Introduction

A ride-sharing application that enables riders to request rides and drivers to accept and complete them.

Team members

name	**BO HUANG**	**Kaiwen QIU**	**Lingrui Liang**	**Lisong Xiao**
student id	1584795	1595215	1522316	1652294
Unimelb username	BOHAUNG1	kaiwenq	LingLiang	lisongx
GitHub username	wffb	hylx4444	llrznhy	EvnyaGH
email	huang.b3@student.unimelb.edu.au	kaiwenq@student.unimelb.edu.au	lingliang@student.unimelb.edu.au	*lisongx@student.unimelb.edu.au*

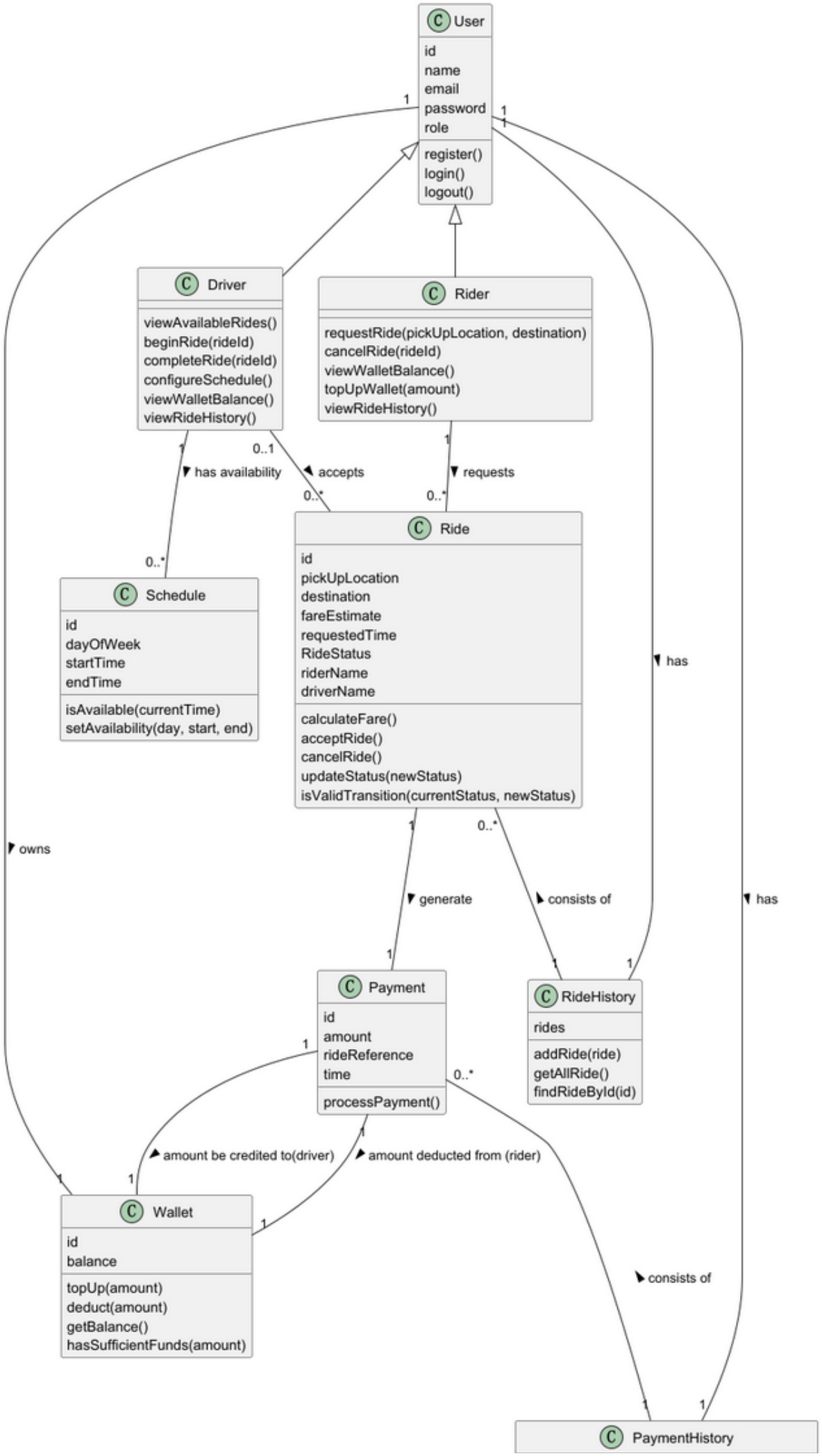
Repository structure

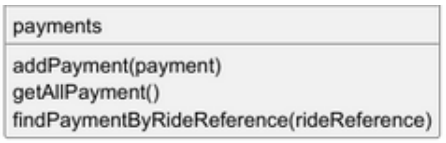
├── src/ # source of the project

├── ├── part1/ #pat1 documentation

├── ├── meetingminutes/ #Meeting minutes documentation

└── README.md


User
id
name
email
password
role
register()
login()
logout()
Driver
viewAvailableRides()
beginRide(rideId)
completeRide(rideId)
configureSchedule()
viewWalletBalance()
viewRideHistory()
Rider
requestRide(pickUpLocation, destination)
cancelRide(rideId)
viewWalletBalance()
topUpWallet(amount)
viewRideHistory()
has availability
accepts
requests
Schedule
id
dayOfWeek
startTime
endTime
isAvailable(currentTime)
setAvailability(day, start, end)
Ride
id
pickUpLocation
destination
fareEstimate
requestedTime
RideStatus
riderName
driverName
calculateFare()
acceptRide()
cancelRide()
updateStatus(newStatus)
isValidTransition(currentStatus, newStatus)
owns
has
generate
consists of
Payment
id
amount
rideReference
time
processPayment()
RideHistory
rides
addRide(ride)
getAllRide()
findRideById(id)
amount be credited to(driver)
amount deducted from (rider)
Wallet
id
balance
topUp(amount)
deduct(amount)
getBalance()
hasSufficientFunds(amount)
PaymentHistory



Ride Sharing Application UML Diagram Description

Overview

This UML diagram shows the main entities in the Ride Sharing Application.

Through the entire functionality, we can confirm that the system only includes two types of participants, and they share certain commonalities in terms of wallets and accounts. Therefore, we created the **User** entity—the foundational entity for all users in the entire system—to align with *Functionality1* and *Functionality2*. It includes all common attributes possessed by users, as well as methods for account control and verification; and the corresponding **Driver** and **Rider** entities. It stores the following attributes:

- `id`
- `name`
- `email`
- `password`
- `role`
  etc.

Rider

Considering the distinction between Riders and Drivers:
Riders are those who request rides and make payments.
A Rider can:

- Request rides by specifying pickup and drop-off locations
- Cancel rides before they are accepted or in progress
- Top up their wallets
- View their ride and payment history
  etc.

Driver

Drivers are service providers who accept and complete trips in exchange for compensation.
A Driver can:

- View available trip requests within their available time window
- Accept trips
- Start and complete trips
- Configure work schedules

- View income history
  etc.

Schedule

Schedule enables drivers to configure their available time windows for accepting trips, implementing *Functionality 3*.
It stores the following core attributes:

- `id`
- `dayOfWeek`
- `startTime`
- `endTime`
  etc.

Each schedule entry represents a weekly recurring time period during which the driver can accept trip requests. The entity includes methods to:

- Check whether the current time is within the available time window
- Set or modify the available time period
  etc.

Wallet

Wallet serves as each user's financial account, maintaining their current balance and providing basic financial operations. It includes methods to:

- Add funds (*available only to Riders*)
- Deduct payments upon trip completion
- Check available balances
- Verify sufficient funds before allowing trip requests
  etc.

Ride

Each ride can be interacted with by Riders and Drivers and will be stored in history after it is complete.
We use the **Ride** entity to represent each ride. It stores:

- `id`
- `pickUpLocation`
- `destination`
- `RideStatus` (Enum variable with only 5 possible states)

Associations:

- **requests**: A Rider can request a ride by specifying pickup and destination
- **accepts**: A Driver can accept a ride if they are available

When a ride is complete:

- It generates a payment record
- It is stored in **RideHistory** for future reference

Payment

Since each payment record is distinct and is transferred between objects, we use the **Payment** entity to represent the payment record of each ride. It stores:

- `id`
- `amount`
- `rideReference`

It has an association **generate** with **Ride**, indicating a one-to-one relationship.
When a payment is processed:

- The `amount` is deducted from the Rider's wallet
- The same amount is credited to the Driver's wallet
- The payment record is stored in **PaymentHistory**

RideHistory & PaymentHistory

RideHistory and **PaymentHistory** are responsible for storing and managing their own records.

We define them as entities because:

- We need entities to manage records
- To maintain encapsulation, both the records and their management methods are placed in the same entity