Honoris united universities - Ecole Marocaine des Sciences de l'Ingénieur

Rapport de projet de fin d'année

Réalisé par

Haitham El Abdioui
Mohamed Amine Kharbouch
Aboubakr Ketoun

Système de Gestion de Véhicules

---

sous la direction de :

Mr Mossab Batal — Encadrant Académique

année universitaire

(2023/2024)

Remerciements

Nous souhaitons exprimer notre profonde gratitude à Mr. Mossab Batal pour son soutien inestimable tout au long de notre projet (Python/django). Sa guidance avisée et son expertise technique ont été des atouts essentiels pour la réussite de ce projet. Son engagement a été crucial pour assurer un développement fluide et efficace de notre application. Mr. Batal a apporté des solutions innovantes aux défis techniques que nous avons rencontrés et a partagé des conseils précieux pour optimiser notre code. Grâce à ses recommandations, nous avons pu améliorer les performances de notre application et implémenter des fonctionnalités complexes avec une grande efficacité. Il a eu un impact positif et durable sur tous les aspects de notre travail. Nous sommes reconnaissants d'avoir pu bénéficier de son expertise et de sa vision, et nous espérons avoir à nouveau l'opportunité de profiter de sa sagesse et de son expérience à l'avenir, afin d'atteindre de nouveaux sommets de réussite. Merci infiniment, Mr. Mossab Batal, pour votre précieuse contribution et pour votre engagement indéfectible envers notre succès.

Table des matières

Table des figures

Introduction

Dans un monde où le commerce en ligne connaît une croissance exponentielle, il est essentiel de disposer d'outils performants et intuitifs pour répondre aux besoins des clients et des administrateurs. C'est dans ce contexte que notre projet de développement d'une application Django s'inscrit. Notre objectif est de créer une plateforme robuste pour la gestion des services d'une agence de vente de véhicules en ligne, alliant efficacité et convivialité. Cette application est conçue pour offrir une expérience utilisateur fluide et agréable. Elle permet aux clients de consulter, rechercher et gérer les produits de manière intuitive, tandis que les administrateurs disposent d'outils puissants pour une gestion efficace des produits. Grâce à une interface utilisateur bien pensée et des fonctionnalités avancées, notre système vise à faciliter la navigation et la gestion des paniers en ligne, rendant ainsi le processus d'achat aussi simple et agréable que possible. En utilisant Django, un framework web hautement flexible et sécurisé, nous nous assurons que notre plateforme est non seulement performante mais également évolutive, capable de s'adapter aux besoins croissants de l'agence et de ses clients. Ce projet représente une avancée significative dans la manière dont les services de vente de véhicules sont gérés en ligne, apportant une solution moderne et efficace pour tous les acteurs impliqués.

Chapitre 1

Analyse fonctionelle

1.1 Cahier des charges

Ce document décrit en détail les spécifications et les fonctionnalités requises pour le développement d'un système de gestion de produits en ligne, visant à améliorer l'expérience utilisateur pour les clients et à faciliter les opérations pour les administrateurs. Le système permet aux clients de consulter les produits disponibles, de lancer des recherches spécifiques, d'ajouter des produits à leur panier, de modifier le contenu de leur panier, et de le supprimer si nécessaire. Pour garantir la sécurité des transactions et des données, toutes les actions relatives à la gestion du panier nécessitent une authentification via un processus de login sécurisé. Les administrateurs, de leur côté, disposent de fonctionnalités pour ajouter de nouveaux produits au catalogue, modifier les informations des produits existants, et supprimer des produits. Ces actions sont également protégées par un login sécurisé. Le diagramme de cas d'utilisation inclus dans ce document illustre les interactions entre les acteurs (clients et administrateurs) et les différentes fonctionnalités du système. Les spécifications fonctionnelles détaillent l'ensemble du processus d'authentification, qui comprend l'accès à la page de login, la saisie des identifiants, et la vérification des informations pour accéder aux différentes fonctionnalités. La gestion des produits inclut des étapes claires pour consulter la liste des produits, effectuer des recherches par mots-clés, et pour les administrateurs, ajouter, modifier et supprimer des produits avec des flux de travail bien définis. En termes de gestion de panier, les clients peuvent ajouter des produits en sélectionnant ceux qui les intéressent, modifier les quantités ou les éléments du panier, et le vider entièrement si besoin. Ce cahier des charges vise à garantir que le système répondra de manière optimale aux besoins des utilisateurs finaux tout en maintenant des standards élevés de sécurité et de facilité d'utilisation.

1.2 Diagramme de cas d'utilisation

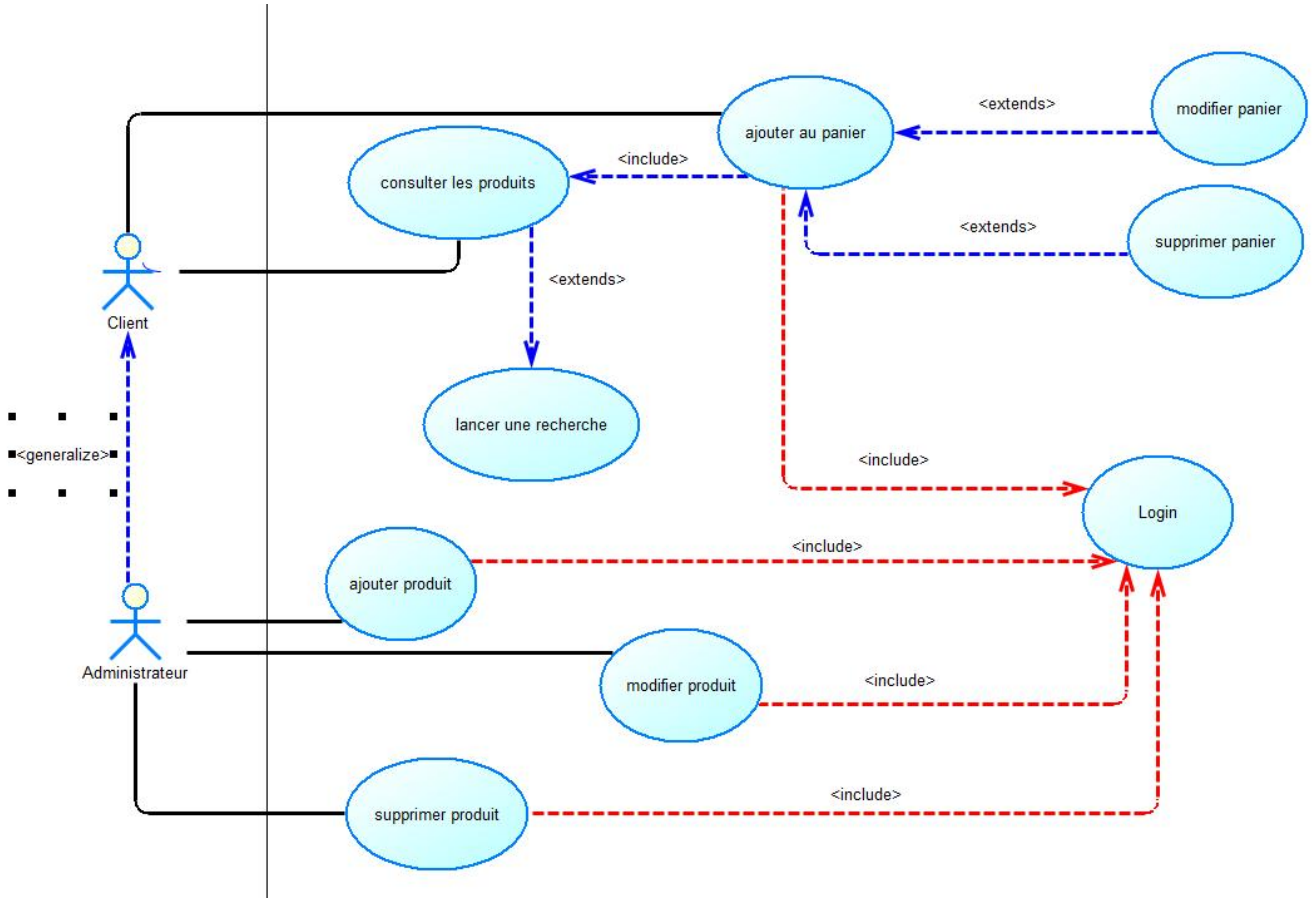

FIGURE 1.1 – Diagramme de cas d'utilisation

1.3 Diagramme de classe

Ce diagramme de classe présente un système de gestion de véhicules pour une agence de vente de véhicules. Il inclut plusieurs classes et leurs relations. La classe "Client" peut utiliser plusieurs "Paniers" et consulter un "Inventaire" de véhicules, chaque inventaire contenant un ou plusieurs "Véhicules". Les véhicules sont gérés par des "Administrateurs". Un véhicule est représenté par la classe parent "Véhicule", qui est spécialisée en deux sous-classes : "Moto" et "Voiture". Les relations entre ces classes montrent que les clients peuvent avoir des paniers, consulter l'inventaire des véhicules, et que les administrateurs ont la responsabilité de gérer ces véhicules. L'inventaire est composé de véhicules, établissant une relation de composition, tandis que les sous-classes Moto et Voiture héritent des propriétés de la classe Véhicule. Ce diagramme illustre les interactions clés et les rôles de chaque entité dans le système de gestion de l'agence de vente de véhicules.

1.4 Outils

Ce projet de développement d'une application pour la gestion des services d'une agence de vente de véhicules en ligne a été réalisé en utilisant une combinaison de technologies modernes et puissantes, afin de garantir une plateforme performante et intuitive.
Pour la partie front-end, nous avons utilisé HTML pour structurer les pages web, CSS pour les styliser et améliorer leur présentation visuelle, et JavaScript pour ajouter des fonctionnalités interactives et

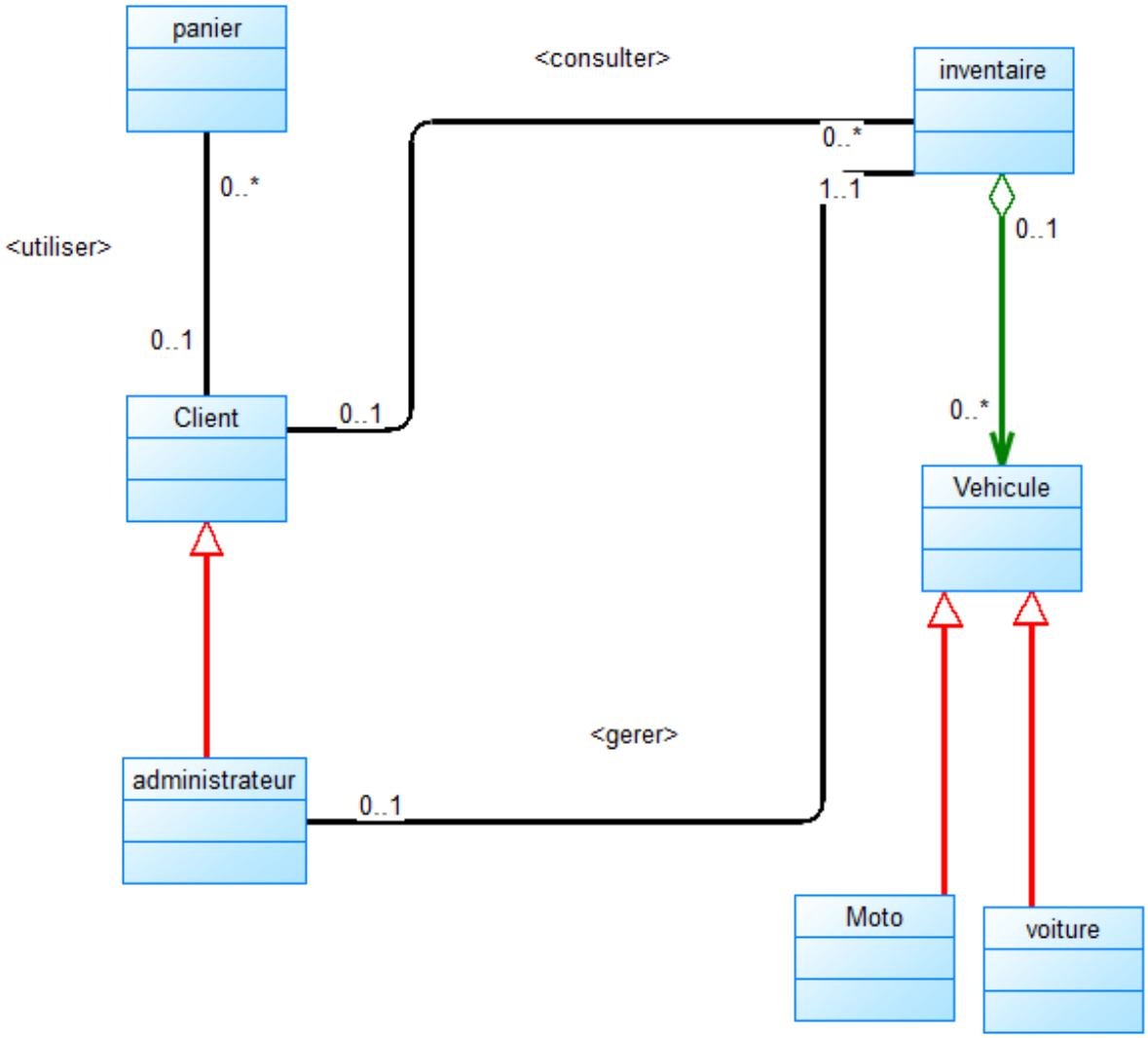

FIGURE 1.2 – Diagramme de classe

dynamiques. Nous avons également intégré Bootstrap, un framework CSS populaire, pour accélérer le développement et garantir une interface utilisateur réactive et esthétique, qui s'adapte parfaitement à différents types de dispositifs, y compris les mobiles et les tablettes.
Le back-end de l'application a été développé en utilisant Django, un framework web Python réputé pour sa robustesse et sa simplicité. Django nous a permis de construire une application sécurisée et évolutive, tout en facilitant la gestion des bases de données et l'implémentation des fonctionnalités complexes nécessaires pour la gestion des produits et des paniers en ligne.
En combinant ces outils, nous avons pu créer une plateforme qui non seulement répond aux besoins des clients en matière de consultation et de recherche de produits, mais aussi aux exigences des administrateurs en termes de gestion efficace des produits. Ce projet représente une solution complète et intégrée, utilisant le meilleur de chaque technologie pour offrir une expérience utilisateur fluide et satisfaisante.

Chapitre 2

Création d'un projet

La réalisation de ce projet de gestion des services d'une agence de vente de véhicules en ligne s'inscrit dans une démarche visant à offrir une solution moderne et performante, répondant aux besoins spécifiques des clients et des administrateurs. Avec l'évolution rapide des technologies et l'augmentation de la demande pour des plateformes de commerce en ligne, il est crucial de développer des applications qui non seulement répondent aux attentes actuelles, mais qui sont également capables de s'adapter aux futurs besoins du marché.
Pour atteindre cet objectif, nous avons défini les étapes de réalisation de notre projet :

2.1 Étapes pour Créer un Projet Django

2.1.1 Créer un Environnement Virtuel

Créez un environnement virtuel pour isoler les dépendances de votre projet :

```
python -m venv nom_de_lenvironnement
```

Activez l'environnement virtuel :

Sur Windows :

```
.\nom_de_lenvironnement\Scripts\activate
```

2.1.2 Installer Django

Installez Django dans votre environnement virtuel :

```
pip install django
```

2.1.3 Créer un Nouveau Projet Django

Utilisez la commande suivante pour créer un nouveau projet :

```
django-admin startproject nom_du_projet
```

Remplacez `nom_du_projet` par le nom de votre projet.

2.1.4 Naviguer dans le Répertoire du Projet

Déplacez-vous dans le répertoire du projet nouvellement créé :

```
cd nom_du_projet
```

2.1.5 Configurer la Base de Données

Dans le fichier `settings.py`, configurez les paramètres de votre base de données si vous souhaitez utiliser une base de données autre que SQLite.

2.2 Settings.py

Le fichier `settings.py` est un composant central dans un projet Django. Il contient toutes les configurations nécessaires pour le fonctionnement du projet. Voici les principales fonctions de ce fichier :

- **Configuration de Base du Projet :**
  - `BASE_DIR` : Définit le répertoire de base du projet, facilitant l'accès aux autres répertoires comme les templates, les fichiers statiques et les fichiers média.
- **Sécurité et Paramètres Globaux :**
  - `SECRET_KEY` : Une clé secrète utilisée pour les opérations cryptographiques, essentielle pour la sécurité des sessions et des cookies.
  - `DEBUG` : Indicateur pour activer ou désactiver le mode débogage. En mode débogage, Django affiche des pages d'erreur détaillées utiles en développement. En production, ce mode doit être désactivé pour des raisons de sécurité.
  - `ALLOWED_HOSTS` : Liste des hôtes/domains autorisés à servir le projet. Cela prévient les attaques de type HTTP Host header.
- **Applications Installées :**
  - `INSTALLED_APPS` : Liste des applications Django et des applications tierces installées. Cette liste permet à Django de savoir quelles applications sont actives et quelles migrations doivent être appliquées.
- **Middleware :**
  - `MIDDLEWARE` : Liste des middlewares utilisés par le projet. Les middlewares sont des composants qui s'interposent entre les requêtes et les réponses, permettant de gérer des tâches comme la sécurité, les sessions, les cookies, etc.
- **Configuration des Templates :**
  - `TEMPLATES` : Configuration des répertoires de templates et des context processors. Cela indique à Django où chercher les fichiers HTML et comment traiter les contextes des vues.
- **Configuration de la Base de Données :**
  - `DATABASES` : Configuration des paramètres de connexion à la base de données. Par défaut, Django utilise SQLite, mais ce paramètre peut être modifié pour utiliser d'autres bases de données comme PostgreSQL, MySQL ou Oracle.
- **Validation des Mots de Passe :**
  - `AUTH_PASSWORD_VALIDATORS` : Liste des validateurs de mots de passe pour renforcer la sécurité des comptes utilisateurs.
- **Internationalisation et Localisation :**
  - `LANGUAGE_CODE` : Définit la langue par défaut du projet.
  - `TIME_ZONE` : Définit le fuseau horaire par défaut du projet.

    - `USE_I18N`, `USE_L10N`, `USE_TZ` : Paramètres pour activer l'internationalisation, la localisation et la gestion des fuseaux horaires.
- **Fichiers Statistiques et Médias :**
    - `STATIC_URL` : URL pour accéder aux fichiers statiques (CSS, JavaScript, images).
    - `MEDIA_URL` et `MEDIA_ROOT` : Configuration des URL et des répertoires pour servir les fichiers média (uploads d'images, vidéos, etc.).
- **Configuration par Défaut des Clés Primaires :**
    - `DEFAULT_AUTO_FIELD` : Définit le type de clé primaire par défaut pour les modèles.

En résumé, le fichier `settings.py` centralise toutes les configurations nécessaires pour que Django puisse exécuter le projet de manière efficace et sécurisée. Il permet une gestion unifiée des paramètres globaux, des applications installées, de la sécurité, des bases de données, des templates, des fichiers statiques et média, ainsi que des options de localisation et d'internationalisation

2.3 Urls.py

Le fichier `urls.py` dans un projet Django est un élément essentiel qui définit les schémas d'URL pour votre application web. Il agit comme un guide pour diriger les requêtes entrantes vers les vues appropriées de votre application. Voici une définition détaillée des composants typiques que vous pourriez trouver dans un fichier `urls.py` :

- **Importations :** Au début du fichier, vous importez généralement les modules Django nécessaires. Cela inclut souvent `path` ou `re_path` (pour définir les schémas d'URL), ainsi que les vues ou les fonctions de vue de votre application.
- **Définition des schémas d'URL :** Les schémas d'URL définissent la correspondance entre les URL entrantes et les vues correspondantes de votre application. Vous pouvez utiliser des expressions régulières (`re_path`) ou des schémas de chemins (`path`).
- **Nommage des URL :** En donnant un nom à chaque URL à l'aide de l'argument `name` dans la fonction `path`, vous pouvez référencer facilement cette URL dans vos templates Django.
- **Inclusions d'URL :** Dans les projets Django de grande taille, vous pourriez avoir plusieurs applications avec leurs propres fichiers `urls.py`. Pour inclure ces URLs dans le fichier principal `urls.py` du projet, vous utilisez généralement `include`.
- **Passage d'arguments aux vues :** Vous pouvez capturer des parties d'une URL et les transmettre comme arguments aux vues correspondantes. Cela est souvent réalisé en utilisant des expressions régulières pour capturer les éléments variables de l'URL.
- **Vue par défaut :** Dans la plupart des configurations, vous spécifiez une vue par défaut pour les URL non valides. Cela peut être une page d'erreur 404 ou une page d'accueil.

En résumé, le fichier `urls.py` dans un projet Django définit comment les URLs sont mappées aux vues de votre application, facilitant ainsi la navigation des utilisateurs à travers votre site web.

Chapitre 3

Création d'une application

Dans le contexte de Django, une "application" est une composante autonome et réutilisable qui accomplit une tâche spécifique au sein d'un projet web. Une application Django est une collection de fonctionnalités liées qui interagissent entre elles pour réaliser un ensemble cohérent de fonctionnalités ou de services.
Voici quelques caractéristiques importantes d'une application Django :

- **Indépendance et modularité :** Les applications Django sont conçues pour être indépendantes et modulaires. Elles peuvent être développées, testées et déployées de manière autonome. Chaque application peut avoir ses propres modèles, vues, URLs et fichiers statiques, ce qui facilite la réutilisation et la maintenance du code.
- **Structure organisée :** Une application Django suit une structure organisée. Elle comprend généralement des fichiers tels que models.py pour définir les modèles de données, views.py pour définir le comportement des vues, urls.py pour définir les schémas d'URL, et éventuellement forms.py pour gérer les formulaires. Cette structure claire facilite la compréhension et la gestion du code.
- **Réutilisabilité :** Les applications Django sont conçues pour être réutilisables. Vous pouvez les intégrer dans différents projets sans avoir à réécrire le code. Par exemple, une application de gestion des utilisateurs ou une application de gestion des articles de blog peut être développée une fois et utilisée dans plusieurs projets.
- **Facilité díntégration :** Les applications Django sont faciles à intégrer dans un projet existant. Vous pouvez les ajouter à votre projet en les incluant dans le fichier INSTALLED_APPS de votre fichier de configuration settings.py.
- **Extensibilité :** Les applications Django sont extensibles. Vous pouvez ajouter de nouvelles fonctionnalités à une application existante ou étendre ses fonctionnalités en ajoutant de nouveaux modules ou en modifiant ceux existants.
- **Simplicité de déploiement :** En raison de leur modularité et de leur structure organisée, les applications Django sont faciles à déployer. Vous pouvez les packager en tant que modules réutilisables et les distribuer via des dépôts de packages Python tels que PyPI.

En résumé, une application Django est une composante autonome et réutilisable d'un projet web Django, conçue pour accomplir une tâche spécifique de manière efficace, modulaire et réutilisable.

3.1 Urls.py

Importations :

— `from django.urls import path` : Importe la fonction `path` de Django pour définir les schémas d'URL.
— `from . import views` : Importe les vues définies dans le fichier `views.py` du répertoire actuel.
— `from django.contrib.auth.views import LogoutView` : Importe la vue de déconnexion fournie par Django pour gérer la déconnexion des utilisateurs.
— `from django.conf.urls.static import static` : Importe la fonction `static` pour servir les fichiers statiques pendant le développement.
— `from django.conf import settings` : Importe les paramètres de configuration de Django.

Définition des schémas d'URL :

```
\begin{verbatim}
    \item \texttt{path('', views.home, name="home")}: Définit le chemin racine de l'applic
    appelant la vue \texttt{home} et lui donnant le nom "home".
    \item \texttt{path('about/',views.about,name="about")}: Définit le chemin "about/",
    appelant la vue \texttt{about} et lui donnant le nom "about".
    \item \texttt{path('contact/',views.contact,name="contact")}: Définit le chemin "conta
    appelant la vue \texttt{contact} et lui donnant le nom "contact".
    \item \texttt{path('products/', views.list_products, name="product_list")}: Définit le
    appelant la vue \texttt{list_products} et lui donnant le nom "product_list".
    \item \texttt{path('fproducts/', views.filterproduit, name='fproduct')}: Définit le ch
    appelant la vue \texttt{filterproduit} et lui donnant le nom "fproduct".
    \item \texttt{path('products/<int:id>/', views.product_detail, name='product_detail')}
    Définit le chemin "products/int:id/", appelant la vue \texttt{product_detail} avec un
    et lui donnant le nom "product_detail".
    \item \texttt{path('signup/', views.signup_or_login, name='signup')}: Définit le chemi
    appelant la vue \texttt{signup_or_login} et lui donnant le nom "signup".
    \item \texttt{path('logout/', views.logout_view, name='logout')}: Définit le chemin "l
    appelant la vue \texttt{logout_view} fournie par Django et lui donnant le nom "logout"
    \item \texttt{path('login/', views.signup_or_login, name='login')}: Définit le chemin
    appelant la vue \texttt{signup_or_login} et lui donnant le nom "login".
    \item \texttt{path('cart/',views.get_cart,name="my_cart")}: Définit le chemin "cart/",
    appelant la vue \texttt{get_cart} et lui donnant le nom "my_cart".
    \item \texttt{path('add_cart/<int:product_id>',views.add_to_cart,name="add_to_cart")}:
    Définit le chemin "add_cart/int:product_id", appelant la vue \texttt{add_to_cart} avec
    et lui donnant le nom "add_to_cart".
    \item \texttt{path('add_cart/<int:product_id>/<int:quantity>',views.add_to_cart,name="
    Définit le chemin "add_cart/int:product_id/int:quantity", appelant la vue \texttt{add_
    \texttt{quantity} de type entier et lui donnant le nom "add_to_cart_qte".
    \item \texttt{path('update_cart/<int:product_id>/<int:quantity>',views.add_to_cart,nam
    Définit le
```

3.2 Forms.py

Ce fichier contient des formulaires Django qui sont utilisés pour valider et traiter les données soumises par les utilisateurs dans une application web Django. Voici une description détaillée de chaque formulaire :

Classe ProductForm :

Ce formulaire est utilisé pour créer un nouveau produit dans l'application.
- Il est basé sur le modèle Product.
- Les champs incluent `name`, `description`, `price`, `stock`, `category` et `type`.

Classe ProductUpdateForm :
Ce formulaire est utilisé pour mettre à jour les informations d'un produit existant dans l'application.
- Il est basé sur le modèle Product.
- Les champs inclus sont les mêmes que ceux de `ProductForm`.

Classe CustomUserCreationForm :
Ce formulaire est utilisé pour créer un nouvel utilisateur personnalisé dans l'application.
- Il est basé sur le modèle CustomUser.
- Les champs incluent `username`, `email` et `password`.
- Le mot de passe est crypté avant d'être enregistré dans la base de données.

Classe LoginForm :
Ce formulaire est utilisé pour permettre aux utilisateurs de se connecter à l'application.
- Il contient des champs pour le nom d'utilisateur (`username`) et le mot de passe (`password`).

Classe FilterProduits :
Ce formulaire est utilisé pour filtrer les produits dans l'application.
- Il contient des champs pour filtrer par nom, prix, catégorie et type de produit.

Classe ContactForm :
Ce formulaire est utilisé pour que les utilisateurs puissent envoyer des messages de contact à l'administrateur de l'application.
- Il contient des champs pour le nom, l'email et le message.

Chaque formulaire est défini en utilisant les fonctionnalités de formulaire fournies par Django. Les champs sont configurés avec différents types de validation et d'attributs HTML pour une meilleure expérience utilisateur. Les formulaires peuvent être utilisés dans les vues Django pour valider et traiter les données soumises par les utilisateurs lors de l'interaction avec l'application web.

3.3 Models.py

```
Classe Product :

Cette classe représente un produit dans votre application.
Champs :
name : Un champ de texte (CharField) pour stocker le nom ou la marque du produit. La longu
description : Un champ de texte plus long (TextField) pour stocker la description du produ
price : Un champ décimal (DecimalField) pour stocker le prix du produit. max_digits défini
stock : Un champ entier (IntegerField) pour stocker la quantité de stock disponible pour l
category : Un champ de choix (CharField) pour spécifier la catégorie du produit (voiture o
type : Un champ de choix (CharField) pour spécifier le type de carburant du produit (diese
image : Un champ pour télécharger et stocker une image du produit (ImageField). Les images

Classe CustomUser (héritée de AbstractUser) :

Cette classe représente un utilisateur personnalisé dans votre application.
Champs :
email : Un champ d'email unique (EmailField) pour l'adresse e-mail de l'utilisateur.
```

Classe Cart :

Cette classe représente le panier d'un utilisateur.
Champs :
user : Une clé étrangère (ForeignKey) vers un utilisateur (CustomUser). Lorsqu'un utilisat
product : Une clé étrangère vers un produit (Product). Lorsqu'un produit est supprimé, tou
quantity : Un champ entier (PositiveIntegerField) pour stocker la quantité de ce produit d

Classe ContactMessage :

Cette classe représente un message de contact envoyé via un formulaire sur le site.
Champs :
name : Un champ de texte (CharField) pour le nom de l'expéditeur du message.
email : Un champ d'email (EmailField) pour l'adresse e-mail de l'expéditeur.
message : Un champ de texte plus long (TextField) pour le contenu du message.

3.4 Views.py

home(request) :

Cette vue renvoie la page d'accueil de l'application.
Elle rend le template acceuil.html.

contact(request) :

Cette vue gère le formulaire de contact.
Si la méthode de la requête est POST et que le formulaire est valide, elle crée un nouvel
Elle rend le template contact.html avec le formulaire.

about(request) :

Cette vue renvoie la page "À propos" de l'application.
Elle rend le template about.html.

list_products(request) :

Cette vue renvoie une liste de tous les produits disponibles dans l'application.
Elle récupère tous les objets Product depuis la base de données et les rend dans le templa

product_detail(request, id) :

Cette vue affiche les détails d'un produit spécifique.
Elle utilise la fonction get_object_or_404 pour récupérer l'objet Product correspondant à
Elle rend le template product_detail.html avec les détails du produit.

```
filterproduit(request) :

Cette vue gère le formulaire de filtrage des produits.
Si la méthode de la requête est POST et que le formulaire est valide, elle filtre les prod
Elle rend le template fproducts.html avec le formulaire et les produits filtrés.

signup_or_login(request) :

Cette vue gère à la fois l'inscription et la connexion des utilisateurs.
Si la méthode de la requête est POST et que l'utilisateur soumet le formulaire d'inscripti
Si la méthode de la requête est POST et que l'utilisateur soumet le formulaire de connexio
Elle rend le template auth.html avec les formulaires d'inscription et de connexion.

logout_view(request) :

Cette vue gère la déconnexion de l'utilisateur.
Elle déconnecte l'utilisateur et redirige vers la page d'accueil.

add_to_cart(request, product_id, quantity=1) :

Cette vue permet à l'utilisateur d'ajouter un produit à son panier.
Si l'utilisateur est connecté, elle ajoute le produit au panier de l'utilisateur avec la q
Elle redirige ensuite vers la page du panier.

get_cart(request) :

Cette vue affiche le contenu du panier de l'utilisateur.
Elle récupère tous les objets Cart associés à l'utilisateur connecté et les rend dans le t

delete_from_cart(request, product_id) :

Cette vue permet à l'utilisateur de supprimer un produit de son panier.
Elle supprime l'objet Cart correspondant au produit spécifié du panier de l'utilisateur.
Elle redirige ensuite vers la page du panier.
```

3.5 Templates

Les templates dans Django sont des fichiers HTML contenant du code HTML, du texte et des balises spéciales appelées tags de template. Ces balises permettent d'insérer dynamiquement des données provenant du serveur dans les pages HTML. Django utilise le moteur de template pour traiter ces fichiers et générer des pages web dynamiques. Les templates sont généralement stockés dans le répertoire templates de chaque application Django. Les vues Django utilisent les templates pour rendre des pages web et renvoyer des réponses aux utilisateurs. Les templates peuvent inclure des balises de contrôle de flux telles que {
Exemple :

```
<ul>
```

```
{% for product in products %}
    <li>{{ product.name }} - {{ product.price }}</li>
{% endfor %}
</ul>
```

3.6 Fichiers statiques

Les fichiers statiques dans Django sont des fichiers tels que des images, des fichiers CSS, des fichiers JavaScript, des fichiers de polices, etc., qui sont utilisés pour styliser et rendre les pages web. Contrairement aux templates, les fichiers statiques ne sont pas générés dynamiquement par le serveur. Ils sont servis directement aux utilisateurs par le serveur web. Les fichiers statiques sont généralement stockés dans le répertoire static de chaque application Django. Pour servir les fichiers statiques pendant le développement, Django fournit une vue spéciale appelée static() et un gestionnaire de fichiers statiques intégré. En production, les fichiers statiques sont généralement servis par un serveur web tel que Nginx ou Apache.
Exemple :

```
body {
    background-color: #f0f0f0;
    font-family: Arial, sans-serif;
}
```

3.7 Interface d'administration Django

Django fournit une interface d'administration intégrée qui permet aux développeurs de gérer les données de leur application sans avoir à écrire de code pour créer une interface d'administration personnalisée. L'interface d'administration Django est générée automatiquement à partir des modèles de données de l'application. Les administrateurs peuvent ajouter, modifier et supprimer des objets de la base de données à l'aide de l'interface d'administration. Pour utiliser l'interface d'administration, les développeurs doivent d'abord enregistrer leurs modèles dans le fichier admin.py de leur application en utilisant les classes ModelAdmin et admin.site.register(). L'interface d'administration peut être personnalisée en utilisant des options de configuration telles que list_display, list_filter, search_fields, etc., dans les classes ModelAdmin.
Exemple :

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

En résumé, les templates, les fichiers statiques et l'interface d'administration Django sont des éléments essentiels pour développer des applications web robustes et conviviales avec Django. Les templates permettent de créer des pages web dynamiques, les fichiers statiques ajoutent du style et de l'interactivité, tandis que l'interface d'administration simplifie la gestion des données de l'application.

Conclusion :

En résumé, une application Django pour un système de gestion de véhicules se compose de plusieurs parties intégrées : une structure de projet bien définie, des modèles pour les données, des vues pour la logique de l'application, des templates pour le rendu des pages, et des fichiers de configuration pour gérer les paramètres du projet. Les migrations facilitent la gestion des modifications de schéma de la base de données, et les tests garantissent que l'application fonctionne correctement. Tous ces composants travaillent ensemble pour créer une application web complète et fonctionnelle, capable de gérer efficacement les opérations d'une agence de vente de véhicules.