

MINI- PROJET

CONTENEURISATION DES APPLICATIONS

Réalisé par: Hafsa TALBI

PARTIE 0





Ce projet, intitulé "**Student Management System**", est une application web conçue pour gérer efficacement les informations des étudiants. Elle offre une interface utilisateur intuitive permettant de réaliser les opérations suivantes :

- **Ajouter un étudiant** : Un formulaire dédié permet de saisir les informations essentielles, notamment le prénom, le nom, l'adresse email, la note, et la date de naissance.
- **Afficher la liste des étudiants** : Une table interactive liste tous les étudiants avec leurs données associées, facilitant la consultation rapide.
- **Modifier un étudiant** : Une fonctionnalité d'édition permet de mettre à jour les informations d'un étudiant existant.
- **Supprimer un étudiant** : Une option de suppression garantit une gestion dynamique et à jour des données.

Technologies utilisées :

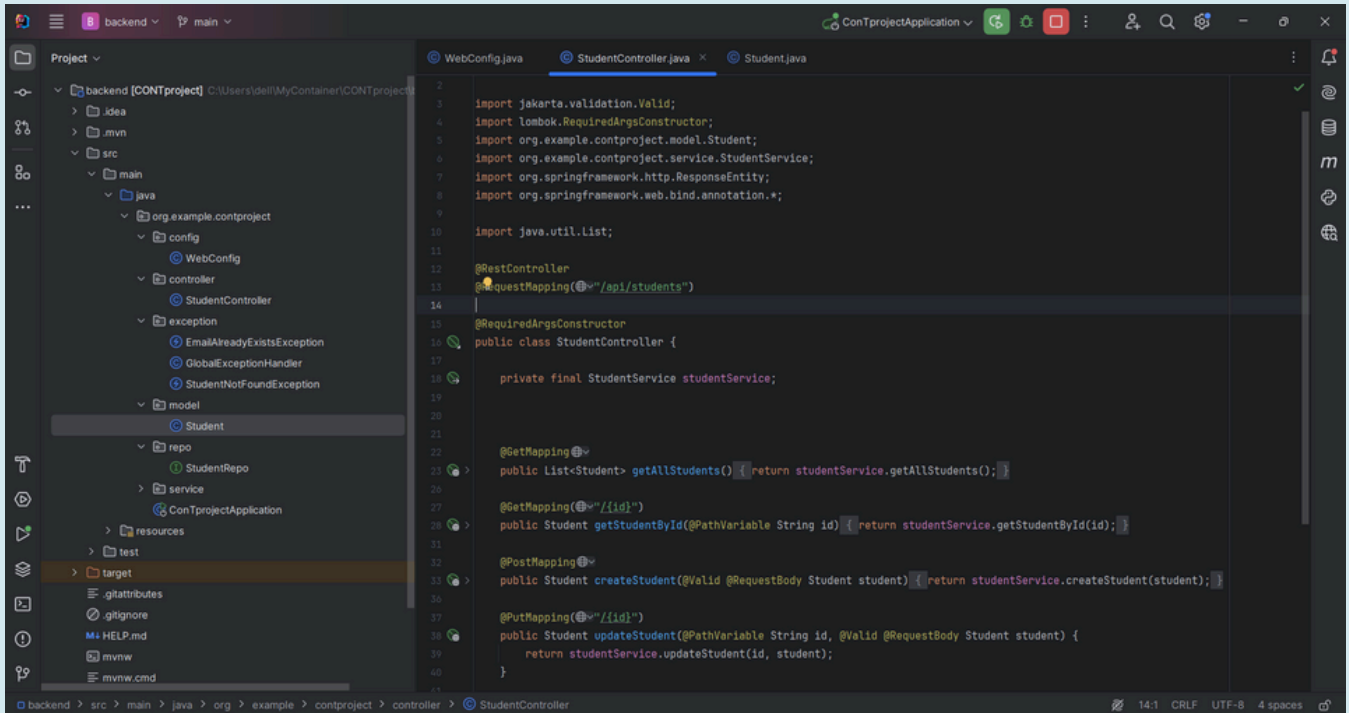
Frontend : Développé avec React, pour créer une interface utilisateur réactive et intuitive.

The screenshot shows a web browser at localhost:5173 displaying the 'Student Management System' interface. The page has a light purple background. At the top, the title 'Student Management System' is centered. Below it, the section 'Add New Student' contains a form with five input fields: 'First Name', 'Last Name', 'Email', 'Grade', and 'Date of Birth' (with a placeholder 'jj/mm/aaaa' and a calendar icon). A blue 'Add Student' button is at the bottom right of the form. Below the form is a table with the following data:

Name	Email	Grade	Date of Birth	Actions
hafa talbi	jjj@gmail.com	20	14/08/2003	 
salima hanane	s.hanane@gmail.com	20	22/07/2003	 

PARTIE 0

Backend : Construit avec Spring Boot en Java, offrant une API RESTful robuste pour la gestion et le traitement des données.



Base de données : Utilisation de MySQL pour stocker et gérer de manière sécurisée et efficace les informations des étudiants.

```
mysql> select * from student ;
+-----+-----+-----+-----+-----+-----+
| id          | date_of_birth | email          | first_name | grade | last_name |
+-----+-----+-----+-----+-----+-----+
| 0b772fcd-c554-4dd4-9d3f-a8ec497d3018 | 2003-08-14    | hafsatlbb@gmail.com | hafsa      | 20    | talbi     |
| f0d01e1e-0bec-4645-8150-78403dd3f4c4 | 2003-07-22    | s.hanane@gmail.com  | salima     | 20    | hanane    |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

PARTIE 1

1. Créer un réseau Docker (tout en lui précisant un driver convenable).

```
PS C:\Users\de11\conteneurisation mini project\conteneurisation> docker network create --driver bridge my_network
0b0e675ee667564954e88b3b312036c93b470fbcc16c767e412d6f00a4d86041
```

2. Créer un conteneur pour la base de données MySQL en instanciant l'image officielle mysql

a. L'utilisation d'un volume Docker pour la persistance des données est la meilleure option, car il offre une gestion plus propre et plus fiable que le montage de répertoires locaux du système d'exploitation. Les volumes sont gérés par Docker, ce qui permet de les déplacer facilement entre les hôtes Docker si nécessaire, et ils sont indépendants du cycle de vie des conteneurs.

b. La commande utilisée :

```
PS C:\Users\de11\MyContainer\CONTproject> docker volume create mysql_data
mysql_data
```

```
docker run -d --name mysql-container --network my_network -v
mysql_data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=TALtal2024**
mysql:8.0.40-debian
```

Vérification de la connexion du conteneur au réseau :

```
PS C:\Users\de11\conteneurisation mini project\conteneurisation> docker network inspect my_network
[
  {
    "Name": "my_network",
    "Id": "0b0e675ee667564954e88b3b312036c93b470fbcc16c767e412d6f00a4d86041",
    "Created": "2024-12-08T12:14:29.931976111Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.20.0.0/16",
          "Gateway": "172.20.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "58ca5befa00c9d0afa6b373b7aa4e2bad32f4221d38066d5f8496f280a97aa5c": {
        "Name": "mysql-container",
        "EndpointID": "7668f6046cb45d8462d8dab6f93fac8c5cdcf24b7eb66939c7d525dafa8f745d",
        "MacAddress": "02:42:ac:14:00:02",
        "IPv4Address": "172.20.0.2/16",
        "IPv6Address": ""
      }
    }
  }
]
```

3. Créer un fichier Dockerfile pour le projet Backend :

```
# Phase 1: Build Stage
FROM maven:3.9.4-eclipse-temurin-21-alpine AS builder

WORKDIR /app

COPY pom.xml mvnw ./
COPY .mvn .mvn/

RUN ./mvnw dependency:resolve

COPY . .
RUN ./mvnw clean package -DskipTests

# Phase 2: Runtime Stage
FROM eclipse-temurin:21-jre-alpine

WORKDIR /app

COPY --from=builder /app/target/student-management-0.0.1-SNAPSHOT.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

4. Lister et expliquer "en détails" ces bonnes pratiques appliquées.

- **Utilisation de Multi-Stage Builds** qui permettent de construire le projet dans une première phase (avec Maven) et de copier uniquement les fichiers nécessaires dans l'image finale afin de réduire la taille de l'image finale.
- **Choix d'images légères (Alpine)** pour garantir une taille minimale de l'image Docker.
- **Gestion des dépendances via Maven:** Le Dockerfile copie le fichier pom.xml en premier pour télécharger les dépendances Maven avant de copier les sources du projet pour que les dépendances ne sont pas téléchargées à chaque modification du code, ce qui accélère les temps de build Docker si seul le code source est modifié.
- **Utilisation d'un répertoire de travail explicite:** Cela garantit que tous les fichiers copiés et exécutés dans l'image se trouvent dans un emplacement connu.
- **Exposition explicite des ports** EXPOSE 8080 indique le port utilisé par l'application afin de clarifier la configuration réseau pour d'autres développeurs ou outils .
- **Commande d'entrée standardisée:** Utilisation de ENTRYPOINT pour exécuter l'application avec Java Standardise le démarrage de l'application et facilite la maintenance et la lisibilité.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker build -t talbihafsa/my_backend:1.0.0 .
```

5.

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

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker build -t my_backend .
```

- **build** : Indique à Docker de créer une image à partir d'un Dockerfile.
- **-t my_backend** : Spécifie un tag pour l'image, ici nommé my_backend, afin de l'identifier facilement.
- **.** : Indique à Docker d'utiliser le répertoire courant comme contexte de construction contenant le Dockerfile et les fichiers nécessaires.

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

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker scout quickview my_backend
```

- **scan** : Exécute une analyse de sécurité pour détecter les vulnérabilités connues dans l'image spécifiée.

Aucune vulnérabilité spécifique n'a été détectée pour l'image Docker my_backend, mais docker scout m'a suggéré de mettre à jour l'image de base (eclipse-temurin:21-jre-alpine) vers une version plus récente (eclipse-temurin:23-jre-alpine)

Target	my_backend:latest	0C	0H	0M	0L
digest	1da90674d6b3				
Base image	eclipse-temurin:21-jre-alpine	0C	0H	0M	0L
Updated base image	eclipse-temurin:23-jre-alpine	0C	0H	0M	0L

c. Publier cette image dans Docker Hub.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker login
Authenticating with existing credentials...
Login Succeeded
```

- **login** : Lance l'authentification auprès de Docker Hub.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker push talbihafsa/my_backend:1.0.0
```

- **push** : Transfère l'image spécifiée vers Docker Hub.
- **talbihafsa/my_backend** : Nouveau nom de l'image, inclut mon nom d'utilisateur Docker Hub.

d. Instancier cette image en créant un conteneur

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker run -d --name backend-container --network my_network -p 5000:5000 talbihafsa/my_backend:1.0.0
1b9836e4941169915718584c9b34c1c1ed73dc14014c19bdc20f6df55c05b88
```

- **run** : Lance un nouveau conteneur à partir de l'image spécifiée.
- **-d** : Exécute le conteneur en mode détaché (en arrière-plan).
- **--name backend-container** : Donne un nom au conteneur pour faciliter sa gestion.
- **--network my_network** : Connecte le conteneur au réseau Docker nommé my_network.
- **-p 8081:8080** : Mappe le port 8081 de l'hôte vers le port 8080 du conteneur, permettant l'accès à l'application via `http://localhost:8081`.
- **talbihafsa/my_backend** : Spécifie l'image Docker à utiliser pour créer le conteneur.

e. Inspecter ce conteneur du backend.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker inspect backend-container
```

- **inspect** : Affiche les métadonnées détaillées du conteneur backend-container.

f. Afficher les logs liés à ce conteneur du backend.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker logs backend-container
```

- **logs** : Affiche les journaux de sortie du conteneur spécifié

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

a. Créer un Dockerfile.

```
FROM node:23.4-alpine3.21 AS builder
WORKDIR /app

COPY ["package.json", "package-lock.json", "./"]
RUN npm cache clean --force

RUN npm install --legacy-peer-deps

COPY . .
RUN npm run build

FROM nginx:1.26.2-alpine3.20-slim AS production
ENV NODE_ENV=production

COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80
```

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

```
(Users\del1\MyContainer\CONTproject\frontend\project> docker build -t my_frontend .
```

c. Scanner l'image des vulnérabilités qu'elle peut contenir

```
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker scout quickview my_frontend
```

Target	local://talbihafsa/my_frontend:1.0.0	0C	0H	0M	0L
digest	606961c8da4b				
Base image	nginx:1.26-alpine-slim	0C	0H	0M	0L
Updated base image	nginx:1.27-alpine-slim	0C	0H	0M	0L

d. Publier cette image dans Docker Hub.

```
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker tag my_frontend talbihafsa/my_frontend
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker push talbihafsa/my_frontend
```

e. Instancier l'image

```
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker run -d --name frontend-container --network my_network -p 3000:80 talbihafsa/my_frontend
2b9cdd41ba8e3bd962ffdda0001f107f0970549d11029c369ae2a13eb741a9fe
```

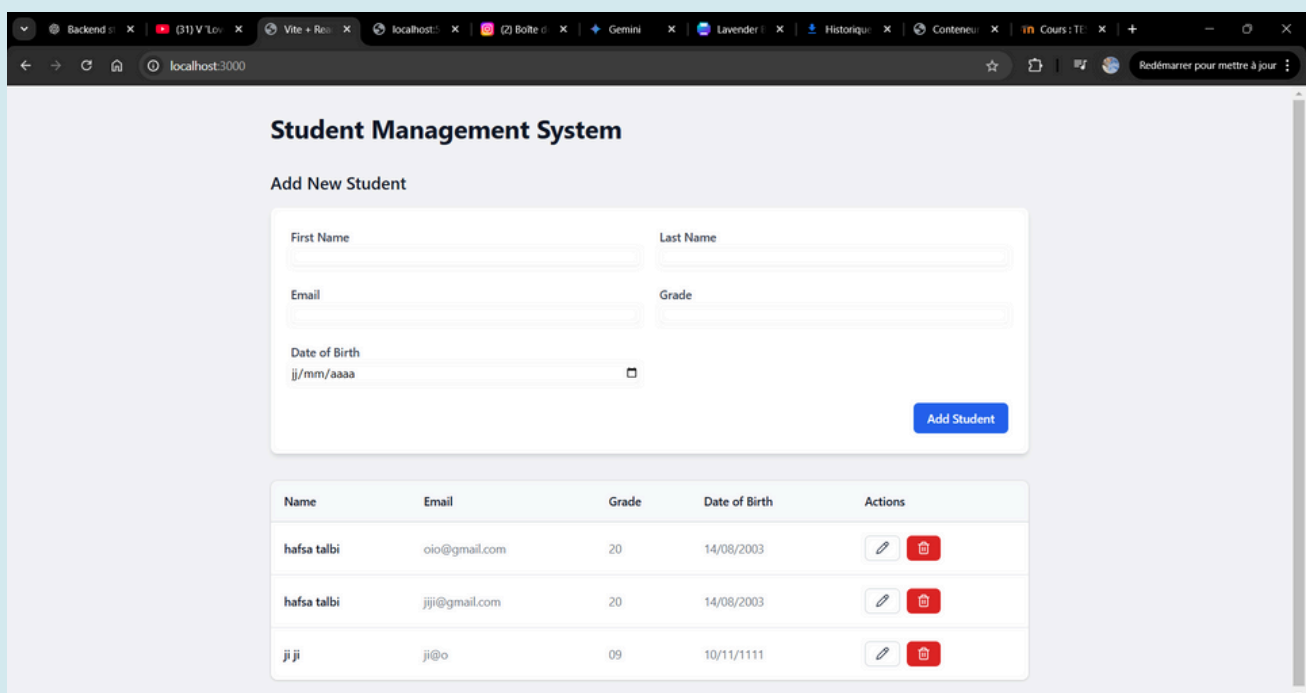
f. Inspecter le conteneur.

```
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker inspect frontend-container
```

g. Afficher les logs

```
PS C:\Users\del1\MyContainer\CONTproject\frontend\project> docker logs frontend-container
```

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



8. Utiliser la commande nécessaire pour supprimer les 3 conteneurs en exécution.

```
PS C:\Users\dell\MyContainer\CONTproject\backend> docker stop backend-container mysql-container frontend-container
backend-container
mysql-container
frontend-container

PS C:\Users\dell\MyContainer\CONTproject\backend> docker rm backend-container mysql-container frontend-container
backend-container
mysql-container
frontend-container
```

9. a. Créer le fichier docker-compose.

```
services:
  mysql:
    image: mysql:8.0.40-debian
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: TALta12024**
      MYSQL_DATABASE: studentdb
    networks:
      - my_network
    volumes:
      - mysql_data:/var/lib/mysql
  backend:
    image: talbihafsa/my_backend:1.0.0
    ports:
      - "5000:5000"
    networks:
      - my_network
  frontend:
    image: talbihafsa/my_frontend:1.0.0
    ports:
      - "3000:80"
    networks:
      - my_network
networks:
  my_network:
    driver: bridge
volumes:
  mysql_data:
```

b. Taper la commande docker permettant de l'exécuter

```
PS C:\Users\dell\MyContainer\CONTproject> docker-compose up -d
[+] Running 4/4
✓ Network contproject_my_network    Created
✓ Container contproject-backend-1   Started
✓ Container mysql-container         Started
✓ Container contproject-frontend-1  Started
```

c. S'assurer que l'application fonctionne correctement

Name	Email	Grade	Date of Birth	Actions
hafsa talbi	o0io@gmail.com	20	14/08/2003	
hafsa talbi	hafsatlb@gmail.com	9	04/12/2024	

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

```
docker rm mysql-container contproject-frontend-1 contproject-backend-1
```

```
docker rmi talbihafsa/my_frontend:1.0.0 talbihafsa/my_backend:1.0.0 mysql:8.0.40-debian
```

11. Créer un registre Docker privé.

a. Lancer le registre privé Docker

Voici la configuration Docker Compose pour lancer un registre privé Docker accessible localement. Il permet de stocker, gérer et supprimer des images Docker en toute sécurité avec un support de contrôle des accès via CORS.

```
registry:
  image: registry:2
  container_name: local-registry
  ports:
    - "7000:5000"
  restart: always
  environment:
    REGISTRY_STORAGE_DELETE_ENABLED: "true" # Allow image deletion
    REGISTRY_HTTP_HEADERS_Access-Control-Allow-Origin: "['http://localhost:8000']"
    REGISTRY_HTTP_HEADERS_Access-Control-Allow-Credentials: "[true]"
    REGISTRY_HTTP_HEADERS_Access-Control-Allow-Headers: "['Authorization', 'Accept']"
    REGISTRY_HTTP_HEADERS_Access-Control-Allow-Methods: "['HEAD', 'GET', 'OPTIONS']"
  volumes:
    - registry-data:/var/lib/registry
```

a. Stocker les images

```
tag talbihafsa/my_backend:1.0.0 localhost:7000/my_backend:1.0.0
tag talbihafsa/my_frontend:1.0.0 localhost:7000/my_frontend:1.0.0
```

```

PS C:\Users\dell\MyContainer\CONTproject> docker push localhost:7000/my_frontend:1.0.0
The push refers to repository [localhost:7000/my_frontend]
9301b75a59e7: Pushed
f69e73dd210e: Pushed
6715a1066dac: Pushed
ede6cd11b305: Pushed
a19f1e837fdf: Pushed
068b4536fb82: Pushed
da9db072f522: Mounted from my_backend
07b39cba6ee7: Pushed
1.0.0: digest: sha256:07f5661de1194c6681dc05cf5b49864893c5720124ba546be1048180e86e0a7d size: 1805

Info -> Not all multiplatform-content is present and only the available single-platform image was pushed
sha256:dccc88749b6ddb07cdf0f1e63069c93f1e29311ed00e4ffe08bd8da7e3bb07 -> sha256:07f5661de1194c6681dc05cf5b49864893c5720124ba546be1048180e86e0a7d

PS C:\Users\dell\MyContainer\CONTproject> docker tag localhost:7000/my_frontend:1.0.0 localhost:7000/my_backend:1.0.0
The push refers to repository [localhost:7000/my_backend]
417fc8c2adaa: Pushed
c78fe47fe453: Pushed
da9db072f522: Pushed
1594e406d9a0: Pushed
7fb22fe02bfc: Pushed
031dc3da24c3: Pushed
ecd0a3cbaf37: Pushed
1.0.0: digest: sha256:7cb8f81520ed75bf888e9b0ff626934f268daa3fd2e2dc8a93e42b472a1e933 size: 1624

Info -> Not all multiplatform-content is present and only the available single-platform image was pushed
sha256:64160b52c88efef6d0a64749c263d0b435ad0c3d2d44f3c7a9f2572cef44dd5f -> sha256:7cb8f81520ed75bf888e9b0ff626934f268daa3fd2e2dc8a93e42b472a1e933

```

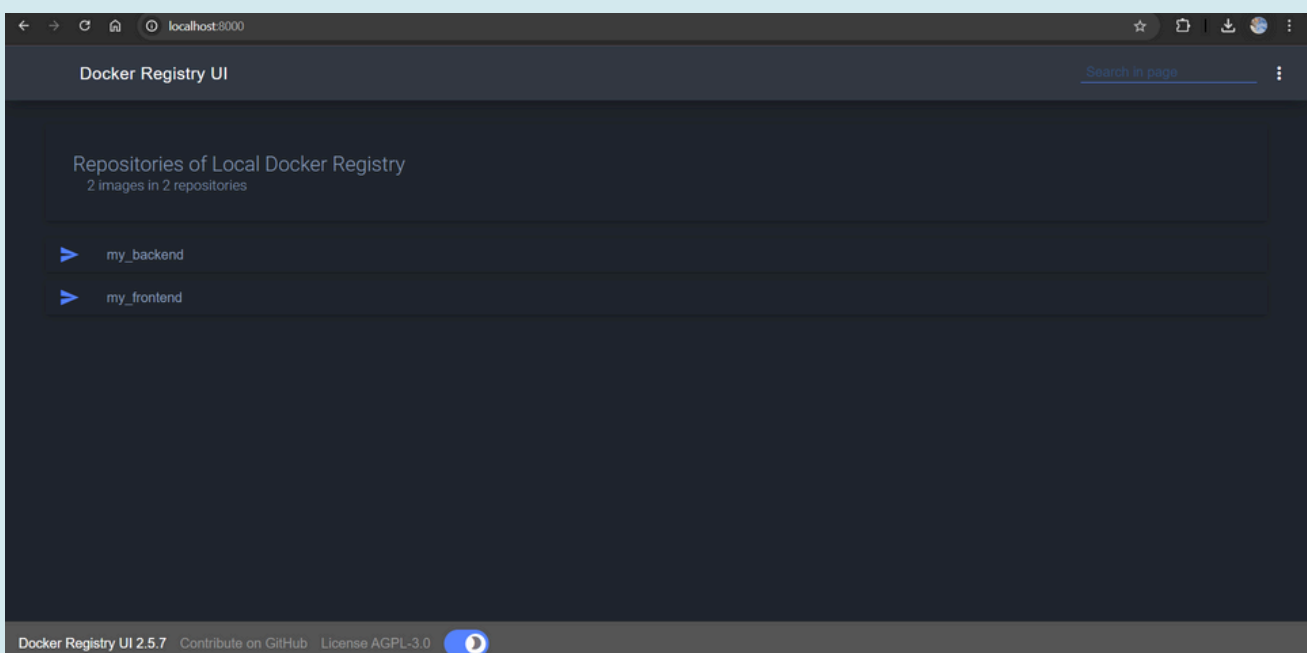
b. Visualiser via une UI les images qu'il contient.

Déploiement d'une interface utilisateur (registry-ui) connectée au registre privé Docker. Cela permet aux utilisateurs de visualiser, gérer et supprimer des images dans le registre via une interface web accessible localement.

```

registry-ui:
  image: joxit/docker-registry-ui:latest
  container_name: local-registry-ui
  ports:
    - "8000:80"
  restart: always
  environment:
    - REGISTRY_TITLE=Local Docker Registry
    - REGISTRY_URL=http://localhost:7000
    - DELETE_IMAGES=true
  depends_on:
    - registry

```



12.Redéployer l'application

Modification du fichier docker-compose afin d'accéder au images en utilisant le registre prive

```
services:
  mysql:
    image: mysql:8.0.40-debian
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: TALta12024**
      MYSQL_DATABASE: studentdb
    networks:
      - my_network
    volumes:
      - mysql_data:/var/lib/mysql
  backend:
    image: localhost:7000/my_backend:1.0.0
    ports:
      - "5000:5000"
    networks:
      - my_network
  frontend:
    image: localhost:7000/my_frontend:1.0.0
    ports:
      - "3000:80"
    networks:
      - my_network
networks:
  my_network:
    driver: bridge
volumes:
  mysql_data:
```

Student Management System

Add New Student

First Name Last Name

Email Grade

Date of Birth

Name	Email	Grade	Date of Birth	Actions
hafsa talbi	hafsatlib@gmail.com	20	13/12/2024	

PARTIE 2

1.a. Définir des services pour chaque composant dans un fichier dockercompose.yml compatible avec Docker Swarm

```
version: "3.8"
services:
  travelo-database:
    image: mysql:8.0.40-debian
    environment:
      MYSQL_ROOT_PASSWORD: TALta12024**
      MYSQL_DATABASE: studentdb
    ports:
      - "3307:3306"
    networks:
      - travelo-network
    volumes:
      - db_data:/var/lib/mysql

  travelo-backend:
    image: localhost:7000/my_backend:1.0.0
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://travelo-database:3306/studentdb
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: TALta12024**
    ports:
      - "5000:5000"
    networks:
      - travelo-network
    depends_on:
      - travelo-database

  travelo-frontend:
    image: localhost:7000/my_frontend:1.0.0
    ports:
      - "3000:80"
    networks:
      - travelo-network
    depends_on:
      - travelo-backend

networks:
  travelo-network:
    driver: overlay
    attachable: true

volumes:
  db_data:
    driver: local
```

b. Assigner chaque service à un réseau commun

J'ai changé le type du réseau à overlay puisqu'il est spécifiquement conçu pour Docker Swarm et permet la communication entre services sur plusieurs nœuds avec des fonctionnalités avancées comme l'équilibrage de charge et la découverte de services.

2-a. Dans le fichier docker-compose.yml, spécifier replicas: 2 pour chaque service.

```
app-backend:
  image: localhost:7000/my_backend:1.0.0
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://app-database:3306/studentdb
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD:
  ports:
    - "5000:5000"
  deploy:
    replicas: 2
  networks:
    - app-network
  depends_on:
    - app-database

app-frontend:
  image: localhost:7000/my_frontend:1.0.0
  ports:
    - "3000:80"
  deploy:
    replicas: 2
  networks:
    - app-network
  depends_on:
    - app-backend
```

3. Limiter les ressources CPU et mémoire de chaque service :

```
version: "3.8"
services:
  app-database:
    image: mysql:8.0.40-debian
    environment:
      MYSQL_ROOT_PASSWORD:
      MYSQL_DATABASE: studentdb
    ports:
      - "3307:3306"
    deploy:
      replicas: 1
    resources:
      limits:
        cpus: "0.5"
        memory: 512M
    networks:
      - app-network
    volumes:
      - db_data:/var/lib/mysql

  app-backend:
    image: localhost:7000/my_backend:1.0.0
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://app-database:3306/studentdb
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: TALta12024**
    ports:
      - "5000:5000"
    deploy:
      replicas: 2
    resources:
      limits:
        cpus: "1.0"
        memory: 1G
    networks:
      - app-network
    depends_on:
      - app-database

  app-frontend:
    image: localhost:7000/my_frontend:1.0.0
    ports:
      - "3000:80"
    deploy:
      replicas: 2
    resources:
      limits:
        cpus: "0.5"
        memory: 512M
    networks:
      - app-network
    depends_on:
      - app-backend
```

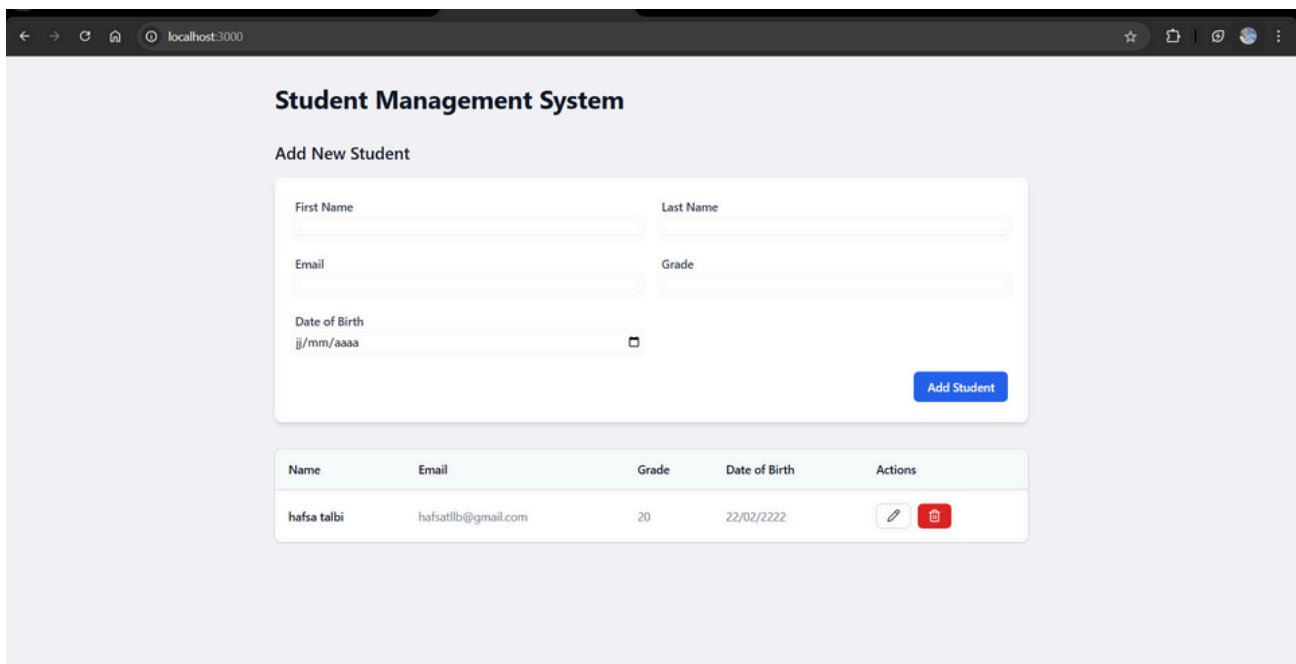
4. Configurer un Restart Policy pour chaque service :

```
restart_policy:
  condition: on-failure
  delay: 5s
  max_attempts: 3
networks:
  - app-network
volumes:
  - db_data:/var/lib/mysql

  app-backend:
    image: localhost:7000/my_backend:1.0.0
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://app-database:3306/studentdb
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD:
    ports:
      - "5000:5000"
    deploy:
      replicas: 2
    resources:
      limits:
        cpus: "1.0"
        memory: 1G
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
    networks:
      - app-network
    depends_on:
      - app-database

  app-frontend:
    image: localhost:7000/my_frontend:1.0.0
    ports:
      - "3000:80"
    deploy:
      replicas: 2
    resources:
      limits:
        cpus: "0.5"
        memory: 512M
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
```

6. Tester et Vérifier l'Application :



Student Management System

Add New Student

First Name

Last Name

Email

Grade

Date of Birth

Name	Email	Grade	Date of Birth	Actions
hafsa talbi	hafsatlbi@gmail.com	20	22/02/2222	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

5. Configurer un accès avec un Reverse Proxy :

```
nginx-proxy:
  image: nginx:latest
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/conf.d:/etc/nginx/conf.d
  networks:
    - app-network
  depends_on:
    - app-frontend
    - app-backend
    - app-database
```

La configuration Docker Compose définit un service Nginx utilisé comme proxy ou serveur web. Il est connecté à un réseau Docker et dépend de plusieurs services (frontend, backend et base de données) pour fonctionner correctement

```
server {
    listen 80;
    server_name frontend.local;

    location / {
        proxy_pass http://app-frontend:80;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

server {
    listen 80;
    server_name backend.local;

    location / {
        proxy_pass http://app-backend:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

configuration des nouveaux url

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping(pathPattern: "/api/**")
            .allowedOrigins(...origins: "http://frontend.local/") // Modify
            .allowedMethods(...methods: "GET", "POST", "PUT", "DELETE")
            .allowedHeaders(...headers: "*");
    }
}
```

```
API_URL=http://backend.local/api/
```

← → 🔍 Non sécurisé frontend.local

Student Management System

Add New Student

First Name



Last Name

Email

Grade

Date of Birth
jj/mm/aaaa

Add Student

Name	Email	Grade	Date of Birth	Actions
hafsa talbi	hafsatlbi@gmail.com	3	28/11/2024	 

PARTIE 3

1. Création des fichier .yaml

backend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: exam
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: mon_backend2:1.0.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5000
          envFrom:
            - configMapRef:
                name: app-config
            - secretRef:
                name: mysql-secrets
          resources:
            limits:
              cpu: "1000m"
              memory: "1Gi"
```

frontend-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: exam
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: mon_frontend4:1.0.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
          resources:
            limits:
              cpu: "500m"
              memory: "512Mi"
```

configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: exam
data:
  SPRING_DATASOURCE_URL: jdbc:mysql://mysql-service:3306/studentdb?allowPublicKeyRetrieval=true&useSSL=false
  SPRING_DATASOURCE_USERNAME: root
```

namespace.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: exam
```

secrets.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secrets
  namespace: exam
type: Opaque
stringData:
  MYSQL_ROOT_PASSWORD:
```

services.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: exam
spec:
  selector:
    app: mysql
  ports:
    - port: 3306
      targetPort: 3306
  clusterIP: None

---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: exam
spec:
  selector:
    app: backend
  ports:
    - port: 5000
      targetPort: 5000
  type: ClusterIP

---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: exam
spec:
  selector:
    app: frontend
  ports:
    - port: 80
      targetPort: 80
  type: NodePort
```

mysql-statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  namespace: exam
spec:
  serviceName: mysql-service
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:8.0.40-debian
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secrets
                  key: MYSQL_ROOT_PASSWORD
            - name: MYSQL_DATABASE
              value: studentdb
          resources:
            limits:
              cpu: "500m"
              memory: "512Mi"
            volumeMounts:
              - name: mysql-data
                mountPath: /var/lib/mysql
      volumeClaimTemplates:
        - metadata:
            name: mysql-data
          spec:
            accessModes: ["ReadWriteOnce"]
            resources:
              requests:
                storage: 1Gi
            storageClassName: standard
```

démarrer un cluster Kubernetes local en utilisant Minikube

```
minikube start
```

charger les images dans l'environnement Minikube

```
minikube image load mon_frontend4:1.0.0
```

```
minikube image load mon_backend2:1.0.0
```

appliquer des configurations

```
kubectl apply -f namespace.yaml
kubectl apply -f secrets.yaml
kubectl apply -f configmap.yaml
kubectl apply -f mysql-statefulset.yaml
kubectl apply -f backend-deployment.yaml
kubectl apply -f frontend-deployment.yaml
kubectl apply -f services.yaml
```

récupérer et afficher toutes les ressources dans le namespace exam

```
PS C:\Users\dell\MyContainer\CONTproject> kubectl get all -n exam
NAME                                READY   STATUS    RESTARTS   AGE
pod/backend-d7f66478d-68m8c         1/1     Running   0           23m
pod/backend-d7f66478d-gr7rr         1/1     Running   0           23m
pod/frontend-7f94bd5c58-lbqg5       1/1     Running   0           14m
pod/frontend-7f94bd5c58-rk6rb       1/1     Running   0           14m
pod/mysql-0                          1/1     Running   0           158m

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/backend-service             ClusterIP     10.111.67.73 <none>        5000/TCP         154m
service/frontend-service            NodePort      10.110.25.146 <none>        80:30135/TCP     154m
service/mysql-service               ClusterIP     None          <none>        3306/TCP         154m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend             2/2     2             2           23m
deployment.apps/frontend            2/2     2             2           14m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-d7f66478d   2         2         2       23m
replicaset.apps/frontend-7f94bd5c58 2         2         2       14m

NAME                                READY   AGE
statefulset.apps/mysql              1/1     160m
```

obtenir l'URL d'accès aux services frontend et backend

```
PS C:\Users\dell\MyContainer\CONTproject\kuber2> minikube service frontend-service -n exam --url
http://127.0.0.1:58863
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

```
PS C:\Users\dell\MyContainer\CONTproject\kuber2> minikube service backend-service -n exam --url
! service exam/backend-service has no node port
! Services [exam/backend-service] have type "ClusterIP" not meant to be exposed, however for local
http://127.0.0.1:58516
```

vérification :

Student Management System

Add New Student

First Name

Last Name

Email

Grade

Date of Birth

Name	Email	Grade	Date of Birth	Actions
John Doe	john.doe@example.com	A	01/01/1990	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

2.Définir des quotas pour la consommation des ressources Mémoire et CPU

D'abord, créons un ResourceQuota pour le namespace :

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: exam-quota
  namespace: exam
spec:
  hard:
    limits.cpu: "4"
    limits.memory: "4Gi"
    requests.cpu: "2"
    requests.memory: "2Gi"
```

définissons un LimitRange pour spécifier les limites par défaut pour chaque pod :

```
apiVersion: v1
kind: LimitRange
metadata:
  name: exam-limits
  namespace: exam
spec:
  limits:
  - default:
      cpu: "500m"
      memory: "512Mi"
    defaultRequest:
      cpu: "200m"
      memory: "256Mi"
    max:
      cpu: "1"
      memory: "1Gi"
    min:
      cpu: "100m"
      memory: "128Mi"
  type: Container
```

```
PS C:\Users\dell\MyContainer\CONTproject\kuber2> kubectl apply -f limitrange.yaml
limitrange/exam-limits created
PS C:\Users\dell\MyContainer\CONTproject\kuber2> kubectl apply -f quota.yaml
resourcequota/exam-quota created
```

3. Contrôler l'accès aux ressources existantes dans le namespace « exam » par un rôle RBAC.

D'abord, créons un ServiceAccount :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: exam-user
  namespace: exam
```

Ensuite, créons un Role avec des permissions spécifiques :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: exam-role
  namespace: exam
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "update"]
```

créons un RoleBinding pour lier le Role au ServiceAccount :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: exam-rolebinding
  namespace: exam
subjects:
- kind: ServiceAccount
  name: exam-user
  namespace: exam
roleRef:
  kind: Role
  name: exam-role
  apiGroup: rbac.authorization.k8s.io
```

Obtenir le token du ServiceAccount

```
PS C:\Users\delly\MyContainer\CD\project\kuber2> kubectl create token exam-user -n exam
```

Tester les permissions

```
PS C:\Users\delly\MyContainer\CONTproject\kuber2> kubectl auth can-i get pods --as=system:serviceaccount:exam:exam-user -n exam
yes
PS C:\Users\delly\MyContainer\CONTproject\kuber2> kubectl auth can-i list services --as=system:serviceaccount:exam:exam-user -n exam
```

Ces configurations :

- Permettent la lecture (get, list, watch) des pods et services

Permettent la lecture des ingresses

Permettent la lecture et la mise à jour des deployments

Limitent ces permissions au namespace "exam"

4. Spécifier un budget de perturbation (Disruption Budget) pour votre application.

Le Disruption Budget est une politique qui définit combien de Pods (instances de votre application) peuvent être indisponibles pendant une mise à jour ou un incident. Cela permet de garantir l'application reste disponible même en cas de perturbations

pdb-mysql.yaml

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: mysql-pdb
  namespace: exam
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: mysql
```

pdb-front.yaml

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: frontend-pdb
  namespace: exam
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: frontend
```

pdb-back.yaml

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: backend-pdb
  namespace: exam
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: backend
```

5. Créer des probs

Liveness Prob

```
containers:
  - name: frontend
    image: mon_frontend5:1.0.0
    imagePullPolicy: IfNotPresent
    ports:
      - containerPort: 80
    livenessProbe:
      httpGet:
        path: /health
        port: 80
      initialDelaySeconds: 30
      periodSeconds: 10
```

Readiness Prob, Startup Prob

```
containers:
- name: backend
  image: mon_backend2:1.0.0
  imagePullPolicy: IfNotPresent
  ports:
  - containerPort: 5000
  readinessProbe:
    httpGet:
      path: /ready
      port: 5000
    initialDelaySeconds: 20
    periodSeconds: 5
  startupProbe:
    httpGet:
      path: /startup
      port: 5000
    initialDelaySeconds: 40
    periodSeconds: 10
```

6. Créer un Ingress pour accéder à l'application depuis un navigateur moyennant un nom de domaine

installation ingress

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml
m1 C:\Users\dell\MyContainer\CONTproject\kuber2>
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
```

back-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: backend-ingress
  namespace: exam
spec:
  ingressClassName: nginx
  rules:
  - host: talbi.backend.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 5000
```

front-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-ingress
  namespace: exam
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - host: minikube.talbi.frontend.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

6. Créer un Ingress pour accéder à l'application depuis un navigateur moyennant un nom de domaine

```
PS C:\Users\dell\MyContainer\CONTproject\kuber2> minikube tunnel  
✓ Tunnel successfully started
```

Student Management System

Add New Student

First Name

Last Name

Email

Grade

Date of Birth

Name	Email	Grade	Date of Birth	Actions
John Doe	john.doe@example.com	A	01/01/1990	<input type="button" value="Edit"/> <input type="button" value="Delete"/>