Université Paris Cité

Faculté des Sciences

UFR de Mathématiques et Informatique

Rapport Projet programmation distribuée

Présenté Par :

-**OSTAPLIUK Nykyta**(M1 Info-IAD)

-**BENNABI Malek Mohamed** (M1 Info-VMI)

Année 2023/2024

Enoncé:

L'Objectif de ce travail est de Réaliser un projet web distribué utilisant des technologies et patterns spécifiés ci dessous:

- Web service (Java, Node. . .)
- Docker
- Multi conteneurs :fKubernetes, Base de données (SQL, NoSQL, hors en dans cloud)
- Déploiement dans un cloud, Front end (Angular, React, VueJS), Android. . . , frameworks

Dans la partie suivante nous allons voir les differentes etapes du projet realisé.

1 Aperçu de notre projet

Dans notre projet nous avons défini un service web basé sur une application Node.js qui gère les notes sur les tableaux des utilisateurs. Elle utilise Mongoose pour interagir avec une base de données MongoDB.

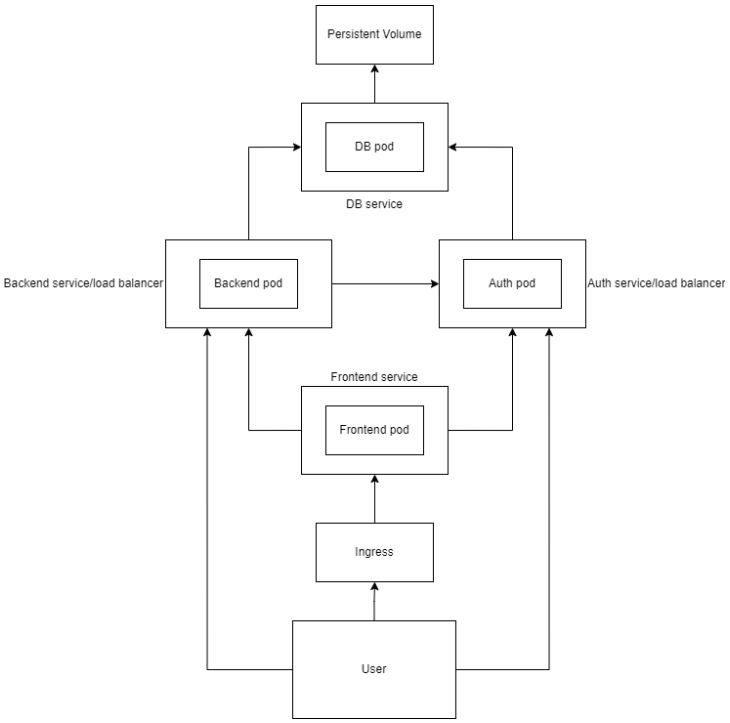

Figure 1: Architecture generale

2 Web Service

Comme cité precedement nous avons utilisé Node.js avec d'autres bibliotheques et framewworks on peut citer:

- body-parser : Un middleware populaire pour analyser les corps des requêtes entrantes dans les applications Express.js.
- cors : Active le partage des ressources entre origines (CORS) pour l'API backend.
- express : Un framework web populaire pour construire des applications web et des API en Node.js.
- mongoose : Une bibliothèque de modélisation des données d'objets (ODM) pour les bases de données MongoDB dans Node.js.

- mongoose-unique-validator : Plugin Mongoose qui valide les champs uniques dans la base de données.

On a aussi specifié un fichier de lockfile du projet pour eviter les problemes de dépendances du projet et leurs versions en cas d'execution sur d'autres appareils.

3 Docker

On a ensuite specifié un fichier docker pour construire l'image docker qui va permettre a notre application de marcher dans des environnements conteneurisés

```
PS C:\Users\Nykyta\IdeaProjects\distributed_project> docker compose up
[+] Running 4/0
 ✓ Container distributed_project-backend-1   Created
 ✓ Container distributed_project-frontend-1  Created
 ✓ Container distributed_project-mongo-1     Created
 ✓ Container distributed_project-auth-1      Created
Attaching to auth-1, backend-1, frontend-1, mongo-1
```

Figure 2: docker compose

4 Multi-Conteneurs

Notre application est multi-conteneurs avec une base de données Mongo, un service d'authentification, un service backend et un service frontend. Ils sont tous connectés via un réseau partagé et utilisent des secrets pour une configuration sécurisée.

Pour securiser le cluster on a utilise la commande
kubectl create secret generic notessecret fromfile=dbpassword=secrets/db_password from-file=jwtkey=secrets/jwt_key qui crée un Kubernetes Secret de type generique.

- mongo : Ce service exécute un conteneur de base de données Mongo en utilisant l'image officielle de Mongo. Il expose le port 27017 sur le conteneur, qui est également mappé au port 27017 sur la machine hôte. Il définit un volume qui monte le répertoire local ./db_data sur la machine hôte vers le répertoire /data/db à l'intérieur du conteneur. Cela permet un stockage persistant des données de la base de données. Il utilise des variables d'environnement pour configurer la base de données Mongo avec un utilisateur root et un mot de passe. Le mot de passe provient du secret db_password à l'aide d'une référence de chemin de fichier (/run/secrets/db_password). Le secret lui-même est monté dans le conteneur à l'aide de la propriété secrets. Enfin, il rejoint le réseau apptier.

- auth : Ce service exécute un conteneur avec l'image epicpigeon/auth_service:1, contenant le service d'authentification. Il expose le port 8080 sur le conteneur, mappé au port 8080 sur la machine hôte. Il lit les variables d'environnement pour le mot de passe de la base de données (MONGO_PASSWORD_FILE) et la clé JWT (JWT_KEY_FILE) à partir des fichiers secrets montés à partir des secrets db_password et jwt_key, respectivement. Il rejoint le réseau app-tier.

- backend : Ce service exécute un conteneur avec l'image epicpigeon/backend:1, contenant l'application backend. Il expose le port 8081 sur le conteneur, mappé au port 8081 sur la machine hôte. Il lit le mot de passe de la base de données dans un fichier secret monté à partir du secret db_password. Il définit une variable d'environnement AUTH_URL qui pointe vers le service d'authentification à l'adresse http://auth:8080/info. Cela permet au backend de communiquer avec le service d'authentification. Il rejoint le réseau app-tier.

- frontend : Ce service exécute un conteneur avec l'image epicpigeon/frontend:1, contenant l'application frontend. Il expose le port 80 sur le conteneur, mappé au port 80 sur la machine hôte. Cela signifie que le frontend sera accessible à l'adresse http://localhost sur votre machine par défaut. Il définit une variable d'environnement THIS_DOMAIN avec la valeur localhost, utilisée par l'application frontend elle-même.

4.1 Kubernetes

Pour chacun des 4 services precedemment cités on a utilisé Kubernetes pour les deployer, les fichiers **.yaml** de configuration et de deploiement ont ete regroupés dans un dossier nommé 'kubernetes'

On a utilisé Ingress dans Kubernetes pour gèrer l'accès externe aux services avec le HTTP.

```
PS C:\Users\Nykyta\IdeaProjects\distributed_project> kubectl apply -f kubernetes
deployment.apps/auth created
service/auth created
deployment.apps/backend created
service/backend created
deployment.apps/frontend created
ingress.networking.k8s.io/frontend-ingress created
service/frontend created
persistentvolumeclaim/mongo-claim created
deployment.apps/mongo created
service/mongo created
```

Figure 3: Deploiement Kubernetes

5 Frontend

Nous avons utilisé un frontend simple afin de remplir les champs de notre application web. Le code définit plusieurs chemins à l'aide de app.post, toutes les connexions requièrent une authentification en appelant d'abord authUser en tant qu'intergiciel.

- User boards :Récupère tous les tableaux de l'utilisateur authentifié en utilisant Board.find(author : req.user.login), Répond avec un objet JSON contenant les tableaux récupérés.
- createBoard : Crée un tableau pour l'utilisateur authentifié avec les propriétés: author : Défini comme le login de l'utilisateur à partir de req.user. notes : Un tableau vide au départ. Enregistre le forum dans la base de données en utilisant board.save(). Répond avec un objet JSON contenant l'ID du forum nouvellement créé.
- updateBoard :Récupère le tableau à l'aide de son identifiant et Vérifie si l'utilisateur est son auteur puis met à jour le propriétaire publique du tableau en fonction du propriétaire contenu dans le corps de la requête, Enregistre le tableau mis à jour dans la base de données.
- deleteBoard :Récupère le tableau à l'aide de getBoard, Vérifie si l'utilisateur est son auteur puis supprime le document du forum en utilisant board.deleteOne().
- addNote :Récupère le tableau à l'aide de getBoard et extrait le contenu de la note du corps de la requête ensuite crée un nouvel objet note avec le contenu fourni et le place dans le tableau des notes du forum et enfin enregistre le tableau mis à jour dans la base de données à l'aide de board.save().
- deleteNote : Récupère le tableau à l'aide de getBoard, extrait le noteId du corps de la requête et supprime le document de la note en utilisant note.deleteOne() ensuite sauvegarde le tableau mis à jour dans la base de données en utilisant board.save().

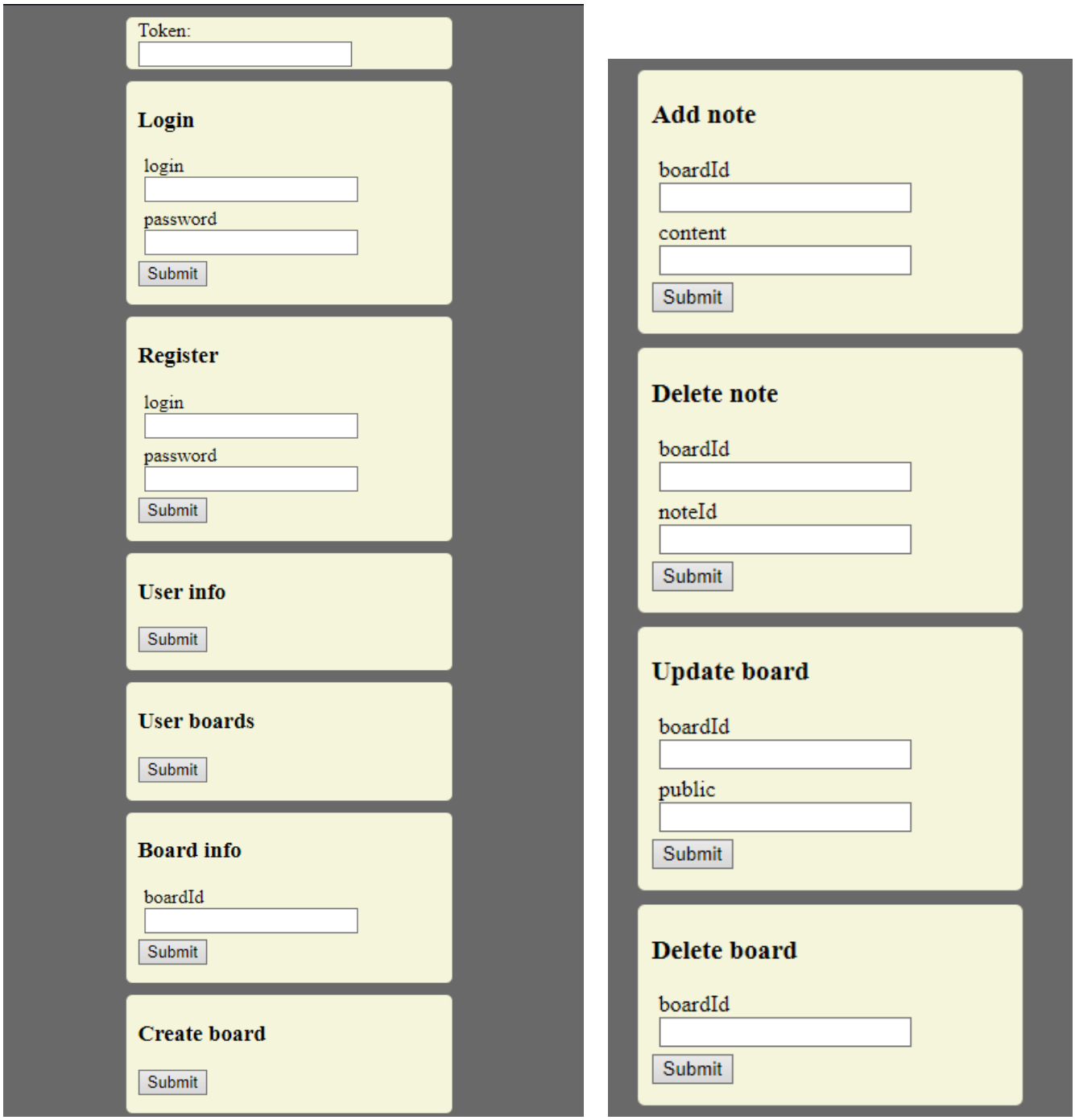

Figure 4: Service frontend

6 Google Labs

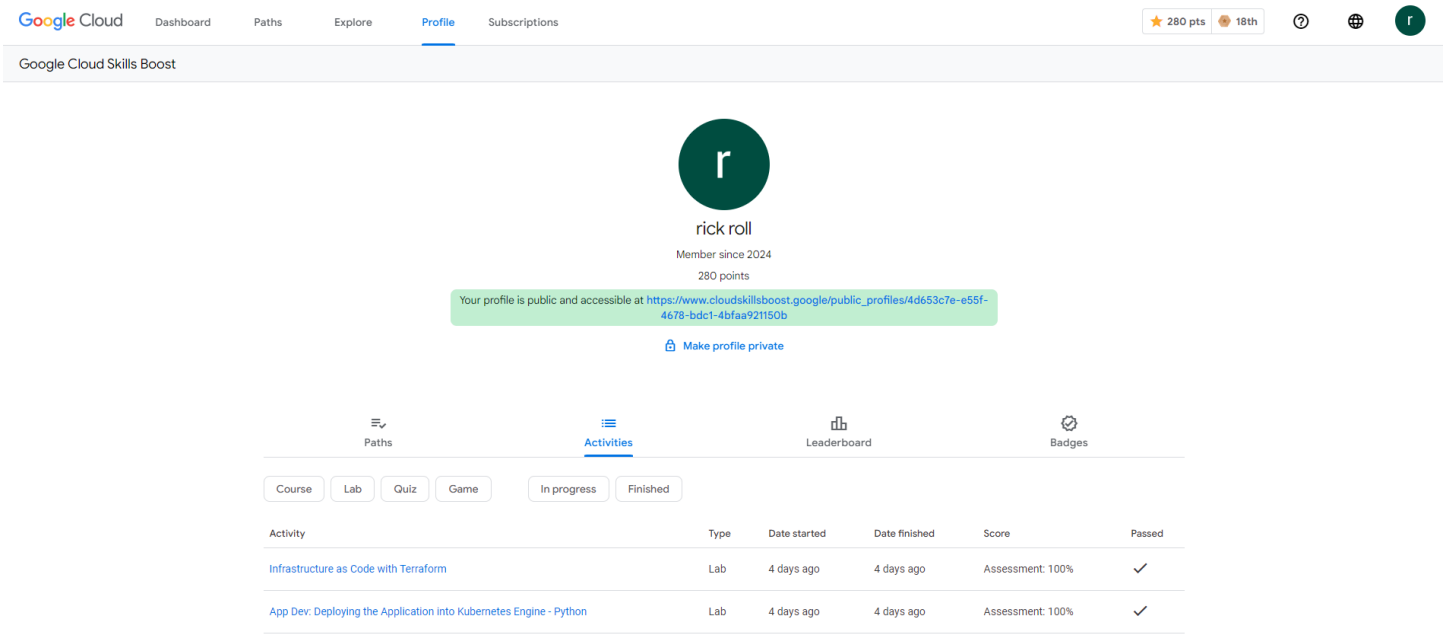

Figure 5: Activite Lab OSTAPLIUK

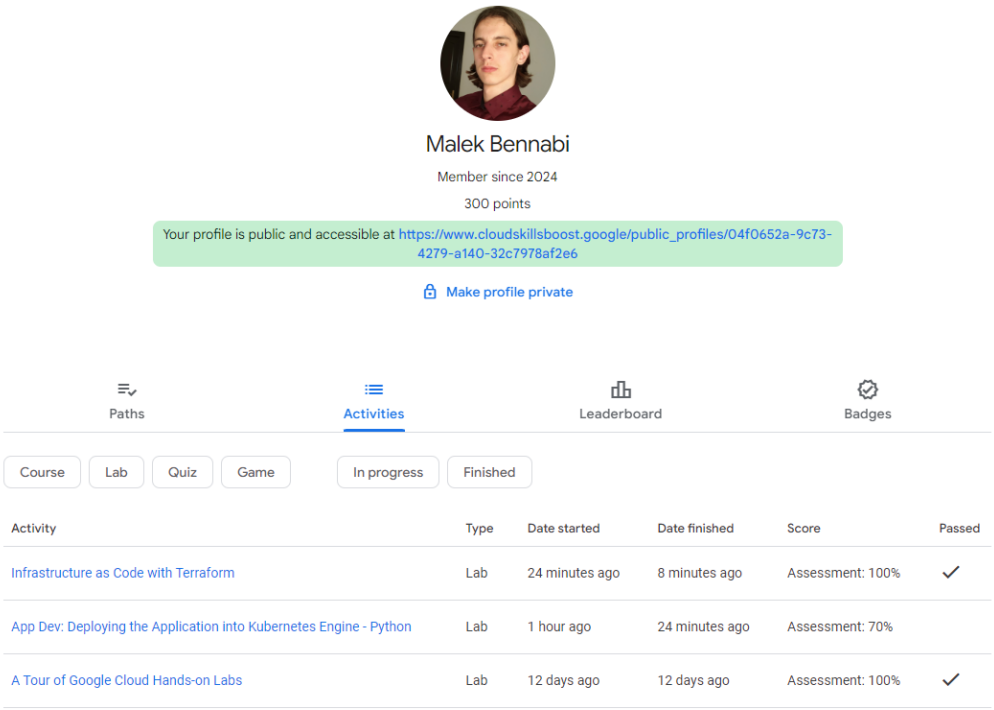

Figure 6: Activité Lab BENNABI

Le code source est disponible à l'adresse:
[Github](**https://github.com/Malekbennabi3/distributed_project.git**).
[Github](**https://github.com/epic-pigeon/distributed_project.git**).