École Marocaine des Sciences de l'Ingénieur

EMSI

Filière : Méthodes Informatiques Appliquées à la Gestion des Entreprises - MIAGE

9ème Semestre

Rapport de Projet de Fin d'Année

AutoStory

Générateur intelligent de contenus publicitaires
pour le secteur automobile

Réalisé par

Youssef MOUSTAID
Fayz OUSSAMA

Encadré par
M. RAOUYANE

Année Universitaire 2025-2026
Dfécembre 2026

Dédicaces

Je tiens à dédier ce travail à toutes les personnes qui ont marqué mon parcours et qui m'ont permis de mener à bien ce **Projet de Fin d'Année**.

À mes chers parents, pour leur amour incommensurable, leur patience et leurs sacrifices. Leur soutien moral, matériel et affectif a été la clé de ma réussite. Je leur dois ce que je suis aujourd'hui et leur exprime ma gratitude éternelle.

À ma famille, frères, sœurs et proches, pour leurs encouragements constants, leur compréhension et leurs prières qui m'ont donné la force de persévérer même dans les moments difficiles.

À mes enseignants et encadrants, qui m'ont transmis le savoir, guidé dans ma formation et accompagné tout au long de ce projet. Leur disponibilité et leurs conseils avisés ont grandement contribué à l'aboutissement de ce travail.

À mes camarades de promotion et amis, avec qui j'ai partagé des moments de travail, de réflexion et d'amitié. Leur collaboration et leur esprit d'équipe ont rendu ce parcours plus agréable et enrichissant.

Enfin, à toutes les personnes, proches ou lointaines, qui ont cru en moi et m'ont soutenu dans cette aventure académique et personnelle. Ce travail est aussi le leur.

Remerciements

Nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce **Projet de Fin d'Année**.

Tout d'abord, nous remercions l'**École Marocaine des Sciences de l'Ingénieur (EMSI)** pour la formation de qualité dont nous avons bénéficié et pour nous avoir offert l'opportunité de travailler sur un projet innovant dans le domaine de l'intelligence artificielle appliquée au marketing automobile.

Nous souhaitons également remercier notre encadrant académique **M. Raouyane** pour son suivi rigoureux, ses conseils avisés et ses recommandations constructives, qui nous ont permis d'améliorer la qualité de ce travail et de mener à bien ce projet.

Nos remerciements vont également à tous les enseignants de la filière **MIAGE** qui nous ont transmis les connaissances théoriques et pratiques nécessaires à la réalisation de ce projet.

Enfin, nous adressons un grand merci à nos familles et nos amis pour leur soutien moral et leur encouragement constant durant cette période exigeante.

Résumé

Ce rapport présente le **Projet de Fin d'Année (PFA)** intitulé *AutoStory*, réalisé à l'**École Marocaine des Sciences de l'Ingénieur (EMSI)** dans le cadre de la filière **MIAGE**. L'objectif principal de ce projet est le développement d'un **système automatisé de génération de vidéos publicitaires** pour le secteur automobile, exploitant les capacités des **Large Language Models (LLMs)** et des technologies de synthèse vocale.

La solution développée se compose d'un **backend Python** utilisant **Google Gemini AI 2.5 Flash** pour la génération de scripts narratifs techniques, **gTTS** pour la synthèse vocale française, et **MoviePy** pour la composition vidéo automatisée. Un **frontend React/TypeScript** est en cours de développement pour offrir une interface utilisateur moderne et intuitive. Le système permet de générer des vidéos publicitaires personnalisées en environ 3 minutes, adaptées au style du véhicule (sportif, luxe, familial, écologique).

Ce projet académique a permis d'acquérir une expérience enrichissante en matière d'**intelligence artificielle générative**, de **développement full-stack**, d'**architecture logicielle modulaire**, et de traitement automatisé du langage naturel, tout en développant une solution innovante pour le marketing automobile.

Mots-clés : Intelligence Artificielle, LLM, Gemini AI, Python, React, Génération automatique de contenu, Marketing automobile, Text-to-Speech, Composition vidéo.

Table des matières

Table des figures

Liste des tableaux

Liste des sigles et abréviations

Chapitre 1

Introduction

Introduction Générale

Ce rapport de **Projet de Fin d'Année (PFA)** présente le développement du projet **AutoStory**, réalisé dans le cadre de la formation en **Ingénierie Logicielle et Intelligence Artificielle** à l'**École Marocaine des Sciences de l'Ingénieur (EMSI)**.

Contexte

Le marketing digital et le secteur automobile connaissent actuellement une transformation profonde portée par l'essor des technologies d'**intelligence artificielle générative**. Les entreprises du secteur automobile ont un besoin croissant de **contenus publicitaires personnalisés et produits rapidement** pour promouvoir leurs véhicules sur les plateformes digitales. Cependant, la production vidéo traditionnelle reste **coûteuse, chronophage et nécessite des compétences spécialisées** (scénaristes, voix-off professionnelles, monteurs vidéo).

L'émergence des **Large Language Models (LLMs)** tels que GPT-4, Claude et Google Gemini ouvre de nouvelles perspectives pour l'automatisation de la création de contenu narratif. Parallèlement, les technologies de **synthèse vocale (Text-to-Speech)** et de **composition vidéo programmatique** permettent aujourd'hui de produire des supports publicitaires de qualité sans intervention humaine extensive.

Problématique

Face à ce contexte, plusieurs questions se posent :

— Comment exploiter les capacités des LLMs pour générer des scripts publicitaires techniques et cohérents pour des véhicules automobiles ?

— Comment automatiser l'ensemble de la chaîne de production vidéo, de la génération du texte à la composition finale ?

— Comment garantir la qualité narrative et la cohérence des contenus générés automatiquement ?

— Quelle architecture logicielle adopter pour assurer modularité, scalabilité et maintenabilité du système ?

Objectifs du projet

Le projet **AutoStory** vise à répondre à ces problématiques en développant un système complet de génération automatisée de vidéos publicitaires pour le secteur automobile. Les objectifs spécifiques sont les suivants :

1. Mettre en place un système de génération de scripts narratifs techniques utilisant **Google Gemini 2.5 Flash**
2. Développer un module de synthèse vocale en français avec **gTTS (Google Text-to-Speech)**
3. Implémenter un système de composition vidéo automatisé avec **MoviePy**
4. Créer un module de classification automatique des styles de véhicules (sportif, luxe, familial, écologique)
5. Concevoir une interface utilisateur moderne avec **React et TypeScript**
6. Adopter une architecture logicielle **modulaire et scalable** facilitant l'évolution future du système

Démarche et structure du rapport

Ce travail s'inscrit dans une démarche pédagogique rigoureuse, combinant apprentissage théorique et application pratique. Il nous permet de mobiliser nos connaissances en **intelligence artificielle**, **développement logiciel**, **traitement du langage naturel** et **architecture applicative**.

Le présent rapport est structuré de la manière suivante :

— Le **Chapitre 1** présente le cahier des charges du projet AutoStory

— Le **Chapitre 2** expose le contexte général, l'état de l'art des technologies IA génératives et le positionnement d'AutoStory

— Le **Chapitre 3** détaille l'analyse et la spécification des besoins fonctionnels et non-fonctionnels

— Le **Chapitre 4** décrit la conception de l'architecture et les choix techniques

— Le **Chapitre 5** présente la réalisation et la mise en œuvre du système

— Le **Chapitre 6** conclut par un bilan du projet et présente les perspectives d'évolution

Chapitre 2

Contexte Général du Projet AutoStory

2.1 Introduction

Dans ce chapitre, nous présentons le contexte général du projet **AutoStory**, en abordant les enjeux actuels du marketing automobile digital, l'état de l'art des technologies d'intelligence artificielle générative, ainsi que le positionnement de notre solution dans cet écosystème. Ce chapitre vise à fournir une compréhension approfondie des motivations techniques et commerciales qui ont guidé la conception de ce système de génération automatisée de vidéos publicitaires.

2.2 Le marketing automobile à l'ère du digital

2.2.1 Évolution du secteur automobile

Le secteur automobile connaît une transformation digitale majeure. Les concessionnaires et constructeurs automobiles investissent massivement dans la présence en ligne et le marketing digital pour atteindre leurs clients. La vidéo publicitaire est devenue un format privilégié pour promouvoir les véhicules sur les plateformes sociales (YouTube, Facebook, Instagram, TikTok) et les sites web commerciaux.

2.2.2 Les défis de la production vidéo traditionnelle

La production de contenus vidéo publicitaires présente plusieurs contraintes majeures :

— **Coûts élevés :** Une vidéo publicitaire professionnelle nécessite un budget significatif (équipe de tournage, matériel, post-production)

— **Délais de production :** Le processus complet peut prendre plusieurs semaines (écriture du script, enregistrement voix-off, montage)

— **Compétences spécialisées :** Rédacteurs, comédiens voix-off, monteurs vidéo professionnels

— **Scalabilité limitée :** Difficulté à produire des contenus personnalisés à grande échelle

2.3 L'intelligence artificielle générative

2.3.1 Les Large Language Models (LLMs)

Les **Large Language Models** représentent une avancée majeure en traitement du langage naturel. Des modèles comme GPT-4 (OpenAI), Claude (Anthropic), et Gemini (Google) sont capables de générer des textes cohérents, techniques et créatifs dans de multiples domaines.

2.3.1.1 Google Gemini 2.5 Flash

Pour AutoStory, nous avons sélectionné **Google Gemini 2.5 Flash** pour plusieurs raisons :

— Excellente capacité de génération de textes techniques et narratifs
— Support multilingue de qualité (notamment le français)
— Vitesse de génération optimale (modèle "Flash")
— API stable et bien documentée
— Coût d'utilisation compétitif

2.3.2 La synthèse vocale (Text-to-Speech)

Les technologies de synthèse vocale ont considérablement progressé ces dernières années. Les voix synthétiques modernes sont difficilement distinguables des voix humaines. Pour notre projet, nous utilisons **gTTS (Google Text-to-Speech)**, qui offre :

— Une qualité vocale naturelle et fluide
— Un support excellent du français
— Une intégration simple via API Python
— Une fiabilité éprouvée

2.3.3 La composition vidéo programmatique

La bibliothèque **MoviePy** permet la création et l'édition de vidéos de manière programmatique en Python. Elle offre :

— Manipulation d'images et de clips vidéo
— Synchronisation audio-vidéo précise
— Ajout de textes, transitions et effets
— Exportation dans divers formats (MP4, AVI, etc.)

2.4 État de l'art des solutions existantes

2.4.1 Solutions commerciales

Plusieurs solutions commerciales existent dans le domaine de la génération automatique de contenus :

Table 2.1. Comparaison des solutions existantes

Solution	Fonctionnalités	Limitations
Synthesia	Génération de vidéos avec avatars IA	Coût élevé, peu de personnalisation
Pictory	Création vidéo à partir de textes	Orienté réseaux sociaux, templates limités
Lumen5	Transformation d'articles en vidéos	Pas de focus automobile
AutoStory	Génération spécialisée secteur auto	Solution académique en développement

2.4.2 Positionnement d'AutoStory

AutoStory se distingue par :

— **Spécialisation automobile :** Scripts techniques adaptés au secteur

— **Classification automatique :** Détection du style de véhicule (sportif, luxe, familial, écologique)

— **Architecture modulaire :** Facilité d'évolution et d'adaptation

— **Open source :** Projet académique avec vocation éducative

— **Personnalisation avancée :** Adaptation du ton narratif au style du véhicule

2.5 Objectifs et portée du projet

2.5.1 Objectifs principaux

Le projet AutoStory vise à :

1. Développer un système end-to-end de génération de vidéos publicitaires automobiles
2. Exploiter les capacités des LLMs pour créer des scripts narratifs de qualité

3. Automatiser entièrement la chaîne de production (texte → voix → vidéo)
4. Fournir une interface utilisateur intuitive et moderne
5. Adopter une architecture logicielle évolutive et maintenable

2.5.2 Périmètre du projet

Le projet couvre :

— **Backend Python** avec API REST
— Module de génération de scripts avec Google Gemini
— Module de synthèse vocale avec gTTS
— Module de composition vidéo avec MoviePy
— Classification automatique des styles de véhicules
— **Frontend React/TypeScript** (en développement)

2.5.3 Valeur ajoutée

AutoStory apporte une valeur ajoutée significative :

— **Réduction des coûts :** Élimination des coûts de production traditionnelle
— **Gain de temps :** Génération en 3 minutes vs plusieurs semaines
— **Scalabilité :** Production de multiples variantes sans effort additionnel
— **Personnalisation :** Adaptation automatique au style du véhicule
— **Accessibilité :** Démocratisation de la production vidéo professionnelle

2.6 Conclusion

Ce chapitre a présenté le contexte global du projet AutoStory, en mettant en évidence les enjeux du marketing automobile digital, l'état de l'art des technologies IA génératives, et le positionnement de notre solution. Dans le chapitre suivant, nous détaillerons l'analyse et la spécification des besoins fonctionnels et non-fonctionnels du système.

Chapitre 3

Analyse et Spécification des Besoins

3.1 Introduction

Dans ce chapitre, nous présenterons l'analyse détaillée des besoins du projet AutoStory. Cette phase constitue une étape fondamentale dans le développement du système, car elle permet d'identifier et de formaliser les exigences fonctionnelles et non-fonctionnelles. Nous établirons également les spécifications techniques qui guideront la conception et l'implémentation de la solution.

3.2 Besoins fonctionnels

Les besoins fonctionnels décrivent les fonctionnalités que le système AutoStory doit offrir aux utilisateurs.

3.2.1 Génération de scripts narratifs

BF1 : Génération automatique de scripts publicitaires

— Le système doit générer des scripts narratifs techniques et cohérents pour des véhicules automobiles

— Le script doit inclure des informations sur les caractéristiques techniques, le design et les innovations

— La longueur du script doit être adaptée à une vidéo de 30-60 secondes

— Le ton et le style doivent s'adapter au type de véhicule (sportif, luxe, familial, écologique)

BF2 : Utilisation d'un Large Language Model

— Le système doit intégrer Google Gemini 2.5 Flash pour la génération de textes

— Les prompts doivent être optimisés pour obtenir des résultats de qualité professionnelle

— Le système doit gérer les erreurs de communication avec l'API Gemini

3.2.2 Classification des véhicules

BF3 : Détection automatique du style de véhicule

— Le système doit classifier automatiquement les véhicules en catégories (sportif, luxe, familial, écologique)

— La classification doit se baser sur les caractéristiques techniques et visuelles du véhicule

— Le système doit adapter le ton narratif en fonction de la catégorie détectée

3.2.3 Synthèse vocale

BF4 : Conversion texte vers audio

— Le système doit convertir le script généré en audio avec une voix française naturelle

— La qualité audio doit être professionnelle (format MP3, bitrate adapté)

— La synthèse vocale doit gérer correctement la prononciation des termes techniques automobiles

3.2.4 Composition vidéo

BF5 : Génération automatique de vidéos

— Le système doit composer automatiquement une vidéo à partir d'images du véhicule

— La vidéo doit synchroniser les images avec la narration audio

— Le système doit ajouter des transitions fluides entre les images

— La vidéo finale doit être exportée en format MP4 haute définition (1080p minimum)

BF6 : Personnalisation visuelle

— Le système doit permettre l'ajout de textes overlay (nom du véhicule, slogan)

— Les effets visuels doivent correspondre au style du véhicule

— Le système doit gérer différents formats d'images en entrée

3.2.5 Interface utilisateur

BF7 : Interface web intuitive

— L'utilisateur doit pouvoir uploader des images du véhicule

— L'utilisateur doit pouvoir saisir les caractéristiques techniques du véhicule

— Le système doit afficher l'état d'avancement de la génération (script, audio, vidéo)

— L'utilisateur doit pouvoir prévisualiser et télécharger la vidéo générée

BF8 : Gestion de multiples générations

— L'utilisateur doit pouvoir consulter l'historique des vidéos générées

— Le système doit permettre de régénérer une vidéo avec des paramètres modifiés

— L'utilisateur doit pouvoir supprimer des vidéos de son historique

3.3 Besoins non-fonctionnels

Les besoins non-fonctionnels définissent les contraintes et les critères de qualité du système.

3.3.1 Performance

BNF1 : Temps de génération

— Le temps total de génération d'une vidéo ne doit pas excéder 5 minutes

— La génération du script doit prendre moins de 30 secondes

— La synthèse vocale doit prendre moins de 20 secondes

— La composition vidéo doit prendre moins de 3 minutes

BNF2 : Optimisation des ressources

— Le système doit optimiser l'utilisation de la mémoire lors du traitement vidéo

— Les fichiers temporaires doivent être nettoyés après la génération

3.3.2 Fiabilité

BNF3 : Gestion des erreurs

— Le système doit gérer gracieusement les échecs d'API (Gemini, gTTS)

— Les erreurs doivent être loggées avec des informations détaillées

— L'utilisateur doit recevoir des messages d'erreur clairs et actionnables

BNF4 : Qualité des contenus

— Les scripts générés doivent être cohérents et sans erreurs grammaticales

— La qualité audio doit être professionnelle sans distorsions

— La qualité vidéo doit être HD (1920x1080 minimum)

3.3.3 Maintenabilité

BNF5 : Architecture modulaire

— Le code doit être organisé en modules indépendants (génération, synthèse, composition)

— Chaque module doit avoir une responsabilité unique et bien définie

— Le système doit permettre le remplacement facile d'un composant (ex : changer de LLM)

BNF6 : Documentation

— Le code doit être commenté et documenté (docstrings Python)

— Un guide d'utilisation doit être fourni pour les utilisateurs

— Une documentation technique doit être disponible pour les développeurs

3.3.4 Sécurité

BNF7 : Protection des données

— Les clés API doivent être stockées de manière sécurisée (variables d'environnement)

— Les fichiers uploadés doivent être validés (type, taille, contenu)

— Les données utilisateur ne doivent pas être partagées avec des tiers

3.3.5 Scalabilité

BNF8 : Évolutivité

— L'architecture doit permettre l'ajout de nouveaux styles de véhicules

— Le système doit supporter l'ajout de nouvelles langues pour la synthèse vocale

— Le système doit être prêt pour une migration vers le cloud si nécessaire

3.3.6 Utilisabilité

BNF9 : Expérience utilisateur

— L'interface doit être intuitive et ne nécessiter aucune formation

— Le système doit fournir des feedbacks visuels clairs sur l'état du traitement

— L'interface doit être responsive et accessible sur différents appareils

3.4 Contraintes techniques

3.4.1 Contraintes technologiques

— **Backend :** Python 3.10+, FastAPI

— **Frontend :** React 18+, TypeScript 5+

— **LLM :** Google Gemini 2.5 Flash

— **TTS :** gTTS (Google Text-to-Speech)

— **Composition vidéo :** MoviePy, Pillow

— **Format vidéo :** MP4 (H.264)

3.4.2 Contraintes d'environnement

— Le système doit fonctionner sur Linux et Windows

— Les dépendances doivent être gérées via pip/conda

— Le système doit supporter Python 3.10 minimum

3.5 Spécifications techniques

3.5.1 Architecture système

— **Backend API REST** avec FastAPI pour la communication frontend-backend

— **Module de génération** utilisant Google Gemini API

— **Module de synthèse vocale** utilisant gTTS

— **Module de composition vidéo** utilisant MoviePy

— **Module de classification** pour détecter le style de véhicule

— **Frontend React** avec interface utilisateur moderne

3.5.2 Format des données

— **Images :** JPEG, PNG (max 10MB par image)

— **Audio :** MP3, 128 kbps minimum

— **Vidéo :** MP4 (H.264), 1920x1080, 30 fps

— **API :** JSON pour les échanges de données

3.6 Conclusion

Ce chapitre a permis d'identifier et de formaliser l'ensemble des besoins fonctionnels et non-fonctionnels du système AutoStory. Ces spécifications constituent la base pour la phase de conception et de développement. Le chapitre suivant présentera l'architecture technique et la modélisation de la solution.

Chapitre 4

Analyse et Conception

Introduction

Dans ce chapitre, nous allons présenter l'analyse et la conception de notre solution. Cette phase cruciale du développement nous permet de définir l'architecture globale du système, de modéliser les différents composants et leurs interactions, ainsi que de spécifier les fonctionnalités à travers différents diagrammes UML.

L'objectif de cette phase est de traduire les besoins identifiés lors de l'analyse fonctionnelle en une solution technique cohérente et réalisable. Nous commencerons par présenter l'architecture générale de la solution, puis nous détaillerons la conception à travers une approche de modélisation structurée.

4.1 Architecture de la solution

4.1.1 Vue d'ensemble de l'architecture

Figure 4.1. Architecture du projet

Pour architecturer notre projet, nous avons adopté une architecture orientée **microservices** afin de garantir l'évolutivité et la flexibilité nécessaires. Cette architecture se compose de plusieurs services indépendants, chacun dédié à un rôle spécifique, collaborant via une **passerelle (gateway)** centralisée qui orchestre les communications et assure une gestion sécurisée des requêtes.

Un module commun (**common**) est également utilisé pour partager des fonctionnalités transversales, telles que les utilitaires ou les configurations standards, réduisant ainsi la redondance et favorisant la cohérence entre les services.

Cette approche offre de nombreux avantages :

— une meilleure modularité,

— une facilité de déploiement et de maintenance,

— une agilité accrue pour répondre aux besoins évolutifs de l'entreprise.

Chaque service peut être développé, testé, déployé et mis à jour indépendamment, favorisant une intégration continue et un déploiement continu alignés sur les pratiques **DevOps** adoptées par l'entreprise.

4.1.2 Architecture globale du projet

L'architecture de notre projet est conçue pour offrir une gestion modulaire et évolutive, intégrant à la fois des composants centraux et des microservices spécifiques.

Le projet **mc-starter** est le cœur de notre architecture et fournit l'infrastructure de base pour le déploiement et l'exécution de l'application. Il est structuré en deux principaux packages :

— **mc-starter-app** : contient les fichiers de configuration nécessaires pour l'initialisation et la gestion des différents environnements (développement, test, production). Les fichiers comme `application.yml` définissent les paramètres globaux et permettent de configurer les services requis.

— **mc-starter-core** : héberge la logique métier de l'application, avec des sous-packages spécialisés.

4.1.3 Approche de sécurité de l'application

Pour renforcer la sécurité de l'application, nous avons choisi, comme pour tous les projets de **Marketing Confort**, d'utiliser **Keycloak**, une solution open source dédiée à la gestion des identités et des accès.

Keycloak permet de centraliser l'authentification des utilisateurs et d'offrir des fonctionnalités essentielles telles que :

— l'authentification unique (SSO),

— l'authentification multifactorielle (MFA),

— une gestion fine des rôles, permissions et tokens d'accès.

Cette intégration nous a permis de sécuriser efficacement l'accès aux différentes parties de l'application tout en simplifiant la connexion pour les utilisateurs. Keycloak s'avère être un atout précieux dans le cadre d'une **architecture microservices**, où la sécurité et la gestion des accès doivent être homogènes et centralisées.

4.2 Conception et modélisation

La phase de conception se base sur une approche de modélisation UML (Unified Modeling Language) qui nous permet de représenter de manière claire et précise les

Figure 4.2. Approche de sécurité de l'application

différents aspects de notre système. Cette modélisation comprend trois types de diagrammes principaux qui couvrent les aspects fonctionnels, structurels et comportementaux de la solution.

4.2.1 Diagrammes de cas d'utilisation

Les diagrammes de cas d'utilisation UML modélisent les interactions entre les acteurs (utilisateurs ou systèmes externes) et les fonctionnalités d'une application. Ils offrent une vision globale du comportement attendu du système, facilitant la communication entre parties prenantes et guidant le développement.

4.2.1.1 Diagramme de cas d'utilisation du module gestion des réservations

Le diagramme ci-dessous illustre les interactions du **Client** avec le système de réservation de véhicules, incluant :

— Consulter la liste des véhicules disponibles

— Sélectionner un véhicule (avec options de filtrage)

— Effectuer une réservation (incluant choix des dates/lieu, options supplémentaires)

— Modifier/Annuler une réservation

— Consulter l'historique

Relations clés :

— **Inclusion :** L'authentification est requise pour certaines actions

— **Extension :** Le choix d'options supplémentaires (GPS, assurance) étend le cas de base « Effectuer une réservation »

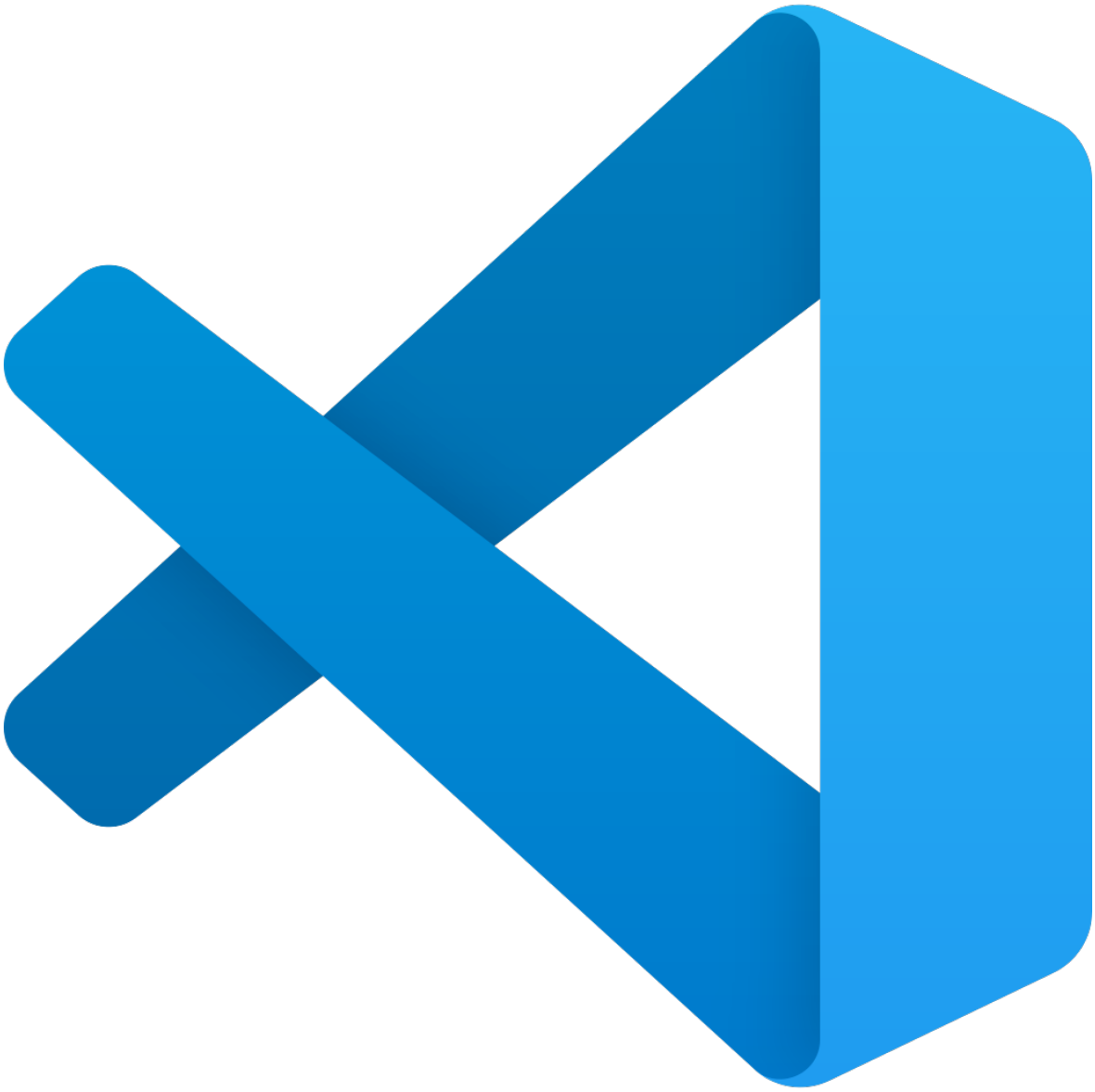

Figure 4.3. Diagramme de cas d'utilisation du module réservation

Cas d'utilisation : Effectuer une réservation

Description	Permet au client de réserver un véhicule après sélection.
Acteurs	Client
Préconditions	Client authentifié. Véhicule disponible aux dates choisies.
Scénario nominal	1. Client sélectionne un véhicule. 2. Choisit dates/lieu. 3. Ajoute des options (GPS). 4. Valide le paiement. 5. Reçoit une confirmation.
Scénario alternatif	Dates indisponibles → le système propose des créneaux alternatifs.

Table 4.1. Cas d'utilisation : Effectuer une réservation

Cas d'utilisation : Modifier une réservation

Description	Permet de modifier les détails d'une réservation existante.
Acteurs	Client, Agent
Préconditions	Réservation existante, non débutée.
Scénario nominal	1. Client accède à « Mes réservations ». 2. Modifie les dates. 3. Système calcule le coût ajusté. 4. Confirme les changements.
Scénario alternatif	Modification impossible (véhicule indisponible) → notification d'erreur.

Table 4.2. Cas d'utilisation : Modifier une réservation

4.2.1.2 Diagramme de cas d'utilisation du module gestion des véhicules

Réservé aux **Agents** et **Administrateurs**, ce diagramme couvre :

— Ajouter / Modifier / Supprimer un véhicule

— Gérer la disponibilité (maintenance, calendrier)

— Assigner un véhicule à une agence

FIGURE 4.4. Diagramme de cas d'utilisation du module véhicule

Cas d'utilisation : Ajouter un véhicule

Description	Ajoute un nouveau véhicule au catalogue.
Acteurs	Administrateur, Agent
Préconditions	Authentifié avec droits d'édition.
Scénario nominal	1. Remplit le formulaire (modèle, plaque, prix). 2. Uploade des photos. 3. Valide. 4. Système ajoute le véhicule.
Scénario alternatif	Champs incomplets → formulaire bloqué jusqu'à correction.

TABLE 4.3. Cas d'utilisation : Ajouter un véhicule

Cas d'utilisation : Assigner un véhicule à une agence

Description	Affecte un véhicule à une agence spécifique.
Acteurs	Administrateur
Préconditions	Véhicule et agence existants.
Scénario nominal	1. Sélectionne le véhicule. 2. Choisit l'agence. 3. Valide. 4. Système met à jour la disponibilité.
Scénario alternatif	Agence déjà saturée → proposition d'une autre agence.

Table 4.4. Cas d'utilisation : Assigner un véhicule à une agence

Objectif : Ces diagrammes servent de base pour les phases de développement et de tests, en clarifiant les attentes fonctionnelles.

4.2.1.3 Diagramme de cas d'utilisation du module gestion des agences

Réservé aux **Administrateurs**, ce diagramme couvre :

— Consulter la liste des agences

— Consulter les détails d'une agence spécifique

— Suivre les performances

— Ajouter une nouvelle agence

— Modifier les détails d'une agence

— Supprimer une agence

FIGURE 4.5. Diagramme de cas d'utilisation du module agences

Cas d'utilisation : Ajouter une nouvelle agence

Description	Ajoute une nouvelle agence au système.
Acteurs	Administrateur
Préconditions	Authentifié avec droits d'administration.
Scénario nominal	1. Remplit le formulaire (nom, adresse, contact, etc.). 2. Valide le formulaire. 3. Le système enregistre l'agence dans la base de données.
Scénario alternatif	Champs manquants ou invalides → message d'erreur et correction demandée.

TABLE 4.5. Cas d'utilisation : Ajouter une nouvelle agence

Cas d'utilisation : Consulter les détails d'une agence spécifique

Description	Affiche toutes les informations relatives à une agence donnée.
Acteurs	Administrateur
Préconditions	L'agence existe dans la base de données.
Scénario nominal	1. Sélectionne une agence dans la liste. 2. Le système affiche les détails (nom, adresse, employés, véhicules liés, etc.).
Scénario alternatif	Agence introuvable → message d'erreur.

Table 4.6. Cas d'utilisation : Consulter les détails d'une agence

Objectif : Ces diagrammes clarifient les fonctionnalités principales de la gestion des agences, et serviront de référence pour le développement et les tests.

4.2.2 Diagrammes de séquence

Les diagrammes de séquence illustrent les interactions entre les objets et services du système dans un ordre chronologique, permettant de représenter les scénarios fonctionnels de bout en bout.

4.2.2.1 Diagramme de séquence – Processus d'authentification

Le diagramme ci-dessous illustre le scénario d'authentification de l'utilisateur au sein de l'application MobiLoca. Il décrit les interactions entre le client, l'application mobile, la passerelle API, le système Keycloak (gestionnaire d'identité) et la base de données des utilisateurs.

Figure 4.6. Diagramme de séquence – Processus d'authentification

L'utilisateur commence par saisir ses identifiants sur l'application mobile, qui sont ensuite vérifiés par Keycloak via la passerelle API. Selon la validité des informations, un jeton d'accès sécurisé est délivré, ou un message d'erreur adapté est retourné en cas d'échec ou de compte désactivé. Ce mécanisme garantit une authentification fiable et sécurisée, essentielle pour protéger l'accès à l'application.

4.2.2.2 Diagramme de séquence – Processus de réservation

Une fois connecté, l'utilisateur peut accéder à la fonctionnalité principale de MobiLoca : la réservation d'un véhicule électrique. Le diagramme suivant illustre précisément ce processus clé, décrivant les interactions nécessaires pour sélectionner, réserver et confirmer une voiture via l'application.

FIGURE 4.7. Diagramme de séquence – Processus de réservation

Ce diagramme illustre le processus complet de réservation d'un véhicule électrique par un utilisateur via l'application MobiLoca, en mettant en évidence les échanges entre l'utilisateur, l'application mobile et les différents microservices composant l'architecture backend :

a. **Consultation du catalogue** : l'utilisateur interroge le microservice Véhicule via la passerelle API pour récupérer la liste des véhicules disponibles.

b. **Affichage des détails et options** : l'application récupère les détails du véhicule et ses packs/options (GPS, siège bébé, etc.) auprès du microservice Véhicule.

c. **Récupération des informations utilisateur** : l'application interroge le microservice Utilisateur pour obtenir ou mettre à jour les informations personnelles nécessaires.

d. **Ajout au panier** : l'utilisateur ajoute sa sélection, transmise au microservice Réservation pour un stockage temporaire.

e. **Paiement sécurisé** : l'application initie une transaction via le microservice Paiement et l'API Stripe. Selon le résultat, la réservation est confirmée ou rejetée.

f. **Confirmation de la réservation** : après paiement accepté, la réservation est créée dans le microservice Réservation, le statut du véhicule est mis à jour et un contrat est généré par le microservice Utilisateur, envoyé par email et application.

Ce scénario garantit une expérience utilisateur fluide et sécurisée, en orchestrant plusieurs services pour gérer la disponibilité, la réservation et la facturation des véhicules, tout en assurant la cohérence et la réactivité du système.

4.2.3 Diagrammes de classes

Les diagrammes de classes représentent la structure statique du système en montrant les classes, leurs attributs, leurs méthodes et les relations entre elles. Ils permettent de visualiser l'architecture des microservices et leurs dépendances internes.

4.2.3.1 Diagramme de classes de service gestion des utilisateurs

Ce diagramme de classes représente la structure du microservice utilisateur, qui centralise la gestion des différents types d'utilisateurs du système, notamment les clients, les agents enregistrés et les agents non enregistrés, en héritant tous de la classe principale **User**.

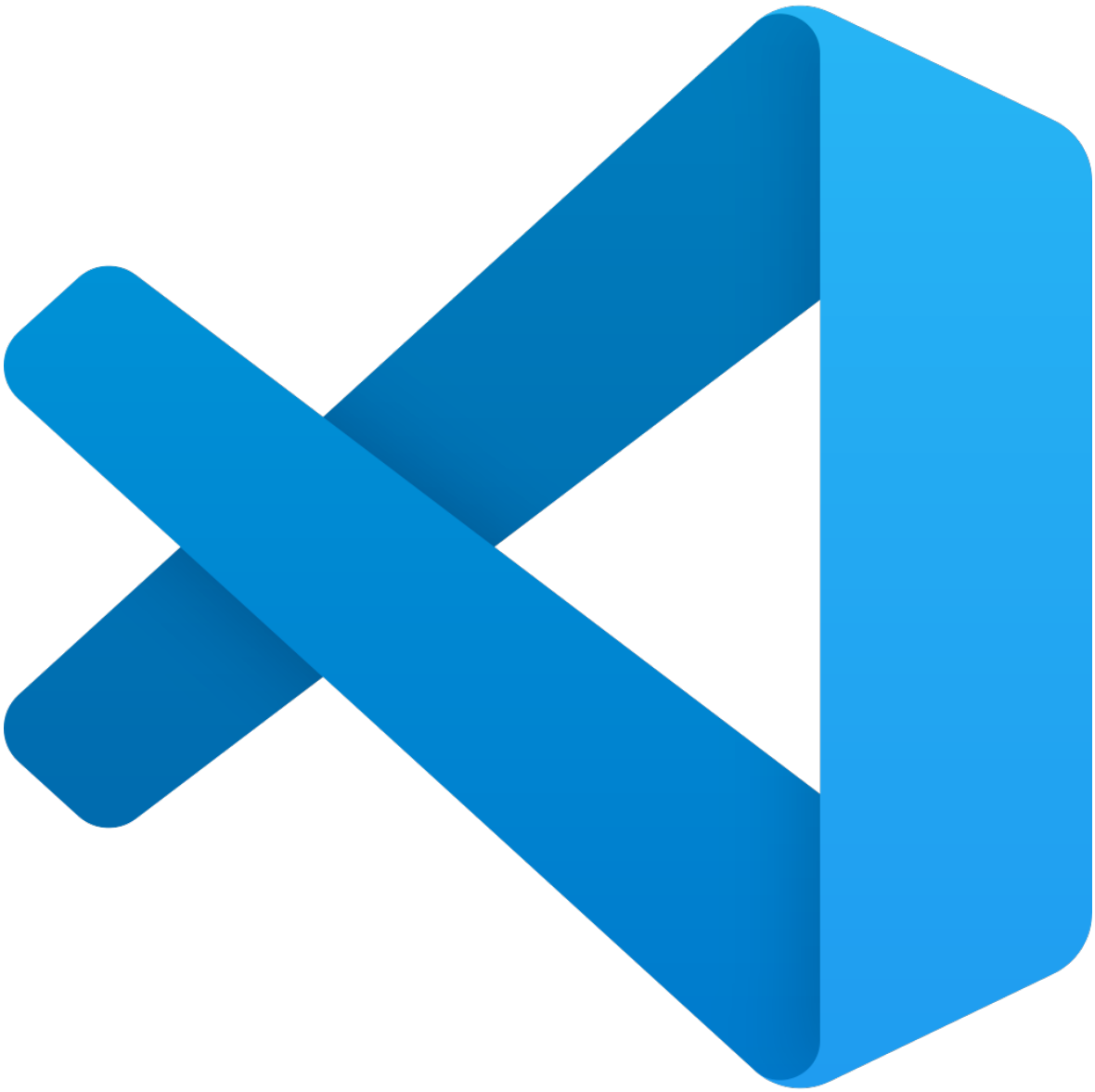

Figure 4.8. Diagramme de classes du module utilisateur

La classe **User** regroupe les informations personnelles (nom, email, téléphone, adresse, etc.). Chaque utilisateur peut avoir plusieurs activités stockées dans **ActivityHistory**. Les agents peuvent être associés à un ou plusieurs **Role**, chacun définissant un ensemble de **Permission**. Chaque permission correspond à une action (lire, créer, mettre à jour, supprimer) sur un module spécifique du système. Ce modèle met en œuvre un contrôle d'accès granulaire basé sur les rôles, permettant de sécuriser et d'adapter dynamiquement les autorisations selon les profils des utilisateurs.

4.2.3.2 Diagramme de classes de service gestion des réservations

Ce diagramme de classes illustre l'architecture du microservice de réservation, responsable de la gestion des processus de réservation de véhicules. Il modélise deux types de réservation : **SingleReservation** (réservation simple) et **GlobalReservation** (ré-

servation groupée), chacune associée à un client.

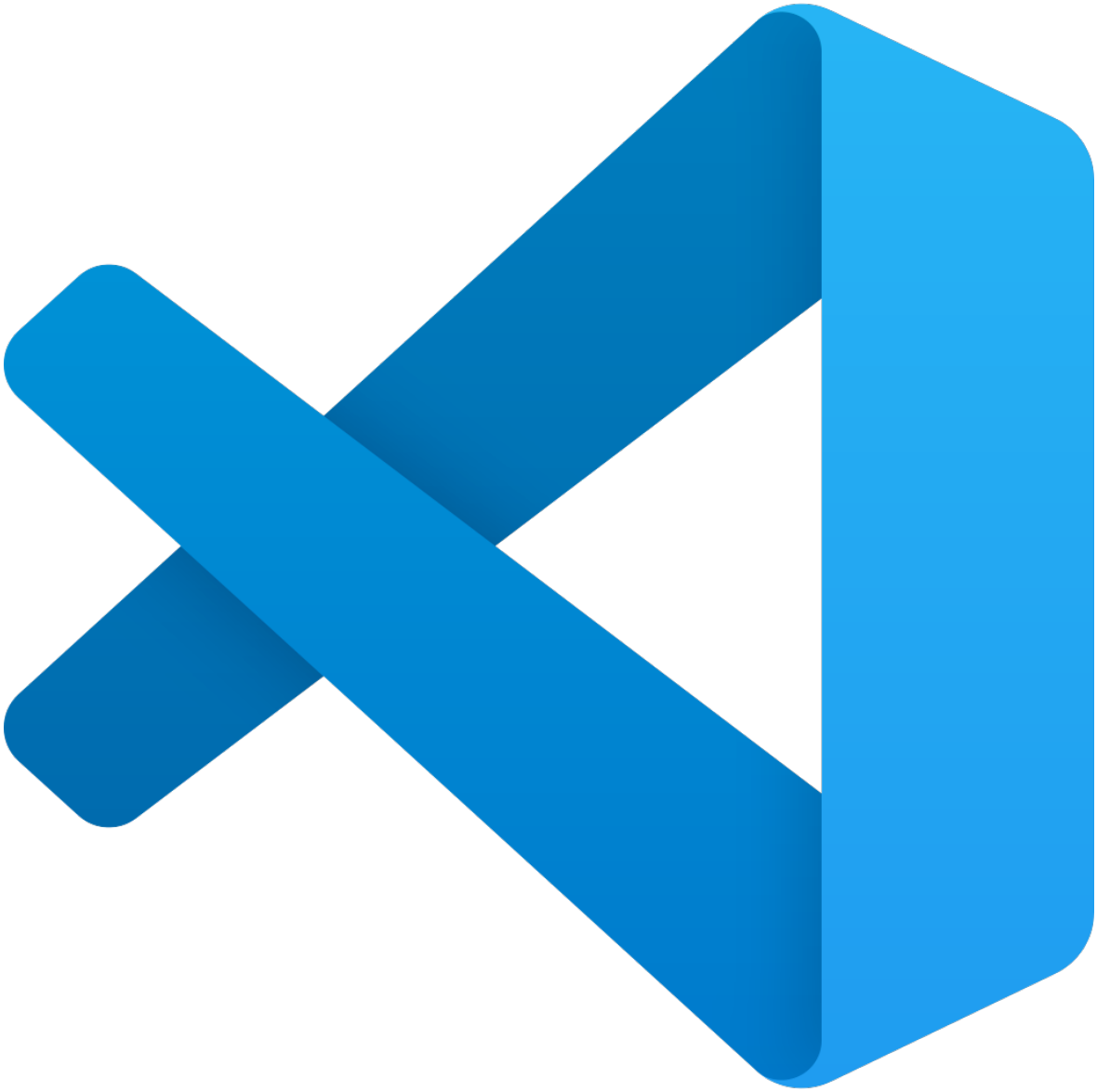

Figure 4.9. Diagramme de classes du module réservation

Chaque réservation est liée à un **Contrat**, qui formalise la relation avec le client, et à un **Deposit**, qui peut générer des pénalités en cas de non-respect des conditions contractuelles. Le dépôt peut être payé par différents moyens (carte bancaire, PayPal, etc.) et suit un cycle de validation.

4.2.3.3 Diagramme de classes de service gestion des agences

Ce diagramme de classes représente la structure du microservice agence, qui centralise la gestion des agences de location de véhicules, incluant leurs informations opérationnelles, statistiques mensuelles et rapports de performance.

Figure 4.10. Diagramme de classes du module agence

La classe principale **Agency** regroupe les informations fondamentales d'une agence (nom, email, téléphone, date de création, description, image, statut). Chaque agence possède une **Adresse** complète et des **OpeningHours** définissant ses horaires d'ouverture.

Les agences génèrent des données mensuelles stockées dans **MonthlyData**, incluant le nombre de locations et le chiffre d'affaires. Le système permet également la création de **Report** (rapports) personnalisés pouvant inclure diverses statistiques (**StatisticType**) et être organisés en dossiers via **ReportFolder**.

Chaque agence est associée à un gestionnaire (**managerId**), une liste d'agents (**agentIds**), une flotte de véhicules (**vehicleIds**) et des équipements (**equipmentIds**). Ce modèle permet une gestion complète des performances opérationnelles et financières des agences, avec une capacité avancée de reporting et d'analyse des données.

4.2.3.4 Diagramme de classes de service gestion des véhicules

Ce diagramme de classes modélise la structure du microservice responsable de la gestion des véhicules, de leurs performances, entretiens, équipements et dépenses.

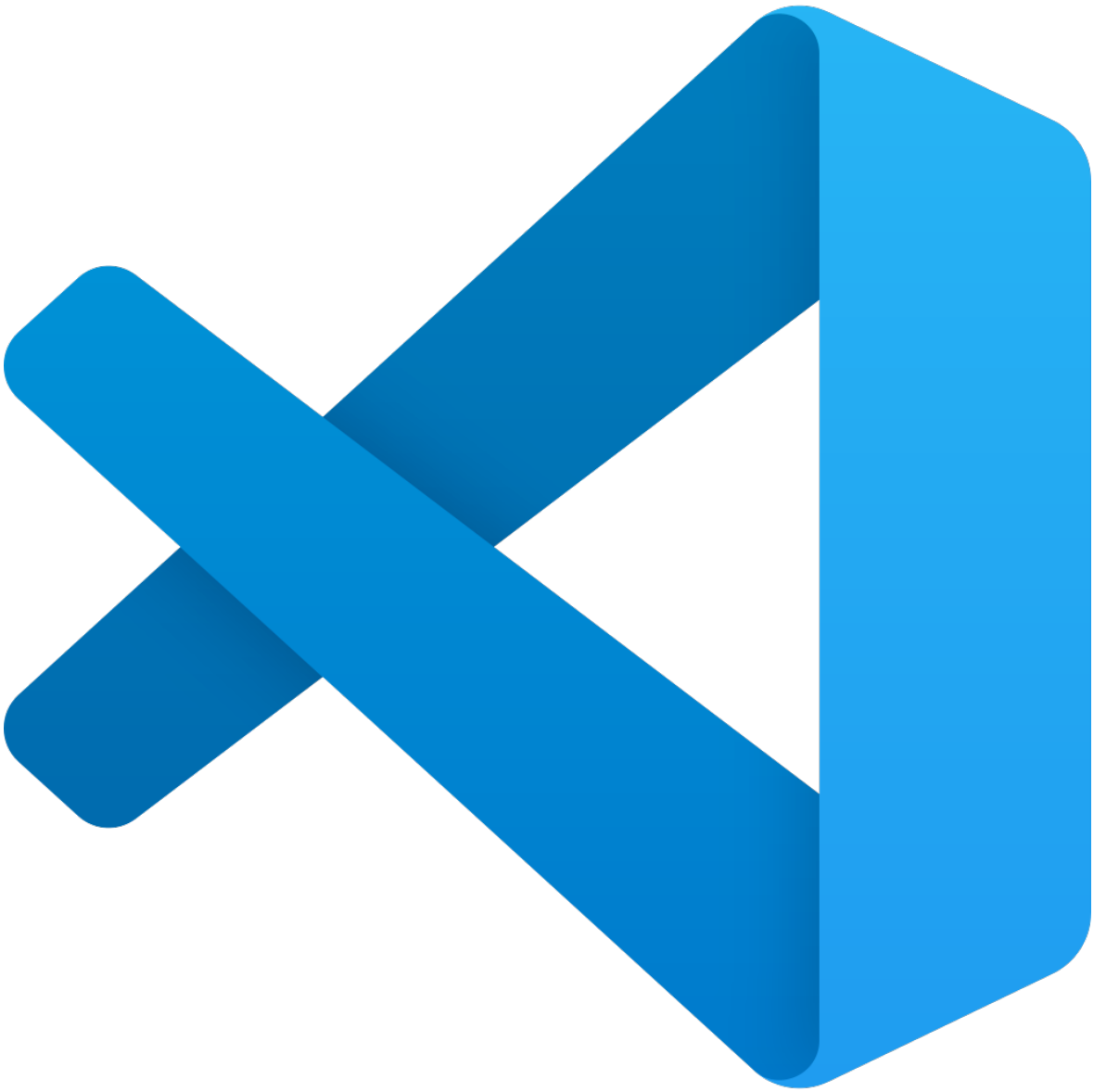

Figure 4.11. Diagramme de classes du module véhicule

La classe centrale est **Vehicle**, qui regroupe les informations essentielles sur chaque véhicule (marque, modèle, année, type de carburant, nombre de places, etc.). Chaque véhicule peut être enrichi via l'entité **VehicleEquipment**, qui permet d'associer des équipements intégrés (**BuiltInEquipment**) ou additionnels (**AddOnEquipment**), catégorisés selon **EquipmentCategory**.

Le suivi de l'activité des véhicules est organisé via la classe **CalendarActivities**, qui se décline en trois sous-types :

— **ReservationActivity** (liée à une réservation),

— **MaintenanceActivity** (planification d'entretien),

— **RegularActivity** (autres activités régulières).

La performance des véhicules est suivie via **VehiclePerformance**, incluant des mesures telles que le kilométrage, la consommation moyenne ou la vitesse maximale. La classe **Expense** enregistre toutes les dépenses associées aux véhicules (carburant, assurance, réparation, etc.), avec leur **ExpenseCategory** et **ExpenseStatus** (Validée ou Rejetée).

4.2.3.5 Diagramme de classes du module Fleet

Ce diagramme de classes illustre l'architecture du microservice **Fleet**, responsable de la gestion post-réservation et du suivi opérationnel des véhicules. Il modélise quatre entités principales : **Complaint** (réclamation), **FuelConsumption** (consommation de carburant), **Incident** et **MileageRecord** (enregistrement kilométrique).

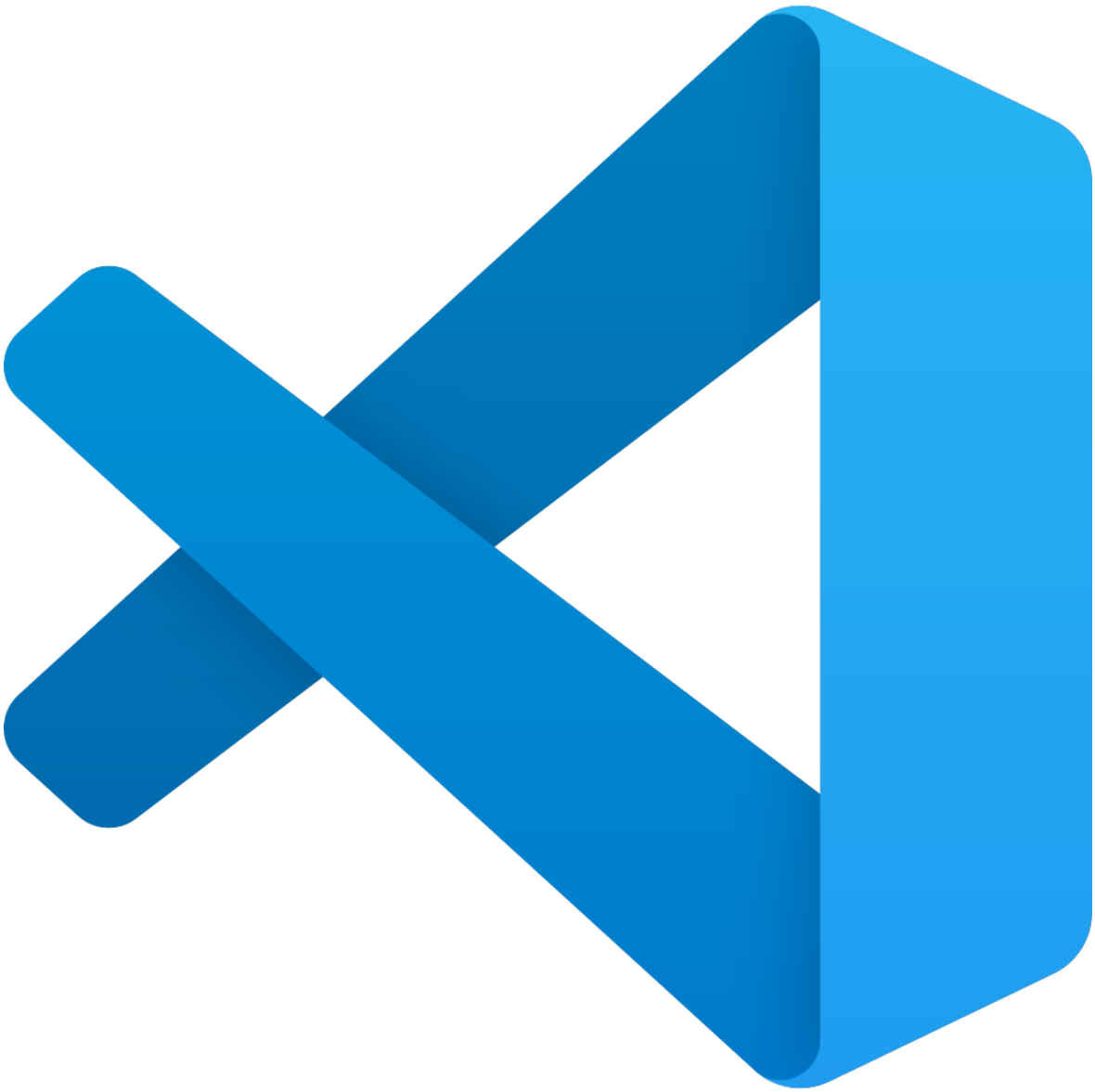

Figure 4.12. Diagramme de classes du module Fleet

Chaque entité est associée à des énumérations spécifiques définissant leurs états et types. Les réclamations (**Complaint**) peuvent être de différents types (facturation, dommages véhicule, qualité de service) et suivent un cycle de traitement avec des statuts allant de `OPEN` à `RESOLVED`. Les incidents couvrent les pannes, dommages, accidents et vols, avec un suivi de leur traitement. La consommation de carburant et les dépassements kilométriques sont liés au statut de paiement (`PENDING`/`BILLED`), permettant la facturation des frais supplémentaires.

Ainsi, ce module assure le suivi complet des événements post-location et la gestion des coûts additionnels.

Conclusion

Ce chapitre a présenté l'analyse et la conception complète de notre solution. L'architecture proposée répond aux besoins identifiés tout en garantissant la scalabilité, la maintenabilité et la performance du système.

La modélisation UML réalisée à travers les diagrammes de cas d'utilisation, de classes et de séquence nous a permis de :

— Clarifier les fonctionnalités attendues et les interactions avec les utilisateurs

— Structurer l'organisation des données et des traitements

— Définir précisément les flux d'exécution des principales fonctionnalités

Cette phase de conception constitue la base solide sur laquelle s'appuiera la phase d'implémentation présentée dans le chapitre suivant. Les choix architecturaux et les modèles définis guideront le développement et assureront la cohérence technique de la solution finale.

Chapitre 5

Réalisation et Mise en Œuvre

5.1 Introduction

5.2 Outils de développement et technologies

5.2.1 Outils de développement

5.2.2 Technologies de développement

5.2.2.1 Backend

5.2.2.2 Frontend

5.2.3 Outils et services complémentaires

5.2.3.1 Sécurité et droits d'accès

5.2.3.2 Base de données

5.2.3.3 Tests et développement API

5.2.4 Mise en place du CI/CD et du versioning

5.3 Tests et Validation

5.3.1 Tests unitaires

5.3.2 Validation des données

5.3.2.1 Backend

5.3.2.2 Frontend

5.4 Présentation des Interfaces Utilisateur

5.4.1 Front Office (Application Mobile)

5.4.1.1 Interface de connexion (Login)

5.4.1.2 Page d'accueil et catalogue de véhicules

5.4.1.3 Interface de filtres avancés

Chapitre 6

Conclusion et perspectives

Conclusion Générale

Au terme de ce **Projet de Fin d'Année (PFA)**, réalisé au sein de **Marketing Confort** pour le client **Adanev**, nous avons conçu et développé **MobiLoca**, une plateforme numérique innovante destinée à moderniser la gestion de la location de véhicules adaptés. Cette solution s'articule autour d'un **back-office web** pour les agents et d'une **application mobile** pour les clients, permettant une gestion centralisée, une automatisation des processus et une expérience utilisateur optimisée.

Ce projet m'a offert une opportunité précieuse de mettre en pratique et de renforcer mes compétences en **développement logiciel**, en **architecture microservices** et en **intégration de solutions sécurisées**. Nous avons mobilisé des technologies modernes telles que **Java 17, Spring Boot, React Native et Next.js**, ainsi que des outils spécialisés comme **Stripe** pour les paiements et **Keycloak** pour la gestion des accès. L'adoption de la méthodologie agile **Scrum** (sprints de deux semaines, feedback utilisateur continu) a permis d'adapter les fonctionnalités aux besoins réels et de livrer une solution fiable et évolutive.

Ce travail m'a également permis de développer des compétences transversales essentielles :

- maîtrise du **cycle de vie d'un projet logiciel** en contexte professionnel,
- conception et intégration d'une **architecture basée sur les microservices**,
- **collaboration en équipe pluridisciplinaire** et gestion des interactions,
- communication efficace et **adaptation aux exigences d'un client réel**.

Par ailleurs, **MobiLoca** ouvre la voie à plusieurs perspectives d'évolution :

- intégration de **modules d'intelligence artificielle** pour la prédiction de la demande et la maintenance proactive ;
- ajout d'un **chatbot intelligent** et de fonctionnalités d'**OCR** pour améliorer l'expérience client ;
- migration vers des applications mobiles natives (**Kotlin/Swift**) afin d'optimiser les performances ;

— adoption d'une **API GraphQL** pour renforcer l'interopérabilité avec les systèmes tiers ;

— déploiement **cloud** avec **Kubernetes**, garantissant scalabilité, haute disponibilité et fiabilité.

En conclusion, ce **PFA** a représenté une étape déterminante dans mon parcours académique et professionnel. Il démontre comment une architecture moderne, associée à des choix technologiques pertinents, peut transformer la gestion interne d'une entreprise tout en améliorant l'expérience de ses clients. Je suis fière d'avoir contribué à cette réalisation et convaincue que les compétences acquises dans ce cadre constitueront un socle solide pour ma future carrière dans le domaine de l'ingénierie logicielle.

Webliographie

[1] Charika.ma. *Fiche d'identité — Marketing Confort*. 2025. URL : https://www.charika.ma/societe-marketing-confort-963077 (visité le 25/08/2025).

[2] diagrams.net. *Draw.io : Online diagram software*. Anglais. URL : https://www.diagrams.net/ (visité le 30/08/2025).

[3] Microsoft. *Visual Studio Code*. Anglais. URL : https://code.visualstudio.com/ (visité le 30/08/2025).

[4] JetBrains. *IntelliJ IDEA*. Anglais. URL : https://www.jetbrains.com/idea/ (visité le 30/08/2025).

[5] JetBrains. *DataGrip*. Anglais. URL : https://www.jetbrains.com/datagrip/ (visité le 30/08/2025).

[6] Pivotal Software. *Spring Boot*. Anglais. URL : https://spring.io/projects/spring-boot (visité le 30/08/2025).

[7] Microsoft. *TypeScript*. Anglais. URL : https://www.typescriptlang.org/ (visité le 30/08/2025).

[8] Vercel. *Next.js*. Anglais. URL : https://nextjs.org/ (visité le 30/08/2025).

[9] Meta Platforms. *React Native*. Anglais. URL : https://reactnative.dev/ (visité le 30/08/2025).

[10] Red Hat. *Keycloak*. Anglais. URL : https://www.keycloak.org/ (visité le 30/08/2025).

[11] PostgreSQL Global Development Group. *PostgreSQL*. Anglais. URL : https://www.postgresql.org/ (visité le 30/08/2025).

[12] Postman, Inc. *Postman*. Anglais. URL : https://www.postman.com/ (visité le 30/08/2025).

[13] Amazon Web Services. *AWS (Amazon Web Services)*. Anglais. URL : https://aws.amazon.com/ (visité le 30/08/2025).

Annexe A

Annexes

Annexes

Annexe 1 : Diagrammes complémentaires

— Diagrammes UML supplémentaires (séquence, activités, etc.) non inclus dans le corps du rapport.

Annexe 2 : Extraits de code

— Exemples d'implémentation des microservices.

— Configurations principales (Spring Boot, sécurité avec Keycloak, intégration Stripe).

Annexe 3 : Documentation technique

— Guide de déploiement de la plateforme.

— Captures d'écran supplémentaires des interfaces web et mobiles.