École polytechnique de Louvain

LINFO2145
Cloud Computing

Année académique :
EPL Master 1

Auteur :
Bouvencourt Thomas 42341900
Gombeer Charles 31541900

29 novembre 2023

TABLE DES MATIÈRES

1 PROJECT

For Part 1 of the project, we successfully created five backend services (users, cart, order, product, log). We ensured that the front end sends different requests for each service. These requests are then handled by a gateway (NGINX) to route them to the appropriate microservices. Our application is now dynamic and persistent. When a user leaves their session, they can resume it at any time. Even though it was not required for this part, we deployed our project on Azure, and it is working perfectly. Due to this, we made slight modifications to the code to adapt to the criteria of the instructions not to push it to Azure yet. This has resulted in a slight increase in the number of commands needed to launch the project.

2 HOW TO RUN

- 1 : You need to change 4 files (In practice, when deploying our project to Azure, we do not need to modify these lines as it automatically considers the DNS URL. For this specific push, we found this workaround solution.) :
  - Store_scapp/front-end/src/routes/admin/+page.svelte : line 22 by your IP host
  - Store_scapp/front-end/src/stores/auth.ts : line 4 by your IP host
  - Store_scapp/front-end/src/stores/product.ts : line 4 by your IP host
  - Store_scapp/front-end/src/stores/carts.ts : line 4 by your IP host
- 2 : Then, you need to create docker swarm :
  docker swarm init –advertise-addr IP_OF_YOUR_HOST
  You will also need to create the docker newtork if it hasn't been created :
  docker network create –driver overlay –attachable scapp-net
- 3 : In the same directory as Makefile(root) => make deploy
- 4 : (Optionnal) you can init the store with this commande : curl -X POST localhost/product/initOrder (In general in our code you cant modify the DB if your are not an admin.

3 THE BACK-END SERVICE

3.1 Back-end microservices are accessed using REST APIs

3.2 State dynamic

All application state is centrally managed in the backend system. Users, when connecting from different browser sessions, experience continuity as their previous actions and data are seamlessly preserved, ensuring a consistent and reliable user experience across sessions.

3.3 User-service

3.3.1 Info

The user service is responsible for managing the registration and login processes. It facilitates the addition of user information to the CouchDB database, ensuring data persistence. Through this service, the registration and login processes become dynamic and persistent. This means that users can exit the application and return later, while retaining their user information associated with their account. This approach ensures a seamless user experience, allowing users to resume their activity where they left off, contributing to a consistent and uninterrupted user experience.

3.3.2 App

— (POST) : /user/create/

— (POST) : /user/authenticate/

— (GET) : /user/validate/

— (GET) : /user/level/

3.4 Product-service

3.4.1 Info

The product service is designed to transform the catalog from a static JSON into a dynamic one. Previously, the catalog was a static JSON file. With the product service, a JSON file with the same format is dynamically generated based on the CouchDB database associated with this service. This enables the retrieval of all items from the database and organizes them into a standardized 'catalog' format.

Additionally, the product service provides functionalities to add, delete, and update items in the catalog. Furthermore, it allows the initialization of the catalog with basic items. This service plays a crucial role in maintaining the catalog's dynamic nature, ensuring that it accurately reflects the contents of the CouchDB database and offering the flexibility to manage and update catalog items seamlessly.

3.4.2 App

— (POST) : /product/createItem/

— (POST) : /product/delItem/

— (POST) : /product/updateItem/

— (POST) : /product/initOrder/

3.5 Cart-service

3.5.1 Info

The cart service is responsible for recording and managing carts associated with a user. When a user adds items to their cart, the service ensures that the cart is 'saved.' This means that the contents of the cart are stored persistently so that the user can resume their shopping later and continue where they left off. This ensures that users have a seamless shopping experience, allowing them to retain and continue their cart contents during subsequent visits.

3.5.2 App

— (GET) : /cart/get/ :

— (POST) : /cart/create/ :

— (PUT) : /cart/update/ :

— (PUT) : /cart/add/ :

3.6 Log-service

3.6.1 Info

The log service is responsible for recovering and saving logs sent by other services. The logs save : the user at the origin of the log ("Guest" if is not logged in), the user level (0 for Guest, 1 for normal users and

2 for admins), the type of log (INFO or ERROR), the name of the service associated with the log (user-service, cart-service, order-service, product-service), the operation producing the log (e.g. "AddItemCart") and finally more precise details on the log, such as the database response or the error if there is one.

3.6.2 App

— (POST) : /log/create/

— (GET) : /log/

3.7 Order-service

3.7.1 Info

The order service is responsible for recording orders placed by logged users into the database. It therefore allows users to retrieve orders.

3.7.2 App

— (POST) : /order/create/

— (GET) : /order/get/

3.8 Gateway

Nginx manages the API gateway. To do this, it redirects requests sent on port 80 to the corresponding services. The corresponding services are identified by the endpoint. For example, a request to http ://-host.com/user/ will redirect to the user-service service. There are five endpoints for the five services.

— /cart/

— /user/

— /product/

— /log/

— /order/

The gateway also provides access to the database interface for the various services. To do this, access port 8080/ and then the service name (e.g. http ://host.com :8080/user/_utils to access the user-service database).

4 THE FRONT-END

In our system, the entire frontend sends requests to our gateway, which then directs the requests to the appropriate service based on the nature of the request.

5 PRIVILEGE

On our website, we have implemented a privilege filter to manage different levels of access, including roles such as administrators. This privilege filter regulates access to various sections or features of the site based on the user's role or permissions.

In addition to the privilege filter, our entire website operates under a token-based security system. This means that user authentication and authorization are facilitated through the use of tokens. After a successful login, users are assigned a token, which contains information about their identity and privileges.

Tokens serve as a secure means of verifying a user's authenticity and determining their level of access. With each request made to the server, the token is validated to ensure that the user has the necessary permissions. This two-tiered approach—utilizing both privilege filters and tokens—provides a robust security infrastructure.