CAHIER DES CHARGES : STOCKFLOW – CLOUD

SOMMAIRE

1. INFORMATIONS GÉNÉRALES

1.1. Identification du projet

Nom du projet : StockFlow - Cloud
Date de création : 10/04/2025
Version du document : 1.0

1.2. Parties prenantes

1.2.1. Maître d'ouvrage (MOA)

Organisation : Gabriel B / Brahim H
Adresse : Lille
Représentant principal : Gabriel B / Brahim H
Fonction : Développeur
Email : 10143@holbertonstudents.com / 10151@holbertonstudents.com

1.3. Historique du document

Version	Date	Auteur	Modifications
1.0	[21/04/2025]	[B / H, Gabriel / Brahim]	Version initiale

1.4. Documents de référence

Liste des documents liés au projet :

- Diagramme UML
- Documentation technique

2. PRÉSENTATION DU PROJET

2.1. Contexte

Situation actuelle Dans le domaine de la gestion de stock, les entreprises font face à des défis croissants concernant l'optimisation de leurs inventaires et la gestion efficace de leur chaîne logistique. Notre projet s'inscrit dans cette dynamique, en cherchant à apporter une solution moderne et adaptée aux besoins actuels. Actuellement, de nombreuses solutions existantes présentent des limitations en termes d'interface utilisateur, de flexibilité ou d'intégration technologique.

Problématiques rencontrées La principale difficulté réside dans notre niveau de maîtrise initial des technologies modernes comme Express.js pour le backend, React pour l'interface utilisateur et Tailwind CSS pour le design. Ces technologies, bien que puissantes et largement adoptées dans l'industrie, représentent un défi d'apprentissage significatif. Par ailleurs, le développement d'une solution de gestion de stock complète nécessite une compréhension approfondie des processus logistiques et une architecture technique solide pour assurer fiabilité et performance.

Enjeux Les enjeux de ce projet sont doubles. D'un point de vue technique, il s'agit de développer une API fonctionnelle et évolutive, capable de gérer efficacement les opérations de stock. Sur le plan personnel et professionnel, l'enjeu est d'acquérir et de démontrer des compétences concrètes dans des technologies de pointe, valorisables sur le marché du travail. La non-réalisation du projet limiterait notre capacité à présenter des compétences techniques avancées à de potentiels employeurs ou clients.

Motivations L'intérêt personnel de longue date pour le domaine de la logistique constitue la motivation fondamentale de ce projet. La gestion de stock, élément crucial de toute chaîne logistique, offre un terrain d'application concret pour mettre en œuvre des solutions techniques innovantes. L'objectif principal est de développer un MVP (Minimum Viable Product) présentable dans notre portfolio, démontrant notre capacité à concevoir et implémenter une API fonctionnelle. Ce projet représente également une opportunité précieuse pour accroître nos compétences techniques et notre compréhension des processus métier liés à la gestion de stock.

Porté conjointement par nous-même et notre partenaire de travail Brahim H., ce projet s'inscrit dans une démarche d'apprentissage collaboratif et de développement professionnel, avec l'ambition de créer une solution qui pourrait éventuellement être utilisée dans un contexte réel ou servir de base à des développements futurs plus ambitieux.

2.2. Description générale

Le projet "Gestionnaire de Stock" est une application web complète visant à moderniser et simplifier la gestion d'inventaire pour les entreprises de toutes tailles. Cette solution se compose d'une API robuste développée avec Express.js, couplée à une interface utilisateur intuitive construite avec React et stylisée avec Tailwind CSS.

Notre application permet aux utilisateurs de suivre en temps réel leurs niveaux de stock, de gérer les entrées et sorties de marchandises, de catégoriser les produits, et de générer des rapports d'analyse. L'objectif principal est de fournir un outil qui centralise l'information, automatise les processus répétitifs et offre une visibilité claire sur l'état des stocks à tout moment.

Le périmètre du projet comprend :

- Une API REST sécurisée pour la gestion des données de stock

- Un système d'authentification et de gestion des droits utilisateurs
- Une interface de gestion des produits (ajout, modification, suppression)
- Un module de suivi des mouvements de stock
- Un système d'alerte pour les seuils minimum de stock
- Des fonctionnalités de reporting et d'analyse basiques

Dans sa version MVP (Minimum Viable Product), l'application se concentrera sur les fonctionnalités essentielles permettant une gestion efficace des stocks, tout en démontrant notre maîtrise des technologies utilisées. L'architecture sera conçue pour permettre des évolutions futures, comme l'intégration avec d'autres systèmes (ERP, e-commerce) ou l'ajout de fonctionnalités avancées (prévisions, optimisation automatique des niveaux de stock).

Ce projet s'adresse principalement aux petites et moyennes entreprises ayant besoin d'une solution flexible de gestion d'inventaire, ainsi qu'aux professionnels de la logistique cherchant à optimiser leurs processus. Il représente également une vitrine de nos compétences techniques en développement web moderne pour notre portfolio professionnel.

2.3. Périmètre du projet

2.3.1. Inclusions

Voici comment vous pourriez compléter ces sections pour votre projet de gestionnaire de stock :

2.3. Périmètre du projet

2.3.1. Inclusions

- **Fonctionnalités Backend (API Express)**
    - Création et gestion complète des produits (CRUD)
    - Gestion des catégories de produits
    - Suivi des mouvements de stock (entrées/sorties)
    - Calcul automatique des niveaux de stock en temps réel
    - Système d'alertes pour les seuils minimaux de stock
    - Authentification des utilisateurs (JWT)
    - Gestion des rôles et permissions
    - Documentation complète de l'API (Swagger/OpenAPI)
- **Interface Utilisateur (React + Tailwind CSS)**
    - Dashboard principal avec indicateurs clés
    - Interface de gestion des produits
    - Système de recherche et filtrage avancé
    - Visualisation des historiques de mouvements
    - Responsive design pour une utilisation sur différents appareils

    - Thème visuel cohérent et professionnel

- **Base de données**
    - Modélisation des données
    - Schéma relationnel optimisé
    - Mécanismes de sauvegarde et récupération
- **Tests et Validation**
    - Tests unitaires pour les fonctionnalités critiques
    - Tests d'intégration pour l'API
    - Tests de l'interface utilisateur

2.3.2. Exclusions

- Intégration avec des systèmes ERP ou solutions comptables
- Fonctionnalités de prévisions ou d'analyse prédictive
- Module de gestion des fournisseurs et commandes automatiques
- Application mobile native (seule une version web responsive sera développée)
- Fonctionnalités de gestion financière (valorisation des stocks, calcul de coûts)
- Gestion multi-entrepôts complexe
- Traçabilité par code-barres ou QR code
- Intégration avec des dispositifs de scanning physiques
- Support multilingue (seul le français sera supporté dans cette version)
- Système de facturation ou de devis

2.4. État des lieux

Actuellement, notre expérience dans le domaine de la gestion de stock est principalement théorique, sans solution logicielle spécifique déjà développée par notre équipe. Nous sommes au stade initial du projet, avec une vision claire des besoins mais sans code existant à maintenir ou à intégrer.

Environnement technologique actuel :

- **Compétences de base en JavaScript et développement web**
- **Connaissance théorique des frameworks Express.js et React**
- **Familiarité limitée avec Tailwind CSS**
- **Expérience en conception de bases de données relationnelles**

Processus et méthodes de travail :

- **Collaboration établie entre les deux membres de l'équipe (nous-même et Brahim H.)**

- **Utilisation de méthodologies agiles pour le suivi du développement**
- **Gestion de code via Git et GitHub**
- **Communication régulière via des outils de visioconférence et messagerie**

Contraintes identifiées :

- **Courbe d'apprentissage pour maîtriser les technologies choisies**
- **Ressources limitées (équipe de deux personnes)**
- **Nécessité de produire un MVP présentable dans un délai raisonnable**
- **Besoin d'équilibrer la qualité technique et l'esthétique de l'interface utilisateur**

Cette phase initiale du projet représente à la fois un défi et une opportunité, nous permettant de concevoir une solution moderne sans les contraintes liées à des systèmes préexistants, tout en développant nos compétences dans les technologies ciblées.

3. BESOINS ET OBJECTIFS

3.1. Objectifs stratégiques

- **Développement de compétences techniques** : Acquérir une maîtrise approfondie des technologies modernes de développement web (Express.js, React, Tailwind CSS) pour renforcer notre profil professionnel.

- **Constitution d'un portfolio solide** : Créer une application démontrable et fonctionnelle qui servira de référence pour nos futures opportunités professionnelles.

- **Positionnement sur le marché de la logistique** : Établir les bases d'une expertise dans le domaine de la gestion de stock et de la logistique, secteur en croissance constante.

- **Évolutivité de la solution** : Concevoir une architecture permettant d'étendre les fonctionnalités dans le futur et potentiellement transformer ce MVP en produit commercialisable.

- **Création de valeur** : Développer une solution qui répond à des besoins réels du marché et qui pourrait être valorisée auprès de petites et moyennes entreprises.

3.2. Objectifs opérationnels

- **Livraison d'un MVP fonctionnel** : Développer une version minimale mais complète du gestionnaire de stock dans un délai de 3 à 4 mois.

- **Maîtrise progressive des technologies** : Acquérir une compétence opérationnelle en Express.js dans le premier mois, puis en React et Tailwind CSS dans les mois suivants.

- **Documentation complète** : Produire une documentation technique détaillée et des guides d'utilisation pour faciliter la présentation du projet.

- **Tests fonctionnels** : Réaliser des tests complets pour garantir la fiabilité et la stabilité de l'application.

- **Optimisation de l'expérience utilisateur** : Concevoir une interface intuitive et efficace permettant une prise en main rapide de l'application.

- **Déploiement de démonstration** : Mettre en place une version de démonstration accessible en ligne pour présentation dans notre portfolio.

3.3. Besoins fonctionnels

- **Gestion des utilisateurs**
    - Création et gestion des comptes utilisateurs
    - Authentification sécurisée
    - Gestion des rôles et des permissions (administrateur, gestionnaire, utilisateur standard)
- **Gestion des produits**
    - Ajout, modification et suppression de produits
    - Catégorisation des produits
    - Définition des attributs (référence, nom, description, unité de mesure, etc.)
    - Gestion des photos/images des produits
- **Gestion des stocks**
    - Suivi des quantités disponibles par produit
    - Enregistrement des entrées (approvisionnements)
    - Enregistrement des sorties (consommations, ventes)
    - Historique complet des mouvements de stock
    - Définition de seuils d'alerte (stock minimum)
- **Reporting et tableaux de bord**
    - Tableau de bord avec indicateurs clés
    - Alertes visuelles pour les produits sous le seuil minimal
    - Rapports d'inventaire exportables
    - Statistiques basiques sur les mouvements de stock
- **Administration du système**
    - Configuration des paramètres généraux
    - Gestion des sauvegardes
    - Journalisation des actions importantes

3.4. Besoins non fonctionnels

- **Performance**
    - Temps de réponse inférieur à 2 secondes pour les opérations courantes
    - Capacité à gérer jusqu'à 10 000 références produits

- Support de plusieurs utilisateurs simultanés (5-10 utilisateurs)

- **Sécurité**
  - Protection contre les injections SQL et autres vulnérabilités web courantes
  - Authentification sécurisée avec tokens JWT
  - Chiffrement des mots de passe
  - Journalisation des accès et modifications sensibles
- **Ergonomie et accessibilité**
  - Interface intuitive nécessitant peu de formation
  - Design responsive adapté aux ordinateurs et tablettes
  - Respect des principes d'accessibilité de base (contraste, taille des éléments)
  - Cohérence visuelle sur l'ensemble de l'application
- **Maintenabilité**
  - Code modulaire et bien documenté
  - Séparation claire entre frontend et backend
  - Tests automatisés pour les fonctionnalités critiques
  - Structure de projet facilitant l'extension future
- **Disponibilité**
  - Application accessible 24/7 via Internet
  - Mécanismes de sauvegarde réguliers
  - Gestion des erreurs avec messages appropriés

3.5. Indicateurs de succès

- **Indicateurs techniques**
  - Couverture de tests > 70% pour les fonctionnalités critiques
  - Temps de réponse moyen < 2 secondes
  - Nombre de bugs critiques identifiés en phase de test < 5
  - API documentée à 100%
- **Indicateurs liés aux objectifs d'apprentissage**
  - Capacité à expliquer et défendre les choix d'architecture technique
  - Maîtrise démontrée des technologies Express.js, React et Tailwind CSS
  - Production d'au moins un article technique ou présentation sur les compétences acquises
- **Indicateurs de qualité**

- Test utilisateur avec au moins 3 utilisateurs potentiels avec un score de satisfaction > 7/10
- Respect des délais de développement fixés (écart < 20%)
- Fonctionnalités du MVP implémentées à 100%

- **Indicateurs de portfolio**
  - Représentation claire du projet dans notre portfolio professionnel
  - Au moins 2 retours positifs de professionnels sur la qualité du projet
  - Application fonctionnelle et démontrable en situation réelle ou simulée

4. SPÉCIFICATIONS FONCTIONNELLES

4.1. Cas d'usage principaux

4.1.1. Cas d'usage 1

Titre : Création d'un nouveau produit dans l'inventaire
Acteurs : Administrateur, Gestionnaire de stock
Description : L'utilisateur ajoute un nouveau produit au système avec toutes ses caractéristiques et définit son stock initial.
Préconditions : L'utilisateur est authentifié et possède les droits suffisants.
Postconditions : Le produit est créé et visible dans l'inventaire avec son stock initial.
Scénario principal :

1. **L'utilisateur accède à la section "Gestion des produits"**
2. **Il clique sur le bouton "Ajouter un produit"**
3. **Il remplit le formulaire avec les informations du produit (référence, nom, description, catégorie, etc.)**
4. **Il définit le stock initial**
5. **Il valide le formulaire**
6. **Le système confirme la création du produit**

Scénarios alternatifs :

- **Si la référence produit existe déjà, le système affiche un message d'erreur et demande une autre référence**
- **Si l'utilisateur annule l'opération, aucun produit n'est créé**
- **Si certains champs obligatoires ne sont pas renseignés, le système met en évidence ces champs et empêche la validation**

4.1.2. Cas d'usage 2

Titre : Enregistrement d'une entrée de stock
Acteurs : Gestionnaire de stock, Utilisateur standard
Description : L'utilisateur enregistre l'entrée de nouvelles quantités pour un produit existant.

Préconditions : L'utilisateur est authentifié, le produit existe dans le système.
Postconditions : Le stock du produit est incrémenté, le mouvement est enregistré dans l'historique.
Scénario principal :

1. **L'utilisateur recherche et sélectionne le produit concerné**
2. **Il clique sur "Ajouter une entrée de stock"**
3. **Il saisit la quantité à ajouter**
4. **Il renseigne éventuellement un motif ou une référence de document**
5. **Il valide l'opération**
6. **Le système met à jour le stock et confirme l'opération**

Scénarios alternatifs :

- **Si l'utilisateur saisit une quantité négative ou nulle, le système affiche un message d'erreur**
- **Si l'utilisateur annule l'opération, aucun mouvement n'est enregistré**

4.1.3. Cas d'usage 3

Titre : Enregistrement d'une sortie de stock
Acteurs : Gestionnaire de stock, Utilisateur standard
Description : L'utilisateur enregistre la sortie de quantités pour un produit existant.
Préconditions : L'utilisateur est authentifié, le produit existe dans le système et le stock disponible est suffisant.
Postconditions : Le stock du produit est décrémenté, le mouvement est enregistré dans l'historique.
Scénario principal :

1. **L'utilisateur recherche et sélectionne le produit concerné**
2. **Il clique sur "Enregistrer une sortie de stock"**
3. **Il saisit la quantité à retirer**
4. **Il renseigne un motif ou une référence de document**
5. **Il valide l'opération**
6. **Le système met à jour le stock et confirme l'opération**

Scénarios alternatifs :

- **Si le stock disponible est insuffisant, le système affiche un avertissement et demande confirmation**
- **Si l'utilisateur saisit une quantité négative ou nulle, le système affiche un message d'erreur**
- **Si l'utilisateur annule l'opération, aucun mouvement n'est enregistré**

4.1.4. Cas d'usage 4

Titre : Consultation du tableau de bord
Acteurs : Administrateur, Gestionnaire de stock, Utilisateur standard

Description : L'utilisateur consulte le tableau de bord pour avoir une vision globale de l'état des stocks.
Préconditions : L'utilisateur est authentifié.
Postconditions : Aucune modification du système.
Scénario principal :

1. **L'utilisateur se connecte à l'application**
2. **Le système affiche automatiquement le tableau de bord**
3. **L'utilisateur visualise les indicateurs clés (produits en alerte, mouvements récents, etc.)**
4. **L'utilisateur peut filtrer ou affiner les données affichées**

Scénarios alternatifs :

- **Si aucune donnée n'est disponible, le système affiche un message approprié**
- **L'utilisateur peut exporter les données du tableau de bord au format PDF ou CSV**

4.2. Exigences fonctionnelles détaillées

4.2.1. Fonctionnalité 1

Identifiant : F-AUTH-001
Priorité : Essentielle
Description : Système d'authentification des utilisateurs
Critères d'acceptation :

- **Les utilisateurs peuvent créer un compte avec email et mot de passe**
- **Les mots de passe sont stockés de manière sécurisée (hachage)**
- **La connexion génère un token JWT avec une durée de validité de 24h**
- **Les tokens peuvent être révoqués en cas de déconnexion**
- **Les routes protégées vérifient la validité du token**

4.2.2. Fonctionnalité 2

Identifiant : F-PROD-001
Priorité : Essentielle
Description : Gestion complète des produits (CRUD)
Critères d'acceptation :

- **Les produits peuvent être créés avec tous leurs attributs (référence unique, nom, description, etc.)**
- **Les produits peuvent être modifiés par les utilisateurs autorisés**
- **Les produits peuvent être supprimés logiquement (soft delete)**
- **La recherche de produits est possible par référence, nom ou catégorie**
- **La liste des produits peut être filtrée et triée selon différents critères**

4.2.3. Fonctionnalité 3

Identifiant : F-STOCK-001
Priorité : Essentielle
Description : Gestion des mouvements de stock
Critères d'acceptation :

- **Les entrées de stock incrémentent la quantité disponible**
- **Les sorties de stock décrémentent la quantité disponible**
- **Chaque mouvement est horodaté et attribué à un utilisateur**
- **L'historique complet des mouvements est consultable par produit**
- **Les mouvements peuvent être filtrés par date, type ou utilisateur**

4.2.4. Fonctionnalité 4

Identifiant : F-ALERT-001
Priorité : Importante
Description : Système d'alertes de stock minimum
Critères d'acceptation :

- **Un seuil minimal peut être défini pour chaque produit**
- **Les produits dont le stock est inférieur au seuil sont signalés visuellement**
- **Un rapport des produits en alerte peut être généré**
- **Les alertes apparaissent sur le tableau de bord**
- **Des notifications peuvent être configurées (dans l'interface uniquement)**

4.2.5. Fonctionnalité 5

Identifiant : F-REPORT-001
Priorité : Importante
Description : Génération de rapports d'inventaire
Critères d'acceptation :

- **Un état des stocks complet peut être généré à la demande**
- **Les rapports peuvent être filtrés par catégorie de produits**
- **Les rapports peuvent être exportés au format CSV ou PDF**
- **Les mouvements de stock sur une période donnée peuvent être analysés**
- **Des graphiques simples illustrent l'évolution des niveaux de stock**

4.3. Interfaces utilisateurs

L'interface utilisateur du gestionnaire de stock sera développée en React avec Tailwind CSS pour un design moderne et responsif. Les principales interfaces seront :

1. Interface de connexion

- **Formulaire de connexion épuré**
- **Option de récupération de mot de passe**

- **Gestion des erreurs d'authentification**

2. Tableau de bord principal

- **Vue d'ensemble avec les indicateurs clés (nombre de produits, valeur du stock, alertes)**
- **Graphiques d'évolution des stocks sur les dernières périodes**
- **Liste des derniers mouvements enregistrés**
- **Raccourcis vers les actions fréquentes**

3. Interface de gestion des produits

- **Liste paginée des produits avec filtres et tri**
- **Formulaire de création/modification de produit**
- **Vue détaillée d'un produit avec son historique de mouvements**
- **Gestion des catégories de produits**

4. Interface de gestion des stocks

- **Formulaires d'entrée et de sortie de stock**
- **Historique des mouvements avec filtres**
- **Visualisation des tendances par produit**
- **Gestion des seuils d'alerte**

5. Interface d'administration

- **Gestion des utilisateurs et des droits**
- **Configuration générale du système**
- **Journaux d'activité**
- **Options de sauvegarde/restauration**

Chaque interface respectera les principes suivants :

- **Design responsive adapté aux écrans d'ordinateur et de tablette**
- **Palette de couleurs cohérente et professionnelle**
- **Formulaires avec validation en temps réel**
- **Feedback visuel clair pour les actions réussies ou en erreur**
- **Aide contextuelle disponible pour les fonctionnalités complexes**

4.4. Interfaces externes

Le système étant un MVP, les interfaces externes seront limitées mais prévues pour des extensions futures :

1. API REST

- **Endpoints documentés via Swagger/OpenAPI**
- **Authentification par token JWT**
- **Formats de données: JSON**
- **Gestion des erreurs standardisée avec codes HTTP appropriés**

2. Export de données

- **Génération de fichiers CSV pour l'export des données d'inventaire**
- **Génération de rapports PDF pour les états de stock**
- **Possibilité d'exporter l'historique des mouvements**

3. Préparation pour intégrations futures

- **Structure d'API permettant une future intégration avec des systèmes ERP**
- **Possibilité d'extension pour une API publique (avec clés d'API)**
- **Architecture préparée pour l'ajout d'un système de webhooks pour des notifications externes**

Aucune intégration directe avec des systèmes externes n'est prévue dans le périmètre du MVP, mais l'architecture sera conçue pour faciliter ces intégrations dans les versions ultérieures.

5. SPÉCIFICATIONS TECHNIQUES

5.1. Architecture générale

L'application suit une **architecture modulaire en 3 couches** :

- **Frontend** : Application React.js avec Redux pour la gestion d'état
- **Backend** : API REST Node.js/Express avec une base de données PostgreSQL
- **Base de données** : PostgreSQL pour le stockage relationnel des données produits/mouvements de stock

Communication entre frontend et backend via requêtes HTTP (REST). Authentification gérée via JWT.

(Note : Les besoins avancés comme Elasticsearch, RabbitMQ et Grafana ne sont pas inclus dans la version actuelle mais pourront être intégrés ultérieurement.)

5.2. Environnement technique

5.2.1. Matériel

- **Serveur de production** :
    - CPU : 4 cœurs (2.5 GHz min)
    - RAM : 8 Go minimum (16 Go recommandé pour une meilleure performance)
    - Stockage : SSD 100 Go (pour la base de données + logs)
- **Postes clients** :

- Navigateur web récent (Chrome, Firefox, Edge)
- Résolution d'écran : 1280x720 minimum

5.2.2. Logiciel

- **Backend** :
  - Node.js (v18+)
  - Express.js
  - PostgreSQL (v15+)
  - Bibliothèques principales : jsonwebtoken (JWT), sequelize (ORM), express-validator
- **Frontend** :
  - React.js (v18+)
  - Redux Toolkit
  - Tailwind CSS
  - Axios pour les requêtes API
- **Développement** :
  - Docker (pour containerisation)
  - Git (gestion de versions)
  - Postman/Insomnia (tests API)

5.2.3. Infrastructure réseau

- **En production** :
  - Bande passante : 10 Mbps minimum
  - HTTPS obligatoire (SSL/TLS via Let's Encrypt ou certificat dédié)
  - Accès RESTRICT aux endpoints API (firewall)
- **En développement** :
  - Accès local (localhost) ou VPN si nécessaire

5.3. Sécurité

- **Authentification** : JWT avec expiration courte (1h) + refresh token
- **RBAC (Rôle-Based Access Control)** :
  - **Admin** : Gestion complète des produits/stocks
  - **Utilisateur standard** : Consultation + entrées/sorties basiques
- **Protection des données** :
  - Chiffrement des mots de passe (bcrypt)

    - Prévention des injections SQL (via Sequelize)
    - Validation stricte des inputs (express-validator)
- **Journalisation (logging)** des accès et modifications critiques

5.4. Performance

- **Temps de réponse API** :
    - < 500 ms pour 90% des requêtes (hors pics de charge)
- **Charge supportée** :
    - Jusqu'à 50 requêtes simultanées (scalable via Docker/Kubernetes si besoin)
- **Disponibilité** :
    - 99.5% en production (hors fenêtres de maintenance)

5.5. Maintenance

- **Sauvegardes :**
    - **Base de données : Sauvegarde automatique quotidienne (via pg_dump)**
    - **Stockage externe des backups (AWS S3 ou équivalent)**
- **Mises à jour :**
    - **Correctives : Délai max. de 72h après détection d'un bug critique**
    - **Évolutives : Planifiées mensuellement (si hors urgence)**
- **Monitoring :**
    - **Logs centralisés (via Winston/Morgan)**
      (Note : Grafana/Elasticsearch pourraient être ajoutés pour du monitoring avancé dans une V2)

5.6. Normes et standards

- **API** : Respect des conventions RESTful (verb HTTP, codes status, etc.)
- **Code** :
    - ESLint (standard Airbnb)
    - Convention de nommage : camelCase (JS), PascalCase (React components)
- **Base de données** : Normalisation SQL (3e forme normale)
- **Sécurité** : OWASP Top 10 (protection contre XSS, CSRF, etc.)

6. CONTRAINTES

6.1. Contraintes légales et réglementaires

- **RGPD/CNIL** :
    - Protection des données personnelles (gestion des accès utilisateurs, journalisation).

    - Droit à l'oubli : suppression des comptes/utilisateurs sur demande.

- **Normes sectorielles** :
    - Respect des bonnes pratiques de traçabilité des stocks (obligatoire dans certains secteurs comme l'agroalimentaire ou la pharmacie).
- **Licences logicielles** :
    - Utilisation de technologies open-source (Node.js, PostgreSQL) pour éviter des coûts de licence.

6.2. Contraintes organisationnelles

- **Interopérabilité** :
    - L'API doit pouvoir s'intégrer à des outils métiers existants (ERP, logiciel de comptabilité, etc.).
- **Formation utilisateurs** :
    - Nécessité de former les équipes à l'interface (en particulier pour les profils non techniques).
- **Processus métier** :
    - Adaptation possible des workflows de gestion de stock (ex : validation hiérarchique pour les sorties critiques).

6.3. Contraintes techniques

- **Compatibilité** :
    - Support des navigateurs récents (Chrome, Firefox, Edge, Safari) – pas de support pour IE11.
- **Dépendances** :
    - Maintenance des librairies critiques (Node.js, React, PostgreSQL) à jour pour éviter les vulnérabilités.
- **Limitations actuelles** :
    - Pas de système de cache (Redis) ni de recherche avancée (Elasticsearch) dans la V1.
    - Génération/scan de QR codes reportée à une version ultérieure.

6.4. Contraintes de délai

- **MVP (Minimum Viable Product)** :
    - Version fonctionnelle (gestion basique des stocks + authentification) à livrer sous **3 mois**.
- **Déploiement final** :
    - Solution complète (dashboard, logs, intégrations) sous **6 mois**.
- **Contraintes externes** :

- Alignement possible avec des audits annuels de stock (si applicable).

6.5. Contraintes budgétaires

- **Développement** :
  - Budget alloué aux outils : < 5 000 € (hébergement, licences éventuelles, services cloud).
- **Maintenance** :
  - Coût mensuel de maintenance estimé à 10-15% du coût initial après la première année.
- **Optimisation** :
  - Priorité aux solutions open-source pour limiter les coûts (ex : PostgreSQL au lieu d'Oracle).

7. LIVRABLES

7.1. Livrables principaux

- **API de gestion de stock** :
  - Code source backend (Node.js/Express) avec tests unitaires et d'intégration
  - Documentation technique des endpoints (OpenAPI/Swagger)
- **Application frontend** :
  - Interface React.js avec gestion des stocks et tableau de bord
  - Build optimisé pour production (prêt à déployer)
- **Base de données** :
  - Scripts SQL de création et peuplement initial
  - Documentation du schéma relationnel
- **Environnement déployable** :
  - Fichiers Docker/Docker-Compose pour conteneurisation
  - Scripts de déploiement (CI/CD optionnel)

7.2. Documentation

- **Documentation technique** :
  - Guide d'installation et configuration
  - Référence API (endpoints, payloads, codes d'erreur)
  - Architecture technique et choix de conception
- **Documentation utilisateur** :
  - Manuel d'utilisation (captures d'écran incluses)

    - FAQ pour résoudre les problèmes courants
- **Rapport de projet** :
    - Bilan des fonctionnalités livrées vs. prévues
    - Recommandations pour les évolutions futures

7.3. Formation

- **Formation administrateurs** (2 sessions) :
    - Gestion des utilisateurs/rôles
    - Supervision des stocks et interprétation des logs
- **Formation utilisateurs** (1 session) :
    - Saisie des mouvements de stock
    - Consultation du dashboard et génération de rapports
- **Supports pédagogiques** :
    - Slides de formation
    - Vidéos tutoriels (optionnel)
    - Environnement de test dédié à la formation

7.4. Support et maintenance

- **Période de garantie** :
    - Correctifs gratuits pendant **3 mois** post-livraison
- **Support technique** :
    - Niveau 1 (assistance utilisateur) : Réponse sous 24h ouvrées
    - Niveau 2 (bugs critiques) : Réponse sous 4h (jours ouvrés)
- **Maintenance évolutive** :
    - Forfait mensuel ajustable en fonction des besoins (ex : ajout de nouvelles fonctionnalités)
    - Mises à jour de sécurité incluses pendant 12 mois

8. PLANNING

8.1. Phasage du projet

Le projet sera découpé en **4 phases principales** :

1. **Phase de Conception (3 semaines)**
    - Spécifications techniques détaillées

- Modélisation de la base de données
- Prototypage UI/UX

2. **Phase de Développement (8 semaines)**
    - Implémentation backend (API + sécurité)
    - Développement frontend (interface utilisateur)
    - Tests unitaires et d'intégration
3. **Phase de Recette et Validation (3 semaines)**
    - Tests utilisateurs (UAT)
    - Corrections des bugs critiques
    - Optimisation des performances
4. **Phase de Déploiement (2 semaines)**
    - Mise en production
    - Formation des utilisateurs
    - Documentation finale

8.2. Jalons principaux

Jalon	Date prévisionnelle	Livrables associés
J1 - Spécifications validées	15/05/2024	Cahier des charges finalisé, schéma DB
J2 - MVP Fonctionnel	20/04/2024	API de base (CRUD stocks), interface admin
J3 - Intégration complète	25/04/2024	Frontend connecté, JWT opérationnel
J4 - Recette terminée	30/05/2024	Rapport de tests, corrections appliquées
J5 - Livraison finale	15/07/2024	Solution en production, docs fournies

8.3. Calendrier détaillé

Semaines 1 : Conception

- **Tâches** :
    - Finalisation des spécifications techniques
    - Création des maquettes UI (Figma)
    - Modélisation PostgreSQL (outil : DbDiagram)
- **Ressources** :
    - Chef de projet, UX Designer, Architecte technique

Semaines 2-3-4 : Développement

- **Tâches** :

Sprint (2 semaines)	Objectifs
Sprint 1	Setup backend + authentification JWT
Sprint 2	CRUD produits + gestion des mouvements
Sprint 3	Dashboard React + connexion API
Sprint 4	Logs + sécurisation finale

- **Dépendances** :
    - Le frontend dépend de l'API stable (à partir du Sprint 2)

Semaines 5-6-7 : Recette

- **Tâches** :
    - Tests manuels (scénarios utilisateurs)
    - Benchmark de performance (JMeter ou équivalent)
    - Audit sécurité (OWASP ZAP)
- **Livrables** :
    - Rapport de tests avec priorités (P1-P3)

Semaines 8-9-10 : Déploiement

- **Tâches** :
    - Déploiement Docker sur serveur client
    - Formation des administrateurs (1 journée)
    - Documentation utilisateur finalisée

Outils de suivi :

- **Gestion de projet** : Jira/Trello (suivi des sprints)

- **Versioning** : Git (branches dev/prod)
- **Collaboration** : Slack/Teams pour les points quotidiens

Risques et marges :

- Buffer de 2 semaines intégré pour les retards éventuels (livraison max. 01/08/2024).

9. BUDGET

9.1. Estimation des coûts

Le projet étant développé **sans budget initial**, les coûts sont limités aux solutions gratuites et open-source :

Poste	Coût estimé (€)	Description
Développement	0	Travail bénévole (temps personnel)
Hébergement	0	Utilisation de services gratuits (Vercel, Railway, NeonDB)
Outils	0	Stack open-source (Node.js, React, PostgreSQL)
Formation	0	Autoformation via documentation/tutoriels
Maintenance	0	Gestion manuelle sans coût immédiat
Total	**0 €**	

Remarque : Si le projet évolue vers une version professionnelle, des coûts pourraient apparaître (hébergement dédié, services premium, etc.).

9.2. Plan de financement

- **Financement actuel** :
    - **Auto-financement** : Temps personnel et ressources gratuites.
- **Financement futur (optionnel)** :
    - Crowdfunding (ex : Ulule) pour une version cloud.
    - Sponsoring via bêta-testeurs (accès anticipé en échange de feedback).

9.3. Modalités de paiement

- **Projet bénévole** :
    - Aucun échéancier ou paiement prévu.
- **Si monétisation ultérieure** :
    - Modèle Freemium (version de base gratuite, fonctionnalités avancées payantes).
    - Paiement unique pour une licence auto-hébergée.

10. MODALITÉS D'ÉVALUATION

10.1. Processus de validation

- **Revues techniques hebdomadaires** :
    - Validation des fonctionnalités développées via des démonstrations (démos)
    - Utilisation de **pull requests** (GitHub/GitLab) pour valider le code avant fusion
- **Validation par livrable** :
    - Chaque module (ex: authentification, gestion de stock) doit passer une **checklist** avant livraison
    - Approbation formelle par le responsable projet (ou auto-validation pour un projet personnel)

10.2. Tests et recette

Types de tests :

Catégorie	Outils/Méthodes	Couverture
Tests unitaires	Jest (Backend), React Testing Library (Frontend)	80% des composants critiques
Tests d'intégration	Supertest (API), Postman	Flux principaux (ex: ajout produit → mise à jour stock)
Tests E2E	Cypress ou Playwright	Scénarios utilisateurs clés
Tests de performance	Artillery (simulation de charge)	Temps de réponse < 1s sous 50 req/s

Recette utilisateur (UAT) :

- **Participants** : 3-5 bêta-testeurs (si disponibles)
- **Durée** : 1 semaine
- **Critères de succès** :
    - 0 bug bloquant (P1)
    - 90% des fonctionnalités utilisables sans documentation

10.3. Critères d'acceptation

Pour l'API :

- ✓ Tous les endpoints répondent avec les codes HTTP appropriés
- ✓ Sécurité : Injection SQL impossible, routes protégées par JWT
- ✓ Performance : 95% des requêtes < 500 ms

Pour le Frontend :

- ✓ Affichage responsive (mobile/desktop)
- ✓ Gestion des erreurs utilisateur claire (ex: stock insuffisant)

Base de données :

- ✓ Intégrité des données (contraintes FK, cascades)
- ✓ Sauvegarde/restauration fonctionnelle

10.4. Gestion des risques

Risque	Probabilité	Impact	Atténuation
Retards de développement	Moyenne	Élevé	Buffer de 2 semaines dans le planning
Bugs critiques en prod	Faible	Critique	Environnement de staging identique à la prod
Dépendance à des libs non maintenues	Moyenne	Modéré	Choix de librairies avec communauté active
Données corrompues	Faible	Critique	Sauvegardes automatiques quotidiennes

Stratégie globale :

- Journalisation détaillée (logs) pour tracer les erreurs
- Mise à jour régulière des dépendances (npm audit, Dependabot)

11. ANNEXES

11.1. Glossaire

Terme	Définition
API REST	Interface de programmation respectant les principes REST (stateless, verbes HTTP).
JWT	JSON Web Token, standard d'authentification sécurisé via tokens signés.
ORM	Outil de mapping objet-relationnel (ex: Sequelize pour PostgreSQL).
UAT	User Acceptance Testing : tests de validation par les utilisateurs finaux.
CI/CD	Intégration et déploiement continus (automatisation des builds/tests).

11.2. Documents complémentaires

1. **Diagrammes UML** :
    - Diagramme de classes (modèle de données)
    - Diagramme de séquence (flux API)
2. **README.md** :
    - Instructions d'installation et configuration
    - Liste des dépendances techniques
