

**LICENCE PROFESSIONNELLE**

Projet de Fin d'Études

Filière : Intelligence Artificielle

Thème : LOCALISER ET CONTROLER LE DEMARRAGE ET L'ARRET DE SON VEHICULE A DISTANCE

**Année Universitaire 2024/2025**

**Rédigé et présenté par :**

*Hamza El ASLY : 758*

**Encadré par :**

*Mr. Adnane TALHA*

# Remerciements

Je remercie mon encadreur

Mr. Adnane TAHA pour son aide, son orientation et ses conseils durant l’accomplissement du projet. Tous mes infinis remerciements aux membres du jury d’avoir accepté de juger ce travail de fin d’études. Je remercie également tous les enseignants qui ont contribué à notre formation durant notre cursus SUP'MANAGEMENT.

Mes remerciements les plus chaleureux vont à mes chers parents pour leurs encouragements, leur patience et leur grand soutien durant toutes ces années d’études. Enfin, je remercie tous ceux qui ont participé de près ou de loin à la réalisation de ce mémoire.

Table de Matière

[Remerciements 1](#_Toc204019950)

[I. Introduction 5](#_Toc204019951)

[1. Objectifs du projet 5](#_Toc204019952)

[2. Contexte et enjeux de la télématique automobile 5](#_Toc204019953)

[3. Problématique du contrôle et de la surveillance à distance 6](#_Toc204019954)

[II. Cahier des charges 6](#_Toc204019955)

[1. Contexte 6](#_Toc204019956)

[2. Objectifs spécifiques 6](#_Toc204019957)

[3. Architecture du système 7](#_Toc204019958)

[4. Planning prévisionnel sur 4 semaines 10](#_Toc204019959)

[5. Livrables attendus 10](#_Toc204019960)

[III. Fonctionnalités implémentées 11](#_Toc204019961)

[1. Interface de connexion et authentification 11](#_Toc204019962)

[2. Contrôle à distance : verrouillage et déverrouillage 11](#_Toc204019963)

[3. Contrôle moteur : démarrage et arrêt à distance 12](#_Toc204019964)

[4. Géolocalisation via GPS 12](#_Toc204019965)

[5. Système de logs et historique des activités 13](#_Toc204019966)

[6. Prédiction de pannes par intelligence artificielle 14](#_Toc204019967)

[7. Réaction d'urgence et alertes 14](#_Toc204019968)

[IV. Modélisation IA 15](#_Toc204019969)

[1. Génération et structure du jeu de données 15](#_Toc204019970)

[2. Stratégie de labellisation 16](#_Toc204019971)

[3. Entraînement du modèle Random Forest 17](#_Toc204019972)

[4. Validation et métriques de performance 18](#_Toc204019973)

[5. Sérialisation et persistance du modèle 19](#_Toc204019974)

[6. Intégration temps réel dans l'interface 19](#_Toc204019975)

[V. Diagrammes 20](#_Toc204019976)

[1. Diagramme de Cas d’utilisation : 20](#_Toc204019977)

[2. Relations entre les cas d'utilisation : 21](#_Toc204019978)

[3. Diagramme de Classes : 22](#_Toc204019979)

[VI. Stockage et analyse des données 25](#_Toc204019980)

[VII. Réalisation technique 26](#_Toc204019981)

[1. Vue du tableau de bord (Dashboard) : 26](#_Toc204019982)

[3. Formulaire d’ajout de véhicule : 27](#_Toc204019984)

[4. Historique des logs : 29](#_Toc204019985)

[VIII. Perspectives d'amélioration 30](#_Toc204019986)

[1. Système d'alertes et notifications temps réel 30](#_Toc204019987)

[2. Évolution vers des modèles d'IA avancés 30](#_Toc204019988)

[3. Renforcement de la sécurité 31](#_Toc204019989)

[4. Scalabilité et architecture distribuée 32](#_Toc204019990)

[IX. Conclusion 33](#_Toc204019991)

[X. Glossaire 34](#_Toc204019992)

Table de figures

|  |  |
| --- | --- |
| Figure 1 : Diagram de Cas d'utilisation | 23 |
| Figure 2 : Diagram de Classes | 26 |
| Figure 3 : Extrait du modèle ORM pour les logs | 27 |
| Figure 4: Interface principale (Dashboard) | 28 |
| Figure 5 : Vue multi-véhicules dans application | 29 |
| Figure 6 : Formulaire d’ajout de véhicule | 29 |
| Figure 7 : Historique des logs | 30 |

# Introduction

Dans un monde en constante évolution technologique, la gestion intelligente des véhicules devient une priorité. Notre projet s'inscrit dans cette dynamique en proposant une application logicielle capable de localiser un véhicule et de contrôler son démarrage et arrêt à distance. Ce projet est conçu dans un cadre purement logiciel, sans intégration de matériel physique, et s’appuie sur les compétences acquises en Intelligence Artificielle, traitement de données et interfaces utilisateur.

## Objectifs du projet

Ce projet vise à développer une solution intégrée de suivi et de contrôle de véhicule à distance, enrichie par des capacités d'intelligence artificielle pour la prédiction de pannes. Les objectifs spécifiques sont :

* 1. **Objectif principal** :

Concevoir et implémenter un système télématique capable de surveiller, contrôler et prédire l'état d'un véhicule à distance.

* 1. **Objectifs secondaires** :
* Développer une interface web intuitive pour la gestion du véhicule
* Implémenter un système de contrôle à distance (verrouillage, démarrage, arrêt)
* Intégrer un module de géolocalisation et de cartographie
* Développer un modèle d'IA pour la prédiction de pannes
* Créer un système de logs et d'historique des activités
* Valider le fonctionnement par simulation et tests

## Contexte et enjeux de la télématique automobile

La télématique automobile représente aujourd'hui un secteur en pleine expansion, combinant les technologies de télécommunication et d'informatique pour révolutionner l'expérience de conduite. Dans un contexte où les véhicules deviennent de plus en plus connectés, les constructeurs automobiles et les gestionnaires de flottes recherchent des solutions innovantes pour optimiser la surveillance, la maintenance et la sécurité des véhicules.

L'essor de l'Internet des Objets (IoT) et des technologies de communication 4G/5G a permis l'émergence de systèmes télématiques sophistiqués, capables de collecter et d'analyser en temps réel une multitude de données véhiculaires. Ces systèmes offrent des perspectives nouvelles en matière de maintenance prédictive, de gestion de flotte, et d'amélioration de la sécurité routière.

## Problématique du contrôle et de la surveillance à distance

La gestion moderne des véhicules, qu'il s'agisse de flottes professionnelles ou de véhicules particuliers, fait face à plusieurs défis majeurs :

**Maintenance réactive vs préventive** : Les approches traditionnelles de maintenance se basent sur des intervalles fixes ou sur l'apparition de pannes, engendrant des coûts élevés et des immobilisations non planifiées.

**Manque de visibilité en temps réel : L'absence d'informations précises sur l'état du** véhicule et sa localisation limite la capacité d'intervention rapide en cas de problème.

**Sécurité et vol de véhicules** : Le vol de véhicules représente un enjeu économique majeur, nécessitant des systèmes de contrôle et de localisation efficaces.

**Optimisation opérationnelle** : La gestion d'une flotte de véhicules requiert une coordination précise pour maximiser l'utilisation et minimiser les coûts d'exploitation.

# Cahier des charges

## Contexte

Ce projet s'inscrit dans le cadre d’un travail pratique en intelligence artificielle et Big Data, **sans dépendance matérielle réelle**. Il est développé uniquement en environnement logiciel à des fins de simulation. L’enjeu est de **valider un concept de télématique intelligente** à travers des outils modernes, et de mettre en avant les compétences techniques des étudiants dans :

* La **programmation Python**,
* La **création d’interfaces interactives**,
* Le **traitement, stockage et analyse des données**,
* Et l’utilisation des outils de visualisation et de prédiction.

## Objectifs spécifiques

Les objectifs opérationnels du projet sont les suivants :

**Simuler la position GPS du véhicule** de manière automatique et réaliste (latitude / longitude évoluant avec le temps).

**Développer une interface graphique** intégrant une carte interactive avec les boutons de commande.

**Implémenter les commandes de démarrage et d'arrêt du moteur**, accessibles à distance via l'interface.

**Sauvegarder les coordonnées GPS** et l’état du véhicule dans une base de données locale.

**Analyser les déplacements du véhicule**, notamment :

* + les distances parcourues,
  + les zones les plus fréquentées,
  + les plages horaires d’utilisation.

Intégrer une **fonction de prédiction des mouvements futurs** en se basant sur un modèle simple.

## Architecture du système

L’architecture générale du système repose sur trois couches principales :

***Présentation (Frontend), Traitement (Backend)*** & ***Stockage (Base de données).***

* 1. **Description des modules principaux**

**Module Simulateur de Capteurs** : Ce module génère des données synthétiques mimant le comportement de capteurs véhiculaires réels. Il simule :

* Température moteur avec variations réalistes et anomalies
* Niveau de carburant avec consommation progressive
* Vitesse du véhicule avec accélérations et freinages
* Position GPS avec trajectoires cohérentes
* Régime moteur corrélé à la vitesse et à la charge

**Module API REST Flask** : Constitue le cœur applicatif du système, exposant des endpoints RESTful pour :

* /api/vehicle/status : Récupération de l'état du véhicule
* /api/vehicle/control : Envoi de commandes (lock/unlock, start/stop)
* /api/logs : Consultation de l'historique des actions
* /api/predict : Déclenchement des prédictions IA
* /api/location : Services de géolocalisation

**Module de Prédiction IA** : Intègre le modèle Random Forest entraîné pour analyser les données de capteurs et prédire les probabilités de panne. Le module inclut :

* Préprocessing des données en temps réel
* Inférence via le modèle pré-entraîné
* Post-traitement et interprétation des résultats
* Gestion des alertes basées sur les seuils de probabilité

**Module Interface Web** : Interface utilisateur responsive développée en HTML5/CSS3/JavaScript, intégrant :

***HTML***

est un langage informatique utilisé sur l'internet. Ce langage est utilisé pour créer des pages web. L'acronyme signifie ***Hyper Text Markup Langage***, ce qui signifie en français "langage de balisage d'hypertexte". Cette signification porte bien son nom puisqu'effectivement ce langage permet de réaliser de l'hypertexte à base d'une structure de balisage.

***CSS & Bootstrap :***

Les CSS, ***Cascading Style Sheets*** (feuilles de styles en cascade), servent à mettre en forme des documents web, type page HTML ou XML. Par l'intermédiaire de propriétés d'apparence (couleurs, bordures, polices, etc.)

***JavaScript :***

*JavaScript* ***(« JS » en abrégé)*** *est un langage de programmation dynamique complet qui, appliqué à un document HTML, peut fournir une interactivité dynamique sur les sites Web.*

* Tableau de bord temps réel des métriques véhiculaires
* Contrôles interactifs pour les commandes à distance
* Cartographie dynamique via Leaflet
* Graphiques de tendance et historiques
* Interface de consultation des prédictions IA
  1. **Flux de données et interactions**

Le système fonctionne selon un modèle événementiel où les interactions utilisateur déclenchent des appels API, qui à leur tour sollicitent les services métier appropriés.

**Flux de contrôle véhicule** :

1. L'utilisateur active une commande via l'interface web
2. Requête AJAX vers l'API Flask correspondante
3. Validation et traitement de la commande
4. Mise à jour de l'état du simulateur
5. Enregistrement de l'action dans les logs
6. Retour de confirmation à l'interface

**Flux de prédiction IA** :

1. Collecte périodique des données capteurs
2. Préprocessing et normalisation des données
3. Inférence via le modèle Random Forest
4. Évaluation des probabilités de panne
5. Génération d'alertes si seuils dépassés
6. Affichage des résultats dans l'interface
   1. **Gestion des états et persistance**

La gestion d'état adopte un modèle hybride combinant état mémoire pour les données volatiles (position temps réel, commandes en cours) et persistance base de données pour les données historiques et de configuration.

**État mémoire** : Variables globales Flask pour l'état instantané du véhicule (position, statut moteur, verrouillage)

**Persistance SQLite** : Tables structurées pour logs, historiques de capteurs, et paramètres de configuration

**Modèle IA** : Sérialisation via joblib pour persistance du modèle entraîné et rechargement à l'initialisation

## Planning prévisionnel sur 4 semaines

Le projet est réparti sur une durée de 4 semaines, avec les jalons suivants :

* **Semaine 1** :
  + Rédaction du cahier des charges
  + Choix des technologies
  + Répartition des tâches entre les membres
* **Semaine 2** :
  + Développement de l’interface utilisateur
  + Intégration de la simulation GPS
* **Semaine 3** :
  + Connexion à la base de données
  + Début des traitements analytiques (Pandas, matplotlib)
* **Semaine 4** :
  + Phase de test et validation
  + Amélioration de l’UX et corrections
  + Rédaction du rapport final
  + Préparation de la soutenance (PowerPoint + vidéo)

## Livrables attendus

À la fin du projet, les livrables suivants devront être produits :

* **Application fonctionnelle**
  + Interface fluide
  + Commandes et GPS simulé opérationnels
* **Code source documenté**
  + Fichiers organisés, avec commentaires
* **Vidéo de démonstration**
  + Courte présentation de l’application en fonctionnement
* **Rapport de projet**
  + Explication de la démarche, architecture, code, IA, résultats
* **Présentation PowerPoint**
  + Pour soutenir le projet devant un jury ou en classe

# Fonctionnalités implémentées

## Interface de connexion et authentification

Le système intègre une interface de connexion sécurisée, conçue pour permettre un accès restreint aux fonctionnalités sensibles de contrôle du véhicule (démarrage, arrêt, logs, prédictions IA, etc.).  
Bien que cette version du projet soit un prototype fonctionnel, l'architecture logicielle a été pensée pour évoluer vers une authentification plus robuste et modulaire, avec gestion d'utilisateurs multiples et rôles distincts (admin, préparateur, viewer…).

**Caractéristiques de l'authentification** :

* Interface de login responsive avec validation côté client
* Gestion des sessions utilisateur via Flask-Session
* Redirection automatique vers le tableau de bord après authentification
* Protection des routes sensibles par décorateurs d'authentification

**Sécurité implémentée** :

* Hachage des mots de passe (bien que simplifié pour le prototype)
* Validation des entrées utilisateur
* Protection contre les injections SQL basiques
* Gestion des timeouts de session

## Contrôle à distance : **verrouillage** et déverrouillage

La fonctionnalité de contrôle d'accès permet la gestion à distance du verrouillage véhiculaire via l'interface web.

**Implémentation technique** :

* Endpoint API REST /api/vehicle/lock et /api/vehicle/unlock
* Mise à jour immédiate de l'état dans le simulateur
* Feedback visuel instantané dans l'interface utilisateur
* Journalisation automatique de toutes les actions

**Interface utilisateur** :

* Boutons toggle avec états visuels distincts (verrouillé/déverrouillé)
* Confirmations d'action pour prévenir les manipulations accidentelles
* Indicateurs de statut en temps réel
* Historique des actions de verrouillage consultable

## Contrôle moteur : démarrage et arrêt à distance

Le système offre la capacité de contrôler l'état du moteur à distance, fonctionnalité critique pour la sécurité et la gestion de flotte.

**Mécanismes de sécurité** :

* Vérification des conditions préalables (véhicule déverrouillé, conditions de sécurité)
* Double confirmation pour les actions critiques
* Timeout automatique en cas de non-réponse
* Logs détaillés avec horodatage précis

**Logique métier** :

* État moteur persistant entre les sessions
* Corrélation avec les données de capteurs (température, régime)
* Simulation réaliste des temps de démarrage/arrêt
* Gestion des états intermédiaires (démarrage en cours, arrêt en cours)

## Géolocalisation via GPS

L'intégration de Flask permet un suivi géospatial avancé avec cartographie interactive et historique de trajectoires.

**Fonctionnalités cartographiques** :

* Affichage temps réel de la position véhiculaire sur carte OpenStreetMap
* Marqueurs personnalisés avec informations contextuelles
* Historique des trajets avec tracés colorés par période
* Zoom automatique et centrage sur la position actuelle

**Données GPS simulées** :

* Génération de trajectoires réalistes avec variation de vitesse
* Simulation de stationnement et de déplacements urbains
* Intégration de waypoints et zones d'intérêt
* Calcul de métriques de déplacement (distance, durée, vitesse moyenne)

**Performance et optimisation** :

* Mise à jour différentielle des positions pour optimiser la bande passante
* Cache côté client pour les tuiles cartographiques
* Limitation de la fréquence de rafraîchissement selon l'activité

## Système de logs et historique des activités

Un système complet de journalisation capture toutes les interactions et événements système pour audit et analyse.

**Catégories de logs implémentées** :

* **Logs d'actions utilisateur** : Connexions, commandes véhiculaires, consultations
* **Logs système** : Démarrages, erreurs, performances
* **Logs capteurs** : Valeurs historiques pour analyse de tendance
* **Logs IA** : Prédictions, alertes, performances modèle

**Structure des logs** :

    \_\_tablename\_\_ = 'activity\_logs'

    id = db.Column(db.Integer, primary\_key=True)

    vehicle\_id = db.Column(db.String(50), db.ForeignKey('vehicles.id'), nullable=False)

    timestamp = db.Column(db.DateTime, default=datetime.utcnow)

    action = db.Column(db.String(100), nullable=False)

    status = db.Column(db.String(20), nullable=False)

    details = db.Column(db.Text, nullable=False)

    user\_id = db.Column(db.String(50), nullable=True)

**Interface de consultation** :

* Filtrage par période, catégorie, niveau de sévérité
* Recherche textuelle dans les détails
* Export CSV pour analyse externe
* Graphiques de tendance des activités

## Prédiction de pannes par intelligence artificielle

Le module IA constitue l'innovation principale du système, offrant des capacités prédictives basées sur l'analyse des données capteurs.

**Pipeline de prédiction** :

1. **Collecte** : Agrégation des données capteurs en temps réel
2. **Préprocessing** : Normalisation, gestion des valeurs aberrantes
3. **Feature Engineering** : Calcul de métriques dérivées (moyennes mobiles, tendances)
4. **Inférence** : Application du modèle Random Forest
5. **Post-processing** : Interprétation et formatage des résultats

**Métriques analysées** :

* Température moteur : dérive, pics, variabilité
* Niveau carburant : consommation anormale, fuites potentielles
* Régime moteur : irrégularités, sur-régime
* Vibrations simulées : patterns anormaux
* Corrélations croisées entre capteurs

**Types de pannes prédites** :

* **Surchauffe moteur** : basée sur température et régime
* **Problèmes carburant** : niveau bas, consommation excessive
* **Usure générale** : analyse des patterns de fonctionnement
* **Maintenance préventive** : recommandations basées sur l'usage

## Réaction d'urgence et alertes

Le système intègre des mécanismes de réaction automatique aux situations critiques détectées par l'IA.

**Niveaux d'alerte** :

* **INFO** : Recommandations de maintenance préventive
* **WARNING** : Anomalies détectées nécessitant surveillance
* **EMERGENCY** : Arrêt automatique pour protection

**Actions automatiques** :

* Notification visuelle et sonore dans l'interface
* Enregistrement détaillé dans les logs d'urgence
* En mode critique : proposition d'arrêt automatique du moteur

**Interface d'urgence** :

* Zone d'alertes prominente dans le tableau de bord
* Procédures d'action recommandées pour chaque type d'alerte
* Bouton d'accusé de réception des alertes
* Historique des incidents avec résolutions

def emergency\_stop(vehicle\_id):

    vehicle = Vehicle.query.get\_or\_404(vehicle\_id)

    vehicle.engine\_status = False

    vehicle.speed = 0

    vehicle.status = 'emergency\_stop'

    vehicle.last\_update = datetime.utcnow()

    db.session.commit()

    DatabaseService.add\_log(

        vehicle\_id,

        'Emergency Stop',

        'warning',

        'Emergency stop activated - engine disabled and vehicle stopped'

    )

    return jsonify({'success': True, 'status': 'emergency\_stop'})

# Modélisation IA

## Génération et structure du jeu de données

La conception d'un modèle de prédiction efficace nécessite un jeu de données représentatif des conditions réelles d'utilisation véhiculaire. En l'absence de capteurs physiques, une approche de simulation sophistiquée a été développée.

**Générateur de données synthétiques** :

Le simulateur de capteurs génère des séries temporelles réalistes en intégrant :

* **Patterns cycliques** : Variations diurnes et saisonnières des conditions d'utilisation
* **Corrélations physiques** : Relations cohérentes entre température, régime moteur et charge
* **Bruit réaliste** : Variabilité stochastique mimant l'imprécision des capteurs réels
* **Événements rares** : Injection contrôlée d'anomalies et situations de panne

**Enrichissement par feature engineering** :

Le jeu de données brut est enrichi par des variables dérivées améliorant le pouvoir prédictif :

* **Moyennes mobiles** : Tendances à court et moyen terme
* **Dérivées temporelles** : Taux de variation des paramètres
* **Écarts à la normale** : Déviation par rapport aux valeurs de référence
* **Ratios et corrélations** : Relations entre variables (température/régime, consommation/vitesse)

## Stratégie de labellisation

La construction d'un modèle supervisé nécessite une stratégie de labellisation cohérente définissant les conditions de panne.

**Règles de labellisation implémentées** :

**Surchauffe moteur** : **carburant** :

    # Simuler des étiquettes de pannes : température élevée OU carburant bas

    data['panne'] = ((data['temperature'] > 90) | (data['fuel'] < 10)).astype(int)

**Anomalie générale** :

* Combinaison de plusieurs indicateurs dépassant les seuils
* Analyse des patterns multi-variables
* Détection d'événements rares par analyse statistique

**Classes de sortie** :

* 0 : Fonctionnement normal
* 1 : Maintenance préventive recommandée
* 2 : Intervention requise sous 48h
* 3 : Situation critique, arrêt recommandé

## Entraînement du modèle Random Forest

Le processus d'entraînement suit une méthodologie rigoureuse garantissant la robustesse et la généralisation du modèle.

**Préparation des données** :

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

import joblib

from models import db, VehicleHistory

MODEL\_PATH = 'model/failure\_predictor.pkl'

def train\_failure\_model():

    """Entraîne un modèle de prédiction de pannes à partir des données VehicleHistory"""

    history = VehicleHistory.query.all()

    if not history:

        print("❌ Aucune donnée disponible dans VehicleHistory.")

        return

    # Préparer les données

    data = pd.DataFrame([{

        'temperature': h.temperature,

        'fuel': h.fuel,

        'speed': h.speed,

        'vehicle\_id': h.vehicle\_id

    } for h in history])

    # Simuler des étiquettes de pannes : température élevée OU carburant bas

    data['panne'] = ((data['temperature'] > 90) | (data['fuel'] < 10)).astype(int)

    # Features et cible

    X = data[['temperature', 'fuel', 'speed']]

    y = data['panne']

    # Split pour test

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Entraînement

    model = RandomForestClassifier(n\_estimators=100, random\_state=42)

    model.fit(X\_train, y\_train)

    # Évaluation

    print("📊 Résultats du modèle :")

    print(classification\_report(y\_test, model.predict(X\_test)))

## Validation et métriques de performance

L'évaluation du modèle s'appuie sur des métriques adaptées au contexte de prédiction de pannes, où les faux négatifs (pannes non détectées) sont plus critiques que les faux positifs.

**Métriques d'évaluation** : **Analyse de l'importance des variables** :

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

model = None

def train\_model(df):

    global model

    X = df[['temperature', 'fuel', 'speed', 'engine\_status']]

    y = df['panne']

    X\_train, \_, y\_train, \_ = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

    clf.fit(X\_train, y\_train)

    model = clf

def predict\_failure(vehicle):

    global model

    if model is None:

        return {'error': 'Model not trained'}

    data = pd.DataFrame([{

        'temperature': vehicle.temperature,

        'fuel': vehicle.fuel,

        'speed': vehicle.speed,

        'engine\_status': 1 if vehicle.engine\_status else 0

    }])

    prediction = model.predict(data)[0]

    proba = model.predict\_proba(data)[0][1]

    return {'prediction': int(prediction), 'probability': round(proba, 2)}

Cette analyse confirme la pertinence de la température moteur comme indicateur principal, tout en révélant l'importance des variables dérivées (moyennes mobiles, variances).

## Sérialisation et persistance du modèle

La sauvegarde du modèle entraîné utilise la bibliothèque joblib, optimisée pour les objets scikit-learn volumineux.

**Chargement et utilisation** :

* Chargement au démarrage de l'application

MODEL\_PATH = 'model/failure\_predictor.pkl'

    # Sauvegarde

    joblib.dump(model, MODEL\_PATH)

    print(f"✅ Modèle enregistré : {MODEL\_PATH}")

* Fonction de prédiction

def generate\_training\_data():

    n = 1000

    df = pd.DataFrame({

        'temperature': np.random.normal(75, 10, n),

        'fuel': np.random.uniform(10, 100, n),

        'speed': np.random.uniform(0, 120, n),

        'engine\_status': np.random.choice([0, 1], n),

        'panne': np.random.choice([0, 1], n, p=[0.9, 0.1])

    })

    return df

df\_train = generate\_training\_data()

train\_model(df\_train)

## Intégration temps réel dans l'interface

L'affichage des résultats de prédiction dans l'interface web combine visualisation intuitive et informations techniques détaillées.

**Composants d'affichage** :

* **Jauge de risque** : Indicateur visuel du niveau de risque global
* **Graphiques de tendance** : Évolution des probabilités dans le temps
* **Alertes contextuelles** : Messages d'explication et recommandations
* **Détail par capteur** : Contribution de chaque variable à la prédiction

**Mise à jour en temps réel** :

* Requêtes AJAX périodiques vers l'endpoint /api/predict
* Mise à jour différentielle des éléments modifiés
* Cache côté client pour optimiser les performances
* Gestion de la connectivité intermittente

# Diagrammes

## Diagramme de Cas d’utilisation :

Ce diagramme de cas d'utilisation modélise les interactions entre différents acteurs et un système de gestion de véhicule à distance.

**Acteurs :**

1. **Système :** Cet acteur représente un système externe ou une entité autonome qui interagit avec le véhicule à distance pour la prédiction de panne.
2. **Admin :** L'administrateur est un utilisateur privilégié qui a des droits étendus sur le système.
3. **Propriétaire :** Le propriétaire est l'utilisateur principal du véhicule à distance, ayant des droits de gestion et de consultation.

**Cas d'utilisation :**

**Cas d'utilisation liés au Système (acteur externe) :**

* **Prédiction de panne AI :** Ce cas d'utilisation indique que le "Système" est capable de prédire les pannes du véhicule à distance en utilisant des techniques d'intelligence artificielle.

**Cas d'utilisation liés à l'Admin :**

* **Démarrage & Arrêt moteur :** L'administrateur peut initier le démarrage et l'arrêt du moteur du véhicule à distance.
* **Verrouillage & Déverrouillage :** L'administrateur a la capacité de verrouiller et déverrouiller le véhicule à distance.
* **Visualisation position :** L'administrateur peut visualiser la position actuelle du véhicule.
* **Consulté logs d'activités :** L'administrateur peut consulter les journaux d'activités du système et du véhicule.
* **Connexion à Interface :** L'administrateur doit se connecter à l'interface du système pour accéder aux fonctionnalités.
* **Configurer les paramètres initiaux :** L'administrateur est responsable de la configuration des paramètres initiaux du système.
* **Ajouter nouvelle véhicule :** L'administrateur a la possibilité d'ajouter de nouveaux véhicules au système.

**Cas d'utilisation liés au "Propriétaire" :**

* **Démarrage & Arrêt moteur :** Le propriétaire peut également démarrer et arrêter le moteur à distance, tout comme l'administrateur.
* **Verrouillage & Déverrouillage :** Le propriétaire a aussi la capacité de verrouiller et déverrouiller le véhicule à distance.
* **Visualisation position :** Le propriétaire peut visualiser la position actuelle de son véhicule.
* **Consulté logs d'activités :** Le propriétaire peut consulter les journaux d'activités liés à son véhicule.
* **Connexion à Interface :** Le propriétaire doit se connecter à l'interface du système pour gérer son véhicule.

**Relations entre les cas d'utilisation :**

* **Réagir en cas d'urgence :** Ce cas d'utilisation est inclus ("include") dans le cas d'utilisation "Consulté logs d'activités". Cela signifie que la consultation des logs d'activités peut potentiellement déclencher une "Réaction en cas d'urgence" si une anomalie est détectée.
* **Suivi GPS (+Suivi GPS) :** Cette note ou extension est liée au cas d'utilisation "Visualisation position", suggérant que la visualisation de la position est réalisée via un suivi GPS. (Il s'agit plus d'une note ou d'un stéréotype que d'une relation formelle "include" ou "extend" ici, mais elle clarifie la méthode).

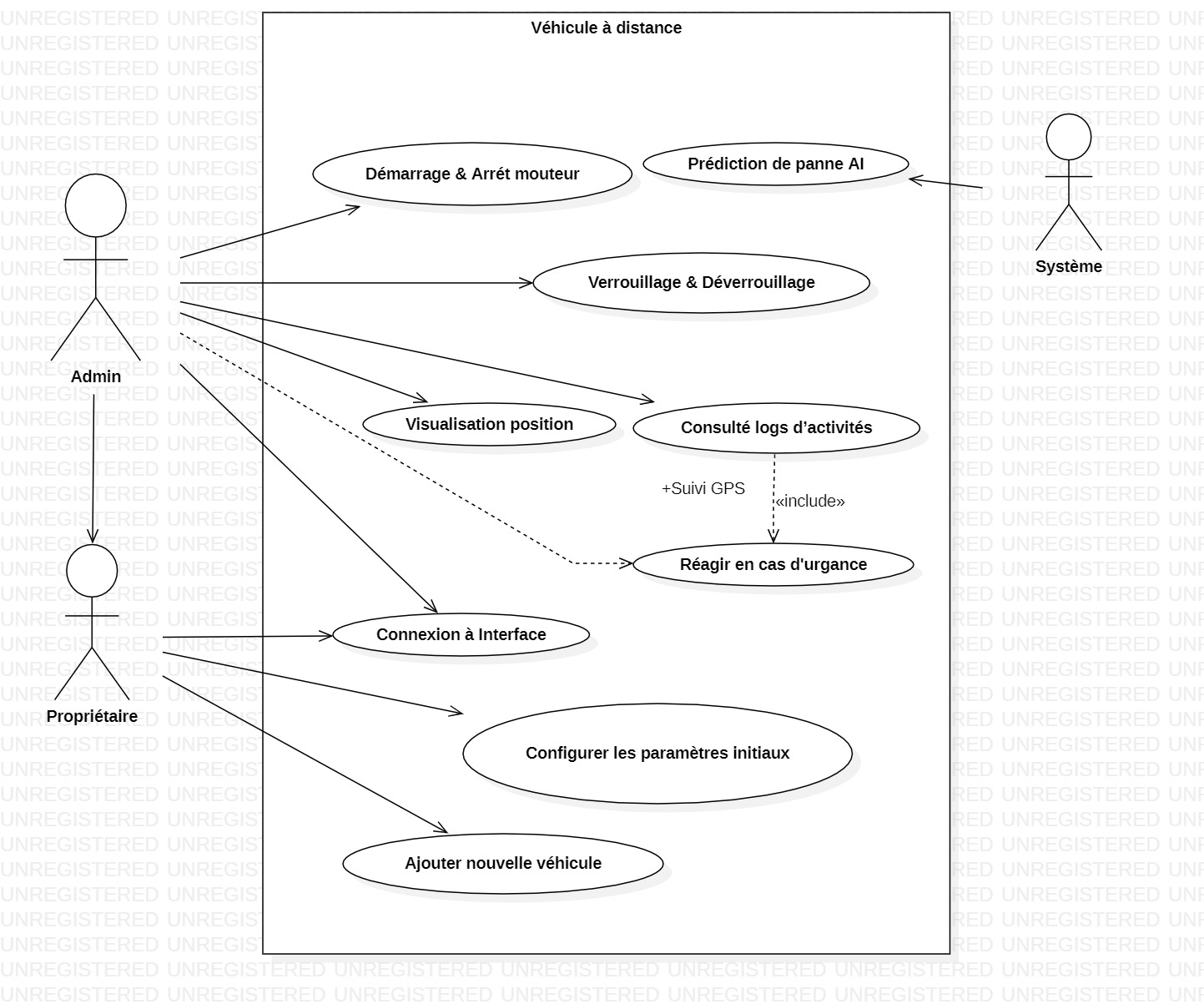


Figure 1 : Diagram de Cas d'utilisation

## Diagramme de Classes :

Le diagramme présente 5 classes principales et leurs relations :

**Vehicle =>** Représente le véhicule en temps réel

**VehicleHistory =>** Archive les états passés du véhicule

**Log =>** Enregistre les actions et événements

**DatabaseService =>** Contient les méthodes utilitaires pour la base

**FailurePredictionModel =>** Implémente le modèle IA (RandomForest)

1. **Classe Véhicule**

Représente l’état actuel et dynamique d’un véhicule dans la base de données.

***🔹 Attributs :***

id: identifiant unique

name: nom ou modèle du véhicule

speed, fuel, temperature, latitude, engine\_status : données de capteurs simulés

***🔹 Méthodes :***

to\_dict() : retourne l’objet sous forme de dictionnaire

as\_dict() : (variante ou alias de to\_dict)

**🔗 *Relations :***

1 — \* avec VehicleHistory (un véhicule possède plusieurs entrées d’historique)

Utilisé par FailurePredictionModel pour la prédiction

1. **Classe VehicleHistory**

Stocke les états précédents du véhicule à chaque simulation (chaque 3s).

***🔹 Attributs :***

timestamp: date et heure de l’enregistrement

speed, fuel, temperature, latitude, engine\_status: comme dans Vehicle

***🔹 Méthodes :***

to\_dict()

save() : insère une nouvelle entrée dans l’historique

1. **Classe Log**

Représente les journaux d’activité du système.

***🔹 Attributs :***

id: identifiant du log

vehicle\_id: lien vers le véhicule concerné

timestamp: date de l’action

level: niveau (success, warning, error)

message: description de l’événement

1. **Classe DatabaseService**

Classe utilitaire avec des méthodes statiques pour gérer les logs et l’historique.

***🔹 Méthodes :***

init\_database() : initialise la base avec des véhicules

get\_recent\_logs(limit) : récupère les derniers événements

save\_vehicle\_history(vehicle) : enregistre l’état actuel dans VehicleHistory

add\_log(vehicle\_id, level, message) : ajoute une ligne dans Log

1. **Classe FailurePredictionModel**

Contient le modèle IA basé sur Random Forest (via scikit-learn).

***🔹 Attributs :***

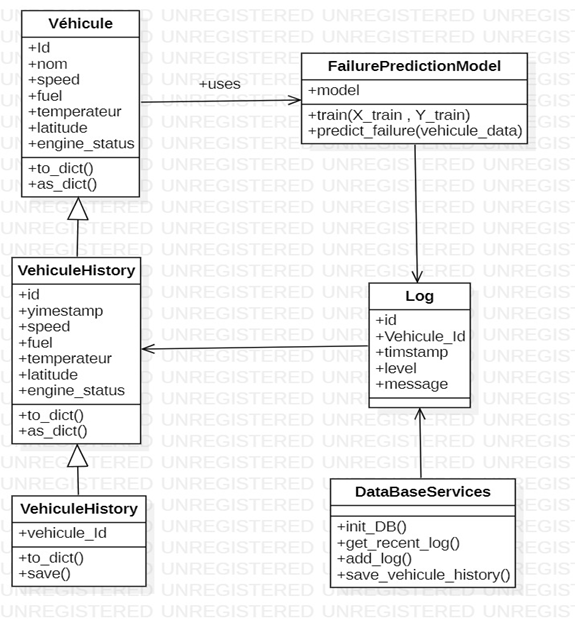
model: objet RandomForestClassifier

***🔹 Méthodes :***

train(X\_train, y\_train) : entraînement du modèle

predict\_failure(vehicle\_data) : retourne une prédiction de panne et une probabilité

**🔗 *Relations :*** Vehicle utilise cette classe pour analyser ses données

**Figure 2 : Diagram de Classes**

# Stockage et analyse des données

Chaque point GPS est stocké avec un horodatage dans une base de données. À partir de ces données, nous analysons :

* La distance totale parcourue
* Les heures d’activité les plus fréquentes
* Les zones les plus visitées

Les résultats sont présentés sous forme de graphiques pour faciliter l’interprétation. Ces visualisations permettent aussi d’identifier les habitudes de déplacement et d’anticiper des comportements futurs.

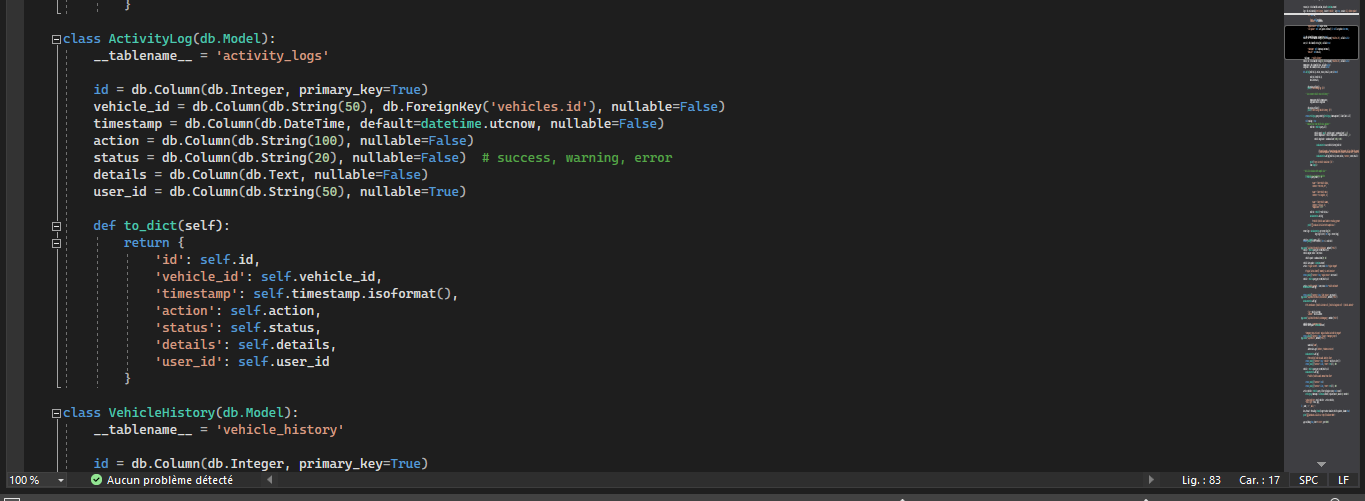


Figure 3 : Extrait du modèle ORM pour les logs

# Réalisation technique

Nous avons conçu une interface utilisateur affichant tous les véhicules et leur état en temps réel. Chaque commande effectuée (ex. démarrer moteur) est enregistrée avec son statut et sa date. Un système de traduction par dictionnaire Python permet de changer dynamiquement la langue du tableau de bord.

## Vue du tableau de bord (Dashboard) :

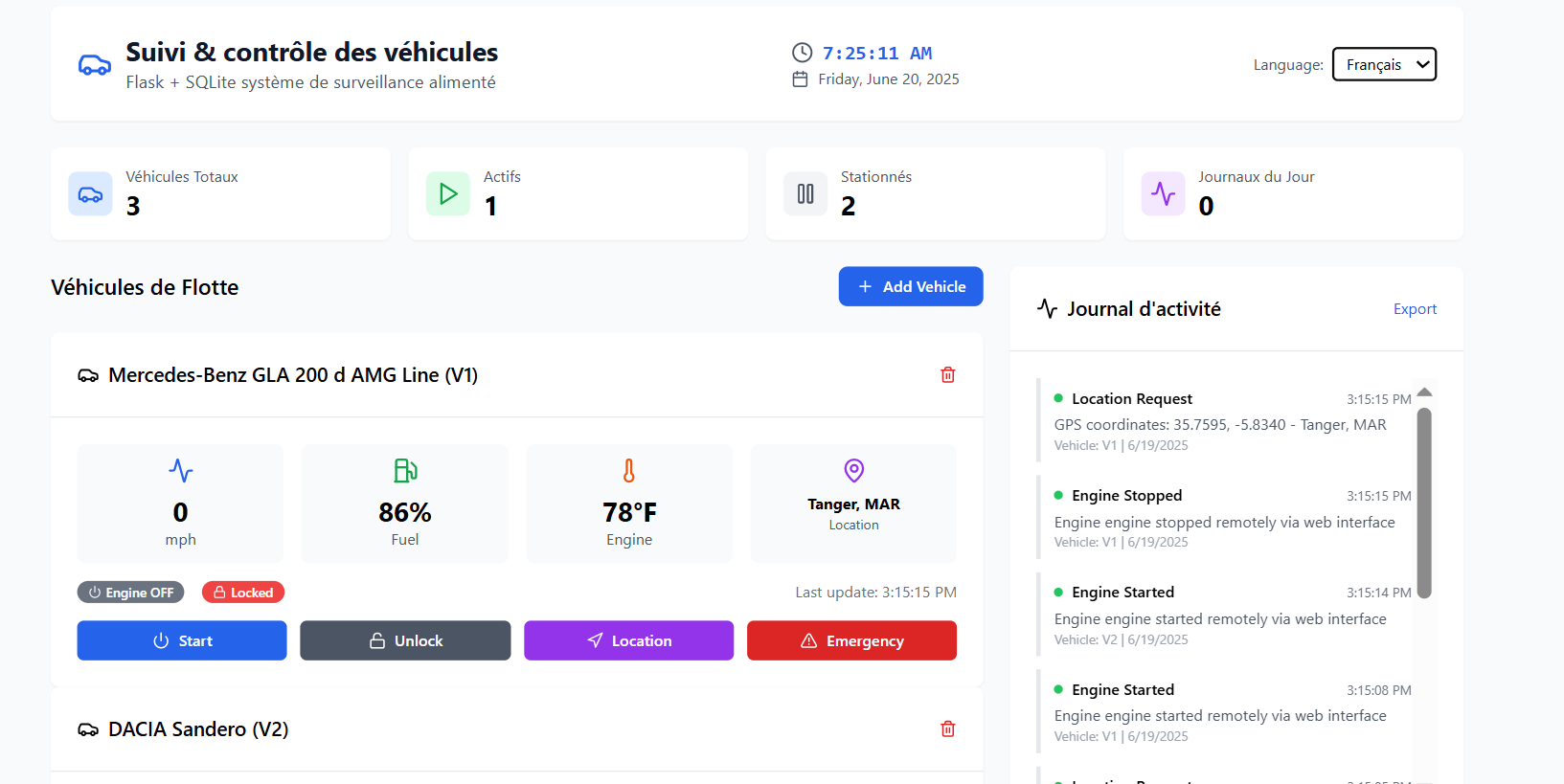


Figure 4: Interface principale (Dashboard)

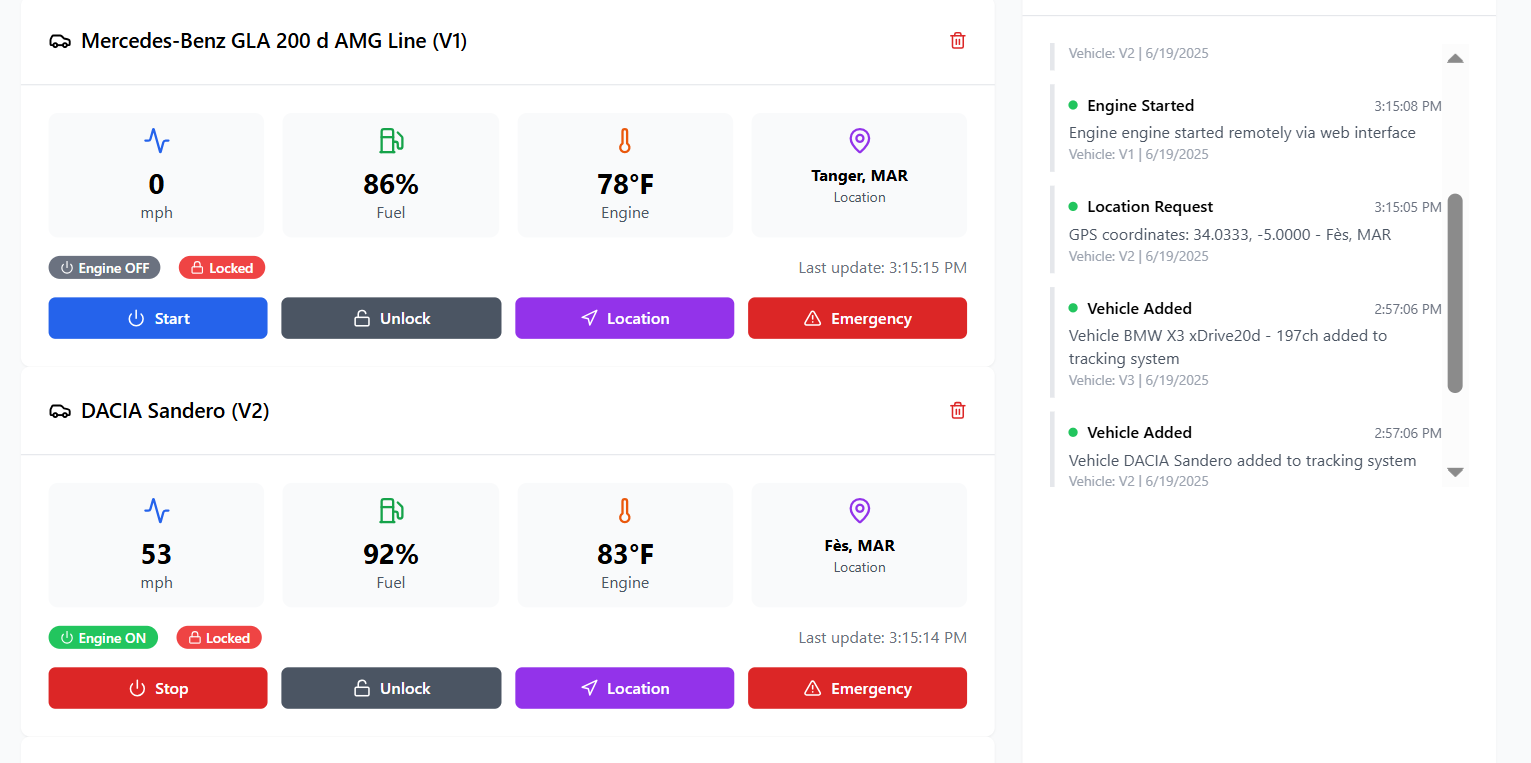
L’interface est fluide, multilingue, et fonctionnelle avec plusieurs véhicules.

Figure 5 : Vue multi-véhicules dans application

## Formulaire d’ajout de véhicule :

Capture d’écran illustrant le formulaire d’ajout de nouveaux véhicules dans l’interface.

Champs : Identifiant du véhicule, nom, position initiale, état par défaut.

Exemple :

* ID : VH005
* Nom : Dacia Ligan
* Position : 67,7639 / -52,0010 initiales
* État : Arrêté

Figure 6 : Formulaire d’ajout de véhicule

## Historique des logs :

Exemple de table de logs :

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Heure | Action | Véhicule | Résultat |
| 2025-04-11 14:05 | Start | VH001 | Succès |
| 2025-04-12 10:06 | Stop | VH001 | Succès |
| 2025-04-12 13:10 | Simulation GPS Position | VH002 | En cours |

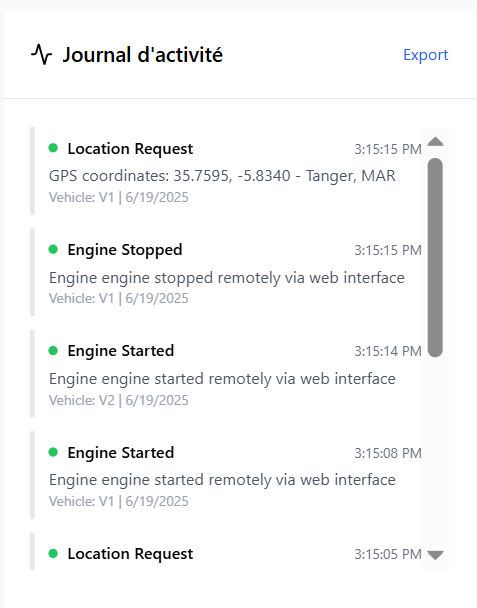


Figure 7 : Historique des logs

# Perspectives d'amélioration

## Système d'alertes et notifications temps réel

L'extension du système vers des capacités d'alerte proactives améliorerait significativement l'utilité pratique.

***Canaux de notification multiples*** *:*

*Notifications push web* : Utilisation des Service Workers pour notifications navigateur persistantes, même application fermée.

*Intégration SMS/Email* : API Twilio ou équivalent pour alertes critiques hors connexion internet.

*Applications mobiles natives* : Développement apps iOS/Android avec notifications push natives et géofencing.

*Intégration systèmes tiers* : Webhooks vers systèmes de gestion de flotte existants (Slack, Teams, ITSM).

***Personnalisation utilisateur :***

* Seuils d'alerte configurables par utilisateur/véhicule
* Plages horaires de notification (mode silencieux nuit)
* Filtrage par type d'événement et priorité
* Accusés de réception et suivi des actions

## Évolution vers des modèles d'IA avancés

L'amélioration continue des capacités prédictives passe par l'adoption de techniques d'IA plus sophistiquées.

***Deep Learning pour séries temporelles :***

*Réseaux LSTM/GRU* : Adaptation à l'analyse de séquences temporelles longues pour détecter des patterns complexes non visibles par Random Forest.

*Autoencoders* : Détection d'anomalies non supervisée, identification de patterns de fonctionnement inconnus.

*Transformers adaptés* : Application des architectures d'attention pour capturer des dépendances long-terme entre capteurs.

***Apprentissage en continu :***

*Online Learning* : Adaptation du modèle aux spécificités de chaque véhicule sans réentraînement complet.

*Federated Learning* : Apprentissage distribué préservant la confidentialité, amélioration collective sans partage de données brutes.

*Active Learning* : Identification automatique des cas ambigus nécessitant validation expert pour amélioration ciblée.

***IA explicable (XAI) :***

*SHAP/LIME* : Explication détaillée des prédictions pour validation métier et confiance utilisateur.

*Visualisations interactives* : Interface permettant exploration des facteurs contribuant aux prédictions.

*Règles métier hybrides* : Combinaison IA + expertise domaine pour prédictions robustes et explicables.

## Renforcement de la sécurité

La sécurité constitue un enjeu critique pour tout système de contrôle véhiculaire connecté.

***Authentification et autorisation robustes :***

*Multi-factor Authentication (MFA)* : Combinaison mot de passe + token/biométrie pour accès sensibles.

*OAuth2/OpenID Connect* : Intégration systèmes d'authentification entreprise (Active Directory, LDAP).

*Certificats PKI* : Infrastructure à clés publiques pour authentification mutuelle device ↔ serveur.

*Gestion des sessions* : Tokens JWT avec expiration, révocation, rotation automatique.

***Chiffrement end-to-end :***

*Communication chiffrée* : TLS 1.3 minimum pour toutes les communications, pinning de certificats.

*Chiffrement données au repos* : Chiffrement base de données (TDE) et fichiers sensibles (AES-256).

*Chiffrement des commandes* : Signature cryptographique des commandes de contrôle véhiculaire.

***Sécurité applicative :***

*Validation stricte entrées* : Sanitisation, validation côté serveur, protection injections.

*Rate limiting* : Protection contre attaques par déni de service et brute force.

*Audit de sécurité* : Logging exhaustif des accès, tentatives d'intrusion, actions sensibles.

*Segmentation réseau* : Isolation des composants critiques, DMZ pour interfaces publiques.

## Scalabilité et architecture distribuée

L'évolution vers un système supportant de multiples véhicules nécessite une refonte architecturale.

***Microservices et conteneurisation :***

*Architecture microservices* : Décomposition en services spécialisés (authentification, contrôle, prédiction, notifications) pour scalabilité indépendante.

*Conteneurisation Docker* : Déploiement cohérent multi-environnements, orchestration Kubernetes pour haute disponibilité.

*API Gateway* : Point d'entrée unique avec load balancing, rate limiting, authentification centralisée.

***Bases de données distribuées :***

*Sharding par véhicule* : Partitionnement horizontal des données pour performances et isolation.

*Réplication maître-esclave* : Haute disponibilité avec basculement automatique.

*Cache distribué* : Redis Cluster pour cache des données fréquemment accédées.

***Monitoring et observabilité :***

*Métriques applicatives* : Prometheus + Grafana pour monitoring performances et alertes.

*Tracing distribué* : Jaeger pour suivi des requêtes inter-services et debugging.

*Logging centralisé* : ELK Stack (Elasticsearch, Logstash, Kibana) pour analyse logs.

# Conclusion

Le présent projet s’inscrit dans une démarche de transformation numérique appliquée à la gestion et au contrôle des véhicules à distance. En mobilisant des technologies modernes telles que Python/Flask pour le backend, Streamlit pour l’interface, ainsi qu’une base de données SQLite, nous avons pu concevoir une solution logicielle complète, modulaire et orientée utilisateur.

La spécificité du projet réside dans son approche purement logicielle, simulant les aspects essentiels du suivi et du contrôle d’un véhicule sans recourir à du matériel physique. Cette orientation a permis de se concentrer sur l’architecture logicielle, la structuration des données, la réactivité de l’interface, ainsi que l’adaptabilité multilingue du système.

La solution développée se distingue par son accessibilité, son interface intuitive et son approche entièrement logicielle, évitant les contraintes matérielles habituelles. Grâce à l'utilisation de technologies modernes telles que Python, Flask, Streamlit et SQLite, nous avons pu créer un environnement fonctionnel, modulable, et apte à évoluer vers des versions plus avancées, notamment avec des applications mobiles, des notifications temps réel ou une extension vers des usages professionnels.

En perspective, cette application pourrait être enrichie par une intégration matérielle (capteurs réels, modules GPS/GSM), un système d’alertes en temps réel, ou encore un déploiement mobile multiplateforme. Ce projet constitue ainsi une base solide pour des développements futurs dans le domaine des véhicules intelligents et de l’Internet des Objets (IoT).

# Glossaire

**API (Application Programming Interface)** : Interface de programmation permettant l'interaction entre différents composants logiciels par l'exposition de fonctions standardisées.

**CAN Bus (Controller Area Network)** : Protocole de communication série développé pour l'industrie automobile permettant la communication entre les différents modules électroniques d'un véhicule.

**Machine Learning** : Domaine de l'intelligence artificielle permettant aux systèmes d'apprendre automatiquement à partir de données sans programmation explicite.

**Deep Learning** : Sous-domaine du machine learning utilisant des réseaux de neurones artificiels multicouches pour l'apprentissage de représentations complexes dans les données.

**Edge Computing** : Paradigme informatique rapprochant le traitement des données de leur source de génération pour réduire la latence et optimiser la bande passante.

**Feature Engineering** : Processus de sélection, modification et création de variables (features) à partir des données brutes pour améliorer les performances des modèles d'apprentissage automatique.

**Flask** : Framework web léger pour Python permettant le développement rapide d'applications web et d'APIs REST.

**IoT (Internet of Things)** : Réseau d'objets physiques connectés équipés de capteurs, logiciels et autres technologies permettant l'échange de données.

**LSTM (Long Short-Term Memory)** : Architecture de réseau de neurones récurrent spécialement conçue pour traiter et prédire des séquences temporelles longues.

**Microservices** : Architecture logicielle structurant une application comme un ensemble de services faiblement couplés, déployables et maintenables indépendamment.

**MQTT (Message Queuing Telemetry Transport)** : Protocole de messagerie léger conçu pour les communications M2M (Machine to Machine) dans des environnements contraints.

**Random Forest** : Algorithme d'apprentissage automatique combinant multiple arbres de décision pour améliorer la précision prédictive et réduire le surapprentissage.

**REST (Representational State Transfer)** : Style architectural pour les services web utilisant les méthodes HTTP standard (GET, POST, PUT, DELETE) pour les opérations sur les ressources.

**Scikit-learn** : Bibliothèque Python open-source proposant des outils simples et efficaces pour l'analyse prédictive et le machine learning.

**SQLite** : Système de gestion de base de données relationnel embarqué, auto-suffisant et ne nécessitant pas de serveur séparé.

**Télématique** : Domaine technologique combinant télécommunications et informatique, appliqué à la transmission d'informations sur de longues distances.

**TLS (Transport Layer Security)** : Protocole cryptographique garantissant la sécurité des communications sur réseau informatique, successeur du SSL.

**WebSocket** : Protocole de communication bidirectionnelle full-duplex sur une connexion TCP, permettant l'échange temps réel entre client et serveur web.