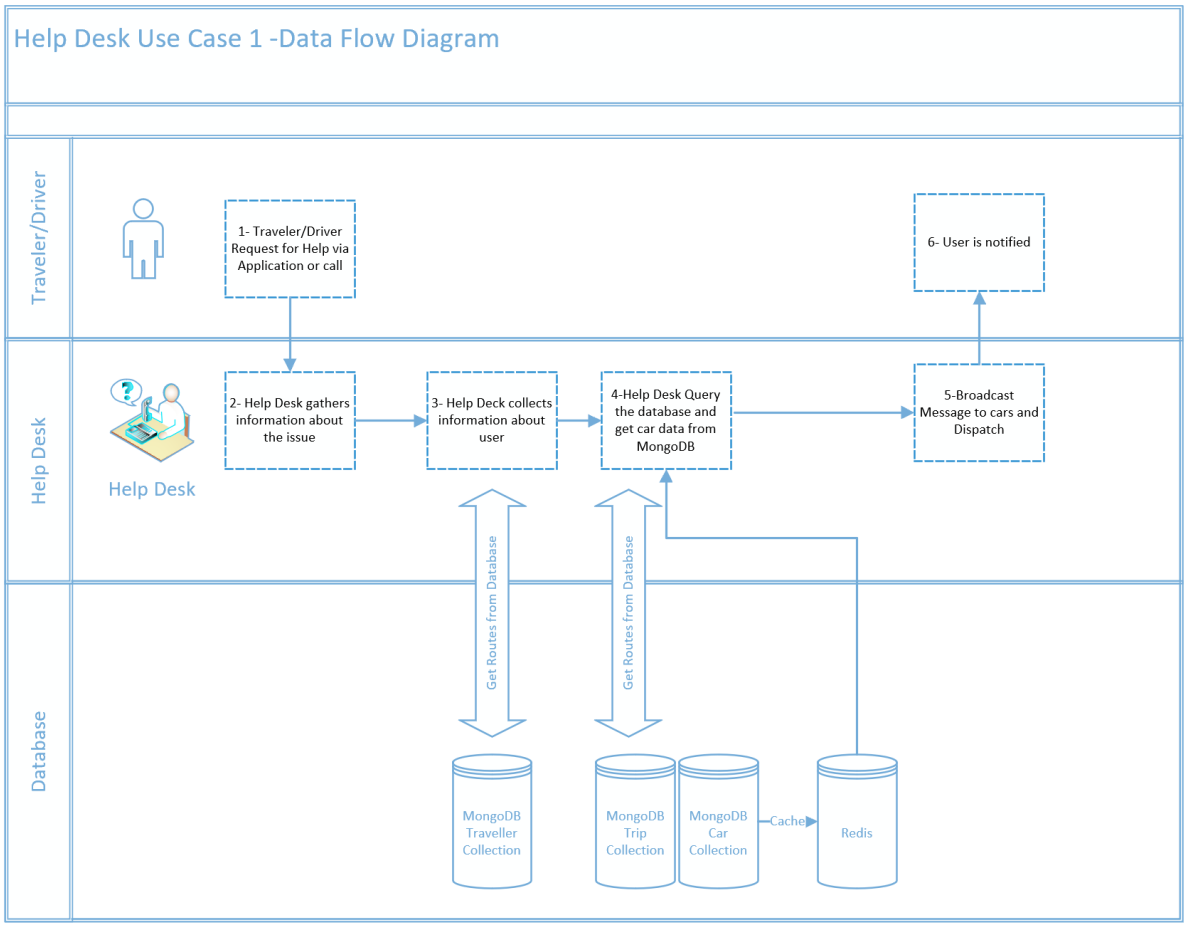

Figure 6.5-1

6.6. Databases for Use Case-2

6.6.1. Databases used:

MongoDB is used to store Traveller, Car and Trip data in three separate collections. All the data is stored in MongoDB cloud for easy access of team in development phase.

Redis is used to cache mongoDB data because of it's efficiency in Geospatial calculations

6.6.2. MongoDB Queries used:

Belo queries are the ones which helpdesk sends to MongoDB in order to get car data. Some of this data will be cached in Redis for Geospatial calculations:

MongoDB Query:

```
db.car.aggregate([
{$lookup:{

        from:"trip",
        localField:"username"",
        foreignField:"car_username",
        as:"trip_data"
}
},
{$match:
    {
    "trip_data.traveller":{$size:2},
    "dest_city" : "Reutlingen",
    "gender" : "Female",
     "trip_data.trip_status":"ongoing"
    }

}

])
```

Equivalent python code:

```
result = collection_car.aggregate([
    {"$lookup": {

        "from": "trip",
        "localField": "username",
        "foreignField": "car_username",
        "as": "trip_data"
    }
    },
    {"$match":
        {
            "trip_data.traveller": {"$size":
int(num_of_travellers)},
            "dest_city": f"{destination_city}",
            "trip_data.trip_status": f"{trip_status}"
        }

    }])
```

6.7. Application:

The application is designed to facilitate Help Desk roll through a dashboard. It is programmed using python utilizing redis, pymongo, Flask packages.

During the development phase below tools has been used as IDE and database tools:

- Redis-cli
- MongoDB Atlas (clous server)
- MongoDB Compass
- PyCharm
- Studio 3T
- VS Code (front-end design)

6.7.1. Application Source Code (MongoDb Queries and Redis Caching):

6.7.1.1. Importing libraries and connecting to databases:

```
import random
import redis
from pymongo import MongoClient
client = MongoClient("mongodb+srv://sasha:sasha123@cluster0.kmfv3.mongodb.net"
                     "/myFirstDatabase?retryWrites=true&w=majority")

r=redis.Redis('127.0.0.1')
```

6.7.1.2. Request help function is generations a user request for car replacement:

- The key field is username in MongoDB Traveller collection. Applications send a query command to MongoDB server to request general and location data
- Using ***random.uniform()*** method The function generates a point between source and destination points. This is to simulate car crash or incident location. Location data is cashed in redis database by using **redis.geoadd** function:

```python
#User Requests for help
def request_help(user_name)-> tuple:
    source_lat,source_lng,dest_lat,dest_lng = 0,0,0,0
    #Gets user travel info from mongo DB
    db = client.mydb
    collection_traveller= db['traveller']
    query = {'username': f'{user_name}'}
    doc = collection_traveller.find(query)
    for i in doc:
        source_lat=float(i['latitude'])
        source_lng=float(i['longtitude'])
        dest_lat=float(i['seclatitude'])
        dest_lng=float(i['seclongtitude'])
        dest_city=i['dest_city']

    #Generates random point in the travel route to simulate car crash location:

    incedent_lat=round(random.uniform(source_lat, dest_lat), 6)
    incedent_lng=round(random.uniform(source_lng, dest_lng), 6)
    r.delete("points")
    r.geoadd("points", [incedent_lng,incedent_lat,user_name])
    return user_name,incedent_lat,incedent_lng
```

- After execution, Geolocation data of the user will be saved in sorted set in redis database. This can be tested by redis-cli commands (below figure):

```
127.0.0.1:6379> keys *
1) "points"
127.0.0.1:6379> zrange points 0 -1 withscores
1) "nsprowson1u"
2) "3666430048822120"
127.0.0.1:6379> geopos points nsprowson1u
1) 1) "9.15894180536270142"
   2) "48.77799275775588228"
127.0.0.1:6379>
```

Figure 6.7-1

6.7.1.3. MongoDB query function

This function is responsible for automating helpdesk user queries. Helpdesk operator enters specification of the car he wants to find around incident location. The function is designed to get the information from MongoDB and return it to helpdesk operator. This function is using queries in section 4.5.2:

```
def mongoQuery(num_of_travellers,destination_city,trip_status):
    db = client.mydb
    collection_trip = db['trip']
    collection_car = db['car']
    result = collection_car.aggregate([
        {"$lookup": {

            "from": "trip",
            "localField": "username",
            "foreignField": "car_username",
            "as": "trip_data"
        }
        },
        {"$match":
            {
                "trip_data.traveller": {"$size": int(num_of_travellers)},
                "dest_city": f"{destination_city}",
                "trip_data.trip_status": f"{trip_status}"
            }

        }])

    return result
```

6.7.1.4. get_nearest_cars function

This Function is responsible for retrieving car data from MongoDb database, cashing location data in redis and finding cars in a radius recived by Helpdesk operator.

- The key field is car owner username in MongoDB car collection. Applications send a query command to MongoDB server to request, car data. The query is reducing the number of cars by filtering the cars with same destination of user.
- Car locations are being cached in redis within a for loop using **redis.geoadd.** multi transaction is used to reduce number of requests to redis. **redis.pipeline()**is used for this purpose

```
def get_nearest_cars(user_name, num_of_travellers, destination_city, trip_status,user_radius):
    print(user_name, num_of_travellers, destination_city, trip_status,user_radius)
    doc=mongoQuery(num_of_travellers,destination_city,trip_status)
    #load all cars from mongoDB car collection to redis points GeoSpatial sorted set:
    pipe=r.pipeline()
    for i in doc:
        lat=float(i['latitude'])
        lng=float(i['longtitude'])
        pipe.geoadd('points',[lng,lat,i['username']])
    pipe.execute()
    #finds nearest cars to user and stores it in dictionary  called nearest_cars
    nearest_cars = dict()
    closest_cars = r.georadiusbymember(name="points", member=user_name, radius=user_radius,
                                       unit='km', withcoord=True)

    for i in closest_cars:

        dist=r.geodist(name="points",place1=user_name,place2=i[0].decode("utf-8"),unit="km")
        if dist==0:
            continue
        nearest_cars[i[0].decode("utf-8")] = f"{dist} km"
        if len(nearest_cars)>10:
            break
    return nearest_cars
```

- After caching all data in sortes set in redis under the name of points. The function executes **redis.geodestinationby** member to find all the cars within 5 Kilometer radius. Equivalent cli command is captured as a sample in below figure
- At the end the function calculates distance of each car to the user, and store all the data to a dictionary for future use in application

```
127.0.0.1:6379> georadiusbymember points nsprowson1u 5 km
 1) "nsprowson1u"
 2) "aalcottff"
 3) "abodely"
 4) "abrommaged5"
 5) "aclement1r"
 6) "acompson6y"
 7) "aconnellyqf"
 8) "afoskewr"
 9) "agosnall76"
10) "agotchco"
11) "ahardisonn7"
12) "aharwickfk"
13) "aimpeyby"
14) "ajosefovicej"
15) "ajouannissonh1"
16) "ajoyes4r"
17) "alantaphak"
18) "alittlecote9o"
19) "alloydre"
20) "amarquese5y"
```

Figure 6.7-2

6.7.2. Application Front End overview:

In order to create interface for user interaction and to build Help Desk employee dashboard. Flask library has been utilized.

Following python codes are used to create sample frontend. It is comprised of 2 web page:

- Help Request Page: Traveller who needs help uses this portal to send request to helpdesk by submitting their username. The page is designed to receive free text message as a message to helpdesk.(The message will be sent to helpdesk operation via redis pubsub)

- Help Desk Dashboard: In this page helpdesk, recives traveller username and perform custom queries in car data to find most suitable car for replacement. On the other hand, helpdesk operator has utilities to broadcast message to cars and to interact with traveller.

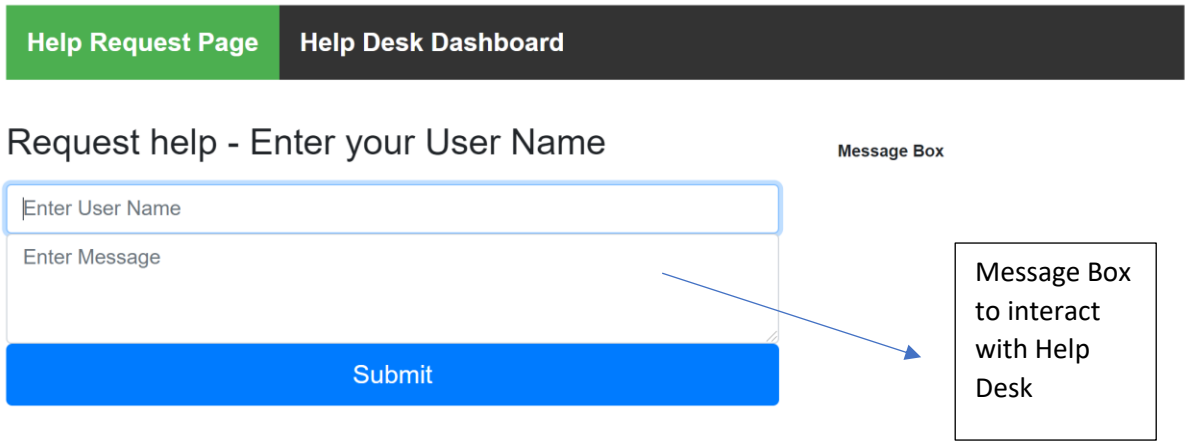

Figure 6.7-3 Help User Page- Portal for asking help from helpdesk

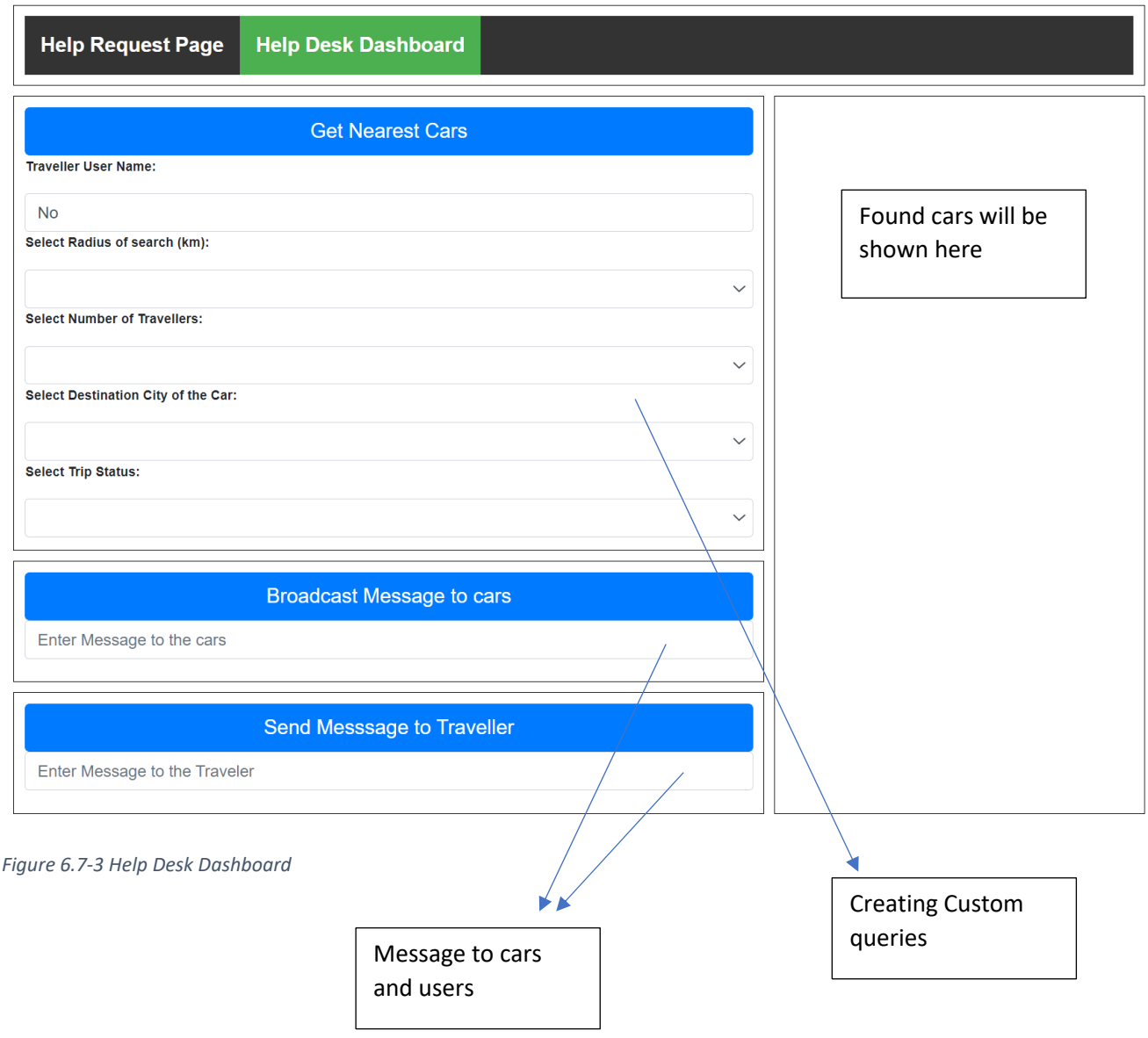

Figure 6.7-3 Help Desk Dashboard

6.7.3. Application Source Code (Back-end with flask):

The following python codes used for developing backend for section 4.6.2

6.7.3.1. Importing libraries:

```
from flask import Flask,render_template
from flask import request as f_request
```

6.7.3.2. Request Help Page Backend code:

```
app=Flask(__name__)
@app.route("/",methods=["GET","POST"])
def request():
    global user_name
    if f_request.method== "POST":
        req = f_request.form
        user_name=req["user_name"]
        msg=req["msg_data"]
        request_help(user_name)
    return render_template("UserPage.html")
```

6.7.3.3. Help Desk Dashboard Backend code:

```
@app.route("/dashboard",methods=["GET","POST"])
def dashboard():
    global user_name,num_of_travellers,destination_city,trip_status,user_radius
    nearest_cars={}
    if f_request.method == "POST":
        req = f_request.form
        user_name= req["user"]
        num_of_travellers=req["Number of travellers"]
        destination_city=req['destination_city']
        trip_status=req["trip status"]
        user_radius=req["radius"]
        nearest_cars=get_nearest_cars(user_name, num_of_travellers, destination_city,
                                      trip_status,user_radius)
        print("hi",user_name, num_of_travellers, destination_city, trip_status,user_radius)

    return render_template("HelpDesk.html",user_name=user_name,nearest_cars=nearest_cars)

if __name__=='__main__':
    app.run(debug=True)
```

6.8. Real-time messaging system Using Redis

Publisher/Subscriber Architecture:

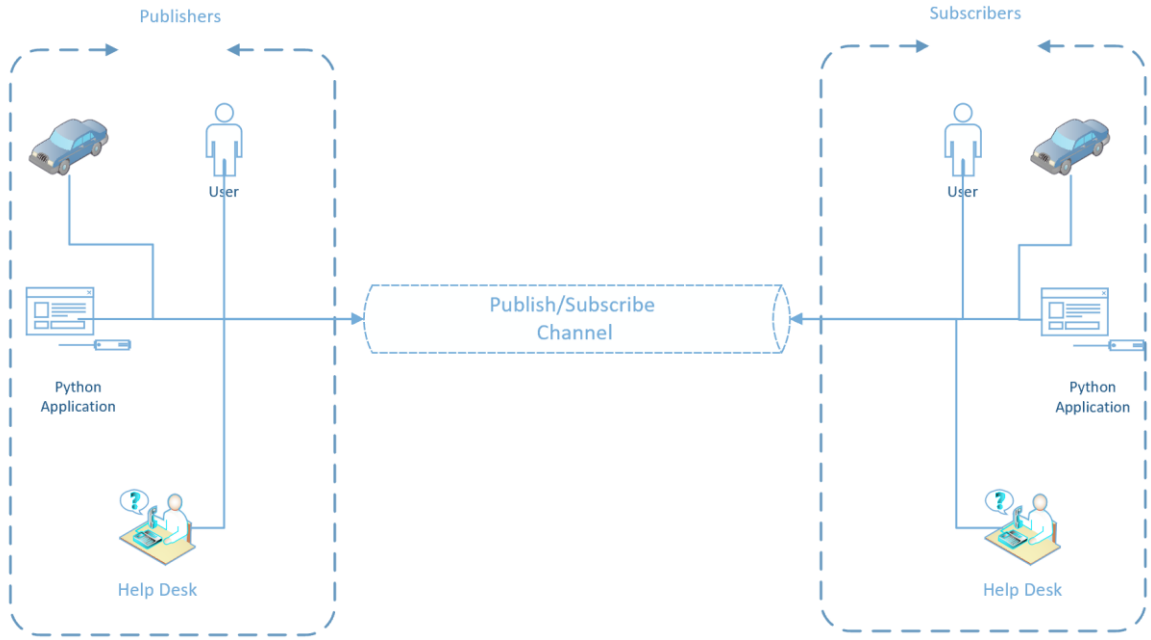

Figure 6.8-1

All the communications are being done through 3 redis pub/sub channel. Application source code is as below. Messages are being saved in redis list for further access:

```
import redis
r=redis.Redis('127.0.0.1')

def pub(msg1,msg2):
    r.publish('ch-1', msg1)
    r.publish('ch-2', msg2)

def sub(channel):
    sub=r.pubsub()
    sub.subscribe(channel)
    for message in sub.listen():
        if message is not None and isinstance(message, dict):
            msg1=message.get('data')
            r.lpush('msg_scoreboard',msg1)
while True:
    sub('ch-1')
    sub('ch-2')
    sub('ch-3')
```

7. Use Case-4 Financial & Operational Services (Nissy)

7.1. User Story

This is a department within carpooling who focuses mainly on sending recommendation to user for joining the carpooling service and route suggestion before the trip begins. Based on user interest, pooling details also shared and track the payment process.

1. Sam who works in the financial & Operation services department need to send pooling recommendations based on the location, trip type, car type, number of co-passengers and expected expense.
2. Sam who works in the financial & Operation service department suggest best route based on user request.
3. Jessey who missed the carpool pickup rejects payment refund and request to assign and carpooling based on her new location.

7.2. Use Case Diagram for Use Case-4

7.2.1. USE CASE 1

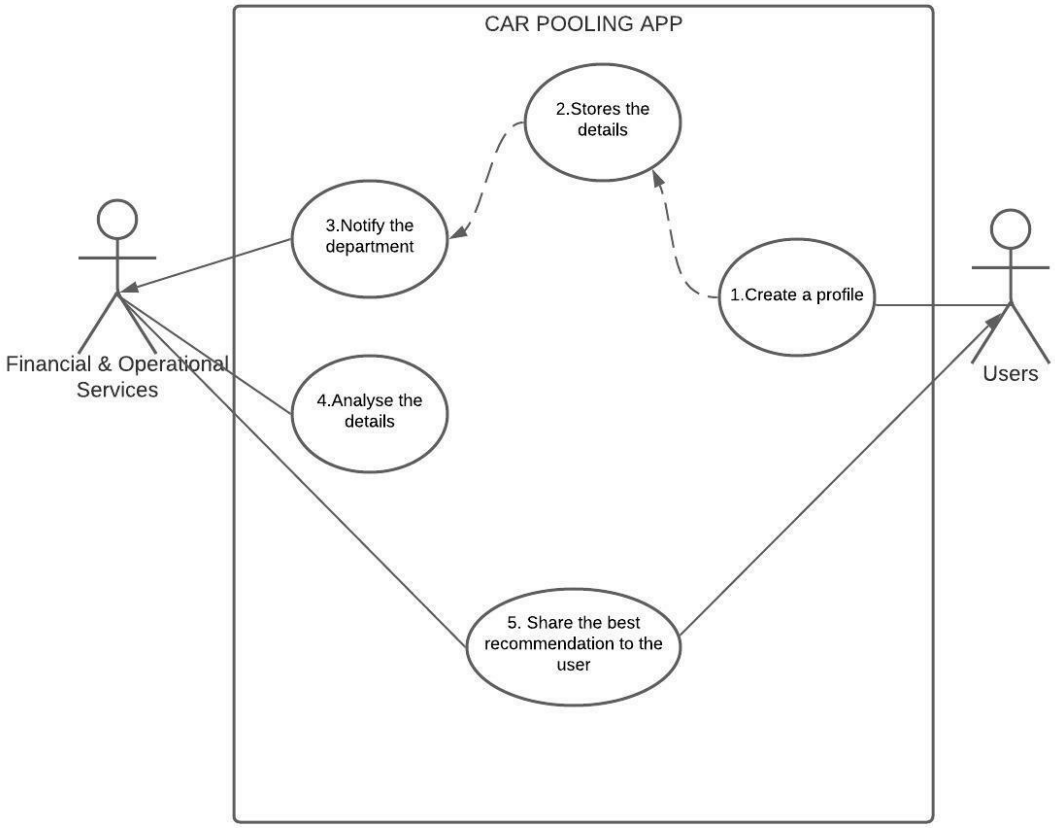

Figure 7.2-1

7.2.2. USE CASE -2

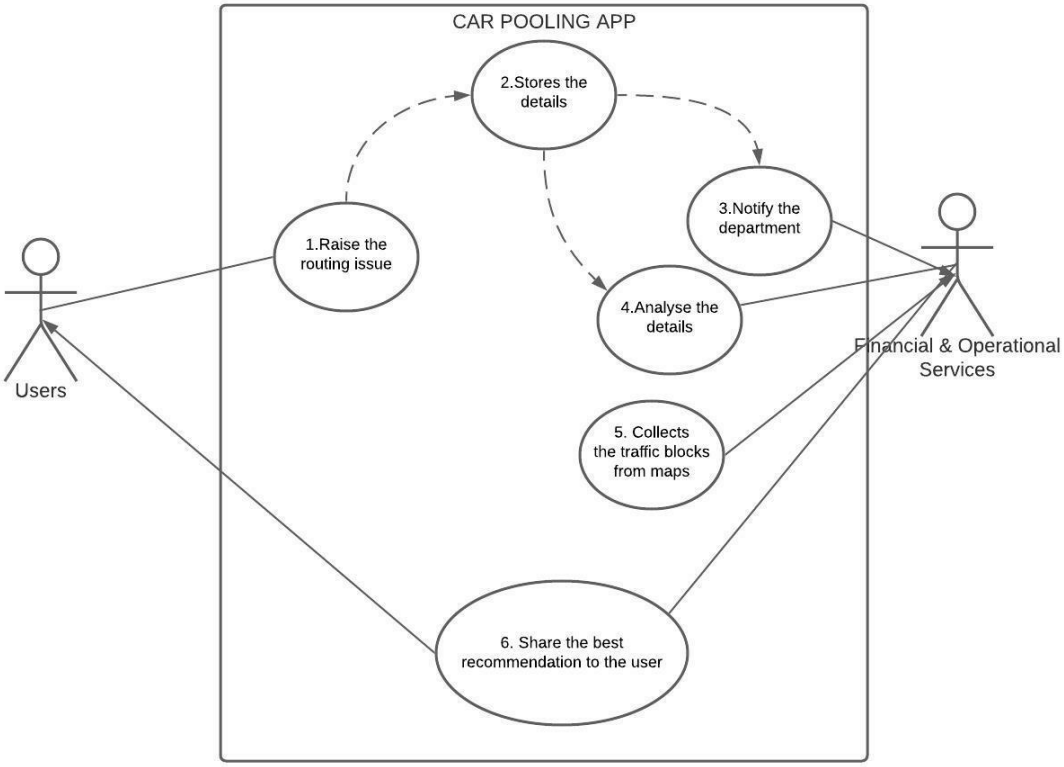

Figure 7.2-2

7.2.3. USE CASE-3

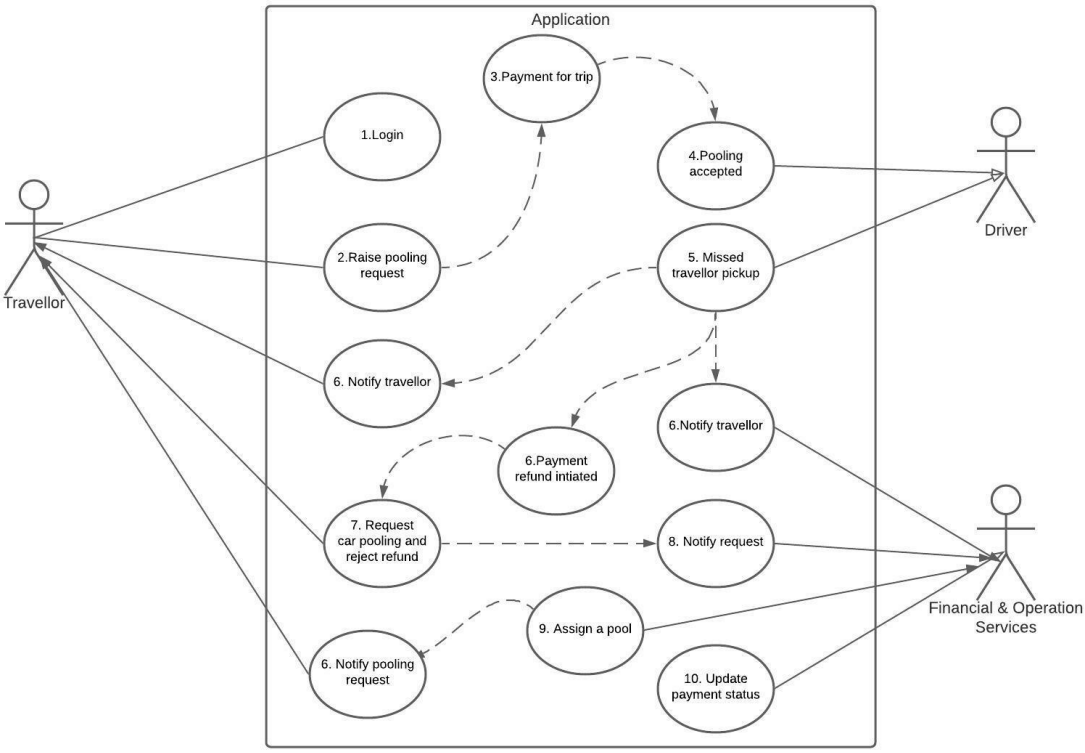

Figure 7.2-3

7.3. Actors

7.3.1. User:

All the users who create a profile in the car pooling application.

7.3.2. Financial & Operation service department:

The department who sending best pooling recommendation and best routes for the trip also tracks the payment process.

7.3.3. Traveller:

All the users who are availing the car pool services using the application.

7.4. Interaction Diagram for Use Case-4

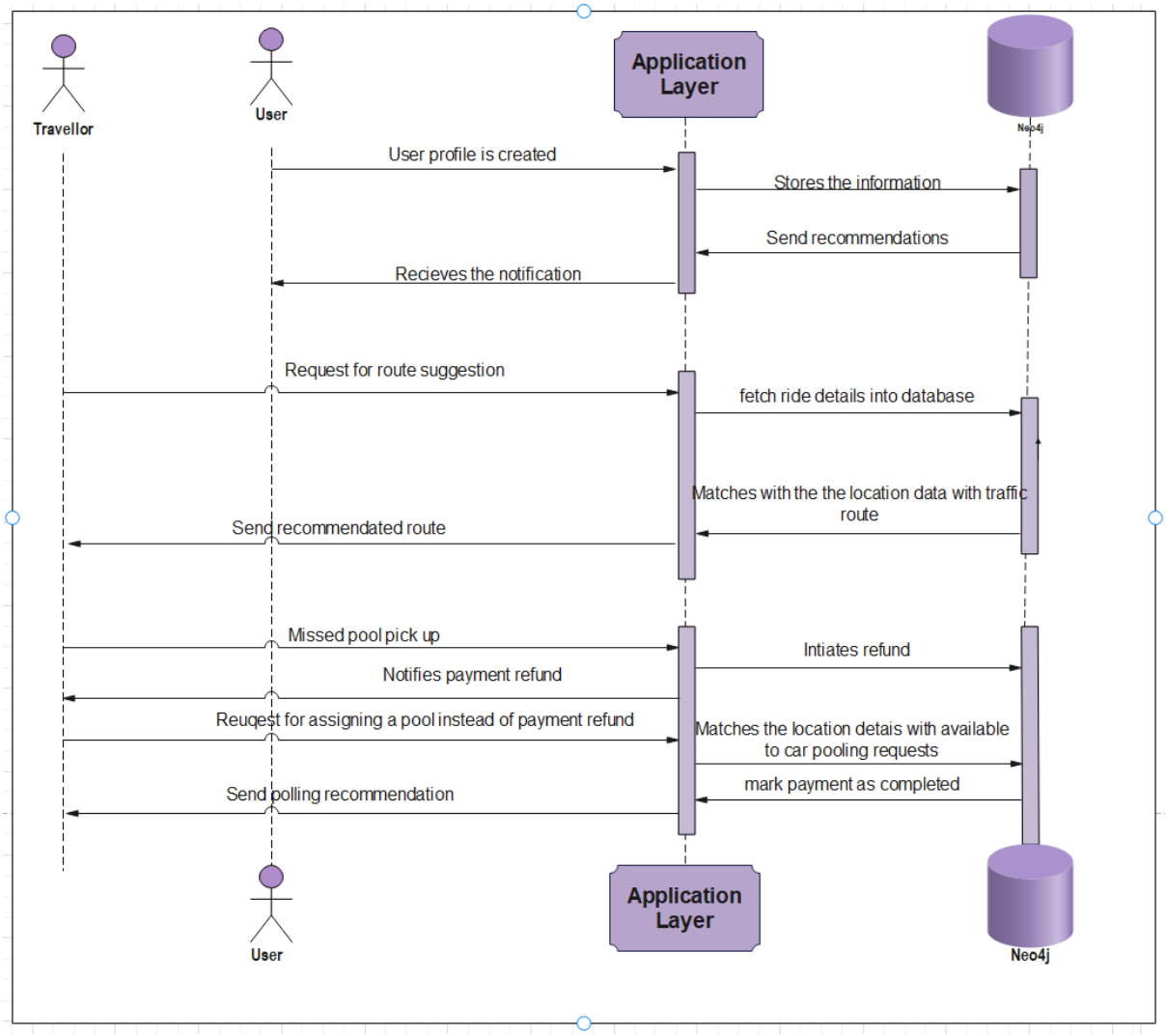

Figure 7.4-1. Descriptions of task with all interactions between actor and database. (Detailed)

7.5. Data Flow for Use Case-4

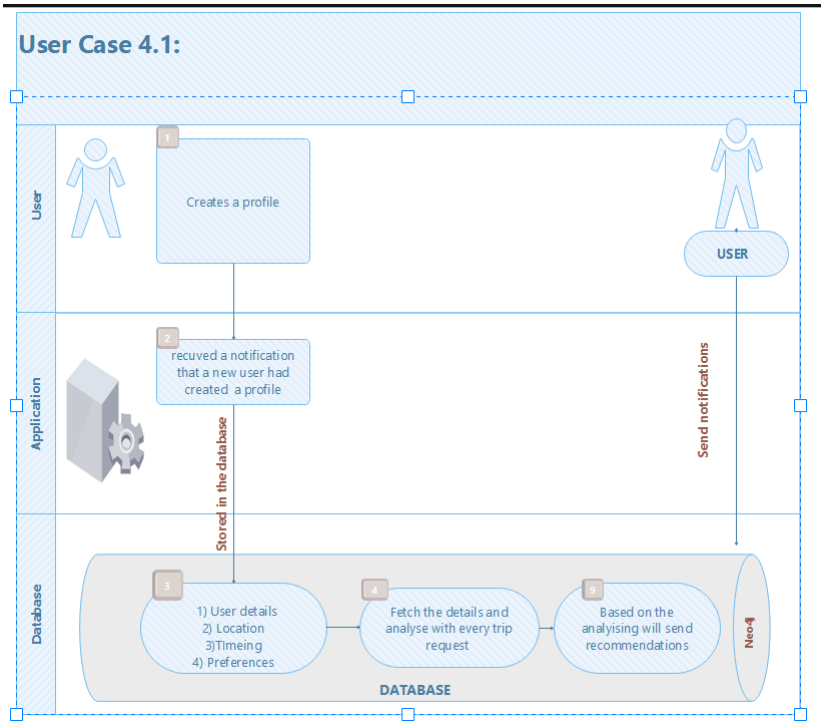

Figure 7.5-1 (Step by step, detailed)

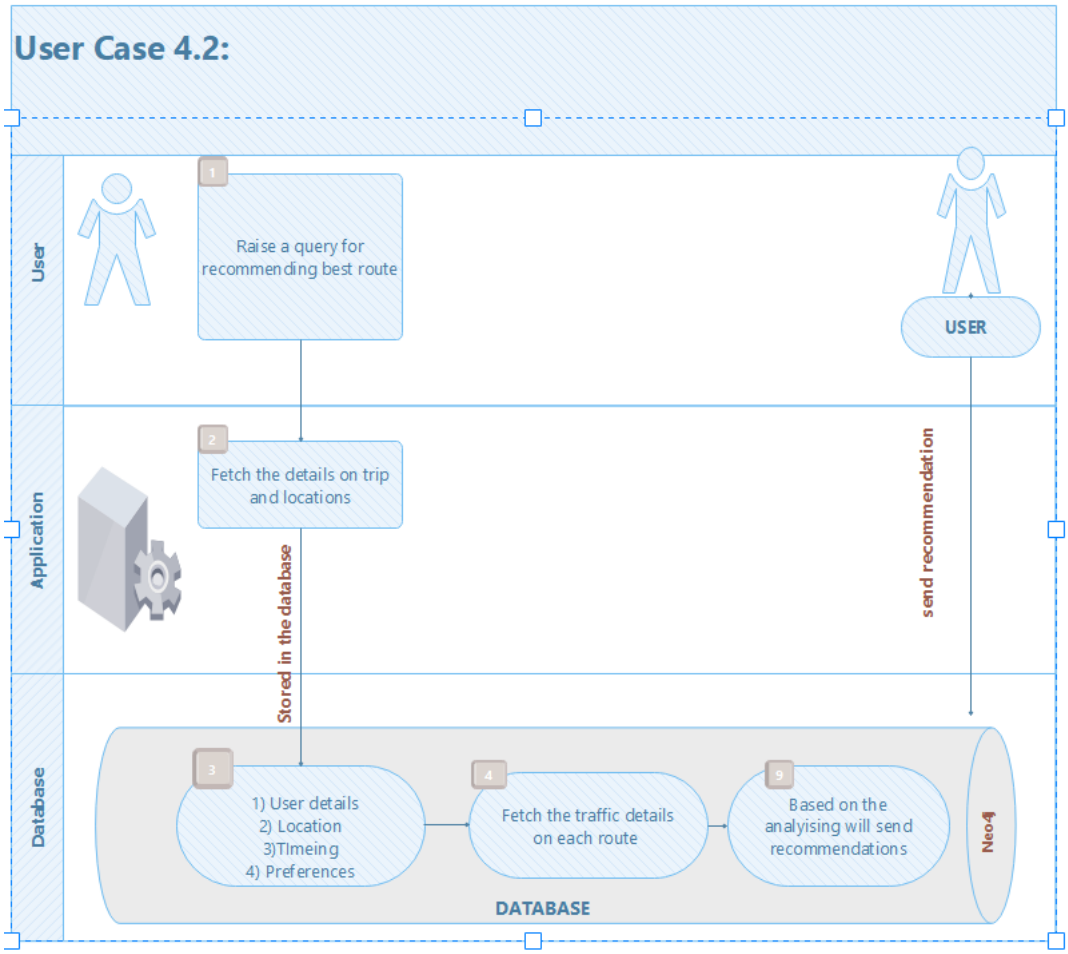
User Case 4.2:
User
Application
Database
1
Raise a query for recommending best route
USER
2
Fetch the details on trip and locations
Stored in the database
send recommendation
3
1) User details
2) Location
3)TImeing
4) Preferences
4
Fetch the traffic details on each route
9
Based on the analyising will send recommendations
Neo4j
DATABASE

Figure 7.5-2

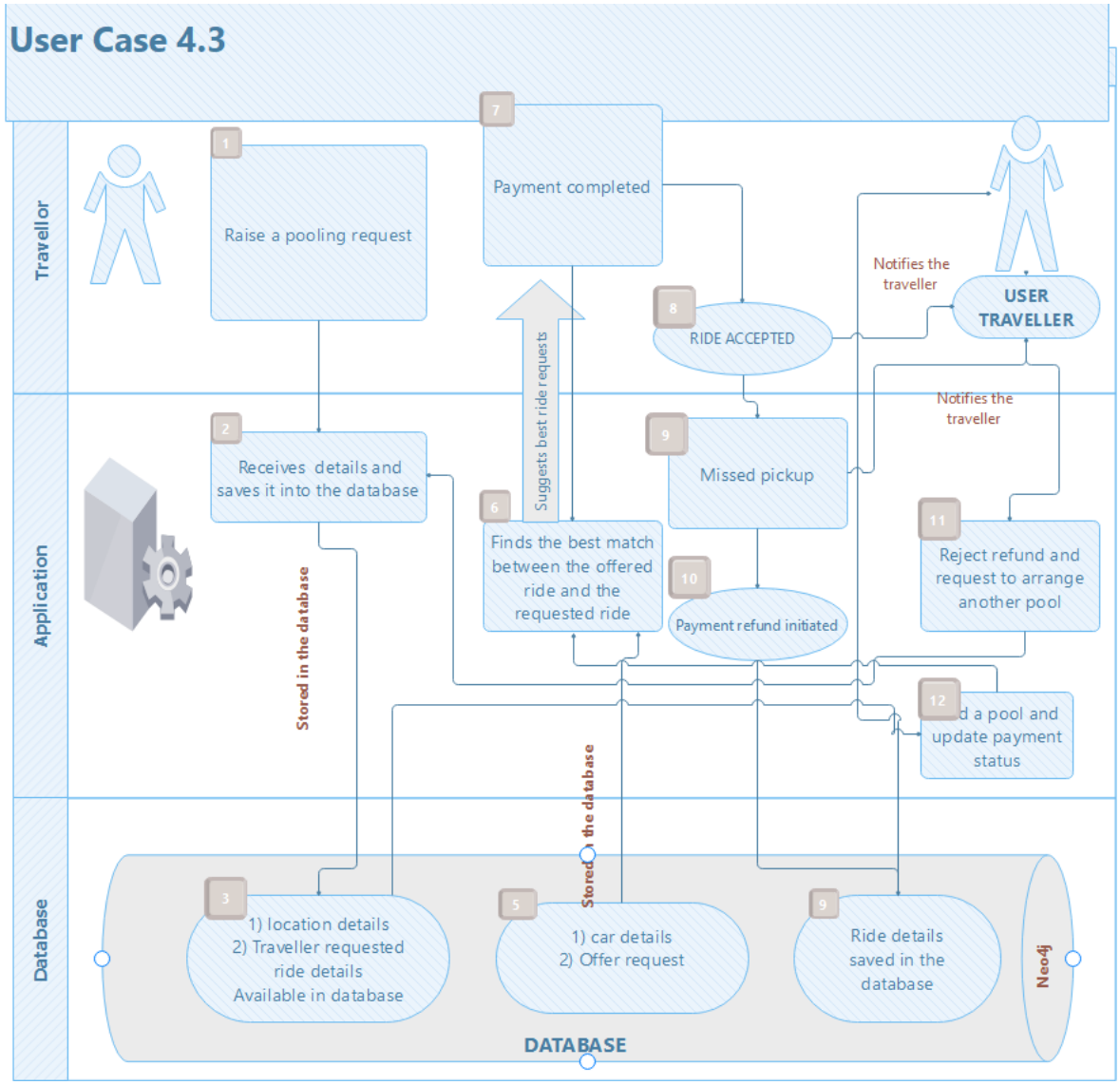

Figure 7.5-3

7.6. Databases for Use Case -4

7.6.1. Database used and why?

Neo4j helps to visualization the data, which will help you improve your results: as you develop an algorithm, it is important to see the results it delivers to evaluate it .

7.6.2. Expressions used for this use case

1. Sam who works in the financial & Operation services department needs to send pooling recommendations based on the location, trip type, No of co-passengers, car type and expected expense for all the users

```
MATCH (h:HomeCity)-[way:TRAVEL_ROUTE]->(m:DestinationCity)
OPTIONAL MATCH
 (p:Rate_Slab)-[t:Rate]->(u:User)
RETURN h.CityName as Homecity,m.CityName as DestinationCity,to
Integer(u.NumOfPasngrs) as NumOfPasngrs,
u.TripType as TripType,u.CarType as CarType,round(way.travelDi
stance/1000,2) as DistanceKM,t.Expected_Expense as Expected_Ex
pense,count(*) as NumOfUsers
```

```
MATCH (h:User)-[:OWN]-(c:Car)
WHERE h.IS_OWNER='Y'
RETURN h.Homecity as Homecity,h.Destin_City as DestinationCity
,toInteger(h.NumOfPasngrs) as NumOfPasngrs,
h.TripType as TripType,h.CarType as CarType,count(*) as NumOfU
sers
```

2. Sam who works in the financial & Operation services department need to sort routing problems in faced in car pooling . hence he sends best route suggestion based on requests.

```
MATCH path = (:PLACE {name:"COLOGNE"}) -[road:LINK*]-
> (:PLACE {name:"ESSEN"})
WHERE ALL (r IN road WHERE NOT EXISTS (r.isBlocked))
WITH path, REDUCE (sum = 0, r IN road | sum + (r.DistanceKM/r.
avgSpeed)) AS time
ORDER BY time
RETURN  REDUCE(cities = head(nodes(path)).name, n IN tail(node
s(path)) | cities + '->' + n.name) AS Route,
  REDUCE (rd = head(relationships(path)).name, r IN tail(relat
ionships(path)) | rd + ' > ' + r.name) AS Roads,
  toString((round(time * 100))/100) + ' hr' AS TotalTime
```

3. Jessey who missed the carpool pickup rejects payment refund and request to assign and carpooling based on her new location.

```
MATCH (n:User)-[t:Distance ]->(m:Location)
WITH
point({ longitude: toInteger(n.longitude), latitude: toInteger
(n.latitude) }) AS p1,
point({ longitude: toInteger(m.longitude), latitude: toInteger
(m.latitude)}) AS p2,t
SET t.Distance=round(distance(p1,p2), 2)
RETURN round(distance(p1,p2), 2) as Dist,t
```

```
MATCH (c:User)-[t:Distance ]->(s:Location)
WHERE c.CityName<>s.CityName and t.Distance>120000
RETURN c.username,c.CityName as FromCity,s.CityName as ToCity,
t.Distance as Distance
```

8. Spark

8.1. Data upload into cluster

Compute

All-purpose clusters | Job clusters

Create Cluster

All | Cre

Name	State	Nodes	Runtime
Yellow	Running	1 (0 spot)	9.1 LTS (includes Apache Spark 3.1.2, Scal…

Yellow | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision histo

Cmd 2

```
# File location and type
file_location = "/FileStore/tables/MOCK_DATA__16_-1.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

▸ (2) Spark Jobs

	username	first_name	last_name	gender	country	state	city	latitude	longtitude	seccountry	secstate	destcity	date
1	ganthonies0	Giffy	Anthonies	Male	Germany	Baden-Württemberg	Freiburg im Breisgau	48.0044582	7.8374651	Germany	Baden-Württemberg	Heilbronn	2022/
2	sannakin1	Sunny	Annakin	Female	Germany	Baden-Württemberg	Reutlingen	48.5373555	9.2004305	Germany	Baden-Württemberg	Reutlingen	2022/
3	etite2	Elberta	Tite	Female	Germany	Baden-Württemberg	Stuttgart	48.7226206	9.1584836	Germany	Baden-Württemberg	Stuttgart	2022/
4	ptheodoris3	Perceval	Theodoris	Female	Germany	Baden-Württemberg	Freiburg im Breisgau	48.0044582	7.8374651	Germany	Baden-Württemberg	Stuttgart	2022/
5	jstuttard4	Jory	Stuttard	Male	Germany	Baden-Württemberg	Stuttgart Stuttgart-Mitte	48.7643143	9.1752629	Germany	Baden-Württemberg	Karlsruhe	2022/
6	bfransman5	Bethany	Fransman	Female	Germany	Baden-Württemberg	Freiburg im Breisgau	47.9963155	7.8107621	Germany	Baden-Württemberg	Reutlingen	2022/
7	vdeavin6	Violante	Deavin	Female	Germany	Baden-Württemberg	Mannheim	49.4637092	8.560081	Germany	Baden-Württemberg	Freiburg im Breisgau	2022/

Showing all 1000 rows.

Figure 8.1-1 Car Database is uploaded in cluster

8.2. Data analysis on PySpark Data frame

Cmd 6

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
dfpd=df.toPandas()
dfpd.head()
```

▸ (1) Spark Jobs

	username	first_name	last_name	gender	country	state	city	latitude	longtitude	seccountry	secstate	destcity	date	seclat	seclon
0	ganthonies0	Giffy	Anthonies	Male	Germany	Baden-Württemberg	Freiburg im Breisgau	48.0044582	7.8374651	Germany	Baden-Württemberg	Heilbronn	2022/02/08	49.1376432	9.1776059
1	sannakin1	Sunny	Annakin	Female	Germany	Baden-Württemberg	Reutlingen	48.5373555	9.2004305	Germany	Baden-Württemberg	Reutlingen	2022/02/07	48.5373555	9.2004305
2	etite2	Elberta	Tite	Female	Germany	Baden-Württemberg	Stuttgart	48.7226206	9.1584836	Germany	Baden-Württemberg	Stuttgart	2022/02/08	48.7893977	9.1962771
3	ptheodoris3	Perceval	Theodoris	Female	Germany	Baden-Württemberg	Freiburg im Breisgau	48.0044582	7.8374651	Germany	Baden-Württemberg	Stuttgart	2022/02/05	48.8096645	9.2415192
4	jstuttard4	Jory	Stuttard	Male	Germany	Baden-Württemberg	Stuttgart Stuttgart-Mitte	48.7643143	9.1752629	Germany	Baden-Württemberg	Karlsruhe	2022/02/04	48.9728482	8.4060268

Figure 8.1-1

```
dfpd.describe()
```

	username	first_name	last_name	gender	country	state	city	latitude	longtitude	seccountry	secstate	destcity	date	seclat	seclon
count	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
unique	1000	936	992	8	1	1	7	10	10	1	1	7	8	10	10
top	csothern2c	Grace	Fries	Male	Germany	Baden-Württemberg	Stuttgart	48.7893977	9.1962771	Germany	Baden-Württemberg	Stuttgart	2022/02/07	48.0044582	7.8374651
freq	1	3	2	457	1000	1000	330	117	117	1000	1000	295	147	120	120

Figure 8.2-2

8.3. Visualization of analysis results

```
x=dfpd['destcity'].unique()
y=[]
for i in x:
    y.append(dfpd.loc[dfpd['destcity']==i].shape[0])
print(y)

fig,ax=plt.subplots()
fig.set_size_inches(15, 8)
ax.bar(x,y,label='Destination Cities')

ax.title('Number of Travellers')
ax.xlabel('Cities')
ax.ylabel('Qty.')

ax.legend()
```

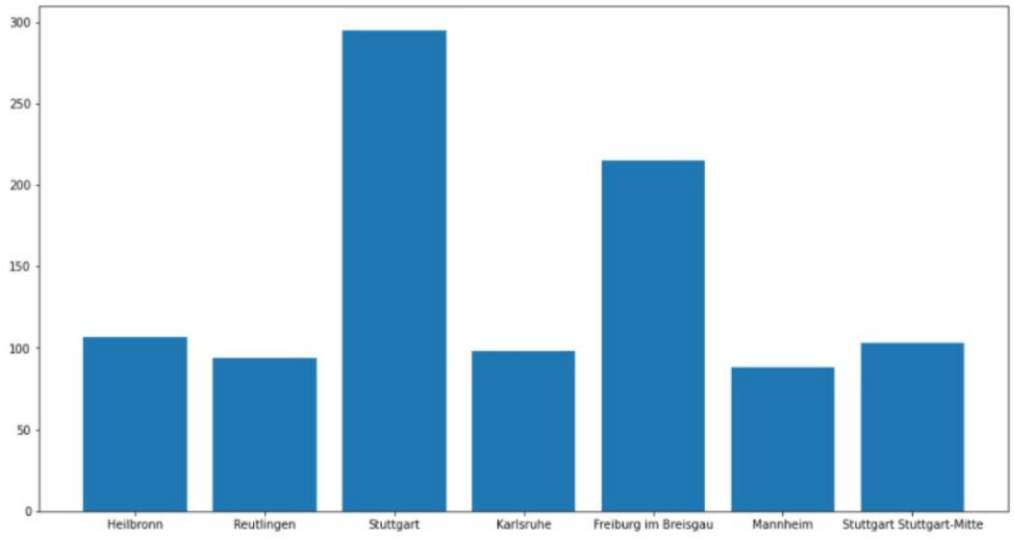

Figure 8.3-1

```
dates=dfpd['date'].unique()
car_count=[]
for i in dates:
   car_count.append(dfpd.loc[dfpd['date']==i].shape[0])

fig,ax=plt.subplots()
fig.set_size_inches(28, 8)
ax.plot(dates,car_count,label='dates')

ax.title('Number of cars in the week')
ax.xlabel('Dates')
ax.ylabel('Number of Cars')

ax.legend()
plt.show()

```

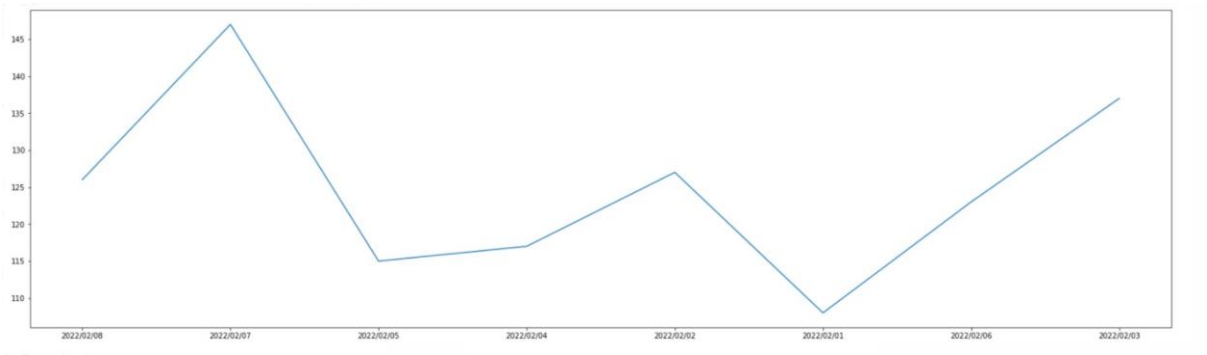

Figure 8.3-2

9. GitHub repository path

https://github.com/Rrschch-6/Data-Engineering-Car-pooling-Application-Team-Yellow

10. Appendix A: References

https://university.redis.com/

https://university.mongodb.com/

https://neo4j.com/graphacademy/university-program/

https://stackoverflow.com

https://docs.mongodb.com/manual/reference/operator/aggregation