# QUESTION 1

Créez un article (POST /products)

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

2.Ajoutez 5 unités au stock de cet article (POST /stocks)

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Vérifiez le stock, votre article devra avoir 5 unités dans le stock (GET /stocks/:id)

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Faites une commande de 2 unités de l'article que vous avez créé (POST /orders)

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Vérifiez le stock encore une fois (GET /stocks/:id)

Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.

**Étape extra**: supprimez la commande et vérifiez le stock de nouveau. Le stock devrait augmenter après la suppression de la commande.

Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

**Réponse à la Question 1**

À la fin du test, mon produit que j’ai créé possède **5 unités en stock**. En effet, après avoir ajouté 5 unités, passé une commande de 2 unités puis supprimé la commande, le stock est revenu à 5.

Pour l’article existant avec **id = 2**, son stock est de **500 unités**.

📸 **Captures d’écran Postman :**

* Création du produit et ajout de stock  
  Une image contenant texte, capture d’écran, logiciel, nombre

  Le contenu généré par l’IA peut être incorrect.

Vérification du stock après commande et suppression  
Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Vérification du stock de l’article id=2  
Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

# QUESTION 2

Dans cette tâche, nous avons modifié la méthode get\_stock\_for\_all\_products afin d’enrichir le rapport de stock avec des informations supplémentaires provenant de la table **Product** (nom, SKU, prix).

L’utilisation de join dans SQLAlchemy permet d’associer deux tables sur une clé commune. Ici, la clé est Stock.product\_id, qui fait référence à Product.id.  
Cela correspond, en SQL, à une requête du type :

SELECT stock.product\_id, stock.quantity, product.name, product.sku, product.price

FROM stock

JOIN product ON stock.product\_id = product.id

ORDER BY stock.product\_id;

Avec SQLAlchemy, on a écrit l’équivalent Python suivant :

def get\_stock\_for\_all\_products():

"""Get stock quantity for all products, including product info via JOIN"""

session = get\_sqlalchemy\_session()

rows = (

session.query(

Stock.product\_id,

Stock.quantity,

Product.name,

Product.sku,

Product.price,

)

.join(Product, Stock.product\_id == Product.id) # JOIN Stock / Product

.order\_by(Stock.product\_id)

.all()

)

data = []

for product\_id, quantity, name, sku, price in rows:

data.append({

"Article": int(product\_id),

"Nom": name,

"Numéro SKU": sku,

"Prix unitaire": float(price) if price is not None else None,

"Unités en stock": int(quantity),

})

return data

**Explication**

* La méthode .join(Product, Stock.product\_id == Product.id) établit la **relation explicite** entre les deux tables.
* Cela nous permet de sélectionner des colonnes provenant à la fois de **Stock** et de **Product** dans la même requête.
* Le résultat retourné contient donc toutes les informations nécessaires pour un rapport clair : l’ID de l’article, son nom, son SKU, son prix unitaire et la quantité en stock.

Capture :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect. Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.

# QUESTION 3

# En utilisant l’endpoint **POST /stocks/graphql-query** avec la requête suggérée, j’ai pu interroger directement un produit en choisissant les colonnes que je voulais obtenir.

# **Exemple de requête envoyée dans Postman :**

# En utilisant l’endpoint **POST /stocks/graphql-query** avec la requête suggérée, j’ai pu interroger directement un produit en choisissant les colonnes que je voulais obtenir.

# **Exemple de requête envoyée dans Postman :**

{

product(id: "20") {

id

name

sku

price

quantity

}

}

Résultat obtenu :

La réponse JSON a bien retourné les informations du produit avec l’id = 20 :

{

"product": {

"id": 20,

"name": "Super Produit",

"sku": "CHA-002",

"price": 129.99,

"quantity": 59

}

}

**Interprétation**

* On peut sélectionner uniquement les colonnes nécessaires (id, name, sku, price, quantity), contrairement à l’endpoint REST classique qui renvoie un format prédéfini.
* GraphQL apporte donc plus de **flexibilité** : pas besoin de multiplier les endpoints pour différents besoins.
* L’exemple montre que le produit avec **id=20** est bien récupéré et que ses données sont cohérentes avec ce qui est présent dans la base de données.

Capture ;

# Une image contenant texte, capture d’écran, logiciel, nombre Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, nombre

Le contenu généré par l’IA peut être incorrect.

# QUESTION 4 :

Code :

def update\_stock\_redis(order\_items, operation):

""" Update stock quantities in Redis """

if not order\_items:

return

r = get\_redis\_conn()

stock\_keys = list(r.scan\_iter("stock:\*"))

if stock\_keys:

pipeline = r.pipeline()

session = get\_sqlalchemy\_session()

try:

for item in order\_items:

if hasattr(item, 'product\_id'):

product\_id = item.product\_id

quantity = item.quantity

else:

product\_id = item['product\_id']

quantity = item['quantity']

current\_stock = r.hget(f"stock:{product\_id}", "quantity")

current\_stock = int(current\_stock) if current\_stock else 0

if operation == '+':

new\_quantity = current\_stock + quantity

else:

new\_quantity = current\_stock - quantity

fields = \_get\_product\_fields(session, product\_id)

pipeline.hset(

f"stock:{product\_id}",

mapping={

"quantity": int(new\_quantity),

"name": fields["name"] or "",

"sku": fields["sku"] or "",

"price": fields["price"] if fields["price"] is not None else "",

},

)

pipeline.execute()

finally:

session.close()

else:

\_populate\_redis\_from\_mysql(r)

# QUESTION 5 :

Requête GraphQL (Postman) :

{

product(id: "20") {

id

name

sku

price

quantity

}

}

Réponse obtenue :

{

    "product": {

        "id": 20,

        "name": "Super Produit",

        "price": 129.99,

        "quantity": 59,

        "sku": "CHA-002"

    }

}

Capture :

Une image contenant texte, capture d’écran, logiciel, affichage

Le contenu généré par l’IA peut être incorrect.

# QUESTION 6 :

Pour cette tâche, j’ai simulé un fournisseur externe en lançant un deuxième conteneur basé sur scripts/supplier\_app.py.

J’ai construit l’image avec le Dockerfile dans scripts/ et démarré le conteneur via scripts/docker-compose.yml.

Ce conteneur exécute périodiquement (toutes les 10 secondes) des appels HTTP POST vers l’endpoint /stocks/graphql-query du service store\_manager.

Comme j’avais modifié le schéma GraphQL à l’étape précédente (colonne name, sku, price, etc.), j’ai pu vérifier dans les logs que la réponse retournait bien un objet produit complet :

Une image contenant texte, capture d’écran, document, Police

Le contenu généré par l’IA peut être incorrect.

Question : Similarités des docker-compose.yml et communication

En regardant attentivement les deux fichiers (docker-compose.yml à la racine et celui dans scripts/), je constate qu’ils ont en commun la **définition d’un réseau Docker** partagé.

**Mécanisme**

* Chaque conteneur attaché à ce réseau peut **résoudre les noms des autres conteneurs** (grâce au DNS interne Docker).
* Concrètement, depuis supplier\_app, j’accède à store\_manager via http://store\_manager:5000 au lieu d’utiliser une IP.
* Les **ports exposés** (5000:5000, 3306:3306, etc.) servent uniquement pour accéder depuis ma machine hôte.
* Mais **entre conteneurs**, on utilise le port interne et le nom du service.

YML – côté racine

services:

store\_manager:

build: .

environment:

- PYTHONUNBUFFERED=1

volumes:

- .:/app

ports:

- "5000:5000"

networks:

- labo03-network

depends\_on:

- mysql

- redis

mysql:

image: mysql:8

restart: unless-stopped

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: labo03\_db

MYSQL\_USER: labo03

MYSQL\_PASSWORD: labo03

networks:

- labo03-network

ports:

- "3306:3306"

volumes:

- mysql\_data:/var/lib/mysql

- ./db-init:/docker-entrypoint-initdb.d

redis:

image: redis:7

restart: unless-stopped

networks:

- labo03-network

ports:

- "6379:6379"

networks:

labo03-network:

driver: bridge

external: true

volumes:

mysql\_data:

YML – côté scripts

services:

supplier\_app:

build: .

environment:

- PYTHONUNBUFFERED=1

volumes:

- .:/app

networks:

- labo03-network

networks:

labo03-network:

driver: bridge

external: true

name: labo03-network

**En résumé** : Les deux docker-compose.yml partagent un **même réseau externe**   
C’est ce réseau qui permet à supplier\_app et store\_manager (ainsi que mysql et redis) de communiquer entre eux directement par leurs **noms de service**.

# TEST

