Project Report

Advanced Databases

[April - May 2024]

Parcel Service Application

Gayatri Samal, 11037887

Ketan Darekar, 11037367

Cris Kohn, 11014674

Vipul Zope, 11037893

Prateeksha Bheemareddy, 11038208

SRH Hochschule Heidelberg

Faculty for Information, Media &Design

Degree: M.Sc. Applied Computer Science

Under the Supervision of

Professor Frank Hefter

1 Introduction

Welcome to **Rapid Route Parcel Services**, where every shipment we handle is sent on a journey designed with precision, care, and an unwavering dedication to quality. In the fast-paced world of package delivery, where the race against time is won in seconds and client trust is as valuable as the goods we transport, we are reinventing the dynamics of courier services. At the heart of our business is a novel approach to logistics, which combines cutting-edge technology and human understanding to transform normal shipping into a thoughtfully created service experience. Our network, which spans continents and cultures, leverages the power of modern routing technology and real-time data analytics to ensure that every package not only reaches its destination safely but also with maximum efficiency and sustainability.

Here, innovation is ongoing. From automated sorting facilities that whisper the future to GPS-enabled vans that track journeys down to the last meter, we provide your deliveries with the pinnacle of logistical technology. But at Rapid Route, we believe that technology alone is insufficient—it's the collaboration with our devoted team that elevates your shipping experience. From pickup to delivery, our dedicated and diligent personnel serve as stewards of your cargo, keeping a keen eye and a kind hand on them. **Rapid Route** places a high priority on security. We recognize that behind every parcel is a person: a sender with expectations and a receiver with eagerness. That's why we implement sophisticated security measures, such as encrypted data protocols and tamper-evident packaging, to ensure the integrity.

Our **Rapid Route Parcel Service** website streamlines the entire process of sending, tracking, and managing deliveries in a user-friendly online environment. Users can quickly create shipments by entering parcel details such as weight, dimensions, and destination, and then choose from various shipping options based on price and delivery speed. The website provides real-time tracking for parcels, allowing both senders and recipients to view the current status and location of their shipment through a tracking number. Additionally, the site offers customer service features, such as scheduling pickups, managing returns, and accessing shipping history. Integrated payment gateways ensure secure transactions, while user accounts help customers save their preferences and details for future convenience.

2 Organization

2.1 Roles and Responsibilities

	Team Members				
Tasks	Gayatri	Ketan	Cris	Vipul	Prateeksha
Project Analysis	R	R	R	R	R
Requirement Gathering	R	R	I	R	R
Selection of Database	R	R	R	R	R
User Stories	R	R	R	R	R
Data Setup	R	I	I	I	I
Neo4j Setup	R	R	R	R	R
MongoDB setup	R	I	I + R	I	I
Redis Setup	R	I	R	I	I
Data Flow/ UML diagram	R	R	R	R	R
Frontend Programming	R	I	R	I	I
Backend Programming	R	R	R	R	R
Report Documentation	R	R	I	R	R

**R= Responsible, C= Consulted, I= Informed

2.2 Meetings

Date	Topic	Outcome	Attendees
22 April	Project Topic decision, Alternatives	Contested ideas related to Use Cases, decided to work on Use Cases, initiated brainstorming session.	Gayatri, Ketan, Vipul, Prateeksha.
23 April	Refining project idea, Defined use cases	Got a clearer picture of the project idea. Decided to do preliminary research into DHL.	Gayatri, Ketan, Vipul, Prateeksha.
24 April	Confirmed use cases,	Use cases were approved by the professor. The first task to complete was our Front-end Part .	Gayatri, Ketan, Vipul, Prateeksha.
25 April	In person session to Discuss progress, Worked on Neo4j issue	Progress on the first task was discussed. Worked and fixed an issue with Merge Conflicts.	Gayatri, Ketan, Vipul, Prateeksha.
26 April	In person session to Discuss progress and divided tasks	Vipul & Gayatri – complete Neo4j data insertion, verify available paths with frontend, prepare bulk insert script/commands. Ketan – work on mongo DB schema for storing landmarks for map. Gayatri & Prateeksha – work on computing shortest path in Neo4j Cris – design and start work on simulated Drone Parcel Deliveries .	Everyone.
27 April	In person session to Discuss progress and planned tasks	Ketan – search locations API Vipul, Gayatri, and Prateeksha work on shortest path alg. with priority	Gayatri, Ketan, Vipul, Prateeksha.

28 April	In person session to Discuss progress and planned tasks	Discussed how to implement Packstation feature. Decided to have long in-person session on Sunday. Gayatri- display landmarks on the map. Vipul, Ketan, Prateeksha - use the weighted relationships to find the shortest path between two points.	Gayatri, Ketan, Vipul, Prateeksha.
29 April	Report Discussion	Discussed neo4j migration Helped each other with common development environment setup. Worked on API endpoint for shortest path calculation	Gayatri, Ketan, Vipul, Prateeksha.
30 April	Report and Progress Discussion	Discussed problems with Aura DB migration. Updated progress. Decided to start work on project report draft	Gayatri, Ketan, Vipul, Prateeksha.
2 May	In person session for Report and Development	Discussion about the report and the content. Shared progress updates.	Everyone.
4 May	Report discussion, Task progression	Discussed implementation issues. Shared diagrams for the report.	Gayatri, Ketan, Vipul, Prateeksha.
5 May	Report discussion, Task progression.	Decided to work on updating the report with priority. Formalised user stories, redistributed tasks.	Gayatri, Ketan, Vipul, Prateeksha.
6 May	Report discussion, Task progression.	Car simulation tasks were implemented. Additional rules added to pick up and transit states.	Gayatri, Ketan, Vipul, Prateeksha.
7 May	Report discussion, Task progression.	Worked on use case diagrams, data flow diagrams and the database ER diagrams. Discussed the next steps regarding the project report.	Gayatri, Ketan, Vipul, Prateeksha.

8 May	Presentation Planning	Discussed what should be shown for the demo part of the presentation. Updated all diagrams	Gayatri, Ketan, Vipul, Prateeksha.
9 May	Final Meeting	Planned the presentation, rehearsed the demo, and exported the final report as pdf.	Gayatri, Ketan, Vipul, Prateeksha.

2.3 Tools

Topic	**Name**
Databases	MongoDB Atlas Neo4j Redis Desktop
IDE	Visual Studio Code, PyCharm Community
Diagrams	Draw.io
Documentation	Microsoft Word
Project Management	Trello Kanban Board

3 User Stories

1. Dynamic Route Optimization for Express Delivery:

As a messenger, I want to receive optimized delivery routes in real-time based on traffic conditions and parcel priorities, so that I can minimize travel time and ensure timely deliveries.

As a customer, I want to see accurate delivery times and tracking information, so that I can plan for parcel receipt efficiently.

2. Real-Time Parcel Tracking:

As a customer, I want to track the real-time location of my parcel and receive notifications at key delivery stages, so that I can monitor its progress and be informed about any delays or exceptions.

3. Pack station Slot Availability and Package Deposit:

As a user, I want to locate nearby pack stations with available slots for package deposit, so that I can efficiently deposit my package at a convenient location and track its delivery.

4. Demand Analysis for Messenger Allocation:

As an admin**,** I want to analyse delivery demand in different areas based on parcel priorities and historical data, so that I can allocate messengers effectively to meet service demands and prioritize urgent deliveries.

5. Drones delivery

As an admin, I want to see the location and the current routes of the drones.

3.1 Use Cases

1. Dynamic Route Optimization for Express Delivery
2. Real-Time Parcel Tracking
3. Packstation Slot Availability and Package Deposit
4. Demand Analysis for Messenger Allocation
5. Drones delivery

4 Details

4.1 Dynamic Route Optimization for Express Delivery (Gayatri)

4.1.1 User Story

As a messenger I want to see how many parcels are pending for deliver for different locations, I want to see a dynamic routing system that provides real-time optimized delivery routes based on traffic, roadworks, and other unforeseen delays, so that I can deliver parcels efficiently and minimize travel time and distance also fuel consumption.

4.1.2 Identified Use Case

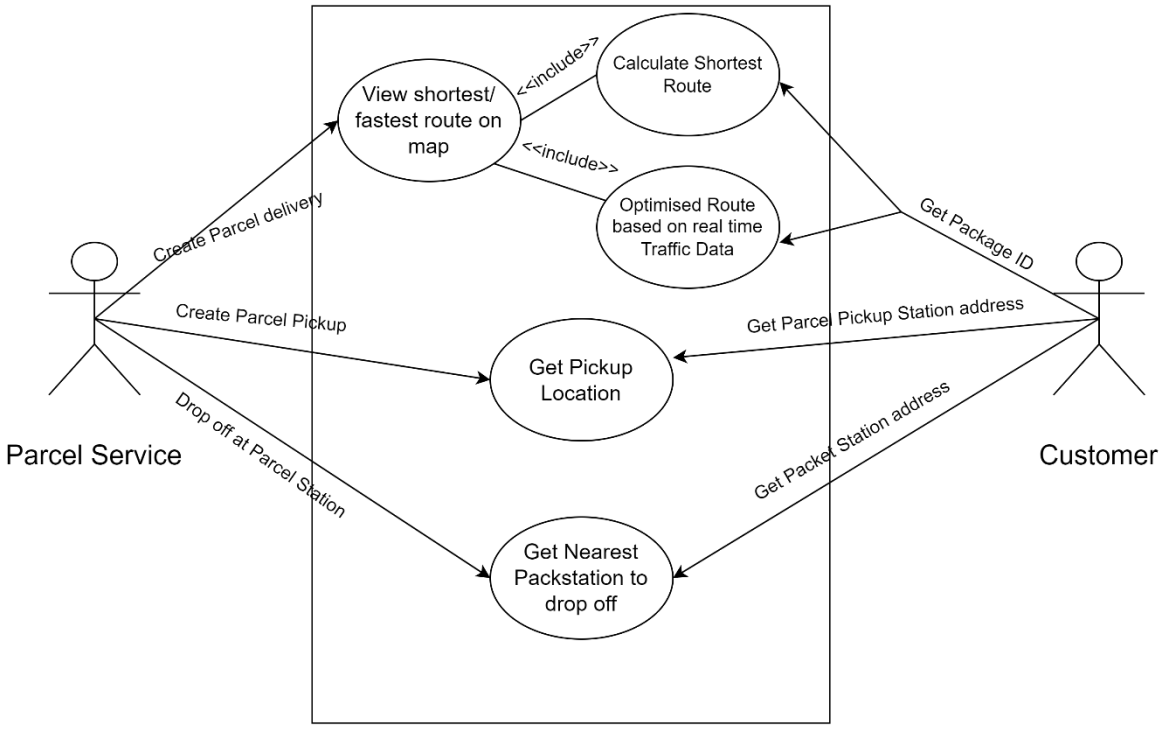


fig (4.1.2) Dynamic Route Optimization UML Diagram

4.1.3 Actors

Parcel Service: Parcel Service can create delivery, pickup and drop off the parcel.

Customer: Customer gets Packet Station address in order to drop of the Parcel to the nearest Station.

4.1.4 Task Description

The process of dynamically optimizing routes for parcel delivery begins at the distribution center, where each messenger is assigned, a van loaded with parcels. The condition of the vans is checked, and they're equipped with GPS and other tracking technologies to ensure efficient navigation. Once the vans are loaded, the routing system calculates the most efficient routes based on various factors such as the messenger's starting point, the parcel load, and expected delivery times.

The routing system is designed to adapt in real-time, adjusting routes based on live traffic data, roadworks, accidents. As messengers set off, they use a mobile app or GPS to follow the optimized route, adjusting as needed based on updated information. At each delivery point, the messenger scans the parcel's barcode or QR code to confirm delivery, and the system records the time, location, and recipient details. These updates are sent to a central system, allowing for real-time tracking and monitoring of the delivery process.

Customer interaction is a significant component of this use case. Customers receive automated notifications with tracking id number with estimated delivery times and can track their parcels through a web or mobile interface. Once the parcel is delivered, customers can confirm receipt through the same interface.

To support this process, the system uses a range of technologies. The routing system provides dynamic route optimization, adjusting in real-time. The mobile app guides messengers through their routes, and the GPS enables real-time tracking. Automated customer notifications are sent through services like SMS, email, or push notifications. A central database stores parcel information, delivery records, and customer data to maintain an accurate record of the entire delivery journey.

4.1.5 Dataflow

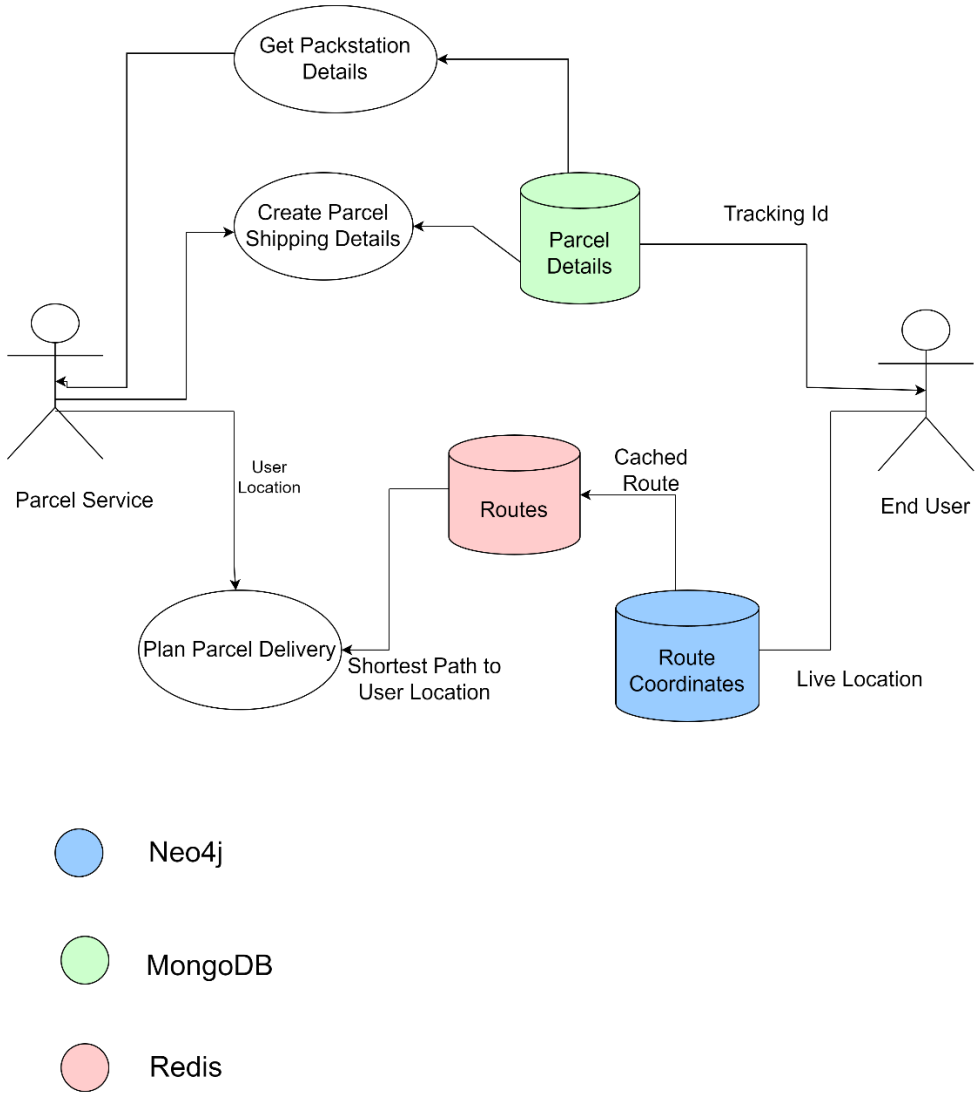


Fig (4.1.5) Dynamic Route Dataflow Diagram

4.1.6 Database

4.1.6.1 Database Used

MongoDB is used to store key information about locations, parcels, and addresses in a flexible and scalable manner.

Redis is used to handle real-time data and state management for dynamic routing.

Using Neo4j to model the delivery network as a graph, with nodes representing addresses and edges representing routes between them.

4.2 Real-Time Parcel Tracking (Ketan)

4.2.1 User Story

As a User, I want to track the real-time location of my parcel and receive notifications at key delivery stages, so that I can monitor its progress and be informed about any delays or exceptions.

4.2.2 Identified Use Case

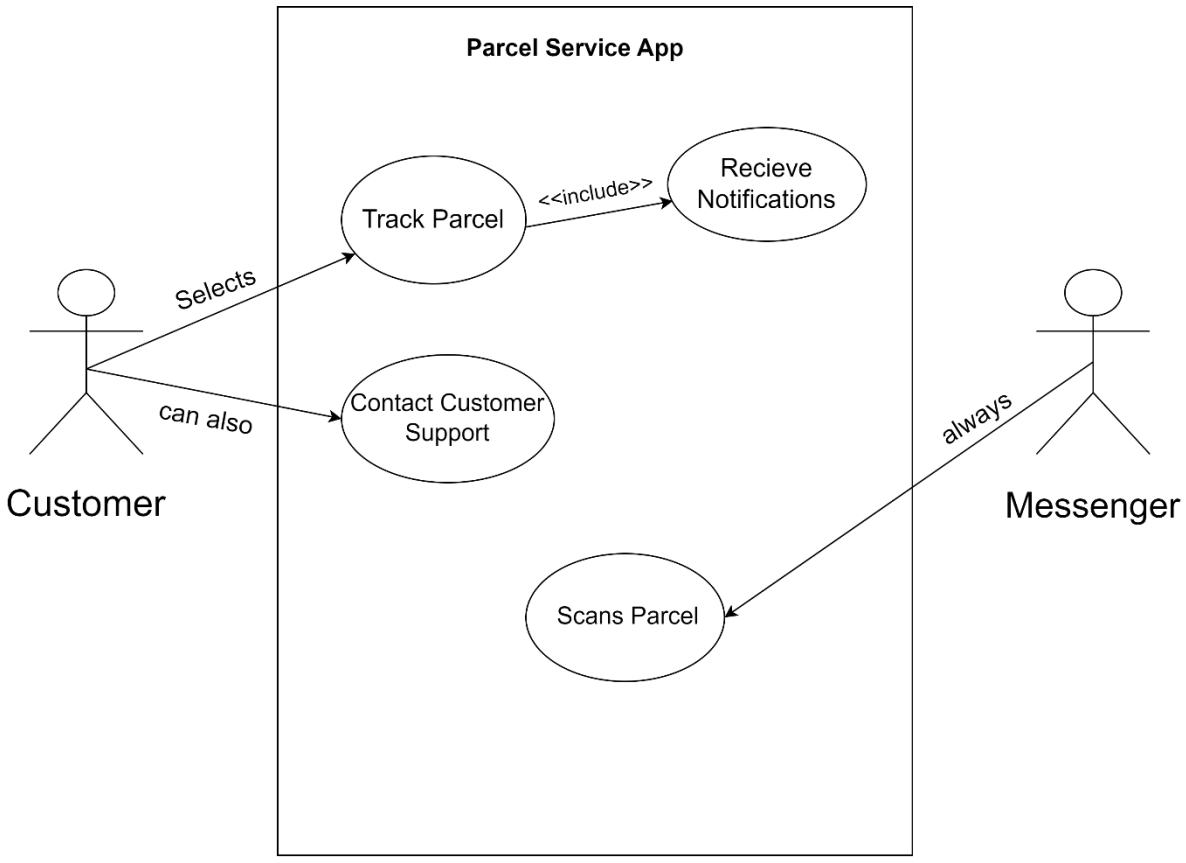


Fig (4.2.2): UML Diagram for Real Time Tracking

4.2.3 Actors

Messenger: Messenger always scans the Parcel before delivering to Customers.

Customer: Whenever Customer selects Track Parcel option, he/she receives notification about Parcel.

4.2.4 Task Description

The real-time parcel tracking system begins with parcel registration, where each parcel is assigned a unique tracking identifier, such as a barcode or QR code. This identifier, along with details like the sender's and recipient's information, is entered into a central tracking system. As parcels move through various checkpoints, including pickup locations, sorting facilities, transit hubs, and delivery vehicles, they are scanned to update their current status and location.

The central tracking system stores these tracking details, providing real-time updates to customers through web and mobile interfaces. Automated notification alerts are sent to customers at key stages of the delivery process. These notifications can include messages like "out for delivery," "delivered," or even exceptions such as "delayed due to weather" or "attempted delivery unsuccessful." Alerts are sent through channels like email, SMS, or push notifications.

Transportation and sorting are crucial tasks within this process. Parcels must be efficiently sorted and routed through transit hubs, ensuring timely delivery. As parcels approach their destination, they are marked "out for delivery," indicating they are on their way to the recipient.

Customers play an active role in the process. They can track parcels in real-time, receive notifications, and even update delivery preferences, such as changing the delivery address or rescheduling delivery times. Upon successful delivery, customers confirm receipt of the parcel, completing the delivery cycle. The system records this final status, including the date and time of delivery, along with additional information like the recipient's signature, if applicable.

4.2.5 Dataflow

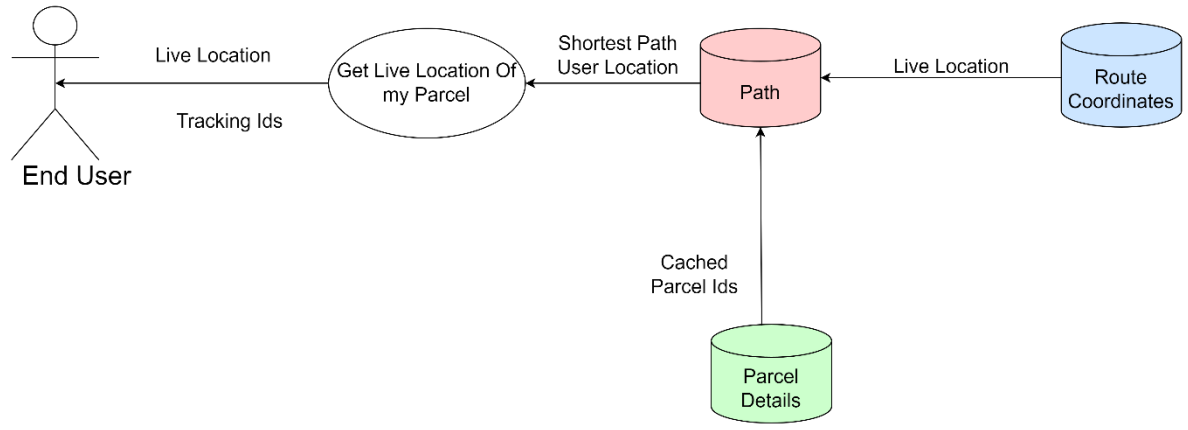


Fig (4.2.5) Data flow Diagram for Real Time Tracking

4.2.6 Database

4.2.6.1 Database Used

MongoDB is a document-oriented database that can be used to store detailed parcel information and associated tracking data. Its flexibility and scalability make it suitable for maintaining a comprehensive record of parcels and their journey through the delivery process.

Neo4j is a graph database designed to manage complex relationships and perform graph-based queries. It is ideal for visualizing delivery networks and identifying optimal routes.

Redis is an in-memory data structure store that excels in caching, session management, and real-time messaging. It's useful for providing fast access to frequently queried data and sending real-time notifications to customers.

4.2.6.2 Expressions Used for this Use Case

1. This Cypher script is designed to process a KML (Keyhole Markup Language) file to create or update nodes in a Neo4j graph database, which is often used for spatial or geographic data analysis.

```
CALL apoc.load.xml("heidelberg-alt.kml", "/kml/Document/Placemark", {}, false)
YIELD value
UNWIND value as item
WITH [i in item._children WHERE i._type = "LineString"][0] as lineString,
     [i in item._children WHERE i._type = "ExtendedData"][0] as extData
UNWIND [item in extData._children WHERE item.name IN ["@id", "highway", "maxspeed", "ref"]] as filteredDataList
WITH lineString,
     COLLECT(CASE WHEN filteredDataList.name = "@id" THEN filteredDataList._children[0]._text END) AS pathName,
     COLLECT(CASE WHEN filteredDataList.name = "highway" THEN filteredDataList._children[0]._text END) AS
pathType,
     COLLECT(CASE WHEN filteredDataList.name = "maxspeed" THEN filteredDataList._children[0]._text END) AS
maxSpeed,
```

```
WITH lineString, pathName[0] AS pathName, pathType[0] AS pathType, maxSpeed[0] AS maxSpeed, refName[0] AS
refName
UNWIND lineString._children as coordsList
WITH [item in coordsList WHERE item._type = "coordinates"][0]._text as coords, pathName, pathType, maxSpeed,
refName
WITH SPLIT(coords, " ") as gpsList, pathName, pathType, maxSpeed, refName
UNWIND range(0, size(gpsList) - 1) AS index
WITH SPLIT(gpsList[index], ",") as gps, pathName, pathType, maxSpeed, refName, index
MERGE (rc: RC { lat: toFloat(gps[1]), lng: toFloat(gps[0])})
ON CREATE SET rc.path = [pathName], rc.pos = [index]
ON MATCH SET rc.path = rc.path + pathName, rc.pos = rc.pos + index
MERGE (p: Way { name: pathName, type: pathType, maxSpeed: COALESCE(maxSpeed, 50), ref: COALESCE(refName,
```

2. This Query is used to create relationships between nodes in a Neo4j graph database, representing geographic or spatial connections based on common paths and relative positions.

```
MATCH (rc1:RC), (rc2:RC)
WHERE rc1 <> rc2
AND ANY(pathName IN rc1.path WHERE pathName IN rc2.path)
WITH rc1, rc2, [pathName IN rc1.path WHERE pathName IN rc2.path][0] as
commonPathName
WITH rc1, rc2, apoc.coll.indexOf(rc1.path,commonPathName) as rc1idx,
apoc.coll.indexOf(rc2.path,commonPathName) as rc2idx
WHERE (rc1.pos[rc1idx] = rc2.pos[rc2idx] + 1 OR rc1.pos[rc1idx] = rc2.pos[rc2idx]
- 1)
CREATE (rc1)-[:WAY { distance: 1}]→(rc2)
RETURN ID(rc1) AS rc1_id, ID(rc2) AS rc2_id, rc1idx, rc2idx.
```

4.3 Packstation Slot Availability and Package Deposit (Prateeksha)

4.3.1 User Story

As a user, I want to ship a package, I can check all the nearby packstations where I can deposit the package for shipping. Each packstation has limited slots (different sizes) and the user can check the available slot details for each packstation.

4.3.2 Identified Use Case

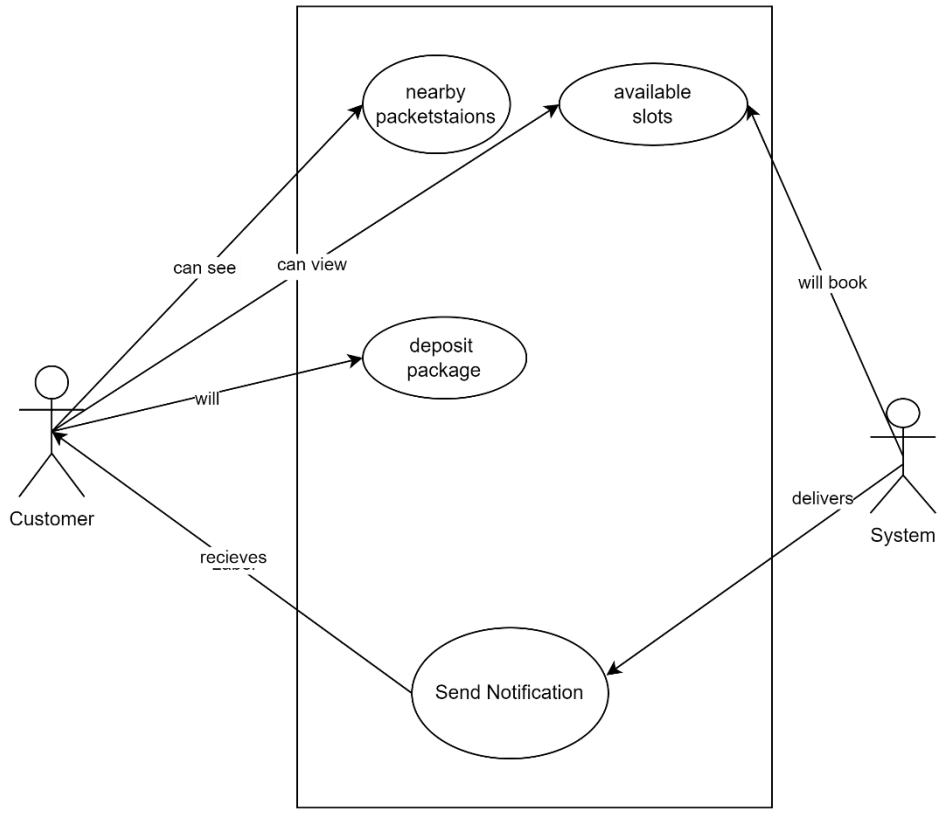


Fig (4.3.2): UML Diagram for Packstation Slot Availability

4.3.3 Actors

Customer: Customer can see the nearby packtstaions and can view the available slots and will deposit the package.

System: System will book the available slots and delivers the notification to the customer.

4.3.4 Task Description

The user, who needs to ship a package, access to the app to begin the shipping process. The app facilitates this process by providing the user with various functionalities. Initially, the user's current location is determined either through automatic retrieval or manual input. This location data serves as a crucial parameter.

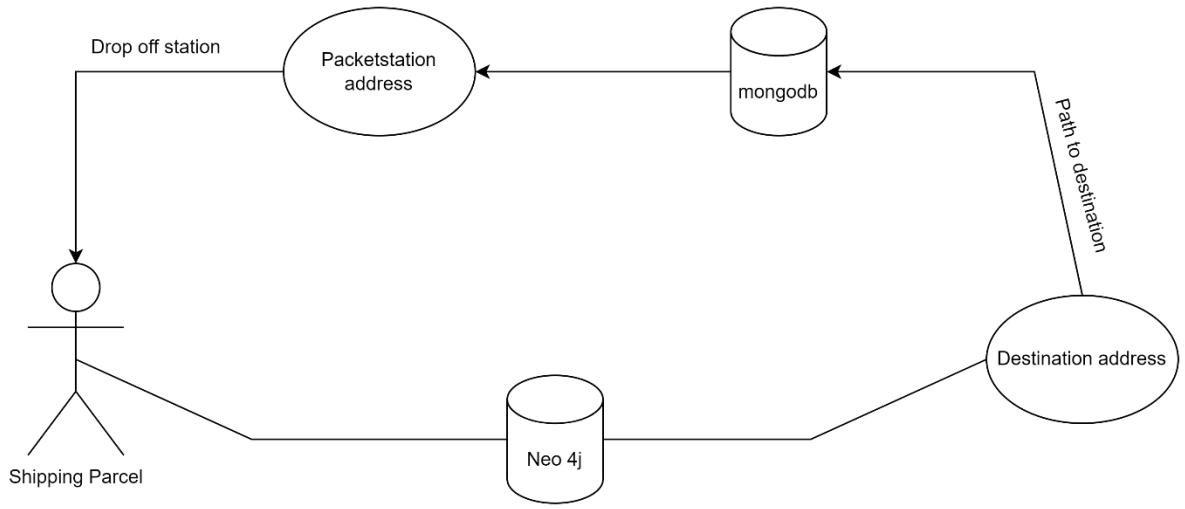
Utilizing the obtained location, the app proceeds to query the integrated database system, aiming to identify nearby packstations where the user can deposit the package for shipping. These packstations are strategically scattered across Heidelberg to ensure accessibility and convenience for the users. Upon retrieving the list of nearby packstations, the app meticulously organizes and presents this information to the user in a clear and intuitive manner. The user, now equipped with the list, can evaluate, and select the most suitable packstation based on factors such as proximity, operational hours, and available services.

Having made a selection, the user proceeds to indicate their choice within the app interface. This action triggers another database query, this time directed towards retrieving detailed information about the available slots at the chosen packstation. Each packstation boasts a finite number of slots, each differing in size and capacity.

The app swiftly processes the retrieved slot data and presents it to the user, empowering them with valuable insights regarding slot availability. Armed with this information, the user can make an informed decision about the most opportune time to visit the packstation for package deposition.

Throughout this entire process, the Rapidroute app ensures seamless navigation and interaction, fostering a user-centric experience characterized by efficiency, transparency, and convenience.

4.3.5 Dataflow



4.3.6 Database

4.3.6.1 Database Used

MongoDB: Used to store user data, including location preferences and shipping history.

Neo4j: Used to model and store relationships between packstations and available slots.

Redis: Used for caching frequently accessed data, such as packstation locations and slot availability.

4.4 Demand Analysis for Messenger Allocation (Vipul)

4.4.1 User Story

As an admin, I want to check the demand for messengers in different areas based on the delivery needs and get insight into which areas need to have more messengers dispatched for better service.

4.4.2 Identified Use Case

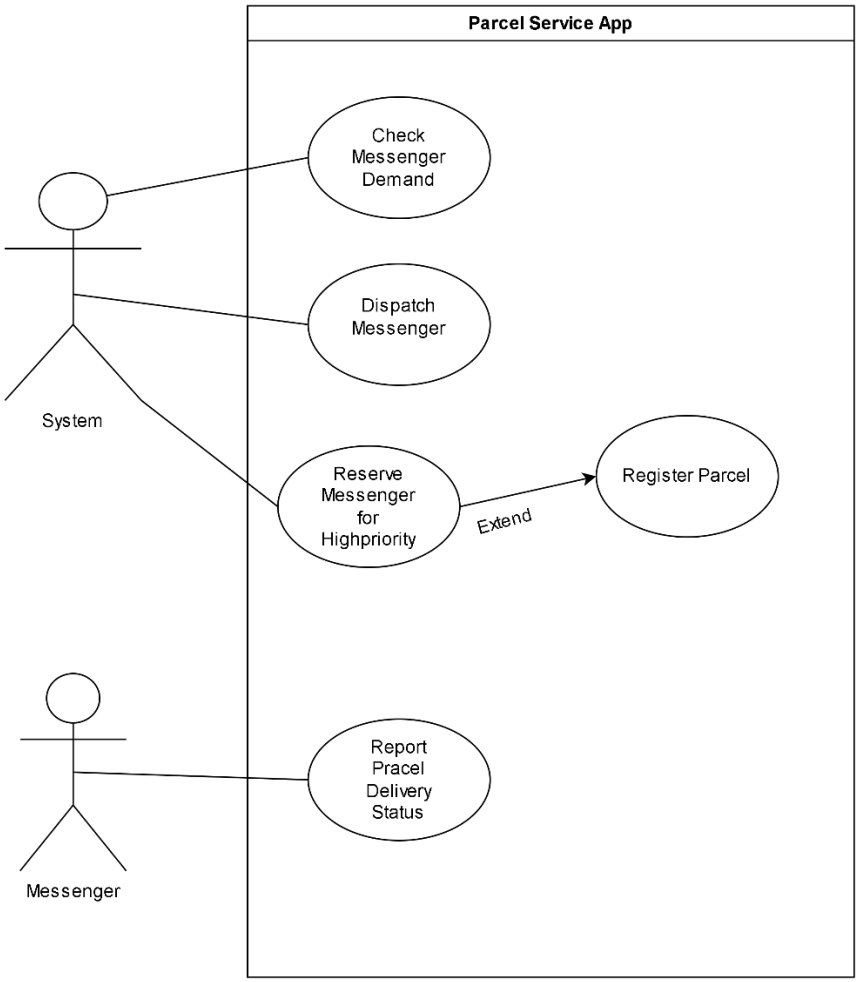


Fig (4.4.2): UML Diagram for analysis of Messenger Allocation

4.4.3 Actors

Admin: Admin checks Messenger's Demand

System: System dispatch the messenger to deliver the parcel and can register parcel itself and and in some of the condition it allows to reserve messenger for high priority

Messenger: Messenger here reports the parcel delivery status.

4.4.4 Task Description

The purpose of this project is to enhance the efficiency of our parcel delivery operations by analyzing demand for messengers in various geographic areas and ensuring that high-priority parcels receive prompt attention. To accomplish this, we will leverage MongoDB for structured data storage and Neo4j for modeling complex relationships. Specifically, MongoDB will store information about delivery areas, messengers, and parcels, while Neo4j will represent the relationships between these entities, such as which messengers are assigned to which areas and which parcels are assigned to which messengers. By maintaining at least one messenger reserved for high-priority parcels, we intend to streamline the delivery process, particularly for critical parcels like those containing medicines or passports.

Deliverables for this project consist of three primary components. First, the MongoDB collections will capture key data elements, including the Areas Collection with information on area names, delivery demand metrics, and the number of assigned messengers. The Messengers Collection will record individual messenger details, such as their current status and whether they are reserved for high-priority parcels. The Parcels Collection will maintain parcel-specific information, including the parcel's description, priority status, and delivery area.

Second, we will design a Neo4j graph model to represent the complex relationships in our delivery network. This graph-based representation will enable us to track which messengers are serving which areas and to which messengers parcels are assigned. The relationships among these entities will

facilitate a more visual understanding of our delivery operations, aiding in resource allocation and optimization.

The third component involves implementing a system for dispatching high-priority parcels. This entails creating a logic-based mechanism to ensure that reserved messengers are assigned to high-priority parcels as a matter of priority, ahead of other parcels. Once the high-priority parcels are dispatched, messengers will be assigned to other parcels based on their availability and other relevant factors.

Expected outcomes from this project include enhanced parcel delivery efficiency, with high-priority parcels being dispatched with minimal delay, and improved resource allocation based on demand metrics. This should lead to a higher level of customer satisfaction through the timely delivery of critical items. Furthermore, the use of Neo4j for relationship modeling will provide valuable insights into messenger assignments, helping management make more informed strategic decisions.

4.4.5 Dataflow

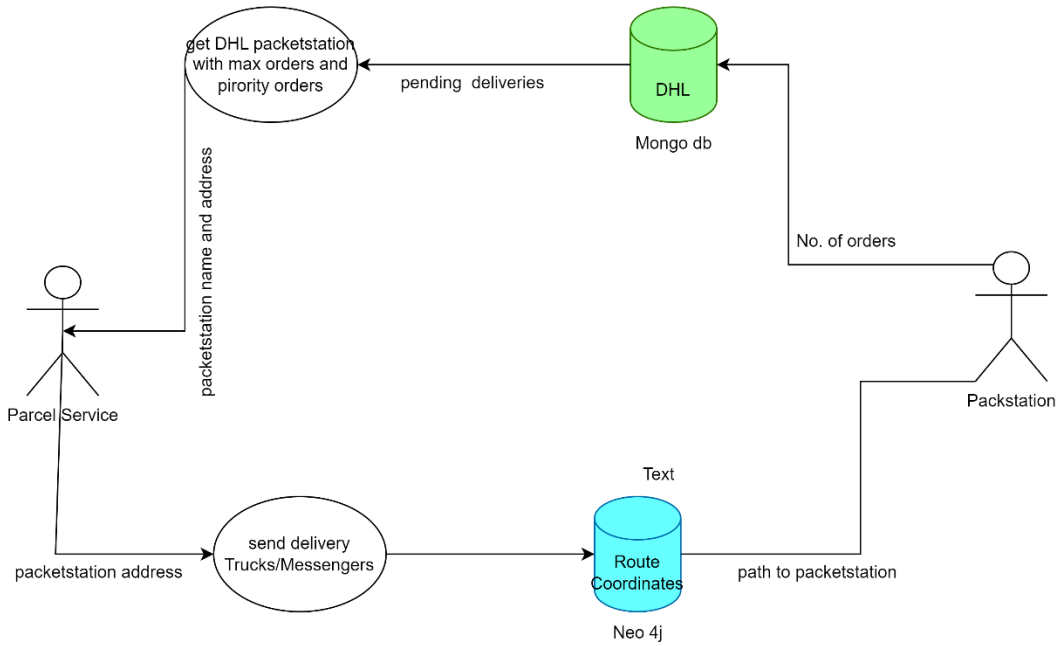


Fig (4.4.5) Data Flow Diagram for Dispatch Messenger

4.4.6 Database

4.4.6.1 Database Used

Using Neo4j to model the delivery network as a graph, with nodes representing addresses and edges representing routes between them.

MongoDB is used to store key information about messengers, parcels, and addresses in a flexible and scalable manner.

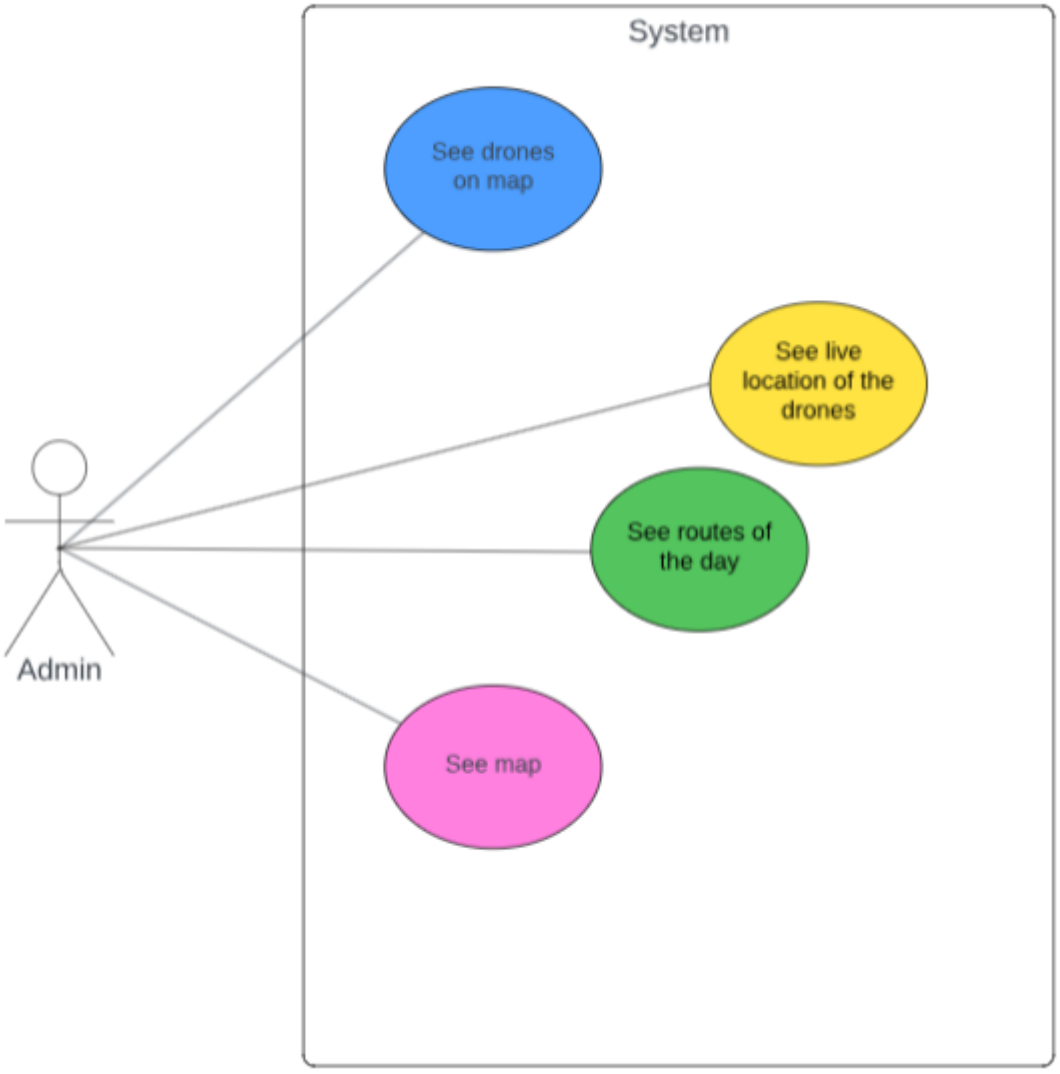
Redis is an in-memory data structure store that excels in caching, session management, and real-time messaging. It's useful for providing fast access to frequently queried data and sending real-time notifications to customers.

4.5 Drones Tracking (Cris)

4.5.1 Task Description

Our parcel delivery service offers the additional service to deliver parcels via drones to some addresses in Heidelberg. The system admin wants to have a whole overview of the drones on the current day and their routes that they are currently flying.

4.5.2 Identified Use Case



4.5.3 Actors

The administrator in this use case is an employee of the parcel delivery enterprise who wants to have a complete overview of the location of the drones and their current location and current routes.

4.5.4 Database interactions

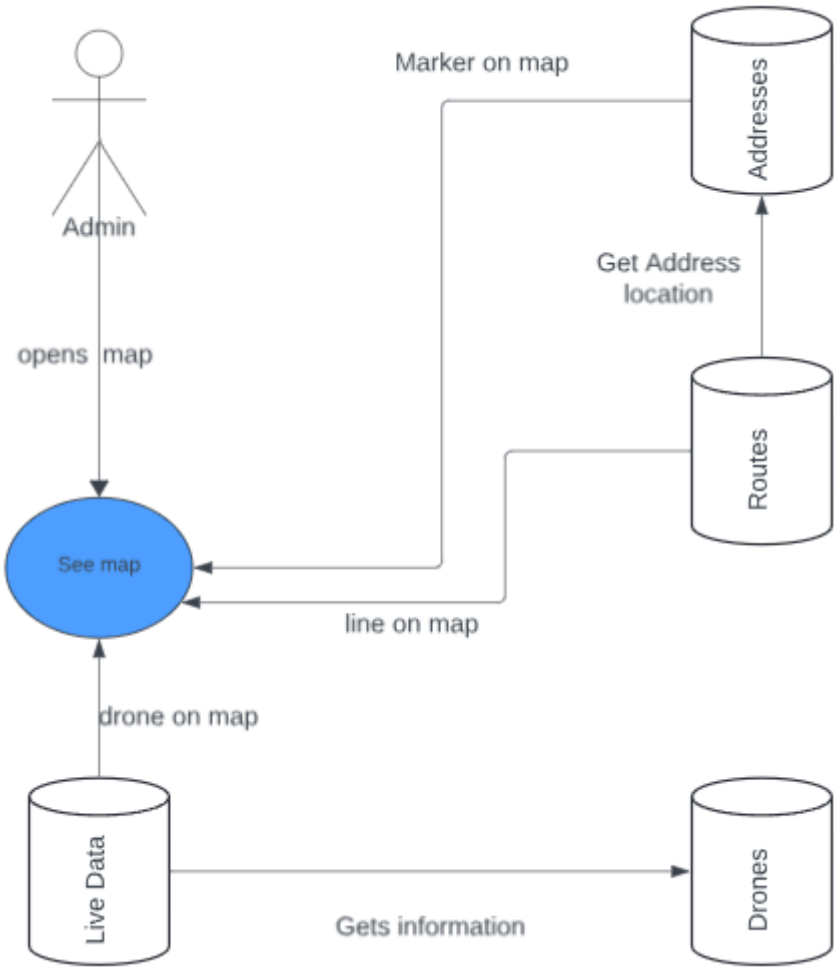
The Addresses are stored in MongoDB, document based with all the details and the geographic locations. In Neo4J, the routes are defined, starting with the route number, then the start point followed by the end point. The start and end points only have the id stored in them as the geographical location will be retrieved from MongoDB. A collection of drones is also saved in MongoDB with the name and if the drone is available.

In Redis the live location and the battery percentage are saved in a set with the drone's name as key. All static information about the drones can be retrieved from MongoDB.

4.5.5 Frontend used.

To run the frontend just un the project in the IDE. Be aware that the databases need to be running.

4.5.6 Data Flow



4.5.7 Databases

4.5.7.1 Database used.

MongoDB has been used for storing the addresses and the drones with all the information that will not change that often and is not in relation with other data.

Neo4J has been used for defining the routes as the location can be connected together with relationships which makes it easier to get the next stop.

Redis has been used for the real-time data as all the other databases need to long to respond.

4.5.7.2 Expressions used

4.5.7.2.1 MongoDB

addresses_collection.find_one({"location.ids.locationId": str(address_id)}, {"location.ids.locationId": 1, "place.geo": 1})

addresses_collection.find({}, {"location.ids.locationId": 1, "place.geo": 1})

4.5.7.2.2 Neo4J

"MATCH (s:Location)-->(e:Location) RETURN s.location_id as start, e.location_id as end"

4.5.2.3 Redis

getall {drone-name}

5 Database

5.1 Overall Structure

Three types of NOSQL database are used in this project:

MongoDB- In our application, the use of MongoDB is to provide Scalable and Schema-flexible Data Storage.

Neo4j- Create Graph from nodes that find the relation and find the shortest and easiest path for Vans.

Redis- Store all information about each van, messenger. Redis will also store the paths that each Vans is assigned to follow.

5.2 Data Model Overview of all Databases

5.2.1 MongoDB (UML)

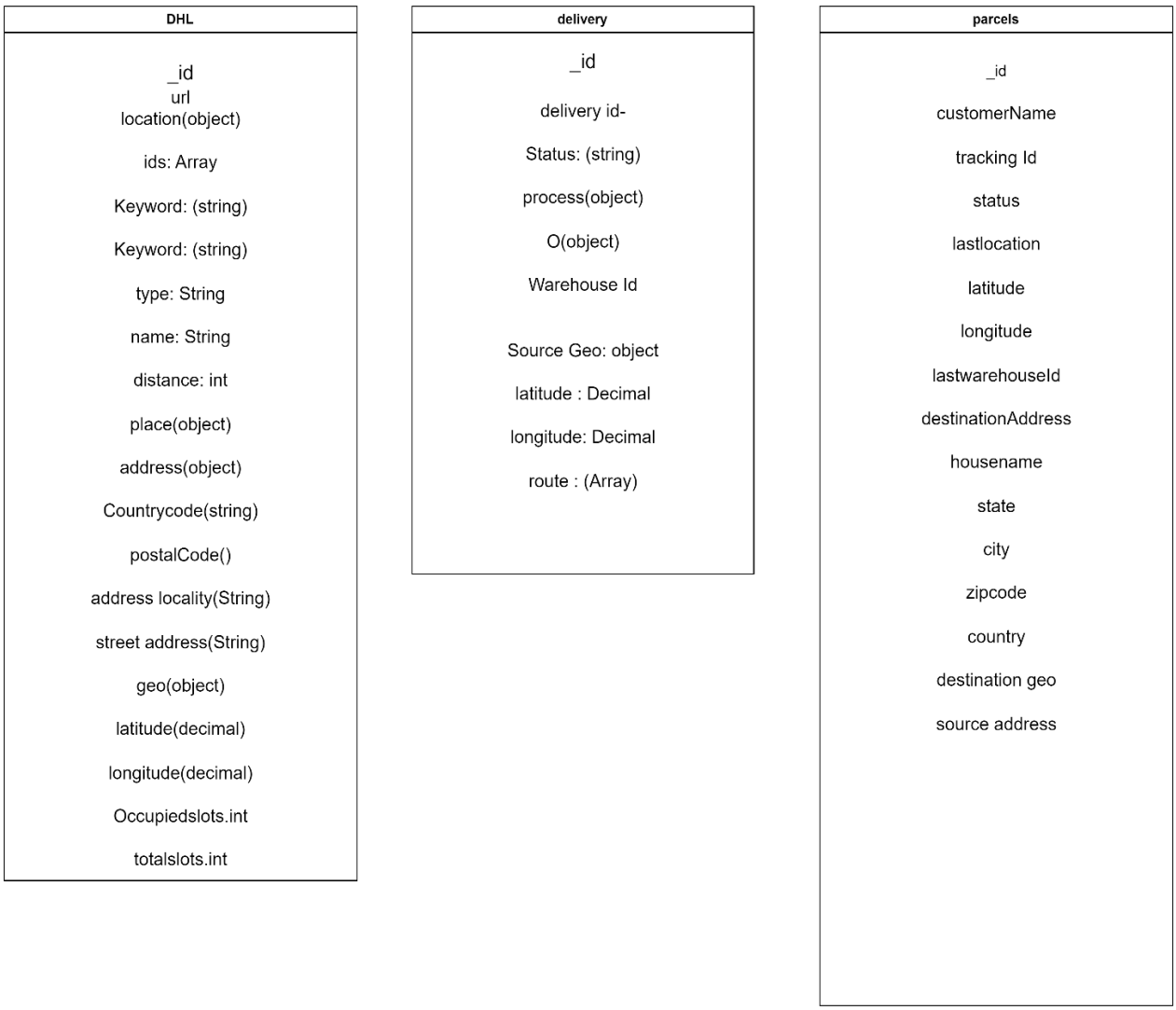
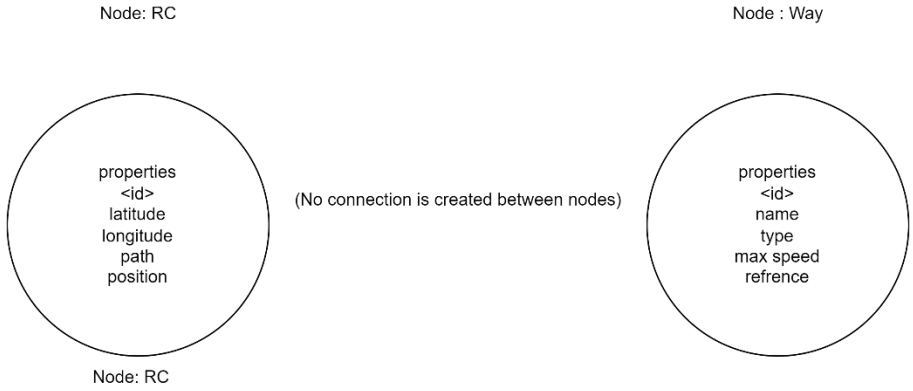
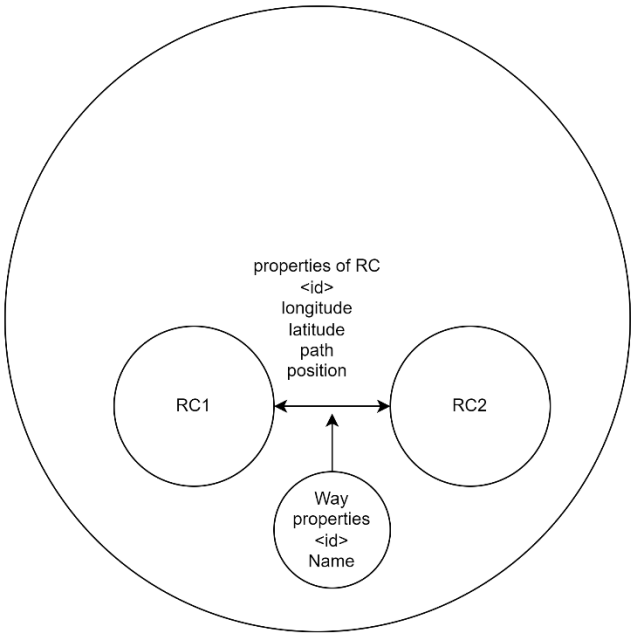


Fig (5.2.1) Integrated MongoDB Uml Diagram

5.2.2 Neo4j (UML)





(Fig 5.2.2) Integrated Neo4j UML Diagram

6 Application

6.1 Languages Used

- Next /JavaScript

  Next js is an react framework for frontend Development and has additional features like server-side rendering (SSR), static site generation (SSG), and built-in routing.
  Additionally, we utilized a selection of React libraries to address specific complex edge case tasks. This allowed us to allocate more time to focus on critical backend issues.
  Next.js provides static site generation capabilities. Which helped us to pre-generate static pages at build time, reducing server load and further enhancing performance.
  Next.js features a simple and intuitive file-based routing system. This eliminates the need for additional routing libraries and helps maintain a clean project structure.

- Python

  Python is a universal, interpreted programming language. Its strength is to deliver a clear but short programming style.
  There are a lot of libraries present in python, so that developers don’t need to program everything from scratch.
  Python was used because of its ease of use for the use-case number five, as there were issues with implementation in NextJs that couldn’t be fixed.

6.2 GitHub Repository

- Github Repository of use case 5 (public):
  https://github.com/Cris9807/databases_drones
  Databases are needed. Files are added in git.

- Github Repository for Use Case 1,2,3,4(Public):
  https://github.com/gayatrisamal7/paketeer-main--3-

7 Evaluation

7.1 Outcome

Throughout the course of this project, our knowledge of NoSQL databases has significantly deepened. At the outset, we had little familiarity with graph databases, but we've since grown to appreciate how they structure information. We were particularly impressed with Neo4j's ability to integrate various algorithms through its plugin system.

Redis has proven to be a powerful tool that we have come to rely on. Its in-memory architecture, along with its seamless integration with Python scripts, is a feature we had not previously realized we needed. Now, we find it preferable to SQLite for smaller hobby projects.

MongoDB has been invaluable for its flexibility and scalability, allowing us to store large volumes of data without worrying about retrieval issues. Its pipelines feature has enabled us to resolve problems that would otherwise have taken us several days to address.

This project marked our first experience using multiple databases to achieve a common objective. We have been impressed by the performance improvements gained by leveraging each database for its specific strengths.

7.2 Possible Extensions

Neo4J has simplified the storage and connection of location and route coordinates on a map. The Graph Data Science (GDS) library has also been useful for pathfinding algorithms. However, while we imported routes from Google My Maps into Neo4J, the imports did not contain key information, such as whether a route is one-way. Including this additional information would be a significant improvement, allowing us to determine the shortest paths more accurately in line with real-world conditions.

The shortest path algorithm we use for calculating emergency personnel pickup routes is not fully optimized for finding the quickest route among multiple points.

It would be better to use alternative methods to achieve the best path when considering multiple destinations.

Neo4J Cloud offers basic database functions for free. However, using advanced pathfinding algorithms from the GDS library incurs charges based on the number of nodes and the specific algorithms and functions used.

Regarding ride management, our system allows users to end a ride only if it is in normal mode. This could be extended to carpooling mode, giving users the flexibility to end their ride during a shared journey. Additionally, a feature to automatically end a ride if an error occurs in an autonomous vehicle could be helpful, with a new car assigned to the user in such cases.

Our car simulation system is somewhat integrated with the main application. Separating these components would make the system structure clearer. Carpooling currently only accommodates users traveling to the same destination, but it could be improved by allowing passengers to join and exit along the route, providing more flexibility for shared rides.

8 References

8.1 Webpage

Next JS Setup: https://nextjs.org/docs

Redis Setup: https://redis.io/docs/latest/

MongoDB Installation: https://www.mongodb.com/docs/

DHL API: https://developer.dhl.com/api-reference/location-finder-unified#get-started-section/

ChatGPT: https://chatgpt.com/

Next js Installation: https://www.youtube.com/watch?v=pCzofI8vPo4

MongoDB Installation: https://www.youtube.com/watch?v=gB6WLkSrtJk

Tailwind.CSS Documentation: https://tailwindui.com/documentation

Python Thinkter Map: https://github.com/TomSchimansky/TkinterMapView