Système de gestion des stocks d’une pharmacie

Rapport de projet de stage

Nom de l'étudiant : Ismail Arrame

Poste de stage : Stagiaire en développement logiciel

Mentor/Superviseur : [Nom du mentor/Nom du superviseur]

Nom de la pharmacie : [Nom de la pharmacie]

Remerciements

Je tiens à exprimer ma sincère gratitude à [Nom de la pharmacie] pour m'avoir offert l'opportunité d'effectuer ce stage. Je suis particulièrement reconnaissant à mon mentor, [Nom du mentor/Nom du superviseur], pour ses précieux conseils, son soutien et sa patience tout au long du développement de ce projet. Leurs conseils et leurs commentaires ont été essentiels à la réalisation du projet et à mon apprentissage.

Je remercie également l'équipe de [Nom de la pharmacie] pour sa coopération et pour m'avoir fourni les informations et le contexte nécessaires sur le fonctionnement de la pharmacie, essentiels à la compréhension des exigences du projet.

Enfin, je remercie [Nom de l'université, le cas échéant] de m'avoir transmis les connaissances fondamentales qui m'ont permis de contribuer à ce projet.

Résumé

Ce rapport détaille le développement d'un système de gestion des stocks pour pharmacie, une application web conçue pour rationaliser et moderniser les processus de contrôle des stocks chez [Nom de la pharmacie]. L'objectif principal de ce projet était de remplacer les

méthodes manuelles ou obsolètes de suivi des stocks par une solution numérique efficace, fiable et conviviale. Ce système facilite la gestion des catégories de produits, des fournisseurs et des mouvements de stock, y compris les achats, les ventes, les ajustements et les retours. Les principaux résultats obtenus comprennent une meilleure précision des niveaux de stock, un meilleur suivi des mouvements de produits, une meilleure gestion des fournisseurs et une efficacité opérationnelle globale de la pharmacie. Le projet a été développé avec le framework Laravel pour le backend et une combinaison de Tailwind CSS et Alpine.js pour le frontend, offrant une interface utilisateur robuste et réactive.

1. Introduction

1.1 Contexte de la pharmacie

[Nom de la pharmacie] est une pharmacie d'officine qui fournit des services de santé essentiels et des produits pharmaceutiques à ses clients. Comme pour de nombreuses pharmacies, une gestion efficace des stocks est essentielle à son fonctionnement quotidien. Elle garantit la disponibilité constante des médicaments et autres produits de santé pour répondre aux besoins des patients, tout en minimisant le gaspillage dû aux dates de péremption ou aux surstocks. Avant ce projet, [Nom de la pharmacie] [Décrivez brièvement le système d'inventaire existant : par exemple, s'appuyait sur une tenue de dossiers manuelle, utilisait un système obsolète aux fonctionnalités limitées, etc.]. Ce système présentait plusieurs défis, notamment [mentionnez des difficultés spécifiques telles que les écarts de stock, la difficulté de suivi des dates de péremption, l'inefficacité des processus de commande, etc.].

1.2 Motivation du projet

Le développement du système de gestion des stocks pour pharmacies est né de la nécessité de remédier aux inefficacités et aux limites des processus d'inventaire existants chez [Nom de la pharmacie]. Une solution numérique moderne a été imaginée pour offrir une visibilité des stocks en temps réel, automatiser les tâches courantes, réduire les erreurs humaines et fournir des données précieuses pour la prise de décision. Le projet visait à doter le personnel de la pharmacie d'outils pour gérer les stocks plus efficacement, contribuant ainsi à l'amélioration des soins aux patients et à la performance de l'entreprise.

1.3 Énoncé du problème

Le problème principal abordé par ce projet était l'absence d'un système intégré et efficace de gestion des stocks en pharmacie. Cela a entraîné :

- Des inventaires de stocks inexacts, pouvant entraîner des ruptures de stock ou des surstocks ;
- Des processus manuels chronophages pour le suivi des mouvements de produits, des commandes et des informations fournisseurs ;
- Des difficultés à générer des rapports complets pour l'analyse et l'audit des stocks ;
- Un risque accru d'erreurs dans la distribution et la gestion des médicaments en raison de données d'inventaire incomplètes.

Le système de gestion des stocks en pharmacie a été conçu pour apporter une solution complète à ces problèmes.

2. Objectifs du projet

Les principaux objectifs du système de gestion des stocks de pharmacie étaient les suivants :

1. **Développer une base de données centralisée** : Stocker et gérer les informations sur les produits, les catégories, les fournisseurs et les niveaux de stock.
2. **Mettre en œuvre les fonctionnalités principales de gestion des stocks** : Ajout de nouveaux produits, mise à jour des quantités en stock, catégorisation des articles et gestion des informations sur les fournisseurs.
3. **Suivre les mouvements de stock** : Enregistrer toutes les entrées (achats, retours clients) et sorties (ventes, dommages, ajustements) de produits avec horodatage et responsabilisation des utilisateurs.
4. **Authentification et autorisation des utilisateurs** : Garantir un accès sécurisé au système grâce à des autorisations basées sur les rôles (le cas échéant, accès général authentifié).
5. **Fournir une interface utilisateur conviviale** : Concevoir une interface web intuitive et conviviale pour permettre au personnel de la pharmacie d'effectuer efficacement les tâches de gestion des stocks. 6. **Améliorer les capacités de reporting** : Permettre la génération de rapports de base sur les niveaux de stock, les

mouvements de produits et l'activité des fournisseurs (bien que des rapports avancés pourraient faire l'objet de travaux ultérieurs).

6. **Améliorer la précision des données** : Réduire les erreurs liées à la saisie manuelle des données et fournir une source fiable d'informations sur les stocks.
7. **Rationaliser la gestion des fournisseurs** : Faciliter l'ajout, la consultation et la mise à jour des informations sur les fournisseurs, y compris leurs coordonnées et leur statut (actif/inactif).

3. Pile technologique

Le choix de la pile technologique a été guidé par les exigences d'une application web robuste, évolutive et maintenable, ainsi que par les ressources et les délais de développement disponibles.

- **Framework back-end :** Laravel (version 12.x)
- **Argumentaire :** Laravel est un framework PHP puissant, reconnu pour sa syntaxe élégante, ses nombreuses fonctionnalités (ORM, routage, moteur de templates) et son fort soutien communautaire. Son architecture Modèle-Vue-Contrôleur (MVC) favorise un code organisé et un développement rapide, ce qui le rend idéal pour la création d'applications web complexes comme un système de gestion des stocks.
- **Langage de programmation (back-end) :** PHP (version 8.2+)
- **Argumentaire :** Laravel étant un framework PHP, PHP s'est imposé comme le choix naturel pour le développement back-end. La version 8.2 offre des améliorations de performances et des fonctionnalités de langage modernes. * **Style Frontend** : Tailwind CSS (version 3.1.0+)
- **Raisonnement** : Tailwind CSS est un framework CSS axé sur les utilitaires qui permet un développement rapide de l'interface utilisateur en intégrant les utilitaires directement dans le balisage HTML. Cette approche offre une grande flexibilité et permet de maintenir un système de conception cohérent sans avoir à écrire de CSS personnalisé pour chaque composant.
- **Framework/Bibliothèque JavaScript Frontend** : Alpine.js (version 3.4.2+)
- **Raisonnement** : Alpine.js est un framework JavaScript minimal et robuste qui offre un comportement réactif et déclaratif au HTML. Léger, il s'intègre parfaitement aux applications rendues par serveur, comme celles développées avec Laravel, ce qui le rend idéal pour ajouter de l'interactivité sans la surcharge d'un framework frontend plus volumineux. * **Serveur Web** : (généralement Apache ou Nginx,

couramment utilisé avec Laravel. Si vous utilisez Laravel Sail, il est basé sur Docker avec Nginx.)

- **Raisonnement** : Serveurs Web standards et fiables, capables de gérer efficacement les applications PHP.
- **Base de données** : (probablement MySQL ou PostgreSQL, couramment utilisé avec Laravel. La configuration du projet utilise SQLite pour le développement local, conformément aux scripts « composer.json ».)
- **Raisonnement** : Les bases de données relationnelles sont particulièrement adaptées aux données structurées comme les inventaires. L'ORM Eloquent de Laravel offre une intégration transparente avec diverses bases de données SQL.
- **Contrôle de version** : Git (et GitHub pour les workflows)
- **Raisonnement** : Essentiel pour le suivi des modifications de code, la collaboration (le cas échéant) et la gestion de l'historique du projet. * **Gestionnaires de paquets** :
- Composer (pour les dépendances PHP)
- NPM (pour les dépendances JavaScript)
- **Raisonnement** : Outils standards pour gérer les dépendances des projets dans leurs écosystèmes respectifs.
- **Environnement de développement** : Laravel Sail (facultatif, mais indiqué par `composer.json`)
- **Raisonnement** : Laravel Sail fournit un environnement de développement local basé sur Docker, simplifiant la configuration et garantissant la cohérence entre les différentes machines de développement.
- **Outil de build front-end** : Vite (version 6.2.4+)
- **Raisonnement** : Vite est un outil de build front-end moderne qui offre un remplacement rapide des modules à chaud (HMR) pour le développement et des builds optimisées pour la production. C'est le bundler de ressources par défaut pour les nouveaux projets Laravel.

4. Architecture du système

Le système de gestion des stocks de pharmacies suit un modèle architectural standard de type Modèle-Vue-Contrôleur (MVC), inhérent au framework Laravel.

4.1 Présentation

- **Modèle :** représente la structure des données et la logique métier. Dans ce projet, les modèles Eloquent (Utilisateur, Catégorie, Produit, Fournisseur, Mouvement de Stock) interagissent avec la base de données, gèrent la validation des données et définissent les relations entre les différentes entités.
- **Vue :** est responsable de la présentation des données à l'utilisateur et de la gestion des interactions. Les modèles Blade (fichiers `.blade.php` dans le répertoire `resources/views`) sont utilisés pour le rendu HTML, stylisés avec Tailwind CSS et enrichis avec Alpine.js pour l'interactivité.
- **Contrôleur :** sert d'intermédiaire entre le modèle et la vue. Les contrôleurs (`App\Http\Controllers`) gèrent les requêtes HTTP entrantes, récupèrent les données des modèles, les traitent et les transmettent à la vue appropriée pour affichage. Ils gèrent également les entrées utilisateur et déclenchent des actions sur les modèles.

4.2 Cycle de vie des requêtes (simplifié)

1. Un utilisateur interagit avec l'interface web (par exemple, il clique sur un bouton pour afficher les produits).
2. Le navigateur envoie une requête HTTP à une URL spécifique.
3. Le système de routage de Laravel (`routes/web.php`) associe l'URL à une action spécifique du contrôleur.
4. L'action du contrôleur est exécutée. Elle peut :
- Interagir avec un ou plusieurs modèles pour récupérer ou manipuler des données (par exemple, `Product::all()` pour obtenir tous les produits).
- Exécuter la logique métier (par exemple, calculer la valeur totale du stock).
5. Le contrôleur transmet ensuite les données traitées à une vue Blade.
6. Le moteur de création de modèles Blade restitue la vue en HTML, qui est renvoyé au navigateur sous forme de réponse HTTP.

4.3 Composants clés et flux

```
graph TD
Utilisateur [Navigateur utilisateur] -- Requête HTTP --> Routeur
[Routeur Laravel (web.php)]
Routeur -- Route vers --> Contrôleur [Contrôleur]
Contrôleur -- Interagit avec --> Modèle [Modèles Eloquents]
```

```
Modèle -- Opérations CRUD --> Base de données [(Base de données)]
Contrôleur -- Transfère les données --> Vue [Vues Blade]
Vue -- Rendu HTML/CSS/JS --> Utilisateur

sous-graphe « Logique applicative »
Contrôleur
Modèle
fin

sous-graphe « Couche de présentation »
Vue
fin

sous-graphe « Couche de données »
Base de données
fin
```

- **Authentification :** Laravel Breeze est utilisé pour l'authentification de base (connexion, inscription, réinitialisation du mot de passe), garantissant que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités de gestion de l'inventaire. * **Routage ingénieux :** L'application utilise le routage ingénieux de Laravel pour les opérations CRUD sur des entités telles que les catégories, les produits, les fournisseurs et les mouvements de stock, ce qui permet des définitions d'itinéraire claires et conventionnelles.

5. Détails de l'implémentation

Cette section décrit les fonctionnalités principales implémentées, la structure du code, les défis rencontrés et la manière dont ils ont été résolus.

5.1 Fonctionnalités principales

1. **Gestion des utilisateurs et authentification** :
- Enregistrement et connexion sécurisés des utilisateurs (gérés par Laravel Breeze).
- Gestion des mots de passe (fonctionnalité de réinitialisation).
- Gestion des profils pour les utilisateurs authentifiés.

2. **Gestion des catégories** :
- Opérations CRUD (Création, Lecture, Mise à jour, Suppression) pour les catégories de produits.
- Chaque produit est associé à une catégorie.
- Implémenté via « CategoryController » et le modèle « Category ».
3. **Gestion des fournisseurs** :
- Opérations CRUD pour les fournisseurs.
- Stockage des informations sur les fournisseurs : nom, adresse e-mail, téléphone, adresse postale, personne à contacter.
- Possibilité de changer de statut de fournisseur (actif/inactif).
- Implémenté via « SupplierController » et le modèle « Supplier ».
4. **Gestion des produits** :
- Opérations CRUD pour les produits.
- Stocke les informations produit : nom, description, prix, quantité disponible et catégorie associée.
- Mise en œuvre via « ProductController » et le modèle « Product ».
5. **Suivi des mouvements de stock** :
- Enregistre toutes les modifications apportées aux quantités de produits.
- Les types de mouvements incluent : « entrée » (achat/réception), « sortie » (utilisation interne/expiration), « ajustement », « vente », « retour », « dommage ».
- Chaque mouvement est lié à un produit, à l'utilisateur ayant effectué l'action et, éventuellement, à un fournisseur (pour les achats).
- Inclut la quantité déplacée et une raison/note pour le mouvement.
- Mise en œuvre via « StockMovementController » et le modèle « StockMovement ». * Le modèle `Product` inclut une méthode d'assistance `updateStock()` pour ajuster sa quantité, qui serait généralement appelée après la création d'un `StockMovement`.

5.2 Code Structure

Le projet suit la structure de répertoire standard de Laravel :

- `app/Http/Controllers/` : Contient les contrôleurs pour la gestion des requêtes HTTP (par exemple, `ProductController.php`, `CategoryController.php`).
- `app/Models/` : Contient les modèles ORM Eloquent (par exemple, `Product.php`, `User.php`).

- `database/migrations/` : Contient les fichiers de migration de base de données définissant le schéma (par exemple, `create_products_table.php`).
- `routes/web.php` : Définit les routes de l'application web.
- `resources/views/` : Contient les fichiers de modèle Blade pour l'interface utilisateur (par exemple, `products/index.blade.php`).
- `public/` : Racine du document du serveur web, contenant le point d'entrée `index.php` et les ressources compilées.
- `config/` : Fichiers de configuration de l'application.

5.3 Schéma de la base de données

Le schéma de la base de données est défini par les fichiers de migration :

- **Table `users` :** Stocke les informations utilisateur (identifiant, nom, adresse e-mail, mot de passe).
- **Table `password_reset_tokens` :** Pour la réinitialisation du mot de passe.
- **Table `sessions` :** Pour la gestion des sessions utilisateur. * **Table `suppliers` :** (identifiant, nom, e-mail, téléphone, adresse, personne à contacter, actif, horodatage)
- **Table `categories` :** (identifiant, nom, horodatage)
- **Table `products` :** (identifiant, nom, description, prix, quantité, category_id (clé étrangère vers les catégories), horodatage)
- **Table `stock_movements` :** (identifiant, product_id (clé étrangère vers les produits), type, quantité, motif, supplier_id (clé étrangère vers les fournisseurs, nullable), user_id (clé étrangère vers les utilisateurs), horodatage)

(Voir l'annexe A pour des extraits de code de migration détaillés)

5.4 Défis rencontrés et solutions

- **Défi 1 : Garantir l'intégrité des données pour les quantités en stock.**
- **Problème :** La mise à jour directe des quantités de produits sans piste d'audit appropriée pouvait entraîner des écarts. La simple incrémentation/décrémentation d'un champ « quantité » dans la table « produits » n'est pas fiable.
- **Résolution :** Mise en œuvre de la table « stock_movements ». Au lieu de modifier directement la table « produits.quantité », chaque modification (vente, achat, ajustement) crée un nouvel enregistrement dans « stock_movements ». La valeur réelle de « produits.quantité » peut ensuite être calculée en additionnant les

mouvements pertinents ou, comme implémenté, mise à jour via une méthode auxiliaire dans le modèle « Produit » après l'enregistrement d'un mouvement de stock. Cela permet une piste d'audit complète.

- **Défi 2 : Conception d'un système flexible de gestion des mouvements de stock.**
- **Problème :** Les stocks peuvent varier pour diverses raisons (achat, vente, avarie, transfert interne, retours). Un système rigide serait difficile à adapter.
- **Résolution :** La table « stock_movements » comprend une colonne « type » (par exemple, « entrée », « sortie », « ajustement », « vente », « achat », « retour », « avarie ») et un champ texte « raison ». Cela permet de catégoriser les mouvements et d'ajouter des détails spécifiques, rendant le système flexible.
- **Défi 3 : Interface utilisateur pour plusieurs opérations CRUD.**
- **Problème :** Créer des interfaces utilisateur intuitives et cohérentes pour la gestion des catégories, des produits, des fournisseurs et des mouvements de stock peut prendre du temps.
- **Résolution :** Exploitation du modèle Blade de Laravel avec Tailwind CSS pour un développement rapide de l'interface utilisateur. Des composants Blade réutilisables (par exemple, pour les formulaires, les tableaux et les fenêtres modales) ont été créés pour garantir la cohérence et réduire la duplication du code. Alpine.js a été utilisé pour les interactions mineures côté client, comme le basculement entre les fenêtres modales ou les éléments de formulaire dynamiques sans rechargement complet de la page.
- **Défi 4 : Gestion de l'activité des fournisseurs.**
- **Problème :** Nécessité de suivre les fournisseurs actifs et inactifs, sans supprimer leurs données historiques (par exemple, leurs achats passés).
- **Solution :** Ajout d'un champ booléen « actif » à la table « fournisseurs ». Au lieu de supprimer un fournisseur, il peut être marqué comme inactif. Cela préserve l'intégrité des données historiques tout en permettant de le filtrer hors des listes de fournisseurs actifs.

6. Tests et validation

Assurer la fiabilité et l'exactitude du système de gestion des stocks de pharmacie était une priorité absolue. Les stratégies de test et de validation suivantes ont été utilisées :

6.1 Tests unitaires (conceptuels)

Bien que la structure de projet fournie inclue PestPHP pour les tests, l'étendue de la mise en œuvre des tests unitaires dépend du processus de développement. Conceptuellement, les tests unitaires se concentreraient sur :

- **Logique du modèle** : tester les relations entre les modèles (par exemple, « Product » appartient à « Category »), les méthodes personnalisées (par exemple, « Product->updateStock() ») et le transtypage des attributs.
- *Exemple* : un test pour s'assurer que lorsque « updateStock() » est appelé sur un modèle « Product », son attribut « quantity » est correctement mis à jour.
- **Logique du contrôleur (basique)** : tester que les actions du contrôleur renvoient les vues ou réponses correctes et que les données sont correctement transmises aux vues.
- *Exemple* : un test pour vérifier que l'action « ProductController@index » renvoie une vue contenant une liste de produits.

6.2 Tests de fonctionnalités (conceptuels)

Les tests de fonctionnalités simulent les interactions de l'utilisateur avec l'application afin de vérifier le bon fonctionnement des différents composants.

- **Opérations CRUD** : Test du cycle de vie complet de création, de lecture, de mise à jour et de suppression de catégories, de produits, de fournisseurs et de mouvements de stock via des requêtes HTTP.
- *Exemple* : Test simulant la soumission d'un formulaire par un utilisateur pour créer un nouveau produit, puis vérifiant que le produit est enregistré dans la base de données et redirigeant l'utilisateur vers la page de liste des produits.
- **Authentification et autorisation** : Test de la connexion, de l'inscription et du contrôle d'accès pour les routes protégées.
- *Exemple* : Test garantissant qu'un utilisateur non authentifié tentant d'accéder à « /products » est redirigé vers la page de connexion.

6.3 Tests manuels

Des tests manuels approfondis ont été réalisés tout au long du cycle de développement afin de couvrir les scénarios difficiles à automatiser et d'évaluer l'utilisabilité :

- **Tests de l'interface utilisateur :** Vérification du bon fonctionnement de tous les formulaires, boutons, liens et éléments d'affichage dans l'application.
- **Tests des workflows :** Simulation de workflows réels en pharmacie, tels que :

1. Ajout d'un nouveau fournisseur.
2. Ajout d'une nouvelle catégorie de produits.
3. Ajout d'un nouveau produit dans cette catégorie, provenant de ce fournisseur.
4. Enregistrement d'un achat (stock « in ») pour ce produit.
5. Vérification de la mise à jour de la quantité du produit.
6. Enregistrement d'une vente (rupture de stock ou mouvement de type « vente » spécifique) pour ce produit.
7. Vérification de la mise à jour de la quantité du produit.
8. Régulation des stocks (par exemple, pour les produits endommagés).

- **Tests de validation des données** : s'assurer que les règles de validation des formulaires (par exemple, les champs obligatoires, le format des e-mails, les valeurs numériques) fonctionnent correctement et fournissent des retours utilisateur appropriés.
- **Tests de cas limites** : tester avec des entrées ou des séquences d'actions inhabituelles afin d'identifier les problèmes potentiels.

6.4 Validation

Les fonctionnalités de validation intégrées de Laravel ont été largement utilisées :

- **Validation des requêtes de formulaire** : pour les scénarios de validation plus complexes, des classes de requêtes de formulaire dédiées (par exemple, dans `app/Http/Requests/`) ont été utilisées pour encapsuler la logique de validation et maintenir les contrôleurs propres.
- **Validation du contrôleur** : pour les cas plus simples, la validation a été gérée directement dans les méthodes du contrôleur à l'aide de la méthode `$request->validate()`.
- **Contraintes de la base de données** : des contraintes uniques (par exemple, e-mail du fournisseur, e-mail de l'utilisateur) et des contraintes de clé étrangère ont été définies lors des migrations de bases de données afin de garantir l'intégrité des données au niveau de la base de données.

6.5 Ajustements effectués

- **Conception initiale :** Initialement, les mises à jour de stock auraient pu être considérées comme des modifications directes de la quantité de produits.
- **Ajustement :** La nécessité d'une piste d'audit a été prise en compte, ce qui a conduit à la création de la table « stock_movements » pour enregistrer chaque transaction, améliorant ainsi la traçabilité et la responsabilisation.
- **Désactivation des fournisseurs :** L'idée initiale était peut-être de supprimer des fournisseurs.
- **Ajustement :** Une désactivation « souple » (indicateur « actif ») a été mise en place pour conserver l'historique des transactions passées avec ce fournisseur.
- **Commentaires des utilisateurs lors des tests manuels :** Des ajustements mineurs ont été apportés à l'interface utilisateur suite aux évaluations de la facilité d'utilisation réalisées lors des tests manuels, notamment l'amélioration de la présentation des formulaires et la simplification de la navigation.

7. Déploiement

Cette section décrit le processus conceptuel de déploiement d'une application Laravel telle que le système de gestion des stocks de pharmacie. Le déploiement réel dépend de l'infrastructure informatique et des préférences de la pharmacie.

7.1 Options d'environnement de déploiement

Plusieurs options sont possibles pour déployer une application Laravel :

1. Hébergement mutualisé traditionnel / VPS :
- **Processus :** Configurer manuellement un serveur (par exemple, Ubuntu avec la pile LEMP/LAMP), cloner le dépôt Git, installer les dépendances (`composer install`, `npm install && npm run build`), configurer le fichier `.env`, configurer le serveur web (Nginx/Apache) et configurer les connexions à la base de données.
- **Considérations :** Nécessite des connaissances en administration serveur. Moins automatisé.
2. **Plateforme en tant que service (PaaS)** :
- **Exemples** : Heroku, Laravel Forge, Ploi.io, AWS Elastic Beanstalk, Google App Engine.

- **Processus** : Ces plateformes simplifient souvent le déploiement. Elles impliquent généralement la connexion au dépôt Git, la configuration des étapes de build, et la plateforme gère le provisionnement, la mise à l'échelle et la maintenance du serveur.
- **Considérations** : Plus faciles à gérer, souvent plus coûteuses, mais permettent un gain de temps considérable sur la gestion de l'infrastructure. Laravel Forge et Ploi sont spécifiquement conçues pour les applications PHP/Laravel.

3. **Conteneurisation (Docker)** :

- **Processus** : L'application (y compris Laravel Sail pour le développement local) est déjà configurée pour être compatible avec Docker. Pour la production, un « Dockerfile » sera créé (ou celui de Sail pourrait être adapté). L'application serait intégrée à une image Docker et déployée sur une plateforme d'orchestration de conteneurs (par exemple, Kubernetes, Docker Swarm) ou un service exécutant des conteneurs (par exemple, AWS ECS, Google Cloud Run).
- **Considérations :** Hautement évolutive, portable et garantissant la cohérence entre les environnements de développement et de production. Courbe d'apprentissage plus raide pour l'orchestration.

7.2 Étapes générales de déploiement (conceptuelles)

Quel que soit l'environnement choisi, les étapes générales incluent :

1. **Préparation du code** : Assurez-vous que la branche « main » (ou « master ») du dépôt Git dispose du code stable le plus récent.
2. **Configuration du serveur** : Provisionnez et configurez le serveur ou la plateforme de déploiement.
3. **Configuration de l'environnement (fichier « .env »)** :

- `APP_ENV=production`
- `APP_DEBUG=false`
- `APP_KEY` : Doit être généré et défini (`php artisan key:generate`).
- `DB_CONNECTION`, `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD` : Configurez pour la base de données de production.
- Paramètres du pilote de messagerie, des files d'attente, du cache, du pilote de session, etc.

4. **Installer les dépendances :** `composer install --optimize-autoloader --no-dev`
5. **Créer les ressources frontend :** `npm install && npm run build` (ou commande Vite équivalente).

6. **Exécuter les migrations de bases de données :** `php artisan migrate --force` (l'option `--force` est généralement utilisée en production pour exécuter des migrations sans confirmation).
7. **Optimiser Laravel :**

- `php artisan config:cache`
- `php artisan route:cache`
- `php artisan view:cache` (le cas échéant)

8. **Définir les autorisations de fichiers :** Assurez-vous que le serveur web dispose d'un accès en écriture aux répertoires nécessaires (par exemple, `storage`, `bootstrap/cache`). 9. **Configurer le serveur Web** : pointez la racine des documents du serveur Web vers le répertoire « public » de l'application Laravel.
9. **Configurer les files d'attente (le cas échéant) : configurez un gestionnaire de file d'attente (par exemple, Supervisor) si l'application utilise les files d'attente Laravel pour les tâches en arrière-plan.
10. **Configurer la planification des tâches (le cas échéant) : configurez la tâche cron pour exécuter « php artisan schedule:run » toutes les minutes si vous utilisez le planificateur de Laravel.
11. **Tests** : testez minutieusement l'application déployée en environnement de production.

7.3 Spécificités de ce projet

- Étant donné que le fichier « composer.json » inclut « laravel/sail », si la pharmacie disposait d'un environnement compatible Docker, le déploiement à l'aide d'une configuration Docker prête pour la production serait une bonne option pour garantir la cohérence. * Le script « dev » dans « composer.json » (`php artisan serve`, `npm run dev`) est destiné au développement local uniquement et ne doit pas être utilisé en production.

8. Conclusion et perspectives

8.1 Réflexion sur l'expérience

Le développement du système de gestion des stocks de pharmacie a été une expérience extrêmement enrichissante. Ce stage m'a permis d'appliquer concrètement mes

connaissances théoriques en développement logiciel, notamment avec le framework Laravel, afin de résoudre un problème concret pour [Nom de la pharmacie].

Principaux enseignements :

- **Développement full-stack :** Expérience pratique du développement back-end avec Laravel (PHP), de la conception et de la gestion de bases de données (SQL, Eloquent ORM) et du développement front-end (Blade, Tailwind CSS, Alpine.js).
- **Architecture MVC :** Compréhension approfondie du modèle Modèle-Vue-Contrôleur et de ses avantages pour l'organisation d'applications complexes.
- **Résolution de problèmes :** Divers défis techniques ont été rencontrés et résolus, notamment la garantie de l'intégrité des données de stock et la conception d'un système flexible pour les mouvements de stock. * **Importance de la planification** : L'importance d'exigences claires et d'une planification dès les premières étapes d'un projet a été reconnue pour guider efficacement le développement.
- **Contrôle des versions** : Git a été utilisé pour le contrôle des versions, renforçant ainsi les bonnes pratiques de gestion du code.
- **Application concrète** : Comprendre les besoins spécifiques et le contexte opérationnel d'une pharmacie a été crucial pour concevoir un système utile et pertinent.

Le projet a atteint ses objectifs principaux en fournissant un système fonctionnel capable de gérer les catégories de produits, les fournisseurs, les produits et de suivre les mouvements de stock. L'interface utilisateur, bien que fonctionnelle, constitue une base solide pour les améliorations futures.

8.2 Suggestions de développement et d'évolution

Le système actuel offre une base solide pour la gestion des stocks. Cependant, plusieurs fonctionnalités pourraient être ajoutées ou améliorées ultérieurement afin d'optimiser son utilité pour [Nom de la pharmacie] :

1. **Rapports et analyses avancés** :
- Mettre en œuvre des fonctionnalités de reporting plus sophistiquées, telles que des rapports sur les articles en rupture de stock, les produits périmés (nécessitant l'ajout d'une date de péremption aux produits), les tendances des ventes et les performances des fournisseurs.
- Tableaux de bord visuels avec graphiques et diagrammes pour une analyse rapide.
2. **Suivi des dates de péremption** :

- Ajouter un champ « expiry_date » à la table « products » (ou une table de suivi des lots distincte si les produits peuvent avoir plusieurs dates de péremption).
- Mettre en place des alertes pour les produits approchant leur date de péremption.

3. **Suivi des numéros de lot** : Pour les produits pharmaceutiques, le suivi des numéros de lot est souvent crucial pour les rappels et le contrôle qualité. 4. **Intégration des codes-barres/codes QR** :

- Permettre la lecture des codes-barres/codes QR des produits pour une recherche et un inventaire plus rapides.

5. **Rôles et autorisations des utilisateurs** :

- Mettre en place un système de contrôle d'accès plus précis basé sur les rôles (par exemple, pharmacien, technicien, administrateur) avec différents niveaux d'autorisation.

6. **Gestion des bons de commande** :

- Développer un module de création et de gestion des bons de commande fournisseurs.

7. **Intégration du module de vente** :

- Si la pharmacie utilise un système de point de vente (PDV) distinct, explorer les possibilités d'intégration pour mettre à jour automatiquement les stocks dès la vente.
- Vous pouvez également étendre le type de mouvement de stock actuel « vente » à une fonctionnalité d'enregistrement des ventes plus complète.

8. **Notifications et alertes** :

- Configurer des notifications automatiques en cas de faible niveau de stock, d'articles proches de la date de péremption ou de commandes en retard.

9. **Améliorations du journal d'audit** : Bien que « stock_movements » fournisse une piste d'audit pour les variations de quantité, un journal d'audit plus complet permettrait de suivre toutes les actions importantes au sein du système (par exemple, qui a modifié les informations d'un produit).
10. **Réactivité mobile et PWA** : Améliorez davantage la réactivité mobile et envisagez de la développer en application web progressive (PWA) pour de meilleures fonctionnalités hors ligne ou une expérience similaire à celle d'une application mobile.
11. **Importation/exportation de données** : Permettez l'importation en masse de données produit (par exemple, à partir d'un fichier CSV) et l'exportation de rapports ou de données d'inventaire.

Ces améliorations potentielles pourraient étendre considérablement les capacités du système de gestion des stocks de pharmacie, offrant ainsi une valeur ajoutée encore plus

importante à [Nom de la pharmacie]. La base actuelle est évolutive et bien structurée pour s'adapter à ces développements futurs.

Annexe

Annexe A : Extraits de code de migration de base de données

create_suppliers_table.php

```
// ... (up() method)
Schema::create('suppliers', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('phone')->nullable();
    $table->text('address')->nullable();
    $table->string('contact_person')->nullable();
    $table->boolean('active')->default(true);
    $table->timestamps();
});
// ...
```

create_categories_table.php

```
// ... (up() method)
Schema::create('categories', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->timestamps();
});
// ...
```

create_products_table.php

```
// ... (up() method)
Schema::create('products', function (Blueprint $table) {
```

```
    $table->id();
    $table->string('name');
    $table->text('description')->nullable();
    $table->decimal('price', 8, 2);
    $table->integer('quantity')->default(0);
    $table->foreignId('category_id')->constrained()-
>onDelete('cascade');
    $table->timestamps();
});
// ...
```

create_stock_movements_table.php

```
// ... (up() method)
Schema::create('stock_movements', function (Blueprint $table) {
    $table->id();
    $table->foreignId('product_id')->constrained()-
>onDelete('cascade');
    $table->string('type'); // 'in', 'out', 'adjustment', 'sale',
'return'
    $table->integer('quantity');
    $table->text('reason')->nullable();
    $table->unsignedBigInteger('supplier_id')->nullable();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
// who made this movement
    $table->timestamps();

    $table->foreign('supplier_id')
                ->references('id')
                ->on('suppliers')
                ->onDelete('set null');
});
// ...
```

Annexe B : Extraits de relations de modèle

app/Models/Product.php

```
// ...
public function category(): BelongsTo
{
    return $this->belongsTo(Category::class);
}

public function stockMovements(): HasMany
{
    return $this->hasMany(StockMovement::class);
}
// ...
```

app/Models/StockMovement.php

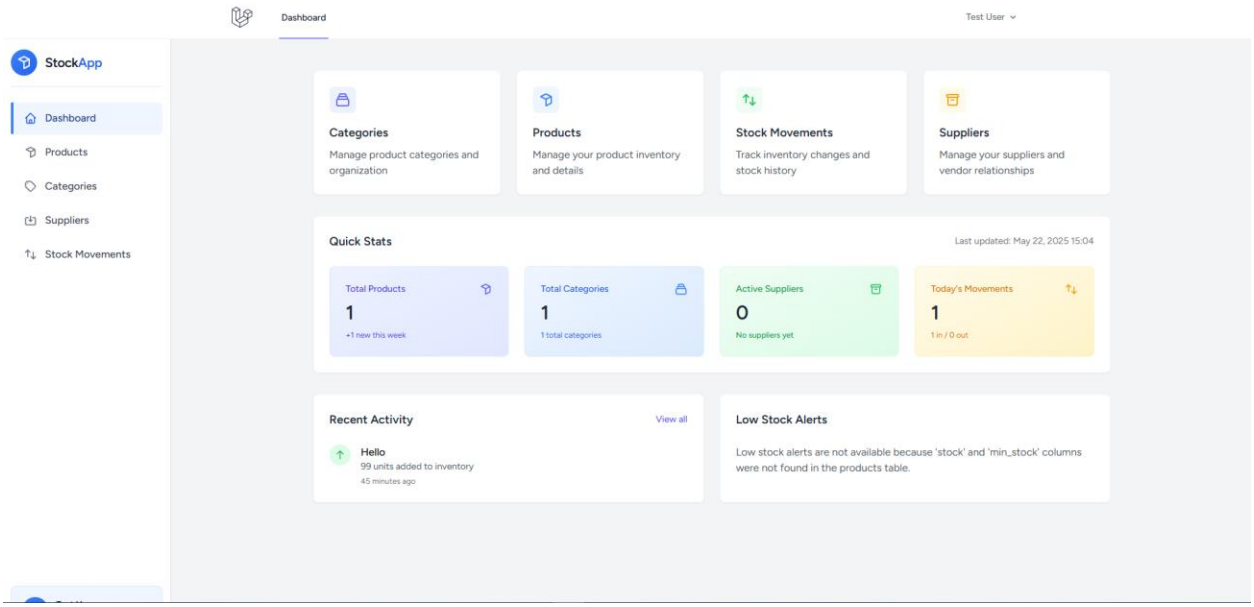
```
// ...
public function product(): BelongsTo
{
    return $this->belongsTo(Product::class);
}

public function supplier(): BelongsTo
{
    return $this->belongsTo(Supplier::class);
}

public function user(): BelongsTo
{
    return $this->belongsTo(User::class);
}
// ...
```

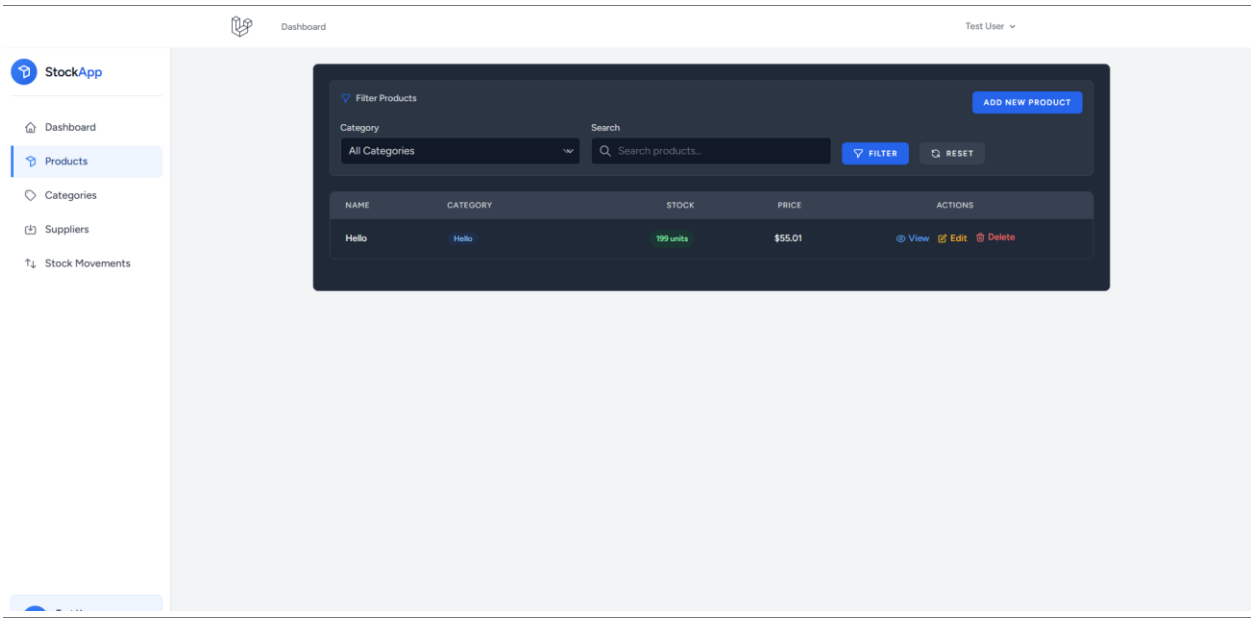
Annexe C : Exemples de définitions d'itinéraire (à partir de `routes/web.php`)

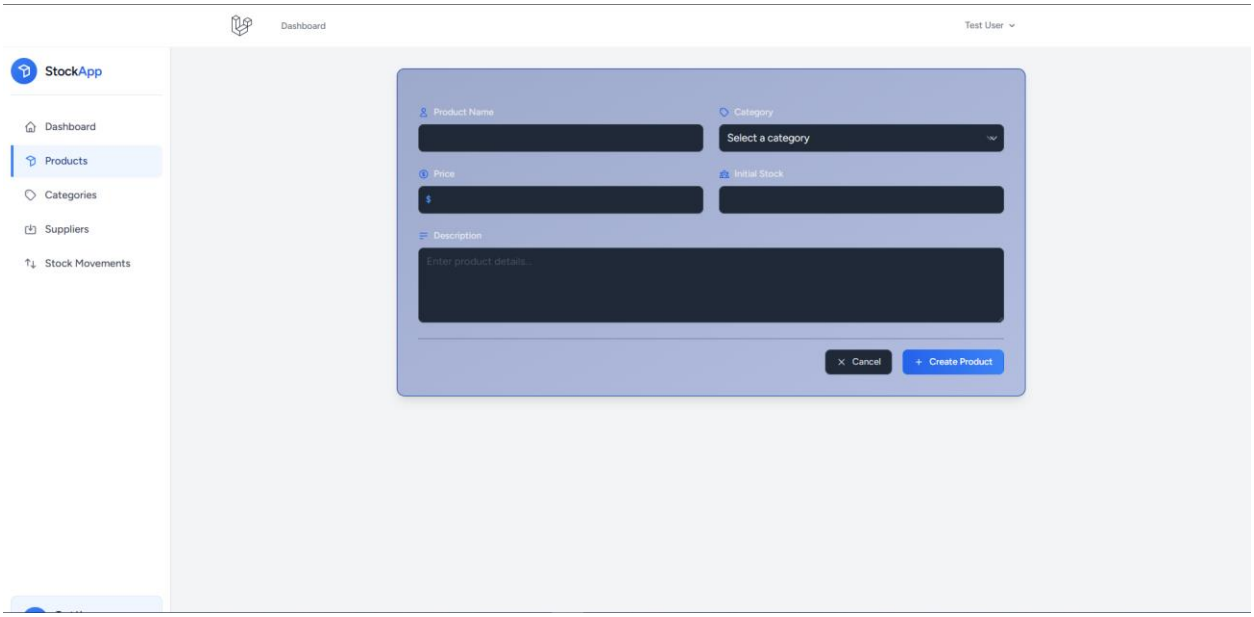
```
// ...
Route::middleware('auth')->group(function () {
    // ... autres routes ...
```

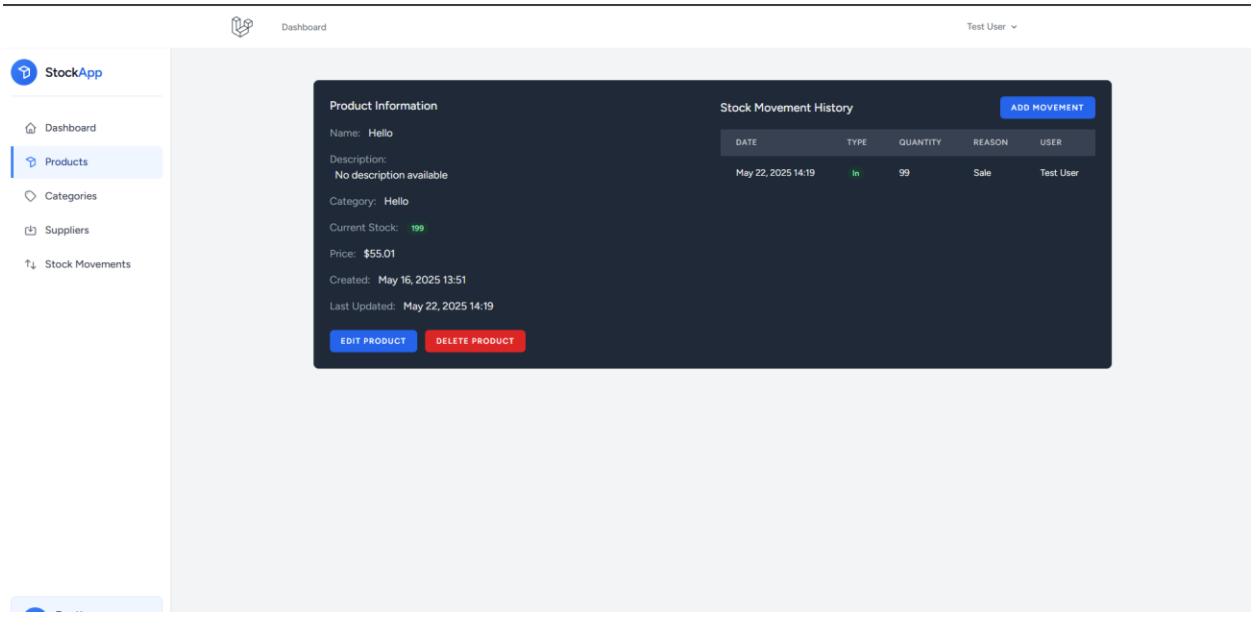
```
    Route::middleware(['verified'])->group(function () {
        Route::resource('categories', CategoryController::class);
        Route::resource('products', ProductController::class);
        Route::resource('suppliers', SupplierController::class);
        Route::patch('suppliers/{supplier}/toggle-active',
[SupplierController::class, 'toggleActive'])-
>name('suppliers.toggleActive');
        Route::resource('stock-movements',
StockMovementController::class);
    });
});
// ...
```

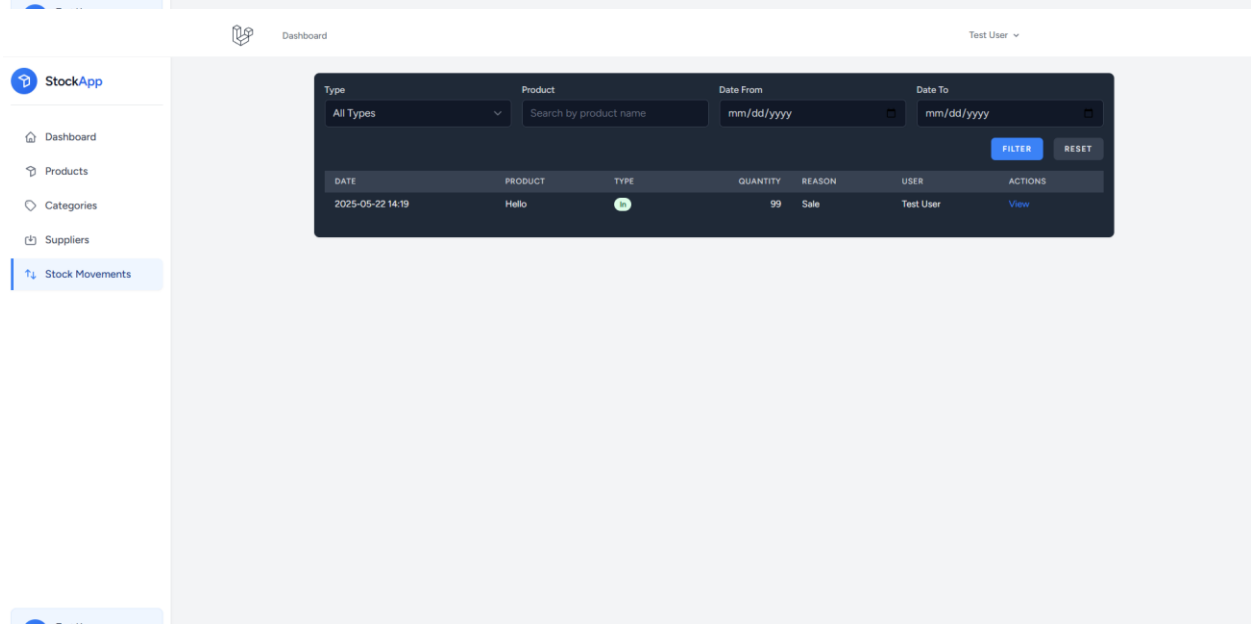
Section de capture:




Dashboard
Test User
StockApp
Dashboard
Products
Categories
Suppliers
Stock Movements
Filter Products
ADD NEW PRODUCT
Category
All Categories
Search
Search products...
FILTER
RESET
NAME
CATEGORY
STOCK
PRICE
ACTIONS
Hello
Hello
199 units
$55.01
View
Edit
Delete


Dashboard
Test User
StockApp
Dashboard
Products
Categories
Suppliers
Stock Movements
Product Name
Category
Select a category
Price
Initial Stock
Description
Enter product details...
Cancel
Create Product





Fin du rapport