Report of Part 1A

A. Team Information

Team Hajimi

Name	Student ID	UoM Username	Github Username	Email
BO Huang	1584795	BOHAUNG1	wffb	huang.b3@student.unimelb.edu.au
Kaiwen QIU	1595215	kaiwenq	hylx4444	kaiwenq@student.unimelb.edu.au
Lingrui Liang	1522316	LingLiang	llrznhy	lingliang@student.unimelb.edu.au
Lisong Xiao	1652294	lisongx	EvnyaGH	lisongx@student.unimelb.edu.au

B. Domain Model Description

This is a UML diagram showing 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 *id*, *name*, *email*, *password,* and *role*.

Considering the distinction between *Riders* and *Drivers*. *Riders* are those who request rides and make payments. *Rider* can request rides by specifying pickup and drop-off locations, cancel rides before they are accepted or in progress, top up their wallets, and view their ride and payment history.

Drivers are a service provider who accepts and completes trips in exchange for compensation. *Drivers* can view available trip requests within their available time window, accept trips, start and complete trips, configure their work schedules, and view their income history.

Schedule enables drivers to configure their available time windows for accepting trips,

implementing Functionality 3. It stores *id*, *dayOfWeek*, *startTime*, and *endTime* as its core attributes. Each schedule entry represents a weekly recurring time period during which the driver can accept trip requests. The entity includes methods for checking whether the current time is within the available time window and for setting or modifying the available time period.

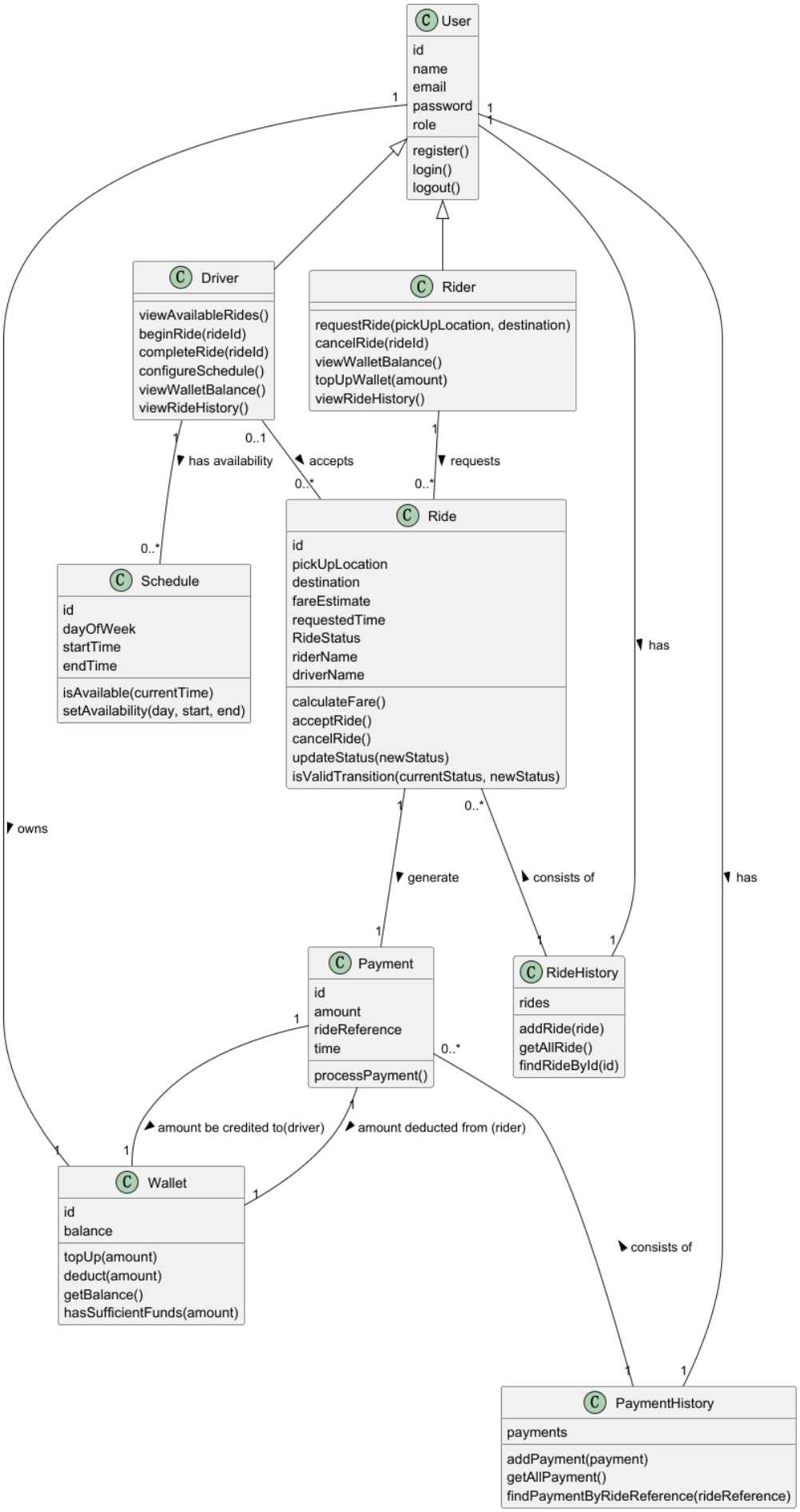
Wallet serves as each user's financial account, maintaining their current balance and providing basic financial operations. It includes methods for adding funds (available only to *Rider*), deducting payments upon trip completion, checking available balances, and verifying sufficient funds before allowing trip requests.

Each ride can be interacted by riders and drivers and it will be stored to history after it is complete. It is natural to consider it as an object, so we use *Ride* entity to represents each ride. It stores *id*, *pickUpLocation*, *destination* and *RideStatus* as its main attributes. *RideStatus* is an Enum variable since there are only 5 possible states of a ride. Association *requests* implies that a rider can request a ride by specifying pickUpLocation and destination. And association *accepts* implies a driver can accept a ride if they are available. When a ride is complete, it generates a payment record and be put into the *RideHistory*, which stores and manages past rides for future reference.

Considering that each payment record is distinct and is transferred between objects, we use *Payment* entity to represent the payment record of each ride. It stores *id*, *amount*, *rideRerence* as its main attributes. It has an association *generate* with *Ride* entity, indicating a one-to-one relationship between them. When a payment is processed, the order amount (*amount*) will be deducted from the rider's wallet and the same amount will be credited to the driver's wallet. After a payment record is generated, it will be put into *PaymentHistory*.

RideHistory and *PaymentHistory* entities are responsible for storing and managing their own records. We define *RideHistory* and *PaymentHistory* as entities because we need entities to manage records. To maintain encapsulation, we place both the records and the methods for managing them within the same entity, thus defining them as entities.

C. Domain Model Diagram


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
has
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)
consists of
PaymentHistory
payments
addPayment(payment)
getAllPayment()
findPaymentByRideReference(rideReference)