Université Cadi Ayyad

École Nationale des Sciences Appliquées de Marrakech

Filière : Génie Cyber-Défense et Systèmes de Télécommunications Embarqués

# Rapport de Projet

## StockPilot

Application web de gestion de stock

**Réalisé par :**

BADR Halima
BAGY Oussama
BENCHELHA Hidaya
BENSASSI Nour Asma
CHBANI Iliass
LAABOUKI Ikram

**Encadré par :**

Pr. NEJEOUI

**Année universitaire :**

2025/2026

# Dédicace

Nous dédions ce travail à nos familles, pour leur soutien constant, leur patience et leurs encouragements tout au long de notre parcours académique. Leur confiance nous a permis d'aborder ce projet avec sérieux et persévérance.

Nous dédions également ce rapport à nos enseignants, qui nous ont transmis les bases nécessaires en développement web, architecture logicielle, bases de données, sécurité applicative et conduite de projet. Les connaissances acquises durant notre formation ont été essentielles pour concevoir une solution complète de gestion de stock.

Enfin, nous dédions ce travail à toutes les personnes qui nous ont aidés directement ou indirectement, que ce soit par un conseil technique, une remarque constructive ou un soutien moral durant les différentes phases de réalisation.

# Remerciements

Nous adressons nos sincères remerciements à notre encadrant pour son accompagnement, ses orientations et ses remarques tout au long de la réalisation de ce projet. Ses recommandations nous ont aidés à structurer notre travail, à clarifier nos choix techniques et à améliorer progressivement la qualité de la solution.

Nous remercions également l'ensemble du corps professoral pour la qualité de la formation dispensée. Les enseignements en programmation orientée objet, bases de données, développement web, sécurité et génie logiciel nous ont permis de construire une application fonctionnelle, structurée et évolutive.

Nous remercions enfin tous les membres de l'équipe pour leur implication et leur esprit de collaboration. La réalisation de **StockPilot** a nécessité une coordination entre plusieurs aspects : modélisation des données, développement des contrôleurs, conception des interfaces, gestion de la sécurité, génération de documents PDF, tests et rédaction du rapport.

# Résumé

Le projet **StockPilot** consiste à concevoir et développer une application web de gestion de stock destinée à centraliser le suivi des produits, des catégories, des fournisseurs, des mouvements de stock, des alertes de seuil et des documents commerciaux. La solution répond à un besoin courant dans les organisations : disposer d'un outil simple, sécurisé et accessible permettant de connaître l'état du stock, d'anticiper les ruptures et de conserver une traçabilité des opérations.

L'application est réalisée avec Spring Boot, Thymeleaf, Spring Security, Spring Data JPA, H2, MySQL et Apache PDFBox. Elle adopte une architecture MVC organisée en couches, séparant les modèles de domaine, les repositories, les services métier, les contrôleurs et les templates d'interface. Cette séparation facilite la maintenance, l'évolution fonctionnelle et la validation des règles métier.

Les fonctionnalités principales incluent l'authentification administrateur, le tableau de bord de pilotage, le catalogue produit, la recherche, la création et la modification de fiches produits, la saisie de mouvements d'entrée, de sortie ou de livraison, la mise à jour automatique des quantités, la génération d'alertes lorsque le stock atteint le seuil défini, la gestion des catégories et fournisseurs ainsi que la création de documents commerciaux téléchargeables en PDF.

Le rapport présente le contexte général du projet, l'état de l'art des solutions de gestion de stock, l'analyse des besoins, la conception fonctionnelle, l'architecture technique, l'implémentation, les interfaces réalisées, les tests effectués et les perspectives d'amélioration. Les captures d'écran intégrées montrent les principaux écrans de l'application et illustrent le fonctionnement réel de la solution.

**Mots-clés :** gestion de stock, Spring Boot, Thymeleaf, JPA, H2, MySQL, Spring Security, PDFBox, alertes de seuil, application web.

# Abstract

The **StockPilot** project aims to design and implement a web-based inventory management application. Its purpose is to centralize the monitoring of products, categories, suppliers, stock movements, threshold alerts and commercial documents. The solution addresses a common operational need : providing a simple, secure and accessible tool to monitor inventory levels, anticipate shortages and maintain traceability of stock operations.

The application is developed using Spring Boot, Thymeleaf, Spring Security, Spring Data JPA, H2, MySQL and Apache PDFBox. It follows a layered MVC architecture that separates domain models, repositories, business services, controllers and user interface templates. This separation improves maintainability, scalability and the validation of business rules.

The main features include administrator authentication, an operational dashboard, product catalog management, search, product creation and update, stock movement registration, automatic quantity update, threshold-based alerts, category and supplier management, and commercial PDF document generation.

This report describes the project context, the state of the art, requirements analysis, functional design, technical architecture, implementation, user interfaces, validation tests and future improvements. The included screenshots illustrate the real behavior and visual identity of the application.

**Keywords :** inventory management, Spring Boot, Thymeleaf, JPA, H2, MySQL, Spring Security, PDFBox, threshold alerts, web application.

# Liste des acronymes

| **Acronyme** | Signification |
|---|---|
| **API** | Application Programming Interface, interface permettant à plusieurs composants logiciels d'échanger des données. |
| **CRUD** | Create, Read, Update, Delete, ensemble des opérations de base sur les données. |
| **CSS** | Cascading Style Sheets, langage de mise en forme des pages web. |
| **DTO** | Data Transfer Object, objet utilisé pour transporter des données entre couches ou services. |
| **HTML** | HyperText Markup Language, langage de structuration des pages web. |
| **HTTP** | HyperText Transfer Protocol, protocole utilisé pour les échanges entre clients et serveurs web. |
| **JPA** | Java Persistence API, spécification Java pour la persistance des objets. |
| **MVC** | Model View Controller, patron d'architecture séparant données, interface et logique de contrôle. |
| **ORM** | Object Relational Mapping, mécanisme permettant de relier les objets Java aux tables de base de données. |
| **PDF** | Portable Document Format, format de document portable. |
| **SGBD** | Système de gestion de base de données. |
| **SQL** | Structured Query Language, langage de manipulation des bases de données relationnelles. |
| **UI** | User Interface, interface utilisateur. |
| **UX** | User Experience, expérience utilisateur. |
| **WMS** | Warehouse Management System, système de gestion d'entrepôt. |

# Table des figures

# Liste des tableaux

# Table des matières

# Introduction générale

La gestion de stock constitue une activité essentielle pour toute organisation manipulant des produits physiques ou des articles consommables. Une mauvaise visibilité sur les quantités disponibles peut provoquer des ruptures, des retards de livraison, des immobilisations financières importantes ou des erreurs dans la planification des achats. À l'inverse, une solution numérique bien structurée permet de centraliser les informations, de suivre les flux et de prendre des décisions plus rapidement.

Dans ce contexte, le projet **StockPilot** propose une application web de gestion de stock orientée simplicité, traçabilité et pilotage quotidien. L'objectif n'est pas seulement de créer un catalogue de produits, mais de mettre en place un système cohérent capable de suivre les mouvements, d'actualiser automatiquement les quantités, de déclencher des alertes et de générer des documents commerciaux.

Le projet a été développé avec une stack Java moderne basée sur Spring Boot. Ce choix permet de bénéficier d'un environnement robuste, maintenable et adapté à la création d'applications web professionnelles. Thymeleaf assure la génération des interfaces côté serveur, Spring Security protège l'accès aux fonctionnalités, Spring Data JPA simplifie la persistance, H2 facilite les tests locaux et MySQL reste disponible pour un déploiement plus réaliste.

Ce rapport reprend une structure académique détaillée inspirée du template fourni. Il commence par le contexte général et les motivations du projet, puis présente l'état de l'art et les solutions retenues. Il développe ensuite les fondements méthodologiques, l'analyse des besoins, la conception fonctionnelle, l'architecture technique, l'implémentation et les interfaces. Enfin, il expose les tests effectués, les limites identifiées et les perspectives d'amélioration.

Notre objectif, en tant qu'équipe, a été de produire une solution équilibrée : assez complète pour couvrir un véritable processus de gestion de stock, mais suffisamment claire pour rester compréhensible, maintenable et démontrable. Nous avons donc privilégié une architecture simple, une interface cohérente et des règles métier placées côté serveur. Cette démarche nous a permis de travailler sur plusieurs compétences complémentaires : analyse, développement backend, interface utilisateur, persistance, sécurité, génération documentaire, tests et rédaction.

Le rapport met aussi en avant les aspects de collaboration. La solution finale est

le résultat d'un travail collectif : certaines décisions concernent la technique, d'autres concernent l'ergonomie, la documentation ou la validation. Nous avons voulu montrer cette dimension afin que le projet apparaisse comme une réalisation de groupe et non comme une simple liste de fonctionnalités.

Chapitre 1

# Contexte général du projet

## 1.1 Introduction

Ce chapitre présente le contexte général du projet **StockPilot**. Il met en évidence l'importance de la transformation numérique dans la gestion des stocks et les limites des méthodes classiques basées sur les fichiers dispersés ou les traitements manuels.

L'objectif est de montrer pourquoi une application web centralisée devient nécessaire pour suivre les produits, contrôler les quantités, gérer les mouvements, détecter les seuils critiques et produire des documents commerciaux. Ce chapitre présente également les objectifs du projet, son périmètre fonctionnel ainsi que la méthodologie adoptée pour sa réalisation.

## 1.2 Contexte et motivation

### 1.2.1 Transformation numérique des processus de stock

Les organisations cherchent de plus en plus à numériser leurs processus internes afin de réduire les erreurs manuelles, d'améliorer la disponibilité des informations et de faciliter la coordination entre les services. La gestion de stock fait partie des domaines où cette transformation apporte une valeur immédiate. Une application centralisée permet à l'équipe responsable du stock de consulter les quantités, de repérer les produits critiques, d'enregistrer les flux et de générer des documents sans passer par des fichiers dispersés.

Dans un fonctionnement manuel, les informations sont souvent fragmentées entre des feuilles de calcul, des notes papier, des échanges par messagerie ou des fichiers non synchronisés. Cette fragmentation crée plusieurs difficultés : manque de visibilité, doublons, oublis de mise à jour, absence de traçabilité et difficulté à justifier les opérations réalisées. Le passage vers une application web permet de concentrer les données dans une base unique et de rendre les actions plus fiables.

### 1.2.2 Enjeux liés à la gestion des stocks

Le stock représente une ressource stratégique. Il influence directement la continuité de service, la satisfaction des clients, les délais d'approvisionnement et le coût global de fonctionnement. Une quantité trop faible expose l'organisation à une rupture, tandis qu'une quantité excessive immobilise inutilement de la trésorerie et de l'espace de stockage.

Les enjeux principaux identifiés sont les suivants :

- connaître rapidement les produits disponibles et leurs quantités ;
- distinguer les articles disponibles des articles à réapprovisionner ;
- suivre les entrées, sorties et livraisons ;
- associer chaque produit à une catégorie et à un fournisseur ;
- conserver un historique des mouvements ;
- produire des documents commerciaux exploitables ;
- sécuriser l'accès aux données de gestion.

### 1.2.3 Limites des approches classiques

Les approches simples basées sur des fichiers Excel ou des documents papier peuvent suffire au départ, mais elles deviennent rapidement limitées. Elles ne garantissent pas toujours l'unicité de la donnée, ne contrôlent pas systématiquement les règles métier et ne produisent pas automatiquement des alertes.

Par exemple, lorsqu'une sortie de stock est saisie manuellement, rien n'empêche toujours de descendre sous zéro si aucune validation n'est prévue. De même, un produit peut atteindre un seuil critique sans qu'un responsable ne soit immédiatement informé. **StockPilot** répond à ces limites en intégrant les règles de contrôle directement dans la couche service.

## 1.3 Présentation générale du projet

### 1.3.1 Objectif global

L'objectif global du projet est de développer une application web de gestion de stock permettant de piloter un inventaire depuis une interface unique. L'application doit être suffisamment simple pour être utilisée au quotidien, tout en intégrant des mécanismes importants comme la sécurité, les alertes, la génération de PDF et la persistance des données.

### 1.3.2 Vision fonctionnelle

La solution est organisée autour de plusieurs modules complémentaires :

- un module d'authentification ;
- un tableau de bord synthétique ;
- un module produit ;
- un module catégorie ;
- un module fournisseur ;
- un module mouvement de stock ;
- un module alerte ;
- un module document commercial.

Chaque module contribue à la cohérence globale du système. Le produit constitue l'élément central. Les mouvements modifient ses quantités. Les alertes dépendent de son seuil. Les catégories et fournisseurs enrichissent son contexte. Les documents commerciaux permettent de formaliser certaines opérations.

### 1.3.3 Périmètre fonctionnel retenu

Le périmètre couvre les fonctionnalités nécessaires à une première version complète :

1. authentification d'un utilisateur administrateur ;
2. consultation des statistiques globales ;
3. recherche et affichage du catalogue produit ;
4. création et modification des produits ;
5. suppression sécurisée des produits et de leurs données liées ;
6. saisie des mouvements d'entrée, de sortie et de livraison ;
7. mise à jour automatique des quantités ;
8. génération et résolution des alertes de seuil ;
9. gestion des catégories et fournisseurs ;
10. génération de documents PDF.

## 1.4 Objectifs spécifiques

Les objectifs spécifiques du projet sont :

- appliquer une architecture MVC avec séparation claire des responsabilités ;
- utiliser Spring Boot comme socle applicatif ;

- sécuriser les routes avec Spring Security ;
- utiliser JPA pour la persistance des entités ;
- rendre l'interface claire, responsive et cohérente ;
- automatiser le calcul des alertes ;
- empêcher les mouvements créant un stock négatif ;
- proposer des PDF professionnels pour les documents commerciaux ;
- valider le bon démarrage de l'application par des tests.

## 1.5 Méthodologie adoptée

La démarche suivie repose sur une approche incrémentale :

1. identification du besoin et des acteurs ;
2. modélisation des entités principales ;
3. développement de la couche persistance ;
4. développement des services métier ;
5. création des contrôleurs web ;
6. intégration des templates Thymeleaf ;
7. ajout de la sécurité ;
8. génération des documents PDF ;
9. tests et validation ;
10. documentation et rédaction du rapport.

Cette démarche permet de construire progressivement une solution fonctionnelle en validant chaque module avant de passer au suivant.

## 1.6 Conclusion

Ce chapitre a permis de définir le cadre général du projet **StockPilot** et de justifier son intérêt dans un contexte de numérisation des processus de stock. Il a montré que la gestion manuelle peut entraîner des erreurs, un manque de visibilité et une faible traçabilité.

La solution proposée vise donc à centraliser les données, automatiser les contrôles, sécuriser l'accès et faciliter le suivi quotidien du stock. Les objectifs et la méthodologie présentés constituent ainsi une base claire pour les chapitres suivants, consacrés à l'analyse, la conception et l'implémentation de l'application.

Chapitre 2

# État de l'art et solutions retenues

## 2.1 Introduction

Ce chapitre présente l'état de l'art et les solutions retenues pour la réalisation de l'application **StockPilot**. Il permet de situer le projet par rapport aux systèmes existants de gestion de stock, tels que les ERP, les WMS et les applications spécialisées.

L'objectif est également de justifier les choix technologiques adoptés dans le projet. L'application repose sur une architecture web MVC et utilise des technologies adaptées comme Spring Boot, Thymeleaf, Spring Security, Spring Data JPA, H2, MySQL, Apache PDFBox et Bootstrap.

Ce chapitre montre donc que les solutions choisies répondent aux besoins du projet en termes de simplicité, sécurité, maintenabilité et évolutivité.

## 2.2 Gestion de stock et systèmes d'information

### 2.2.1 Rôle d'une application de stock

Une application de gestion de stock permet de centraliser les informations relatives aux articles, aux quantités, aux fournisseurs et aux mouvements. Elle joue un rôle de référentiel opérationnel. Dans un environnement professionnel, elle peut être reliée à d'autres systèmes comme la comptabilité, les achats, la vente ou la logistique.

Dans le cadre de ce projet, le système reste volontairement centré sur les fonctionnalités essentielles afin de produire une preuve de concept stable. Il ne s'agit pas d'un ERP complet, mais d'un outil modulaire pouvant évoluer vers un système plus large.

### 2.2.2 ERP, WMS et applications spécialisées

Les ERP couvrent plusieurs domaines de gestion comme les finances, les ressources humaines, les ventes et les stocks. Les WMS sont plus spécialisés dans la gestion d'entrepôt

et couvrent souvent les emplacements, les préparations de commandes, les codes-barres, les lots et les inventaires physiques.

**StockPilot** se positionne entre ces deux niveaux : il ne cherche pas à remplacer un ERP complet, mais propose une base applicative solide pour gérer un catalogue, des flux et des alertes.

## 2.3 Architecture web MVC

### 2.3.1 Principe du modèle MVC

Le modèle MVC sépare l'application en trois responsabilités :

- le modèle, qui représente les données et les règles métier ;
- la vue, qui présente les informations à l'utilisateur ;
- le contrôleur, qui reçoit les requêtes, appelle les services et choisit la vue.

Cette séparation améliore la lisibilité du code et facilite la maintenance. Dans **StockPilot**, les entités Java représentent le modèle, les templates Thymeleaf représentent les vues, et les contrôleurs Spring MVC assurent la coordination.

### 2.3.2 Intérêt pour le projet

Le choix MVC est pertinent pour une application de gestion car les pages sont principalement orientées formulaires, listes et tableaux. Thymeleaf permet de produire directement les vues HTML à partir du modèle transmis par les contrôleurs, ce qui simplifie l'intégration.

## 2.4 Technologies retenues

### 2.4.1 Spring Boot

Spring Boot est un framework Java qui simplifie la création d'applications web autonomes. Il fournit un serveur embarqué, une configuration automatique et une intégration simple avec Spring MVC, Spring Security et Spring Data JPA.

Dans ce projet, Spring Boot version 3.5.0 est utilisé. Le projet s'appuie sur Java 17 comme version cible déclarée dans le fichier Maven.

### 2.4.2 Thymeleaf

Thymeleaf est un moteur de templates utilisé pour générer les pages HTML côté serveur. Il permet d'écrire des expressions comme `th:text`, `th:each`, `th:href` ou `th:field`. Ces expressions relient les données Java aux éléments HTML.

### 2.4.3 Spring Security

Spring Security protège les routes de l'application. Le projet utilise un utilisateur administrateur en mémoire, dont les identifiants sont configurés dans `application.properties`. Les pages publiques sont limitées aux ressources statiques et à la page de connexion.

### 2.4.4 Spring Data JPA

Spring Data JPA fournit les repositories permettant d'effectuer les opérations de base sur les entités sans écrire manuellement toutes les requêtes SQL. Les interfaces héritent de `JpaRepository` et peuvent déclarer des méthodes comme `findAllByOrderByCreatedAtDesc`.

### 2.4.5 H2 et MySQL

H2 est utilisé par défaut pour un lancement local rapide. La base est configurée en mode fichier avec l'URL `jdbc:h2:file:./data/gestion-stock`. Une configuration MySQL est également préparée dans les commentaires du fichier de configuration afin de faciliter une migration vers un SGBD de production.

### 2.4.6 Apache PDFBox

Apache PDFBox est utilisé pour générer les documents commerciaux. La génération est codée côté service et produit un fichier PDF contenant l'en-tête, les blocs d'information, un tableau de détails, des observations, un cachet et une zone de signature.

### 2.4.7 Bootstrap et Bootstrap Icons

L'interface s'appuie sur Bootstrap pour les composants visuels, les grilles responsive et les formulaires. Bootstrap Icons est utilisé pour enrichir la navigation, les boutons et les indicateurs.

## 2.5 Justification des choix techniques

TABLE 2.1 – Justification des principales technologies retenues

| Technologie | Rôle | Justification |
|---|---|---|
| Spring Boot | Socle applicatif | Démarrage rapide, serveur embarqué, intégration native avec MVC, sécurité et JPA. |
| Thymeleaf | Vues HTML | Adapté aux applications serveur avec formulaires, tableaux et navigation simple. |
| Spring Security | Protection | Gestion fiable de l'authentification, sessions et restriction des routes. |
| JPA/Hibernate | Persistance | Mapping objet-relationnel, repositories simples, maintenance facilitée. |
| H2 | Base locale | Rapide à lancer, pratique pour les tests et les démonstrations. |
| MySQL | Base cible | Option plus réaliste pour un déploiement durable. |
| PDFBox | PDF | Génération programmatique sans dépendre d'un service externe. |

## 2.6 Conclusion

Ce chapitre a présenté les principales solutions existantes dans le domaine de la gestion de stock ainsi que les technologies retenues pour développer **StockPilot**. L'étude a montré que l'application se positionne comme une solution intermédiaire entre un outil simple de suivi de stock et un système plus complet de type ERP ou WMS.

Les choix techniques adoptés, notamment Spring Boot, Thymeleaf, Spring Security, Spring Data JPA et PDFBox, permettent de construire une application web structurée, sécurisée et évolutive. Ainsi, ces technologies constituent une base solide pour la conception et l'implémentation pratique du système.

Chapitre 3

# Fondements théoriques et méthodologiques

## 3.1 Introduction

Ce chapitre présente les fondements théoriques et méthodologiques sur lesquels repose l'application **StockPilot**. Avant de détailler l'implémentation technique, il est important de définir les notions essentielles liées à la gestion de stock, telles que le produit, la quantité disponible, le seuil d'alerte et les mouvements de stock.

Ce chapitre décrit également les principales règles métier appliquées dans l'application, notamment la détection du stock bas, la mise à jour automatique des quantités et la synchronisation des alertes. Enfin, il aborde les aspects liés à la traçabilité et à la sécurité applicative, qui garantissent un suivi fiable des opérations et un accès contrôlé aux fonctionnalités du système.

## 3.2 Notions de gestion de stock

### 3.2.1 Produit

Le produit est l'élément principal du système. Il possède une référence, un nom, une description, une image optionnelle, un prix unitaire, une quantité, un seuil d'alerte, une catégorie et un fournisseur. La référence permet d'identifier rapidement l'article dans le catalogue.

### 3.2.2 Quantité disponible

La quantité disponible représente le nombre d'unités actuellement présentes dans le stock. Elle est modifiée automatiquement lors de l'enregistrement d'un mouvement. Une entrée augmente la quantité, tandis qu'une sortie ou une livraison la diminue.

### 3.2.3 Seuil d'alerte

Le seuil d'alerte correspond à la quantité minimale acceptable. Lorsque la quantité disponible est inférieure ou égale à ce seuil, le produit est considéré comme critique. L'application crée alors une alerte active afin d'attirer l'attention de l'administrateur.

### 3.2.4 Mouvement de stock

Un mouvement est une opération qui modifie le niveau de stock. Dans l'application, trois types sont définis :

- `ENTREE` : ajout de quantité au stock ;
- `SORTIE` : retrait de quantité ;
- `LIVRAISON` : retrait lié à une livraison client.

## 3.3 Règles métier

### 3.3.1 Détection de stock bas

La règle de stock bas est définie dans l'entité `Product`. Elle compare la quantité au seuil d'alerte.

Listing 3.1 – Règle de détection du stock bas dans l'entité Product

```
public boolean isLowStock() {
    return quantity <= alertThreshold;
}
```

Cette méthode est simple, mais elle joue un rôle central. Elle est utilisée par le service d'alerte pour décider s'il faut créer, conserver ou résoudre une alerte.

### 3.3.2 Mise à jour des quantités

La mise à jour des quantités est réalisée dans le service `StockMovementService`. Le service récupère le produit, calcule la nouvelle quantité, vérifie qu'elle ne devient pas négative, enregistre le mouvement, sauvegarde le produit et synchronise les alertes.

Listing 3.2 – Principe de calcul d'un mouvement de stock

```
int newQuantity = product.getQuantity();
if (movement.getType() == MovementType.SORTIE
        || movement.getType() == MovementType.LIVRAISON) {
    newQuantity -= movement.getQuantity();
} else {
    newQuantity += movement.getQuantity();
```

```
}
if (newQuantity < 0) {
    throw new IllegalArgumentException("Stock insuffisant pour ce
        mouvement");
}
```

Cette règle protège l'application contre les sorties incohérentes et garantit que la quantité ne descend pas sous zéro.

### 3.3.3 Synchronisation des alertes

Le service `AlertService` applique une logique de synchronisation :

- si le produit est en stock bas et qu'aucune alerte active n'existe, une nouvelle alerte est créée ;
- si le produit n'est plus en stock bas et qu'une alerte active existe, elle est résolue ;
- si la situation n'a pas changé, aucune action inutile n'est effectuée.

Ce comportement évite la multiplication d'alertes identiques pour un même produit.

## 3.4 Traçabilité

La traçabilité est assurée par les mouvements et les documents. Chaque mouvement conserve le produit, le type, la quantité, la référence documentaire, la note et la date de création. Les documents commerciaux conservent le type, le numéro, le destinataire, la description et la date.

## 3.5 Sécurité applicative

### 3.5.1 Authentification

L'application utilise une authentification par formulaire. L'utilisateur doit se connecter avant d'accéder aux fonctionnalités. Le compte de démonstration est configuré comme suit :

TABLE 3.1 – Compte de test configuré

| Paramètre | Valeur |
|---|---|
| Utilisateur | `admin` |
| Mot de passe | `admin123` |
| Rôle | `ADMIN` |

### 3.5.2 Contrôle d'accès

Les routes relatives aux produits, catégories, fournisseurs, mouvements, alertes et documents nécessitent le rôle `ADMIN`. Les ressources statiques restent accessibles afin d'afficher correctement la page de connexion.

### 3.5.3 Gestion de session

La configuration active la protection des cookies HTTP-only, le mode SameSite strict, la migration de session après authentification et la limitation à une session active par utilisateur.

## 3.6 Conclusion

Ce chapitre a permis de présenter les notions fondamentales nécessaires à la compréhension du fonctionnement de **StockPilot**. Les concepts de produit, quantité disponible, seuil d'alerte et mouvement de stock constituent la base du système de gestion.

Les règles métier définies assurent la cohérence des opérations, notamment en empêchant les stocks négatifs et en déclenchant les alertes lorsque le niveau de stock devient critique. La traçabilité des mouvements et la sécurité applicative renforcent également la fiabilité du système.

Ainsi, ces fondements théoriques et méthodologiques préparent la suite du projet, en servant de base à la conception technique et à l'implémentation pratique de l'application.

Chapitre 4

# Analyse des besoins et conception fonctionnelle

## 4.1 Introduction

Ce chapitre présente l'analyse des besoins et la conception fonctionnelle de l'application **StockPilot**. Cette étape est essentielle avant le développement, car elle permet d'identifier les utilisateurs du système, les fonctionnalités attendues, les contraintes à respecter et les règles de gestion nécessaires au bon fonctionnement de l'application.

L'objectif principal de **StockPilot** est de proposer une solution simple, sécurisée et efficace pour la gestion de stock. L'application doit permettre à l'utilisateur de suivre les produits, gérer les mouvements de stock, surveiller les alertes de seuil, organiser les catégories et les fournisseurs, ainsi que générer des documents commerciaux.

Ce chapitre décrit donc les acteurs concernés, les besoins fonctionnels et non fonctionnels, les principaux cas d'utilisation ainsi que les règles de gestion appliquées dans le système.

## 4.2 Acteurs du système

Table 4.1 – Acteurs identifiés

| Acteur | Description |
|---|---|
| Administrateur | Utilisateur principal de l'application. Il gère les produits, catégories, fournisseurs, mouvements, alertes et documents. |
| Responsable stock | Profil métier chargé de surveiller les quantités, d'anticiper les ruptures et de saisir les flux. Dans cette version, il est représenté par le compte administrateur. |
| Service commercial | Utilisateur indirect intéressé par les documents générés, tels que factures et bons. |
| Direction | Utilisateur indirect consultant les indicateurs de valeur de stock, alertes et volumes. |

## 4.3 Besoins fonctionnels

### 4.3.1 Authentification

Le système doit empêcher l'accès aux données internes sans authentification. La page de connexion doit être claire et orientée administrateur.

### 4.3.2 Tableau de bord

Le tableau de bord doit fournir une vue synthétique :

- nombre de produits suivis ;
- valeur globale du stock ;
- nombre de fournisseurs ;
- nombre d'alertes actives ;
- derniers mouvements ;
- alertes actives.

### 4.3.3 Gestion des produits

L'utilisateur doit pouvoir consulter, rechercher, créer, modifier et supprimer un produit. La liste doit distinguer visuellement les produits disponibles et les produits à réapprovisionner.

### 4.3.4 Gestion des mouvements

L'utilisateur doit pouvoir saisir un mouvement en choisissant un produit, un type, une quantité, une référence documentaire et une note. Le système doit mettre à jour la quantité et empêcher un stock négatif.

### 4.3.5 Gestion des alertes

L'utilisateur doit pouvoir consulter les alertes, connaître le produit concerné, lire le message, voir le statut et résoudre une alerte lorsque le problème a été traité.

### 4.3.6 Gestion documentaire

L'utilisateur doit pouvoir créer un document commercial en choisissant son type, son numéro, son destinataire et sa description. Il doit ensuite pouvoir télécharger le PDF généré.

## 4.4 Besoins non fonctionnels

Table 4.2 – Besoins non fonctionnels

| Besoin | Description |
|---|---|
| Sécurité | Accès restreint par authentification et rôle administrateur. |
| Fiabilité | Règles métier appliquées côté serveur, notamment pour le stock négatif. |
| Maintenabilité | Architecture en couches et classes dédiées par responsabilité. |
| Lisibilité | Interface claire avec tableaux, badges, formulaires structurés et navigation persistante. |
| Évolutivité | Possibilité d'ajouter des rôles, des inventaires, des exports ou une base MySQL. |
| Traçabilité | Historique des mouvements et documents archivés. |

## 4.5 Cas d'utilisation principaux

Table 4.3: Cas d'utilisation principaux

| Cas d'utilisation | Précondition | Résultat attendu |
|---|---|---|
| Se connecter | L'utilisateur possède les identifiants | Accès au tableau de bord. |
| Consulter le tableau de bord | Utilisateur authentifié | Affichage des indicateurs. |
| Créer un produit | Catégories/fournisseurs disponibles ou optionnels | Produit ajouté au catalogue. |
| Modifier un produit | Produit existant | Informations mises à jour. |
| Supprimer un produit | Produit existant | Produit et données liées supprimés. |
| Saisir une entrée | Produit existant | Quantité augmentée et alerte synchronisée. |
| Saisir une sortie | Stock suffisant | Quantité diminuée et mouvement historisé. |
| Consulter les alertes | Alertes existantes ou non | Liste des alertes avec statut. |
| Résoudre une alerte | Alerte active | Alerte marquée comme résolue. |
| Créer un document | Utilisateur authentifié | Document archivé. |
| Télécharger un PDF | Document existant | Fichier PDF retourné. |

## 4.6 Règles de gestion

Table 4.4 – Règles de gestion principales

| Code | Règle |
|---|---|
| RG01 | Un produit doit avoir une référence et un nom. |
| RG02 | Le prix unitaire, la quantité et le seuil doivent être positifs ou nuls. |
| RG03 | Un mouvement doit avoir un produit, un type et une quantité supérieure à zéro. |
| RG04 | Une sortie ou une livraison ne peut pas produire une quantité négative. |
| RG05 | Une alerte active est créée lorsque la quantité est inférieure ou égale au seuil. |
| RG06 | Une alerte active est résolue automatiquement lorsque le produit repasse au-dessus du seuil. |
| RG07 | La suppression d'un produit supprime d'abord ses alertes et mouvements associés. |
| RG08 | Les documents PDF sont générés côté serveur à partir des données archivées. |

## 4.7 Conclusion

Ce chapitre a permis de définir les besoins principaux de l'application **StockPilot** ainsi que sa conception fonctionnelle. L'analyse a identifié les différents acteurs du système, les fonctionnalités attendues, les contraintes non fonctionnelles, les cas d'utilisation et les règles de gestion.

Cette étape montre que l'application doit répondre à des besoins concrets : sécuriser l'accès, gérer les produits, suivre les mouvements, prévenir les ruptures, organiser les fournisseurs et catégories, puis générer des documents commerciaux.

Ainsi, cette analyse constitue une base solide pour la conception technique et l'implémentation de l'application. Elle permet de garantir que le développement répondra aux attentes fonctionnelles et aux contraintes métier liées à la gestion de stock.

Chapitre 5

# Conception et architecture du système

## 5.1 Introduction

Ce chapitre présente la conception et l'architecture générale de l'application **StockPilot**. Il décrit l'organisation du système, les principales couches techniques, le modèle de données, les routes web et le flux de traitement des mouvements de stock.

L'objectif est de montrer comment les différents composants de l'application sont structurés afin d'assurer une gestion claire, cohérente et évolutive du stock.

## 5.2 Objectif du chapitre

Ce chapitre présente la conception générale de l'application **StockPilot**. L'objectif est de montrer l'organisation du système, les principales couches techniques, les entités métier importantes et le flux de traitement d'une opération de stock.

L'application repose sur une architecture en couches afin de séparer clairement :

- l'interface utilisateur ;
- les contrôleurs web ;
- les services métier ;
- l'accès aux données ;
- la base de données.

Cette séparation rend le projet plus lisible, plus maintenable et plus facile à faire évoluer.

## 5.3 Architecture globale

L'application suit une architecture MVC enrichie par une couche service. Les templates Thymeleaf assurent l'affichage, les contrôleurs Spring MVC reçoivent les requêtes, les

services appliquent les règles métier, les repositories communiquent avec la base de données et les entités représentent le modèle métier.

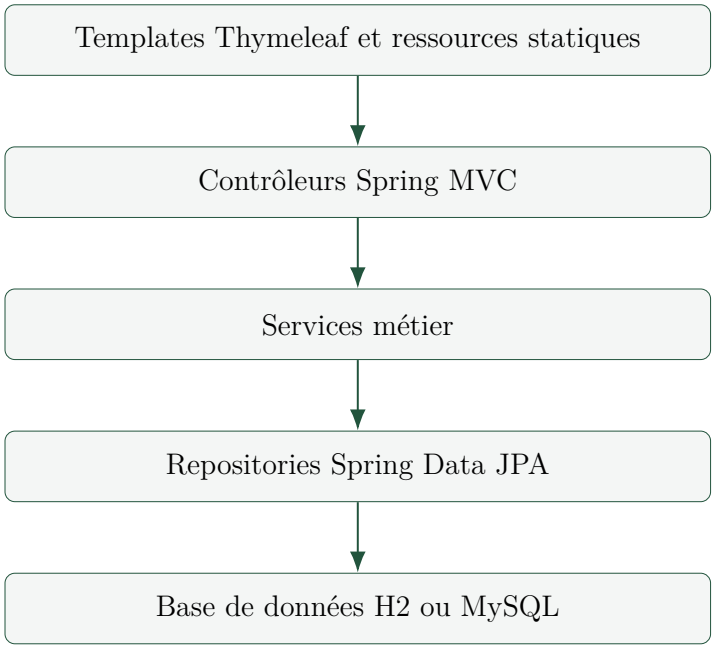


FIGURE 5.1 – Architecture générale en couches de StockPilot

## 5.4 Organisation du projet

Le projet est organisé par responsabilité. Chaque package contient un type précis de composants.

TABLE 5.1: Organisation logique du projet

| Package ou dossier | Rôle principal |
|---|---|
| `config` | Configuration de sécurité et initialisation des données. |
| `controller` | Gestion des routes web et coordination entre vues et services. |
| `model` | Entités JPA et énumérations métier. |
| `repository` | Interfaces Spring Data JPA pour l'accès aux données. |
| `service` | Règles métier, traitement du stock, alertes et génération PDF. |
| `templates` | Pages HTML Thymeleaf affichées à l'utilisateur. |
| `static` | Fichiers CSS, JavaScript et ressources statiques. |

## 5.5 Fonctionnalités principales

L'administrateur est l'acteur principal du système. Il peut gérer les produits, les catégories, les fournisseurs, les mouvements de stock, les alertes et les documents commerciaux.

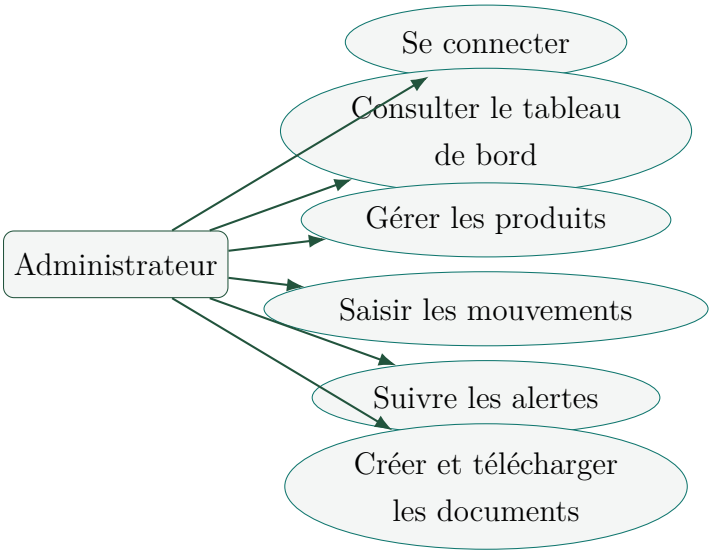


FIGURE 5.2 – Cas d'utilisation principaux de l'application

## 5.6 Modèle de données

### 5.6.1 Entités principales

Le modèle métier est centré sur le produit. Les autres entités permettent de le classer, de l'associer à un fournisseur, de suivre ses mouvements et de générer des alertes lorsque le stock devient critique.

TABLE 5.2: Description simplifiée des entités

| Entité | Attributs principaux | Rôle |
|---|---|---|
| `Product` | référence, nom, description, image, prix, quantité, seuil | Représente un article du stock. |
| `Category` | nom, description | Classe les produits par famille. |
| `Supplier` | nom, email, téléphone, adresse | Représente le fournisseur d'un produit. |
| `StockMovement` | produit, type, quantité, référence, note, date | Trace les entrées, sorties et livraisons. |

| | | |
|---|---|---|
| `StockAlert` | produit, message, statut, dates | Signale un produit en stock critique. |
| `CommercialDocument` | type, numéro, client, description, date | Archive les documents commerciaux. |

### 5.6.2 Relations entre les entités

Un produit peut appartenir à une catégorie et être lié à un fournisseur. Chaque mouvement de stock concerne un produit. Chaque alerte concerne aussi un produit. Les documents commerciaux restent indépendants dans cette version afin de simplifier leur création et leur téléchargement.

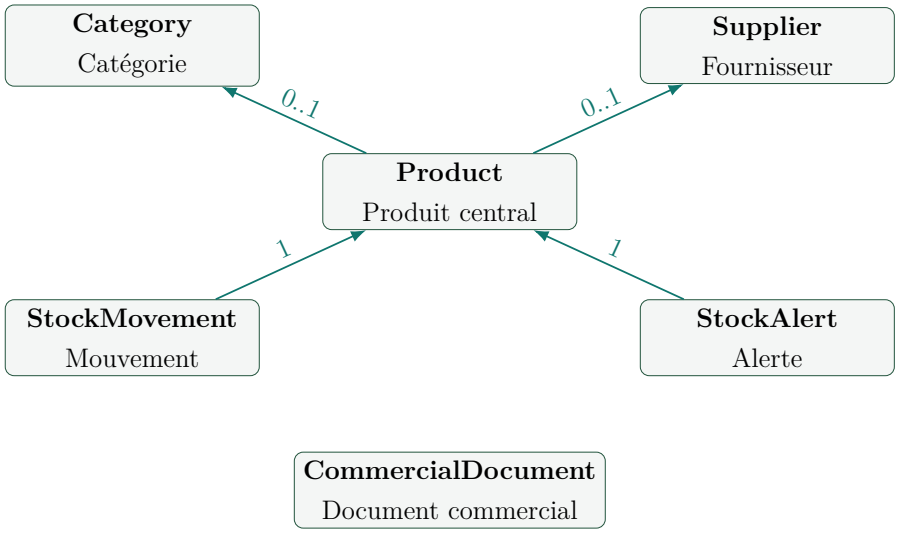


FIGURE 5.3 – Vue simplifiée du modèle de données

## 5.7 Routes principales

Les routes suivantes représentent les points d'entrée web essentiels de l'application.

TABLE 5.3: Routes web principales

| Route | Méthode | Rôle |
|---|---|---|
| `/login` | GET/POST | Affichage et traitement de la connexion. |
| `/` | GET | Affichage du tableau de bord. |
| `/products` | GET/POST | Liste, recherche et ajout des produits. |
| `/products/{id}/edit` | GET | Formulaire de modification d'un produit. |
| `/products/{id}/delete` | POST | Suppression d'un produit. |

| | | |
|---|---|---|
| `/movements` | GET/POST | Consultation et saisie des mouvements de stock. |
| `/alerts` | GET | Liste des alertes de stock. |
| `/alerts/{id}/resolve` | POST | Résolution d'une alerte. |
| `/documents` | GET/POST | Création et affichage des documents commerciaux. |
| `/documents/{id}/pdf` | GET | Téléchargement d'un document au format PDF. |
| `/categories` | GET/POST | Gestion des catégories. |
| `/suppliers` | GET/POST | Gestion des fournisseurs. |

## 5.8 Flux de traitement d'un mouvement de stock

Le mouvement de stock est le traitement métier le plus important. Lorsqu'un utilisateur saisit une entrée, une sortie ou une livraison, le système calcule la nouvelle quantité, vérifie que le stock ne devient pas négatif, sauvegarde l'opération puis met à jour les alertes.

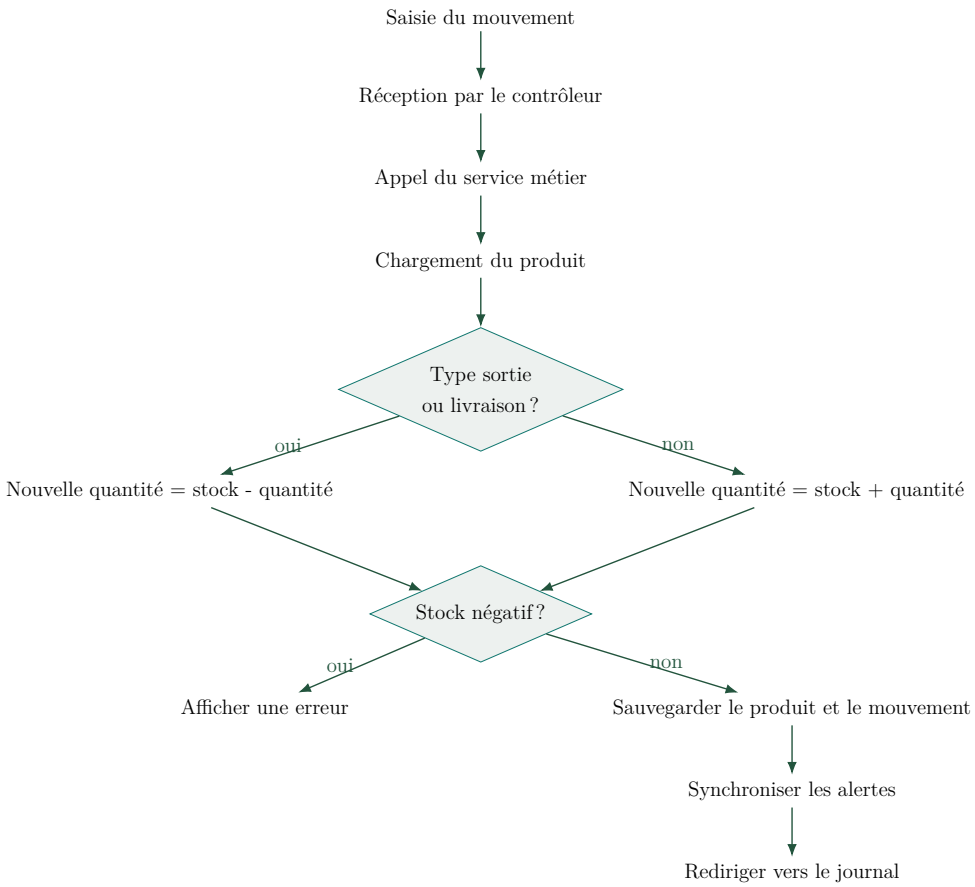


FIGURE 5.4 – Flux de traitement d'un mouvement de stock

## 5.9 Déploiement du système

En mode local, l'application est exécutée comme une application Spring Boot avec un serveur Tomcat embarqué. Elle est accessible depuis un navigateur web et communique avec une base H2 ou MySQL via JDBC.

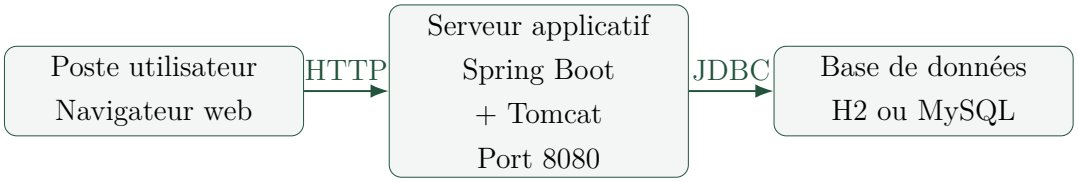


FIGURE 5.5 – Déploiement simplifié de l'application

## 5.10 Synthèse de conception

La conception de **StockPilot** est basée sur une architecture simple et claire. Les contrôleurs gèrent les requêtes web, les services concentrent les règles métier, les repositories assurent l'accès aux données et les templates affichent les pages.

Les principaux choix de conception sont les suivants :

- séparation claire entre interface, logique métier et persistance ;
- centralisation des règles de stock dans les services ;
- utilisation de Spring Data JPA pour simplifier l'accès aux données ;
- génération des documents PDF dans un service dédié ;
- modèle de données centré sur le produit et les mouvements de stock.

Cette organisation permet de faire évoluer l'application facilement, par exemple en ajoutant de nouveaux rôles, un module d'inventaire ou une API REST.

## 5.11 Conclusion

Ce chapitre a présenté l'architecture de **StockPilot** ainsi que les principaux choix de conception adoptés. L'application repose sur une séparation claire entre les vues, les contrôleurs, les services, les repositories et la base de données.

Cette organisation rend le système plus lisible, maintenable et évolutif. Elle permet aussi de gérer efficacement les produits, les mouvements, les alertes, les fournisseurs, les catégories et les documents commerciaux.

Chapitre 6

# Implémentation pratique

## 6.1 Introduction

Ce chapitre présente l'implémentation pratique de l'application **StockPilot**. Il décrit la configuration du projet, les principaux modules développés et les technologies utilisées, notamment Spring Boot, Maven, Thymeleaf, JPA, Spring Security et PDFBox.

L'objectif est de montrer comment les fonctionnalités de gestion de stock ont été réalisées techniquement, depuis la gestion des produits jusqu'à la génération des documents PDF.

## 6.2 Configuration Maven

Le fichier `pom.xml` déclare les dépendances nécessaires :

- `spring-boot-starter-web` pour le web MVC ;
- `spring-boot-starter-thymeleaf` pour les vues ;
- `spring-boot-starter-security` pour la sécurité ;
- `spring-boot-starter-data-jpa` pour la persistance ;
- `spring-boot-starter-validation` pour la validation ;
- `h2` et `mysql-connector-j` pour les bases de données ;
- `pdfbox` pour la génération PDF ;
- `spring-boot-starter-test` et `spring-security-test` pour les tests.

## 6.3 Configuration applicative

Le fichier `application.properties` définit le port serveur, le compte applicatif, la base H2, la console H2 et les options JPA.

Listing 6.1 – Configuration principale de l'application

```
spring.application.name=gestion-stock
server.port=8080

app.security.username=admin
app.security.password=admin123

spring.datasource.url=jdbc:h2:file:./data/gestion-stock
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
```

## 6.4 Initialisation des données

Une classe `DataInitializer` insère des données de démonstration si la base ne contient aucun produit. Les données initiales incluent deux catégories, un fournisseur et deux produits. L'un des produits est volontairement proche de la rupture afin de déclencher une alerte.

Table 6.1 – Produits de démonstration

| Référence | Nom | Catégorie | Fournisseur | Quantité |
|---|---|---|---|---|
| PRD-001 | Clavier mécanique | Informatique | Atlas Distribution | 25 |
| PRD-002 | Toner imprimante | Bureau | Atlas Distribution | 3 |

## 6.5 Implémentation du module produit

Le module produit repose sur l'entité `Product`, le `ProductRepository`, le `ProductController`, le `ProductService` et les vues `products/list.html` et `products/form.html`.

La liste des produits affiche la référence, le nom, la catégorie, le fournisseur, le prix, le stock et le statut. Les lignes en alerte sont mises en évidence par une couleur spécifique. La recherche permet de filtrer par nom ou référence.

La suppression d'un produit passe par le service `ProductService`, qui supprime d'abord les alertes et mouvements liés. Ce choix évite les incohérences de clé étrangère.

## 6.6 Implémentation du module mouvement

Le module mouvement permet de saisir les flux opérationnels. Il utilise l'énumération `MovementType` avec les valeurs `ENTREE`, `SORTIE` et `LIVRAISON`. La logique métier est

centralisée dans `StockMovementService`.

Lorsqu'un mouvement est enregistré, le service :

1. récupère le produit depuis la base ;
2. calcule la nouvelle quantité ;
3. refuse l'opération si le stock devient négatif ;
4. associe le produit réel au mouvement ;
5. sauvegarde le mouvement ;
6. sauvegarde la quantité mise à jour ;
7. synchronise les alertes.

## 6.7 Implémentation du module alerte

Le module alerte affiche les alertes dans l'ordre chronologique décroissant. Une alerte contient un message, un statut résolu/non résolu, une date de création et éventuellement une date de résolution.

L'alerte active de démonstration correspond au produit `Toner imprimante`, dont la quantité est inférieure au seuil.

## 6.8 Implémentation du module document

Le module document permet de créer des pièces commerciales de type :

- `BON_ENTREE` ;
- `BON_SORTIE` ;
- `BON_LIVRAISON` ;
- `FACTURE`.

Le contrôleur retourne les fichiers PDF via un `ResponseEntity<byte[]>`. Le type MIME est `application/pdf`, et l'en-tête `Content-Disposition` force le téléchargement.

## 6.9 Génération PDF

La génération PDF utilise PDFBox. Le service crée un document A4, dessine un en-tête vert, ajoute les informations de l'émetteur et du document, construit un tableau de détails, puis ajoute les observations, le cachet et la signature.

Cette approche est entièrement côté serveur. Elle ne dépend pas d'un navigateur ni d'une conversion HTML vers PDF. Elle permet donc de contrôler précisément la structure du document.

## 6.10 Implémentation de l'interface

L'interface repose sur un fragment de layout contenant l'en-tête, la marque **StockPilot**, la barre de recherche visuelle, l'utilisateur connecté, le bouton de déconnexion et la navigation principale.

Le fichier CSS personnalise fortement l'apparence :

- palette verte et dorée adaptée au domaine logistique ;
- cartes d'indicateurs ;
- badges de statut ;
- tableaux lisibles ;
- formulaires structurés ;
- page de connexion avec fond illustratif ;
- navigation responsive.

## 6.11 Conclusion

Ce chapitre a présenté la réalisation technique de l'application **StockPilot**. Les différents modules ont été implémentés de manière organisée à travers les entités, services, contrôleurs, repositories et vues.

Ainsi, l'application obtenue est fonctionnelle, structurée et adaptée à la gestion de stock, avec des fonctionnalités essentielles comme la sécurité, le suivi des mouvements, les alertes et la génération PDF.

Chapitre 7

# Présentation de l'interface utilisateur et parcours fonctionnel

## 7.1 Introduction

Ce chapitre présente les principales interfaces de l'application **StockPilot**. L'objectif est de montrer le parcours utilisateur depuis la connexion jusqu'à la gestion complète du stock. Chaque capture d'écran illustre une fonctionnalité importante : authentification, tableau de bord, gestion des produits, mouvements de stock, alertes, documents commerciaux, catégories et fournisseurs.

Ces interfaces ont été conçues pour offrir une navigation simple, une lecture rapide des informations et une gestion efficace des opérations courantes liées au stock.

## 7.2 Page de connexion

La page de connexion constitue le point d'entrée sécurisé de l'application. Elle permet de limiter l'accès aux fonctionnalités internes uniquement à l'administrateur autorisé. Cette page met également en avant l'identité visuelle de **StockPilot** à travers le nom de l'application, les couleurs utilisées et une présentation claire du formulaire d'authentification.

L'utilisateur doit saisir ses identifiants avant d'accéder au tableau de bord. Cette étape garantit que les données de stock, les fournisseurs, les mouvements et les documents commerciaux ne sont pas accessibles publiquement.

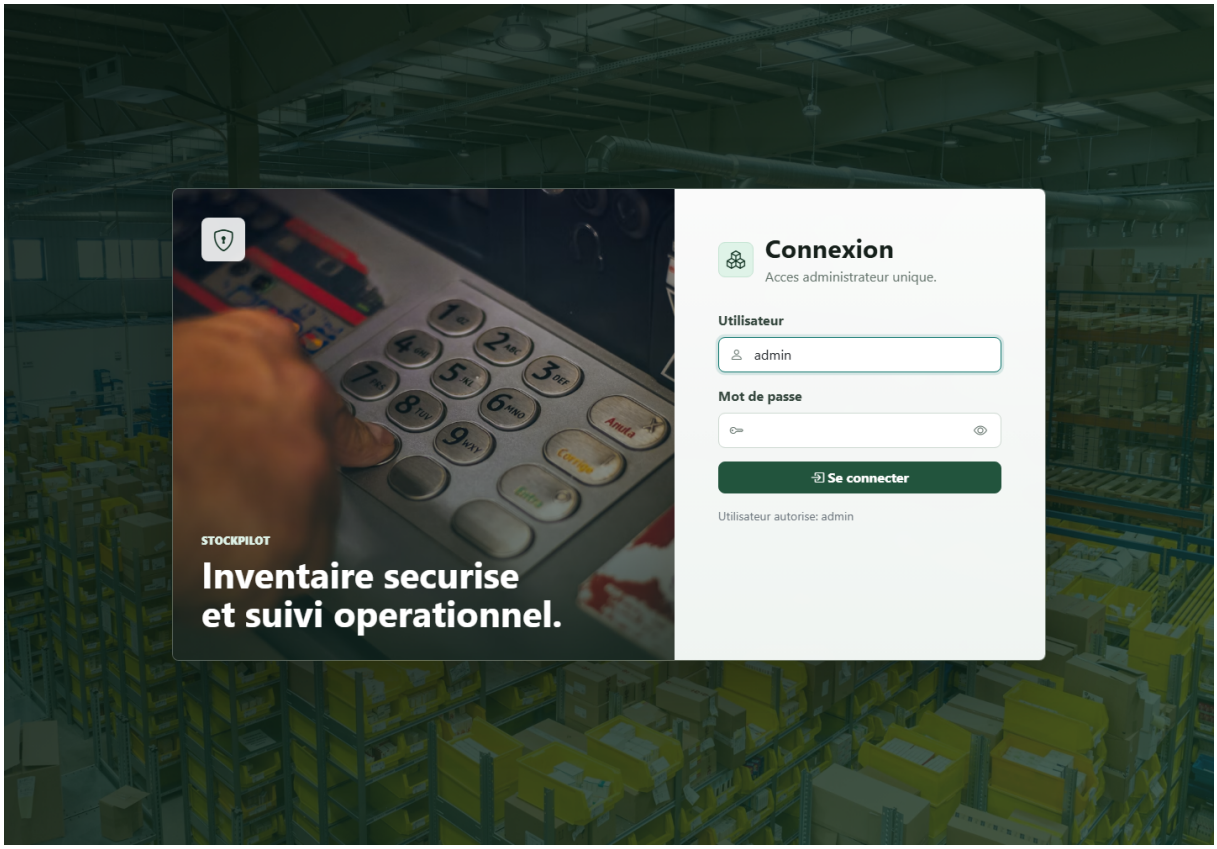


FIGURE 7.1 – Page de connexion sécurisée de StockPilot

## 7.3 Tableau de bord

Après authentification, l'utilisateur accède au tableau de bord. Cette page donne une vision globale et rapide de l'état du stock. Elle regroupe les indicateurs essentiels tels que le nombre total de produits, la valeur du stock, le nombre de fournisseurs et le nombre d'alertes actives.

Le tableau de bord affiche aussi les derniers mouvements enregistrés et les produits qui nécessitent une attention particulière. Cette interface joue donc un rôle de synthèse : elle permet à l'administrateur de comprendre immédiatement la situation générale sans consulter chaque module séparément.

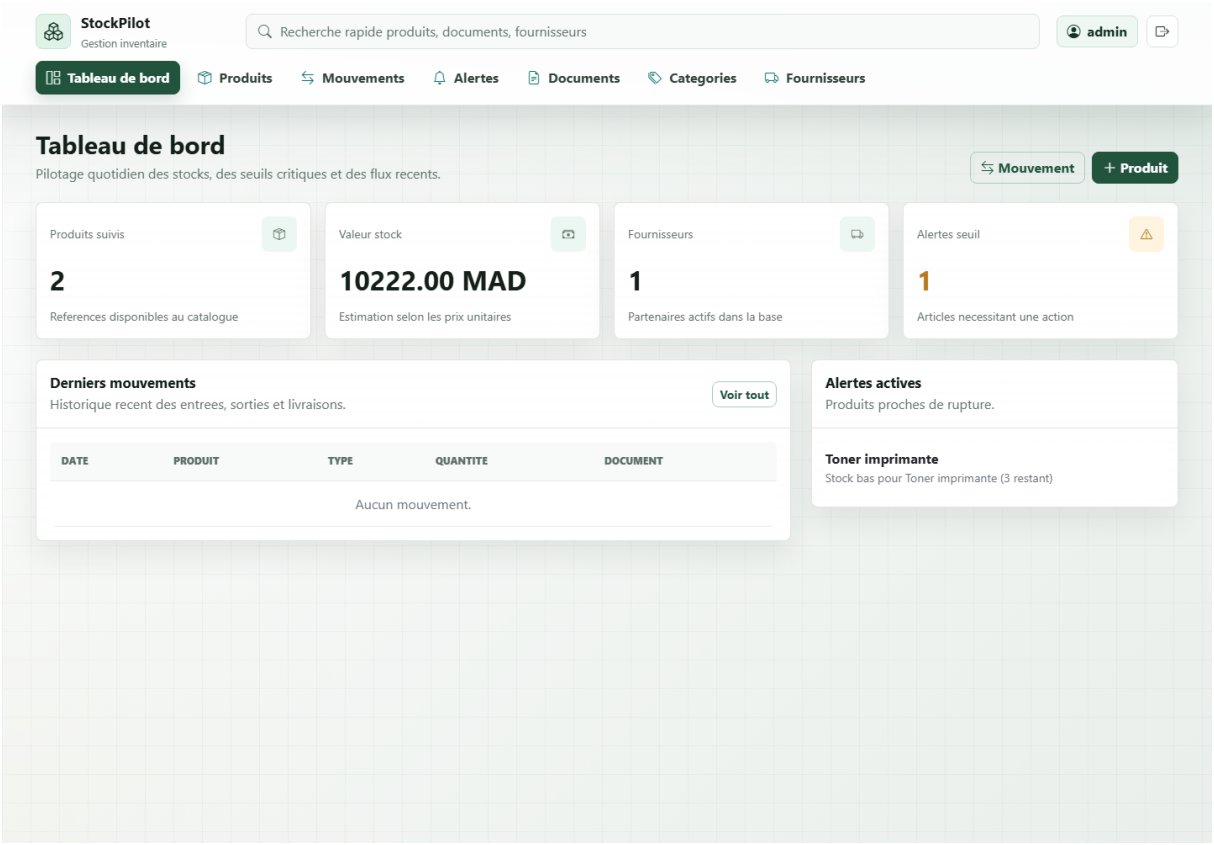


FIGURE 7.2 – Tableau de bord avec indicateurs principaux et alertes actives

## 7.4 Catalogue des produits

Le catalogue des produits centralise tous les articles enregistrés dans l'application. Chaque produit est présenté avec ses informations principales : référence, nom, prix, quantité disponible, catégorie, fournisseur et état du stock.

L'interface permet de distinguer rapidement les produits disponibles et les produits à réapprovisionner. Les badges et les couleurs facilitent la lecture visuelle, notamment lorsqu'un produit atteint ou dépasse son seuil critique.

Cette page constitue donc le module principal pour consulter, rechercher, modifier ou supprimer les articles du stock.

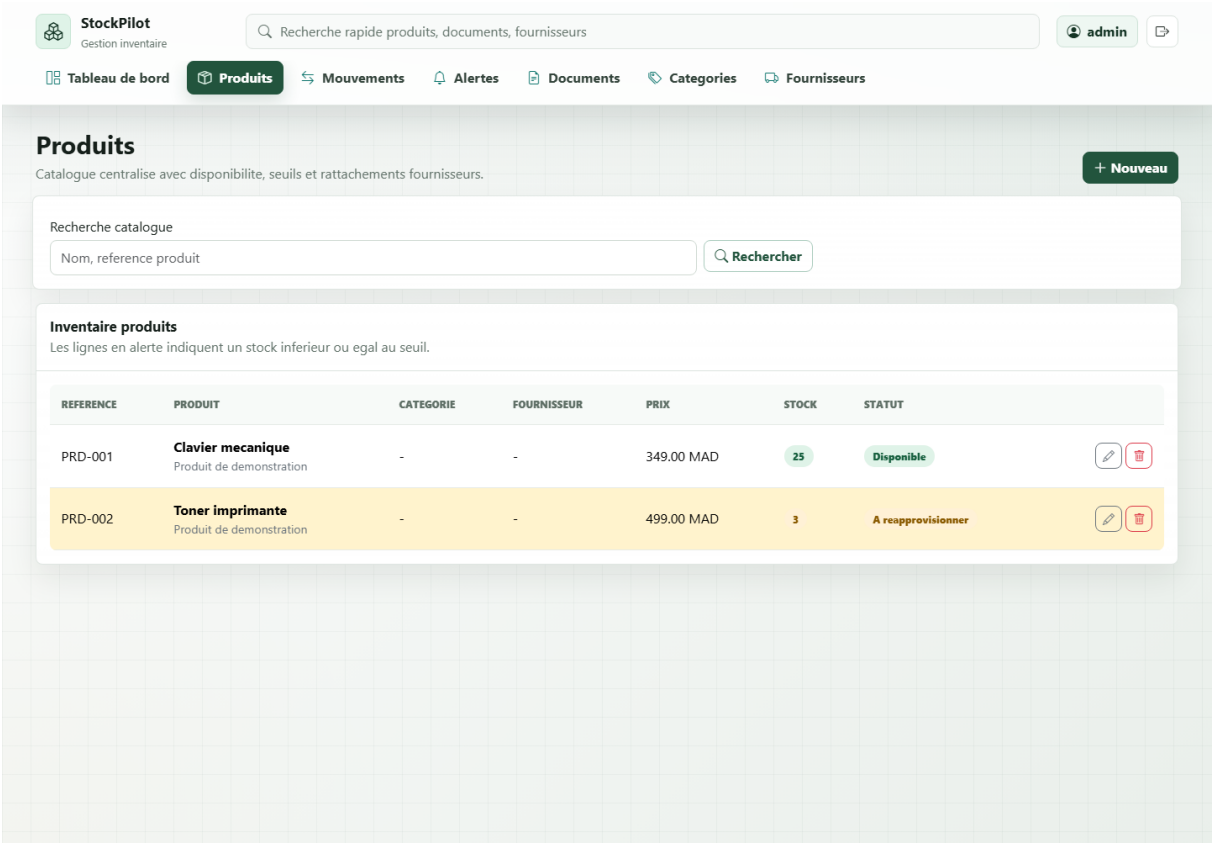


FIGURE 7.3 – Catalogue des produits avec statut de disponibilité

## 7.5 Formulaire de produit

Le formulaire produit permet d'ajouter un nouveau produit ou de modifier un produit existant. Il regroupe toutes les informations nécessaires à la gestion d'un article : référence, nom, description, prix unitaire, quantité initiale, seuil d'alerte, catégorie, fournisseur et image.

La présence du seuil d'alerte est importante, car elle permet au système de détecter automatiquement les produits en stock critique. Le formulaire contribue donc directement au bon fonctionnement du centre d'alertes.

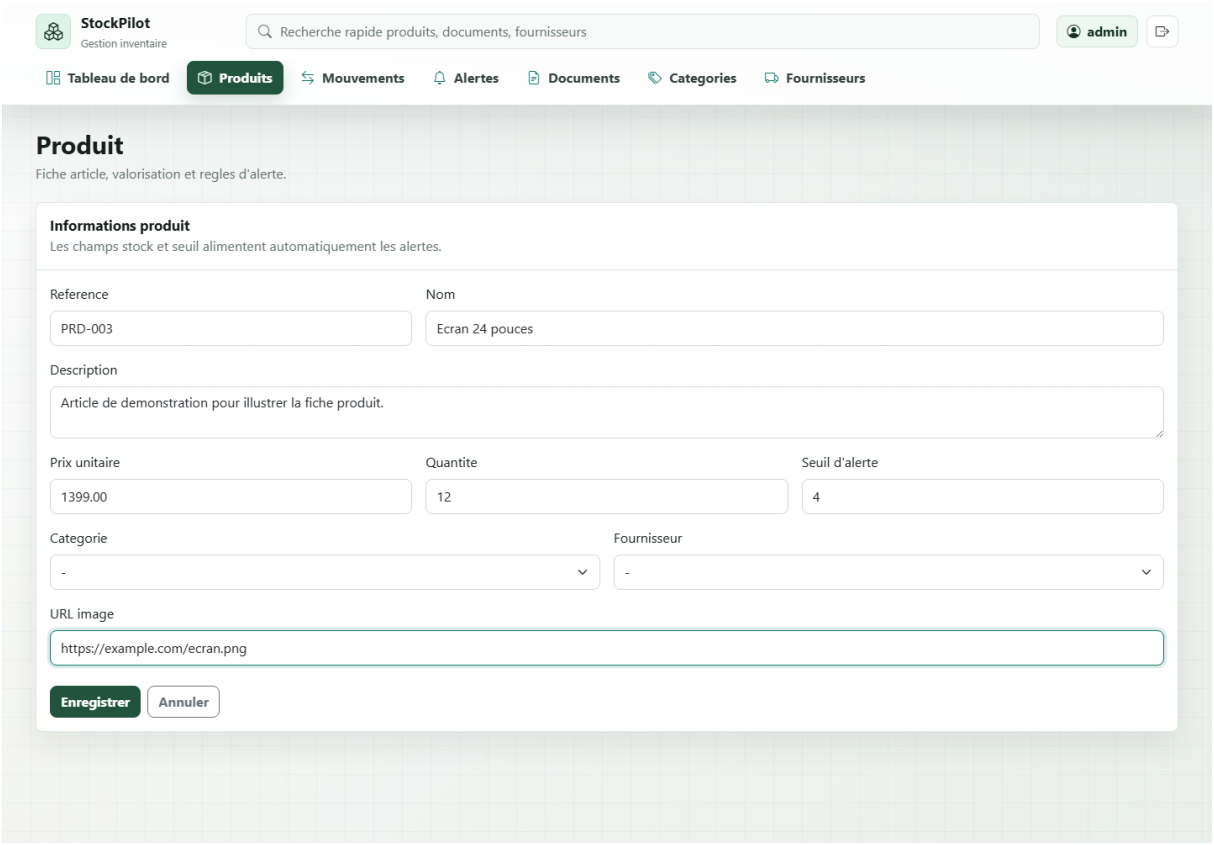


FIGURE 7.4 – Formulaire de création ou de modification d'un produit

## 7.6 Mouvements de stock

La page des mouvements permet d'enregistrer les opérations qui modifient la quantité d'un produit. Un mouvement peut correspondre à une entrée, une sortie ou une livraison. Lors de la saisie, l'utilisateur choisit le produit concerné, le type de mouvement et la quantité.

Après validation, le système met à jour automatiquement le stock du produit. Il conserve également une trace de l'opération dans le journal des mouvements. Cette traçabilité est essentielle pour comprendre l'historique des variations de stock.

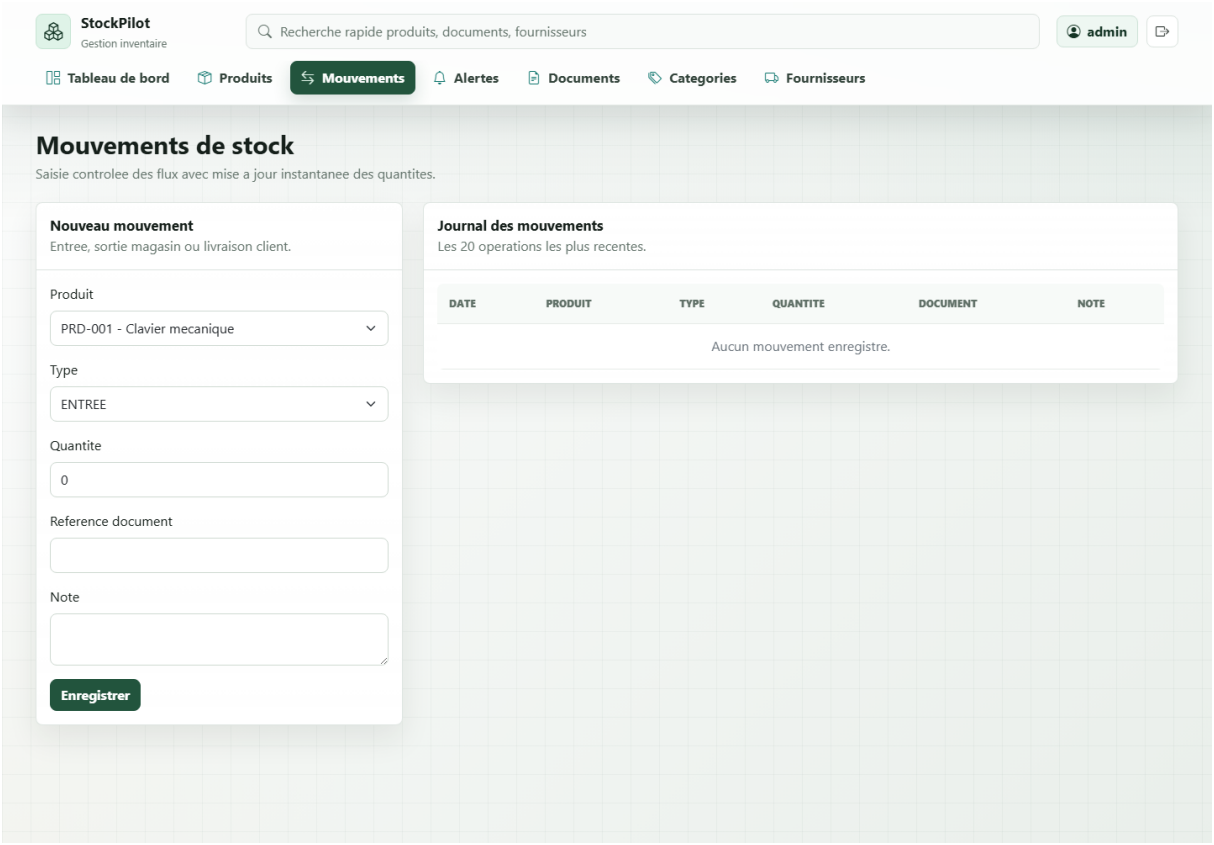


FIGURE 7.5 – Module de saisie et journal des mouvements de stock

## 7.7 Alertes de seuil

Le centre d'alertes affiche les produits dont la quantité est inférieure ou égale au seuil défini. Cette page aide l'administrateur à identifier rapidement les articles à réapprovisionner.

Chaque alerte contient le produit concerné, le message associé et son état. Une alerte peut être active ou résolue. Lorsque le stock est corrigé ou lorsque l'utilisateur traite l'alerte, son statut peut évoluer.

Ce module permet donc d'éviter les ruptures de stock et d'améliorer le suivi des produits sensibles.

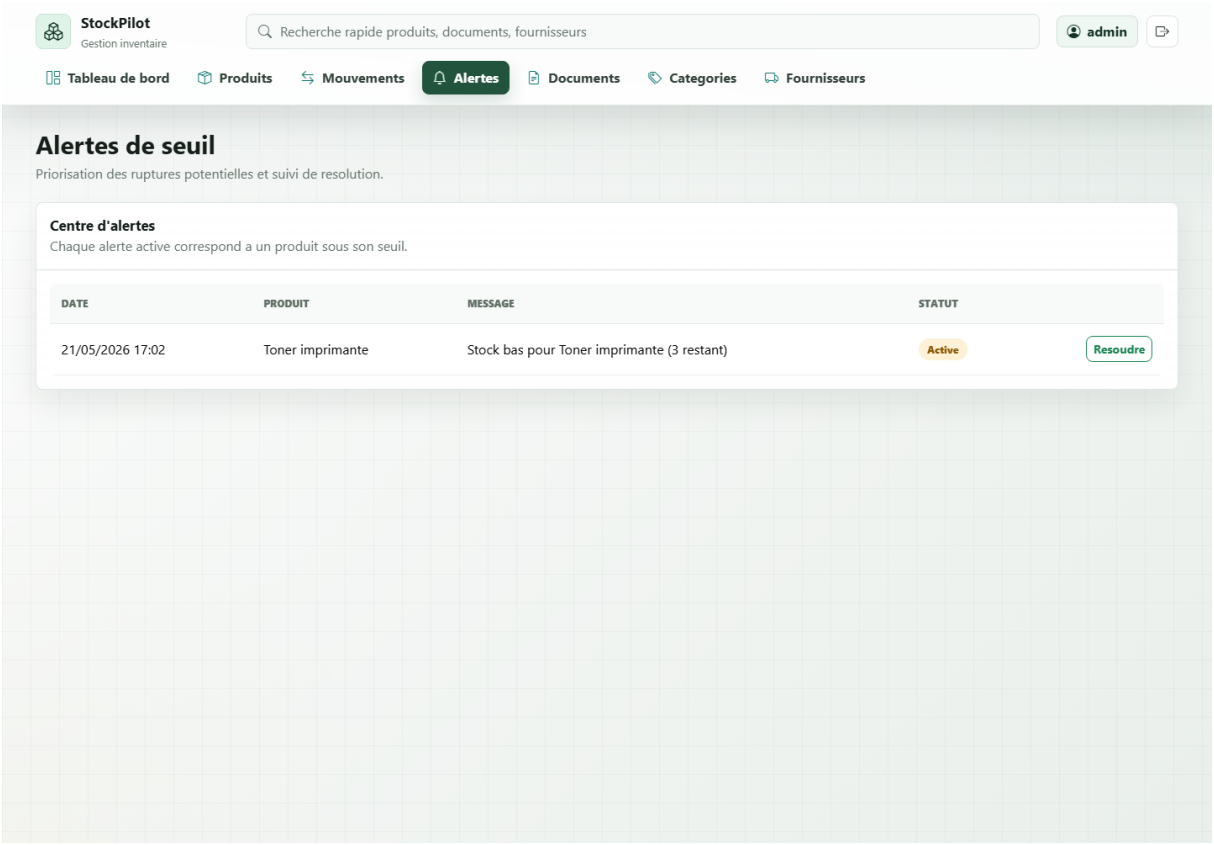


FIGURE 7.6 – Centre d'alertes des produits en seuil critique

## 7.8 Documents commerciaux

La page des documents commerciaux permet de créer et d'archiver des documents liés à la gestion du stock. L'utilisateur peut saisir les informations nécessaires, comme le type de document, le numéro, le client et la description.

L'application permet ensuite de télécharger le document au format PDF. Cette fonctionnalité facilite la conservation des pièces commerciales et donne un aspect plus professionnel au système.

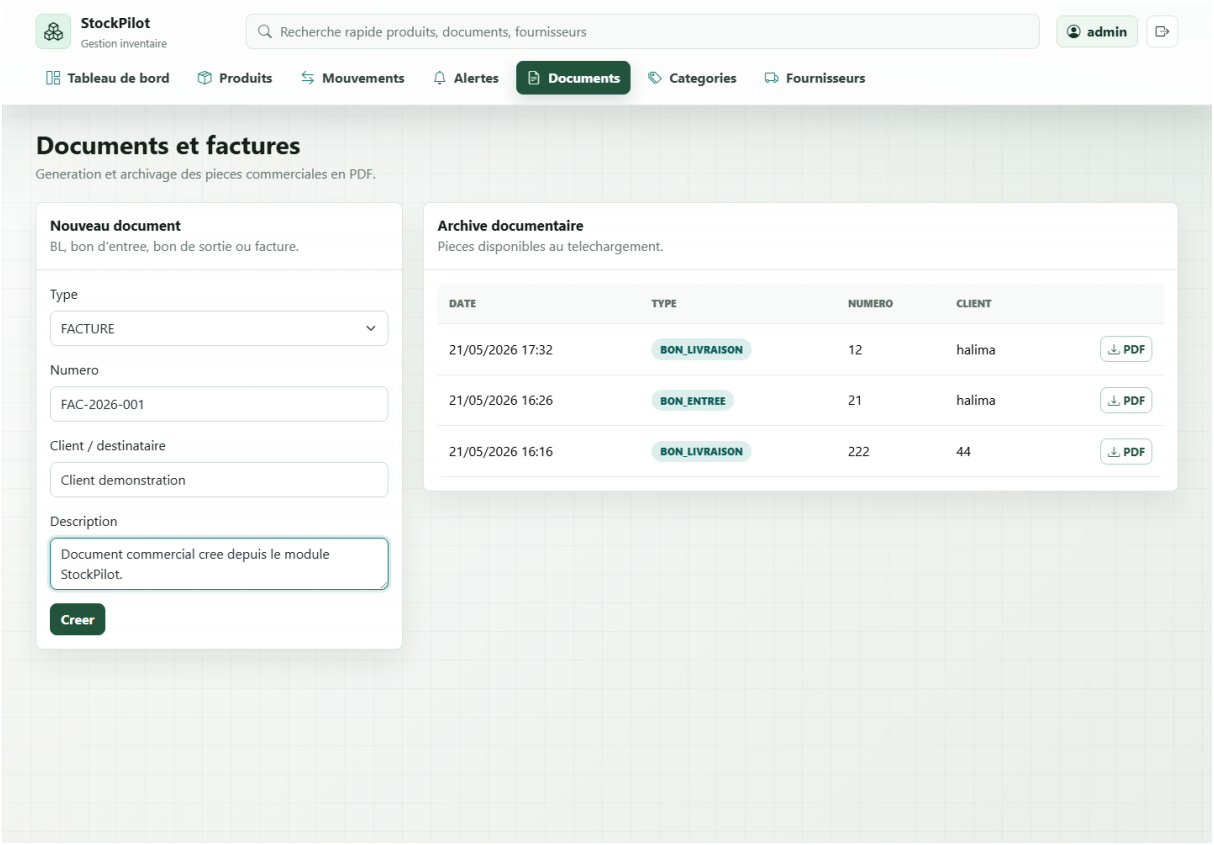


FIGURE 7.7 – Module de documents commerciaux avec génération PDF

## 7.9 Gestion des catégories

La page des catégories permet d'organiser les produits par familles. Chaque catégorie contient un nom et une description. Cette organisation rend le catalogue plus clair et facilite la recherche des articles.

L'utilisation des catégories est particulièrement utile lorsque le nombre de produits devient important. Elle permet de structurer le stock et d'améliorer la lisibilité du catalogue.

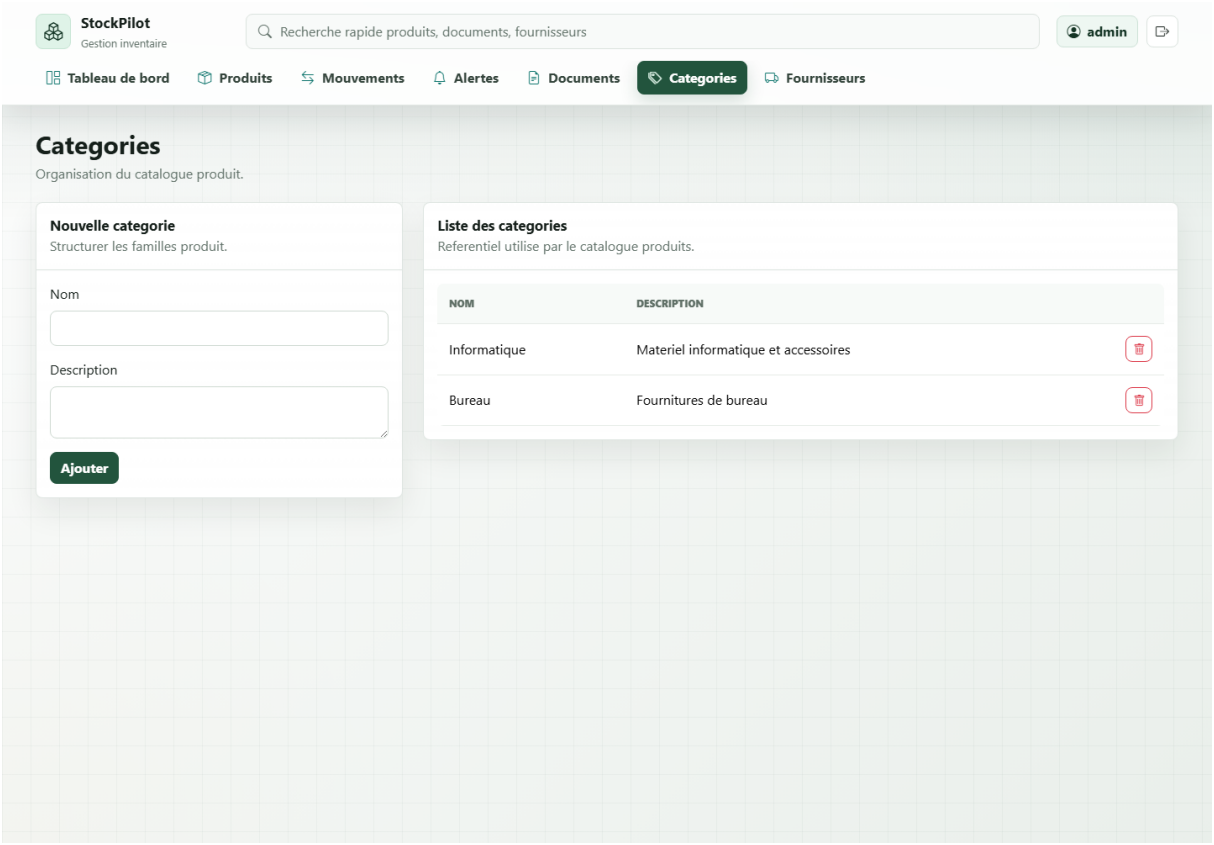


FIGURE 7.8 – Gestion des catégories de produits

## 7.10 Gestion des fournisseurs

La page des fournisseurs regroupe les informations relatives aux partenaires commerciaux. Elle permet d'enregistrer le nom du fournisseur, son adresse, son email et son numéro de téléphone.

Associer un produit à un fournisseur facilite le suivi des achats et le réapprovisionnement. En cas d'alerte de seuil, l'administrateur peut rapidement identifier le fournisseur concerné.

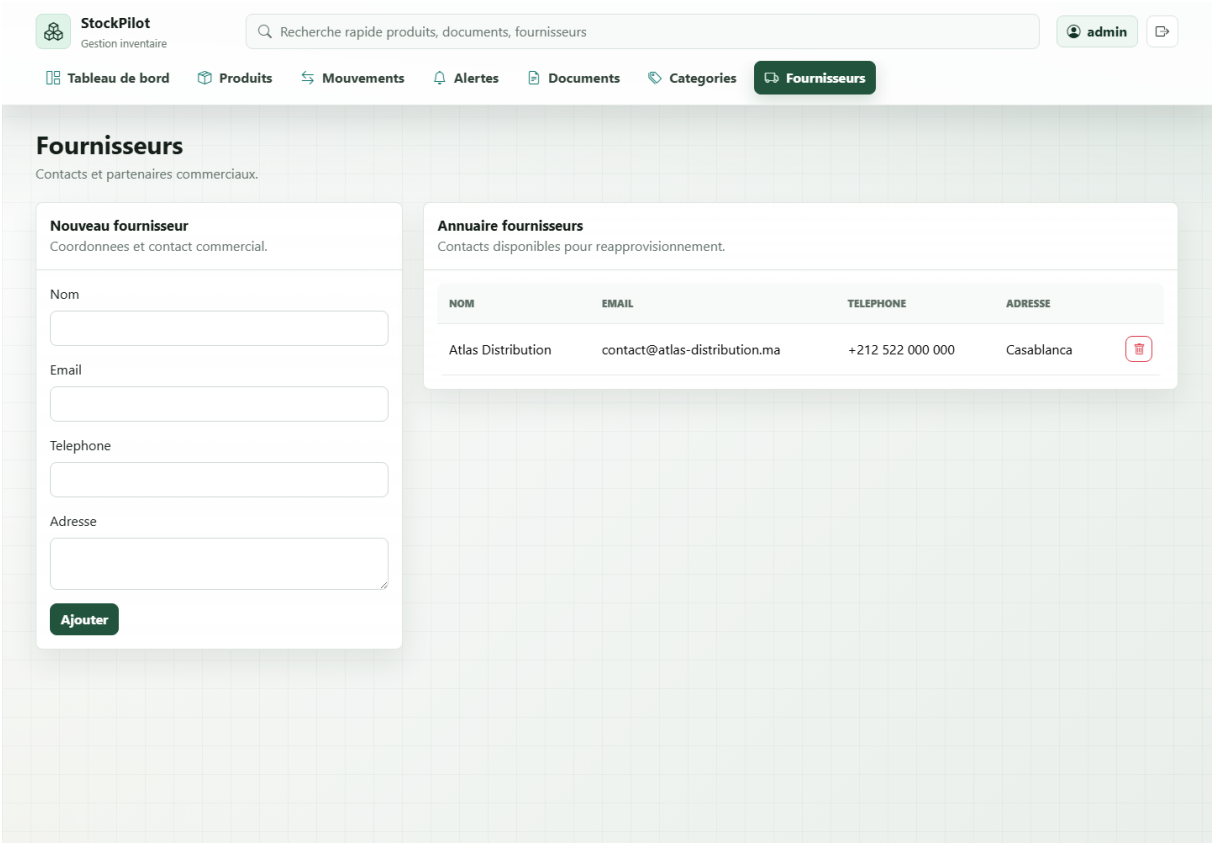


FIGURE 7.9 – Gestion des fournisseurs et informations de contact

## 7.11 Synthèse du parcours utilisateur

L'interface de **StockPilot** suit un parcours simple et logique. L'utilisateur commence par se connecter, puis accède au tableau de bord. À partir de cette page, il peut consulter les produits, enregistrer des mouvements, suivre les alertes, gérer les fournisseurs, organiser les catégories et générer des documents commerciaux.

Les captures présentées montrent que l'application couvre les besoins essentiels d'une gestion de stock :

- accès sécurisé à l'application ;
- consultation rapide des indicateurs ;
- gestion complète du catalogue produit ;
- suivi des entrées et sorties de stock ;
- détection des seuils critiques ;
- génération de documents PDF ;
- organisation des produits par catégories ;
- conservation des informations fournisseurs.

Ainsi, l'application propose une interface cohérente, structurée et adaptée aux opérations quotidiennes d'un responsable de stock.

## 7.12 Conclusion

Ce chapitre a présenté les principales interfaces de l'application **StockPilot** et le parcours suivi par l'utilisateur. Les captures montrent les fonctionnalités essentielles : connexion, tableau de bord, gestion des produits, mouvements de stock, alertes, documents, catégories et fournisseurs.

Ainsi, **StockPilot** offre une interface claire, organisée et adaptée à la gestion quotidienne du stock. Elle facilite le suivi des produits, la prise de décision et l'amélioration des opérations de gestion.

# Chapitre 8

# Déploiement et exploitation

## 8.1 Introduction

Ce chapitre présente les conditions de lancement, d'exploitation et d'évolution de l'application **StockPilot**. Après la phase de développement, il est important de montrer que l'application peut être exécutée dans un environnement local, qu'elle utilise une base de données fonctionnelle et qu'elle peut évoluer vers une configuration plus proche d'un déploiement professionnel.

L'objectif de cette partie est donc de décrire le lancement de l'application, la configuration de la base H2, l'accès à la console d'administration, l'organisation des tables générées par JPA, la possibilité de migration vers MySQL ainsi que les mesures de sécurité à prévoir dans un contexte réel.

## 8.2 Lancement local de l'application

### 8.2.1 Principe général

Le lancement local de **StockPilot** s'effectue à l'aide de Maven Wrapper. Cet outil permet d'exécuter le projet sans obliger l'utilisateur à installer manuellement une version précise de Maven sur sa machine. Le fichier `mvnw.cmd` est fourni avec le projet et permet de lancer les commandes Maven directement sous Windows.

Ce choix facilite la portabilité du projet, car chaque membre de l'équipe peut démarrer l'application avec la même commande. Il limite également les problèmes liés aux différences de versions entre les environnements de développement.

### 8.2.2 Commande de démarrage

Le lancement local s'effectue avec la commande suivante :

Listing 8.1 – Lancement de l'application avec Maven Wrapper

```
.\mvnw.cmd spring-boot:run
```

Cette commande réalise plusieurs opérations :

- chargement des dépendances Maven déclarées dans le fichier `pom.xml` ;
- compilation du code source Java ;
- initialisation du contexte Spring Boot ;
- démarrage du serveur embarqué Tomcat ;
- chargement de la configuration applicative ;
- connexion à la base de données H2 ;
- exécution éventuelle de l'initialiseur de données ;
- mise à disposition de l'application sur le port configuré.

### 8.2.3 Adresse d'accès

Une fois le serveur démarré, l'application devient accessible depuis un navigateur web à l'adresse suivante :

Listing 8.2 – Adresse locale de l'application

```
http://localhost:8080
```

L'adresse `localhost` indique que l'application s'exécute sur la machine locale. Le port `8080` correspond au port par défaut utilisé par Spring Boot lorsque le serveur embarqué Tomcat est lancé.

### 8.2.4 Authentification

L'accès à l'application est protégé par Spring Security. L'utilisateur doit passer par la page de connexion avant de pouvoir accéder au tableau de bord, aux produits, aux mouvements, aux alertes ou aux documents.

Dans la version actuelle, un compte administrateur de démonstration est configuré :

Table 8.1 – Compte de démonstration

| Élément | Valeur |
|---|---|
| Nom d'utilisateur | `admin` |
| Mot de passe | `admin123` |
| Rôle | Administrateur |

Ce compte permet de tester l'ensemble des fonctionnalités de l'application dans un cadre académique. Dans un environnement réel, il devra être remplacé par une gestion plus sécurisée des utilisateurs.

## 8.3 Base de données H2

### 8.3.1 Choix de H2

Pour la version locale du projet, la base de données utilisée est H2. Il s'agit d'un système de gestion de base de données léger, intégré et adapté aux phases de développement, de test et de démonstration.

Le choix de H2 présente plusieurs avantages :

- installation simple ;
- démarrage rapide ;
- compatibilité avec Spring Data JPA ;
- possibilité de stocker les données dans un fichier local ;
- présence d'une console web d'administration ;
- facilité de réinitialisation pendant le développement.

Dans **StockPilot**, H2 permet de tester rapidement les entités, les repositories, les services métier et les contrôleurs sans mettre en place immédiatement un serveur MySQL externe.

### 8.3.2 Stockage local de la base

La base H2 locale est stockée dans le dossier `data`. Cela signifie que les données peuvent être conservées entre deux lancements de l'application, contrairement à une base en mémoire qui serait réinitialisée à chaque arrêt.

Le stockage local permet de conserver :

- les produits enregistrés ;
- les catégories ;
- les fournisseurs ;
- les mouvements de stock ;
- les alertes générées ;
- les documents commerciaux.

### 8.3.3 Console H2

La console H2 est disponible à l'adresse suivante :

Listing 8.3 – Adresse de la console H2

```
http://localhost:8080/h2-console
```

Elle permet de visualiser directement la base de données utilisée par l'application. À travers cette console, il est possible de consulter les tables, d'exécuter des requêtes SQL et de vérifier les données créées par l'application.

### 8.3.4 Tables principales de la base

Les tables sont générées automatiquement à partir des entités JPA de l'application. Chaque entité importante du modèle correspond à une table relationnelle.

Les principales tables sont les suivantes :

Table 8.2 – Tables principales de la base de données

| Table | Rôle |
|---|---|
| `PRODUCT` | Stocke les produits du catalogue |
| `CATEGORY` | Stocke les catégories de produits |
| `SUPPLIER` | Stocke les fournisseurs |
| `STOCK_MOVEMENT` | Stocke l'historique des mouvements |
| `STOCK_ALERT` | Stocke les alertes de seuil |
| `COMMERCIAL_DOCUMENT` | Stocke les documents commerciaux |

### 8.3.5 Table des produits

La table `PRODUCT` représente le coeur du système. Elle contient les informations nécessaires au suivi du stock : référence, nom, description, prix, quantité disponible, seuil d'alerte, catégorie et fournisseur associés.

Cette table permet de vérifier que les produits affichés dans l'interface web sont bien persistés dans la base de données.

### 8.3.6 Table des mouvements

La table `STOCK_MOVEMENT` conserve l'historique des entrées, sorties et livraisons. Elle joue un rôle essentiel dans la traçabilité de l'application, car elle permet de comprendre comment le stock a évolué dans le temps.

Chaque mouvement est lié à un produit et contient notamment :

- le type de mouvement ;
- la quantité concernée ;
- la date du mouvement ;
- le produit associé.

### 8.3.7 Table des alertes

La table `STOCK_ALERT` contient les alertes générées lorsque la quantité disponible d'un produit devient inférieure ou égale au seuil défini. Elle permet de vérifier que la logique métier de détection de stock bas fonctionne correctement.

Une alerte peut être active ou résolue selon l'état du produit. Cette information permet à l'utilisateur de distinguer les problèmes encore ouverts des alertes déjà traitées.

### 8.3.8 Table des fournisseurs et catégories

Les tables `SUPPLIER` et `CATEGORY` complètent les informations relatives aux produits. Elles permettent d'organiser le catalogue et de relier chaque produit à un contexte métier plus précis.

Les catégories facilitent le classement des articles, tandis que les fournisseurs permettent de conserver les informations utiles pour le réapprovisionnement.

### 8.3.9 Table des documents commerciaux

La table `COMMERCIAL_DOCUMENT` stocke les documents créés depuis le module documentaire. Elle permet de conserver les informations relatives aux factures, bons ou documents générés par l'application.

La génération PDF s'appuie ensuite sur ces données pour produire un fichier téléchargeable.

## 8.4 Migration vers MySQL

### 8.4.1 Intérêt de la migration

Même si H2 est pratique pour le développement, une base MySQL est plus adaptée à un environnement réel. MySQL permet une meilleure gestion des volumes de données, une administration plus complète, une intégration avec des outils externes et une exploitation plus proche d'un contexte professionnel.

La migration vers MySQL permettrait notamment :

- de centraliser la base sur un serveur dédié ;
- de faciliter les sauvegardes ;
- de gérer plusieurs utilisateurs ;
- d'améliorer la robustesse en production ;
- de connecter l'application à d'autres services.

### 8.4.2 Étapes de migration

Le fichier `application.properties` contient une configuration MySQL prête à être activée. Pour migrer vers MySQL, il faut :

1. créer une base de données nommée `gestion_stock` ;
2. vérifier que le serveur MySQL est démarré ;
3. adapter l'URL de connexion JDBC ;
4. renseigner l'utilisateur MySQL ;
5. renseigner le mot de passe ;
6. vérifier la présence du driver MySQL dans `pom.xml` ;
7. relancer l'application ;
8. contrôler la création des tables.

### 8.4.3 Exemple de configuration MySQL

Listing 8.4 – Exemple de configuration MySQL

```
spring.datasource.url=jdbc:mysql://localhost:3306/gestion_stock
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

La propriété `spring.jpa.hibernate.ddl-auto=update` permet à Hibernate de mettre à jour automatiquement le schéma de la base à partir des entités JPA. Cette option est pratique en développement, mais elle doit être utilisée avec prudence en production.

## 8.5 Exploitation de l'application

### 8.5.1 Suivi du fonctionnement

L'exploitation consiste à maintenir l'application disponible, stable et sécurisée. Dans le cadre local, cela revient principalement à vérifier que le serveur démarre correctement, que la base est accessible et que les fonctionnalités principales répondent comme prévu.

Les points à surveiller sont :

– le démarrage sans erreur du serveur Spring Boot ;

- l'accès à la page de connexion ;
- la connexion avec le compte administrateur ;
- l'affichage du tableau de bord ;
- la présence des produits dans la base ;
- le bon enregistrement des mouvements ;
- la génération des alertes ;
- la création des documents PDF.

### 8.5.2 Sauvegarde des données

En exploitation, les données représentent l'élément le plus important du système. Une perte de données pourrait rendre impossible le suivi du stock ou l'analyse des mouvements passés.

Il est donc recommandé de prévoir :

- une sauvegarde régulière de la base ;
- une exportation périodique des tables importantes ;
- une conservation des documents PDF générés ;
- une procédure de restauration en cas d'erreur.

## 8.6 Sécurité d'exploitation

### 8.6.1 Limites de la configuration actuelle

La configuration actuelle convient à une démonstration académique, mais elle ne doit pas être utilisée telle quelle en production. Le compte `admin/admin123` est simple à retenir, mais il représente un risque si l'application est exposée sur un réseau réel.

De plus, la console H2 est utile en développement, mais elle peut devenir dangereuse si elle reste accessible à des utilisateurs non autorisés.

### 8.6.2 Mesures recommandées

Pour un déploiement réel, plusieurs mesures sont recommandées :

- remplacer le mot de passe par défaut ;
- stocker les identifiants dans des variables d'environnement ;
- activer HTTPS ;
- créer plusieurs profils utilisateurs ;
- séparer les rôles administrateur, magasinier et lecteur ;

- désactiver ou restreindre la console H2 ;
- sauvegarder régulièrement la base ;
- journaliser les opérations sensibles ;
- limiter l'accès aux routes critiques ;
- utiliser une base MySQL protégée par un utilisateur dédié ;
- éviter d'exposer les informations de configuration dans le code source.

### 8.6.3 Sécurisation des mots de passe

Dans une version évoluée, les mots de passe ne doivent pas être écrits directement dans le fichier de configuration. Ils doivent être injectés à travers des variables d'environnement.

Exemple :

Listing 8.5 – Utilisation de variables d'environnement

```
app.security.username=${APP_ADMIN_USERNAME}
app.security.password=${APP_ADMIN_PASSWORD}
```

Cette approche évite de stocker les identifiants sensibles dans le dépôt du projet.

### 8.6.4 Protection de la base de données

La base de données doit également être protégée. Dans le cas d'une migration vers MySQL, il est préférable de créer un utilisateur spécifique à l'application plutôt que d'utiliser le compte `root`.

Cet utilisateur doit disposer uniquement des permissions nécessaires :

- lecture ;
- insertion ;
- modification ;
- suppression contrôlée.

## 8.7 Conclusion

Le déploiement local de **StockPilot** montre que l'application peut être lancée facilement grâce à Maven Wrapper et Spring Boot. L'utilisation de H2 facilite la démonstration et permet de vérifier rapidement les données générées par les modules de l'application.

La console H2 joue un rôle important dans la validation technique, car elle permet d'observer directement les tables, les produits, les mouvements, les alertes et les documents commerciaux. Ces captures renforcent le rapport en montrant que l'application ne se limite pas à une interface graphique, mais repose bien sur une base de données structurée.

Enfin, la possibilité de migrer vers MySQL montre que le projet peut évoluer vers un environnement plus professionnel. Pour une mise en production réelle, il sera toutefois nécessaire de renforcer la sécurité, de protéger les identifiants, de désactiver les outils de développement exposés et de mettre en place une stratégie de sauvegarde fiable.

Chapitre 9

# Tests illustrés et validation fonctionnelle détaillée

## 9.1 Introduction

La validation d'une application de gestion ne doit pas se limiter à affirmer que le projet démarre. Elle doit démontrer que les parcours essentiels fonctionnent, que les erreurs sont traitées correctement et que les règles métier produisent les résultats attendus. Ce chapitre complète donc la partie de validation par des captures d'écran réalisées sur l'application exécutée localement.

Les captures présentées dans cette section servent de preuves visuelles. Elles montrent les états de l'interface pendant des scénarios précis :

- erreur d'authentification ;
- recherche dans le catalogue ;
- tentative de sortie de stock impossible ;
- mouvement d'entrée accepté ;
- préparation d'un document PDF ;
- contrôle du centre d'alertes.

## 9.2 Environnement utilisé pour les tests illustrés

Les tests ont été réalisés sur l'application Spring Boot lancée localement. Les paramètres d'environnement sont les suivants :

Table 9.1 – Environnement de validation illustrée

| Élément | Valeur |
|---|---|
| Application | **StockPilot** / `gestion-stock` |
| Mode d'exécution | Spring Boot local |
| Adresse | `http://localhost:8080` |
| Base de données | H2 fichier local `./data/gestion-stock` |
| Compte utilisé | `admin / admin123` |
| Navigateur de capture | Google Chrome automatisé |
| Date de validation | 05/06/2026 |
| Objectif | Vérifier les parcours utilisateur, les erreurs et les règles métier. |

## 9.3 Jeu de données observé

Le jeu de données de démonstration contient deux produits principaux. Le produit `Clavier mecanique` est disponible, tandis que le produit `Toner imprimante` est sous son seuil d'alerte. Cette situation permet de vérifier à la fois l'affichage d'un stock normal et l'affichage d'un stock critique.

Table 9.2 – Jeu de données utilisé pendant les captures

| Référence | Produit | Prix | Quantité initiale | Statut attendu |
|---|---|---|---|---|
| PRD-001 | Clavier mecanique | 349.00 MAD | 25 | Disponible |
| PRD-002 | Toner imprimante | 499.00 MAD | 3 | À réapprovisionner |

## 9.4 Scénario T01 : erreur d'authentification

### 9.4.1 But du test

Ce test vérifie que l'application refuse les identifiants incorrects et affiche un message d'erreur compréhensible à l'utilisateur. Il s'agit d'une validation importante, car l'accès aux données de stock doit rester protégé.

### 9.4.2 Procédure

1. ouvrir la page `/login` ;
2. saisir l'utilisateur `admin` ;
3. saisir un mot de passe incorrect ;

4. soumettre le formulaire ;
5. vérifier l'affichage du message d'erreur.

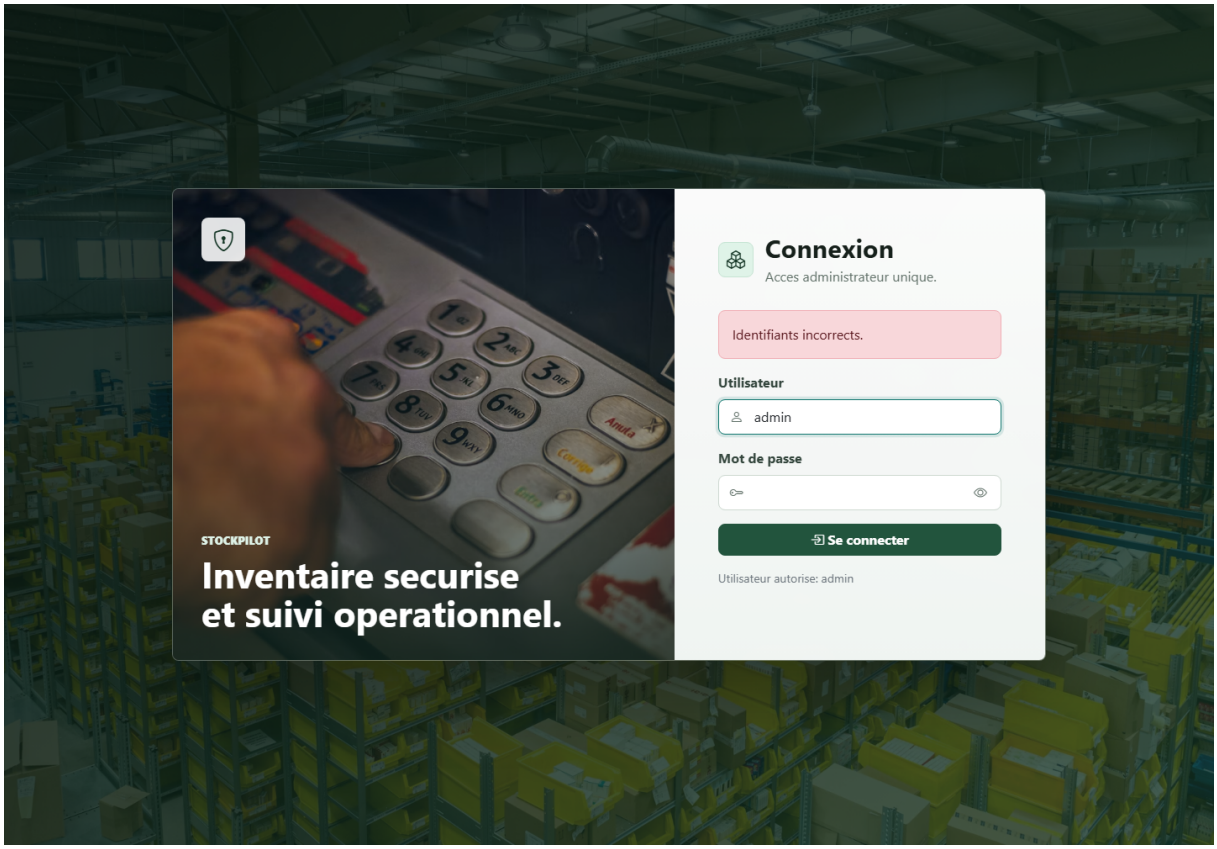


FIGURE 9.1 – Test T01 : refus d'une authentification incorrecte

### 9.4.3 Résultat

Le message *Identifiants incorrects* apparaît dans une alerte visuelle. Le formulaire reste affiché et l'utilisateur n'est pas redirigé vers les pages internes. Le résultat est conforme à l'objectif de sécurité.

TABLE 9.3 – Résultat du test T01

| Critère | Attendu | Obtenu |
|---|---|---|
| Refus d'accès | L'utilisateur reste sur la page de connexion | Conforme |
| Message d'erreur | Message visible et clair | Conforme |
| Protection des routes | Aucun accès au tableau de bord | Conforme |

## 9.5 Scénario T02 : recherche d'un produit

### 9.5.1 But du test

Ce test vérifie que le catalogue produit peut être filtré par mot-clé. La recherche est importante pour une application de stock, car le nombre de références peut augmenter rapidement.

### 9.5.2 Procédure

1. se connecter avec le compte administrateur ;
2. ouvrir la page `/products` ;
3. saisir le mot-clé `toner` dans le champ de recherche ;
4. cliquer sur le bouton *Rechercher* ;
5. vérifier que le résultat correspond au produit recherché.

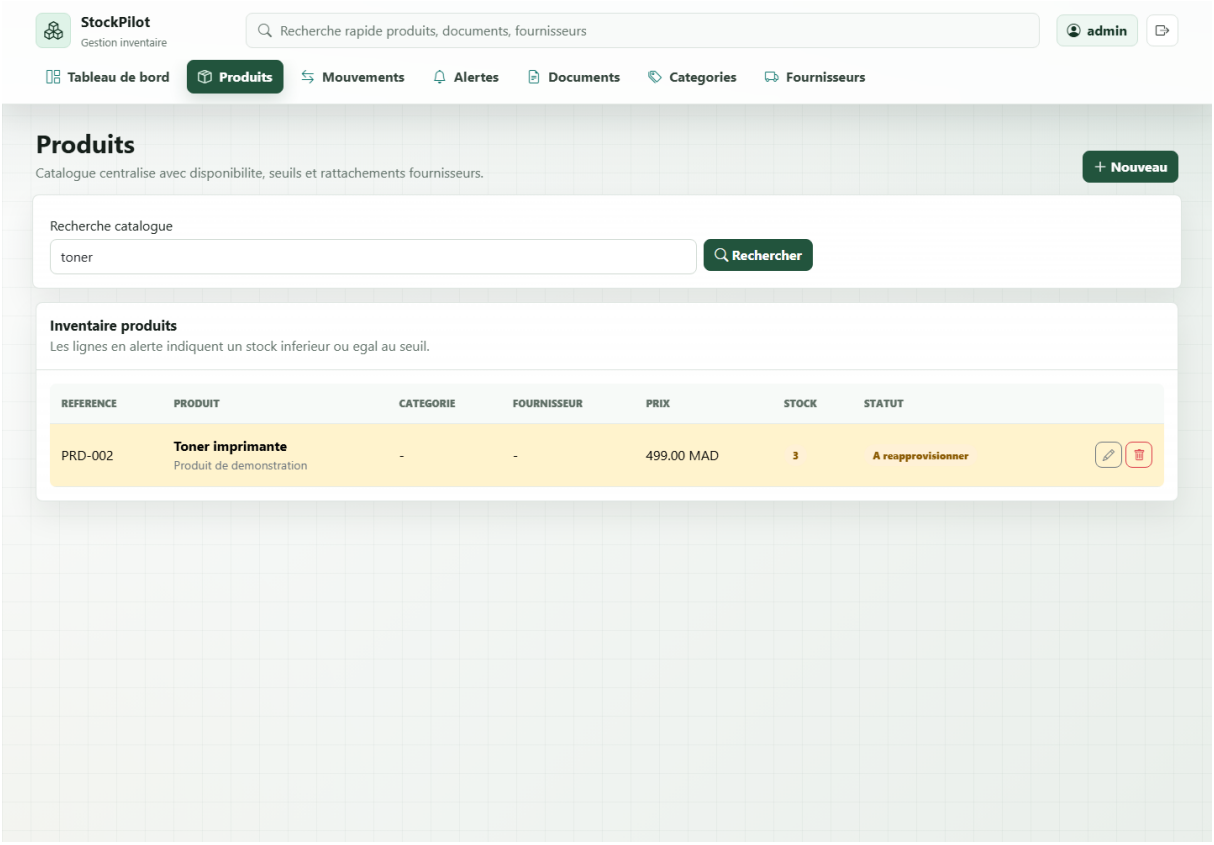


FIGURE 9.2 – Test T02 : recherche du produit Toner

### 9.5.3 Analyse

La liste filtrée affiche le produit `Toner imprimante`. Le statut reste visible et indique que le produit est à réapprovisionner. Le test valide donc deux points : le filtrage fonctionne et les indicateurs de stock restent affichés même après recherche.

TABLE 9.4 – Résultat du test T02

| Critère | Attendu | Obtenu |
|---|---|---|
| Filtrage | Affichage du produit contenant `toner` | Conforme |
| Statut de stock | Badge de réapprovisionnement visible | Conforme |
| Maintien interface | Navigation et formulaire de recherche conservés | Conforme |

## 9.6 Scénario T03 : refus d'un mouvement impossible

### 9.6.1 But du test

Ce test vérifie la règle métier la plus critique : une sortie de stock ne doit pas rendre la quantité négative. Sans cette règle, le système pourrait afficher des stocks incohérents et perdre sa fiabilité.

### 9.6.2 Procédure

1. ouvrir la page `/movements` ;
2. sélectionner un produit disponible ;
3. choisir le type `SORTIE` ;
4. saisir une quantité très supérieure au stock disponible ;
5. soumettre le formulaire ;
6. vérifier que le mouvement est refusé.

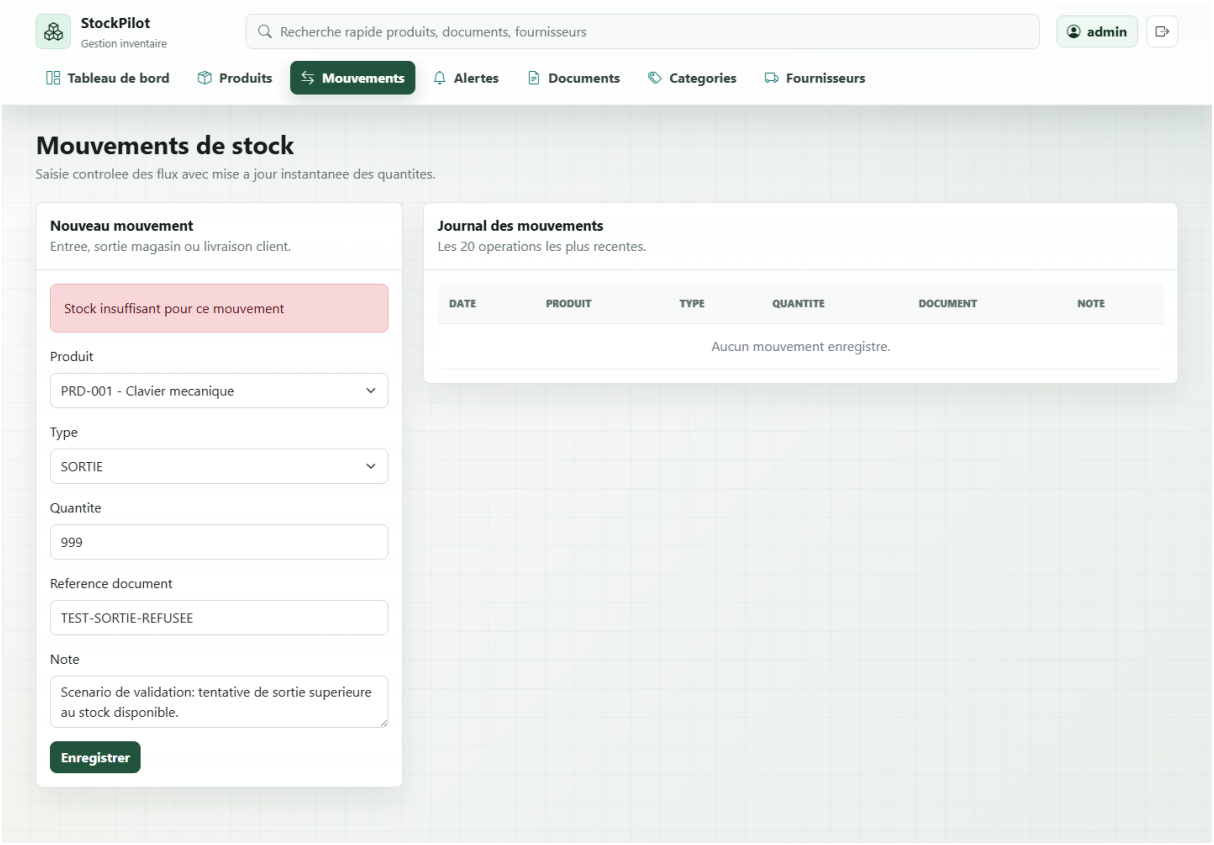


FIGURE 9.3 – Test T03 : refus d'une sortie supérieure au stock disponible

### 9.6.3 Résultat

L'application affiche le message *Stock insuffisant pour ce mouvement*. Le journal des mouvements ne reçoit pas la ligne refusée. La règle métier est donc appliquée côté serveur et protège les données.

TABLE 9.5 – Résultat du test T03

| Critère | Attendu | Obtenu |
|---|---|---|
| Contrôle quantité | Refus si la nouvelle quantité est négative | Conforme |
| Message utilisateur | Erreur claire dans le formulaire | Conforme |
| Persistance | Aucun mouvement invalide sauvegardé | Conforme |

## 9.7 Scénario T04 : enregistrement d'un mouvement valide

### 9.7.1 But du test

Ce test vérifie que le système accepte un mouvement d'entrée valide, met à jour les données et affiche le mouvement dans le journal.

### 9.7.2 Procédure

1. ouvrir la page `/movements` ;
2. sélectionner le produit `Clavier mecanique` ;
3. choisir le type `ENTREE` ;
4. saisir la quantité `2` ;
5. renseigner une référence de document de test ;
6. soumettre le formulaire ;
7. vérifier que le mouvement apparaît dans le journal.

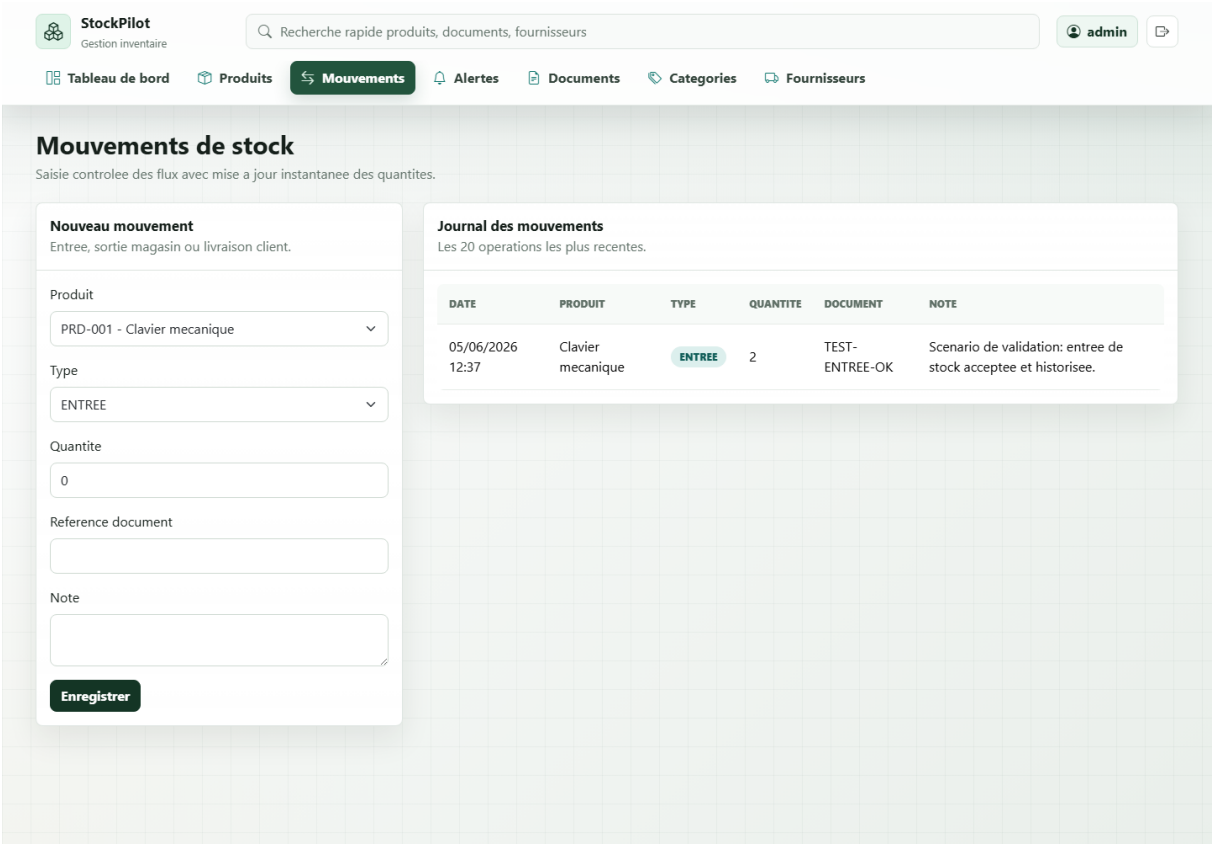


FIGURE 9.4 – Test T04 : mouvement d'entrée accepté et historisé

### 9.7.3 Analyse

Le mouvement enregistré apparaît dans le journal avec la date, le produit, le type, la quantité, la référence documentaire et la note. Ce résultat confirme que l'application ne se contente pas de modifier une quantité : elle conserve aussi la trace de l'opération.

Table 9.6 – Résultat du test T04

| Critère | Attendu | Obtenu |
|---|---|---|
| Acceptation mouvement | Entrée valide sauvegardée | Conforme |
| Journalisation | Ligne visible dans le journal | Conforme |
| Traçabilité | Référence et note conservées | Conforme |

## 9.8 Scénario T05 : préparation d'un document PDF

### 9.8.1 But du test

Ce test vérifie que le module documentaire permet de préparer une pièce commerciale. La capture montre le formulaire rempli et l'archive documentaire disponible à droite.

### 9.8.2 Procédure

1. ouvrir la page `/documents` ;
2. sélectionner le type `FACTURE` ;
3. saisir un numéro de test ;
4. saisir un client ;
5. renseigner une description ;
6. vérifier l'archive documentaire et le bouton PDF.

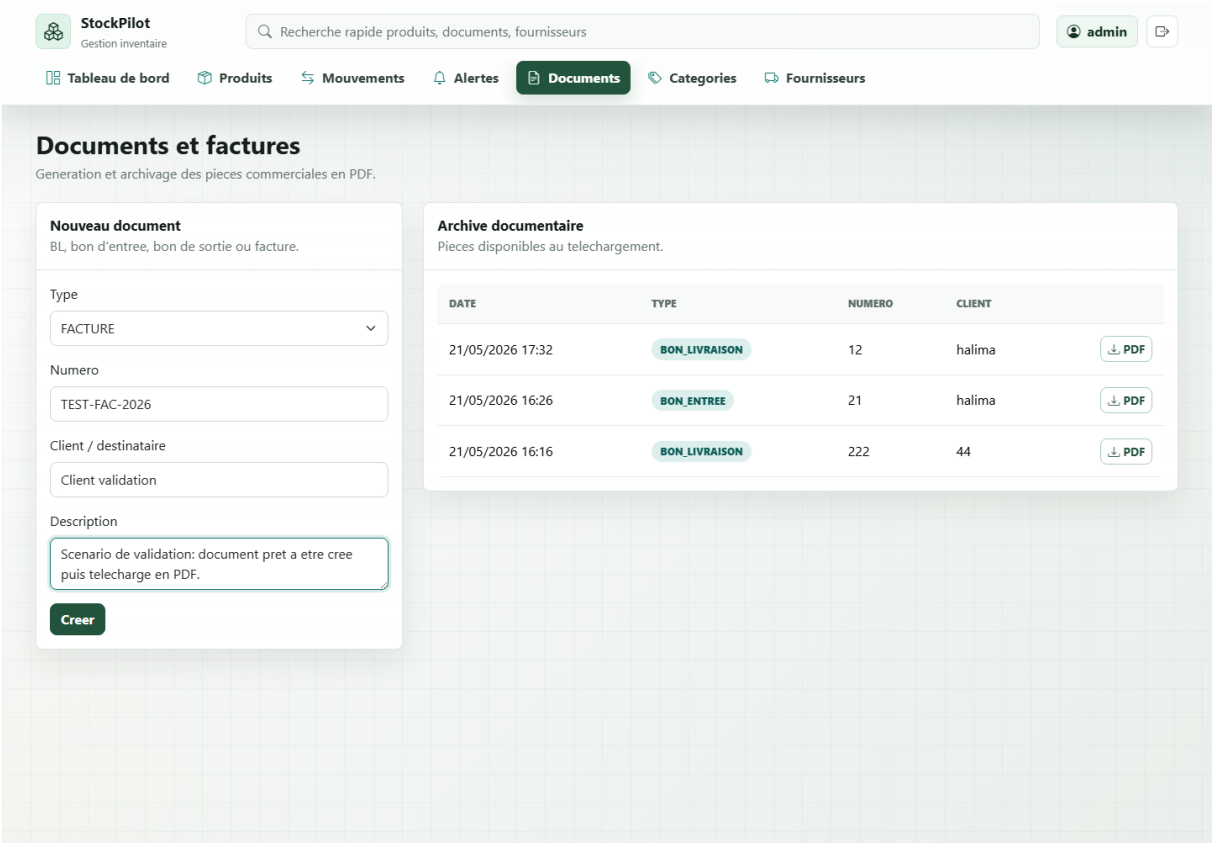


FIGURE 9.5 – Test T05 : formulaire documentaire et archive PDF

### 9.8.3 Analyse

Le module présente clairement la zone de création et l'archive. Les documents archivés possèdent un bouton *PDF*, ce qui valide l'accès utilisateur au téléchargement. La génération elle-même est également couverte par le test automatisé `PdfDocumentServiceTests`.

TABLE 9.7 – Résultat du test T05

| Critère | Attendu | Obtenu |
|---|---|---|
| Formulaire | Champs type, numéro, client et description visibles | Conforme |
| Archive | Documents existants listés | Conforme |
| Accès PDF | Bouton de téléchargement visible | Conforme |

## 9.9 Scénario T06 : contrôle du centre d'alertes

### 9.9.1 But du test

Ce test vérifie que les alertes de seuil sont visibles dans une page dédiée. Le responsable stock doit pouvoir identifier rapidement les produits nécessitant une action.

### 9.9.2 Procédure

1. ouvrir la page `/alerts` ;
2. vérifier la présence du produit critique ;
3. lire le message d'alerte ;
4. vérifier le statut actif ;
5. vérifier la présence du bouton de résolution.

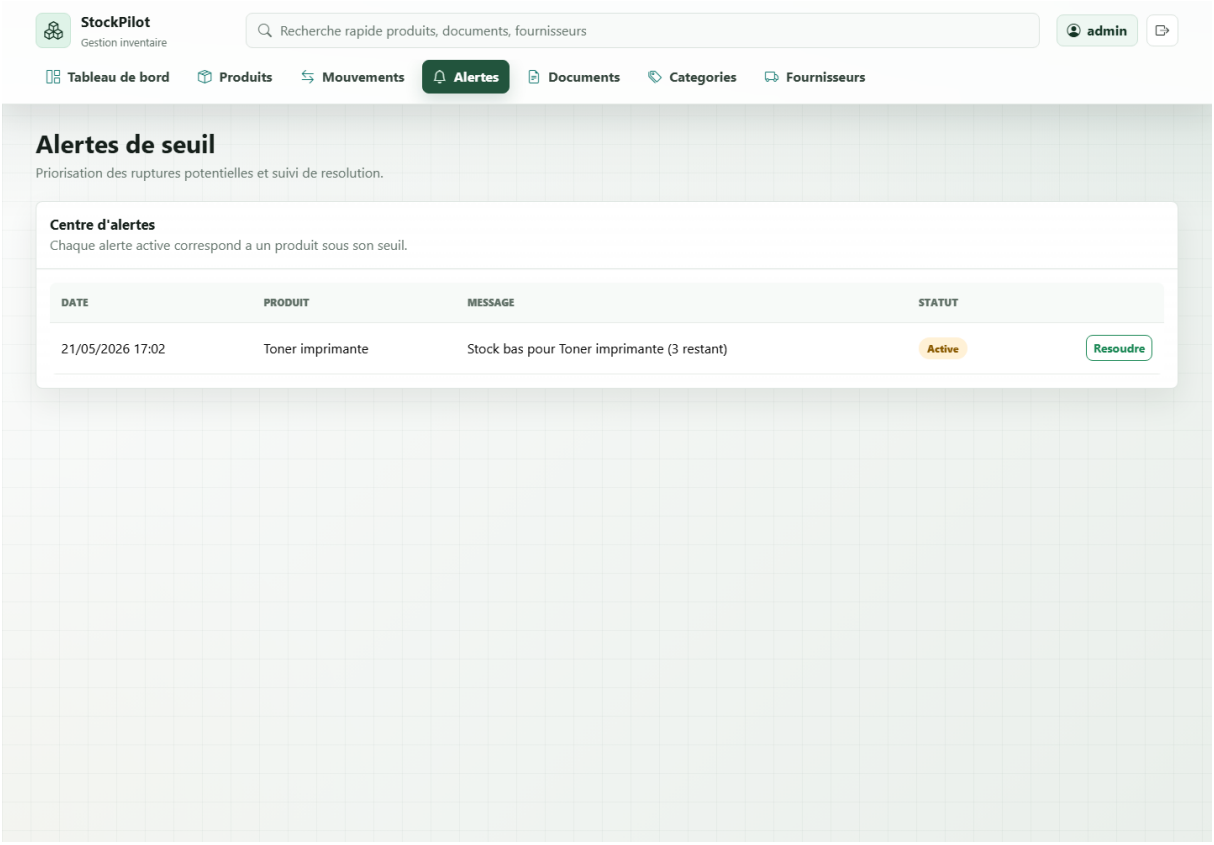


FIGURE 9.6 – Test T06 : contrôle du centre d'alertes

### 9.9.3 Analyse

L'alerte du produit `Toner imprimante` est visible avec le statut *Active*. Le message précise le produit et la quantité restante. Cette présentation répond au besoin de priorisation des ruptures potentielles.

Table 9.8 – Résultat du test T06

| Critère | Attendu | Obtenu |
|---|---|---|
| Produit critique | Toner imprimante visible | Conforme |
| Message | Quantité restante indiquée | Conforme |
| Statut | Alerte active affichée | Conforme |
| Action | Bouton Résoudre visible | Conforme |

## 9.10 Matrice globale de validation illustrée

Table 9.9: Matrice des tests illustrés

| Code | Fonction testée | Preuve visuelle | Statut |
|---|---|---|---|
| T01 | Authentification incorrecte | Figure 8.1 | Conforme |
| T02 | Recherche produit | Figure 8.2 | Conforme |
| T03 | Refus stock insuffisant | Figure 8.3 | Conforme |
| T04 | Mouvement valide | Figure 8.4 | Conforme |
| T05 | Module documentaire | Figure 8.5 | Conforme |
| T06 | Centre d'alertes | Figure 8.6 | Conforme |

## 9.11 Analyse transversale des tests

### 9.11.1 Validation de la sécurité

Le test d'authentification incorrecte confirme que Spring Security bloque l'accès lorsque le mot de passe est invalide. La présence d'un message d'erreur est utile pour l'expérience utilisateur, mais elle reste suffisamment générique pour ne pas révéler si le nom d'utilisateur ou le mot de passe est faux.

### 9.11.2 Validation de l'ergonomie

Les captures montrent une navigation persistante et cohérente. Les pages utilisent les mêmes composants visuels : en-tête, barre de navigation, panneaux, formulaires, tableaux et badges. Cette homogénéité réduit l'effort d'apprentissage.

### 9.11.3 Validation de la règle de stock

Le refus d'un mouvement impossible est une preuve directe que la règle métier est appliquée au bon endroit. Même si un utilisateur modifie le formulaire côté navigateur, le serveur recalcule la quantité et refuse les opérations incohérentes.

### 9.11.4 Validation de la traçabilité

Le mouvement accepté apparaît dans le journal avec ses informations. Cette trace est essentielle pour comprendre l'historique du stock. Dans une version future, ce journal pourrait être enrichi par le nom de l'utilisateur, l'adresse IP ou une signature d'opération.

### 9.11.5 Validation du suivi des alertes

Le centre d'alertes permet d'isoler les produits critiques du reste du catalogue. Cela évite à l'utilisateur de parcourir manuellement toutes les références pour identifier les ruptures potentielles.

## 9.12 Plan de tests recommandé pour une version future

La version actuelle contient des tests automatisés de base et des validations visuelles. Pour une version plus avancée, il serait recommandé d'ajouter les tests suivants :

Table 9.10: Tests recommandés pour extension future

| Type de test | Objectif | Exemple |
|---|---|---|
| Test unitaire service | Vérifier les règles métier isolées | Sortie impossible, entrée valide, alerte créée. |
| Test repository | Vérifier les requêtes personnalisées | Recherche par nom et référence. |
| Test contrôleur MVC | Vérifier les routes et vues | Accès `/products`, redirections, erreurs. |
| Test sécurité | Vérifier les droits d'accès | Accès interdit sans session. |
| Test PDF avancé | Vérifier contenu textuel du PDF | Numéro, client, description. |
| Test intégration base | Vérifier persistance réelle | Sauvegarde mouvement et produit. |
| Test UI automatisé | Vérifier parcours navigateur complet | Connexion, création produit, mouvement, PDF. |

| | | |
|---|---|---|
| Test responsive | Vérifier mobile/tablette | Navigation et tableaux sur petit écran. |

## 9.13 Fiches de tests détaillées

### 9.13.1 Fiche T01

Table 9.11 – Fiche détaillée T01

| Champ | Description |
|---|---|
| Nom | Authentification incorrecte |
| Précondition | Application démarrée, page `/login` accessible |
| Données | utilisateur `admin`, mot de passe invalide |
| Étapes | Saisir les identifiants, soumettre le formulaire |
| Résultat attendu | Affichage d'une erreur et refus d'accès |
| Résultat obtenu | Message d'erreur visible, pas de redirection interne |
| Statut | Conforme |

### 9.13.2 Fiche T02

Table 9.12 – Fiche détaillée T02

| Champ | Description |
|---|---|
| Nom | Recherche catalogue |
| Précondition | Utilisateur authentifié |
| Données | mot-clé `toner` |
| Étapes | Ouvrir les produits, saisir le mot-clé, rechercher |
| Résultat attendu | Affichage du produit Toner imprimante |
| Résultat obtenu | Le produit recherché est affiché avec son statut |
| Statut | Conforme |

### 9.13.3 Fiche T03

Table 9.13 – Fiche détaillée T03

| Champ | Description |
|---|---|
| Nom | Stock insuffisant |
| Précondition | Utilisateur authentifié, produit existant |
| Données | type `SORTIE`, quantité `999` |
| Étapes | Saisir le mouvement et enregistrer |
| Résultat attendu | Refus de l'opération |
| Résultat obtenu | Message *Stock insuffisant pour ce mouvement* |
| Statut | Conforme |

### 9.13.4 Fiche T04

Table 9.14 – Fiche détaillée T04

| Champ | Description |
|---|---|
| Nom | Mouvement valide |
| Précondition | Utilisateur authentifié, produit existant |
| Données | type `ENTREE`, quantité `2` |
| Étapes | Saisir le mouvement et enregistrer |
| Résultat attendu | Mouvement historisé |
| Résultat obtenu | Ligne visible dans le journal |
| Statut | Conforme |

### 9.13.5 Fiche T05

Table 9.15 – Fiche détaillée T05

| Champ | Description |
|---|---|
| Nom | Module documentaire |
| Précondition | Utilisateur authentifié |
| Données | facture de test |
| Étapes | Remplir le formulaire document |
| Résultat attendu | Formulaire exploitable et archive visible |
| Résultat obtenu | Champs visibles et boutons PDF présents |
| Statut | Conforme |

### 9.13.6 Fiche T06

Table 9.16 – Fiche détaillée T06

| Champ | Description |
|---|---|
| Nom | Centre d'alertes |
| Précondition | Produit sous seuil existant |
| Données | Toner imprimante avec stock bas |
| Étapes | Ouvrir la page `/alerts` |
| Résultat attendu | Alerte active visible |
| Résultat obtenu | Alerte affichée avec statut et message |
| Statut | Conforme |

## 9.14 Conclusion

Les tests illustrés confirment que l'application répond aux principaux besoins opérationnels. L'authentification protège l'accès, le catalogue permet de retrouver un produit, les mouvements appliquent les règles métier, les opérations valides sont historisées, les documents sont accessibles et les alertes sont clairement affichées.

La validation visuelle complète les tests automatisés en montrant l'expérience réelle de l'utilisateur. Elle est particulièrement utile dans un rapport académique, car elle permet d'associer chaque exigence à une preuve concrète issue de l'application.

# Conclusion générale

Le projet **StockPilot** a permis de concevoir et développer une application web complète de gestion de stock. La solution couvre les besoins essentiels : authentification, tableau de bord, gestion du catalogue, saisie des mouvements, alertes de seuil, gestion des catégories, gestion des fournisseurs et génération de documents PDF.

L'utilisation de Spring Boot, Thymeleaf, Spring Security, JPA, H2/MySQL et PDFBox a permis de construire une architecture cohérente, maintenable et adaptée à une application de gestion. Les règles métier importantes, comme la prévention du stock négatif et la synchronisation des alertes, sont centralisées dans les services, ce qui améliore la fiabilité du système.

Les tests Maven réalisés indiquent que le contexte applicatif se charge correctement et que le service PDF produit un document valide. Les captures d'écran intégrées montrent que les interfaces principales sont fonctionnelles, lisibles et adaptées à un usage quotidien.

Ce projet constitue donc une base solide pouvant évoluer vers une solution plus avancée de gestion d'inventaire, avec davantage de rôles, d'indicateurs, d'exports, de statistiques et d'intégrations.

# Perspectives et améliorations futures

Plusieurs perspectives peuvent être envisagées :

- ajouter des rôles distincts comme administrateur, magasinier et lecteur ;
- relier les documents commerciaux aux mouvements de stock ;
- ajouter des exports Excel et CSV ;
- intégrer des codes-barres ou QR codes ;
- ajouter une gestion des emplacements de stockage ;
- créer un historique détaillé par produit ;
- ajouter des graphiques d'évolution du stock ;
- renforcer les tests unitaires et d'intégration ;
- mettre en place une API REST ;
- préparer un déploiement Docker ;
- ajouter des notifications par email lorsque le seuil est atteint ;
- améliorer la recherche globale ;
- connecter l'application à un système d'achat ou de facturation.

# Bibliographie


[1] Spring Boot Documentation, https://docs.spring.io/spring-boot/.

[2] Spring Security Documentation, https://docs.spring.io/spring-security/reference/.

[3] Spring Data JPA Documentation, https://docs.spring.io/spring-data/jpa/reference/.

[4] Thymeleaf Documentation, https://www.thymeleaf.org/documentation.html.

[5] H2 Database Engine Documentation, https://www.h2database.com/html/main.html.

[6] Apache PDFBox Documentation, https://pdfbox.apache.org/.

[7] Bootstrap Documentation, https://getbootstrap.com/docs/.

# Annexe : structure des fichiers

Listing 9.1 – Structure simplifiée du projet

```
src/main/java/com/gestionstock
  config/
  controller/
  model/
  repository/
  service/

src/main/resources
  application.properties
  static/css/app.css
  static/js/app.js
  templates/
```

# Annexe : commandes utiles

Listing 9.2 – Commandes principales

```
.\mvnw.cmd test
.\mvnw.cmd spring-boot:run
```