**📝 Compte Rendu sur le Fonctionnement du Projet E-commerce API**

**📌 Introduction**

Le projet **E-commerce API** est une application permettant de gérer des **produits, clients et commandes** via une API RESTful. Développée avec **Node.js**, **Express** et **MongoDB**, elle utilise **Docker** pour garantir une exécution stable et indépendante de l'environnement.

Ce document explique **le fonctionnement du projet**, depuis son installation jusqu'à son utilisation complète.

**🏗 1. Mise en Place du Projet**

**Clonage et Installation**

Pour récupérer le projet, il faut d'abord le **cloner depuis GitHub** :

git clone https://github.com/Meisseu/ecommerce-api.git

cd ecommerce-api

Cela crée une copie locale du code source.

**Démarrage avec Docker**

Le projet utilise **Docker** pour gérer son environnement, notamment la base de données **MongoDB**. Il existe **deux méthodes** pour démarrer l'API :

**✅ Méthode 1 : Exécution Automatisée avec run.sh**

Si **Git Bash** est installé, il suffit d’exécuter :

bash run.sh

Ce script **automatise** :

1. La création d'un réseau Docker.
2. Le lancement d’un conteneur MongoDB.
3. La construction et l’exécution de l’API Express.

**✅ Méthode 2 : Exécution Manuelle**

Si run.sh ne fonctionne pas, on peut **exécuter les commandes individuellement** :

docker network create backend-test-network

docker run -d --name db-container --network backend-test-network --network-alias db-container mongo:4.4

docker build -t express-project .

docker run -d --name express-app --network backend-test-network -p 5000:5000 express-project

Une fois lancé, l’API est accessible sur **http://localhost:5000**.

**🔍 2. Fonctionnement de l’API**

L’API suit une structure **RESTful**, avec les fonctionnalités suivantes :

* **Gestion des Produits** (/api/products)
* **Gestion des Clients** (/api/customers)
* **Gestion des Commandes** (/api/orders)
* **Passation de Commandes Directes** pour un client (/api/orders/direct/:customerId)

Les échanges se font via des **requêtes HTTP (GET, POST, PUT, DELETE)** et les données sont envoyées au format **JSON**.

**📌 3. Endpoints de l’API**

**📦 Gestion des Produits**

| **Méthode** | **URL** | **Description** |
| --- | --- | --- |
| POST | /api/products | Ajouter un produit |
| GET | /api/products | Récupérer tous les produits |
| GET | /api/products/:id | Voir un produit spécifique |
| PUT | /api/products/:id | Modifier un produit |
| DELETE | /api/products/:id | Supprimer un produit |

**Exemple : Ajouter un produit**

curl -X POST -H "Content-Type: application/json" -d "{\"name\":\"Laptop\",\"price\":1000,\"stock\":10}" http://localhost:5000/api/products

**Réponse attendue** :

{

"\_id": "65a42b1fcf6a1c3d9b654321",

"name": "Laptop",

"price": 1000,

"stock": 10

}

**👤 Gestion des Clients**

| **Méthode** | **URL** | **Description** |
| --- | --- | --- |
| POST | /api/customers | Ajouter un client |
| GET | /api/customers | Voir la liste des clients |

**Exemple : Ajouter un client**

curl -X POST -H "Content-Type: application/json" -d "{\"name\":\"John Doe\",\"email\":\"john@example.com\"}" http://localhost:5000/api/customers

**Réponse attendue** :

{

"\_id": "65b12345cf6a1c3d9b987654",

"name": "John Doe",

"email": "john@example.com"

}

**🛒 Gestion des Commandes**

| **Méthode** | **URL** | **Description** |
| --- | --- | --- |
| POST | /api/orders | Créer une commande |
| GET | /api/orders/:id | Voir une commande |
| PUT | /api/orders/:id | Modifier une commande |
| DELETE | /api/orders/:id | Supprimer une commande |

**🔄 Passer une Commande Directe pour un Client**

Cette route permet à un **client existant** de passer directement une commande avec des produits.

curl -X POST -H "Content-Type: application/json" -d "{\"products\":[\"65a42b1fcf6a1c3d9b654321\"]}" http://localhost:5000/api/orders/direct/65b12345cf6a1c3d9b987654

**Réponse attendue** :

{

"message": "Order created successfully",

"order": {

"\_id": "65a42b1fcf6a1c3d9b999999",

"customer": "65b12345cf6a1c3d9b987654",

"products": ["65a42b1fcf6a1c3d9b654321"]

}

}

**🛠 4. Tests Automatisés avec test.sh**

Un script test.sh est fourni pour **tester automatiquement l’API**.

**Exécuter le script**

**Sous Git Bash ou Linux/macOS**

./test.sh

**Sous Windows CMD ou PowerShell**

bash test.sh

Le script effectue les actions suivantes :

1. Créer un client
2. Ajouter un produit
3. Passer une commande

📌 **Réponse attendue après exécution** :

{

"message": "Order created successfully",

"order": {

"\_id": "65a42b1fcf6a1c3d9b999999",

"customer": "65b12345cf6a1c3d9b987654",

"products": ["65a42b1fcf6a1c3d9b654321"]

}

}

**🧹 5. Arrêter et Nettoyer Docker**

Lorsque l’API n’est plus nécessaire, il est recommandé de **stopper et supprimer les conteneurs** pour libérer des ressources.

docker stop express-app db-container

docker rm express-app db-container

docker network rm backend-test-network

**🎯 Conclusion**

L’API E-commerce offre une solution **complète** pour la gestion des produits, des clients et des commandes via des requêtes HTTP.

**✅ Ce que nous avons fait :**

* Installation et configuration avec **Docker**
* Gestion des **produits, clients et commandes**
* Tests automatisés avec **test.sh**
* Nettoyage et maintenance de l’API

**🔥 Prochaines Améliorations :**

* **Sécurisation avec JWT** pour gérer l’authentification
* **Ajout de Swagger** pour documenter l’API
* **Pagination et recherche** pour améliorer l’affichage des produits

L’API est maintenant **fonctionnelle, testée et prête à être utilisée en production !** 🚀