Porteføljе opgave til Softwareteknologi i Cyber-fysiske Systemer

Navn: Jakub Bouzan

Email: jabou19@student.sdu.dk

Project idé

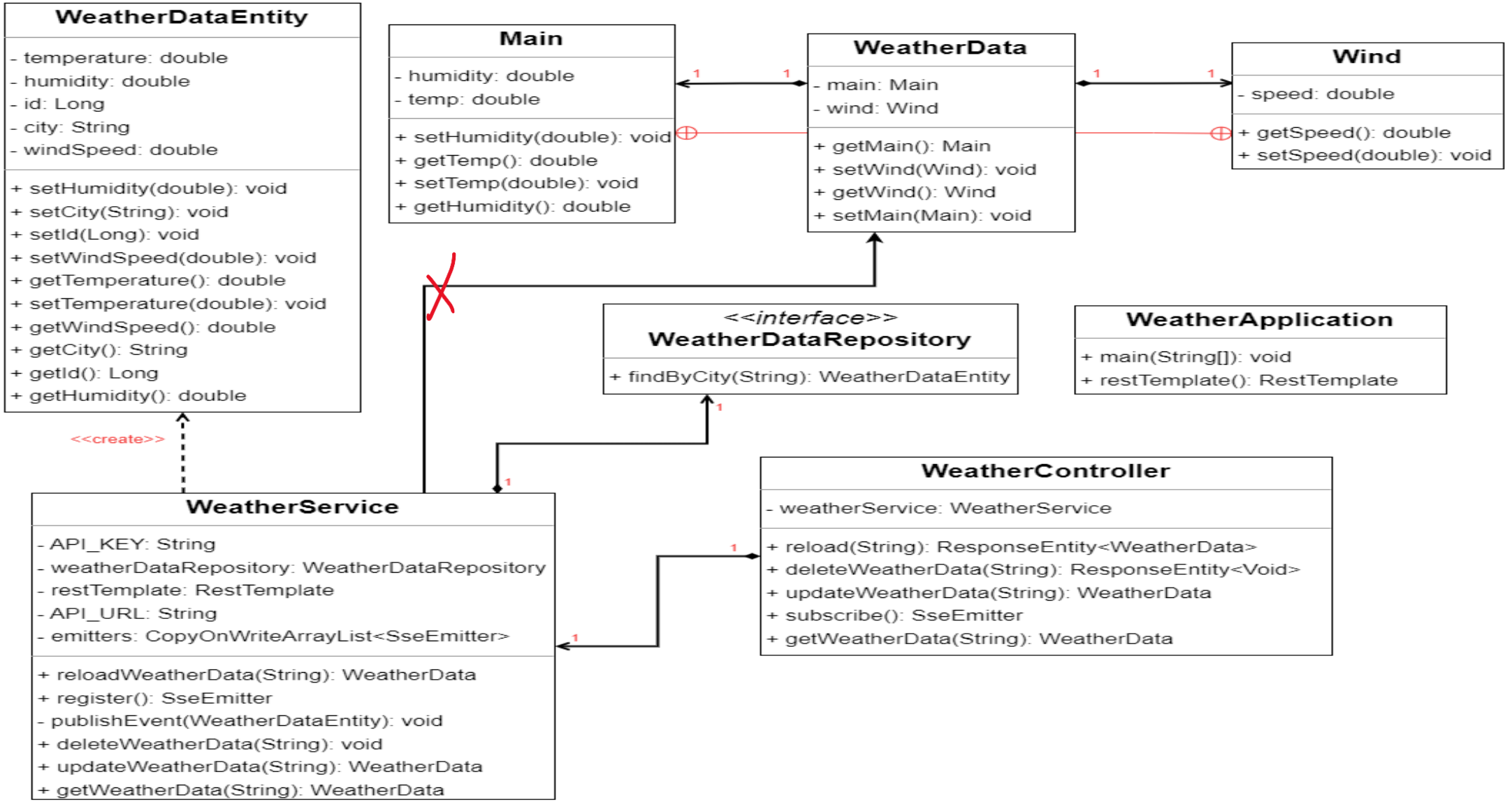
- The idea behind this project is to be able to find the weather information in a given city in the world. It is more specific the 3 pieces of information on the temperature, humidity and wind speed in the city at the given time. These will then be displayed visually in the form of a bar chart(søjlediagram).

- It is a Fullstack application where a user vists on the website, has the option to write, in a field, the name of the city they want information from. Mechanism By pressing the button, send a request from the front end and via the backend retrieve data from an external API, after which the results are presented visually via bare Chart graphs of the 3 pieces of information. The information is then stored in the database.
- When there is a change in the information from OpenWheather(API key), the new changes are updated and saved in the database and then it is also updated in the front end so that you can see the changes visually.

Hvordan skal appliktionen køres?

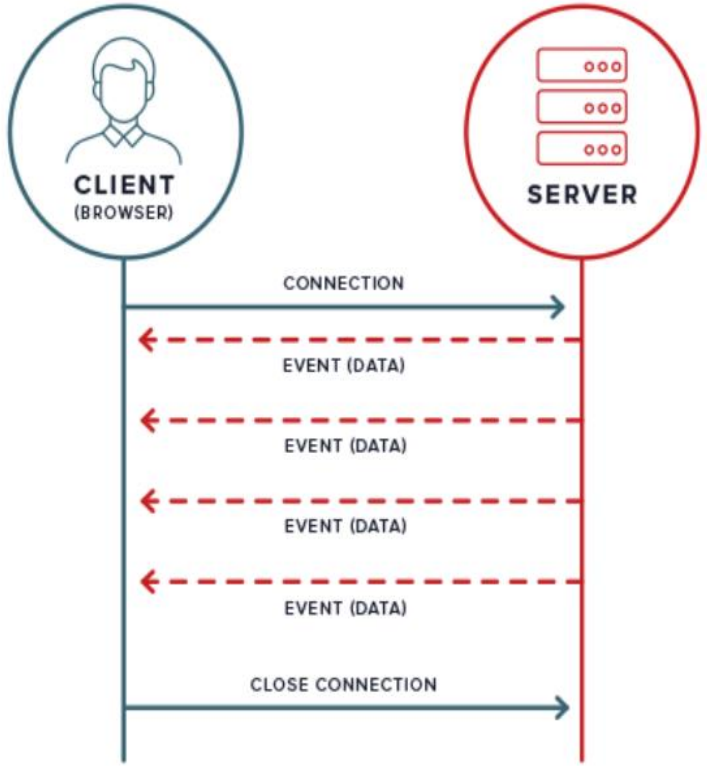
- (a Fullstack) All the 3 parts of the application ie. database, backend and frontend must be started separately, and it must be done in the following order.
- 1. Database: MySQL is set up via Docker. So therefore, man must start by having the docker app up and running, and then man must run docker-compose.
- 2. The backend is made in Spring Boot and must be started by pressing the start button, which can be found under the com/weather/service folder and then under the WeatherApplication class.
- 3. Finally, the frontend must be started up. React is used for the front end and it can be started by running "npm start" in the terminal.
- , React is a JavaScript library for building user interfaces

UML - klassediagram



Integration strategy

I am using No-Polling for my portefølje where it involves the use of Server-Sent Events (SSE) for real-time communication(the front end and via the backend). SSE is a unidirectional communication channel where the server (backend) sends data to the frontend when changes occur in the backend, without having to make any requests in the frontend



Design Pattern

We have several design patterns that are part of our code base:

Observer pattern: The WeatherService class maintains a list (emitters) that is used to send real-time weather updates to clients using Server-Sent Events (SSE). When the weather observations are updated, the publishEvent method is called to send the updated data to all registered emitters. This is an example of the Observer design pattern, where the WeatherService class is the subject and the SseEmitter instances are the observers.

```
// Publishes a weather data event to all registered SseEmitter instances, and sends the weather data entity
// as a server-sent event to each emitter.
private void publishEvent(WeatherDataEntity weatherDataEntity) {
    emitters.forEach(emitter -> {
        try {
            emitter.send(weatherDataEntity, MediaType.APPLICATION_JSON);
        } catch (IOException e) {
            emitter.complete();
            emitters.remove(emitter);
        }
    });
}
```

Repository Pattern: The WeatherDataRepository interface is an example of the Repository pattern. The Repository pattern acts as an intermediary between the objects and the database by abstracting access to the data source. The ORM framework handles the mapping to and from the underlying database.

```
@Repository
public interface WeatherDataRepository extends JpaRepository<WeatherDataEntity, Long> {
    WeatherDataEntity findByCity(String city);
}
```

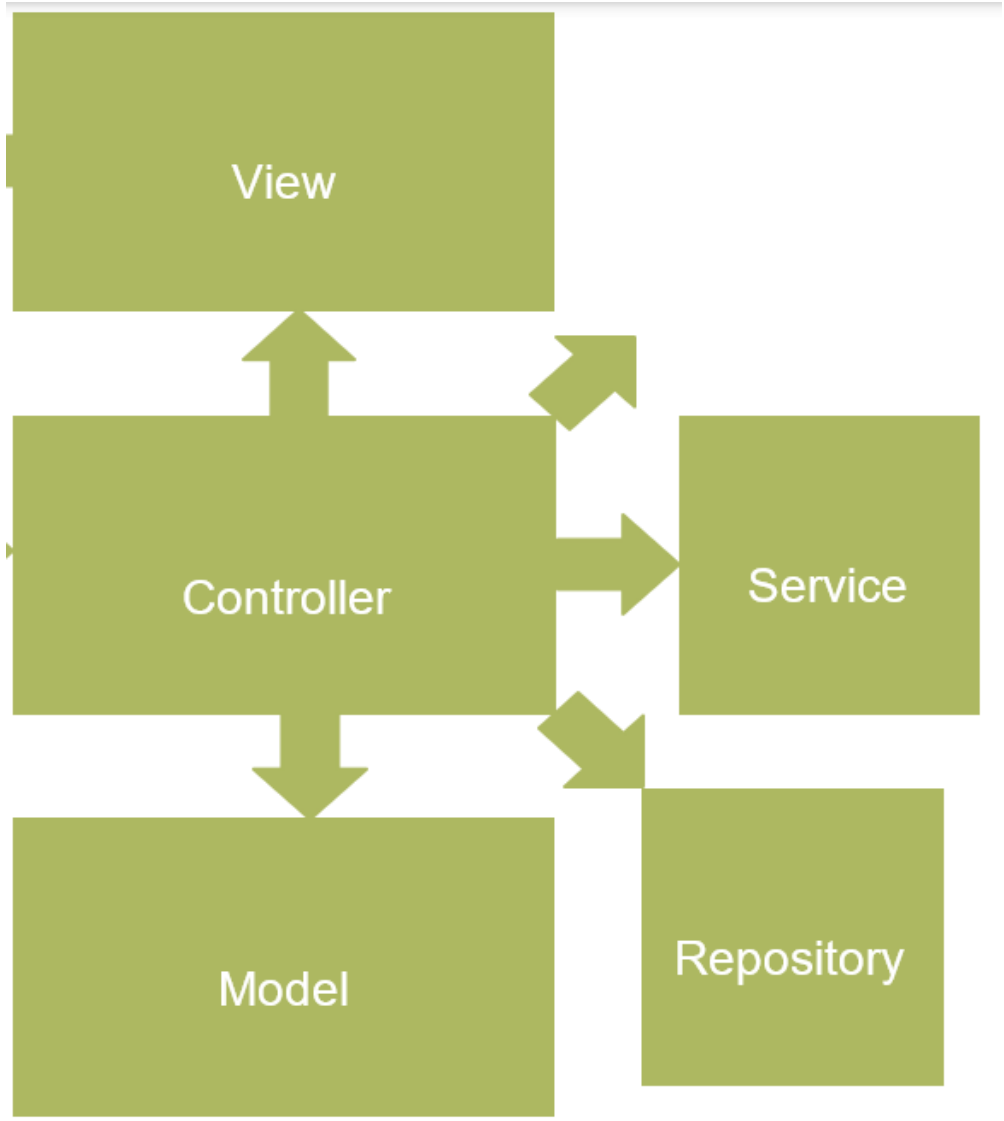
Software Architecture

MVC

I have chosen a variant of the Model View Controller software architecture for our full-stack application. We have something extra; we have **service and repository.**

This architecture divides the application into three connected layers: **the model** (data classes), **view** (user interface), and **controller** (handles user input and updates the model and view, orchestrates communication).

Service is used to implement the complex business logic.



Test

End to end:

I have chosen to do end-to-end testing to verify that our application works as expected.In this connection, we have used the Cypress framework, which is one of the more popular frameworks in java for this kind of tests.