Application of Big Data: TP

Database:

First, we build our postgres server with :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

To make sure that our data can not be easily accessed by anyone, we put our login data in a .env file :

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Now that we have our files, we build our server with :

* docker build -t Linkaon/list .
* docker run -e POSTGRES\_PASSWORD=your\_secure\_password -p 5432:5432 Linkaon/list

Now our server is running.

We need to run it again, but with Adminer. So, in order to do that, we first need to create a docker-compose file :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

After that, we create a network to enable a connection between the database and Adminer:

* docker network create app-network

We finally create a database and an Adminer, link with a network. Next thing to do is to use our database, by creating some sql file :

* 01 – CreateScheme.sql :

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

* 02 – InsertData.sql

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

We put these files in a script folder, and we re-run our image.

We notice that our data are reset when we destroyed the container. So, we modify the docker-compose file in order to create persistent data :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Now, we have a database with persistent data that will not be erase when we destroyed the container, an Adminer to manipulate the data and a network between them.

Backend API :

For our backend, we will use a simple java class :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

We compile our class and we create a Dockerfile associated with this :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Now that we have our first step for our API, we need to improve it. So, we deploy a Springboot application in order to have a full backend API:

Une image contenant texte, Police, blanc, algèbre

Description générée automatiquement

With our Springboot generated, we add it to our project. In this Springboot, we add a controller class :

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Now that our controller is created, we create our Dockerfile for the backend:

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

So far, we have our database and our backend that can be running together.

HTTP Server :

In order to have a frontend to our app, we need to create a landing page to display our data.

We create out index.html :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

This page only exist to display the data present in our database.

Now that we have the frontend, the backend and the database, we need to connect them altogether in order to make our application running well.

In our project, we create a docker-compose file :

Une image contenant texte, capture d’écran

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

Now, when we run the whole application, we can see our data display in our landing page, thanks to the main.java file.