

# **SWENG 837 Final Project**

**Real-Time Ride-Sharing Service**

**Sowmya Talasila**

**SWENG 837**

**Santosh Nalubandhu**

## Table of Contents

Title	Page 1
Table of Contents	Page 2
Problem Statement and Requirements	Pages 3-4
UML Use Case Diagram	Page 5
UML Domain Model Diagram	Page 6
UML Class Diagram	Page 7
UML Sequence Diagram	Page 8
UML State Diagram	Page 9
UML Activity Diagram	Page 10
UML Component Diagram	Page 11
Cloud Deployment Diagram	Page 12
Skeleton Classes and Table Definition	Page 13-15
Design Pattern	Page 16

# Problem Statement and Requirements

## Problem statement:

- The main problem the system aims to solve is efficiently connecting passengers to the available drivers to get them to different places within a ride-sharing ecosystem. This is ensuring everyone is being picked up in a timely manner, implementing a dynamic pricing system that responds to fluctuating supply and demand conditions, and optimizing routes to save time for both customers and passengers.

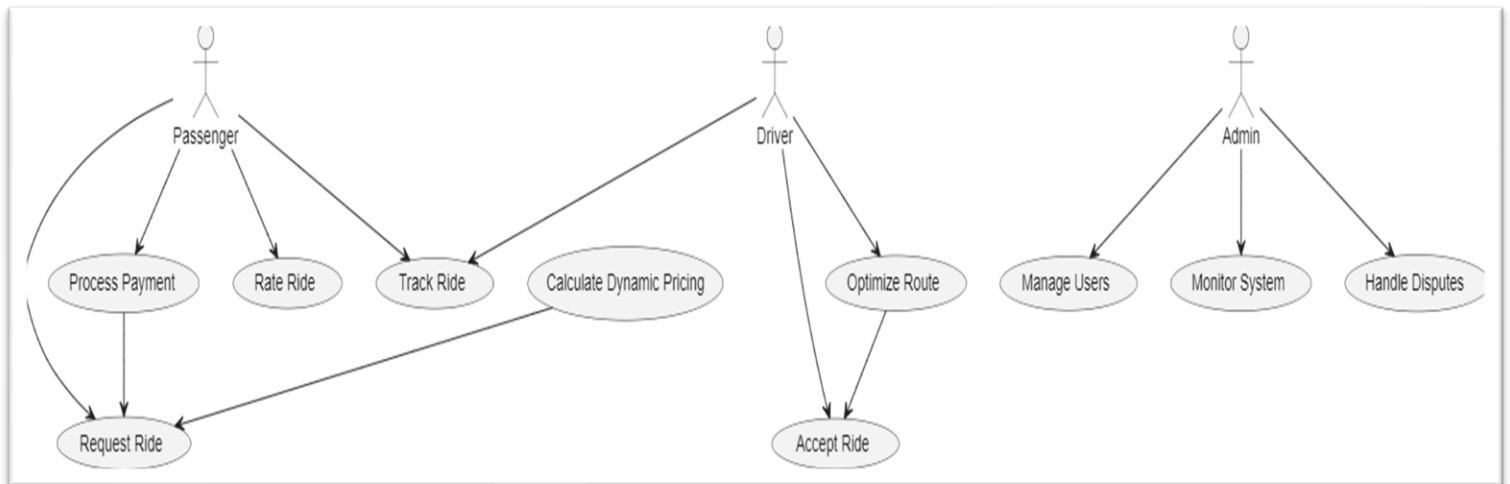
## Business and Non-Functional Requirements:

- **Business Goals:**
  - Customer Retention: All the target users should be satisfied with the system
  - Revenue Growth: The system should be able to generate profit through optimizing pricing models
  - Market Expansion: Scales the system so it fits the needs of new geographical areas and user bases.
- **Target Users:**
  - Customers who need reliable transportation between places at fair prices.
  - Drivers who need regular ride request at an optimized route with fair compensation price.
  - Administrators who need tools for managing the system and dealing with any disputes.
- **Functional Requirements:**
  - The system should allow user to create and manage their profiles
  - It should display the drivers within a 10-mile radius so the customers can see their options and pick the one that suites their needs
  - The system should have an option that lets users pick if they want to share the ride with someone or not.
  - The system should let passengers book their rides using their current location and their destination
  - The system should have an option where we can tract the driver's location to see how far they are from the user's current locations
  - The system should offer optimized routes which take traffic, road closures, and accidents into consideration.
  - The system should dynamically calculate ride sharing costs which reflect the current market trends

- The system should have the capability that lets users pay through the app by using a credit card. This should also consider fair splitting options
- The system should have the capability where the customer can leave reviews of their rides.
- The system should also let the target users contact customer support if they have any problems they want to report.
- **Non-Functional Requirements:**
  - Performance Requirements:
    - The system should scale to handle the increasing number of users and demands.
    - The system should be able to handle 1000s of requests per minute especially during peak times.
    - The response time of the should be less than 2 seconds.
  - Maintainability Requirements:
    - The system should have a modular architecture to simplify updated, big fixed, and feature additions.
    - The system should have comprehensive documentation of everything for easy maintenance.
    - The system should have automated testing and CI/CD pipeline to ensure the reliability of code changes.
  - Security Requirements:
    - The system should have mechanisms that verify the identities of all users.
    - The system should make sure all the sensitive data is protected from unauthorized access.
    - The system should have mechanisms that detect fraudulent activities.
    - The system should ensure that the users have only access to the things based on their role.
  - Other Requirements:
    - The system should be user friendly making sure everything is easy for the users to find.
    - The system should have a high availability rate of 99.99% uptime to make sure the users can use it without any problems.
    - The system should be compliant with all the regulatory requirements.

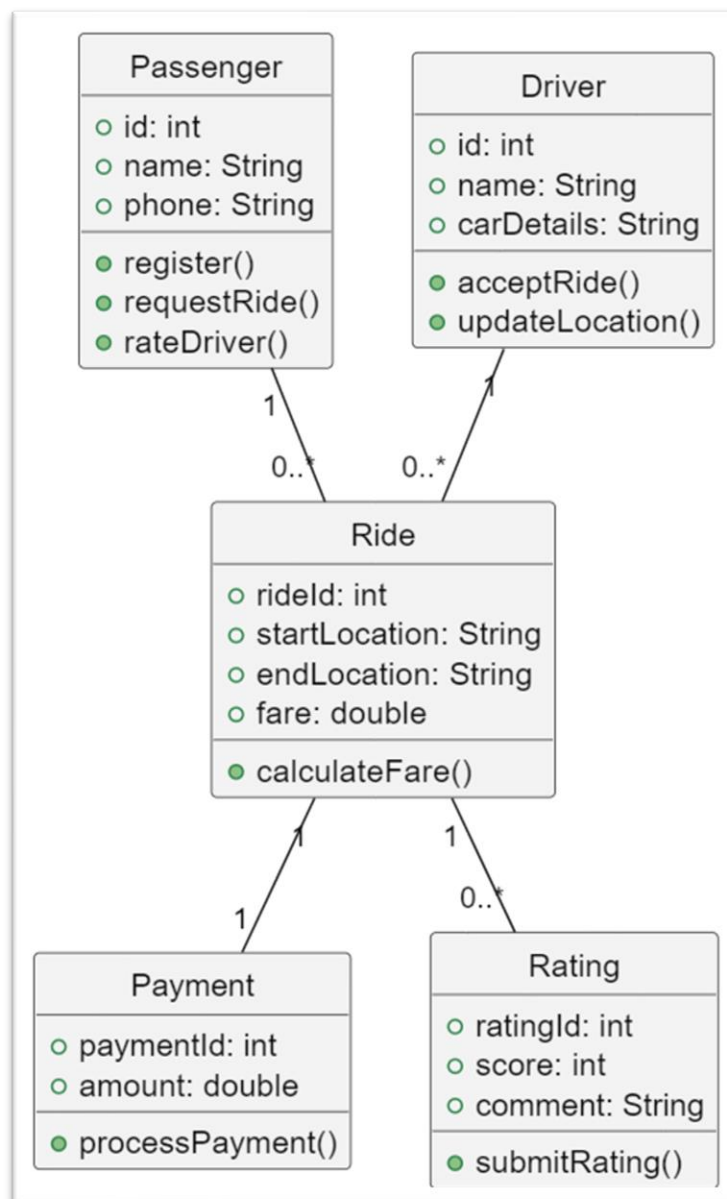
## UML Use Case Diagram

This diagram helps us identify the interactions between all the users. The user can do actions like request a ride, rate one, track the ride, and process the payment after. The driver can track the ride as well and they can accept a ride and get an optimized route. The admin can monitor the system and manage users and any disputes they submit.



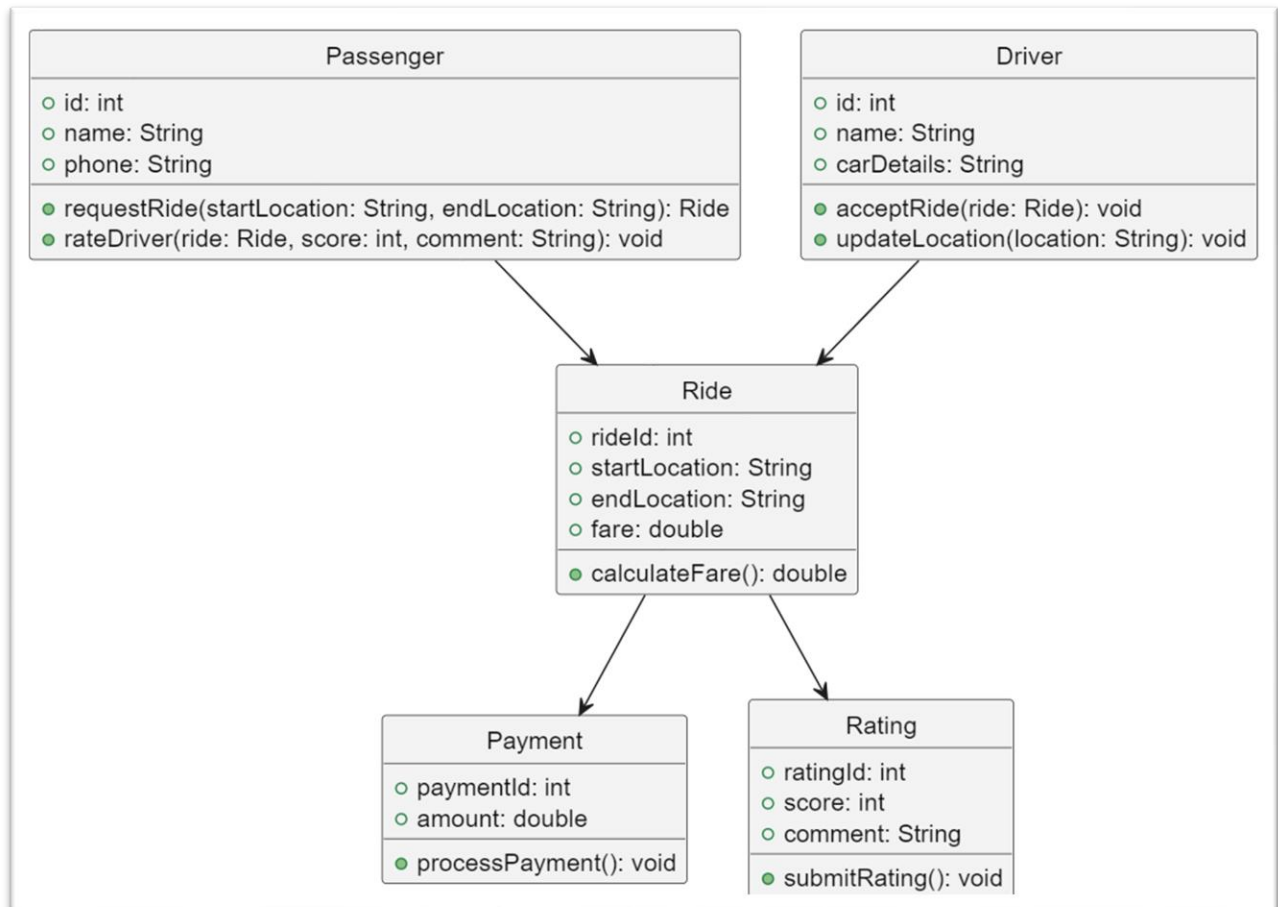
## UML Domain Model

The domain model shows the classes and the relationships between them. In this diagram there is a one-to-many relationship between passenger and ride, driver and ride, and ride and rating which means one attribute can belong to multiple classes at a time. There is a one-to-one relationship between ride and payment because one ride can only have one payment attached to it



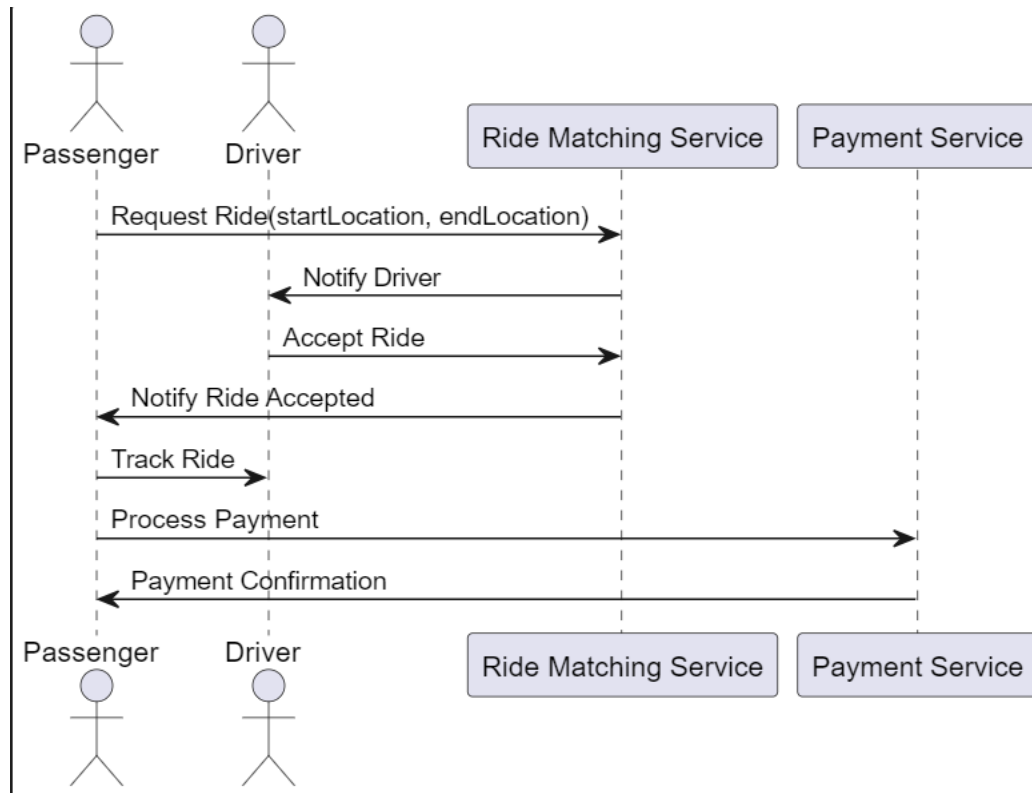
## UML Class Diagram

The class diagram shows how multiple classes work together to make the system functional. In this case the passenger requests a ride, the driver accepts it, once the ride is done the ride can calculate the fare and then the user can make the payment and submit the rating.



## UML Sequence Diagram

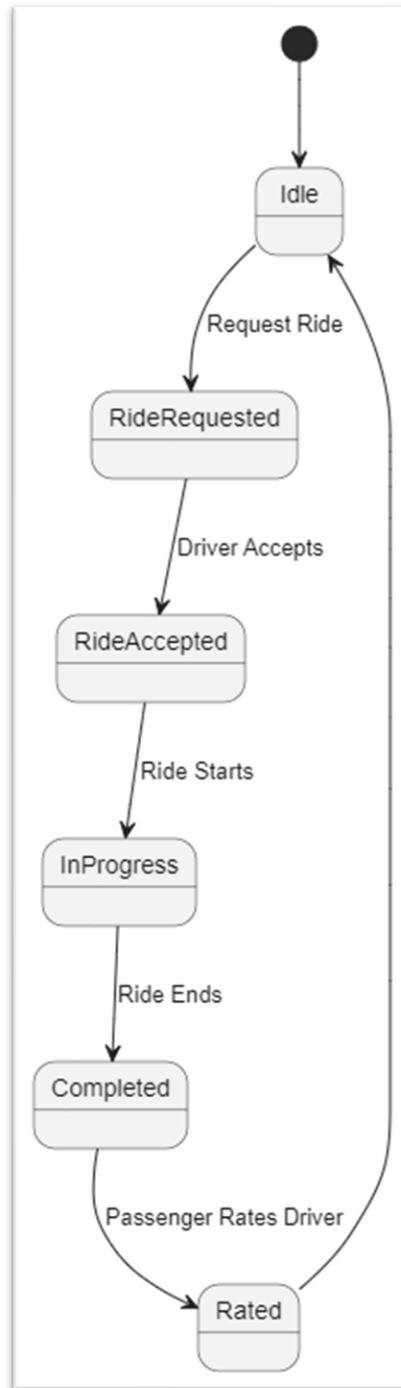
The sequence diagram shows the exact step by step process of what happens when the user requests a ride. The information above the arrow in this diagram shows this information and where the arrow is pointing towards is the stage of the system.





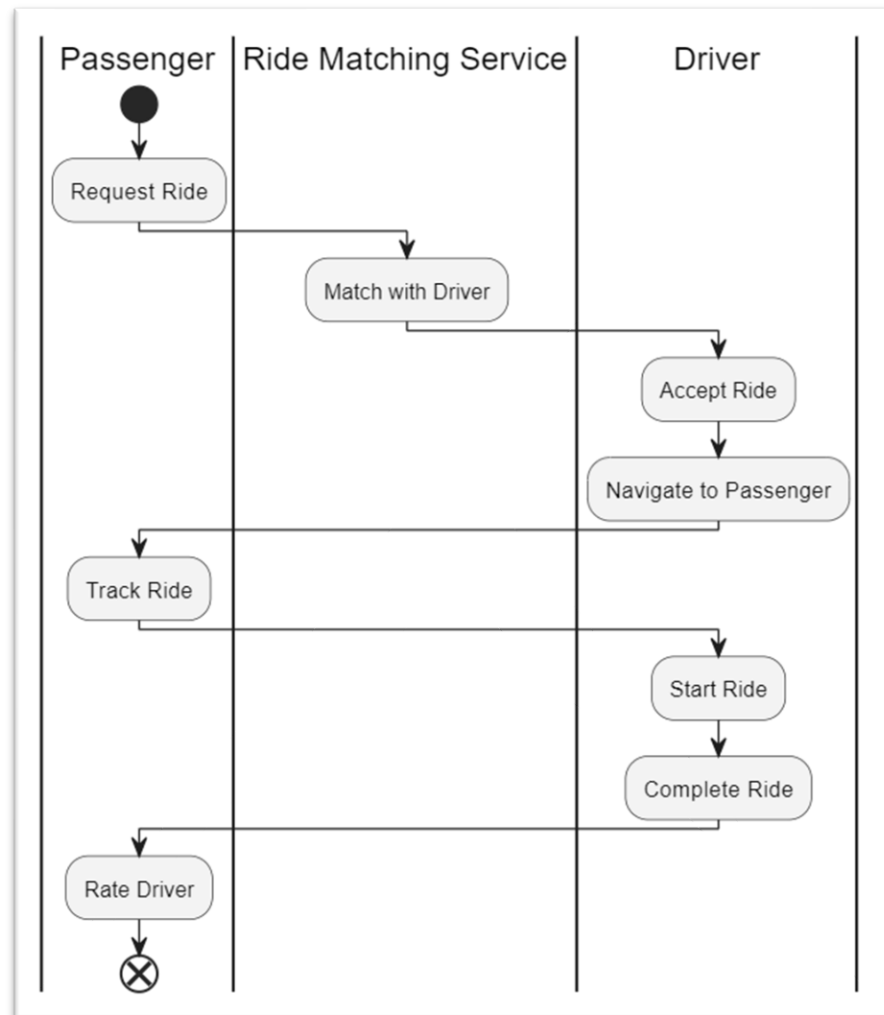
## UML State Diagram

The state diagram shows the state of the system and which level each step is at and it also shows which actions led to that state.



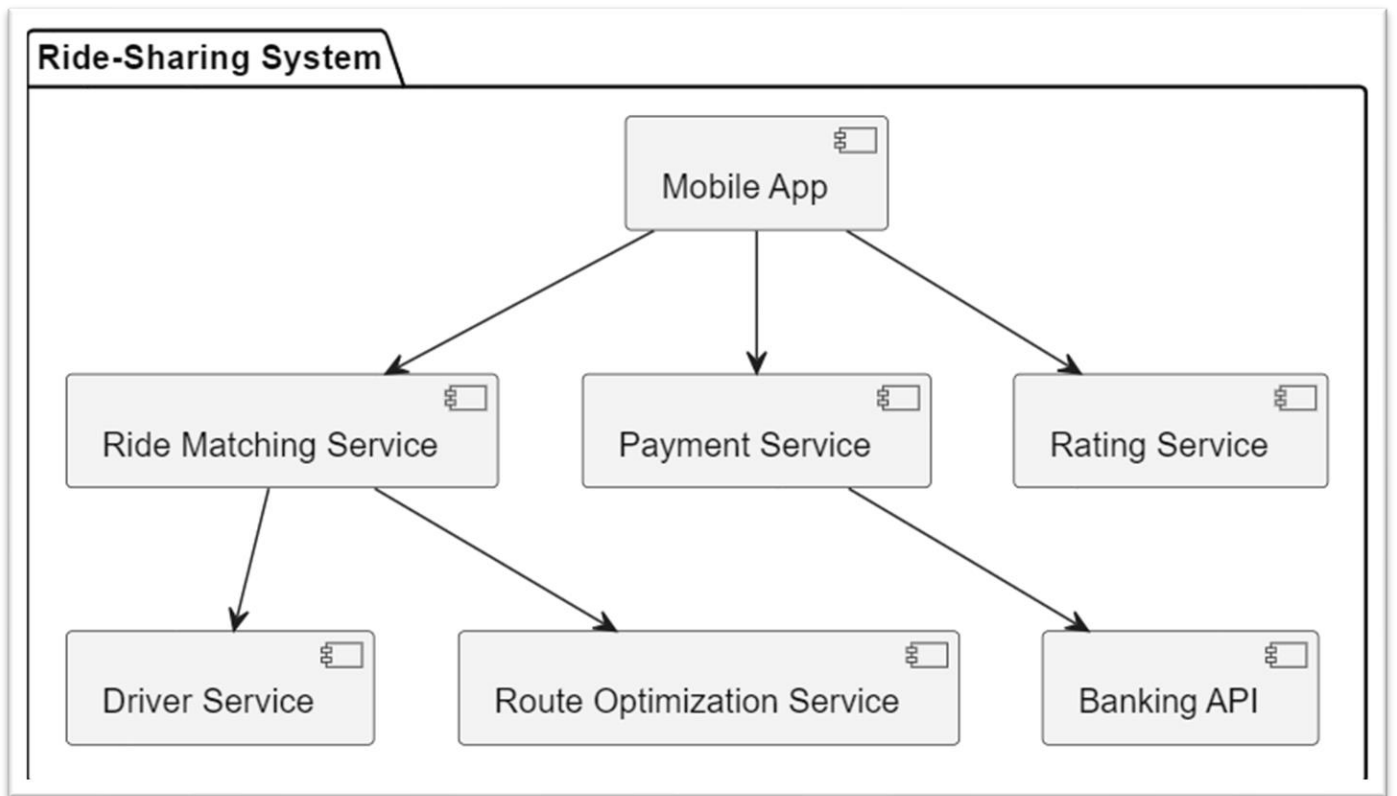
## UML Activity Diagram

The activity diagram shows all the steps taken by the user in a chronological order.



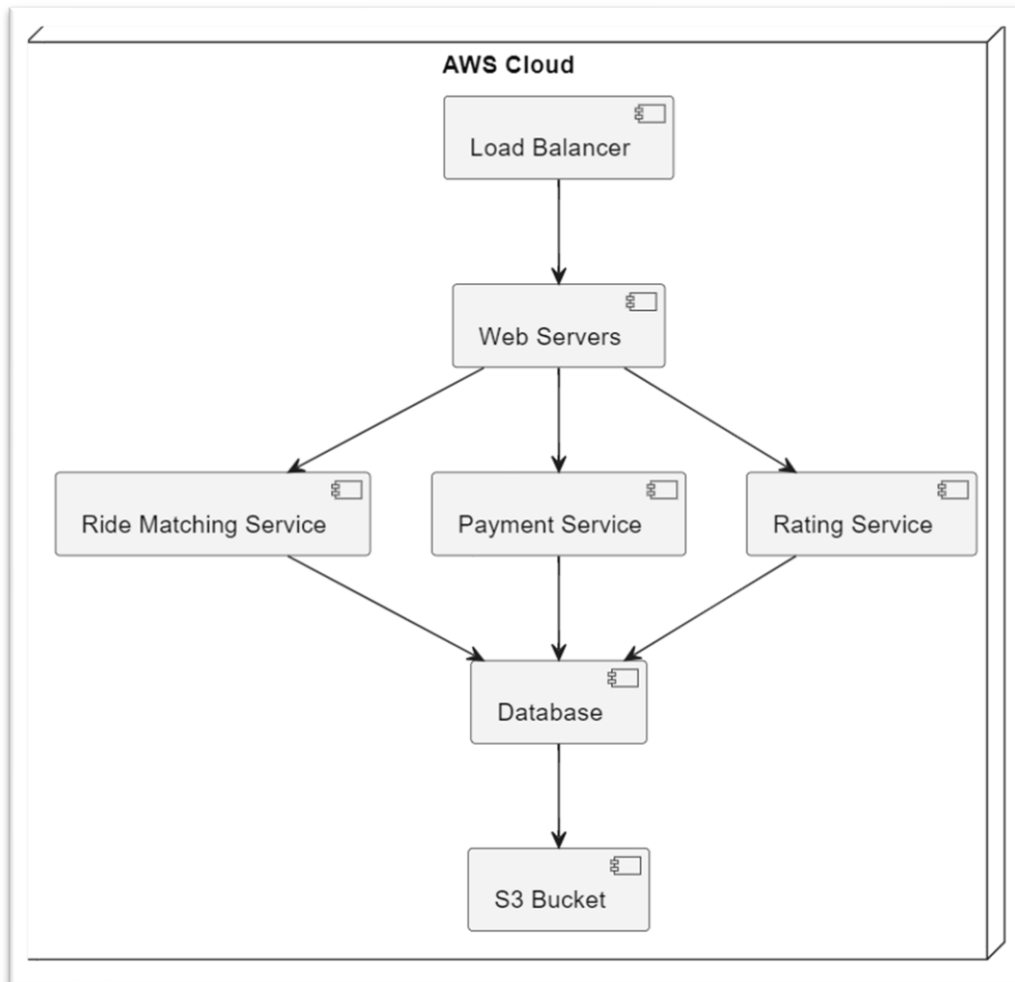
## UML Component Diagram

The component diagram shows all the major components needed for the system to function appropriately.



## Cloud Deployment Diagram

The deployment diagram shows how the system is divided into smaller, individual services



## Skeleton Classes and Table Definition

- User
  - Attributes:
    - user\_id: Integer
    - name: String
    - email: String
    - password\_hash: String
    - phone\_number: String
    - address: String
    - user\_type: Enum (Passenger, Driver)
  - Methods:
    - register()
    - login()
    - updateProfile()
    - viewProfile()
    - deleteAccount()
- Ride
  - Attributes:
    - ride\_id: Integer
    - passenger\_id: Integer (foreign key to User)
    - driver\_id: Integer (foreign key to User)
    - pickup\_location: String
    - dropoff\_location: String
    - status: Enum (Pending, Accepted, In-Progress, Completed, Canceled)
    - fare: Float
    - timestamp: DateTime

- Methods:
  - requestRide()
  - updateRideStatus()
  - calculateFare()
  - cancelRide()
  - completeRide()
- Driver
  - Attributes:
    - driver\_id: Integer (foreign key to User)
    - vehicle\_type: String
    - license\_plate: String
    - rating: Float
    - availability\_status: Enum (Available, Unavailable)
  - Methods:
    - updateAvailability()
    - viewCurrentRides()
    - updateRating()
- RideRequest
  - Attributes:
    - request\_id: Integer
    - passenger\_id: Integer (foreign key to User)
    - pickup\_location: String
    - dropoff\_location: String
    - request\_time: DateTime
    - status: Enum (Pending, Accepted, Rejected)
  - Methods:

- `createRequest()`
- `updateRequestStatus()`

## **Design Patterns**

Some of the principles I have used for this are:

- Single Responsibility from SOLID. This means each class has its own responsibility
- Interface Segregation Rule which means each user will have their own interface
- Dependency Inversion Rule which means high level modules don't depend on low level modules.
- Strategy Pattern from GOF will be applied for dynamic pricing. We can use various strategies to implement this.
- It also uses API gateways and Service Separations which means each functionality has its own service and the gateway handles all the requests and routes them to appropriate service.