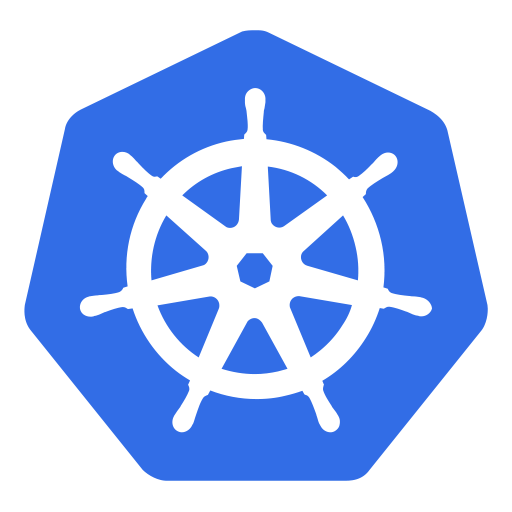
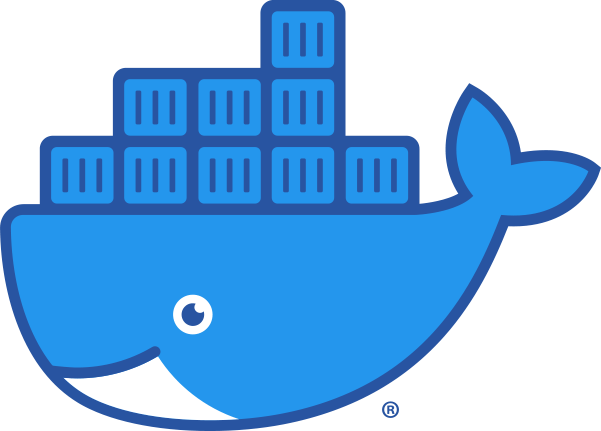


Conteneurisation des Applications :





Mini Projet

Prof. Driss ALLAKI

Présente par :

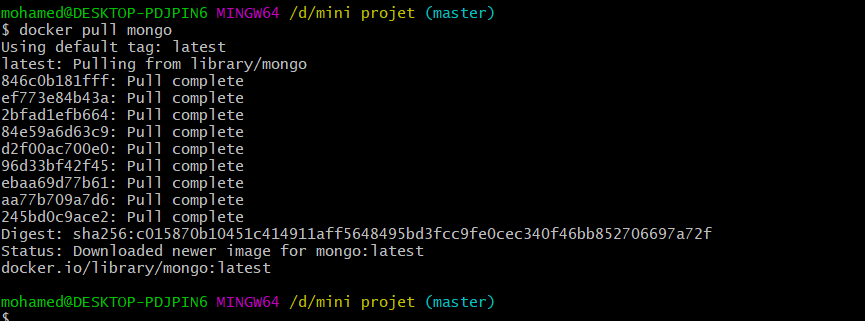
EL HAJJAMI MOHAMED

Filière : ASEDS Année universitaire : 2022/2023

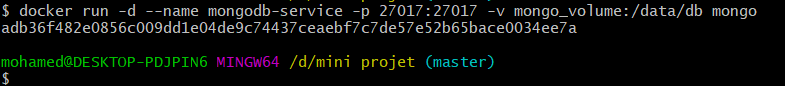
# Partie 1 :

Dans cette partie, il est demandé de réaliser les manipulations suivantes en utilisant un outil de conteneurisation comme Docker.

1. Créer un conteneur (nommé mongodb-service) pour la base de données MongoDB en instanciant l’image officielle mongo.

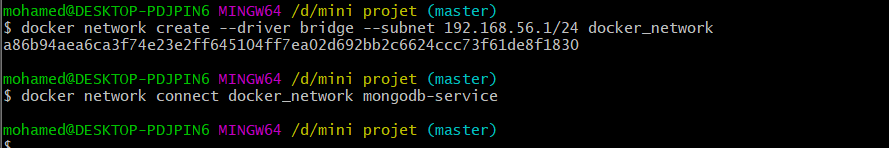
On va télécharger l'image MongoDB la plus récente disponible dans le dépôt Docker Hub et la stocker localement sur votre ordinateur

Puis on va créer et de démarrer le conteneur mongodb-service en arrière-plan et on monte également le volume externe "mongo\_volume" sur le répertoire "/data/db" du conteneur.

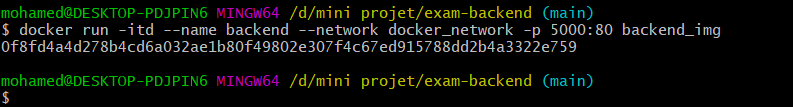


1. Créer un réseau Docker (tout en lui précisant un driver convenable). Puis connecter le conteneur mongodb avec ce réseau.

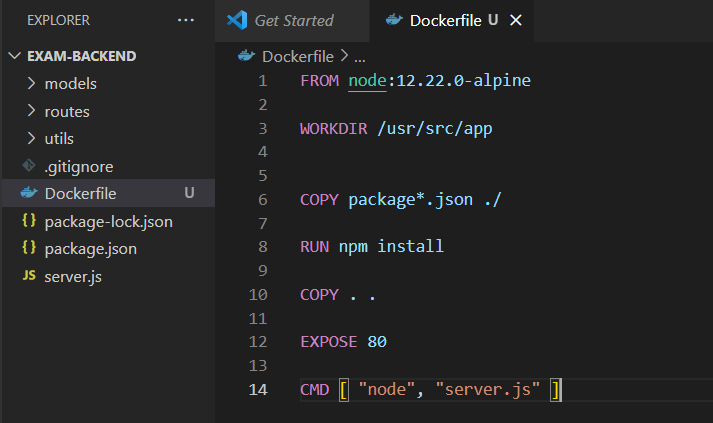
On crée un réseau Docker nommé "docker\_network" avec le pilote "bridge" et une adresse IP de départ de 192.168.56.1



Maintenant on connect mongodb-service a ce network



1. Créer un fichier Dockerfile pour le projet Backend.



1. Enumérer et expliquer "en détails" les bonnes pratiques utilisées pour l’écriture de ce fichier Dockerfile.

Voici une explication de chaque ligne de ce fichier :

• **FROM node:12.22.0-alpine :** on a utiliser l'image Node.js 12.22.0-alpine comme base pour notre image Docker.

• WORKDIR /usr/src/app : le répertoire de travail de notre conteneur. Toutes les commandes suivantes seront exécutées dans ce répertoire.

• **COPY package\*.json ./ :** cette ligne copie tous les fichiers qui correspondent à "package\*.json" ("package.json" et "package-lock.json") du répertoire courant (.) vers le répertoire de travail du conteneur (./).

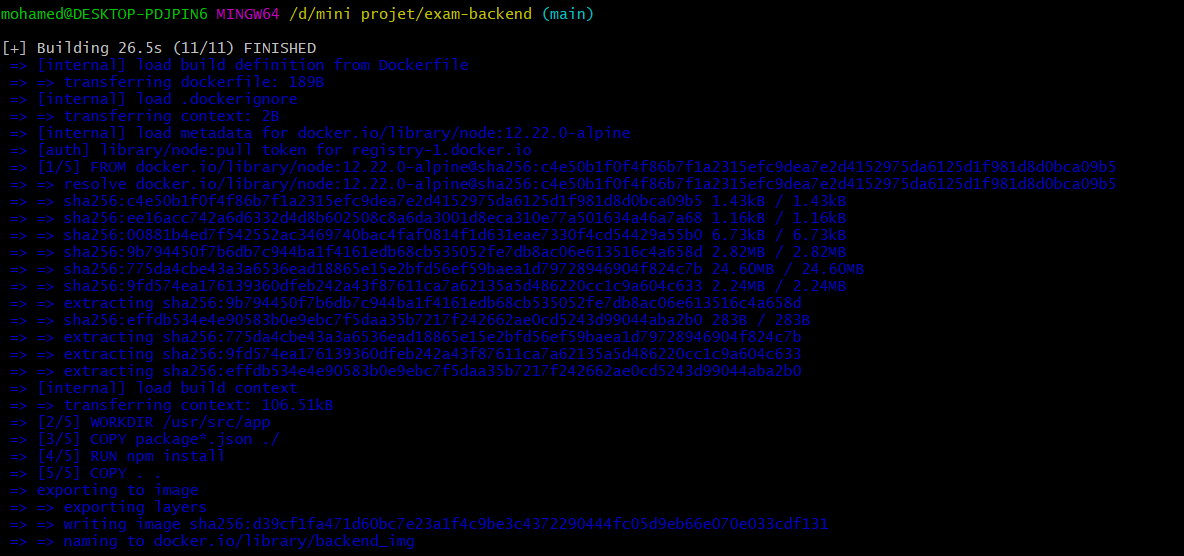
• **RUN npm install :** cette ligne exécute la commande "npm install" dans le conteneur, ce qui installe toutes les dépendances de votre application listées dans "package.json".

• **COPY . . :** cette ligne copie tous les fichiers du répertoire courant (.) vers le répertoire de travail du conteneur (.). Cela inclut notre application et tous les autres fichiers nécessaires pour exécuter votre application.

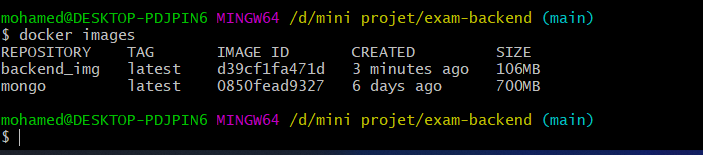
• **EXPOSE 80 :** cette ligne indique que notre application écoute sur le port 80 dans le conteneur.

1. Taper les commandes docker nécessaires pour :
   1. Créer une image docker en local à partir de ce Dockerfile.

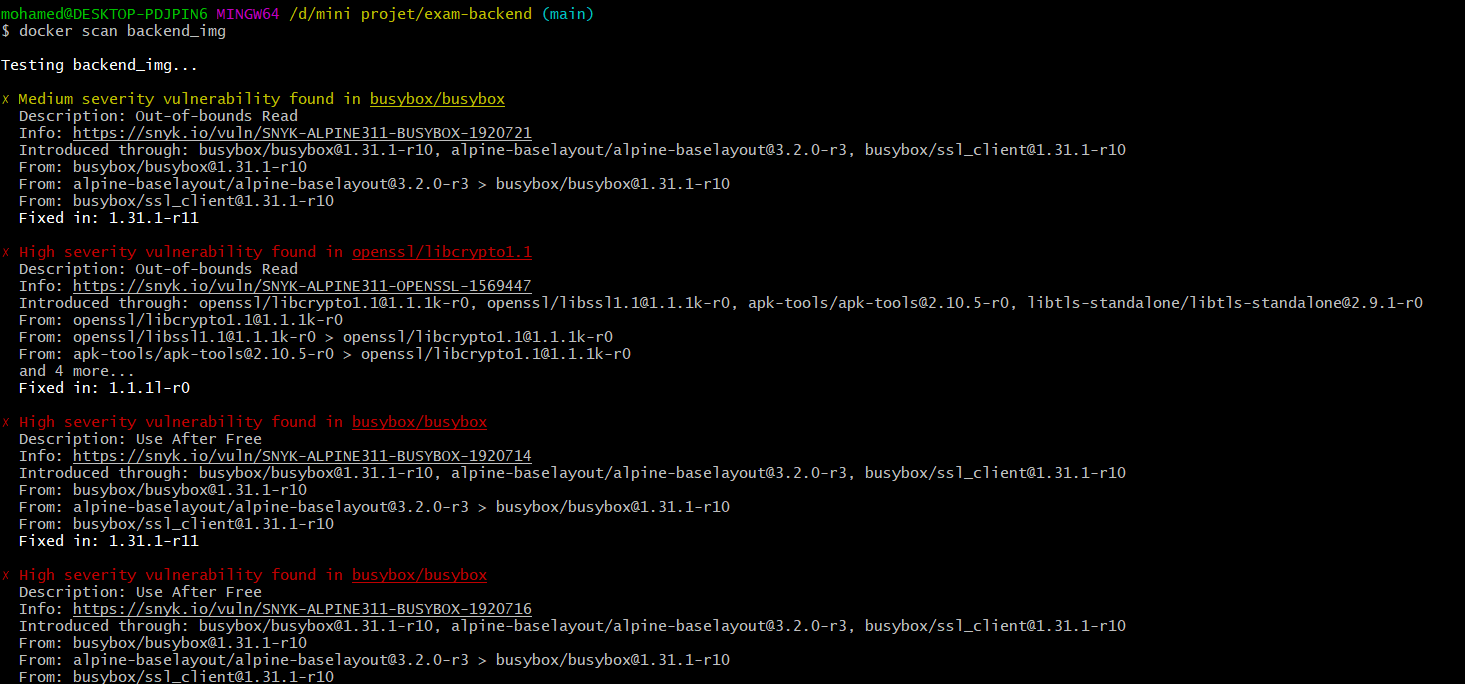
On lance la commande docker build . -t backend\_img

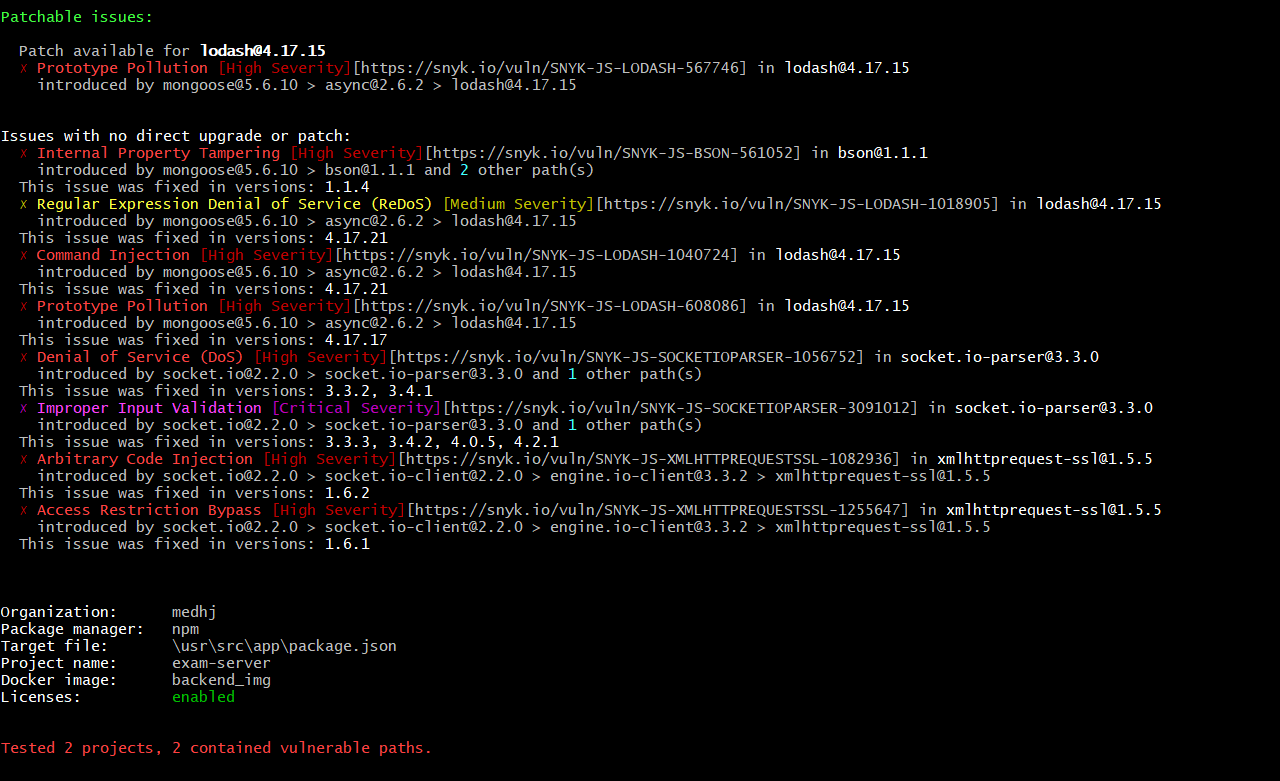


on verifier qu’on a builder l’image



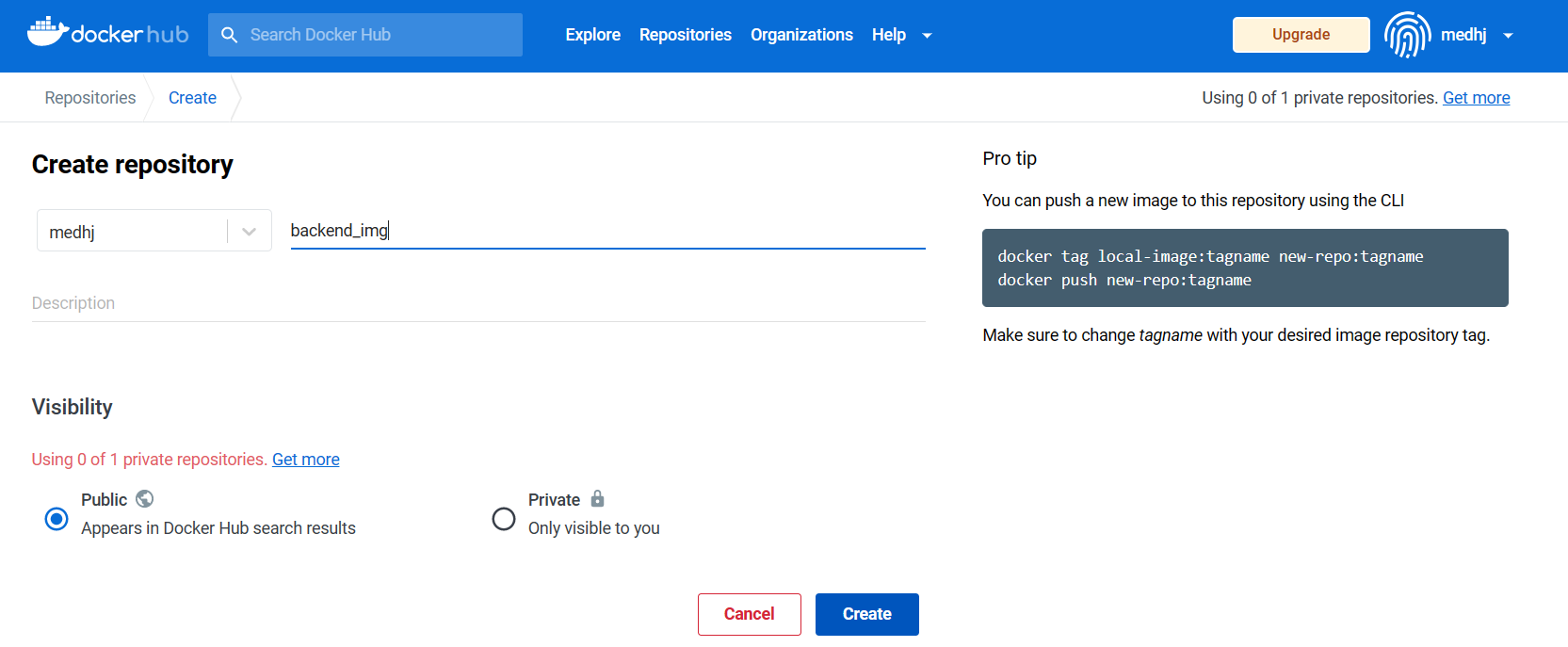
* 1. Scanner l’image des vulnérabilités qu’elle peut contenir.





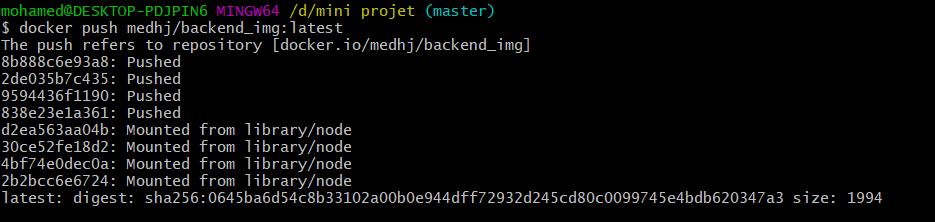
* 1. Publier cette image dans Docker Hub.

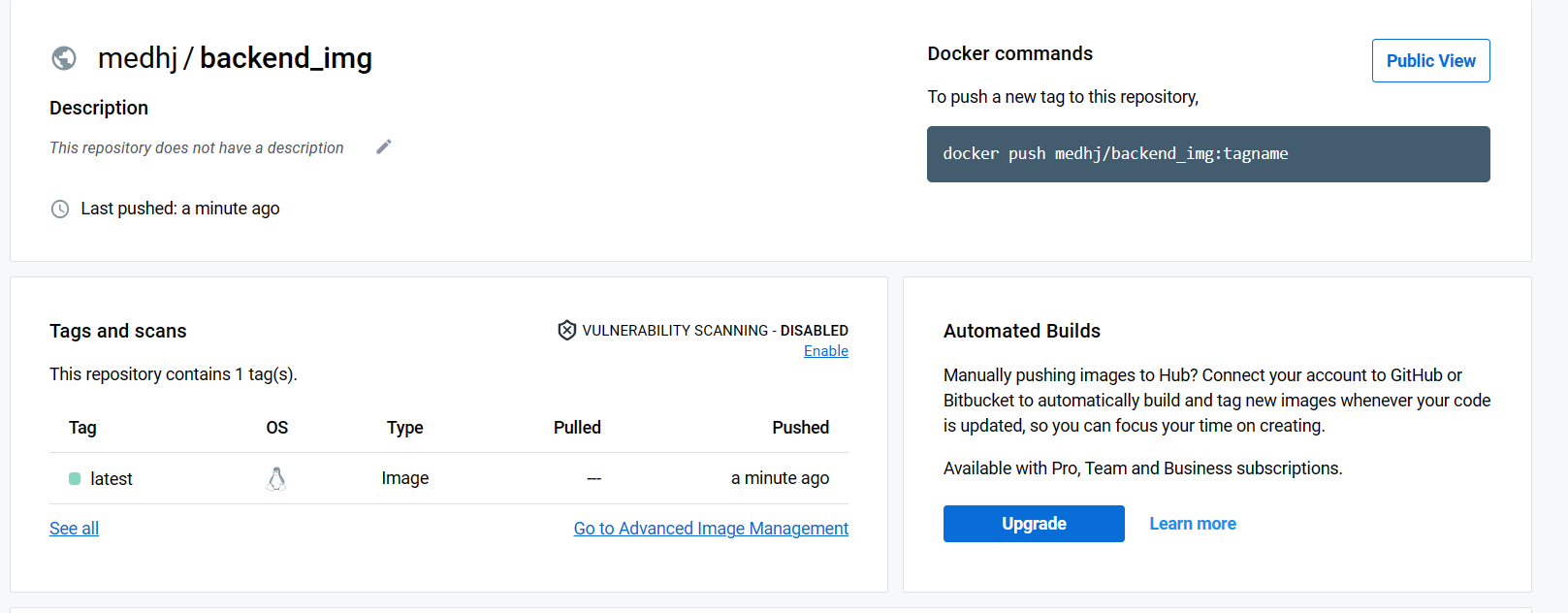
On ajoute un nouveau repositorie dans docker hub

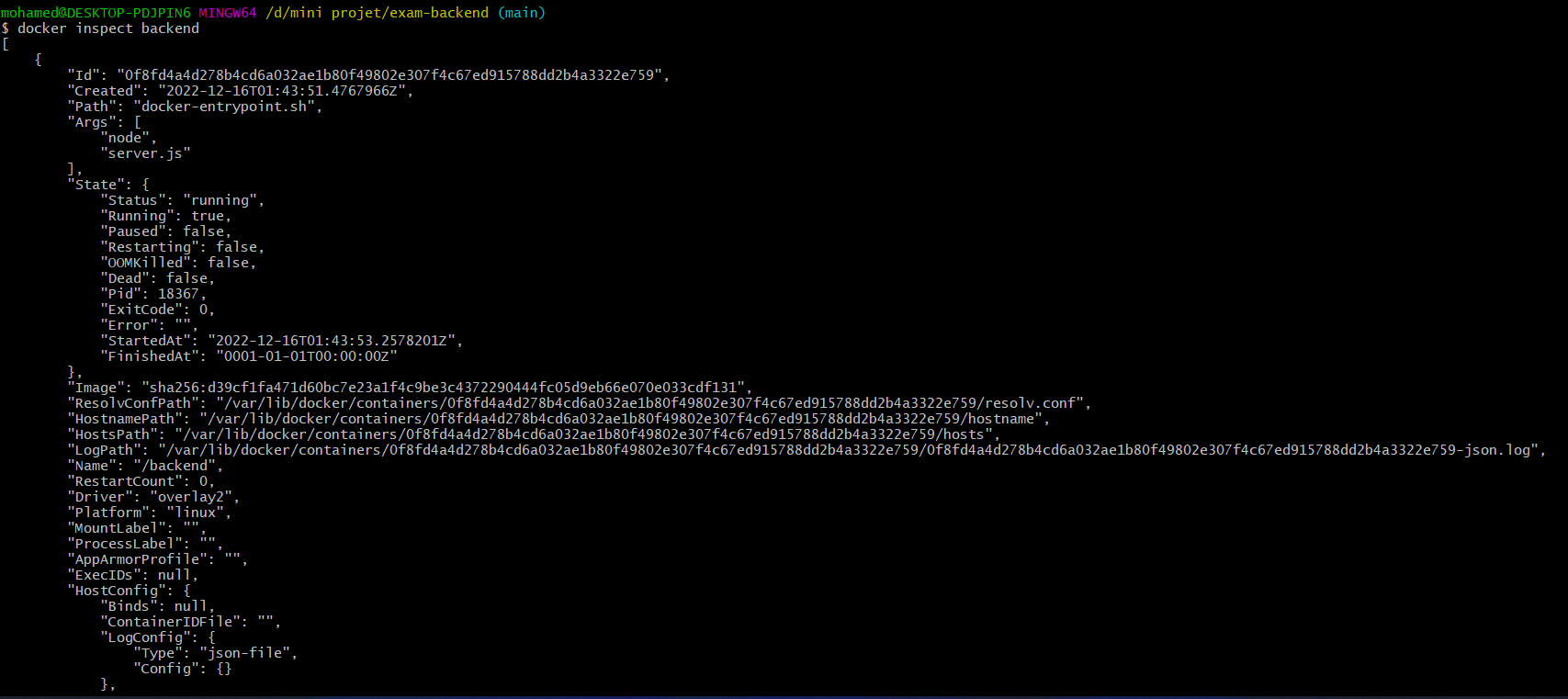


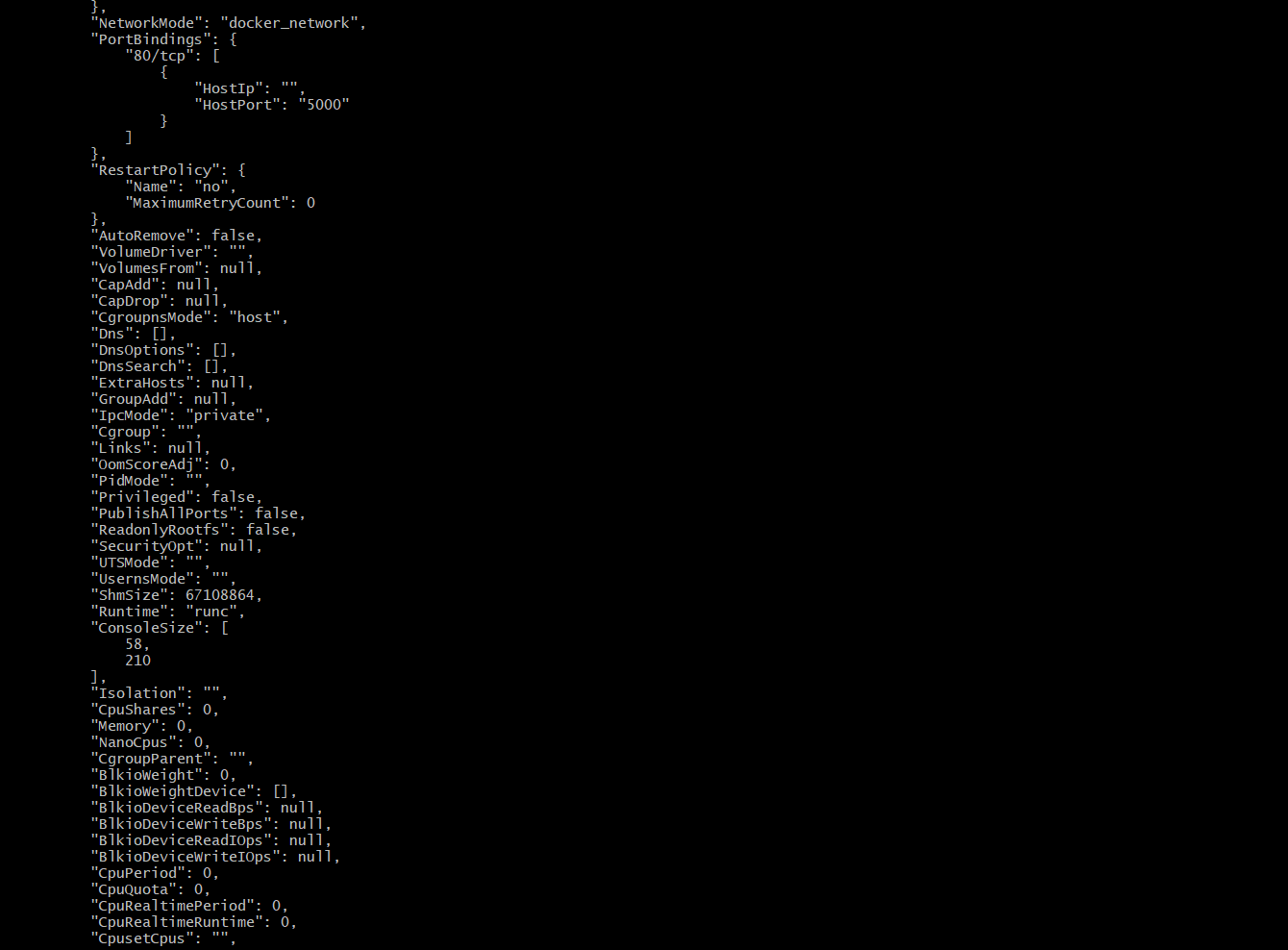
On tag notre image de backend





Maintenat on verifier qu’on pusher l’image

* 1. Instancier cette image en créant un conteneur (nommé backend) s’exécutant en local et partageant le même réseau avec le conteneur de la base de données mongodb.
  2. Inspecter ce conteneur backend.

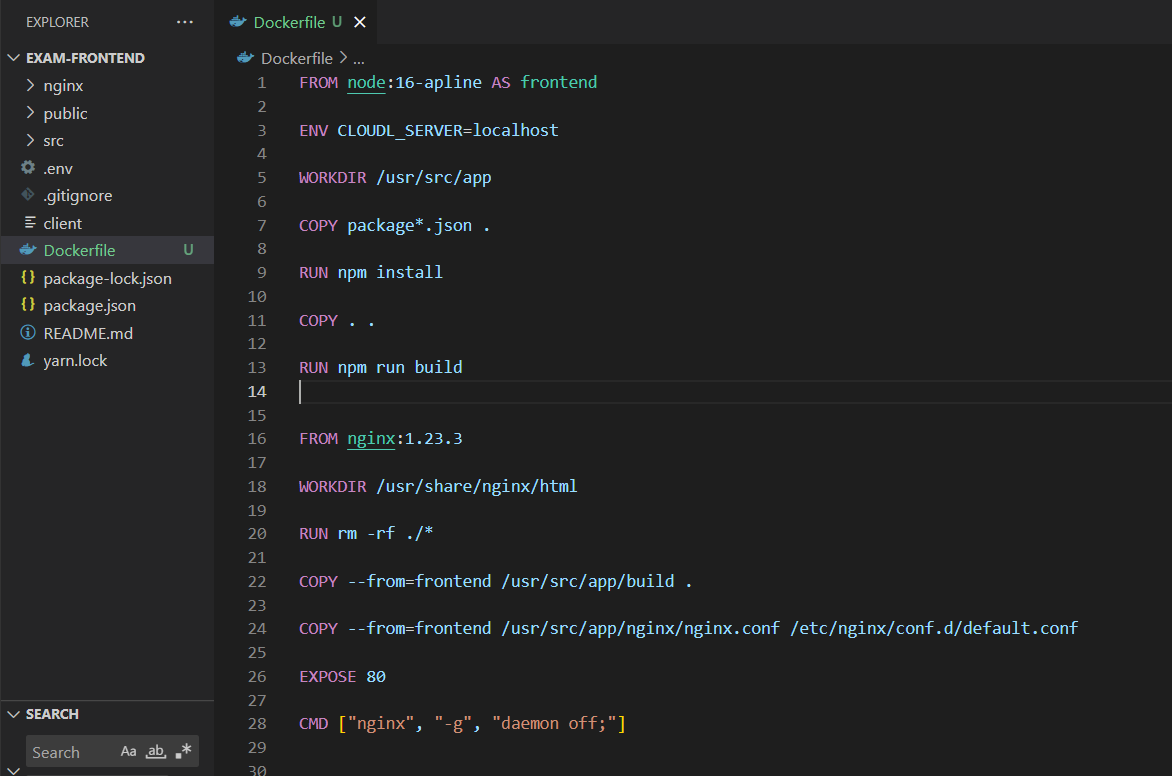


* 1. Afficher les logs liés à ce conteneur backend.



1. Refaire le travail demandé dans les questions 3), 4) et 5) pour le projet Frontend.

3- Créer un fichier Dockerfile pour le projet frontend.



4- Enumérer et expliquer "en détails" les bonnes pratiques utilisées pour l’écriture de ce fichier Dockerfile.

Voici une explication de chaque étape de ce fichier :

**FROM node:16.19-alpine AS frontend :** on utilise l'image Node.js 16.19-alpine comme base pour la première étape de notre image Docker. on donne également un nom à cette étape ("frontend")

**ENV CLOUDL\_SERVER=localhost :** cette ligne définit une variable d'environnement nommée "CLOUDL\_SERVER" avec la valeur "localhost" dans le conteneur de cette étape.

**WORKDIR /usr/src/app :** cette ligne définit le répertoire de travail de mon conteneur. Toutes les commandes suivantes seront exécutées dans ce répertoire.

**COPY package\*.json . :** cette ligne copie tous les fichiers qui correspondent à "package\*.json" ("package.json" et "package-lock.json") du répertoire courant (.) vers le répertoire de travail du conteneur (.).

**RUN npm install :** cette ligne exécute la commande "npm install" dans le conteneur, ce qui installe toutes les dépendances de l’application listées dans "package.json".

**COPY . . :** cette ligne copie tous les fichiers du répertoire courant (.) vers le répertoire de travail du conteneur (.). Cela inclut notre application et tous les autres fichiers nécessaires pour exécuter votre application.

**RUN npm run build :** cette ligne exécute la commande "npm run build" dans le conteneur, qui génère les fichiers statiques de votre application (généralement dans un répertoire "build").

**FROM nginx:1.23.3 :** on a utilisé l'image Nginx 1.23.3 comme base pour la seconde étape de notre image Docker.

**WORKDIR /usr/share/nginx/html :** cette ligne définit le répertoire de travail du conteneur. Toutes les commandes suivantes seront exécutées dans ce répertoire.

**RUN rm -rf ./\* :** cette ligne exécute la commande "rm -rf ./\*" dans le conteneur, qui supprime tous les fichiers du répertoire de travail.

**COPY --from=frontend /usr/src/app/build . :** cette ligne copie les fichiers générés par la première étape (stockés dans "/usr/src/app/build") dans le répertoire de travail du conteneur (.).

**COPY --from=frontend /usr/src/app/nginx/nginx.conf /etc/nginx/conf.d/default.conf :** permet de copier un fichier "nginx.conf" depuis la première étape de construction (identifiée par le nom "frontend") vers le répertoire "/etc/nginx/conf.d" de l'image de la seconde étape.

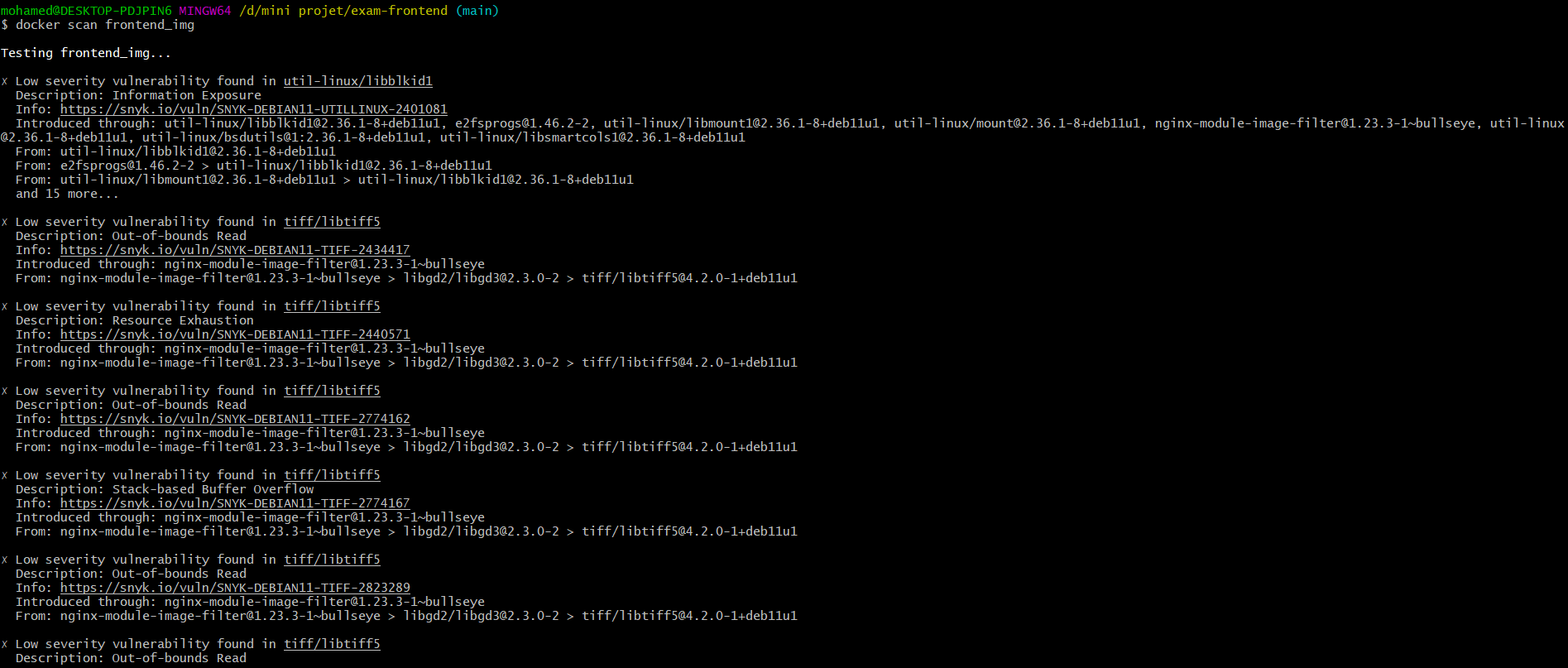
**EXPOSE 80 :** cette ligne indique que votre application écoute sur le port 80 dans le conteneur.

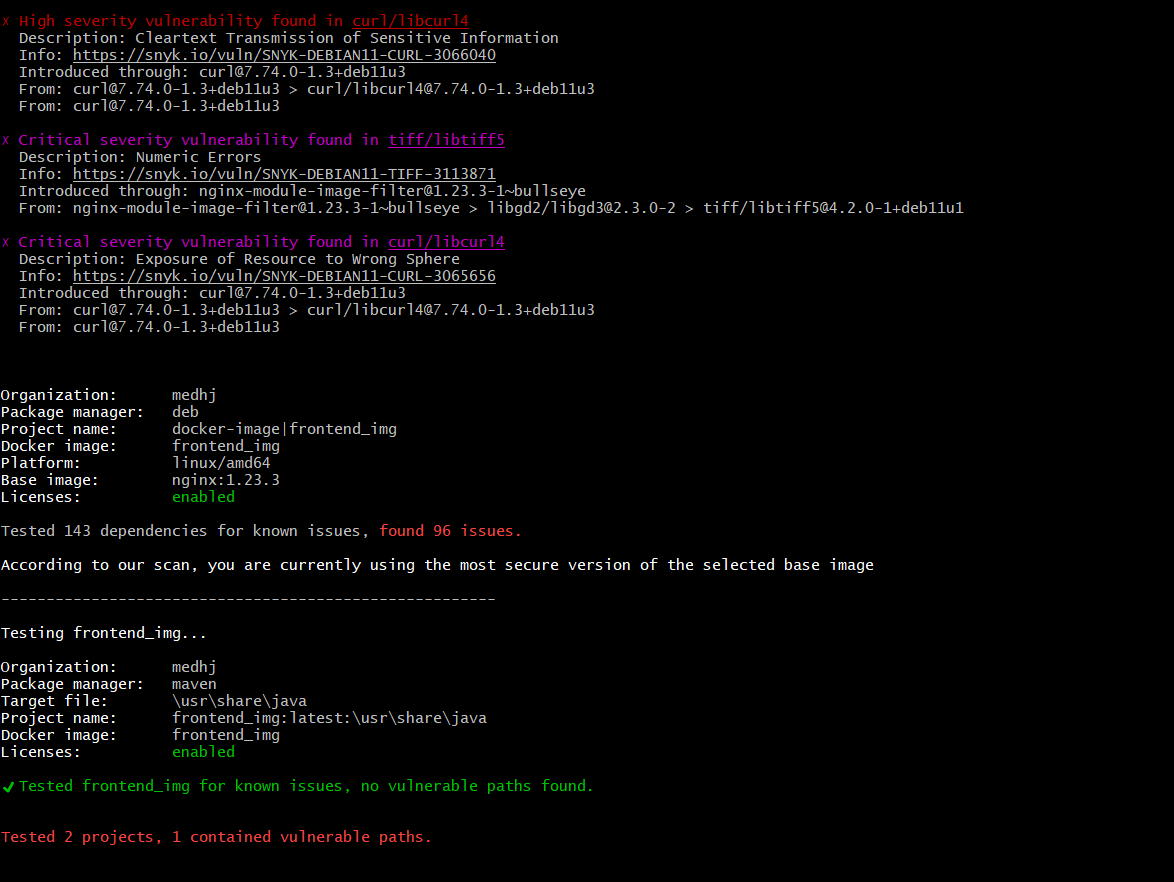
**CMD ["nginx", "-g", "daemon off;"] :** cette ligne définit la commande à exécuter lorsque le conteneur est démarré. Dans ce cas, elle lance le serveur Nginx avec les options "-g" et "daemon off;".

5. Taper les commandes docker nécessaires pour :

 a. Créer une image docker en local à partir de ce Dockerfile

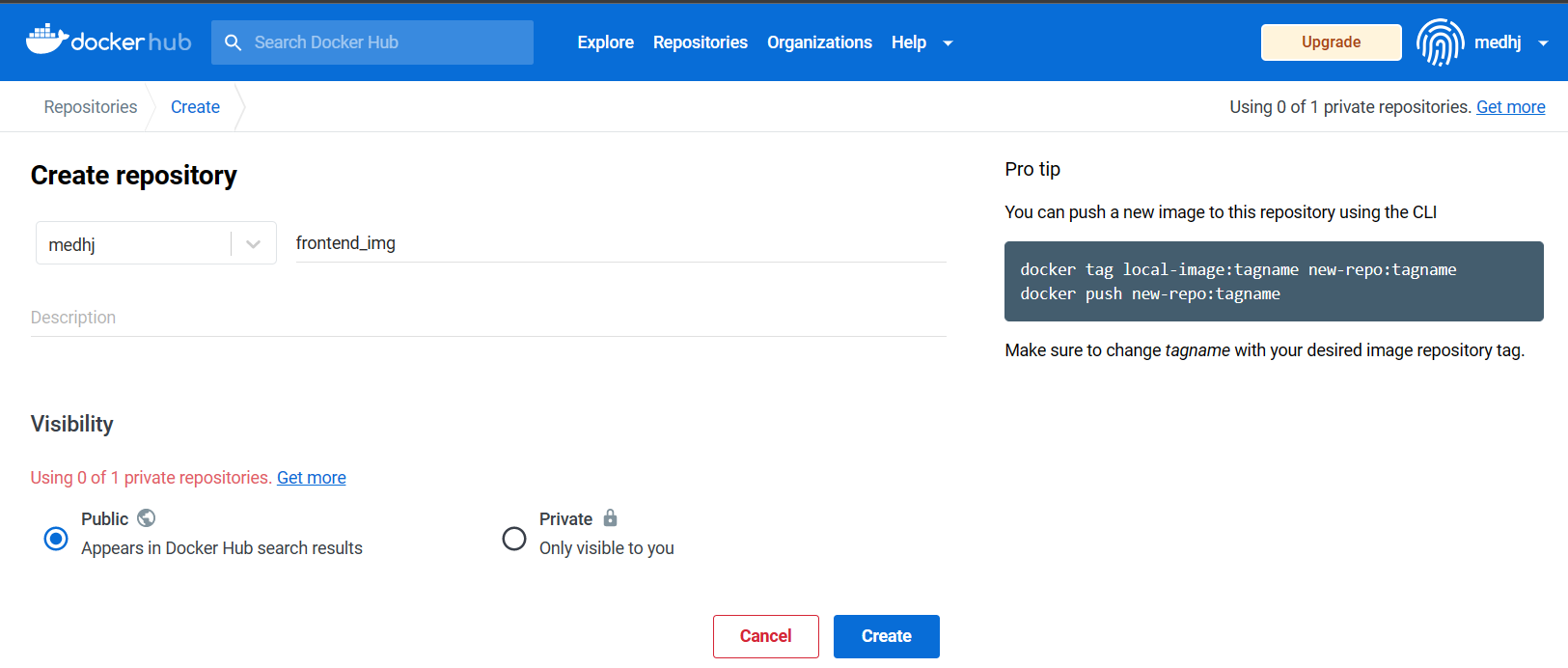
b. Scanner l’image des vulnérabilités qu’elle peut contenir.

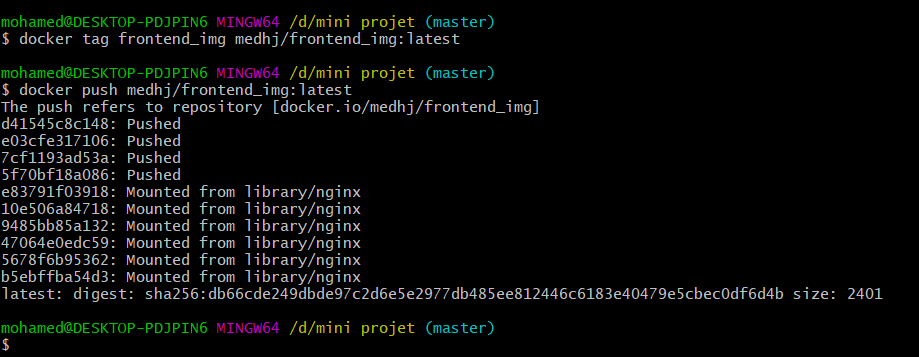


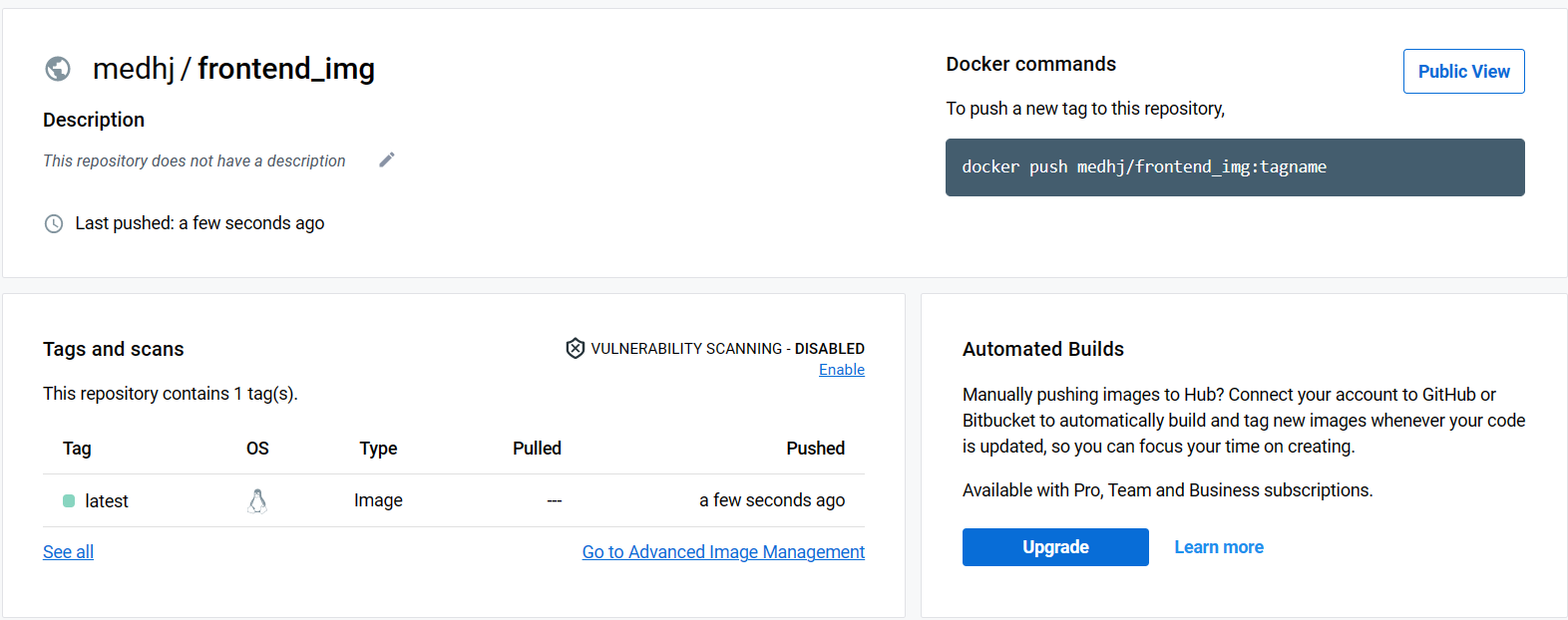


c. Publier cette image dans Docker Hub.

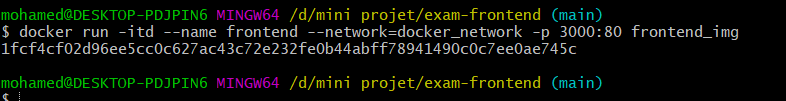
On ajoute un nouveau repositorie dans docker hub

On tag notre image de frontend



Maintenat on verifier qu’on pusher l’image 

d. Instancier cette image en créant un conteneur (nommé backend) s’exécutant en local et partageant le même réseau avec le conteneur de la base de données mongodb.

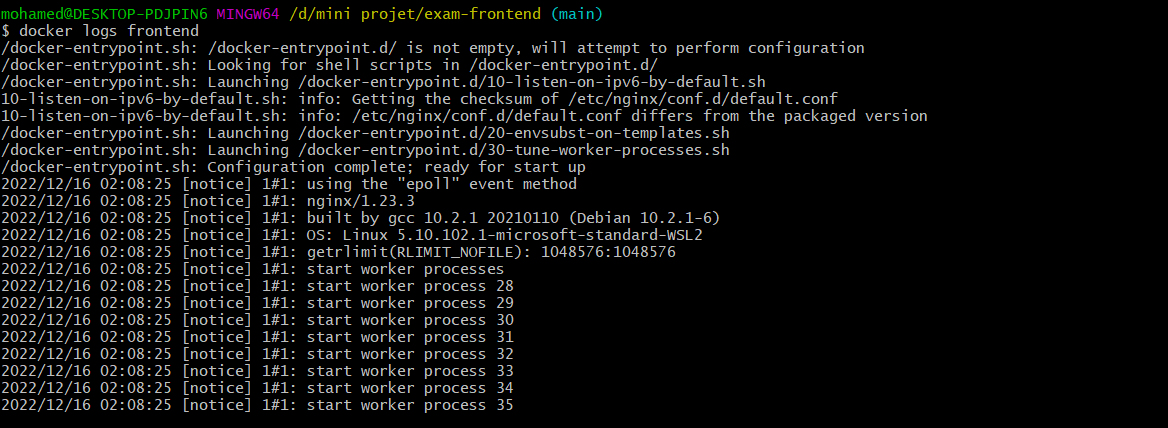


e. Inspecter ce conteneur backend.



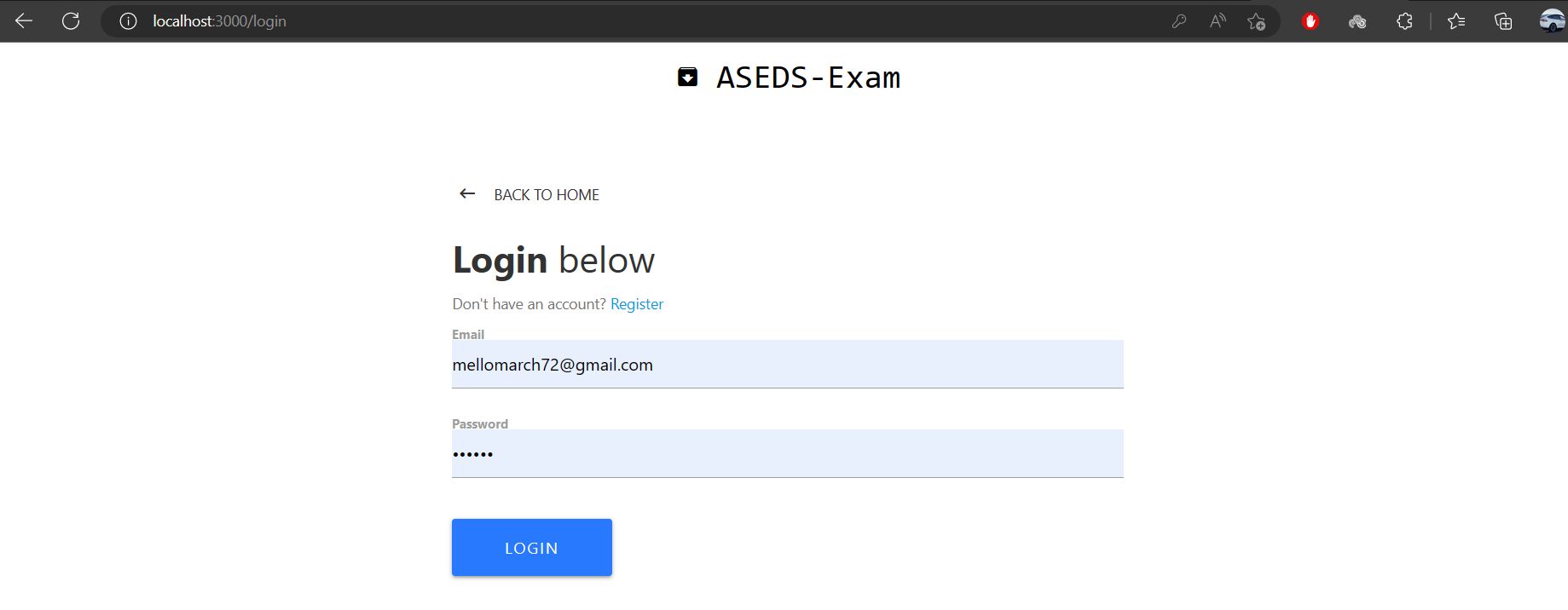
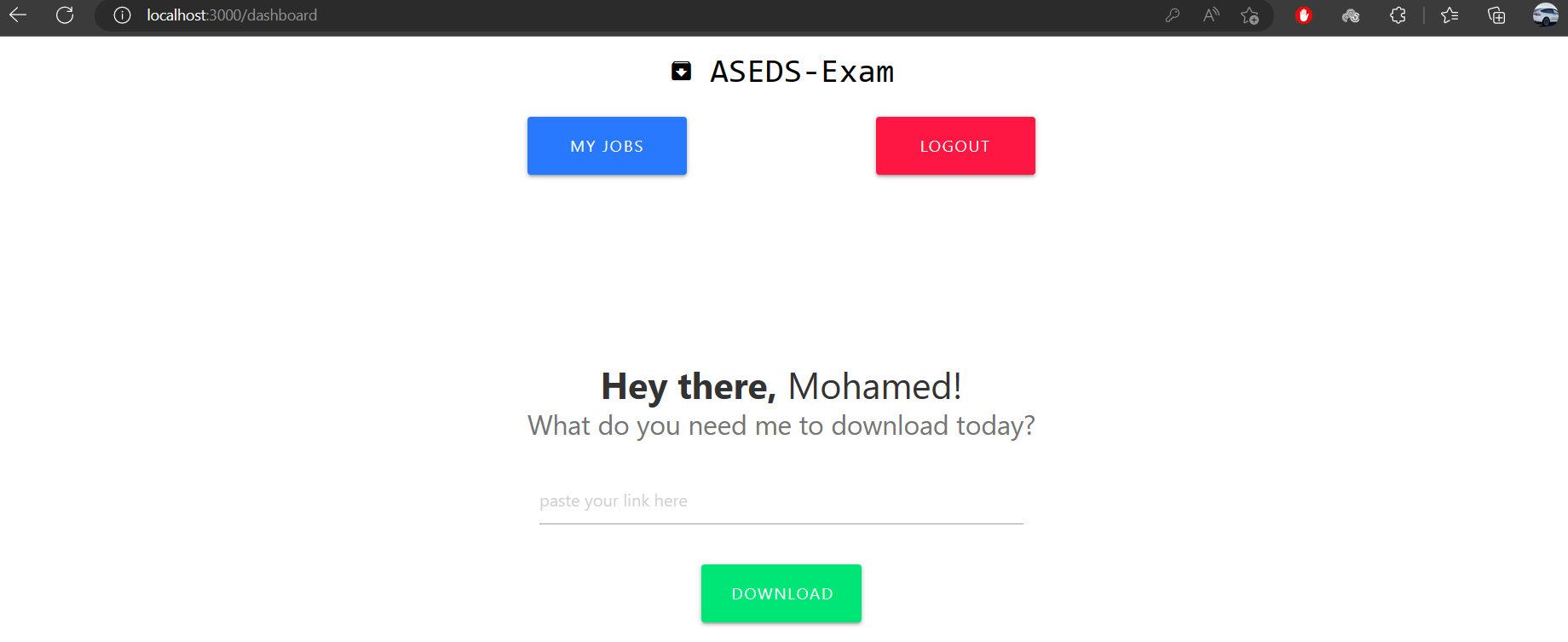


f. Afficher les logs liés à ce conteneur backend.



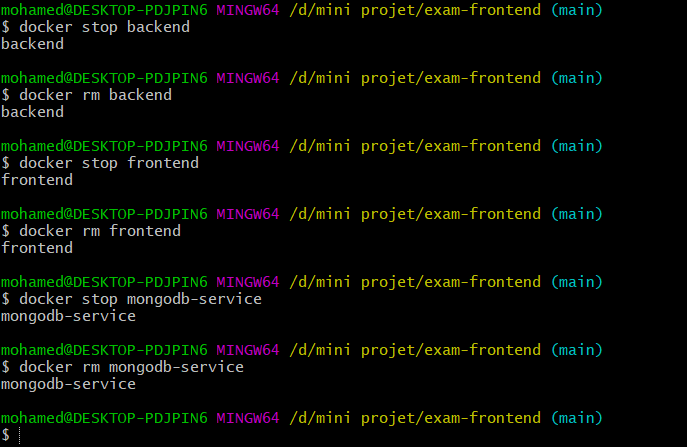
1. S’assurer que l’application a été bien conteneurisée et qu’elle fonctionne correctement. (Prendre des prises d’écran du navigateur)



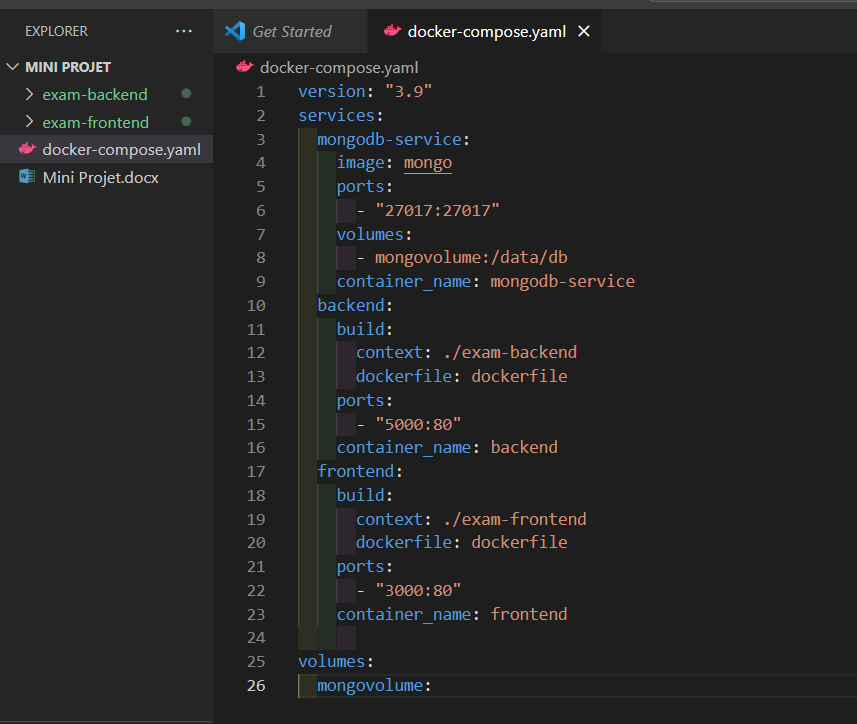
  

1. Supprimer les 3 conteneurs en exécution.

On arrête les conteneurs puis on les suprême

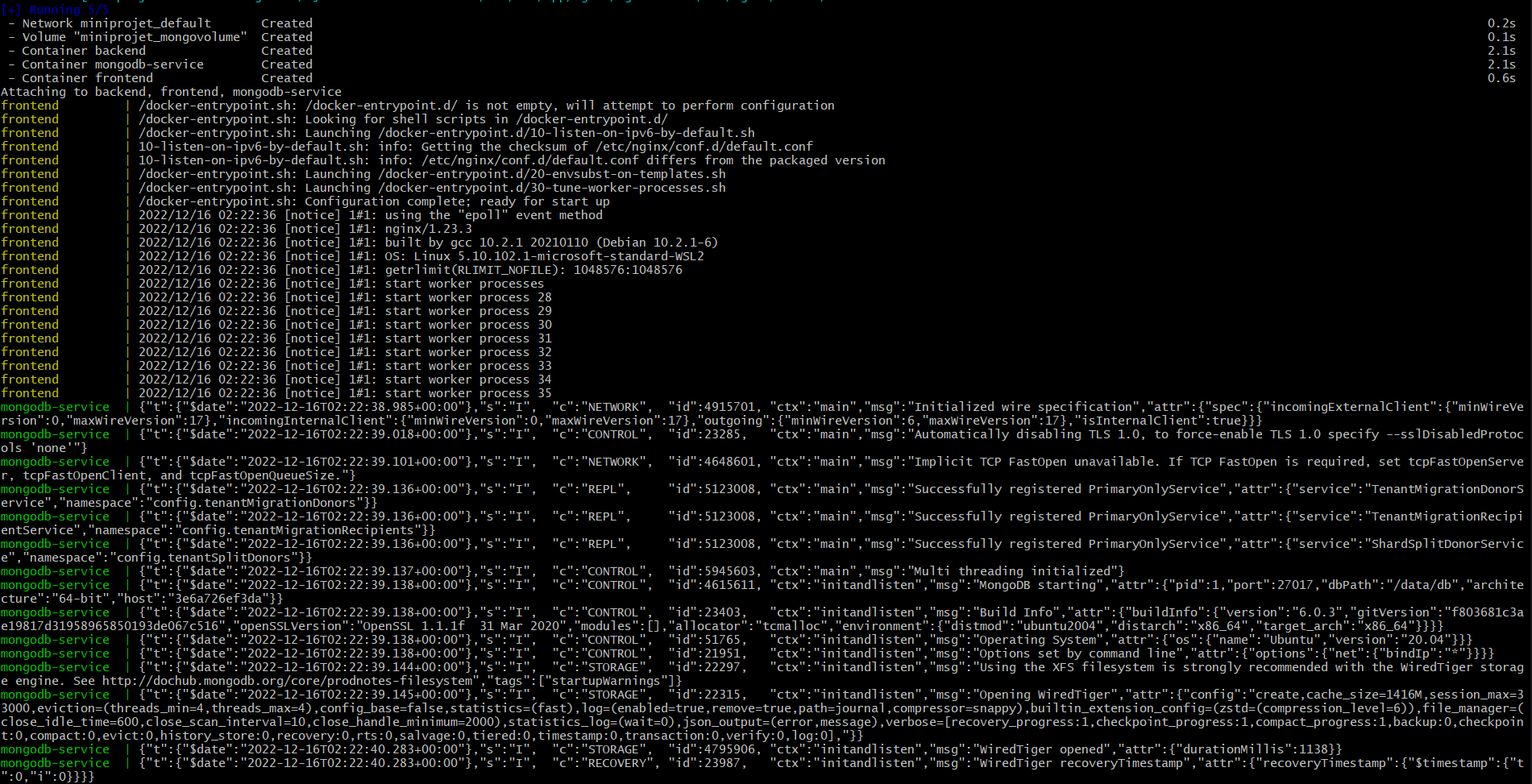


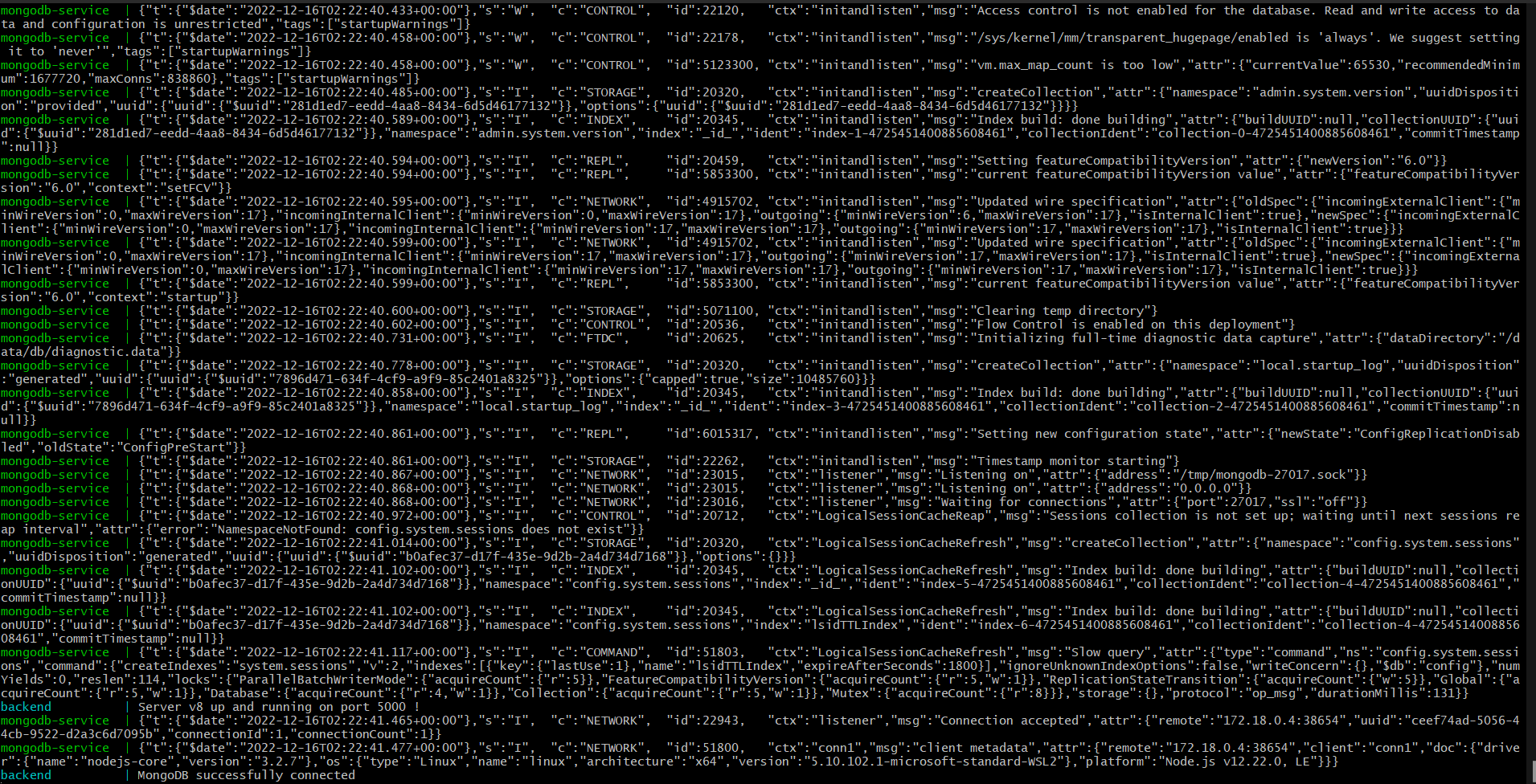
1. Redéployer l’application (frontend, backend et base de données) en utilisant docker-compose.
   1. Créer le fichier docker-compose.

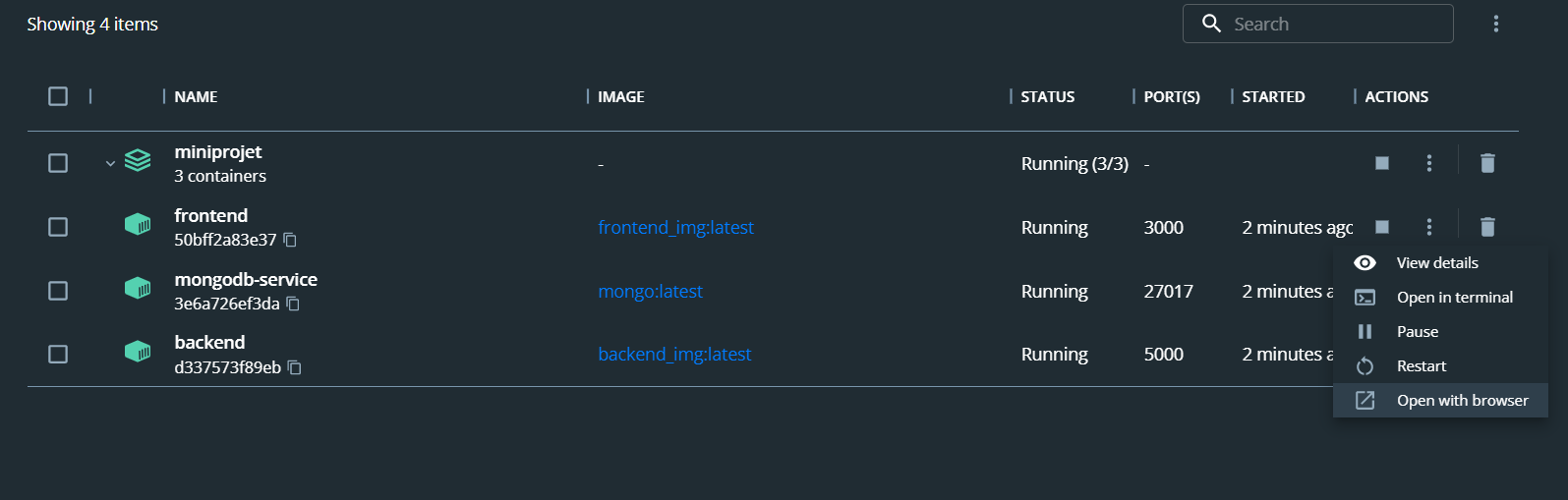


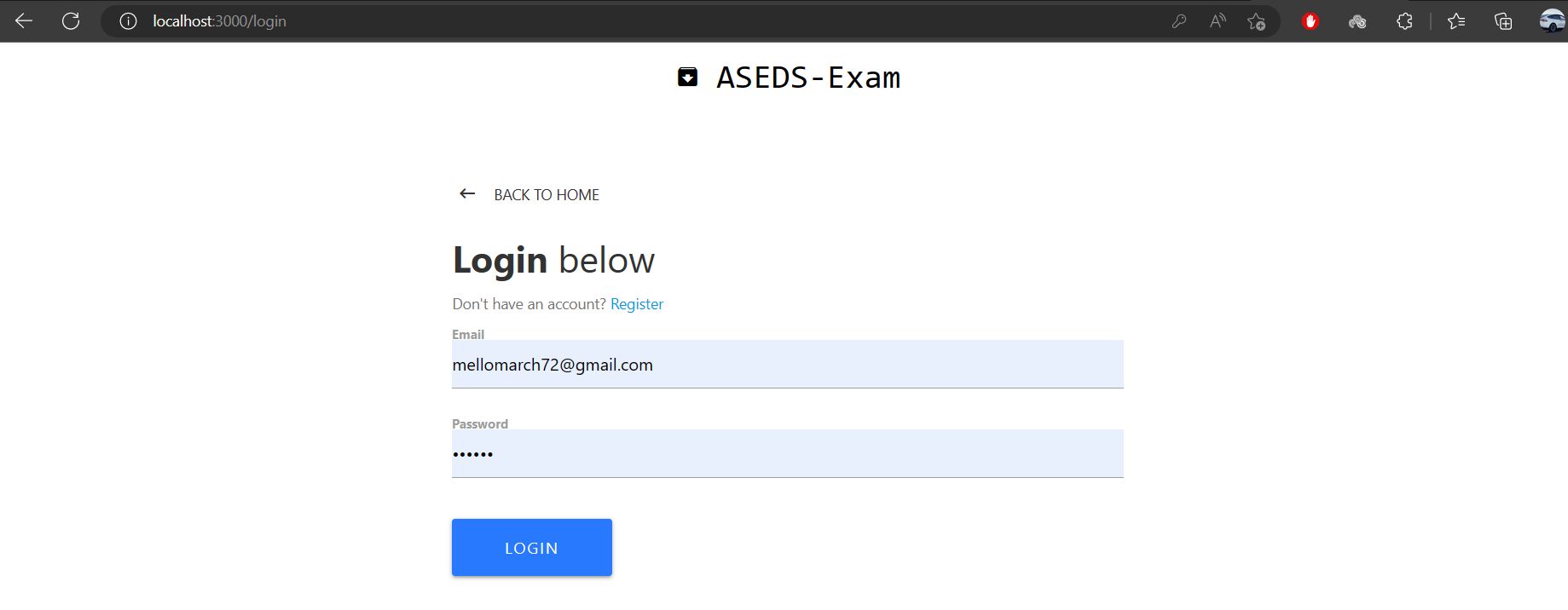
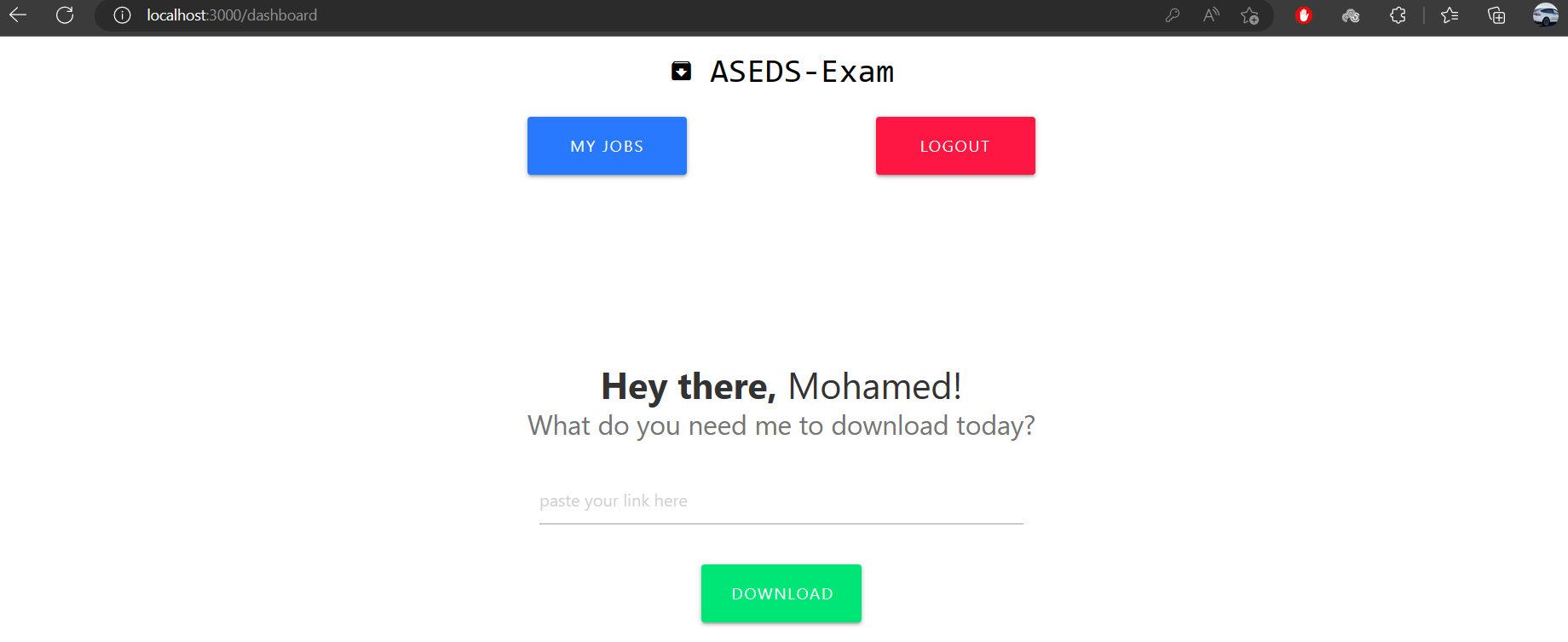
* 1. Taper la commande docker permettant de l’exécuter.



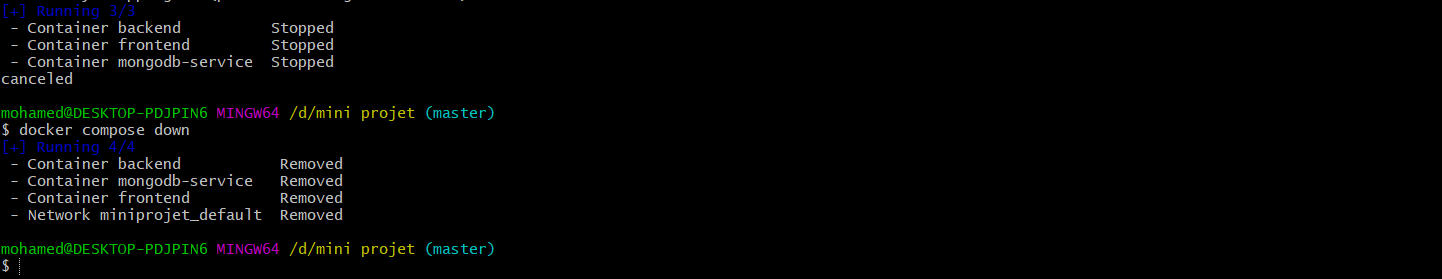


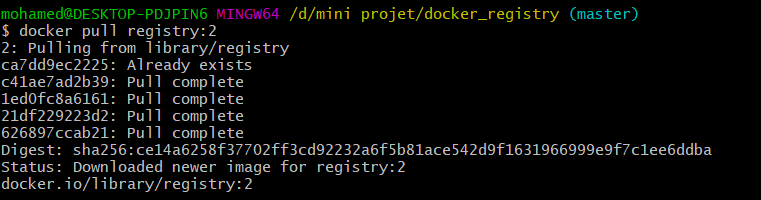


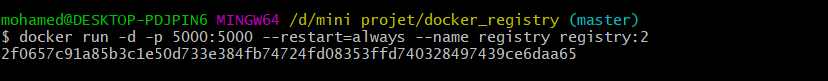
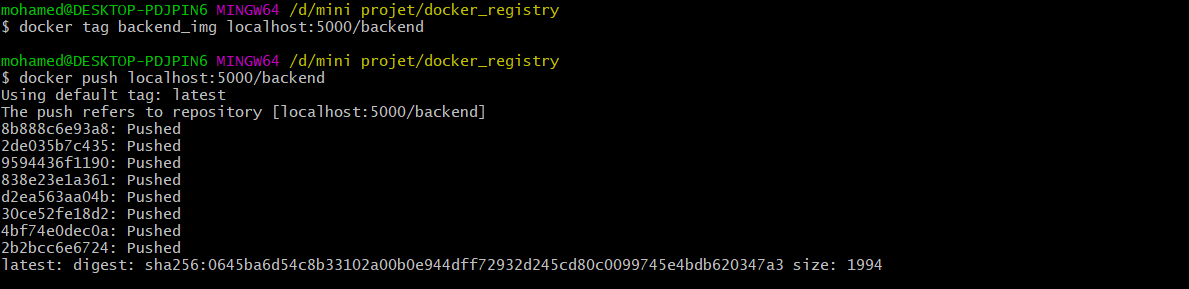
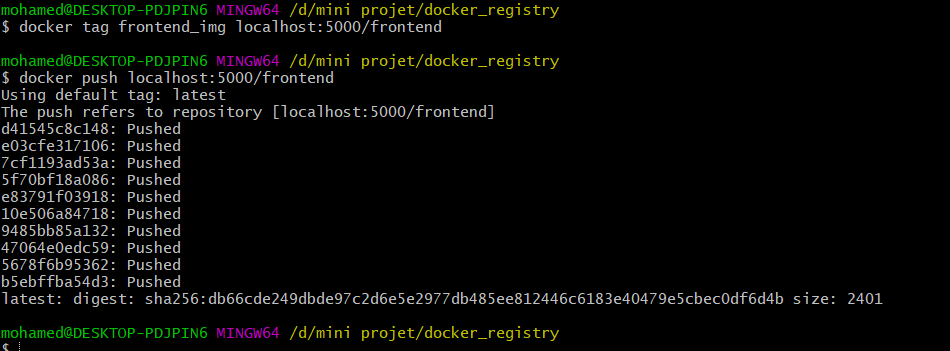
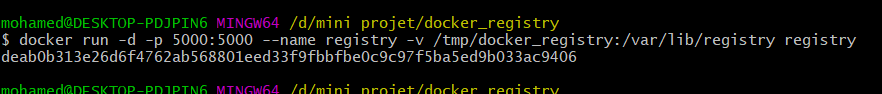
* 1. S’assurer que l’application fonctionne correctement (prises d’écran du navigateur)

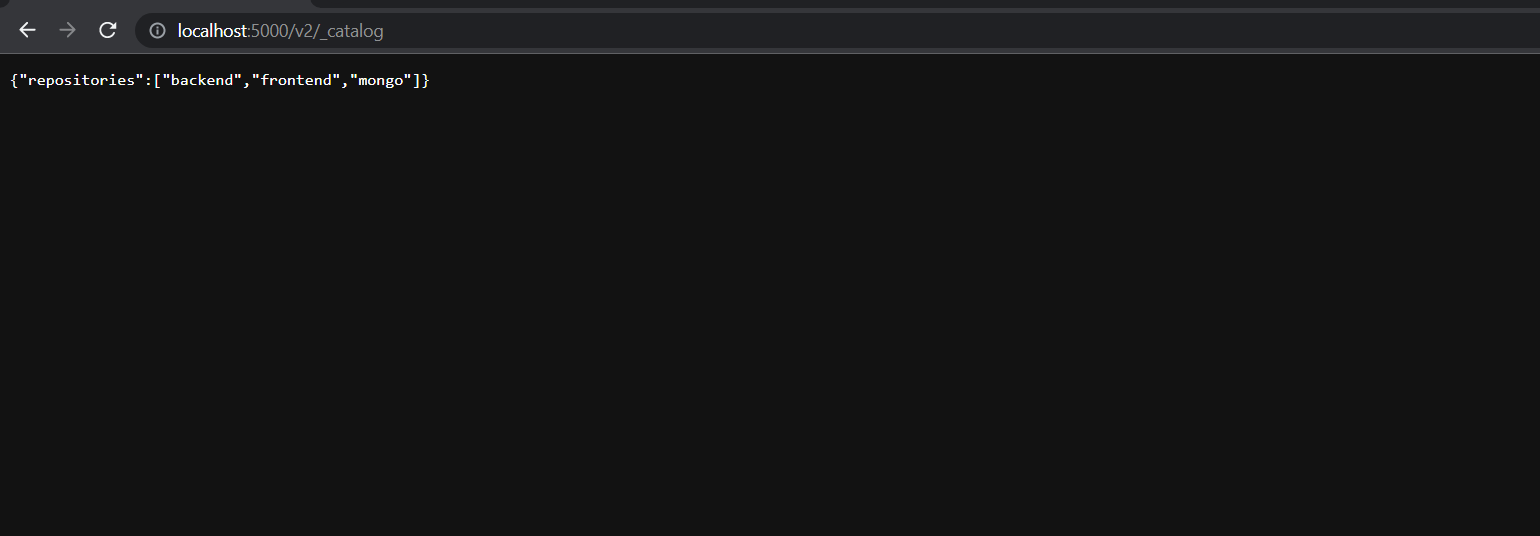
1. Supprimer les conteneurs qui s’exécutent ainsi que les images se trouvant dans le docker host.

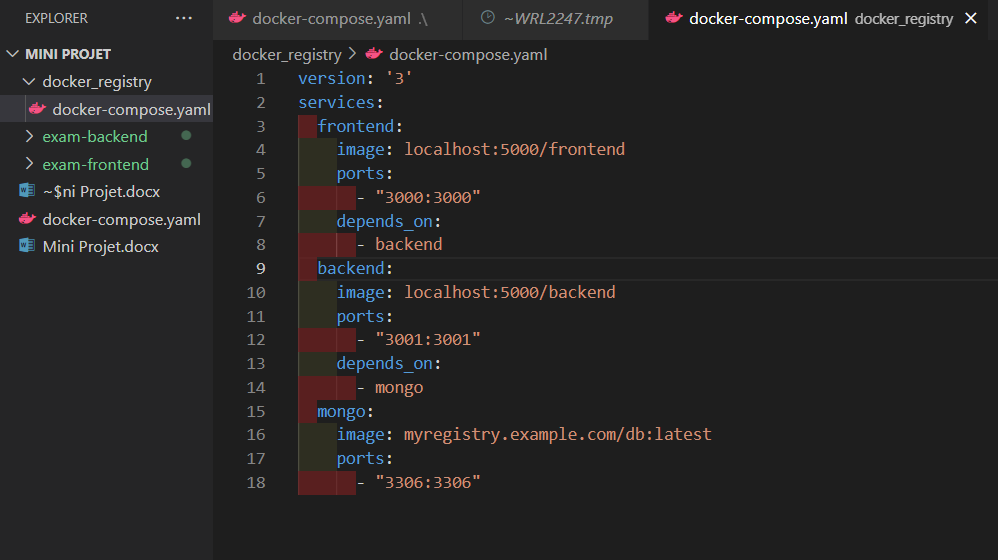


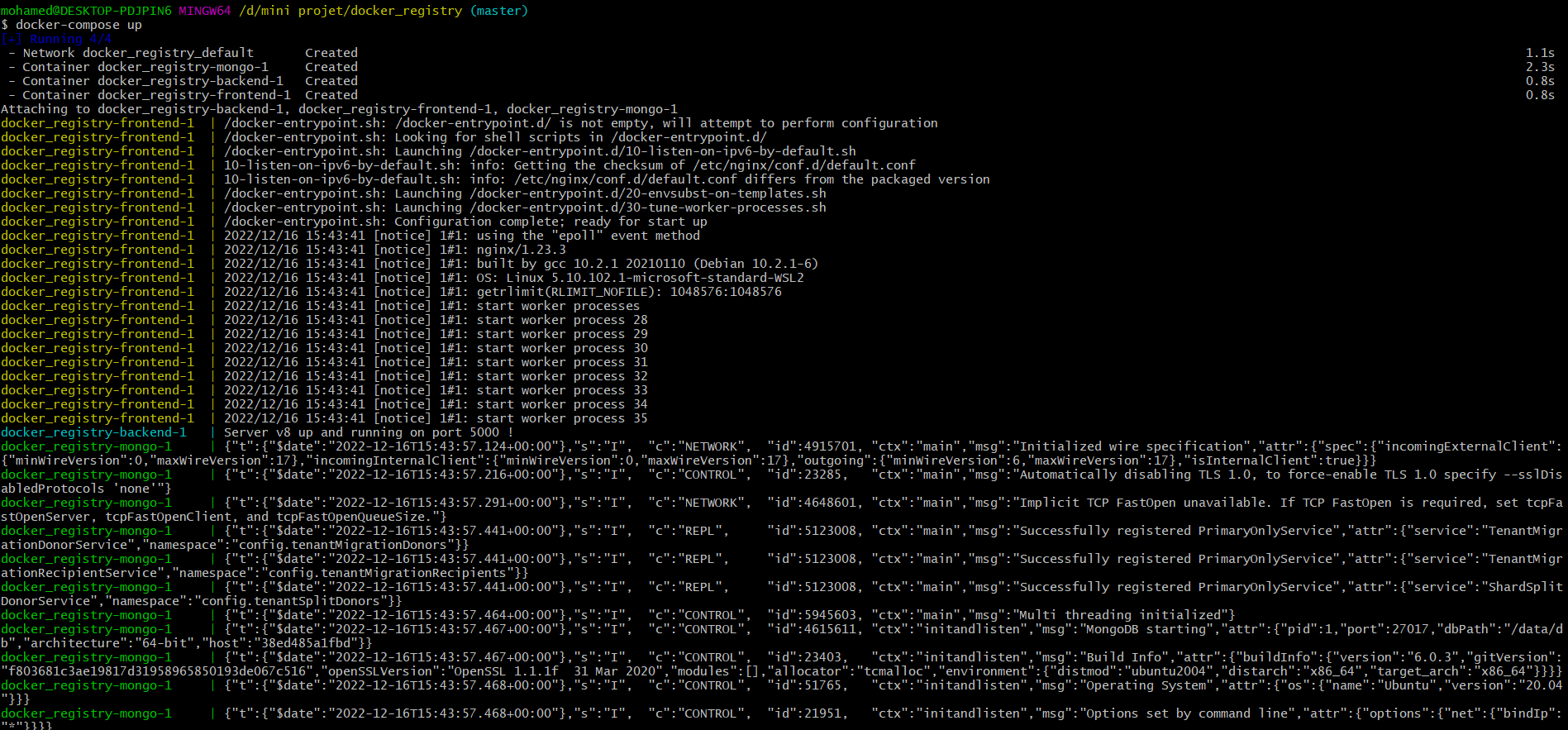
1. Créer un registre privé d’images docker en local qui vous permettra de :
   1. Stocker les images que vous avez construites

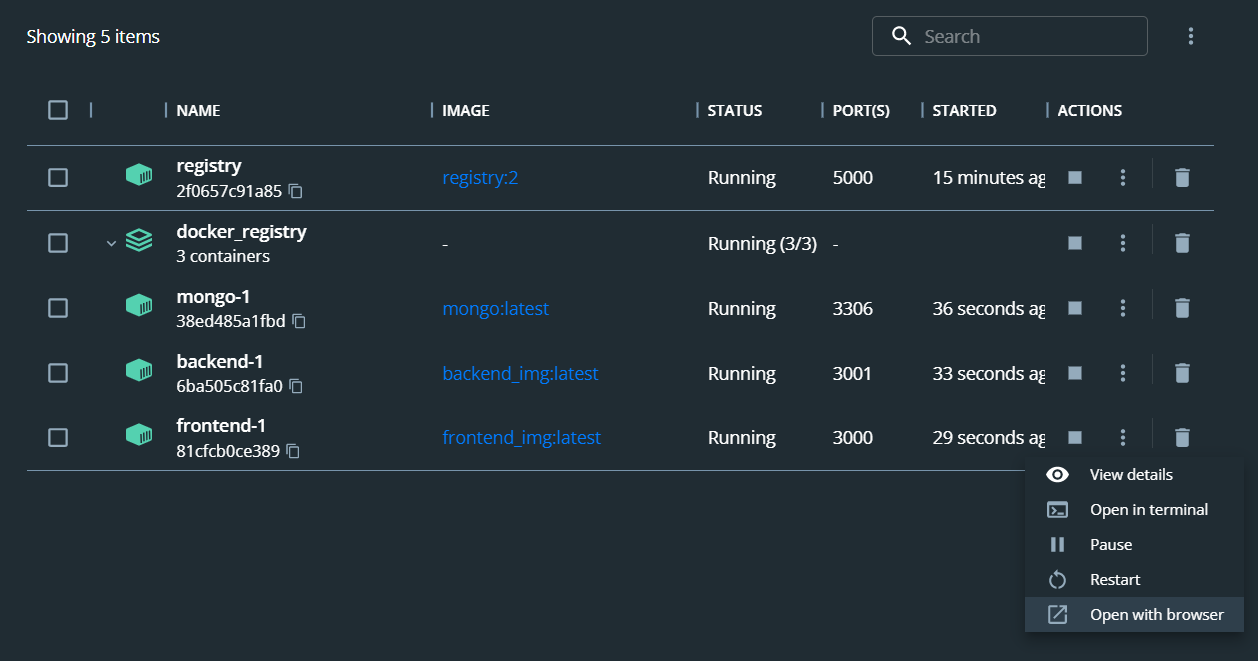


* 1. Visualiser via une UI les images qu’il contient.

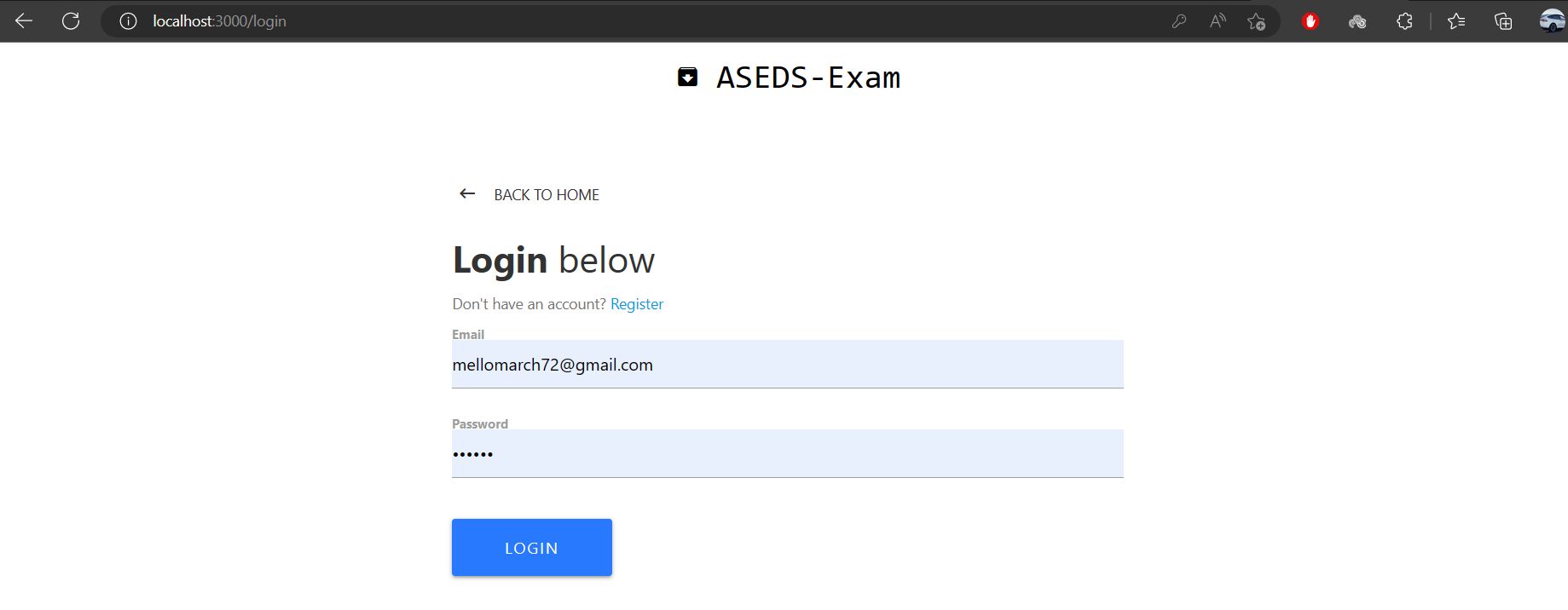
1. Redéployer l’application (frontend, backend et base de données) en utilisant docker-compose tout en récupérant l’image des conteneurs depuis ce nouveau registre privé.

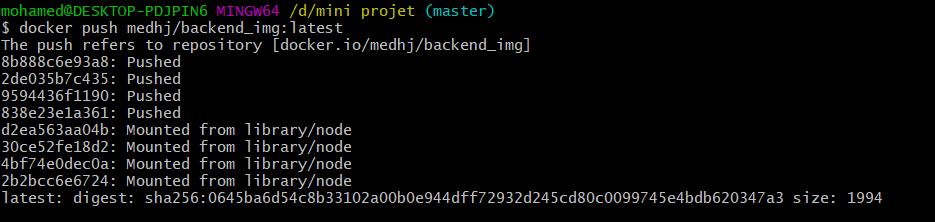
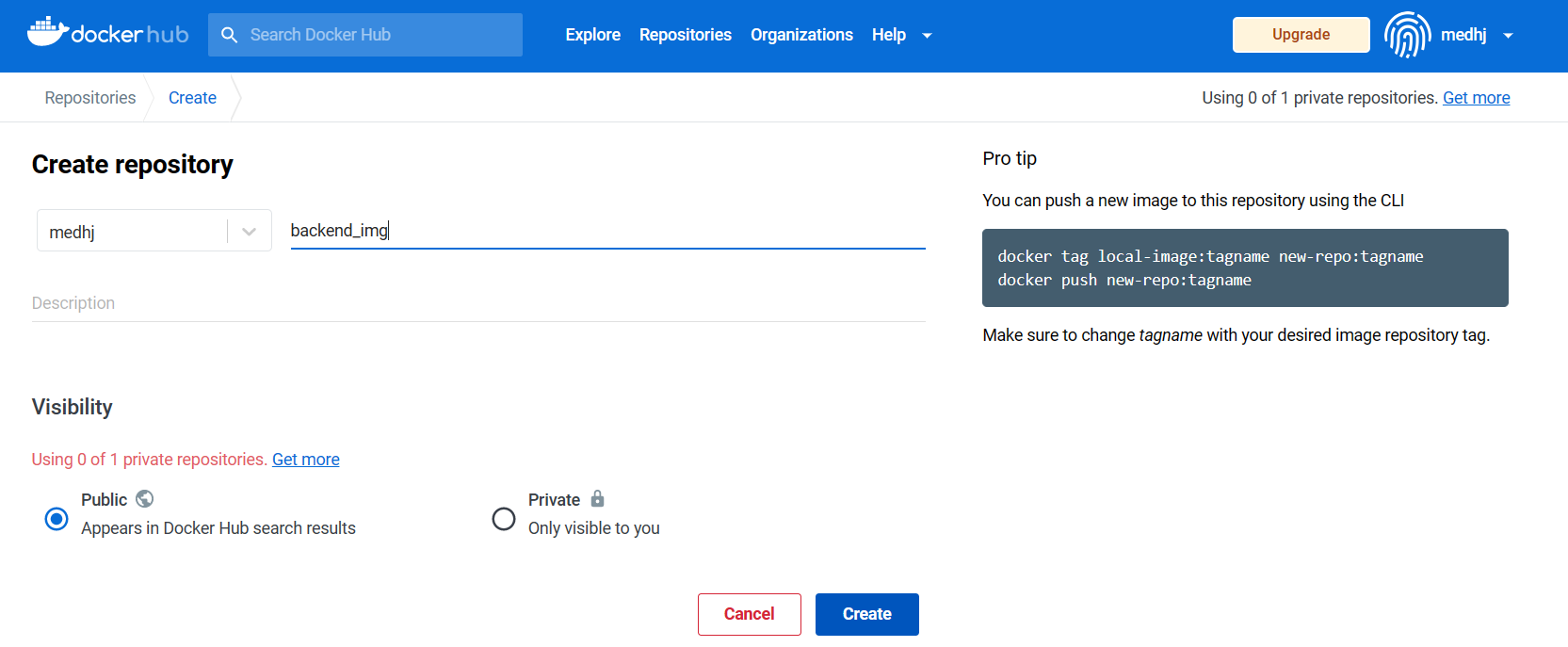


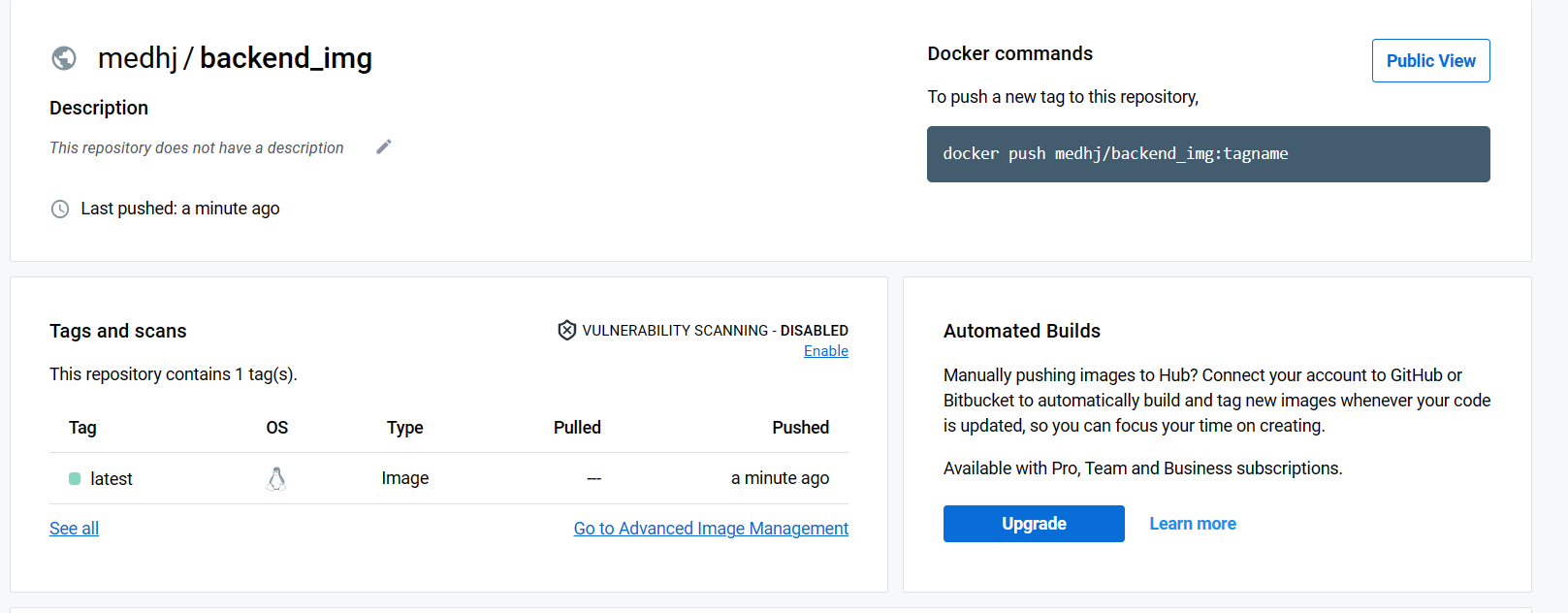


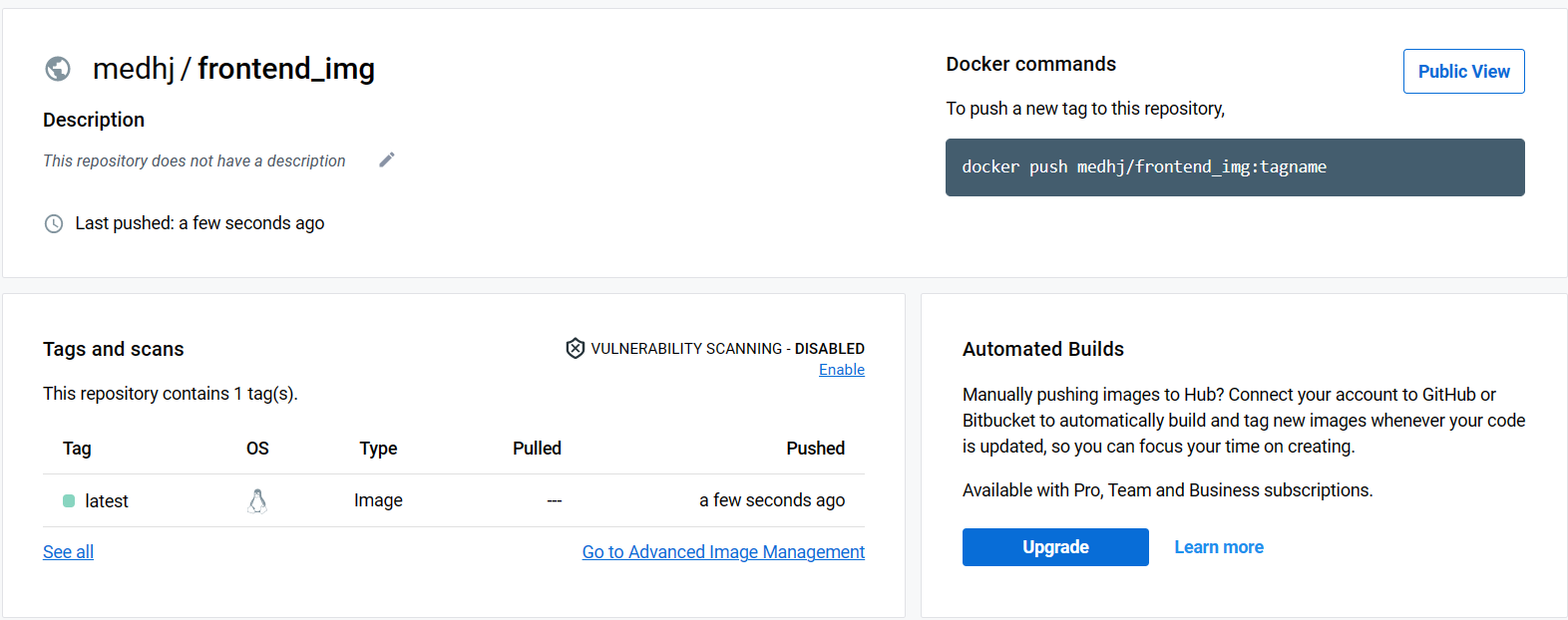
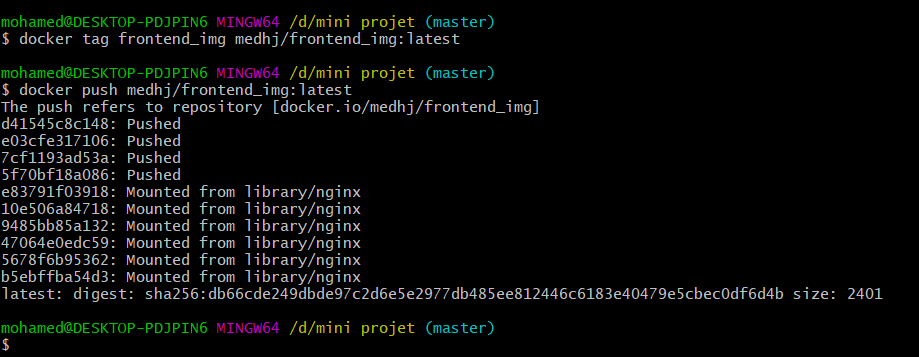
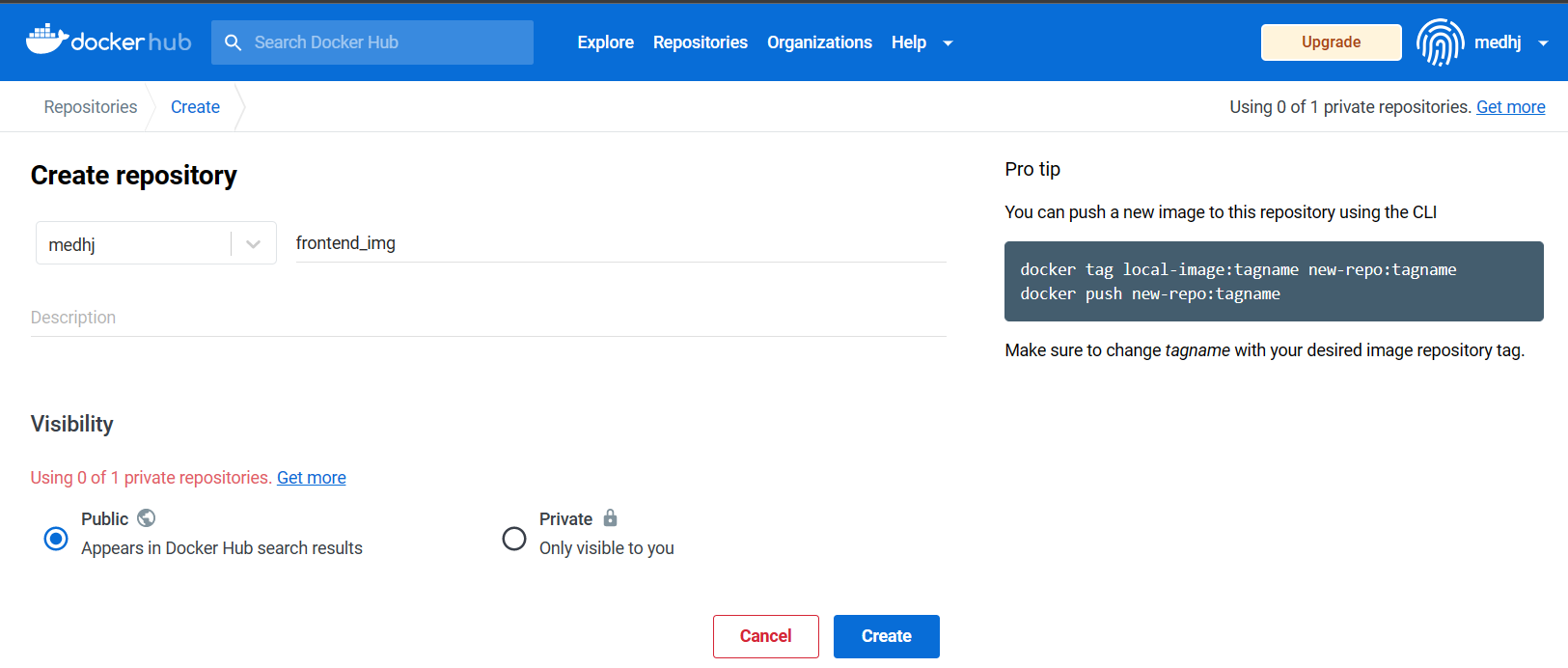






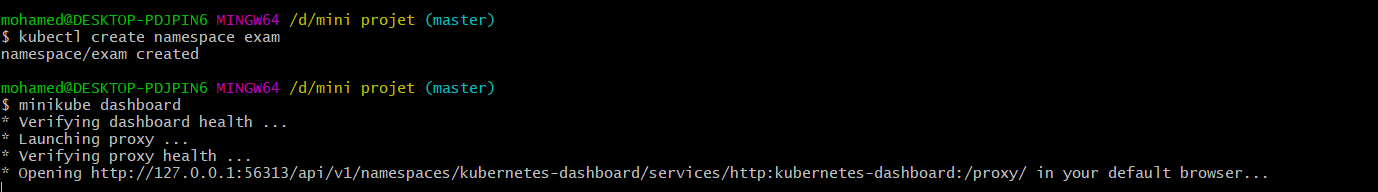




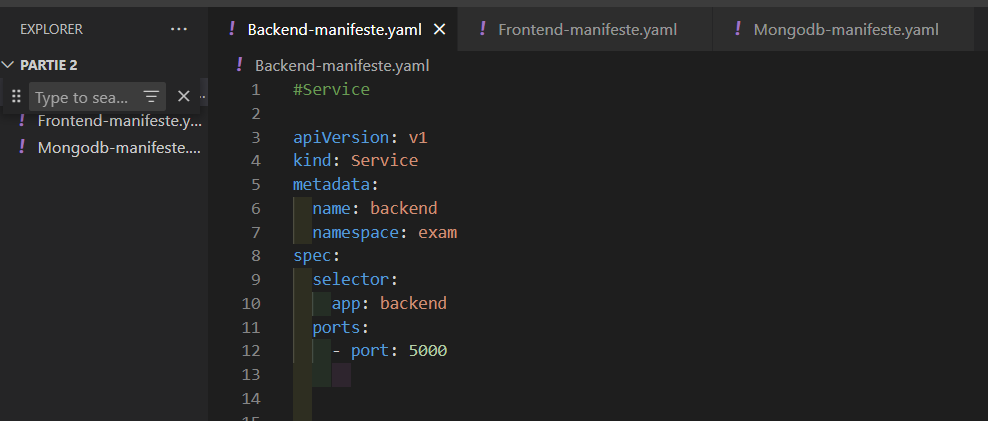
# Partie 2 :

Dans cette partie, il est permis d’utiliser n’importe quel outil permettant de créer un cluster Kubernetes en local (ex : Minikube, MicroK8s, kind, k3d, k3s, etc.)

1. Créer les Kubernetes manifests (fichiers YAML) nécessaires pour déployer cette application dans un cluster Kubernetes (sous un namespace nommé "exam").
   1. Deployment
   2. Service
   3. ConfigMap
   4. Secret
   5. PV / PVC

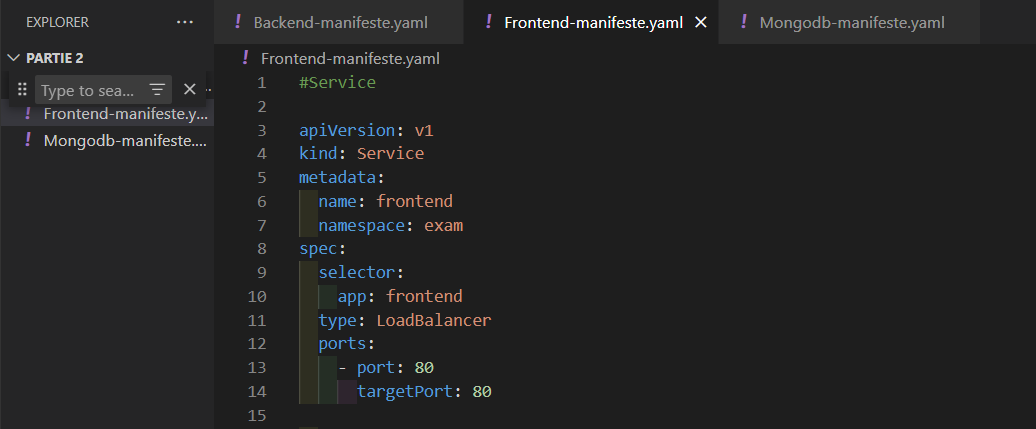
On crée le namespace exam ……………………………………………………………………………………..

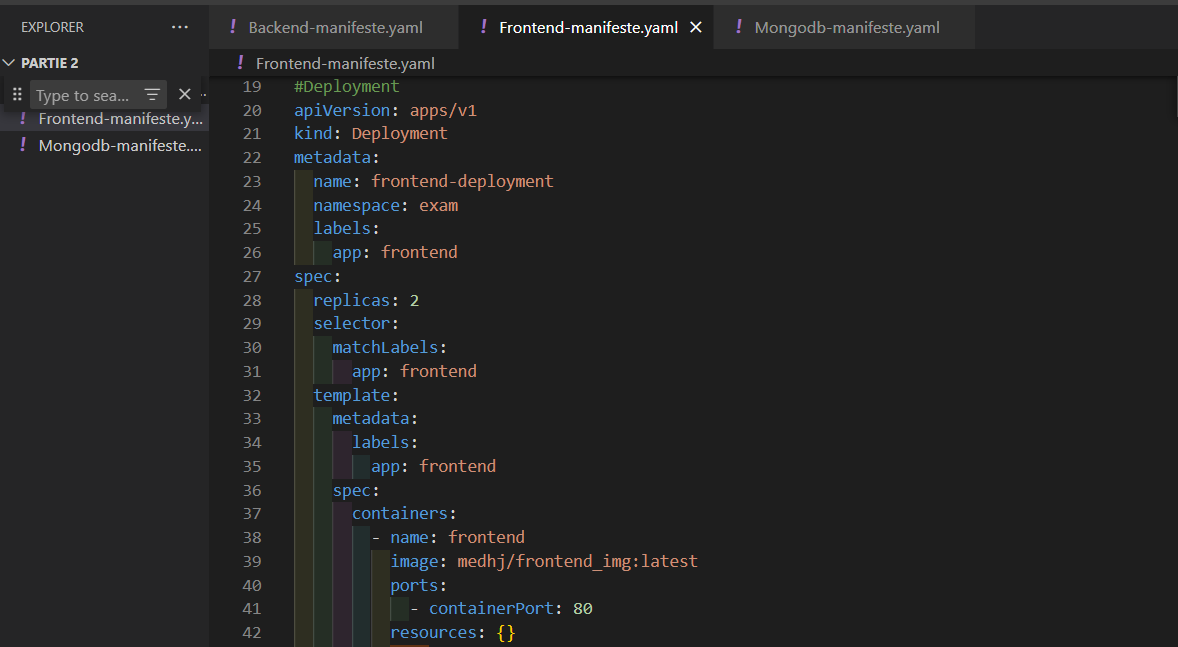
On créer le fichier manifeste pour la backend

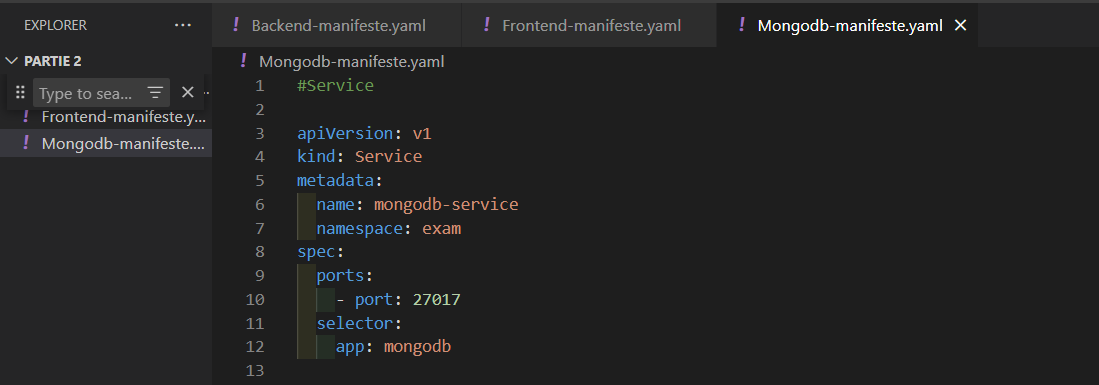


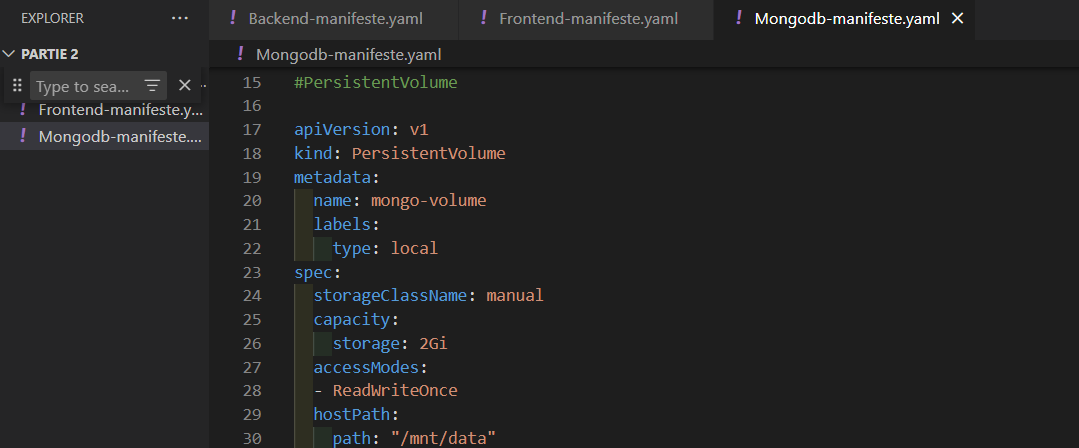


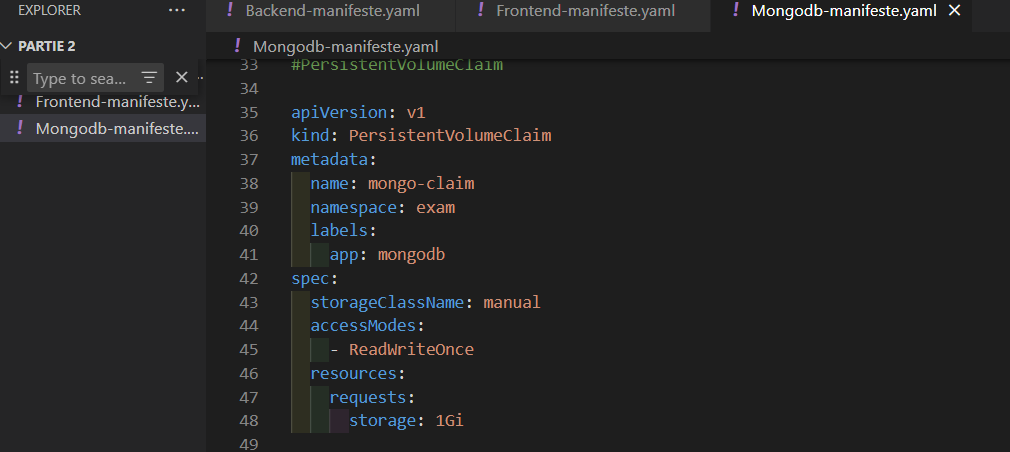
On créer le fichier manifeste pour la frontend

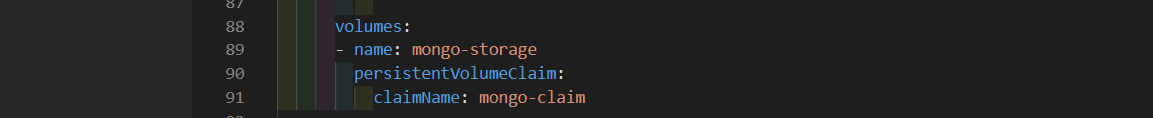
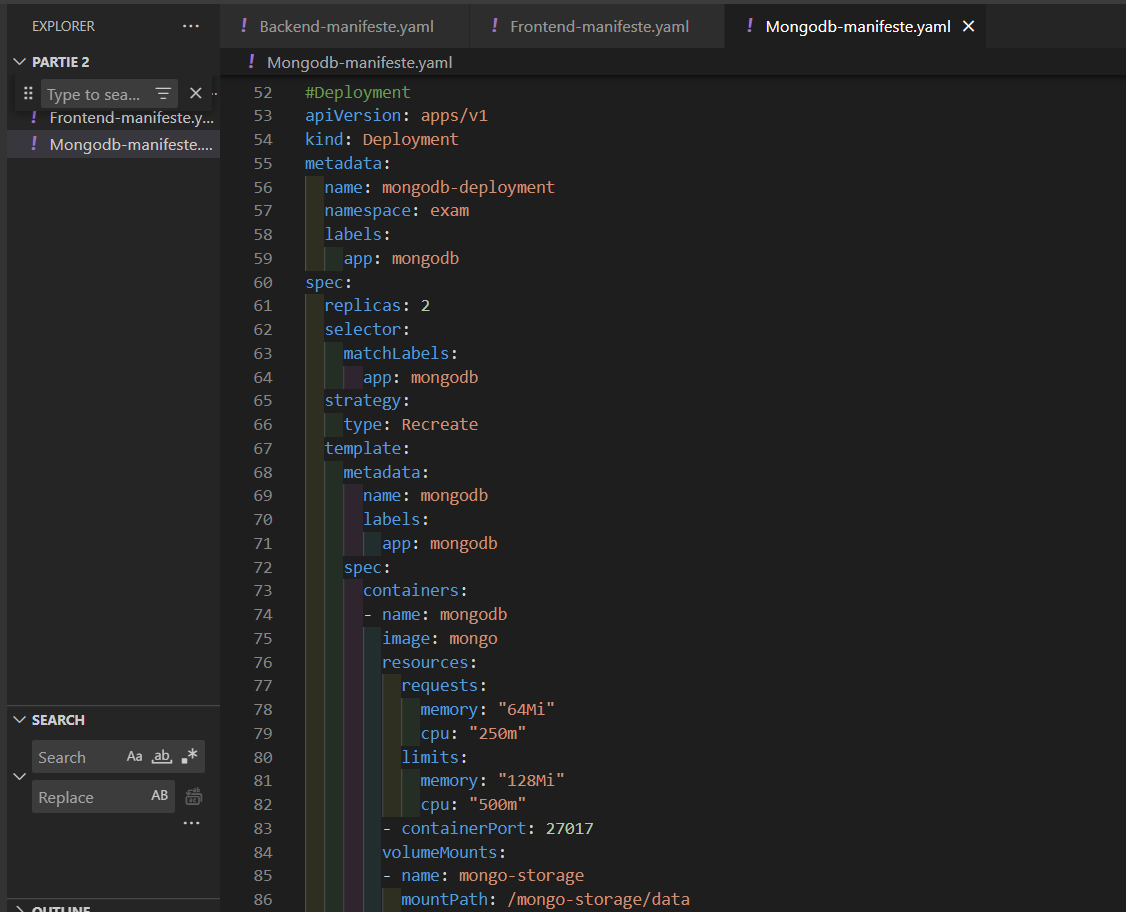




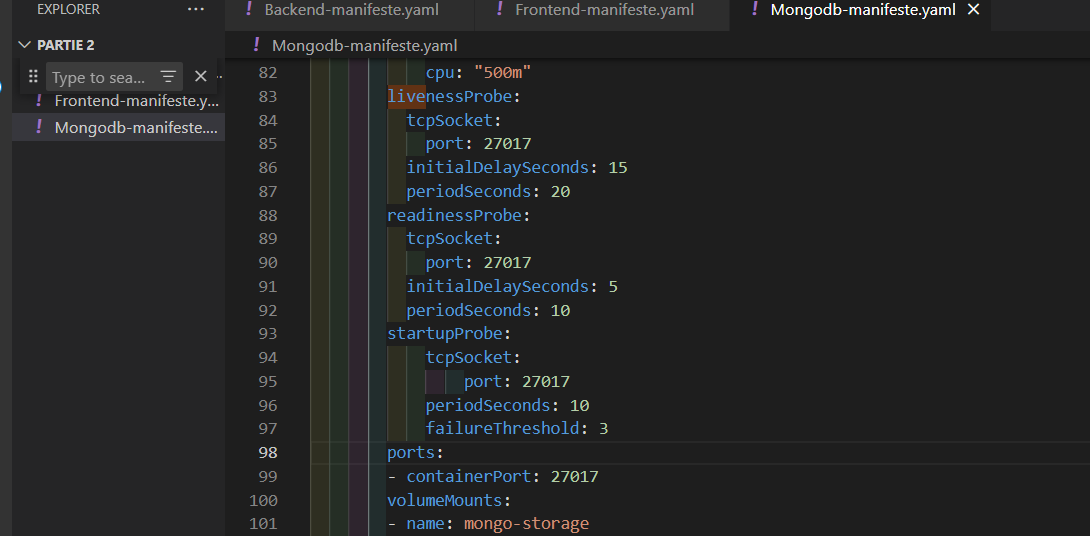
On créer le fichier manifeste pour la base de données

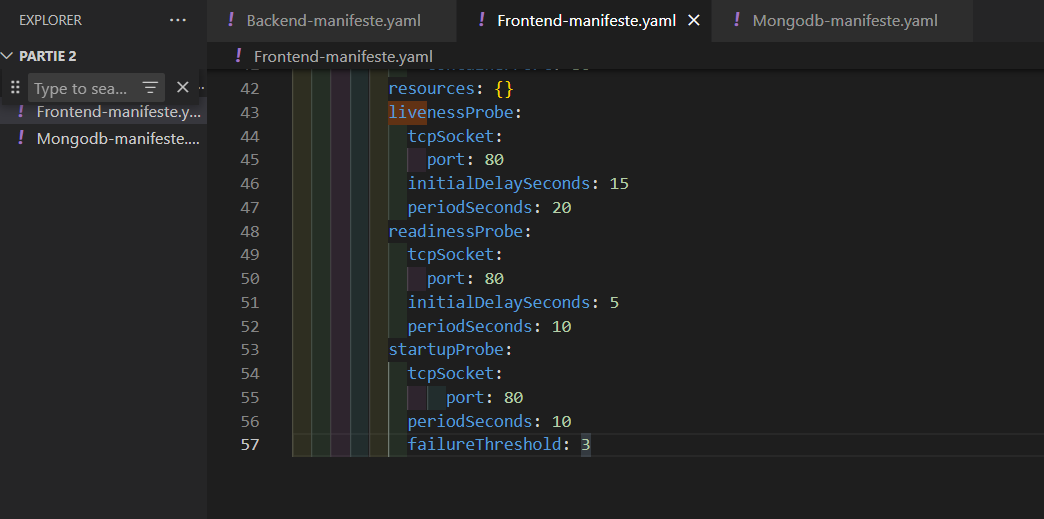




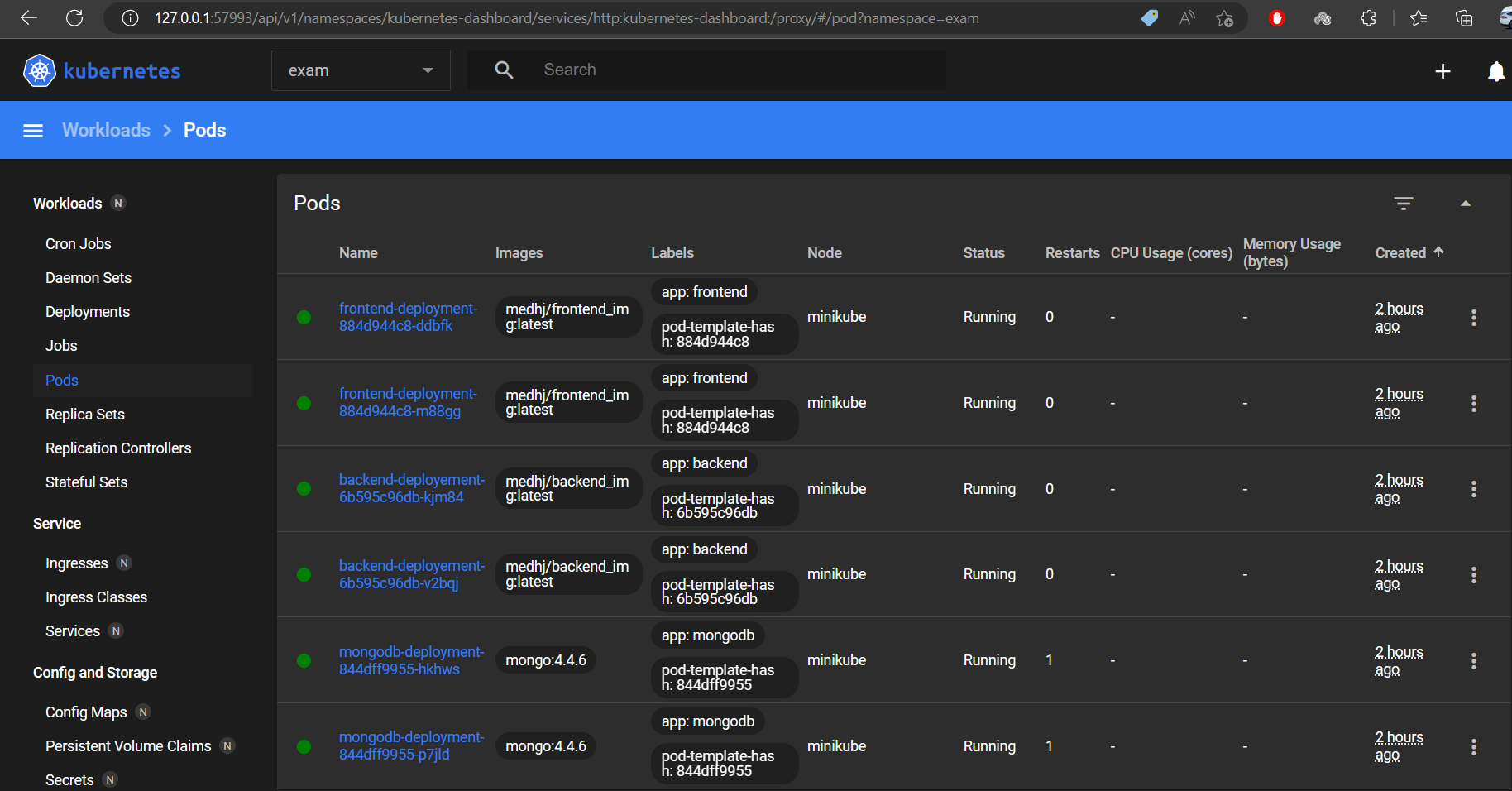


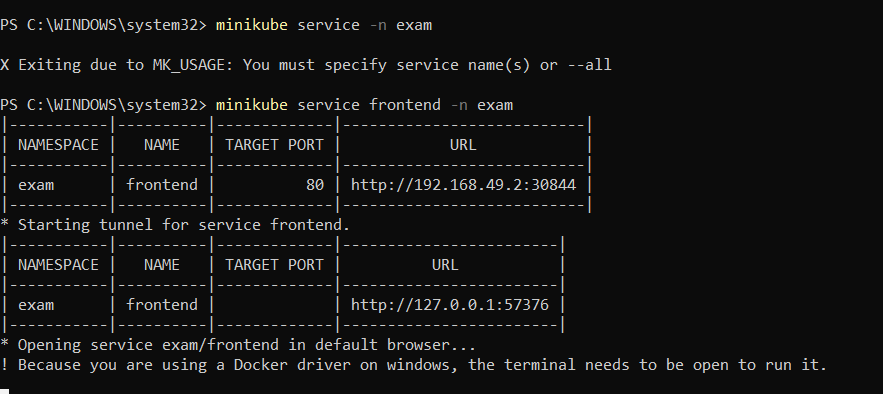
1. Créer et configurer pour cette application un :
   1. Liveness Prob
   2. Readiness Prob
   3. Startup Prob



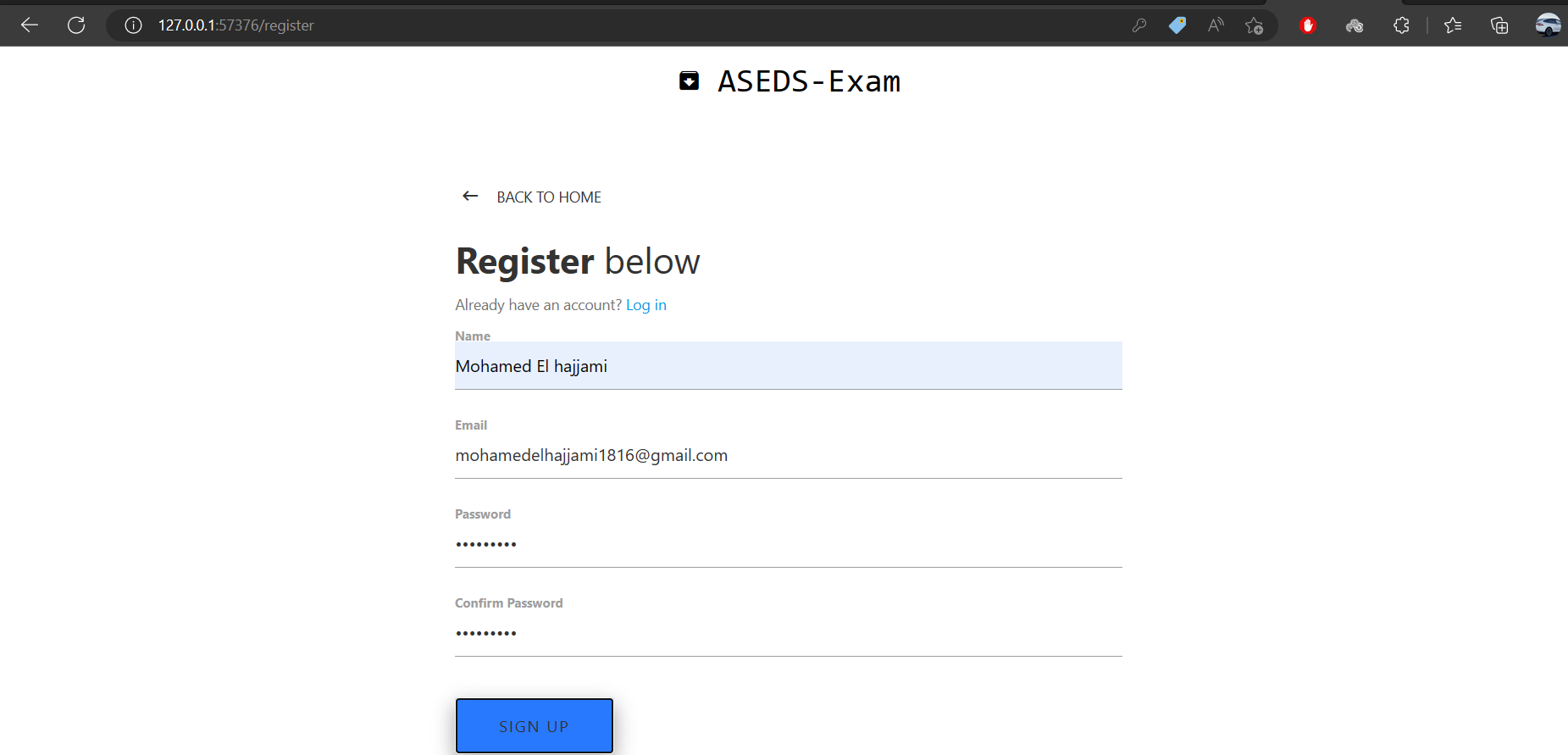


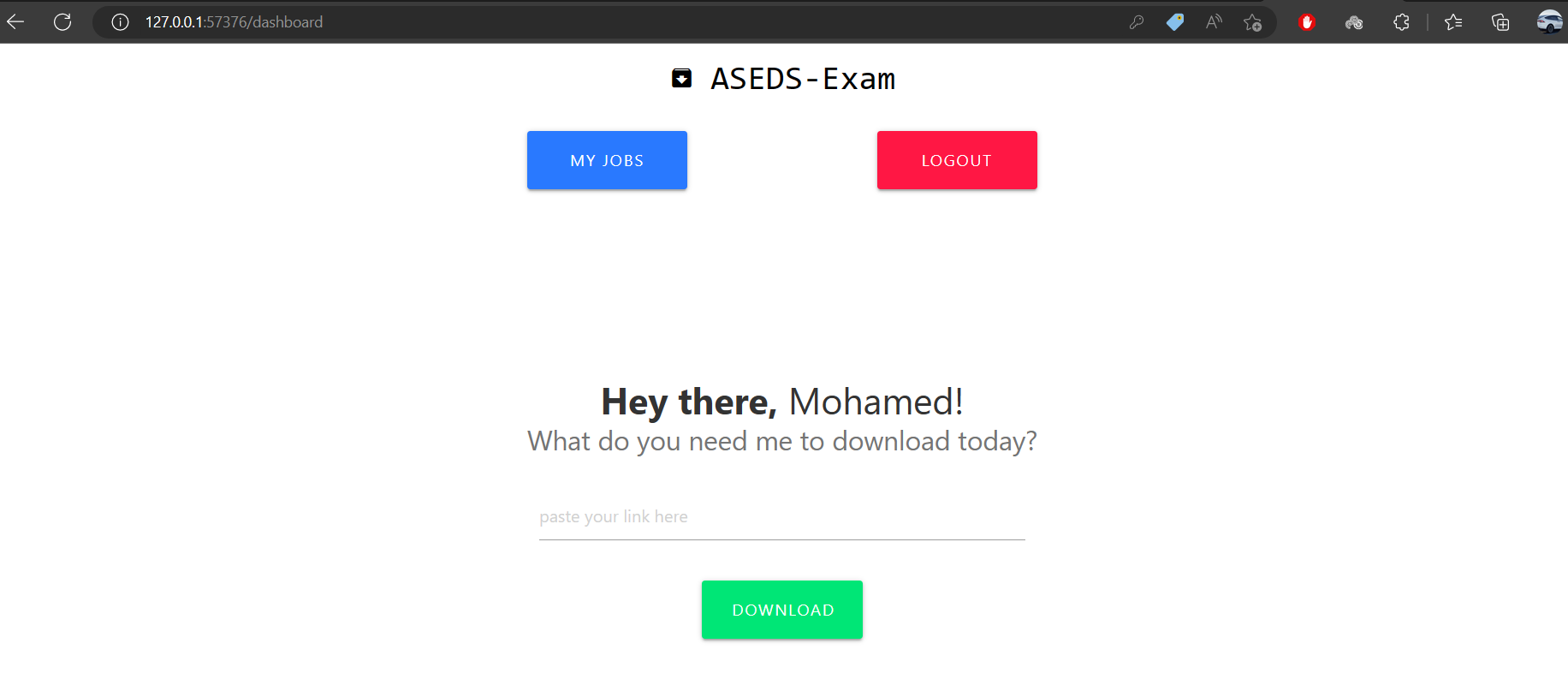
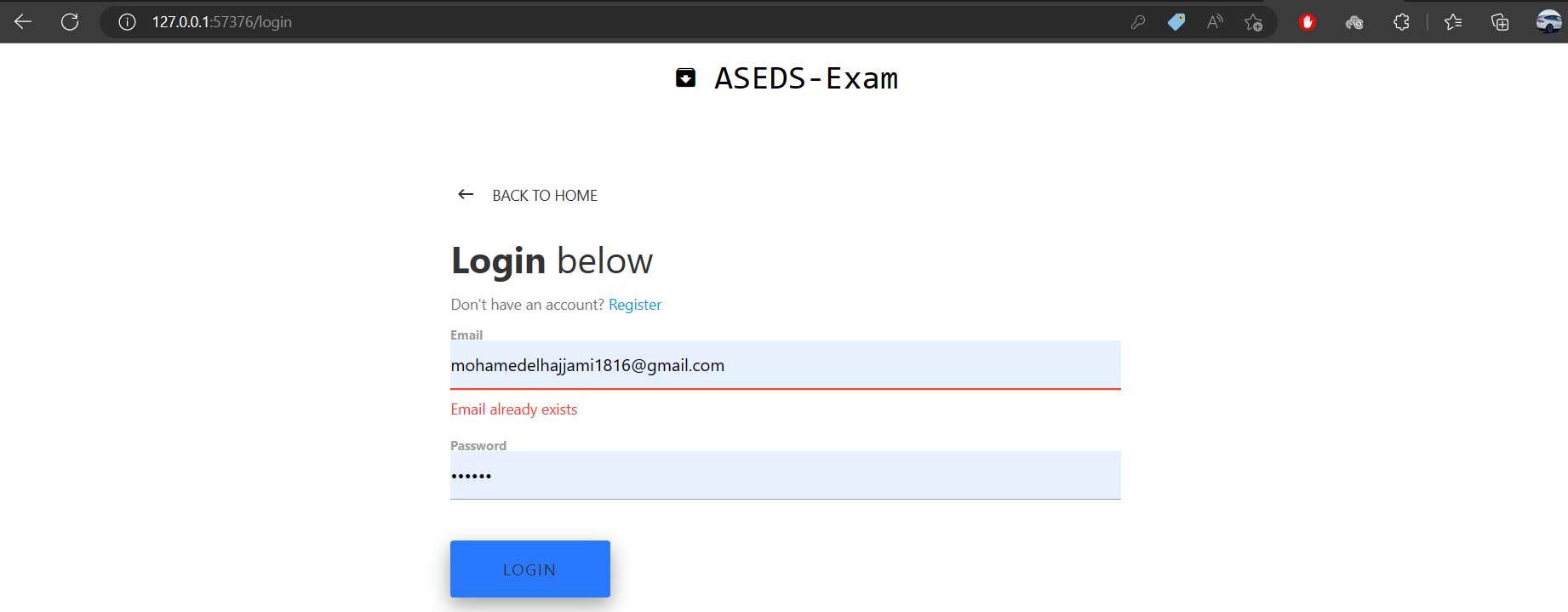




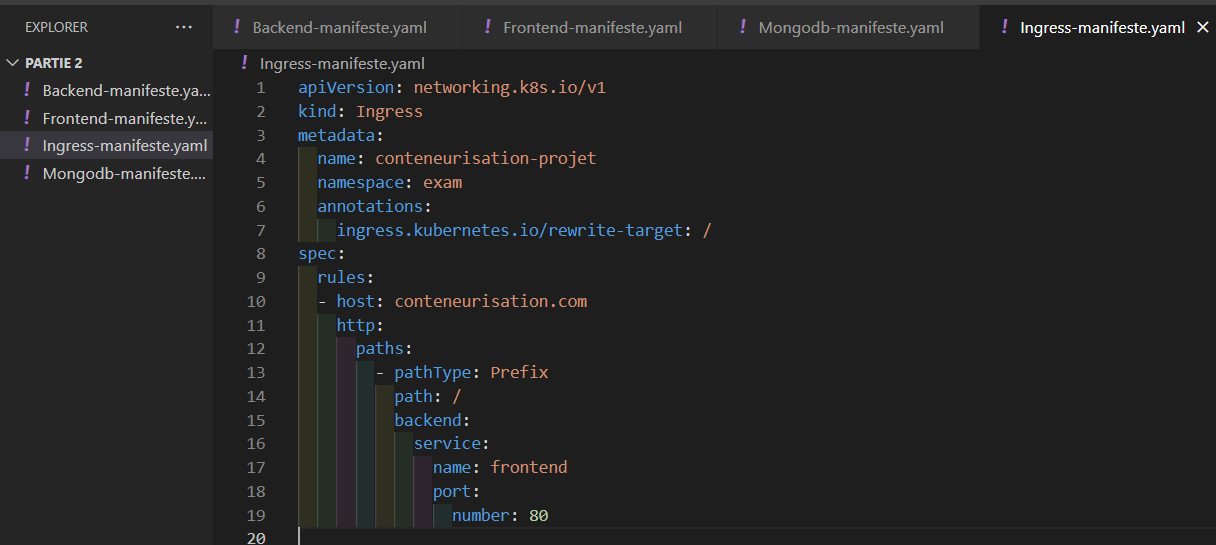


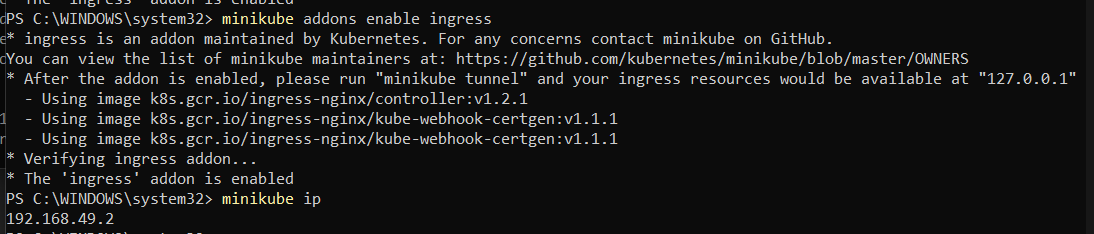
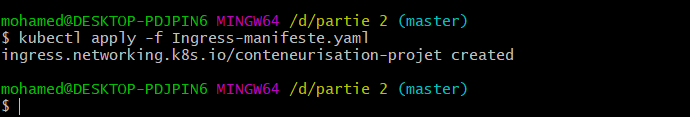
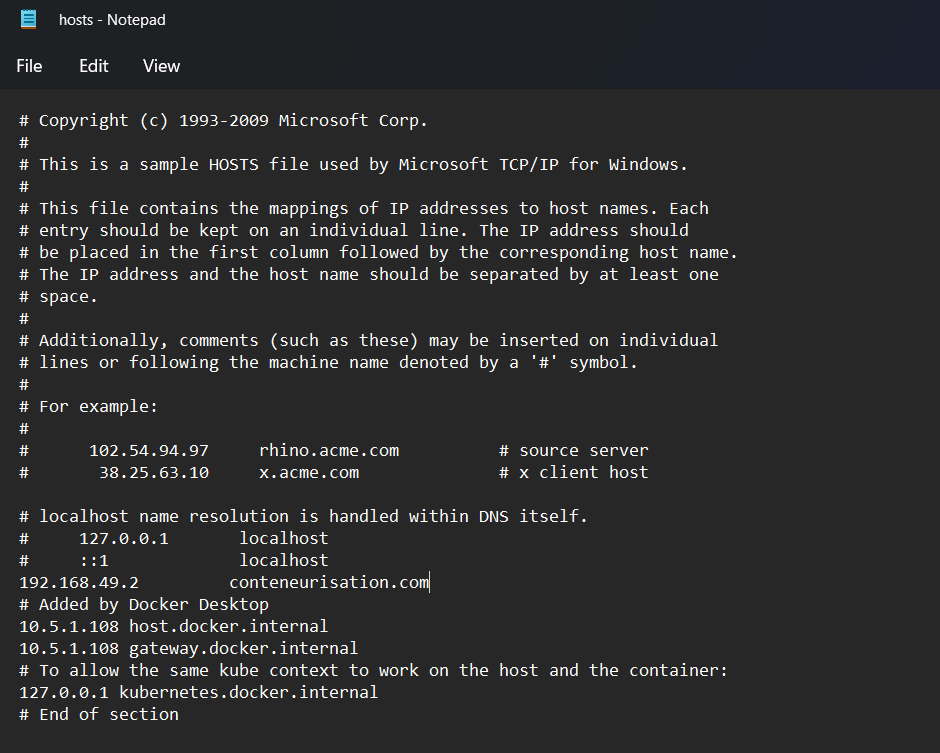






1. Créer un Ingress pour accéder à l’application depuis un navigateur moyennant un nom de domaine.







**INDEX**

Voici le URL de mon repo GitHub :