

2025-01-17

Annéé scolaire 2024/2025

*UNITE D’ENSEIGNEMENT: PROJET D’INTÉGRATION DE DÉVELOPPEMENT*

Professeur: Martel Alain

SHOP CONNECT : APPLICATION DE GESTION COMMERCIALE

*AUTEUR: VANHEES VINCENT*

Table des matières

[1. Introduction 3](#_Toc188810307)

[1.1 Contexte de développement 3](#_Toc188810308)

[1.2 Contexte technique et solution proposée 3](#_Toc188810309)

[1.2.1 Contexte initial : 3](#_Toc188810310)

[1.2.2 Technologies utilisées : 4](#_Toc188810311)

[1.3 Modèle Conceptuel des Données (M.C.D) 8](#_Toc188810312)

[1.4 Modèle Logique des Données (M.L.D) 8](#_Toc188810313)

[1.5 Analyse de la Base de Données 9](#_Toc188810314)

[1.5.1 Structure de la Base de Données 9](#_Toc188810315)

[1.5.2 Relations entre les Tables 9](#_Toc188810316)

[1.5.3 Flux des Données et Scénarios d'Utilisation 9](#_Toc188810317)

[2. Back-end 10](#_Toc188810318)

[2.1 Package config 10](#_Toc188810319)

[2.2 Package controller 10](#_Toc188810320)

[2.2.1 Fichier : HomeController.java 11](#_Toc188810321)

[2.2.2 Fichier : UserController. 11](#_Toc188810322)

[2.2.3 Fichier : AdminController.java 12](#_Toc188810323)

[2.3 Package model 12](#_Toc188810324)

[2.3.1 Fichier : Cart.java 12](#_Toc188810325)

[2.3.2 Fichier : Category.java 13](#_Toc188810326)

[2.3.3 Fichier : OrderAddress.java 13](#_Toc188810327)

[2.3.4 Fichier : OrderRequest.java 13](#_Toc188810328)

[2.3.5 Fichier : Product.java 13](#_Toc188810329)

[2.3.6 Fichier : ProductOrder.java 13](#_Toc188810330)

[2.3.7 Fichier : UserDtls.java 14](#_Toc188810331)

[2.4 Package Repository 14](#_Toc188810332)

[2.4.1 Fichier : CartRepository.java 14](#_Toc188810333)

[2.4.2 Fichier : CategoryRepository.java 14](#_Toc188810334)

[2.4.3 Fichier : ProductOrderRepository.java 15](#_Toc188810335)

[2.4.4 Fichier : ProductRepository.java 15](#_Toc188810336)

[2.4.5 Fichier : UserRepository.java 16](#_Toc188810337)

[2.5 Package service/impl (fichiers des services d’implémentation (impl)) 16](#_Toc188810338)

[2.5.1 Rôle de impl : 16](#_Toc188810339)

[2.5.2 Fichier : CartServiceImpl.java 17](#_Toc188810340)

[2.5.3 Fichier : CategoryServiceImpl.java 17](#_Toc188810341)

[2.5.4 Fichier : CommnServiceImpl.java 17](#_Toc188810342)

[2.5.5 Fichier : OrderServiceImpl.java 17](#_Toc188810343)

[2.5.6 Fichier : ProductServiceImpl.java 17](#_Toc188810344)

[2.5.7 Fichier : UserDetailsServiceImpl.java 18](#_Toc188810345)

[2.5.8 Fichier : UserServiceImpl.java 18](#_Toc188810346)

[2.6 Package service 18](#_Toc188810347)

[2.6.1 Fichier : CartService.java 18](#_Toc188810348)

[2.6.2 Fichier : CategoryService.java 19](#_Toc188810349)

[2.6.3 Fichier : CommonService.java 19](#_Toc188810350)

[2.6.4 Fichier : OrderService.java 19](#_Toc188810351)

[2.6.5 Fichier : ProductService.java 20](#_Toc188810352)

[2.6.6 Fichier : UserService.java 20](#_Toc188810353)

[2.7 Package util 20](#_Toc188810354)

[2.7.1 AppConstant 20](#_Toc188810355)

[2.7.2 Fichier : CommonUtil.java 21](#_Toc188810356)

[2.7.3 Fichier : OrderStatus.java 21](#_Toc188810357)

[3. Front End 22](#_Toc188810358)

[3.1 Vues par défaut 22](#_Toc188810359)

[3.1.1 Base Template (base.html)​ 22](#_Toc188810360)

[3.1.2 Page de mot de passe oublié (forgot\_password.html)​ 22](#_Toc188810361)

[3.1.3 Page d'accueil (index.html)​ 23](#_Toc188810362)

[3.1.4 Page de connexion (login.html)​ 23](#_Toc188810363)

[3.1.5 Page des messages (message.html)​ 24](#_Toc188810364)

[3.1.6 Page des produits (product.html)​ 25](#_Toc188810365)

[3.1.7 Page d'inscription (register.html)​ 25](#_Toc188810366)

[3.1.8 Page de réinitialisation du mot de passe (reset\_password.html)​ 26](#_Toc188810367)

[3.1.9 Page des détails du produit (view\_product.html)​ 26](#_Toc188810368)

[3.2 Vues du profil administrateur (admin) 26](#_Toc188810369)

[3.2.1 Page Ajouter un Administrateur (add\_admin.html)​ 26](#_Toc188810370)

[3.2.2 Page Ajouter un Produit (add\_product.html)​ 27](#_Toc188810371)

[3.2.3 Page Gérer les Catégories (category.html)​ 27](#_Toc188810372)

[3.2.4 Page Modifier un Produit (edit\_product.html)​ 28](#_Toc188810373)

[3.2.5 Page d'Accueil Administrateur (index.html)​ 28](#_Toc188810374)

[3.2.6 Page Commandes (orders.html)​ 29](#_Toc188810375)

[3.2.7 Page Liste des Produits (products.html)​ 29](#_Toc188810376)

[3.2.8 Page Profil Administrateur (profile.html)​ 30](#_Toc188810377)

[3.2.9 Page Utilisateurs (users.html)​ 30](#_Toc188810378)

[3.3 Vues du profil Utilisateur (user) 30](#_Toc188810379)

[3.3.1 Page du Panier (cart.html)​ 30](#_Toc188810380)

[3.3.2 Page d'Accueil Utilisateur (home.html)​ 31](#_Toc188810381)

[3.3.3 Page de Mes Commandes (my\_orders.html)​ 31](#_Toc188810382)

[3.3.4 Page Commander (order.html)​ 32](#_Toc188810383)

[3.3.5 Page Profil Utilisateur (profile.html)​ 32](#_Toc188810384)

[3.3.6 Page Confirmation de Commande (success.html)​ 33](#_Toc188810385)

[4. Gestion de l'Authentification 33](#_Toc188810386)

[4.1 Technologies Utilisées 33](#_Toc188810387)

[4.1.1 Frameworks et Bibliothèques 33](#_Toc188810388)

[4.1.2 Intégration des Composants 33](#_Toc188810389)

[4.2 Partie Back End 33](#_Toc188810390)

[4.2.1 AuthFailureHandlerImpl (Gestion des échecs d'authentification) 33](#_Toc188810391)

[4.2.2 AuthSucessHandlerImpl (Gestion des succès d'authentification) 34](#_Toc188810392)

[4.2.3 CustomUser (détails des utilisateurs) 34](#_Toc188810393)

[4.2.4 SecurityConfig (Configuration de Spring Security) 35](#_Toc188810394)

[4.3 Partie Front End (Intégration des Vues HTML) 35](#_Toc188810395)

[4.3.1 register.html 35](#_Toc188810396)

[4.3.2 login.html 36](#_Toc188810397)

[4.3.3 forgot\_password.html 36](#_Toc188810398)

[5. Conclusion 36](#_Toc188810399)

[5.1 Perspectives d'Évolution 37](#_Toc188810400)

# Introduction

Shop Connect est une application conçue pour moderniser la gestion commerciale de commerçants locaux. Elle centralise la gestion des produits, des commandes et des promotions tout en offrant une expérience utilisateur intuitive et sécurisée.

Cette documentation détaille les choix technologiques, les réalisations techniques, et les perspectives d'évolution pour ce projet.

## Contexte de développement

Un entrepreneur local souhaite moderniser la gestion commerciale d’un réseau de petits commerçants en développant une plateforme numérique complète. Cette plateforme a pour objectif de centraliser la gestion des boutiques en ligne des commerçants, leur permettant d'afficher leurs produits, de suivre les commandes des clients, et de promouvoir leurs offres de manière efficace.

Le projet répond à un double enjeu :

* Pour les commerçants : Simplifier la gestion des produits et des commandes tout en offrant un outil de marketing intégré.
* Pour les clients : Offrir une expérience utilisateur fluide et moderne pour naviguer, acheter, et gérer leurs commandes.

## Contexte technique et solution proposée

### Contexte initial :

Le projet a été conçu initialement avec une architecture moderne et découplée, combinant Spring Boot pour le backend et Angular pour le frontend. Cette approche permettait une séparation nette entre la logique métier et la présentation, garantissant une évolutivité optimale et une expérience utilisateur interactive.

Cependant, des contraintes liées au temps de développement et à la complexité d’apprentissage d'Angular ont rendu cette architecture difficile à mettre en place et à finaliser dans les délais impartis. En conséquence, une transition stratégique a été opérée vers une intégration plus fluide basée sur Thymeleaf, un moteur de templates Java.

Le choix de Thymeleaf a permis d’accélérer le développement en supprimant la complexité liée à Angular. La génération des pages dynamiques directement dans Spring Boot a réduit la nécessité des appels API tout en garantissant une interface responsive et fonctionnelle.

#### Pourquoi ce changement ?

L’utilisation de Thymeleaf, directement intégré à Spring Boot, a permis de :

* Simplifier le développement en supprimant la nécessité des appels API constants entre le backend et le frontend.
* Générer des pages HTML dynamiques avec des données injectées directement depuis les contrôleurs Spring Boot.
* Respecter les délais tout en conservant la robustesse et la modularité de l’architecture initiale.

### Technologies utilisées :

#### Spring Boot

Le backend de l’application est développé avec Spring Boot, un framework Java moderne et puissant qui simplifie la création d’applications Java, en particulier pour les APIs REST, la gestion des dépendances et l'intégration avec des bases de données relationnelles comme PostgreSQL.

Spring Boot offre une architecture légère et flexible pour construire des APIs RESTful. Grâce à son système de configuration automatique, il réduit considérablement la complexité de la mise en place d'un backend robuste. Spring Boot fournit une infrastructure REST prête à l'emploi, incluant la gestion des requêtes HTTP, des contrôleurs et des filtres pour interagir facilement avec des clients web, des applications mobiles ou d'autres services.

**Structure et Architecture :**

* Modularité et Maintenance

L'architecture de l’application suit un modèle bien défini pour assurer la modularité et la maintenabilité du code. L’application est généralement divisée en trois couches principales :

* Les contrôleurs (Controllers) : Ils gèrent les requêtes HTTP et renvoient les réponses correspondantes. Les contrôleurs sont responsables de l'interface entre le frontend (ou autres services) et la logique métier.
* Les services (Services) : Cette couche contient la logique métier de l'application. Elle permet de séparer la gestion des données (repositories) de la gestion des flux et des processus d'affaires. Le service interagit avec les repositories pour effectuer les opérations sur les données.
* Les repositories (Repositories) : Ce sont des interfaces qui facilitent les interactions avec la base de données. Spring Data JPA, par exemple, permet de définir des repositories sans implémenter des méthodes complexes de gestion des données. Les repositories offrent un accès facile aux opérations CRUD (création, lecture, mise à jour, suppression).

**Rôle :**

* Framework Java principal pour le développement du backend, utilisé pour gérer :
* La logique métier (gestion des produits, utilisateurs, commandes, etc.).
* Les API REST.
* La sécurité via Spring Security (authentification, gestion des rôles).

**Avantages :**

* Configuration rapide grâce à l’approche « convention over configuration ».
* Intégration fluide avec Spring Data JPA pour la gestion des bases de données.

#### Thymeleaf

Le frontend de l'application utilise Thymeleaf, un moteur de templates Java qui s'intègre parfaitement avec Spring Boot pour générer des pages dynamiques basées sur les données fournies par le backend. Thymeleaf permet de rendre des pages HTML dynamiques, en injectant des données directement dans les vues à l'aide de expressions Thymeleaf.

**Thymeleaf : Présentation et Fonctionnalités :**

Thymeleaf est conçu pour fonctionner avec des applications web basées sur Java, et il est très souvent utilisé avec Spring Boot. Ce moteur de templates permet de séparer la logique métier du rendu des pages, offrant ainsi une architecture claire et maintenable. Thymeleaf se distingue par son approche HTML valide, ce qui signifie que les templates peuvent être visualisés dans un navigateur même sans être exécutés par un serveur Spring Boot.

Les templates Thymeleaf sont construits autour de balises HTML classiques auxquelles sont ajoutées des attributs spécifiques pour lier dynamiquement les données à partir du backend.

**Balises Thymeleaf et Leur Fonctionnalité :**

Thymeleaf dispose de nombreuses balises et attributs permettant de manipuler le contenu d’une page HTML, de faire des boucles, d’ajouter des conditions, ou encore de lier des données au frontend.

Voici un aperçu des principales balises et attributs Thymeleaf avec leurs fonctions :

|  |  |
| --- | --- |
| Balise/Attribut | Description |
| th:text | Afficher du texte dynamique |
| th:utext | Afficher du texte HTML dynamique |
| th:each | Itérer sur une collection d'objets |
| th:if | Conditionner l’affichage d’un élément |
| th:unless | Afficher un élément si la condition est fausse |
| th:attr | Modifier plusieurs attributs d’un élément HTML |
| th:href | Définir dynamiquement l'URL d’un lien |
| th:src | Définir dynamiquement la source d’une ressource |
| th:class | Appliquer dynamiquement des classes CSS |
| th:style | Appliquer dynamiquement des styles CSS |
| th:object | Lier un objet au formulaire HTML |
| th:field | Lier un champ de formulaire à une propriété |
| th:value | Définir la valeur d’un champ de formulaire |
| @{} | Résoudre dynamiquement des URL dans le template |
| th:include | Inclure un fragment d'un autre template |
| th:replace | Remplacer un élément HTML par un fragment d’un autre template |
| th:switch et th:case | Conditionner l’affichage avec un switch |

**Avantages :**

* Syntaxe intuitive permettant d’itérer sur des listes (th:each), d’ajouter des conditions (th:if), et d’afficher des données dynamiquement.
* Intégration native avec Spring Boot, simplifiant la gestion des vues et réduisant la complexité des appels API.

**Exemple d’utilisation :**

* Une page listant les produits utilise th:each pour afficher chaque produit et th:if pour afficher des badges conditionnels (ex. : "Promotion").

#### Bootstrap

**Rôle :**

* Framework CSS utilisé pour concevoir une interface moderne, responsive et cohérente.

**Fonctionnalités clés utilisées :**

* Grille Responsive : Organisation des éléments de la page en colonnes et lignes pour une compatibilité multi-produits.
* Composants Préconçus : Utilisation de boutons, modales, carrousels et autres éléments pour enrichir l’expérience utilisateur.
* Classes CSS Utilitaires : Classes pour gérer rapidement les marges, alignements, couleurs et typographies.

**Exemple d’utilisation :**

* Formulaires utilisateurs : Créés avec des composants Bootstrap (champs de saisie, boutons) pour une apparence professionnelle et une validation visuelle immédiate.
* Liste des produits : Grilles stylisées affichant les produits avec des boutons d’action (ajout au panier, consultation des détails).
* Barre de navigation : Responsive et adaptée à la gestion des rôles (ex. : éléments administrateurs uniquement visibles pour les utilisateurs avec le rôle approprié).

#### PostgreSQL

**Rôle :**

* Base de données relationnelle utilisée pour stocker et gérer les entités clés du projet :
* Utilisateurs, Produits, Commandes, Catégories, et Promotions.

**Avantages :**

* Support des relations complexes entre les entités grâce à des clés étrangères.
* Performances élevées pour les requêtes multi-table et les agrégations de données.

**Structure Exemple :**

* Table produit : Contient les informations principales des produits (nom, prix, stock).
* Table commande : Reliée à utilisateur et produit via des relations one-to-many et many-to-many.

J'ai opté pour PostgreSQL pour plusieurs raisons, principalement sa performance, sa robustesse, et sa compatibilité avec Spring Boot.

**Pourquoi choisir PostgreSQL ?**

PostgreSQL est un système de gestion de base de données relationnelle open-source, qui est très performant et largement utilisé pour des applications modernes. Il est particulièrement adapté quand il s'agit de travailler avec de grandes quantités de données, car il peut gérer des transactions complexes tout en garantissant la sécurité et l'intégrité des données. Voici quelques raisons pour lesquelles j'ai choisi PostgreSQL :

* Performances élevées : Même avec de nombreuses données, PostgreSQL reste très rapide et réactif.
* Conformité ACID : Il assure l’intégrité des données à travers les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité). Cela garantit que les données sont toujours cohérentes, même en cas de défaillance.
* Extensibilité : PostgreSQL permet d'ajouter des fonctionnalités personnalisées grâce à des types de données, des fonctions et des extensions.
* Sécurité avancée : La gestion des droits d'accès est robuste, ce qui est important pour protéger les données sensibles.
* Facilité d'intégration avec Spring Boot : Grâce à des bibliothèques comme Spring Data JPA, l'intégration entre Spring Boot et PostgreSQL est fluide et très simple à configurer.

**Comment PostgreSQL fonctionne avec Spring Boot ?**

Dans l’ application, Spring Boot gère l’accès aux données via Spring Data JPA. Cela permet de travailler avec des entités Java qui correspondent aux tables de la base de données PostgreSQL. En d'autres termes, je peux manipuler les données comme des objets Java sans avoir à écrire des requêtes SQL complexes. C’est très pratique et cela permet de gagner beaucoup de temps.

#### PgAdmin

**Rôle :**

* Outil d’administration pour PostgreSQL.
* Permet de gérer les schémas, exécuter des requêtes SQL, et visualiser les données stockées.

**Avantages :**

* Interface graphique conviviale pour effectuer des tests sur les données pendant le développement.
* Outil permettant d’identifier les erreurs de structure ou de relations dans la base.

**Choix stratégique :**

* Le choix d’adopter une architecture intégrée avec Thymeleaf et de conserver un backend robuste avec Spring Boot et PostgreSQL a permis de :
* Respecter les délais du projet : Réduction du temps nécessaire pour développer des interfaces dynamiques.
* Simplifier la maintenance : Moins de couches à synchroniser (backend et frontend intégrés).
* Assurer la qualité technique : Utilisation de technologies éprouvées comme Spring Boot et PostgreSQL.
* Génération de pages dynamiques tout en conservant la sécurité et la modularité du code.

Pour l'administration de la base de données, j'utilise PgAdmin, un outil graphique très pratique qui me permet de gérer la base de données de manière intuitive, sans avoir à écrire constamment des commandes SQL en ligne. Cela facilite vraiment la gestion des schémas, des tables et des requêtes dans le projet.

## Modèle Conceptuel des Données (M.C.D)

## Modèle Logique des Données (M.L.D)

## Analyse de la Base de Données

### Structure de la Base de Données

#### Thésaurus

|  |  |  |
| --- | --- | --- |
| **Nom de la table** | **Description** | **Rôle dans le flux** |
| Table « user\_dtls » | Contient les informations des utilisateurs. | Point d'origine pour la gestion des comptes clients et administrateurs. |
| Table « category » | Permet de regrouper les produits sous des classifications logiques. | Facilite la recherche et le filtrage des produits. |
| Table « product » | Contient tous les détails des produits disponibles à la vente | Source de toutes les informations sur les produits affichés et ajoutés aux paniers. |
| Table « cart » | Gère le panier temporaire de l'utilisateur. | Point intermédiaire entre la navigation des produits et la création des commandes. |
| Table «  product\_order » | Stocke les commandes finalisées, avec leurs détails. | Enregistre toutes les informations de paiement, produits, et statut de la commande. |
| Table «  order\_address » | Contient les informations de livraison pour chaque commande. | Associée à une commande unique pour la livraison. |

### Relations entre les Tables

Utilisateur -> Panier - Relation : 1:N.

* Description : Chaque utilisateur peut avoir plusieurs paniers (lignes dans cart).

Produit -> Panier - Relation : 1:N.

* Description : Chaque produit peut être ajouté à plusieurs paniers.

Utilisateur -> Commande - Relation : 1:N.

* Description : Chaque utilisateur peut créer plusieurs commandes.

Produit -> Commande - Relation : 1:N.

* Description : Un produit peut être présent dans plusieurs commandes.

Commande -> Adresse - Relation : 1:1.

* Description : Une commande est liée à une adresse unique.

### Flux des Données et Scénarios d'Utilisation

#### Inscription/Connexion

* Les utilisateurs s'inscrivent en fournissant leurs informations de base (nom, e-mail, mot de passe).
* Les mots de passe sont hashés et stockés dans la table user\_dtls.
* Navigation et Sélection de Produits
* L'utilisateur navigue dans les produits filtrés par catégorie (table category).
* Les détails des produits (titre, description, prix, stock) proviennent de la table product.

#### Ajout au Panier

* L'utilisateur ajoute un produit à son panier (table cart) en spécifiant la quantité souhaitée.
* Chaque ligne du panier est liée à un produit (étrangère product\_id) et à l'utilisateur (étrangère user\_id).
* Validation du Panier et Création de la Commande
* Une fois les produits sélectionnés, l'utilisateur passe à la validation.

#### Création d’une commande

* Une commande est créée (table product\_order) contenant les informations suivantes :
* Liste des produits achetés.
* Quantités et prix total.
* Mode de paiement et statut.
* Ajout d'une Adresse
* Une adresse est ajoutée à la commande via la table order\_address.
* Finalisation et Paiement
* La commande est marquée comme "en cours", "livrée" ou "annulée" en fonction du statut dans product\_order.

# Back-end

## Package config

Les classes dans ce dossier sont responsables de la configuration de la sécurité et de la personnalisation des comportements liés à l'authentification. Cela inclut :

* AuthFailureHandlerImpl et AuthSuccessHandlerImpl : Gèrent les scénarios d'échec et de succès d'authentification.
* CustomUser : Représente un utilisateur personnalisé pour Spring Security.
* SecurityConfig : Fournit la configuration centrale de Spring Security.

## Package controller

**Les contrôleurs sont segmentés par rôle ou fonctionnalité :**

* AdminController : Probablement pour les actions spécifiques à l'administrateur.
* HomeController : Gère l'affichage des pages principales.
* UserController : Responsable des actions utilisateur (inscription, connexion, etc.).

**Séparation des responsabilités :**

* Chaque contrôleur cible un domaine fonctionnel précis.

**Gestion robuste des messages utilisateur :**

* Utilisation des sessions HTTP pour les retours utilisateur (succès ou erreurs).

**Pagination et recherche :**

* Implémentées dans les listes (produits, commandes, utilisateurs).

**Flux d'authentification sécurisé :**

* Validation des utilisateurs via Principal.

### Fichier : HomeController.java

* Objectif : Gérer les pages publiques (connexion, inscription, page d'accueil) et les fonctionnalités accessibles sans connexion utilisateur spécifique.

**Fonctionnalités principales :**

* *@ModelAttribute* - Préchargement des données utilisateurs : Charge les détails de l'utilisateur connecté (Principal) et les catégories actives dans toutes les vues.
* Exemple : user (Détails de l'utilisateur) et categorys (Liste des catégories actives).
* Page d'accueil (index) : Récupère les catégories et produits actifs, triés par ID décroissant. Limite l'affichage à 12 produits récents.
* Page de connexion et d'inscription : Méthodes simples pour charger les vues login et register.
* Produits avec pagination et filtres (/products) : Affiche des produits filtrés par catégorie ou mot-clé et pagination intégrée (numéro de page, taille de page, total des éléments).
* Détails d'un produit (/product/{id}) : Récupère un produit par son ID et affiche ses détails dans une vue.
* Enregistrement d'utilisateur (/saveUser) : Sauvegarde les informations d'un utilisateur et gère l'upload de photo de profil. Si aucune photo n'est uploadée, une image par défaut est utilisée.
* Réinitialisation de mot de passe (/forgot-password) : Génère un lien de réinitialisation avec un token unique envoyé par email. Le token est validé pour autoriser le changement de mot de passe.
* Recherche de produits (/search) : Recherche des produits correspondant à un mot-clé et affiche les résultats.

### Fichier : UserController.

* Objectif : Gérer les interactions utilisateur après connexion, comme le panier, les commandes, et le profil.

**Fonctionnalités principales :**

* Page d'accueil utilisateur (/user/) : Charge la vue user/home pour afficher un tableau de bord utilisateur.
* Ajout au panier (/user/addCart) : Ajoute un produit au panier d'un utilisateur et gère les messages de succès ou d'échec via la session HTTP.
* Page du panier (/user/cart) : Affiche les articles du panier et calcule le prix total.
* Mise à jour de la quantité dans le panier (/user/cartQuantityUpdate) : Met à jour la quantité d'un produit dans le panier.

**Gestion des commandes :**

* Page de commande (/user/orders) : Affiche les articles du panier prêts pour la commande.
* Sauvegarde d'une commande (/user/save-order) : Crée une commande à partir des informations du panier et des détails utilisateur.
* Page des commandes utilisateur (/user/user-orders) : Liste toutes les commandes passées par l'utilisateur.
* Mise à jour du statut d'une commande (/user/update-status) : Change le statut d'une commande (exemple : livrée, annulée).

**Profil utilisateur :**

* Afficher et mettre à jour le profil (/user/profile) : Met à jour les informations utilisateur et la photo de profil.
* Changer le mot de passe (/user/change-password) : Valide le mot de passe actuel et met à jour avec un nouveau mot de passe.

### Fichier : AdminController.java

* Objectif : Gérer les fonctionnalités réservées à l'administrateur, comme la gestion des catégories, des produits, des utilisateurs, et des commandes.

**Fonctionnalités principales :**

* Page d'accueil administrateur (/admin/) : Charge le tableau de bord admin (admin/index).

**Gestion des catégories :**

* Ajout de catégories (/admin/loadAddProduct, /admin/saveCategory) : Gère les catégories avec possibilité d'upload d'images.
* Modification de catégories (/admin/loadEditCategory/{id}) : Charge les détails d'une catégorie pour édition.
* Suppression de catégories (/admin/deleteCategory/{id}) : Supprime une catégorie par son ID.

**Gestion des produits :**

* Ajout et modification de produits (/admin/saveProduct, /admin/updateProduct) : Permet l'édition et l'upload d'images associées aux produits.
* Suppression de produits (/admin/deleteProduct/{id}) : Supprime un produit par son ID.
* Recherche de produits avec pagination (/admin/products) : Affiche les produits en fonction de critères de recherche.

**Gestion des utilisateurs :**

* Liste des utilisateurs par rôle (/admin/users) : Récupère et affiche les utilisateurs ou administrateurs.
* Mise à jour du statut utilisateur (/admin/updateSts) : Active ou désactive un compte utilisateur.

**Gestion des commandes :**

* Liste paginée des commandes (/admin/orders) : Affiche toutes les commandes avec pagination.
* Mise à jour du statut d'une commande (/admin/update-order-status) : Change le statut d'une commande (exemple : en traitement, expédiée).
* Recherche de commandes (/admin/search-order) : Trouve une commande par son ID.

## Package model

* Structure : Chaque classe correspond à une table ou une logique métier précise.
* Les annotations Lombok : Simplifient le code en générant automatiquement les méthodes standard.
* Relations JPA : Modélisent correctement les relations entre les entités.

### Fichier : Cart.java

* Description : Cette classe représente un panier d'achat associé à un utilisateur. Elle permet de suivre les produits ajoutés au panier.

**Points clés :**

* Annotations JPA :

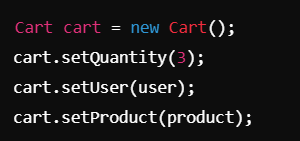
*@Entity* : Représente une table en base de données.

*@Id* : Identifie la clé primaire.

*@GeneratedValue*(strategy = GenerationType.IDENTITY) : Auto-incrémente la clé primaire.

*@ManyToOne* : Définit des relations avec UserDtls (un utilisateur) et Product (un produit).

*@Transient* : Les champs totalPrice et totalOrderPrice ne sont pas persistés.

**Exemple d'utilisation** :

### Fichier : Category.java

* Description : Cette classe représente une catégorie de produit.

**Points clés :**

* Annotations JPA :

*@Id* : Identifie la clé primaire.

*@GeneratedValue*(strategy = GenerationType.IDENTITY) : Auto-incrémente la clé primaire.

### Fichier : OrderAddress.java

* Description :Cette classe modélise une adresse associée à une commande.

**Points clés :**

* Annotations JPA :

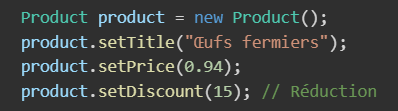
*@Data* : Génère les getters, setters, equals, hashCode, et toString.

### Fichier : OrderRequest.java

* Description : Classe qui encapsule les données nécessaires pour créer une commande.

### Fichier : Product.java

* Description : Cette classe représente un produit dans l'application shop connect.

**Exemple d'utilisation :**

### Fichier : ProductOrder.java

* Description : Cette classe représente une commande d'un produit par un utilisateur.

**Points clés :**

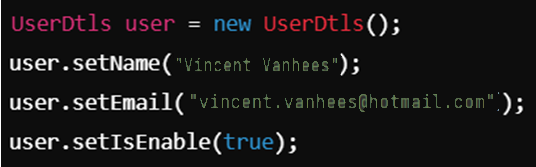
* Relations JPA :

*@ManyToOne* : Relation avec Product et UserDtls.

*@OneToOne* : Relation avec OrderAddress.

### Fichier : UserDtls.java

* Description : Cette classe modélise un utilisateur dans le système.

**Exemple d'utilisation :**

## Package Repository

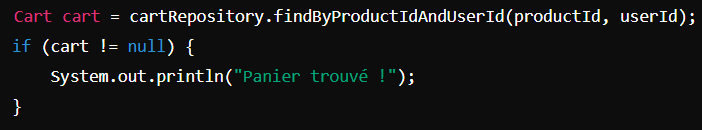
* Les méthodes y sont nommées selon les conventions Spring Data JPA.
* Une séparation claire des responsabilités facilite la maintenance et l’extension du projet.
* Intégration des fonctionnalités de recherches, pagination et filtrage.

### Fichier : CartRepository.java

Le CartRepository gère les opérations liées au panier d'achat.

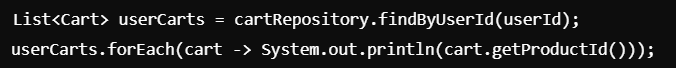
**Méthodes principales :**

* findByProductIdAndUserId : Recherche un panier spécifique pour un produit donné et un utilisateur donné.

**Exemple d'utilisation :**

* countByUserId : Compte le nombre d'articles dans le panier d'un utilisateur.
* Utilité : Afficher le nombre total d'articles dans le panier.
* findByUserId : Récupère tous les paniers associés à un utilisateur.

**Exemple :**



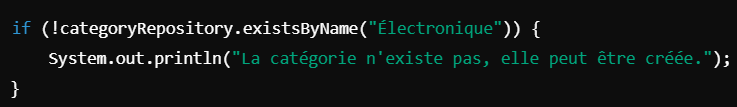
### Fichier : CategoryRepository.java

Le CategoryRepository gère les catégories de produits.

**Méthodes principales :**

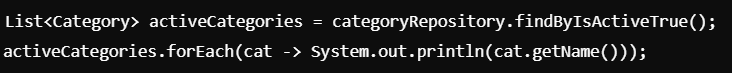
* existsByName : Vérifie si une catégorie existe en fonction de son nom.
* Utilité : Empêcher les doublons lors de la création d'une catégorie.

**Exemple :**



* findByIsActiveTrue : Récupère toutes les catégories actives.

**Exemple :**

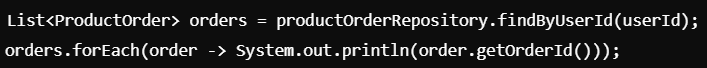


### Fichier : ProductOrderRepository.java

Le ProductOrderRepository gère les commandes utilisateurs.

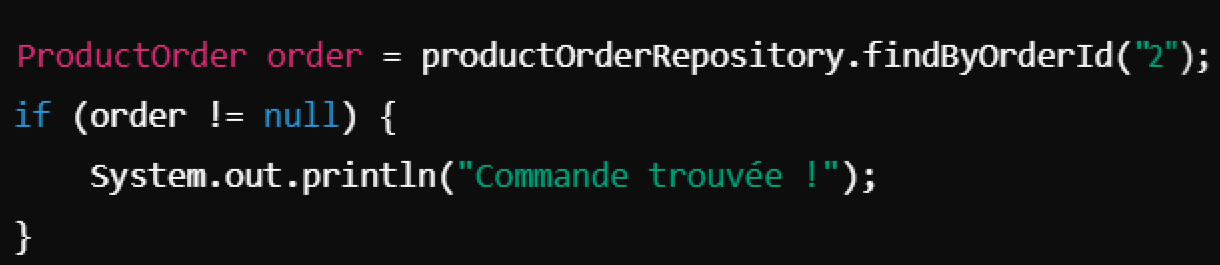
**Méthodes principales :**

* findByUserId : Récupère toutes les commandes d'un utilisateur.

**Exemple :** 

* findByOrderId : Trouve une commande par son identifiant unique.

**Exemple :**



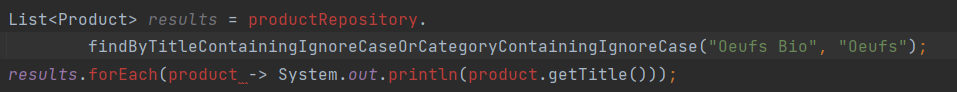
### Fichier : ProductRepository.java

Le ProductRepository gère les produits.

**Méthodes principales :**

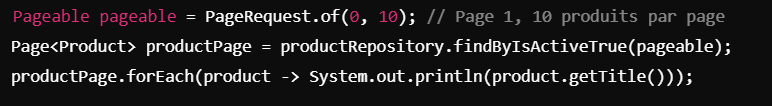
* findByIsActiveTrue : Récupère tous les produits actifs,
* findByTitleContainingIgnoreCaseOrCategoryContainingIgnoreCase : recherche des produits par mot-clé dans leur titre ou leur catégorie.

**Exemple :**



Fournit une liste paginée de produits actifs.

**Exemple :**



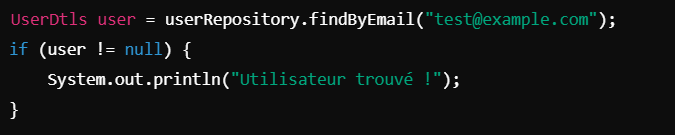
* Recherche avancée avec filtre d'activation : Recherche des produits actifs en filtrant par titre ou catégorie avec pagination.

### Fichier : UserRepository.java

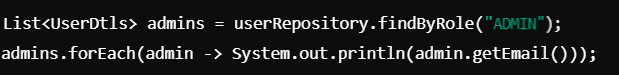
Le UserRepository gère les utilisateurs.

**Méthodes principales :**

* findByEmail : Trouve un utilisateur par son email.

**Exemple :**

* findByRole : Récupère tous les utilisateurs ayant un rôle spécifique.

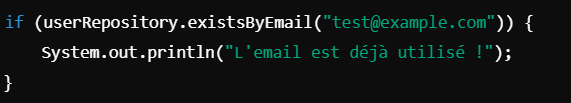
**Exemple** : 

* findByResetToken : Trouve un utilisateur via un token de réinitialisation.

**Utilité :** Réinitialisation de mot de passe.

* existsByEmail : Vérifie si un utilisateur existe par son email.

**Exemple :**



## Package service/impl (fichiers des services d’implémentation (impl))

* Interfaces (service) : Définissent les comportements.
* Implémentations (service/impl) : Gèrent la logique métier.

La séparation entre les interfaces et leurs implémentations est une pratique pour respecter le principe d’inversion de dépendances. Chaque fichier correspond à une implémentation concrète des interfaces définies dans les services.

L'ajout de impl dans le nom indique qu'il s'agit de l'implémentation de l'interface, ce qui est une convention pour les développeurs Java.

### Rôle de impl :

Les classes impl fournissent les implémentations concrètes des interfaces des services. Cela sépare clairement la logique métier (dans impl) des définitions (dans les interfaces).

### Fichier : CartServiceImpl.java

**Fonctionnalités principales :**

* Ajout de produit au panier (saveCart) : Vérifie si le produit existe déjà dans le panier. Si oui, incrémente la quantité et met à jour le prix total. Sinon, crée un nouveau panier pour l’utilisateur et le produit.
* Récupération des produits du panier (getCartsByUser) : Retourne tous les produits dans le panier d’un utilisateur. Calcule le prix total pour chaque produit et la commande complète.
* Comptage des articles dans le panier (getCountCart) : Retourne le nombre total d'articles dans le panier d’un utilisateur.
* Mise à jour de la quantité (updateQuantity) : Permet d’augmenter ou diminuer la quantité d’un produit dans le panier. Supprime le produit si la quantité atteint 0.

### Fichier : CategoryServiceImpl.java

**Fonctionnalités principales :**

* Sauvegarde d'une catégorie (saveCategory) : Ajoute ou met à jour une catégorie dans la base de données.

**Récupération des catégories :**

* getAllCategory : Toutes les catégories.
* getAllActiveCategory : Catégories actives uniquement.
* getAllCategorPagination : Catégories avec pagination.
* Existence d'une catégorie (existCategory) : Vérifie si une catégorie existe par son nom pour éviter les doublons.
* Suppression et récupération par ID : Permet de supprimer ou récupérer une catégorie par son identifiant.

### Fichier : CommnServiceImpl.java

**Fonctionnalité principale :**

* Suppression des messages de session (removeSessionMessage) : Supprime les messages de succès ou d’erreur stockés dans la session utilisateur.

### Fichier : OrderServiceImpl.java

**Fonctionnalités principales :**

* Création de commandes (saveOrder) : Génère des commandes à partir des articles dans le panier. Enregistre chaque commande avec un ID unique et une adresse associée. Envoie un email de confirmation.
* Mise à jour du statut de commande (updateOrderStatus) : Modifie le statut d’une commande (ex. : en cours, livré, annulé).

**Récupération des commandes :**

* getOrdersByUser : Commandes d’un utilisateur spécifique.
* getAllOrdersPagination : Commandes avec pagination.
* getOrdersByOrderId : Commande par ID unique.

### Fichier : ProductServiceImpl.java

**Fonctionnalités principales :**

* Gestion des produits (saveProduct, updateProduct) : Sauvegarde un produit ou met à jour ses informations. Calcule et met à jour le prix après remise.
* Gestion des images de produits : Enregistre les images dans un répertoire dédié. Met à jour les informations de l'image si elle est fournie.
* Recherche et récupération : Recherche par mot-clé, catégorie ou pagination. Filtre les produits actifs.
* Suppression d'un produit (deleteProduct) : Supprime un produit de la base de données.

### Fichier : UserDetailsServiceImpl.java

**Fonctionnalité principale :**

* Chargement des utilisateurs pour Spring Security (loadUserByUsername) : Recherche un utilisateur par son email. Retourne un objet CustomUser contenant les informations nécessaires pour l’authentification et l’autorisation.

### Fichier : UserServiceImpl.java

**Fonctionnalités principales :**

* Gestion des utilisateurs (saveUser, saveAdmin) : Enregistre un utilisateur ou un administrateur avec des rôles par défaut. Encode les mots de passe avec un encodeur BCrypt.
* Gestion du profil : Permet de mettre à jour les informations utilisateur et d’uploader une photo de profil.

**Vérification et gestion des comptes** :

* updateAccountStatus : Active ou désactive un compte.
* increaseFailedAttempt : Incrémente les tentatives de connexion échouées.
* userAccountLock : Verrouille le compte après plusieurs échecs.
* Réinitialisation de mot de passe : Génère et assigne un token pour réinitialiser le mot de passe. Permet de mettre à jour le mot de passe en utilisant ce token.
* Déverrouillage des comptes (unlockAccountTimeExpired) : Vérifie si la durée de verrouillage est expirée et réactive le compte.

## Package service

Ces fichiers définissent les interfaces des services et servent de contrats pour leurs implémentations dans les classes impl. Cela permet de maintenir une séparation claire entre les définitions et les logiques métier. Les interfaces définissent les fonctionnalités et les contraintes de chaque service.

Cela permet :

* Une séparation nette entre la définition (services) et l’implémentation (services/impl).
* Une meilleure testabilité, car les services peuvent être simulés ou remplacés.

Chaque service gère une fonctionnalité spécifique : panier, catégories, commandes, produits ou utilisateurs.

Les dépendances sont injectées grâce à @Autowired, facilitant les tests et la maintenance.

### Fichier : CartService.java

Objectif : Gérer les fonctionnalités liées au panier.

**Méthodes principales :**

* saveCart : Ajoute un produit au panier d’un utilisateur.
* Paramètres : productId, userId.
* Retour : L'objet Cart sauvegardé.
* getCartsByUser : Récupère tous les articles dans le panier d’un utilisateur.
* Retour : Liste d'objets Cart.
* getCountCart : Compte le nombre total d'articles dans le panier.
* Paramètre : userId.
* Retour : Nombre total.
* updateQuantity : Met à jour la quantité d’un produit dans le panier.
* Paramètres : Action ("incrémenter" ou "décrémenter"), ID du produit dans le panier.

### Fichier : CategoryService.java

Objectif : Gérer les catégories de produits.

**Méthodes principales :**

* saveCategory : Sauvegarde ou met à jour une catégorie.
* Paramètre : L'objet Category.
* Retour : La catégorie sauvegardée.
* existCategory : Vérifie si une catégorie existe par son nom.
* Retour : Booléen.

**Récupération :**

* getAllCategory : Toutes les catégories.
* getAllActiveCategory : Catégories actives uniquement.
* getAllCategorPagination : Catégories avec pagination.

**Suppression et recherche** :

* deleteCategory : Supprime une catégorie par son ID.
* getCategoryById : Récupère une catégorie par son ID.

### Fichier : CommonService.java

Objectif : Fournir des fonctionnalités utilitaires communes.

**Méthode principale :**

* removeSessionMessage : Supprime les messages de session (succès/erreur). Utilisé pour nettoyer les sessions utilisateur après des actions.

### Fichier : OrderService.java

Objectif : Gérer les commandes des utilisateurs.

**Méthodes principales :**

* Création de commande (saveOrder) : Enregistre des commandes à partir des articles dans le panier.
* Paramètres : userId, orderRequest.

**Récupération des commandes :**

* getOrdersByUser : Commandes d’un utilisateur.
* getAllOrders : Toutes les commandes.
* getAllOrdersPagination : Commandes avec pagination.

**Statut et recherche :**

* updateOrderStatus : Change le statut d’une commande.
* getOrdersByOrderId : Recherche une commande par son ID unique.

### Fichier : ProductService.java

Objectif : Gérer les produits de l’application.

**Méthodes principales :**

* Gestion des produits :
* saveProduct : Sauvegarde un produit.
* updateProduct : Met à jour un produit et son image.
* deleteProduct : Supprime un produit par son ID.

**Récupération et recherche :**

* getAllProducts : Tous les produits.
* getAllActiveProducts : Produits actifs par catégorie.
* searchProduct : Recherche par mot-clé.
* Méthodes avec pagination pour les produits actifs ou toutes les catégories.

### Fichier : UserService.java

Objectif : Gérer les utilisateurs et leurs actions liées.

**Méthodes principales :**

* Gestion utilisateur :
* saveUser et saveAdmin : Sauvegarde des utilisateurs avec rôles par défaut.
* updateUserProfile : Met à jour le profil et l’image.

**Gestion des comptes :**

* updateAccountStatus : Change le statut actif/inactif d’un compte.
* increaseFailedAttempt : Augmente le compteur d’échecs de connexion.
* userAccountLock : Verrouille un compte après plusieurs échecs.
* unlockAccountTimeExpired : Déverrouille un compte après expiration.

**Réinitialisation de mot de passe :**

* Génère un token (updateUserResetToken) pour réinitialiser le mot de passe.
* Permet de mettre à jour le mot de passe avec le token.

**Recherche :**

* getUserByEmail : Recherche par email.
* existsEmail : Vérifie si un email est déjà utilisé.

## Package util

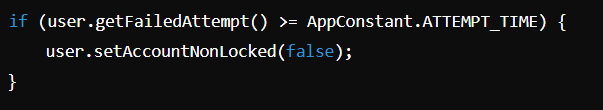
Le package util contient des classes qui centralisent des fonctionnalités partagées, des constantes et des énumérations, afin de réduire la duplication de code et d'améliorer la maintenabilité de l'application. Ces classes jouent un rôle essentiel pour uniformiser la gestion des constantes, des utilitaires et des statuts.

### AppConstant

La classe AppConstant est dédiée à la gestion des constantes globales utilisées dans l'application. Elle permet de centraliser les paramètres configurables et d'assurer une cohérence à travers les différents modules.

**Exemple de constantes :**

* UNLOCK\_DURATION\_TIME : Durée de déblocage après plusieurs tentatives infructueuses (par défaut : 3 secondes).
* ATTEMPT\_TIME : Nombre maximal de tentatives d'authentification (par défaut : 3).

**Exemple d'utilisation :**

### Fichier : CommonUtil.java

La classe CommonUtil regroupe des fonctions utilitaires réutilisables, principalement liées aux fonctionnalités de messagerie et aux interactions utilisateur.

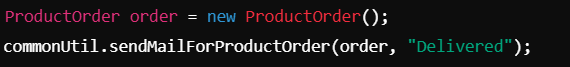
**Méthodes principales :**

* sendMail : Envoie un email de réinitialisation de mot de passe.

**Paramètres :**

* url : Lien pour réinitialiser le mot de passe.
* recipientEmail : Email du destinataire.
* Retourne : true si l'email est envoyé avec succès.
* generateUrl : Génère une URL complète basée sur la requête actuelle. Utilisé pour inclure l'hôte et le port dans des liens.
* sendMailForProductOrder : Envoie un email contenant les détails de commande. Personnalise le contenu en remplaçant les balises dynamiques ([[name]], [[orderStatus]], etc.).
* getLoggedInUserDetails : Récupère les informations de l'utilisateur actuellement connecté via l'objet Principal.

**Exemple d'utilisation :** Envoyer un email après une commande

.

### Fichier : OrderStatus.java

Objectif : Représenter les différents statuts possibles pour une commande dans l'application.

**Champs :**

* id : Identifiant numérique du statut.
* name : Nom textuel du statut (par exemple, "In Progress").

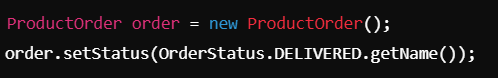
**Valeurs définies :**

* IN\_PROGRESS : Commande en cours de traitement.
* ORDER\_RECEIVED : Commande reçue.
* PRODUCT\_PACKED : Produit emballé.
* OUT\_FOR\_DELIVERY : Produit en cours de livraison.
* DELIVERED : Produit livré.
* CANCEL : Commande annulée.
* SUCCESS : Commande réussie.

**Méthodes :**

* getId et getName : Récupèrent respectivement l'identifiant et le nom du statut.
* setId et setName : Permettent de modifier ces valeurs.

**Exemple d'utilisation :** Changer le statut d'une commande.



# Front End

## Vues par défaut

### Base Template (base.html)​

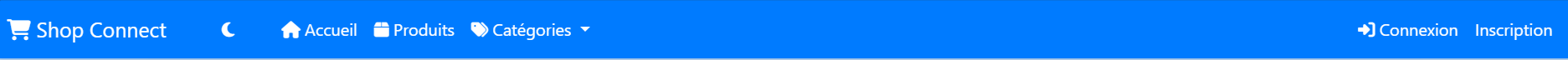
Objectif : Ce fichier est le layout principal utilisé comme base pour toutes les autres pages grâce à la directive Thymeleaf th:replace.

**Points clés :**

* Barre de navigation : Personnalisée avec des liens pour l'accueil, les produits, les catégories, et les actions utilisateur (connexion, inscription, déconnexion).

**Le menu s'adapte dynamiquement :**

* Si user == null, affiche les liens pour Connexion et Inscription.
* Si un utilisateur est connecté, affiche son nom, son rôle, et des actions comme Profil et Déconnexion.
* Contenu dynamique (<div th:replace="${content}"></div>) : Permet d'injecter du contenu spécifique provenant des autres vues.
* Thème sombre/clair : Fonctionnalité pour basculer entre les deux thèmes via JavaScript et des variables CSS.



**Footer :**

Contient des liens vers les réseaux sociaux avec des icônes FontAwesome.



### Page de mot de passe oublié (forgot\_password.html)​

Objectif : Permet aux utilisateurs de demander un lien de réinitialisation de mot de passe en entrant leur email.

**Points clés :**

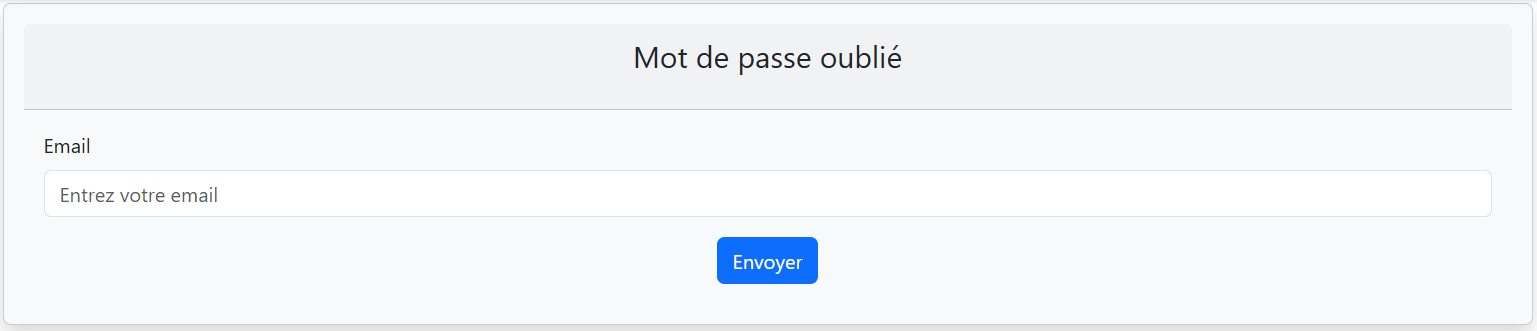
* Formulaire de réinitialisation : Champ email requis avec validation HTML5. Envoie une requête POST à /forgot-password.

**Messages dynamiques :**

* Affiche un message de succès ou d’erreur (${session.succMsg} ou ${session.errorMsg}).

**Thymeleaf :**

* Utilisation de th:block pour conditionnellement afficher les messages.



### Page d'accueil (index.html)​

Objectif : Page principale affichant les catégories disponibles et les derniers produits.

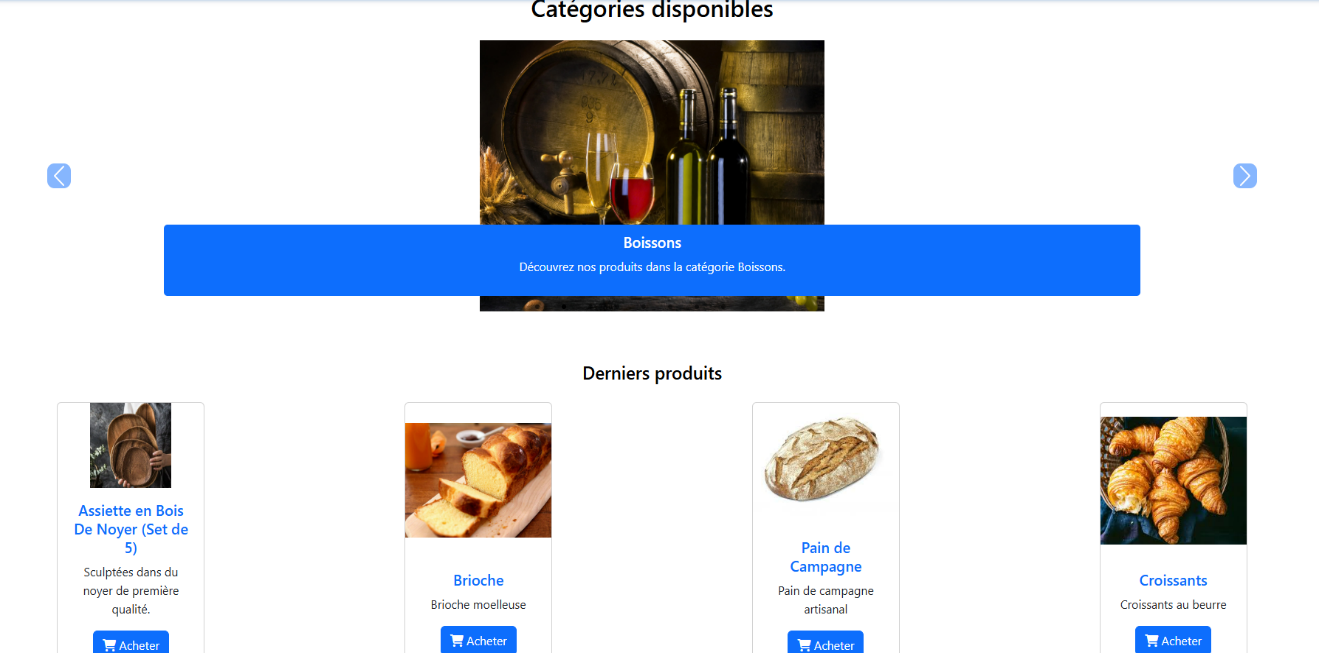
**Points clés :**

* Carrousel des catégories :
* Dynamique grâce à Thymeleaf :
* th:each pour itérer sur les catégories.
* Liens cliquables pour rediriger vers les produits de chaque catégorie.

**Section des derniers produits** :

* Affiche une grille de produits, chaque carte contenant :
* Une image.
* Un lien vers la page des détails du produit.
* Un bouton Acheter.

**Bootstrap :**

* Mise en page responsive avec des colonnes et des composants prêts à l’emploi.

### Page de connexion (login.html)​

Objectif : Fournit une interface pour que les utilisateurs puissent se connecter à l'application.

**Points clés :**

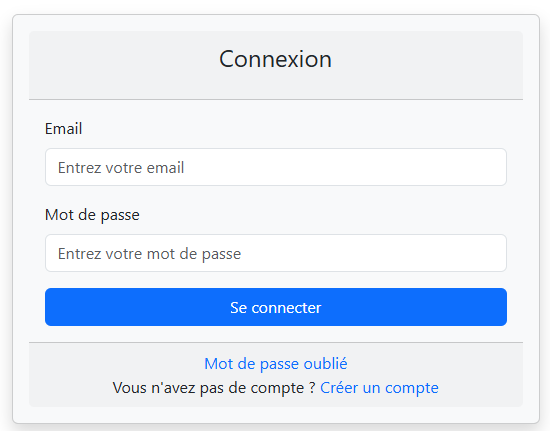
* Formulaire de connexion :
* Champs pour l'email (username) et le mot de passe (password).
* Soumission via POST à /login.

**Messages d'état :**

* th:if="${param.error} : Affiche une alerte en cas d'échec.
* th:if="${param.logout} : Affiche une alerte de déconnexion réussie.

**Liens supplémentaires :**

* Redirection vers Mot de passe oublié et Créer un compte.

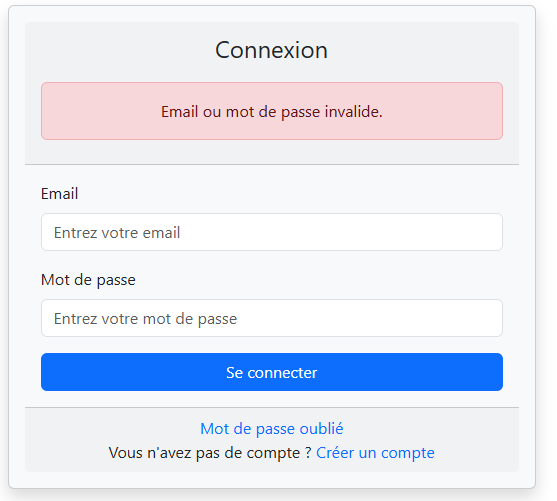


### Page des messages (message.html)​

Objectif : Affiche un message générique transmis depuis le backend.

**Points clés :**

* Utilise [[${msg}]] pour afficher dynamiquement un message.
* Mise en page centrée, adaptée pour des notifications temporaires.



### Page des produits (product.html)​

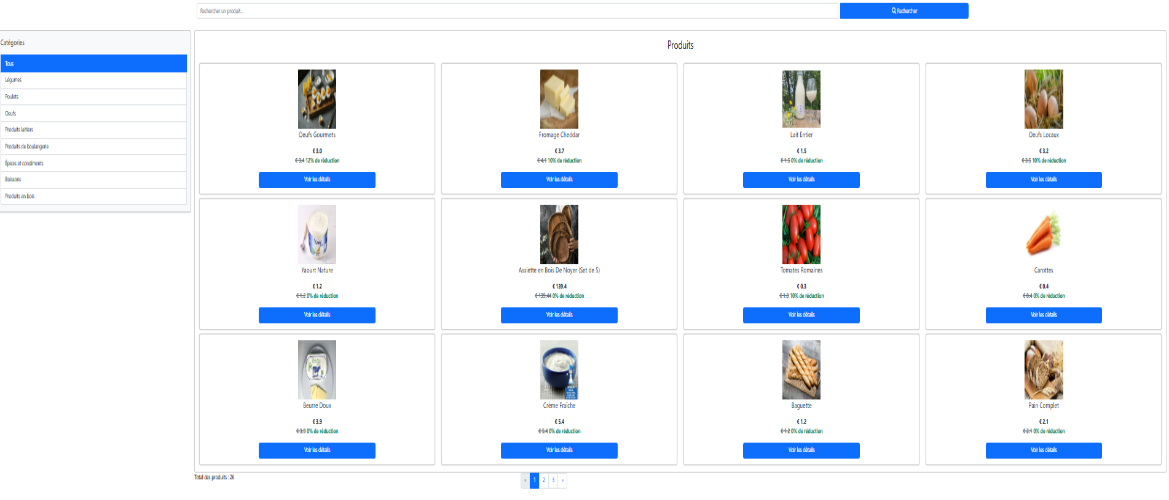
Objectif : Affiche une liste de produits avec options de recherche, filtrage par catégorie, et pagination.

**Points clés :**

* Barre de recherche : Soumet une requête GET à /products avec un mot-clé (ch).
* Filtrage par catégorie : Liste des catégories dynamiques avec th:each.

**Liste des produits :**

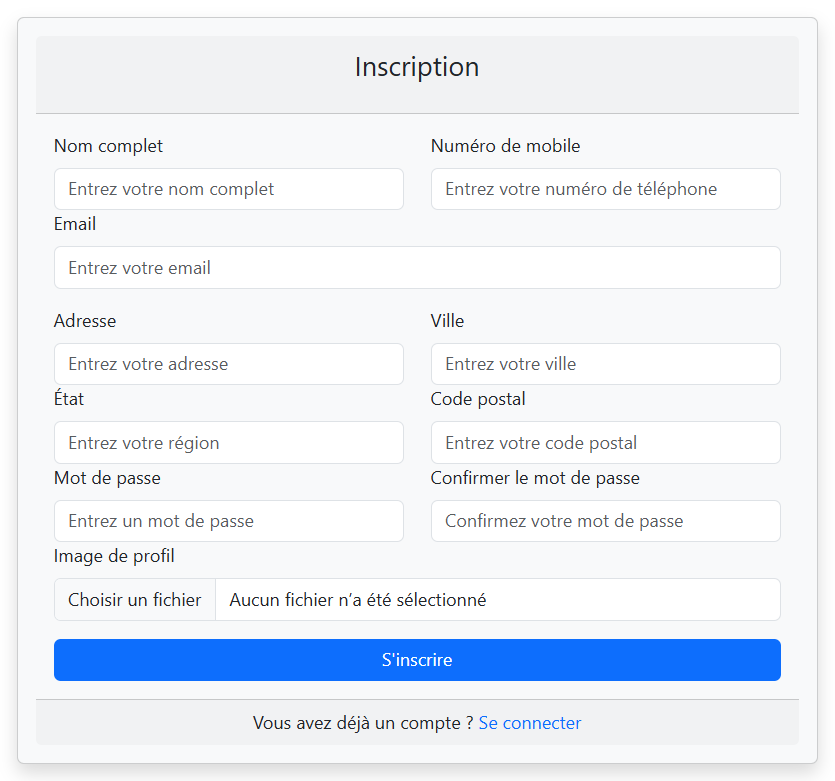
* Itération sur products pour générer une carte par produit :
* Image, titre, prix avec réduction.
* Bouton pour voir les détails du produit.
* Pagination : Affiche les pages disponibles avec th:each sur une séquence générée dynamiquement.



### Page d'inscription (register.html)​

Objectif : Permet aux utilisateurs de créer un compte.

**Points clés :**

* Formulaire d'inscription :
* Champs pour les informations personnelles, les coordonnées, et l'image de profil.
* Validation des champs avec l’attribut required.
* Messages dynamiques : Affiche les messages de succès ou d'erreur liés à l'inscription.

**Thymeleaf :**

* Gestion des messages via th:block.

### Page de réinitialisation du mot de passe (reset\_password.html)​

Objectif : Permet à l'utilisateur de définir un nouveau mot de passe.

**Points clés :**

* Formulaire :
* Champs pour entrer et confirmer le mot de passe.
* Envoie un jeton caché (${token}) avec la requête.
* Messages dynamiques : Similaires à ceux des autres formulaires.

### Page des détails du produit (view\_product.html)​

Objectif : Affiche les informations détaillées d'un produit spécifique.

**Points clés :**

* Affichage du produit :
* Image, titre, description, prix avec réduction.
* Stock disponible ou message Rupture de stock.
* Bouton d'action : Si l’utilisateur est connecté, permet d’ajouter au panier.
* Icônes des options : Paiement à la livraison, retour possible, livraison gratuite.

## Vues du profil administrateur (admin)

Thymeleaf : Intégration dynamique des données avec des blocs conditionnels (th:block) et des itérations (th:each).

Bootstrap : Design responsive pour une compatibilité multi-appareils.

Actions : Gérées via des formulaires ou des liens, assurant une navigation fluide.

### Page Ajouter un Administrateur (add\_admin.html)​

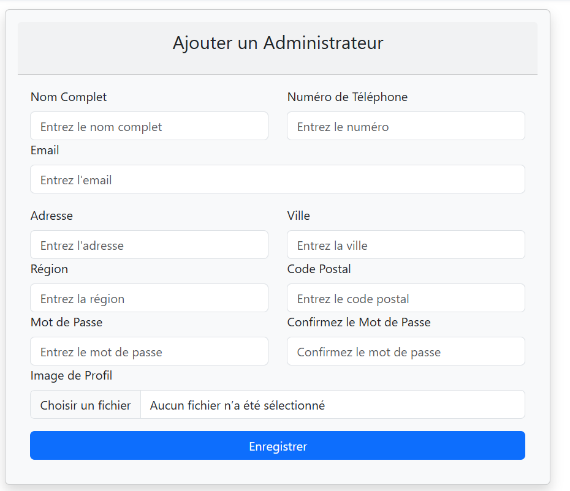
Objectif : Permet à un administrateur de créer un nouvel administrateur.

**Points clés :**

* Formulaire d'ajout : Inclut des champs pour le nom, l’email, le numéro de téléphone, l’adresse, la ville, le code postal, et l’image de profil.
* Envoie une requête POST à /admin/save-admin.

**Validation :**

* Utilise des attributs HTML5 comme required pour garantir la saisie des champs obligatoires.

Affiche les messages de succès/erreur dynamiques via Thymeleaf (th:block).

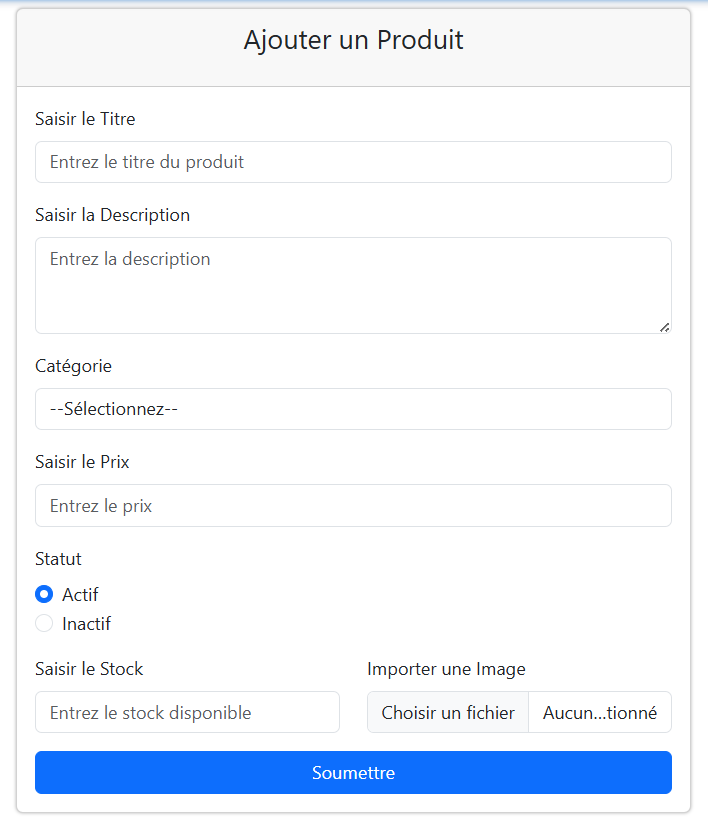
### Page Ajouter un Produit (add\_product.html)​

Objectif : Permet à un administrateur d'ajouter un nouveau produit.

**Points clés :**

* Formulaire d'ajout : Champs pour le titre, la description, la catégorie, le prix, le stock et l'image.
* Statut actif/inactif géré par des boutons radio.
* Sélection dynamique des catégories (th:each).

**Validation :**

* Utilisation de JavaScript pour la validation client-side, notamment pour le prix.

### Page Gérer les Catégories (category.html)​

Objectif : Permet de gérer les catégories (ajout, modification, suppression).

**Points clés :**

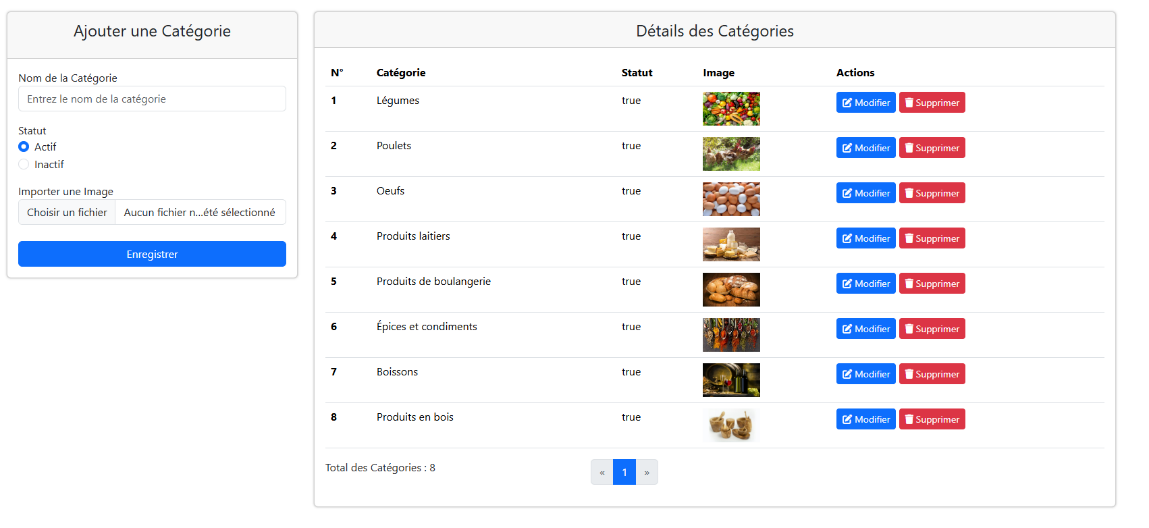
* Ajout de catégorie : Formulaire avec des champs pour le nom, le statut, et une image.
* Envoie une requête POST à /admin/saveCategory.

**Tableau des catégories :**

* Affiche les catégories existantes avec des actions pour les modifier ou les supprimer.
* Utilise th:each pour itérer sur les catégories.

**Pagination :**

* Navigation dynamique avec th:classappend pour activer/désactiver les boutons.



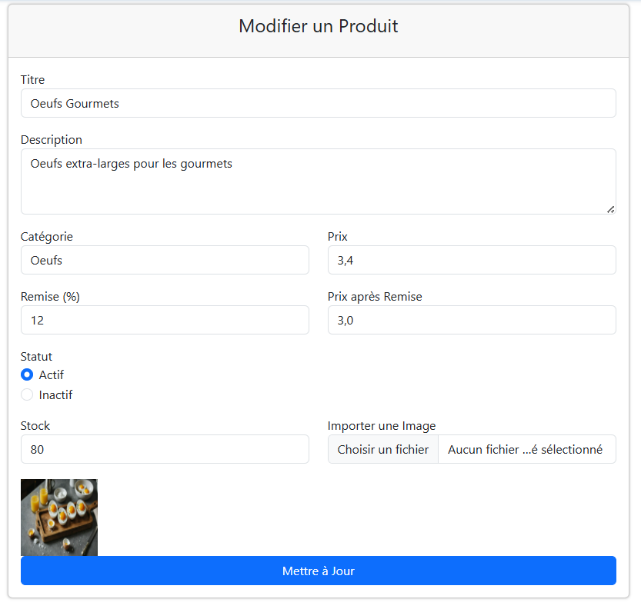
### Page Modifier un Produit (edit\_product.html)​

Objectif : Permet de modifier les détails d’un produit existant.

**Points clés :**

* Pré-remplissage des champs : Utilisation de Thymeleaf (th:value) pour afficher les informations existantes.

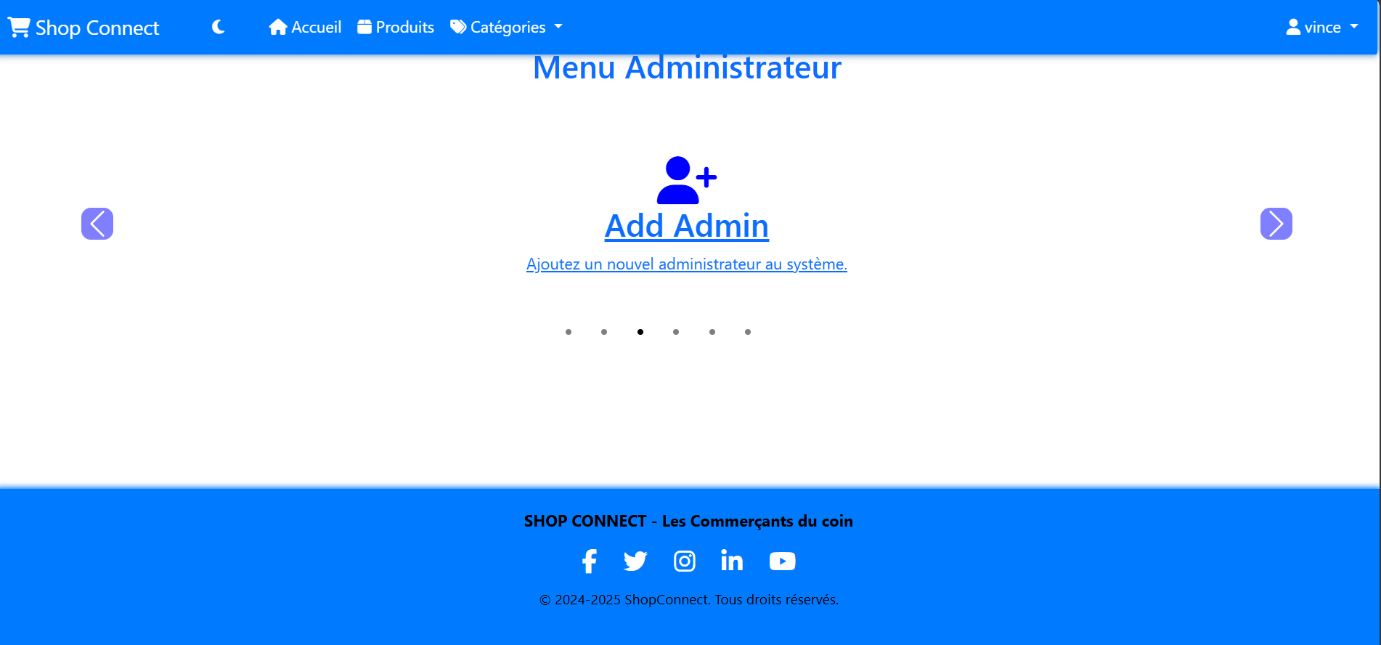
**Validation et mise à jour :**

* Champ caché pour l’ID du produit.
* Actions gérées via une requête POST à /admin/updateProduct.

### Page d'Accueil Administrateur (index.html)​

Objectif : Fournit un tableau de bord pour accéder aux fonctionnalités administratives.

**Points clés :**

* Carrousel interactif : Chaque diapositive redirige vers une section (ex. : produits, commandes).
* Utilisation d’icônes FontAwesome pour l’interface.

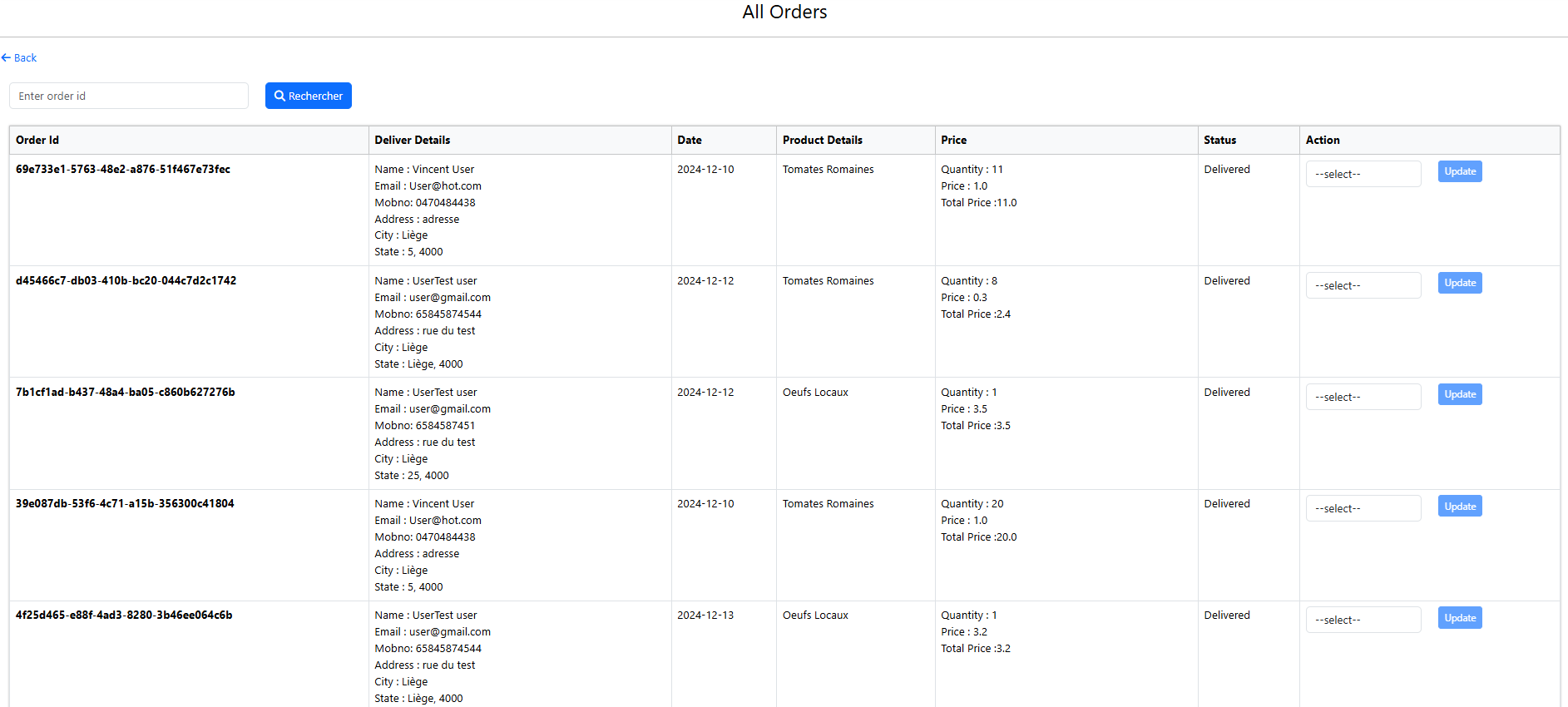
### Page Commandes (orders.html)​

Objectif : Affiche et permet de gérer toutes les commandes.

**Points clés :**

* Tableau des commandes : Affiche les détails (ID, client, produits, prix, statut).
* Actions pour mettre à jour le statut d’une commande.

**Recherche et pagination :**

* Recherche par ID via une requête GET.
* Pagination dynamique basée sur le nombre total de commandes.

### Page Liste des Produits (products.html)​

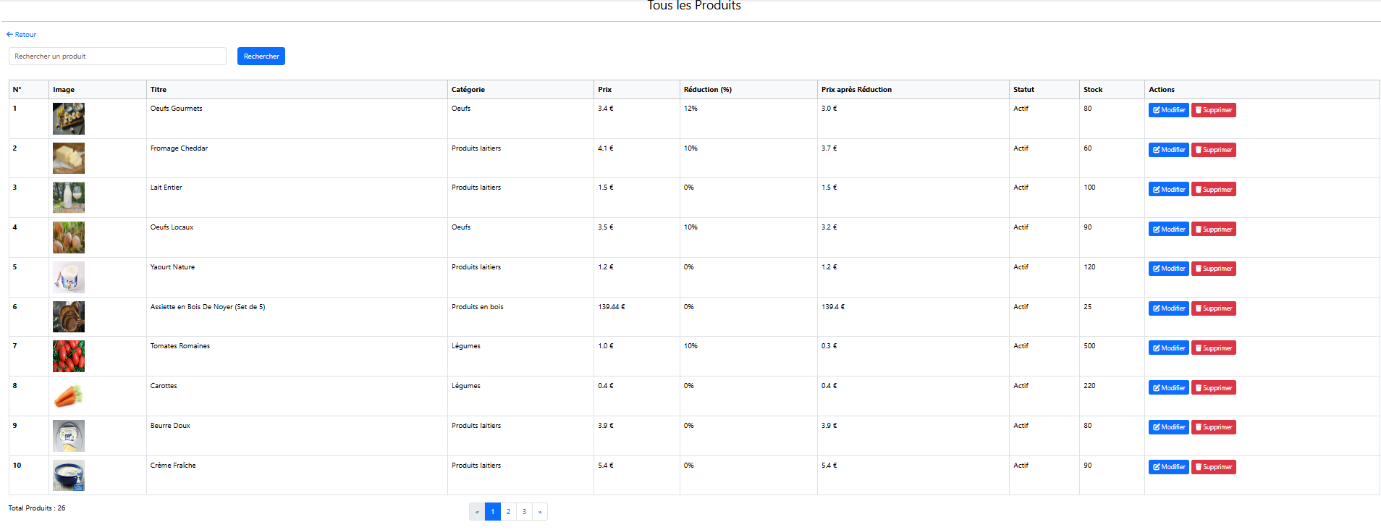
Objectif : Affiche tous les produits avec des options de modification et suppression.

**Points clés :**

* Tableau des produits : Affiche l’image, le titre, la catégorie, le prix, le stock, et les actions.
* Statut actif/inactif affiché dynamiquement avec Thymeleaf (th:text).

**Recherche et pagination :**

* Recherche par mot-clé via une requête GET.
* Navigation avec th:each pour générer les numéros de page.



### Page Profil Administrateur (profile.html)​

Objectif : Permet de visualiser et de modifier les informations du profil administrateur.

**Points clés :**

**Formulaire de mise à jour :**

* Champs pour les informations personnelles et l’image.
* Les rôles et le statut sont en lecture seule.

**Modification du mot de passe :**

* Formulaire séparé pour changer le mot de passe.

### Page Utilisateurs (users.html)​

Objectif : Affiche et gère les utilisateurs ou administrateurs.

**Points clés :**

**Tableau des utilisateurs :**

* Affiche l’image de profil, le nom, l’email, le téléphone, et le statut.
* Actions pour activer/désactiver un utilisateur.

**Gestion du type d’utilisateur :**

* Affiche dynamiquement le titre (utilisateur ou administrateur) via th:if.

## Vues du profil Utilisateur (user)

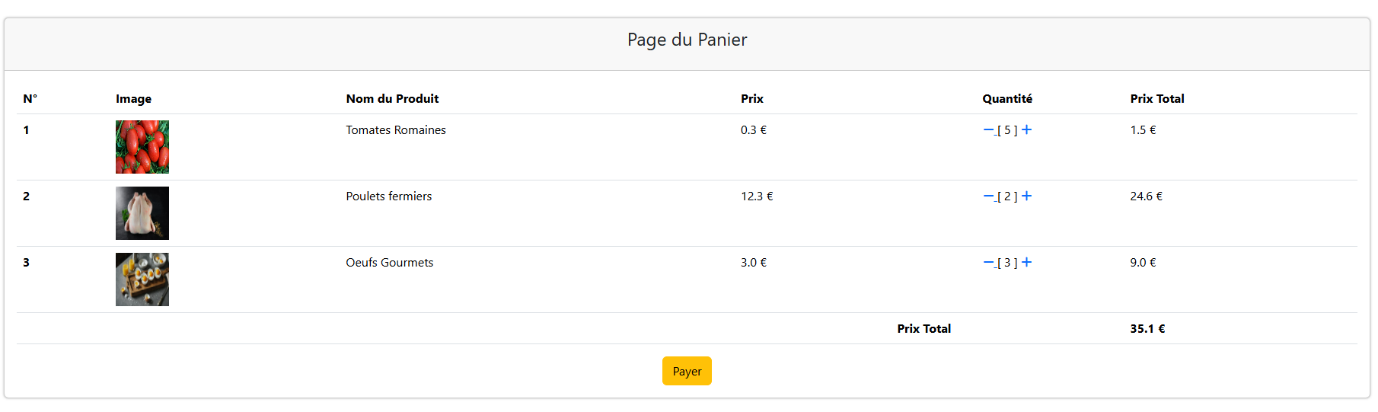
Ces fichiers HTML offrent une interface utilisateur intuitive :

* Thymeleaf : Gestion dynamique des données grâce aux blocs conditionnels et itérations.
* Design : Utilisation de Bootstrap pour une mise en page responsive et attrayante.
* Fonctionnalités orientées utilisateur : Gestion du panier, commandes, profil, et processus de commande clairement séparés.

### Page du Panier (cart.html)​

Objectif : Permet à l'utilisateur de visualiser et gérer les articles ajoutés à son panier.

**Points clés :**

* Tableau des articles : Utilise th:each pour itérer sur la liste des paniers (${carts}).
* Affiche l'image, le nom, la quantité, et le prix total par produit.
* Actions : augmentation/diminution de la quantité avec des liens dynamiques (/user/cartQuantityUpdate).
* Total général : Calculé dynamiquement et affiché via ${totalOrderPrice}.
* Bouton de paiement : Redirige vers /user/orders pour passer la commande.

### Page d'Accueil Utilisateur (home.html)​

Objectif : Servir de tableau de bord principal pour l'utilisateur connecté.

**Points clés :**

**Structure simple :**

* Texte de bienvenue statique et descriptif.
* Utilisation d'un conteneur centré avec une mise en page minimaliste.

### Page de Mes Commandes (my\_orders.html)​

Objectif : Affiche l'historique des commandes passées par l'utilisateur.

**Points clés :**

* Tableau des commandes :
* Itère sur orders pour afficher chaque commande.
* Colonne "Action" : Permet l'annulation de commandes non annulées via /user/update-status.
* Statut des commandes : Dynamique, affiché via ${o.status}.

### Page Commander (order.html)​

Objectif : Permet à l'utilisateur de passer une commande en entrant les détails de facturation et en choisissant un mode de paiement.

**Points clés :**

**Formulaire de facturation :**

* Champs obligatoires pour le prénom, l'adresse, l'email, etc.
* Envoie les données via POST à /user/save-order.

**Type de paiement :**

* Dropdown dynamique avec des options (Paiement à la livraison ou Paiement en ligne).

**Calcul des coûts :**

* Affiche le prix total, les frais de livraison, et les taxes.

### Page Profil Utilisateur (profile.html)​

Objectif : Permet à l'utilisateur de mettre à jour ses informations personnelles et son mot de passe.

**Points clés :**

**Mise à jour des informations :**

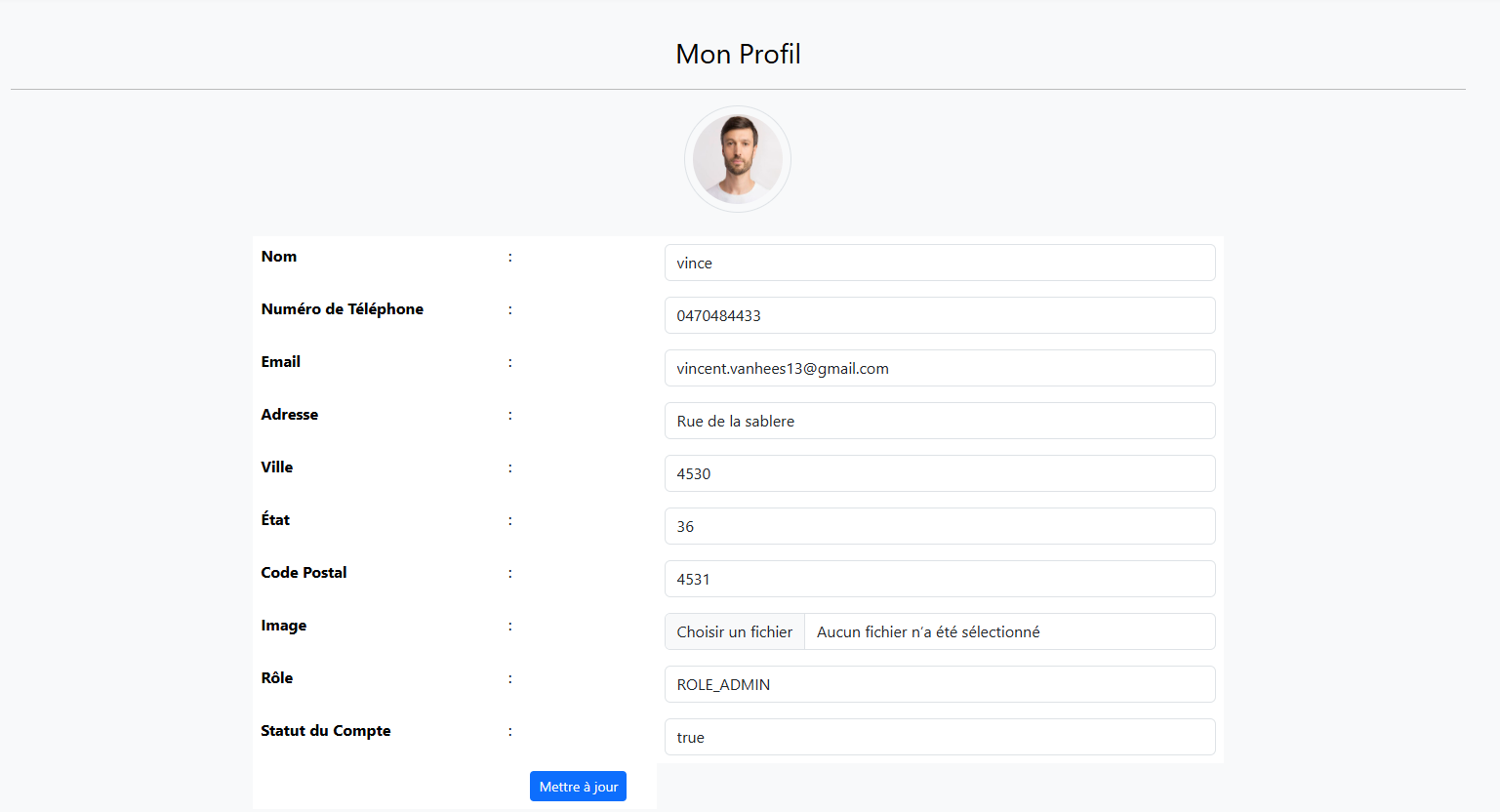
* Formulaire pré-rempli avec les données existantes (${user}).
* Champs pour le nom, l'adresse, l'email, etc.

**Image de profil :**

* Affichée dynamiquement via th:src.
* Option pour uploader une nouvelle image.

**Changement de mot de passe :**

* Formulaire distinct avec validation pour confirmer le mot de passe.



### Page Confirmation de Commande (success.html)​

Objectif : Affiche un message de confirmation après le passage d'une commande.

**Points clés :**

* Icône et message de succès : Icône FontAwesome pour indiquer que la commande a été passée avec succès.
* Texte dynamique pour informer l'utilisateur.

**Actions après commande :**

* Boutons pour revenir à l'accueil ou consulter les commandes.

# Gestion de l'Authentification

## Technologies Utilisées

L’authentification dans l’application Spring Boot est structurée avec une logique pour les échecs, les succès, et la gestion des rôles. Les vues Thymeleaf complètent cette fonctionnalité avec des formulaires conviviaux et dynamiques.

### Frameworks et Bibliothèques

* Spring Boot : Infrastructure principale pour l’application.
* Spring Security : Gère l’authentification et l’autorisation.
* Thymeleaf : Moteur de template pour les vues HTML.
* BCrypt : Algorithme de hachage pour sécuriser les mots de passe.

### Intégration des Composants

* Interaction entre UserRepository, UserService, et SecurityConfig pour charger et valider les utilisateurs.
* Utilisation de Thymeleaf pour afficher dynamiquement les erreurs et messages d’information.

## Partie Back End

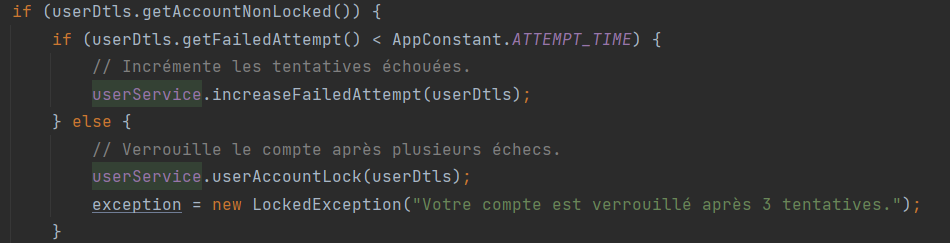
### AuthFailureHandlerImpl (Gestion des échecs d'authentification)

Cette classe gère les échecs d’authentification et fournit une logique avancée pour la gestion des tentatives de connexion.

**Rôle :**

* Traite les échecs de connexion.
* Incrémente les tentatives échouées.
* Verrouille le compte après un nombre maximal de tentatives (égal à AppConstant.ATTEMPT\_TIME).

**Extrait de code :** Gestion des tentatives échouées



**Flux :**

* Extraction de l’e-mail fourni dans le formulaire de connexion.
* Recherche de l’utilisateur dans la base de données via UserRepository.

**Application de la logique suivante :**

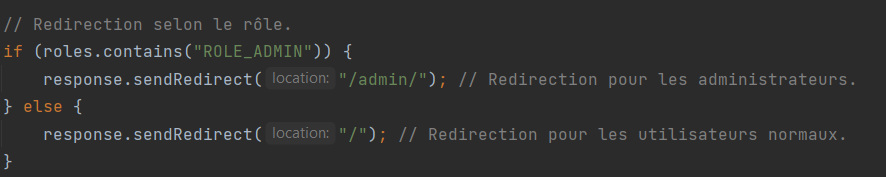
* Si le compte est actif et non verrouillé : incrémenter les tentatives ou verrouiller le compte.
* Si le compte est verrouillé : vérifier si le temps de déverrouillage est expiré.
* Rediriger vers /signin?error en cas d’échec.

### AuthSucessHandlerImpl (Gestion des succès d'authentification)

**Rôle :**

* Récupère les rôles utilisateur.
* Redirige les administrateurs vers /admin/ et les utilisateurs standards vers /.

**Extrait de code :** Gestion des redirections :



**Flux :**

* Extraction des autorités utilisateur via authentication.getAuthorities().
* Conversion des autorités en rôles (à l’aide de AuthorityUtils).
* Application des redirections selon les rôles.

### CustomUser (détails des utilisateurs)

Cette classe implémente UserDetails pour adapter les détails utilisateur au framework Spring Security.

**Rôle :**

* Fournir les informations utilisateur essentielles (mot de passe, e-mail, rôles).
* Gérer l’état du compte (actif, verrouillé, etc.).
* Extrait de code : Gestion des autorités utilisateur

**Extrait de code :** Gestion des autorités utilisateur



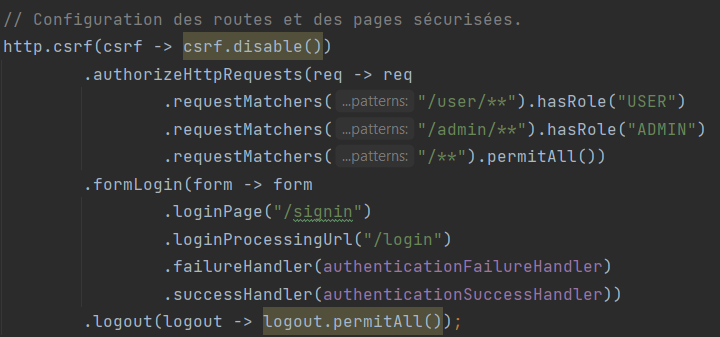
### SecurityConfig (Configuration de Spring Security)

Cette classe configure Spring Security, notamment l’authentification et l’autorisation.

**Rôle :**

* Activer/Désactiver CSRF (Cross-Site Request Forgery).
* Définir les permissions d’accès aux routes selon les rôles utilisateur.
* Configurer l’encodage des mots de passe avec BCryptPasswordEncoder.

**Extrait de code :** Configuration des routes et gestionnaires



**Flux :**

**Gestion des autorisations :**

* Routes /admin/\*\* accessibles uniquement aux administrateurs.
* Routes /user/\*\* accessibles aux utilisateurs connectés.
* Routes publiques accessibles sans connexion

**Authentification :**

* Configuration de la page de connexion personnalisée.
* Gestion des succès et échecs via les gestionnaires personnalisés.

## Partie Front End (Intégration des Vues HTML)

### register.html

Formulaire d’inscription permettant à un utilisateur de créer un compte.

**Fonctionnalités :**

* Champ pour les détails utilisateur (nom, e-mail, mot de passe, etc.).
* Validation des champs requis.
* Gestion des messages de succès et d’erreur via Thymeleaf.

**Extrait de code :** Validation des champs



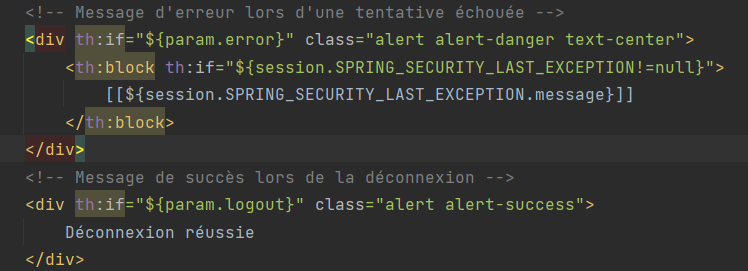
### login.html

Formulaire de connexion permettant aux utilisateurs de s’authentifier.

**Fonctionnalités :**

* Champ pour l’e-mail et le mot de passe.
* Messages dynamiques pour les erreurs d’authentification.

**Extrait de code :** Gestion des messages d’erreur



### forgot\_password.html

Formulaire permettant à un utilisateur de réinitialiser son mot de passe.

**Fonctionnalités :**

* Champ pour saisir l’e-mail.
* Bouton pour envoyer une demande de réinitialisation.

**Extrait de code :** Champ d’e-mail

<input required="required" class="form-control" name="email" type="email" placeholder="Entrez votre email">

# Conclusion

Ce projet a permis de concevoir un outil centralisé répondant aux besoins concrets des commerçants, comme la gestion des produits, des commandes et des promotions, tout en offrant une expérience utilisateur claire et accessible.

Le choix des technologies, telles que Spring Boot pour le backend et PostgreSQL pour la base de données, ainsi que l'utilisation de Thymeleaf et Bootstrap pour le frontend, s’est appuyé sur des critères de simplicité, de maintenabilité et d’efficacité. Bien que le développement ait présenté des défis, notamment en termes de conception des flux et d’intégration des fonctionnalités, les solutions mises en œuvre se sont avérées adaptées au contexte et aux contraintes du projet.

## Perspectives d'Évolution

L’application Shop Connect pourra évoluer pour répondre encore mieux aux besoins des commerçants et des utilisateurs grâce à diverses fonctionnalités prévues. Les paiements en ligne constitueront une avancée importante, avec l’intégration d’outils tels que Stripe ou PayPal, permettant de réaliser des transactions de manière sécurisée via carte de crédit ou autres moyens de paiement modernes. La mise en œuvre de ces options renforcera la confiance des utilisateurs tout en simplifiant leurs processus d’achat.

Une autre amélioration envisagée est la prise en charge de plusieurs langues. Chaque utilisateur pourra ainsi choisir l’affichage dans sa langue préférée, comme le français ou l’anglais, rendant l’application accessible à un public plus large et diversifié. Cette fonctionnalité s’appuiera sur des systèmes de localisation performants pour adapter dynamiquement l’interface en fonction de la préférence linguistique.

Pour enrichir l’expérience utilisateur et optimiser les performances SEO, les URL du site deviendront explicites. Cette approche permettra non seulement d’améliorer le référencement naturel sur les moteurs de recherche, mais aussi de rendre la navigation plus intuitive pour les utilisateurs. Par exemple, un produit ou une catégorie aura une adresse compréhensible directement reflétant son contenu.

La gestion des données administratives sera également un axe central d’évolution. Les administrateurs auront accès à une interface complète leur permettant de gérer efficacement les informations relatives aux produits, utilisateurs, commandes, promotions et catégories. Ce tableau de bord simplifié renforcera le contrôle et la fluidité des opérations de maintenance.

Un aspect clé pour dynamiser l’utilisation de ShopConnect sera l’intégration de fonctionnalités marketing. Les responsables pourront configurer des promotions spécifiques sur les produits ou les catégories, en paramétrant notamment des dates d’application et des conditions spécifiques. Ces outils permettront de stimuler les ventes tout en offrant aux clients des opportunités intéressantes.

En termes de sécurité, l’accent sera mis sur la protection des données et des API. Une authentification sécurisée via OAuth ou JWT sera implémentée pour garantir un haut niveau de sûreté. De plus, l’application sera conçue pour résister aux attaques courantes, afin d’assurer une expérience utilisateur sereine. Enfin, l’apparence visuelle restera une priorité : ShopConnect continuera de proposer une interface moderne, esthétique et totalement responsive, adaptée à tous les types de supports, qu’ils soient mobiles, tablettes ou ordinateurs.

Ces perspectives d’évolution placeront ShopConnect dans une position idéale pour offrir une expérience utilisateur optimale, tout en soutenant les commerçants dans leur transition numérique.