WEATHER FORECAST

Developed By

Duțu Alin Călin

CONTENTS

1 INTRODUCTION

1.1 Context

A web application, often called a web app, is a software application that runs on a web server and can be accessed via a web browser over the internet. Unlike traditional desktop applications, which are installed on a local computer, web apps are hosted on remote servers, making them accessible from any device with an internet connection. Web apps range from simple websites with basic interactivity to complex platforms that provide robust functionality, such as online banking systems, e-commerce platforms, and social media networks.

Considering the importance of such apps, Spring which is one of the most popular frameworks in the Java ecosystem for building robust, scalable, and maintainable web applications, has a big impact on the actual industry.

1.2 Objectives

This project aims to provide an overview of developing a backend application and serves as practice for testing Spring API along with other tools for backend development. The following objectives are taken into consideration:

- Create a database that manages at least 3 entities
- Create the layer in the backend application that connects to the database
- Implement multiple REST endpoints that will manage the entities from the app using different URLs and inputs

2 PROPOSED SOLUTION

2.1 Solution presentation

The project aims to implement a web app that registers and stores data about the weather in different cities and countries for a forecast. Knowing the project's purpose and complexity, the following architecture has been considered:

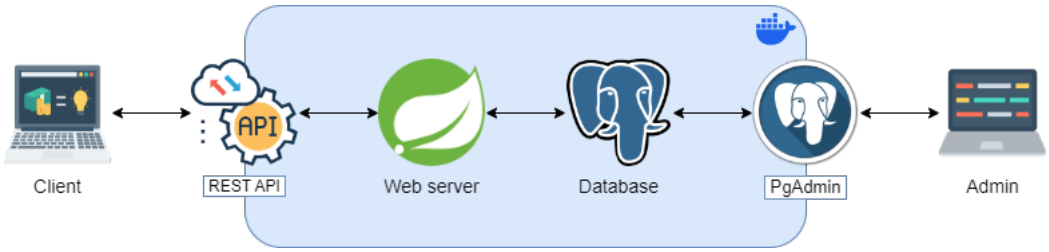

Figure 1: Architecture

There are 6 components identified in the architecture:

- Client - The device that uses the app through HTTP requests
- REST API - The middleware component between the client and the Web server
- Web server - The component that executes client's requests and returns the results
- Database - The component that stores data about weather and provides it when requested
- PgAdmin - The middleware tool to administrate the database
- Admin - The authorized device that administers the database

The client is any device that has access to a web browser or that can call the web server using HTTP requests. The client communicates with the web server using the REST API component, that can interpret HTTP requests received by the client and call the corresponding functions in the Web server.

The web server is an app developed in Spring that receives the HTTP request from the client, executes a series of commands that can interact with the database, and then provides an output that is sent to the client in an HTTP reply.

The database is an instance of a PostgreSQL[1] that stores the provided data. This storage

provides an API for administering the instance and its data using the Spring web server and the web browser to which an administrator can connect.

The web server and database components work as independent Docker services that communicate with each other in an isolated environment through a dedicated network.

3 IMPLEMENTATION DETAILS

3.1 Proposed implementation

3.1.1 Docker

As stated, the Weather Forecast app requires Docker to compile and deploy the components as isolated services that communicate internally within a dedicated network and externally with the client through the REST API component and with the admin using the pgAdmin tool.

The whole configuration is written in the Docker Compose file.

3.1.2 Database

The database service is used for storing data provided by the client and fetched at request. This service is also administered by the admin using the PgAdmin tool.

In Docker, this service is created with the image of a PostgreSQL database (postgres) and it comes with a persistent volume, the default 5432 port exposed internally in Docker, and access to an internal network to communicate with the spring app (spring-postgres) and one to communicate with the PgAdmin tool (postgres-pgadmin). Additionally, the service has some environment variables that define the database name and the default credentials for access to the database.

3.1.3 PgAdmin

The pgAdmin tool is used to connect and administer the PostgreSQL database through an internal network and can be accessed externally by an administrator.

In Docker, this tool is created using a pgAdmin image and it comes with a persistent volume, the external 5050 port that binds to the internal port 80, and access to the "postgres-pgadmin" internal network to communicate with the database. Additionally, the service has some environment variables that define the default credentials for access to the administration tool.

3.1.4 Spring app

The Spring App is the main backend component, also named as web server, developed to handle operations requested by the client. The entities stored and processed in the app are: Countries, Cities, and Temperatures. The dependencies between these entities that are applied in the database, but also considered in the app can be found below:

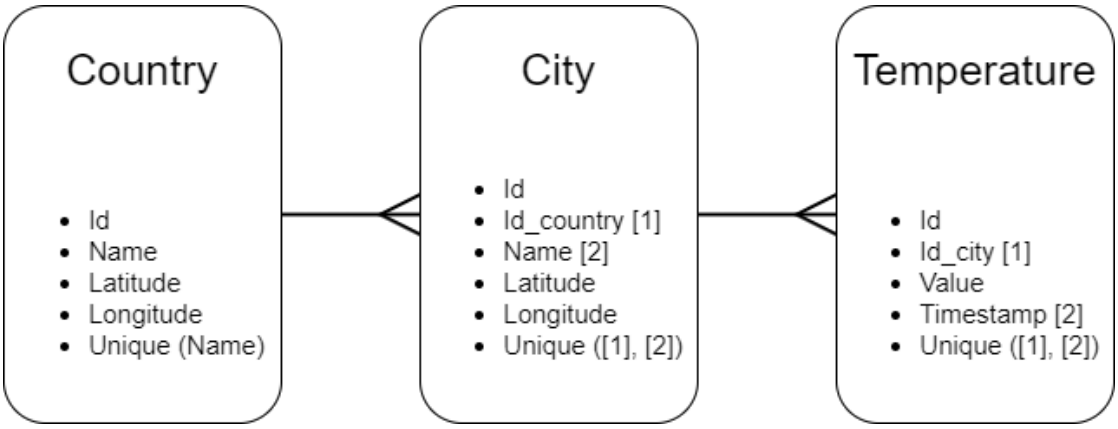

Figure 2: Entity dependency

REST API

The REST API is the middleware component between the client and the web server app. This component captures the HTTP requests sent by the client and lets the web server execute them. The output of the execution will be provided by the web server and sent back to the client as an HTTP reply. The following operations can be processed:

- Add a country
  - URL: "/api/countries"
  - Request type: POST
  - Payload: New country data
  - Results:
    - Success - HTTP Code 201
    - Data Conflict - HTTP code 409
- Get all countries
  - URL: "/api/countries"
  - Request type: GET

  - Results:
    * Success - HTTP Code 200 + List of countries

- Modify a country
  - URL: "/api/countries/id"
  - Request type: PUT
  - Payload: Country id + modified data
  - Results:
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404
    * Data conflict - HTTP Code 409

- Delete a country
  - URL: "/api/countries/id"
  - Request type: DELETE
  - Payload: Country id
  - Results:
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404

- Add a city
  - URL: "/api/cities"
  - Request type: POST
  - Payload: New city data
  - Results:
    * Success - HTTP Code 201
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404
    * Data conflict - HTTP Code 409

- Get all cities
  - URL: "/api/cities"
  - Request type: GET
  - Results:
    * Success - HTTP Code 200 + List of cities

- Get all cities from a country

  - URL: "/api/cities/country/country_id"
  - Request type: GET
  - Payload: Country id
  - Results:
    * Success - HTTP Code 200 + List of cities

- Modify a city
  - URL: "/api/cities/id"
  - Request type: PUT
  - Payload: City id + modified data
  - Results:
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404
    * Data conflict - HTTP Code 409

- Delete a city
  - URL: "/api/cities/id"
  - Request type: DELETE
  - Payload: City id
  - Results:
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404

- Add temperature
  - URL: "/api/temperatures"
  - Request type: POST
  - Payload: New temperature data
  - Results:
    * Success - HTTP Code 201
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404

- Get the temperatures from a location over a certain time interval
  - URL: "/api/temperatures"
  - Request type: GET
  - Payload: Latitude, Longitude, Date interval
  - Results:
    * Success - HTTP Code 200 + List of temperatures

- Get all temperatures for a city over a certain time interval
  - URL: "api/temperatures/cities/city_id"
  - Request type: GET
  - Payload: City id + Date interval
  - Results:
    * Success - HTTP Code 200 + List of temperatures
    * Not Found - HTTP Code 404

- Get all temperatures for a country over a certain time interval
  - URL: "api/temperatures/countries/country_id"
  - Request type: GET
  - Payload: Country id + Date interval
  - Results:
    * Success - HTTP Code 200 + List of temperatures
    * Not Found - HTTP Code 404

- Modify temperature data
  - URL: "/api/temperatures/id"
  - Request type: PUT
  - Payload: Temperature id + modified data
  - Results:
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404
    * Data conflict - HTTP Code 409

- Delete temperature data
  - URL: "/api/cities/id"
  - Request type: DELETE
  - Payload: Temperature id
  - Results
    * Success - HTTP Code 200
    * Bad request - HTTP Code 400
    * Not Found - HTTP Code 404

Additionally, there are other libraries used in the project that made developing much easier:

- postgresql - library that provides an adapter for postgresql database
- lombok[2] - for providing constructors using tags only
- mapstruct[3] - for converting instances of objects to entities and back

The image for this app is created using a Docker file that compiles, builds, and packs the app as a Docker image with the default port 5000 exposed using Maven[4]. This image will be used in Docker Compose to create the service along with an exposed 8080 that points to the internal service and access to the "spring-postgres" network to let the app set the connection to the PostgreSQL.

The connection to the database is set using the Hibernate[5]. The connection parameters are set in a resource file where the database URL and credentials are found along with other SQL properties.

3.2 Testing

For testing the proposed solution, the following methods have been used:

- Postman - Used for testing the REST API using the general cases
- Manual testing - Used for testing corner cases

4 CONCLUSIONS

In conclusion, this program aims to provide an overview of backend development experience by developing and deploying a full backend app for weather Forecasts. The app serves as practice for discovering multiple backend tools such as Spring, PostgreSQL, and other libraries that have helped implement the project at a very fast pace, along with popular deployment tools such as Docker that were used to host the required services for the app.

BIBLIOGRAPHY

[1] The PostgreSQL Global Development Group. PostgreSQL Database. https://www.postgresql.org/. Latest access: 01.09.2024.

[2] Roel Spilker and Reinier Zwitserloot. Project Lombok. https://projectlombok.org/. Latest access: 01.09.2024.

[3] Map Struct - Java bean mapping the easy way! https://mapstruct.org/. Latest access: 01.09.2024.

[4] The Apache Software Foundation. Apache Maven. https://maven.apache.org/. Latest access: 01.09.2024.

[5] Commonhaus Foundation. Hibernate. Latest access: 01.09.2024.

[6] SPRC team @ Polytechnic University of Bucharest. REST API & Docker. https://archive.curs.upb.ro/2022/pluginfile.php/258913/mod_folder/content/0/SPRC_2022_2023_Tema_2.pdf?forcedownload=1. Latest access: 01.09.2024.