MGL7760 - Qualité et Productivité des outils logiciels
Hiver 2023

Projet partie 1 - **Environnement de développement avec Docker**

Groupe 4

Présenté par :
Ahmat Acheik,Al Aaguid,
Oumar Marame NDIONE

Professeur : **Moustapha Boulgoudan**

UQÀM

Université du Québec à Montréal

Février 2023

Table des matières

1 Introduction

Dans ce présent rapport on a un projet qui consiste à développer une application Web de gestion d'une bibliothèque personnelle en utilisant le langage de programmation Python 3, le micro-framework Flask et le toolkit/ORM SQLAlchemy. Notre objectif principal est de nous familiariser avec les outils de développement modernes afin comprendre l'importance des enjeux reliés à la qualité et la productivité engendrée par l'utilisation d'outils logiciels modernes. Notamment les conteneurs Docker, l'environnement de développement intégré Visual Studio Code, les dépôts de code Git et Github. L'application sera hébergée dans des conteneurs Docker, et l'environnement de développement local sera créé à l'aide de Docker Compose. Le projet se déroulera en quatre étapes, allant de la mise en place d'un serveur web et d'un serveur applicatif jusqu'à l'environnement de développement final avec un serveur de répartition de charge, trois serveurs applicatifs, une base de données et un serveur de cache mémoire. Le travail à réaliser comprend la création d'un dépôt de code source privé sur Github, le développement de l'application Web, la création d'un script Shell pour gérer les conteneurs et l'importation de nouveaux livres dans la base de données. Enfin, on présentera l'architecture logicielle de l'application développée.

2 Démarche pour la création de l'environnement de développemnt avec conteneurs

2.1 Étape 1 :

2.1.1 Étape 1 : 1 serveur web + 1 serveur applicatif

Cette première étape consistait à afficher un simple "Hello World !" à la racine de l'application.
Pour cela, nous avons créé deux conteneurs Docker : un conteneur **web** pour le serveur web Nginx et un autre conteneur **app** pour le serveur applicatif Flask.

Pour cela on a créer un fichier docker-compose.yml à la racine du projet contenant les configurations pour deux conteneurs, puis configurer le serveur web pour rediriger les requêtes HTTP vers le serveur applicatif.
Configurer le serveur applicatif pour afficher un "Hello World !" à la racine de l'application.
Tester l'application en exécutant "docker-compose up" et en accédant à l'URL localhost :8000

```
version: '3.9'
services:

  web:
    image: nginx:latest
    container_name: Nginx
    restart: 'on-failure'
    # working_dir: "/var/www"
    volumes:
      - ./web:/usr/share/nginx/html
      #- ./:/var/www
      #- ./default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "90:80"
    networks:
      - 'biblio'
    depends_on:
      - app

  app:
```

```
    image: tiangolo/meinheld-gunicorn:latest
    container_name: WSGI
    #command: gunicorn --bind 0.0.0.0:8000 --workers 3 "app.create_app:create_app()"
    #restart: 'on-failure'
    build:
      context: .
      dockerfile: ./Dockerfile
    ports:
      - '8000:5000'
    volumes:
      - .:/var/www/html
    environment:
      FLASK_DEBUG: "true"
    networks:
      - 'biblio'
    depends_on:
      - cache
      - db
networks:
  biblio:
    driver: bridge
```

Figure 1 : Le docker-compose.yml avec **web + app**

Dans cette étape, on affiche simplement un Hello World ! à la racine de notre application.

2.2 Étape 2 :

2.2.1 Étape 2 : 1 serveur web + 1 serveur applicatif + 1 base de données

Dans cette seconde étape, nous avons ajouté un base de données **biblio** pour permettre l'accès à la base de données via l'application.
Il est génère par un fichier **bd.sql**, dont on fait appel au niveau du volume :

```
  volumes:
    - './db:/docker-entrypoint-initdb.d'
```

Pour cela, nous avons créé un troisième conteneur Docker pour la base de données
avec l'O.R.M **SQLAlchemmy** une boîte à outils SQL Python et le mappage relationnel des objets.
Cela se fait en ajoutant une configuration pour le conteneur de la base de données dans le fichier docker-compose.yml.
Ensuite on configure l'application pour se connecter à la base de données en utilisant SQLAlchemy.
Puis on peut le tester l'application en exécutant "docker-compose up" et en vérifiant que les données sont correctement affichées.

```
version: '3.9'
services:

  web:
  #nginx:
    image: nginx:latest
    container_name: Nginx
```

```
8       restart: 'on-failure'
9       # working_dir: "/var/www"
10      volumes:
11        - ./web:/usr/share/nginx/html
12        #- ./:/var/www
13        #- ./default.conf:/etc/nginx/conf.d/default.conf
14      ports:
15        - "90:80"
16      networks:
17        - 'biblio'
18      depends_on:
19        - app
20        # - db
21
22    app:
23      image: tiangolo/meinheld-gunicorn:latest
24      container_name: WSGI
25      #command: gunicorn --bind 0.0.0.0:8000 --workers 3 "app.create_app:create_app()"
26      #restart: 'on-failure'
27      build:
28        context: .
29        dockerfile: ./Dockerfile
30      ports:
31        - '8000:5000'
32      volumes:
33        - .:/var/www/html
34      environment:
35        FLASK_DEBUG: "true"
36      networks:
37        - 'biblio'
38      depends_on:
39        - db
40
41    db:
42      image: mysql:latest
43      container_name: MySQL
44      #restart: unless-stopped
45      restart: 'on-failure'
46      ports:
47        - '3306:3306' #si on ne le fixe pas il met des port al oratoire    chaque
        d marage
48      # expose:
49      #   - "3306"
50      # env_file:
51      #  - .env
52      volumes:
53        - './db:/docker-entrypoint-initdb.d'
54        #- mysql-data:/var/lib/mysql
55      environment:
56        MYSQL_DATABASE: biblio #ceci vas   tre   la bd vide cr e
57        MYSQL_ROOT_PASSWORD: rootpw
58        MYSQL_USER: db
59        MYSQL_PASSWORD: dbpw
60        # MYSQL_DATABASE: ${MYSQL_DATABASE}
61        # MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
62        # MYSQL_PASSWORD: ${MYSQL_PASSWORD}
63        # MYSQL_USER: ${MYSQL_USER}
64        # SERVICE_TAGS: dev
65        # SERVICE_NAME: mysql
66      networks:
67        - 'biblio'
```

Figure 2 : Le docker-compose.yml avec **web + app+ bd**

Dans cette étape, on vérifie que l'on peut accéder à la base de données via notre application.

2.3 Étape 3 :

2.3.1 Étape 3 : 1 serveur web + 1 serveur applicatif + 1 base de données + 1 serveur de cache mémoire

Cette étape consistait à ajouter un serveur de cache mémoire pour améliorer les performances de l'application. Pour cela, nous avons créé un quatrième conteneur **cache** pour le serveur de cache Redis dans le fichier docker-compose.yml.
Ensuite de configurer l'application pour utiliser le serveur de cache en utilisant Flask-Caching.
Enfin tester l'application en exécutant "docker-compose up" et en vérifiant que les données sont correctement mises en cache.

```
version: '3.9'
services:

  web:
  #nginx:
    image: nginx:latest
    container_name: Nginx
    restart: 'on-failure'
    # working_dir: "/var/www"
    volumes:
      - ./web:/usr/share/nginx/html
      #- ./:/var/www
      #- ./default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "90:80"
    networks:
      - 'biblio'
    depends_on:
      - app
      # - db

  app:
    image: tiangolo/meinheld-gunicorn:latest
    container_name: WSGI
    #command: gunicorn --bind 0.0.0.0:8000 --workers 3 "app.create_app:create_app()"
    #restart: 'on-failure'
    build:
      context: .
      dockerfile: ./Dockerfile
    ports:
      - '8000:5000'
    volumes:
      - .:/var/www/html
    environment:
      FLASK_DEBUG: "true"
    networks:
      - 'biblio'
    depends_on:
      - cache
      - db

  db:
    image: mysql:latest
```

```
44      container_name: MySQL
45      #restart: unless-stopped
46      restart: 'on-failure'
47      ports:
48        - '3306:3306' #si on ne le fixe pas il met des port al oratoire    chaque
        d marage
49      # expose:
50      #   - "3306"
51      # env_file:
52      #  - .env
53      volumes:
54        - './db:/docker-entrypoint-initdb.d'
55        #- mysql-data:/var/lib/mysql
56      environment:
57        MYSQL_DATABASE: biblio #ceci vas  tre  la bd vide cr e
58        MYSQL_ROOT_PASSWORD: rootpw
59        MYSQL_USER: db
60        MYSQL_PASSWORD: dbpw
61        # MYSQL_DATABASE: ${MYSQL_DATABASE}
62        # MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
63        # MYSQL_PASSWORD: ${MYSQL_PASSWORD}
64        # MYSQL_USER: ${MYSQL_USER}
65        # SERVICE_TAGS: dev
66        # SERVICE_NAME: mysql
67      networks:
68        - 'biblio'
69
70    cache:
71      image: redis:latest
72      container_name: Redis
73      # volumes:
74      #   - redis-data:/data
75      restart: 'on-failure'
76      ports:
77        - 6379
78      #  - "6379:6379"
79      networks:
80        - biblio
81
82    # volumes:
83    #   #mysql-data:
84    #   redis-data:
85  networks:
86    biblio:
87      driver: bridge
```

Figure 3 : Le docker-compose.yml avec **web + app+ bd + cache**

Dans cette étape, on est censé vérifier que l'on peut accéder à la base de données et à la mémoire cache via notre application.

2.4 Étape 4 :

2.4.1 Étape 4 : 1 serveur web / répartiteur de charge + 3 serveurs applicatifs + 1 base de données + 1 serveur de cache mémoire

Cette étape correspond à l'environnement de développement final de notre application pour compléter le projet.

```
#command: gunicorn --bind 0.0.0.0:8000 --workers 3 "app.create_app:
 create_app()"
```

Dans cette dernière étape, on est censé ajouter un serveur web / répartiteur de charge pour équilibrer la charge entre les différents serveurs applicatifs et améliorer les performances de l'application. Nous devons également créé deux autres conteneurs Docker pour les serveurs applicatifs supplémentaires.

1 serveur web / répartiteur de charge + 3 serveurs applicatifs + 1 base de données + 1 serveur de cache mémoire.

On doit aller modifier la configuration du fichier docker-compose.yml pour ajouter trois conteneurs pour le serveur applicatif et un conteneur pour le serveur web / répartiteur de charge (HAProxy). Configurer le serveur web / répartiteur de charge pour répartir la charge entre les serveurs applicatifs en utilisant HA-Proxy. `https://hub.docker.com/_/haproxy`
Et tester l'application en exécutant "docker-compose up" et en vérifiant que la charge est correctement répartie entre les serveurs applicatifs.

La démarche pour la création de l'environnement de développement avec conteneurs pour les quatre étapes mentionnées consiste à créer un fichier docker-compose.yml contenant les configurations pour chaque conteneur nécessaire (serveur web, serveur applicatif, base de données, serveur de cache, etc.), configurer l'application pour utiliser ces conteneurs et tester l'application en exécutant "docker-compose up". Il est également important de développer un script Shell gerer-conteneurs.sh qui permet de gérer facilement les conteneurs (créer, importer des données, supprimer, démarrer, arrêter).

3 Explication du fonctionnement du script Shell gerer-conteneurs.sh

3.1 script Shell

Le script Shell **gerer-conteneurs.sh** permet de gérer l'environnement de développement avec des conteneurs Docker en effectuant différentes actions.
Voici une explication détaillée de chaque fonctionnalité :
1. Créer l'environnement de développement avec tous les conteneurs à partir de votre dépôt de code source et d'une base de données initiale : Cette fonctionnalité permet de créer les conteneurs nécessaires pour l'exécution de l'application et de la base de données. Elle utilise Docker Compose pour créer les différents conteneurs en se basant sur la configuration définie dans le fichier docker-compose.yml.

2. Importer de nouveaux livres, à partir d'un fichier CSV, dans la base de données : Cette fonctionnalité permet d'importer des données de livres à partir d'un fichier CSV dans la base de données. Elle utilise le langage Python pour lire le fichier CSV et insérer les données correspondantes dans la base de données.

3. Supprimer l'environnement de développement avec tous les conteneurs : Cette fonctionnalité permet de supprimer tous les conteneurs créés pour l'environnement de développement. Elle utilise Docker Compose pour arrêter et supprimer les conteneurs correspondants.

4. Démarrer des conteneurs : Cette fonctionnalité permet de démarrer des conteneurs spécifiques en se basant sur leur nom ou leur ID. Elle utilise la commande docker start pour démarrer les conteneurs.

5. Arrêter des conteneurs : Cette fonctionnalité permet d'arrêter des conteneurs spécifiques en se basant

sur leur nom ou leur ID. Elle utilise la commande docker stop pour arrêter les conteneurs.

• Le script contient des fonctions pour créer, importer et supprimer des conteneurs. La fonction "creer-conteneurs" crée l'environnement de développement en utilisant Docker Compose. Cette fonction commence par vérifier si Docker Compose est installé et disponible sur la machine. Si Docker Compose n'est pas disponible, le script télécharge et installe Docker Compose.
• La fonction "creer-conteneurs" crée ensuite les conteneurs en utilisant Docker Compose. Les conteneurs créés sont un serveur web, un serveur applicatif, une base de données et un serveur de cache mémoire. Cette fonction importe également des données initiales à partir d'un fichier CSV dans la base de données.
• La fonction "supprimer-conteneurs" supprime tous les conteneurs créés en utilisant Docker Compose.
• Le script contient également des fonctions pour démarrer et arrêter des conteneurs individuels. La fonction "demarrerconteneur" démarre un conteneur spécifique et la fonction "arreter-conteneur" arrête un conteneur spécifique.
• En fin de script, une fonction "menu" est utilisée pour afficher un menu de choix des options disponibles pour l'utilisateur.

Pour résumer, le script Shell gerer-conteneurs.sh permet de gérer l'environnement de développement avec des conteneurs Docker en facilitant la création, l'importation de données et la suppression des conteneurs. Il permet également de démarrer ou d'arrêter des conteneurs spécifiques en fonction des besoins de l'utilisateur. Ce dernier peut choisir une option en entrant le numéro correspondant à l'option. Selon l'option choisie, le script appelle la fonction correspondante pour effectuer l'action demandée.
Le script est conçu pour être facilement utilisable par les développeurs même sans connaissances avancées en Docker.

```
#!/bin/bash

# Variables pour les noms des conteneurs et images
web_container="web"
app_container="app"
db_container="db"
cache_container="cache"
image_name="biblio_app"

# Cr er l'environnement de d veloppement
function create_environment() {
  # Lancer tous les conteneurs avec Docker Compose
  docker-compose up -d
}

# Importer de nouveaux livres dans la base de donn es
function import_books() {
  # Importer les livres    partir d'un fichier CSV
  docker exec -it $db_container bash -c "command for importing books from csv"

  # V rifier que les livres ont   t   correctement import s
  docker exec -it $db_container bash -c "command to check if books were
    imported"
}

# Supprimer l'environnement de d veloppement
function delete_environment() {
```

```
  # Arrter et supprimer les conteneurs avec Docker Compose
  docker-compose down --volumes
  #docker-compose down -v
}

# Demarrer les conteneurs
function start_containers() {
  # Demarrer les conteneurs existants avec Docker Compose
  docker-compose start
}

# Arrter les conteneurs
function stop_containers() {
  # Arrter les conteneurs existants avec Docker Compose
  docker-compose stop
}

# Analyse des options
case $1 in
  "create") create_environment;;
  "import") import_books;;
  "delete") delete_environment;;
  "start") start_containers;;
  "stop") stop_containers;;
  *) echo "Usage: gerer-conteneurs.sh [create|import|delete|start|stop]"
esac
#if [ "$1" == "create" ]; then
#create_env
#elif [ "$1" == "import" ]; then
#import_books
#elif [ "$1" == "remove" ]; then
#remove_env
#elif [ "$1" == "start" ]; then
#start_containers
#elif [ "$1" == "stop" ]; then
#stop_containers
#else
#echo "Usage : gerer-conteneurs.sh {create|import|remove|start|stop}"
#fi
```

4 Explication de l'importance des enjeux reliés à la qualité et la productivité engendré par l'utilisation d'outils logiciels modernes.

Les enjeux reliés à l'utilisation d'outils logiciels modernes tels que Github, les conteneurs et Visual Studio Code, sont nombreux et importants, permetant d'augmenter la qualité et la productivité du développement logiciel. En effet, Github facilite la collaboration et la gestion des versions du code source, ce qui évite les conflits et les pertes de données. Les conteneurs permettent de créer des environnements de développement

isolés et reproductibles, ce qui garantit une meilleure qualité et une plus grande fiabilité des applications. Visual Studio Code est un IDE puissant qui facilite le développement et le débogage des applications, ce qui permet de gagner du temps et d'augmenter la productivité.

4.0.1 bénéfices d'utilisation de Github

- Github, favorise la collaboration et la gestion des versions du code source, ce qui évite les conflits et les pertes de données. Les développeurs peuvent travailler sur des branches différentes et fusionner leur code en toute sécurité. Les problèmes et les demandes d'extraction peuvent être gérés de manière centralisée, ce qui facilite la communication et la résolution des problèmes.

Cela permet d'améliorer la qualité du code en réduisant les conflits de versions, en permettant des revues de code efficaces et en favorisant la communication et la collaboration entre les développeurs.

4.0.2 bénéficies d'utilisation des conteneurs

- Docker permet de créer des environnements de développement isolés et reproductibles. Cela garantit que l'application fonctionne de manière cohérente sur tous les systèmes et permet de résoudre les problèmes plus rapidement. Les conteneurs peuvent être déployés de manière transparente sur différentes plateformes, ce qui facilite le déploiement de l'application, on est dans un environnement de développement reproductible et isolé.

En utilisant des conteneurs, il est possible de tester et de déployer l'application dans des environnements identiques, ce qui contribue à améliorer la qualité de l'application et à réduire les erreurs liées aux différences d'environnement.

4.0.3 bénéficies d'utilisation de Visual Studio Code

- Visual Studio Code permet de bénéficier d'un IDE puissant qui facilite le développement et le débogage des applications. Les développeurs peuvent utiliser des fonctionnalités telles que l'autocomplétion, la mise en évidence de la syntaxe et le débogage en temps réel pour travailler plus rapidement et plus efficacement.

Dans le cadre du projet, l'utilisation de ces outils modernes permettra de faciliter la mise en place d'un environnement de développement local avec des conteneurs Docker, d'améliorer la qualité et la productivité du développement en utilisant des outils de gestion de versions et de collaboration tels que Github, et de développer une application performante et fiable grâce à l'utilisation d'un environnement de développement intégré comme Visual Studio Code.

5 Explication de l'architecture logicielle de notre application

L'architecture logicielle de notre application de gestion de bibliothèque personnelle est basée sur le micro-framework Flask pour la partie serveur web et l'ORM SQLAlchemy pour la gestion de la base de données. L'application est développée en Python 3.

Il utilise une architecture de type Modèle-Vue-Contrôleur (MVC) où les modèles représentent les objets manipulés dans l'application (livres, catégories, auteurs, éditeurs), les vues représentent l'interface utilisateur et les contrôleurs gèrent les interactions entre les modèles et les vues.

Les modèles de données sont définis à l'aide de classes Python et sont mappés à des tables dans la base de données à l'aide de SQLAlchemy. Les vues sont des fonctions Python qui renvoient des templates HTML pour afficher les données. Enfin, les contrôleurs sont responsables de la logique métier de l'application et interagissent avec les modèles et les vues pour fournir une expérience utilisateur cohérente.

L'application est hébergée dans des conteneurs Docker. Les conteneurs sont gérés par Docker Compose qui permet de lancer plusieurs conteneurs simultanément et de les interconnecter. Les différents conteneurs utilisés sont :

- Un conteneur "web" pour le serveur web basé sur le serveur web Nginx.
- Un conteneur "app" pour le serveur applicatif Flask.
- Un conteneur "db" pour la base de données MySQL.
- Un conteneur "cache" pour le serveur de cache mémoire Redis.

Le serveur web Nginx agit comme un répartiteur de charge qui redirige les requêtes entrantes vers l'un des trois serveurs applicatifs Flask. Cela permet d'améliorer les performances et la disponibilité de l'application.

L'application utilise Git et Github pour la gestion de versions du code source. On a pu travailler en équipe sur le même code source en utilisant des branches et des pull requests.

Le script Shell gerer-conteneurs.sh permet de gérer l'environnement de développement avec des conteneurs Docker. Il permet de créer l'environnement avec tous les conteneurs à partir du code source et d'une base de données initiale, d'importer de nouveaux livres à partir d'un fichier CSV dans la base de données, de supprimer l'environnement et de démarrer/arrêter des conteneurs.

6 Conclusion

Ce projet nous a permis de nous familiariser avec certains outils de développement modernes en utilisant des conteneurs Docker. L'utilisation de Visual Studio Code, avec Git et GitHub ont permis de maîtriser l'utilisation de dépôts de code, le développement collaboratif et la préservation de l'historique de développement du code. La création de l'application Web de gestion de bibliothèque personnelle a été réalisée avec Python 3, le micro-framework Flask et le toolkit/ORM SQLAlchemy. L'environnement de développement a été mis en place avec des conteneurs Docker et l'outil Docker Compose a été utilisé pour créer les serveurs web, applicatifs, base de données et serveur de cache mémoire. Le script Shell gerer-conteneurs.sh a été développé pour gérer l'environnement de développement avec tous les conteneurs à partir du dépôt de code source et d'une base de données initiale, importer de nouveaux livres à partir d'un fichier CSV et supprimer l'environnement de développement avec tous les conteneurs. Ce projet a permis de comprendre l'importance des enjeux reliés à la qualité et la productivité engendrés par l'utilisation d'outils logiciels modernes.

7 Annexe / Glossaire

Installation des outils

Le processus d'installation des outils pour ce projet peut être divisé en plusieurs étapes, qui sont les suivantes :

- Installer Docker : Docker est une plateforme open source qui permet de créer, de déployer et de gérer des applications dans des conteneurs. Il est nécessaire de l'installer pour utiliser des conteneurs dans l'environnement de développement.

- Installer Visual Studio Code : C'est un environnement de développement intégré (IDE) qui prend en charge plusieurs langages de programmation. Il doit être installé avec les extensions adéquates pour Python, Git, Github, GitHub Pull Requests and Issues et Docker.

- Installer Git : C'est un système de gestion de version qui permet de suivre l'évolution du code source d'un projet. Il doit être installé pour gérer le dépôt de code source sur Github.

- Créer un compte Github : Github est un service d'hébergement de code source qui permet de collaborer sur des projets. Il est nécessaire de créer un compte pour héberger le dépôt de code source de l'application.

- Cloner le dépôt Git localement : Le dépôt de code source doit être cloné localement sur la machine de développement pour y travailler dessus.

- Installer Docker Compose : C'est un outil qui permet de définir et de lancer des applications Docker multi-conteneurs. Il doit être installé pour créer l'environnement de développement local avec des conteneurs Docker.

- Définir les fichiers de configuration de Docker Compose : Il faut créer les fichiers de configuration de Docker Compose pour définir les services, les images, les réseaux et les volumes nécessaires pour l'application.

- Construire les images Docker : Les images Docker doivent être construites en utilisant les fichiers de configuration de Docker Compose.

- Lancer les conteneurs Docker : Les conteneurs Docker peuvent être lancés en utilisant les commandes de Docker Compose. Les conteneurs sont créés en fonction des images construites précédemment.

- Développer l'application Web : L'application Web de gestion de bibliothèque personnelle doit être développée en utilisant le langage de programmation Python 3, le micro-framework Flask et le toolkit/ORM SQLAlchemy.

- Créer le script Shell gerer-conteneurs.sh : Le script Shell gerer-conteneurs.sh doit être développé pour permettre de créer, importer de nouveaux livres à partir d'un fichier CSV, supprimer l'environnement de développement et démarrer/arrêter des conteneurs.

En suivant ces étapes, il est possible d'installer tous les outils nécessaires pour mettre en place un environnement de développement local utilisant des conteneurs Docker et développer l'application Web de gestion de bibliothèque personnelle.

Glossaire

ORM (Object-Relational Mapping) : est une technique de programmation qui permet de représenter des données d'une base de données relationnelle sous forme d'objets dans un langage de programmation.

SQLAlchemy : est une bibliothèque de mapping objet-relationnel (ORM) pour Python qui permet d'interagir avec une base de données SQL de manière intuitive.

HTTP (Hypertext Transfer Protocol) : est un protocole de communication utilisé pour transférer des données sur le web. Il est principalement utilisé pour le transfert de pages web, mais il peut également être utilisé pour le transfert de tout autre type de données.

URL (Uniform Resource Locator) : est une adresse web qui permet d'identifier de manière unique une ressource sur internet, telle qu'une page web, une image, une vidéo, etc.

CSV (Comma-Separated Values) : est un format de fichier utilisé pour stocker des données sous forme de tableau. Les valeurs sont séparées par des virgules, et chaque ligne représente une entrée distincte dans le tableau.

ID (Identifiant) : est un numéro unique attribué à chaque objet ou entrée dans une base de données. Il permet de référencer de manière univoque chaque élément dans la base de données.

MVC (Modèle-Vue-Contrôleur) : est un modèle de conception logiciel qui permet de séparer les données, la présentation et la logique de traitement d'une application en trois composants distincts pour faciliter la maintenance et l'évolutivité de l'application.

8 Références

[1] : https://code.visualstudio.com/docs/sourcecontrol/overview
[2] : https://code.visualstudio.com/docs/python/tutorial-flask
[3] : https://code.visualstudio.com/docs/containers/overview
[4] : https://docs.docker.com/compose/gettingstarted/
[5] : https://www.digitalocean.com/community/tutorials/how-to-use-one-to-many-database-relationships-with-flask-sqlalchemy
[6] : https://www.digitalocean.com/community/tutorials/how-to-use-many-to-many-database-relationships-with-flask-sqlalchemy
[7] : https://ena01.uqam.ca/mod/resource/view.php?id=3403224
[8] : https://ena01.uqam.ca/pluginfile.php/6250691/mod_resource/content/0/01%20Ligne%20de%20commande.pdf
[9] : https://ena01.uqam.ca/pluginfile.php/6250692/mod_resource/content/0/02%20Scripts%20Shell.pdf
[10] : https://ena01.uqam.ca/pluginfile.php/6279112/mod_resource/content/0/03%20Les%20Conteneurs.pdf
[11] : https://ena01.uqam.ca/pluginfile.php/6280892/mod_resource/content/0/projet1.pdf
[12] : https://hub.docker.com/r/tiangolo/meinheld-gunicorn
[13] : https://www.baeldung.com/ops/docker-mysql-container
[14] : https://onexlab-io.medium.com/docker-compose-mysql-entry-point-fa6335346791
[15] : https://geshan.com.np/blog/2022/02/mysql-docker-compose/
[16] : https://openclassrooms.com/en/courses/2035766-optimisez-votre-deploiement-en-creant-des-conteneurs-avec-docker/6211677-creez-un-fichier-docker-compose-pour-orchestrer-vos-conteneurs
[17] : https://www.datanovia.com/en/fr/lessons/forcer-docker-compose-a-attendre-un-conteneur-en-utilisant-loutil-wait/docker-compose-attendre-que-le-conteneur-mysql-soit-pret/
[18] : https://www.youtube.com/playlist?list=PLkA60AVN3hh_YpK8t2ASZZJAEkz8fX2wj
[19] : https://www.youtube.com/watch?v=3Lg2ijbghR8&t=5s
[20] : https://www.youtube.com/watch?v=7F0bj9Vnjic&t=2s
[21] : https://www.youtube.com/watch?v=kphq2TsVRIs
[22] : https://www.youtube.com/watch?v=uZnp21fu8TQ

[22] : https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/