A white sign with green text

Description automatically generated

**Rapport de projet**

**3ème année**

**Ingénierie Informatique et Réseaux**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gestion de la flotte de véhicules

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Réalisé par :**

Abdelkhalek AIT-IDIR & Aymane ADIDI

**Encadré par :**

Tuteur de l’école : Dr. Mariame AMINE

**Introduction**

**Contexte général**

Le projet de gestion de flotte de véhicules vise à optimiser la gestion et l'utilisation des véhicules au sein d'une organisation. Ce système aide à suivre la disponibilité des véhicules, leurs réservations par les employés, et leur kilométrage. De plus, il permet une gestion efficace des véhicules électriques en prenant en compte leur autonomie.

**Introduction**

Le projet répond au besoin de digitalisation de la gestion des véhicules dans une organisation. Avec une interface simple et conviviale, il permet de réduire les erreurs humaines et d'améliorer l'efficacité.

**Objectifs du projet**

* Offrir un système pour réserver des véhicules.
* Gérer les détails des véhicules, y compris leur kilométrage et disponibilité.
* Suivre les réservations effectuées par les employés.
* Intégrer la gestion des véhicules électriques.

**Technologies utilisées**

Le projet est entièrement réalisé en **C++**, avec l'utilisation des bibliothèques standard (par exemple, **iostream**, **vector**, et **string**).

**Conclusion**

Ce système répond aux besoins organisationnels pour une gestion simplifiée et efficace des ressources véhiculaires.

**Analyse des besoins**

**Introduction**

L'analyse des besoins est essentielle pour comprendre et définir les fonctionnalités attendues.

**Exigences fonctionnelles**

* Permettre l'ajout de nouveaux véhicules à la flotte.
* Enregistrer les employés et leurs réservations de véhicules.
* Mettre à jour les kilométrages des véhicules après usage.
* Afficher les véhicules disponibles et la liste des réservations.

**Exigences non fonctionnelles**

* **Facilité d'utilisation** : Interface intuitive et simple d'interaction.
* **Performance** : Traitement rapide des réservations et des mises à jour.
* **Maintenabilité** : Code clair, modulaire, et évolutif.

**Conclusion**

Les exigences fonctionnelles et non fonctionnelles garantissent un système répondant aux attentes des utilisateurs tout en assurant une gestion optimale.

**Conception du système**

**Introduction**

La conception inclut des modèles pour chaque composant du système, assurant une implémentation organisée et compréhensible.

**Modélisation des classes et objets**

**Classes principales**

1. **Vehicule**
   * **Attributs** : Marque, modèle, kilométrage, disponibilité.
   * **Méthodes** : getMarque(), getModele(), setKilometrage(), afficherDetails().
2. **VehiculeElectrique** (Hérite de Vehicule)
   * **Attributs supéplémentaires** : Autonomie.
   * **Méthodes** : afficherAutonomie().
3. **Employe**
   * **Attributs** : Nom, liste des véhicules réservés.
   * **Méthodes** : reserverVehicule(), afficherReservations().
4. **Flotte**
   * **Attributs** : Liste des véhicules et des employés.
   * **Méthodes** : ajouterVehicule(), ajouterEmploye(), reserverVehicule().

**Schéma relationnel (MCD simplifié)**

Employe (ID\_Employe, Nom)

Vehicule (ID\_Vehicule, Marque, Modele, Kilometrage, Disponible)

VehiculeElectrique (ID\_Vehicule [FK], Autonomie)

Flotte (ID\_Reservation, Date\_Reservation, ID\_Employe [FK], ID\_Vehicule [FK])

**Diagramme des Cas d'Utilisation (UML)**

Voici les interactions principales entre les acteurs et les fonctionnalités :

**Acteurs :**

* **Employé :**
  + Réserver un véhicule.
  + Consulter les véhicules disponibles.
* **Administrateur :**
  + Ajouter un véhicule.
  + Mettre à jour l'état d'un véhicule.
  + Visualiser les réservations.
  + Mettre à jour le kilométrage des véhicules.

**Cas d'utilisation :**

* Association entre chaque acteur et ses fonctionnalités directes (réserve, ajout, mise à jour).

**Surcharge des opérateurs**

L'opérateur << a été surchargé pour afficher les détails d'un véhicule de manière concise.

**Conclusion**

La conception modulaire du système facilite l'implémentation et l'extensibilité future.

**Détails d'implémentation**

**Introduction**

L'implémentation traduit les concepts de conception en un code fonctionnel et performant.

**Code source**

Voici un extrait de chaque classe :

**Classe Vehicule**

A screen shot of a computer program

Description automatically generated

**Classe VehiculeElectrique**

A computer code on a black background

Description automatically generated

**Classe Employe**

A computer screen shot of text

Description automatically generated

**Classe Flotte**

A screen shot of a computer program

Description automatically generated

**Description de l’implémentation**

* **Structures de données** : std::vector pour gérer dynamiquement les listes de véhicules et d'employés.
* **Encapsulation** : Utilisation d'attributs privés pour garantir la protection des données.
* **Surcharge d’opérateur** : Amélioration de la lisibilité des sorties en formatant les détails des véhicules.

**Défis rencontrés et solutions**

1. **Vérification des disponibilités** : Utilisation d'attributs booléens pour indiquer l'état du véhicule.
2. **Gestion des réservations conflictuelles** : Une boucle parcourt les employés pour éviter les doublons.

**Conclusion**

L'implémentation respecte les bonnes pratiques de codage et répond aux besoins spécifiés.

**Tests et validation**

**Introduction**

Les tests vérifient si le système fonctionne correctement dans des cas d'utilisation réels.

**Scénarios de test**

1. Ajout de véhicules et affichage de la flotte.
2. Réservation d'un véhicule disponible.
3. Mise à jour du kilométrage et vérification.

**Résultats des tests**

Tous les tests ont été concluants :

* Les véhicules ont été ajoutés correctement.
* Les réservations ont été enregistrées sans conflits.
* La mise à jour du kilométrage fonctionne comme prévu.

**Conclusion**

Les tests confirment que le programme fonctionne comme attendu.

**Conclusion & Perspectives**

**Bilan du projet**

Le projet répond aux besoins de gestion de flotte en offrant des fonctionnalités pratiques et efficaces.

**Limites du projet**

* Les objets Vehicule sont stockés directement dans un vector, ce qui empêche d'exploiter le polymorphisme.
* les objets sont manipulés statiquement sans différenciation par type
* La classe Flotte ne permet pas de libère correctement la mémoire allouée pour éviter les fuites.
* les employés sont stockés dans un vector, ce qui nécessite une recherche séquentielle.

**Perspectives d’amélioration**

* Utilisation des pointeurs pour stocker les véhicules dans la classe Flotte, ce qui permet de gérer des objets polymorphiques (Vehicule et VehiculeElectrique) dans un conteneur unique.
* implémentation de polymorphisme avec la méthode virtuelle afficherDetails et la méthode getType, permettant un comportement spécifique pour les véhicules électriques (VehiculeElectrique) tout en traitant tous les véhicules via des pointeurs de type Vehicule.
* Destructeur : La classe Flotte dispose d’un destructeur qui libère correctement la mémoire allouée pour éviter les fuites.
* Les employés sont stockés dans un map avec leur nom comme clé. Cela permet un accès rapide et efficace aux employés pour gérer leurs réservations.
* Une méthode genererRapport qui fournit un aperçu des statistiques globales de la flotte :
* Nombre total de véhicules.
* Nombre de véhicules disponibles.
* Répartition par type de véhicule.

**Annexes**

**Code source complet**

Le code source est fourni dans le fichier attaché.

Voici un extrait de la surcharge d'opérateur :

A computer screen with text

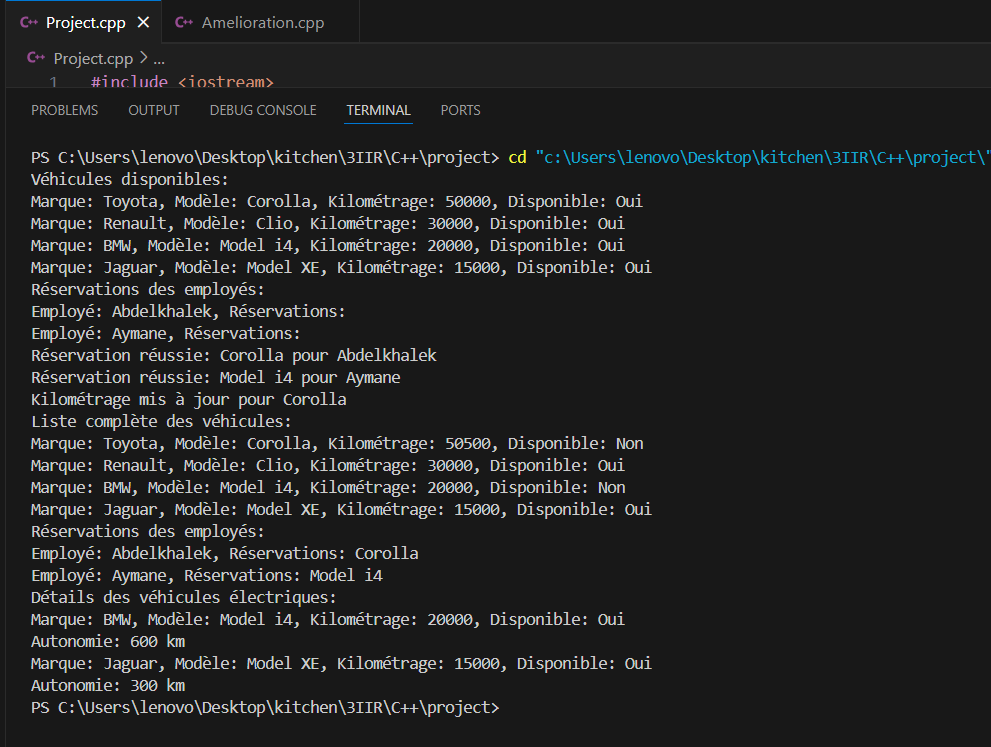
Description automatically generated

Voici un extrait de la Mis à jour de kilométrage du véhicule après usage :

A screen shot of a computer code

Description automatically generated

**L’exécution du code source**

****

**Code source après l’amélioration**

Le code source après l’amélioration est fourni dans le fichier attaché.

Voici un extrait de la méthode genererRapport:

**A screen shot of a computer program

Description automatically generated**

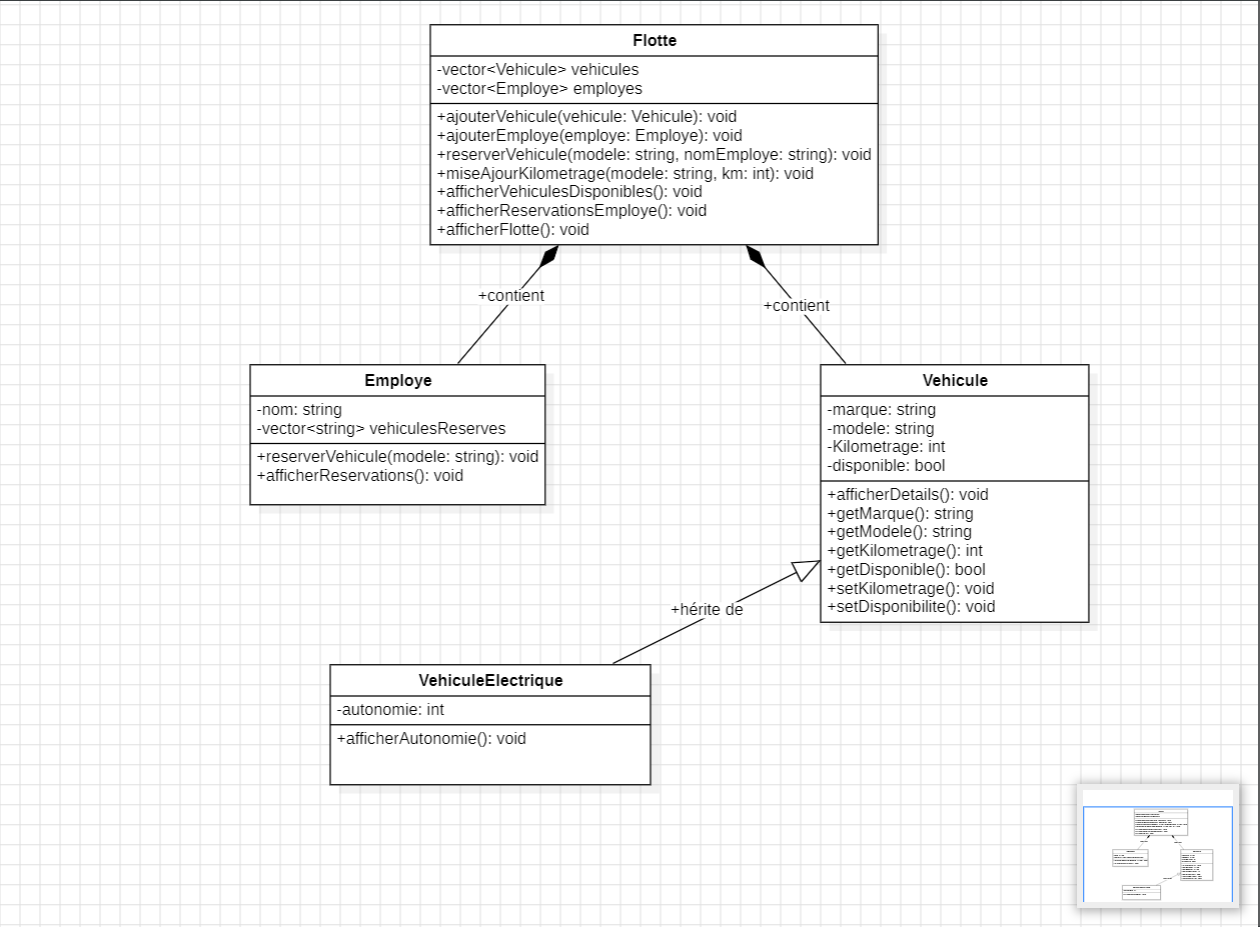
**L’exécution du code source après l’amélioration**

**A screenshot of a computer program

Description automatically generated**

**Diagrammes UML**

Le diagramme UML représentant les relations entre les classes :



**Autres ressources**

* Documentation C++ pour les bibliothèques standards utilisées.

Nous avons utilisé les bibliothèques standard de C++ suivantes :

* **iostream** : Pour les entrées/sorties standard, telles que std::cout et std::cin, permettant l'affichage des informations et la lecture des données saisies.
* **vector** : Une structure de données de la bibliothèque standard utilisée pour gérer dynamiquement des collections, comme la liste des véhicules ou des employés.
* **string** : Fournit la gestion des chaînes de caractères, utilisée pour stocker des informations comme les noms des employés, les marques et modèles de véhicules.