Data Station Collector

About The Project

The Data Station Collector is a project where a distributed system with a REST-based API, a RabbitMQ message queue and a JavaFX UI was implemented.

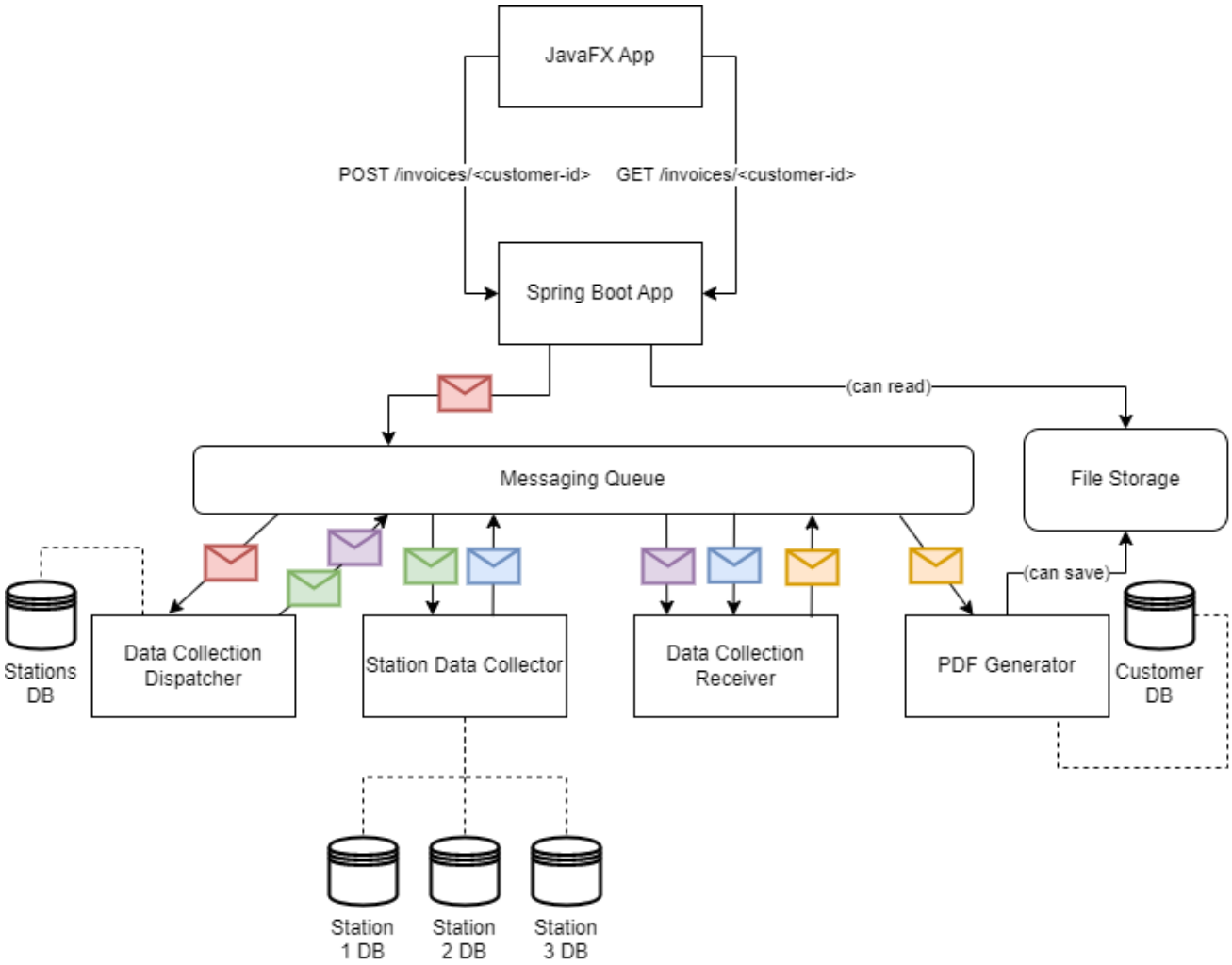
The workflow:

- Customer can input a customer id into the UI and click “Generate Invoice”
- A HTTP Request calls the REST-based API
- The application starts a new data gathering job
- When the data is gathered, it gets send to the PDF generator
- The PDF generator generates the invoice and saves it on the file system
- The UI checks every couple seconds if the invoice is available

It is a docker-compose project that sets up five databases and a queue. One database stores user information, one database stores the access information of the other three databases, which itself store the charging information (every charging station has its own database).

The System Overview

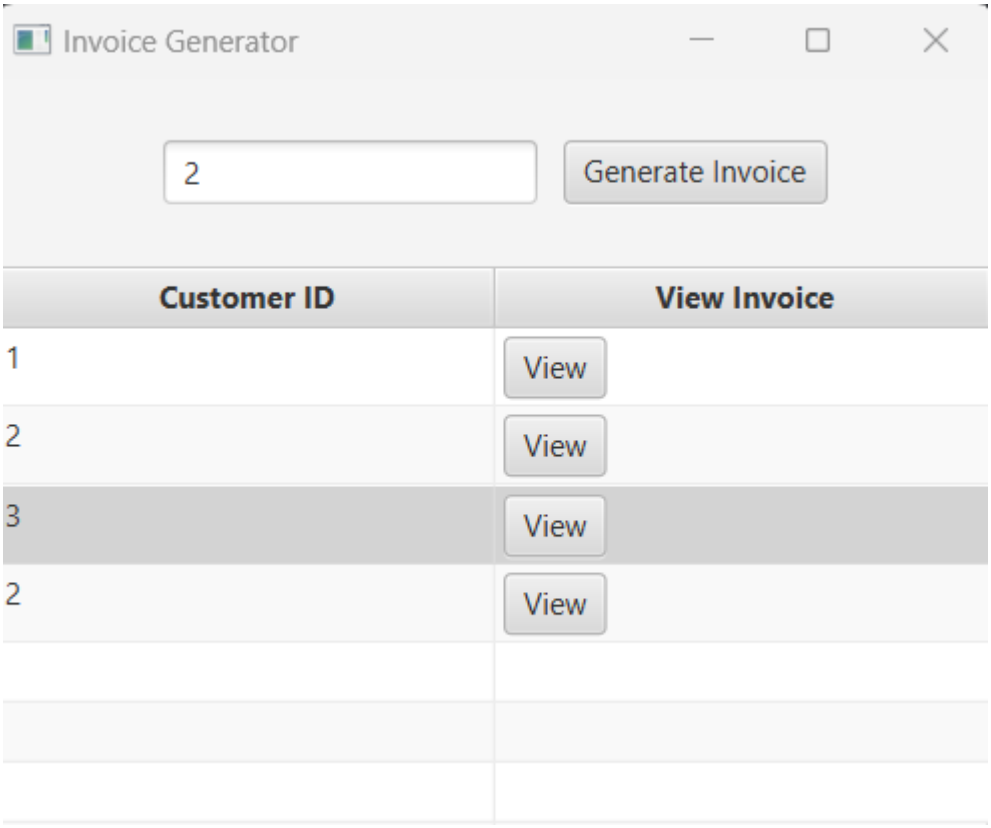
A graph of the system:



The Components

Station JavaFX Application

The PDF Generator is a JavaFX Application wher the customer can enter the customer_id. Afterwards a list with all corresponding invoices is displayed. The customer can select the invoice which should be generated as PDF.



Spring Boot Application

The Spring Boot Application starts the process by sending a start message to the Data Collection Dispatcher. It contains the Data Collection Controller which is responsible for the REST API and starts the data gathering job as well as returns the invoice PDF. The Data Collection Service within this component contains the business logic for the data collection Spring Boot Application. Further more the Name and the Service Queue are part of the Spring Boot Application.

Data Collection Dispatcher

The Data Collection Dispatcher starts the data gahtering job, has knowledge about the available stations, sends a message for every charging station to the Station Data Collector and sends a message to the Data Collection Receiver, that a new job started. It is responsible for dispatching the data collection to the correct data collector. Furthermor it includes the class Database Connector which is responsible for the connection to the database POSTGRES and the Station Class which represents a station central database. In addition the Name and the Service Queue are included.

Station Data Collector

The Station Data Collecotr gathers data for a specific customer from a specific charging station and sends data to the Data Collection Reciever. It includes the class Database Connector which is responsible for the connection to the database POSTGRES. It also contains the Charge class which represents a charge of a cutsomer and again the Name and the Service Queue.

Data Collection Receiver

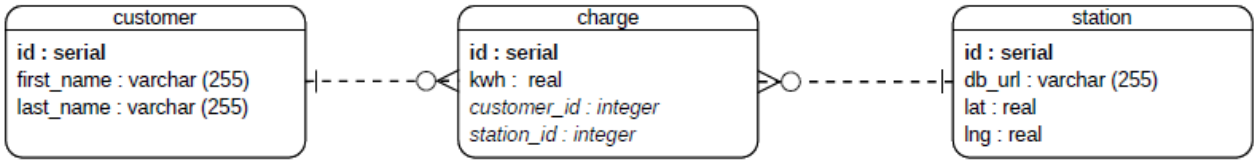
The Data Collection Receiver receives all collected data, sorts the data to the according gathering job and sends data to the PDF Generator when the data is complete. The class Data Collection Receiver is responsible for receiving the data collection from the data collector. Furthermore it contains the model classes invoice and station as well as the Name and the Service Queue.

PDF Generator

The PDF Generator generates the invoice from data and saves PDF to the file system. The service class PDF Helper is responsible for generating PDF files. It uses the iText library. The component also contains the class Database Connector which is responsible for the connection to the database POSTGRES as well as the model classes Customer, Invoice and Station. In addition the Name and the Service Queue are part of the component.

Entity Relationship Diagram

The following entity relationship diagram shows the tables of the database and how they are related:



The table customer contains all relevant customer data and is connected to the charge table via customer_id. The charge table contains the data about kwh and an id for tracking the record. The station table contains the data about the station and is connected to the charge table via station_id. The station table definition is the same for all station databases.

User Guide

This guide will help you set up and run the project using IntelliJ and Docker.

Requirements

- IntelliJ IDEA (Community or Ultimate edition) installed on your system
- Docker installed on your system

Installation

1. Clone or download the project from the GitHub repository.
2. Open the project in IntelliJ IDEA.

Starting Docker

1. Navigate to the project directory.
2. Run the `start_docker.bat` file by double-clicking on it.
    - This script will start the required Docker images using Docker Compose.

Running the Application

1. Open the project in IntelliJ IDEA.
2. In the IntelliJ toolbar, select the desired run configuration from the dropdown menu.
3. Click the Run button or press Shift+F10 to start the application.
    - This will start all the necessary services and applications defined in the run configuration.

Generating PDF

1. Once the applications have started, open the GUI for the desired application.
2. Enter the number of customers for whom you want to generate an invoice.
3. Wait for the PDF to be generated. The progress will be displayed in the application.
4. After the PDF is generated, click the "View" button to open the PDF file.
    - The PDF file will be opened using the default PDF viewer on your system.

That's it! You can now use the application to generate invoices and view them in PDF format.

Please note that this guide assumes basic familiarity with IntelliJ IDEA and Docker. If you encounter any issues or have specific questions about the project, refer to the project documentation or consult the project's support resources.

For more information and the latest updates, visit the GitHub repository.

Lessons Learned

- Active participation during teaching events and completing the exercises helps to understand the basic concept of distributed systems, but is not enough for successfully creating the project.
- A project as complex and big as this one takes always longer than estimated. Despite time buffers it took longer as estimated. Starting very early helped us to keep the deadline and not having an extremly busy time in the last view days.
- Considering the Project Grading Scheme from the very beginning helps to focus on most relevant tasks and to not forget any requirement.
- Cloud-based repositories are cruciual for successful team work. In additon communication tools like Discord are essential.
- Creating a basic coding concept and structure bevor starting to code in detail helps to reduce complexity and to ensure consistency. Afterwards it helps to split the tasks.
- For a group of not experienced programmers or even programming beginners the project would be hardly feasible.
- The support / help and explanations from experienced team members are by far more helpful than any online resource.

Effort

The time needed was not tracked, but we would estimate it took us approximately 20 hours per person, so about 80 hours in total.

Credits

Developer

The distributed system was coded by the following developers:

- Altin KELMENDI (WI21B054)
- Julian HOFFMANN (WI21B050)
- Sandra GRADWOHL-PREM (WI20B063)

- Sara SEIEDMIRZAEI (WI21B040)

Date of creation

The project was completed in June 2023.