Final project for Service Design and Engineering

Letizia Girardi

1. INTRODUCTION

The project **WebArticles** offers an advanced service for efficiently managing products. This includes the integration of independent services to facilitate the acquisition of promotional offers and pricing information. The service will offer valuable insights for available articles, associated promotions, and the most favorable pricing options.

The submitted composition of services is presented in detail in §3, while their implementation is outlined in §4. Section 2 below portrays the experience of a user making use of the service provided.

The implementation has been submitted and is also available at `https://github.com/letiziagirardi/WebArticles`.

2. THE WORKFLOW

A user of the WebArticle service can perform the following workflow:

1. Sign up for an account.
2. Log in to the platform.
3. Depending on the user's role (admin/user/admin and user), they can perform the following actions repeatedly:

   a) Execute a range of associated actions, which encompass:

   - Inserting a new user.
   - Searching for a specific user.
   - Retrieving information about all users.
   - Deleting a user.
   - Finding an article using its Ean.
   - Locating an article through its CodArt.
   - Searching for an article based on its description.
   - Adding a new article.
   - Modifying an existing article.
   - Removing an article.
   - Searching for the price associated with a specific CodArt.
   - Inserting a price if not already present, or modifying it if it exists.
   - Exploring all available promotions.
   - Searching for a promotion by its unique ID.
   - Discovering promotions available for a particular article.
   - Adding a new promotion.
   - Removing a promotion.

This comprehensive set of actions is designed to offer users a versatile and efficient experience within the WebArticle service. Overall, five services and one command line interface have been developed.

3. DESIGN OF SERVICES

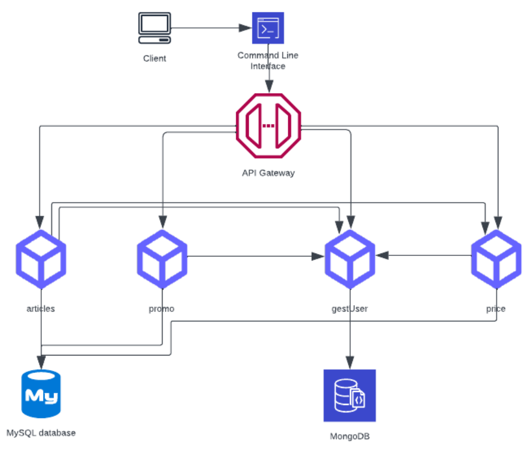

Figura 1. Structure and interactions of the services.

The service employs a **microservices architecture**: instead of building a single monolithic application, its functionality has been divided into separate services that can be developed, deployed, and scaled independently. This approach offers benefits such as improved scalability, easier maintenance, and better isolation of concerns.

Within the microservices architecture, distinct services operate to fulfill specific roles:

- Articles Service: Manages articles and interacts with the price and promo services to provide article information and related price.
- Promo Service: This service handles promotional activities, providing active promos.
- Price Service: Deals with price-related information. Receives requests from the articles service.

- Gestuser Service: This service is responsible for user authentication and authorization, providing access control to other services. Moreover, it is possible to create and managing users' profiles.

In order to simplify client interaction, handle authentication and authorization across all microservices and distribute incoming requests across multiple instances of services, the API Gateway service has been implemented. It efficiently routes incoming requests to the appropriate microservice based on endpoints and HTTP methods. This enables clean separation of concerns for clients, as they don't need to know the individual microservice URLs. Clients now engage through a sole entry point, accessed via `http://localhost:8080`, which approach simplifies interactions with the various microservices. Within this framework, clients interact with the API Gateway, which orchestrates communication with the underlying microservices through internal requests:

- When a client makes a request to the API Gateway, the Gateway translates this into requests that are sent to the relevant microservices.
- Microservices process these requests and return responses to the API Gateway.
- The API Gateway aggregates the responses if needed and sends an appropriate response back to the client.

In Figure 1 there is a depiction of the developed services and of their interactions.

A. Logical Layers

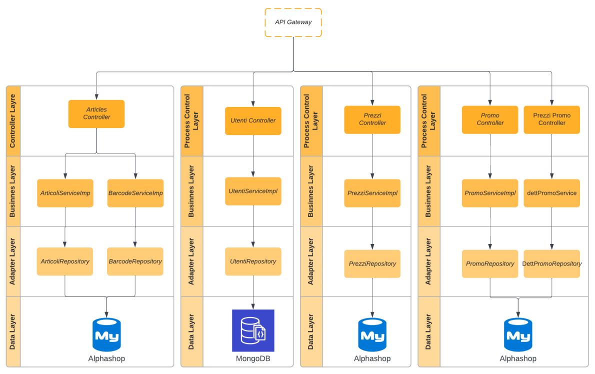

Figura 2. Logical Layers of the services.

Since each of these microservices has been developed as individual Spring Boot applications, the follow structure has been implemented:

- Controller Layer / Presentation Layer: Exposes RESTful endpoints to handle incoming requests.
- Service Layer / Businnes Logic Layer: Implements business logic, orchestrates operations, and interacts with repositories.
- Repository Layer / Adapter Layer : Manages data storage and retrieval.
- Database Layer / Data Layer: The database layer contains all the useful databases: MySql and MongoDB. It is responsible for performing the CRUD operations.

B. The CLI

Eventhough not functioning as a standalone service, the CLI client serves as the primary interface for interacting with the broader service, specifically as the designated client for WebArticle. This does not exclude the possibility for users to directly initiate HTTP requests to the WebArticle gateway. The design of the CLI client is oriented towards delivering a user-friendly experience to end-users.

4. IMPLEMENTATION OF SERVICES

Five services have been developed in Java. Each service is stored in a separate folder in the project, and can be run in a Docker container; see §4.B below for details on building and running the provided containers.

All of these services have been developed utilizing the open-source Java Spring Framework, with a specific focus on Java Spring Boot—an extended module of the framework. In terms of authentication and authorization for Java applications, the implementation leverages the Spring Security framework.

The Command Line Interface, instead, has been developed in Python. Read `example-run.txt` file for a comprehensive guidance on running the interface.

A. Structure of the Submission

Inside the `version2` folder of the submission, you'll find a well-structured arrangement of folders, here what you'll find:

- **Java Web Projects**: Each service has a Java package that contains the actual source code for that service. This code likely implements the functionality of the respective service. Alongside the Java package, there is a `configuration file`. This file contains various settings and parameters required for the proper functioning of the service. Moreover, a `Dockerfile` is present for each service. This file defines the instructions for building a Docker image of the service.
- **docker-compose.yml**: The docker-compose.yml file is used for orchestrating the deployment of

multiple Docker containers. It defines how the various services should interact and how they should be configured when deployed locally using Docker Compose.

- **Databases**: There are two folders, one named `mysqlData` and the other named `database`. `mysqlData` contains the data related to users information. The database folder, instead, contains data related to articles and promotional content.
- **shop.sh**: A Bash script, providing a commandline interface (CLI), with the goal of automating the build, deployment, and management of a set of services in a distributed application environment.

B. Docker

Each service is exposed through a separate Docker container. Each of them is listed in the `docker-compose.yml` file at the version2 folder of the project repository, so as to ease the building and running of the relevant Docker images.

For comprehensive guidance on launching the services, refer to the instructions detailed in the `README.md` file. This document outlines the necessary steps to initiate and manage the services effectively.

C. Authentication and Authorization

Authentication and authorization are managed by the `gestuser` service due to its direct connection to the database. This proximity ensures efficient handling of authentication processes. The authentication process is carried out as follows:

1. A client performs a request and provides a value in the Authorization header (e.g. Bearer abc123...);
2. The requeired service receives the request and before fulfilling it, sends a simple static request to the gestuser service;
3. If the token is invalid:
   a) The Gestuser responds with an HTTP status of 401 Unauthorized; and
   b) The WebArticle responds with the same status to the client;
4. If the token is valid:
   a) The Gestuser responds with an HTTP status of 200 OK and provides information about the user including his role (admin/user); and
   b) The required service proceeds with handling the request.

Authorization has been established through the conventional method of utilizing a bearer token, furnished to the client upon successful completion of the password challenge. Certain endpoints need basic authentication, while others require higher user privileges, such as administrator status, to access enhanced functionality.

For user convenience, the CLI client retains the username, his role and the associated token within a configuration file after a successful execution of the login command. In this way, users do not need to re-insert their credentials for every request.

D. Inter-service Communications

In the proposed microservices architecture, communication between services is facilitated through web APIs using HTTP REST calls and JSON data exchange. Typically, to ensure compatibility and ease of parsing, JSON is the preferred data format for both requests and responses. Since the microservices have been implemented through Spring Boot, the RestTemplate class has been employed to make synchronous HTTP requests to other microservices' RESTful APIs. This has streamlined data exchange and promotes effective communication.

E. API Documentation

In addition to their APIs, each service offers a dedicated API documentation. To access this documentation, simply navigate to the `/swagger-ui` page of the respective service. For instance, to explore the API documentation for articles service, visit `http://localhost:8080/articles/swagger-ui`. Notably, the API documentation has been crafted using Swagger, ensuring a comprehensive and intuitive reference for developers.