MyUber Ride-Sharing Platform: Detailed Report

# 1. Introduction

The MyUber Ride-Sharing Platform is designed to be a secure, distributed system that enables riders to request rides and drivers to accept, reject, or complete these ride requests. The system leverages gRPC for high-performance communication and implements SSL/TLS with mutual authentication (mTLS) to ensure secure communication between riders, drivers, and the server. The platform handles ride assignments, timeout and rejection scenarios, driver availability, and client-side load balancing across multiple servers for scalability and fault tolerance.  
This report details the implementation of the system, including API design, security features, interceptors, load balancing mechanisms, and timeout handling.

# 2. Architecture Design

## 2.1 System Overview

The MyUber platform consists of:  
- Riders: Clients who request rides by providing their pickup and destination locations.  
- Drivers: Clients who accept or reject ride requests based on availability and status.  
- gRPC Servers: Multiple distributed servers that handle ride requests, load balance across servers, and communicate securely with riders and drivers using SSL/TLS.  
- SSL/TLS: Provides encryption and authentication, ensuring that only valid riders and drivers can participate.

## 2.2 Communication Flow

1. Rider requests a ride: The rider sends a ride request with their pickup and destination location.  
2. Driver accepts/rejects the ride: The server assigns the ride to an available driver. The driver either accepts or rejects the request within a specified timeout.  
3. Ride reassignment: If the driver does not respond or explicitly rejects the request, the system automatically reassigns the ride to another available driver.  
4. Ride completion: Once the driver completes the ride, the ride status is updated, and the driver becomes available for new requests.

## 2.3 Server Architecture

The platform uses a multi-server setup with gRPC clients connecting to multiple backend servers. Requests are load balanced across available servers using a round-robin algorithm, ensuring even distribution of ride requests.

# 3. API Design

## 3.1 Rider Services

• RequestRide(RideRequest) → RideResponse:  
Riders submit a ride request by providing their pickup location and destination.  
  
RideRequest:  
```  
message RideRequest {  
 string pickup\_location = 1;  
 string destination = 2;  
}  
  
message RideResponse {  
 string status = 1;  
 string driver\_id = 2;  
}  
```  
  
• GetRideStatus(RideStatusRequest) → RideStatusResponse:  
Riders can check the status of their requested or ongoing rides.  
  
RideStatusRequest and RideStatusResponse:  
```  
message RideStatusRequest {  
 string ride\_id = 1;  
}  
  
message RideStatusResponse {  
 string status = 1; // e.g., ASSIGNED, IN\_PROGRESS, COMPLETED, CANCELED  
}  
```

## 3.2 Driver Services

• AcceptRide(AcceptRideRequest) → AcceptRideResponse:  
Drivers can accept a ride request assigned to them. The system waits for a response within a timeout period.  
  
AcceptRideRequest and AcceptRideResponse:  
```  
message AcceptRideRequest {  
 string ride\_id = 1;  
 string driver\_id = 2;  
}  
  
message AcceptRideResponse {  
 string status = 1; // e.g., ACCEPTED, TIMEOUT  
}  
```  
  
• RejectRide(RejectRideRequest) → RejectRideResponse:  
Drivers can explicitly reject a ride if they are unavailable or unwilling to accept it. The ride is automatically reassigned.  
  
RejectRideRequest and RejectRideResponse:  
```  
message RejectRideRequest {  
 string ride\_id = 1;  
 string driver\_id = 2;  
}  
  
message RejectRideResponse {  
 string status = 1; // e.g., REJECTED  
}  
```  
  
• CompleteRide(RideCompletionRequest) → RideCompletionResponse:  
Drivers can mark the ride as complete after dropping off the rider.  
  
RideCompletionRequest and RideCompletionResponse:  
```  
message RideCompletionRequest {  
 string ride\_id = 1;  
 string driver\_id = 2;  
}  
  
message RideCompletionResponse {  
 string status = 1; // e.g., COMPLETED  
}  
```