Autonomous Vehicle Rental Service Project System Design

By
Group 5

Sahithi Bommadi (015949219)
Vaishak Melarcode Kallampad (015017496)
Venkata Lakshmi Praneetha Moturi (015913495)
Pavan Karthik Gollakaram (015945670)

Table of Contents

Diagrams

1. Introduction

The rapid advancement of technology has aided in the transformation of people's travel habits. People have moved from using public transport to private vehicles and now we have rental car services like Uber and Lyft. With the advent of self-driving cars it’s just a matter of time when these companies release their own service of rental AVs. Autonomous vehicles (AVs) are already on our roads, with experiments underway in locations around the country. Thousands of autonomous vehicles will soon be on the roads, autonomous buses and transit vehicles will be delivering rides in the foreseeable future.

Autonomous vehicles was the much awaited technology in the field of automobiles due to the increase in the number of crashes everyday which further increased the need for driver assistance technologies. Most of the accidents these days are caused by human errors, and using Autonomous vehicles improves safety to a great extent. It also reduces the traffic congestion,and using such systems people will be able to smooth out the traffic flow for all the cars. Over time, as the frequency of accidents is reduced, Vehicles could be made much lighter which would eventually increase fuel economy even more. It will also reduce travel time and transportation costs.

Smart linked vehicles are expected to be driverless, smart, and vehicle collisions in the future, making them the best mode of urban transportation. Autonomous vehicles are man-made intelligence-driven vehicles that use sensors and LIDAR. To appear genuine, automakers have started working around here to recognize the potential and resolve the existing measures being taken to appear at the expected result. They are self-sufficient and, like source and objective, make course decisions with little input from the customer. As a result, the primary test will be to modify and check existing advancements in regular automobiles in order to understand how they might be applied to near-future driverless vehicles.

Purpose - To build a Cloud-based autonomous vehicle rental service system using CARLA simulator.

Objectives - To design a cloud-based autonomous vehicle rental service system using CARLA simulator offering the best possible user interface and interactions. Will leverage a web development framework in order to deploy the model in real-time. We build a system using the latest architecture called Elastic Beanstalk.

Market analysis - The market of autonomous cars is currently valued at USD 1.64 billion in the year 2021 and is expected to grow to USD 11.03 billion by the year 2028. Similarly, the cab booking service industry is currently valued at USD 213.41 billion and is projected to reach USD 354.2 billion by 2027. With companies like Waymo, Uber, Zoox ramping up their research and development, combined with the big players like General Motors, and Ford jumping in this domain, some predictions go way higher up to USD 2000 billion. The only where the research is lagging is the cloud-based renting system which is struggling to keep up with the rapid development in Artificial intelligence. But significant costs and resources can be saved using cloud-based rental systems in the right way.

2. SYSTEM REQUIREMENTS

System Goals:

The primary goals of our projects are as follows:

1. Improve the Autonomous car rental system
2. Keeps the system Centralized - All forms of data and system operations will be centralized in one location, making it easier to monitor the system and deliver the best possible service to users.
3. Handles the system's scalability and reliability features.
4. To ensure that a user receives his desired vehicle as soon as possible, the autonomous rental management system will respond more quickly to complete the process.
5. Provides an interface with minimum downtime and minimal performance effect.

2.1. User groups are discussed in detail in the User Perspective diagram of the cloud system.

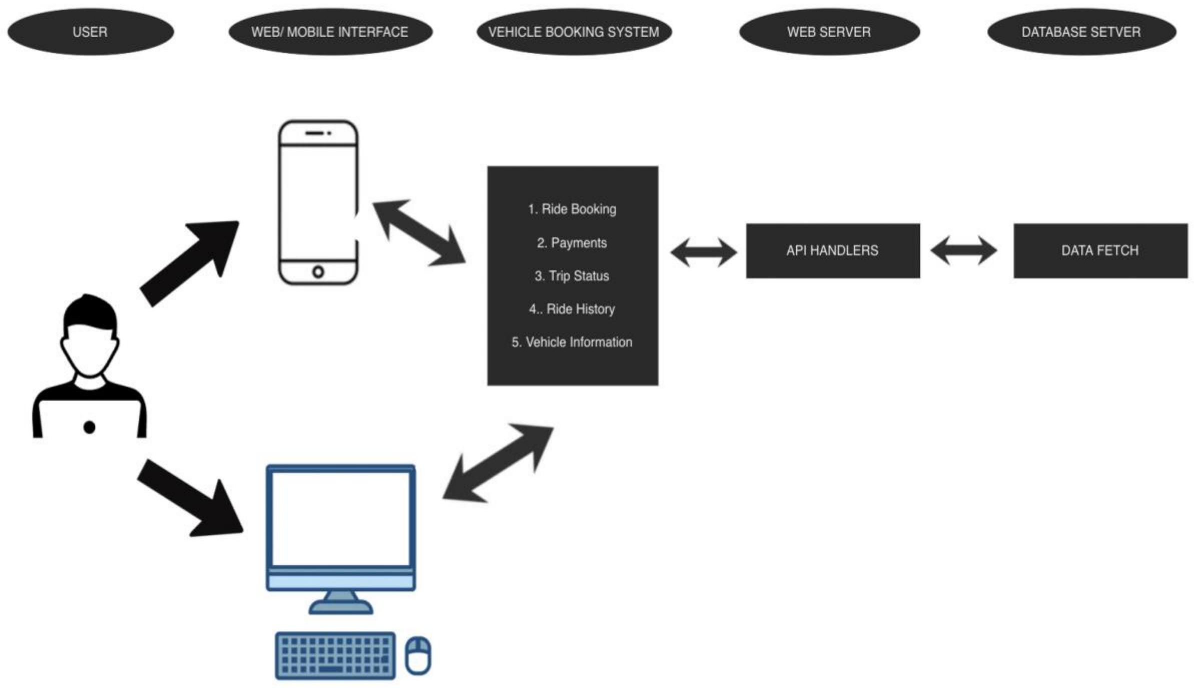
(a) User group specifications:

Users	**Responsibility**
User/Customer	Reserve a ride.
Admin	Manage the real-time updation of car fleet
Fleet Maintenance	Maintain each car and clean it.

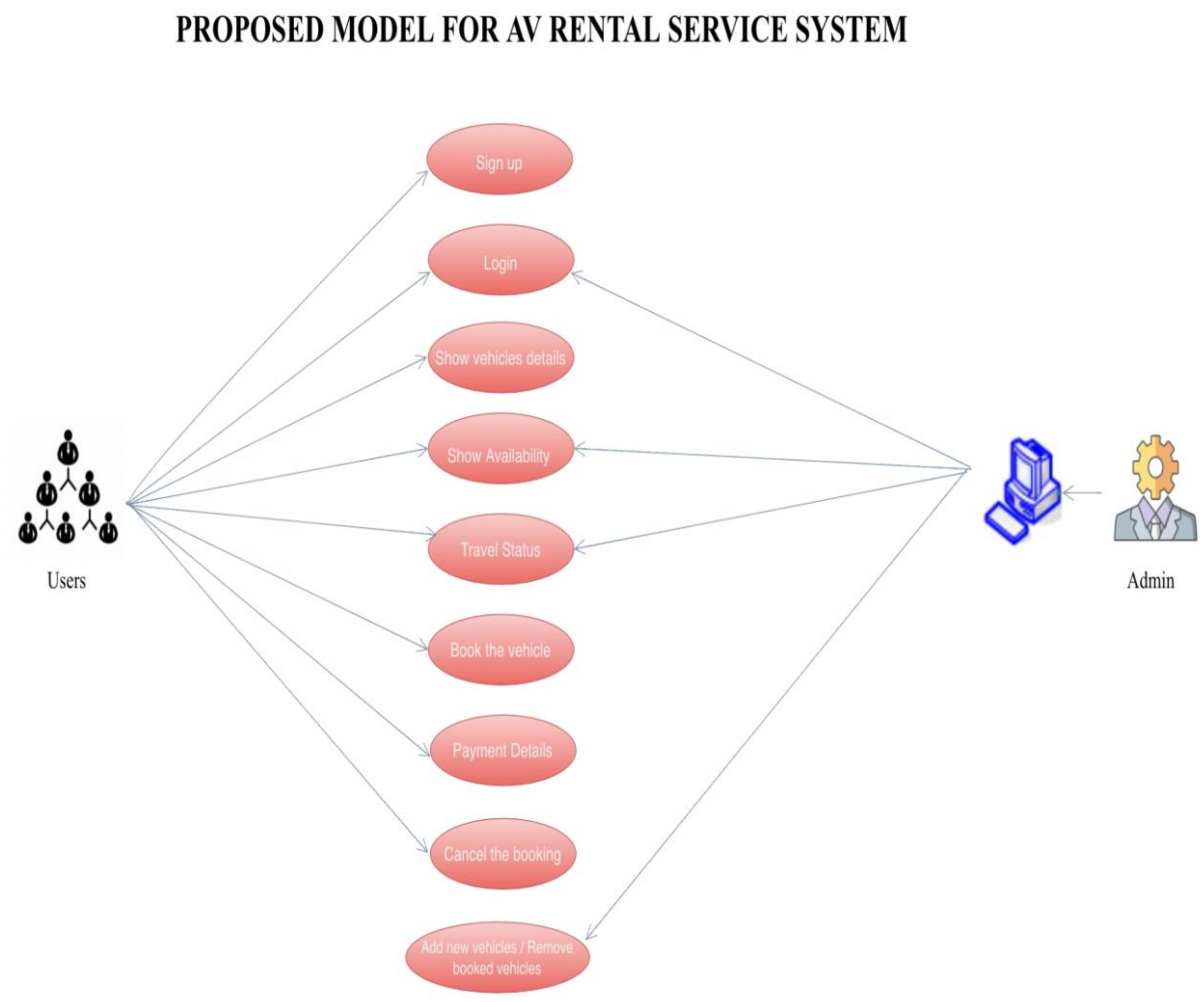
(b) User group service function specifications

Services	**Functions**
User/Customer	Search and book an AV ride.
Admin	Manages all the backend, requests and system.
Fleet maintenance	Takes care of the conditions of every car in the fleet.

CURRENT SYSTEM OVERVIEW FOR AV RENTAL SERVICE SYSTEM



(I) System Overview

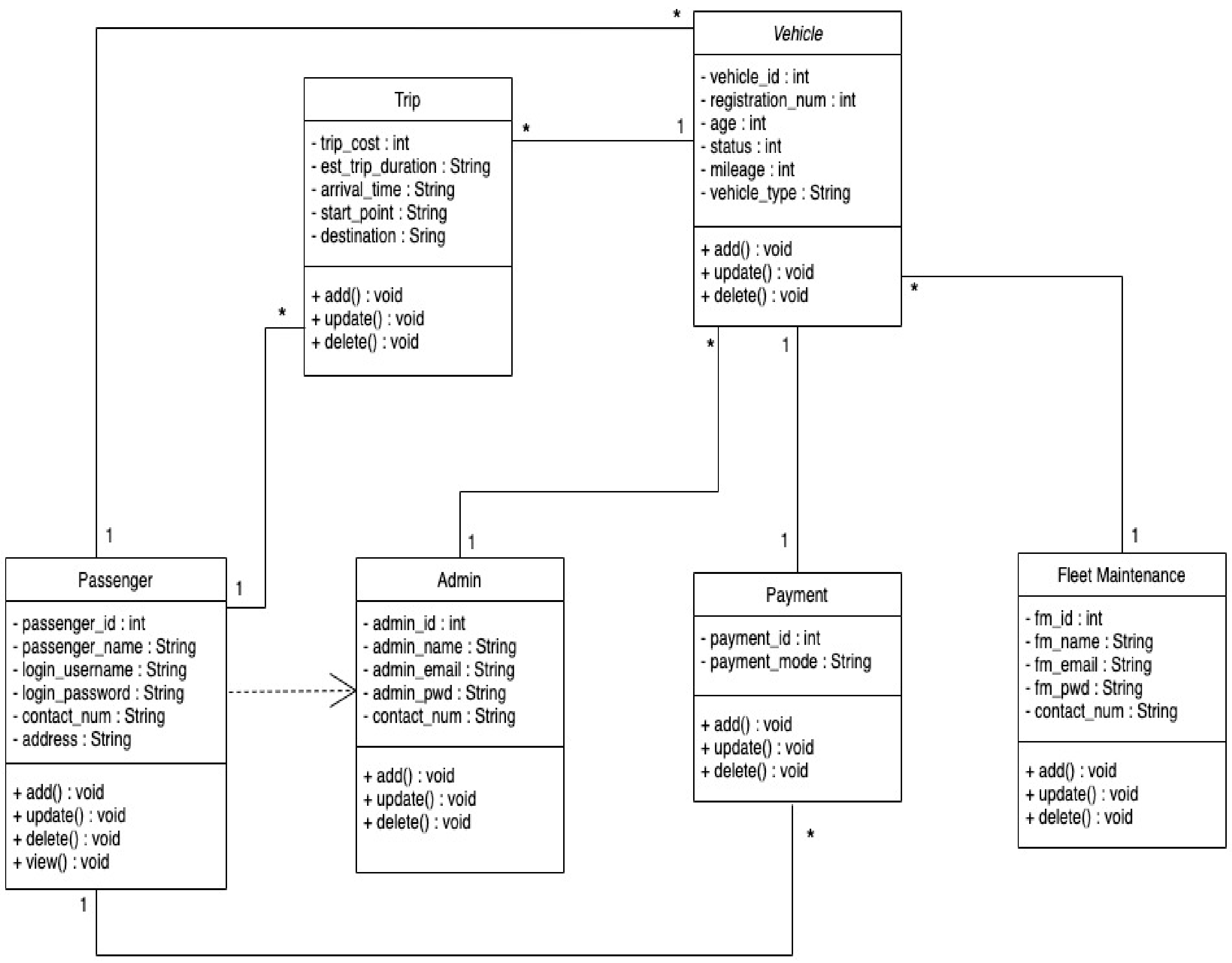


(II). Proposed System Model

2.2 Cloud System User Scenario analysis

The AV Rental system that we aspire to build has a total of three actors - User, Admin, Fleet Maintenance. Below is a Class Diagram illustrating the system.

Class Diagram



III. Class Diagram

The system architecture contains the following classes:

1. For Admin

Admin_id : primary key	int
admin_name	String
admin_email	String
Admin_password	String
contact_number	String

2. For Vehicle

vehicle_id	int
registration_number	int
age	int
status	int
mileage	int
vehicle_type	String

3. For Payment

payment _id	int
payment_mode	String

4. For Fleet Maintenance

fm_id : primary key	int
fm_name	String
fm_email	String
fm_password	String
contact_number	String

As per the class diagram included above, the system architecture is made up of the following classes:

- **Admin :** As shown in the class diagram, Admin has the privilege to update the fleet.
- **User/Customer/Passenger:** The only update that the passenger can do is the destination address and pickup address.
- **Trip :** One or more actors can update the Trip.
- **Vehicle :** A vehicle's details are open to all three actors but updating privileges are limited.
- **Payment :** Transactions are the most important part of the system.
- **Fleet Maintenance:** Every member of the fleet maintenance can update and fix the fleet.

Our initial functionalities enabled for the user:

- Input a pickup and destination location through text or geolocation.
- Search for all the available cars for rental and select one of the options.
- Manage the booking and access the details of the ride and the car.
- Manage payments for the ride taken.

The Admin would have access to and the responsibilities of:

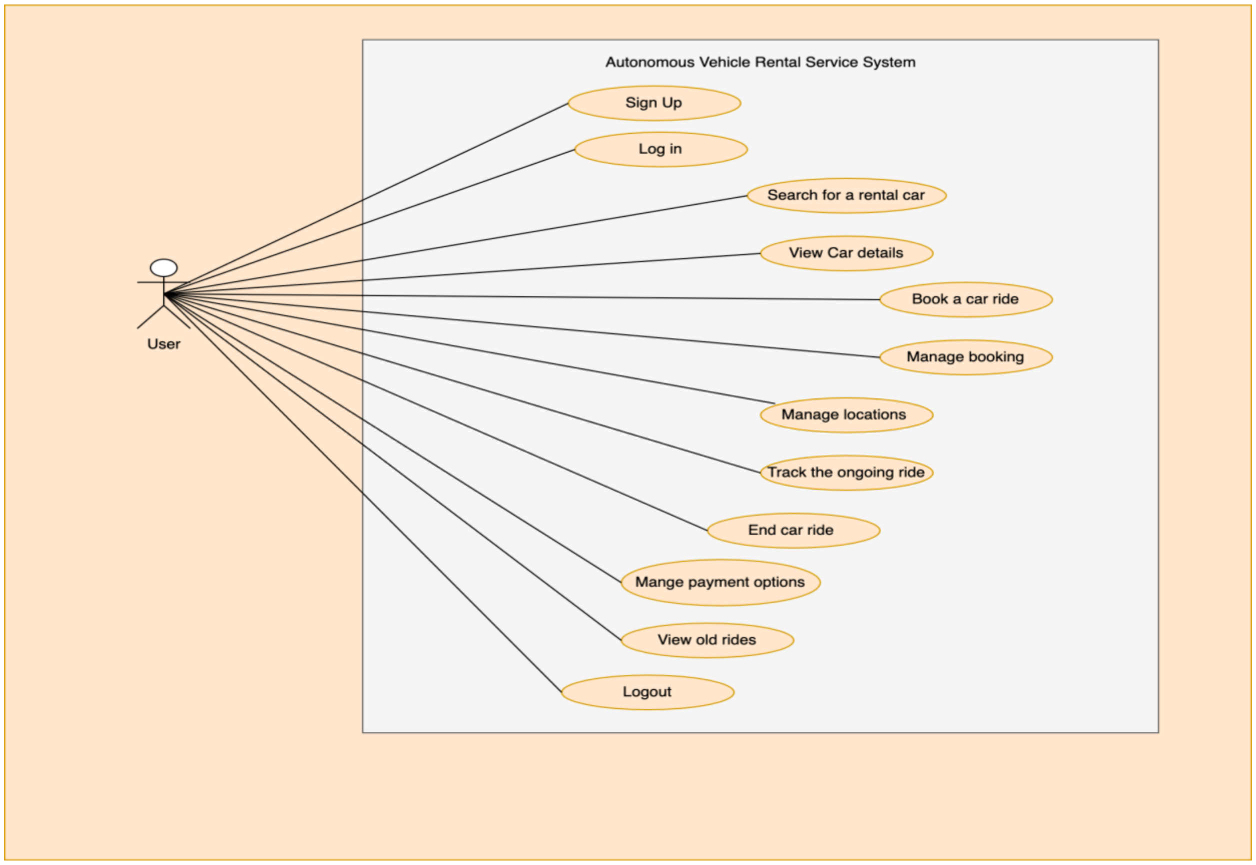
- View, edit, delete, modify the details of the car fleet available.
- Have access to all the users' information.
- Manage the locations where the rentals are available.
- Manage the trips in case of any emergencies.

Likewise, the fleet maintenance team would have the following responsibilities:

- Add, delete any car from the fleet.
- Repair/fix a damaged car.
- Clean a car.
- Manage the fleet in real-time.

Use Case Diagrams :

User side :

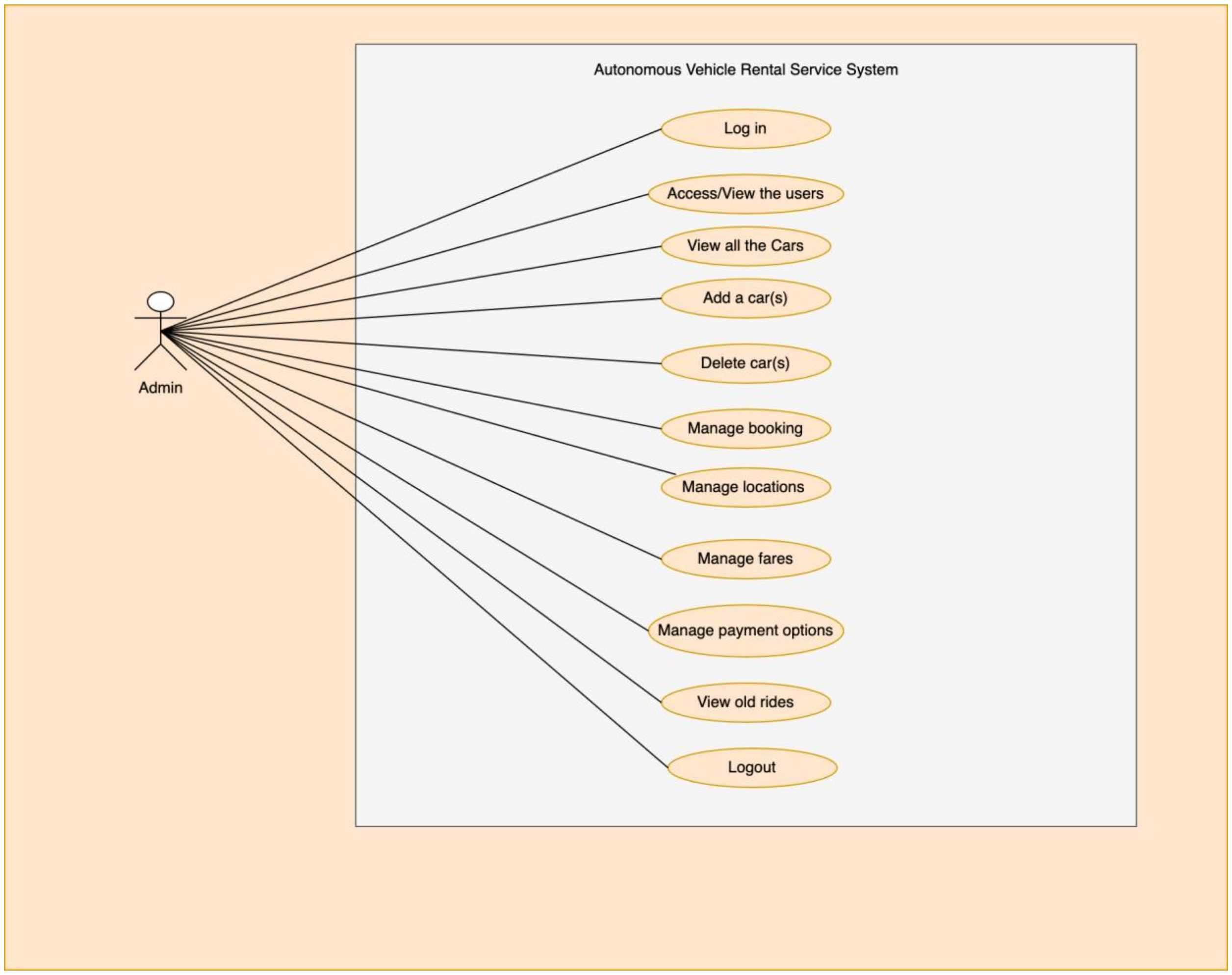


IV. User side Use-case diagram.

Functionalities to which the 'User' has access in this system are as listed below:

- The **User** can **Sign Up, Login, Logout** of the system.
- The **User** can **search for a car.**
- The **User** can **view all the car details.**
- The **User** can **book a ride.**
- The **User** can **manage booking and locations.**
- The **User** can **track the ongoing ride** and **end the ride.**
- The **User** can choose and **manage payment options**.
- The **User** has the access to **view old rides** taken only by them.
- The **User** can give **reviews/ratings and comments** on his experience

Admin side Use Case Diagram :

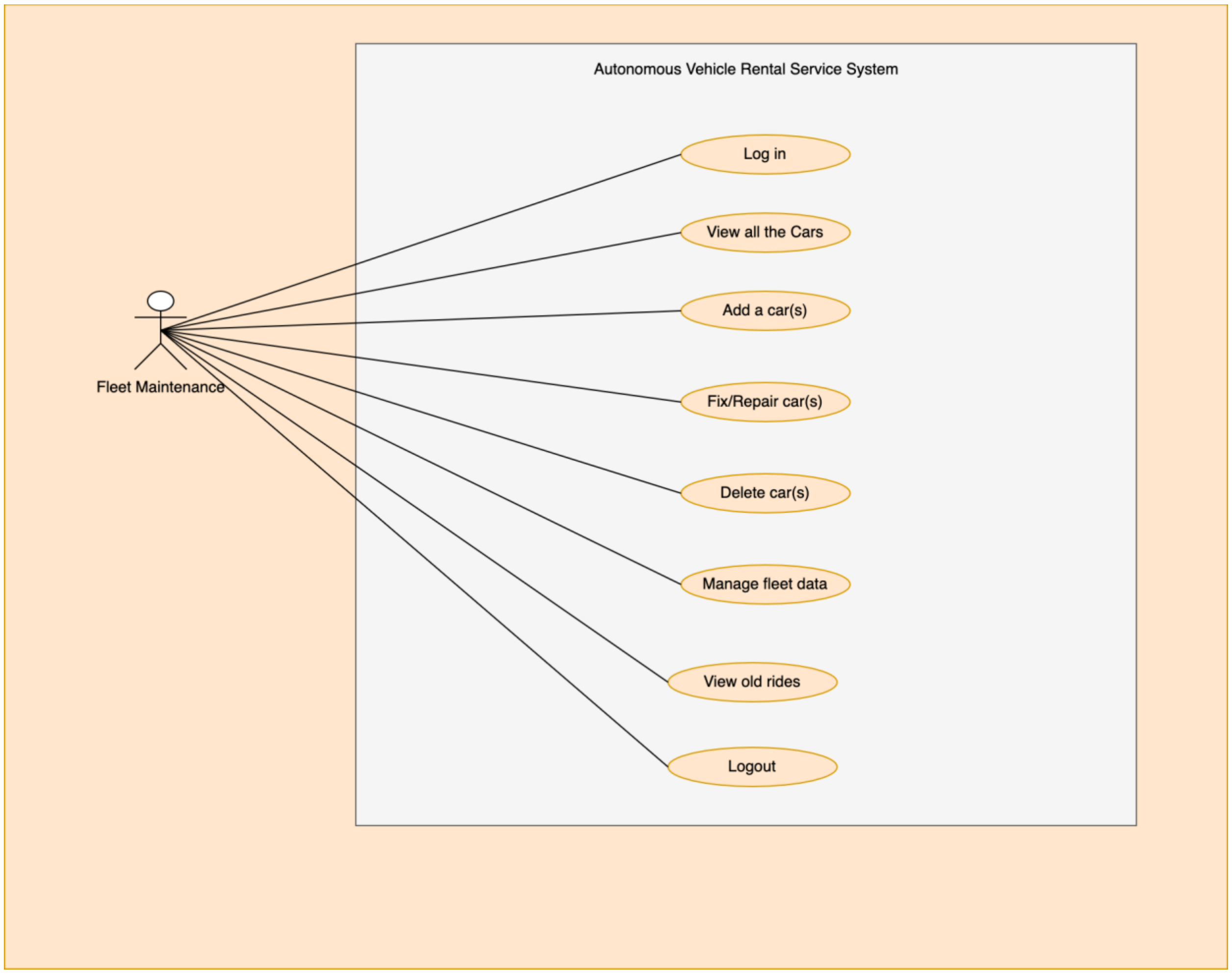


V. Admin Side Use-case diagram

The following are the functions handled by the 'Admin' in our AV system:

- **Admins** can access the system at any time and **log in** or **log out.**
- **Admin** can **access** all the relevant **user's details** and can use them further to their liking.
- **Admin** can also retrieve the **car's details.**
- Once, accessing the data, **Admin** can **add car(s)** and can **delete car(s).**
- **Admin** is granted the privilege to **Manage Booking.**
- **Admin** can **Manage fares.**
- **Admin** can **Manage Payment Options** - adding, deleting, and modifying, solving issues in payments.

Use Case Diagram - Fleet Maintenance side



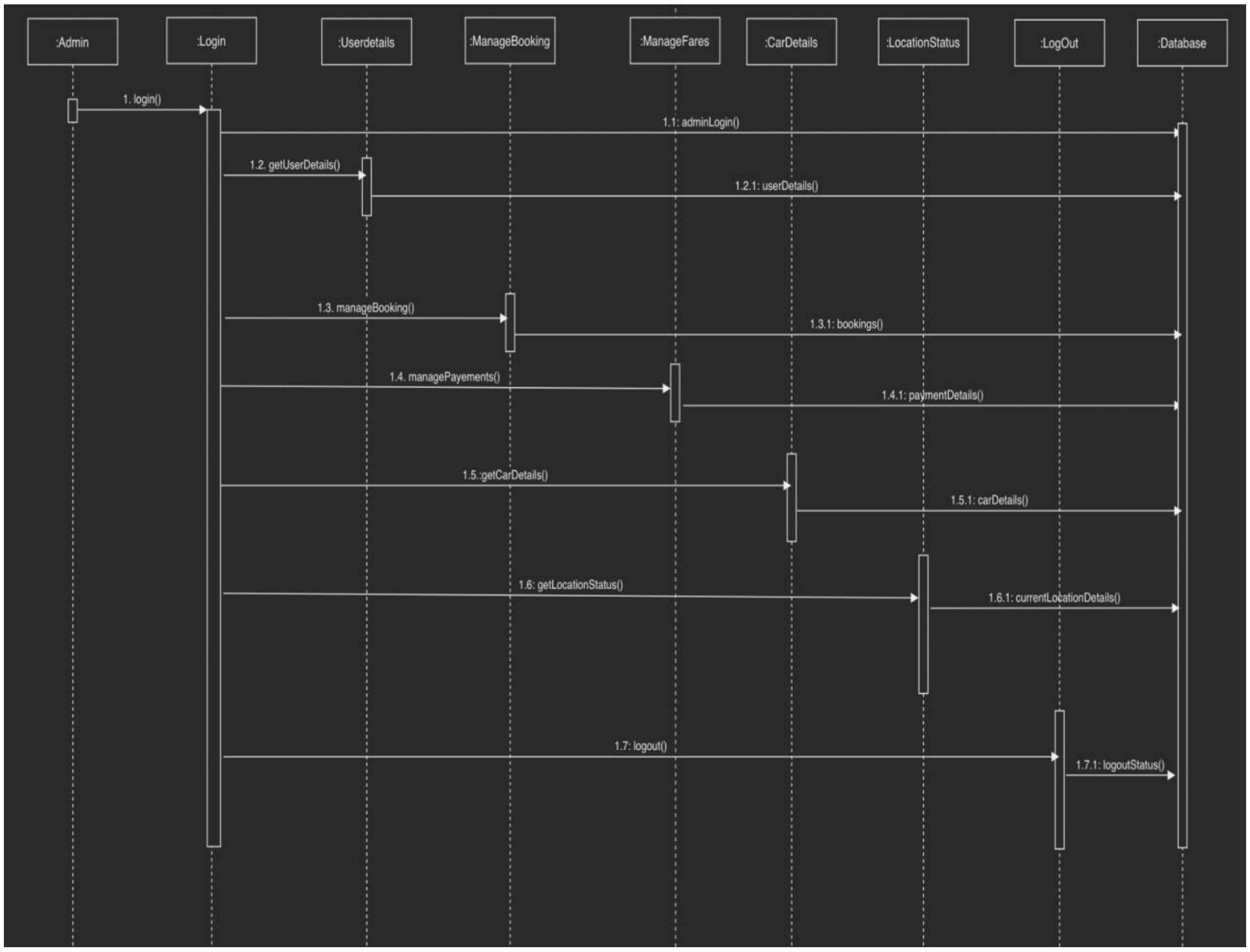
VI. Fleet Maintenance side Use-case diagram

Functionalities handled by 'Fleet Maintenance' in this system are as listed below:

- The **Fleet Maintenance team** can either **Login** or **Logout** of the system anytime.
- The **Fleet Maintenance team** can **access car details** and can use them further to their liking.
- The **Fleet Maintenance team** can **add, delete car(s).**
- The **Fleet Maintenance team** can **repair/fix car(s).**
- The **Fleet Maintenance team** can **view old rides** to see who has rented the car.

SEQUENCE DIAGRAM OF THE SYSTEM

The sequence diagram below shows the event timeline from the admin side. The process occurs as follows:

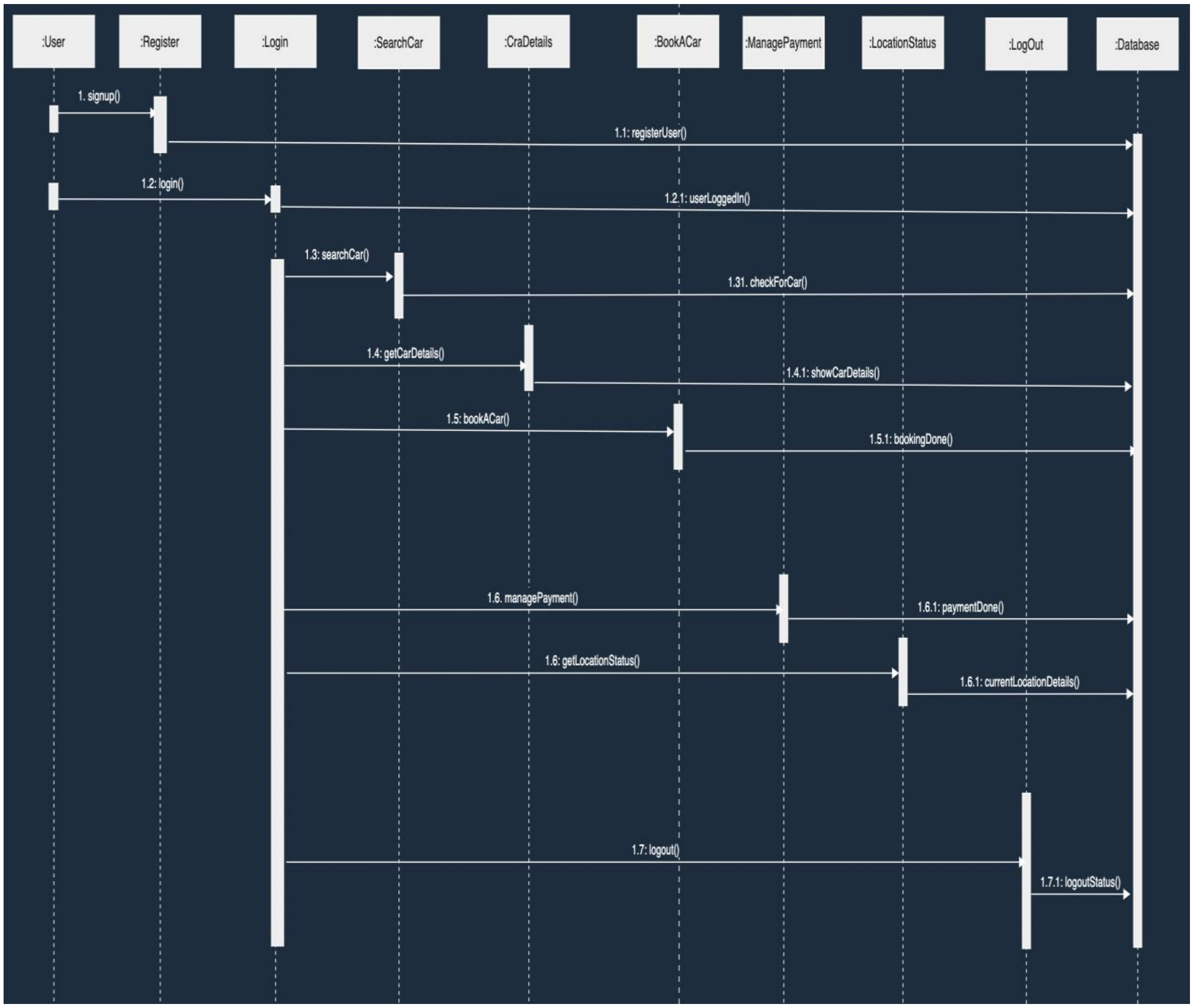


VII. Admin side sequence diagram.

<u>ADMIN SIDE ACTIVITIES :</u>

- The system's admin logs in.
- The admin can manage/view the following details after logging in.
- The administrator will be able to see the user's information.
- He would validate Manage Booking, which is done by the user.
- Organize fares for rides.
- Adding and deleting vehicle information as needed
- Keep an eye on the current location. The location is updated in real time.
- The database keeps track of all the details and changes in the stages.
- The admin logs off.

The Sequence Diagram from the User Side is as follows:



VIII. User side sequence diagram.

<u>USER SIDE ACTIVITIES :</u>

- In the order listed below, the user performs the following tasks:
- Users must first register for the online application before they can log in. Once logged in, the following events occur in the order specified:
- The user looks for a car in a location.
- Then he will look into the car details and then the user can book it.
- After booking the car, he will make the payment and the details about the car are presented on the online portal.

- The real-time location of the car is regularly updated in the web application, and the user can check it at any moment.
- The database stores all of the information and modifications that occur at each stage.
- At the end of the session, the user logs out.

API design of the proposed system

User

End Point	Method	Description
User/signup	**POST**	**For a user to register.**
User/nearbycars	**GET**	**Finds the nearby cars for the specific user.**
User/confirm	**POST**	**Allows the user to manage his trip destination and starting point and book the ride.**
User/getcar	**GET**	**Gets the details of the vehicle.**
User/map	**GET**	**To view the real time location of the AV.**
User/confirmpayment	**POST**	**Allows user to make payment**
User/previousrides	**GET**	**Gets the ride history for the user**
User/review	**POST**	**To give feedback about the autonomous driving system.**

Admin

End Point	Method	Description
admin/register	POST	For an admin to register.
admin/getuser	GET	Allows admin to access user details.
admin/ getcar	POST	Gets the details of the vehicle.
admin/addcar	POST	Allows the admin to add a car into the system.
admin/removecar	POST	Allows the admin to remove a car from the system.
admin/setfare	POST	Allows admin to change fare details.
admin/complaints	GET	Allows admin to get complaints.

Fleet Maintenance

End Point	Method	Description
fm/getcar	GET	It gets the details of the car and the status of the car, repairs to be done etc.
fm/carfixed	UPDATE	To update the status of the car and the repairs history.

Car

End Point	Method	Description
car/status	**POST**	**To update the rides ongoing/completed and the status of the vehicle.**
car/location	**POST**	**Allows to track the AV in real time**

2.3 System Services

These can be divided into two forms:

- High-level functional services
- High-level data services

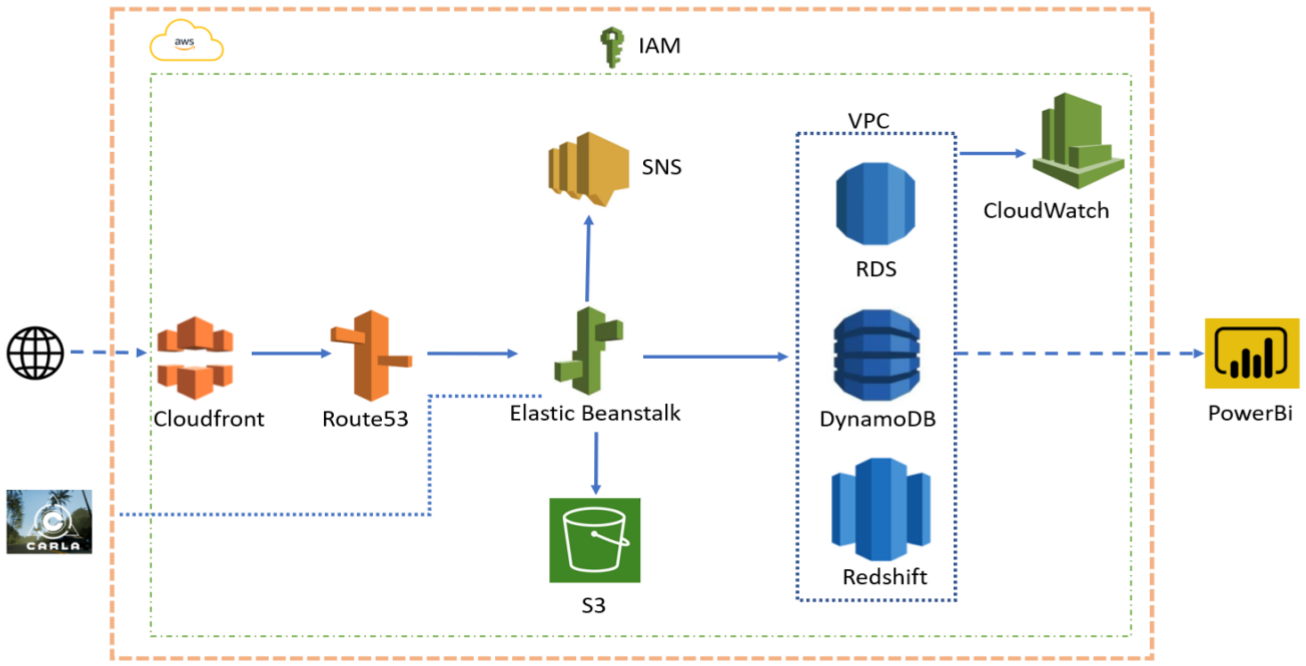
Clubbing together, a few services that we came up with are as follows:

1. **Client/User service:** Dubbed to be the Passenger while booking a ride, this would include managing a functional User Interface which is also user-friendly at its best. It also consists of User profile management.
2. **Booking service (data and functional):** A web server/API specifically dedicated for a faster and smoother booking process. Fetches the required data and manages relevant details until the ride ends.
3. **Payment or Billing service (data and functional):** This would house the different options for payment alongside tracking all the details. Also allows asynchronous services to make sure no transaction is delayed because of the others.
4. **Car communication service:** This would function as the backend and interface logic behind the actual CARLA simulator and API deployments over and through the hosted environment.
5. **Location Data service:** All the while during the trips and non-rental, the locations of every car in the fleet have to be tracked to avoid complications of any sort. Can be used from every side: the user, the admin, the fleet maintenance side.
6. **Log Data Service:** This service is dedicated to managing the defects, logs, and monitoring pieces of the system.

3. Cloud System infrastructure design

3.1. Infrastructure Design

Autonomous Vehicle Car Implementation is deployed and maintained on AWS Cloud. Our project's AWS cloud architecture is depicted in the diagram below.



IX. Infrastructure Design

Components of cloud system:

Amazon VPC: We'll create our own VPC (virtual private network) for the databases, giving us full control over the virtual network infrastructure, including resource allocation, connectivity, and encryption.

Cloudfront: AWS CloudFront is an Amazon Web Services global network that distributes applications, Development tools, multimedia, and other forms of material to users securely and quickly.

Route 53: Route 53 is a scalable and simply accessible DNS web service. It allows users to be redirected to other AWS components, such as SNS, Elastic Beanstalk, or Amazon S3 buckets.

Elastic Beanstalk: AWS Elastic Beanstalk enables you to swiftly install and maintain apps in the AWS Cloud without worrying about the infrastructure that supports them. Elastic Beanstalk simplifies management without limiting options or control. A logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations, is an Elastic Beanstalk application. It automatically applies changes to current resources or deletes and deploys new resources when an environment's configuration settings are updated, depending on the change.

SNS: The Amazon Simple Notification Service (Amazon SNS) is a cloud-based messaging service for both A2A(application-to-application) and A2P(application-to-person) communication. We'll use it to send users messages and push alerts.

Amazon S3 Bucket: The application files and any log files generated will be stored in an Amazon S3 bucket.

RDS: Amazon RDS (Amazon Relational Database Service) makes it simple to set it up, run, and deploy a relational database in the cloud. The 'MySQL' engine will be used to store information on users, automobiles, and rides.

DynamoDB: Amazon DynamoDB is a cloud - hosted, fully managed, key-value NoSQL database built to run high-performance applications at any scalability. We intend to keep data statistics linked to automobiles, rides, and high-volume payment transactions, and we require minimal latency in payment services.

Redshift: A data warehouse is provided by this Amazon web service. We chose it since it has a wide range of integrations with applications like Amazon QuickSight, PowerBi for analytical purposes. We will be able to extract data and deliver relevant insights and business information as a result of this.

CloudWatch: The monitoring tool we've chosen is AWS CloudWatch.CloudWatch collects and analyzes surveillance and operational data in the form of logs, metrics, and events to provide a unified view of AWS resources, apps, and software running on AWS and on-premises web server.

PowerBi: Power BI is a collection of software services, apps, and connectors that combine to transform disparate data sources into logical, visually immersive, and interactive insights. Your data could be in the form of an Excel spreadsheet or a collection of cloud-based and on-premises hybrid data warehouses. Connecting to and visualizing your data sources is made simple by Power BI.

3.2. System architecture with components and connectivity:

Components of System Architecture:

User Interface: The frontend of the program is built with ReactJS, and the backend is built with NodeJS. Customers may use the User Interface to book a ride, track the car, and pay for it. Our app will work on both mobile phones and computers with greater resolution devices, such as pcs and laptops.

ReactJS: It is a free open-source JavaScript library that provides online applications with simple, rapid, and adaptive frontends. The React system's core features include a virtual DOM program and server-side delivery, allowing even the most complicated apps to operate in seconds.

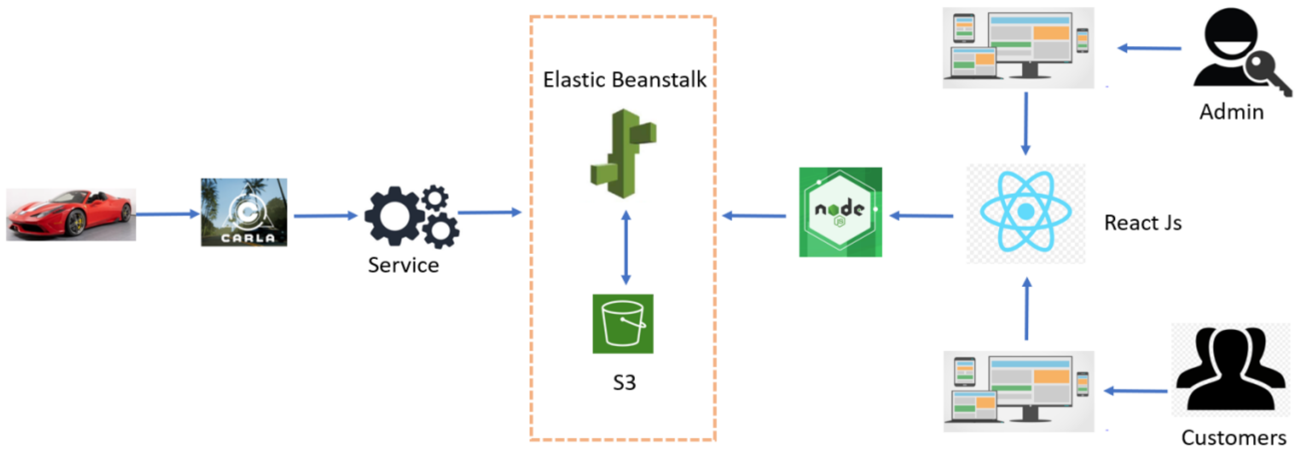
NodeJS: It's a cross-stage server running that's free and open-source, and it's used to handle sudden increases in JavaScript demand. Engineers may stack portable programs and run capacities at the same time with NodeJS without crashing or slowing down the servers. The environment is likely the most accessible option due to its non-impeding, input-yield activities. Code runs faster, which benefits the entire server running environment.

AWS: AWS cloud computing delivers a cloud framework platform that is easy, adaptable, and extremely dependable. AWS provides limitless server capacity that can accommodate any load. It has auto-scaling to create a self-monitoring framework that adjusts to the actual need based on the vehicle used. As a result, building an application on AWS will allow us to work more efficiently and smoothly.

Service Manager: To control the simulation of data, we will use the Python and C++ libraries supplied by the CARLA community. The Traffic Manager provided by CARLA can be used to simulate a realistic traffic situation.

CARLA Simulator: It's an open-source driving simulator for self-driving cars. It was designed from the ground up to help with the development and training of automated vehicles. It is an open-source code that can be used for free. Open digital assets include architecture, urban layouts, and automobiles. It includes a robust API that allows users to manipulate all aspects of the simulation, such as pedestrian behavior, traffic generation, and climatic conditions.

3.3. Cloud System Deployment Design:



X. Cloud System Deployment Design

- Admin and customer interfaces are the two types of user interfaces. Both user interfaces are linked to the same backend server, which is connected to a cloud network.
- A Elastic Beanstalk distributes traffic evenly among the instances by routing connections between the backend and the cloud. All the relevant application files and configurations are stored in an S3 bucket.
- The CARLA simulator, through which the automobiles converse, is linked to a service manager, who communicates with the database on a regular basis using Python and C++ APIs. DynamoDB is used to store the sensor data generated by the CARLA.
- The database is accessizble using cloud-based backend services.

Section 4

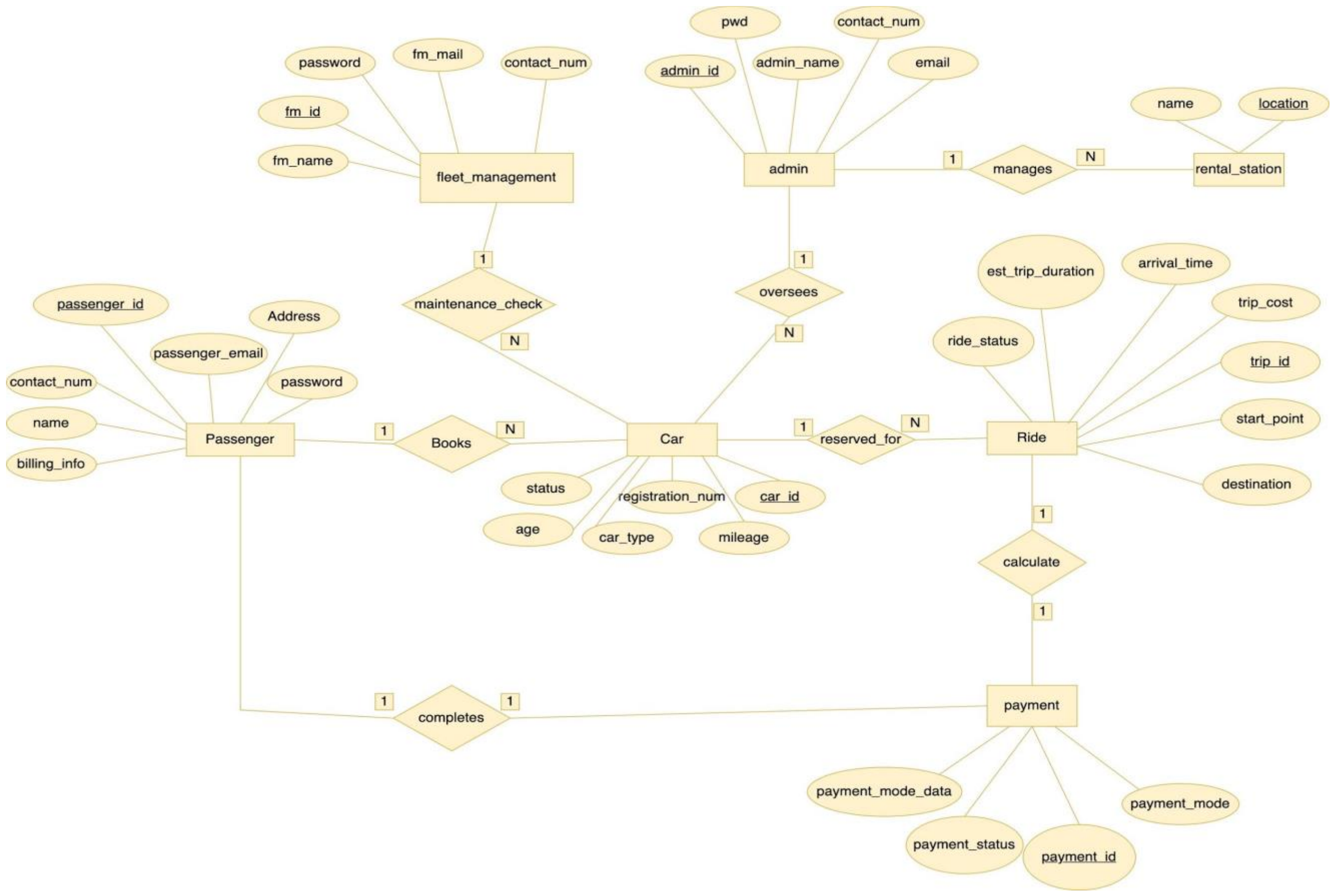
Cloud DB design

4.1. Cloud system selection:

While the traditional database - MySQL would suffice most of the times, for the larger and continuous data, we might need MongoDB or any other such NoSQL databases. Services that only involve Client, fleet, booking and scheduling, might work fine with the traditional methods whilst for logging and location services, we might pick another database management system.

4.1.1. SQL Database - Entity - Relationship (ER) diagram

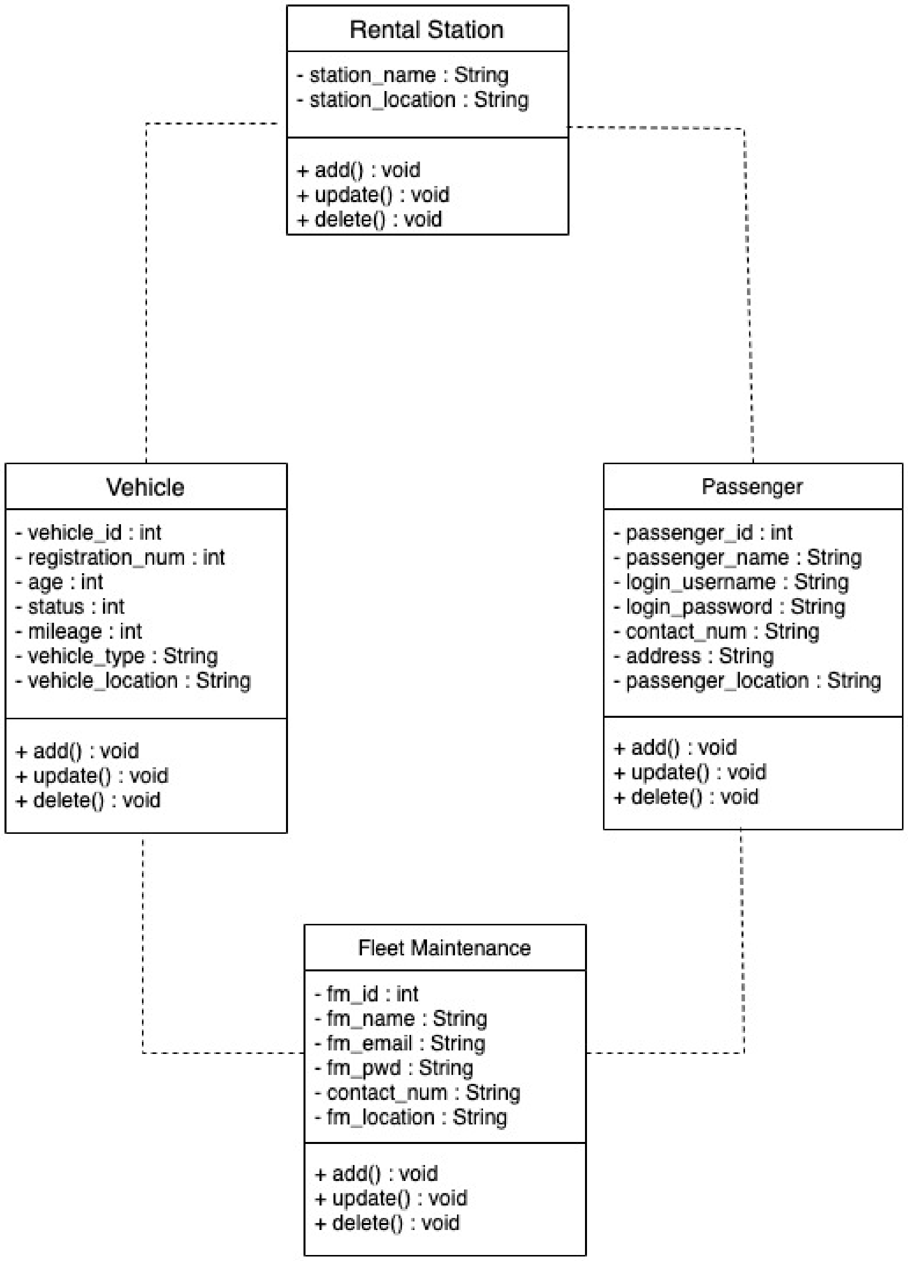
An Entity-relationship diagram successfully describes the interrelated entities in the system. In the below diagram, it explains the relationship between 6 different entities. Each entity is also mapped to their attributes and the relationship between the attributes and the entities too has been shown adeptly in a single diagram.



XI. Database Entity-Relationship Diagram

4.1.2. NoSQL DB design

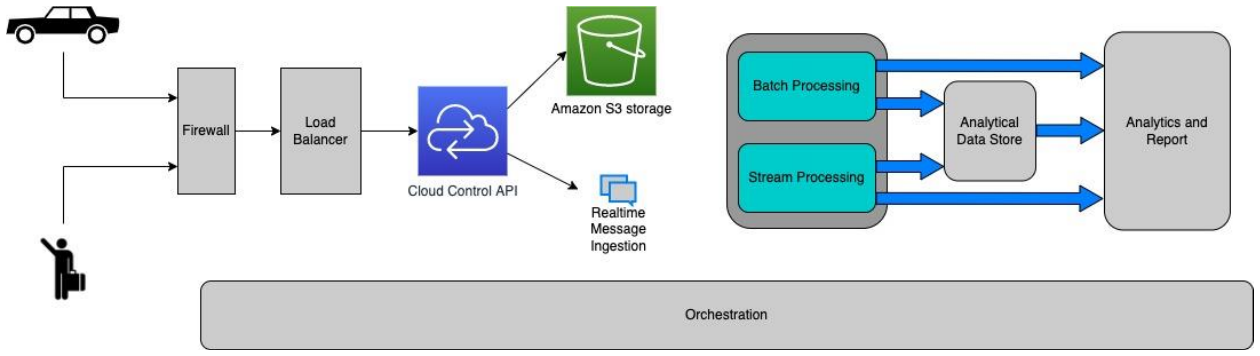
While the user and fleet data could be managed through traditional databases, the sensor data along with the images from all the angles, steering angles, information about steering angles and locations, that are going to be contiguously received, stored and used for further processing needs non-relational databases.



XII. No-SQL Database design

4.1.3. Big Data System Design

With the plethora of data having to be received, modified, analyzed, and the eclectic data types and formats involved, leveraging a data warehouse would be the best way to evaluate data. The bigger organizations leverage cloud data warehouses like Google BigQuery, Microsoft Azure, Amazon Redshift, Snowflake and so on. Compiling and getting the information into a regularized pattern and store in a warehouse would be possible using an ETL (Extract, Transform, Load) tool. Using one of such tool, we can retrieve the necessary data.

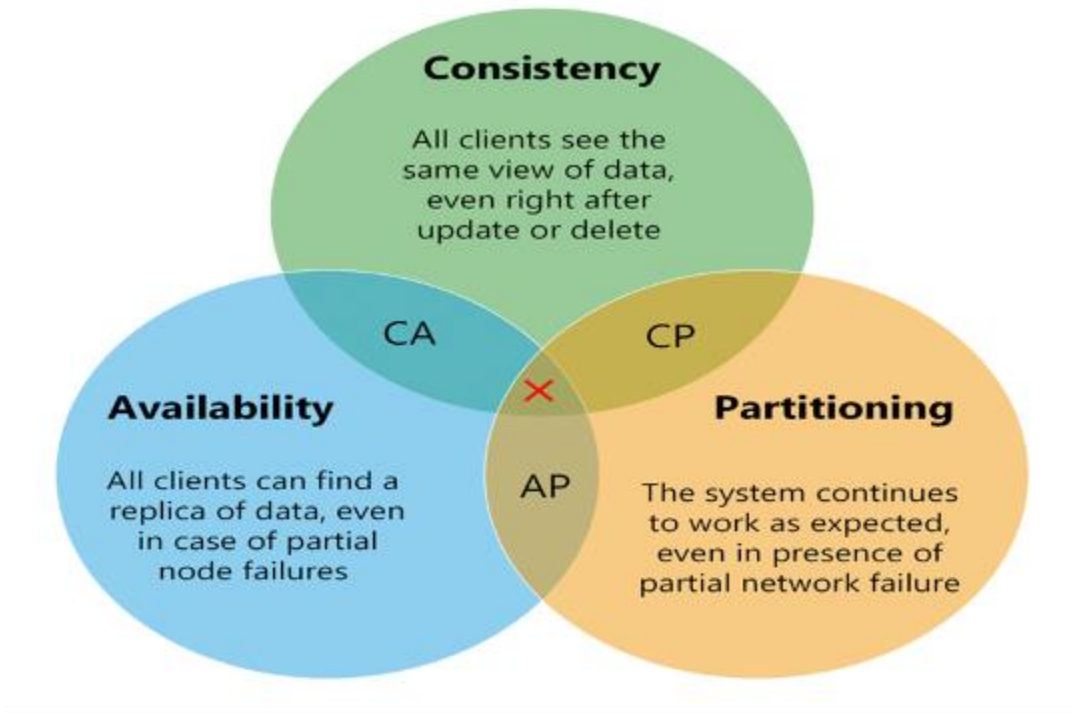


XIII. Proposed BigData system design

5. Cloud system design issues and solutions DB designs

5.1. Cloud system design issues

Every Distributed system design has the following three characteristics according to CAP theorem.


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
×
Availability
Partitioning
AP
All clients can find a replica of data, even in case of partial node failures
The system continues to work as expected, even in presence of partial network failure

XIV. CAP theorem representation

There are three ultimate things to be taken into consideration while designing any large-scale systems

- Safety
- Availability
- Consistency

As the client/user load increases the following issues might occur in an Autonomous vehicle cloud.

1. In certain situations, like a system failure, there is a possibility that a part of the system might fail causing the loss of some of the data. So, in that case we can simply store all the data and transactions in the data storage blocks regularly so that if anything happens further, the data won't be lost permanently and can easily get retrieved.

2. In the same situations as when the system fails an active switchover will be performed from one system to another system which helps the system function without any loss or extra costs. They are kept in an active and standby condition at all times to ensure that they are always in sync.

3. As the amount of API calls to the web servers grows, the load balancer may automatically scale.

The cloud system architecture for the autonomous car rental system will be created and deployed with all of the potential obstacles in mind, as well as an appropriate set of algorithms to address the issues that arise.

5.2. Strategies and solutions relating to cloud issues in application Resource Management, Scalability, and Security

Scalability and Resource Utility Management

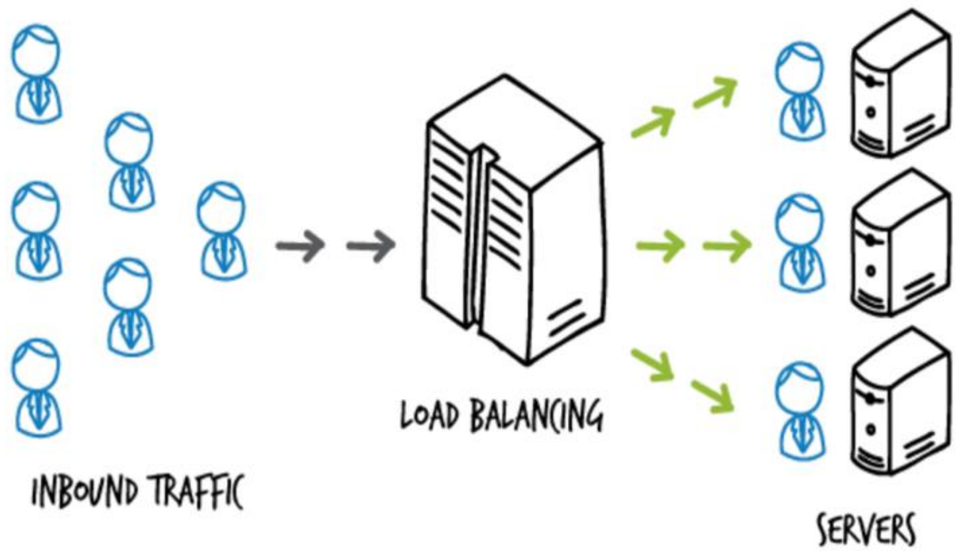
The key goal is to assign resources and capabilities dynamically in order to efficiently handle the workload. The components listed below are resource management components that help with cloud database design concerns.

- **Scheduling**:
  An event calendar must be developed in order to distribute resources among the workload.
- **Discovery**
  This is the process of determining which resources are accessible to help with the burden.
- **Allocation**
  In this case, resources are allocated to competing workloads on a demand basis.
- **Provisioning**
  This is the process of allocating resources to workloads.

Strategies to Address Workload Limitation:

Network load balancing:

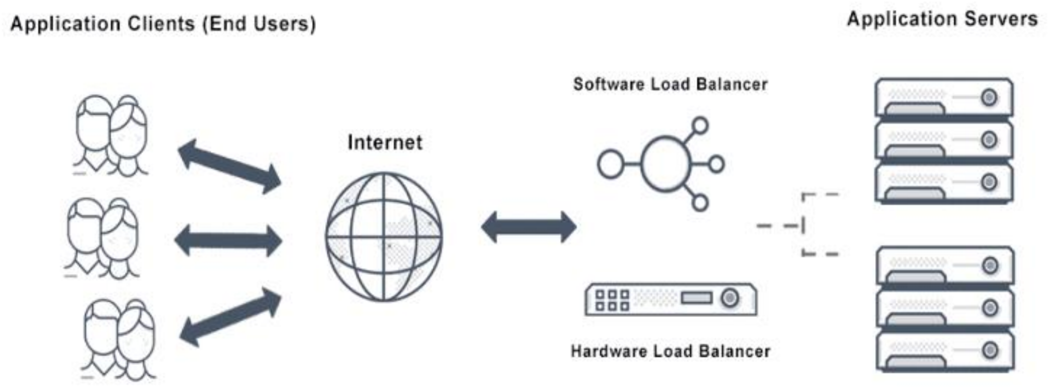
Multiple requests from the client(s) need to be load-balanced across the backend servers so that no single backend server gets overloaded. It efficiently distributes incoming traffic across backend servers. Load balancers sit in the middle receiving and distributing incoming requests to any servers.



XV. Network load balancing

Server-side load-balancing

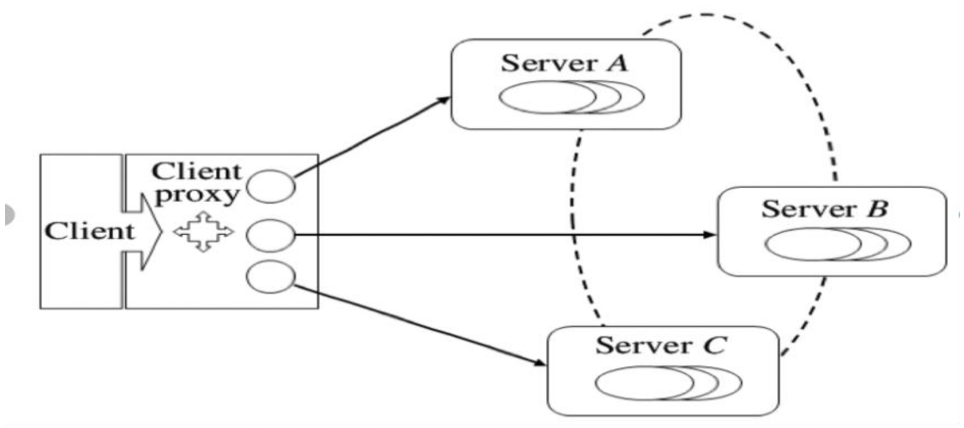
In our arrangement a load balancer is put in between the backend server instances / application servers. It acts as a middle component for all the requests coming from the application clients. It then directs requests coming from clients to servers based on algorithms like round-robin etc.



XVI. **Server-side load-balancing**

It prioritizes responses to the specific requests from clients over the network.

Client-side load-balancing
In this arrangement, the load balancing decision is left to the client itself. The client here takes the help of the server to identify the server instances and then the request goes to client-side load balancer and then goes to specific backend server instance.



XVII. **Client-side load-balancing**

1. **RAFT Algorithm:**

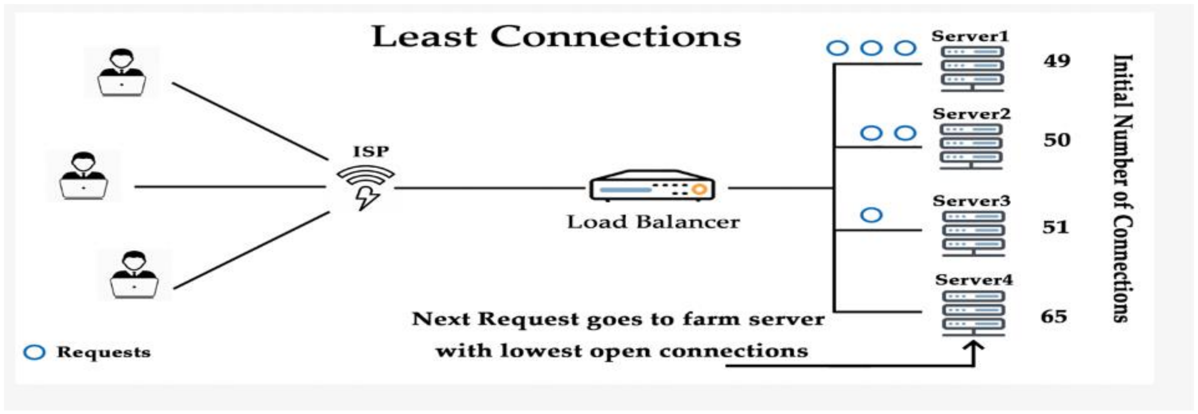
This is a distributed consensus leader-based algorithm where all the requests are sent to only one place: the Leader. As the repository of requests has been centralized, now it's the turn of the leader node to manage the requests. All other nodes are prone to only receive the requests as assigned by the leader, no nodes are in direct communication with the requests. And the leader is to be relied upon for a balanced distribution of the load. The four fundamental safety rules of the algorithm make it easier for load balancing. The rules would be

- Election security and safety
- Leader appendment
- Log equivalence.
- Completeness of the leader.

One of the famous large-scale applications of this algorithm is the Kubernetes, in which the RAFT algorithm is the heart of the key-value store hood ETCD.

2. **Least Connection algorithm:**

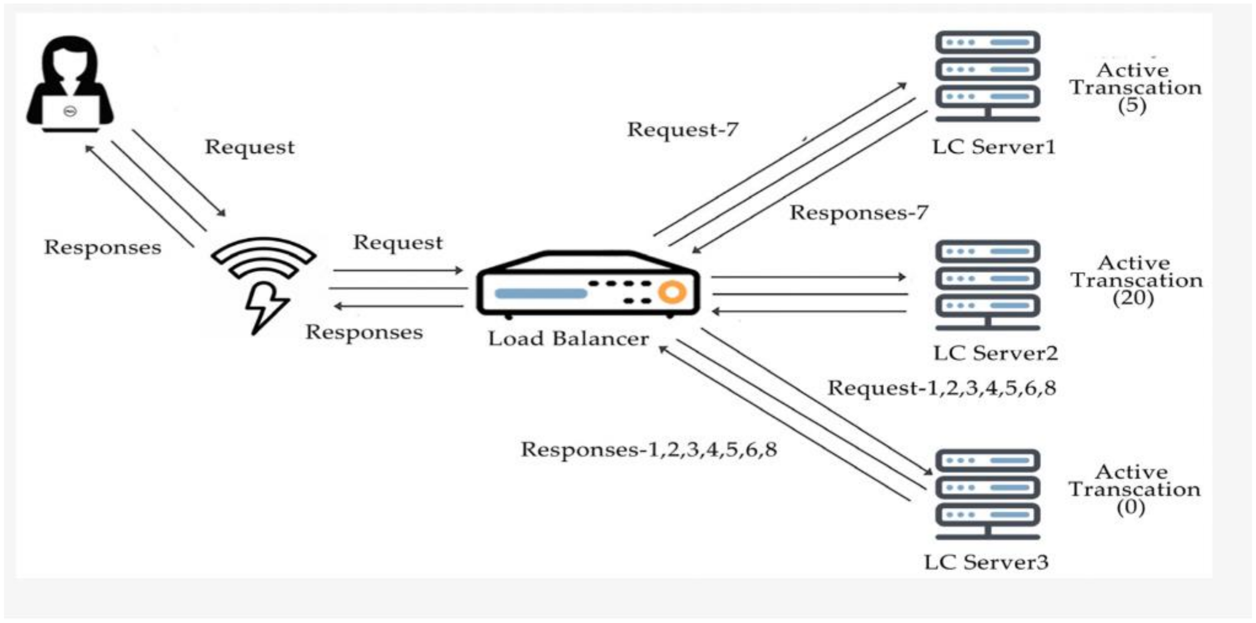
The algorithm here sends traffic to those servers which have the fewest connections open at that time assuming that all connections require almost equal processing power. It depends on the current number of active connections on each server and forwards the new connection to the server with the least number of active open connections.


Least Connections
ISP
Load Balancer
Server1
Server2
Server3
Server4
49
50
51
65
Initial Number of Connections
Next Request goes to farm server with lowest open connections
Requests

XVIII. Least Connection algorithm

3. Weighted least connection:

This provides the ability to assign different weights to each server, assuming some servers can handle more connections compared to other servers. Here a new connection is forwarded to a server based on the number of active connections or its proportion to the weight. If two servers have equal number of active connections, then the new request will be directed to the server with greater weight.

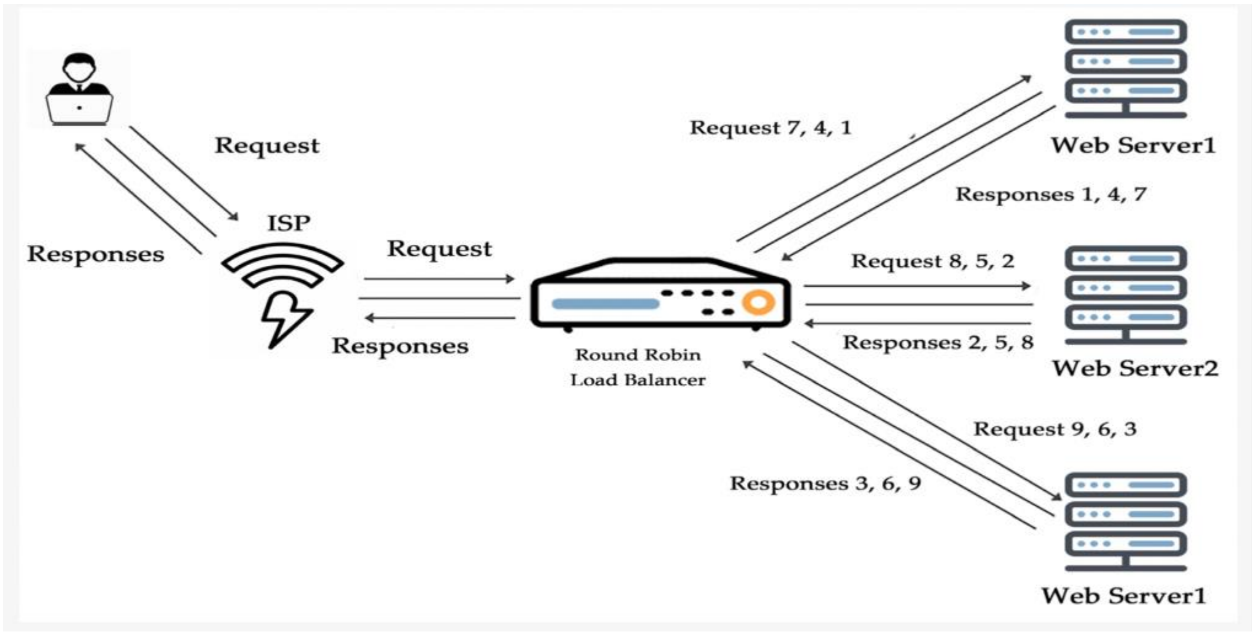


XIX. **Weighted least connection**

4. **Round-robin algorithm:**

This is a widely used load balancing strategy that works best when all the available servers have similar configurations, computational and storage capacities. Assigning the client/user requests to the available servers in a cyclical manner ensures that no one server is being overloaded while the others are being underused.

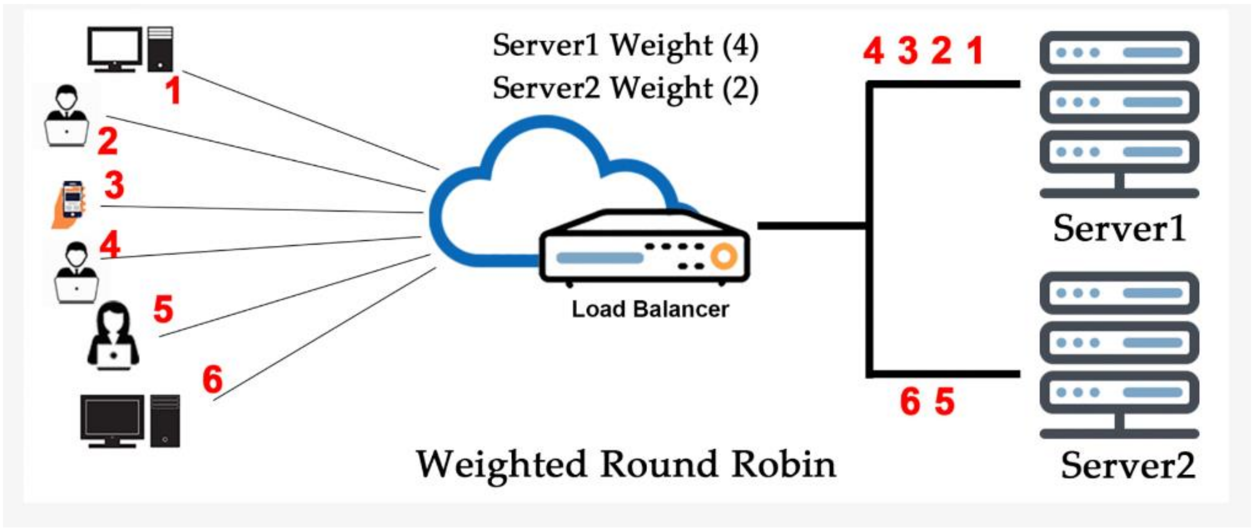
The only disadvantage when using this algorithm is that it assumes that all the servers have similar features and can manage a similar amount of loads which is not always the case. Apart from load balancing, all these algorithms also help in securing the networks and systems as they mitigate the cyber threat from occurring, keeping the systems up at all times, in turn, the firewalls.



XX. **Round-robin algorithm**

5. **Weighted Load Balancing :**

Turning the disadvantage of the Round-robin algorithm, weighted load balancing algorithm helps in sending the handler only the amount of load that they can bear. This makes sure that the application server with more weight receives more traffic and vice-versa. This enables the system to never overload. The most popular cloud service: Amazon Web Services(AWS) and the Elastic in it uses this weighted load balancing strategy.



XXI. Weighted Load Balancing

6. Team Members and responsibilities.

Team Members	Role/Responsibility
Sahithi Bommadi	Cloud Architecture & Backend Development
Vaishak Melarcode Kallampad	Frontend Development & Database Administration
Venkata Lakshmi Praneetha Moturi	Cloud Architecture & Backend Development
Pavan Karthik Gollakaram	Frontend Development & Database Administration