



une école de la



Rapport Architecture & Micro-services
NetFloux

Yonnel NYOKAS - Antonio VILLAZON - Yassine GOUMBARK

Sommaire

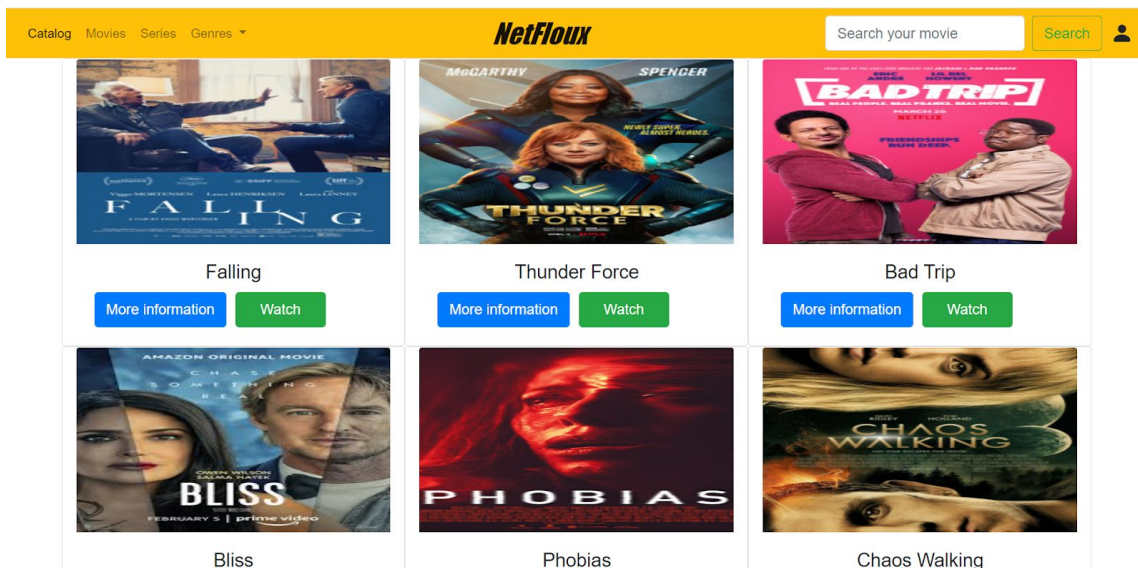
Conception de l'architecture orientée micro-services	3
Développement du site web	3
Les Web Apis	4
Les difficultés rencontrées	5
Conteneurisation des services : recherche de solutions	8
Vérification du bon fonctionnement de nos services	8
Conteneurisation et l'accès aux services conteneurisés	9
Conclusion	10

Conception de l'architecture orientée micro-services

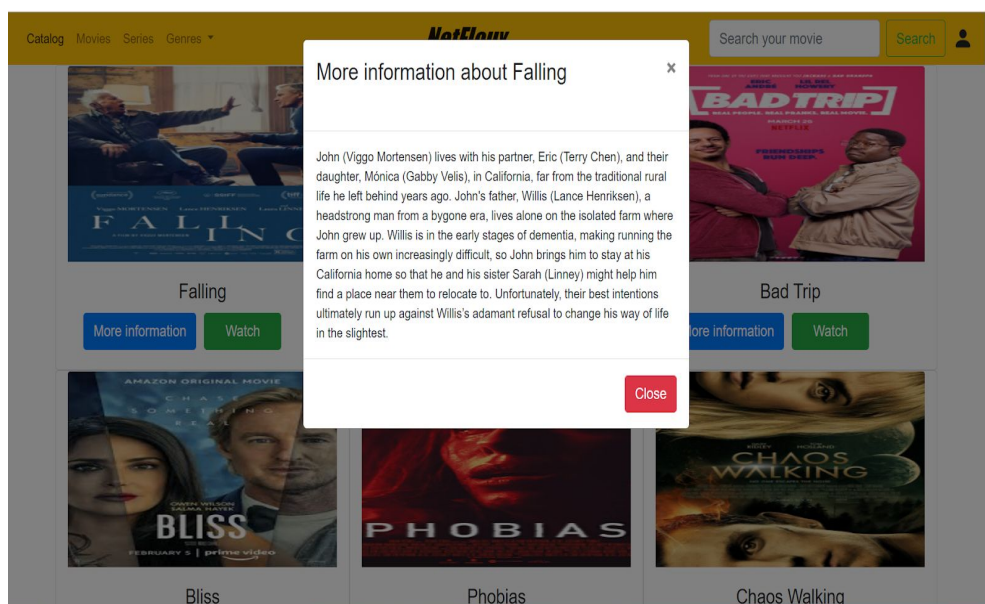
cf. prez_ARCHITECTURE_ET_MICROSERVICES.pdf

Développement du site web

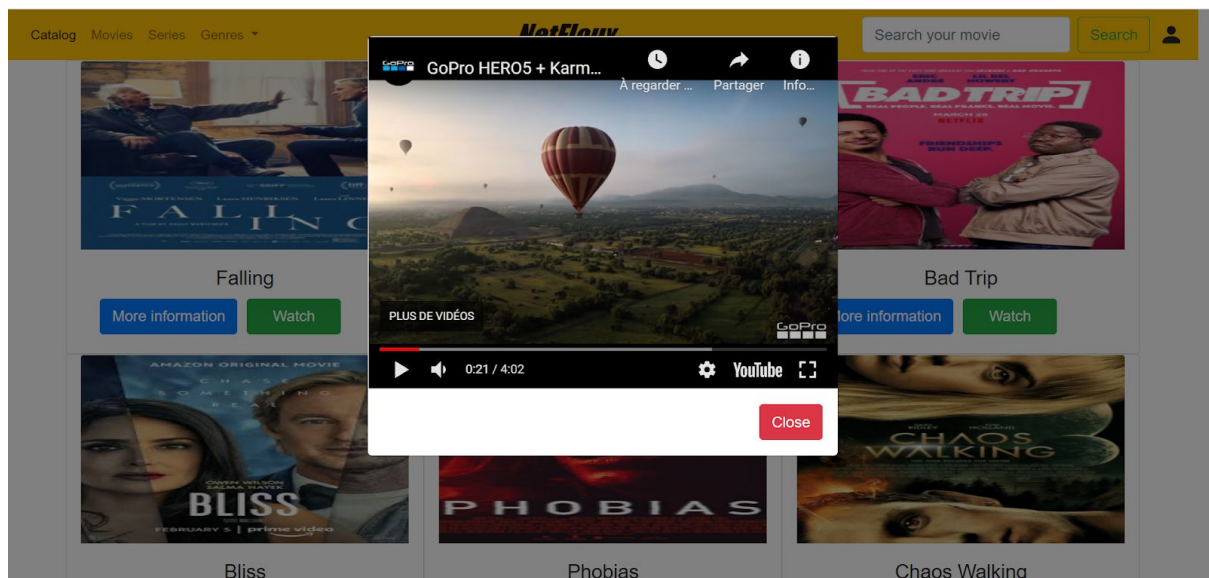
Pour la partie front, nous avons décidé d'utiliser Angular car nous avons déjà eu l'occasion de développer avec ce framework en entreprise. Après avoir conçu l'application, notre front donne le résultat suivant :



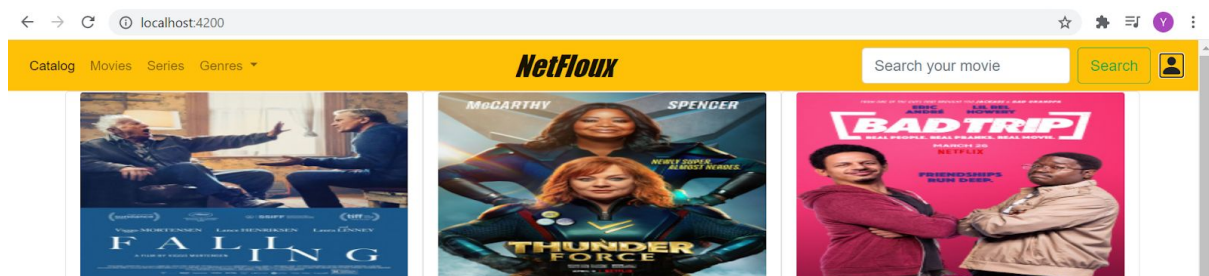
Lorsque l'on clique sur le bouton "More Information" d'un des films, une pop-up s'ouvrir et affiche des informations sur ce film :



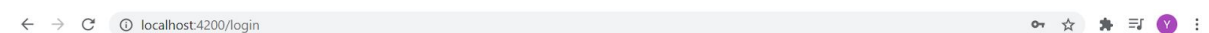
Ensuite, le bouton play sert à lancer le film de notre choix (NB: pour des raisons d'autorisation, nous n'avons pas pu intégrer les liens Youtube pour chaque film) :



Concernant l'authentification, nous avons voulu implémenter une authentification sécurisée JWT qui utilise des tokens permettant d'échanger les informations de manière sécurisée, mais nous n'avons pas abouti à cet objectif à cause de plusieurs bugs sur lesquels nous avons passé beaucoup de temps afin de les régler. Par conséquent, nous avons implémenté une authentification basic qui permet à un utilisateur de s'authentifier :

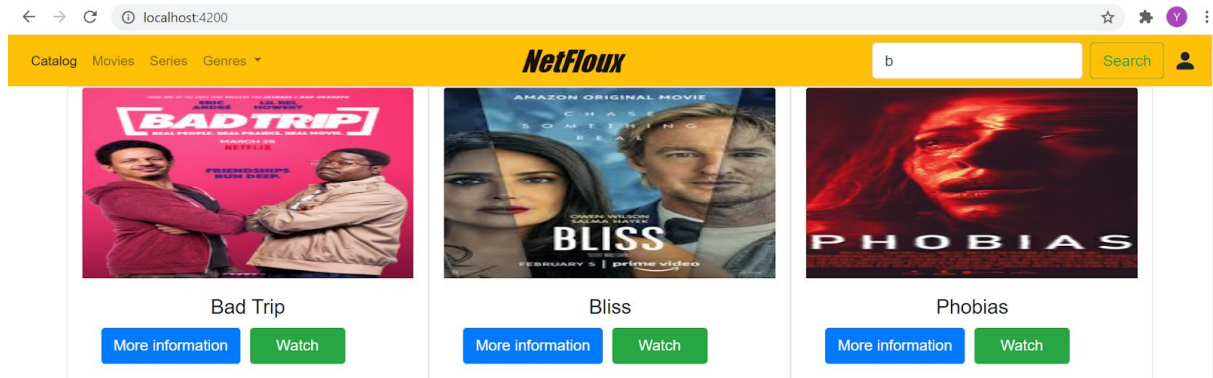


Lorsque l'utilisateur clique sur le bouton (haut à droite), il est redirigé vers la page d'authentification "<http://localhost:4200/login>" :

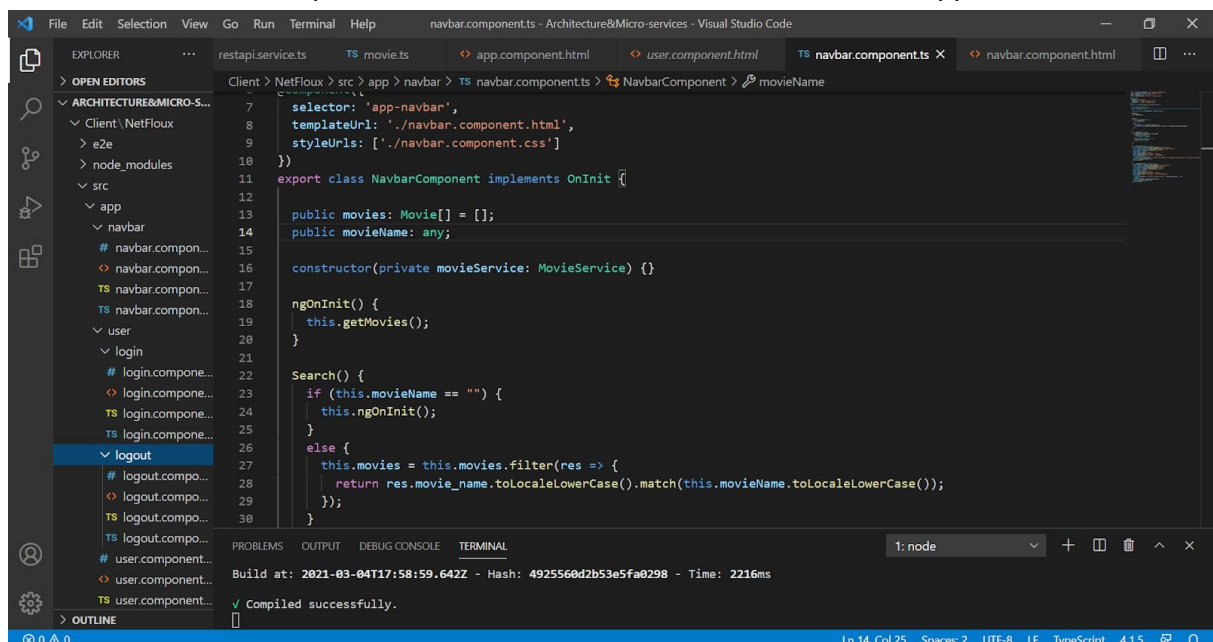


NetFloux

Par ailleurs, nous avons réussi à réaliser la fonctionnalité de recherche. Par exemple, si un utilisateur recherche un film qui commence par la lettre b. Le site lui affichera non seulement les films dont le nom débute par un b, mais aussi ceux qui contiennent la lettre b :

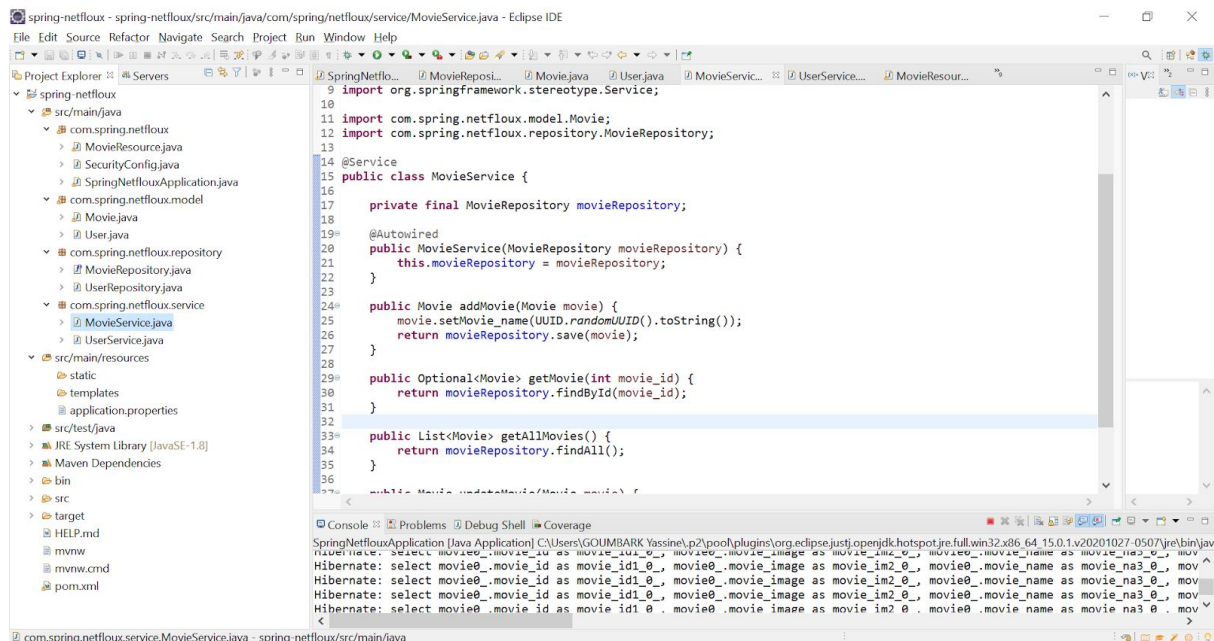


Voici ci-dessous une capture d'écran de notre environnement de développement front :

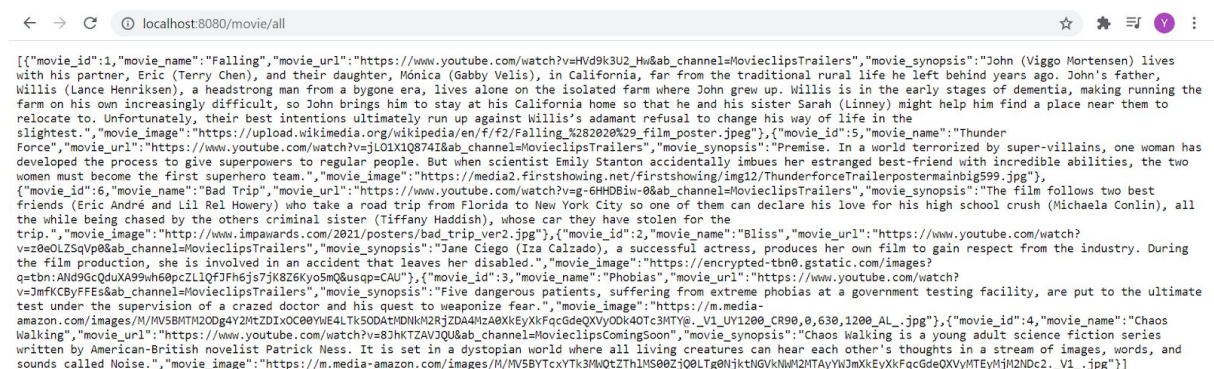


Partie SpringBoot :

Passons maintenant à la partie Back-End. Pour cette partie, nous avons fait le choix d'utiliser le framework SpringBoot avec Eclipse comme IDE :

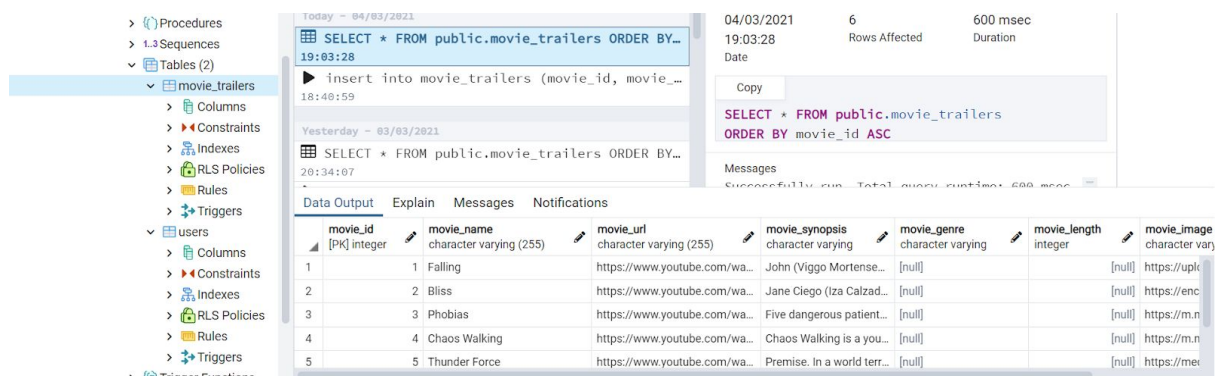


Nous avons réussi à extraire les données qui se trouvent dans notre base de données postgres afin de les afficher sur le front. Voici un exemple de données extraites :



Partie PostgreSQL :

Pour la partie base de données, nous avons opté pour PostgreSQL. Ci-dessous, nous avons notre base de données contenant deux tables **movie_trailers** et **users** :












Les Web Apis

En parallèle avec le développement du Front Web sous Angular, on a voulu faire 6 micro-services :






- **Catalog** : Ce micro-service Catalog a été développé pour d'afficher l'ensemble des films disponibles dans la base de données.
- **Recommandation** : afin d'afficher les films ayant une bonne réputation en fonction de la notation obtenue (ici on a mis pour tous les films ayant plus de 3.5 de note en tant que note fixe).
- **Search** : dans ce micro-service, on gère toutes les actions qui serviront à rechercher un film en fonction de son titre ou un ensemble de film en fonction du genre, de la date de parution, la notation (bonus car un peu similaire à recommandation).
- **Films** : dans Films on s'occupe de toutes les actions liées à la gestion des objets films stockés dans la base de données. Elle nous permet d'ajouter un nouveau film, de supprimer un film, de modifier les infos concernant un film comme sa note ou encore son URL (on a imaginé le cas où l'URL ne serait plus active et devrait être remplacée).
- **Profils** : Le micro-service Profils nous permet de faire des actions un peu similaires à celles de Films, mais sur les profils utilisateurs. On peut afficher les login et noms des utilisateurs, ajouter un utilisateur dans la base de données, supprimer un utilisateur et modifier le mot de passe.
- **Credits** : C'est un micro-services qui ne fait que de l'affichage, il renvoie vers une simple page HTML recensant nos noms en tant que développeurs de ce projet, ainsi que les versions des différentes applications et technologies utilisées pour réaliser ce projet

Après avoir testé tous les web api indépendamment les unes des autres, nous sommes passés à la partie docker afin de pouvoir déployer l'ensemble de l'application. Voici à quoi ressemble l'architecture de notre projet en comptant le docker-compose.yml et les Dockerfiles :

Nom	Modifié le	Type	Taille
 docker-compose.yml	04/03/2021 17:38	Fichier YML	3 Ko
 backup.zip	04/03/2021 16:50	Dossier compressé	6 Ko
 compose working with url pb.JPG	04/03/2021 00:00	Fichier JPG	195 Ko
 Catalog	03/03/2021 21:56	Dossier de fichiers	
 Profils	03/03/2021 19:20	Dossier de fichiers	
 Search	03/03/2021 19:18	Dossier de fichiers	
 Recommandation	03/03/2021 19:17	Dossier de fichiers	
 Credits	03/03/2021 19:10	Dossier de fichiers	
 Films	03/03/2021 19:08	Dossier de fichiers	

Nous avons à la racine le docker-compose.yml et les répertoires correspondants à nos micro-services. Dans chaque répertoire se trouvent un script python, un fichier requirements.txt pour indiquer les librairies/dépendances à installer dans le conteneur et un Dockerfile afin de définir les actions à faire pour créer les images, puis les conteneurs de ces micro-services.

» Documents » Docker_project » project » Catalog

Nom	Modifié le
 Dockerfile	04/03/2021 17:52
 films_catalogue.py	03/03/2021 23:25
 moviesDB.db	03/03/2021 17:31
 profilsDB.db	03/03/2021 18:10
 requirements.txt	03/03/2021 22:50

Dans le répertoire, étant donné qu'on voulait d'abord voir si on pouvait faire fonctionner l'ensemble des web apis dans des conteneurs sans prendre en compte le front dans un premier temps, on a 2 bases de données moviesDB.db et profilsDB.db sur lesquels on a testé nos apis.

Les difficultés rencontrées

Sans la partie docker, chaque web api indépendamment les unes des autres fonctionnait (il suffit de mettre la base de données sur laquelle le script travaille dans le même répertoire que le script).

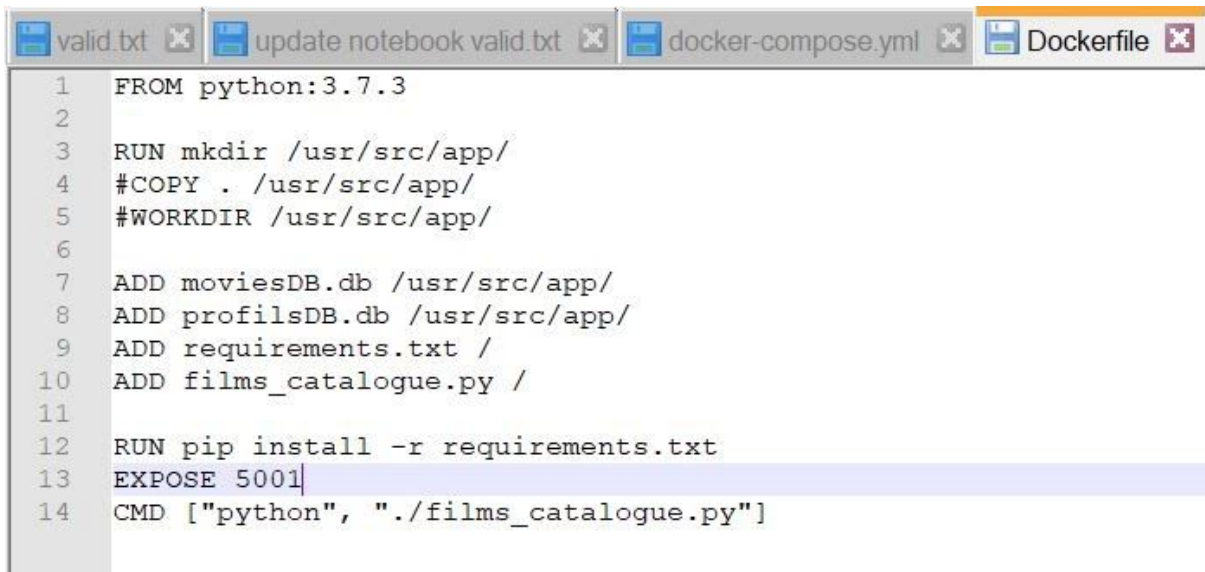
En déployant l'ensemble de ces web apis avec docker, nous arrivons bien à créer les conteneurs pour chaque micro-services avec la commande `docker-compose up` jouant les instructions que nous avons écrits dans le `docker-compose.yml` et les Dockerfiles :

```
Creating project_films_1 ... done
Creating project_search_1 ... done
Creating project_recommandation_1 ... done
Attaching to project_credits_1, project_profils_1, project_catalog_1, project_films_1, project_recommandation_1, project_search_1
catalog_1 | * Serving Flask app "films_catalogue" (lazy loading)
catalog_1 | * Environment: production
catalog_1 | WARNING: This is a development server. Do not use it in a production deployment.
catalog_1 | Use a production WSGI server instead.
catalog_1 | * Debug mode: off
catalog_1 | * Running on http://127.0.0.1:5001/ (Press CTRL+C to quit)
credits_1 | * Serving Flask app "credits" (lazy loading)
credits_1 | * Environment: production
credits_1 | WARNING: This is a development server. Do not use it in a production deployment.
credits_1 | Use a production WSGI server instead.
credits_1 | * Debug mode: off
credits_1 | * Running on http://127.0.0.1:5006/ (Press CTRL+C to quit)
profils_1 | * Serving Flask app "manage_profils" (lazy loading)
profils_1 | * Environment: production
profils_1 | WARNING: This is a development server. Do not use it in a production deployment.
profils_1 | Use a production WSGI server instead.
profils_1 | * Debug mode: off
profils_1 | * Running on http://127.0.0.1:5005/ (Press CTRL+C to quit)
films_1 | * Serving Flask app "manage_films" (lazy loading)
films_1 | * Environment: production
films_1 | WARNING: This is a development server. Do not use it in a production deployment.
films_1 | Use a production WSGI server instead.
films_1 | * Debug mode: off
films_1 | * Running on http://127.0.0.1:5004/ (Press CTRL+C to quit)
recommandation_1 | * Serving Flask app "recommandation" (lazy loading)
recommandation_1 | * Environment: production
recommandation_1 | WARNING: This is a development server. Do not use it in a production deployment.
recommandation_1 | Use a production WSGI server instead.
recommandation_1 | * Debug mode: off
recommandation_1 | * Running on http://127.0.0.1:5002/ (Press CTRL+C to quit)
search_1 | * Serving Flask app "search" (lazy loading)
search_1 | * Environment: production
search_1 | WARNING: This is a development server. Do not use it in a production deployment.
search_1 | Use a production WSGI server instead.
search_1 | * Debug mode: off
search_1 | * Running on http://127.0.0.1:5003/ (Press CTRL+C to quit)
```

Mais nous avons rencontrés 4 difficultés :

La première est que contrairement aux fois où nous jouons les scripts python sans docker, les urls qui étaient fournies après le déploiement de nos conteneurs étaient invalides alors qu'on a bien spécifié les ports et les hosts sur lesquels nos apis devaient être jouées.

Pour les Dockerfiles, pour le micro-service Catalog, on a mis les commandes suivantes :



```
1 FROM python:3.7.3
2
3 RUN mkdir /usr/src/app/
4 #COPY . /usr/src/app/
5 #WORKDIR /usr/src/app/
6
7 ADD moviesDB.db /usr/src/app/
8 ADD profilsDB.db /usr/src/app/
9 ADD requirements.txt /
10 ADD films_catalogue.py /
11
12 RUN pip install -r requirements.txt
13 EXPOSE 5001
14 CMD ["python", "./films_catalogue.py"]
```

Il y a l'import de python en tant qu'environnement pour le conteneur, la création du répertoire /usr/src/app/ qui va nous servir de volume pour stocker les bases de données afin qu'on ne perdent pas les données qu'elles contiennent à l'arrêt des conteneurs et qu'on puisse directement travailler dans le volume, l'ajout des bases de données dans le répertoire qui va servir de volume, l'installation des dépendances requises pour notre api (flask), l'exposition du port d'écoute qui va être utilisé pour cette api et la dernière commande permettant de jouer le script python.

Pour les autres apis, il y a aussi la première et les 5 dernières lignes qui sont écrites dans le Dockerfile de Catalog, avec les noms et le port respectifs pour chacune.

On pense que les instructions dans les Dockerfiles ne sont pas totalement bonnes pour réaliser notre projet mais nous ne savons pas exactement comment corriger ces fichiers pour obtenir le bon fonctionnement des urls.

Cette erreur pourrait aussi venir du docker-compose.yml mais nous pensons plus que nos problèmes viennent de la rédaction des Dockerfiles.

La deuxième difficulté venait du fait que nous ne voyons pas comment connecter les conteneurs pour les apis avec un conteneur contenant une base de données postgresQL, en « dockerisant » l'ensemble. D'où la présence de base de données dans le répertoire Catalog. Ces bases de données devaient être utilisées seulement dans un premier temps afin de voir le fonctionnement des apis avec docker. On voulait dans un second temps les lier avec les bases de données postgresQL utilisées avec le front Web, afin que l'ensemble des fonctionnalités (Front Web, BDD, APIs) soient réunis. C'est ce qu'on aurait essayé de faire dans le cas où nous n'aurions pas eu la 1^{ère} erreur.

La 3^{ème} difficulté

« Dockerization » du front Web :

Parmi les difficultés rencontrées, nous avons eu quelques erreurs lors de la “dockerization” de notre Front-End. Lorsque nous lançons la commande “**docker-compose up**”, le conteneur de la partie front “**angular app**” est bien généré par docker (avec quelques warnings), en revanche, nous n’arrivons pas à accéder à notre conteneur :

```
✓ Browser application bundle generation complete.
- Copying assets...
✓ Copying assets complete.
- Generating index.html...
✓ Index.html generation complete.

Initial Chunk Files | Names | Size
vendor.js           | vendor | 2.70 MB
polyfills.js        | polyfills | 141.27 kB
styles.css          | styles | 74.04 kB
main.js             | main | 66.61 kB
runtime.js          | runtime | 6.15 kB

| Initial Total | 2.98 MB

Build at: 2021-03-04T16:24:44.289Z - Hash: 632947b40ec52f27262f - Time: 9777ms

Warning: ./src/styles.css (./node_modules/css-loader/dist/cjs.js??ref--12-1!./node_modules/postcss-loader/dist/cjs.js??ref--12-2!./src/styles.css)
Module Warning (from ./node_modules/postcss-loader/dist/cjs.js):
Warning
(3:1) postcss-import: It looks like you didn't end your @import statement correctly. Child nodes are attached to it.

Removing intermediate container 8906e98fa8f9
----> ff851ac0517a
Step 10/10 : EXPOSE 80
----> Running in d8fe5fff76d4
Removing intermediate container d8fe5fff76d4
----> 9db9c6a81e71
Successfully built 9db9c6a81e71
Successfully tagged angular-app:latest
WARNING: Image for service angular-app was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating architecturemicro-services_angular-app_1 ... done
Attaching to architecturemicro-services_angular-app_1
architecturemicro-services_angular-app_1 exited with code 0
goumbark@goumbark-VirtualBox:~/Desktop/docker/Architecture@Micro-services_docker/Architecture@Micro-services$ ls
client docker-compose.yml package-lock.json Server
```

Dernière difficulté :

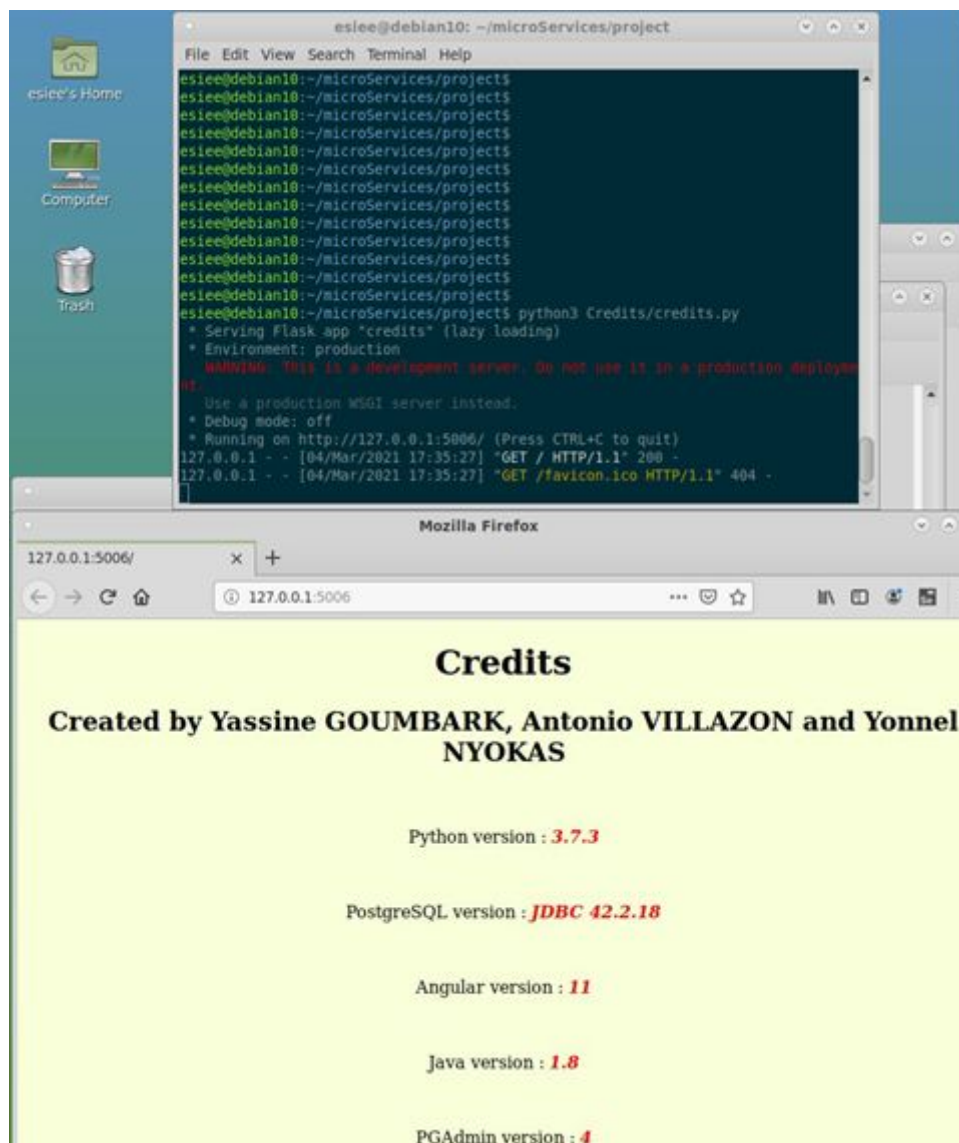
Lors du développement de notre site, nous avons eu plusieurs bugs au niveau de SpringBoot et cela nous a beaucoup ralenti dans l’avancement de notre projet.

Conteneurisation des services : recherche de solutions

Nous avons des problèmes lors du déploiement des conteneurs avec docker-compose.
Voici les résultats de notre recherche de résolution.

Vérification du bon fonctionnement de nos services

Exemple d'une api Flask renvoyant du code HTML correspondant à la présentation du groupe et outils :



Conteneurisation et l'accès aux services conteneurisés

La conteneurisation via docker fonctionne, et nous pouvons par exemple accéder à nos APIs... à condition que lors du lancement du conteneur à partir d'une image, on spécifie que l'on permet au conteneur d'avoir accès au réseau local de notre machine ou notre VM avec les argument `--network host`

The screenshot displays a terminal window and a web browser. In the terminal, the command `docker run --network host -d credit_img1` is executed, with `--network host` highlighted by a red box. The browser window shows the output of a curl command, displaying the 'Credits' page with version information for Python, PostgreSQL, Angular, Java, and PGAdmin.

```
eslee@debian10: ~/microServices/project/Credits$ docker run --network host -d credit_img1
eslee@debian10: ~/microServices/project$ curl http://127.0.0.1:5006
<!DOCTYPE html>
<html lang="en">
<style>
body {
background-color: #F9FFD8;
text-align: center;
color : black
}
</style>
<body>
<h1>Credits</h1>
<h2>Created by Yassine GOUMBARK, Antonio VILLAZON and Yonnel NYOKAS</h2>
<br>
<p>Python version : <b style="color:red;"><i>3.7.3</i></b></p>
<br>
<p>PostgreSQL version : <b style="color:red;"><i>10BC 42.2.18</i></b></p>
<br>
<p>Angular version : <b style="color:red;"><i>11</i></b></p>
<br>
<p>Java version : <b style="color:red;"><i>1.8</i></b></p>
<br>
<p>PGAdmin version : <b style="color:red;"><i>4</i></b></p>
</body>
eslee@debian10: ~/microServices/project$ curl http://0.0.0.0:5006
<!DOCTYPE html>
<html lang="en">
<style>
body {
background-color: #F9FFD8;
text-align: center;
color : black
}
</style>
<body>
<h1>Credits</h1>
<h2>Created by Yassine GOUMBARK, Antonio VILLAZON and Yonnel NYOKAS</h2>
<br>
<p>Python version : <b style="color:red;"><i>3.7.3</i></b></p>
<br>
<p>PostgreSQL version : <b style="color:red;"><i>10BC 42.2.18</i></b></p>
<br>
<p>Angular version : <b style="color:red;"><i>11</i></b></p>
<br>
<p>Java version : <b style="color:red;"><i>1.8</i></b></p>
<br>
<p>PGAdmin version : <b style="color:red;"><i>4</i></b></p>
```

127.0.0.1:5006/

Credits

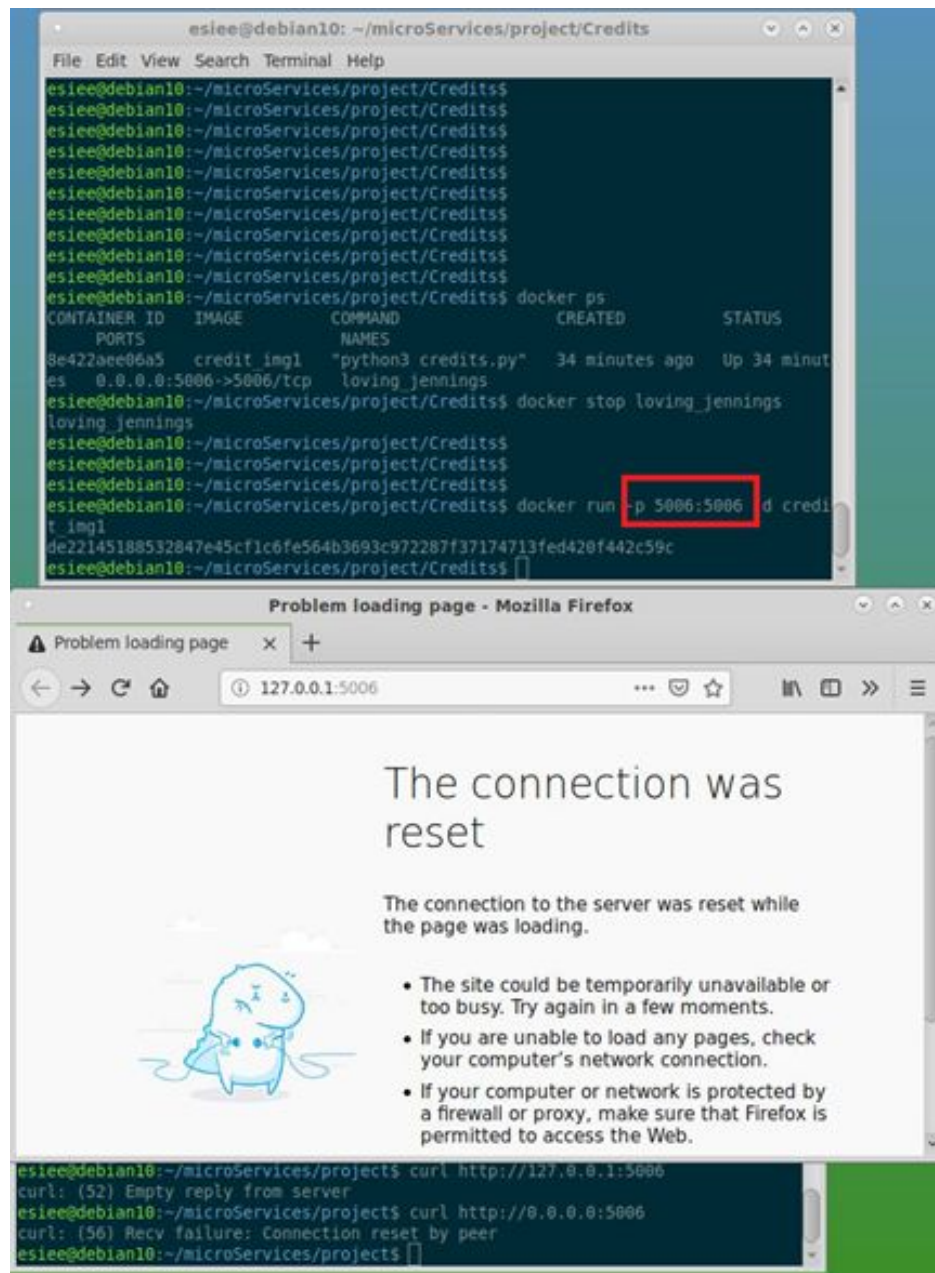
Created by Yassine GOUMBARK, Antonio VILLAZON and Yonnel NYOKAS

Python version : **3.7.3**

PostgreSQL version : **10BC 42.2.18**

Angular version : **11**

Cependant, lorsqu'on essaie de publier le port d'entrée du service, une erreur survient quand on tente d'y accéder :



The image shows two windows. The top window is a terminal with the following commands and output:

```
eslee@debian10: ~/microServices/project/Credits
File Edit View Search Terminal Help
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
PORTS         NAMES
8e422aee06a5   credit    "python3 credits.py"    34 minutes ago Up 34 minutes
0.0.0.0:5006->5006/tcp   loving_jennings
eslee@debian10:~/microServices/project/Credits$ docker stop loving_jennings
loving_jennings
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$
eslee@debian10:~/microServices/project/Credits$ docker run -p 5006:5006 d credit
t img1
de22145188532847e45cf1c6fe564b3693c972287f37174713fed428f442c59c
eslee@debian10:~/microServices/project/Credits$
```

The bottom window is a Mozilla Firefox browser showing a "Problem loading page" error. The address bar shows "127.0.0.1:5006". The main content area displays the message "The connection was reset" with a cartoon character and a list of troubleshooting steps:

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

At the bottom of the browser window, a terminal snippet shows the following commands and output:

```
eslee@debian10:~/microServices/projects$ curl http://127.0.0.1:5006
curl: (52) Empty reply from server
eslee@debian10:~/microServices/projects$ curl http://0.0.0.0:5006
curl: (56) Recv failure: Connection reset by peer
eslee@debian10:~/microServices/projects$
```

Conclusion

Donc lors de la construction de l'image par un dockerfile, nous n'arrivons pas à maîtriser les ports de communication. Cela nous empêche alors de réaliser des liens entre conteneurs.

On ne peut donc pas faire fonctionner le docker-compose que nous avons préparé puisqu'il utilise les dockerfiles pour construire les images et déployer les services.