DISTRIBUTED SYSTEMS PROJECTDOCUMENTION

MARIA MIRNIC, NAZIA NAZARI, MICHAEL REITER

DATE: 18.06.2023

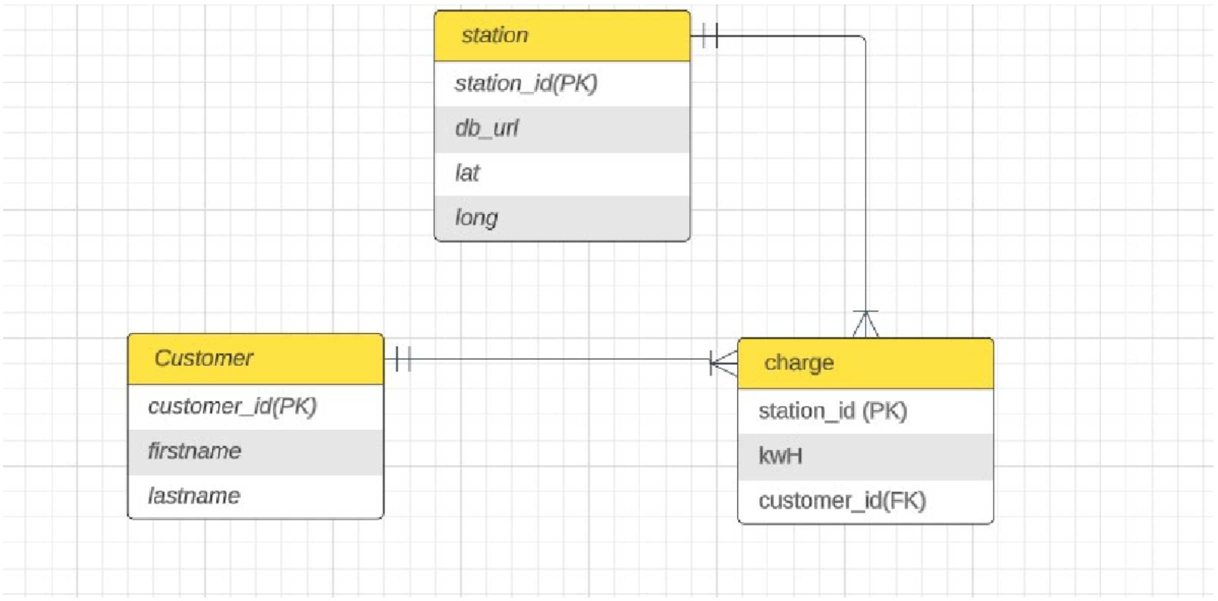
Documentation

Name	Feature	Date	Time Investment	Description
Nazia	Data Collection Dispatcher	11.05.2023	2 hours	Worked on Database Connection
Michael	FrontEnd	11.05.2023	2 hours	Started Frontend (JavaFX)
Maria	Station Data Collector	11.05.2023	2 hours	Started Queue for StationDataCollector
Nazia	Data Collection Dispatcher	19.05.2023	2 hours	Worked on Queue (Sender & Receiver)
Michael	Springboot Application	19.05.2023	2 hours	Worked on Springboot Application
Maria	Data Collection Receiver	19.05.2023	2 hours	Worked on Data Collection Receiver
Maria	Station Data Collector	05.06.2023	5 hours	StationDataCollector Done
Nazia	Station Data Collector - Testing	12.06.2023	2 hours	Finished Station Data Collector. Started writing unit tests.
Nazia	Documentation & UML Diagram	15.06.2023	3 hours	Created Documentation & UML Diagram
Maria	Data Collection Receiver	15.06.2023	5 hours	Finished Data Collection Receiver
Nazia	Testing & UML	17.06.2023	3 hours	Corrected UML & Testing
Nazia	Testing	18.06.2023	1 hour	Finished testing
Maria & Michael	Hotfixes	18.06.2023	4 hours	Worked on some hotfixes

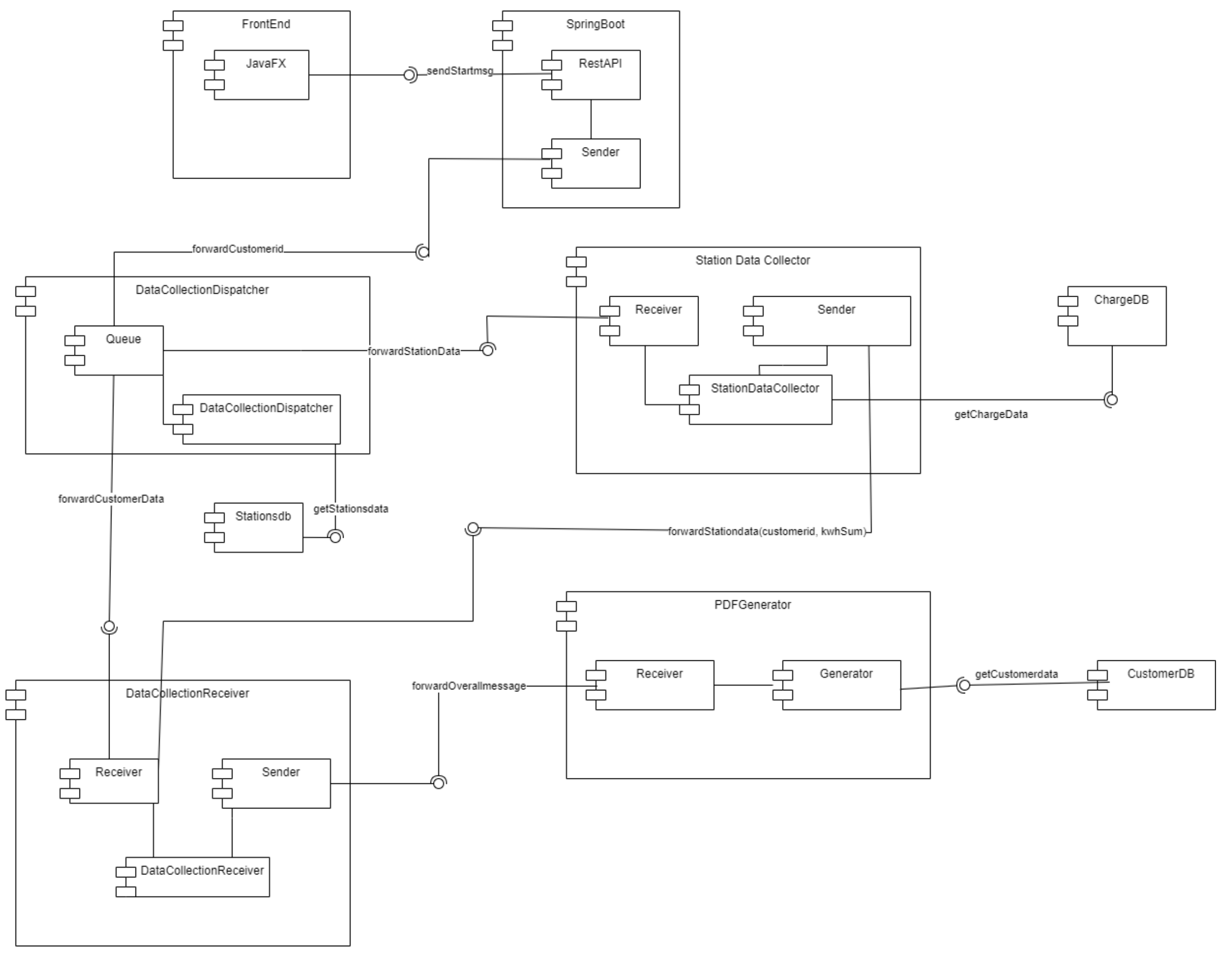
Divison of Responsibilty

Name	Responsibility
Michael Reiter	Spring Boot App
Michael Reiter	Java FX App
Michael Reiter	PDF-Generator
Michael Reiter	Unit Tests
Maria Mirnic	Station Data Collector
Maria Mirnic	Data Collection Receiver
Nazia Nazari	Data Collection Dispatcher
Nazia Nazari	Unit Tests
Nazia Nazari	UML Diagram

Database



UML Diagram



FrontEnd
JavaFX
sendStartmsg
SpringBoot
RestAPI
Sender
forwardCustomerid
DataCollectionDispatcher
Queue
DataCollectionDispatcher
forwardStationData
Station Data Collector
Receiver
Sender
StationDataCollector
ChargeDB
getChargeData
forwardCustomerData
Stationsdb
getStationsdata
forwardStationdata(customerid, kwhSum)
PDFGenerator
Receiver
Generator
getCustomerdata
CustomerDB
forwardOverallmessage
DataCollectionReceiver
Receiver
Sender
DataCollectionReceiver

Set Up, Installation & User Guide

Welcome to the User Guide for the Fuel Data Station Collector. This guide provides detailed instructions on how to use it.

<u>Set up and installation:</u>

First step: Install docker and run the database with docker.

Second step: Open the project.zip in IntelliJ and run following classes:

Main (pdfgen), HelloApplication(Client), Main(Station DataCollector), RestAPIApplication (Rest API), Main (DataCollectionDispatcher), Main (DataCollectionReceiver)

Third step: Enter a customerid in the FrontEnd

Fourth step: Click on "Gather Data" in the FrontEnd

Fifth step: Click on "Show Invoice" in the FrontEnd

Now an invoice pdf should open.

<u>Key Features:</u>

As a user you have to enter your customerid in the Frontend. This way you can print your invoice. First you have to click on "Gather Data", then on "Show Invoice".

Lessons Learned

<u>Cleary defined requirements:</u>

It is essential to have clearly defined requirements. A thorough requirement analysis helps understand the project's scope. Furthermore, it helps understand the structure and the functionality of the project. Additionally, it is important to not only know the functionality and structure of the own service but the whole project in order to see the bigger picture.

<u>Division of responsibility:</u>

It is important to divide the project into smaller steps and split them up among the group. This also includes defining what each group member is responsible for, not only including the implementation, but also the documentation. One should also not forget to clarify the dependencies among the steps. This way, a group member is aware how the other members depend on him and essentially does not take too long to do his part.

Architecture and design:

Before the implementation of the project, one should establish a solid architecture and a suitable design. In this case, frameworks such as Spring Boot, JavaFX, Docker were already predetermined, but for future projects it is important to define these factors beforehand.

Continuous tests:

During the implementation, the code should be tested continuously. This can be done by running the code and comparing the results to the desired outcome. One can also write unit tests to test certain functions independently.

Time-management:

Before the group starts implementing the program, it is important to plan the project. The plan should not only include the time for the implementation but also time for the documentation and a time buffer. One needs to consider that installing and configuring the frameworks can take a lot of time. Sometimes the frameworks do not work and need to be reinstalled. These are some issues that should be considered in the timetable.

Unit test descisions

Unit test	Description	Reason
DataCollectionReceiver - checkKWHSum	This test checks whether the sum of the KWH is formed correctly	This is an essential part of our project. Without this, the invoice is not correct.
StationDataCollector – checkMessage	This message checks whether the message is correctly formed. It should consist of kwhSum;customerid	This test is important, because it creates the message for the DataCollectionReceiver.
APIController – testPost	Mocks the Send class, then compare the expected and actual results to ensure that the API controller behaves correctly.	To ensures that the `post()` method behaves correctly.

Link to Github

https://github.com/michi-1/ProjectDisys