

Projet Informatique Location de Véhicules

Rapport

Auteurs :
Ilyann ARAGON
Paul RENAUD

Date de rendu : 10 mai 2025



INSTITUT
POLYTECHNIQUE
DE PARIS

ENSTA
Année universitaire 2024–2025

Table des matières

Description générale du problème	1
Introduction	1
Modélisation et fonctionnalités du projet	1
Résumé des fonctionnalités du système	2
Hypothèses réductrices	3
Description de l'application	4
Présentation générale du programme	4
Description des principales classes et méthodes	4
Application	4
Reservation_DSL	5
Classe User et héritage	5
Classe Vehicule	6
Conclusion	7
Table des figures imposées et choisies	7
Table des figures imposées	7
Table des figures choisies	9
Tests unitaires	10
Tests des utilisateurs	10
Tests des véhicules	10
Tests des réservations DSL	10
Tests des fonctions de chargement	11
Conclusion des tests	11
Limitations	11
Apports personnels	11
Présentation des résultats	12
Fonctionnalités côté client	12
Fonctionnalités côté vendeur	13
Fonctionnalités côté administrateur	13
IHM	14
Annexe	
Annexe A	
Annexe B	
Annexe C	
Annexe D	
Annexe E	

Description générale du problème

Introduction

Ce projet vise à développer un système de réservation de véhicules permettant à des clients de formuler des demandes selon des critères précis (type, volume, nombre de places, etc.) et d'obtenir une location adaptée, ou un véhicule de catégorie supérieure ou égales en cas d'indisponibilité. Le système proposera également une interface dédiée au client pour effectuer ses recherches et réservations de manière autonome.

Côté loueur, l'application permettra de gérer le parc de véhicules, suivre la disponibilité, calculer les coûts de possession, éditer des factures PDF, et analyser les performances annuelles afin d'optimiser le parc. Une interface graphique permettra au loueur de traiter les demandes, suivre l'activité, et consulter un indicateur de rentabilité (chiffre d'affaire, etc.).

Le projet sera réalisé en deux phases : une première centrée sur la logique métier, et une seconde sur les interfaces utilisateur (IHM) pour le client et le loueur.

Modélisation et fonctionnalités du projet

Pour pouvoir organiser le projet de manière simple, nous avons décidé de l'organiser de la manière suivante :

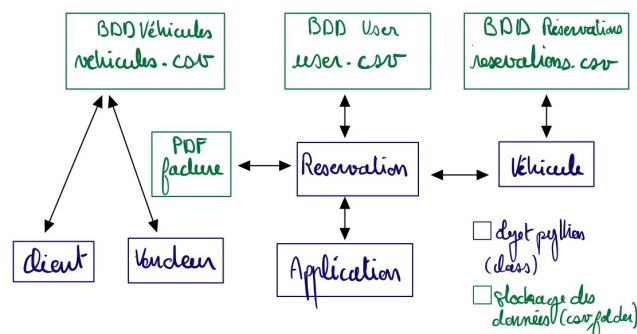


FIGURE 1 – Modélisation du projet

La figure 1 illustre l'organisation générale de notre programme. Nous avons structuré notre système autour de trois bases de données : une pour les véhicules, une pour les utilisateurs du logiciel, et une troisième dédiée aux réservations. Le choix du format CSV s'est imposé pour sa simplicité de modification en cas d'erreurs de manipulation, ainsi que pour sa facilité d'utilisation en matière de stockage et de traitement des données.

L'objet Réservation occupe une place centrale dans notre architecture : c'est celui qui interagit le plus avec les bases de données. Cependant, le cœur du fonctionnement du logiciel se trouve dans l'objet Application, qui pilote l'ensemble des opérations.

L'accès aux bases de données est restreint : les clients, par exemple, n'ont accès qu'au catalogue des véhicules, présenté de manière descriptive, sans visibilité complète sur les autres données.

Afin d'avoir une vue plus fonctionnelle des classes, de leurs méthodes et de leurs interactions, nous vous proposons le schéma suivant (Figure : 2) :

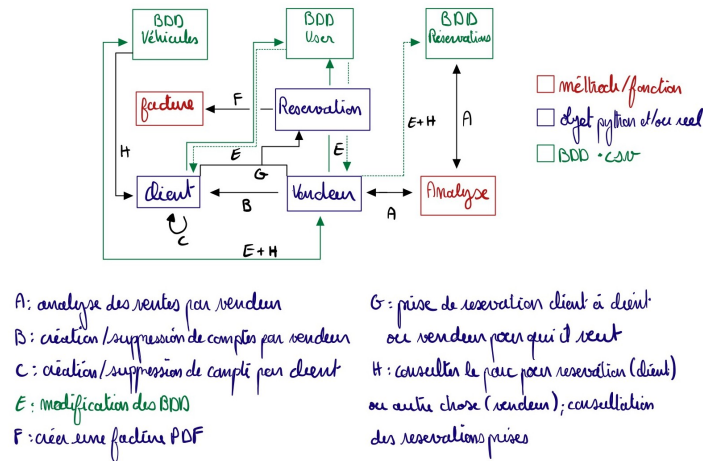


FIGURE 2 – Architecture simplifiée du système de location de véhicules

Résumé des fonctionnalités du système

Le système de location développé couvre l'ensemble du cycle de gestion, depuis la réservation jusqu'à l'analyse historique des données. Les principales fonctionnalités sont détaillées ci-dessous.

- **BDD Réservations** : Les réservations sont stockées dans un même fichier CSV pour faciliter l'analyse des ventes et le stockage.
- **Facturation et génération de documents** : Une facture PDF est générée automatiquement à partir des informations de réservation.
- **Transformation des réservations en factures complètes** : Le système intègre les données clients pour automatiser la production de factures détaillées.
- **Analyse des données sur plusieurs années** : Les performances ou tendances d'utilisation peuvent être comparées entre années via des indicateurs visuels.
- **Visualisation graphique (Matplotlib)** : Les données sont affichées sous forme de graphiques clairs facilitant l'analyse rapide.
- **Consultation du catalogue par le client** : Un moteur de recherche avancé permet de filtrer les véhicules selon les critères du client.
- **Gestion du catalogue par les vendeurs** : Les employés peuvent ajouter, supprimer ou modifier un véhicule et son statut.
- **Réservations côté client** : Les clients disposent d'un espace dédié pour effectuer eux-mêmes leurs réservations.
- **Réservations par un employé pour le compte d'un client** : Les vendeurs peuvent aussi enregistrer une réservation manuellement pour un client.
- **Ajout de clients** : Un compte peut être créé par le client lui-même ou par un employé, selon le contexte.
- **Consultation du planning de réservation** : Une vue globale des réservations est disponible pour les vendeurs via l'interface de gestion.

Dans un souci de simplicité et de stabilité, l'intégration de Google Agenda a été écartée. Bien qu'utile pour la visualisation des réservations, cette fonctionnalité impliquait des contraintes techniques importantes (gestion d'API, authentification OAuth2, dépendances externes).

Le système initial prévoyait deux fichiers de réservations : un pour l'année en cours et un autre pour les archives. Cette structure, bien que logique d'un point de vue archivistique, compliquait les recherches et comparaisons. Elle a été remplacée par une base unique contenant toutes les réservations, avec une colonne indiquant l'année.

Enfin, un rôle d'administrateur a été ajouté tardivement pour gérer la création et la suppression des comptes vendeurs directement via le fichier CSV, sans intervention sur la base de données.

Hypothèses réductrices

Dans un souci de simplicité et de cohérence, plusieurs hypothèses ont été adoptées pour cadrer le développement du projet :

- **Réservation par journée complète uniquement** : aucun horaire ni demi-journée n'est pris en compte afin de simplifier la gestion du calendrier.
- **Tarification sans prise en compte des kilomètres parcourus** : seule la durée (en jours) est utilisée pour calculer le prix, évitant ainsi la complexité liée au suivi des distances.
- **Coût d'entretien annuel fixe par véhicule** : un coût moyen annuel est attribué à chaque véhicule, sans gestion des pannes ou entretiens spécifiques, pour faciliter l'analyse de rentabilité.

Ces hypothèses ont été choisies pour assurer la faisabilité technique et pédagogique du projet tout en maintenant sa cohérence fonctionnelle.

Description de l'application

Présentation générale du programme

Notre programme est structuré en différents modules et dossiers pour les bases de données. Pour lancer notre application de location, rendez vous dans le fichier `app_ihm.py` et lancer le code. Pour vous connectez, vous pouvez utiliser les identifiants suivants (rôle vendeur) : ID : 954644716 mdp : mdp_obi-wan.

Description des principales classes et méthodes

Classe `Application`

La classe `Application` (diagramme : 7) (ou `AppIHM` dans le fichier `app_ihm.py`) représente le noyau logique et fonctionnel de notre plateforme de gestion de location de véhicules. Véritable point d'entrée du système, elle centralise toutes les interactions entre l'utilisateur et les différentes entités métier, qu'il s'agisse de la gestion des utilisateurs (clients, vendeurs et administrateurs), des réservations ou encore du catalogue de véhicules.

Conçue dans une optique de clarté et de modularité, cette classe pilote l'ensemble des fonctionnalités de la plateforme via une série de menus dynamiques, adaptés au rôle de l'utilisateur connecté. Elle gère ainsi les connexions sécurisées, la navigation dans les options disponibles, et l'accès conditionnel aux différentes actions possibles selon le profil.

Parmi les fonctionnalités offertes, on peut citer :

- La consultation, l'ajout, la suppression ou la modification de véhicules dans le parc.
- La gestion des comptes clients, y compris la création de nouveaux profils, la modification des informations ou le changement de mot de passe.
- Le processus de réservation complet, incluant la recherche de véhicules disponibles, la réservation effective, l'annulation ou encore le surclassement automatique si applicable.
- L'analyse des ventes, avec un accès à des bilans détaillés et à la génération de rapports tels que les factures.

Grâce à des méthodes bien segmentées et à une logique de navigation intuitive, la classe `Application` garantit une expérience utilisateur fluide et cohérente. Elle intègre par ailleurs des outils avancés comme la recherche multicritères, la consultation des réservations à venir par véhicule, ou encore un système de surclassement intelligent visant à optimiser l'utilisation du parc.

Enfin, sa structure orientée objet permet une maintenance facilitée et une extensibilité naturelle, rendant la plateforme adaptable aux évolutions futures, tant en termes de fonctionnalités que de volume d'utilisation.

Classe `Reservation_DSL`

La classe `Reservation_DSL` (diagramme : 8) a été développée dans le but de modéliser une réservation de véhicule tout en offrant une interface textuelle flexible grâce à l'utilisation d'un DSL (Domain Specific Language). Inspirée de la classe `Reservation`, elle étend ses fonctionnalités en introduisant une méthode de parsing automatique, permettant de transformer une chaîne de caractères textuelle structurée en une instance de réservation exploitable par le système.

Cette classe remplit deux objectifs principaux :

1. Structurer les données d'une réservation à l'aide d'attributs explicites tels que les identifiants client et véhicule, les dates de réservation, la durée, le prix, ou encore le statut de surclassement.
2. Faciliter l'importation de réservations sous forme de texte grâce à une méthode d'analyse qui interprète un DSL prédéfini.

Par exemple, une réservation peut être exprimée par la chaîne suivante :

```
RESERVATION 123456789 CLIENT 987654321 VEHICULE AB-123-CD DU 05-01-2025 AU 05-05-2025 JOURS 3 PRIX 400.00 SURCLASSEMENT True.
```

La méthode `from_dsl()` utilise des expressions régulières pour extraire les données pertinentes, puis construit automatiquement un objet `Reservation_DSL`. En cas de format incorrect, une exception informative est levée, renforçant ainsi la robustesse du système.

Outre ce mécanisme de parsing, la classe propose plusieurs méthodes utilitaires :

- `to_dict()` : convertit l'objet en dictionnaire Python pour faciliter son exportation vers un fichier CSV.
- `enregistrer()` : enregistre la réservation dans un fichier CSV, en gérant l'ajout de l'en-tête si besoin.
- `__str__()` : fournit une représentation textuelle claire de la réservation.

Un attribut spécifique de type booléen permet également d'indiquer si un surclassement a été appliqué à la réservation.

En résumé, la classe `Reservation_DSL` (diagramme : 8) allie la rigueur d'une structure orientée objet à la souplesse de l'interprétation textuelle, ce qui la rend particulièrement adaptée pour les interfaces en ligne de commande, les tests automatisés ou l'intégration avec des systèmes externes.

Classe `User` et héritage

La classe `User` (diagramme : 8) représente une entité utilisateur du système, qu'il s'agisse d'un client ou d'un vendeur. Elle centralise les informations essentielles à la gestion des utilisateurs : nom, prénom, identifiant, mot de passe, rôle et coordonnées de contact. Cette structure permet de gérer efficacement l'authentification et la personnalisation des interactions sur la plateforme.

Deux méthodes principales sont proposées :

- `to_dict()` : transforme l'objet utilisateur en dictionnaire Python, facilitant ainsi son enregistrement dans un fichier CSV ou son export vers d'autres systèmes.

- `__str__()` : fournit une représentation lisible de l'utilisateur, sous la forme "**Prénom Nom - Rôle**", utile pour les interfaces et le débogage.

La classe **User** (diagramme : 8) est conçue pour servir de classe mère à trois sous-classes spécialisées : **Client**, **Vendeur** et **Admin** (diagramme : 8). Cette structure par héritage permet de mieux séparer les responsabilités et d'introduire des comportements ou attributs spécifiques à chaque type d'utilisateur comme un pourcentage de réduction pour les vendeurs et les administrateurs par exemple.

Grâce à cette hiérarchisation (diagramme : 8), les méthodes génériques comme `to_dict()` ou `__str__()` sont mutualisées, évitant toute redondance de code et facilitant la maintenance. Cette organisation orientée objet prépare ainsi le système à une évolutivité propre et maîtrisée.

Classe **Vehicule**

La classe **Vehicule** (diagramme : 8) constitue l'un des éléments fondamentaux du système de gestion de location automobile. Elle a été conçue pour modéliser de manière complète et précise chaque véhicule présent dans le parc, en encapsulant l'ensemble des données nécessaires à sa gestion, sa réservation, et son affichage pour les clients.

Chaque instance de la classe représente un véhicule unique, identifié par une plaque d'immatriculation au format **AA-123-AA**, utilisée comme identifiant principal dans les fichiers ou bases de données. Outre cette clé d'identification, de nombreux attributs viennent décrire le véhicule de manière détaillée :

- Informations commerciales : marque, modèle, prix journalier.
- Caractéristiques techniques : masse, puissance, vitesse maximale, volume utile, nombre de places et hauteur.
- Spécificités mécaniques : type de motorisation (essence, diesel, hybride, électrique...), type de boîte de vitesses (manuelle ou automatique).
- Informations complémentaires : coût d'entretien annuel, description textuelle, disponibilité.

La classe propose un constructeur complet ainsi qu'une méthode d'affichage conviviale pour présenter les informations à l'utilisateur. Elle inclut également un ensemble de getters et setters robustes, intégrant des mécanismes de validation afin de garantir la cohérence et l'intégrité des données manipulées.

Parmi les fonctionnalités avancées, on trouve :

- `to_dict()` : une méthode de conversion automatique de l'objet en dictionnaire Python, facilitant son enregistrement dans un fichier CSV ou son intégration à une base de données.
- `set_valeur()` : une méthode générique permettant de modifier dynamiquement n'importe quel attribut de l'objet, tout en effectuant les vérifications nécessaires à la validité de la modification.

Grâce à cette structuration rigoureuse, la classe **Vehicule** offre une grande souplesse tout en assurant la fiabilité du système. Elle constitue une brique logicielle centrale dans l'architecture du programme, et joue un rôle clé dans la sélection, la réservation, et la maintenance des véhicules au sein de la plateforme.

Conclusion

Table des figures imposées et choisies

Table des figures imposées

Figure imposée : Création d'objets

Dans le cadre de notre projet, nous avons créer différents objets qui sont les suivants :

- **Vehicule** : représente un véhicule. Ce véhicule a différentes caractéristiques (masse, vitesse maximale...).
- **Client** : représente un client. Cet objet hérite d'une classe mère **User**.
- **Vendeur** : représente un vendeur. Cet objet hérite d'une classe mère **User**.
- **Admin** : représente un administrateur. Cet objet hérite d'une classe mère **User**.
- **Reservation_DSL** : représente une réservation. Une réservation a différentes caractéristiques (prix total, client, véhicule...)

Ces différents objets python représentent chacun un objet réel (ou une personne). La modélisation de ces objets réels en objet python est indispensable afin de modéliser (simplement) les interactions et les actions que chaque objet réel peut faire.

Figure imposée : Structuration du code en modules

Notre code est structuré en différents fichiers et modules schématisé ci-dessous (Figure : 3) :

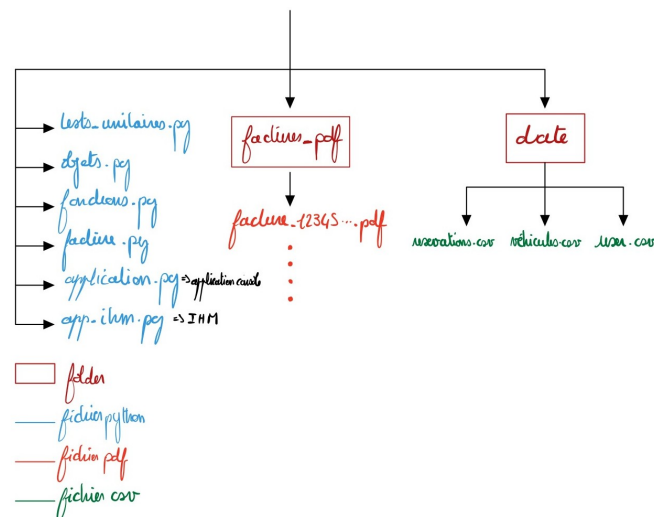


FIGURE 3 – Schéma de l'organisation des fichiers et dossiers de notre projet

- `data` : contient les fichiers `.csv`.
- `factures_pdf` : contient les factures en PDF.
- `fonctions.py` : fichier `.py` avec des fonctions utiles pour le bon fonctionnement de l'application (`load_vehicules`, `load_users_P00...`)
- `app_ihm.py` : version avec un interface homme machine (IHM) de notre application (version fonctionnelle).
- `application.py` : version sans un interface homme machine (IHM) de notre application (cette version est actuellement obsolète).
- `objects.py` : fichier `.py` avec les objets python (`Vehicule`, `User...`)
- `facture.py` : fichier contenant la fonction `facture` qui permet de générer une facture au format pdf avec des objets donnés.

Figure imposée : Héritage

Nous avons fait le choix d'utiliser l'héritage sur `User` pour modéliser les différents rôles en interaction dans notre projet (Client, Vendeur et Administrateur) car ils ont une chose en commun : ils sont tous utilisateurs de notre application. Ainsi il faut les différencier (via leur rôle), ainsi l'héritage permet de répondre à ce critère. Les utilisateurs sont donc différenciés avec une variable rôle qui prend une valeur différente en fonction du rôle de chacun. Cette variable est utilisée ensuite pour autoriser ou non des actions dans l'application.

Nous avons fait le choix de ne pas utiliser l'héritage pour les objets véhicules, car implémenter un nouveau type de véhicule aurait nécessité la création d'une nouvelle classe associée à ce type (une classe mère Véhicule et plusieurs classes filles comme Utilitaire, Camion, etc.). L'un des objectifs majeurs que nous nous étions fixés était de proposer une large gamme de véhicules disponibles à la location, tout en conservant une certaine simplicité dans la gestion du code lié aux véhicules. Ainsi, si l'utilisateur souhaite ajouter un nouveau type de véhicule, il peut le faire en modifiant simplement une liste.

Figure imposée : Documentation du code

Notre code est documenté comme demandé dans les consignes. Il est plutôt facile de comprendre le fonctionnement ou l'utilité d'une fonction, d'une classe ou d'une méthode grâce à notre documentation.

Figure imposée : Tests Unitaires

Nous avons réalisé quelques tests unitaires sur nos objets et des fonctions jugées comme importantes pour notre projet. Plus de détails dans la partie Tests unitaires.

Figure imposée : Stockage des données

Nos données sont stockées dans trois fichiers `.csv` :

- `users.csv` : contient toutes les informations (nom, prénom, mot de passe...) concernant les utilisateurs (une ligne par utilisateur).
- `vehicules.csv` : contient toutes les informations concernant les véhicules de notre parc (une ligne par véhicule).
- `reservation.csv` : contient toutes les informations concernant les réservations effectuées via notre application (une ligne par réservation).
- `factures_pdf` : ce dossier contient toutes les factures sous forme de fichier PDF générées après avoir effectué une réservation.

L'utilisation de fichiers `.csv` est conforme aux consignes, et elle facilite à la fois la vérification des attentes et le débogage.

Table des figures choisies

Figure choisie : Domain Specific Language (DSL) pour les réservations

Nous avons choisi d'implémenter un DSL (Domain Specific Language) pour représenter chaque réservation sous forme de chaîne de caractères structurée. Chaque ligne encode toutes les informations essentielles (identifiants, dates, véhicule, tarif, etc.). Ce choix permet une lecture humaine facilitée (dans le fichier `application.py` ou `appli_ihm.py`), un traitement automatique via des expressions régulières et une gestion simplifiée dans un fichier texte, sans base de données. De manière générale, le DSL permet d'exprimer des concepts métier de manière directe, compréhensible par des non-développeurs ou des experts du domaine.

Figure choisie : Fonction récursive

Méthode récursive de surclassement

La méthode récursive `surclassement` dans la classe `AppIHM` propose au client un véhicule surclassé selon ses critères. La première proposition concerne un surclassement du même type de véhicule. Si ce choix ne convient pas, la méthode s'appelle de nouveau pour offrir un autre type de véhicule. Le client peut répéter ce processus autant de fois qu'il le souhaite ou l'interrompre à tout moment. Le prix final du véhicule choisi reste inférieur ou égal à celui du véhicule initial non disponible.

Figure choisie : Gestion des utilisateurs via des comptes dédiés (permissions,...)

Nous avons décidé de créer un compte pour chaque utilisateur car cela est plus simple et cela permet au client de mieux gérer ses réservations (annulations, prise de réservations, etc). Cela permet aussi d'empêcher les clients d'avoir certaines permissions que les vendeurs et administrateurs possèdent (modification des BDD, ajout de véhicule, etc). Cette démarche permet aussi d'optimiser la création de réservation en ne récupérant uniquement les informations du véhicule (celles du client étant déjà renseignées dans son compte).

Tests unitaires

Afin d'assurer la fiabilité, la robustesse et la conformité fonctionnelle de notre application de gestion de location de véhicules, nous avons mis en place une suite de tests unitaires (Figure : 4) utilisant le module `unittest` de Python. Ces tests couvrent les principales classes et fonctions critiques du projet, à savoir : les utilisateurs (`User`, `Client`, `Vendeur`, `Admin`), les véhicules (`Vehicule`), les réservations au format DSL (`Reservation_DSL`) et les fonctions de chargement des données.

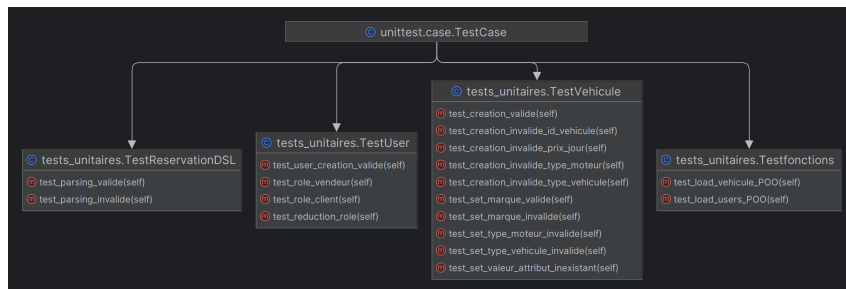


FIGURE 4 – Diagramme de classes des tests unitaires.

Tests des utilisateurs

Les tests de la classe `User` et de ses sous-classes vérifient :

- La création correcte d'un utilisateur avec des attributs valides.
- La cohérence entre le rôle ("C", "V", "A") et la classe instanciée (`Client`, `Vendeur`, `Admin`).
- Le bon fonctionnement des coefficients de réduction selon le rôle.

Tests des véhicules

Les tests de la classe `Vehicule` valident :

- La création correcte avec des paramètres valides (format de l'identifiant, types autorisés, prix, etc.).
- La détection d'erreurs lors de la création avec des paramètres invalides (prix négatif, identifiant incorrect, types non autorisés).
- Le bon fonctionnement des *setters*, notamment celui de la marque.

Tests des réservations DSL

Nous avons testé la classe `Reservation_DSL` via deux cas principaux :

- La conversion correcte d'une chaîne DSL bien formée en objet `Reservation_DSL`.
- La détection d'une erreur en cas de DSL malformée.

Tests des fonctions de chargement

Les fonctions `load_vehicules` et `load_users_P00` ont été testées afin de garantir que :

- Le chargement des données depuis les fichiers CSV s’effectue correctement.
- Les objets instanciés sont bien des instances des classes attendues (`Vehicule`, `Client`, `Vendeur` et `Admin`).

Conclusion des tests

Les tests unitaires implémentés nous ont permis de vérifier la cohérence de l’architecture orientée objet, la validité des entrées utilisateurs et l’intégrité des données manipulées. Ils constituent une base solide pour la validation du projet et pour faciliter les maintenances et évolutions futures.

Limitations

Abus possibles du système de surclassement par le client

Le client peut exploiter ses droits pour annuler ses réservations et profiter de prix bas grâce au système de surclassement. Il réserve d’abord un véhicule au tarif le plus bas, ce qui fixe le prix initial. Ensuite, il lance une nouvelle réservation pour le véhicule souhaité. Le système propose automatiquement un surclassement du même type, puis un autre sans restriction si le premier est refusé. Le client choisit finalement son véhicule réel et supprime la réservation initiale.

Apports personnels

Gestion des utilisateurs via des comptes dédiés

Bien que les consignes du projet ne spécifiaient pas la nécessité de gérer des comptes utilisateurs distincts, nous avons choisi d’introduire cette fonctionnalité pour améliorer la modélisation du système. Nous avons ainsi créé des comptes client et comptes vendeur, permettant une gestion plus fine des actions spécifiques à chaque type d’utilisateur.

Chaque utilisateur dispose maintenant d’un compte associé à ses propres attributs et comportements, ce qui permet de mieux séparer les responsabilités. Le compte client est responsable des réservations et de la gestion de ses besoins, tandis que le compte vendeur gère le parc de véhicules, les disponibilités, les réservations effectuées et il possède les mêmes fonctionnalités que le client. Le compte administrateur a les mêmes permissions que le vendeur mais il peut en plus créer et supprimer des comptes vendeur.

Cette approche permet de réduire les conditions imbriquées et d’exploiter le polymorphisme, facilitant l’évolution du système. En outre, elle améliore la modularité et la flexibilité du projet, offrant une meilleure évolutivité en cas d’ajout de nouveaux rôles comme un administrateur (rôle qui a été ajouté tardivement dans le projet).

Ainsi, même si la gestion des utilisateurs n’était pas explicitement demandée dans les consignes, cette amélioration s’inscrit dans une logique de développement de l’application tout en respectant les bonnes pratiques de la programmation orientée objet.

Vérification des variables des classes objets

Chaque classe qui représente un objet possède un bloc de plusieurs lignes vérifiant que chaque variable d'instance est dans le format et/ou du type attendu (un numéro de téléphone est au format 0102030405, une plaque d'immatriculation de véhicule (`id_vehicule`) est au format AA-123-AA)... L'implémentation de ces blocs de vérification est donc indispensable car ils permettent aussi de faire remonter tout de suite un problème critique pour le bon fonctionnement de l'application en phase de maintenance et de développement de cette dernière.

Contrôle des valeurs qui entre dans une BDD

Les utilisateurs de notre application peuvent modifier nos bases de données (fichiers `.csv`) via leurs actions. Pour la suppression de valeurs, nous avons implémenté un système de permission via l'attribution de rôle qui évite que les clients suppriment des véhicules du parc par exemple. Pour l'ajout de données, chaque valeur entrée dans un formulaire (boîte de dialogue de l'IHM...) est contrôlée par une fonction souvent encapsulée dans la méthode gérant la boîte de dialogue. Ces contrôles sont nécessaires afin d'éviter tout problème majeur dans l'application. Par exemple, pour la description d'un véhicule lors de son ajout au parc, le programme contrôle la ponctuation pour éviter l'ajout de virgules ou de ponctuation dérangeante dans les fichiers `.csv`.

Présentation des résultats

Afin d'évaluer l'efficacité de notre solution, nous avons conçu une interface utilisateur intuitive adaptée aux deux types d'utilisateurs principaux : le client et le vendeur. Chacun bénéficie d'un accès personnalisé avec des fonctionnalités spécifiques répondant à ses besoins.

Fonctionnalités côté client

Lors du lancement du programme, l'utilisateur a la possibilité de se connecter en tant que client ou de créer un nouveau compte s'il n'en possède pas. Une fois connecté, l'interface met à sa disposition plusieurs fonctionnalités essentielles :

- **Consultation du catalogue** : le client peut visualiser l'ensemble des véhicules disponibles dans la base de données.
- **Recherche ciblée** : un champ de recherche permet de retrouver rapidement un véhicule spécifique à partir de sa plaque d'immatriculation (Figure : 6).
- **Réservation de véhicule** : si le véhicule est disponible, le client peut effectuer une réservation, qui génère automatiquement une facture au format PDF.
- **Gestion des réservations** : l'utilisateur peut consulter l'historique de ses réservations, annuler une réservation en cours ou en créer une nouvelle.
- **Gestion du compte** : le client peut modifier ses informations personnelles (nom, prénom, email, téléphone, mot de passe) ou supprimer son compte définitivement.

Cette interface vise à offrir une expérience fluide, ergonomique et complète pour toute interaction client avec le système.

Fonctionnalités côté vendeur

Le mode vendeur donne accès à une interface de gestion enrichie, axée sur la supervision des données et le pilotage commercial. Les principales fonctionnalités disponibles sont les suivantes :

- **Consultation des bases de données** : accès complet au catalogue des véhicules, à la liste des utilisateurs et à l'ensemble des réservations enregistrées.
- **Gestion des véhicules** : ajout, modification ou suppression de véhicules à partir de critères prédéfinis (modèle, plaque, carburant, etc.).
- **Gestion des comptes clients** : création, suppression ou modification des comptes et des informations personnelles des utilisateurs.
- **Réservations manuelles** : possibilité de faire ou d'annuler une réservation pour un client, par exemple en cas de demande directe.
- **Suivi logistique** : consultation des prochaines réservations par véhicule afin d'anticiper leur disponibilité.
- **Analyse des ventes** : il est possible de consulter des histogrammes concernant les chiffres de l'entreprise (chiffre d'affaire par an, chiffre d'affaire par véhicule, indice de rentabilité par véhicule ...) (Figure : 11)

Fonctionnalités côté administrateur

Le mode administrateur donne accès à une interface de gestion globale de l'application. Ce mode regroupe les mêmes fonctionnalités que le mode vendeur, mais un administrateur peut aussi gérer les comptes vendeurs (création et suppression).

Analyse des performances

Une fonctionnalité avancée a été intégrée pour répondre aux besoins d'analyse financière spécifiés dans le cahier des charges. Pour éviter une exploration manuelle des bases de données, nous avons développé un module d'analyse visuelle.

Ce module permet de générer des graphiques et histogrammes dynamiques facilitant l'extraction d'indicateurs clés tels que :

- la rentabilité individuelle de chaque véhicule,
- le bénéfice total généré par la flotte,
- l'évolution des bénéfices sur une période définie, ventilée par véhicule.

Grâce à cette représentation graphique, le vendeur peut réaliser un bilan financier clair et complet, tout en identifiant rapidement les évolutions de performance (pertes ou gains). Cette visualisation rend la prise de décision plus efficace et évite de manipuler manuellement les données brutes.

En résumé, les résultats obtenus montrent que notre interface permet une exploitation complète et intuitive du système, que ce soit pour les clients ou pour les vendeurs, tout en respectant les objectifs fonctionnels fixés initialement.

Interface Homme-Machine (IHM)

Présentation générale de l'IHM

Conformément au cahier des charges, nous avons développé une interface homme-machine afin de faciliter l'utilisation de notre application. Pour cela, nous avons choisi d'utiliser le module `PyQt5`, qui offre une riche bibliothèque de composants graphiques tels que des boutons, des champs de texte, des boîtes de dialogue, etc. Ces outils nous étaient familiers grâce aux travaux pratiques réalisés en informatique.

La conception de l'interface suit la même logique que celle de notre classe `Application` : un système de connexion sécurisé avec identifiant, permettant d'authentifier soit un client (interface client 9), soit un vendeur (interface vendeur 10). Chaque type d'utilisateur dispose alors de fonctionnalités spécifiques adaptées à ses droits.

L'utilité de cette interface se manifeste notamment dans les fonctions d'analyse des ventes, qui permettent aux utilisateurs de visualiser facilement les données commerciales (Figure : 11).

Par ailleurs, la réservation d'un véhicule est simplifiée grâce à un calendrier interactif, facilitant la sélection des dates (Figure : 5). De plus, une fenêtre déroulante permet de choisir le client concerné lorsque la réservation est effectuée par un vendeur (Figure : 5), rendant le processus intuitif et rapide.

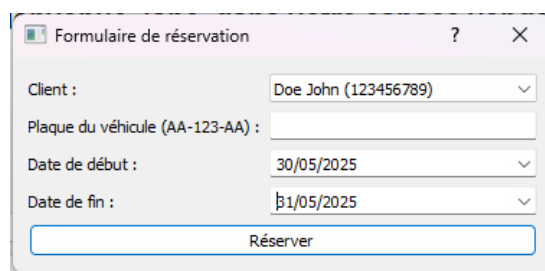
The image shows a software window titled "Formulaire de réservation" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are four labeled input fields arranged vertically: "Client :" with a dropdown menu showing "Doe John (123456789)", "Plaque du véhicule (AA-123-AA) :" with a text input field, "Date de début :" with a date picker showing "30/05/2025", and "Date de fin :" with a date picker showing "31/05/2025". At the bottom of the form is a wide button labeled "Réserver".

FIGURE 5 – Interface pour réserver un véhicule

De plus, l'interface permet aux clients comme aux vendeurs d'accéder directement aux bases de données, facilitant ainsi la consultation et la gestion des informations. L'utilisation de `PyQt5` offre un affichage plus structuré et agréable visuellement, bien plus organisé et intuitif qu'un simple fichier CSV, qui correspond au format brut de nos bases de données.

Fonctionnalité de surclassement et recherche de véhicule

Une autre fonctionnalité importante de l'IHM est celle du **surclassement de véhicule**, étroitement liée à la recherche de véhicules. Cette option s'active automatiquement lorsque le véhicule initialement souhaité par le client est déjà réservé. Dans ce cas, une nouvelle interface dédiée apparaît (voir Figure 6), proposant au client de rechercher une alternative de surclassement.

Cette interface permet tout d'abord de sélectionner un **type de véhicule** via une liste déroulante (par exemple : berline, SUV, citadine, etc.). Pour affiner davantage la recherche, l'utilisateur peut ensuite ajouter un ou plusieurs **critères de surclassement** afin d'orienter la proposition vers un véhicule équivalent ou supérieur à celui initialement envisagé. Parmi les critères disponibles, on peut citer : *vitesse maximale supérieure à 250 km/h*, *marque*, *modèle*, ou encore *prix journalier maximum*.

Ce système permet d'offrir une solution de remplacement intelligente et personnalisée, tout en garantissant une expérience fluide et satisfaisante pour le client.

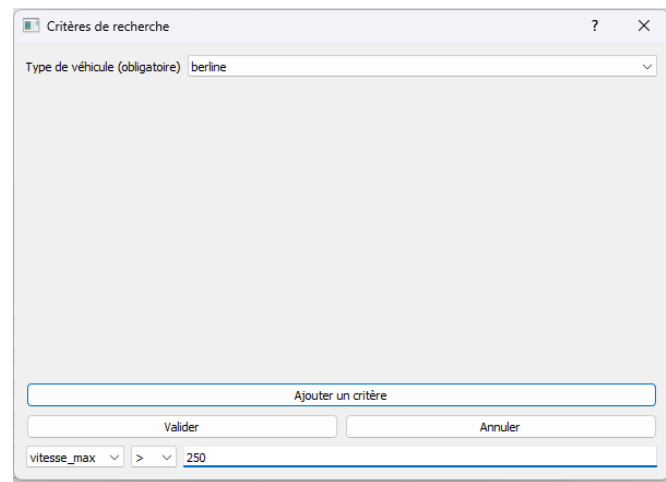


FIGURE 6 – Interface de sélection du type de véhicule et du critère de surclassement

Annexe

Annexe A

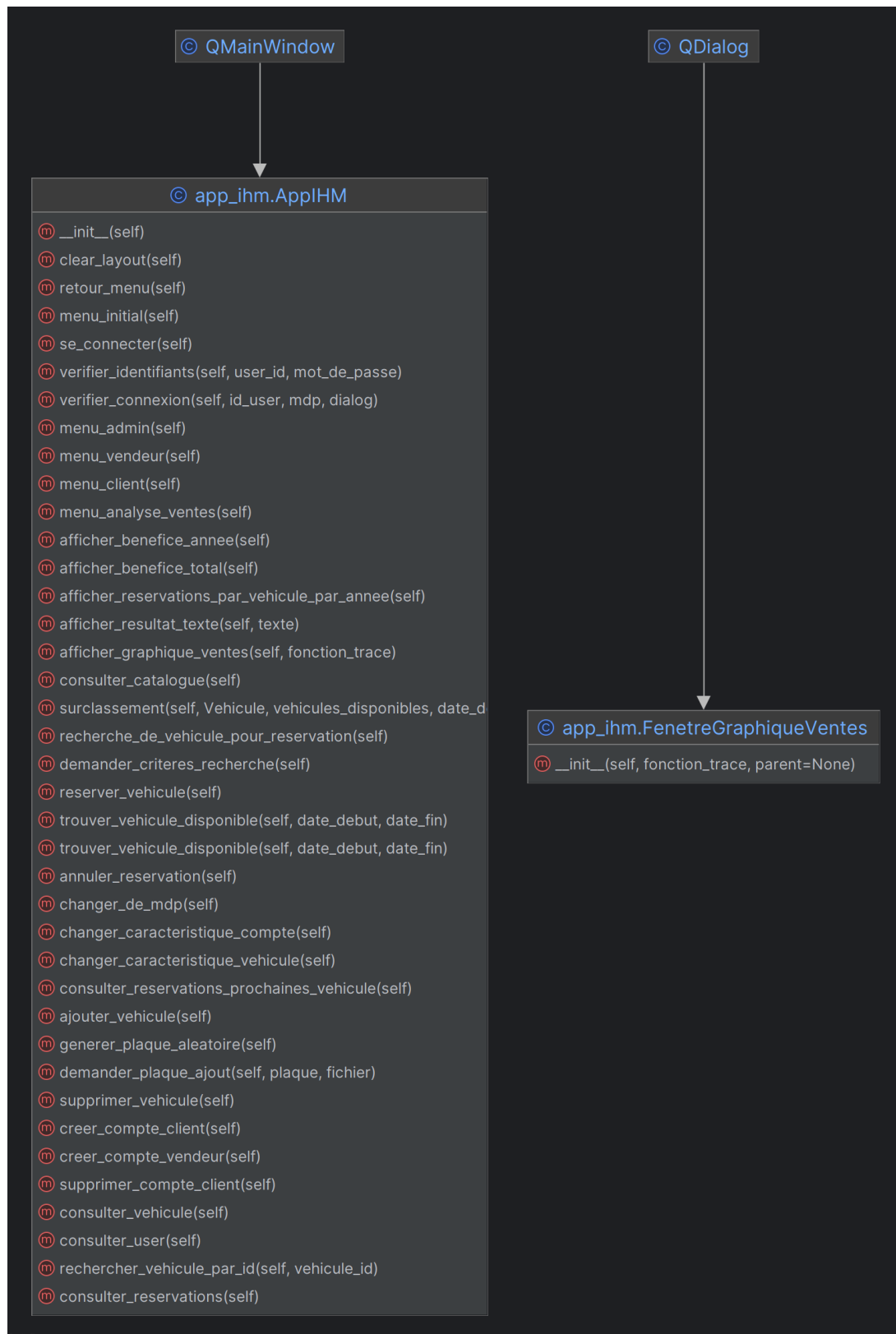


FIGURE 7 – Diagramme de classes de l'Application.

Annexe B

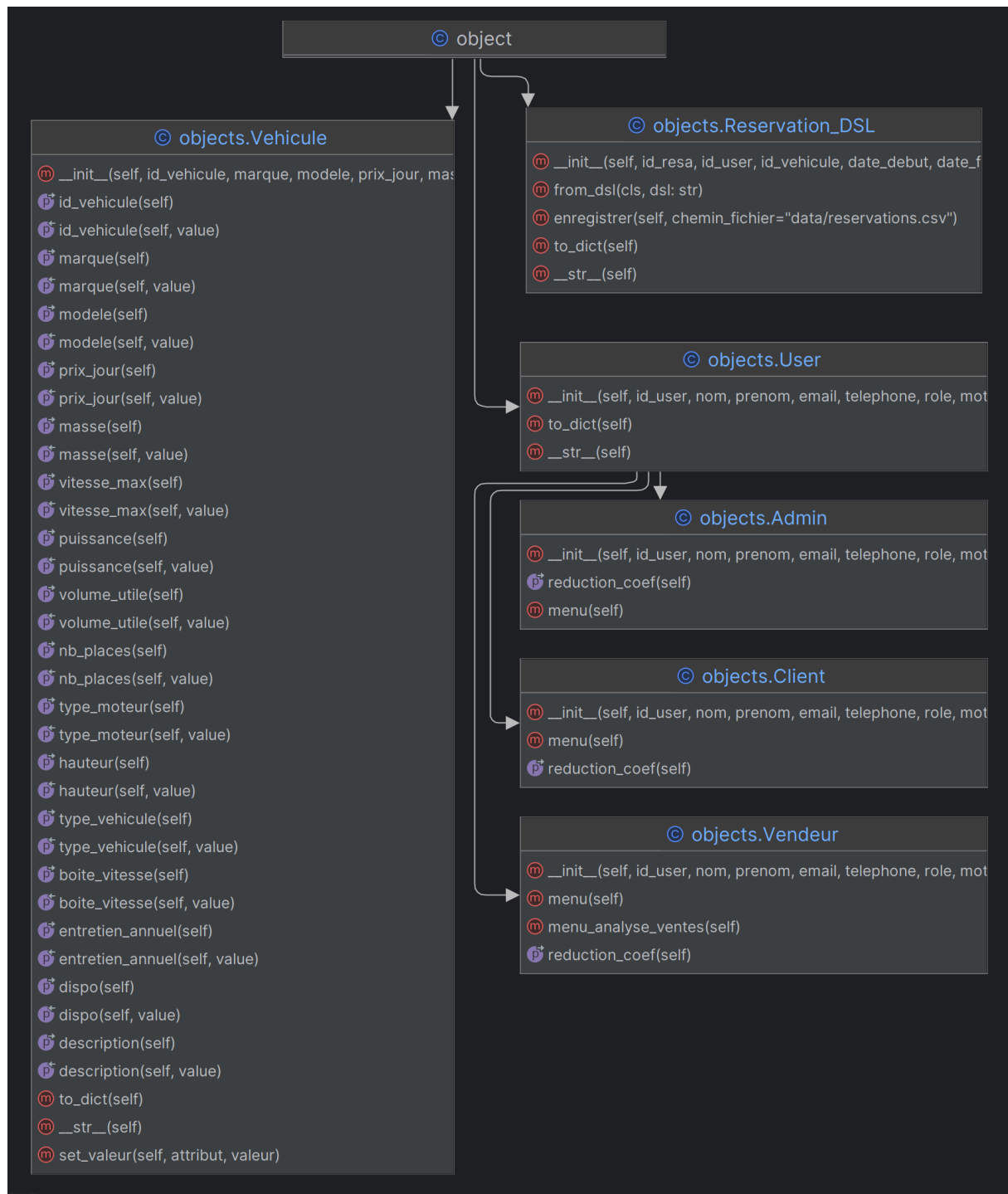


FIGURE 8 – Diagramme de classes des objets : User, Vendeur, Client, Admin, Reservation_DSL et Vehicule.

Annexe C

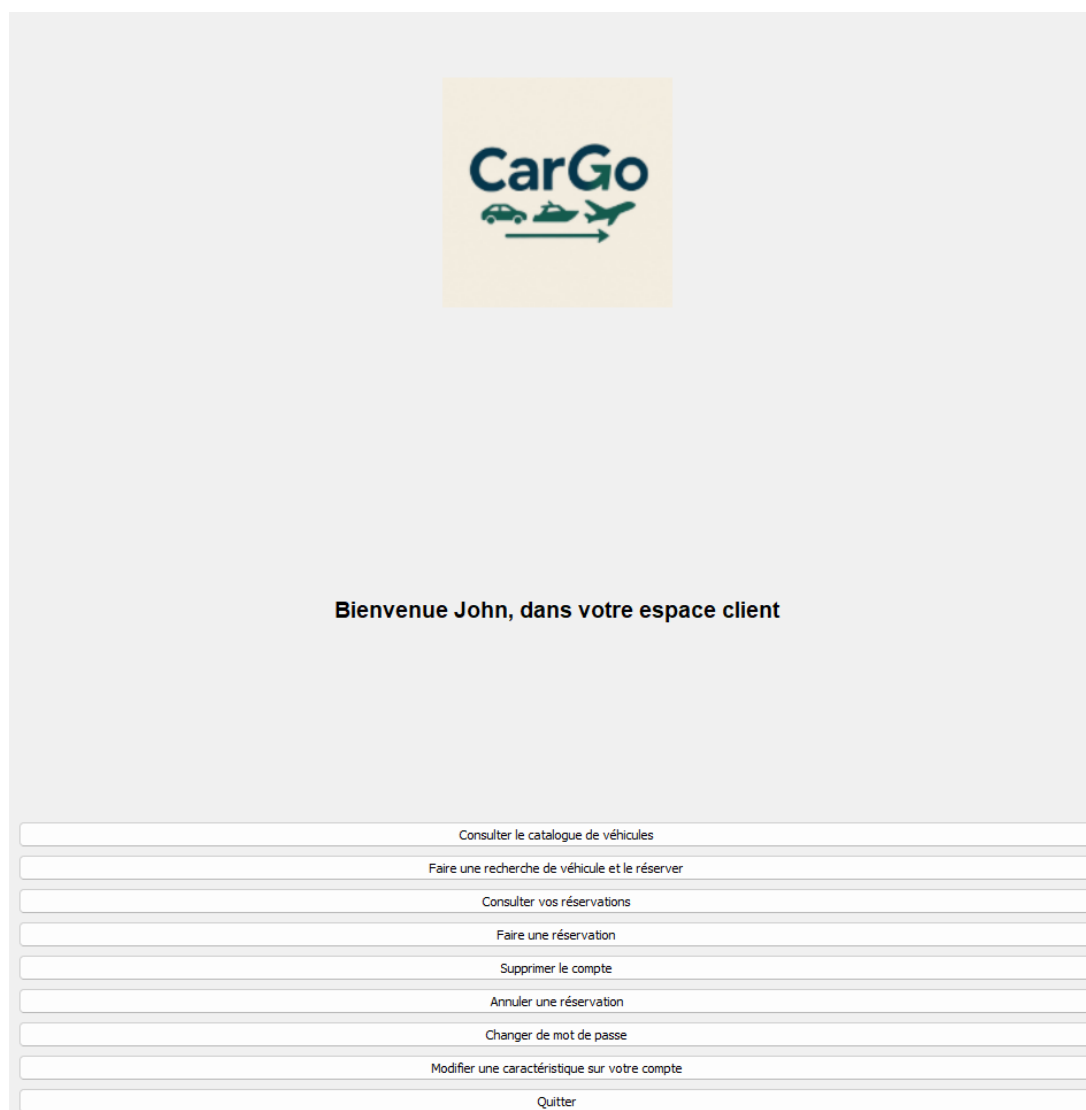


FIGURE 9 – Interface de l'espace client

Annexe D

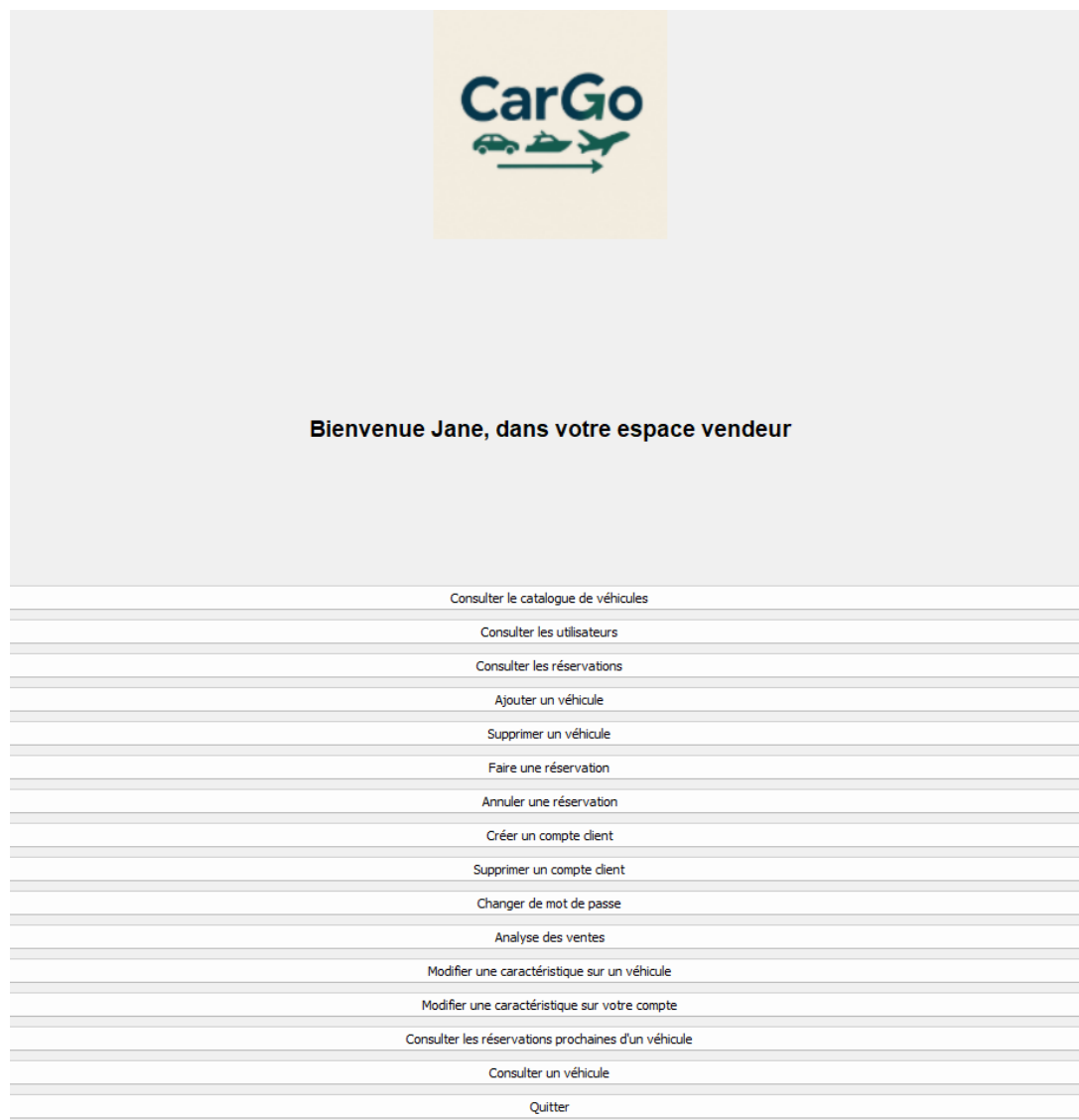


FIGURE 10 – Interface de l'espace vendeur

Annexe E

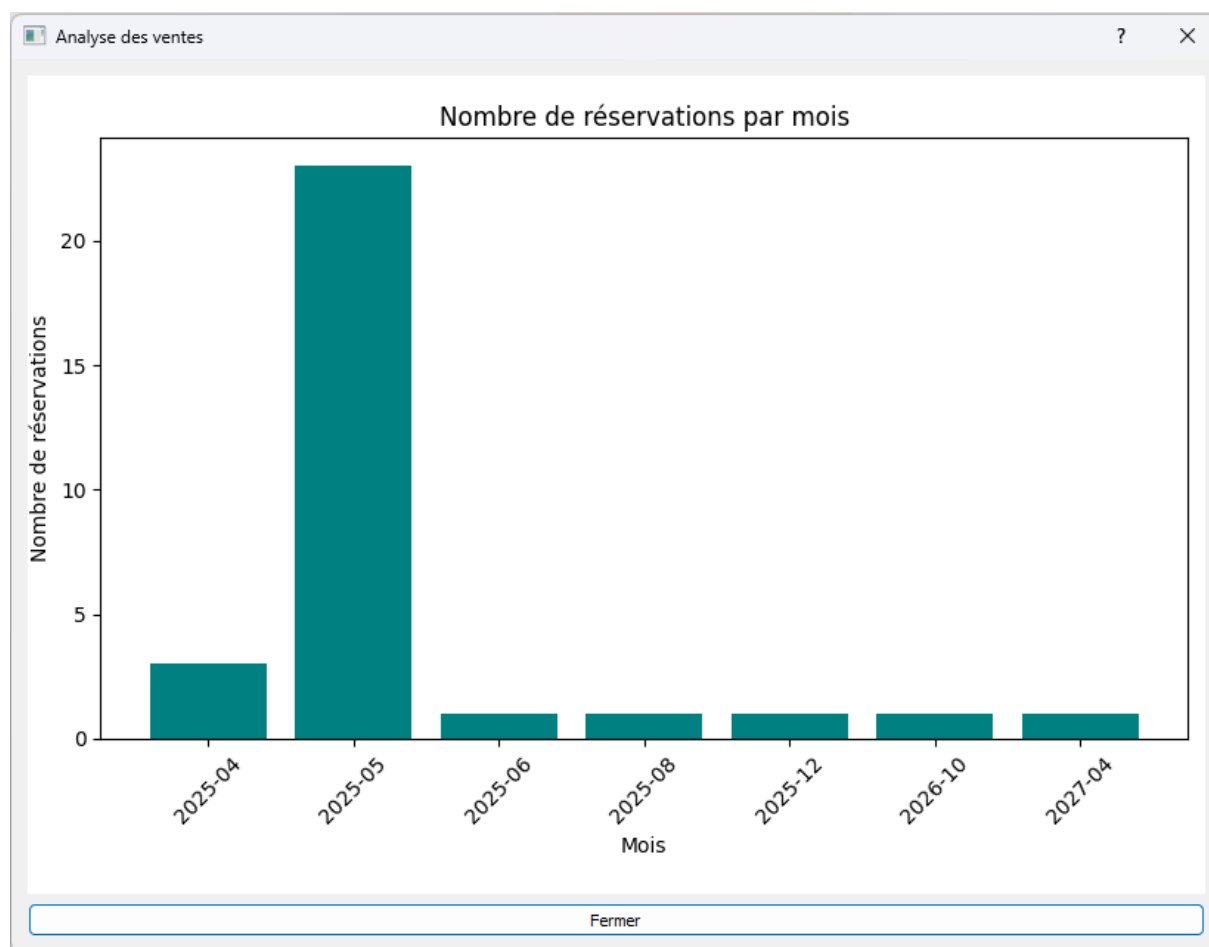


FIGURE 11 – Interface d'analyse des ventes