Project 1

Design and implement a microservice application called *LastMile* which can be used by commuters to hire a drop service from metro stations to nearby locations. The application consists of riders and drivers. Drivers indicate their route and the metro stations near the route with a number of free seats for riders. Riders indicate the time at which they reach the target metro station and their destination. Drivers continously update the their location and pickup from a metro station. You can mimic this by sending periodic updates of the location. The system matches a one or more riders just minutes before the driver reaches the metro station. For each metro station, you can keep a list of “nearby” locations and when a driver reaches that location, you can trigger the match. The target destination must be same as the drivers target destination for a match. Provide the API reference using Google RPC for implementing this as microservices. You have to deploy this application on Kubernetes and demonstrate running it matching riders with drivers. You must demonstrate how the application continues to work in the presence of failed services and scales from 1 matching service to a maximum of 5 matching services. The following are a list of microservices that constitute the application

- **User Service**
  - Manages rider and driver profiles.
  - Authentication and authorization.
- **Driver Service**
  - Allows drivers to register routes, metro stations, and available seats.
  - Updates driver location and pickup status.
- **Rider Service**
  - Allows riders to register their metro arrival time and destination.
  - Tracks ride status.
- **Matching Service**
  - Matches riders with drivers based on location, time, and destination.
  - Sends notifications to both parties.
- **Trip Service**
  - Manages trip lifecycle: scheduled, active, completed.
  - Tracks pickup and drop-off events.
- **Notification Service**
  - Sends real-time updates to riders and drivers.
  - Push notifications
- **Location Service**
  - Handles real-time location updates from drivers.
  - Proximity detection.
- **Station Service**
  - Maintains metadata about metro stations.
  - Maps stations to nearby areas.

Project 2

Build a distributed key value store with a single controller node and multiple worker nodes. The key-value store supports two operations – get and put. The get operation takes a key as input and returns a value. The put operation takes a key and value as input and sets the value to be associated with the key. The keys are organized as in a key-space and are partitioned into n where is the number of workers. Each worker is primarily responsible for a set of keys. The controller node knows how the keys are partitioned among the workers and each client must

first query the controller to get this mapping. It then proceeds to query the primary worker for that key. Each worker replicates all the data to 2 more workers so there are 3 replicas for each key. The put operation only succeeds when the data is written to 2 replicas. The 3rd replica is written in the background.

Handling failures

- Each worker node sends a heartbeat to the controller
- On worker failure, the keys stored on the worker have less than 3 copies, so the controller initiates transfer from the other nodes.

Design a REST API to query the controller and implement the get and put operations as REST APIs.

Your final solution should have 1 controller and 4 workers.