avenga

Java CC Developer Task

01.07.2025

Prepared for: Jarosław Pawlik
Owner: Adrian Wąsik

Workflow for the Task

Presented application is just a basis. We will proceed step by step, adjusting the solution, incorporating changes, and introducing new technologies.

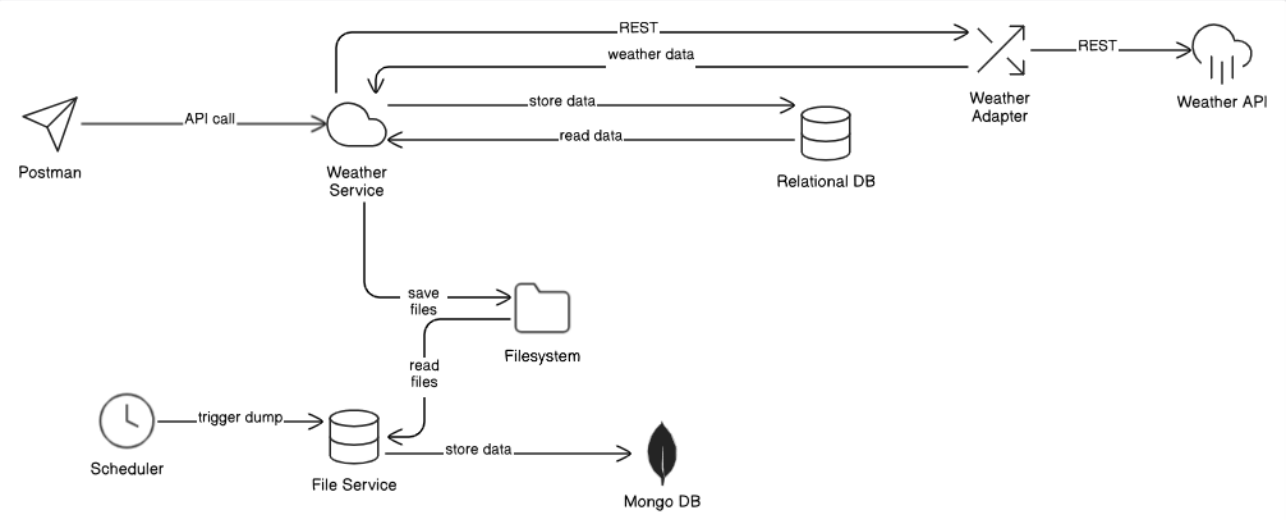
Let me know once you accomplish a specific step for review and further actions. You can always post me code for review.

Contact me in case of any problems or if you need any clarification or guidance.

General description

You will build an application, consisting of 3 separate microservices for querying weather API, processing its responses, fetching historical data, and archiving mechanisms.

Weather application diagram:



Functional requirements

1. I want to be able to use Postman to query **WeatherService** for weather conditions for current day and provided city and have the results in response.

2. I want my input city, query date and results retrieved saved for future reference.
3. I want to be able to retrieve my historical queries, their dates and weather results with optional filtering by city and dates range.
4. I want to be able to dump my results to file.
5. I want my dump files processed and saved to MongoDB on a regular basis.

Technical specifications

Technical details

- Each microservice is in its own project and with separate Maven POM.
- Configured via application.yaml with env variables.
- Each microservice exposing its REST interface documented with OpenAPI specification (Swagger).
- **WeatherService**
    - Calls WeatherAdapter using RestTemplate.
    - Saving city and response from WeatherAdapter to relational DB connection (DB engine of your choice) using Spring Data.
    - Moving data from relational DB to files in configured folder.
- **WeatherAdapter**
    - Call external weather API of your choice using RestTemplate.
    - Returns its response with no additional processing, parsing (full JSON).
- **FileService**
    - Every 10 minutes (Spring Scheduling) checks preconfigured folder for new files,
        - if any are available it reads and stores them in MongoDB collection and
        - deletes the files.
    - Expose REST interface to check the number of documents currently stored in Mongo collection.

Technology stack

- UML
- OpenAPI Specification
- Maven, Java 20+
- Spring:
  - Boot 3.5.3
  - Data (relational, Mongo)
  - Web
  - Scheduling
- REST Template
- Java I/O
- Junit, Mockito, Jacoco

Implementation approach

1. Think about your solution and design it.
2. Prepare skeleton (scaffolding) with empty methods calling each other.
3. Write all types of tests (unit, integration and E2E), adhering to testing pyramid. Reach test coverage at minimum 70%, check with Jacoco.
4. Fill the skeleton, implement business logic, validating it with prepared tests.
5. Verify the whole solution using Postman.

Steps

1. Architecture diagrams

Please prepare the following diagrams:

1. Use cases diagram,
2. Sequence diagrams (for: search process, archive query and dump request),
3. Components diagram.

2. API specification

Use OpenAPI design service e.g.: Apicurio preferably Apicurio WEB to prepare OpenAPI specification for each microservice.

3. Projects and REST interfaces

Prepare a project for each microservice using spring initializr or its adapter in IDE.

Use **org.openapitools.openapi-generator-maven-plugin** to incorporate generation of REST interfaces during build phase, based on prepared OpenAPI specification. It should give you classes to implement.

4. Prepare scaffolding

Prepare sceleton of the app with classes and empty methods calling each other.

5. Write tests

6. …