**Documentation - Fuel Station Data Collector - Group R**

# GitHub-Link

<https://github.com/wi22b085/DistributedSystems>

# Software Architecture

The **Spring Boot App** receives the customer ID entered in **JavaFX** via an HTTP request (GET). The program then sends the data read from the Rabbit MQ stream ECHO\_IN\_QUEUE\_ID.

The data read into the ECHO\_IN\_QUEUE\_ID is then taken by the **Data Collection Dispatcher**. This dispatcher is connected to a database, which contains the URL as well as the longitude and latitude values of each current location. The System is concepted like this, so that if needed other stations can be easily added and the current system can be expanded. An Entity is created to extract the values for each station, which are then stored in a list. The Collection dispatcher goes through each element of the list to extract the url so that the other components can also access the necessary data. For each row in the list a new queue is defined which contains the db-url aswell as the current customer id. This is then sent to the ECHO\_IN\_URL\_ID. Another queue is sent to the Data collection receiver, with a message, which informs that a new job has started.

The **Station Data Collector** is

The **Data Collection Receiver** receives for every job four messages from two different queues: One message comes from the Data Collection Dispatcher, that informs that a new job has started. The other three messages come from the Station Data Collector and contain the used kWh of a specific customer for all three charging stations. For all these three messages a key is created, that contains the customer-ID and the charging station. The data is then saved together with the key into a HashMap. If all three keys for a specific customer are present in the HashMap, the data is extracted from the HashMap. Then the corresponding costs are calculated. Finally, all the sorted data is sent together with the current datetime to the next queue and the entries get removed from the HashMap.

The **PDF Generator** receives a message from the Data Collection Receiver. This message contains all the data needed for the PDF-Invoice creation except for the first- and lastname of the customer. Therefore, a CustomerEntity is created with the data of the customerdb-Database to collect the first- and lastname. The database is connected via a Jdbc-Template. Then the actual PDF-Creation function gets invoked and the PDF is created in the defined path with the customer-ID and the date as a name.

# Setup/Installation- and User-Guide

# Unit Testing Decisions

**Data Collection Receiver:**

We decided to run the “sendDataPdf” (has the data for a specific customer from a specific charging station as input) four times: Three times for every charging station of Customer 2 and one time for a different customer. Then we verify, that data would be sent to the next expected queue exactly one time (only happens, if the data for every charging station is present for the specific customer), with the expected message. Finally, we verify that the three entries of that specific customer get removed and the single entry of the different customer persists.

**PDF Generator:**

We decided to verify, that the defined query is executed exactly once on the database via the Jdbc-Template and that the PDF Generator creates a PDF-File with the expected name at the expected location. Furthermore, we verify, that the created PDF-File contains the expected text (e.g. Customer-ID, Name of Customer, Costs). Finally, at the end of the unit test, the created PDF-File gets deleted.

# Lessons Learned

# Tracked Time

|  |  |  |
| --- | --- | --- |
| Feature | Time | Team-Member |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Reno (vorerst):

Data Collection Receiver: 16h

Test: 6h

PDF Generator: 12h

Test: 4h

Documentation: 6h