Documentation du Projet
E-Commerce Microservices

ESGI
école supérieure de
génie informatique

Contents

Documentation du Projet E-Commerce Microservices

Introduction

Documentation détaillée du **Projet E-Commerce Microservices**. Ce projet est une application e-commerce complète basée sur une architecture microservices, conçue pour offrir modularité, scalabilité et maintenabilité. Cette documentation couvre la structure du projet, les composants individuels, les configurations nécessaires, les méthodes de déploiement, ainsi que les pratiques de développement et de test mises en œuvre.

Architecture du Projet

Vue d'Ensemble

Le projet est structuré autour d'une architecture microservices, permettant de séparer les différentes fonctionnalités en services indépendants. Cette approche facilite la maintenance, la scalabilité et l'évolution de l'application.

Composants Principaux :

- **Frontend** : Application utilisateur développée avec Vue.js.
- **Backend** : Trois microservices principaux :
  - **Auth Service** : Gestion de l'authentification et des utilisateurs.
  - **Product Service** : Gestion des produits et du panier.
  - **Order Service** : Gestion des commandes.
- **Base de Données** : Chaque microservice dispose de sa propre base de données MongoDB.
- **Conteneurisation** : Utilisation de Docker et Docker Compose pour la conteneurisation et l'orchestration, avec Docker Swarm pour la production.
- **Intégration Continue** : Utilisation de GitLab CI/CD avec des runners Docker internes à l'entreprise.
- Les machines du projets utilisées sont toutes des **debian 12**, veuillez prendre en compte cette information que ce soit ou un déploiement local ou pour votre future pipeline :) ### Microservices

1. **Auth Service** : Gère l'inscription, la connexion, la gestion des profils utilisateurs et l'authentification via JWT.
2. **Product Service** : Gère l'affichage des produits, les opérations sur le panier (ajout, modification, suppression).
3. **Order Service** : Gère la création et la consultation des commandes des utilisateurs.

Structure des Répertoires

Voici la structure complète du projet, excluant les dossiers ignorés (`node_modules`, `dist`, `.git`, `coverage`):

```
.
├── docker-compose.prod.yml
├── docker-compose.yml
├── frontend
│   ├── build-front.yml
│   ├── Dockerfile
│   ├── Dockerfile.dev
│   ├── index.html
│   ├── package.json
│   ├── package-lock.json
│   ├── public
│   ├── README.md
│   ├── server.cjs
│   ├── src
│   │   ├── App.vue
│   │   ├── assets
│   │   ├── components
│   │   ├── main.js
│   │   ├── router
│   │   ├── services
│   │   └── style.css
│   ├── test-results
│   ├── tests
│   ├── vite.config.js
│   └── vitest.config.js
├── package.json
```

```
├── package-lock.json
├── README.md
├── scripts
│   ├── deploy.sh
│   ├── init-products.sh
│   ├── run-tests.sh
│   ├── setup.sh
│   └── tests.notes
└── services
    ├── auth-service
    │   ├── build-auth.yml
    │   ├── Dockerfile
    │   ├── jest.config.js
    │   ├── package.json
    │   ├── package-lock.json
    │   ├── src
    │   └── tests
    ├── order-service
    │   ├── build-order.yml
    │   ├── Dockerfile
    │   ├── jest.config.js
    │   ├── package.json
    │   ├── package-lock.json
    │   ├── src
    │   └── tests
    └── product-service
        ├── build-product.yml
        ├── Dockerfile
        ├── jest.config.js
        ├── package.json
        ├── package-lock.json
        ├── src
        └── tests
```

Environnement et Dépendances

Environnement d'Exécution

L'application peut être déployée de trois manières principales :

1. **Déploiement Manuel avec PM2 pour la Pré-production** : Exécution directe des services sur votre machine locale en utilisant PM2 pour gérer les processus, idéal pour la pré-production ou les environnements de test.
2. **Déploiement avec Docker et Docker Compose** : Utilisation de Docker pour conteneuriser les services et Docker Compose pour orchestrer les conteneurs en environnement de développement.
3. **Déploiement en Production avec Docker Swarm** : Utilisation de Docker Swarm pour orchestrer les conteneurs en environnement de production, avec le fichier `docker-compose.prod.yml`.

Logiciels Nécessaires

Pour le Déploiement Manuel

- **Node.js** : Version 14.x ou supérieure.
- **npm** : Version 6.x ou supérieure (généralement inclus avec Node.js).
- **Git** : Pour cloner le dépôt du projet.
- **MongoDB** : Version 4.4 installée et en cours d'exécution localement.
- **PM2** : Pour gérer les processus Node.js en production ou pré-production.

Pour le Déploiement avec Docker

- **Docker** : Version 20.x ou supérieure.
- **Docker Compose** : Version 1.27 ou supérieure.
- **Docker Swarm** : Inclus avec Docker (activation via `docker swarm init`).

Automatisation avec GitLab CI/CD

- **GitLab Runner** : Configuré pour exécuter les pipelines CI/CD avec des runners Docker (Docker-in-Docker - dind) internes à l'entreprise.

Outils de Développement (Facultatif)

- **Visual Studio Code** ou tout autre éditeur de code.

- **Postman** : Pour tester les API backend.
- **MongoDB Compass** : Pour gérer visuellement les bases de données MongoDB.

Dépendances

Chaque composant (frontend et services backend) possède ses propres dépendances gérées via `package.json`. Voici un aperçu des dépendances principales :

Frontend

- **Vue.js** : Framework JavaScript pour construire l'interface utilisateur.
- **Vue Router** : Gestion des routes côté client.
- **Axios** : Pour les requêtes HTTP vers les APIs backend.
- **Vite** : Outil de bundling et de développement rapide.
- **Vitest** : Framework de test pour Vue.js.

Services Backend

- **Express.js** : Framework web pour Node.js.
- **Mongoose** : ODM pour interagir avec MongoDB.
- **jsonwebtoken** : Pour la génération et la vérification des tokens JWT.
- **bcrypt** : Pour le hachage des mots de passe.
- **Cors** : Middleware pour gérer les en-têtes CORS.
- **dotenv** : Pour gérer les variables d'environnement.
- **Jest** : Framework de test pour Node.js.

Détails des Composants

Frontend Le frontend est une application Vue.js qui sert d'interface utilisateur pour l'application e-commerce.

- **Technologies** : Vue.js, Vue Router, Axios, Vite.
- **Structure** :
  - **`index.html`** : Point d'entrée HTML de l'application.
  - **`src/`** : Contient le code source de l'application.
    - **`main.js`** : Point d'entrée JavaScript, configure l'application Vue et le routeur.
    - **`App.vue`** : Composant racine de l'application.
    - **`components/`** : Contient les composants Vue réutilisables.

        - **`AuthTest.vue`** : Composant pour l'inscription, la connexion et l'affichage du profil utilisateur.
        - **`OrderHistory.vue`** : Composant pour afficher l'historique des commandes.
        - **`ProductList.vue`** : Composant pour afficher la liste des produits disponibles.
        - **`ShoppingCart.vue`** : Composant pour gérer le panier d'achat de l'utilisateur.
    * **`services/`** : Contient les services pour communiquer avec les APIs backend via Axios.
        - **`authService.js`**, **`cartService.js`**, **`orderService.js`**, **`productService.js`**.
    * **`router/`** : Configure les routes de l'application avec Vue Router.
    * **`assets/`** : Contient les fichiers statiques tels que les images et les styles CSS.
- **`server.cjs`** : Serveur Express qui sert les fichiers statiques et configure les proxies vers les microservices backend (pour la production)
- **`tests/`** : Contient les tests unitaires du frontend.
- **`vite.config.js`** et **`vitest.config.js`** : Configurations pour Vite et Vitest (pour le developement)

Fonctionnalités Clés :

- **Authentification** : Inscription, connexion, gestion du profil via JWT.
- **Gestion des Produits** : Affichage de la liste des produits, détails des produits.
- **Gestion du Panier** : Ajout, modification, suppression d'articles dans le panier.
- **Gestion des Commandes** : Passation de commandes, affichage de l'historique des commandes.

Services Backend Chaque microservice backend est développé en Node.js avec Express et possède sa propre base de données MongoDB.

Auth Service

- **Fonctionnalités** :
    - Inscription, connexion, gestion du profil utilisateur.
    - Génération et vérification des tokens JWT.
- **Structure** :
    - **`src/app.js`** : Point d'entrée du service.
    - **`src/controllers/authController.js`** : Contient les fonctions pour l'inscription, la connexion et la récupération du profil.
    - **`src/models/user.js`** : Modèle Mongoose pour les utilisateurs.

- **`src/routes/authRoutes.js`** : Définit les routes pour l'authentification.
- **`src/middleware/auth.js`** : Middleware pour vérifier le token JWT.
- **Tests** : Situés dans **`tests/`**.

Product Service

- **Fonctionnalités** :
  - Gestion des produits : liste, détails.
  - Gestion du panier : ajout, suppression, mise à jour.
- **Structure** :
  - **`src/app.js`** : Point d'entrée du service.
  - **`src/controllers/productController.js`** : Gère les opérations liées aux produits.
  - **`src/models/product.js`** : Modèle Mongoose pour les produits.
  - **`src/routes/productRoutes.js`** : Définit les routes pour les produits.
  - **`src/controllers/cartController.js`** : Gère les opérations sur le panier.
  - **`src/models/cart.js`** : Modèle Mongoose pour le panier.
  - **`src/routes/cartRoutes.js`** : Définit les routes pour le panier.
  - **Tests** : Situés dans **`tests/`**.

Order Service

- **Fonctionnalités** :
  - Création et consultation des commandes.
- **Structure** :
  - **`src/app.js`** : Point d'entrée du service.
  - **`src/controllers/orderController.js`** : Gère la création et la récupération des commandes.
  - **`src/models/order.js`** : Modèle Mongoose pour les commandes.
  - **`src/routes/orderRoutes.js`** : Définit les routes pour les commandes.
  - **Tests** : Situés dans **`tests/`**.

Automatisation avec GitLab CI/CD

Le projet utilise GitLab CI/CD pour automatiser les processus de build, test et déploiement. Chaque microservice et le frontend possèdent leur propre fichier de configuration de pipeline situé à la racine

de leur répertoire respectif, nommé `build-*.yml` (par exemple, `build-front.yml` pour le frontend).

- **Runners Docker Internes** : Les pipelines s'exécutent sur des runners Docker-in-Docker (dind) internes à l'entreprise, ce qui permet de construire des images Docker pendant les jobs CI/CD.
- **Variables CI/CD** : Les variables d'environnement nécessaires (comme `CI_REGISTRY_IMAGE`, `IMAGE_FULL`, `IMAGE_TAG`, `JWT_SECRET`) sont définies dans les variables GitLab CI/CD.

Configuration et Déploiement

Déploiement Manuel avec PM2 pour la Pré-production

Le déploiement manuel consiste à exécuter directement les services sur votre machine locale en utilisant PM2 pour gérer les processus. Le script `deploy.sh` est utilisé pour automatiser ce processus et est idéal pour les environnements de pré-production ou de test.

Utilisation de `deploy.sh`

- **Chemin** : `scripts/deploy.sh`
- **Fonctionnalités** :
  - Installe les dépendances pour le frontend et les services backend.
  - Construit le frontend avec `npm run build` (comme en production).
  - Démarre les services backend et le frontend en utilisant PM2.

Étapes :

1. **Installer PM2** :

```
npm install -g pm2
```

2. **Rendre le script exécutable** :

```
chmod +x scripts/deploy.sh
mkdir /opt/e-commerce
cp -r e-commerce-vue/* /opt/e-commerce
cd /opt/e-commerce
```

3. **Exécuter le script** :

```
./scripts/deploy.sh
./scripts/init-products.sh
```

Remarques :

- Assurez-vous que MongoDB est installé et en cours d'exécution sur votre machine.
- Configurez les variables d'environnement nécessaires dans des fichiers `.env` situés dans chaque répertoire de service.
- PM2 offre des fonctionnalités de surveillance, de gestion des logs et de redémarrage automatique.
- Exemple avec SSH :

```
sync -avz --delete --exclude '.git' -e "ssh -o StrictHostKeyChecking=no" ./
↪  root@192.168.1.108:/opt/e-commerce
```

Déploiement avec Docker et Docker Compose

Le projet utilise Docker pour conteneuriser les services et le frontend, et Docker Compose pour orchestrer les conteneurs. Le fichier `docker-compose.yml` est configuré pour déployer la stack en mode développement. En modifiant la cible (`target`) dans les Dockerfiles ou les variables d'environnement, il est possible de déployer en mode production.

Environnement de Développement

1. **Prérequis** :
   - **Docker** et **Docker Compose** installés sur votre machine.
   - **Git** pour cloner le dépôt du projet.
2. **Cloner le Dépôt** :

   ```
   git clone <URL_DU_DÉPÔT>
   cd <NOM_DU_PROJET>
   ```

3. **Construire et Démarrer les Conteneurs** :

   ```
   docker-compose up --build
   ```

4. **Initialisation des Données** :

```
./scripts/init-products.sh
```

5. **Accès à l'Application** :
    - Ouvrez votre navigateur et accédez à `http://localhost:8080`.

Environnement de Production avec Docker Swarm Pour le déploiement en production, le fichier `docker-compose.prod.yml` est utilisé avec Docker Swarm.

1. **Prérequis** :
    - **Docker** installé sur le serveur de production.
    - Accès SSH au serveur.
2. **Initialiser Docker Swarm** :

```
docker swarm init
```

3. **Déployer la Stack en Production** :

```
docker stack deploy -c docker-compose.prod.yml e-commerce
```

4. **Vérifier le Déploiement** :

```
docker stack services e-commerce
```

5. **Accès à l'Application** :
    - Accédez à l'adresse IP ou au domaine de votre serveur suivi du port approprié (par exemple, `http://192.168.1.108:8080`).

Remarques :

- Le fichier `docker-compose.prod.yml` est configuré pour utiliser les images Docker pré-construites.
- Les variables comme `IMAGE_TAG` et `CI_REGISTRY_IMAGE` sont gérées par GitLab CI/CD.
- Adapter le script init-products.sh si nécessaire pour ajouter les produits.

Scripts d'Automatisation

Le répertoire **`scripts/`** contient plusieurs scripts pour faciliter le déploiement et la gestion du projet :

- **`deploy.sh`** : Automatise le déploiement manuel de l'application avec PM2 pour la pré-production.
- **`init-products.sh`** : Initialise la base de données des produits avec des données par défaut.
- **`run-tests.sh`** : Exécute l'ensemble des tests pour les services backend et le frontend.
- **`setup.sh`** : Prépare l'environnement de développement en installant les dépendances nécessaires.
- **`tests.notes`** : Contient des notes ou des instructions supplémentaires liées aux tests.

Exemples d'utilisation : - **Mise en place initiale sur le serveur de dev** :

```
./scripts/setup.sh
```

- **Exécuter les Tests** :

```
./scripts/run-tests.sh
```

- **Initialiser les Données des Produits** :

```
./scripts/init-products.sh
```

Fonctionnalités Principales

Authentification JWT

- **Inscription et Connexion** : Les utilisateurs peuvent s'inscrire avec un email et un mot de passe, puis se connecter pour obtenir un token JWT.
- **Protection des Routes** : Les routes sensibles sont protégées par un middleware qui vérifie le token JWT.
- **Gestion du Profil** : Les utilisateurs peuvent consulter leur profil.

Gestion des Produits et du Panier

- **Liste des Produits** : Affichage de tous les produits disponibles.
- **Détails du Produit** : Affichage des informations détaillées sur un produit.
- **Panier** :
  - **Ajout au Panier** : Les utilisateurs peuvent ajouter des produits à leur panier.
  - **Modification du Panier** : Mise à jour ou suppression d'articles du panier.
  - **Consultation du Panier** : Affichage des articles dans le panier.

Gestion des Commandes

- **Passation de Commande** : Les utilisateurs peuvent passer une commande basée sur le contenu de leur panier.
- **Historique des Commandes** : Consultation des commandes précédentes.

Flux de Données

1. **Authentification** :
   - **Inscription** :
     - Frontend → `/api/auth/register` → Auth Service → Base de Données → Frontend.
   - **Connexion** :
     - Frontend → `/api/auth/login` → Auth Service → Base de Données → Frontend.
   - **Profil** :
     - Frontend (avec JWT) → `/api/auth/profile` → Auth Service → Base de Données → Frontend.
2. **Gestion des Produits et du Panier** :
   - **Liste des Produits** :
     - Frontend → `/api/products` → Product Service → Base de Données → Frontend.
   - **Gestion du Panier** :
     - Frontend → `/api/cart/*` → Product Service → Base de Données → Frontend.
3. **Gestion des Commandes** :
   - **Passation de Commande** :
     - Frontend → `/api/orders/create` → Order Service → Base de Données → Frontend.
   - **Historique des Commandes** :
     - Frontend → `/api/orders/user/:userId` → Order Service → Base de Données → Frontend.

Tests et Qualité du Code

Tests de Sécurité

- **SonarCloud** : Peut être implémenté pour une analyse continue de la qualité et de la sécurité du code.
- **Trivy** : Outil de scan de vulnérabilités pour le code et les conteneurs.

Tests Frontend

- **Framework** : Vitest
- **Localisation** : `frontend/tests/`
- **Types de Tests** :
  - **Unitaires** : Tests des composants Vue.js et des services Axios.

Exécution des Tests :

```
cd frontend
npm run test
npm run test:unit
npm run test:coverage
npm run lint:report || true
```

Tests Backend

- **Framework** : Jest
- **Localisation** : `services/<service-name>/tests/`
- **Types de Tests** :
  - **Unitaires** : Tests des contrôleurs, des modèles et des routes.

Exécution des Tests :

```
cd services/<service-name>
npm test
npm run lint || true # Si disponible
```

Bonnes Pratiques et Considérations

- **Modularité** : Chaque service est indépendant, ce qui facilite la maintenance et la scalabilité.
- **Sécurité** :
  - Utilisation de tokens JWT pour sécuriser les communications.
  - Protection des routes sensibles avec des middlewares d'authentification.
  - Stockage sécurisé des mots de passe (hachage avec bcrypt).
  - Gestion des secrets avec des variables d'environnement sécurisées.
- **Gestion des Erreurs** : Implémentation de messages d'erreur clairs pour faciliter le débogage.
- **Logs** :
  - Les services backend utilisent des middlewares de logging pour enregistrer les requêtes reçues.
- **Configuration du Proxy** :
  - Le fichier `server.cjs` est configuré pour rediriger correctement les requêtes du frontend vers les services backend.
- **Conteneurisation et Orchestration** :
  - Utilisation de Docker pour isoler les environnements de développement et de production.
  - Docker Swarm pour l'orchestration en production, offrant une haute disponibilité et un load balancing.
- **Tests Automatisés** : Maintien d'une couverture de tests élevée pour assurer la qualité et la fiabilité du code.
- **CI/CD avec GitLab** :
  - Intégration de pipelines CI/CD pour automatiser les tests, la construction des images Docker et le déploiement.
  - Utilisation de runners Docker-in-Docker internes à l'entreprise.
- **Utilisation de PM2 pour la Pré-production** :
  - PM2 est utilisé pour gérer les processus Node.js dans les environnements de pré-production ou de test.
- **Gestion des Variables d'Environnement** :
  - Les variables sensibles, comme `JWT_SECRET`, sont gérées via des variables d'environnement et doivent être correctement configurées.

Annexes

Exemples de Commandes CURL

Pour faciliter les tests des différentes APIs de votre application, voici une série d'exemples de commandes `curl` que vous pouvez utiliser.

Auth Service

- **Inscription d'un Nouvel Utilisateur**

```
curl -X POST http://localhost:3001/api/auth/register \
     -H "Content-Type: application/json" \
     -d '{"email": "user@example.com", "password": "password123"}'
```

```
{"message":"Utilisateur créé avec suc-
↪  cès","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NzNiNmIyYzAyNTJmZ
↪  pCKl_bGiXbqT8aSFghE2MQBkCmXo","userId":"673b6b2c0252fd5bfe997dac"}
```

- **Connexion d'un Utilisateur**

```
curl -X POST http://localhost:3001/api/auth/login \
     -H "Content-Type: application/json" \
     -d '{"email": "user@example.com", "password": "password123"}'
```

```
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NzNiNmIyYzAyNTJmZDViZmU5O
```

- **Récupération du Profil Utilisateur**

```
curl -X GET http://localhost:3001/api/auth/profile \
     -H "Authorization: Bearer <JWT_TOKEN>"
```

```
{"_id":"673b6b2c0252fd5bfe997dac","email":"user@example.com","createdAt":"2024-
↪  11-18T16:28:28.239Z","__v":0}
```

Product Service

- **Récupération de la Liste des Produits**

```
curl -X GET http://localhost:3000/api/products
```

```
[{"_id":"673b6ba048c194f02ade2aac","name":"Smartphone Galaxy
↪  S21","price":899,"description":"Dernier smartphone Samsung avec appareil
↪  photo 108MP","stock":15,"createdAt":"2024-11-
↪  18T16:30:24.472Z","__v":0},{"_id":"673b6ba048c194f02ade2aae","name":"MacBook
↪  Pro M1","price":1299,"description":"Ordinateur portable Apple avec puce
↪  M1","stock":10,"createdAt":"2024-11-
↪  18T16:30:24.489Z","__v":0},{"_id":"673b6ba048c194f02ade2ab0","name":"PS5","price":49
↪  de jeu dernière génération","stock":5,"createdAt":"2024-11-
↪  18T16:30:24.498Z","__v":0},{"_id":"673b6ba048c194f02ade2ab2","name":"Écouteurs
↪  AirPods Pro","price":249,"description":"Écouteurs sans fil avec
↪  réduction de bruit","stock":20,"createdAt":"2024-11-
↪  18T16:30:24.506Z","__v":0},{"_id":"673b6ba048c194f02ade2ab4","name":"Nintendo
↪  Switch","price":299,"description":"Console de jeu
↪  portable","stock":12,"createdAt":"2024-11-
↪  18T16:30:24.516Z","__v":0},{"_id":"673b6ba048c194f02ade2ab6","name":"iPad
↪  Air","price":599,"description":"Tablette Apple avec écran
↪  Retina","stock":8,"createdAt":"2024-11-
↪  18T16:30:24.526Z","__v":0},{"_id":"673b6ba048c194f02ade2ab8","name":"Montre
↪  connectée","price":199,"description":"Montre intelligente avec suivi
↪  d'activité","stock":25,"createdAt":"2024-11-
↪  18T16:30:24.535Z","__v":0},{"_id":"673b6ba048c194f02ade2aba","name":"Enceinte
↪  Bluetooth","price":79,"description":"Enceinte portable
↪  waterproof","stock":30,"createdAt":"2024-11-18T16:30:24.544Z","__v":0}]1
```


- **Ajout d'un Produit au Panier**

```
curl -X POST http://localhost:3000/api/cart/add \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer efrei_super_pass" \
  -d '{
    "userId": "673b6b2c0252fd5bfe997dac",
    "productId": "673b6ba048c194f02ade2aba",
    "quantity": 2
  }'
```


```
{"userId":"673b6b2c0252fd5bfe997dac","items":[{"productId":{"_id":"673b6ba048c194f02ade2
↪  Bluetooth","price":79,"description":"Enceinte portable
↪  waterproof","stock":30,"createdAt":"2024-11-
↪  18T16:30:24.544Z","__v":0},"quantity":1,"_id":"673b6f7f48c194f02ade2ad0"}],"_id":"67
↪  11-18T16:46:55.284Z","__v":0}
```


Order Service

- **Passation d'une Commande**

```
curl -X POST http://localhost:3002/api/orders \
  -H "Authorization: Bearer eyJhbGciOiJI-
  ↪  UzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NzNiNmIyYzAyNTJmZDViZmU5OTdkYWMiLCJlbWFpb
  ↪  \
  -H "Content-Type: application/json" \
  -d '{
    "products": [{
      "productId": "673b6ba048c194f02ade2aba",
      "quantity": 1
    }],
    "shippingAddress": {
      "street": "123 Test St",
      "city": "Test City",
      "postalCode": "12345"
    }
  }'
```

```
{"userId":"673b6b2c0252fd5bfe997dac","products":[{"productId":"673b6ba048c194f02ade2aba"
 ↪  Blue-
 ↪  tooth","price":79,"quantity":1,"_id":"673b70050a40d45c0f920818"}],"totalAmount":79,"
 ↪  Test St","city":"Test
 ↪  City","postalCode":"12345"},"_id":"673b70050a40d45c0f920817","createdAt":"2024-
 ↪  11-18T16:49:09.281Z","__v":0}
```

- **Consultation de l'Historique des Commandes**

```
  curl -X GET http://localhost:3002/api/orders \
       -H "Authorization: Bearer <JWT_TOKEN>"
```

```
[{"shippingAddress":{"street":"123 Test St","city":"Test
 ↪  City","postalCode":"12345"},"_id":"673b70050a40d45c0f920817","userId":"673b6b2c0252f
 ↪  Blue-
 ↪  tooth","price":79,"quantity":1,"_id":"673b70050a40d45c0f920818"}],"totalAmount":79,"
 ↪  11-18T16:49:09.281Z","__v":0}]
```

Comment Exécuter le Projet

Déploiement Manuel avec PM2 pour la Pré-production

1. **Prérequis** :
   - **Node.js** et **npm** installés sur votre machine.
   - **MongoDB** installé et en cours d'exécution localement.
   - **Git** pour cloner le dépôt.
   - **PM2** installé globalement.
2. **Cloner le Dépôt** :

```
git clone
cd  /opt/e-commerce-vue
```

3. **Exécuter le Script de Déploiement** :

```
./scripts/deploy.sh
```

4. **Initialisation des Données** :

```
./scripts/init-products.sh
```

5. **Accès à l'Application** :
   - Ouvrez votre navigateur et accédez à `http://localhost:8080`.

Déploiement avec Docker et Docker Compose

1. **Prérequis** :
   - **Docker** et **Docker Compose** installés sur votre machine.
   - **Git** pour cloner le dépôt.
2. **Cloner le Dépôt** :

```
git clone <URL_DU_DÉPÔT>
cd <NOM_DU_PROJET>
```

3. **Construire et Démarrer les Conteneurs** :

```
docker-compose up --build
```

4. **Initialisation des Données** :

```
./scripts/init-products.sh
```

5. **Accès à l'Application** :
   - Ouvrez votre navigateur et accédez à `http://localhost:8080`.

Déploiement en Production avec Docker Swarm

1. **Prérequis** :
   - **Docker** installé sur le serveur de production.
   - Accès SSH au serveur.
2. **Initialiser Docker Swarm** :

```
docker swarm init
```

3. **Déployer la Stack en Production** :

```
docker stack deploy -c docker-compose.prod.yml e-commerce
```

4. **Vérifier le Déploiement** :

```
docker stack services e-commerce
```

5. **Accès à l'Application** :
   - Accédez à l'adresse IP ou au domaine de votre serveur suivi du port approprié.

Automatisation avec GitLab CI/CD

1. **Configuration des Pipelines CI/CD** :
   - Les pipelines sont définis dans les fichiers `build-*.yml` situés dans chaque répertoire de service.
   - Assurez-vous que les variables CI/CD sont correctement configurées dans GitLab.
2. **Déclencher les Pipelines** :
   - Les pipelines sont déclenchés automatiquement lors des commits sur les branches `develop` et `main`.
3. **Surveillance des Pipelines** :
   - Utilisez l'interface GitLab pour surveiller l'exécution des pipelines, vérifier les rapports de sécurité et consulter les résultats des tests.