PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
HIGHER SCHOOL OF COMPUTER SCIENCE

2nd Year Superior Cycle (2CS)

2022-2023

**Software Architecture Project**

# Distributed Microservices Architecture for Supply Chain Management System


**Realized By :**

- Mohammed Abderrahmane Bensalem
- Safa Zakaria Abdellah
- Oussama Hadj Aissa Fekhar
- Abderrahmane Boucenna

**Supervised By :**

- M.Riyadh ABDMEZIEM


Year : 2022-2023

**Abstract**

The supply chain management domain includes a vast amount of systems interacting with each other within a specified time frame in order to cooperate according to the standards of the industry. one observed room for optimizaing the supply chain process is the empty return trips that are under utilized .

To apply this solution we must first study the software architecture of the system and design it in a way that responds effectively to the system needs that were observed notably interoperability and performance .

In this document we study the environment , the structure, the driving quality attributes and the tactics of implementing the solutions to our software architecture and the diffrent architectural styles involved.

# Contents

# 1

# Project Description

## 1.1 The Proposed Project : freight management system for supply chain optimization

Algeria has an asymmetrical freight transport network that is largely the result of poor logistics organization. Transport companies are often contractually obliged to make an empty return trip, sometimes unpaid, after delivering their goods. This creates an inefficient and costly network where a considerable number of trucks make an empty return trip. adding to that the lack of communication or a clear network between shippers and transporters creates points of confusion. Our objective here is to design and implement a software solution for the management of heavy empty transports dedicated to empty returns which should ensure the automatic planning and organization of the carriers' returns and effectively Minimize carrier and shipping costs and automate the process of the return shipment.along side additional functionalities related to communication.

## 1.2 System Description

Our System is a platform composed of a web app for the administrator and transporter and a mobile app shippers and transporters, the system will have 3 types of actors :

### 1.2.1 The Transporters

Represents companies or corporations that have a fleet of transportation vehicles. A representative of this company enters the company's vehicles into the system

to participate in the process. The representative is mainly responsible for sharing empty return offers.

### 1.2.2 The Shipper

A shipper is a person who belongs to a company and who organizes the transport of goods in his own name, on a professional basis. He chooses shared offers of empty returns that are compatible with his needs and orders them through the system.

### 1.2.3 Administrator

Manages and supervises system users, their roles and privileges, he also manages the activities, operations and dashboard related to the empty return control system.

## 1.3 Architectural Challenges

The project contains a number of architectural challenges related to it's nature, we cite the obvious ones

- Integration with existing systems ,since the project will deal with a number of systems such as logistics,inventory management system ,commands system ,planification system... along side external systems such as mailing and maps ,this could lead to a difficult data synchronization and handling .
- the System must support a high number of vehicles and should be able to expand and scale up when adding new clients .
- the system contains real time data that is the location of the transportation vehicle, the performance could face degradation as the system scales up .

## 1.4 Stakeholders Description

### 1.4.1 Project Owner

#### 1.4.1.1 Ayrade

Ayrade is a service company that is specialized in web hosting, integration of information systems, ERP/CRM management and digitalization of companies. it will be the project owner .

### 1.4.2 Clients:

#### 1.4.2.1 Bejaia Logistiques

Founded in 2008, the Sarl BEJAIA LOGISTIQUE is today one of the actors of references in the field of road transport in ALGERIA. Their activities are extended from the public transport of goods, renting of machines and and equipment for construction, public works and handling, rental of vehicles...

#### 1.4.2.2 NUMILOG

In 2007, Numilog was created by the Cevital group to accompany the development of its activities and to ensure its logistical support. The company has been able to capitalize on its experience in the food, household appliance, retail, automotive and construction industries. In 2014, Numilog opened up to the external market and Today, Numilog has a turnover of 75 million euros and 1,400 employees.

### 1.4.3 Private investors :

Private investors provide financial investment for the development of the tourism project and may have a network of contacts and connections that can be valuable in promoting and marketing the solution.

## 1.5 Functional Description

### 1.5.1 Common

- The system allows any type of user to authentify.

- The system allows any type of user to see their profile.
- the system stores the actions of each user and allows to visualize it.

### 1.5.2 The Transporter

- The system allows transporter to access,add,modify or delete a vehicle to their list of vehicles along with its technical specifications.
- The system allows the transporter to add an empty return trip.
- The system allows to visualize an empty return trip to the transporter and allows for him to modify it before the supply chain process.
- The system allows the transporter to see the current geographic location of a chosen vehicle.
- The system calculates the distance taken by a chosen vehicle and displays it to the transporter and for statistics.
- The system notifies the transporter about new commands in his return.
- The system visualizes the waiting commands concerning his return trip .
- The system allows the transporter to choose the convenient command or commands or refuse it for his return trip.
- The system allows the user to visualize their dashboard.

### 1.5.3 The Shipper

- The system allows the shipper to access his geographical location,and see the geographical location of near by shippers.
- The system allows the shipper to see the empty return road plans.
- The system notifies the shipper about the modifications in empty return plan .
- The system allows the shipper to communicate with other shippers via mailing .
- The system allows the shipper to choose a return plan that fits him .

- The system allows the shipper to add,modify,cancel a command in a waiting state.
- The system notifies the shipper in case of acceptance or refusal of their command.
- The system allows the user to visualize their dashboard.

### 1.5.4 The Administrator

- The system allows the administrator to see the list of user and add to it.
- The system allows the administrator to disable a user.
- The system allows the administrator to see the list of vehicles and modify it.
- The system allows the administrator to see the statistics related to vehicles and usage.

### 1.5.5 Class Diagram :

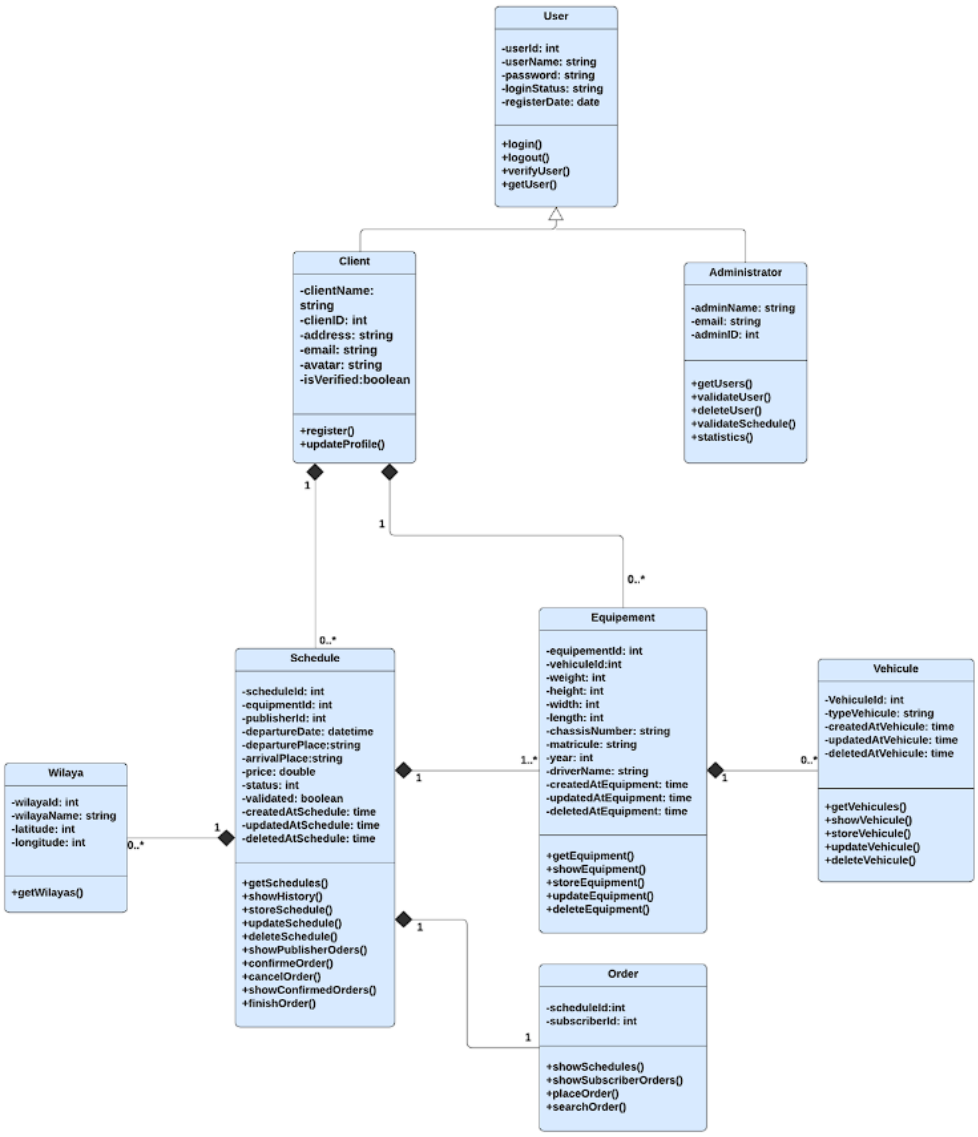



Figure 1.1: Class diagram of the freight management system

### 1.5.6 Use case diagrams

#### 1.5.6.1 Transporter use case diagram

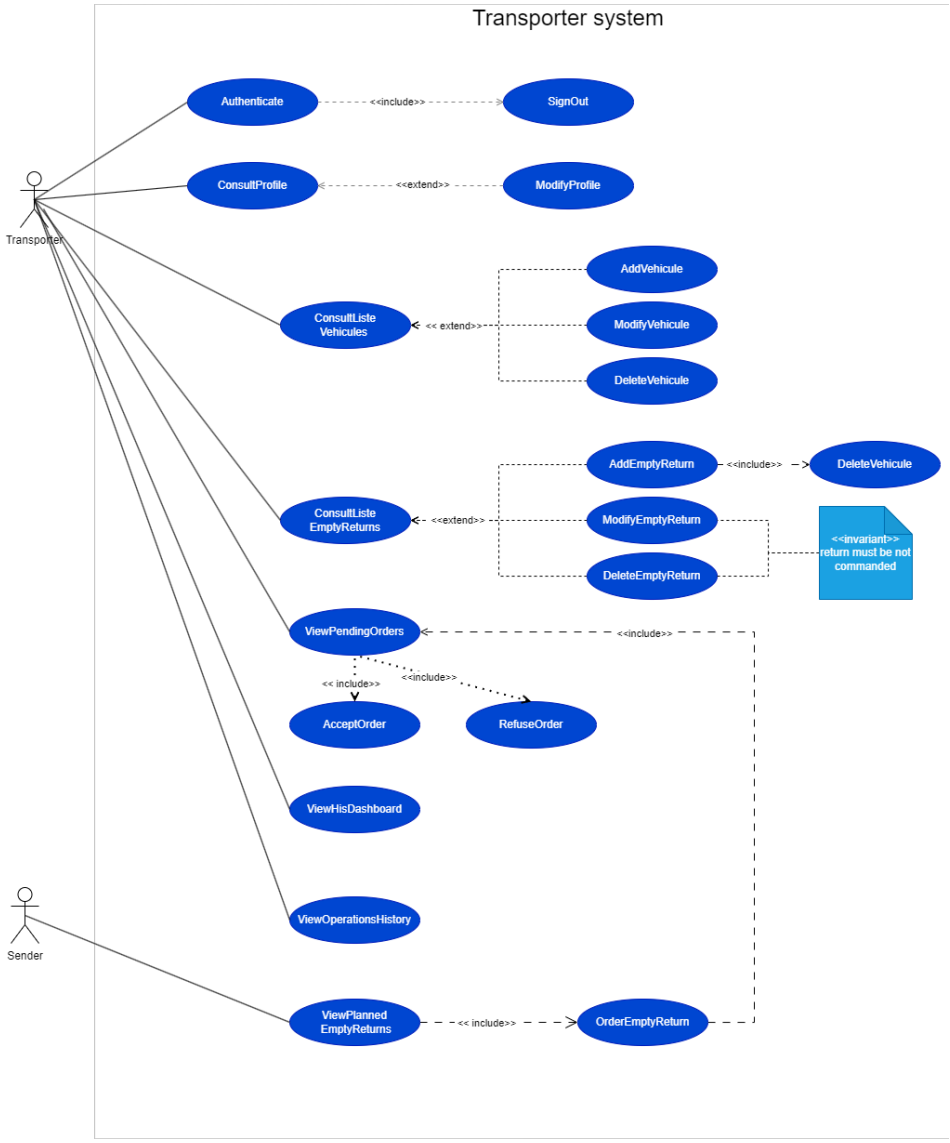



Figure 1.2: Transporter use case diagram

#### 1.5.6.2 Shipper use case diagram

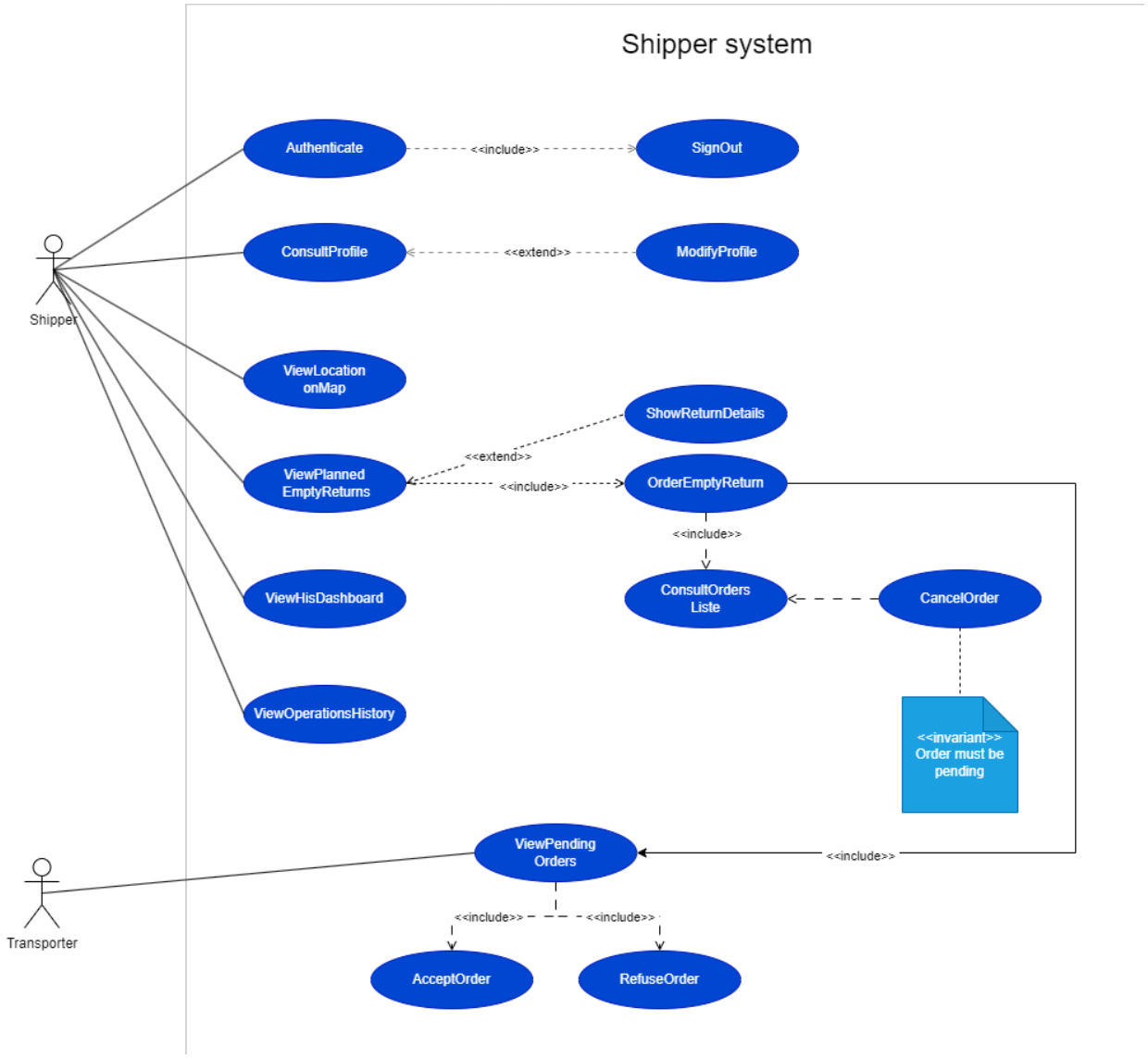



Figure 1.3: Shipper use case diagram

#### 1.5.6.3 Administrator use case diagram

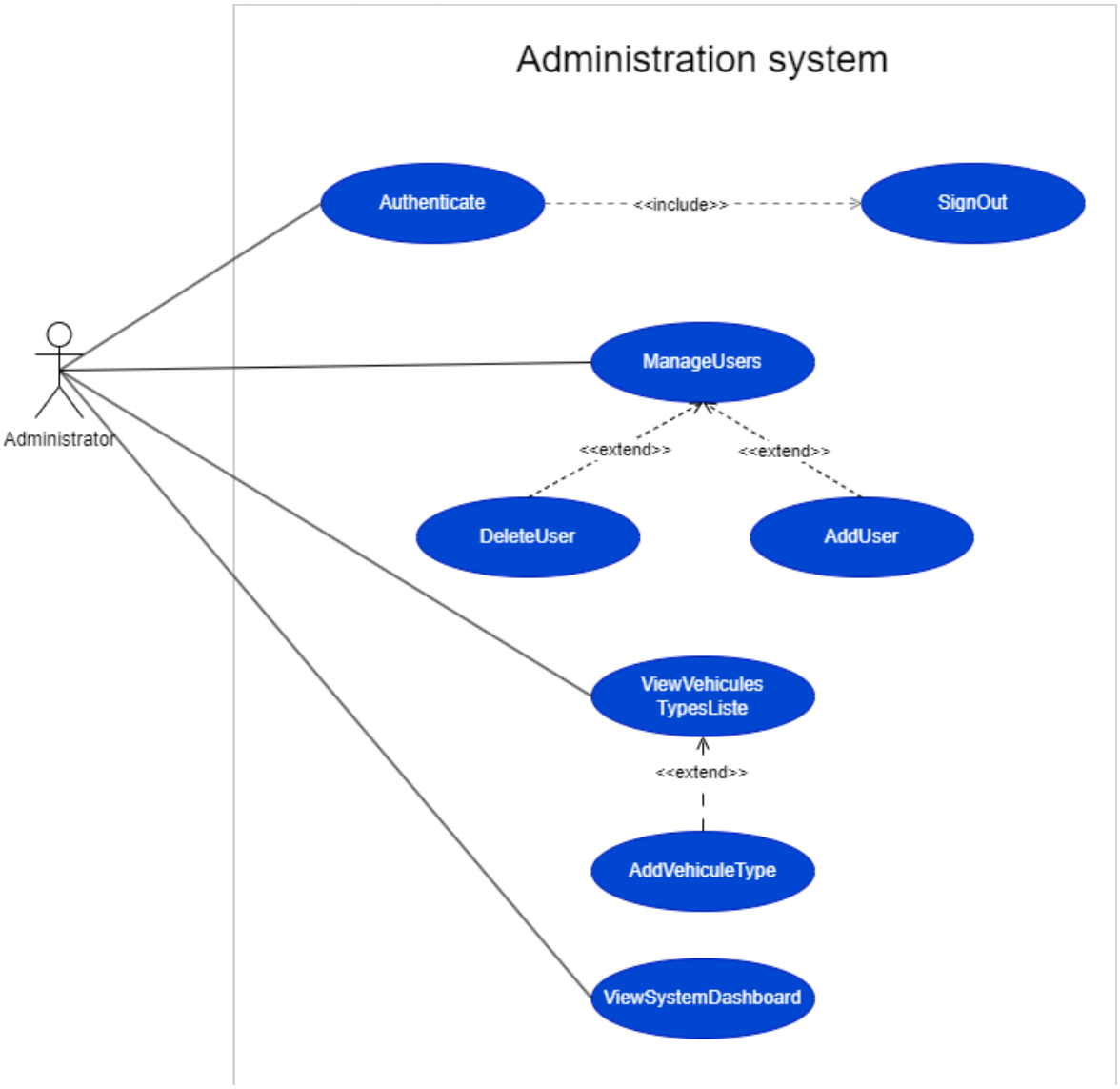



Figure 1.4: Administrator use case diagram

## 1.6 Objectives

This systems first Aim is to optimize and ameliorate the process of supply chain from various perspectives, to achieve that a number of objectives must be met , we cite the following :

- the system must Improve communication between the two actors, carrier and shipper
- the system must Rent a customizable private "user" space for each company.
- The system Must ensure the best return plans for transporters .
- The system should respond to future expansion in case of new clients.
- The solution must ensure access to data at all times and under all circumstances.
- The system must coordinate between various systems .
- The system must provide an effective mailing and commanding system.
- The system must take advantage of the empty return plans and maximize the number of commands achieved in the given time and vehicles access.
- The system must minimize the cost of transportation and take full use of vehicles .

## 1.7 Work Distribution

Since we still have too few information about the Next Parts of the system we prefer to leave this part for last, we envision a team work for each part of the ongoing Project.

# 2

# Objectives Definition according to Quality Attributes

## 2.1 Introduction

Having seen the description of our supply chain solution , we now move on to Our quality attributes,The domain of supply chain is a domain distinct for having many systems interconnecting with each other ,beginning from storage management,their planification,organization to the transportation management to command system and distribution so naturally their related systems would be similarly complex that is having many complex systems working together to create an optimized flow for the supply chain process to work on .we now cite the principal quality attributes that we will be focusing on ,this doesn't mean that our system will not provide any technique to assure the rest but the priority quality attributes are ones who will take the most of work to be assured.

## 2.2 Interoperability

Our most important Quality Attribute, since the system contains many subsystems interacting with each other we view interoperability to be the most crucial one .Now to illustrate interoperability we must go through it's dimensions.

### 2.2.1 The Importance of interoperability for the system

#### 2.2.1.1 Service Discovery

Service discovery is a mechanism or process used in distributed systems to locate and identify available services within a network.a possible scenario where this

would be crucial is the following :

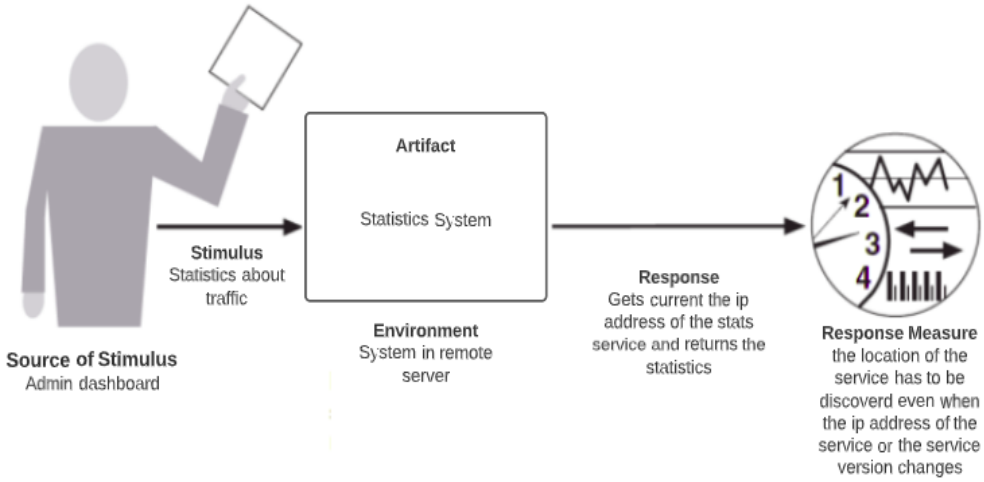



Figure 2.1: service discovery scenario

- This is a common scenario where let's say the admin wants to visualize statistics generated from the statistics server .to first access the statistics generated we must know their ip adress . and what if this ip address is unknown or it's ip address has changed . this makes the service undiscovered and rendering the system nonfunctional.
- if in the case where our services are inside docker containers. the ip addresses frequently change .meaning there's no way we could frequently and manually change services.

#### 2.2.1.2 Service Orchestration

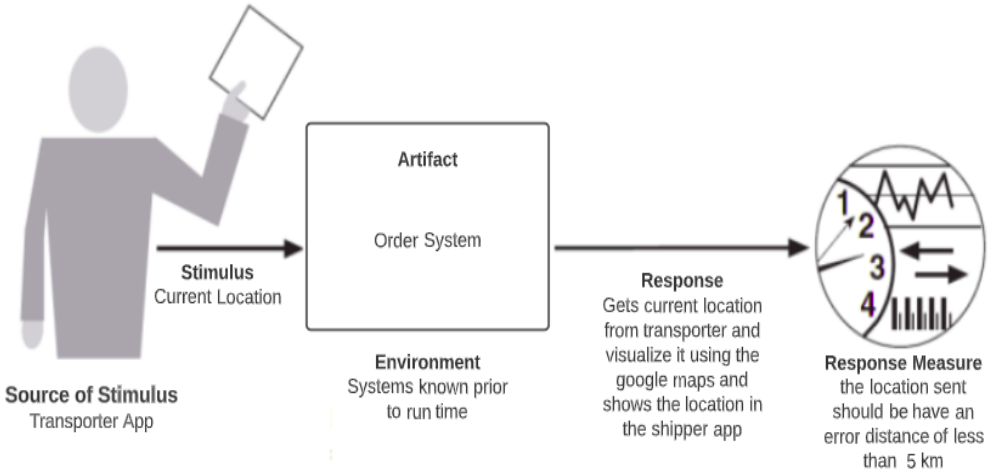



Figure 2.2: service orchestration scenario

- A transporter starts a trip and launches the location discovery option . a shipper wants to see location of the transporter so he access the option using the mobile app the data of the location is transmitted to from the mobile app of the transporter and with the maps api is visualized on the map .
- Another scenario is when a shipper wants to make an order .first it shows the list of available empty return trips from the freight management system . then once the order chooses it sends a notification to the chosen transporter and he chooses whether he accepts or refuses . once he accepts it an estimation of the time and distance are calculated by the map system . this scenario exposes the importance of orchestration between various systems .
- if the transporter refuses to accept the order he puts the reason. this is sent to the shipper and the admin , and the stats system so it adds it to the users profile .

The previously mentioned scenarios require a high level of orchestration where these services cooperate .this task becomes increasingly hard to maintain without some sort of a system dedicated to it . if we don't have one and tomorrow we don't have an orchestration mechanism we must reconfigure the interactions .

#### 2.2.1.3 Tailor Interface

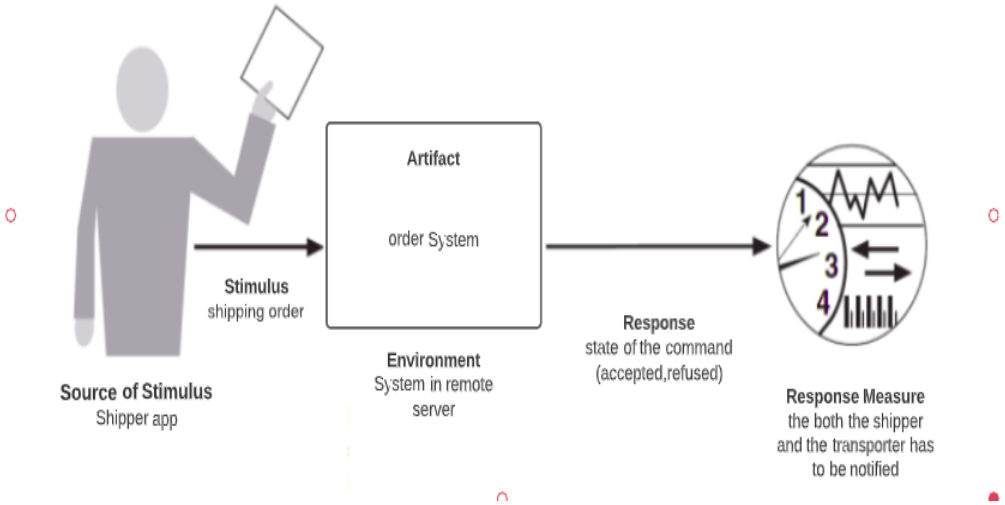



Figure 2.3: tailor interface scenario

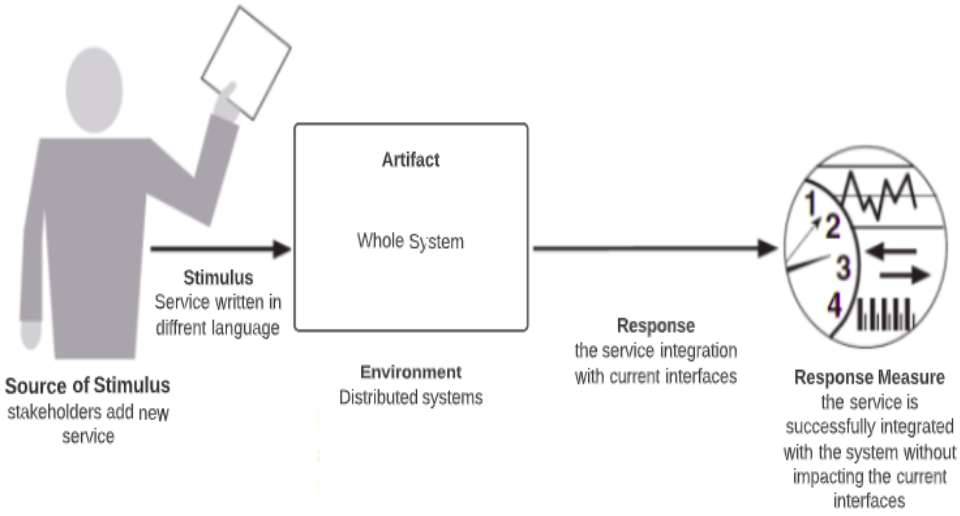



Figure 2.4: tailor interface scenario 2

- let's say a service is accessed by a consumer this consumer tries to apply malicious activities on the service or repeatedly sends requests to the same transporter we have to apply mechanisms that allow block or limit the consumers capabilities via a tailored interface that limits and controls the access to these services .
- a service written in js like command is trying to communicate with another service like statistics build with python that is written in java .both have

their own data handling methods . if there's no standard data formats this leads to inconsistency .

- a service is trying to access a service that has already been upgraded or changed version this could lead to system crash .

### 2.2.2 The Importance of performance for the system

#### 2.2.2.1 System Caching

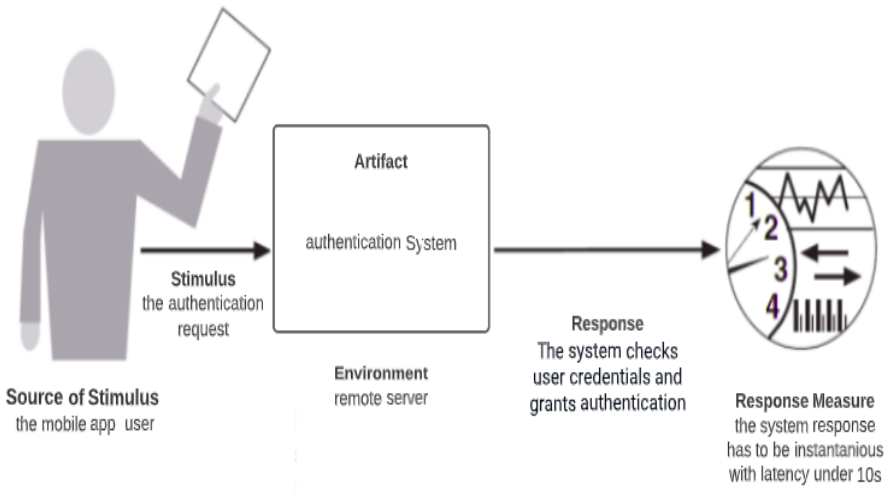



Figure 2.5: caching scenario

- when a user tries to authentify instead of accessing a remote database to compare the values we should find a way to store his user credentials in a manner that makes this operation as fast as possible.

- another possible scenario would be frequently accessing the data related to an order. when launching an order it is highly possible that the shipper will frequently access the order information .if this data is frequently accessed we shouldn't have to wait everytime for the request to reach the remote server as it makes the network heavier and adds more work on the db server.

#### 2.2.2.2 System Workload Balancer

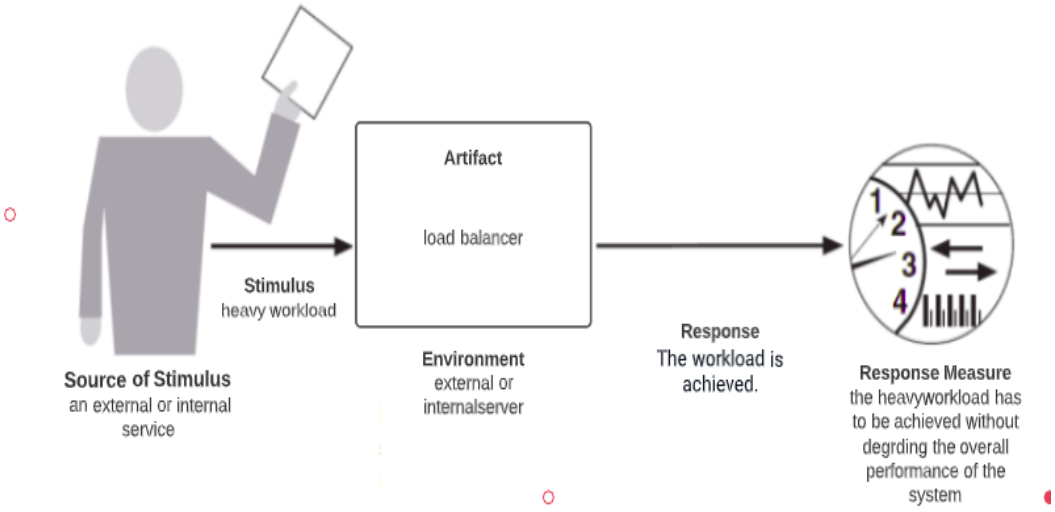



Figure 2.6: workload balancer scenario

at the end of the month we want to calculate the statistics for the entire year.the statistics server is having heavy load of information,if there are multiple instances of the system instead of having one instance take all the heavy load ,we should find a mechanism to distribute this workload to take the maximum advantage of our hardware

### 2.2.3 Scalability

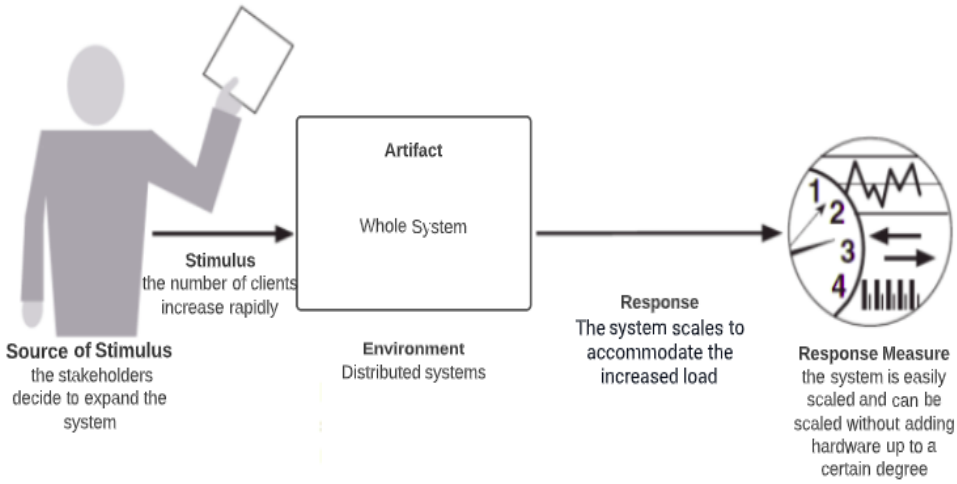



Figure 2.7: scalability scenario

The system at it's core is an inter-organizations effort but it is more likely to expand in the future reaching a national scale.if we want to add new service in the future or accommodate to higher consumption we will need to avoid as much as possible hardware expansion as it is costly ,this means that we should allow the system to expand without the need of adding physical ressources.

# 3

# Architectural Tactics Documentation

## 3.1 Introduction

After deriving the quality attributes viewed critical for our supply chain project we provide the tactics that we envision will ensure these quality attributes. The process of documenting these tactics will be through a checklist for each desired quality attribute , we will also examine the effect of each tactic on the rest of quality attributes.

## 3.2 Interoperability Tactics

The tactics concerning this quality attribute are devised into two main categories : service discovery as discussed in the previous part , there are a plethora of services both internally and externally to handle and interface management ,other important tactics would be data format standardization.

### 3.2.1 Service Discovery

In order to choose among these approaches we had to add another dimension into consideration which is the performance and this is why we chose service registry as it provides the best performance with a centralized directory of available services that can be queried quickly and efficiently. A service registry can also use load balancing algorithms to distribute requests to available services, improving performance and scalability.

This means that information about services will go be stored in a centralized registry and requests to use these services will go through a service registry software. the most common one would be Eureka.for example we want to access

the map service stored in Eureka registry Our system uses a variety of services both and external which creates the need for service discovery mechanisms.the service oriented nature of our system provides a mechanism to achieve this through the use of service registry. there are several approaches for this such as service registry which is centralized directory that maintains a list of all available services and their metadata,UDDI( Universal Description, Discovery, and Integration) which is a standard protocol for service registry and and dynamic discovery where services advertise themselves to the network using multicast or broadcast messages and finally Directory Services such as LDAP or active directory, can be used for service discovery .

In order to choose among these approaches we had to add another dimension into consideration which is the performance and this is why we chose service registry as it provides the best performance with a centralized directory of available services that can be queried quickly and efficiently. A service registry can also use load balancing algorithms to distribute requests to available services, improving performance and scalability.

This means that information about services will go be stored in a centralized registry and requests to use these services will go through a service registry software. the most common one would be Eureka.for example we want to access the map service stored in Eureka registry

```
 const Eureka = require('eureka-js-client').Eureka;
const L = require('leaflet');

const client = new Eureka({
  instance: {
    app: 'your-app-name',
    instanceId: 'your-instance-id',
    hostName: 'your-host-name',
    ipAddr: 'your-ip-address',
    port: {
      '$': 3000,
      '@enabled': 'true',
    },
    vipAddress: 'your-vip-address',
    dataCenterInfo: {
      '@class': 'com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo',
```

```
      name: 'MyOwn',
    },
  },
  eureka: {
    host: 'eureka-server-host',
    port: 8761,
    servicePath: '/eureka/apps/',
    preferIpAddress: true,
  },
});

client.logger.level('debug');

client.start((error) => {
  if (error) {
    console.log(error);
  } else {
    console.log('Eureka client started');
    const mapServiceUrl = client.getInstancesByAppId('map-service-name')[0]
    console.log(`Map Service URL: ${mapServiceUrl}`);
    // Use the map service URL to create a Leaflet map
    const map = L.map('map').setView([51.505, -0.09], 13);
    L.tileLayer(`${mapServiceUrl}/tiles/{z}/{x}/{y}.png`).addTo(map);
  }
});
```

this solution is suitable in a number of ways , as its :

- Allows services to be more autonomous by lowering coupling with the external services.
- Removes concerns about a service version as they automatically update.
- It's language agnostic,making it independant of the programming languages used.
- it's easier to monitor as it allows us to track available and none available services and services in use.

### 3.2.2 Orchestration

Since our system contains many services which rely upon each other to each certain tasks , orchestration is considered a key aspect in interoperability.if we take the following example :

A process begins by receiving a command from the command system which will then be processed in the freight management system which will use the map to find the best route if available and executes it . this process along side many others needs complex orchestration . for this we have several strategies to achieve this from applying an event driven architecture ,using a centralized orchestrator or using a distributed workflow engine. event driven architecture is the most complex to implement compared to the two and it has the risks of creating event loops while the centralized orchestrator is the easiest it has a downfall on performance as it will be treat all interactions in the distributed system.this is why we will go to the approach of a distributed workflow engine. A distributed workflow engine can help manage the flow of data and events between services especially if they are distributed.to implement this workflow engine we will

- use Zeebe as a tool for this.initially we would use Camunda but it's centralized nature meant a future obstacle for scalability ,as we dove deeper into the world of workflow engines we met Zeebe a relatively new workflow engine that is distributed meaning scalable and provides low latency.

- install distributed versions of it across servers that host the services that are interdependant

- implement the broker node or the central workflow engine instance that will be responsible for creating,monitoring the rest of nodes. (it is worth mentioning that the nodes will communicate with each other to coordinate , the broker is there essentially to monitor them and communication doesn't have to pass by him.

- define the workflow schema for each service depending on the task..moving back to the previous example the process of commanding an empty return trip would be like this :

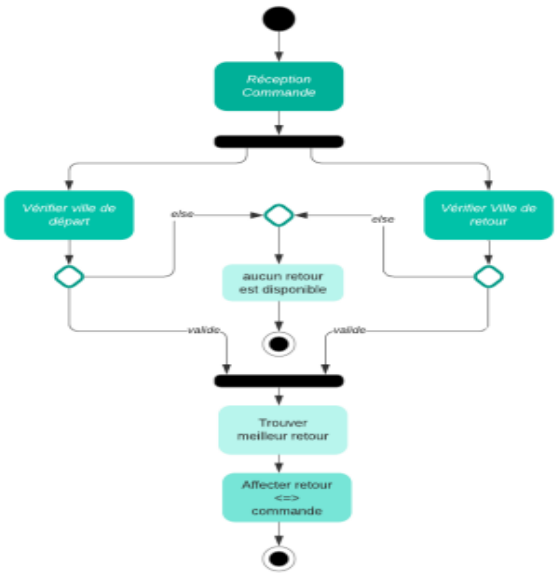



Figure 3.1: The process of launching an empty return command

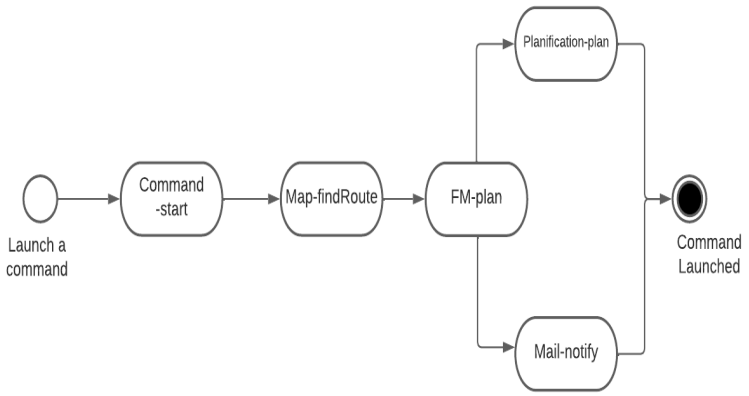



Figure 3.2: Workflow diagram for the tasks and their order in Zeebe

Each node in the diagram corresponds to a task and each task has it's own id , we could easily using in the NodeJs environment like this

```
const ZB = require('zeebe-node')

const zbc = new ZB.ZBClient('localhost:26500')
const zbWorker = zbc.createWorker('test-worker', 'FM-plan', handler)

function handler(job, complete) {
  // here: business logic that is executed with every job
  //.....
  // and let the workflow engine know we are done
  complete.success()
}
```

### 3.2.3 tailor interface

Though not so crucial to the interoperability , we view it as a useful tactic to reduce complexity of interacting with the service.having already established our service registry for service discovery , we add this tactic to further more facilitate the process of handling these services, for this we envision using an api gateway not only to facilitate using services but also discovering them as well as protocols translation.

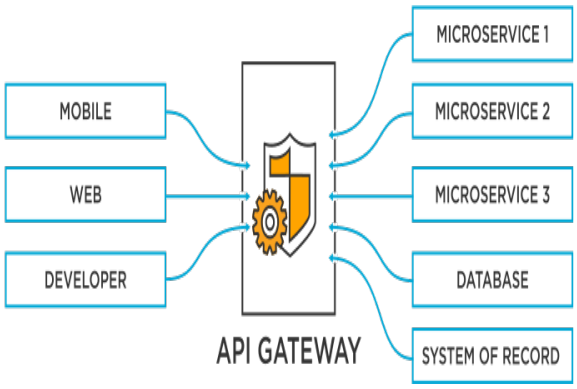



Figure 3.3: The use of api gateway

## 3.3 Interoperability Checklist

### 3.3.1 Responsibility Allocation

The system contains a number of services with some being internal like the users module and other external like the various APIs, to determine which of our systems will need to interoperate with the other we first have to assign responsibilities

- A "Logistics Integration Module" : responsible for handling interactions with the logistics system.
- A "Commands Integration Module" : responsible for handling interactions with the commands system.
- A "Planification Integration Module" : responsible for handling interactions with the planification system.
- A "Mailing Integration Module" : responsible for handling interactions with the mailing system.
- A "Maps Integration Module" : responsible for handling interactions with the maps system.
- A "Distance Matrix Integration Module" : responsible for handling interactions with the Distance Matrix system.(these three mentioned are external api services)
- A " Directions Integration Module" : responsible for handling interactions with the Directions system.
- A "Freight Management Module" : responsible for processing the shipments. This module will use the data received from the different Integration Modules and Maps Integration Module to manage the shipments and delivery routes. It will also use the Mailing Integration Module to send delivery notifications and updates to customers.

the request detection and handling will all be done through the orchestrator node which will have the appropriate response giving the task ordered the communication with unknown services will be done through the api gateway which in of itself can give permissions or block certain services the discovery of services will be managed by a centralized service registry

### 3.3.2 Coordination Module

For the coordination model we will discuss the degree in which our mechanisms respond to our quality attributes being the interoperability and performance and to a lesser extent scalability.

- *Service Discovery:* the approach is to use a centralized service registry , this allows to find and register the services and easily handle their occurring updates , the centralized approach allows for better performance as the synchronization of a distributed service registry is less performance efficient however this might have a negative effect on the scalability of the system as the number of services grow,however this effects negatively on the availability of the system as having one service register down would threaten the ability to find services in the system.

- *Orchestration:* for orchestration the approach being a distributed workflow engine distributed across the servers holding the services that need to be managed. this approach is scalable as adding a future interacting service would require adding a new node, it also performance efficient as it leverages streaming into the client and use binary communication protocol this is very efficient and performant.however this makes system modifiability harder as each node has to be configured separately.

- *Tailored Interfaces:* Monitoring services is hard to say the least especially as the system grows , using simple decorator won't be sufficient , the use of a more performant and scalable solution is a must , we previously discussed the use of an API gateway this is one of the most common approaches in such system, it's good for monitoring , it can control the behavior and interactions of unknown services, and for interoperability it provides a unified interface to handle API's meaning that the language implementation,data formats won't effect the interoperability ,it can help with scalability as it distribute requests to multiple backend services or servers, improving scalability and fault tolerance. This allows for horizontal scaling where additional instances of a subsystem can be added to handle increasing traffic,on the other hand it might face a bit of performance issues with network latency or high traffic due to it's centralized nature, afcourse we cannot say that our solution will be 100 % efficient but we will handle this further more in the performance tactics (load balancing ) .

### 3.3.3 Data Model

The main data formats that would be used for the system is JSON , the majority of the system is build on JS technologies , the database is a document oriented database providing JSON like documents,which will be exchanged mostly using the http protocol,websockets are also a solution for real time data concerning the tracking system. In some instances data will take the form of XML and for this there are relatively easy ways to guarantee data consistency such as using a converter class for example when defining a new workflow in the zeebe workflow engine , we have built in XML support but we could send the format using JSON by applying a simple BPMXML diagram within the JSON like this

```
 // Define the workflow to deploy
const workflow = {
  bpmnProcessId: 'processId',
  name: 'Workflow Name',
  // Set the version of the workflow to deploy
  version: 1,
  // Define the BPMN diagram as a JSON object
  bpmnXml: `
    <?xml version="1.0" encoding="UTF-8"?>
    <bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MOI
      <bpmn:process id="processId" name="Process Name">
        <bpmn:startEvent id="start" />
        <bpmn:serviceTask id="task" name="Task Name" />
        <bpmn:endEvent id="end" />
        <bpmn:sequenceFlow sourceRef="start" targetRef="task" />
        <bpmn:sequenceFlow sourceRef="task" targetRef="end" />
      </bpmn:process>
    </bpmn:definitions>
  `,
};
```

with that being said we show the different operations concerning data abstractions in the system as follows : as for the major data abstractions we state the following :

- User

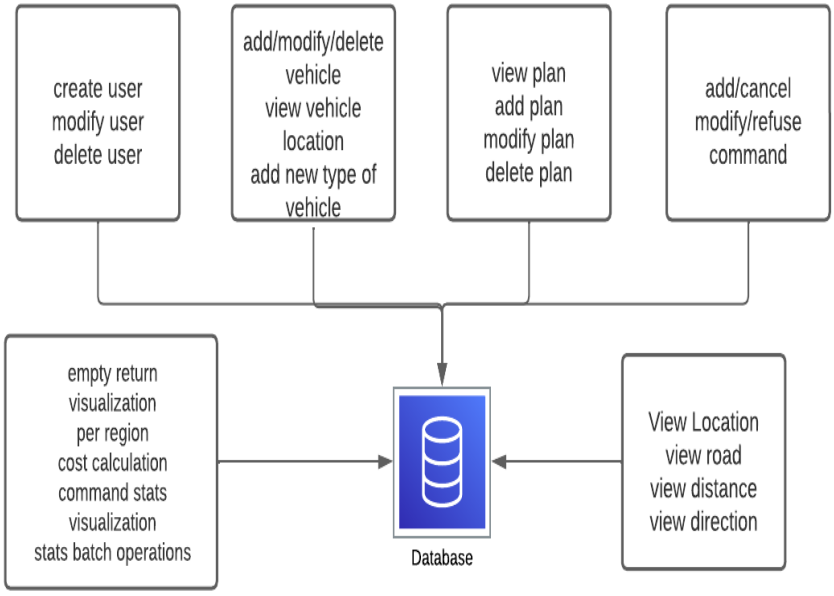



Figure 3.4: different data operations in the system

```
{
  "name": "...",
  "user_type": "...",
  "id": "...",
  "vehicles": [...],
  "email": "...",
  "password": "...",
  "company": "..."
}
```

- Vehicle Location

```
{
  "vehicle_id": ...,
  "company" :...,
  "timestamp": "2022-05-01T10:30:00Z",
  "latitude": ...,
  "longitude":...,
  "speed": ...,
```

```
  "heading": ...,
}
```

- empty return road Plan

```
{
{
  "name": "...",
  "description": "...",
  "waypoints": [
    {
      "name": "...",
      "latitude": ...,
      "longitude": ...
    },.....],
  "distance": 30.2,
  "duration": "00:35:00"
}
```

- Zeebe Workflow

```
{
  bpmnProcessId: '..',
  name: '..',
  version: ..,
  bpmnXml: `..`,
};
```

### 3.3.4 Management of Resources

as for the management of ressources we tried to take two factors into consideration scalability and performance , as for the ressource identification and management,the system is distributed across multiple servers.each server will host a service or a group of closely bonded services .

- the database will be hosted in a centralized manner in a db server .
- the service registry, API gateway and work flow engine broker will be all hosted in the same server , the API gateway will provide control access to both internal services stored in the registry and external access . the service registry hill be have one instance as it's central. the work flow manager will monitore,create and update the nodes .each node will communicate with each other in a peer to peer manner without any central control. the broker is there just for monitoring.
- the mailing service will be hosted in the mail server and will have a work flow engine node for integration with the freight manager.
- the freight management server will host the closely related components of command,planification and freight management and will cooperate with the maps and mailing services.the map service will be an external API.
- the statistics system will take care of the heavy calculations concerning the massive amounts of data.

### 3.3.5 Mapping across architectural elements

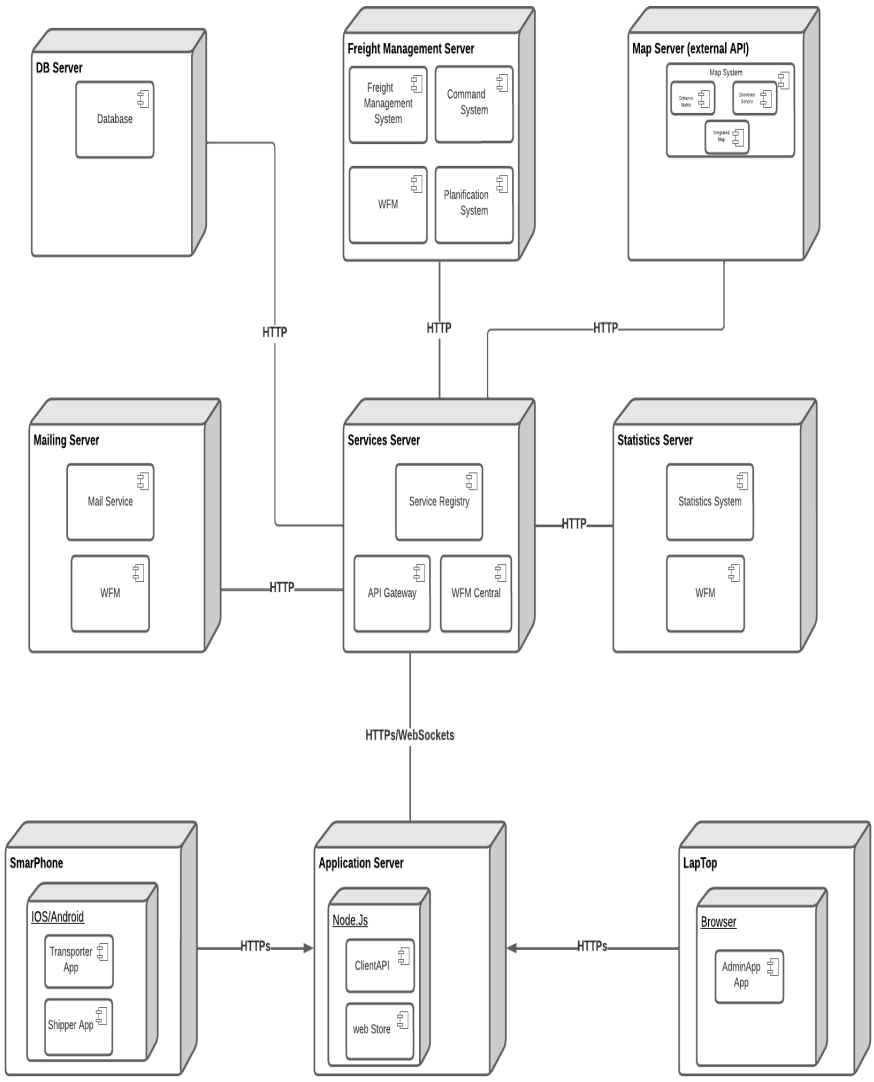



Figure 3.5: Component/deployment diagram of the services and the tactics ensuring them

### 3.3.6 Binding Time Decisions

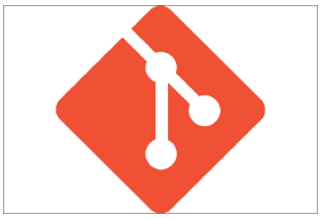
The binding time decisions of our system are done through the combined effort of the service registry,api gateway and the workflow engines. the decision of when to interact with a service is done mainly through the use of the workflow engine as it has predefined schemes for achieving tasks , meaning that if at a given state a service a has to interact with a service b then the orchestrator decides that. now say that an interaction with a service is denied for any reason such as it's not available or too busy , this will be handled through the API gateway as it's a powerful approach towards monitoring and controlling service interactions,the API gateway will determine who will get to interact with what service. finally for service discovery it is a combination between the service registry which keeps track of the ports and addresses even if they change and the API gateway for reaching external services . so to sum up the orchestrator responds to the question of (when does service a interacts with service b),the API gateway answers to the question(can this service interact with this service?) and the service registry responds to (where is this service located?).

### 3.3.7 Choice of technology

the choice of software for binding time decisions depends on the specific requirements of the program , it includes the compiler ,interpreter,preprocessor...

#### 3.3.7.1 git

A version control tool that will allow for modifications in case of obsolescence and incompatibility. this is used for changes in the explicit versioning . using dynamic versioning would reduce the the use of this tool for modification for example using getInfo or hasCapability functions



#### 3.3.7.2 Travis CI

An automation tool for continuous integration,it can help identify and resolve issues that may arise during the different stages of binding time. By frequently integrating code changes, developers can catch errors that may arise during compile time, load time, or runtime, and fix them before they become larger

problems. This can help ensure that the software is stable and reliable throughout its development life cycle.

#### 3.3.7.3 Heroku


An excellent automation tool for continuous deployment,automating the deployment process, ensures that the latest version of the software is always running in production, which can improve the speed and reliability of the software. this allows for faster release cycles and can improve the overall quality of the software by catching and fixing issues earlier in the development process.

#### 3.3.7.4 Node Package Manager


NPM (Node Package Manager) is a package manager for the JavaScript programming language. It is used to manage dependencies for NodeJs projects, including libraries and modules that can be used in a project,one binding time decision that must be made when using NPM is when to install dependencies. Dependencies can be installed during development time or runtime. Another binding time decision when using NPM is which version of a package to install. NPM uses semantic versioning to manage package versions, where each version number consists of three parts: major, minor, and patch.

#### 3.3.7.5 Eureka Registry

Eureka is a service registry created by Netflix, used for building resilient and scalable microservice-based applications. A service registry is a centralized directory of services, where each service can register itself and its location, making it easy for other services to find and connect to it. Eureka allows services to register and deregister themselves dynamically, making it easy to add or remove instances of a service as demand changes. It also provides a REST API for querying the registry and discovering available services, enabling client-side load balancing and failover.

#### 3.3.7.6 Zeebe Work flow engine


Zeebe is a modern, cloud-native workflow engine designed to orchestrate workflows at scale. It is developed and maintained by Camunda, an open-source software company.

Zeebe is built on a distributed architecture, allowing it to handle high volumes of workflow instances and scale horizontally as needed. Workflows are modeled using the BPMN 2.0 standard, which provides a visual notation for modeling business processes. Zeebe workflows can be executed on any cloud or on-premise environment, and can integrate with a variety of other tools and systems.

#### 3.3.7.7 Kong gateway

Kong: An open-source API gateway that can run on-premise or in the cloud, providing features such as traffic routing, rate limiting, and authentication.


## 3.4 Side effects of interoperability tactics

Afcourse for large systems not all quality attributes can be ensured to the highest quality, and basing our solutions on specific quality attributes would mean negatively impacting the others, we state the possible negative effect of these solutions on the rest of quality attributes

### 3.4.1 Availability

- having a centralized service registry,API gateway means less fault tolerance as having the server holding them down would mean the loss of the ability to locate services .
- the use of tailor interfaces in the form of the API gateway might block certain services rendering them unavailable , this would ultimately cause the system to stop and therefore be unavailable when trying to achieve tasks that require services cooperation.

It's worth mentioning that having a centralized database would also harm the availability of the system as its significantly less fault tolerance.

### 3.4.2 Security

- though the use of the API gateway should drop the risk of interacting with a dangerous external service , it remains a possible threat.
- the orchestration between services which is of the form of apis creates vulnerable points as api's are vulnerable to numerous attacks such as injection, spoofing, and man-in-the-middle attacks.
- relies on services and external apis makes it crucial to regularly update and patch these dependencies to avoid vulnerabilities that can be exploited by attackers overall another case for augmenting the security risk.
- the distributed nature and multi service nature of the project means more surface for attacks and more effort into securing the system .
- the orchestration implemented means more data exchange and more data exposure this means a higher risk of security data attacks .

### 3.4.3 modifiablity

the decentralized nature of the project means that adding a new service in the future would be much harder and complex as it has to be discoverd and stored in the service registry,integrated and orchestrated with the WFM and be controlled and configured by the api gateway, this makes the modifiablity of the system a much more complex task, on the otherhand modifying the behaviore of a service means that the interactions with it would be modified, the permission to it would also be modified. overall we estimate that this is the most damaged quality attribute

### 3.4.4 Summary

though the tactics proposed for interoperability have various negative sides on the other quality attributes, they only add to the risk of them being violated. we can't have a system that has 0 security and availability.However modifiability remains the only quality attribute that would be seriously impacted by our tactics to a noticable desgree

## 3.5 Performance Tactics

### 3.5.1 Manage sampling rate

Managing sampling rate is a performance tactic that involves controlling the rate at which data is collected and processed. This tactic can improve the performance of the system by reducing the amount of unnecessary data that is collected and processed, while ensuring that important data is captured in a timely manner. For our system it is important to use Manage Sampling Rate. As the system deals with a large volume of data. To process this data in a timely and efficient manner. This involves selecting a suitable interval for sampling the data at a rate that is appropriate for the system's processing capabilities. To avoid overloading the system with more data than it can handle, leading to slow performance or even system crashes.

#### 3.5.1.1 Mechanisms

- **Adaptive Sampling:** This strategy adjusts the sampling rate based on the current system load and resource availability. For example, if the system is experiencing heavy traffic, the sampling rate can be decreased to reduce the load on the system.
- **Event-Triggered Sampling:** This strategy samples data only when certain events occur, such as a change in system state or a threshold being reached. For example, in a temperature monitoring system, data may only be sampled when the temperature exceeds a certain threshold.
- **Burst sampling :** it's a technique in which a high-frequency data stream is sampled at a lower rate for a specified period, and then the sampling rate is increased again. This technique is useful when there are short bursts of activity in the data stream that need to be captured, while the overall data stream can be sampled at a lower rate to conserve resources.
- **Stratified sampling :** it's a technique in which a population is divided into subgroups or strata based on certain characteristics, and then samples are taken from each stratum to ensure that the sample is representative of the population as a whole. This technique is useful when the population being sampled has significant variations or differences, and the sampling

needs to capture those differences to provide an accurate representation of the population.

- **Periodic sampling :** where data is sampled at regular intervals.
- **Threshold-based sampling :** setting thresholds based on certain performance metrics. When the metric exceeds a certain threshold, the system increases the sampling rate to capture more detailed information. This mechanism is useful for systems where performance metrics vary widely over time and space, and where it is difficult or impractical to use a fixed sampling rate.

#### 3.5.1.2 The data concerned with sampling rate

- **Location Data**The system needs to collect and process real-time data from sensors and GPS devices to track the location of the freight
- **Traffic data** The system needs to collect and process traffic data to optimize the delivery routes and schedules.
- **Freight data** The system needs to collect and process data about the freight, such as weight, dimensions, and value, to optimize the supply chain and ensure that the freight is delivered on time and in good condition.
- **Customer data** The system needs to collect and process customer data, such as delivery addresses and preferences, to optimize the delivery routes and schedules and ensure customer satisfaction.

### 3.5.2 Introduce concurrency

To improve system performance in the Freight Management System for Supply Chain Optimization architecture we can apply Introducing concurrency to :

- improve the system's ability to handle heavy loads or time-critical events
- ensure that the data is processed quickly and accurately.
- enhance resource utilization in the system.reducing blocked time, which is the time a request has to wait before being processed.
- the processing of large amounts of data and requests in parallel using different threads or streams of events to improve the overall system throughput.

- achieve specific performance goals, such as maximizing fairness or throughput using scheduling policies

### 3.5.3 Mechanisms :

- **Parallel processing:** The system can process multiple tasks simultaneously, such as tracking shipments and scheduling transportation... This can reduce the overall processing time and increase system throughput.
- **Multi-threading:** The system can use multiple threads to perform different tasks concurrently, such as processing customer orders and updating inventory. This can improve system responsiveness and reduce the time it takes to complete tasks.
- **Distributed processing:** The system can distribute tasks across multiple servers, allowing for more efficient use of resources and improved fault tolerance.
- **event-based concurrency:** involves processing different streams of events on different threads to improve system performance.such as freight pickup requests, delivery requests(this is handled in the interoperability as well)
- **scheduling policies:** Once concurrency has been introduced, scheduling policies can be used to achieve the desired goals. Different scheduling policies can be used to maximize fairness, throughput,Response time,Quality of service...

### 3.5.4 Load balancing

In our system a lot of queries arrive in a short period of time which might force a server or a resource to become overwhelmed by the workload. This may lead to system failure or significant performance degradation , or even it may lead to state "system out of service" . In that case the queries arriving must be distributed over the resources and suitable servers to make a balance over all those components and avoid overcharging one of them. this is a shared concern between interoperability and performance as deciding the responding service ultimately means deciding the responding server due to services being

dedicated to specific servers, the tactics to ensure this has already been specified in the tailored interface tactic

### 3.5.5 Maintaining multiple copies of computations and data

One of the common tactics used to reduce latency and improve the time of execution is cashing , by storing the frequently searched queries and computations in limited memory to retrieve them easily .

#### 3.5.5.1 Mechanisms

Two mécanismes are suitable to our project which are :

- **In-memory caching:** involves storing frequently accessed data and computations on a networked cache server , those data and computations are filtered first in order to be the most searched and requested data by large number of users , the frequently accessed data by a single user are cached on another level . This mechanism can be implemented using a dedicated caching software or service such as Redis, Memcached, or Hazelcast .

  - Installation and configuration of the caching software .
  - Identifying the data to be cached .
  - Defining the cache expiration policy .
  - Identifying the rights of read and write operations .

- **Client-side caching:** which consists of caching the data in the web browser or mobile app to be loaded more quickly and reduce the load on the server. Three different types of Client -side caching are considered :

  - Browser caching : is a default mechanism that conserves the files and the assets of a web page to be reloaded later .
  - Local storage : HTML5 added recently a new API called "local storage" that allows caching a small amount of data such as user preferences or authentication tokens in the form of pairs (key-value).

- Service workers : Are JS background functionalities that can intercept network requests made by the application and caches frequently accessed resources . The first mechanism is used for the frequently accessed data by one user and is implemented in his local machine , on the other hand the second solution is used for the frequently accessed data by all the users of the system .

### 3.5.6 Increase Resources

Finally , as a final solution if none of the tactics above didn't fully satisfy the goal of improving the performance of the system , increasing the amount of resources available (CPU , memory , network bandwidth, and storage capacity)

#### 3.5.6.1 Mechanisms

Two main implementation of this mechanism are considered :

- **Upgrading hardware or adding servers:** Upgrading the hardware is one straightforward way to increase resources by adding powerful CPUs and replacing hard drives with a faster SSD. Another way is to add new servers to the system and setting up load balancing to distribute the traffic .which is already established given the distributed nature of the project .afcourse this solution is costly but given the scope of the project and it's cooperative stakeholders the cost isn't that much of an issue.
- **Virtualization:** This mechanism is used to avoid adding more complex components and to reduce the number of resources in the system (many virtual machines are used on the same physical machine) .we envision using it in the form of containers on the freight management server given it hosts several closely related services such as the command and planification systems.

This tactic is the best and most costly tactic among all those mentioned. That's why we listed it at the end as a final solution in case of a huge usage of our system in the future .

### 3.5.7 Prioritize Events:

The freight management system contains events with different degree of priority , so we can define a priority scheme that rank events according to their :

- **Cruciality :** Assigning priority based on the criticality of the event is very important . Critical events should be given the highest priority, while less critical events, such as user requests for non-critical functionality, can be given a lower priority. For example : adding vehicles and their technical information is more important than visualizing the list of historic trips .
- **User impact:** Prioritizing events based on their impact on the user experience can help ensure that the system is providing a high-quality experience for users.Since one of our objectives is to make sure that the UI/UX must be simple and comfortable for the users we should pay attention to events that have a big time of use by users in order to give them more priority
- **Time sensitivity:** in our system we should prioritize events based on their time sensitivity , this can help to ensure that the system is meeting time-critical requirements. for example events with minimal deadlines such as an empty return plan and its different commands ..........
- **Frequency:** In order to ensure that the system is efficiently handling the most frequently occurring events, High-frequency events must be given the highest priority such as adding empty return trips by the transporter

### 3.5.8 Increase resource efficiency

we already established the use of Caching which involves storing frequently accessed data in memory or on disk to reduce the need to access the data from the original source. This can reduce resource usage and improve system responsiveness on the other hand Resource sharing which allows multiple applications to share resources, such as databases in order to reduce resource usage and increase efficiency, has already been established through the use of a shared service registry and database.

#### 3.5.8.1 Mechanisms

a single mecanism not tackled yet to increase the resources efficiency is : **Resource monitoring:** Resource monitoring involves tracking resource usage, such as memory, CPU, or network usage, to identify inefficiencies and optimize resource allocation.

### 3.5.9 Schedule Ressources

A scheduling policy conceptually has two parts: a priority assignment and dispatching. Competing criteria for scheduling include optimal resource usage, request importance, minimizing the number of resources used, minimizing latency, maximizing throughput, preventing starvation to ensure fairness, and so forth. A high-priority event stream can be dispatched only if the resource to which it is being assigned is available. Sometimes this depends on pre-empting the current user of the resource.

#### 3.5.9.1 Mechanisms

**Fixed-priority scheduling :** Fixed-priority scheduling assigns each source of resource requests a particular priority and assigns the resources in that priority order. This strategy ensures better service for higher-priority requests but admits the possibility of a low-priority, but important, request taking an arbitrarily long time to be serviced because it is stuck behind a series of higher-priority requests. The prioritization strategy is : - semantic importance. Each stream is assigned a priority statically according to some domain characteristic of the task that generates it.

## 3.6 Performance Checklist

### 3.6.1 Allocation Of Responsibilities

- **Server Access Management:** This Responsibility includes all the function of the system that are sending a simple queries to the server such as (Authentication , adding modifying or deleting a vehicle , adding and modifying an empty return ,... )according to their priority , this type of queries have to be balanced and distributed between servers using the tactic mentioned previously including the two layers . Another tactic can be used

as well , which is maintaining multiple copies of data in the client-side to preserve the data recently used such as authentication by generating tokens in the app mobile or client's browser to save the credentials of the user .

- **Road estimation and time calculation:** Our application should have an efficient module to choose the best road to take by the transporter while having an empty return , and so to calculate the total distance and time spent in the trip , while having a huge amount of users this may take a long time for some users . That's why caching tactic seems one of the best solution to minimize the time execution by saving the frequently accessed data and computations to avoid recalculate the same information over and over , this tactic may reduce the response time by 50% , however adding new resources or upgrading the existing resources will accelerate the calculation part of the system while doing other tasks .Another solution is sharing resources between different modules of the app in order to make the calculation process more efficient and optimal .

- **Statistics generation:** All the users of the system can consult the details of their planification and their vehicles as well as the information and the planification of other users so they can ask for a shipment command , in addition they can visualize their dashboard . On the other side the administrator can see the list of users , vehicles and the statistics related to vehicles and usage. All those statistics are overwhelming the system in the normal process , so balancing the workload is necessary by distributing them over the nearest server in the first place to reduce the time response. In case of surcharging this last , looking for the least connections server is the best solution . Cache tactic is used as well by its both form : client-side and in memory , and the most important tactic is introducing concurrency to boost the computation part .

- **Analytics and Reporting:** This responsibility involves generating reports and analyzing data to optimize the supply chain. To manage the processing requirements and ensure timely response, a mechanism like stratified sampling can be implemented to focus on processing data that is most relevant to the current analysis or report.

- **Freight Tracking:** To manage the processing requirements and ensure timely response, a burst sampling can be implemented to handle sudden

spikes in the volume of data.

### 3.6.2 Coordination model:

The elements of the system that must coordinate with each other are : Freight management system , Planification system , Empty return plan management , Distance calculation elements , Geographic location information , Directions system , Statistic generation system . These elements support event prioritization and can be prioritized according to aspects mentioned in the tactic description ,according to this prioritization we can schedule ressources using fixed-priority scheduling . These elements have different degrees of occurrence,such as geographic location are synchronized in periodic moments(for example each 10 milliseconds). Communication is essential to coordinate the activities of different elements. The appropriate mechanism used in this situation is a synchronous mechanism in order to guarantee the transfer of the latest changed data between the different elements. Additionally,in order for the resource to coordinate when using the introduce concurrency tactic we can implement the following mechanisms :

- **Message Passing:** In a distributed processing environment, message passing can be used between interconnected elements of the system. This can be achieved by sending messages between different servers or between different threads of execution.
- **Remote Procedure Call (RPC):** RPC can be used to invoke procedures or functions on remote servers. This mechanism can be used to coordinate the processing of tasks between interconnected elements of the system. For example, the routing optimization responsibility can invoke a remote procedure on a server that is responsible for calculating the optimal route for each shipment.
- **Synchronization Mechanisms:** Synchronization mechanisms such as semaphores, mutexes, and condition variables can be used to coordinate the processing of tasks between different threads of execution.
- **Publish/Subscribe:** is a messaging pattern where senders of messages, called publishers that categorize published messages into topics. This mechanism can be used to coordinate the processing of tasks between interconnected elements of the system by allowing them to subscribe to relevant topics. For example, the freight tracking responsibility can publish

messages related to the status and location of shipments, which can be subscribed to by other elements of the system such as the order processing responsibility.

Other orchestration mechanisms are already specified in the interoperability part .

### 3.6.3 Data model:

The portions of data that are heavily loaded are those which contain a list of users and vehicles for the administrator user , which may take a long time to be fully loaded and may be reloaded again in a short period of time . For those data preserving a copy in client-side may reduce the time in case of re-executing the same query multiple time , another mechanism partitioning data may reduce the size And therefore reduce time of transmission , a first portion of the list will be sent and showed , after the second one if the users press more and so on , in case of pre-sign previous the list is stored in the cache . For the portion of data that includes processing big data such as generating statistics (a big amount of data must be treated to get accurate results ) , increasing resource efficiency or adding new resources will reduce the bottlenecks by reducing the time of generating statistics .

### 3.6.4 Mapping among architectural elements

introducing concurrency can improve the mapping among architectural elements by using parallel processing,Multi-threading can also be used to perform different tasks concurrently, This can improve system responsiveness and reduce the time it takes to complete tasks.As well as using Distributed processing to reduce the likelihood of a single point of failure.

### 3.6.5 Resource Management

In This system components that are responsible for calculation (server) and big data manipulation are the one exposed to be overwhelmed , that's why balancing the workload as identified previously is very important and introducing concurrency may reduce the bottlenecks. Parallel processing, multi-threading, or distributed processing can help optimize the use of network resources and improve system performance by handling more requests faster .

### 3.6.6 Binding Time :

There is a need for the tactic of bound execution time in the binding time,to ensure that certain components of the system are executed within a specified time frame to meet the performance requirements of the system. This can be achieved through techniques such as time-slicing, iteration capping, and resource allocation.For example:by implementing the tactic of bound execution time in the routing optimization responsibility we can ensure that the calculation is bounded within an acceptable time frame to meet the performance requirements of the system and ensure that the system remains responsive to user requests.

### 3.6.7 Choice of technology :

For the technologies which are used : javascript and HTML5 for web applications are very important for maintaining multiple copies of data client-side by the API and service workers mentioned in the tactic part . Other technologies are the following :

#### 3.6.7.1 Prometheus

Prometheus is an open-source monitoring system that allows you to collect metrics from a variety of sources, including applications, servers, and networks. It has a flexible query language and supports alerting and visualization. Prometheus can be integrated with various systems and is particularly useful mutli system architectures.

#### 3.6.7.2 Grafana

Grafana is a popular open-source dashboarding platform
that allows you to visualize and analyze data from various sources, including Prometheus. It supports a wide range of data sources, such as databases, messaging systems, and logs, and provides various plugins and extensions. Grafana can help you monitor system performance and detect anomalies.

#### 3.6.7.3 Threading libraries

These are libraries that provide APIs for creating and managing threads in a program. Examples pthreads library for C/C++

#### 3.6.7.4 Message queuing

These are systems that allow components to communicate with each other through a message queue.this is already established in the interoperability with zeebe

#### 3.6.7.5 Hadoop

These are frameworks that enable the distribution of computing tasks across multiple machines or nodes in a network.we will be using Hadoop for it's ability to process large data, the usage of map reduce, the integration with MongoDB,real time data processing with spark,support for batch processing.

#### 3.6.7.6 Containerization technologies:

These are technologies that allow applications to be packaged into containers, which can be deployed and run in a consistent and isolated manner across different environments. Examples include Docker and Kubernetes.

#### 3.6.7.7 No SQL Database

For the sake of scalability and calculations. no-SQL databases exceed SQL databases. the amount of data that will be stored and the integration with batch processing system like Hadoop requires a no-SQL database.MongoDB is one of the most famous and easy to use databases.it also provides a caching mechanism

## 3.7 Side effects of performance tactics

### 3.7.1 modifiability

Implementing caching, increasing resources and load balancing can increase the complexity of a system, as additional infrastructure and software components may need to be introduced. This can lead to higher maintenance and development costs, as well as potential performance overhead due to increased communication and synchronization . This may have a negative impact on the modifiability attribute which consists of reducing the complexity .

### 3.7.2 Security

Increasing the number of resources or introducing caching may improve performance but could also increase the risk of security vulnerabilities such as Denial of Service (DoS) attacks or unauthorized access to cached data. Similarly, implementing load balancing may require the use of third-party tools or services, which could introduce additional security risks. It is important to consider these potential trade-offs and take appropriate steps to mitigate any security risks.

## 3.8 Scalability as an additional focus

Although not a major quality attribute , it is a concern for our system.as the stakeholders envision the system growing out to a national scale the system has to reply to the needs,naturally each of chosen tactics for the performance and interoperability had a hidden concern that is scalability . some of the approaches towards making the system scalable were :

- addition of ressources as it made the system distributed and scalable as the system is brocken down to services or subsystems
- implementing Load balancing which involves distributing the workload evenly across multiple nodes. This can be achieved using a load balancer, which can route traffic to the appropriate node based on various criteria, such as CPU usage or network traffic..
- Data caching which involves storing frequently accessed data in memory, so it can be retrieved quickly. This can reduce the load on the system and improve performance. There are several caching solutions available, such as Redis or Memcached.

Although many design decisions were taken with respect the performance and interoperability and scalability .sometimes these presented conflicts . the choice of a centralized database which although respects performance as it gets rid of synchronization issues , it does leave scalability damaged as the centralized database and service registry would limit the scalability

# 4

# Architectural Views and Styles Documentation

## 4.1 Architectural Views

An architectural view is a representation of a system that captures its important characteristics and properties, from a particular perspective, for a specific purpose.we will take a look through various perspectives to further understand the system.

### 4.1.1 Decomposition View

The decomposition view is used to describe how a system is broken down into smaller, more manageable parts or components. This view helps to understand the organization and structure of the system . The system is divided into smaller components that are responsible for specific tasks or functions. Each component may be responsible for handling a specific business process or technical concern.The relationships between components are defined to show how they interact with each other.

the logical decomposition for our system system regroups the tightly coupled components into one subsystem , the logical decomposition will highly influence the physical decomposition of the distributed system later on.

### 4.1.2 Deployment View

A deployment view shows the configuration of run time processing nodes and the components that live on them. Deployment views show the deployment structures used in modeling the physical aspects of an object-oriented system.

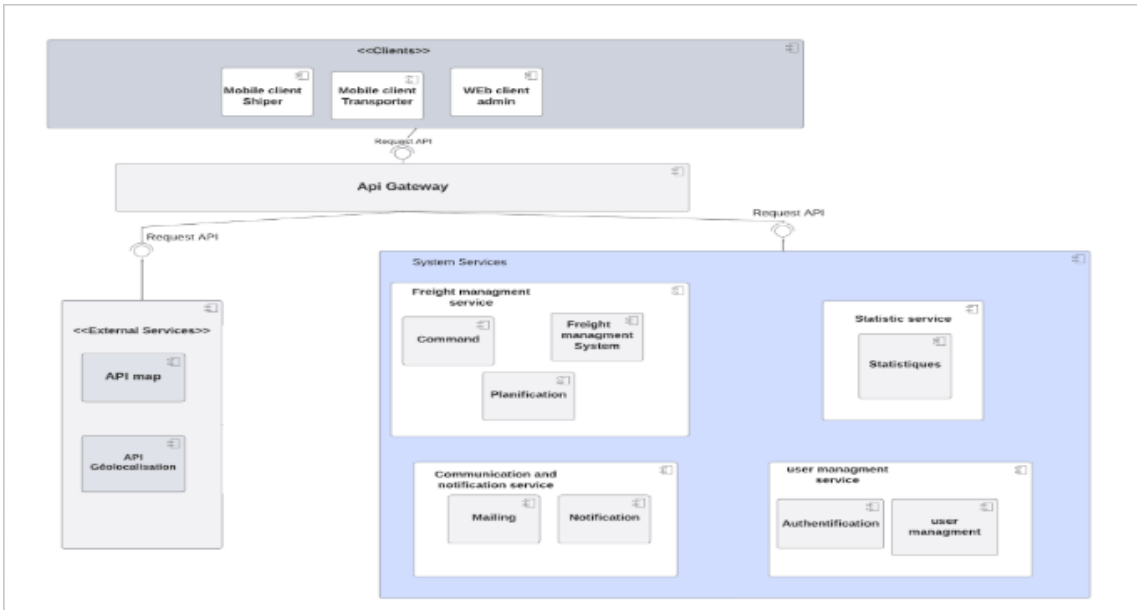



Figure 4.1: Decomposition view of the system

They are often be used to model the static deployment view of a system .our system being a distributed multi platform SaaS microservice system,where services are spread across servers.each service is hosted inside a container environment where it is independent from the other services. closely bonded services are hosted inside the same physical server .

Taking a closer look now at the inside deployment for our services . to provide the maximum scalability and performance for our distributed system . we said we will be using containers. services will be hosted inside containers where they have their own storage and computation. the moment a service is overloaded the load balancer will intervene to balance the workload . when the system scales up new instances of the service will be created.

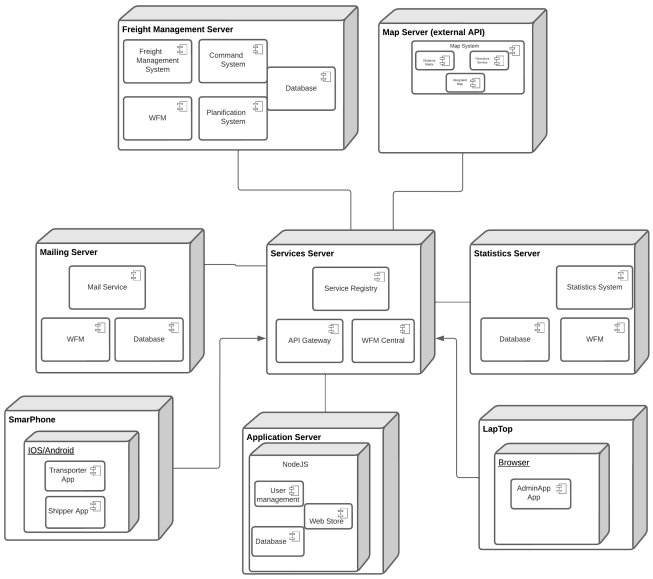



Figure 4.2: Deployment diagram

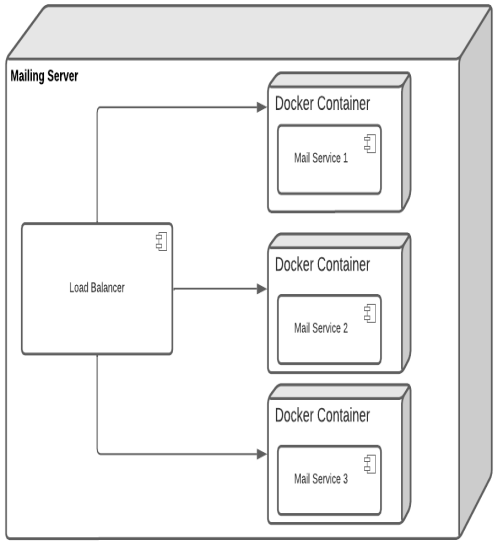



Figure 4.3: Deployment diagram for single server

### 4.1.3 Layered View

this just provides a global vision on the level of each component in the system, with each layer representing a different level of abstraction , each layer is designed to be independent of the other layers and has well-defined interfaces for communicating with the other layers.software layers:

- **Presentation layer:** This layer is responsible for presenting information to the user and capturing user input. for our system the web application and mobile application interfaces used by customers and employees to access the system.
- **API layer:**represents the interface that allows clients and services to interact with the underlying system.it is responsible for handling incoming requests and returning appropriate responses .
- **Application layer:** responsible for the business logic and processing of data. This layer includes the different services representing responsibilities of the system, such as order processing,Road estimation, and freight tracking.
- **Data layer:** This layer is responsible for managing data storage and retrieval,This layer includes : the database systems used to store information such as customer profiles, order history, and shipment tracking data.

### 4.1.4 Communication View

This view gives a global idea on the communication between various services . the main communication protocol is HTTP with the exception of the SMTP for the mailing service . for visual purposes direct communication between services isn't shown here but it does happen since the wfm work in a kind of peer to peer architecture .JSON will be the data exchange format commonly used . as for the lower level communication procotol . TCP/IP will be the dominant due to the nature of http and smtp. TLS/UDP will be used for the communication between the WFM nodes(zeebe documentation)

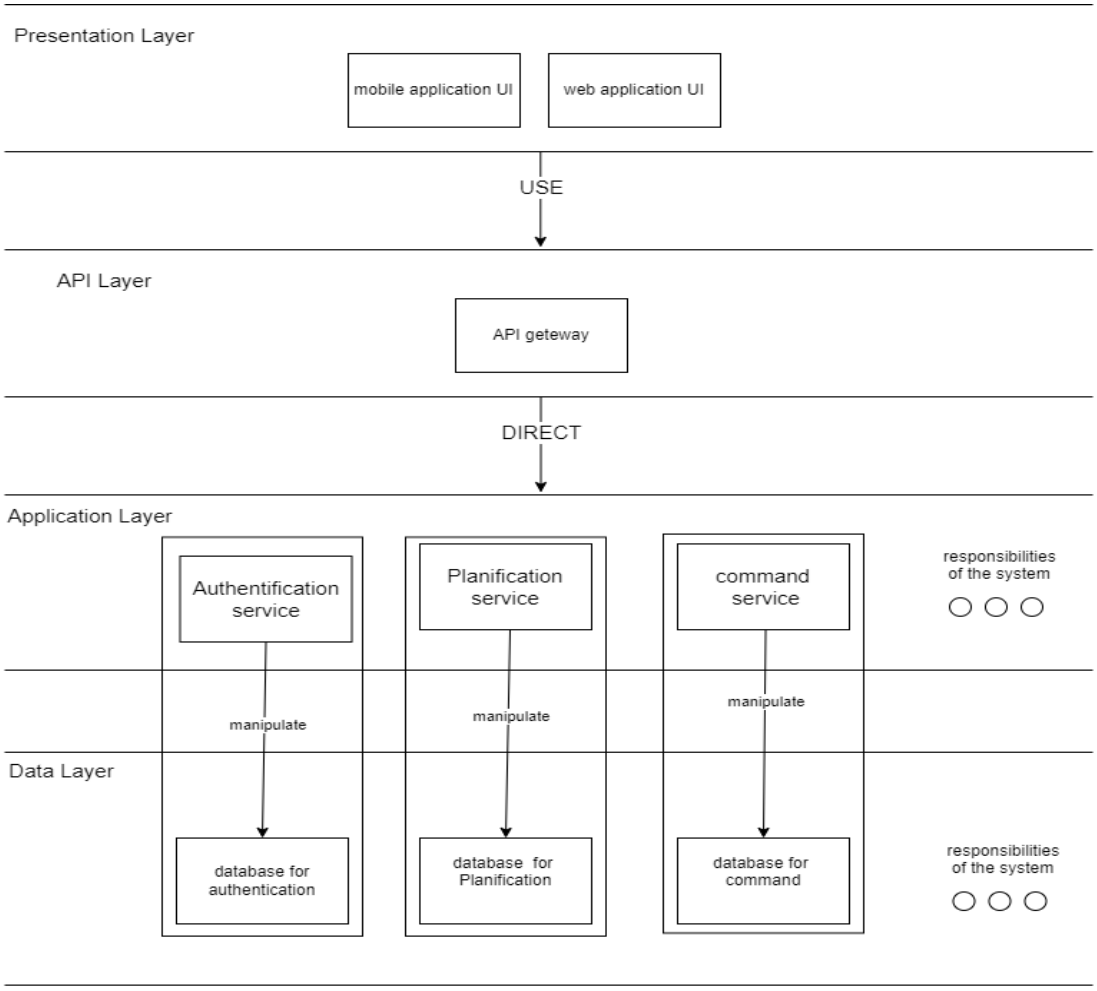



Figure 4.4: Layered View of the system

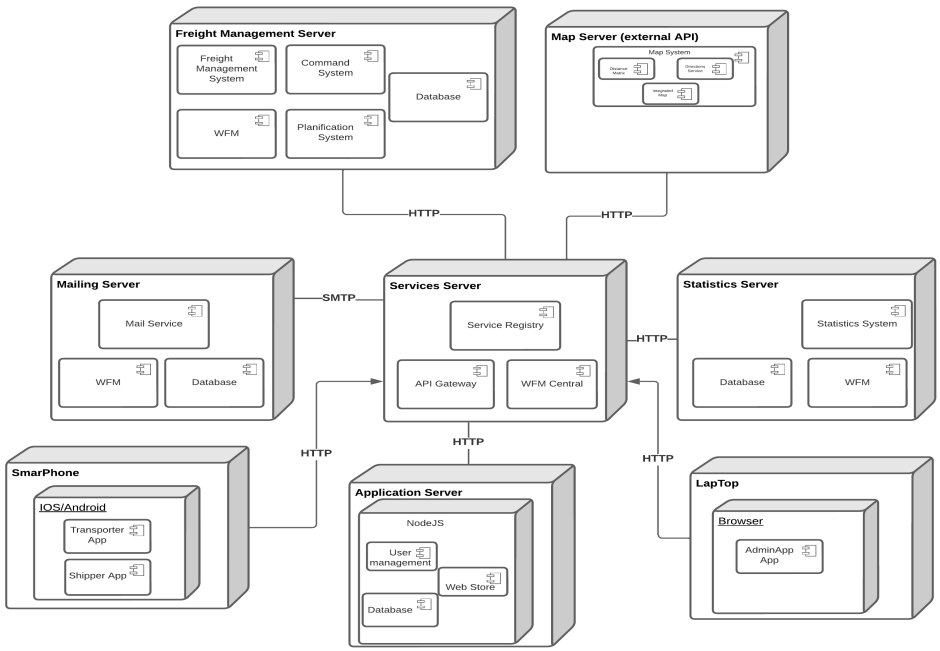



Figure 4.5: High level communication view

## 4.2 Architectural Styles

Architectural styles tell us, in very broad strokes, how to organise our code. It's the highest level of granularity and it specifies layers, high-level modules of the application and how those modules and layers interact with each other, the relations between them.

Although our system in general is a microservice system . it does contain and corporate other architectural styles to a certain extent . we will take a closer look at each architectural style largely used in our system one by one.

### 4.2.1 Micro-services

Microservices refers to a style of application architecture where a collection of independent services communicate through lightweight APIs.Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.this type of architecture is used in our system for multiple reasons :

- **The independent nature of subsystems :** the subsystems in our system are highly decoupled . on one hand each subsystems handles different data from the other, for example the user management deals with

the data related to users ,companies.. the mail system contains mailing data . these systems are highly isolated , with each system needing it's own database, type of computation ... .

- **Ensuring interoperability:** Along with virtualization technologies, microservices have enabled the loose-coupling of both service interfaces (message passing) and service integration (form and fit).microservices provides a slew of ready to use mechanisms for ensuring interoperability . a service could be using xml and another could be using json. the statistics db is of type no-sql while the user db is sql. micro services could be the best architecture to provide interoperability in a distributed virtualized heterogeneous environment .
- **Scalability improvements :** One of our primary concerns ,Since each microservice runs independently, it is easier to add, remove, update or scale each cloud microservice.developping a system for future deployment on a national or even international scale means that the architecture should allow scalability without the need for expensive horizontal or vertical sclaing. For instance, if a particular microservice experiences increased demand because of seasonal buying periods, more resources can be efficiently devoted to it. If demand drops as the season changes, the microservice can be scaled back, allowing resources or computing power to be used in other areas.

### 4.2.2 Client-server Rest Architecture

One of the most common architectural style used in most applications nowadays is based on the separation between the client and the server side (frontend and backend ), this approach offers a high scalability and flexibility to the system , which means both backend and frontend can maintain and scale up their part independently of the other one , Even in the development phase both sides can be developed in parallel regardless to the other side . This kind of approach suits our system application for reasons of :

- client-server architecture can improve interoperability and performance by standardizing communication protocols, load balancing, caching frequently accessed data, and encouraging modularization and separation of concerns.
- Standardizing :Client-server architecture is based on standardized protocols for communication between client and server components. This allows

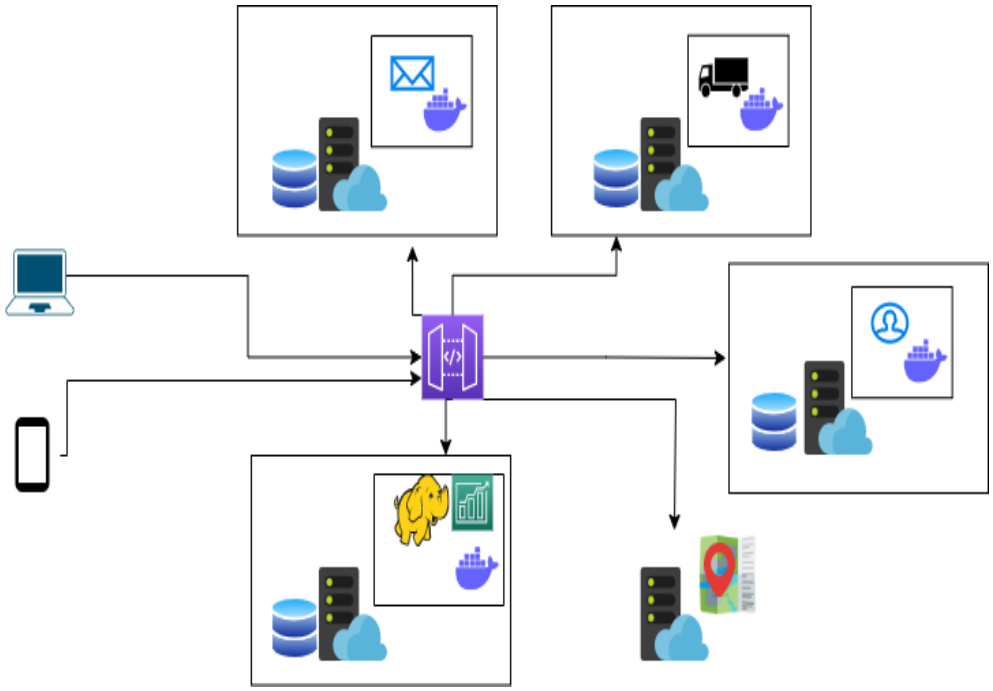


Figure 4.6: Microservices in the supply chain system

different clients and servers to be developed independently and still work seamlessly together as long as they adhere to the same protocol. Standardization also simplifies integration with third-party systems and services.

- Caching: Client-server architecture allows for caching of frequently accessed data on the client or server side. This can reduce the need for repeated data requests and improve overall performance by reducing the amount of data that needs to be transmitted over the network. Load balancing: Client-server architecture allows for load balancing across multiple servers, which can improve performance and scalability. When multiple servers are used, client requests can be distributed across them in a way that balances the workload and ensures that no single server is overloaded.

- Modularization: Client-server architecture encourages modularization and separation of concerns, which makes it easier to develop, test, and maintain both the client and server components. This can improve overall system performance and reduce the risk of errors or bugs

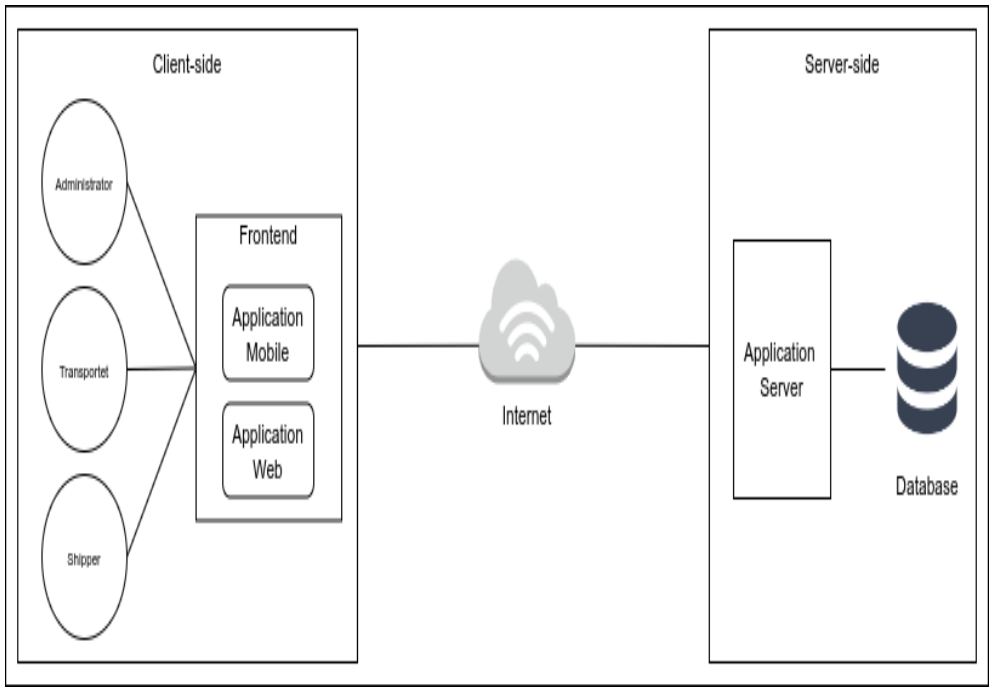



Figure 4.7: client-server in a sub part of the system

### 4.2.3 Big Data Architecture

This one is specific to the statistics server. which will be used for decision making and business intelligence processes. this will have the biggest amount of computations as the amount of data gathered is enormous especially as the system scales out . this means that special storage techniques,computation techniques,threading have to be provided for the assurance of performance which could only be found in the big data architectures.we will discuss these following topics

- **No-SQL database** traditional databases are stored in rows , this makes it defficult to parallalize batch processing and as a result a slower time to do operations on columns. for example if i need to get the mean time of travel trip am only needing to access the time column but in sql databases i will have to pass row by row to find them , this ultimately hurts the performance which is a key quality attribute, thankfully big data architectures provide a solution that is no-sql databases these databases are typically stored in columns,documents , object in any case they allow for faster treatment on data we previously discussed mongodb and it's benefits over traditional sql .

- **Data Query**In order to effectively apply map reduce on the database we must use effective optimized queries. now this could be done directly by the mongodb queries but instead we opt for a better solution that is the use of pig. traditionally pig only supported hbase but now it support mongo db as well .

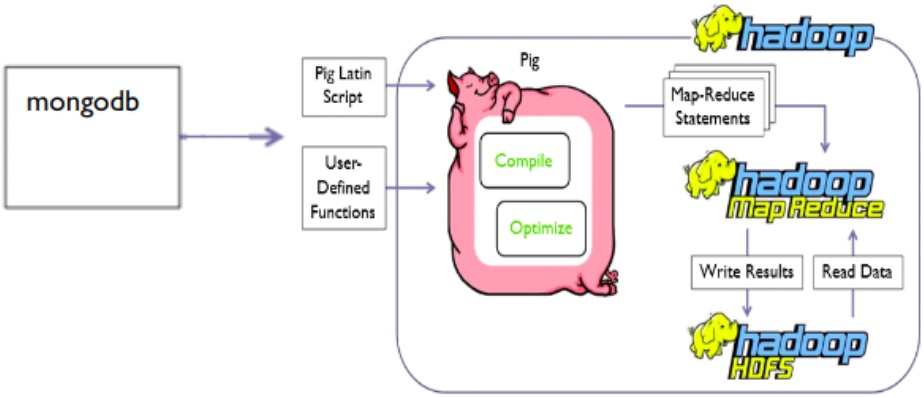



Figure 4.8: the use of pig for map reduce

- **Data Analysis and computations** map reduce uses data clusters to effectively parallelize the computations, the mechanisms start by applying batch processing on the mongodb clusters which will pass through a mapping area where similar data are regrouped and finally the reduce part for the final result .

### 4.2.4 Broker

This concerns the distributed workflow manager of the system. this functions as a broker architecture with four three components

#### 4.2.4.1 Clients

Each of our services will be configured as a Zeebe client.Clients send commands to Zeebe to:

- Deploy processes
- Carry out business logic
- Start process instances

- Publish messages
- Activate jobs
- Complete jobs
- Fail jobs
- Handle operational issues
- Update process instance variables
- Resolve incidents

#### 4.2.4.2 Brokers

The Zeebe broker is the distributed workflow engine that tracks the state of active process instances. Brokers can be partitioned for horizontal scalability and replicated for fault tolerance. A Zeebe deployment often consists of more than one broker.It's important to note that no application business logic lives in the broker. Its only responsibilities are Processing commands sent by clients Storing and managing the state of active process instances Assigning jobs to job workers Brokers form a peer-to-peer network in which there is no single point of failure. This is possible because all brokers perform the same kind of tasks and the responsibilities of an unavailable broker are transparently reassigned in the network.

#### 4.2.4.3 Exporters

The exporter system provides an event stream of state changes within Zeebe. This data has many potential uses, including but not limited to Monitoring the current state of running process instances Analysis of historic process data for auditing, business intelligence, etc. Tracking incidents created by Zeebe The exporter includes an API you can use to stream data into a storage system of your choice. Zeebe includes an out-of-the-box Elasticsearch exporter, and other community-contributed exporters are also available.

### 4.2.5 MVVM

used in the mobile app it has three levels view,viewmodel and the model

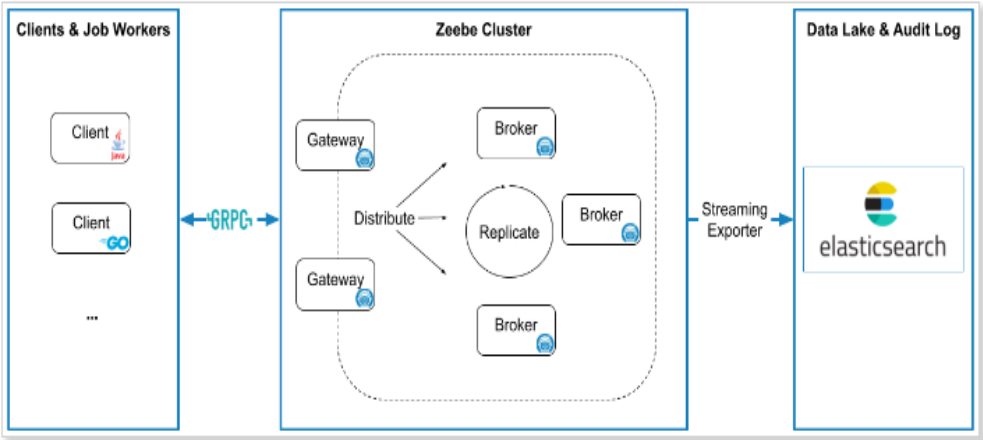



Figure 4.9: zeebe broker architecture

- **View** the UI of the mobile app , the view will be similar for both shipper and transporter with slight additional options.it observes changes in the viewmodel and updates the view accordingly.
- **Model** the model represents the scheme of the data of our application , there are various data classes involved in the application.
- **ViewModel** the central layer of the system , it takes data gathered from the various api's and converts them to the data that follows the models scheme .

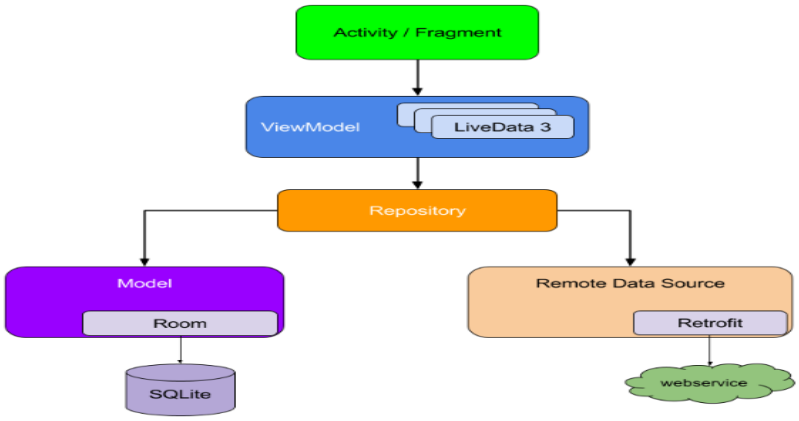



Figure 4.10: MVVM architecture

### 4.2.6 MVC

Similar to MVVM, this will be used for our web app.In MVC, the controller is the entry point to the Application, while in MVVM, the view is the entry point to the Application.

# Conclusion

In conclusion, the architecture of a distributed microservice application in the supply chain context is a powerful tool for managing complex supply chain processes. By breaking down the application into smaller, more manageable microservices, it becomes easier to maintain and scale the system as needed. Additionally, the use of distributed architecture helps to ensure that the system is fault-tolerant and can recover quickly from failures. The distributed microservices architecture breaks down the system into independently deployable and scalable microservices. Each microservice focuses on a specific business capability, such as inventory management, order processing, logistics, or analytics. This modular approach allows for easier development, testing, and maintenance of individual services while enabling rapid adaptation to evolving business requirements.