École Nationale Supérieure
d'Informatique et d'Analyse des Systèmes

Ingénierie Intelligence Artificielle - 2IA

Web-Based Smart Grocery Box Interface for Catalog Management and Transactions

Élèves :
SOSSEY Salmane, SALHI Aymane
ABAYAD Mehdi, SAOUD Omar
ROCHDI Zakaria

Enseignant :
Mr. S. Ohamouddou

2025-2026

Contents

Chapter 1

Introduction et Analyse des besoins

Le projet **Smart Grocery Box Web App** est une application web conçue pour faciliter l'utilisation d'un système de type *Smart Grocery Box*. L'objectif est de proposer une interface simple, réactive et fiable permettant de :

- gérer un **catalogue de produits** (CRUD),
- constituer un **panier** et modifier les quantités,
- effectuer un **checkout** (validation du panier) pour créer une transaction,
- consulter un **historique des transactions**,
- proposer un **mode Kiosk** (QR code + plein écran) comme élément d'innovation minimal.

Le périmètre du projet est volontairement minimal et respecte les exigences essentielles du cahier des charges : **gestion complète des données**, **UI ergonomique**, **validation robuste**, **gestion d'erreurs**, **architecture claire** et **bonnes pratiques de sécurité web**. Dépôt GitHub : https://github.com/Eymeee/Web_app

1.1 Problématique

Dans un contexte magasin/borne (ou assistant d'achat), l'utilisateur doit pouvoir sélectionner rapidement des articles, ajuster les quantités, puis finaliser l'achat. Sans application, l'usage du système reste incomplet : il manque un parcours clair et une persistance fiable.

1.2 Exigences fonctionnelles (essentielles)

1. **CRUD Produits** : créer, consulter, modifier, supprimer des produits (nom, prix, SKU optionnel).
2. **Panier** : ajouter un produit avec quantité, modifier quantité, supprimer un item.
3. **Checkout** : transformer le panier en transaction enregistrée, puis vider le panier.

4. **Transactions** : lister les transactions et consulter le détail d'une transaction.
5. **Innovation minimale** : mode Kiosk (QR code + plein écran) pour accès rapide.

1.3 Exigences non-fonctionnelles

1. **UI réactive et ergonomique** (mobile-first, navigation claire).
2. **Validation robuste** côté client et serveur.
3. **Gestion d'erreurs** cohérente et compréhensible (messages + codes).
4. **Architecture claire** (séparation UI / API / logique / données).
5. **Sécurité web de base** : validation serveur systématique, erreurs sûres, pas de secrets en dur.

1.4 User stories

1. Consulter le catalogue pour choisir des produits.
2. Ajouter des produits au panier avec une quantité.
3. Ajuster les quantités ou supprimer des items.
4. Valider le panier pour enregistrer une transaction.
5. Consulter l'historique et le détail des transactions.
6. Accéder à l'app via un QR code (mode Kiosk).

Chapter 2

Conception

2.1 Architecture générale

L'application suit une architecture par couches :

- **Couche Présentation (UI)** : pages et composants (Dashboard, Products, Cart, Transactions, Kiosk).
- **Couche API** : endpoints internes (routes Next.js) pour le CRUD et les opérations panier/checkout.
- **Couche Logique Métier** : règles panier et checkout (calcul total, création transaction, vidage panier).
- **Couche Données** : base SQLite via ORM (Prisma), migrations et seed.

2.2 Choix technologiques

Les technologies retenues répondent aux objectifs de rapidité, maintenabilité et compatibilité locale :

- **Next.js 14 + TypeScript** : framework web moderne full-stack.
- **Tailwind CSS + shadcn/ui** : UI cohérente, composantisée, productive.
- **Prisma + SQLite** : persistance locale, migrations, seed.
- **Zod (validation)** : cohérence client/serveur.

2.3 Modèle de données

Le modèle est centré sur 4 entités principales :

Entité	Champs principaux	Rôle
Product	id, name, price, sku (optionnel), createdAt, updatedAt	Catalogue produits
CartItem	id, productId, quantity, createdAt, updatedAt	Panier courant
Transaction	id, total, createdAt	Achat finalisé
TransactionItem	id, transactionId, productId, quantity, unitPrice, lineTotal	Détails transaction

2.4 Cas d'usage (texte)

Les cas d'usage principaux sont :

- **Utilisateur** : consulter catalogue, gérer panier, checkout, consulter transactions, utiliser mode Kiosk.
- **Administrateur** : CRUD sur produits.

2.5 Scénario de séquence : Checkout (description)

1. L'utilisateur clique sur **Checkout** dans la page Panier.
2. L'API lit les **CartItem** et récupère les prix des **Product**.
3. L'API calcule les **lineTotal** et le **total**.
4. L'API crée une **Transaction** et ses **TransactionItem**.
5. L'API vide le panier (suppression des CartItem).
6. L'UI affiche un message de succès et redirige vers l'historique.

Chapter 3

Réalisation (Implémentation)

3.1 Fonctionnalités réalisées

3.1.1 Catalogue produits (CRUD)

La page `/products` permet de :

- afficher la liste des produits,
- ajouter un produit (nom, prix, SKU optionnel),
- modifier un produit,
- supprimer un produit.

Les données sont validées (nom obligatoire, prix strictement positif).

3.1.2 Panier

La page `/cart` permet de :

- ajouter un produit au panier avec une quantité,
- modifier la quantité (quantité ≥ 1),
- supprimer un item,
- afficher le total.

3.1.3 Checkout et Transactions

Le checkout :

- enregistre une transaction avec ses lignes (produit, quantité, prix unitaire, total ligne),
- vide le panier après succès,
- gère les cas limites (panier vide).

La page `/transactions` liste les transactions et une page de détail affiche les items.

3.1.4 Pré-remplissage du catalogue (seed)

Pour éviter un catalogue vide au premier lancement, un **seed** initialise une liste de produits de supermarché (produits courants avec prix). Cela permet à l'utilisateur de tester immédiatement : *catalogue* → *panier* → *checkout*.

3.2 API interne (endpoints)

Les routes API suivent un style REST :

- `GET/POST /api/products`
- `GET/PUT/DELETE /api/products/:id`
- `GET/POST /api/cart`
- `PUT/DELETE /api/cart/:id`
- `POST /api/checkout`
- `GET /api/transactions`
- `GET /api/transactions/:id`

3.3 Validation et gestion d'erreurs

- Validation côté client (formulaires) et côté serveur (API).
- Messages d'erreurs compréhensibles côté UI.
- Réponses API cohérentes pour faciliter le debug et la robustesse.

Chapter 4

Innovation et Tests

4.1 Innovation

L'innovation minimale demandée est satisfaite via le **Mode Kiosk** :

- affichage d'un **QR code** pointant vers l'application (accès mobile rapide),
- bouton **plein écran** pour usage sur tablette/borne,
- partage du lien (copie).

Ce choix est simple, utile, et adapté à un contexte magasin (borne d'accès ou caisse).

4.2 Scénario de test manuel

1. Lancer l'application et vérifier l'affichage du catalogue sur `/products`.
2. Ajouter 2 ou 3 produits au panier sur `/cart`.
3. Modifier une quantité et supprimer un item.
4. Cliquer sur **Checkout** et vérifier que le panier devient vide.
5. Ouvrir `/transactions` et vérifier l'apparition de la transaction.
6. Ouvrir le détail d'une transaction et vérifier les lignes et total.
7. Tester `/kiosk` : QR code visible + plein écran.

4.3 Sécurité de base

- Validation serveur systématique sur les routes d'écriture.
- Erreurs sûres (pas d'informations sensibles affichées).
- Persistance via ORM (Prisma) et schéma contrôlé.

Chapter 5

Exécution et Conclusion

5.1 Guide d'exécution

1. `npm install`
2. `npm run db:generate`
3. `npm run db:migrate`
4. `npm run db:seed`
5. `npm run dev` puis ouvrir `http://localhost:3000`

5.2 Conclusion

L'application **Smart Grocery Box Web App** répond aux exigences essentielles du projet : CRUD complet, UI réactive, validation robuste, gestion d'erreurs, architecture claire, et une innovation minimale via le mode Kiosk. Le périmètre reste volontairement minimal afin de respecter le cahier des charges sans ajouter de fonctionnalités optionnelles.

Dépôt GitHub : https://github.com/Eymeee/Web_app