Cab Booking Application

Description:

Implement a cab booking application. Below are the expected features from the system.

Features:

- 1. The application allows users to book rides on a route.
- 2. Users can register themself and make changes to their details.
- 3. Driving partner can onboard on the system with the vehicle details
- 4. Users can search and select one from multiple available rides on a route with the same source and destination based on the nearest to the user

Requirements:

- 1. Application should allow user onboarding.
 - a. add_user(user_detail)
 - i. Add basic user details
 - b. update user(username, updated details)
 - i. User should be able to update its contact details
 - c. update userLocation(username,Location):
 - i. This will update the user location in X , Y coordinate to find nearest in future
- 2. Application should allow Driver onboarding
 - a. add driver(driver details, vehicle details, current location)
 - i. This will create an instance of the driver and will mark his current location on the map
 - b. update_driverLocation(driver_name)
 - i. This will mark the current location of driver
 - c. change driver status(driver name, status)
 - i. In this driver can make himself either available or unavailable via a boolean
- 3. Application should allow the user to find a ride based on the criteria below
 - a. find ride (Username, Source, destination)
 - i. It will return a list of available ride
 - b. choose ride(Username, drive name)
 - i. It will choose the drive name from the list

Note: Only the driver which is at a max distance of 5 unit will be displayed to a user and the driver should be in available state to confirm the booking

- c. calculateBill(Username):
 - i. It will return the bill based on the distance between the source and destination and will display it
- 4. Application should at the end calculate the earning of all the driver onboarded in the application **find_total_earning**()

Other Notes:

- 1. Write a driver class for demo purposes. Which will execute all the commands at one place in the code and have test cases.
- 2. Do not use any database or NoSQL store, use in-memory data-structure for now.
- 3. Do not create any UI for the application.
- 4. Please prioritize code compilation, execution and completion.
- 5. Work on the expected output first and then add bonus features of your own.

Expectations:

- 1. Make sure that you have a working and demo-able code.
- 2. Make sure that code is functionally correct.
- 3. Use of proper abstraction, entity modelling, separation of concerns is good to have.
- 4. Code should be modular, readable and unit-testable.
- 5. Code should easily accommodate new requirements with minimal changes.
- 6. Proper exception handling is required.
- 7. Concurrency Handling (BONUS) [Good if you do this]

Sample Test Cases:

- 1. Onboard 3 users
 - a. add_user("Abhishek, M, 23"); update_userLocation("Abhishek",(0,0))
 - b. add_user("Rahul, M, 29"); update_userLocation("Rahul",(10,0))
 - c. add user("Nandini, F, 22") ;update userLocation("Nandini",(15,6))
- 2. Onboard 3 driver to the application
 - a. add_driver("Driver1, M, 22", "Swift, KA-01-12345",(10,1))
 - b. **add_driver**("Driver2, M, 29", "Swift, KA-01-12345", (11,10))
 - c. add_driver("Driver3, M, 24", "Swift, KA-01-12345", (5,3))
- 3. User trying to get a ride
 - a. **find ride**("Abhishek",(0,0),(20,1))

Output: No ride found [Since all the driver are more than 5 units away from user]

b. **find ride**("Rahul",(10,0),(15,3))

Output : Driver1 [Available] choose_ride("Rahul","Driver1")

Output : ride Started calculateBill("Rahul")

Output: ride Ended bill amount \$ 6

Backend API Call: update_userLocation("Rahul",(15,3))
update driverLocation("Driver1",(15,3))

- c. **change_driver_status**("Driver1",False)
- d. **find_ride(**"Nandini",(15,6),(20,4))

Output: No ride found [Driver one in set to not available]

4. Total earning by drivers

a. find_total_earning()

- i. Driver1 earn \$6
- ii. Driver2 earn \$0
- iii. Driver3 earn \$0