**Cours : Projet Développement Web et Mobile (Master)**

**1️⃣ Objectifs du module**

À la fin du cours, l’étudiant sera capable de :

* Concevoir une application web et mobile complète (full-stack).
* Développer et intégrer **API REST** ou **GraphQL**.
* Déployer une application web et mobile en production.
* Travailler en équipe avec **Git** et une **méthodologie agile**.
* Présenter un projet final complet : **Web + Mobile + Documentation**.

**2️⃣ Contenu détaillé (Plan du cours)**

**Semaine 1 : Introduction et cadrage**

* **Contenu :**
  + Présentation du module et des technologies (Web + Mobile).
  + Panorama des architectures : Monolithique, API REST, Microservices.
  + Introduction à la méthodologie Agile (Scrum).
* **TP :**
  + Création d’un dépôt GitHub/GitLab par équipe.
  + Définition des sujets de projet (application fil rouge).

**Semaine 2 : Analyse et conception du projet**

* **Contenu :**
  + Analyse des besoins et rédaction du **cahier des charges**.
  + Introduction à UML : cas d’utilisation, classes, séquences.
  + Maquettage UI/UX avec Figma ou Adobe XD.
* **TP :**
  + Réalisation d’une maquette interactive de l’application.
  + Rédaction du backlog produit.

**Semaine 3 : Mise en place du back-end**

* **Contenu :**
  + Introduction à **Node.js + Express** ou **Laravel/PHP**.
  + Création d’une API REST connectée à MySQL/PostgreSQL/MongoDB.
  + Introduction à Postman pour tester l’API.
* **TP :**
  + Développer l’API de gestion des utilisateurs et authentification (JWT).

**Semaine 4 : Front-end Web – Fondamentaux**

* **Contenu :**
  + Introduction à **Angular** ou **React.js**.
  + Composants, routes, gestion des états et formulaires.
  + Consommation d’API REST avec Axios/Fetch.
* **TP :**
  + Créer la page de connexion et la page d’accueil connectée à l’API.

**Semaine 5 : Front-end Web – Approfondissement**

* **Contenu :**
  + Gestion des rôles et permissions côté front.
  + Gestion des erreurs et notifications.
  + Structure modulaire du projet Web.
* **TP :**
  + Créer une interface pour afficher/éditer les données de l’API.

**Semaine 6 : Développement Mobile – Fondamentaux**

* **Contenu :**
  + Présentation de **Flutter** ou **React Native**.
  + Création d’écrans mobiles et navigation.
  + Interaction avec l’API REST (GET/POST).
* **TP :**
  + Créer une application mobile simple qui se connecte à l’API.

**Semaine 7 : Développement Mobile – Fonctions avancées**

* **Contenu :**
  + Stockage local (SQLite ou SharedPreferences).
  + Notifications push (Firebase Cloud Messaging).
  + Design responsive et bonnes pratiques UX mobile.
* **TP :**
  + Créer un module mobile complet (ex: gestion de profil).

**Semaine 8 : Gestion de projet et intégration**

* **Contenu :**
  + Utilisation avancée de **Git et GitHub/GitLab** (branches, merge, PR).
  + Introduction à **CI/CD** (GitHub Actions, GitLab CI).
  + Débogage et logs côté serveur et mobile.
* **TP :**
  + Mettre en place un workflow Git complet pour le projet.

**Semaine 9 : Sécurité et optimisation**

* **Contenu :**
  + Sécurisation des API (JWT, CORS, HTTPS).
  + Bonnes pratiques de gestion des mots de passe et tokens.
  + Optimisation des performances Web et Mobile.
* **TP :**
  + Implémenter la sécurisation des routes API et test des accès.

**Semaine 10 : Déploiement et mise en production**

* **Contenu :**
  + Déploiement Web sur **Heroku, Vercel ou serveur Linux**.
  + Génération d’APK mobile (Android) et test sur émulateur ou smartphone.
  + Notions sur Docker pour déploiement rapide.
* **TP :**
  + Déployer l’application Web + générer le premier APK mobile.

**Semaine 11-12 : Finalisation et soutenance**

* **Contenu :**
  + Revue des livrables : Code, Documentation, Vidéo Démo.
  + Préparation à la soutenance finale.
* **TP :**
  + Présentation de l’application Web & Mobile + démonstration live.

PROJET : Comparateur de prix

1. Introduction

1.1 Contexte

Les consommateurs recherchent des solutions pour identifier rapidement les meilleures offres sur les produits qu’ils souhaitent acheter. Les variations de prix entre différents magasins, en ligne ou physiques, rendent cette tâche complexe. Grâce à la popularisation des smartphones et des technologies de lecture de codes-barres, il est possible de collecter des données de prix en temps réel pour offrir une solution collaborative et géolocalisée.

1.2 Objectifs du projet

* Développer une application mobile et web permettant aux utilisateurs de scanner les codes-barres des produits pour collecter et partager les prix.
* Construire une base de données géolocalisée centralisant les informations sur les prix des produits dans différents magasins.
* Fournir des visualisations graphiques de l’évolution des prix par magasin et par produit.
* Permettre une comparaison rapide des prix pour un même produit entre différents points de vente.

1.3 Portée

* Une application mobile native ou hybride pour iOS et Android.
* Une base de données collaborative alimentée par les contributions des utilisateurs.
* Une interface utilisateur intuitive avec des fonctionnalités de recherche, de comparaison et de visualisation.
* Une couverture géographique initiale limitée à une région pilote (par exemple, une grande ville ou un pays) avec possibilité d’extension.

2. Méthodologie Scrum

2.1 Organisation de l’équipe

* Product Owner : Responsable de la vision du produit, de la priorisation du backlog et de la validation des livrables.
* Scrum Master : Facilite les cérémonies Scrum, élimine les obstacles et assure le respect de la méthodologie.
* Équipe de développement : Développeurs full-stack, designers UX/UI, testeurs QA, et administrateur de base de données.
* Parties prenantes : Utilisateurs finaux, partenaires commerciaux (magasins, plateformes e-commerce), équipe marketing.

2.2 Cérémonies Scrum

* Sprint Planning (2 heures par sprint) : Définition des objectifs du sprint et sélection des user stories du backlog.
* Daily Scrum (15 minutes/jour) : Point quotidien pour synchroniser l’équipe.
* Sprint Review (2 heures par sprint) : Présentation des livrables aux parties prenantes et collecte des retours.
* Sprint Retrospective (1 heure par sprint) : Analyse des points forts et des axes d’amélioration.
* Durée des sprints : 2 semaines.

2.3 Artefacts Scrum

* Product Backlog : Liste priorisée des fonctionnalités et exigences (détaillée ci-dessous).
* Sprint Backlog : Sous-ensemble du product backlog sélectionné pour chaque sprint.
* Increment : Version fonctionnelle du produit livrée à la fin de chaque sprint.

3. Exigences fonctionnelles3.1 Fonctionnalités principales

1. Scan de codes-barres :
   * Les utilisateurs peuvent scanner les codes-barres des produits via l’appareil photo de leur smartphone.
   * Reconnaissance des codes-barres (EAN, UPC, QR codes).
   * Liaison du code-barre à une fiche produit (nom, marque, description).
2. Ajout de prix :
   * Les utilisateurs saisissent manuellement ou via scan le prix d’un produit dans un magasin spécifique.
   * Géolocalisation automatique du magasin ou saisie manuelle de l’adresse.
3. Base de données collaborative :
   * Stockage des informations sur les produits (code-barre, nom, marque, prix, magasin, date, localisation).
   * Validation des contributions pour éviter les doublons ou les données erronées.
4. Comparaison des prix :
   * Recherche d’un produit par nom, catégorie ou code-barre.
   * Affichage des prix actuels et historiques pour un produit dans différents magasins géolocalisés.
5. Visualisation graphique :
   * Graphiques montrant l’évolution des prix dans le temps pour un produit donné.
   * Comparaison des prix entre magasins sous forme de graphiques ou tableaux.
6. Notifications :
   * Alertes personnalisées lorsque le prix d’un produit suivi baisse ou atteint un seuil défini par l’utilisateur.

3.2 Fonctionnalités secondaires

1. Compte utilisateur :
   * Inscription et connexion via email, réseaux sociaux ou numéro de téléphone.
   * Gestion de profil (préférences, produits favoris, historique des scans).
2. Mode hors-ligne :
   * Possibilité de scanner et enregistrer des données hors connexion, avec synchronisation une fois connecté.
3. Partage communautaire :
   * Partage des prix trouvés via réseaux sociaux ou messageries.
   * Système de gamification (points, badges) pour encourager les contributions.
4. Filtres avancés :
   * Filtrage des résultats par distance, type de magasin (supermarché, boutique spécialisée, e-commerce), ou fourchette de prix.

3.3 Exigences non fonctionnelles

* Performance : Temps de réponse de l’application < 2 secondes pour le scan et la recherche.
* Sécurité : Protection des données personnelles (RGPD-compliant), chiffrement des données sensibles.
* Scalabilité : Base de données capable de gérer des milliers de contributions quotidiennes.
* Compatibilité : iOS 14+ et Android 10+.
* Accessibilité : Interface conforme aux normes WCAG 2.1 pour les utilisateurs en situation de handicap.
* Multilingue : Support initial du français, avec possibilité d’ajouter d’autres langues.

1. Product Backlog

Le backlog est structuré en épics, décomposés en user stories avec des critères d’acceptation. Les priorités sont indiquées (Haute, Moyenne, Basse) et les estimations d’effort sont exprimées en story points (échelle de Fibonacci : 1, 2, 3, 5, 8, 13, etc.).

Épic 1 : Scan et ajout de prix

5. Architecture technique

5.1 Stack technologique

* Frontend : Framework hybride (React Native ou Flutter) pour iOS et Android.
* Backend : Node.js avec Express ou Django pour l’API RESTful.
* Base de données : MongoDB (NoSQL) pour la flexibilité des données géolocalisées ou PostgreSQL avec PostGIS pour les fonctionnalités spatiales.
* Hébergement : Cloud AWS/GCP pour scalabilité, avec S3 pour stockage des images.
* APIs externes :
  + API de reconnaissance de codes-barres (ex. : ZXing ou service tiers).
  + API de géolocalisation (Google Maps ou OpenStreetMap).
* Outils DevOps : Docker pour conteneurisation, CI/CD avec GitHub Actions.

5.2 Infrastructure

* Serveur API pour gérer les requêtes utilisateurs et contributions.
* Base de données géospatiale pour stocker les informations sur les prix et les magasins.
* Système de cache (Redis) pour accélérer les recherches fréquentes.
* Système de monitoring (Prometheus, Grafana) pour suivre les performances.

6. Planification initiale

6.1 Sprints prévus

* Sprint 1-2 : Base de l’application (4 semaines)
  + US1.1, US1.2, US6.1 : Scan de codes-barres, ajout de prix, inscription/connexion.
  + Mise en place de l’API et de la base de données.
* Sprint 3-4 : Recherche et comparaison (4 semaines)
  + US3.1, US3.2, US2.2 : Recherche de produits, comparaison des prix, stockage des contributions.
* Sprint 5-6 : Visualisation et notifications (4 semaines)
  + US4.1, US4.2, US5.1 : Graphiques d’évolution des prix, comparaisons visuelles, notifications.
* Sprint 7-8 : Fonctionnalités secondaires et tests (4 semaines)
  + US1.3, US1.4, US2.3, US5.2, US6.2 : Géolocalisation, mode hors-ligne, gamification, favoris.
  + Tests d’intégration et validation UX.

6.2 Jalons

* Mois 1 : Prototype fonctionnel (scan, ajout de prix, inscription).
* Mois 2 : MVP avec comparaison des prix et base de données collaborative.
* Mois 3 : Visualisations graphiques et notifications.
* Mois 4 : Lancement bêta avec fonctionnalités secondaires et tests utilisateurs.

7. Contraintes et risques

7.1 Contraintes

* Budget : Limité aux ressources allouées pour une équipe de 5-7 personnes sur 4 mois.
* Délais : Lancement bêta prévu dans 4 mois.
* Qualité des données : Dépendance aux contributions des utilisateurs, risque de données erronées.

7.2 Risques

* Risque technique : Problèmes d’intégration avec APIs externes (codes-barres, géolocalisation).
  + Mitigation : Tests unitaires et choix d’APIs fiables.
* Risque utilisateur : Adoption lente si l’interface n’est pas intuitive.
  + Mitigation : Tests UX réguliers avec utilisateurs cibles.
* Risque légal : Conformité RGPD pour la collecte de données géolocalisées.
  + Mitigation : Audit RGPD dès le sprint 1.

8. Livrables

* Application mobile : Versions iOS et Android disponibles sur les stores.
* Documentation : Guide utilisateur, documentation technique pour les développeurs.
* Rapports : Rapports de sprint, tests QA, et retours utilisateurs post-bêta.
* Base de données : Initialement peuplée avec des données tests et contributions pilotes

1. Wireframes de l’interface

Les wireframes proposés sont des esquisses low-fidelity pour illustrer la structure et la navigation de l’application mobile. Ils couvrent les écrans principaux pour répondre aux fonctionnalités clés (scan, recherche, comparaison, visualisation). Les wireframes sont décrits textuellement pour plus de clarté, mais ils peuvent être traduits en maquettes graphiques (par exemple, via Figma) si besoin.1.1 Écran d’accueil

* Objectif : Point d’entrée avec accès rapide aux fonctionnalités principales.
* Éléments :
  + Barre de navigation supérieure : Logo, icône de profil, icône de recherche.
  + Bouton principal : « Scanner un produit » (grand bouton central).
  + Section « Produits récents » : Liste des derniers produits scannés ou recherchés (nom, image, meilleur prix).
  + Section « Favoris » : Aperçu des produits suivis avec icône de notification si prix modifié.
  + Pied de page : Navigation (Accueil, Recherche, Comparaison, Profil).

1.2 Écran de scan de code-barre

* Objectif : Permettre à l’utilisateur de scanner un produit et d’ajouter un prix.
* Éléments :
  + Zone de scan : Fenêtre pour l’appareil photo avec cadre pour aligner le code-barre.
  + Bouton « Saisie manuelle » : Alternative si le scan échoue.
  + Après scan : Affichage de la fiche produit (nom, marque, image si disponible).
  + Formulaire : Champ pour le prix, sélection du magasin (géolocalisé ou manuel), bouton « Soumettre ».

1.4 Écran de visualisation des prix

* Objectif : Afficher l’historique des prix et une comparaison graphique.
* Éléments :
  + Fiche produit : Nom, marque, image, catégorie.
  + Graphique linéaire : Évolution du prix sur 1 mois/3 mois/6 mois.
  + Tableau comparatif : Liste des magasins, prix, date de soumission.
  + Bouton « Ajouter aux favoris » et « Configurer alerte prix ».

1.5 Écran de profil

* Objectif : Gérer les informations de l’utilisateur et ses contributions.
* Éléments :
  + Détails du profil : Nom, email, points de gamification, badges.
  + Liste des contributions : Produits ajoutés, statut (validé/rejeté).
  + Liste des favoris : Produits suivis avec notifications actives.
  + Paramètres : Notifications, langue, déconnexion.

2. Détails techniques

2.1 Architecture globale

* Frontend : React Native pour une application multiplateforme (iOS et Android). Utilisation de TypeScript pour une meilleure maintenabilité.
* Backend : Node.js avec Express pour une API REST légère et performante. Alternative : Django avec Django REST Framework pour une gestion robuste des modèles.
* Base de données :
  + MongoDB : Choisi pour sa flexibilité avec les données non structurées (produits, prix, géolocalisation).
  + Indexes géospatiaux : Pour des requêtes rapides sur les données de localisation (ex. : magasins à moins de 5 km).
* Stockage des images : AWS S3 pour héberger les images des produits (téléchargées par les utilisateurs ou récupérées via APIs externes).
* Cache : Redis pour mettre en cache les recherches fréquentes (ex. : produits populaires).
* Authentification : JWT (JSON Web Tokens) pour sécuriser les sessions utilisateur. Intégration OAuthმო�

System: OAuth 2.0 pour l’inscription/connexion via Google ou Apple (SSO).

* APIs externes :
  + ZXing ou QuaggaJS pour la reconnaissance des codes-barres.
  + Google Maps API ou OpenStreetMap pour la géolocalisation et le calcul des distances.
  + Cloudinary ou AWS Rekognition pour la reconnaissance d’images (optionnel, pour extraire des informations produit depuis des photos si le scan échoue).
* Infrastructure Cloud :
  + AWS ECS (Elastic Container Service) pour déployer l’API.
  + AWS RDS (si PostgreSQL est choisi) ou MongoDB Atlas pour la base de données.
  + AWS Lambda pour des tâches asynchrones (ex. : validation des contributions).
* Sécurité :
  + Chiffrement HTTPS/TLS pour toutes les communications.
  + Conformité RGPD : Consentement explicite pour la géolocalisation, anonymisation des données sensibles.
  + OWASP Top 10 protections (SQL injection, XSS, etc.).
* Performances :
  + Optimisation des requêtes MongoDB avec indexes sur barcode, location, et date.
  + CDN (CloudFront) pour les assets statiques (images, JS, CSS).
  + Mise en cache des résultats de recherche populaires (TTL de 24h dans Redis).

2.2 Flux de données

1. Scan de code-barre :
   * L’utilisateur scanne un code-barre → requête API ZXing → récupération des données produit.
   * Données produit affichées → utilisateur saisit le prix et le magasin → envoi à l’API backend.
   * Backend valide les données (règles de validation : prix positif, magasin géolocalisé valide) → stockage dans MongoDB.
2. Recherche et comparaison :
   * Requête utilisateur (nom ou code-barre) → recherche dans MongoDB avec index plein texte.
   * Résultats triés par prix ou distance → renvoyés au frontend avec pagination.
3. Visualisation :
   * Requête historique des prix pour un produit spécifique → agrégation pour générer le graphique.
   * Comparaison multi-magasins → jointure sur store et tri par prix.
4. Notifications :
   * Tite tâche planifiée (AWS Lambda) vérifie les changements de prix toutes les 24h.
   * Envoi de notifications push via Firebase Cloud Messaging.

3. Plan de test3.1 Types de tests

1. Tests unitaires :
   * Couverture : API endpoints, fonctions de validation des contributions, calculs de distance géospatiale.
   * Outils : Jest (pour Node.js) ou Pytest (pour Django).
   * Ex-Exemple : Tester la fonction de scan de code-barre.
     + Entrée : Code-barre valide → Résultat attendu : Fiche produit correcte.
     + Entrée : Code-barre invalide → Résultat attendu : Message d’erreur clair.
2. Tests d’intégration :
   * Scénarios : Scan → ajout de prix → recherche → comparaison → visualisation.
   * Outils : Postman pour tester les endpoints API, Cypress pour les tests E2E de l’interface.
   * Exemple : Simuler un scan, vérifier que les données sont correctement stockées et récupérées.
3. Tests de performance :
   * Charge : 1000 requêtes/seconde sur l’API de recherche.
   * Latence : Temps de réponse < 2 secondes pour le scan et la recherche.
   * Outils : JMeter pour simuler la charge, New Relic pour monitoring.
4. Tests UX :
   * Scénarios : Navigation fluide, formulaires intuitifs, graphiques lisibles.
   * Méthode : Tests avec 10 utilisateurs cibles (5 iOS, 5 Android).
   * Critères : Temps pour scanner et ajouter un prix < 10 secondes, taux de satisfaction > 90%.
5. Tests de sécurité :
   * Vérification : Protection contre injections SQL, XSS, et fuite de données géolocalisées.
   * Outils : OWASP ZAP pour scans de vulnérabilités.
6. Tests de compatibilité :
   * Plates-formes : iOS 14+, Android 10+.
   * Résolution : 1080x1920, 750x1334 (iPhone), etc.
   * Outils : BrowserStack pour tests multi-appareils.

3.2 Planification des tests

* Sprint 1 :
  + Tests unitaires sur le scan et l’ajout de prix.
  + Tests d’intégration pour l’inscription/connexion.
* Sprint 2 :
  + Tests unitaires sur la recherche et la comparaison.
  + Tests UX sur l’écran de scan et de recherche.
* \*\* hungarian translation for "cahier des charges"\*\*

Étude MERISE : Comparateur de prix

1. Niveau ConceptuelLe niveau conceptuel décrit les besoins métier indépendamment des contraintes techniques. Il inclut le Modèle Conceptuel des Données (MCD) et le Modèle Conceptuel des Traitements (MCT).

1.1 Modèle Conceptuel des Données (MCD)Le MCD représente les entités, leurs attributs et les relations entre elles. Voici le MCD pour le projet :Entités principales :

1. Produit :
   * Attributs : code\_barre (clé primaire, unique), nom, marque, catégorie, image (URL optionnelle).
2. Magasin :
   * Attributs : id\_magasin (clé primaire), nom, adresse, coordonnees (latitude, longitude), type (supermarché, boutique, e-commerce).
3. Prix :
   * Attributs : id\_prix (clé primaire), montant, date\_soumission, code\_barre (clé étrangère vers Produit), id\_magasin (clé étrangère vers Magasin), id\_utilisateur (clé étrangère vers Utilisateur).
4. Utilisateur :
   * Attributs : id\_utilisateur (clé primaire), email, mot\_de\_passe (haché), nom, points\_gamification, date\_inscription.
5. Favori :
   * Attributs : id\_favori (clé primaire), id\_utilisateur (clé étrangère), code\_barre (clé étrangère).
6. Alerte :
   * Attributs : id\_alerte (clé primaire), id\_utilisateur (clé étrangère), code\_barre (clé étrangère), seuil\_prix, active (booléen).

Relations :

* Soumettre : Entre Utilisateur et Prix (un utilisateur soumet plusieurs prix, un prix est soumis par un seul utilisateur).
* Vendre : Entre Produit et Magasin via Prix (un produit peut avoir plusieurs prix dans différents magasins, un magasin vend plusieurs produits).
* Suivre : Entre Utilisateur et Produit via Favori (un utilisateur peut suivre plusieurs produits, un produit peut être suivi par plusieurs utilisateurs).
* Notifier : Entre Utilisateur et Produit via Alerte (un utilisateur peut configurer plusieurs alertes, un produit peut être lié à plusieurs alertes).

Règles de gestion :

* Un produit doit avoir un code-barre unique.
* Un prix est toujours associé à un produit, un magasin et un utilisateur.
* Un utilisateur peut avoir plusieurs favoris et alertes, mais chaque favori/alerte est lié à un seul produit.

1.2 Modèle Conceptuel des Traitements (MCT)Le MCT décrit les processus métier sans préciser leur implémentation. Voici les traitements principaux :

1. Scanner un produit :
   * Entrée : Code-barre scanné via l’appareil photo.
   * Traitement : Identifier le produit dans la base de données ou créer une nouvelle entrée si inconnu.
   * Sortie : Affichage de la fiche produit (nom, marque, catégorie).
2. Ajouter un prix :
   * Entrée : Code-barre, prix, magasin (géolocalisé ou manuel), utilisateur connecté.
   * Traitement : Valider les données (prix positif, magasin existant) et enregistrer dans la base.
   * Sortie : Confirmation de soumission, mise à jour des points de gamification.
3. Rechercher un produit :
   * Entrée : Nom, catégorie ou code-barre.
   * Traitement : Recherche dans la base avec autocomplétion, application des filtres (distance, type de magasin).
   * Sortie : Liste des produits avec prix et magasins associés.
4. Comparer les prix :
   * Entrée : Produit sélectionné, filtres (distance, type de magasin).
   * Traitement : Récupérer les prix associés au produit, trier par prix ou distance.
   * Sortie : Tableau ou graphique comparatif.
5. Visualiser l’évolution des prix :
   * Entrée : Produit, période (1 mois, 3 mois, 6 mois).
   * Traitement : Agréger les prix historiques, générer un graphique linéaire.
   * Sortie : Graphique d’évolution des prix.
6. Configurer une alerte :
   * Entrée : Produit, seuil de prix, utilisateur.
   * Traitement : Enregistrer l’alerte, vérifier périodiquement les prix.
   * Sortie : Notification push si le prix descend sous le seuil.

2. Niveau OrganisationnelLe niveau organisationnel précise l’organisation des flux et des traitements dans le contexte de l’entreprise et des utilisateurs. Il inclut le Modèle Organisationnel des Données (MOD) et le Modèle Organisationnel des Traitements (MOT).2.1 Modèle Organisationnel des Données (MOD)Le MOD adapte le MCD aux contraintes organisationnelles (par exemple, rôles des utilisateurs, validation des données).

* Rôles :
  + Utilisateur standard : Scanne, ajoute des prix, consulte et configure des alertes.
  + Administrateur : Valide les contributions, gère les signalements d’erreurs.
* Flux de données :
  + Les utilisateurs soumettent des prix → en attente de validation par un administrateur (sauf si utilisateur de confiance).
  + Les données validées sont accessibles à tous via recherche/comparaison.
  + Les données géolocalisées sont stockées avec une précision définie (100 m).
* Contraintes :
  + Validation manuelle ou automatique (via règles : prix dans une fourchette réaliste, magasin existant).
  + Archivage des prix obsolètes (> 6 mois) pour optimiser la base.

2.2 Modèle Organisationnel des Traitements (MOT)Le MOT décrit les processus en tenant compte des acteurs et des outils.Exemple de flux pour "Ajouter un prix" :

1. L’utilisateur scanne un code-barre.
2. L’application envoie une requête à l’API pour vérifier le produit.
3. Si produit inconnu, l’utilisateur saisit manuellement les informations (nom, marque).
4. L’utilisateur entre le prix et sélectionne un magasin (géolocalisé ou manuel).
5. L’API valide les données (contrôle des doublons, prix réaliste).
6. Les données sont stockées en attente de validation (administrateur ou algorithme).
7. Une fois validées, les données sont disponibles pour recherche/comparaison.

Rôles et responsabilités :

* Utilisateur : Fournit des données (prix, localisation).
* Administrateur : Vérifie la qualité des données, résout les signalements.
* Système : Gère les requêtes, stocke les données, génère des graphiques.

1. Niveau Opérationnel

Le niveau opérationnel traduit les modèles en implémentation technique. Il inclut le Modèle Logique des Données (MLD) et le Modèle Opérationnel des Traitements (MOT).

3.1 Modèle Logique des Données (MLD)Le MLD transforme le MCD en tables pour une base de données (ici, MongoDB pour une approche NoSQL flexible, mais décrit comme relationnel pour clarté).Tables :

1. Produit :
   * code\_barre (PK, string), nom (string), marque (string), categorie (string), image (string, nullable).
2. Magasin :
   * id\_magasin (PK, string), nom (string), adresse (string), coordonnees (GeoJSON: {type: "Point", coordinates: [lon, lat]}), type (string).
3. Prix :
   * id\_prix (PK, string), montant (float), date\_soumission (datetime), code\_barre (FK, string), id\_magasin (FK, string), id\_utilisateur (FK, string), valide (boolean).
4. Utilisateur :
   * id\_utilisateur (PK, string), email (string, unique), mot\_de\_passe (string), nom (string), points\_gamification (int), date\_inscription (datetime).
5. Favori :
   * id\_favori (PK, string), id\_utilisateur (FK, string), code\_barre (FK, string).
6. Alerte :
   * id\_alerte (PK, string), id\_utilisateur (FK, string), code\_barre (FK, string), seuil\_prix (float), active (boolean).

Indexes :

* Produit.code\_barre : Index unique pour recherche rapide.
* Magasin.coordonnees : Index géospatial pour requêtes de proximité.
* Prix.date\_soumission : Index pour trier les prix historiques.

3.2 Modèle Opérationnel des Traitements (MOT)

Le MOT opérationnel décrit les algorithmes et les outils techniques pour chaque traitement.

Exemple : Traitement "Ajouter un prix" :

1. Entrée : Code-barre (string), prix (float), magasin (id ou géolocalisation), utilisateur (id).
2. Algorithme :
   * Vérifier l’existence du produit (findOne({code\_barre}) dans MongoDB).
   * Si inexistant, créer un produit temporaire (en attente de validation).
   * Vérifier la validité du prix (ex. : 0 < prix < 10000).
   * Récupérer ou créer un magasin via géolocalisation (API Google Maps/OpenStreetMap).
   * Insérer le prix dans la collection Prix avec valide: false.
   * En file d’attente pour validation (Lambda déclenche une vérification automatique ou notifie un administrateur).
3. Outils :
   * Frontend : React Native (composant de scan avec ZXing).
   * Backend : Node.js/Express, endpoint POST /prix.
   * Base de données : MongoDB, collection Prix.
   * Validation : Règles automatiques (fourchette de prix, historique) + interface admin (React Admin).

Exemple : Traitement "Rechercher un produit" :

1. Entrée : Terme de recherche (string), filtres (distance, type de magasin).
2. Algorithme :
   * Recherche plein texte sur Produit.nom et Produit.code\_barre (index MongoDB).
   * Appliquer filtres : géolocalisation ($geoNear), type de magasin.
   * Joindre Prix pour récupérer les prix récents ($lookup).
   * Trier par prix ou distance.
3. Outils :
   * Frontend : Composant de recherche avec autocomplétion (React Native).
   * Backend : Endpoint GET /produits?query=...&filters=....
   * Cache : Redis pour stocker les résultats fréquents (TTL 24h).

4. Intégration avec ScrumL’étude MERISE s’intègre dans la méthodologie Scrum via :

* Product Backlog : Les entités et traitements du MCD/MCT sont traduits en user stories (voir backlog dans le cahier des charges initial).
* Sprints :
  + Sprint 1 : Mise en place du MLD (base MongoDB), API pour scan et ajout de prix.
  + Sprint 2 : Recherche et comparaison (endpoints API, requêtes géospatiales).
  + Sprint 3 : Visualisation (composants graphiques avec Chart.js).
  + Sprint 4 : Notifications et gamification (Firebase, algorithmes de scoring).
* Validation : Chaque sprint livre un incrément testé (tests unitaires, intégration, UX) basé sur les critères d’acceptation des user stories.

5. Contraintes et hypothèses

* Contraintes :
  + Base de données géolocalisée nécessitant des indexes performants pour les recherches de proximité.
  + Validation des contributions pour éviter les doublons ou erreurs (ex. : algorithme de détection des prix aberrants).
  + Conformité RGPD pour les données utilisateur (email, localisation).
* Hypothèses :
  + Les utilisateurs acceptent de partager leur localisation pour associer les prix aux magasins.
  + Les APIs externes (ZXing, Google Maps) sont fiables et disponibles.
  + Le volume initial de contributions permet de constituer une base de données pertinente.

6. Livrables MERISE

1. MCD : Schéma des entités et relations (voir section 1.1).
2. MCT : Liste des traitements métier avec flux (voir section 1.2).
3. MOD/MOT : Flux organisationnels et responsabilités (voir section 2).
4. MLD : Schéma de base de données NoSQL (voir section 3.1).
5. Documentation technique : Spécifications des endpoints API, algorithmes, et outils utilisés.
6. Wireframes : Intégrés dans le cahier des charges précédent pour l’interface utilisateur.