Système de Vente en Ligne de Véhicules

UML et Design Patterns

UNIVERSITE DE YAOUNDE I

DEPARTEMENT D'INFORMATIQUE
Sous la supervision du Dr :
Valéry MONTHE

February 14, 2025

Membres du groupe

Nom	Matricule
MAHAMAT SALEH MAHAMAT	21T2423
TCHAMI TAMEN SORELLE	20U2855
TSAKEU NGUEMO MARILYN FLORA	21T2627

Table des matières

Introduction

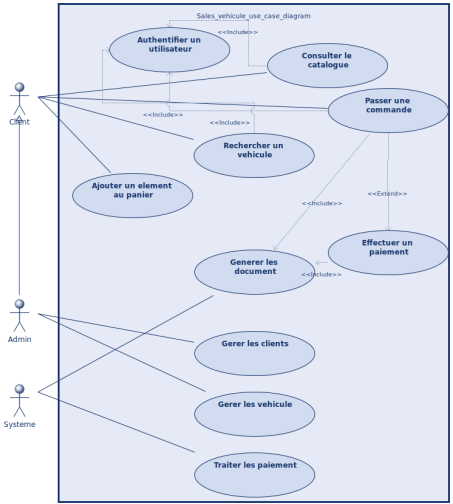
- Présentation du projet : Vente en ligne de véhicules.
- Objectif : Implémentation d'une architecture avec UML et Design Patterns.
- Technologies utilisées : Spring Boot, React.js, H2.

Architecture Technique

L'application est construite avec les technologies suivantes :

- **Backend** : implémenté en **Spring Boot**, exposant des API REST sécurisées.
- **Frontend** : développé avec **React.js** et utilisant Axios pour les requêtes API.
- **Base de données** : gérée avec **H2**.
- **Authentification** : basée sur **Spring Security** avec gestion des sessions HTTPS.
- **Stockage des images** : fichiers stockés sur le serveur et accessibles via une API.
- **Latex** : Pour la rédaction des documents.

Diagrammes UML - Cas d'utilisation



- Creation de compte.
- Passer une commande.
- Effectuer un paiement.
- Gestion des utilisateurs.
- Ajout et suppression de véhicules.
- Gestion des commandes et paiements.
- Authentification et autorisation.

Diagramme de classes

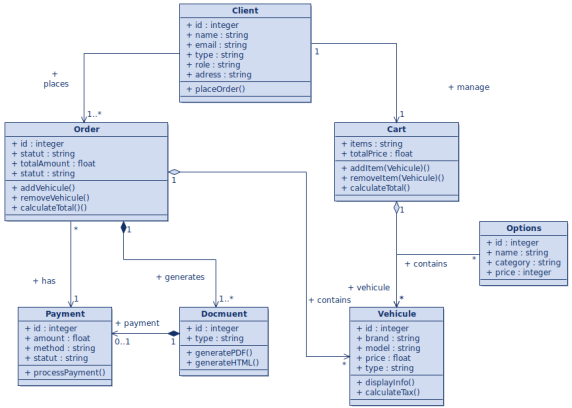
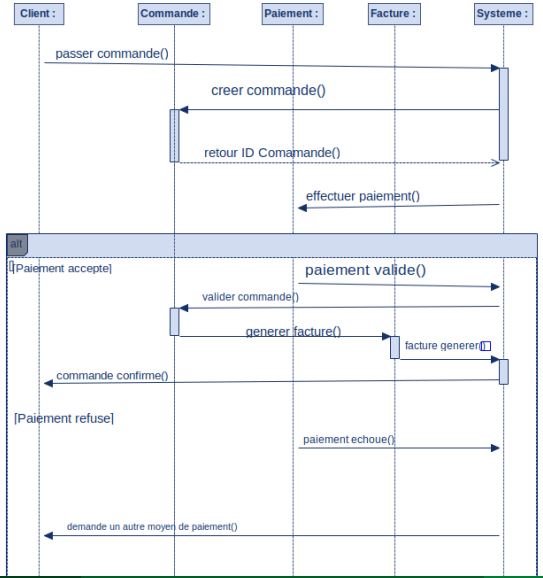


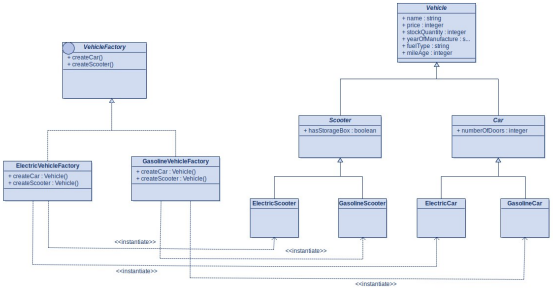
Diagramme de séquence



Répartition des Patterns

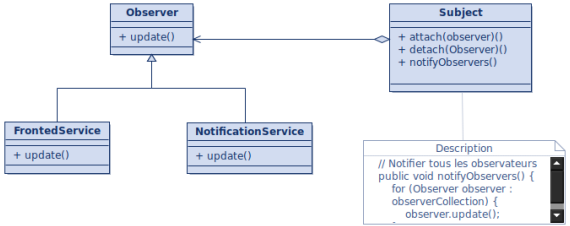
- **Catalogue de véhicules** : Abstract Factory, Observer Pattern, Iterator Pattern, Decorator Pattern.
- **Gestion des commandes** : Factory Method Pattern.
- **Gestion des documents** : Builder Pattern, Adapter Pattern, Bridge Pattern, Singleton.
- **Gestion des clients** : Composite Pattern.
- **Gestion des paiements** : Template Method Pattern.
- **Authentification** : Singleton Pattern.

Gestion du Catalogue - Abstract Factory



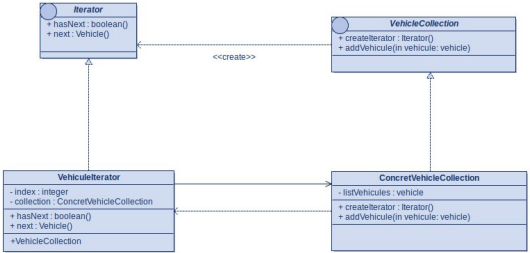
Ce pattern permet de créer des objets sans spécifier leurs classes concrètes, assurant ainsi une flexibilité dans la gestion du catalogue.

Gestion du Catalogue - Observer Pattern



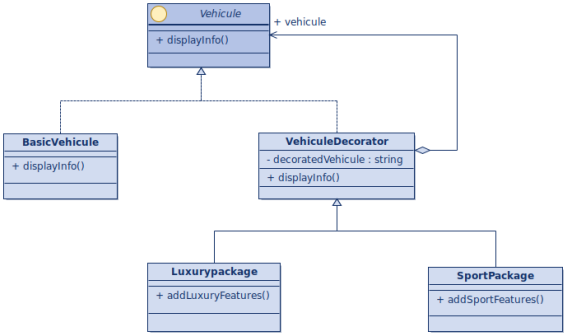
Ce pattern permettre a plusieurs module de recevoir des notifications automatiques lorsqu'un changement est effectue sur un sujet observe (Subject)

Gestion du Catalogue - Iterator Pattern



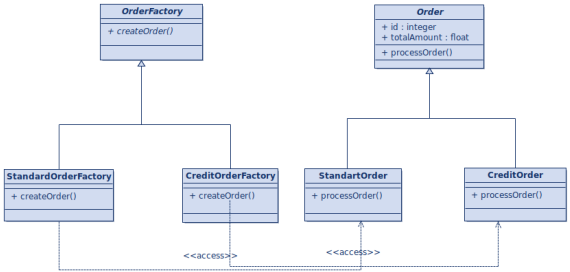
Permet de parcourir une collection d'objets sans exposer sa structure interne.

Gestion du Catalogue - Decorator Pattern



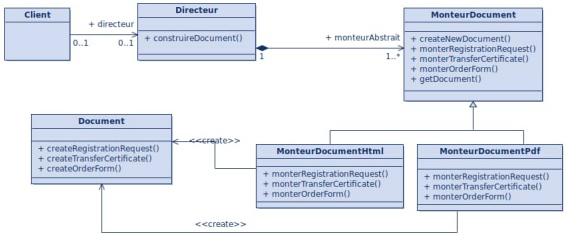
Permet d'ajouter dynamiquement des fonctionnalités aux objets sans modifier leur structure.

Gestion des Commandes - Factory Method



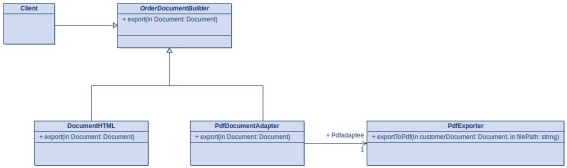
Définit une interface pour la création d'objets, permettant une gestion dynamique des types de commandes.

Gestion des Documents - Builder Pattern



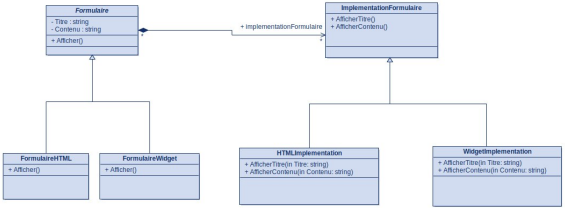
Simplifie la construction des documents complexes tels que les factures et les contrats.

Gestion des Documents - Adapter Pattern



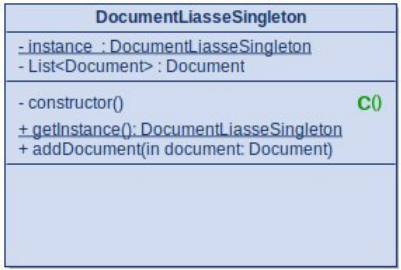
Permet d'adapter l'interface d'un objet existant pour être utilisée dans un autre contexte.

Gestion des Documents - Bridge Pattern



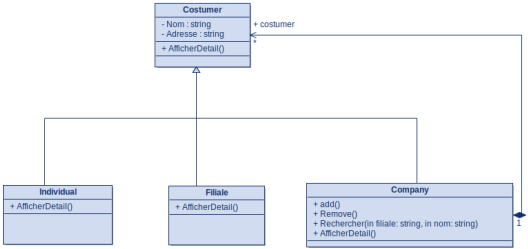
Sépare une abstraction de son implémentation pour permettre leur évolution indépendamment.

Gestion des Documents - Singleton



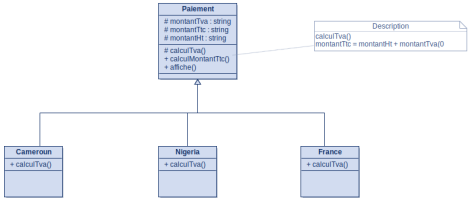
Garantir qu'une seule instance de la classe DocumentTemplate existe et qu'elle est accessible globalement.

Gestion des Clients - Composite Pattern



Permet de traiter les clients sous forme hiérarchique en les organisant en arbres.

Gestion des Paiements - Template Method Pattern



Définit l'algorithme général de traitement des paiements tout en permettant aux sous-classes de définir des étapes spécifiques.

Conclusion

- Le projet applique des principes avancés d'architecture logicielle.
- Utilisation des Design Patterns pour une meilleure structuration.
- Perspectives : Modularisation, tests automatisés, migration vers microservices.


Merci de
votre
attention