Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université / Institut . . . . . .

Institut Supérieur d'Informatique
et de Multimédia de Gabes

RAPPORT DE STAGE TECHNICIEN

Entreprise : TeyebCom

Période de stage : Juillet 2025

Titre : Application de Gestion de Stock

Étudiant : Mohamed Teyeb

Encadrant entreprise : Khalil Teyeb

Année universitaire : 2024/2025

Remerciements

Je remercie sincèrement M. Khalil Teyeb pour son encadrement méthodique, son exigence de qualité et sa disponibilité constante. Mes remerciements s'adressent également à l'équipe TeyebCom pour l'accueil, l'entraide et les retours constructifs. Enfin, j'exprime ma gratitude envers le corps enseignant et administratif de mon établissement pour l'accompagnement académique qui a rendu ce stage possible.

Table des matières

Table des figures

CHAPITRE 1

Introduction générale

Sommaire

1.1 Contexte

La digitalisation des opérations commerciales exige des solutions fiables de suivi de stock. TeyebCom, société tunisienne de services numériques et de gestion commerciale, vise à offrir une application centralisée permettant de contrôler les produits, les mouvements et les alertes, tout en intégrant les contraintes métier de réassort et de traçabilité.

1.2 Problématique

Comment livrer, en un stage technicien, une application web full-stack qui :

— assure la cohérence des mouvements (entrées, sorties, transferts),
— prévient les ruptures par des seuils configurables,
— sécurise les opérations sensibles (authentification, autorisations),
— reste performante et maintenable pour accompagner la croissance des données ?

1.3 Objectifs

— Concevoir et implémenter une application de gestion de stock avec React.js (front) et Express.js (API).
— Structurer une base MySQL (XAMPP) couvrant produits, catégories, mouvements, utilisateurs.
— Offrir un tableau de bord synthétique, des filtres avancés et des alertes de seuil.
— Intégrer l'authentification et des contrôles de validation pour sécuriser les opérations.
— Documenter l'architecture, les choix techniques, et les scénarios d'usage.

1.4 Méthodologie

— Recueil des besoins auprès de l'équipe TeyebCom.

— Modélisation UML (cas d'utilisation, classes).
— Architecture en couches (React $\rightarrow$ Express $\rightarrow$ MySQL).
— Développement incrémental avec tests fonctionnels via Postman et validations front.
— Itérations guidées par l'encadrant, consolidation documentaire.

CHAPITRE 2

Présentation de l'Entreprise TeyebCom

Sommaire

2.1 Introduction

TeyebCom, basée en Tunisie, accompagne les PME dans la digitalisation de la gestion commerciale, avec un focus sur le pilotage de stock, la facturation et la relation client.

2.2 Historique & implantation

Créée en Tunisie, TeyebCom s'est spécialisée dans les solutions numériques adaptées aux commerces locaux et régionaux, avec un ancrage opérationnel proche des besoins terrain.

2.3 Mission, vision, valeurs

Mission : Concevoir des outils numériques intégrés pour fiabiliser et accélérer la gestion commerciale.
Vision : Être le partenaire de référence en solutions de stock pour les acteurs tunisiens.
Valeurs : Proximité client, fiabilité, innovation pragmatique, transparence.

2.4 Activités & services

— Développement d'applications web et mobiles orientées gestion.
— Intégration de solutions de stock et de facturation.
— Conseil en organisation des flux logistiques et commerciaux.
— Support et maintenance applicative.

2.5 Réalisations

TeyebCom a livré des solutions de suivi de stock et de facturation pour des commerces de détail et des distributeurs, contribuant à réduire les ruptures et à améliorer les délais de réassort.

2.6 Culture & équipe

Culture collaborative : développeurs, consultants métiers et support travaillent en synergie. L'entreprise valorise la réactivité, la qualité de service et la formation continue.

2.7 Conclusion

Le positionnement de TeyebCom sur la gestion commerciale offre un cadre idéal pour développer une application de stock robuste et adaptée aux besoins locaux.

CHAPITRE 3

Étude Théorique

Sommaire

3.1 Analyse des besoins

3.1.1 Besoins fonctionnels (toutes les fonctionnalités du projet)

- **Authentification & gestion des sessions :** connexion, token (JWT si activé), protection des routes sensibles.
- **Gestion des utilisateurs (optionnel) :** création, rôles simples (admin/gestionnaire), suivi des actions.
- **Gestion des produits (CRUD) :** création, édition, suppression, recherche par référence/nom/catégorie.
- **Gestion des catégories :** ajout, édition, suppression, hiérarchisation simple.
- **Mouvements de stock :** entrées, sorties, transferts internes, ajustements ; journal horodaté.
- **Alertes & seuils :** définition des seuils critiques, notifications visuelles sur le tableau de bord.
- **Tableau de bord :** indicateurs clés (stock total, alertes, derniers mouvements, valeur estimée).
- **Recherche & filtrage avancés :** par catégorie, par type de mouvement, par plage de dates, par statut d'alerte.
- **Historique & traçabilité :** journal des opérations avec utilisateur et timestamp.
- **Validation métier :** contrôle de stock non négatif, cohérence des quantités, formats de données.
- **Exports/rapports (optionnel) :** extraction CSV pour audit ou reporting.
- **Accessibilité & ergonomie :** navigation fluide, formulaires validés, messages d'erreur explicites.

3.1.2 Besoins non fonctionnels

— **Sécurité :** validation serveur, protection des entrées, contrôle d'accès.
— **Performance :** pagination, filtres côté serveur, requêtes optimisées.
— **Accessibilité :** interface lisible, navigation clavier, contrastes adéquats.
— **Scalabilité :** séparation front/back, services modulaires, schéma relationnel clair.
— **Maintenabilité :** architecture par couches (routes, contrôleurs, services), documentation.
— **Fiabilité :** gestion des erreurs API, retours cohérents, tests via Postman.
— **Compatibilité :** exécution locale via XAMPP/MySQL et déploiement cloud envisageable.

3.2 Modélisation UML

3.2.1 Diagramme de cas d'utilisation

Acteurs : Administrateur, Gestionnaire, Utilisateur authentifié.
Cas : gérer produits, gérer catégories, enregistrer mouvements (entrée/sortie/transfert), consulter historique et alertes, authentifier, visualiser le tableau de bord, exporter les données (optionnel).

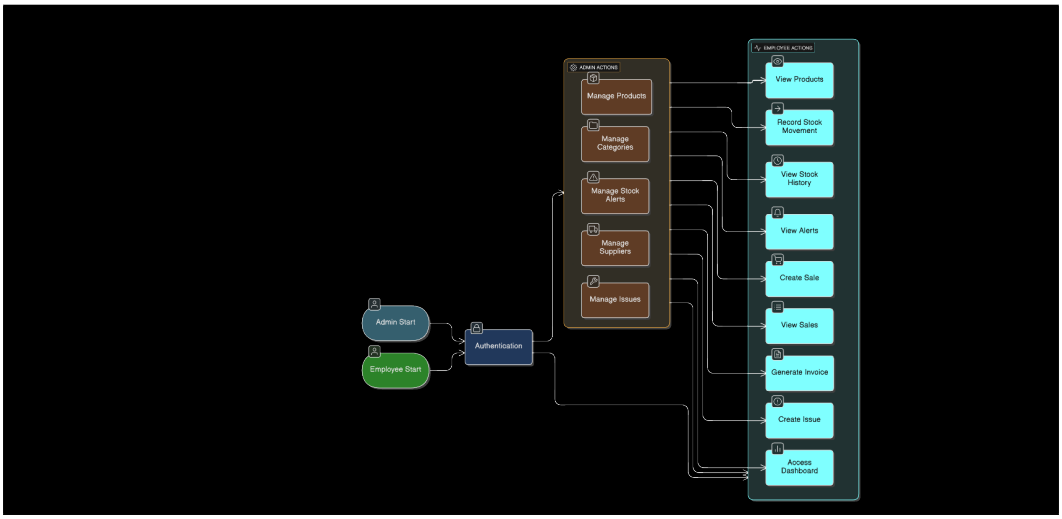

FIGURE 3.1 – Diagramme de cas d'utilisation : interactions acteurs/fonctionnalités (fond noir).

3.2.2 Diagramme de classes

Classes principales :
— **Produit** (id, nom, référence, categoryId, stockActuel, seuilAlerte, prixUnitaire, statut).
— **Categorie** (id, libelle, description).
— **Mouvement** (id, produitId, type, quantite, dateMvt, utilisateurId, commentaire).
— **Utilisateur** (id, nom, email, motDePasseHash, role).

Relations :

— Categorie 1..* — 0..* Produit.

— Produit 1..* — 0..* Mouvement.

— Utilisateur 1..* — 0..* Mouvement.

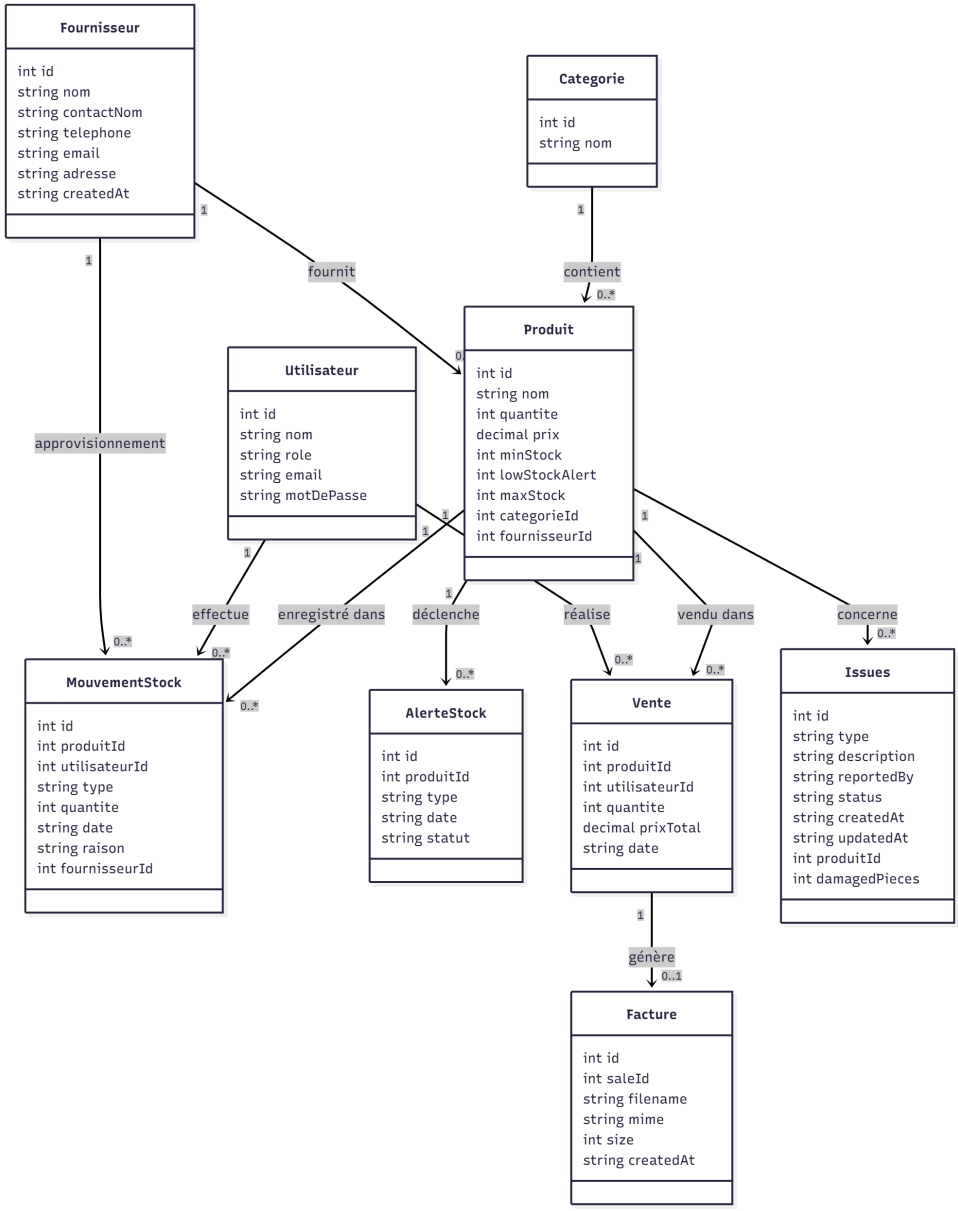

FIGURE 3.2 – Diagramme de classes : entités et associations.

3.3 Technologies utilisées

Frontend : React.js (SPA, hooks, React Router, services API).

Backend : Express.js (API REST, middleware).

Serveur local : XAMPP (Apache, MySQL).

Base de données : MySQL (tables `products`, `categories`, `stock_movements`, `users`).
Outils : GitHub (versionnement), Postman (tests API), VS Code (IDE).

Dépendances du workspace (racine)

Dépendances principales :

- `axios` : gestion des appels HTTP vers l'API.
- `cors` : configuration du Cross-Origin Resource Sharing.
- `react-router-dom` : navigation entre les pages de l'application React (SPA).

Dépendances de développement :

- `tailwindcss` : framework CSS utilitaire pour le prototypage rapide d'interfaces.

Dépendances front-end (`stock-manager`)

Dépendances principales :

- `react`, `react-dom` : bibliothèque et moteur de rendu pour l'interface utilisateur.
- `react-router-dom` : gestion des routes côté client.
- `axios` : appels HTTP vers le back-end Express.
- `firebase` : persistance et services potentiels (authentification, etc.).
- `chart.js` et `react-chartjs-2` : création de graphiques (courbes, barres, etc.) pour le tableau de bord.
- `react-scripts` : scripts de build, test et démarrage de l'application.
- `web-vitals` : mesure d'indicateurs de performance front-end.

Dépendances de test :

- `@testing-library/dom`, `@testing-library/jest-dom`, `@testing-library/react`, `@testing-library/u` tests unitaires et fonctionnels des composants React.

Dépendances de développement :

- `@tailwindcss/postcss7-compat` : compatibilité TailwindCSS avec PostCSS 7.
- `autoprefixer` : ajout automatique des préfixes navigateurs aux règles CSS.
- `postcss`, `postcss-flexbugs-fixes`, `postcss-preset-env` : pipeline de transformation CSS moderne et corrections de bugs de flexbox.
- `tailwindcss` : système de classes utilitaires pour le design.

Dépendances back-end (`inventory-sql-express-backend`)

Dépendances principales :

- `express` : framework minimaliste pour créer l'API REST.
- `dotenv` : gestion centralisée des variables d'environnement (configuration BDD, secrets, etc.).
- `bcryptjs` : hachage des mots de passe utilisateurs.
- `jsonwebtoken` : génération et vérification des tokens JWT pour l'authentification.
- `express-rate-limit` : limitation du nombre de requêtes (protection contre les attaques par force brute).

— `express-validator` : validation et sanitation des données reçues depuis le front-end.

Ces dépendances front-end et back-end structurent l'architecture technique globale : React pour l'interface, Express pour l'API sécurisée, MySQL pour la persistance, avec un ensemble de bibliothèques assurant la validation, la sécurité, les graphiques et la qualité de l'expérience utilisateur.

CHAPITRE 4

Réalisation Technique

Sommaire

4.1 Architecture du système

Architecture en couches :

- **React (client) :** interfaces, formulaires, tableau de bord, filtres.
- **Express (API REST) :** routes `/auth`, `/products`, `/categories`, `/movements`, gestion des erreurs.
- **MySQL (XAMPP) :** persistance des entités et contraintes relationnelles.

Flux : React → requêtes HTTP → Express → MySQL.

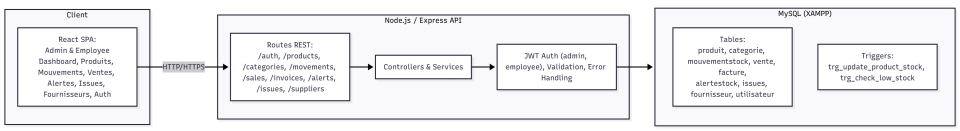

FIGURE 4.1 – Architecture globale : React → Express → MySQL.

4.2 Implémentation Front-End (React.js)

4.2.1 Structure du projet

- `src/components/` : tables, modals, alertes, formulaires réutilisables.

— `src/pages/` : Dashboard, Produits, Mouvements, Auth.
— `src/services/` : appels API (auth, products, categories, movements).
— `src/context/` : éventuel contexte d'authentification et gestion du token.

4.2.2 Interfaces principales

Dashboard : indicateurs clés (stock total, valeur estimée, alertes actives, derniers mouvements).
Produits : tableau filtrable (catégorie, seuil d'alerte, statut), CRUD complet.
Formulaire produit : saisie nom, référence, catégorie, stock initial, seuil, prix unitaire, statut.
Mouvements : journal daté avec type (entrée/sortie/transfert), quantités, utilisateur, filtres par date/type.
Authentification : formulaire de connexion, gestion d'état connecté/déconnecté.

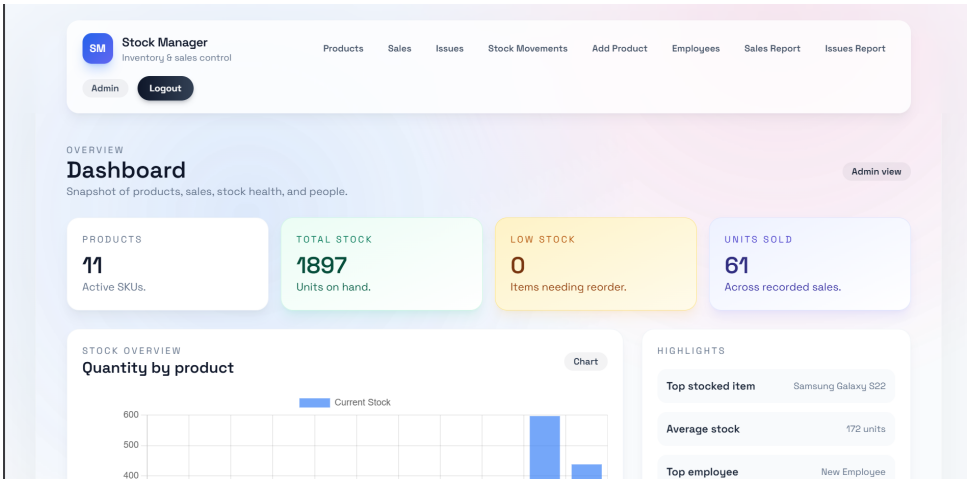

FIGURE 4.2 – Dashboard : synthèse et alertes.

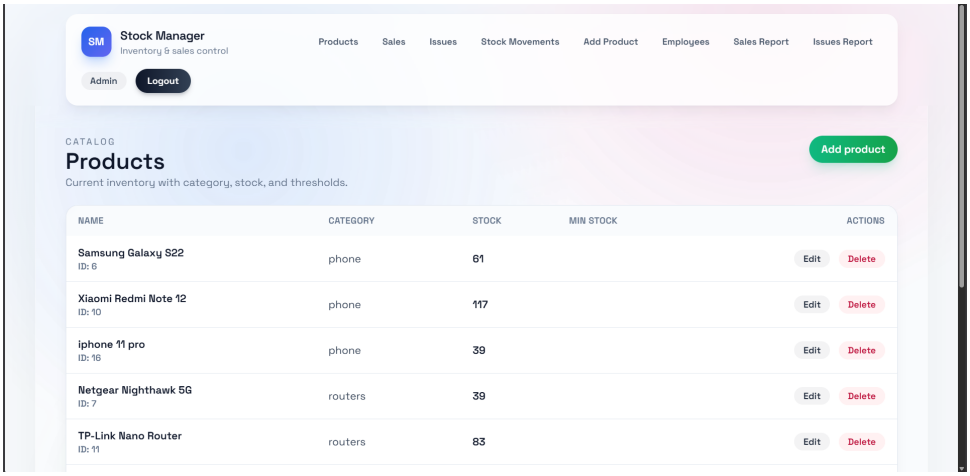

FIGURE 4.3 – Liste des produits : tableau filtrable, actions CRUD.

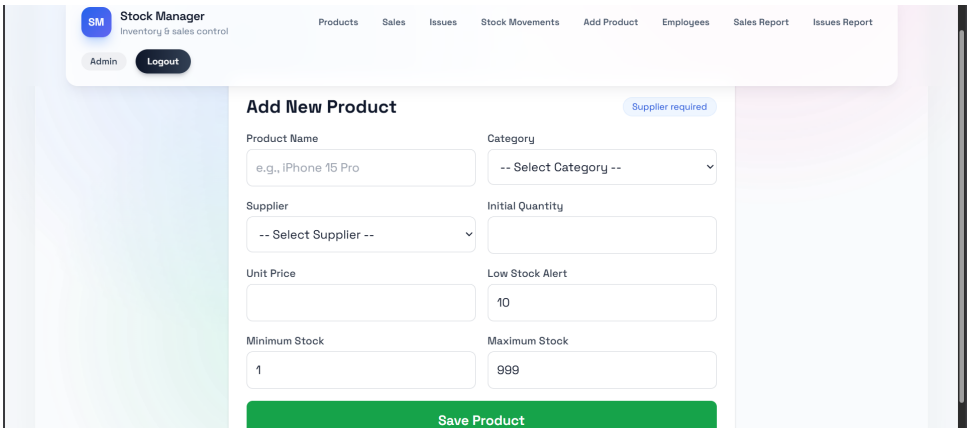

FIGURE 4.4 – Formulaire produit : création/édition validée.

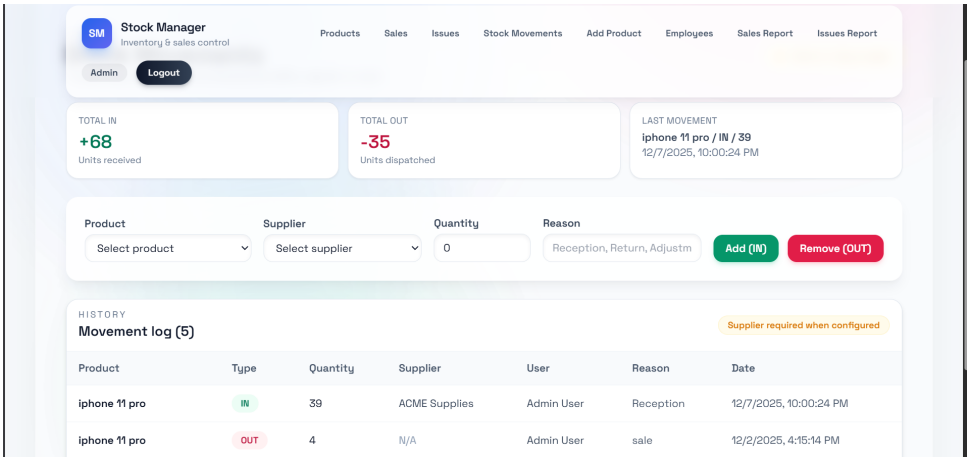

FIGURE 4.5 – Mouvements : historique des entrées/sorties/transferts.

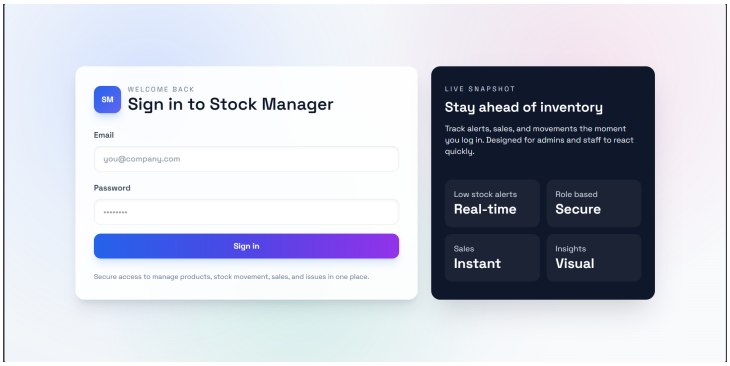

FIGURE 4.6 – Interface d'authentification de l'application.

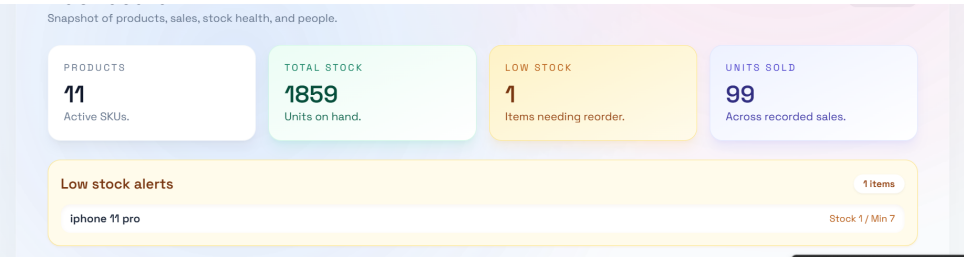

FIGURE 4.7 – Alertes de seuil de stock dans le tableau de bord.

4.2.3 Comportements clés

- Validation client (champs requis, formats numériques, seuil cohérent).
- États UI : chargement, succès, erreur ; notifications utilisateur.
- Navigation : React Router, redirection après connexion.
- Gestion du token : stockage local sécurisé (selon configuration), injection dans les requêtes.

4.3 Implémentation Back-End (Express.js)

4.3.1 Structure des dossiers

- `routes/` : `authRoutes`, `productRoutes`, `categoryRoutes`, `movementRoutes`.
- `controllers/` : logique métier (auth, CRUD produits/catégories, enregistrement mouvements).
- `models/` : requêtes SQL paramétrées pour MySQL.
- `middleware/` : authentification (JWT si activé), validation d'entrée, gestion centralisée des erreurs.

4.3.2 API REST (principales)

- `POST /auth/login` : authentification, retour token.
- `GET /products`, `POST /products`, `PUT /products/:id`, `DELETE /products/:id`.
- `GET /categories`, `POST /categories`, `PUT /categories/:id`, `DELETE /categories/:id`.
- `GET /movements`, `POST /movements` (types : entrée, sortie, transfert).

Filtres : pagination, recherche par libellé/référence, filtres par type de mouvement et date.

4.3.3 Authentification & sécurité

- Vérification des identifiants ; génération et validation du token.
- Middleware de protection pour les routes de mouvement et de gestion des produits.
- Validation des entrées serveur (quantités > 0, stock non négatif, références uniques si requis).

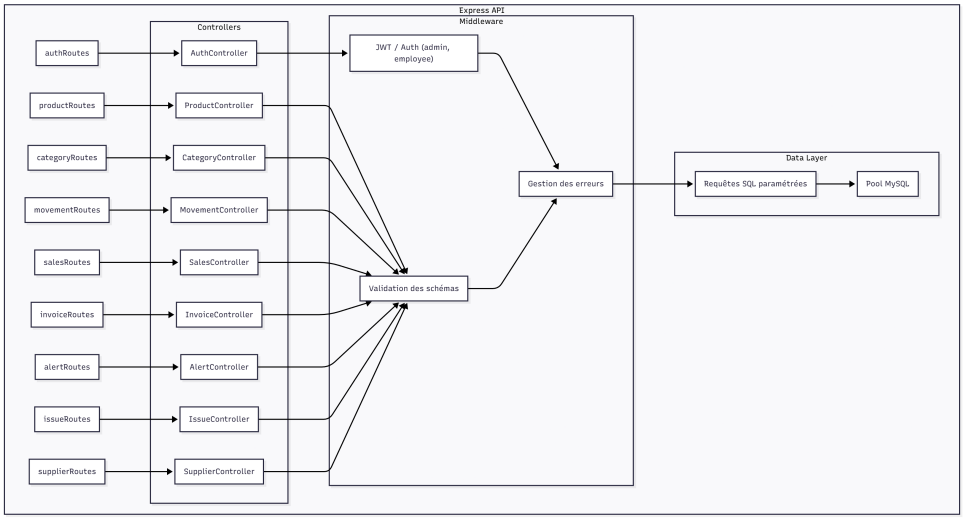

FIGURE 4.8 – Flux API : requêtes client, contrôleurs et accès à la base de données.

4.4 Base de données (MySQL via XAMPP)

4.4.1 Tables

— **products** : id, name, reference, category_id, stock, threshold, unit_price, status, created_at.
— **categories** : id, label, description.
— **stock_movements** : id, product_id, type, quantity, created_at, user_id, note.
— **users** : id, name, email, password_hash, role.

4.4.2 Relations et contraintes

— **categories** 1..* — **products**.
— **products** 1..* — **stock_movements**.
— **users** 1..* — **stock_movements** (auteur).
— Index : `product_id`, `category_id`, `type`, `created_at` pour accélérer les filtres.
— Contrôles : quantités positives, seuils par produit, stock non négatif sur sorties.

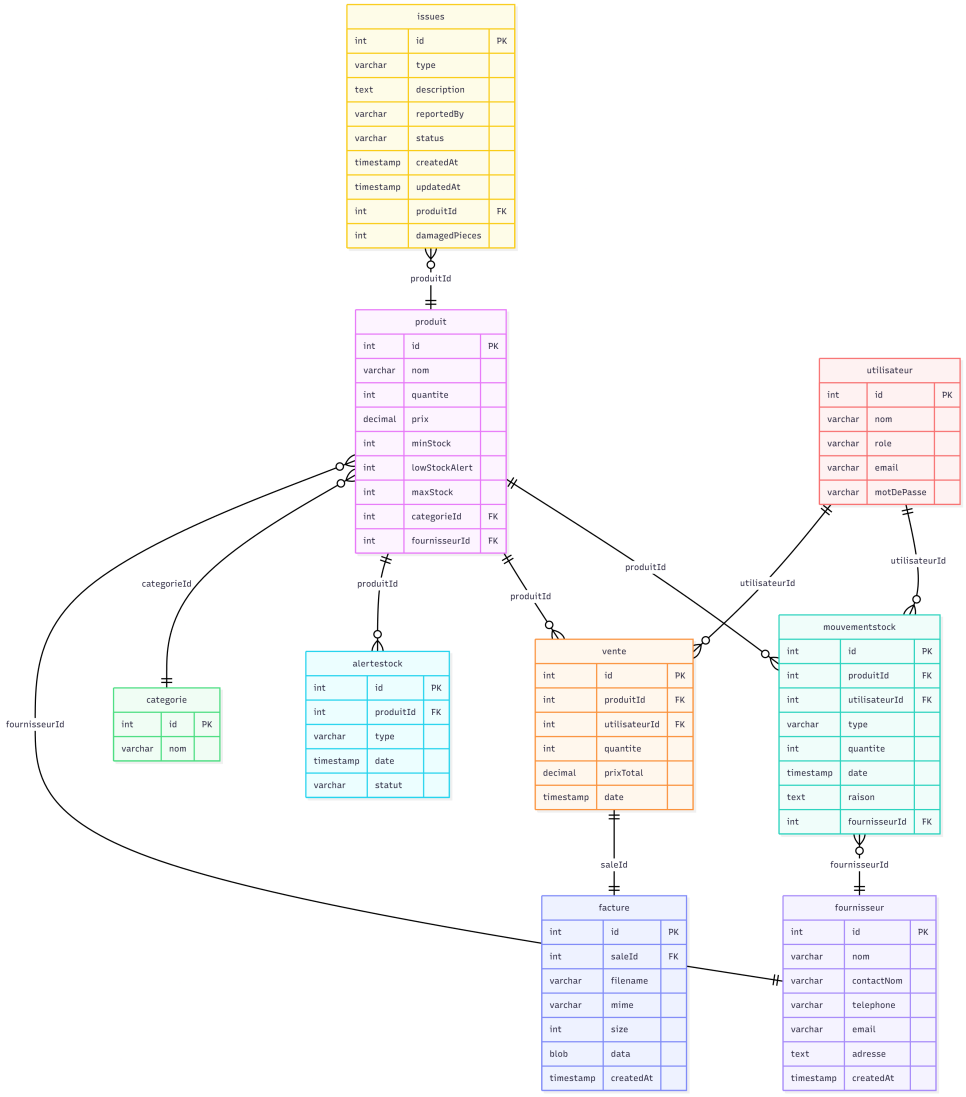

FIGURE 4.9 – Schéma relationnel : tables, clés étrangères et index.

4.5 Défis techniques & solutions

- **Synchronisation front/back :** standardisation des formats (dates, types), schémas de validation partagés.
- **Gestion des erreurs API :** middleware d'erreurs, codes HTTP cohérents, messages structurés.
- **Sécurité des opérations :** contrôle des sorties pour éviter stock négatif, authentification sur routes sensibles.
- **Optimisation BDD :** index ciblés, requêtes paramétrées, séparation produits/catégories.
- **UX et performance :** pagination côté serveur, filtres efficaces, loaders pour réduire l'attente perçue.

CHAPITRE 5

Conclusion et perspectives

Le stage a abouti à une application de gestion de stock couvrant l'ensemble des fonctionnalités attendues : authentification, gestion des produits et catégories, enregistrement des mouvements, alertes de seuil, tableau de bord et traçabilité. L'architecture React/Express/MySQL a favorisé la modularité et la maintenabilité. Les acquis incluent la modélisation UML, la sécurisation des flux et la validation métier.
Technologies apprises : React.js (routing, hooks, services API), Express.js (middleware, contrôleurs), MySQL (modélisation, indexation), Postman (tests), GitHub (versionnement).
Apports de l'expérience : maîtrise d'un flux full-stack, structuration d'un schéma relationnel orienté stock, rigueur de validation et de documentation.

Améliorations possibles

- Application mobile d'inventaire (scannage rapide).
- Intégration code-barres/QR pour accélérer les mouvements.
- Rôles avancés (admin/opérateur/auditeur) et audit renforcé.
- Notifications push/email sur seuils et prévisions de rupture.
- Tableau de bord analytique (prévision de demande, rotation, saisonnalité).