Synopsis Document:

Ride-Sharing App Using Spring Boot

Team Members:

- C068 Kaustub Mule
- C074 Soham Agarwal
- C095 Anuj Chandak

Introduction to the Project

In today's fast-paced world, urban commuting has become increasingly challenging due to traffic congestion, high fuel costs, and environmental concerns. Ride-sharing services provide a convenient, cost-effective, and eco-friendly solution by allowing individuals to share rides with others traveling in the same direction. This project aims to develop a ride-sharing application using Spring Boot, providing a seamless and efficient platform for users to book and share rides in real-time while addressing key limitations of existing platforms like Ola and Uber.

Problem Statement

Traditional modes of transportation often lead to increased traffic congestion, pollution, and higher commuting costs. Existing ride-sharing platforms may have limitations such as high service charges, lack of transparency, inadequate route optimization, and driver dissatisfaction. This project seeks to develop a user-friendly, scalable, and secure ride-sharing application that enhances the ride-booking experience while minimizing operational inefficiencies and introducing unique features.

Objectives

1. Develop a robust and scalable ride-sharing application using Spring Boot.
2. Implement user authentication and authorization for secure access.
3. Enable real-time ride booking, matching, and route optimization using AI.
4. Provide seamless payment integration and fare negotiation between drivers and riders.
5. Introduce a decentralized, community-driven model to empower drivers.
6. Implement a rating and feedback system for service improvement.
7. Ensure data security and user privacy through encrypted communications.
8. Utilize Spring Cloud for microservices architecture and seamless API communication.
9. Deploy the application using Docker for easy scalability and containerization.

Scope of the Project

- **User Roles:**
    - Riders: Users looking for a ride.
    - Drivers: Individuals offering ride-sharing services.
    - Admins: Manage users, rides, and app settings.

- **Core Features:**
    - User registration and authentication.
    - Ride creation, search, and booking functionality.
    - Real-time location tracking and ride matching.
    - Fare negotiation and transparent pricing model.
    - Secure payment processing and diverse payment options.
    - Ride history, trip management, and carbon footprint tracking.
    - Reviews, ratings, and gamification-based loyalty programs.
    - Private ride groups for trusted connections.
    - Microservices-based architecture using Spring Cloud for better scalability.
    - Docker-based deployment for efficient resource management and CI/CD integration.

Technologies Used

- **Backend:** Spring Boot, Spring Security, Spring Data JPA, Spring Cloud
- **Database:** MySQL
- **Frontend:** React.js
- **API Gateway and Service Discovery:** Spring Cloud Gateway, Eureka Server
- **Mapping & Navigation:** Google Maps API/OpenStreetMap
- **Containerization & Deployment:** Docker, Kubernetes

Expected Outcome

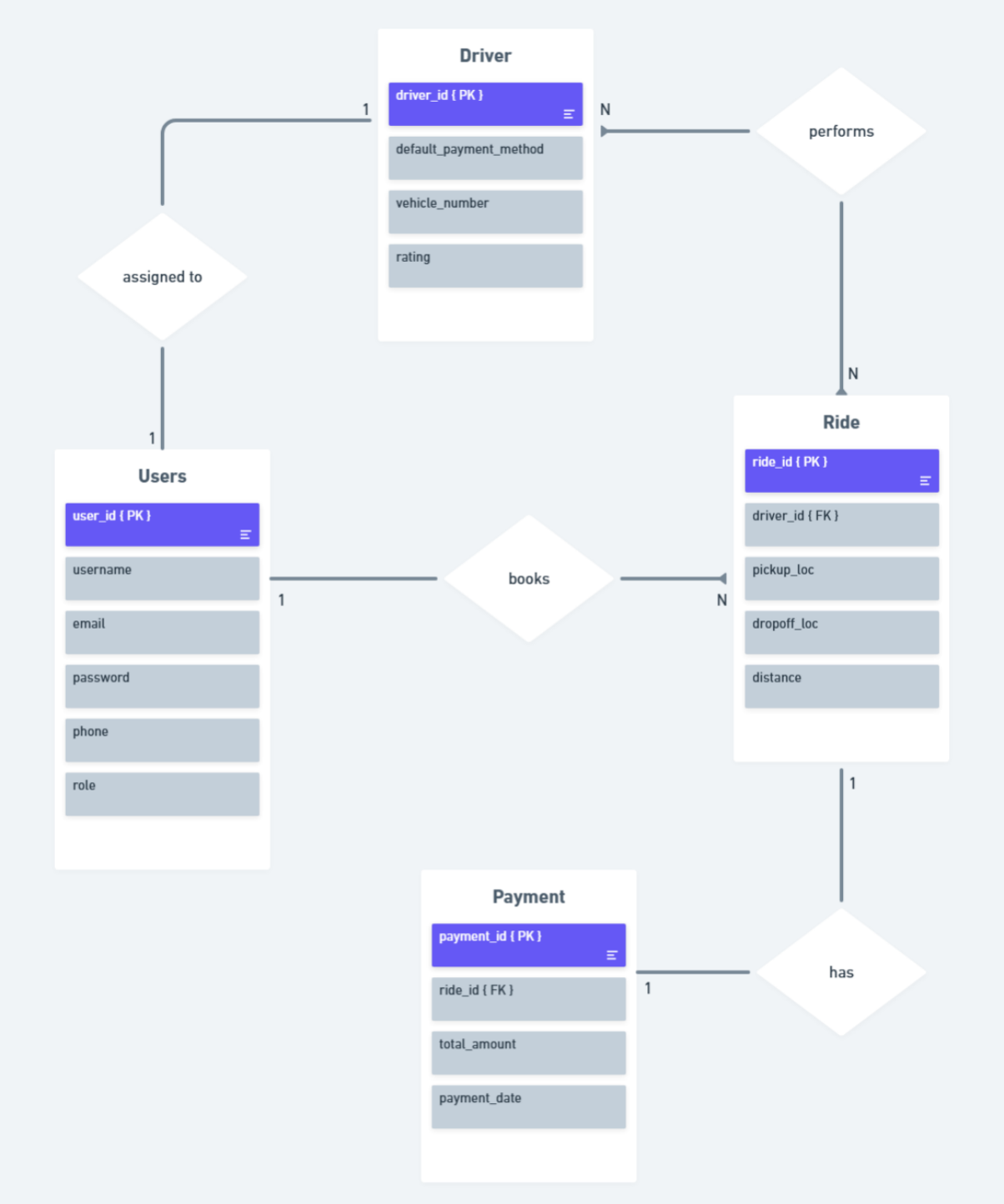
1. A fully functional ride-sharing application with a secure and scalable backend.
2. A seamless user experience with efficient ride booking, fare negotiation, and real-time tracking.
3. Reduction in traffic congestion and travel costs through optimized ride-sharing and AI-driven route selection.
4. A more driver-centric approach with lower commission fees, incentives, and better financial support.
5. A highly scalable, microservices-based architecture using Spring Cloud for better service management.
6. Efficient deployment and scalability using Docker, allowing easier updates and cloud integration.

Entities and Attributes:

1. **Users**
    - user_id (PK)
    - username
    - email
    - password
    - phone
    - role
2. **Driver**
    - driver_id (PK)
    - default_payment_method
    - vehicle_number
    - rating
3. **Ride**
    - ride_id (PK)
    - driver_id (FK)
    - pickup_loc
    - dropoff_loc
    - distance
4. **Payment**
    - payment_id (PK)
    - ride_id (FK)
    - total_amount
    - payment_date

Relationships and Cardinalities:

1. **User - Driver** (assigned to)
    - Cardinality: One-to-One (1:1)
    - A User can be assigned to exactly one Driver (1)
    - A Driver must be assigned to exactly one User (1)
    - Total participation from both sides
2. **User - Ride** (books)
    - Cardinality: One-to-Many (1:N)
    - A User can book many Rides (N)
    - A Ride must be booked by exactly one User (1)
    - Total participation from Ride side
    - Partial participation from User side
3. **Driver - Ride** (performs)
    - Cardinality: One-to-Many (1:N)
    - A Driver can perform many Rides (N)
    - A Ride must be performed by exactly one Driver (1)
    - Total participation from Ride side
    - Partial participation from Driver side
4. **Ride - Payment** (has)
    - Cardinality: One-to-One (1:1)
    - A Ride has exactly one Payment (1)
    - A Payment belongs to exactly one Ride (1)
    - Total participation from both sides



Relational Model:

user

user_id	username	email	password	phone	role

driver

driver_id	vehicle_number	rating

Driver_rating

ride_id {FK}	driver_id(FK)	payment_method

Ride

ride_id	user_id(FK)	driver_id(FK)	pickup_loc	dropoff_loc	distance

Payment

payment_id	ride_id(FK)	total_amount	payment_date

Normalization

1NF (First Normal Form):

- Users Table: Already in 1NF - atomic values, primary key exists
- Driver Table: Already in 1NF - atomic values, primary key exists
- Driver_Payment_Methods Table: Already in 1NF - atomic values, primary key exists
- Ride Table: Already in 1NF - atomic values, primary key exists
- Payment Table: Already in 1NF - atomic values, primary key exists

2NF (Second Normal Form): All tables are in 2NF as all non-key attributes are fully dependent on their primary keys with no partial dependencies.

3NF (Third Normal Form):

- Users Table: Already in 3NF
- Driver Table: Violates 3NF (rating has transitive dependency)
    - Split into: Driver (driver_id, user_id, vehicle_number) and Driver_Rating (driver_id, rating)
- Driver_Payment_Methods Table: Already in 3NF
- Ride Table: Already in 3NF
- Payment Table: Already in 3NF

BCNF: After 3NF changes, all tables are in BCNF as all determinants are candidate keys.

Final Tables:

1. Users (user_id, username, email, password, phone, role)
2. Driver (driver_id, user_id, vehicle_number)
3. Driver_Rating (driver_id, rating)
4. Driver_Payment_Methods (rider_id, driver_id, payment_method)
5. Ride (ride_id, user_id, driver_id, pickup_loc, dropoff_loc, distance)
6. Payment (payment_id, ride_id, total_amount, payment_date)