**Internal Hire Assignment – Response**

**Developer – Neha Vishwarupe**

Pl use attached springboot-kafka-jpa.zip file

**Running the application**

1. Unzip springboot-kafka-jpa.zip file
2. Run “docker-compose up”
3. Zip includes source code for project

**Design/ Implementation Details**

**REST API**

* REST API
* POST - <http://localhost:9000/api/driver>s //Add or update new driver position
* POST - <http://localhost:9000/api/store>s //Add new store
* GET - [http://localhost:9000/api/drivers?storeId=<id>&numerofdrives=<number](http://localhost:9000/api/drivers?storeId=%3cid%3e&numerofdrives=%3cnumber)> //Get N nearest drivers
* Implemented **SWAGGER API** but disabled for the production “profile”

**Testing**

* **Mockito/Junit Framework**
* Scale testing: N nearest driver selection from pool of 200 drivers in 5 seconds

**Multi-Module Maven Project**

To demonstrate use multiple modules – in addition to parent project, 2 separate modules are created “application” and “library”.

application module uses “service” exposed by library module.

**App Container**

* Packaged application as “jar” and build container with Open JDK 11 as base image
* docker-compose.yml updates
  + Added “app-tier” network for “db” to make sure all 4 containers are part of same bridge network
  + Added “myapp” w/ dependency on “db” container. Docker builds container with “Dockerfile” when docker-compose starts for the 1st time.

**Scale Consideration**

* As all drivers needs to be visited in nearest driver selection, maintained in-memory cache for drivers
* Minimalistic Locking for liveliness of application – Synchronization/lock at driver instance as the location may get updated while performing ‘get nearest driver’ query
* Kafka Future Enhancements
  + As there can be large # of driver updates, we can create multiple partitions for KAFKA topic and have multiple consumers in the “group” process events in parallel. However, to make this design work, we need “ordering guarantee” to play events for given driver in sequence. This can be achieved via setting up “driver id” as KAFKA KEY. This will ensure given driver will always hash to the same partition. Note that this is not implemented as there is expectation that application should work with direct CLI and there is no mention of KAFKA key for the same. BTW in addition to producing KAFKA event via REST API, tested by generating events via CLI (kafka-console-producer).
  + In-memory cache makes DB persistence lower priority for driver table. This can be achieved via DB connection pool and by periodic batch DB update (say every minute) instead of hitting DB w/ every single driver update. However, with this change, KAFKA “offset” should be committed only after processing these events fully (persisting in DB) else we may lose driver updates in case of application crash. BTW existing drivers are loaded from DB in case of application restart.