Development Phase - Reflection Phase - Task 1

To start developing the data system, a GitHub-repository was created and linked to a corresponding GitHub-project where tickets can be drafted and progress can be tracked. After drafting a few tickets, development started by creating the initial .NET-project and adding the chosen dataset to the project. Before setting up the Docker integration and automation scripts, or even adding the PostgreSQL-container to the project, the database was drafted by analyzing the chosen dataset and the system requirements, creating the Entity Framework Core classes and generating the initial migration script. With database entities and migration script in place, the first gRPC-endpoint was created, and Protocol Buffers for this service were implemented. For testing purposes, the project creates client-side and server-side Protocol Buffers. Since basic functionality and one endpoint was present, the next step was the implementation of the Docker-integration and test the integration. The official documentation provides a basic starting point and a comprehensive explanation of how a Dockerfile is supposed to be used with a .NET-project. With a running .NET-container and existing database-structure, it was time to put the Docker-composition together. To achieve this, the project makes use of the `docker compose` command which enables dependent container structures. The .NET-project was set up to be dependent on the PostgreSQL-container and requires the database to be started before the data system. For initializing the PostgreSQL-container with the migration file, a database initialization script is generated with the `dotnet ef` utility and mounted into the PostgreSQL-container. The database container supports a script to be executed on first container start and initializes the database with the base structure. After combining two containers together, the python script still needs to be executed at startup. A third container was added that checks a file inside the virtual file system and determines if the container was already started and initialized. If no initialization happened yet, it loads the dataset from the CSV-file and batch inserts the data into the database. With a running database and existing gRPC-endpoint, as well as a simple API-controller, it was time to write missing texts and add some basic authorization. For testing, another Tests-project was created, and all needed dependencies were added to the project. In this project, the endpoints were tested intensively with multiple `DataRow` per test and debugged accordingly. The last step was the integration of Swagger and code-/endpoint-documentation, where code summaries and proto-annotations were used to automatically generate a JSON-based API-endpoint for the gRPC-service and a basic documentation. All the tickets have been converted to issues and linked to the pull-request that closed them.