# Plateforme E-commerce
# Vue d'ensemble de l'architecture

## 1 Aperçu général

Ce dépôt contient une plateforme e-commerce full–stack organisée en trois applications principales :

— **backend/** : API REST en Flask + SQLAlchemy, authentification, panneau d'administration, téléversement de fichiers (Cloudinary) et chat propulsé par OpenAI.

— **frontend/** : boutique en ligne en Next.js (App Router, TypeScript) avec pages compte, blog, panier / paiement et interface de chat.

— **model3d/** : customiseur 3D de produits en Vite + React, exposé sur la route `/customerProduct`.

Des scripts Docker et shell au niveau racine facilitent les environnements local / développement / production (`DOCKER_README.md`, fichiers `docker-compose.*.yml`, `run.sh`, `quick-start.sh`, etc.).

## 2 Backend (API Flask)

Répertoire : `backend/`

### 2.1 Points d'entrée

— `backend/app.py` : crée et lance l'application Flask via `create_app()`. Hôte par défaut `0.0.0.0`, port `8080`, mode debug pour le développement.

— `backend/wsgi.py` : point d'entrée WSGI utilisé pour le déploiement (`Procfile`, Docker, etc.).

### 2.2 Configuration et factory

— `backend/app/__init__.py`
  — Crée une instance globale de `Flask`.
  — Charge la configuration depuis `backend/config.py` (classe `Config`).
  — Active CORS pour tous les domaines via `flask_cors.CORS(app)`.
  — Enregistre les blueprints principaux :
  authentification (`/api/auth`), produits (`/api`), commandes (`/api/orders`), promotions (`/api/offers`), blogs (`/api/blogs`), chat IA (`/api/opnai`), téléversements (`/api/upload`) et panneau admin (`/admin`).
  — La route racine `/` redirige vers `/admin`.
  — Un gestionnaire `teardown_appcontext` ferme la session BD à la fin de chaque requête.

— `backend/config.py`
  — Charge les variables d'environnement depuis `.env` (`SECRET_KEY`, `DATABASE_URL`, etc.).
  — `DATABASE_URL` pointe par défaut sur `sqlite:///./ecommerce.db` mais peut cibler PostgreSQL ou MySQL.

### 2.3 Accès aux données (SQLAlchemy)

La couche base de données se trouve dans `backend/app/database.py` et `backend/app/models.py`.

**database.py**

— Création d'un `engine` SQLAlchemy à partir de `Config.DATABASE_URL` (gestion spéciale de SQLite pour le multithreading).

— Définition d'une fabrique de sessions `SessionLocal` et d'une session portée (`db_session`).

— `init_db()` crée toutes les tables via `Base.metadata.create_all`.

— `get_db()` retourne une session à utiliser dans les routes (avec commit/rollback manuel).

— `get_db_context()` fournit un gestionnaire de contexte qui gère commit / rollback automatiquement.

— `DBHelper` centralise quelques opérations courantes (lecture d'un enregistrement, listage, suppression par id).

**models.py** Les principaux modèles métier :

— **User** (table `utilisateurs`) : informations de compte (email unique, mot de passe hashé, date de création) et relation `orders`.

— **Product** (table `products`) : nom, description, prix, images (cover + liste), couleurs, quantité, catégorie (enum), date de création, relations vers `Offer` et `LineOrder`.

— **Offer** (table `offers`) : promotion associée à un produit (pourcentage de remise, début/fin de validité, état actif/inactif).

— **Order** (table `orders`) : commande d'un utilisateur, montant total, statut (`Pending`, `Completed`, `Shipped`), lignes de commande.

— **LineOrder** (table `line_orders`) : ligne de commande (produit, quantité, prix unitaire, taille, couleur).

— **Blog** (table `blogs`) : articles de blog (titre, contenu, image, statut de publication).

Chaque modèle expose une méthode `to_dict()` pour faciliter la sérialisation JSON dans l'API.

### 2.4 Fonctionnalités du backend

— Authentification utilisateur par email / mot de passe, avec émission de JWT (`/api/auth`).

— Gestion de catalogue produits : catégories, création, mise à jour, suppression, récupération par id ou liste (`/api/products`).

— Gestion des commandes : création d'une commande à partir d'un panier (lignes detail produit / taille / couleur), consultation et mise à jour (`/api/orders`).

— Gestion des offres promotionnelles liées aux produits (`/api/offers`).

— Gestion des articles de blog (`/api/blogs`).

— Téléversement d'images sur Cloudinary (une ou plusieurs images) via `/api/upload`.

— Chat IA contextualisé par le catalogue (produits, offres, blogs récents) via `/api/opnai/chat` avec l'API Chat Completions d'OpenAI.

— Panneau d'administration HTML sous `/admin` pour gérer utilisateurs, produits, commandes, offres et blogs.

## 3 Frontend (boutique Next.js)

Répertoire : `frontend/`

### 3.1 Technologies

— Next.js avec App Router (dossier `src/app/`).

— TypeScript, Tailwind CSS / PostCSS.

— Gestion d'état globale avec Zustand (`src/store/`).

### 3.2 Structure principale

— `src/app/` : layout racine, page d'accueil et sous-routes fonctionnelles :

  — `home/` : composition de la page d'accueil.

  — `products/` : liste et détail des produits.

  — `checkout/` : tunnel de paiement.

  — `orders/` : historique des commandes utilisateur.

  — `profile/` : profil utilisateur.

  — `wishlist/` : liste de souhaits.

  — `blogs/` : liste et détail des articles de blog.

  — `chat/` : interface de l'assistant d'achat IA.

  — `(auth)/` : pages de connexion / inscription.

  — `about/`, `contact/`, `success/` : pages informatives.

— `src/common/` : composants de layout partagés (barre de navigation, pied de page).

— `src/components/` : composants UI réutilisables (ex. carte produit, composants `ui/`).

— `src/services/` : fonctions d'appel à l'API Flask (authentification, commandes, blogs, etc.), basées sur `NEXT_PUBLIC_API_URL`.

— `src/store/` : stores Zustand persistés dans `localStorage` (auth, panier, wishlist, chat).

### 3.3 Flux d'authentification

— Le service `auth.ts` appelle `/api/auth/register`, `/api/auth/login` et `/api/auth/profile`.

— Le JWT renvoyé est sauvegardé dans `localStorage` et dans `useAuthStore`.

— Les composants peuvent vérifier si l'utilisateur est connecté via `isLoggedIn()` et récupérer le profil via `getProfile()`.

### 3.4 Panier et wishlist

— `useCartStore` stocke les lignes de panier (produit, taille, couleur, quantité), fusionne les doublons et permet ajout / suppression / mise à jour / vidage.

— `useWishList` conserve une liste de produits favoris.

### 3.5 Chat IA côté client

— `useChatStore` gère les conversations (id, titre, messages, date de création) et les persiste.

— L'interface Next.js envoie les messages utilisateur à l'endpoint backend `/api/opnai/chat` et ajoute les réponses de l'assistant dans le store.

## 4 Application 3D (model3d)

Répertoire : `model3d/`

### 4.1 Objectif

— Fournir un configurateur 3D d'articles (par exemple un t–shirt) où l'utilisateur peut changer couleurs, textures, logos, etc.

### 4.2 Structure

— `src/main.jsx` : point d'entrée React qui monte `<App />`.

— `src/App.jsx` : utilise `react-router-dom` et expose au moins la route `/customerProduct` qui affiche :

  — `Home` : introduction / contrôles.
  — `Canvas` : scène 3D (React Three Fiber / Drei).
  — `Customizer` : interface de configuration.

— `src/canvas/` : composants 3D (caméra, arrière–plan, modèle de vêtement, etc.).

— `src/components/` : composants de contrôle (sélecteur de couleur, téléverseur d'image, éventuellement AI picker, ...).

— `src/store/` : état local du configurateur (couleurs choisies, logos, textures).

## 5 Résumé des flux de données

— Le frontend Next.js consomme l'API Flask via `fetch` ou via les services dans `src/services/`.

— L'authentification est basée sur des JWT : le backend émet un jeton, stocké côté client et envoyé dans l'en-tête `Authorization` pour les routes protégées.

— SQLAlchemy gère les modèles e-commerce (utilisateurs, produits, offres, commandes, lignes de commande, blogs).

— Les images sont téléversées par le frontend ou l'admin vers Cloudinary via les routes `/api/upload`.

— L'assistant d'achat IA (chat) s'appuie sur OpenAI, enrichi avec un contexte issu des produits, des offres et des blogs récents.

— L'application `model3d` est un SPA séparé, accessible depuis la boutique (par exemple depuis une page produit) via la route `/customerProduct`.