**CSMS (charging station management system) Project Setup and Execution Guide**

# 1. Introduction

Created 3 individual project and added each project in a separate Github repository   
**1. csms-common  
2. csms-transaction-service  
3. csms-authentication-service**  
  
For use the project created a combined **csms** and added all of them in that Github repository   
 **csms Github Repository** : <https://github.com/avinitjsr/csms>  
 **Clone and execution** of the project in the below section (**Github Details and Execution**)  
  
**Note**: 1. Not added any database just added the test data manually Hard coded for identify the driverID  
 **Example**: driverIdentifier id : aaaaaaaaaaaaaaaaaaaa then Accepted  
 driverIdentifier id : bbbbbbbbbbbbbbbbbbbb then Rejected  
 2. Created separate repository for each project beneficial to use by another project also

# 2. Prerequisites

Following setup require for execution:

- Java 21

- Gradle

- Docker & Docker Compose (for containerized setup)

- Confluent Platform OR Schema Registry binaries (for local setup without Docker)

# 3. Github Details and Execution

Created **csms** Github repository which consists of all 3 root project   
 **csms gihub repository**: <https://github.com/avinitjsr/csms>  
  
 **To clone the project locally, run**:  
 *git clone* [*https://github.com/avinitjsr/csms.git*](https://github.com/avinitjsr/csms.git)

It will clone all 3 project in a single **csms** project  
 **csms-common  
 csms-transaction-service  
 csms-authentication-service  
 docker-compose.yml  
 README.md**

# 3.1 Execution with Docker (Recommended)

**Use below steps for run the project using Docker Compose:**

1. Navigate to the project root where docker-compose.yml is located.  
    Example: inside your csms project which clone from github  
    cd csms
2. Start Kafka and Schema Registry using:  
   docker-compose up -d
3. Verify containers are running:  
   docker ps
4. Move csms-common file to local repository:  
    cd csms-common  
   ./gradlew clean build publishToMavenLocal
5. Build and Start csms-authentication-service  
   cd csms-authentication-service  
   ./gradlew clean build --refresh-dependencies  
   ./gradlew bootRun
6. Build and Start csms-transaction-service  
   cd csms-transaction-service  
   ./gradlew clean build --refresh-dependencies  
   ./gradlew bootRun

# 3.2 Execution Manual

**Use below steps for Run Kafka and Schema Registry manually:**

1. Download 7.4.1 version and tar or archive as compare to your system(windows,linux,mac) from the url  
   <https://packages.confluent.io/archive/7.4/>
2. After download extract the confluent-community-7.4.1 folder and go in that folder
3. Example:  
   cd confluent-7.4.1
4. **Start zookeeper:**  
   bin/zookeeper-server-start etc/kafka/zookeeper.properties
5. **Start kafka:**  
   bin/kafka-server-start etc/kafka/server.properties
6. **Start Schema Registry:**  
   bin/schema-registry-start etc/schema-registry/schema-registry.properties
7. Move csms-common file to local repository:  
    cd csms-common  
   ./gradlew clean build publishToMavenLocal
8. Build and Start csms-authentication-service  
   cd csms-authentication-service  
   ./gradlew clean build --refresh-dependencies  
   ./gradlew bootRun
9. Build and Start csms-transaction-service  
   cd csms-transaction-service  
   ./gradlew clean build --refresh-dependencies  
   ./gradlew bootRun

3.3 Application Execution Test

**After execution of above step 3.1 or 3.2 test the below scenario :**  
 **1. Accepted scenario** : once driverIdentifier id in between 20 to 80 characters and data set true in authentication-service  
 **Input command:**curl -X POST localhost:8080/transaction/authorize \-H "Content-Type: application/json" \-d '{"stationUuid":"25aac66b-6051-478a-95e2-6d3aa343b025","driverIdentifier":{"id":"aaaaaaaaaaaaaaaaaaaa"}}'  
  
**output**: {"authorizationStatus":"Accepted"}  
  
**2. Rejected scenario** : once driverIdentifier id in between 20 to 80 characters and data set false in authentication-service  
 **Input command:**  
curl -X POST localhost:8080/transaction/authorize \-H "Content-Type: application/json" \-d '{"stationUuid":"25aac66b-6051-478a-95e2-6d3aa343b025","driverIdentifier":{"id":"bbbbbbbbbbbbbbbbbbbb"}}'  
  
**output**: {"authorizationStatus":"Rejected"}  
  
**3. Invalid scenario** : once driverIdentifier id less than 20 characters   
 **Input command:**curl -X POST localhost:8080/transaction/authorize \-H "Content-Type: application/json" \-d '{"stationUuid":"25aac66b-6051-478a-95e2-6d3aa343b025","driverIdentifier":{"id":"aaaaaa"}}'  
  
**output**: {"authorizationStatus":" Invalid"}

**4. Unknown scenario** : once driverIdentifier id in between 20 to 80 characters and data does not set in authentication-service  
 **Input command:**  
curl -X POST localhost:8080/transaction/authorize \-H "Content-Type: application/json" \-d '{"stationUuid":"25aac66b-6051-478a-95e2-6d3aa343b025","driverIdentifier":{"id":"cccccccccccccccccccc"}}'  
  
**output**: {"authorizationStatus":"Unknown"}

3.4 Integration and Unit Test Execution

I have written **Integration** and **Unit** test for both services(csms-transaction-service and csms- authentication -service).  
csms-common is the simple DTOS so no required Test classes for this   
  
**1. csms-transaction-service Test cases:**

**Integration Test**: src -> test -> kotlin  
 com.csms.ts  
 integrationTest -> TransactionServiceIT.kt  
  
 **Unit Test cases**: src-> test -> Kotlin  
 com.csms.ts  
 controller -> TransactionControllerTest.kt  
 kafka   
 -> consumer -> AuthenticationResponseConsumerTest.kt  
 ->producer -> TransactionProducerTest.kt  
  
 **Build and Start Test cases:** cd csms-transaction-service  
 ./gradlew test  
  
 **Result -> Show the test result in html format** csms-transaction-service/build/reports/tests/test/index.html

**2. csms- authentication -service Test cases:**

**Unit Test cases**: src-> test -> Kotlin  
 com.csms.as  
 service -> AuthenticationServiceTest.kt  
 kafka -> AuthenticationRequestConsumerTest.kt

**Build and Start Test cases:**  
 cd csms- authentication -service  
 ./gradlew test  
  
 **Result -> Show the test result in html format** csms- authentication -service/build/reports/tests/test/index.html

# 4. Project Implementation Details

1. **csms-common**  
   The csms-common module defines shared Avro schemas used for communication between services via Kafka. It acts as a contract module, ensuring consistent message formats across services.  
     
   **Files:**

* **AuthenticationRequest.avsc** Represents a request from a charging station to authorize a driver.  
   Fields:  
   requestId: Unique ID of the request.  
   stationUuid: Identifier of the station.  
   driverIdentifier: Nested record containing driver ID.
* **AuthenticationResponse.avsc**  
   Represents the response to an authentication request.  
   Fields:  
   requestId: ID to correlate with the request.

authorizationStatus: Enum indicating the outcome.

* **AuthorizationStatus.avsc**  
  Enum defining possible statuses for authorization:

Fields:  
 Accepted, Invalid, Unknown, Rejected

1. **csms-transaction-service**  
    Transaction service handles incoming driver authorization requests from clients and communicates with the authentication (csms-authentication-service) service via Kafka. It waits for an asynchronous response to complete the request  
     
   **Responsibilities**

* Exposes a REST API to accept authorization requests
* Sends AuthenticationRequest messages to Kafka.
* Waits for and consumes AuthenticationResponse messages.
* Returns the final authorization status to the client.

**Components Details**

* **Controller**: TransactionController.kt Handle /transaction/authorize endpoint.
* **Kafka** : TransactionProducer.kt: Produce authorization requests to Kafka.

AuthenticationResponseConsumer.kt: Consume and processes responses from Kafka.

* **DTOs**: AuthorizeRequest, AuthorizeResponse, DriverId Request/response models for the REST API.

1. **csms-authentication-service**  
   Authentication service processes authentication requests received via Kafka and determines whether a given driver identifier is valid, known, and authorized.  
     
   **Responsibilities**

* Consumes AuthenticationRequest messages from Kafka.
* Validates the driver ID based on predefined logic.
* Publishes AuthenticationResponse messages back to Kafka.

**Components Details**

* **Service**: AuthenticationService.kt Contains logic to resolve the AuthorizationStatus of a driver ID.
* **Kafka** : AuthenticationRequestConsumer.kt Consume authorization requests and publishes responses.