Project Deliverable 3

Date	Version	Description	Author
01/10/2023	1.0	Software Architecture Document	RideWave

Table of Contents

Introduction

Introduction

1.1 Purpose

This document outlines the architecture of the RideWave carpooling application tailored for IBA university students. It aims to provide a comprehensive overview of the system's architecture, including its key components, interactions, and technologies used.

1.2 Scope

The RideWave application is designed to address transportation challenges faced by IBA university students by facilitating carpooling among peers with similar schedules or commuting routes. This Software Architecture Document covers the high-level architectural aspects of the system, outlining its structure and key features.

1.3 Definitions, Acronyms, and Abbreviations

- UI/UX: User Interface/User Experience
- SDK: Software Development Kit
- REST: Representational State Transfer
- API: Application Programming Interface
- Firestore: Firebase's NoSQL database service
- Flutter: A cross-platform framework for mobile application development
- Firebase: A cloud-based platform for building mobile and web applications

Functional and Non-Functional Requirements

In the development of the RideWave carpooling application, a comprehensive set of functional and non-functional requirements has been established to guide the system's design and ensure its effectiveness. These requirements are pivotal in shaping the user experience, system performance, and adherence to security and privacy standards.

Functional Requirements:

- User Registration:
    - Users should be able to create accounts using their IBA email addresses and ERP IDs.
    - Registration should require necessary user information, including name, contact information, and password.
    - Users should have the ability to create, update, and manage their profiles.
    - Profiles should include information such as class schedules, geographic coordinates, vehicle details (for drivers), and preferences.
- Matching Algorithm:
    - Implement an intelligent matching algorithm that considers factors like location, time, and user preferences.
    - The algorithm should connect students sharing comparable schedules and routes for carpooling.
- Ride Listing and Requests:
    - Drivers should be able to create ride listings specifying their travel plans, including the origin, destination, date, and time.
    - Riders should be able to view a list of rides (with time, capacity, gender, etc.) available when they open the app.
    - Rider should be able to send ride requests from the list, and drivers should have the option to accept or decline them.
- Filtering and Search:
    - Users should have the ability to filter and search for potential carpooling partners based on criteria such as vehicle type, seating capacity, air-conditioning availability, and gender preferences (for safety and comfort).
- Reviews:
    - The system should allow users to leave reviews and ratings for fellow carpoolers.

Non-Functional Requirements:

- Responsiveness:
    - The system should provide quick response times for user interactions, even during peak usage.
- Scalability:

- It should handle a substantial number of concurrent users without significant performance degradation.

❖ Security:

- User data, including personal information and ride history, must be securely stored and transmitted.
- Implement proper authentication and authorization mechanisms to protect user accounts and data.

❖ Usability:

- The user interface should be intuitive and user-friendly to ensure that users can easily navigate the application.
- Accessibility considerations should be taken into account for a diverse user base.

❖ Reliability:

- The application should have a high level of uptime and availability.

❖ Data Backup:

- Data backup and recovery mechanisms should be in place to prevent data loss.

❖ Privacy:

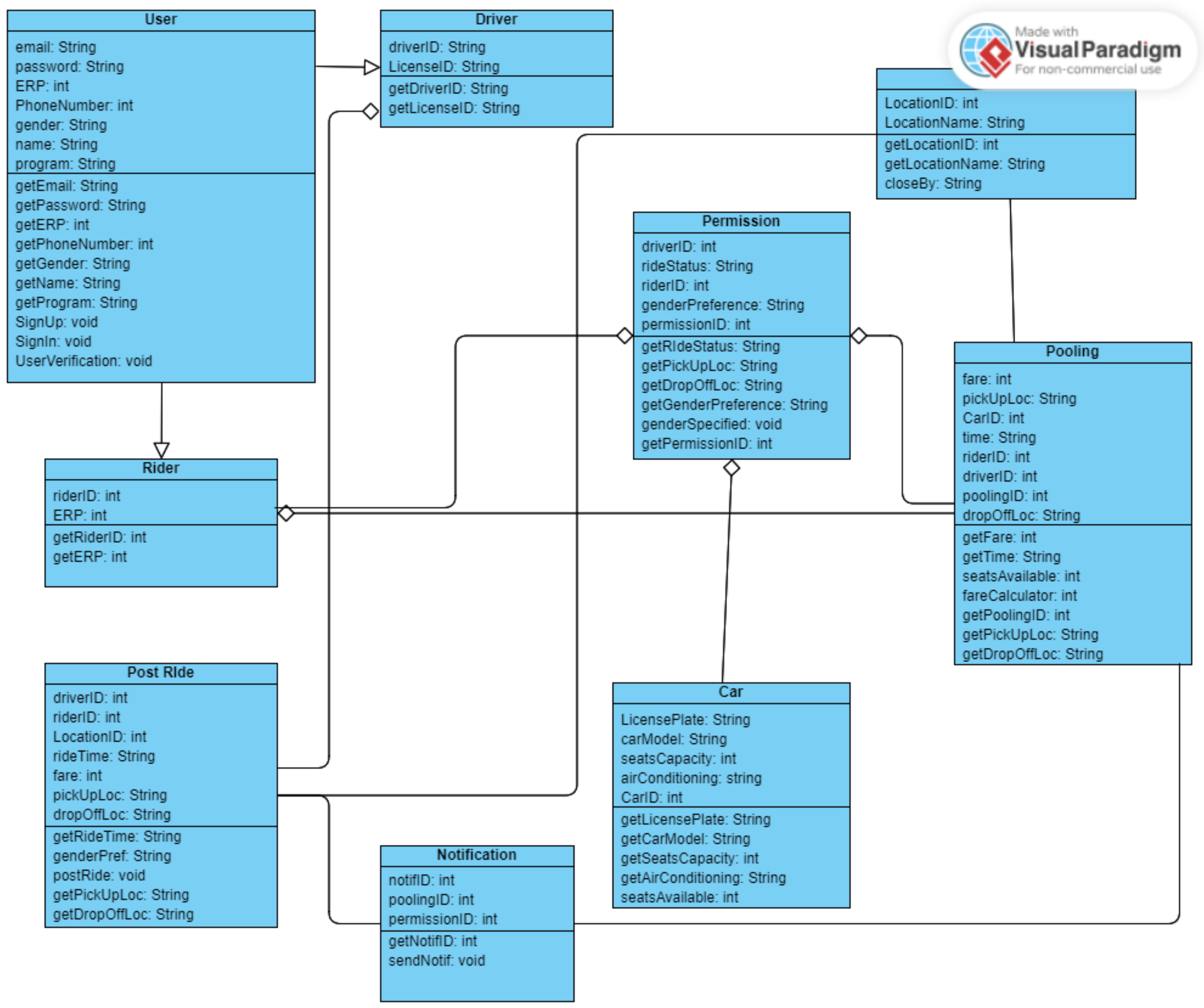
- User privacy should be respected, and data should only be used for the intended purpose (carpooling).

❖ Regulatory Compliance:

- Ensure compliance with relevant data protection and privacy regulations, such as GDPR or local laws.

All Use Cases

UML Diagram:



Explanation:

<u>Class Descriptions</u>

1. User Class

The User class serves as a fundamental entity within the RideWave application, representing individuals who use the platform. It encompasses user-specific information such as their name, contact details, and authentication credentials. This class is the basis for both drivers and riders, providing a common structure for managing user profiles and interactions.

2. Driver Class:

The Driver class represents users who offer rides within the RideWave carpooling system. It extends the User class and includes additional information related to driving and preferences. Drivers utilize this class to eventually create and manage ride listings, connecting with riders seeking transportation.

3. Rider Class:

The Rider class represents users who are looking for rides within the RideWave application. Like the Driver class, it extends the User class but emphasizes preferences from the rider's perspective. Riders use this class to eventually request rides from available drivers and to manage their carpooling experiences.

4. Notification Class:

The Notification class handles communication and updates within the RideWave system. It encompasses notifications sent to users regarding ride requests, accepted rides, and messages from other users. This class ensures that users stay informed and can efficiently coordinate their carpooling activities.

5. Location Class:

The Location class represents locations within the RideWave application. It's used to eventually pinpoint the starting and ending locations of rides, allowing users to specify where they want to be picked up and dropped off. This class is essential for the matching algorithm and the efficient organization of rides.

6. Car Class:

The Car class provides a structured representation of vehicles associated with drivers within the RideWave system. It includes details such as the model, and seating capacity of the vehicle. This class helps users assess the suitability of rides offered by drivers, considering factors like comfort and capacity.

7. Pooling Class:

The Pooling class represents a carpooling request made by a rider within the RideWave application. It encapsulates the rider's preferences, such as the desired departure time and location, as well as any specific requirements or considerations. This class plays a vital role in matching riders with compatible drivers and facilitating efficient carpooling arrangements.

8. Post Ride Class:

The PostRide class represents a ride posting within the RideWave carpooling application. It encapsulates information about a ride, including its origin, destination, date, time, and any additional details provided by the driver. Instances of this class allow drivers to offer rides to other users within the system, making it a central component for organizing and coordinating carpools.

Relationships between Classes

1. User to Rider and Driver *(Association):*

There is a simple association represented by a non-colored arrow going from User to both Rider and Driver. This signifies that a user can have associations with both the Rider and Driver classes. In other words, a user can be either a rider or a driver, or even both.

2. PostRide to Driver *(Aggregation):*

The diamond arrow going from PostRide to Driver indicates an aggregation relationship. This suggests that a PostRide "has" a Driver, meaning that a ride posting is associated with a specific driver. It implies that a PostRide is composed of a Driver object.

3. Rider to Permission *(Composition):*

The diamond arrow going from Rider to Permission represents a composition relationship. This indicates that a Rider is composed of a Permission object, signifying that permissions are an integral part of a rider's profile.

4. Pooling to Rider *(Aggregation):*

The diamond arrow going from Pooling to Rider signifies an aggregation relationship, indicating that a pooling request is associated with a specific rider. It implies that a Pooling request is composed of a Rider object.

5. PostRide to Location *(Association):*

The line without an arrow going from PostRide to Location represents a simple association between a ride posting and a location. It signifies that a PostRide includes information about the ride's origin and destination, which are represented by Location objects.

6. Location and Pooling *(Association):*

The single line with no arrow going from Location to Pooling indicates a simple association between the location and pooling. It suggests that the Pooling class uses or references Location information when specifying ride preferences or requirements.

7. Pooling to Permission *(Composition):*

The diamond arrow going from Pooling to Permission signifies a composition relationship. This implies that a Pooling request is composed of a Permission object, indicating that permissions play a crucial role in defining ride preferences and requirements.

8. Car to Permission *(Composition):*

The diamond arrow going from Car to Permission represents a composition relationship, indicating that a Car is composed of a Permission object. This suggests that permissions are essential in defining the characteristics and accessibility of a car.

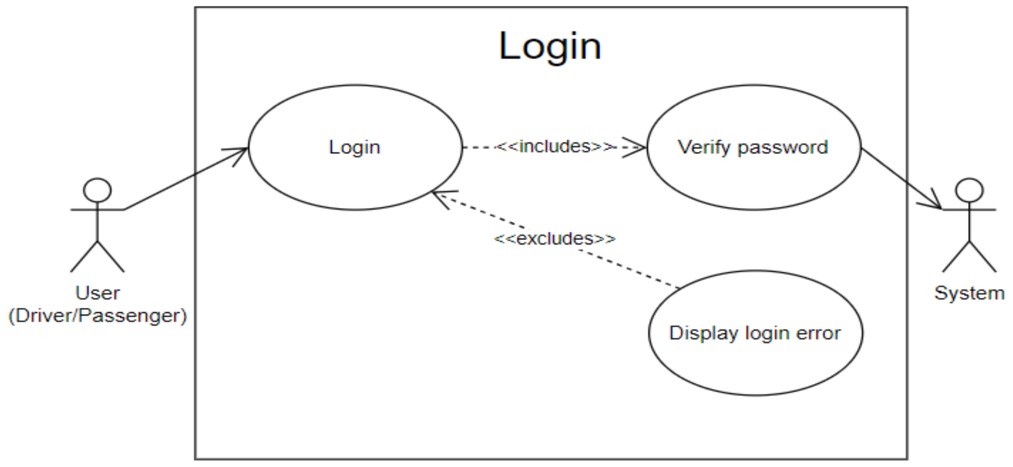
9. Notification to PostRide and Pooling *(Association):*

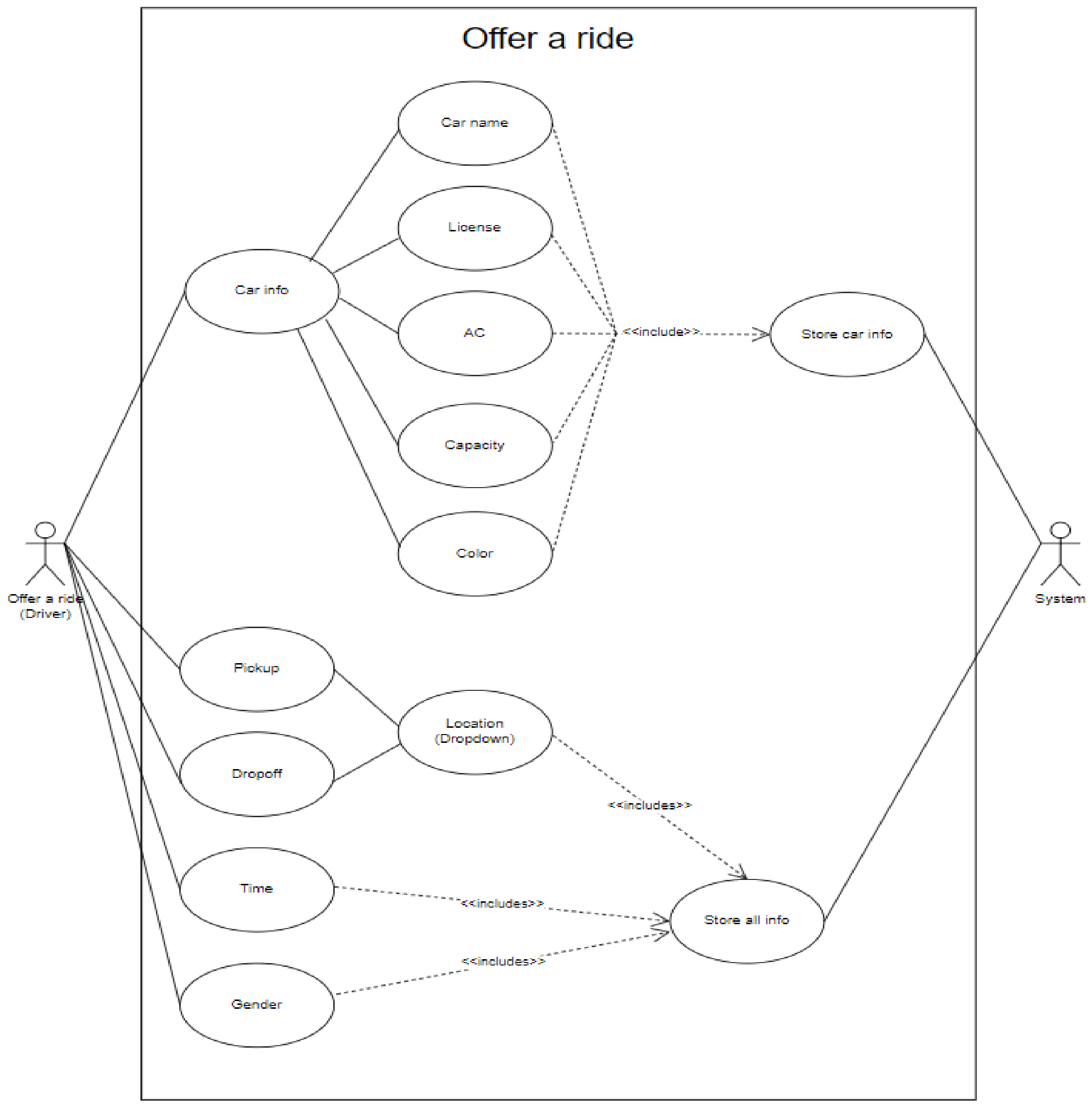
The non-arrow lines connecting Notification to PostRide and Pooling represent simple associations, indicating that notifications can be associated with both ride postings and pooling requests. This allows notifications to be linked to specific events related to rides and pooling.

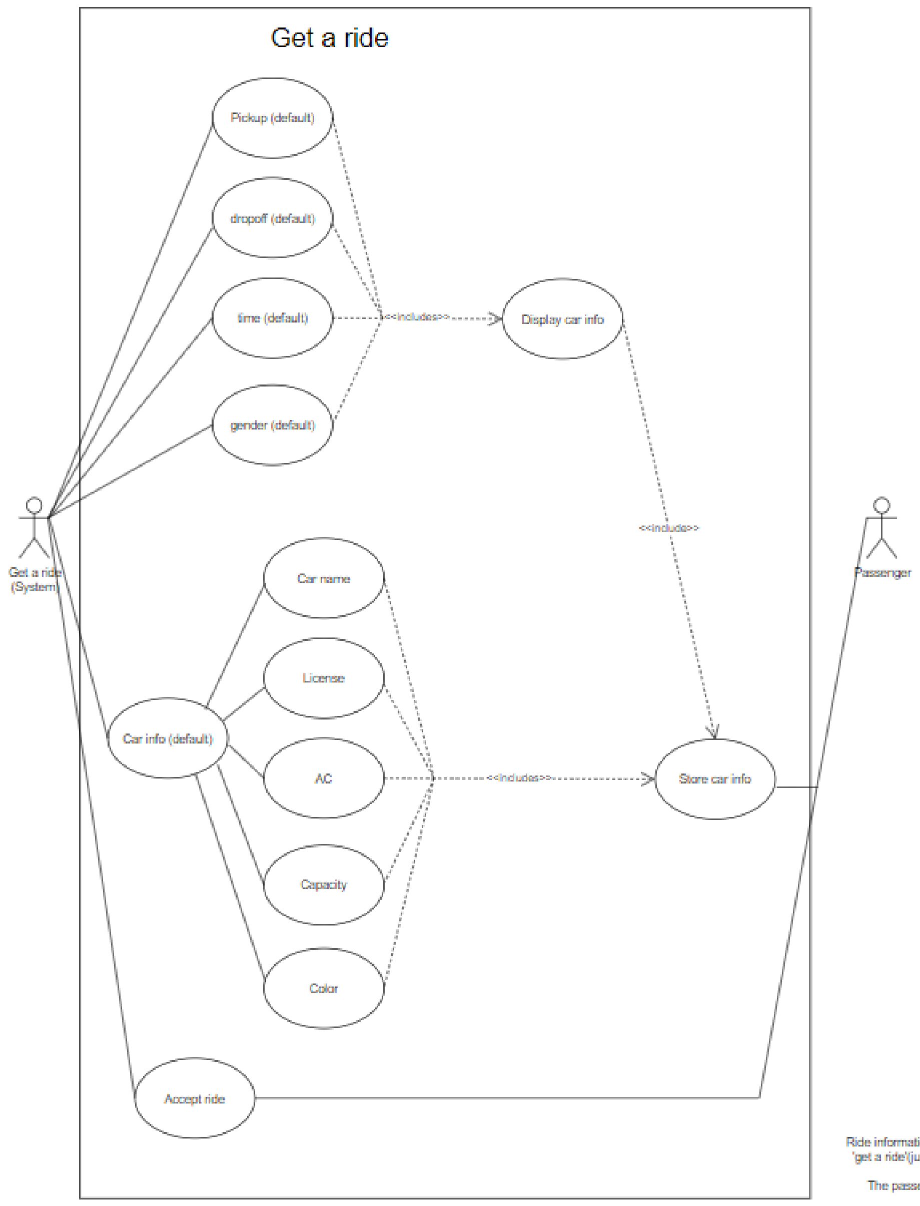
10. Pooling to Rider *(Aggregation):*

The diamond arrow going from Pooling to Rider signifies an aggregation relationship, indicating that a Pooling request is associated with a specific rider. It implies that a Pooling request is composed of a Rider object, representing the rider's preferences and requirements.

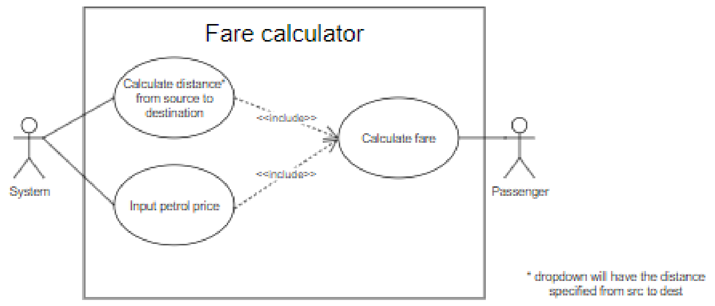
<u>All Class Diagrams</u>

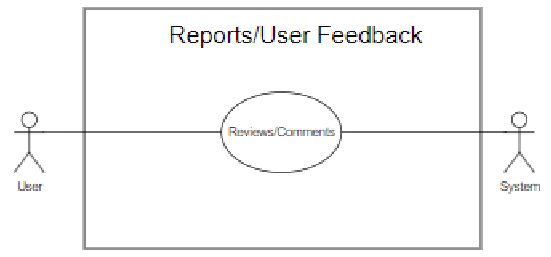



Offer a ride
Car name
License
Car info
AC
<<include>>
Store car info
Capacity
Color
Offer a ride (Driver)
System
Pickup
Location (Dropdown)
Dropoff
<<includes>>
Time
<<includes>>
Store all info
<<includes>>
Gender

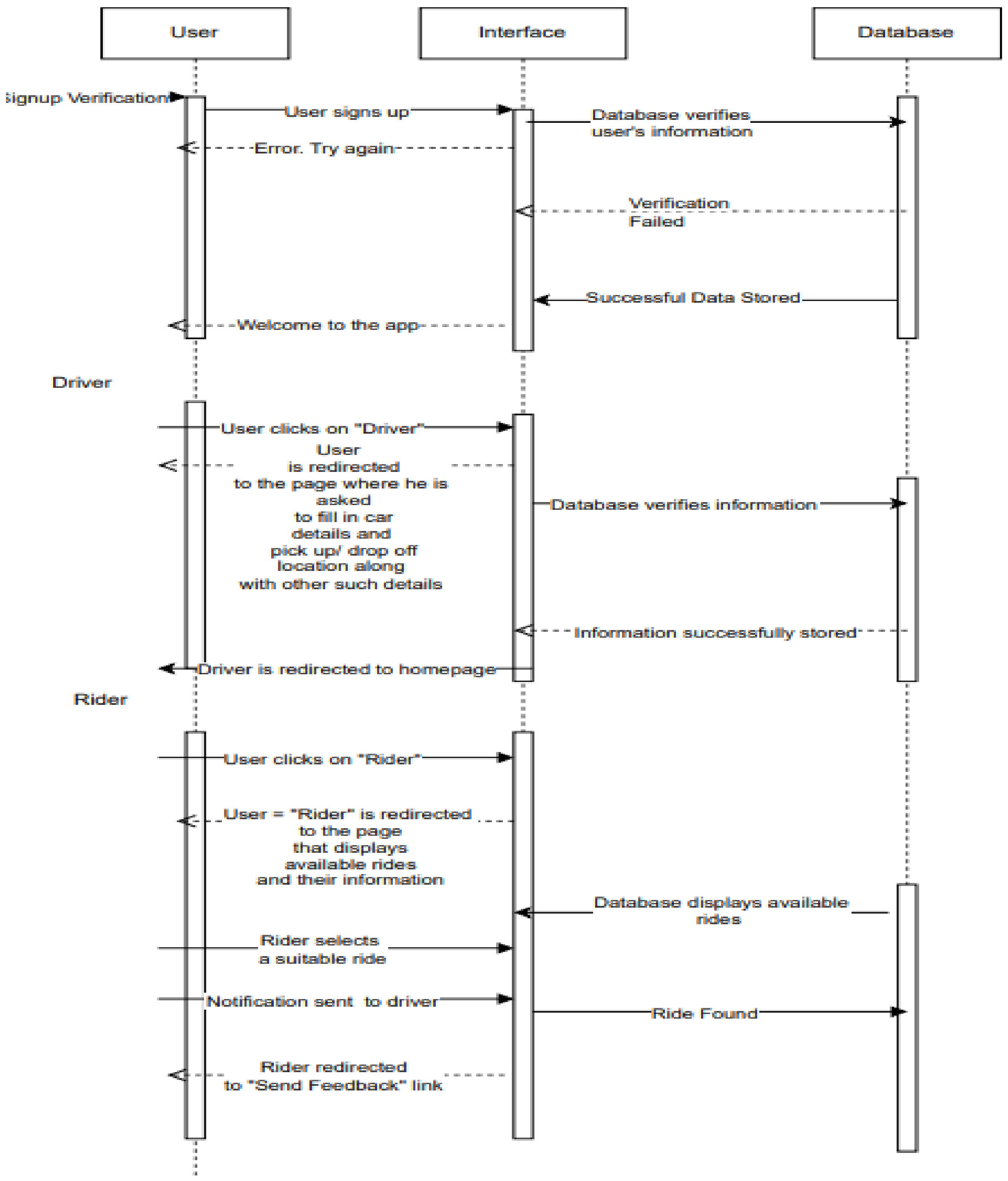


Ride information will be displayed when you click on 'get a ride'(just like in SWVL) by the system to the passenger.
The passenger can accept the suitable ride.


Fare calculator
Calculate distance* from source to destination
Input petrol price
Calculate fare
<<include>>
<<include>>
System
Passenger
* dropdown will have the distance specified from src to dest



<u>All Sequence Diagrams</u>


User
Interface
Database
Signup Verification
User signs up
Database verifies user's information
Error. Try again
Verification Failed
Successful Data Stored
Welcome to the app
Driver
User clicks on "Driver"
User is redirected to the page where he is asked to fill in car details and pick up/ drop off location along with other such details
Database verifies information
Information successfully stored
Driver is redirected to homepage
Rider
User clicks on "Rider"
User = "Rider" is redirected to the page that displays available rides and their information
Database displays available rides
Rider selects a suitable ride
Notification sent to driver
Ride Found
Rider redirected to "Send Feedback" link