API Backend
Service Flask

Exemples d'URL de base (configurables selon l'environnement de déploiement) :

— Développement local (Docker ou lancement direct) : `http://localhost:8080`

— Côté Next.js : `${process.env.NEXT_PUBLIC_API_URL}`

Toutes les réponses JSON sont encodées en UTF-8 et, sauf mention contraire, retournent le type `application/json`.

1 Authentification – /api/auth

Module : `backend/app/routes/auth_routes.py`

1.1 POST /api/auth/register

Inscrire un nouvel utilisateur.

Corps de requête (JSON)

— `email` (string, requis)

— `password` (string, requis)

Réponses

— **201 Created** : `"message": "User registered"`

— **409 Conflict** : `"error": "Email already exists"`

— **500 Internal Server Error** : `"error": <string>`

1.2 POST /api/auth/login

Authentifier un utilisateur et renvoyer un JWT.

Corps de requête (JSON)

— `email` (string, requis)

— `password` (string, requis)

Réponses

— **200 OK** : JSON contenant un message, un jeton (`token`) et un objet utilisateur (`id`, `email`, `name`).

— **400 Bad Request** : email / mot de passe manquants ou JSON invalide.

— **401 Unauthorized** : `"error": "Invalid credentials"`

— **500 Internal Server Error** : `"error": "Login error: ..."`

Le JWT contient les champs `id`, `email` et `name`, avec une durée de vie de 10 jours.

1.3 GET /api/auth/profile

Récupérer le profil utilisateur contenu dans le JWT.

En-têtes

— `Authorization: Bearer <jwt>` (requis)

Réponses

— **200 OK** : `"user": {...jwtPayload}`

— **400 Bad Request** : `"error": "Token missing"`

— **401 Unauthorized** : `"error": "Invalid or expired token"`

1.4 GET /api/auth/logout

Déconnexion côté serveur (stateless) : le client doit simplement oublier le jeton.

Réponses

— **200 OK** : `"message": "Logged out"`

1.5 GET /api/auth/users

Lister tous les utilisateurs.

Réponses

— **200 OK** : tableau d'objets utilisateur (`id`, `email`, `created_at`).

— **500 Internal Server Error** : `"error": <string>`

1.6 PUT /api/auth/users/<user_id>

Mettre à jour un utilisateur.

Corps de requête (JSON)

— `email` (string, requis)

— `password` (string, optionnel ; remplace le mot de passe existant si fourni)

Réponses

— **200 OK** : `"message": "User updated"`

— **404 Not Found** : `"error": "User not found"`

— **409 Conflict** : `"error": "Email already exists"`

— **500 Internal Server Error** : `"error": <string>`

1.7 DELETE /api/auth/users/<user_id>

Supprimer un utilisateur par identifiant.

Réponses

— **200 OK** : `"message": "User deleted"`

— **404 Not Found** : `"error": "User not found"`

— **500 Internal Server Error** : `"error": <string>`

2 Produits – /api/products

Module : `backend/app/routes/product_routes.py`

2.1 GET /api/products/categories

Retourner la liste des catégories produit disponibles.

Réponses

— **200 OK** : tableau d'objets `{value, label}`.
— **500 Internal Server Error** : `"error": <string>`

2.2 POST /api/products

Créer un nouveau produit.

Corps de requête (JSON)

— `name` (string, requis)
— `description` (string, optionnel)
— `price` (number, requis)
— `image_cover` (string, optionnel ; URL)
— `image_details` (tableau de chaînes, optionnel)
— `colors` (tableau de chaînes, optionnel ; défaut `["black", "white"]`)
— `quantity` (number, requis)
— `category` (string, requis ; nom d'une valeur de l'énumération `ProductCategory`)

Réponses

— **201 Created** : `"message": "Product created", "id": <number>`
— **400 Bad Request** : champs manquants ou invalides.
— **500 Internal Server Error** : `"error": <string>`

2.3 GET /api/products

Lister les produits, éventuellement filtrés par catégorie et limités en nombre.

Paramètres de requête

— `category` (string, optionnel ; nom d'énumération)
— `elements` (integer, optionnel ; nombre maximal de produits)

Réponses

— **200 OK** : tableau de produits, chacun pouvant contenir un champ `discount_percentage` s'il existe une offre active.
— **400 Bad Request** : catégorie invalide.
— **500 Internal Server Error** : `"error": <string>`

2.4 GET /api/products/recommande

Récupérer jusqu'à huit produits recommandés aléatoirement, éventuellement filtrés par catégorie.

Paramètres de requête

— `category` (string, optionnel ; nom d'énumération)

Réponses

— **200 OK** : tableau de produits.

— **400 Bad Request** : catégorie invalide.

— **500 Internal Server Error** : `"error": <string>`

2.5 GET /api/products/<product_id>

Obtenir un produit par identifiant.

Réponses

— **200 OK** : objet produit.

— **404 Not Found** : `"error": "Product not found"`

— **500 Internal Server Error** : `"error": <string>`

2.6 PUT /api/products/<product_id>

Mettre à jour un produit existant.

Réponses

— **200 OK** : `"message": "Product updated", "id": <number>`

— **400 Bad Request** : champs invalides ou manquants.

— **404 Not Found** : `"error": "Product not found"`

— **500 Internal Server Error** : `"error": <string>`

2.7 DELETE /api/products/<product_id>

Supprimer un produit.

Réponses

— **200 OK** : `"message": "Product deleted"`

— **404 Not Found** : `"error": "Product not found"`

— **500 Internal Server Error** : `"error": <string>`

3 Offres – /api/offers

Module : `backend/app/routes/offers_routes.py`

3.1 GET /api/offers/

Obtenir toutes les offres *actives* avec détails produit.

Réponses

— **200 OK** : tableau d'offres enrichies (champ produit : nom, prix, images, couleurs, catégorie, etc.).

— **500 Internal Server Error** : `"error": <string>`

3.2 GET /api/offers/all

Obtenir toutes les offres, actives ou non.

3.3 GET /api/offers/<offer_id>

Obtenir une offre par identifiant.

Réponses

— **200 OK** : offre avec détails produit.
— **404 Not Found** : `"error": "Offer not found"`
— **500 Internal Server Error** : `"error": <string>`

3.4 POST /api/offers/

Créer une nouvelle offre promotionnelle.

Corps de requête (JSON)

— `product_id` (number, requis ; produit existant)
— `title` (string, requis)
— `discount_percentage` (number, requis ; entre 0 et 100)
— `start_date` (string, ISO-8601, requis)
— `end_date` (string, ISO-8601, requis)

Réponses

— **201 Created** : offre créée.
— **400 Bad Request** : remise invalide ou champs manquants.
— **404 Not Found** : `"error": "Product not found"`
— **500 Internal Server Error** : `"error": <string>`

3.5 PUT /api/offers/<offer_id>

Mettre à jour une offre (tous les champs optionnels).

Réponses

— **200 OK** : offre mise à jour.
— **400 Bad Request** : remise invalide.
— **404 Not Found** : `"error": "Offer not found"`
— **500 Internal Server Error** : `"error": <string>`

3.6 DELETE /api/offers/<offer_id>

Supprimer une offre.

Réponses

— **200 OK** : `"message": "Offer deleted"`
— **404 Not Found** : `"error": "Offer not found"`
— **500 Internal Server Error** : `"error": <string>`

4 Commandes – /api/orders

Module : `backend/app/routes/order_routes.py`

4.1 POST /api/orders

Créer une nouvelle commande avec ses lignes.

Corps de requête (JSON)

— `utilisateur_id` (number, requis)
— `items` (tableau, requis) : pour chaque ligne, `product_id`, `quantity` (>0), `size`, `color`.
— `status` (string, optionnel ; par défaut `"Pending"`).

Réponses

— **201 Created** : `"message": "Order created", "order_id": <number>, "status": <string>`.
— **400 Bad Request** : erreurs de validation ou produit introuvable.

4.2 GET /api/orders

Lister toutes les commandes (sans les lignes de détail).

Réponses

— **200 OK** : tableau de `Order.to_dict()`.
— **500 Internal Server Error** : `"error": <string>`

4.3 GET /api/orders/<order_id>

Récupérer une commande avec ses lignes et les noms des produits.

Réponses

— **200 OK** : objet commande incluant la liste `items`.
— **404 Not Found** : `"error": "Order not found"`
— **500 Internal Server Error** : `"error": <string>`

4.4 GET /api/orders/user/<user_id>

Récupérer toutes les commandes pour un utilisateur donné.

Réponses

— **200 OK** : tableau d'objets commande.
— **500 Internal Server Error** : `"error": <string>`

4.5 PUT /api/orders/<order_id>

Mettre à jour les lignes d'une commande et, éventuellement, son statut.

Réponses

— **200 OK** : `"message": "Order updated", "order_id": <number>`.
— **400 Bad Request** : erreurs de validation.
— **404 Not Found** : `"error": "Order not found"`

4.6 DELETE /api/orders/<order_id>

Supprimer une commande (les lignes associées sont supprimées par cascade).

Réponses

— **200 OK** : `"message": "Order deleted"`
— **404 Not Found** : `"error": "Order not found"`
— **400 Bad Request** : `"error": <string>`

5 Blogs – /api/blogs

Module : `backend/app/routes/blogs_routes.py`

5.1 GET /api/blogs/

Lister tous les articles de blog, du plus récent au plus ancien.

Réponses

— **200 OK** : tableau de `Blog.to_dict()`.
— **500 Internal Server Error** : `"error": <string>`

5.2 GET /api/blogs/<blog_id>

Obtenir un article par identifiant.

Réponses

— **200 OK** : objet blog.
— **404 Not Found** : `"error": "Blog not found"`
— **500 Internal Server Error** : `"error": <string>`

5.3 POST /api/blogs/

Créer un article de blog.

Corps de requête (JSON)

— `title` (string, requis)
— `content` (string, requis)
— `image_url` (string, optionnel)
— `is_published` (booléen ou string, optionnel ; true par défaut)

Réponses

— **201 Created** : `"message": "Blog created successfully"`
— **400 Bad Request** : JSON invalide ou champs manquants.
— **500 Internal Server Error** : `"error": <string>`

5.4 PUT /api/blogs/<blog_id>

Mettre à jour un article existant.

Réponses

— **200 OK** : blog mis à jour.
— **404 Not Found** : `"error": "Blog not found"`
— **500 Internal Server Error** : `"error": <string>`

5.5 DELETE /api/blogs/<blog_id>

Supprimer un article.

Réponses

— **200 OK** : `"message": "Blog deleted"`
— **404 Not Found** : `"error": "Blog not found"`
— **500 Internal Server Error** : `"error": <string>`

6 Téléversement de fichiers – /api/upload

Module : `backend/app/routes/upload_routes.py`

6.1 POST /api/upload/image

Téléverser une seule image sur Cloudinary.

Requête (multipart/form-data)

— `file` (fichier, requis)
— `folder` (string, optionnel ; défaut `"ecommerce"`)

Réponses

— **200 OK** : JSON contenant `url`, `public_id`, format et dimensions.
— **400 Bad Request** : par exemple absence de fichier.

6.2 POST /api/upload/images

Téléverser plusieurs images.

Requête (multipart/form-data)

— `files` (liste de fichiers, requis)
— `folder` (string, optionnel ; défaut `"ecommerce"`)

Réponses

— **200 OK** : résumé du nombre d'images téléversées / échouées lorsque au moins une réussit.
— **400 Bad Request** : tous les téléversements ont échoué ou données invalides.

7 Chat IA – /api/opnai

Module : `backend/app/routes/opnai_routes.py`

7.1 POST /api/opnai/chat

Transmettre une conversation à l'API OpenAI avec un contexte catalogue.

Environnement

— `OPENAI_API_KEY` (requis ; sinon erreur 501).

— `OPENAI_MODEL` (optionnel ; défaut `gpt-4o-mini`).

Corps de requête (JSON)

— Soit `"prompt": <string>` (mode simple), soit `"messages": [{role, content}, ...]` (mode avancé).

— Paramètres optionnels : `temperature` (float, défaut 0,7), `max_tokens` (int, défaut 512).

Réponses

— **200 OK** : objet contenant la réponse texte de l'assistant et la réponse brute OpenAI.

— **400 Bad Request** : ni `prompt` ni `messages` fournis.

— **501 Not Implemented** : clé API absente.

— **502 Bad Gateway** : erreur renvoyée par l'API OpenAI.

— **500 Internal Server Error** : `"error": <string>`

8 Panneau d'administration – /admin

Module : `backend/app/admin/main.py`

Le panneau d'administration est principalement HTML (templates Jinja) mais repose sur les mêmes modèles et API.

— Authentification d'un administrateur via `ADMIN_EMAIL` et `ADMIN_PASSWORD` (variables d'environnement).

— Routes : formulaire de connexion (`/admin/login`), déconnexion (`/admin/logout`), tableau de bord (`/admin/`) et vues de gestion pour produits, utilisateurs, commandes, offres et blogs.

Ces pages consomment ou complètent les API JSON documentées ci-dessus.