Project documentation DISYS Semester-Project Group-X

Setup/Installation- and user-guide:

The Project uses JavaFX, Spring Boot, RabbitMQ and a Postgres Database. To be able to use the Programme the project Container on Docker from the Moodle course must be started. The Project consists of 6 independent IntelliJ Projects, which all must be started for the Project to work. Before using the Programme, you also must adjust the path in the PDFGeneratore and RestAPI (DownloadLinkController and InvoiceController) to the FileStorage directory. As a user you only interact with the JavaFXApp. The user interface opens automatically when you start the JavaFXApp IntelliJ Project. To generate a PDF, you must enter a Customer Id into the Text field that shows “Customer Id” and click the Button “Generate Invoice”. While the invoice is loading the Button is disabled so you can only request one pdf at a time. A Label below the Button shows the current statues of the request. And the Textfield below that shows the download link for the PDF as soon as it is ready. This is a Textfield so the link can be easy copied but writing in it will have no effect. The Pdf will also be stored in the directory “FileStorage”.

The CustomerId should be an integer or else an error message will appear. If there is no customer with the entered ID a PDF with the total of zero and a message that shows “No Customer with this ID” will be generated.

If an error occurs later in another service, then the JavaFXApp or the RestAPI an Output with the Error message will show in one of the Consoles.

Lessons learned:

We learned a lot about combining different tools. Especially since no one of use worked with a messaging queue or docker before. Also, I had to research some new things that we did not learn before, for example creating a PDF with java.

Unit Testing Decisions:

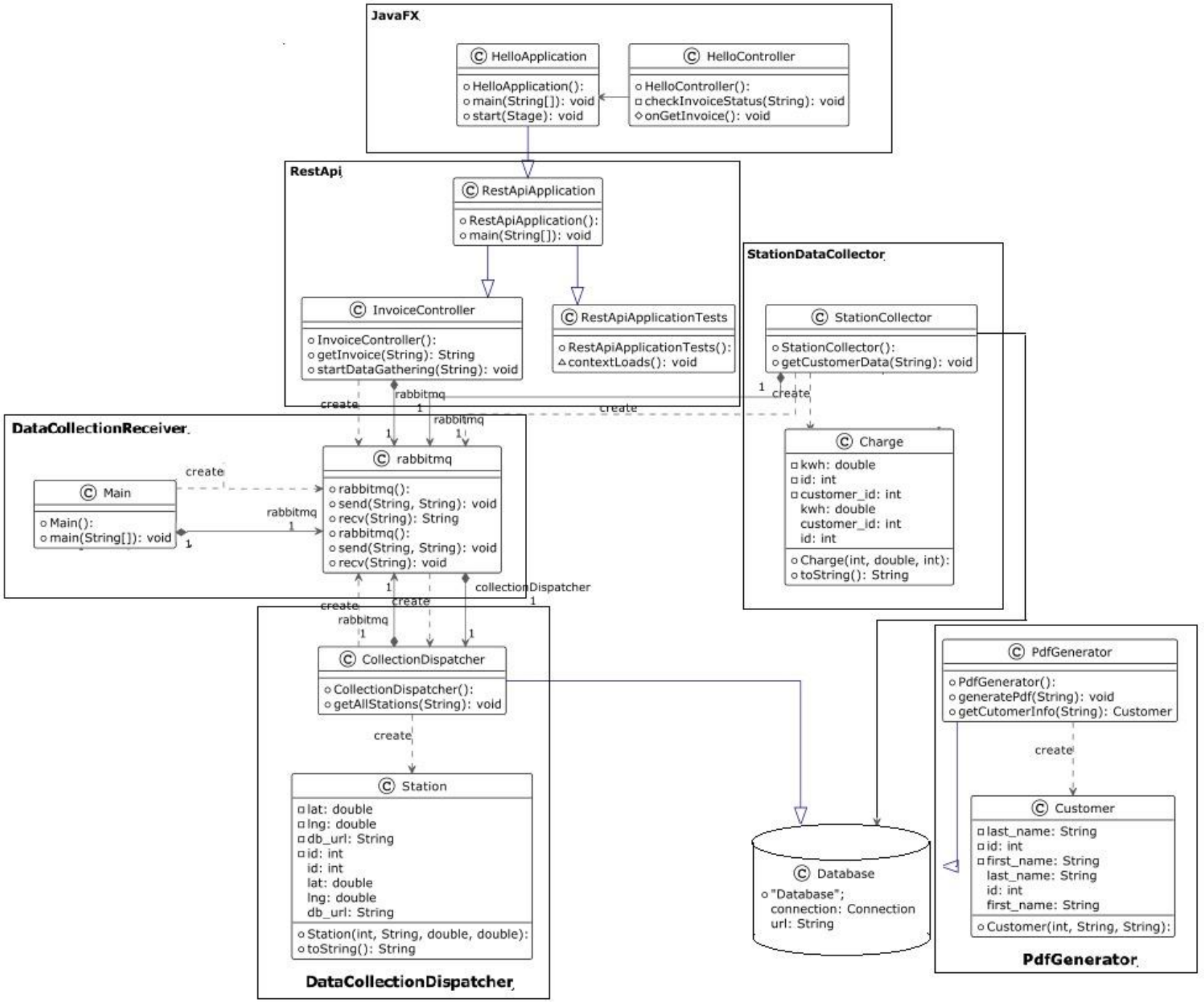
3 out of our 4 Service were tested with Unit Tests. The DataCollectionReceiver was not tested with Unit Tests because this service only interacts with the messaging queue. For the DataCollectionDispatcher only one Test was written because it was not possible to use mock data and if a station is added/removed the Test does not work anymore. The Test checks if the Dispatcher gets the right number of stations from the database. It was possible to make more Unit tests for the StationDataCollector, also using mock Data. In total there are 5 Tests which check the getCustomerData() and the getChargeList() methodes. We test the different Error Messages, String messages and if the right Charges are returned. For the PDF-Generator it was also possible to create some tests with mock data. There are in total 6 Test for the PDF-Generator. Both the generatePDF() and the getCustomerInfo() methods were tested. One Test checks if an PDF with the right Path exists after the creation, and others if the right Customers were created. Also some Error cases where tested.

Documentation on Github:

At the beginning of the project (only JavaFX was finished) there was a Problem with the main branch on Github and the Programme did not work anymore. Because of that a new branch called test2 was created. After that, the commits worked without a Problem and are accurately.

Github link: https://github.com/VanessaMaria02/DISYS.git

UML-Diagram:


JavaFX
HelloApplication
HelloApplication():
main(String[]): void
start(Stage): void
HelloController
HelloController():
checkInvoiceStatus(String): void
onGetInvoice(): void
RestApi
RestApiApplication
RestApiApplication():
main(String[]): void
InvoiceController
InvoiceController():
getInvoice(String): String
startDataGathering(String): void
RestApiApplicationTests
RestApiApplicationTests():
contextLoads(): void
StationDataCollector
StationCollector
StationCollector():
getCustomerData(String): void
Charge
kwh: double
id: int
customer_id: int
kwh: double
customer_id: int
id: int
Charge(int, double, int):
toString(): String
DataCollectionReceiver
Main
Main():
main(String[]): void
rabbitmq
rabbitmq():
send(String, String): void
recv(String): String
rabbitmq():
send(String, String): void
recv(String): void
create
rabbitmq
collectionDispatcher
CollectionDispatcher
CollectionDispatcher():
getAllStations(String): void
Station
lat: double
lng: double
db_url: String
id: int
id: int
lat: double
lng: double
db_url: String
Station(int, String, double, double):
toString(): String
DataCollectionDispatcher
Database
"Database";
connection: Connection
url: String
PdfGenerator
PdfGenerator():
generatePdf(String): void
getCutomerInfo(String): Customer
Customer
last_name: String
id: int
first_name: String
last_name: String
id: int
first_name: String
Customer(int, String, String):
PdfGenerator