



République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique



RAPPORT DE PROJET D'INTÉGRATION

Réalisé par :

Ikram Guesmi, Eya Farhaoui et Wissal Soltani

Système de Gestion de Location de Voitures entre particuliers

Mme Mariem Baccouche

Encadré par :

M.Faouzi Maddouri

Réalisé au sein de l'**ISET Rades**



Année Universitaire : 2024-2025

Table des matières

Introduction générale	1
1 Présentation du cadre du projet	2
1.1 Présentation de l'ISET de Rades	3
1.1.1 Missions de l'ISET Rades	3
1.1.2 Contexte du projet	4
1.2 Présentation du projetl	4
1.2.1 Description de l'existant	4
1.2.2 Critique de l'existant	4
1.2.3 Solution proposée	5
1.3 Méthodologie de conception et Langage de modélisation	5
1.3.1 Choix de la méthodologie de conception	5
1.3.2 Langage de modélisation	6
2 Planification du cadre du projet	8
2.1 Identification des besoins fonctionnels et non fonctionnels	9
2.1.1 Les besoins fonctionnels	9
2.1.2 Besoins non fonctionnels :	10
2.2 Les Diagramme utilisé	10
2.2.1 Diagramme du cas d'utilisation global	10
2.2.2 diagramme de déploiement	12
2.3 Pilotage du projet avec Scrum	13
2.3.1 backlogProduit	13
2.4 Environnement de développement	14
2.4.1 Environnement matériel	14
2.4.2 Environnement logiciel	15
2.5 Architecture Logicielle	18
2.5.1 Architecture SOA	18
2.5.2 Modèle MVC au niveau Backend et Frontend	19
2.5.3 Avantages de cette architecture	22

3 Sprint 1 : Mise en Place des Fonctionnalités Fondamentales pour les Clients et Propriétaires	23
3.1 Présentation de l'équipe	24
3.2 Objectifs	24
3.3 Sprint Backlog	25
3.4 Conception	26
3.4.1 Diagramme de classe	26
3.4.2 Diagramme de séquence :	26
3.5 Les interfaces	28
3.5.1 Interface de l'accueil :	28
3.5.2 Interface des offres disponibles	30
3.5.3 Interface d'authentification pour le propriétaire	30
3.5.4 Interface de publication d'annonces du propriétaire	31
3.5.5 Interface de publication d'annonces des Clients	32
3.6 Tests effectuées	33
3.6.1 Test Unitaire	33
3.6.2 Test d'Intégration Composants	34
3.6.3 Test d'Intégration Système	34
3.6.4 Test d'Acceptation Utilisateur	34
4 sprint 2 : Implémentation des Fonctionnalités Clés	36
4.1 Présentation de l'équipe	37
4.2 Objectifs	37
4.3 sprint Backlog	38
4.4 Conception	38
4.4.1 Diagramme de classe	38
4.4.2 Diagramme de séquence :	39
4.5 Les interfaces :	41
4.5.1 Interface d'authentification pour le Client	41
4.5.2 Interface pour la réservation de voiture	41
4.5.3 Interface de Consultation de la liste des réservations propres à un Client	42
4.6 Tests :	43
4.6.1 Test Unitaire	43
4.6.2 4.3.2 Test d'Intégration	44

4.6.3	4.3.3 Test d'Acceptation	45
5	sprint 3 : Implémentation des Fonctionnalités Clés	46
5.1	Présentation de l'équipe	47
5.2	Objectifs	47
5.3	Sprint Backlog	48
5.4	Conception :	48
5.4.1	Diagramme de séquence :	49
5.5	Les interfaces :	50
5.5.1	Interface du propriétaire pour consulter les avis faits sur un client	50
5.5.2	Interface du propriétaire pour consulter et répondre sur les avis faits sur lui	51
5.5.3	Interface du propriétaire pour donner son avis sur un client	51
5.5.4	Interface du propriétaire pour consulter ses avis et les réponses faites dessus	52
5.5.5	Interface du client pour consulter et répondre sur les avis faits sur lui	52
5.5.6	Interface du client pour consulter ses avis et les réponses faites dessus	53
5.6	Tests	54
5.6.1	Test Unitaire	54
5.6.2	Test d'Intégration	55
5.6.3	Test d'Acceptation	56
6	sprint 4 : Implémentation des Fonctionnalités Clés	58
6.1	Présentation de l'équipe	59
6.2	Objectifs	59
6.3	Sprint Backlog	60
6.4	Conception :	60
6.4.1	Diagramme de séquence :	61
6.5	Les interfaces :	62
6.5.1	Tableau de bord du propriétaire :	62
6.5.2	Tableau de bord du Client :	63
6.5.3	Création du compte de l'agent :	64
6.5.4	Connexion de l'agent :	65
6.5.5	Réclamation du propriétaire :	66
6.5.6	La liste des réclamation des propriétaire :	67
6.6	Tests	67
6.6.1	Test Unitaire	67

7 Application Mobile pour les clients	69
7.1 Connexion à la base de données	70
7.1.1 Configuration du client Retrofit (<code>ApiClient.java</code>)	70
7.1.2 Définition des services API (<code>ApiService.java</code>)	72
7.2 Package com.example.eliteauto.model	73
7.2.1 Classe AvisProprietaire	73
7.2.2 Classe client	74
7.2.3 Classe Disponibilite	75
7.3 Classe Voiture	76
7.4 Package com.example.eliteauto.adapter	80
7.4.1 AvisAdapter	80
7.4.2 LocalDateAdapter	82
7.4.3 VoitureAdapter	82
7.5 connexion via Facebook	86
7.5.1 Explication de la logique du login Facebook :	86
7.5.2 LoginActivity (Authentification via Facebook)	86
7.5.3 AndroidManifest.xml	90
7.5.4 values/strings.xml	91
7.6 Fichiers XML	92
7.6.1 activity_main.xml	92
7.6.2 item_voiture.xml	94
7.6.3 menu.xml	98
7.6.4 detailvoiture.xml	100
7.7 Vue d'ensemble du fichier MainActivity.java	106
7.7.1 Usage des intents	106
7.7.2 RecyclerView Setup	107
7.7.3 SwipeRefreshLayout	107
7.7.4 VoitureAdapter	107
7.7.5 Glide pour le chargement des images	107
7.7.6 OnItemClickListener	108
7.8 La Classe DetailsActivity	112
7.8.1 Initialisation de la Vue	112
7.8.2 Récupération des Données depuis l'Intent	112
7.8.3 Affichage de l'Image et des Informations	112

7.8.4	Interaction avec le Bouton de Réservation	112
7.9	Package com.example.eliteauto.Service.DisponibiliteService	115
7.9.1	Principaux Composants et Fonctionnalités	115
7.10	Les tests faits	124
7.10.1	introduction	124
7.10.2	Conclusion	128
	Conclusion générale	129

Table des figures

1.1	Logo de l'ISET Rades	3
1.2	Mode de fonctionnement Scrum	6
1.3	logo UML	7
2.1	Diagramme de cas d'utilisation	11
2.2	Diagramme de déploiement	12
2.3	Logo Angular	15
2.4	Logo SpringBoot	15
2.5	Système de gestion de bases de données Oracle	16
2.6	Langage de programmation JAVA	16
2.7	Visual Studio Code	17
2.8	Postman	17
2.9	Architecture SOA	19
2.10	Modèle MVC	21
3.1	Diagramme de classe du sprint 1	26
3.2	Authentification propriétaire	27
3.3	Publication du client	27
3.4	Recherche de voiture par date	28
3.5	Accueil du site	29
3.6	Les offres disponibles	30
3.7	Connexion du propriétaire	31
3.8	Publication du propriétaire	32
3.9	Publication du client	32
3.10	Publication du client	33
4.1	Diagramme de classe du sprint 2	39
4.2	Diagramme de séquence du scénario de proposition d'une offre	40
4.3	Diagramme de séquence du scénario de réservation d'une voiture	40
4.4	Connexion du Client	41
4.5	Réservation d'une voiture	42
4.6	liste de reservation propre à un proprietaire	43

5.1	Diagramme de classe du sprint 3	49
5.2	Diagramme de séquence du sprint 3	50
5.3	Vue de « Consultation des avis faits sur un client»	51
5.4	Vue de « Consultation et réponse du propriétaire sur les avis faits sur lui»	51
5.5	Vue de « ajout du propriétaire d'un avis sur un client»	52
5.6	Vue de « consultation du propriétaire de la liste de ses avis et des réponses des clients faites dessus »	52
5.7	Vue de « Consultation et réponse du client sur les avis faits sur lui»	53
5.8	Vue de « consultation du client de la liste de ses avis et des réponses des propriétaires faites dessus »	53
6.1	Diagramme de classe du sprint 4	61
6.2	Diagramme de séquence du sprint 4	62
6.3	Vue de « La premiere partie du tableau de bord du propriétaire»	63
6.4	Vue de « La deuxième partie du tableau de bord du propriétaire»	63
6.5	Vue de « Tableau de bord du Client»	64
6.6	Vue de « email de création du compte d'un agent»	65
6.7	Vue de « connexion d'un agent»	66
6.8	Vue de « l'ajout d'une réclamation par un propriétaire»	66
6.9	Vue de « La liste des réclamation des propriétaire»	67
7.1	Layout de l'activité principale	94
7.2	Layout de itemvoiture	98
7.3	Layout de menu	100
7.4	Layout de l'activité detail voiture	106
7.5	demarrage de service en arriere plan	118
7.6	liste des voitures disponible	119
7.7	details de voiture	120
7.8	Liste des avis faits sur un propriétaire	121
7.9	menu	122
7.10	Connexion du client	123
7.11	Dashbord du client	124

Liste des tableaux

3.1 Backlog du Sprint 1	25
4.1 Backlog du Sprint	38
5.1 Backlog du Sprint	48
6.1 Backlog du Sprint	60
7.1 Tableau des tests effectués	127

Introduction générale

Dans le cadre de notre projet d'intégration, nous avons été amenés à travailler sur le développement d'une plateforme innovante de location de voitures entre particuliers. Ce projet s'inscrit dans une démarche visant à résoudre les problèmes couramment rencontrés dans ce domaine, notamment la difficulté de garantir la confiance entre les propriétaires de voitures et les clients, due à l'absence de systèmes de notation fiables et de vérification des profils.

Pour répondre à ces enjeux, nous avons adopté la méthodologie agile **Scrum**, qui nous a permis de collaborer efficacement en équipe et de livrer des incrémentations fonctionnelles à chaque sprint. Notre solution repose sur le développement :

- d'un **site web** pour offrir une expérience utilisateur fluide aux propriétaires et aux clients,
- et d'une **application mobile** pour garantir une accessibilité optimale et une gestion intuitive des réservations.

L'objectif principal est de proposer une plateforme complète qui facilite les interactions entre les utilisateurs, tout en répondant à des besoins tels que :

- la mise en place d'un système de notation pour renforcer la transparence,
- la sélection de véhicules selon des critères spécifiques,
- la gestion des demandes et des offres,
- ainsi que le suivi des transactions.

Ce rapport décrit les étapes clés de notre démarche, les choix techniques réalisés, ainsi que les résultats obtenus au cours de ce projet.

PRÉSENTATION DU CADRE DU PROJET

Plan

1	Présentation de l'ISET de Rades	3
2	Présentation du projetl	4
3	Méthodologie de conception et Langage de modélisation	5

Introduction

Ce premier chapitre est consacré à la présentation du projet dans son contexte général. Nous détaillons d'abord le cadre académique de l'ISET de Radès, puis nous procédons à l'étude de l'existant, en mettant en évidence les critiques et problèmes qui en découlent pour proposer les solutions à adopter.

1.1 Présentation de l'ISET de Rades

L'Institut Supérieur des Études Technologiques (ISET) de Rades, créé dans le cadre du réseau des ISET en Tunisie, est un établissement public d'enseignement supérieur et de recherche.

L'ISET de Radès a pour mission principale de former des cadres techniques et des professionnels dans divers domaines technologiques. Il joue également un rôle central dans l'accompagnement des étudiants dans leurs projets académiques et professionnels.

Cet institut se distingue par son approche pédagogique orientée vers la pratique et l'intégration de projets innovants. Il encourage les étudiants à réaliser des projets d'intégration qui combinent leurs compétences techniques, leur esprit d'analyse et leur capacité à résoudre des problématiques concrètes dans divers secteurs.

1.1.1 Missions de l'ISET Rades

Dans le cadre de ses activités, l'ISET Rades poursuit plusieurs objectifs stratégiques :

- Former des techniciens et des ingénieurs spécialisés répondant aux besoins du marché.
- Favoriser le développement de compétences pratiques à travers des projets d'intégration.
- Soutenir l'innovation et l'utilisation des nouvelles technologies dans l'enseignement et la recherche.
- Renforcer les liens avec les entreprises et les acteurs économiques régionaux pour assurer une meilleure insertion professionnelle des diplômés.



FIGURE 1.1 : Logo de l'ISET Rades

1.1.2 Contexte du projet

Le projet présenté dans ce rapport s'inscrit dans le cadre des projets d'intégration de l'ISET Radès. Ces projets ont pour but de permettre aux étudiants de mettre en pratique les connaissances acquises au cours de leur formation et de développer des solutions concrètes répondant à des problématiques réelles.

1.2 Présentation du projet

Ce projet consiste à développer une plateforme qui gère la location des voitures entre particuliers. Cette dernière combine un site web et une application mobile qui offrent des fonctionnalités similaires. En fait, ils permettent principalement aux clients (futurs locataires) d'ajouter des demandes de location, de consulter la liste des voitures disponibles et de réserver d'une façon sécurisée une voiture . De leur côté, les propriétaires peuvent ajouter des offres de location et consulter les demandes effectuées par les clients. En fait, ce projet représente une plateforme de rencontre entre les propriétaires des voitures et les personnes à la recherche d'un véhicule.

1.2.1 Description de l'existant

Actuellement, le processus de location repose sur des interactions physiques ou des échanges directs entre particuliers. Cette méthode présente plusieurs limitations :

- **Mode de fonctionnement** : Les propriétaires et locataires se rencontrent en personne pour discuter des termes et finaliser la transaction.
- **Absence de centralisation** : Il n'existe pas de plateforme dédiée pour faciliter les échanges ou regrouper les offres.
- **Communication limitée** : Les informations sur les véhicules disponibles ou les besoins des locataires sont partagées de manière informelle, souvent via des réseaux sociaux ou des contacts directs.
- **Promotions restreintes** : Les propriétaires n'ont pas d'outil structuré pour promouvoir leurs offres à grande échelle.

Ces contraintes freinent l'expansion et la fluidité des services de location de voitures entre particuliers.

1.2.2 Critique de l'existant

Les limitations actuelles entraînent plusieurs problématiques :

- **Perte de temps** : La nécessité de rencontres en personne et la recherche manuelle d'un véhicule

approprié allongent inutilement les délais.

- **Manque de transparence :** Les utilisateurs (locataires et propriétaires) ne disposent pas d'outils pour vérifier la fiabilité et la réputation de l'autre partie.
- **Processus de paiement peu sécurisé :** Les transactions financières se font de manière informelle, augmentant les risques de fraude ou de litige.
- **Manque d'évolutivité et Absence de fonctionnalités modernes :** Le modèle actuel n'intègre pas d'éléments essentiels comme les évaluations des utilisateurs, les historiques de transactions, les dashboards ou les outils d'analyse de données.

1.2.3 Solution proposée

Pour répondre aux problèmes identifiés, nous proposons de développer une application web et mobile innovante, intégrant les fonctionnalités suivantes :

- **Publication d'offres :** Les propriétaires pourront publier leurs véhicules avec des critères détaillés (modèle, prix, état, etc.).
- **Recherche et réservation :** Les locataires auront accès à une interface de recherche avancée leur permettant de filtrer selon leurs besoins, et pourront réserver directement en ligne.
- **Gestion des paiements :** Intégration d'un système de paiement sécurisé en deux étapes (un tiers lors de la réservation, le reste lors de la location).
- **Notation et transparence :** Les locataires et propriétaires pourront s'évaluer mutuellement après chaque transaction, renforçant la confiance et la transparence.
- **Support utilisateur :** Une assistance en ligne sera disponible pour répondre aux questions des clients et des propriétaires et pour résoudre les problèmes.
- **Accessibilité multiplateforme :** L'application sera accessible sur le web et les appareils mobiles, garantissant ainsi une utilisation fluide et universelle.

Ces solutions visent à créer une expérience utilisateur optimale et à révolutionner le marché de la location de voitures entre particuliers.

1.3 Méthodologie de conception et Langage de modélisation

1.3.1 Choix de la méthodologie de conception

Pour la réalisation de notre projet, nous avons adopté le Framework **SCRUM**, faisant partie des méthodes agiles pour le cycle de développement d'un logiciel, mais également pour la gestion et le suivi de nos activités journalières tout au long de la période de son réalisation.

Le choix de cette méthodologie s'explique par les spécificités suivantes liées à notre projet :

— **Contrôle régulier et gestion proactive des obstacles :**

Notre projet nécessite un suivi quotidien pour vérifier les progrès réalisés et identifier les obstacles éventuels. Scrum répond à ce besoin grâce à son principe d'Inspection, qui consiste à examiner régulièrement l'avancement du travail afin de détecter et de traiter rapidement les problèmes.

— **Communication et collaboration optimales :**

La réalisation de notre projet au sein d'une équipe exige une communication fluide et une coopération efficace. Ces aspects sont garantis par le principe de Transparence dans Scrum, qui favorise une compréhension commune et claire de l'état du projet parmi tous les membres de l'équipe.

— **Adaptabilité aux mises à jour fréquentes et aux données évolutives :**

Notre projet doit être adaptable aux évolutions fréquentes des données et aux mises à jour de la plateforme. Cela est assuré par la méthodologie Scrum, dont l'un des principes fondamentaux est l'adaptation. Ce principe repose sur la capacité d'apporter les ajustements nécessaires pour garantir l'atteinte des objectifs, en réponse aux besoins changeants.

Ces spécificités justifient notre choix d'adopter l'agilité comme méthodologie de développement, en utilisant le framework **SCRUM**.

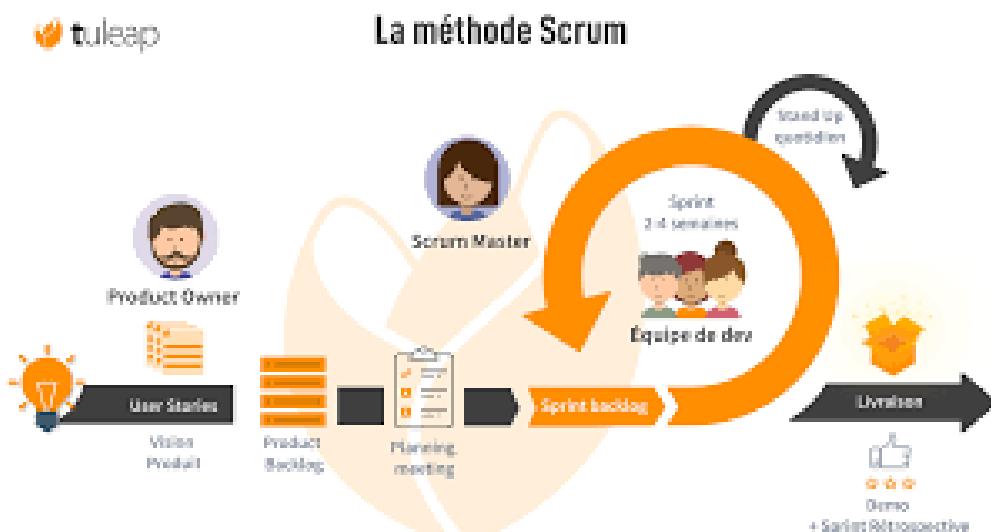


FIGURE 1.2 : Mode de fonctionnement Scrum

1.3.2 Langage de modélisation

Pour concevoir notre projet, et pour visualiser, spécifier, construire et documenter nos artefacts logiciels nous avons besoin d'un langage de modélisation qui peut nous permettre de réaliser des **diagrammes structurels** comme les diagrammes de classes et d'objets, et des **diagrammes comportementaux**.

comme les diagrammes de séquence et de cas d'utilisationopté.

Pour cela, on a opté pour **UML** (*Unified Modeling Language*) qui est couramment utilisé dans les projets logiciels,et qui offre une multitude de diagrammes permettant ainsi de donner une représentation informatique d'un ensemble d'éléments et de problèmes standards du monde réel.



FIGURE 1.3 : logo UML

L'utilisation d'UML dans notre projet nous a permis de clarifier et d'organiser efficacement les différentes étapes de conception.

Conclusion

Dans ce chapitre, nous avons présenté le cadre académique dans lequel s'inscrit notre projet.

Ensuite, nous avons procédé à l'étude de l'existant, mettant en lumière les lacunes des solutions actuelles pour la location de voitures entre particuliers. Ces analyses nous ont permis d'identifier des problématiques spécifiques et de proposer des solutions techniques et fonctionnelles pour pallier ces insuffisances.

Enfin, nous avons présenté la méthodologie adoptée pour la conception de notre projet, basée sur le Framework **SCRUM**, et le langage de modélisation **UML**, qui ont été sélectionnés pour leur pertinence dans un contexte de développement agile et structuré.

PLANIFICATION DU CADRE DU PROJET

Plan

1	Identification des besoins fonctionnels et non fonctionnels	9
2	Les Diagramme utilisé	10
3	Pilotage du projet avec Scrum	13
4	Environnement de développement	14
5	Architecture Logicielle	18

Introduction

Au niveau de ce chapitre, nous allons aborder la spécification des besoins, où nous allons dégager les besoins fonctionnels et non fonctionnels auxquels doit répondre notre application. Et nous allons présenter le Product Backlog.

2.1 Identification des besoins fonctionnels et non fonctionnels

2.1.1 Les besoins fonctionnels

Gestion des Véhicules et Offres :

- Publication d'un véhicule par les propriétaires avec détails complets : modèle, état, nombre de portes, type de carburant, boîte de vitesses, nombre de passagers, prix par jour, montant de la caution.
- Consultation par les clients des offres disponibles, avec possibilité de filtrer selon le type de véhicule, les dates, et les lieux de retrait et de retour.
- Demande de location par les client .
- Acceptation des demandes de location par les propriétaires.

Réservation et Paiement :

- Réservation en ligne par le client avec paiement d'un tiers du montant total lors de la réservation.
- Paiement du reste (les deux tiers) et de la caution directement au propriétaire le jour de la location.
- Suivi des paiements des propriétaires par l'agent de location.

Gestion des Retours et Notation :

- Notation et avis par les clients sur les propriétaires après la location pour assurer transparence et confiance. .
- Notation par les propriétaires sur les locations.
- Consultation des avis sur les propriétaires par un locataire .
- Consultation des avis sur les locations par les propriétaires .

Support et suivi :

- Assistance en ligne pour les locataires et propriétaires en cas de problème.
- Suivi des paiements et gestion des transactions par les agents de location.

2.1.2 Besoins non fonctionnels :

Facilité d'utilisation :

- Interface utilisateur intuitive : Le Site doit être facile à naviguer pour les futurs locataires novices et offrir une expérience utilisateur fluide avec une organisation claire des fonctionnalités.
- Pour les agents, le système doit être suffisamment intuitif pour qu'un utilisateur puisse être formé en moins de 2 heures pour utiliser toutes les fonctionnalités principales.

Maintenabilité :

- Architecture modulaire : Le Projet doit suivre une architecture modulaire pour faciliter la maintenance et les futures extensions en faisant une séparation claire entre le frontend et le backend.
- Tests automatisés : Des tests unitaires et fonctionnels doivent être mis en place pour garantir la stabilité du système lors des mises à jour.

Portabilité :

- Accessibilité universelle : ce projet permet aux utilisateurs d'accéder au service où qu'ils soient et sur l'appareil de leur choix, ce qui est essentiel pour une bonne expérience utilisateur.
- Gestion des erreurs et Gestion des exceptions :Le système doit gérer correctement les erreurs utilisateur (ex. : formulaire incorrect)et afficher des messages d'erreur clairs et compréhensibles.

2.2 Les Diagramme utilisé

2.2.1 Diagramme du cas d'utilisation global

Pour représenter le comportement fonctionnel de notre système, définir les relations entre les utilisateurs (client, propriétaire et agent) et les cas d'utilisation, ainsi qu'illustrer la structure des principales fonctionnalités requises, nous avons utilisé des diagrammes de cas d'utilisation (DCU).//

Dans la figure suivante nous représentons le diagramme du cas d'utilisation global.

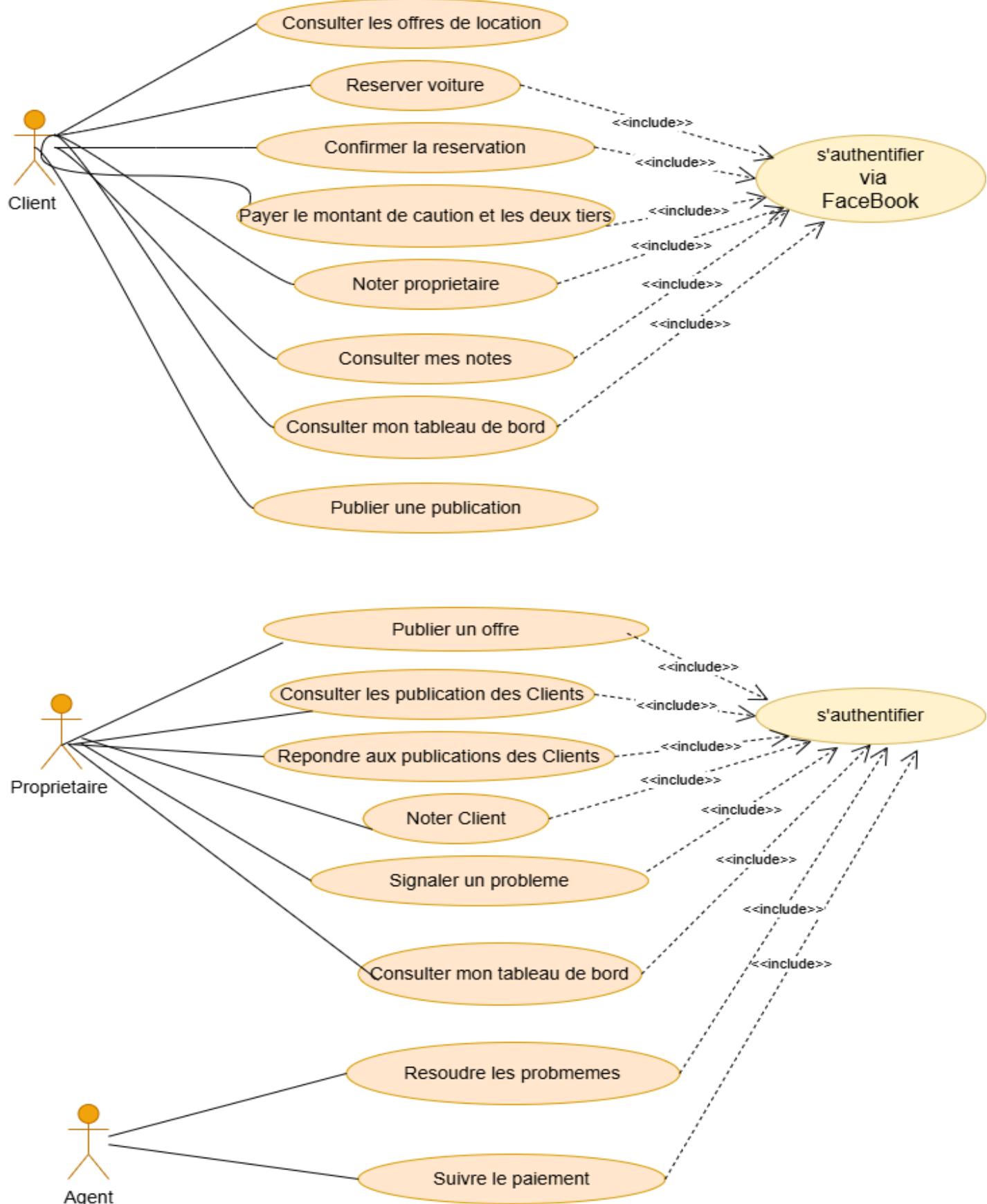


FIGURE 2.1 : Diagramme de cas d'utilisation

Ce diagramme illustre comment le client, le propriétaire et l'agent interagissent avec les différentes fonctionnalités de notre système ; la réservation d'une voiture, la consultation des offres, la publication

d'une publication. Il illustre aussi la nécessité d'authentification (à travers «include»).

2.2.2 diagramme de déploiement

Nous avons eu recours à un diagramme de déploiement pour décrire comment les composants logiciels sont installés et exécutés sur l'infrastructure matérielle, ainsi que les interactions entre les différents éléments du système. ;

- l'interaction entre le browser et le frontend Angular lors de demande et d'affichage des données.
- l'interaction entre le frontend Angular et le backend Spring Boot à travers des http requests et des responses json .
- l'interaction entre le backend Spring Boot et le DATABASE

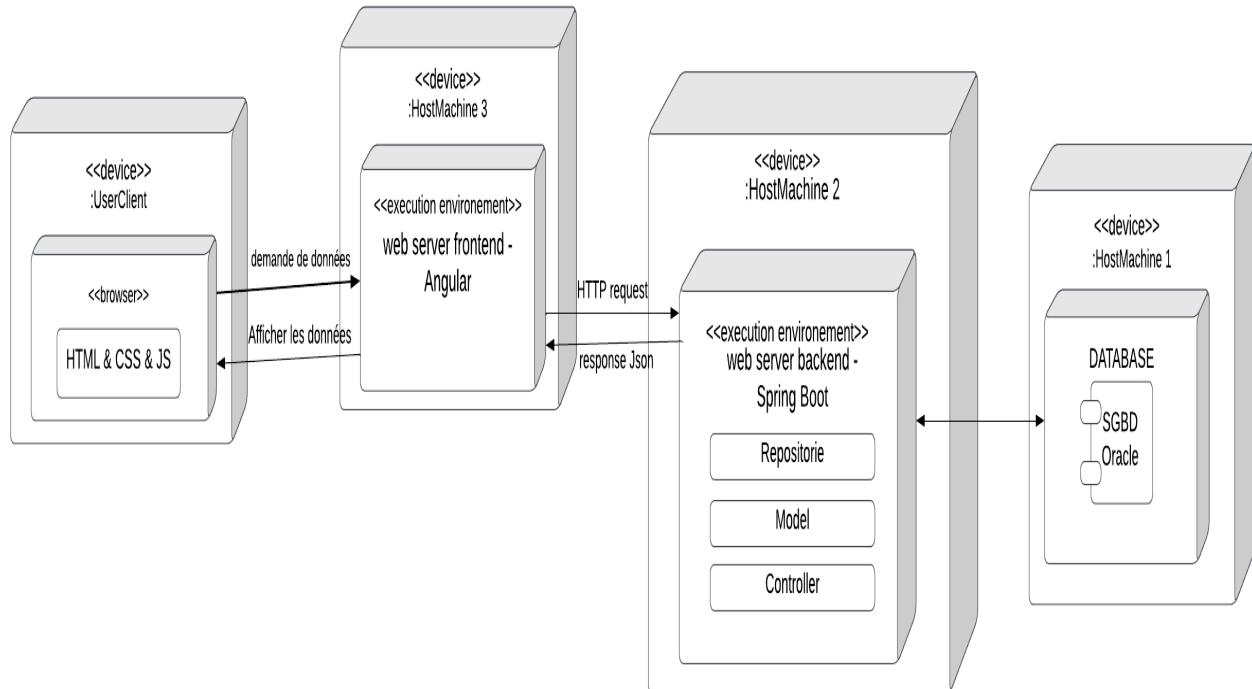


FIGURE 2.2 : Diagramme de déploiement

2.3 Pilotage du projet avec Scrum

2.3.1 backlogProduit

Sprint	User Story	Priorité	Scrum Master
1	<ul style="list-style-type: none"> — En tant que propriétaire, je veux publier une annonce de voiture avec tous les détails requis. — En tant que client, je veux consulter les offres disponibles selon mes critères de recherche (dates, lieu, type de voiture). — En tant que client, je veux annoncer une demande de location. — En tant que propriétaire, je veux consulter les annonces des demandes de location. 	Haute	Farhaoui Eya,
2	<ul style="list-style-type: none"> — En tant que client, je veux réserver une voiture en payant un tiers du montant total. — En tant que client, je veux payer la caution et les deux tiers restants le jour de la location. — En tant que propriétaire, je peux proposer une offre à une demande de location . 	Haute	Guesmi ikram

3	<ul style="list-style-type: none"> — En tant que client, je veux noter le propriétaire après la location. — En tant que propriétaire, je veux noter le client après la location. — En tant que client , je veux pouvoir repondre à une notation qui me concerne. — En tant que propriétaire, je veux pouvoir repondre à une notation qui me concerne. — En tant que client, je veux consulter les avis sur un propriétaire. — En tant que propriétaire, je veux consulter les avis sur un client. 	Normale	Soltani wissal
4	<ul style="list-style-type: none"> — En tant que propriétaire, je veux avoir accès à un tableau de bord qui présente les statistiques de mes locations. — En tant que locataire, je veux avoir un tableau de bord qui présente mon historique de locations et mes notes. — En tant que client ou propriétaire, je veux avoir accès à un service d'assistance pour poser des questions ou signaler un problème. 	Normale	Farhaoui Eya

2.4 Environnement de développement

2.4.1 Environnement matériel

- Lenovo ideapad Gaming i5 10th Gen 16G RAM
- HP EliteBook i5 16G RAM

2.4.2 Environnement logiciel

2.4.2.1 Angular

Nous avons choisi Angular pour le développement front-end de notre projet de location de voiture entre particuliers en raison de sa robustesse, de sa flexibilité et de son architecture moderne basée sur des composants. Cette structure nous permet de concevoir une application bien organisée, facile à maintenir et évolutive, répondant ainsi aux besoins spécifiques de notre projet. De plus, Angular offre un large éventail de fonctionnalités intégrées, telles que le routage, la gestion avancée des formulaires et la liaison bidirectionnelle des données, qui simplifient et accélèrent pour nous le processus de développement. Grâce à Angular, nous pouvons garantir une expérience utilisateur fluide et performante tout en optimisant la productivité de notre équipe de développement.



FIGURE 2.3 : Logo Angular

2.4.2.2 SpringBoot

Nous avons choisi Spring Boot pour le développement back-end de notre projet de location de voiture entre particuliers en raison de sa puissance, de sa flexibilité et de son architecture modulaire. Ce framework Java open source nous assure une application évolutive et facile à maintenir, tout en simplifiant l'intégration des différents composants nécessaires. Spring Boot offre également des outils et des fonctionnalités avancées, tels que la gestion des dépendances et une configuration simplifiée, ce qui accélère le développement et garantit une performance optimale pour notre projet.



FIGURE 2.4 : Logo SpringBoot

2.4.2.3 Oracle

Nous avons opté pour Oracle comme système de gestion de bases de données (SGBD) pour notre projet de location de voiture entre particuliers en raison de sa robustesse, de sa fiabilité et de ses performances élevées. Oracle est particulièrement adapté pour gérer de grandes quantités de données tout en garantissant une sécurité optimale, ce qui est essentiel pour notre application. Sa capacité à prendre en charge des scénarios complexes et à offrir des fonctionnalités avancées, telles que la gestion des transactions et l'optimisation des requêtes, en fait un choix idéal pour assurer la stabilité et l'efficacité de notre système.



FIGURE 2.5 : Système de gestion de bases de données Oracle

2.4.2.4 JAVA

Nous avons choisi Java pour le développement d'applications mobiles dans notre projet de location de voiture entre particuliers en raison de sa robustesse. Ce langage de programmation offre une syntaxe claire et une vaste collection de bibliothèques, ce qui facilite la création d'applications performantes, réactives et adaptées aux besoins des utilisateurs. De plus, Java garantit une compatibilité optimale avec les plateformes Android, assurant ainsi la fiabilité et la fluidité de notre application mobile.



FIGURE 2.6 : Langage de programmation JAVA

2.4.2.5 Visual Studio Code

Nous avons choisi Visual Studio Code (VS Code) pour le développement de notre projet en raison de sa polyvalence et de sa compatibilité avec Angular et Spring Boot. Cet éditeur léger offre des extensions dédiées, telles qu'Angular Language Service et Java Extension Pack, qui simplifient le développement frontend et backend. De plus, il intègre des outils puissants pour le débogage, la gestion de versions avec Git. Son interface moderne et intuitive, combinée à des fonctionnalités comme IntelliSense et le split-screen, améliore considérablement la productivité. Enfin, VS Code est gratuit et open source, ce qui le rend particulièrement adapté à notre projet académique.



FIGURE 2.7 : Visual Studio Code

2.4.2.6 Postman

Nous avons utilisé Postman pour tester et valider les API de notre projet de location de voiture entre particuliers. Cet outil nous a permis de simuler et de vérifier le bon fonctionnement des points de terminaison (endpoints) de l'application grâce à son interface conviviale. Avec Postman, nous avons pu envoyer des requêtes HTTP, analyser les réponses et identifier rapidement d'éventuelles erreurs, garantissant ainsi la fiabilité et l'efficacité de la communication entre le front-end et le back-end.



FIGURE 2.8 : Postman

2.5 Architecture Logicielle

Dans ce projet, nous avons adopté une architecture logicielle qui repose sur deux concepts fondamentaux : l'**architecture SOA** et le **modèle MVC**, chacun répondant à des besoins spécifiques dans la conception et l'organisation de notre système.

2.5.1 Architecture SOA

L'**architecture orientée services (SOA - Service Oriented Architecture)** a été utilisée comme base de notre architecture logicielle. Ce paradigme nous permet de structurer notre application en un ensemble de services indépendants, réutilisables et interconnectés via des interfaces bien définies, telles que les API REST. Les principaux avantages de cette approche sur notre projet incluent :

- **Modularité** : Chaque service est conçu pour effectuer une tâche spécifique, facilitant la maintenance et l'évolution.
- **Réutilisabilité** : Les services peuvent être réutilisés dans plusieurs contextes ou applications.
- **Interopérabilité** : Les services communiquent via des protocoles standard, garantissant une intégration fluide entre les différentes parties du système.

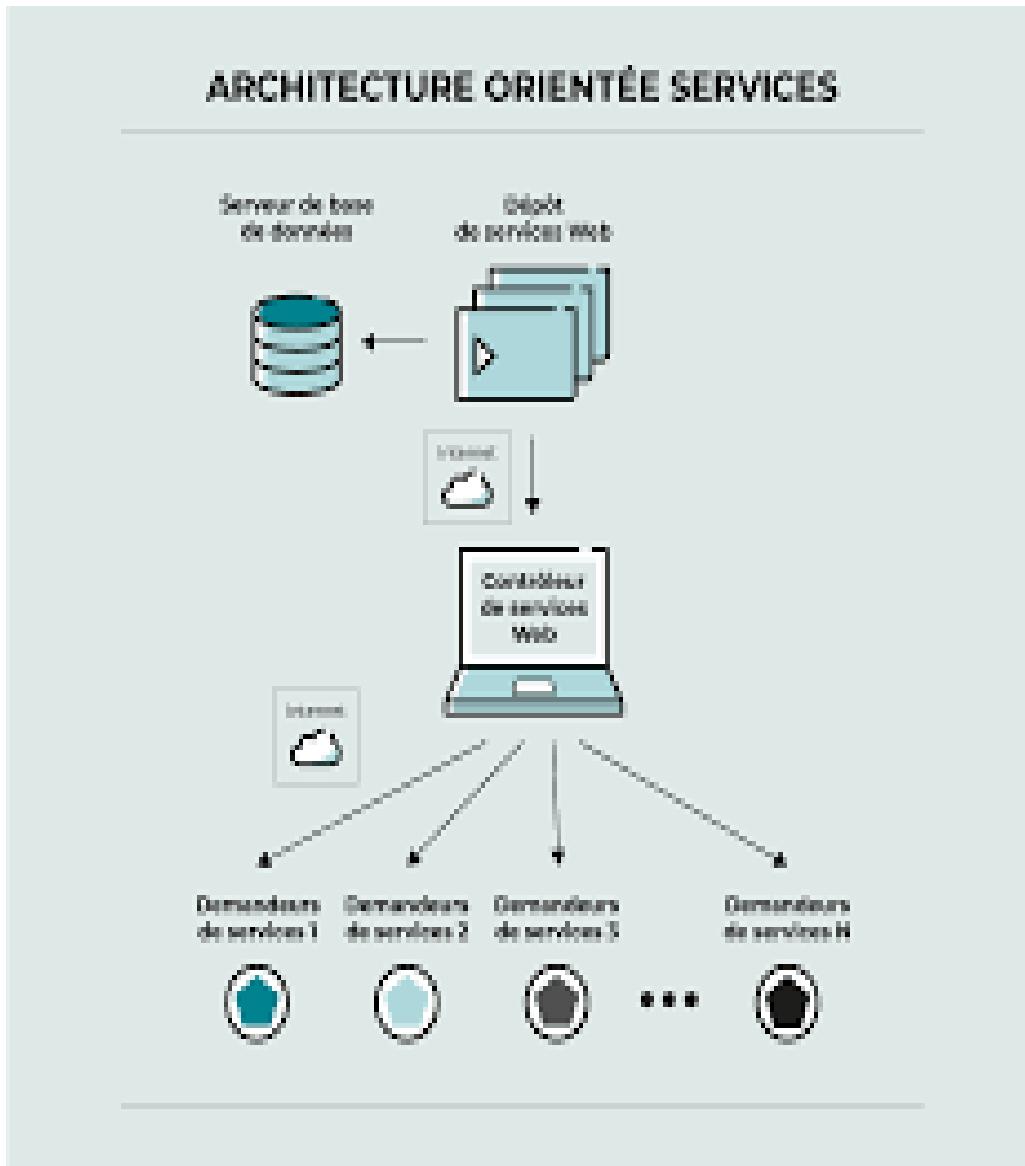


FIGURE 2.9 : Architecture SOA

2.5.2 Modèle MVC au niveau Backend et Frontend

Pour la conception interne des composants, nous avons utilisé le **modèle MVC** au niveau du backend et du frontend. Ce modèle garantit une séparation claire des responsabilités, améliorant ainsi l'organisation du code et facilitant sa maintenance.

2.5.2.1 Backend (Spring Boot)

Au niveau du backend, l'architecture adoptée repose sur une structure organisée autour des composants suivants :

- **Modèle (Model)** : Représente la couche de gestion des données, incluant les entités et les classes métier, définissant la structure des données manipulées dans l'application.
- **Contrôleur (Controller)** : Responsable de recevoir les requêtes des utilisateurs, d'interagir

avec les couches de service et de renvoyer des réponses adaptées, souvent sous forme de données JSON via des API REST.

- **Référentiel (Repository)** : Fournit une abstraction pour les opérations de persistance en interagissant avec la base de données, simplifiant les requêtes grâce à l'utilisation de Spring Data JPA.
- **Service (Service)** : Contient la logique métier et agit comme un intermédiaire entre le contrôleur et le référentiel, garantissant une séparation claire des responsabilités et une meilleure maintenabilité du code.

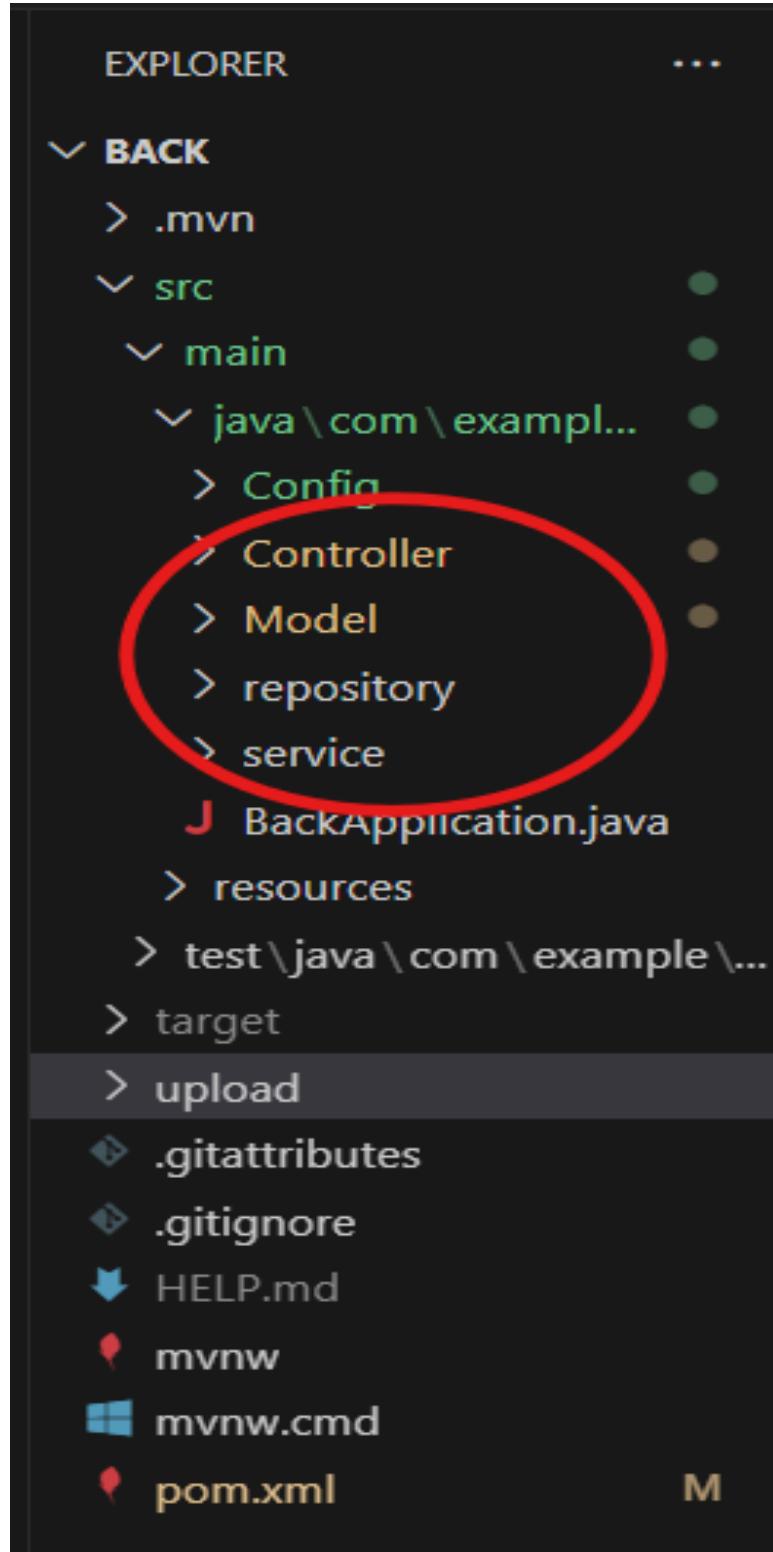


FIGURE 2.10 : Modèle MVC

2.5.2.2 Frontend (Angular)

Au niveau du frontend, le modèle MVC est également appliqué, comme suit :

- **Modèle (Model)** : Inclut les objets et les services Angular utilisés pour gérer les données et la logique métier.

- **Vue (View)** : Correspond aux composants Angular et aux templates HTML qui affichent l’interface utilisateur.
- **Contrôleur (Controller)** : Représenté par les classes des composants Angular, qui gèrent les interactions utilisateur et la logique de présentation.

2.5.3 Avantages de cette architecture

L’adoption de l’architecture SOA et du modèle MVC offre plusieurs avantages pour notre projet :

- **Flexibilité** : Les services peuvent être développés, testés et déployés indépendamment.
- **Maintenance facilitée** : La séparation des responsabilités simplifie la localisation et la correction des bugs.
- **Évolutivité** : La modularité et la réutilisabilité des composants permettent une extension facile du système.
- **Cohérence** : Le modèle MVC garantit une organisation claire du code, améliorant sa lisibilité et sa compréhension.

Conclusion

Dans ce chapitre nous avons cité les différents besoins fonctionnels et non fonctionnels, ce qui nous permet d’avoir une vision très claire et plus profonde de notre projet. Puis nous avons établi le Product Backlog et divisé notre solution en des releases, et enfin nous avons défini l’architecture.

SPRINT 1 : MISE EN PLACE DES FONCTIONNALITÉS FONDAMENTALES POUR LES CLIENTS ET PROPRIÉTAIRES

Plan

1	Présentation de l'équipe	24
2	Objectifs	24
3	Sprint Backlog	25
4	Conception	26
5	Les interfaces	28
6	Tests effectuées	33

Introduction

Dans ce premier sprint, Nous allons détaillé le Product Backlog, qui regroupe les fonctionnalités prioritaires à implémenter dans ce sprint, afin d'assurer une progression structurée et cohérente du projet.

3.1 Présentation de l'équipe

Les membres de l'équipe sont définis comme suit :

- **Scrum Master** : Chef de projet de l'équipe de développement, *Mme Baccouche Mariem*.
- **Product Owner** : Responsable produit, *Farhaoui Eya*.
- **Membres de l'équipe des développeurs** :
 - *Guesmi ikram*
 - *Soltani Wissal*

Durée : de 16/10/2024 à 25/10/2024

3.2 Objectifs

Pour ce premier sprint, l'objectif principal consiste à implémenter des fonctionnalités de base permettant l'interaction entre les clients et les propriétaires sur la plateforme :

- Authentification des propriétaires.
- Ajout d'une demande de location par le client.
- Ajout d'une offre de location par le propriétaire.
- Consultation de la liste des voitures.
- Consultation de la liste des demandes de location par le propriétaire.

3.3 Sprint Backlog

User Story	Scrum team	Durée	État
En tant que propriétaire, je veux m'inscrire avec mon numéro de téléphone	Wissal	2 jours	achevé
En tant que propriétaire, je veux publier une annonce de voiture avec tous les détails requis.	Ikram	2 jours	achevé
En tant que client, je veux consulter les offres disponibles.	Wissal	3 jours	achevé
En tant que client(futur locataire), je veux faire des recherches filtrées sur les voitures.	Wissal	3 jours	achevé
En tant que client, je veux annoncer une demande de location	Ikram	1 jour	achevé
En tant que propriétaire, je veux consulter les annonces des demandes de locations	Ikram	2 jours	achevé

TABLEAU 3.1 : Backlog du Sprint 1

3.4 Conception

3.4.1 Diagramme de classe

Pour avoir une vue d'ensemble des structures de données du système dans ce sprint, on a conçu un diagramme de classe qui illustre la capacité du client de publier plusieurs publications et celle du propriétaire de publier aussi plusieurs offres tout en mentionnant le modèle de la voiture, le prix de location par jour, le nombre de chevaux, le type de la boîte de vitesse et le nombre des portes.

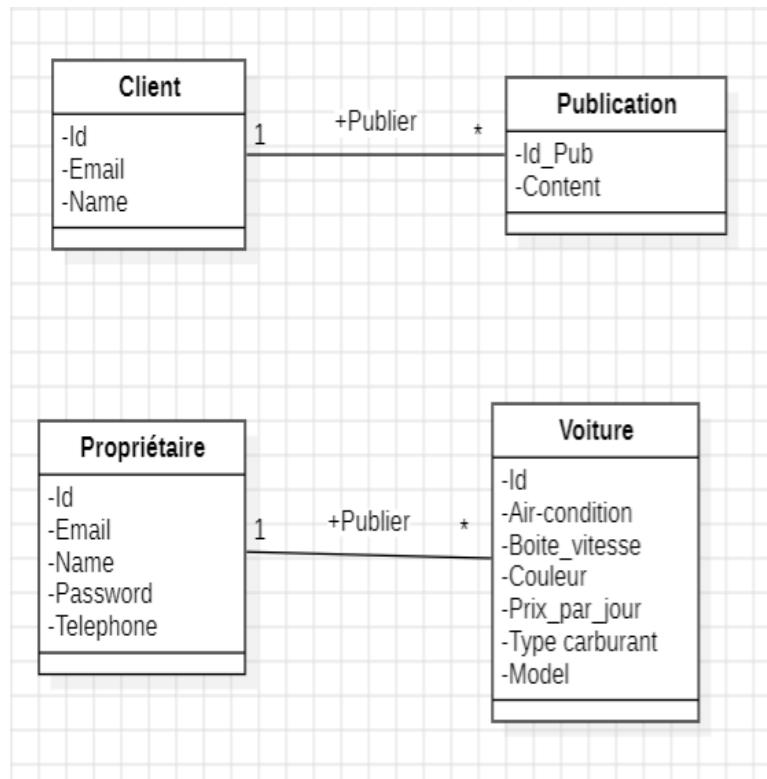


FIGURE 3.1 : Diagramme de classe du sprint 1

3.4.2 Diagramme de séquence :

Pour montrer comment les clients et les propriétaires interagissent à travers le temps afin d'accomplir une tâche précise, on a réalisé ces diagrammes de séquence.

3.4.2.1 Scénario : authentification propriétaire

Ce diagramme illustre comment un propriétaire s'authentifie :

- Le propriétaire envoie une requête de connexion en entrant ses coordonnées.
- Le système vérifie les données dans la base de données des propriétaires et connecte le propriétaire ou affiche un message d'erreur si les données sont incorrectes. Cela se répète tant que les données entrées sont erronées.

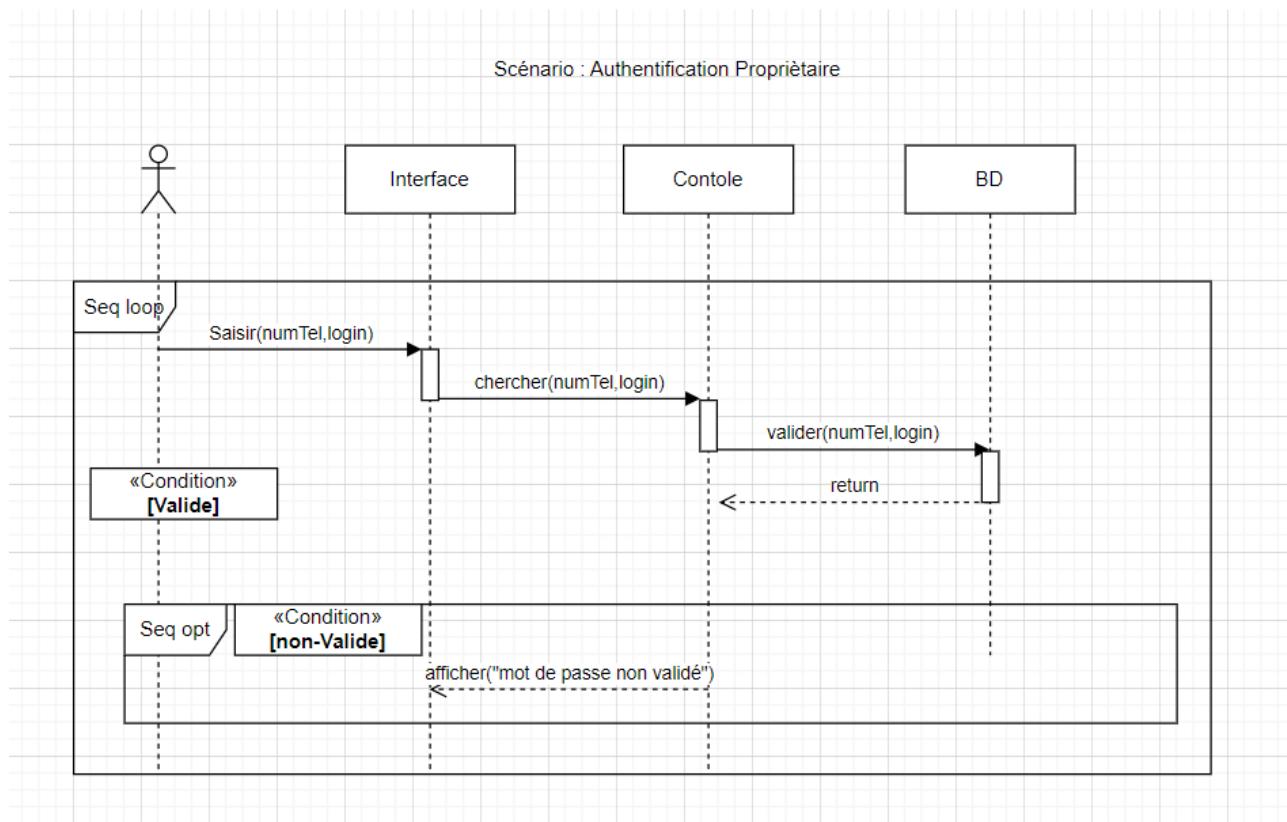


FIGURE 3.2 : Authentification propriétaire

3.4.2.2 Scénario : publication du client

- Le client envoie une requête d'ajout en cliquant sur le bouton publier après avoir entré du texte.
- Le système enregistre cette publication dans la base de données et affiche un message pour valider cet enregistrement.

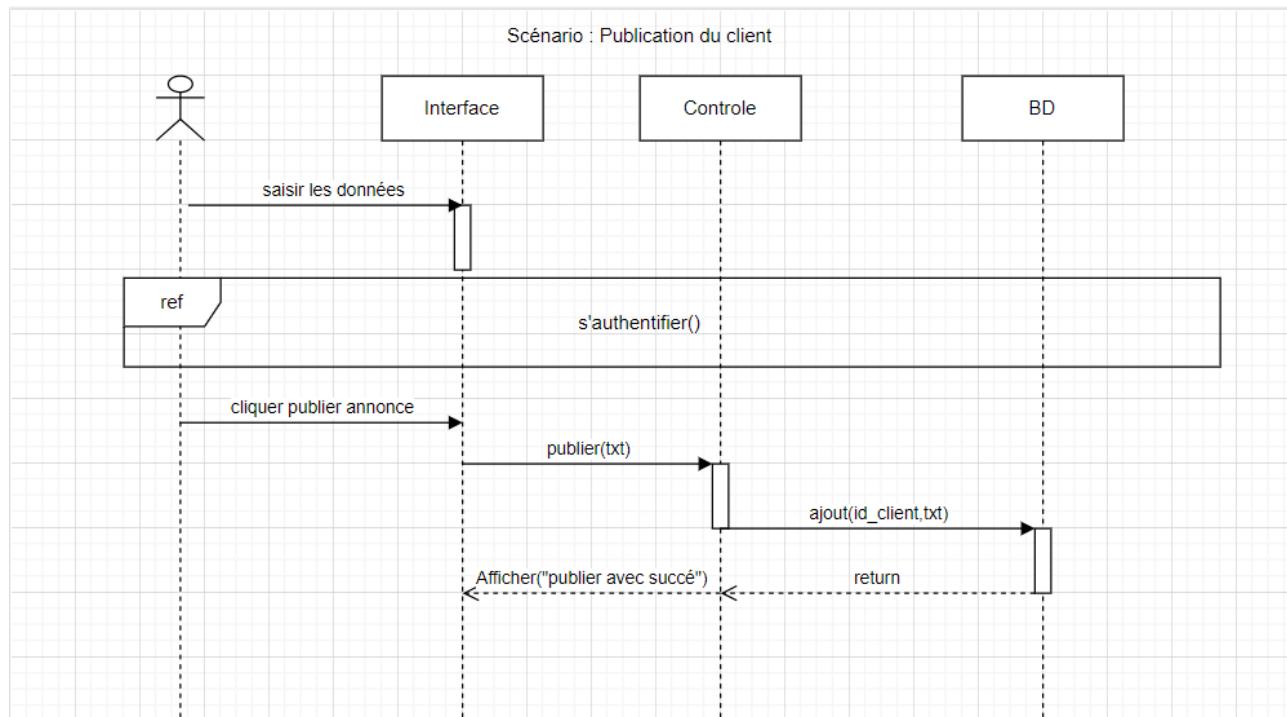


FIGURE 3.3 : Publication du client

3.4.2.3 Scénario : filtrage des voitures par date

- Le client envoie une requête de recherche au système en mentionnant la date de début et de fin de la disponibilité de la voiture recherchée.
- Le système recherche et retourne le message « il n'y a pas de voitures disponibles à ces dates » si aucune voiture n'est disponible. Sinon, il retourne les voitures disponibles correspondant aux critères.

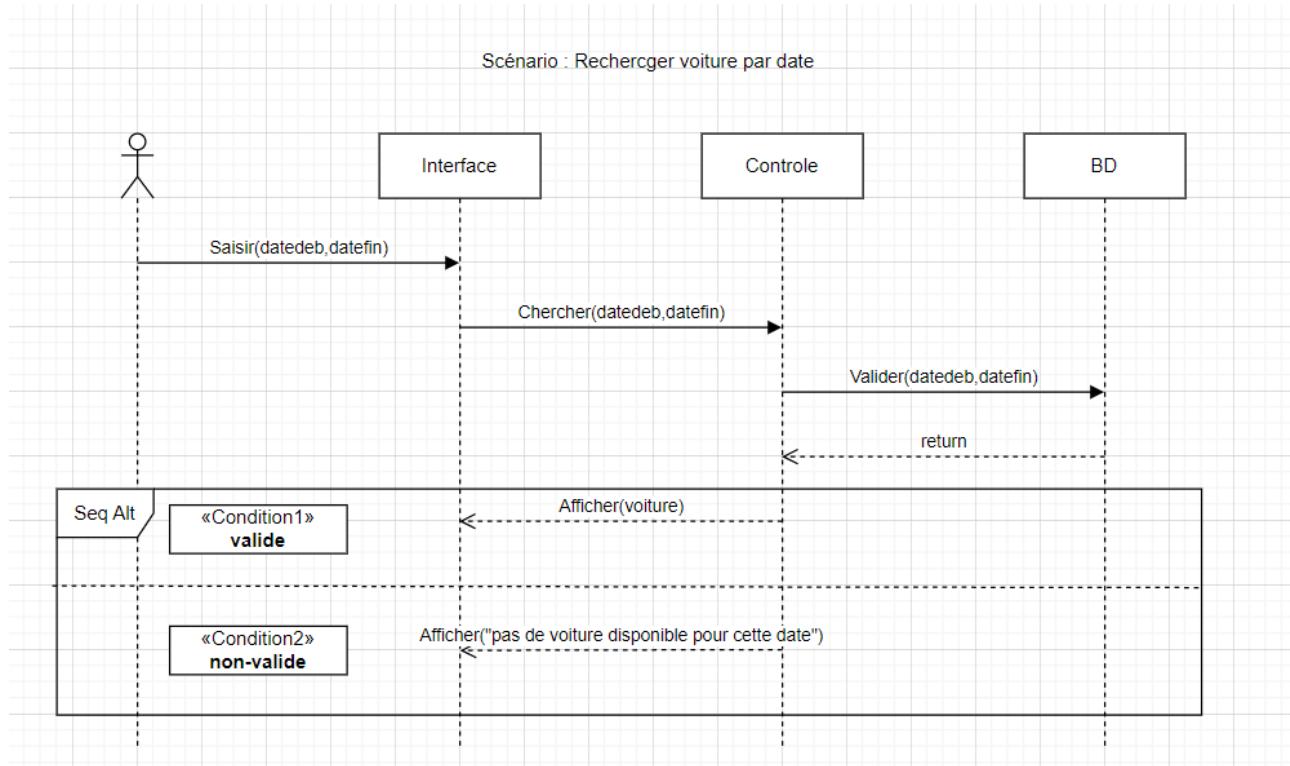


FIGURE 3.4 : Recherche de voiture par date

3.5 Les interfaces

Une maquette d'interface est une représentation visuelle statique ou interactive d'un design d'interface utilisateur (UI) pour une application, un site web ou un logiciel. Elle est créée dans le but de fournir une vue préliminaire et visuelle de l'apparence et de la disposition des éléments de l'interface.

3.5.1 Interface de l'accueil :

Le visiteur du site se trouve en premier lieu devant l'accueil du site.

EliteAuto

Accueil Nos Voitures Publier une annonce signOut Contact

EliteAuto - Location de Voitures Premium

Découvrez le luxe et le confort de notre flotte de véhicules haut de gamme

VOIR NOS VOITURES

EliteAuto

Accueil Nos Voitures Publier une annonce signOut Contact

Nos Voitures de Luxe



Mercedes-Benz S-Class
Berline de luxe, confort exceptionnel
À partir de 250 TND/jour
RÉSERVER



BMW X7
SUV spacieux et élégant
À partir de 300 TND/jour
RÉSERVER



Audi A8
Technologie de pointe et design raffiné
À partir de 280 TND/jour
RÉSERVER

EliteAuto

Accueil Nos Voitures Publier une annonce signOut Contact



"Un service exceptionnel et des voitures impeccables. Je recommande vivement EliteAuto pour toute location de voiture de luxe."

khairi D.



"La meilleure expérience de location que j'ai jamais eue. Le personnel est professionnel et les véhicules sont de première classe."

Iheb M.



"EliteAuto a rendu mon voyage d'affaires beaucoup plus agréable. Je n'hésiterai pas à faire appel à leurs services à nouveau."

sami L.

À propos d'EliteAuto
EliteAuto est votre partenaire de confiance pour la location de voitures de luxe. Nous nous engageons à fournir une expérience de conduite exceptionnelle.

Liens Rapides
Accueil
Nos Voitures
Contact

Contact
Email: info@eliteauto.com
Téléphone: *****
Adresse: *****

Suivez-nous


FIGURE 3.5 : Accueil du site

3.5.2 Interface des offres disponibles

Pour consulter la liste des voitures disponibles à louer, le visiteur doit cliquer sur « Nos voitures » pour afficher cette page. L'utilisateur peut également effectuer des recherches selon les dates de disponibilité.

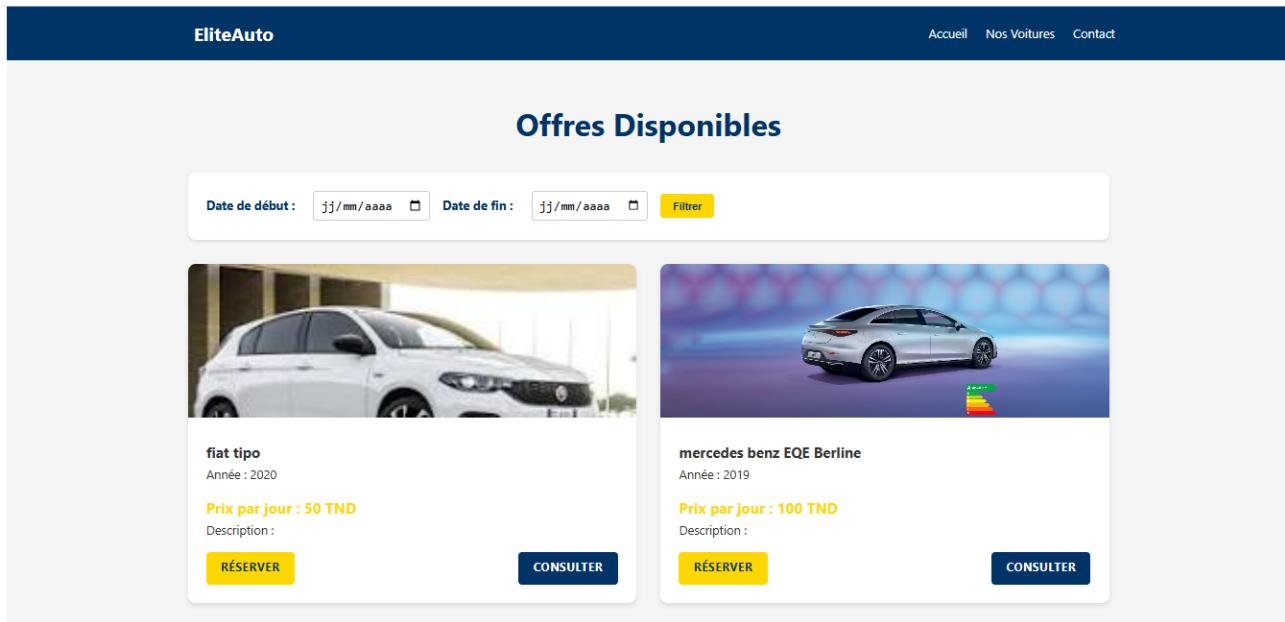


FIGURE 3.6 : Les offres disponibles

3.5.3 Interface d'authentification pour le propriétaire

Pour publier une annonce, il faut cliquer sur « Publier une annonce » en haut à droite au niveau de l'accueil. Une alerte apparaît pour choisir le mode de connexion (se connecter en tant que propriétaire ou client).

Le propriétaire souhaite se connecter à son compte en utilisant son numero de telephone et son mot de passe.

Connexion

Téléphone:

Mot de passe:

Se connecter

FIGURE 3.7 : Connexion du propriétaire

3.5.4 Interface de publication d'annonces du propriétaire

Le propriétaire peut publier une voiture dans ce site selon un formulaire.

Publier une annonce de voiture

Marque
mercedes benz

Modèle
EQE Berline

Année
2019

Prix
100.000

Couleur
BLANC

Type de carburant
Electrique

Nombre de passagers
5

Nombre de chevaux
70

Nombre de portes
5

Date de début :
08/11/2024

Date de fin :
07/12/2024

Montant de la caution
2000

Image
 EQE-Exe...e-Line.avif

Publier

Voiture publiée avec succès !

FIGURE 3.8 : Publication du propriétaire

3.5.5 Interface de publication d'annonces des Clients

Si le visiteur clique sur « Connexion client », la page permettant au client de faire une publication s'affiche.

EliteAuto

Accueil Nos Voitures Caractéristiques Témoignages Contact

Publier une annonce

Announce

JE CHERCHE UNE VOITURE A DEUX PLACES DE LA MARQUE KIA AVEC UN PRIX DE LOCATION QUI NE DÉPASSE PAS 200 DT PAR JOUR

Publier

FIGURE 3.9 : Publication du client

Pour consulter la liste des publications faites par les clients, il faut cliquer sur Publication client pour afficher la page suivante.



FIGURE 3.10 : Publication du client

3.6 Tests effectuées

3.6.1 Test Unitaire

- **Objectif :** Vérifier la création et l'enregistrement d'une nouvelle offre de location par le propriétaire.
- **Objet de Test :** OffreLocation
- **Propriété à Tester :** Champs obligatoires comme dateDebut et dateFin.
- **Données de Test :**
 - dateDebut : 2024-11-01
 - dateFin : 2024-12-01
 - Autres champs comme prix, disponibilites, etc.
- **Cas de Test :**
 - Créer une offre avec des dates valides et un prix.
 - Vérifier que l'enregistrement est effectué avec succès.
 - Essayer d'enregistrer une offre avec des champs vides (ex. dateDebut ou dateFin) pour vérifier la validation des champs obligatoires.

3.6.2 Test d'Intégration Composants

- **Objectif :** Vérifier la communication entre le composant Angular et le backend Spring Boot pour l'affichage de l'offre.
- **Objet de Test :** `ListeOffresComponent` (Angular) et `OffreController` (Spring Boot).
- **Propriété à Tester :** Récupération des données de l'offre via l'API.
- **Données de Test :** Vérifier que le composant Angular envoie correctement les requêtes et reçoit les réponses attendues du backend.
- **Cas de Test :**
 - Tester la requête GET pour récupérer toutes les offres.
 - Vérifier si les données retournées sont correctement affichées dans le composant.

3.6.3 Test d'Intégration Système

- **Objectif :** Vérifier l'intégration complète entre Angular, Spring Boot, et la base de données Oracle.
- **Objet de Test :** Système complet (Angular + Spring Boot + Oracle).
- **Propriété à Tester :** Fonctionnalité d'authentification et récupération des données utilisateur.
- **Données de Test :** Un utilisateur authentifié dans la base de données.
- **Cas de Test :**
 - Tester la fonctionnalité d'authentification en utilisant des identifiants valides.
 - Vérifier la récupération des données d'une voiture via l'API.
 - Tester la réponse correcte d'Oracle lors de la requête pour récupérer les voitures disponibles.

3.6.4 Test d'Acceptation Utilisateur

- **Objectif :** Vérifier que les utilisateurs peuvent interagir avec le système comme prévu.
- **Objet de Test :** Interface utilisateur pour publier une offre de location et consulter les voitures.
- **Propriété à Tester :** Interface utilisateur, flux de publication d'offre et consultation de voitures.
- **Données de Test :**
 - Utilisateur connecté en tant que propriétaire.
 - Offre de location et voitures existantes.
- **Cas de Test :**
 - Se connecter en tant que propriétaire.

- Publier une nouvelle offre de location.
- Vérifier si l'offre apparaît dans la liste des offres disponibles.
- Consulter une voiture et vérifier les informations affichées.

Conclusion

À travers ce premier sprint, les fonctionnalités clés permettant les interactions de base entre clients et propriétaires ont été implémentées avec succès. Les tests ont permis de valider l'intégration des différents composants et de garantir une expérience utilisateur satisfaisante. Ces fondations établies facilitent les développements à venir dans les prochains sprints.

SPRINT 2 : IMPLÉMENTATION DES FONCTIONNALITÉS CLÉS

Plan

1	Présentation de l'équipe	37
2	Objectifs	37
3	sprint Backlog	38
4	Conception	38
5	Les interfaces :	41
6	Tests :	43

Introduction

Ce chapitre détaille les étapes de la mise en place de l'environnement de développement et les travaux réalisés lors du deuxième sprint du projet. Ce sprint a été crucial pour intégrer des fonctionnalités clés de la plateforme, telles que l'authentification via Facebook, la gestion des réservations et paiements, ainsi que l'interaction entre clients et propriétaires. Nous abordons la composition de l'équipe, les objectifs définis, le backlog du sprint, les diagrammes de conception, les interfaces développées et les tests effectués pour assurer la qualité des fonctionnalités implémentées.

4.1 Présentation de l'équipe

Les membres de l'équipe sont définis comme suit :

- **Scrum Master** : Chef de projet de l'équipe de développement, *Mme Baccouche Mariem*.
- **Product Owner** : Responsable produit, *Guesmi ikram*.
- **Membres de l'équipe des développeurs** :
 - *Farhaoui eya*
 - *Soltani Wissal*

Durée : 3/11/2024 à 15/11/2024

4.2 Objectifs

Pour ce deuxième sprint, l'objectif principal était l'implémentation des fonctionnalités suivante sur la plateforme :

- Authentification des Clients à l'aide du FaceBook.
- réalisation d'une reservation pour une voiture.
- paiement de la première tranche lors de la reservation.
- Consultation de la liste des reservation propre a un utilisateur .
- paiement de la 2ème tranche avec le montant de caution .
- Consultation de la liste des location par le propriétaire
- création d'une reponse a une publication des clients par le propriétaire.
- Consultation par le client des réponses faites par les propriétaires à ses publications.

4.3 sprint Backlog

User Story	Scrum team	Durée	État
En tant que Client, je veux réserver une voiture en payant un tiers du montant total	Wissal	2 jours	achevé
En tant que Client, je veux payer la caution et les deux tieres restant	Eya	2 jours	achevé
En tant que propriétaire je veux proposer un offre à une demande de location	Wissal	2 jours	achevé
En tant que propriétaire je veux consulter les réservation	Eya	1 jour	achevé
En tant que Client , je veux me connecter avec mon compte FaceBook 1	Wissal	3 jours	achevé

TABLEAU 4.1 : Backlog du Sprint

4.4 Conception

4.4.1 Diagramme de classe

Le diagramme de classe illustre les entités clés du système et leurs relations. La classe Client est associée aux Publications, qu'il peut créer, et aux Réservations, qui sont liées aux Offres des Propriétaires. Les Propriétaires peuvent répondre aux Publications des clients via une classe Réponse.

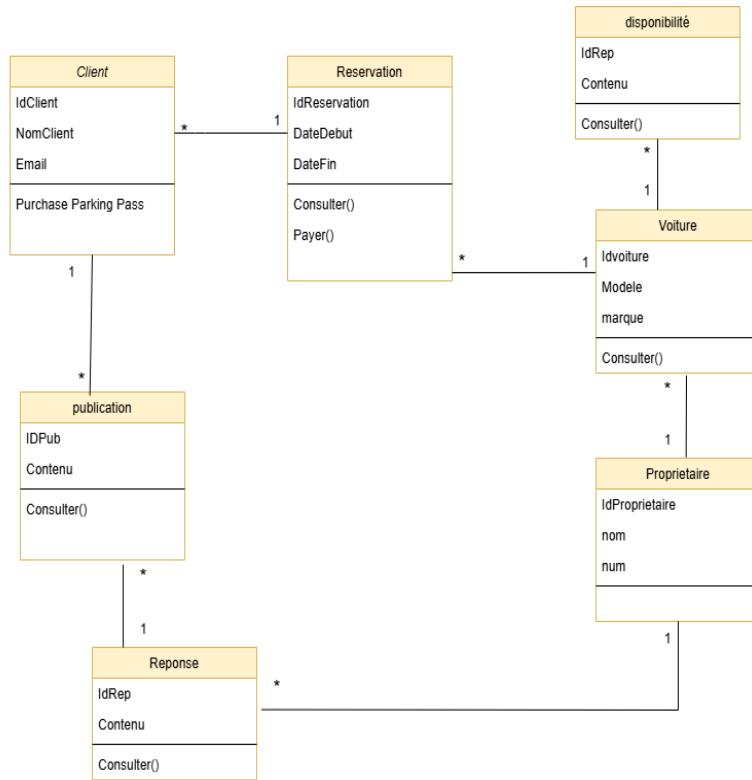


FIGURE 4.1 : Diagramme de classe du sprint 2

4.4.2 Diagramme de séquence :

Pour montrer comment les clients et les propriétaires interagissent à travers le temps afin d'accomplir une tâche précise, on a réalisé ces diagrammes de séquence.

4.4.2.1 Scénario : Proposition d'une offre par le propriétaire à une demande d'un client :

Après s'être authentifié ;

Le propriétaire envoie une requête d'ajout d'une offre en cliquant sur le bouton ajouter offre après avoir entré cette dernière.

Le système enregistre cette offre dans la base de données et affiche un message pour valider cet enregistrement.

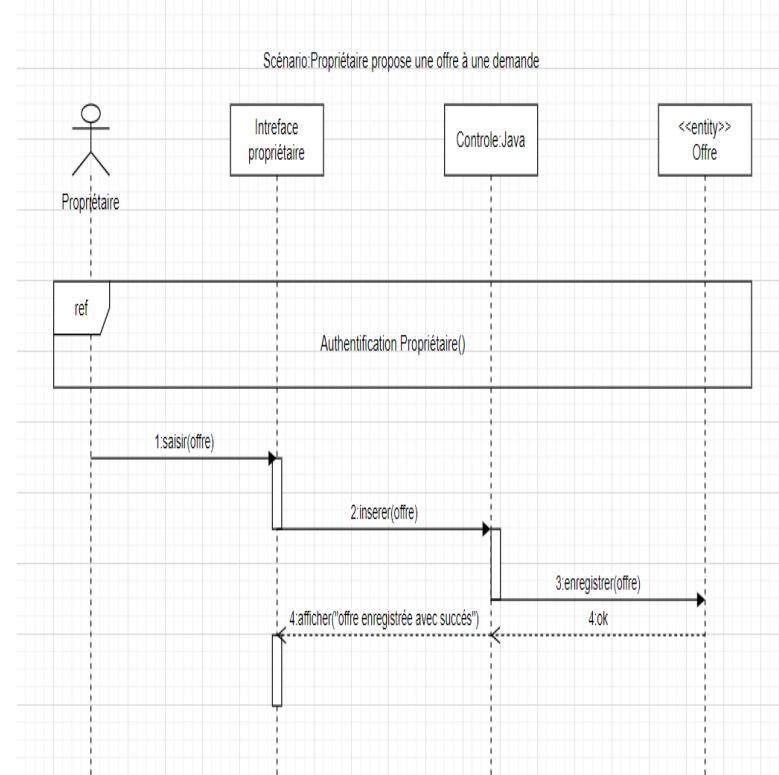


FIGURE 4.2 : Diagramme de séquence du scénario de proposition d'une offre

4.4.2.2 Scénario :Réserver une voiture

Ce diagramme illustre comment un Client peut réserver une voiture en payant la première tranche du montant de location.

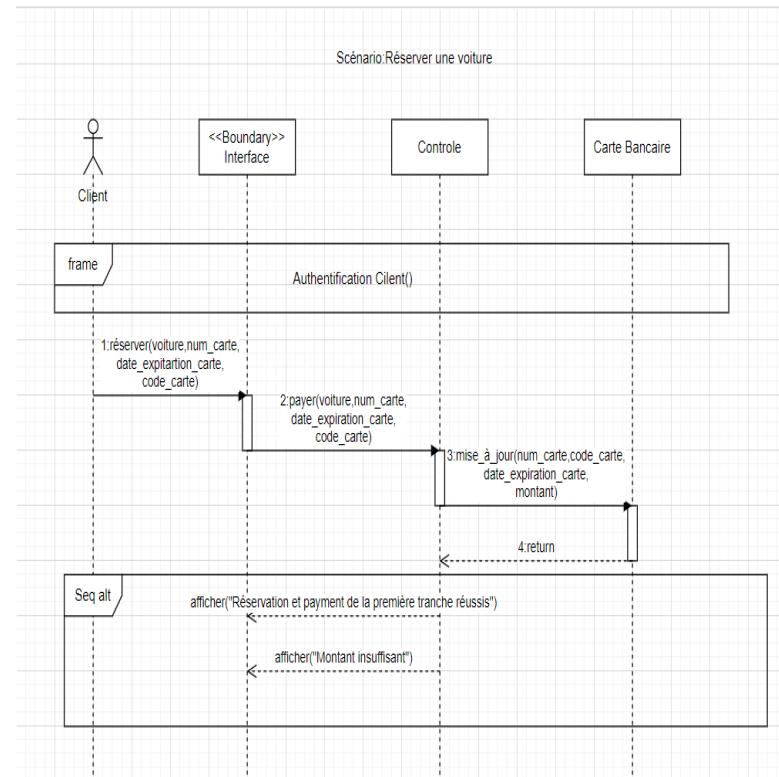


FIGURE 4.3 : Diagramme de séquence du scénario de réservation d'une voiture

4.5 Les interfaces :

4.5.1 Interface d'authentification pour le Client

Le Client souhaite se connecter à son compte en utilisant son Compte FaceBook.

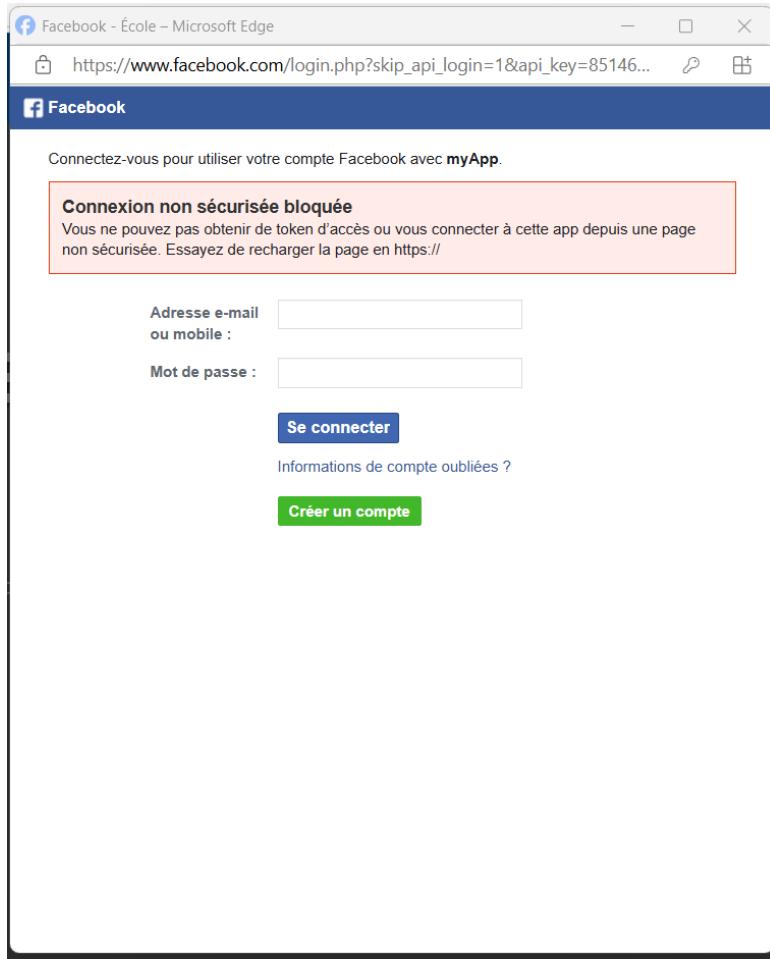


FIGURE 4.4 : Connexion du Client

4.5.2 Interface pour la réservation de voiture

Lorsqu'un client clique sur le bouton "Réserver", une interface de réservation s'affiche. Cette interface permet au client de saisir les informations nécessaires, notamment :

- La date de début et la date de fin de la réservation.
- Les données de la carte bancaire, telles que le numéro de carte, le nom du titulaire, le CVV, et la date d'expiration.

Une fois ces informations remplies, le client peut confirmer la réservation en soumettant le formulaire.



Ford Mustang
Année : 2018

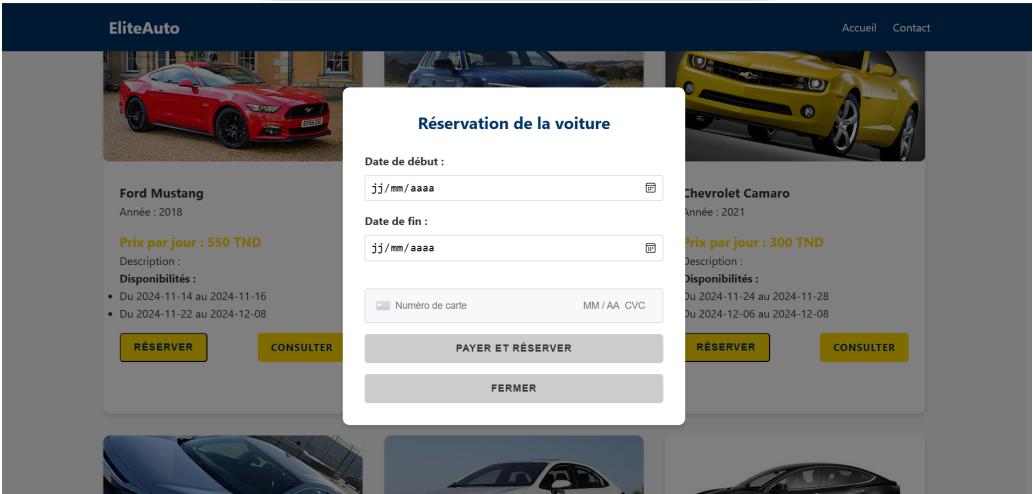
Prix par jour : 550 TND

Description :

Disponibilités :

- Du 2024-11-14 au 2024-11-16
- Du 2024-11-22 au 2024-12-08

RÉSERVER **CONSULTER**



Réservez une voiture

Date de début : Date de fin :

Numéro de carte MM / AA CVC

PAYER ET RÉSERVER **FERMER**

FIGURE 4.5 : Réervation d'une voiture

4.5.3 Interface de Consultation de la liste des réservations propre à un Client

le client de ce site peut consulter la liste de ses réservations à travers cette interface :

EliteAuto								Accueil	Nos Voitures	Publier une annonce	signOut	Contact
Marque	Modèle	nom de propriétaire	Date Début réservation	Date Fin réservation	Montant Total	Montant Payé	montantCaution	Actions				
Toyota	Corolla	Baccouche Mariem	2024,11,15	2024,11,20	500TND	166.67TND	2000TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		
Audi	A4	arij	2024,11,21	2024,12,1	2600TND	866.67TND	1200TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		
Tesla	Model 3	Baccouche Mariem	2024,11,17	2024,11,19	1200TND	400TND	6500TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		
Audi	A4	arij	2024,11,13	2024,11,14	260TND	86.67TND	1200TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		
Audi	A4	arij	2024,12,4	2024,12,5	260TND	86.67TND	1200TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		
Honda	Civic	Baccouche Mariem	2024,12,21	2024,12,26	500TND	166.67TND	1500TND	<button>payer la 2 éme tranche du montant</button>	<button>Ajouter un avis</button>	<button>Consulter les avis</button>		

FIGURE 4.6 : liste de reservation propre à un proprietaire

4.6 Tests :

4.6.1 Test Unitaire

4.6.1.1 Objectif : Authentification des Clients à l'aide de Facebook

- **Objet de Test :** Composant `FacebookLoginComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Identifiants Facebook valides.
 - Identifiants Facebook invalides.
- **Cas de Test :**
 - Tester si un utilisateur peut se connecter avec un compte Facebook valide.
 - Vérifier le traitement des erreurs en cas d'échec de connexion (ex. mot de passe incorrect ou compte non associé).

4.6.1.2 Objectif : Réalisation d'une réservation pour une voiture

- **Objet de Test :** Méthode `ReserverVoiture` dans le contrôleur Spring Boot et formulaire Angular associé.
- **Données de Test :**
 - Informations valides (voiture disponible, dates correctes).
 - Informations invalides (voiture déjà réservée ou dates manquantes).
- **Cas de Test :**

- Vérifier la création d'une réservation avec des informations valides.
- Tester la validation des champs obligatoires.

Objectif : Paiement de la première tranche lors de la réservation

- **Objet de Test :** Service de paiement (`StripeService`) et composant Angular pour le paiement.
- **Données de Test :**
 - Carte bancaire valide.
 - Carte bancaire invalide (ex. numéro incorrect).
- **Cas de Test :**
 - Tester le paiement avec des informations valides.
 - Vérifier le traitement des erreurs en cas de paiement échoué.

4.6.2 4.3.2 Test d'Intégration

Consultation de la liste des réservations propre à un utilisateur

- **Objet de Test :** Composant `ReservationListComponent` (Angular) et méthode Spring Boot pour récupérer les réservations.
- **Données de Test :** Réservations existantes pour un utilisateur.
- **Cas de Test :**
 - Tester si les réservations d'un utilisateur authentifié sont correctement affichées.
 - Vérifier qu'un utilisateur sans réservations voit un message approprié.

Paiement de la deuxième tranche avec le montant de la caution

- **Objet de Test :** Méthode `PayerDeuxiemeTranche` dans le backend et interface utilisateur.
- **Données de Test :**
 - Montant total, montant déjà payé, montant restant (caution comprise).
- **Cas de Test :**
 - Vérifier le calcul correct du montant restant.
 - Tester le paiement final et confirmer la mise à jour de l'état de la réservation.

Consultation de la liste des locations par le propriétaire

- **Objet de Test :** Composant `LocationListComponent` et API Spring Boot pour récupérer les locations.

- **Données de Test :** Locations associées au propriétaire connecté.
- **Cas de Test :**
 - Vérifier l'affichage correct des locations pour le propriétaire.
 - Tester l'absence de locations avec un message approprié.

4.6.3 4.3.3 Test d'Acceptation

Création d'une réponse à une publication des clients par le propriétaire

- **Objectif :** Vérifier que le propriétaire peut répondre à une publication.
- **Objet de Test :** Interface utilisateur et méthode `RepondrePublication` dans le backend.
- **Données de Test :** Publication existante.
- **Cas de Test :**
 - Publier une réponse valide et vérifier qu'elle est enregistrée.
 - Tester les validations en cas de réponse vide.

Consultation des réponses par le client

- **Objectif :** Vérifier que le client peut consulter les réponses à ses publications.
- **Objet de Test :** Interface utilisateur et méthode `AfficherReponses` dans le backend.
- **Données de Test :** Réponses existantes pour les publications d'un client.
- **Cas de Test :**
 - Afficher correctement les réponses associées à une publication.
 - Vérifier qu'un client sans réponse voit un message approprié.

Conclusion

Le deuxième sprint a permis d'avancer significativement dans le développement de la plateforme en intégrant des fonctionnalités essentielles pour l'expérience utilisateur et les interactions entre clients et propriétaires. Les tests unitaires, d'intégration, et d'acceptation ont permis de valider la fiabilité des fonctionnalités et d'assurer leur bon fonctionnement.

SPRINT 3 : IMPLÉMENTATION DES FONCTIONNALITÉS CLÉS

Plan

1	Présentation de l'équipe	47
2	Objectifs	47
3	Sprint Backlog	48
4	Conception :	48
5	Les interfaces :	50
6	Tests	54

Introduction

Ce chapitre présente les étapes de mise en place de l'environnement de développement ainsi que les travaux réalisés lors du 3ème sprint du projet, axé principalement sur l'intégration du système de notation et de feedback entre les clients et les propriétaires.

5.1 Présentation de l'équipe

Les membres de l'équipe sont définis comme suit :

- **Scrum Master** : Chef de projet de l'équipe de développement, *Mme Baccouche Mariem*.
- **Product Owner** : Responsable produit, *Soltani wissal*.
- **Membres de l'équipe des développeurs** :
 - *Farhaoui eya*
 - *Guesmi ikram*

Durée : 16/11/2024 à 29/11/2024

5.2 Objectifs

Pour ce 3eme sprint, l'objectif principal était l'implémentation des fonctionnalités suivantes sur la plateforme :

- Notation du propriétaire par le client après une location.
- Notation du client par le propriétaire après une location.
- Consultation des avis sur un propriétaire par le client.
- Consultation des avis sur un client par le propriétaire.
- Possibilité pour le client de répondre à une notation le concernant.
- Possibilité pour le propriétaire de répondre à une notation le concernant.

5.3 Sprint Backlog

User Story	Scrum team	Durée	État
En tant que client, je veux noter le propriétaire après la location	ikram	2 jours	achevé
En tant que propriétaire, je veux noter le client après la location	Eya	2 jours	achevé
En tant que client, je veux pouvoir répondre à une notation qui me concerne	ikram	2 jours	achevé
En tant que propriétaire, je veux pouvoir répondre à une notation qui me concerne	Eya	2 jours	achevé
En tant que client, je veux consulter les avis sur un propriétaire	ikram	1 jour	achevé
En tant que propriétaire, je veux consulter les avis sur un client	Eya	1 jour	achevé

TABLEAU 5.1 : Backlog du Sprint

5.4 Conception :

Le diagramme de classe illustre les entités clés du système et leurs relations. La classe Client est associée aux Avis, qu'il peut recevoir après une location. La classe Propriétaire est également reliée à des Avis qu'il peut recevoir de la part des clients après une location. Les Avis permettent aux clients

de noter les propriétaires et vice-versa.

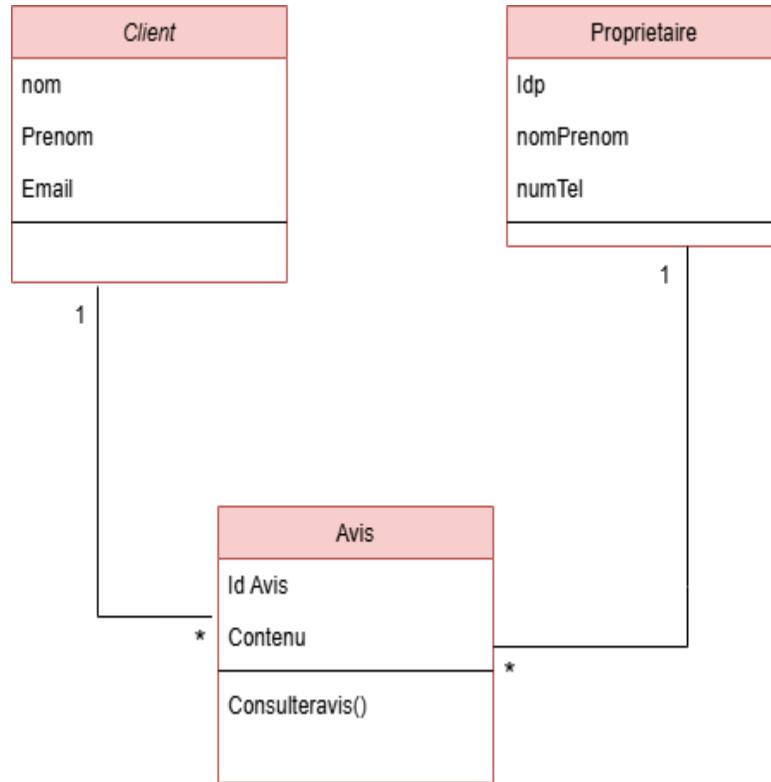


FIGURE 5.1 : Diagramme de classe du sprint 3

5.4.1 Diagramme de séquence :

Pour montrer comment les clients et les propriétaires interagissent à travers le temps afin d'accomplir une tâche précise, on a réalisé ces diagrammes de séquence.

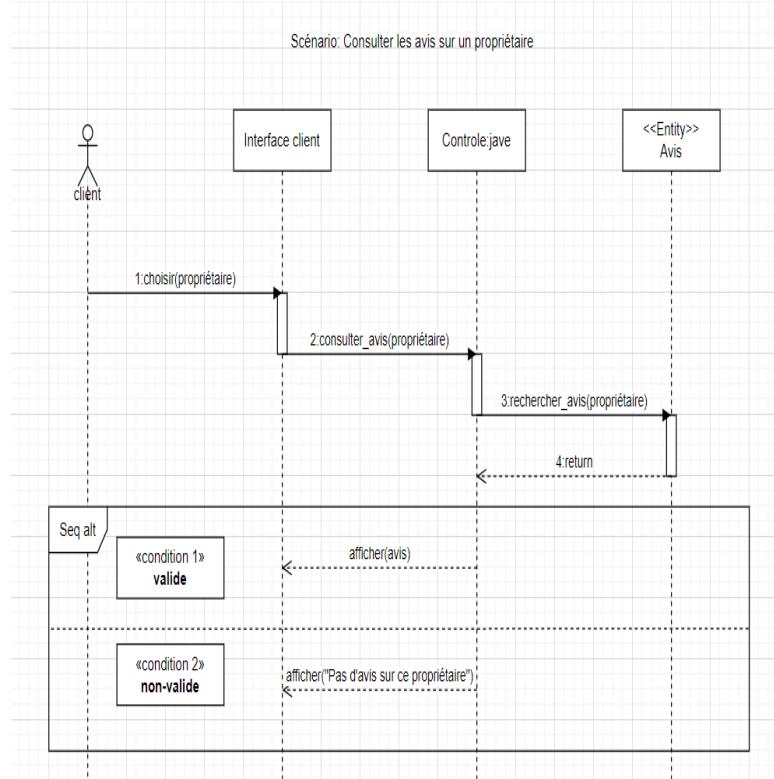


FIGURE 5.2 : Diagramme de séquence du sprint 3

5.4.1.1 Scénario : Consulter les avis faits sur un propriétaire

Ce diagramme illustre comment un client peut consulter la liste des avis faits sur un propriétaires bien déterminé

Après s'être authentifié :

Le client envoie une requête de recherche au système en choisissant le propriétaire visé

Le système recherche et retourne le message « il n'y a pas encore d'avis faits sur ce propriétaire » si aucun avis n'est disponible. Sinon, il retourne les avis disponibles concernant le propriétaire visé.

5.5 Les interfaces :

5.5.1 Interface du propriétaire pour consulter les avis faits sur un client

la capture d'écran de l'interface ci-dessous permet au propriétaire de consulter les avis faits par d'autres propriétaires sur un client bien déterminé :

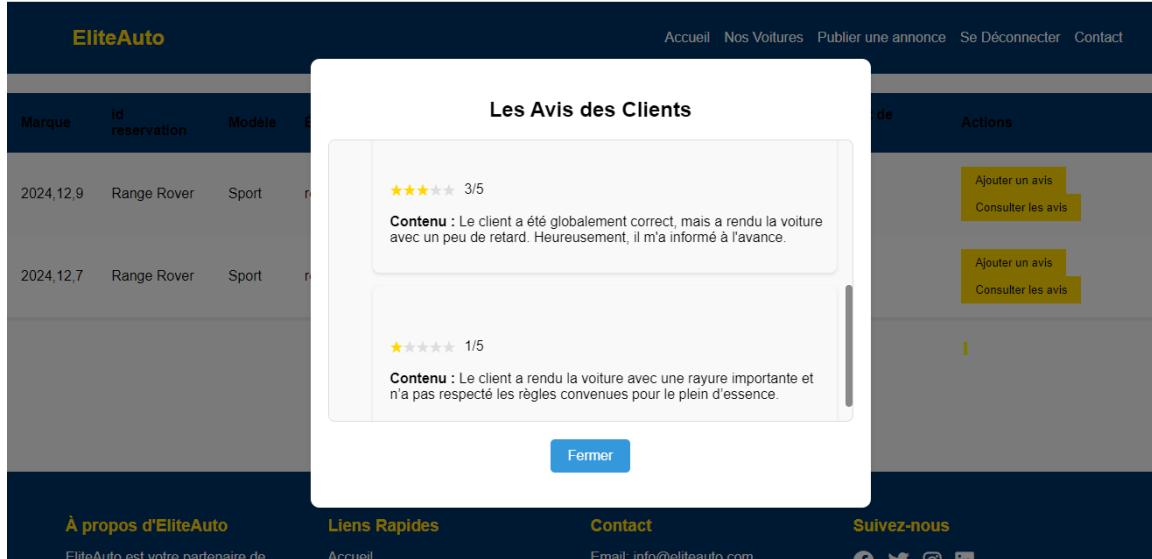


FIGURE 5.3 : Vue de « Consultation des avis faits sur un client»

5.5.2 Interface du propriétaire pour consulter et répondre sur les avis faits sur lui

la capture d'écran de l'interface ci-dessous permet au propriétaire de consulter les avis faits sur lui par des clients qui ont réservé sa voiture. Elle lui permet aussi de faire des réponses sur ces avis :



FIGURE 5.4 : Vue de « Consultation et réponse du propriétaire sur les avis faits sur lui»

5.5.3 Interface du propriétaire pour donner son avis sur un client

la capture d'écran de l'interface ci-dessous permet au propriétaire de noter un client et de donner son avis sur lui :

The screenshot shows a table of car reservations with columns for Marque, id réservation, Modèle, Etat, Date Début réservation, Date Fin réservation, Montant Total, Montant de caution, and Actions. Two rows are visible. A modal window titled "Ajouter un Avis" (Add Review) is open over the second row. The modal contains a note field with placeholder text "Le client a été globalement correct," a dropdown menu for "Note" showing values 1 through 5 with 5 selected, and two buttons "Soumettre" (Submit) and "Annuler" (Cancel).

FIGURE 5.5 : Vue de « ajout du propriétaire d'un avis sur un client»

5.5.4 Interface du propriétaire pour consulter ses avis et les réponses faites dessus

la capture d'écran de l'interface ci-dessous permet au propriétaire de consulter la liste des avis qu'il a fait sur ses clients et les réponses de ces clients sur ces avis :

The screenshot shows a sidebar with navigation links: Publier une annonce, Consulter les publications des clients, Consulter les réservations de mes voitures, Consulter les réponses à mes avis (highlighted), Consulter les avis faite sur moi, Service d'assistance, and Déconnexion. The main area displays a list of reviews with their notes and client names. A modal window titled "Les réponses sur vos avis" (Responses on your reviews) is open, showing three responses to a review from a client named Eya Farhaoui.

Commentaire : Client	Note	Client	Commentaire
Client tres respectueux et ponctuel. La voiture m a ete rendue dans un excellent etat. Merci beaucoup	★★★★★	Eya Farhaoui	Merci beaucoup pour votre confiance. La voiture etait impeccable, et je n hésiterai pas a louer chez vous a nouveau
			merci bien
			derienderien

FIGURE 5.6 : Vue de « consultation du propriétaire de la liste de ses avis et des réponses des clients faites dessus »

5.5.5 Interface du client pour consulter et répondre sur les avis faits sur lui

la capture d'écran de l'interface ci-dessous permet au client de consulter les avis faits sur lui par des propriétaires .Elle lui permet aussi de faire des réponses sur ces avis :

The screenshot shows the EliteAuto application interface. On the left is a sidebar with the following menu items:

- Publier une annonce
- Consulter les offres
- Mes réservations
- consulter les réponses à mes avis
- consulter les avis faits sur moi** (highlighted in blue)
- Service d'assistance
- Déconnexion

The main content area is titled "Avis faîtes sur vous". It displays three reviews:

- Contenu :** CLIENT BIEN RESPECTUEUX
Note : ★★★★★
Répondre
- Contenu :** Client très respectueux et ponctuel. La voiture m'a été rendue dans un excellent état. Merci beaucoup.
Note : ★★★★★
Répondre
- Contenu :** Le client a été globalement correct, mais a rendu la voiture avec un peu de retard. Heureusement, il m'a informé à l'avance.
Note : ★★★★ 1
Répondre
Je comprends votre frustration. La rayure est survenue accidentellement et j'aurais dû vous en informer immédiatement. Je suis prêt à couvrir les frais de réparation.

A "Commenter" button is located at the bottom of the third review.

FIGURE 5.7 : Vue de « Consultation et réponse du client sur les avis faits sur lui »

5.5.6 Interface du client pour consulter ses avis et les réponses faites dessus

la capture d'écran de l'interface ci-dessous permet au client de consulter la liste des avis qu'il a fait sur les propriétaires des voitures qu'il a loué et les réponses de ces propriétaires sur ces avis :

The screenshot shows the EliteAuto application interface. On the left is a sidebar with the following menu items:

- Publier une annonce
- Consulter les voitures
- Consulter mes réservations
- Consulter les avis faits sur moi
- Service d'assistance
- Déconnexion

The main content area is titled "Les réponses sur vos avis". A modal window is open, showing a review and its response:

Réponses pour l'avis :
La voiture était top, mais ce que j'ai préféré, c'est le café que le propriétaire m'a offert en attendant que je la livre. Service VIP

Haha, ravi que le café ait ajouté une touche spéciale à votre expérience. Merci pour votre bonne humeur, au plaisir de vous revoir bientôt.
Propriétaire : mourad

Fermer

Commentaire : La location s'est bien passée dans l'ensemble, mais le réservoir n'était pas complètement plein au départ, ce qui m'a un peu gêné. Sinon, tout était parfait

Note : ★★★★ 1

Voir réponses

FIGURE 5.8 : Vue de « consultation du client de la liste de ses avis et des réponses des propriétaires faites dessus »

5.6 Tests

5.6.1 Test Unitaire

5.6.1.1 Objectif : Notation du propriétaire par le client après la location

- **Objet de Test :** Composant `NoterProprietaireComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Note valide (entre 1 et 5).
 - Note invalide (hors plage autorisée, ex. 0 ou 6).
- **Cas de Test :**
 - Vérifier si un client peut attribuer une note valide à un propriétaire.
 - Tester la validation de la note invalide (hors plage).

5.6.1.2 Objectif : Notation du client par le propriétaire après la location

- **Objet de Test :** Composant `NoterClientComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Note valide (entre 1 et 5).
 - Note invalide (hors plage autorisée, ex. 0 ou 6).
- **Cas de Test :**
 - Vérifier si un propriétaire peut attribuer une note valide à un client.
 - Tester la validation de la note invalide (hors plage).

5.6.1.3 Objectif : Réponse du client à une notation

- **Objet de Test :** Composant `RepondreNotationClientComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Réponse valide (texte non vide).
 - Réponse invalide (réponse vide ou texte non conforme).
- **Cas de Test :**
 - Vérifier si un client peut répondre à une notation reçue.
 - Tester la validation de la réponse vide ou non conforme.

5.6.1.4 Objectif : Réponse du propriétaire à une notation

- **Objet de Test :** Composant `RepondreNotationProprietaireComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Réponse valide (texte non vide).
 - Réponse invalide (réponse vide ou texte non conforme).
- **Cas de Test :**
 - Vérifier si un propriétaire peut répondre à une notation reçue.
 - Tester la validation de la réponse vide ou non conforme.

5.6.2 Test d'Intégration

5.6.2.1 Consultation des avis sur un propriétaire par un client

- **Objet de Test :** Composant `ConsultationAvisProprietaireComponent` (Angular) et méthode backend pour récupérer les avis sur un propriétaire.
- **Données de Test :**
 - Avis existants sur un propriétaire.
 - Pas d'avis disponibles.
- **Cas de Test :**
 - Vérifier si les avis d'un propriétaire sont correctement affichés pour un client.
 - Tester le cas où aucun avis n'est disponible et afficher un message approprié.

5.6.2.2 Consultation des avis sur un client par un propriétaire

- **Objet de Test :** Composant `ConsultationAvisClientComponent` (Angular) et méthode backend pour récupérer les avis sur un client.
- **Données de Test :**
 - Avis existants sur un client.
 - Pas d'avis disponibles.
- **Cas de Test :**
 - Vérifier si les avis d'un client sont correctement affichés pour un propriétaire.
 - Tester le cas où aucun avis n'est disponible et afficher un message approprié.

5.6.3 Test d'Acceptation

5.6.3.1 Création d'une notation par le client sur le propriétaire

- **Objectif** : Vérifier qu'un client peut noter un propriétaire après la location.
- **Objet de Test** : Interface utilisateur et méthode `NoterProprietaire` dans le backend.
- **Données de Test** :
 - Note valide (entre 1 et 5).
 - Note invalide (hors plage autorisée).
- **Cas de Test** :
 - Publier une note valide et vérifier son enregistrement.
 - Tester les validations pour une note invalide.

5.6.3.2 Création d'une notation par le propriétaire sur le client

- **Objectif** : Vérifier qu'un propriétaire peut noter un client après la location.
- **Objet de Test** : Interface utilisateur et méthode `NoterClient` dans le backend.
- **Données de Test** :
 - Note valide (entre 1 et 5).
 - Note invalide (hors plage autorisée).
- **Cas de Test** :
 - Publier une note valide et vérifier son enregistrement.
 - Tester les validations pour une note invalide.

5.6.3.3 Réponse du client à une notation reçue

- **Objectif** : Vérifier qu'un client peut répondre à une notation reçue.
- **Objet de Test** : Interface utilisateur et méthode `RepondreNotationClient` dans le backend.
- **Données de Test** :
 - Texte de réponse valide (non vide).
 - Réponse invalide (vide ou non conforme).
- **Cas de Test** :
 - Publier une réponse valide et vérifier son affichage.
 - Tester les validations pour une réponse vide.

5.6.3.4 Réponse du propriétaire à une notation reçue

- **Objectif** : Vérifier qu'un propriétaire peut répondre à une notation reçue.
- **Objet de Test** : Interface utilisateur et méthode `RepondreNotationProprietaire` dans le backend.
- **Données de Test** :
 - Texte de réponse valide (non vide).
 - Réponse invalide (vide ou non conforme).
- **Cas de Test** :
 - Publier une réponse valide et vérifier son affichage.
 - Tester les validations pour une réponse vide.

5.6.3.5 Consultation des avis sur un propriétaire par le client

- **Objectif** : Vérifier qu'un client peut consulter les avis sur un propriétaire.
- **Objet de Test** : Interface utilisateur et méthode `ConsultationAvisProprietaire` dans le backend.
- **Données de Test** :
 - Avis existants sur le propriétaire.
 - Aucun avis disponible.
- **Cas de Test** :
 - Vérifier l'affichage des avis pour un propriétaire.
 - Tester le message d'absence d'avis.

Introduction

Le 3ème sprint a permis d'avancer de manière significative dans le développement de la plateforme, notamment en intégrant un système de notation complet, tant pour les clients que pour les propriétaires. Les fonctionnalités telles que la notation réciproque, la consultation des avis et la possibilité de répondre aux évaluations ont été implémentées, offrant ainsi une expérience utilisateur plus interactive et transparente. Les tests unitaires, d'intégration et d'acceptation ont validé la fiabilité et l'efficacité de ces nouvelles fonctionnalités.

SPRINT 4 : IMPLÉMENTATION DES FONCTIONNALITÉS CLÉS

Plan

1	Présentation de l'équipe	59
2	Objectifs	59
3	Sprint Backlog	60
4	Conception :	60
5	Les interfaces :	62
6	Tests	67

Introduction

Ce chapitre présente les avancements réalisés lors du 4 ème sprint du projet, axé principalement sur les tableaux de bord et le développement d'un service d'assistance.

6.1 Présentation de l'équipe

Les membres de l'équipe sont définis comme suit :

- **Scrum Master** : Chef de projet de l'équipe de développement, *Mme Baccouche Mariem*.
- **Product Owner** : Responsable produit, *Farhaoui Eya*.
- **Membres de l'équipe des développeurs** :
 - *Soltani Wissal*
 - *Guesmi Ikram*

Durée : 29/11/2024 à 10/12/2024

6.2 Objectifs

Pour ce 4 ème sprint, l'objectif principal était l'implémentation des fonctionnalités suivantes sur la plateforme :

- **Tableau de bord pour les propriétaires** : En tant que propriétaire, je veux avoir accès à un tableau de bord qui présente les statistiques de mes locations.
- **Tableau de bord pour les locataires** : En tant que locataire, je veux avoir un tableau de bord qui présente mon historique de locations et mes notes.
- **Service d'assistance** : En tant que client ou propriétaire, je veux avoir accès à un service d'assistance pour poser des questions ou signaler un problème.

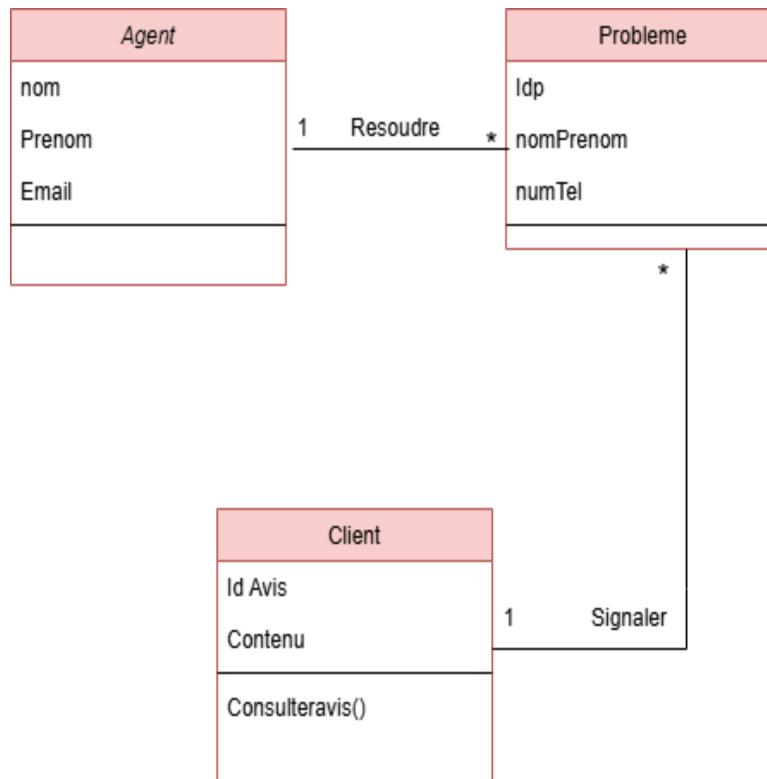
6.3 Sprint Backlog

User Story	Scrum team	Durée	État
En tant que propriétaire, je veux avoir accès à un tableau de bord qui présente les statistiques de mes locations	Ikram	2 jours	Achevé
En tant que locataire, je veux avoir un tableau de bord qui présente mon historique de locations et mes notes	Eya	2 jours	Achevé
En tant que client ou propriétaire, je veux avoir accès à un service d'assistance pour poser des questions ou signaler un problème	Ikram	2 jours	Achevé

TABLEAU 6.1 : Backlog du Sprint

6.4 Conception :

Le diagramme de classe illustre les entités clés du système et leurs relations. La classe Client et la est associée à la classe Problème, qui représente les demandes d'assistance. Un client peut créer un objet Problème pour signaler une question ou une difficulté. La classe Agent est responsable de la gestion des problèmes signalés par les utilisateurs.

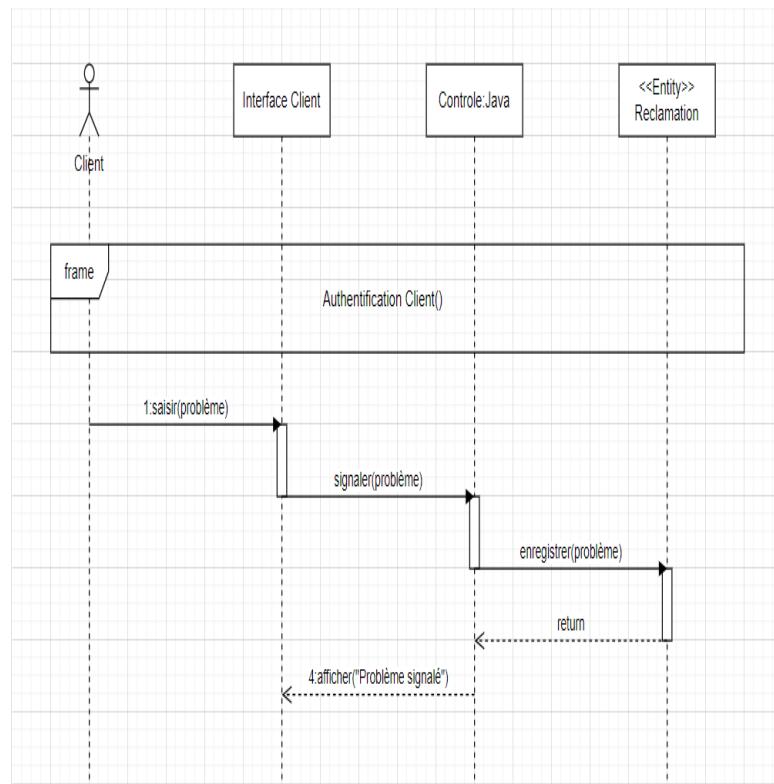
**FIGURE 6.1 :** Diagramme de classe du sprint 4

6.4.1 Diagramme de séquence :

Pour montrer comment les clients et les agents interagissent à travers le temps afin d'accomplir une tâche précise, on a réalisé ce diagrammes de séquence.

6.4.1.1 Scénario :Singnale d'un problème par un client

Ce diagramme illustre comment un Client peut signaler un problème :

**FIGURE 6.2 :** Diagramme de séquence du sprint 4

6.5 Les interfaces :

6.5.1 Tableau de bord du propriétaire :

Ce tableau de bord affiche des graphiques et des statistiques pour permettre au propriétaire de visualiser le nombre de location de ses voitures par moi et le pourcentage des locations payés.

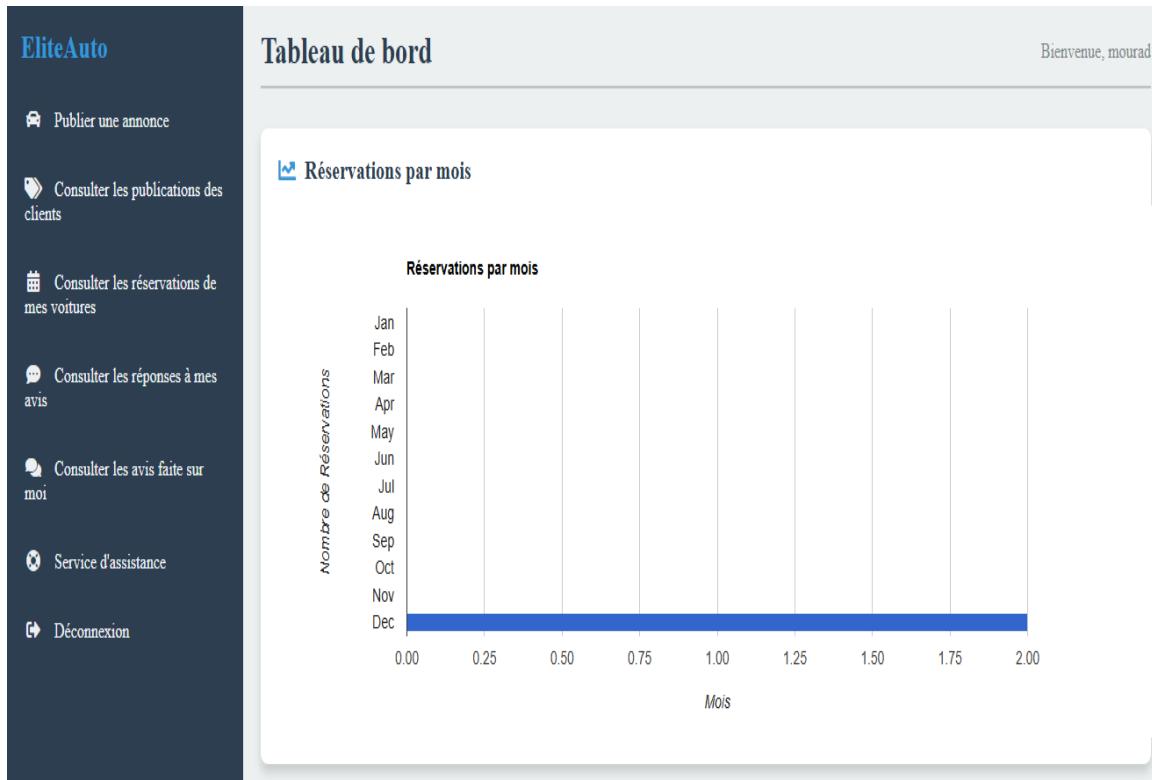


FIGURE 6.3 : Vue de « La premiere partie du tableau de bord du propriétaire»

Paiements

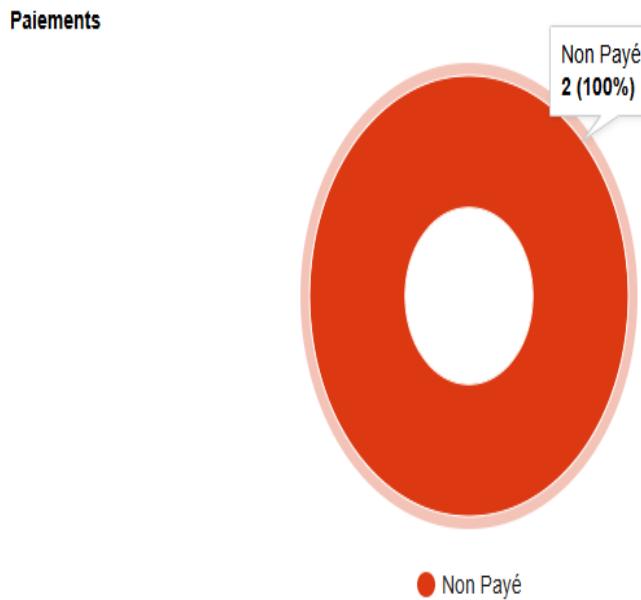


FIGURE 6.4 : Vue de « La deuxième partie du tableau de bord du propriétaire»

6.5.2 Tableau de bord du Client :

Ce tableau de bord permet au client de visualiser ses réservations par mois.

EliteAuto

- Publier une annonce
- Consulter les réponses
- Consulter les offres
- Mes réservations
- consulter les réponses à mes avis
- consulter les avis faite sur moi
- Service d'assistance
- Déconnexion

Tableau de bord

Bienvenue,Eya Farhaoui

Lil Réservations par mois

Mois	Nombre de Réservations
1	2

FIGURE 6.5 : Vue de « Tableau de bord du Client»

6.5.3 Crédation du compte de l'agent :

L'agent ne va pas crée manuellement un compte .Il va recevoir un email automatique qui contient le nom d'utilisateur et le mot de passe qu'il doit saisir à chaque fois qu'il veut se connecter à son compte.

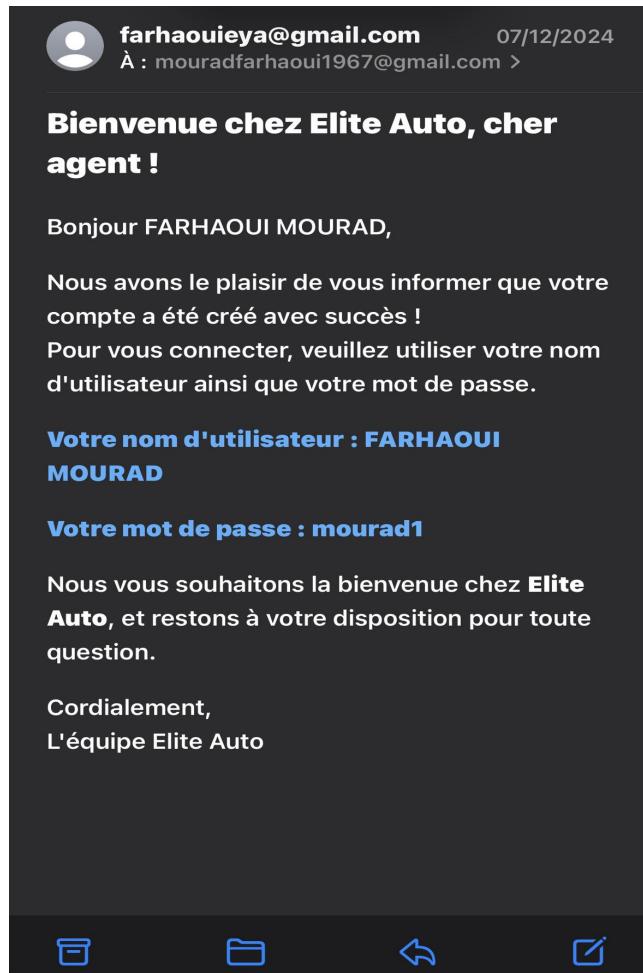


FIGURE 6.6 : Vue de « email de création du compte d'un agent»

6.5.4 Connexion de l'agent :

Après avoir reçu son nom d'utilisateur et son mot de passe, l'agent peut se connecter d'après la page d'accueil en cliquant au premier lieu sur se connecter et au second lieu sur connexion agent

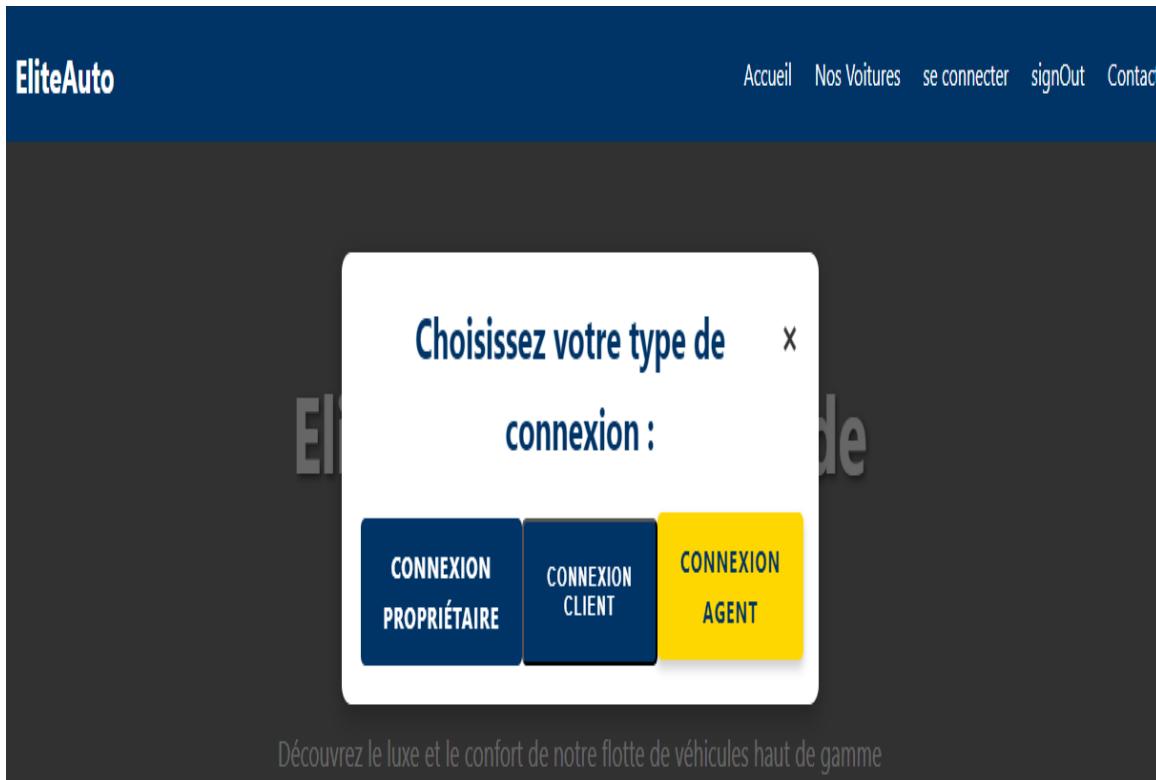


FIGURE 6.7 : Vue de « connexion d'un agent»

6.5.5 Réclamation du propriétaire :

En cliquant sur Service d'assistance, le propriétaire peut poser des questions ou faire des réclamations



FIGURE 6.8 : Vue de « l'ajout d'une réclamation par un propriétaire»

6.5.6 La liste des réclamation des propriétaire :

En cliquant sur les réclamations des propriétaires, l'agent peut consulter et répondre aux réclamations faites par les propriétaires.

The screenshot displays the 'Réclamations des Propriétaires' (Owner Complaints) section of the EliteAuto application. On the left, a sidebar menu lists 'Les Réclamations des clients' and 'Les Réclamations des propriétaires' (which is selected and highlighted in blue). Other options include 'Déconnexion'. The main content area shows two complaints:

- Owner: eya**
Phone: 554...
Complaint: rec de prop
Répondre à cette réclamation
- Owner: mourad**
Phone: 552...
Complaint: j'arrive pas à rejoindre la cliente Eya
Répondre à cette réclamation
Saisissez votre réponse ici...
Envoyer

FIGURE 6.9 : Vue de « La liste des réclamation des propriétaire»

6.6 Tests

6.6.1 Test Unitaire

6.6.1.1 Objectif : Tableau de bord du propriétaire

- **Objet de Test :** Composant TableauDeBordProprietaireComponent (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Statistiques de location valides.
 - Absence de données de location.
- **Cas de Test :**
 - Vérifier si les statistiques de location sont correctement affichées.
 - Tester le cas où il n'y a pas de données et afficher un message approprié.

6.6.1.2 Objectif : Tableau de bord du locataire

- **Objet de Test :** Composant `TableauDeBordLocataireComponent` (Angular) et méthode d'intégration dans le backend.
- **Données de Test :**
 - Historique des locations.
 - Notes sur les propriétaires.
- **Cas de Test :**
 - Vérifier si l'historique des locations et les notes sont correctement affichés.
 - Tester l'affichage des données vides et l'affichage d'un message approprié.

6.6.1.3 Objectif : Service d'assistance

- **Objet de Test :** Composant `ServiceAssistanceComponent` (Angular) et méthode backend.
- **Données de Test :**
 - Demandes d'assistance valides.
 - Absence de demande.
- **Cas de Test :**
 - Vérifier si la soumission d'une demande d'assistance fonctionne correctement.
 - Tester le cas où aucune demande n'a été soumise et afficher un message approprié.

Conclusion

Ce sprint a permis de mener à bien deux fonctionnalités essentielles pour nos utilisateurs. Tout d'abord, le tableau de bord pour le propriétaire, offrant une vue d'ensemble des statistiques de ses locations, et celui pour le locataire, qui présente son historique de locations et ses évaluations. Ces deux user stories ont été réalisées avec succès et sont maintenant achevées. Enfin, nous avons bien avancé sur le développement d'un service d'assistance, visant à répondre aux besoins des clients et des propriétaires pour toute question ou signalement de problème. Notre équipe a respecté les délais et a livré les fonctionnalités dans les temps, contribuant ainsi à améliorer l'expérience utilisateur.

APPLICATION MOBILE POUR LES CLIENTS

Plan

1	Connexion à la base de données	70
2	Package com.example.eliteauto.model	73
3	Classe Voiture	76
4	Package com.example.eliteauto.adapter	80
5	connexion via Facebook	86
6	Fichiers XML	92
7	Vue d'ensemble du fichier MainActivity.java	106
8	La Classe DetailsActivity	112
9	Package com.example.eliteauto.Service.DisponibiliteService	115
10	Les tests faits	124

Introduction

Dans ce chapitre, nous allons aborder l'implémentation de l'application mobile *EliteAuto* pour les clients. Nous allons détailler la façon dont l'application se connecte à un back-end pour récupérer des données sur les voitures disponibles, en utilisant une architecture API REST pour échanger des données au format JSON. L'intégration avec le back-end est facilitée par la bibliothèque Android **Retrofit**, qui simplifie les requêtes HTTP et la gestion des données.

Explication du code :

7.1 Connexion à la base de données

L'application mobile *EliteAuto* se connecte à un serveur local pour interroger la base de données et obtenir les informations nécessaires sur les voitures disponibles. Cette communication se fait via une API REST qui expose des données au format JSON. La connexion au back-end est gérée à l'aide de Retrofit, une bibliothèque Android dédiée à l'intégration des API REST. Le processus de connexion se divise en plusieurs étapes, que nous détaillons ci-dessous.

7.1.1 Configuration du client Retrofit (`ApiClient.java`)

Le client Retrofit est configuré pour gérer les requêtes HTTP et convertir les données JSON en objets Java. La classe `ApiClient` joue un rôle central dans cette configuration en spécifiant l'URL de base de l'API et en utilisant un convertisseur Gson personnalisé pour gérer le format particulier des dates, notamment le type `LocalDate` en Java.

Les étapes de la configuration sont les suivantes :

- **Gson personnalisé avec LocalDateAdapter** : Gson est utilisé pour convertir les données JSON en objets Java. Dans ce contexte, un adaptateur personnalisé (`LocalDateAdapter`) est utilisé afin de gérer la conversion des dates au format `LocalDate`, en tenant compte de la structure particulière des données de l'API.
- **Configuration du Retrofit Builder** : Le client Retrofit est configuré dans la méthode `getClient()`. Si une instance de `retrofit` n'existe pas encore, elle est créée avec la configuration nécessaire, incluant l'URL de base et le convertisseur Gson pour le traitement des données JSON. L'URL de base est définie comme "`http://10.0.2.2:8081/api/`", pointant vers le serveur local utilisé pendant le développement.

```
1
2     @Override
3
4         protected void onCreate(Bundle savedInstanceState) {
5
6             package com.example.eliteauto.network;
7
8
9
10            import com.example.eliteauto.adapter.LocalDateAdapter;
11
12            import com.google.gson.Gson;
13
14            import com.google.gson.GsonBuilder;
15
16
17            import java.time.LocalDate;
18
19
20            import retrofit2.Retrofit;
21
22            import retrofit2.converter.gson.GsonConverterFactory;
23
24
25            public class ApiClient {
26
27
28                private static final String BASE_URL = "http://10.0.2.2:8081/api/";
29
30                private static Retrofit retrofit;
31
32
33                public static Retrofit getClient() {
34
35                    if (retrofit == null) {
36
37                        // Configuration de Gson avec l'adaptateur LocalDate
38
39                        Gson gson = new GsonBuilder()
40
41                            .registerTypeAdapter(LocalDate.class, new LocalDateAdapter()) // Ajoute
42
43                            ↪ l'adaptateur pour LocalDate
44
45                            .create();
46
47
48                        // Configuration de Retrofit avec le Gson personnalisé
49
50                        retrofit = new Retrofit.Builder()
51
52                            .baseUrl(BASE_URL)
53
54                            .addConverterFactory(GsonConverterFactory.create(gson)) // Utilise Gson
55
56                            ↪ pour les conversions
57
58                            .build();
59
60
61                    }
62
63
64                    return retrofit;
65
66                }
67
68            }
```

36

37

7.1.2 Définition des services API (ApiService.java)

L’interface `ApiService` définit les points de terminaison (endpoints) que l’application peut interroger pour effectuer des opérations sur les données. Elle expose plusieurs méthodes, dont `getAllVoitures()`, qui permet de récupérer la liste des voitures disponibles en envoyant une requête HTTP GET vers le chemin "voitures". La méthode `loginOrRegister()` permet de gérer l’authentification ou l’enregistrement d’un client en envoyant un objet `Client` via une requête POST. De plus, la méthode `getAvisByProprietaireId()` permet de récupérer les avis d’un propriétaire en fonction de son identifiant, en envoyant une requête GET avec le paramètre `proprietaireId` dans l’URL "avis/avis-proprietaire/{proprietaireId}".

Ces méthodes utilisent la bibliothèque `Retrofit` pour simplifier la gestion des requêtes HTTP et le traitement des réponses, facilitant ainsi l’intégration avec le backend de l’application.

```

1 package com.example.eliteauto.network;

2

3 import com.example.eliteauto.model.AvisProprietaire;
4 import com.example.eliteauto.model.Client;
5 import com.example.eliteauto.model.Voiture;
6 import java.util.List;
7 import retrofit2.Call;
8 import retrofit2.http.Body;
9 import retrofit2.http.GET;
10 import retrofit2.http.POST;
11 import retrofit2.http.Path;

12

13 public interface ApiService {
14
15     @POST("clients/login")
16     Call<Client> loginOrRegister(@Body Client client);
17
18     @GET("voitures")
19     Call<List<Voiture>> getAllVoitures();
20
21     @GET("avis/avis-proprietaire/{proprietaireId}")

```

```
22     Call<List<AvisProprietaire>> getAvisByProprietaireId(@Path("proprietaireId") Long
23         ↵  proprietaireId);
24 }
```

7.2 Package com.example.eliteauto.model

Le dossier model contient des classes représentant les entités principales de l'application, telles que Client, AvisProprietaire, Disponibilite et Voiture. Chaque classe est responsable de la gestion des données associées, comme les informations sur les clients, les avis sur les propriétaires, les disponibilités des voitures et les caractéristiques des véhicules.

7.2.1 Classe AvisProprietaire

cette classe représente un avis sur un propriétaire, contenant trois attributs : le contenu de l'avis, la note attribuée (de type entier) et la date de l'avis. Elle inclut des méthodes getter et setter pour chaque attribut, permettant d'accéder et de modifier les informations.

```
1 package com.example.eliteauto.model;
2
3 public class AvisProprietaire {
4
5     private String contenu;
6
7     private int note;
8
9     private String dateAvis;
10
11    // Getters and Setters
12
13    public String getContenu() {
14
15        return contenu;
16    }
17
18    public void setContenu(String contenu) {
19
20        this.contenu = contenu;
21    }
22
23    public int getNote() {
24
25        return note;
26    }
27}
```

```
20
21     public void setNote(int note) {
22         this.note = note;
23     }
24
25
26     public String getDateAvis() {
27         return dateAvis;
28     }
29
30     public void setDateAvis(String dateAvis) {
31         this.dateAvis = dateAvis;
32     }
33 }
```

7.2.2 Classe client

cette classe définit un objet client avec trois propriétés principales : l'ID Facebook, le nom, et l'email.

```
1 package com.example.eliteauto.model;
2
3
4 public class Client {
5
6     private String facebookId;    //l'ID Facebook
7     private String name;          // Le nom du client
8     private String email;         // Email du client
9     public Client() {
10
11 }
12     public Client(String email) {
13         this.email=email;
14     }
15
16     // Getters et setters
17 
```

```
18     public String getFacebookId() {
19
20         return facebookId;
21     }
22
23     public void setFacebookId(String facebookId) {
24
25         this.facebookId = facebookId;
26     }
27
28     public String getName() {
29
30         return name;
31     }
32
33
34     public String getEmail() {
35
36         return email;
37     }
38
39     public void setEmail(String email) {
40
41         this.email = email;}}
```

7.2.3 Classe Disponibilite

La classe Disponibilite contient deux attributs principaux : dateDebutDisponibilite et dateFinDisponibilite, qui sont des chaînes de caractères représentant des dates. Pour faciliter leur gestion dans les formats adaptés, la classe fournit des méthodes permettant de convertir ces dates en objets LocalDate grâce à l'utilisation de DateTimeFormatter.ISO_DATE. Cela permet d'assurer une manipulation plus flexible des dates tout en respectant les formats utilisés dans le backend.

```
1 package com.example.eliteauto.model;
2
3
4 import java.time.LocalDate;
5 import java.time.format.DateTimeFormatter;
6 import com.google.gson.annotations.SerializedName;
```

```
7
8 public class Disponibilite {
9     @SerializedName("id")
10    private Long id;
11
12    @SerializedName("dateDebutDisponibilite")
13    private String dateDebutDisponibilite;
14
15    @SerializedName("dateFinDisponibilite")
16    private String dateFinDisponibilite;
17
18    // Méthodes pour convertir les dates au format LocalDate si nécessaire
19    public LocalDate getDateDebutDisponibilite() {
20        return LocalDate.parse(dateDebutDisponibilite, DateTimeFormatter.ISO_DATE);
21    }
22
23    public LocalDate getDateFinDisponibilite() {
24        return LocalDate.parse(dateFinDisponibilite, DateTimeFormatter.ISO_DATE);
25    }
26}
```

7.3 Classe Voiture

La classe Voiture représente un objet voiture avec divers attributs, comme marque, modele, prix, nombrePortes, etc. Elle contient également un attribut disponibilites, qui est une liste d'objets Disponibilite, permettant de suivre les périodes où la voiture est disponible.

Le getter getImageUrl() permet de générer dynamiquement l'URL de l'image de la voiture, ce qui facilite l'intégration des images dans l'interface utilisateur.

```
1 package com.example.eliteauto.model;
2
3 import java.math.BigDecimal;
4 import java.util.List;
5
6 public class Voiture {
7     private Long id;
```

```
8     private String marque;
9
10    private String modele;
11
12    private Integer annee;
13
14    private BigDecimal prix;
15
16    private Integer nombrePortes;
17
18    private String typeCarburant;
19
20    private BigDecimal montantCaution;
21
22    private Integer nombreChevaux;
23
24    private Integer nombrePassagers;
25
26    private String imagePath; // Can be URL or base64 string (BLOB)
27
28    private List<Disponibilite> disponibilites; // Ajout de ce champ
29
30
31    // Getters and Setters
32
33    public String getImageUrl() {
34
35        return "http://10.0.2.2:8081/api/voitures/getImage/" + id;
36
37    }
38
39    public Integer getAnnee() {
40
41        return annee;
42
43    }
44
45    public void setAnnee(Integer annee) {
46
47        this.annee = annee;
48
49    }
50
51    public Long getId() {
52
53        return id;
54
55    }
56
57    public Integer getNombrePortes() {
58
59        return nombrePortes;
60
61    }
62
63    public void setNombrePortes(Integer nombrePortes) {
64
65        this.nombrePortes = nombrePortes;
66
67    }
68
69
70    public String getTypeCarburant() {
71
72        return typeCarburant;
73
74    }
```

```
45
46     public void setTypeCarburant(String typeCarburant) {
47         this.typeCarburant = typeCarburant;
48     }
49
50     public BigDecimal getMontantCaution() {
51         return montantCaution;
52     }
53
54     public void setMontantCaution(BigDecimal montantCaution) {
55         this.montantCaution = montantCaution;
56     }
57
58     public Integer getNombreChevaux() {
59         return nombreChevaux;
60     }
61
62     public void setNombreChevaux(Integer nombreChevaux) {
63         this.nombreChevaux = nombreChevaux;
64     }
65
66     public Integer getNombrePassagers() {
67         return nombrePassagers;
68     }
69
70     public void setNombrePassagers(Integer nombrePassagers) {
71         this.nombrePassagers = nombrePassagers;
72     }
73
74     public void setId(Long id) {
75         this.id = id;
76     }
77
78     public String getMarque() {
79         return marque;
80     }
81
```

```
82     public void setMarque(String marque) {
83
84         this.marque = marque;
85     }
86
87     public String getModele() {
88
89         return modele;
90     }
91
92     public void setModele(String modele) {
93
94         this.modele = modele;
95     }
96
97
98     public BigDecimal getPrix() {
99
100        return prix;
101    }
102
103    public BigDecimal getPrixParJour() {
104
105        return this.prix;
106    }
107
108    public String getImage() {
109
110        return imagePath;
111    }
112
113
114    public List<Disponibilite> getDisponibilites() {
115
116        return disponibilites;
117    }
118
119    public void setDisponibilites(List<Disponibilite> disponibilites) {
```

```
119     this.disponibilites = disponibilites;
120 }
121 }
122
```

7.4 Package com.example.eliteauto.adapter

Les classes dans ce package permettent de gérer la conversion JSON pour les objets LocalDate ainsi que l'affichage des avis faits sur un propriétaire et les informations des voitures dans une RecyclerView Android.

7.4.1 AvisAdapter

Cette classe est un adaptateur pour un RecyclerView qui affiche une liste d'avis de propriétaires(AvisProprietaire). Elle utilise le modèle ViewHolder pour créer et lier des vues à partir d'une mise en page XML définie dans itemavis.xml. Chaque élément de la liste affiche le contenu d'un avis grâce à la méthode 'bind()'.

```
1 package com.example.eliteauto.adapter;
2
3 import android.view.LayoutInflater;
4 import android.view.View;
5 import android.view.ViewGroup;
6 import android.view.View;
7 import android.view.ViewGroup;
8 import android.widget.TextView;
9 import androidx.recyclerview.widget.RecyclerView;
10
11 import com.example.eliteauto.R;
12 import com.example.eliteauto.model.AvisProprietaire;
13 import java.util.List;
14
15
16 import androidx.recyclerview.widget.RecyclerView;
17
18 public class AvisAdapter extends RecyclerView.Adapter<AvisAdapter.AvisViewHolder> {
19     private List<AvisProprietaire> avisList;
20 }
```

```
21  public AvisAdapter(List<AvisProprietaire> avisList) {
22
23      this.avisList = avisList;
24
25
26      @Override
27
28      public AvisViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
29
30          View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_avis,
31
32              parent, false);
33
34          return new AvisViewHolder(view);
35
36
37      @Override
38
39      public void onBindViewHolder(AvisViewHolder holder, int position) {
40
41          AvisProprietaire avis = avisList.get(position);
42
43          holder.bind(avis);
44
45      }
46
47
48      public int getItemCount() {
49
50          return avisList.size();
51
52      }
53
54
55      public static class AvisViewHolder extends RecyclerView.ViewHolder {
56
57          private TextView textViewAvis;
58
59
60          public AvisViewHolder(View itemView) {
61
62              super(itemView);
63
64              textViewAvis = itemView.findViewById(R.id.textAvis);
65
66          }
67
68
69          public void bind(AvisProprietaire avis) {
70
71              textViewAvis.setText(avis.getContenu()); // affichage des commentaires
72
73          }
74
75      }
76
77  }
```

7.4.2 LocalDateAdapter

Cette classe sert à adapter les objets LocalDate en chaînes JSON et inversement. Le format yyyy-MM-dd est utilisé pour la sérialisation et la désérialisation des dates.

```
1 package com.example.eliteauto.adapter;  
2  
3 import com.google.gson.*;  
4 import java.lang.reflect.Type;  
5 import java.time.LocalDate;  
6 import java.time.format.DateTimeFormatter;  
7  
8 public class LocalDateAdapter implements JsonSerializer<LocalDate>,  
9     JsonDeserializer<LocalDate> {  
10  
11     private static final DateTimeFormatter formatter =  
12         DateTimeFormatter.ofPattern("yyyy-MM-dd");  
13  
14     @Override  
15     public JsonElement serialize(LocalDate date, Type typeOfSrc, JsonSerializationContext  
16         context) {  
17         return new JsonPrimitive(date.format(formatter)); // Convert LocalDate en chaîne  
18         JSON  
19     }  
20  
21     @Override  
22     public LocalDate deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext  
23         context) throws JsonParseException {  
24         return LocalDate.parse(json.getAsString(), formatter); // Convert chaîne JSON en  
25         LocalDate  
26     }  
27 }
```

7.4.3 VoitureAdapter

Cette classe est responsable de l'affichage des objets Voiture dans une RecyclerView et de la liaison des voitures à leurs avis, en permettant la navigation vers l'activité dédiée pour consulter les commentaires faits sur un propriétaire. En plus, Elle formate les disponibilités pour les afficher dans

l’interface utilisateur et charge les images à l’aide de la bibliothèque Glide.

```
1 package com.example.eliteauto.adapter;  
2  
3 import android.content.Intent;  
4 import android.view.LayoutInflater;  
5 import android.view.View;  
6 import android.view.ViewGroup;  
7 import android.widget.Button;  
8 import android.widget.ImageView;  
9 import android.widget.TextView;  
10  
11 import androidx.annotation.NonNull;  
12 import androidx.recyclerview.widget.RecyclerView;  
13  
14 import com.bumptech.glide.Glide;  
15 import com.bumptech.glide.load.engine.DiskCacheStrategy;  
16 import com.example.eliteauto.R;  
17 import com.example.eliteauto.model.Voiture;  
18 import com.example.eliteauto.model.Disponibilite;  
19 import com.example.eliteauto.ui.AvisActivity; // Import AvisActivity  
20  
21 import java.util.List;  
22  
23 public class VoitureAdapter extends RecyclerView.Adapter<VoitureAdapter.VoitureViewHolder>  
24 {  
25     private List<Voiture> voitures;  
26     private OnItemClickListener listener;  
27  
28     public VoitureAdapter(List<Voiture> voitures, OnItemClickListener listener) {  
29         this.voitures = voitures;  
30         this.listener = listener;  
31     }  
32  
33     @NonNull  
34     @Override  
35     public VoitureViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
```

```
36         View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_voiture,
37             ↪ parent, false);
38
39         return new VoitureViewHolder(view);
40     }
41
42     @Override
43     public void onBindViewHolder(@NonNull VoitureViewHolder holder, int position) {
44
45         Voiture voiture = voitures.get(position);
46
47         holder.textMarqueModele.setText(voiture.getMarque() + " " + voiture.getModele());
48         holder.textPrix.setText(voiture.getPrixParJour() + " €/jour");
49
50
51         // Formatage des disponibilités
52         String disponibilitesFormatted = formatDisponibilites(voiture.getDisponibilites());
53         holder.textDisponibilites.setText("Disponibilités : " + disponibilitesFormatted);
54
55
56         // Chargement de l'image avec Glide
57         Glide.with(holder.itemView.getContext())
58             .load(voiture.getImageUrl())
59             .diskCacheStrategy(DiskCacheStrategy.ALL)
60             .into(holder.imageVoiture);
61
62
63         holder.buttonConsulter.setOnClickListener(v -> listener.onItemClick(voiture));
64
65
66         // ajouter OnClickListener pour "View Avis" button
67         holder.btnViewAvis.setOnClickListener(v -> {
68
69             Intent intent = new Intent(v.getContext(), AvisActivity.class);
70             intent.putExtra("proprietaireId", Long.valueOf(
71                 ↪ voiture.getProprietaireId()));v.getContext().startActivity(intent);
72         });
73
74
75     @Override
76     public int getItemCount() {
77
78         return voitures.size();
79     }
80
81 }
```

```
71 // Méthode pour formater les disponibilités
72
73 private String formatDisponibilites(List<Disponibilite> disponibilites) {
74
75     if (disponibilites == null || disponibilites.isEmpty()) {
76
77         return "Aucune";
78
79     }
80
81
82     StringBuilder disponibilitesBuilder = new StringBuilder();
83
84     for (Disponibilite dispo : disponibilites) {
85
86         disponibilitesBuilder.append("Du ")
87             .append(dispo.getDateDebutDisponibilite())
88             .append(" au ")
89             .append(dispo.getDateFinDisponibilite())
90             .append("\n");
91
92     }
93
94
95     return disponibilitesBuilder.toString().trim();
96
97 }
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2196
2197
2198
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2217
2218
2219
2219

```

7.5 connexion via Facebook

Notre application EliteAuto permet aux utilisateurs de se connecter via leur compte Facebook. En utilisant le SDK Facebook, l'email de l'utilisateur est récupéré et envoyé au backend pour l'inscription ou la connexion. Cette méthode d'authentification simplifie le processus de connexion tout en garantissant une expérience utilisateur fluide et sécurisée. Dans ce qui suit, nous vous présenterons en détail les fichiers concernés, notamment LoginActivity, ApiService, le fichier AndroidManifest.xml et les ressources associées.

7.5.1 Explication de la logique du login Facebook :

- Lorsque l'utilisateur clique sur le bouton de connexion, le SDK de Facebook gère la connexion.
- Si la connexion est réussie, l'email de l'utilisateur est extrait et envoyé au backend via l'API loginOrRegister.
- Si la connexion ou l'inscription est réussie, l'utilisateur est redirigé vers l'écran principal (DashboardActivity).

7.5.2 LoginActivity (Authentification via Facebook)

La classe LoginActivity implémente l'authentification des utilisateurs via Facebook en utilisant le SDK Facebook. Elle configure un bouton de connexion (LoginButton) pour obtenir les permissions nécessaires, gère les callbacks avec un CallbackManager, récupère l'email utilisateur à l'aide d'une requête Graph Facebook, puis envoie cette information au backend via Retrofit pour effectuer l'inscription ou la connexion de l'utilisateur.

Gestion des événements de connexion avec CallbackManager :

Le CallbackManager est un gestionnaire central qui capture et traite les réponses du SDK Facebook liées aux actions de connexion.

Bouton de connexion LoginButton

Le LoginButton intégré permet à l'utilisateur de se connecter à son compte Facebook. Il est configuré pour demander des autorisations spécifiques comme l'accès à l'email et au profil public.

Écoute des événements avec FacebookCallback :

Cette interface détecte trois types d'événements :

- Succès : L'utilisateur s'est connecté avec succès. L'application récupère ensuite l'AccessToken pour extraire les informations de l'utilisateur.
- Annulation : L'utilisateur a annulé la tentative de connexion.
- Erreur : Une erreur s'est produite lors de la tentative de connexion.

Méthode fetchEmailFromFacebook(AccessToken accessToken) :

Une requête Graph est effectuée avec l'AccessToken pour obtenir l'email de l'utilisateur connecté. Cette information est extraite et utilisée pour l'inscription ou la connexion.

Méthode loginOrRegisterClient(String email) :

Une fois l'email récupéré, cette méthode utilise Retrofit pour envoyer une requête au backend. L'API vérifie si l'utilisateur existe :

Si l'utilisateur existe : Il est connecté.

Sinon : L'utilisateur est inscrit, puis connecté.

```
1
2
3 import android.content.Intent;
4 import android.os.Bundle;
5 import android.util.Log;
6 import android.widget.Toast;
7
8 import androidx.appcompat.app.AppCompatActivity;
9
10 import com.example.eliteauto.R;
11 import com.example.eliteauto.model.Client;
12 import com.example.eliteauto.network.ApiClient;
13 import com.example.eliteauto.network.ApiService;
14 import com.facebook.AccessToken;
15 import com.facebook.CallbackManager;
16 import com.facebook.FacebookCallback;
17 import com.facebook.FacebookException;
18 import com.facebook.GraphRequest;
19 import com.facebook.HttpMethod;
20 import com.facebook.login.LoginResult;
21 import com.facebook.login.widget.LoginButton;
22
23 import org.json.JSONObject;
24
25 import retrofit2.Call;
26 import retrofit2.Callback;
27 import retrofit2.Response;
```

```
29 public class LoginActivity extends AppCompatActivity {  
30  
31     private CallbackManager callbackManager;  
32  
33     @Override  
34     protected void onCreate(Bundle savedInstanceState) {  
35         super.onCreate(savedInstanceState);  
36         setContentView(R.layout.activity_login);  
37  
38         callbackManager = CallbackManager.Factory.create();  
39  
40         LoginButton loginButton = findViewById(R.id.login_button);  
41         loginButton.setPermissions("email", "public_profile");  
42  
43         loginButton.registerCallback(callbackManager, new FacebookCallback<LoginResult>() {  
44             @Override  
45             public void onSuccess(LoginResult loginResult) {  
46                 AccessToken accessToken = loginResult.getAccessToken();  
47                 fetchEmailFromFacebook(accessToken);  
48             }  
49  
50             @Override  
51             public void onCancel() {  
52                 Toast.makeText(LoginActivity.this, "Login annulé",  
53                             Toast.LENGTH_SHORT).show();  
54             }  
55             @Override  
56             public void onError(FacebookException error) {  
57                 Toast.makeText(LoginActivity.this, "Erreur de login : " +  
58                             error.getMessage(), Toast.LENGTH_SHORT).show();  
59             }  
60         });  
61  
62         private void fetchEmailFromFacebook(AccessToken accessToken) {  
63             new GraphRequest(  
64                 ...  
65             ).registerCallback(callbackManager);  
66         }  
67     }  
68  
69     @Override  
70     protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
71         super.onActivityResult(requestCode, resultCode, data);  
72         callbackManager.onActivityResult(requestCode, resultCode, data);  
73     }  
74 }  
75  
76 // Fetching the user's email from Facebook  
77 private void fetchEmailFromFacebook(AccessToken accessToken) {  
78     GraphRequest request = new GraphRequest(accessToken, "/me", null, HttpMethod.GET, new GraphRequest.Callback() {  
79         @Override  
80         public void onCompleted(GraphResponse response) {  
81             try {  
82                 JSONObject json = response.getJSONObject();  
83                 String email = json.getString("email");  
84                 Log.d("Email", "Email : " + email);  
85             } catch (JSONException e) {  
86                 e.printStackTrace();  
87             }  
88         }  
89     });  
90     request.executeAsync();  
91 }
```



```

97             Intent intent = new Intent(LoginActivity.this, DashboardActivity.class);
98             startActivity(intent);
99             finish();
100         } else {
101             Log.e("BackendResponse", "Erreur: " + response.code());
102             Toast.makeText(LoginActivity.this, "Erreur de connexion ou
103             ↳ d'inscription", Toast.LENGTH_SHORT).show();
104         }
105     }
106
107     @Override
108     public void onFailure(Call<Client> call, Throwable t) {
109         Log.e("BackendRequest", "Erreur lors de l'appel API: " + t.getMessage());
110         Toast.makeText(LoginActivity.this, "Erreur réseau",
111             ↳ Toast.LENGTH_SHORT).show();
112     });
113 }
114
115     @Override
116     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
117         super.onActivityResult(requestCode, resultCode, data);
118         callbackManager.onActivityResult(requestCode, resultCode, data);
119     }

```

7.5.3 AndroidManifest.xml

Permissions

Le fichier AndroidManifest.xml déclare les permissions nécessaires pour permettre à l'application de fonctionner correctement :

- **Internet** : Permet d'effectuer des requêtes réseau, notamment pour communiquer avec le backend et les API Facebook.

```
<uses-permission android:name="android.permission.INTERNET" />
```

- **État du réseau** : Vérifie l'état de la connexion réseau avant d'effectuer des requêtes :

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Configuration du SDK Facebook

Le manifeste configure le SDK Facebook en ajoutant les métadonnées suivantes pour identifier l'application auprès des serveurs Facebook :

- **Application ID** : L'identifiant unique de l'application.

```
<meta-data  
    android:name="com.facebook.sdk.ApplicationId"  
    android:value="@string/facebook_app_id" />
```

- **Client Token** : Un jeton client utilisé pour sécuriser les interactions avec les serveurs Facebook

```
<meta-data  
    android:name="com.facebook.sdk.ClientToken"  
    android:value="@string/facebook_client_token" />
```

7.5.4 values/strings.xml

Le fichier strings.xml contient les informations essentielles pour la configuration du SDK Facebook ainsi que d'autres chaînes nécessaires au fonctionnement de l'application. Voici les détails :

- **ID de l'application Facebook (facebook_app_id)** : Cet identifiant unique est fourni par Facebook lors de la création de l'application sur le tableau de bord Facebook Developers. Il permet de lier l'application mobile à son projet Facebook.

```
<string name="facebook_app_id">851462143702467</string>
```

- **Token client (facebook_client_token)** : Ce jeton est utilisé pour sécuriser les interactions entre l'application mobile et Facebook, garantissant que seules des requêtes légitimes sont effectuées.

```
<string name="facebook_client_token">dc03b6b632f9891777ae857a7e260589</string>
```

7.6 Fichiers XML

7.6.1 activity_main.xml

Ce fichier XML définit la structure principale de l'interface utilisateur de l'activité principale.

Voici un résumé de sa composition :

- **CoordinatorLayout** : Le conteneur racine, qui permet une gestion avancée des vues et de leur comportement lors du défilement, avec une barre d'outils intégrée et un comportement de vue qui défile avec la barre.
- **AppBarLayout** : Un composant contenant la **MaterialToolbar** qui sert de barre d'outils en haut de l'écran, où le titre “EliteAuto” est affiché.
- **SwipeRefreshLayout** : Permet de rajouter une fonctionnalité de rafraîchissement par glissement vers le bas (pull-to-refresh). Il contient un **RecyclerView** pour afficher une liste de voitures avec leurs détails.
- **RecyclerView** : Utilisé pour afficher une liste d'éléments, dans ce cas, des objets **Voiture**. C'est une vue très flexible et optimisée pour les listes longues.
- **FloatingActionButton (FAB)** : Un bouton flottant en bas à droite pour permettre des actions supplémentaires comme un filtrage (par exemple, appliquer un filtre sur les voitures listées).

Le fichier met en place une interface simple et fluide, avec une possibilité de rafraîchir la liste des voitures et d'ajouter des actions supplémentaires via un bouton flottant.

```
1
2 <?xml version="1.0" encoding="utf-8"?>
3 <androidx.coordinatorlayout.widget.CoordinatorLayout
4     xmlns:android="http://schemas.android.com/apk/res/android"
5     xmlns:app="http://schemas.android.com/apk/res-auto"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent">
8
9     <com.google.android.material.appbar.AppBarLayout
10        android:id="@+id/appBarLayout"
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content">
13
14         <com.google.android.material.appbar.MaterialToolbar
```

```
15         android:id="@+id/toolbar"
16
17         android:layout_width="match_parent"
18
19         android:layout_height="?attr/actionBarSize"
20
21         android:background="?attr/colorPrimary"
22
23         app:title="EliteAuto"
24
25         app:titleTextColor="@android:color/white"
26
27         app:layout_scrollFlags="scroll|enterAlways"/>
28
29
30
31     </com.google.android.material.appbar.AppBarLayout>
32
33
34     <androidx.swiperefreshlayout.widget.SwipeRefreshLayout
35
36         android:id="@+id/swipeRefreshLayout"
37
38         android:layout_width="match_parent"
39
40         android:layout_height="match_parent"
41
42         app:layout_behavior="@string/appbar_scrolling_view_behavior">
43
44
45         <androidx.recyclerview.widget.RecyclerView
46
47             android:id="@+id/recyclerView"
48
49             android:layout_width="match_parent"
50
51             android:layout_height="match_parent"/>
52
53
54     </androidx.swiperefreshlayout.widget.SwipeRefreshLayout>
55
56
57     <com.google.android.material.floatingactionbutton.FloatingActionButton
58
59         android:id="@+id/fabFilter"
60
61         android:layout_width="wrap_content"
62
63         android:layout_height="wrap_content"
64
65         android:layout_gravity="bottom|end"
66
67         android:layout_margin="16dp"
68
69         app:fabSize="normal"/>
70
71
72     </androidx.coordinatorlayout.widget.CoordinatorLayout>
73
```

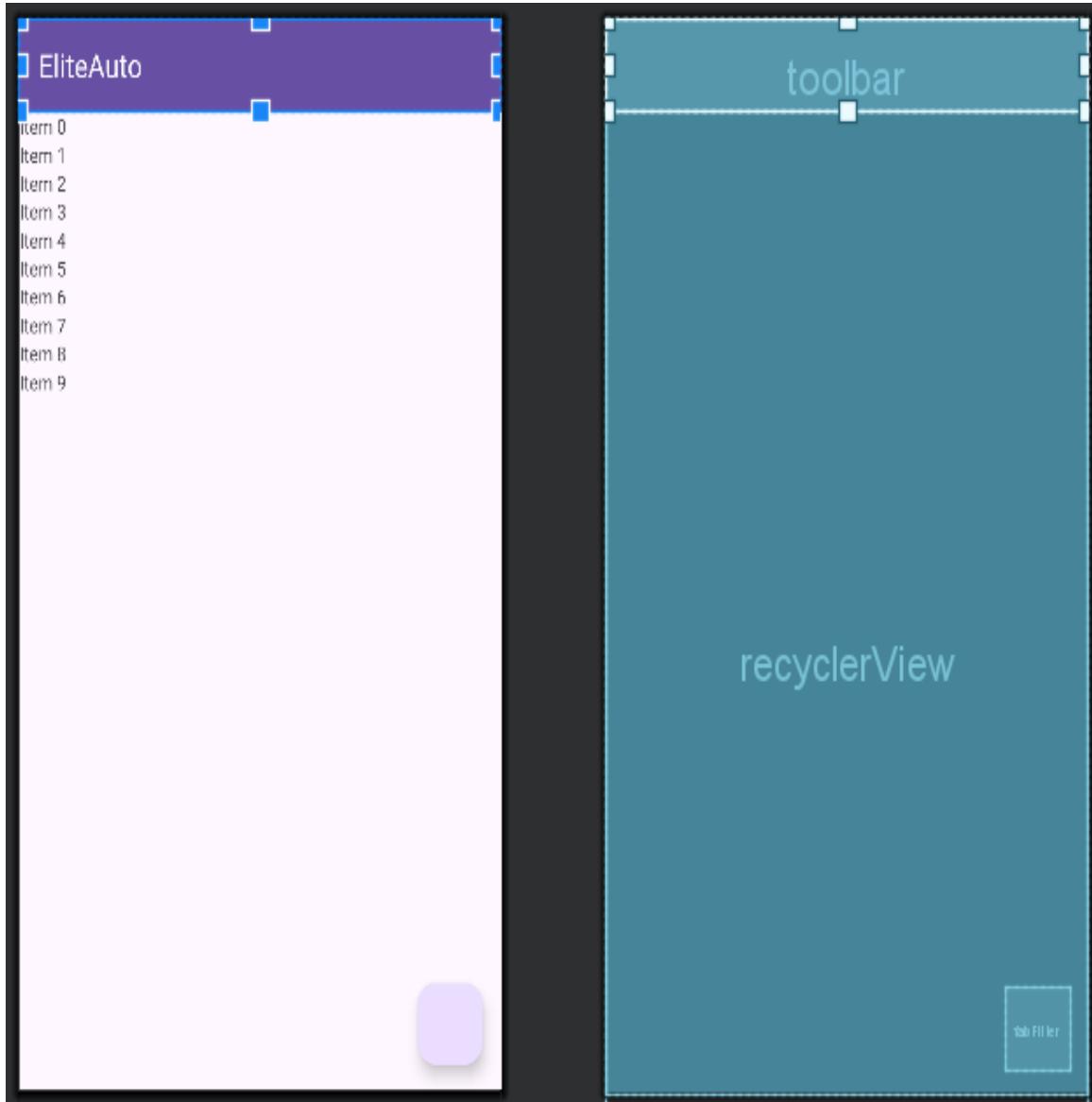


FIGURE 7.1 : Layout de l'activité principale

7.6.2 item_voiture.xml

Ce fichier définit l'apparence de chaque élément individuel dans la liste des voitures, affiché par le `RecyclerView`.

- **MaterialCardView** : Un composant de carte avec des coins arrondis et une élévation pour donner un effet d'ombre et de profondeur.
- **LinearLayout** : Organise les éléments à l'intérieur de la carte verticalement.
- **ImageView** : Affiche l'image de la voiture avec un identifiant spécifique pour permettre son accès dans le code. Elle utilise `Glide` pour charger l'image de manière optimisée.
- **TextView** : Affiche le nom et le prix de la voiture ainsi que ses disponibilités.

Ce fichier est utilisé pour formater et organiser l'affichage des informations sur chaque voiture dans la liste.

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <com.google.android.material.card.MaterialCardView
4      xmlns:android="http://schemas.android.com/apk/res/android"
5      xmlns:app="http://schemas.android.com/apk/res-auto"
6      android:layout_width="match_parent"
7      android:layout_height="wrap_content"
8      android:layout_margin="8dp"
9      app:cardElevation="4dp"
10     app:cardCornerRadius="8dp">
11
12
13     <LinearLayout
14         android:layout_width="match_parent"
15         android:layout_height="wrap_content"
16         android:orientation="vertical"
17         android:padding="16dp">
18
19         <!-- Image de la voiture -->
20
21         <ImageView
22             android:id="@+id/imageVoiture"
23             android:layout_width="match_parent"
24             android:layout_height="200dp"
25             android:scaleType="centerCrop"
26             android:transitionName="imageVoiture"/>
27
28
29         <!-- Conteneur pour les informations -->
30
31         <LinearLayout
32             android:layout_width="match_parent"
33             android:layout_height="wrap_content"
34             android:orientation="vertical"
35             android:layout_marginTop="8dp">
36
37             <!-- Marque et modèle de la voiture -->
38
39             <TextView
40                 android:id="@+id/textMarqueModele"
41                 android:layout_width="wrap_content"
42                 android:layout_height="wrap_content"
43                 android:textStyle="bold"
```

```
38         android:textSize="18sp"
39         android:textColor="@android:color/black"/>
40
41     <!-- Prix de la voiture -->
42
43     <LinearLayout
44         android:layout_width="match_parent"
45         android:layout_height="wrap_content"
46         android:orientation="horizontal"
47         android:layout_marginTop="8dp">
48
49         <ImageView
50             android:layout_width="24dp"
51             android:layout_height="24dp"
52             android:src="@drawable/ic_euro"
53             app:tint="@android:color/holo_green_dark"/>
54
55         <TextView
56             android:id="@+id/textPrix"
57             android:layout_width="wrap_content"
58             android:layout_height="wrap_content"
59             android:textSize="16sp"
60             android:textColor="@android:color/holo_green_dark"
61             android:layout_marginStart="4dp"/>
62
63     </LinearLayout>
64
65
66     <!-- Disponibilité de la voiture -->
67
68     <LinearLayout
69         android:layout_width="match_parent"
70         android:layout_height="wrap_content"
71         android:orientation="horizontal"
72         android:layout_marginTop="8dp">
73
74         <ImageView
```

```
75         android:textSize="14sp"
76
77         android:textColor="@android:color/black"
78         android:layout_marginStart="4dp"/>
79
80     <!-- Bouton Consulter -->
81
82     <com.google.android.material.button.MaterialButton
83         android:id="@+id/buttonConsulter"
84         android:layout_width="match_parent"
85         android:layout_height="wrap_content"
86         android:text="Consulter"
87         android:layout_marginTop="16dp"
88
89         app:iconGravity="textStart"
90         style="@style/Widget.MaterialComponents.Button.OutlinedButton"/>
91
92     </LinearLayout>
93
94 </LinearLayout>
95
96
```

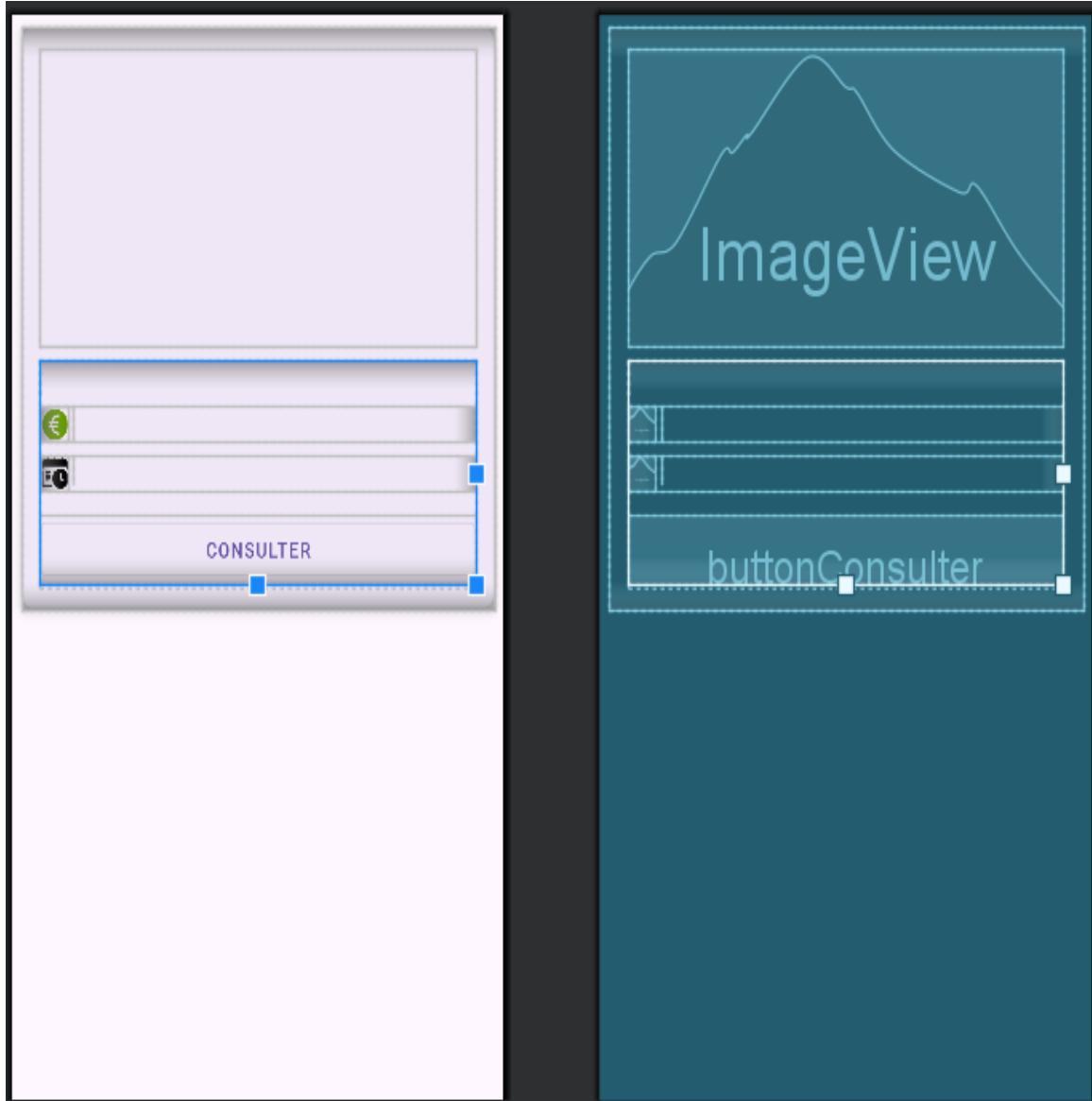


FIGURE 7.2 : Layout de itemvoiture

7.6.3 menu.xml

Le fichier XML définit un menu d'options qui sera affiché dans la barre d'outils (Toolbar) ou dans le menu d'options de l'activité Android. Ce menu contient deux éléments, chacun ayant une icône, un identifiant unique, et un titre associé.

7.6.3.1 Élément action_connect

- `android:id="@+id/action_connect"` : Cet attribut attribue un identifiant unique à cet élément de menu. Cela permet de faire référence à cet élément dans le code Java pour lui associer des actions spécifiques lorsque l'utilisateur clique dessus.
- `android:title="Se connecter"` : Le titre affiché dans le menu pour cet élément est "Se connecter". C'est le texte qui sera vu par l'utilisateur dans le menu d'options.

- `android:icon="@android:drawable/ic_menu_add"` : L'icône associée à cet élément est l'icône de "ajout" par défaut fournie par Android. Elle est affichée à côté du titre "Se connecter".

7.6.3.2 Élément `action_disconnect`

- `android:id="@+id/action_disconnect"` : Cet identifiant unique permet de gérer cet élément dans le code, par exemple pour effectuer des actions lors de son clic.
- `android:title="Se déconnecter"` : Le titre affiché dans le menu est "Se déconnecter". Ce texte sera visible dans le menu d'options lorsque l'utilisateur interagira avec ce menu.
- `android:icon="@android:drawable/ic_menu_close_clear_cancel"` : L'icône associée à cet élément est une icône représentant l'action "annuler" ou "fermer", ce qui est pertinent pour une action de déconnexion.

7.6.3.3 Fonctionnement du Menu

Ce menu est généralement utilisé pour afficher des actions accessibles par l'utilisateur à partir de la barre d'outils (`Toolbar`) ou via un bouton de menu dans l'application. Les deux éléments de menu, "Se connecter" et "Se déconnecter", sont des actions qui pourraient être utilisées pour permettre à un utilisateur de se connecter ou se déconnecter d'une session.

- Lorsqu'un utilisateur clique sur l'élément "Se connecter", il pourrait être dirigé vers une interface de connexion.
- Lorsque l'utilisateur clique sur "Se déconnecter", il pourrait être redirigé vers une page de déconnexion ou la session de l'utilisateur pourrait être fermée.

```
1 <menu xmlns:android="http://schemas.android.com/apk/res/android">
2     <item
3         android:id="@+id/action_connect"
4         android:title="Se connecter"
5         android:icon="@android:drawable/ic_menu_add" />
6     <item
7         android:id="@+id/action_disconnect"
8         android:title="Se déconnecter"
9         android:icon="@android:drawable/ic_menu_close_clear_cancel" />
10    </menu>
```

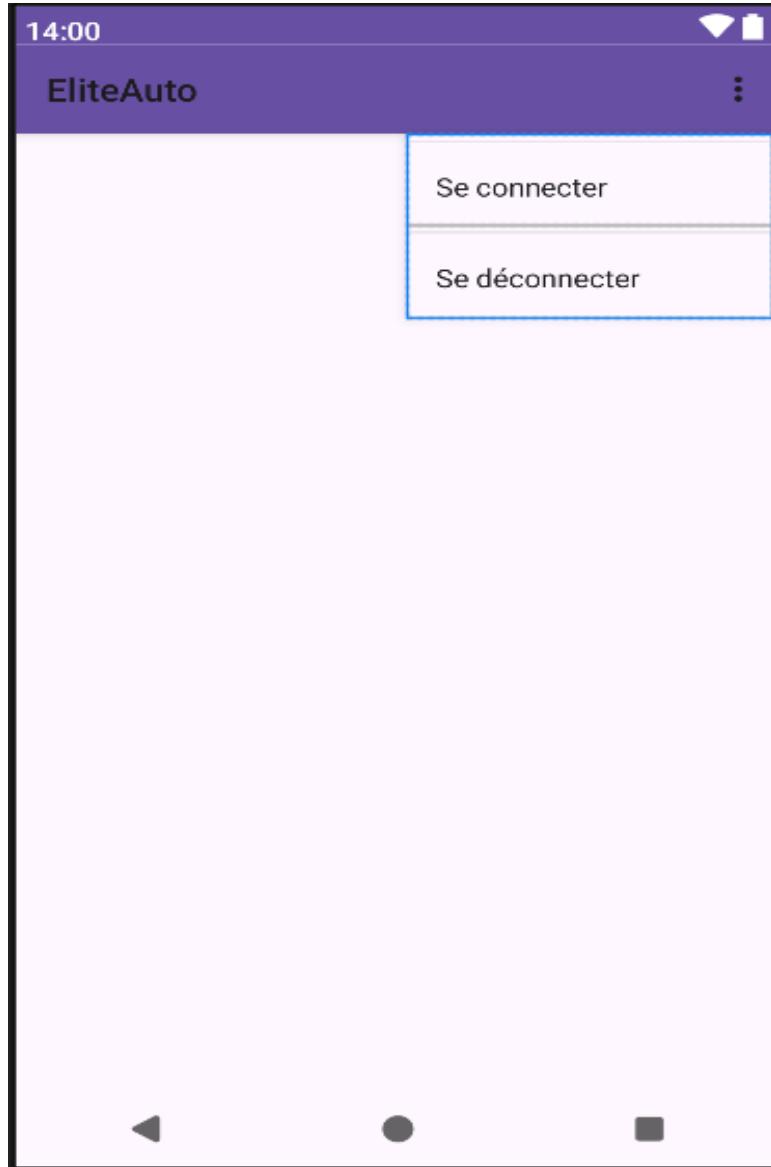


FIGURE 7.3 : Layout de menu

7.6.4 detailvoiture.xml

Le fichier XML utilisé pour la mise en page de la section des détails est basé sur un `CoordinatorLayout`, qui contient un `AppBarLayout` avec un `CollapsingToolbarLayout`. Cela permet d'afficher une image de la voiture dans un style interactif qui peut se réduire lorsque l'utilisateur fait défiler la page. Le layout inclut également un `NestedScrollView` contenant un `LinearLayout`, ce qui permet de faire défiler le contenu verticalement et d'afficher les informations de manière fluide et dynamique.

Principales Parties du Layout :

- **AppBarLayout & CollapsingToolbarLayout** : Contient l'image de la voiture et un toolbar. L'image est définie avec un effet de parallax, ce qui crée une interaction visuelle agréable lors du défilement de la page.

- **TextViews** : Divers **TextView** sont utilisés pour afficher les informations du véhicule, telles que le modèle, le prix, l'année, le type de carburant, etc.
- **LinearLayouts** : Plusieurs **LinearLayout** horizontaux sont utilisés pour aligner les informations relatives aux caractéristiques du véhicule.
- **MaterialButton** : Un bouton permettant à l'utilisateur de réserver la voiture, avec une icône représentant la réservation.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.coordinatorlayout.widget.CoordinatorLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent">
7
8      <com.google.android.material.appbar.AppBarLayout
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content">
11
12         <com.google.android.material.appbar.CollapsingToolbarLayout
13             android:layout_width="match_parent"
14             android:layout_height="256dp"
15             app:layout_scrollFlags="scroll|exitUntilCollapsed"
16             app:contentScrim="?attr/colorPrimary"
17             app:expandedTitleMarginStart="16dp"
18             app:expandedTitleMarginBottom="16dp">
19
20             <ImageView
21                 android:id="@+id/imageVoiture"
22                 android:layout_width="match_parent"
23                 android:layout_height="match_parent"
24                 android:scaleType="centerCrop"
25                 android:transitionName="imageVoiture"
26                 app:layout_collapseMode="parallax"/>
27
28             <androidx.appcompat.widget.Toolbar
29                 android:id="@+id/toolbar"
30                 android:layout_width="match_parent"
```

```
31         android:layout_height="?attr/actionBarSize"
32         app:layout_collapseMode="pin"/>
33
34     </com.google.android.material.appbar.CollapsingToolbarLayout>
35   </com.google.android.material.appbar.AppBarLayout>
36
37   <androidx.core.widget.NestedScrollView
38       android:layout_width="match_parent"
39       android:layout_height="match_parent"
40       app:layout_behavior="@string/appbar_scrolling_view_behavior">
41
42     <LinearLayout
43         android:layout_width="match_parent"
44         android:layout_height="wrap_content"
45         android:orientation="vertical"
46         android:padding="16dp">
47
48       <TextView
49           android:id="@+id/textMarqueModele"
50           android:layout_width="match_parent"
51           android:layout_height="wrap_content"
52           android:textSize="24sp"
53           android:textStyle="bold"/>
54
55       <TextView
56           android:id="@+id/textPrix"
57           android:layout_width="match_parent"
58           android:layout_height="wrap_content"
59           android:textSize="20sp"
60           android:textColor="@android:color/holo_green_dark"
61           android:layout_marginTop="8dp"/>
62
63       <com.google.android.material.divider.MaterialDivider
64           android:layout_width="match_parent"
65           android:layout_height="1dp"
66           android:layout_marginTop="16dp"
67           android:layout_marginBottom="16dp"/>
```

```
68
69    <LinearLayout
70        android:layout_width="match_parent"
71        android:layout_height="wrap_content"
72        android:orientation="horizontal">
73
74        <ImageView
75            android:layout_width="24dp"
76            android:layout_height="24dp"
77            android:src="@drawable/ic_calendar"/>
78
79        <TextView
80            android:id="@+id/textAnnee"
81            android:layout_width="0dp"
82            android:layout_height="wrap_content"
83            android:layout_weight="1"
84            android:textSize="16sp"
85            android:layout_marginStart="8dp"/>
86
87        <ImageView
88            android:layout_width="24dp"
89            android:layout_height="24dp"
90            android:src="@drawable/ic_door"/>
91
92        <TextView
93            android:id="@+id/textNombrePortes"
94            android:layout_width="0dp"
95            android:layout_height="wrap_content"
96            android:layout_weight="1"
97            android:textSize="16sp"
98            android:layout_marginStart="8dp"/>
99    </LinearLayout>
100
101    <LinearLayout
102        android:layout_width="match_parent"
103        android:layout_height="wrap_content"
104        android:orientation="horizontal"
```

```
105     android:layout_marginTop="16dp">  
106  
107     <ImageView  
108         android:layout_width="24dp"  
109         android:layout_height="24dp"  
110         android:src="@drawable/ic_fuel"/>  
111  
112     <TextView  
113         android:id="@+id/textTypeCarburant"  
114         android:layout_width="0dp"  
115         android:layout_height="wrap_content"  
116         android:layout_weight="1"  
117         android:textSize="16sp"  
118         android:layout_marginStart="8dp"/>  
119  
120     <ImageView  
121         android:layout_width="24dp"  
122         android:layout_height="24dp"  
123         android:src="@drawable/ic_euro"/>  
124  
125     <TextView  
126         android:id="@+id/textMontantCaution"  
127         android:layout_width="0dp"  
128         android:layout_height="wrap_content"  
129         android:layout_weight="1"  
130         android:textSize="16sp"  
131         android:layout_marginStart="8dp"/>  
132     </LinearLayout>  
133  
134     <LinearLayout  
135         android:layout_width="match_parent"  
136         android:layout_height="wrap_content"  
137         android:orientation="horizontal"  
138         android:layout_marginTop="16dp">  
139  
140     <ImageView  
141         android:layout_width="24dp"
```

```
142         android:layout_height="24dp"
143     />
144
145     <TextView
146         android:id="@+id/textNombreChevaux"
147         android:layout_width="0dp"
148         android:layout_height="wrap_content"
149         android:layout_weight="1"
150         android:textSize="16sp"
151         android:layout_marginStart="8dp"/>
152
153     <ImageView
154         android:layout_width="24dp"
155         android:layout_height="24dp"
156         android:src="@drawable/ic_person"/>
157
158     <TextView
159         android:id="@+id/textNombrePassagers"
160         android:layout_width="0dp"
161         android:layout_height="wrap_content"
162         android:layout_weight="1"
163         android:textSize="16sp"
164         android:layout_marginStart="8dp"/>
165     </LinearLayout>
166
167     <com.google.android.material.button.MaterialButton
168         android:id="@+id/buttonReserver"
169         android:layout_width="match_parent"
170         android:layout_height="wrap_content"
171         android:text="Réserver"
172         android:layout_marginTop="24dp"
173         app:icon="@drawable/ic_calendar_check"
174         app:iconGravity="textStart"/>
175
176     </LinearLayout>
177 </androidx.core.widget.NestedScrollView>
178 </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

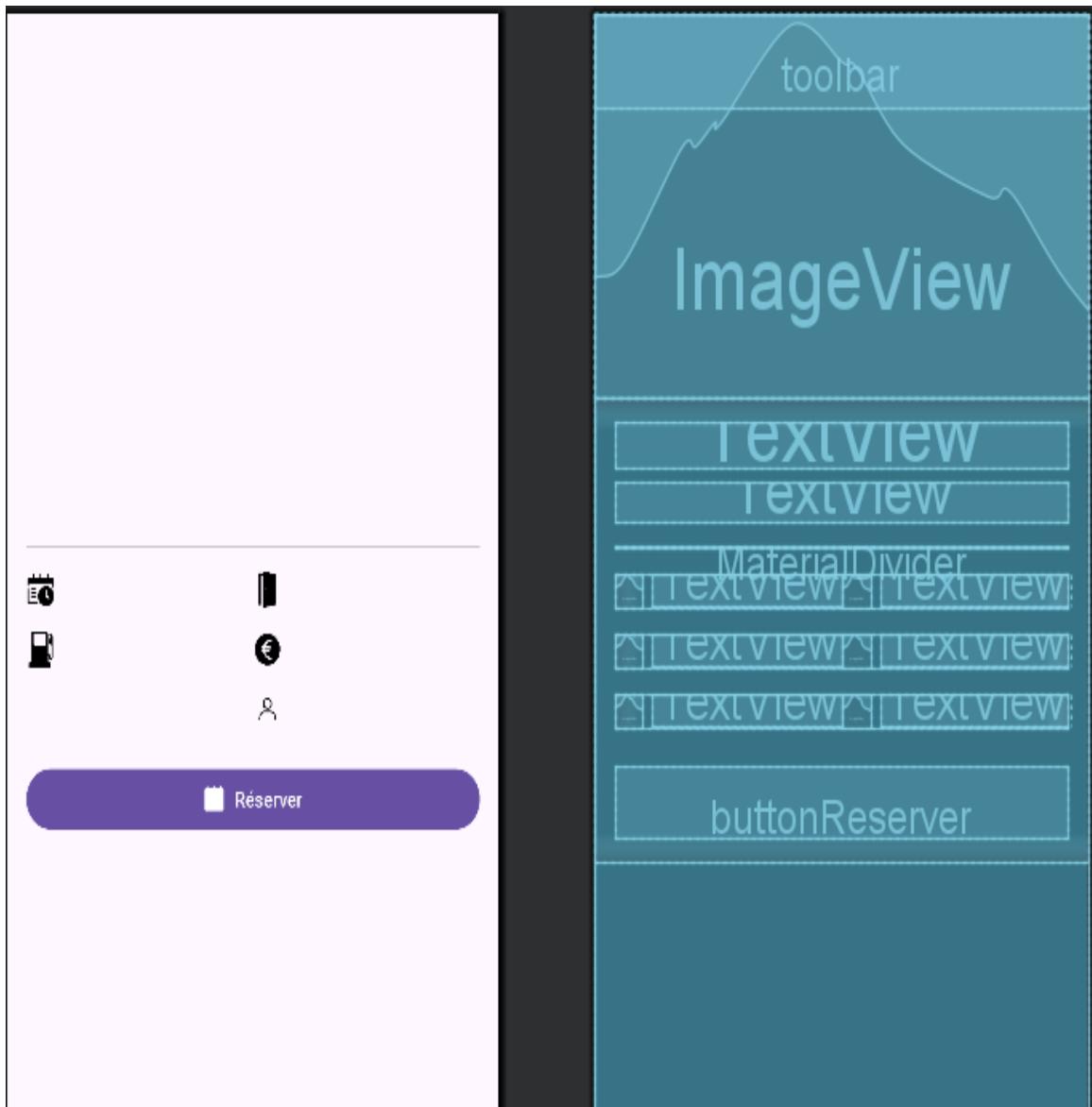


FIGURE 7.4 : Layout de l'activité detail voiture

7.7 Vue d'ensemble du fichier MainActivity.java

7.7.1 Usage des intents

Ligne 64 :Intent pour démarrer un service

- Type d'Intent : Explicit Intent
- Utilisation :Démarrage du service DisponibiliteService

Ligne 76 :Intent pour ouvrir l'activité des détails de la voiture

- Type d'Intent : Intent Explicite avec transfert des données

- Données transférées :Les données sont transférées via la méthode **putExtra()** :imageUrl, marqueModele, annee, prix, nombrePortes, etc.

Ces données seront accessibles dans l'activité cible (DetailsActivity) via **getIntent().getStringExtra()**

Ligne 118 : Intent pour ouvrir l'activité de connexion

- Type d'Intent : Explicit Intent
- Utilisation :Naviguer vers l'activité LoginActivity pour permettre à l'utilisateur de se connecter.

7.7.2 RecyclerView Setup

Dans le code de **MainActivity**, le **RecyclerView** est initialisé et configuré pour afficher une liste de voitures. L'adaptateur **VoitureAdapter** est utilisé pour lier les données des voitures à la vue. Cela permet d'afficher efficacement une liste d'éléments avec une gestion de mémoire optimisée pour des ensembles de données plus grands.

7.7.3 SwipeRefreshLayout

Le **SwipeRefreshLayout** est configuré pour permettre à l'utilisateur de rafraîchir la liste des voitures en glissant vers le bas. Lors du rafraîchissement, une nouvelle liste de voitures est chargée depuis le serveur ou la base de données. Ce mécanisme est couramment utilisé pour mettre à jour dynamiquement l'affichage des données sans quitter l'écran actuel.

7.7.4 VoitureAdapter

L'adaptateur **VoitureAdapter** est responsable de l'affichage de chaque voiture dans un élément de la liste. L'adaptateur récupère les données et les affiche dans les vues correspondantes du fichier **item_voiture.xml**. Chaque voiture a :

- Un **ImageView** pour son image,
- Un **TextView** pour le nom et le prix de la voiture,
- Un autre **TextView** pour afficher ses disponibilités formatées.

7.7.5 Glide pour le chargement des images

Glide est utilisé pour charger et afficher les images des voitures de manière asynchrone et optimisée. L'URL de l'image est récupérée via la méthode **getImageUrl()** de la classe **Voiture** et est ensuite chargée dans le **ImageView**. Cela permet un chargement rapide et fluide des images, même lorsqu'elles proviennent de sources distantes.

7.7.6 OnItemClickListener

L’interface `OnItemClickListener` est implémentée dans `MainActivity` pour permettre de gérer les interactions de l’utilisateur avec les éléments de la liste. Cela permet de définir une action lorsqu’un utilisateur clique sur une voiture, qui est dans notre cas le lancement d’une nouvelle activité pour afficher les détails complets de la voiture sélectionnée.

```
1
2 package com.example.eliteauto.ui;
3
4 import android.content.Intent;
5 import android.os.Bundle;
6 import android.util.Log;
7 import android.view.Menu;
8 import android.view.MenuItem;
9 import android.view.View;
10 import android.widget.Toast;
11
12 import androidx.appcompat.app.AppCompatActivity;
13 import androidx.core.app.ActivityOptionsCompat;
14 import androidx.recyclerview.widget.LinearLayoutManager;
15 import androidx.recyclerview.widget.RecyclerView;
16 import androidx.swiperefreshlayout.widget.SwipeRefreshLayout;
17
18 import com.example.eliteauto.R;
19 import com.example.eliteauto.adapter.VoitureAdapter;
20 import com.example.eliteauto.model.Voiture;
21 import com.example.eliteauto.network.ApiClient;
22 import com.example.eliteauto.network.ApiService;
23 import com.example.eliteauto.services.DisponibiliteService;
24 import com.google.android.material.floatingactionbutton.FloatingActionButton;
25
26 import java.util.List;
27
28 import retrofit2.Call;
29 import retrofit2.Callback;
30 import retrofit2.Response;
```

```
32 public class MainActivity extends AppCompatActivity {  
33  
34     private RecyclerView recyclerView;  
35     private VoitureAdapter voitureAdapter;  
36     private ApiService apiService;  
37     private FloatingActionButton fabFilter;  
38     private SwipeRefreshLayout swipeRefreshLayout;  
39  
40     @Override  
41     protected void onCreate(Bundle savedInstanceState) {  
42         super.onCreate(savedInstanceState);  
43         setContentView(R.layout.activity_main);  
44         startDisponibiliteService();  
45         setSupportActionBar(findViewById(R.id.toolbar));  
46  
47         recyclerView = findViewById(R.id.recyclerView);  
48         recyclerView.setLayoutManager(new LinearLayoutManager(this));  
49  
50         swipeRefreshLayout = findViewById(R.id.swipeRefreshLayout);  
51         swipeRefreshLayout.setOnRefreshListener(this::fetchVoitures);  
52  
53         fabFilter = findViewById(R.id.fabFilter);  
54         fabFilter.setOnClickListener(v -> {  
55             // TODO: Implémenter la logique de filtrage  
56             Toast.makeText(MainActivity.this, "Filtrage à implémenter",  
57             → Toast.LENGTH_SHORT).show();  
58         });  
59  
60         apiService = ApiClient.getClient().create(ApiService.class);  
61  
62         fetchVoitures();  
63     }  
64     private void startDisponibiliteService() {  
65         Intent serviceIntent = new Intent(this, DisponibiliteService.class);  
66         startService(serviceIntent); // Démarre le service en arrière-plan  
         Toast.makeText(this, "Service de disponibilité démarré",  
         → Toast.LENGTH_SHORT).show();
```

```
67 }
68
69     private void fetchVoitures() {
70
71         swipeRefreshLayout.setRefreshing(true);
72
73         apiService.getAllVoitures().enqueue(new Callback<List<Voiture>>() {
74
75             @Override
76
77             public void onResponse(Call<List<Voiture>> call, Response<List<Voiture>>
78
79                 response) {
80
81
82                 swipeRefreshLayout.setRefreshing(false);
83
84                 if (response.isSuccessful() && response.body() != null) {
85
86                     voitureAdapter = new VoitureAdapter(response.body(), voiture -> {
87
88                         Intent intent = new Intent(MainActivity.this,
89
90                             DetailsActivity.class);
91
92                         intent.putExtra("imageUrl", voiture.getImageUrl());
93
94                         intent.putExtra("marqueModele", voiture.getMarque() + " " +
95
96                             voiture.getModele());
97
98                         intent.putExtra("annee", String.valueOf(voiture.getAnnee()));
99
100                        intent.putExtra("prix", voiture.getPrixParJour().toString());
101
102                        intent.putExtra("nombrePortes",
103
104                            String.valueOf(voiture.getNombrePortes()));
105
106                        intent.putExtra("typeCarburant", voiture.getTypeCarburant());
107
108                        intent.putExtra("montantCaution",
109
110                            voiture.getMontantCaution().toString());
111
112                        intent.putExtra("nombreChevaux",
113
114                            String.valueOf(voiture.getNombreChevaux()));
115
116                        intent.putExtra("nombrePassagers",
117
118                            String.valueOf(voiture.getNombrePassagers()));
119
120
121                         ActivityOptionsCompat options = ActivityOptionsCompat.
122
123                             makeSceneTransitionAnimation(MainActivity.this,
124
125                                 findViewById(R.id.imageVoiture),
126
127                                 "imageVoiture");
128
129                         startActivity(intent, options.toBundle());
130
131                     });
132
133                     recyclerView.setAdapter(voitureAdapter);
134
135                 } else {
136
137                     Log.e("MainActivity", "Erreur : " + response.code());
138
139                     Toast.makeText(MainActivity.this, "Erreur lors du chargement des
140
141                         voitures", Toast.LENGTH_SHORT).show();
142
143                 }
144
145             }
146
147         });
148
149     }
150
151 }
```

```
97         }
98     }
99
100    @Override
101   public void onFailure(Call<List<Voiture>> call, Throwable t) {
102       swipeRefreshLayout.setRefreshing(false);
103       Log.e("MainActivity", "Erreur lors de la récupération des voitures", t);
104       Toast.makeText(MainActivity.this, "Erreur réseau",
105                     Toast.LENGTH_SHORT).show();
106   }
107 }
108
109 @Override
110 public boolean onCreateOptionsMenu(Menu menu) {
111     getMenuInflater().inflate(R.menu.main_menu, menu);
112     return true;
113 }
114
115 @Override
116 public boolean onOptionsItemSelected(MenuItem item) {
117     if (item.getItemId() == R.id.login_button) {
118         Intent loginIntent = new Intent(this, LoginActivity.class);
119         startActivity(loginIntent);
120         return true;
121     } else if (item.getItemId() == R.id.action_disconnect) {
122         Toast.makeText(this, "Se déconnecter", Toast.LENGTH_SHORT).show();
123         return true;
124     }
125     return super.onOptionsItemSelected(item);
126 }
127 }
```

7.8 La Classe DetailsActivity

La classe `DetailsActivity` gère la logique de cette section de l'application. Elle hérite de `AppCompatActivity` et est responsable de la configuration des éléments UI et du traitement des données qui y sont affichées. Les points clés du code Java sont les suivants :

7.8.1 Initialisation de la Vue

Dans la méthode `onCreate()`, tous les composants de l'interface utilisateur sont liés à leurs éléments XML correspondants à l'aide de `findViewById()`.

7.8.2 Récupération des Données depuis l'Intent

Les informations du véhicule sont passées à l'activité via l'Intent. Elles sont récupérées en utilisant la méthode `getIntent().getStringExtra()` pour chaque attribut du véhicule (image, marque, modèle, prix, etc.).

7.8.3 Affichage de l'Image et des Informations

L'image de la voiture est chargée à l'aide de la bibliothèque `Glide`, ce qui permet un chargement efficace et rapide des images. Les autres informations sont simplement affichées dans leurs `TextView` respectifs.

7.8.4 Interaction avec le Bouton de Réservation

Un `OnClickListener` est attaché au bouton de réservation. Lorsqu'il est cliqué, une alerte (`Toast`) est affichée indiquant que la logique de réservation n'a pas encore été implémentée.

```
1 package com.example.eliteauto.ui;  
2  
3 import android.os.Bundle;  
4 import android.view.MenuItem;  
5 import android.view.View;  
6 import android.widget.ImageView;  
7 import android.widget.TextView;  
8 import android.widget.Toast;  
9  
10 import androidx.appcompat.app.AppCompatActivity;  
11 import androidx.appcompat.widget.Toolbar;
```

```
13 import com.bumptech.glide.Glide;
14 import com.example.eliteauto.R;
15 import com.google.android.material.button.MaterialButton;
16
17 import java.util.ArrayList;
18
19 public class DetailsActivity extends AppCompatActivity {
20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_details);
25
26         Toolbar toolbar = findViewById(R.id.toolbar);
27         setSupportActionBar(toolbar);
28         getSupportActionBar().setDisplayHomeAsUpEnabled(true);
29         getSupportActionBar().setDisplayShowTitleEnabled(false);
30
31         ImageView imageVoiture = findViewById(R.id.imageVoiture);
32         TextView texteMarqueModele = findViewById(R.id.textMarqueModele);
33         TextView textePrix = findViewById(R.id.textPrix);
34         TextView texteAnnee = findViewById(R.id.textAnnee);
35         TextView texteNombrePortes = findViewById(R.id.textNombrePortes);
36         TextView texteTypeCarburant = findViewById(R.id.textTypeCarburant);
37         TextView texteMontantCaution = findViewById(R.id.textMontantCaution);
38         TextView texteNombreChevaux = findViewById(R.id.textNombreChevaux);
39         TextView texteNombrePassagers = findViewById(R.id.textNombrePassagers);
40         MaterialButton buttonReserver = findViewById(R.id.buttonReserver);
41
42         // Récupérer les données passées par l'intent
43         String imageUrl = getIntent().getStringExtra("imageUrl");
44         String marqueModele = getIntent().getStringExtra("marqueModele");
45         String annee = getIntent().getStringExtra("annee");
46         String prix = getIntent().getStringExtra("prix");
47         String nombrePortes = getIntent().getStringExtra("nombrePortes");
48         String typeCarburant = getIntent().getStringExtra("typeCarburant");
49         String montantCaution = getIntent().getStringExtra("montantCaution");
```

```
50     String nombreChevaux = getIntent().getStringExtra("nombreChevaux");
51     String nombrePassagers = getIntent().getStringExtra("nombrePassagers");
52
53     // Afficher les données dans les vues
54     Glide.with(this).load(imageUrl).into(imageVoiture);
55     textMarqueModele.setText(marqueModele);
56     textPrix.setText(prix + " €/jour");
57     textAnnee.setText(annee);
58     textNombrePortes.setText(nombrePortes + " portes");
59     textTypeCarburant.setText(typeCarburant);
60     textMontantCaution.setText("Caution : " + montantCaution + " €");
61     textNombreChevaux.setText(nombreChevaux + " ch");
62     textNombrePassagers.setText(nombrePassagers + " passagers");
63
64     buttonReserver.setOnClickListener(new View.OnClickListener() {
65         @Override
66         public void onClick(View v) {
67             // TODO: Implémenter la logique de réservation
68             Toast.makeText(DetailsActivity.this, "Réservation à implémenter",
69             →   Toast.LENGTH_SHORT).show();
70         }
71     });
72
73     @Override
74     public boolean onOptionsItemSelected(MenuItem item) {
75         if (item.getItemId() == android.R.id.home) {
76             onBackPressed();
77             return true;
78         }
79         return super.onOptionsItemSelected(item);
80     }
81 }
```

7.9 Package com.example.eliteauto.Service.DisponibiliteService

Le service `DisponibiliteService` est un service en arrière-plan dans l'application Android, qui permet de vérifier périodiquement la disponibilité des voitures. Ce service est responsable de la récupération des informations concernant les véhicules et de leur disponibilité via une API, tout en maintenant une notification pour informer l'utilisateur que le service fonctionne en arrière-plan.

7.9.1 Principaux Composants et Fonctionnalités

1. Initialisation du Service

Lors de la création du service (`onCreate()`), le service se connecte à un client API (`ApiClient`) pour effectuer des requêtes réseau. Un `NotificationCompat.Builder` est utilisé pour créer une notification indiquant que le service est en cours d'exécution. Cette notification est affichée en avant-plan pour que l'utilisateur puisse voir que le service est actif. Ensuite, un `ScheduledExecutorService` est initialisé pour exécuter la vérification de disponibilité des voitures à intervalles réguliers.

2. Vérification de la Disponibilité des Voitures

La méthode `checkDisponibilite()` est appelée périodiquement toutes les 10 secondes par le `ScheduledExecutorService`. Elle effectue une requête API en utilisant l'interface `ApiService` pour obtenir la liste de toutes les voitures via `getAllVoitures()`. Une fois la réponse reçue, chaque voiture est vérifiée et son statut de disponibilité est consigné dans les logs via `Log.d()`.

3. Gestion des Erreurs

En cas de défaillance du réseau ou d'autres erreurs lors de l'appel API, un message d'erreur est consigné dans les logs avec `Log.e()`, ce qui permet de suivre les problèmes potentiels.

4. Gestion du Cycle de Vie du Service

Le service utilise la méthode `onDestroy()` pour s'assurer que toutes les tâches périodiques sont annulées et que les ressources sont correctement libérées lorsque le service est arrêté. Le `ScheduledExecutorService` est arrêté et le service est détruit proprement.

```
1 package com.example.eliteauto.services;  
2  
3 import android.app.Notification;  
4 import android.app.Service;  
5 import android.content.Intent;
```

```
6 import android.os.IBinder;
7 import android.util.Log;
8
9 import androidx.annotation.Nullable;
10 import androidx.core.app.NotificationCompat;
11 import androidx.core.app.NotificationManagerCompat;
12
13 import com.example.eliteauto.R;
14 import com.example.eliteauto.network.ApiClient;
15 import com.example.eliteauto.network.ApiService;
16 import com.example.eliteauto.model.Voiture;
17
18 import java.util.List;
19 import java.util.concurrent.Executors;
20 import java.util.concurrent.ScheduledExecutorService;
21 import java.util.concurrent.TimeUnit;
22
23 import retrofit2.Call;
24 import retrofit2.Callback;
25 import retrofit2.Response;
26
27 public class DisponibiliteService extends Service {
28
29     private ApiService apiService;
30     private ScheduledExecutorService scheduler;
31
32     @Override
33     public void onCreate() {
34         super.onCreate();
35         apiService = ApiClient.getClient().create(ApiService.class);
36
37         // Create a notification to show that the service is running
38         NotificationCompat.Builder builder = new NotificationCompat.Builder(this,
39             "default")
40             .setContentTitle("Disponibilité des voitures")
41             .setContentText("Service en cours de vérification...")
```

```
42         .setPriority(NotificationCompat.PRIORITY_DEFAULT);
43
44     Notification notification = builder.build();
45
46     // Start the service in the foreground
47     startForeground(1, notification);
48
49     // Schedule the periodic task to check availability
50     scheduler = Executors.newSingleThreadScheduledExecutor();
51     scheduler.scheduleAtFixedRate(this::checkDisponibilite, 0, 10, TimeUnit.SECONDS);
52 }
53
54 private void checkDisponibilite() {
55     apiService.getAllVoitures().enqueue(new Callback<List<Voiture>>() {
56
57         @Override
58         public void onResponse(Call<List<Voiture>> call, Response<List<Voiture>>
59             → response) {
60
61             if (response.isSuccessful() && response.body() != null) {
62
63                 // Processing the response: Log the availability of each car
64                 for (Voiture voiture : response.body()) {
65
66                     Log.d("DisponibiliteService", "Voiture " + voiture.getMarque() + "
67                         → " + voiture.getModele() + " disponible.");
68
69                 }
70             }
71         }
72
73         @Override
74         public void onFailure(Call<List<Voiture>> call, Throwable t) {
75
76             Log.e("DisponibiliteService", "Erreur lors de la vérification de la
77                 → disponibilité des voitures", t);
78
79         }
80     });
81 }
82
83 @Nullable
84 @Override
85 public IBinder onBind(Intent intent) {
```

```
76     return null;
77 }
78
79 @Override
80 public void onDestroy() {
81     super.onDestroy();
82     // Stop the scheduler and service when it's no longer needed
83     if (scheduler != null && !scheduler.isShutdown()) {
84         scheduler.shutdownNow();
85     }
86     stopForeground(true);
87     stopSelf();
88 }
89 }
```

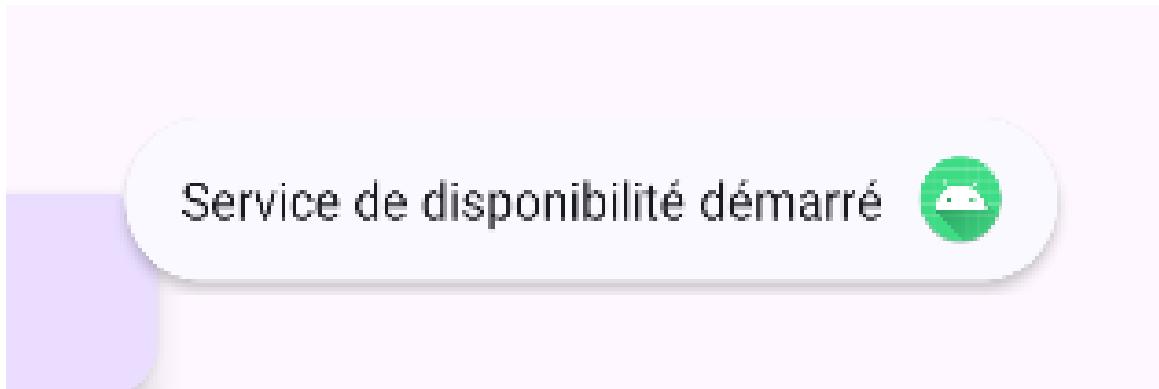


FIGURE 7.5 : demarrage de service en arriere plan

Les interfaces utilisateurs

1. Liste des Voitures Disponibles

Voici une capture d'écran montrant l'interface où les voitures disponibles sont affichées :

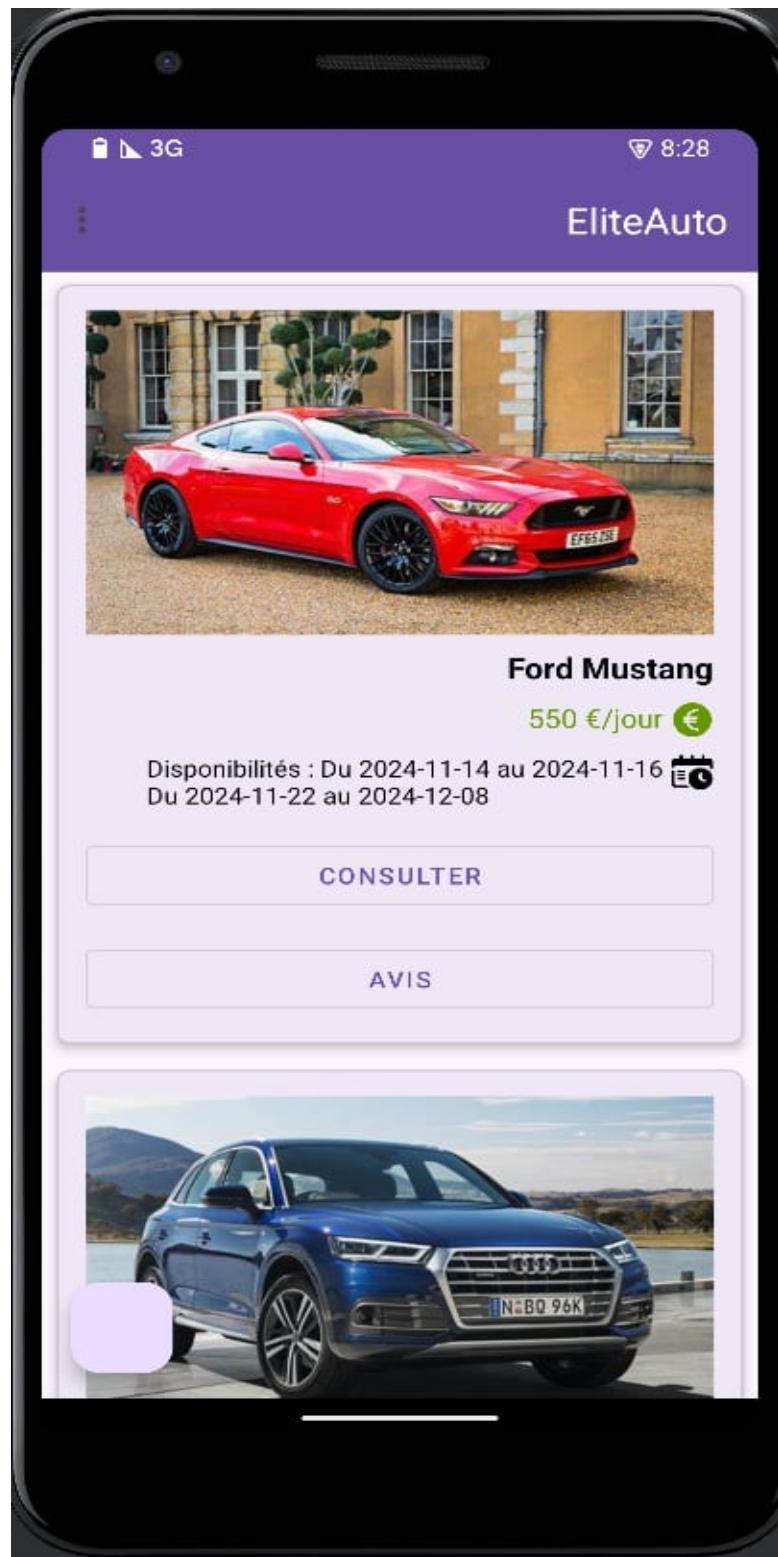


FIGURE 7.6 : liste des voitures disponibles

2. Détails d'une Voiture

Lorsqu'un utilisateur clique sur le bouton **Consulter**, il est dirigé vers une nouvelle interface affichant les détails de la voiture sélectionnée. Voici une capture d'écran de cette interface :

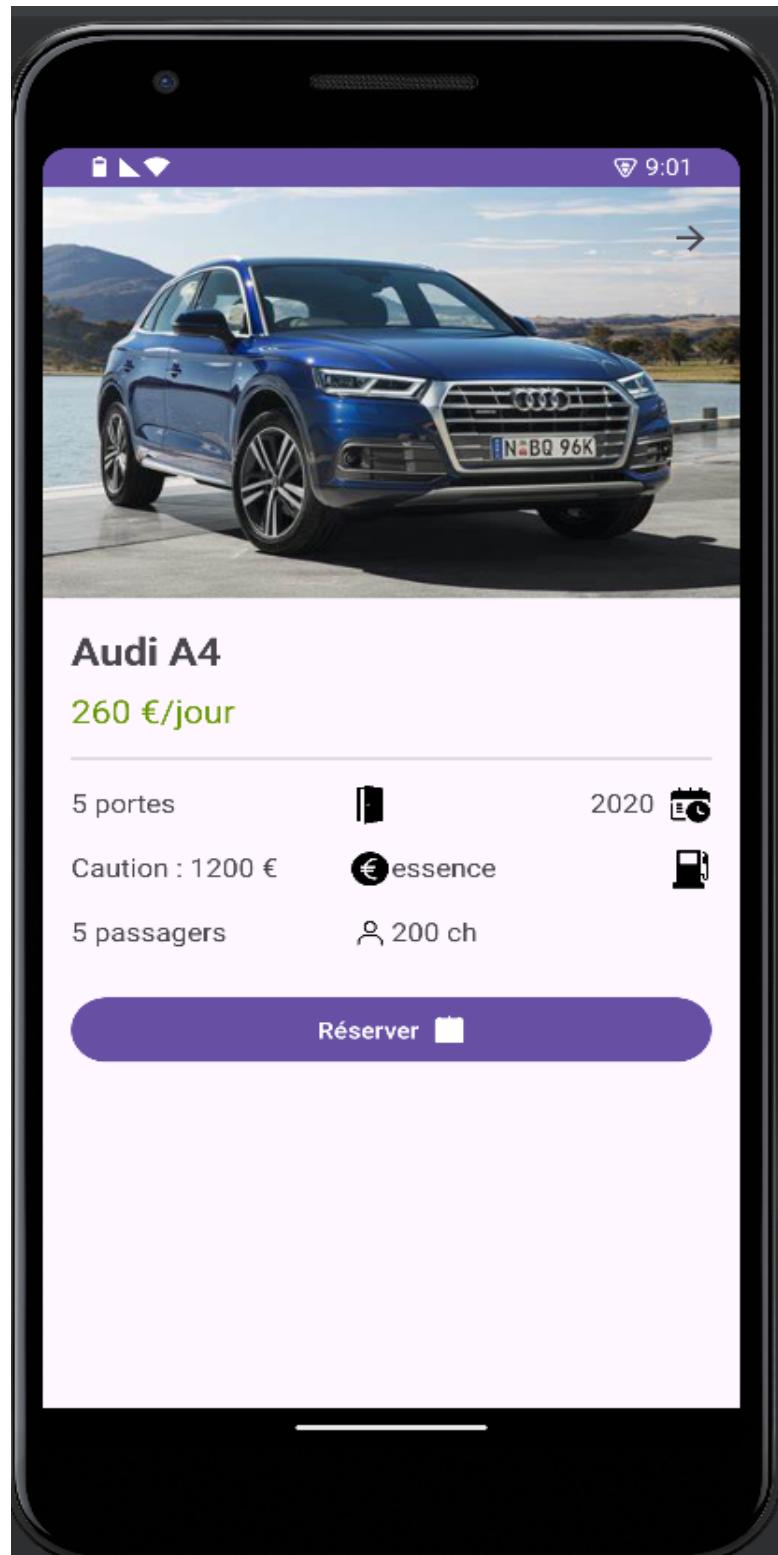


FIGURE 7.7 : details de voiture

3.Avis faits sur un propriétaire

Lorsqu'un utilisateur est dans la pae des listes des voitures et il clique sur le bouton **Avis**, il est dirigé vers une nouvelle interface affichant la liste des avis faits sur le propriétaire de la voiture selectionnée Voici une capture d'écran de cette interface :

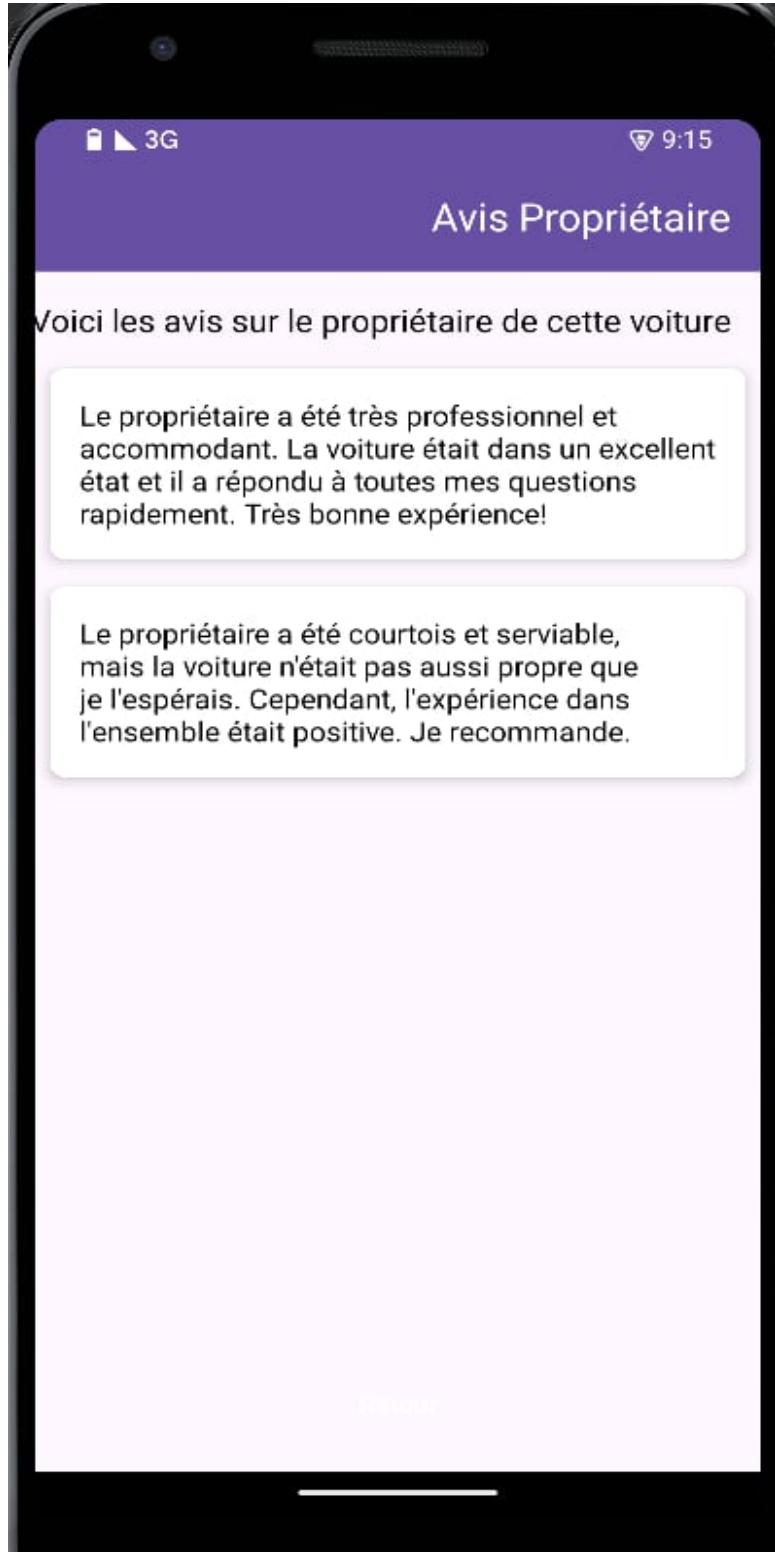


FIGURE 7.8 : Liste des avis faits sur un propriétaire

4.Menu

Lorsqu'un utilisateur clique sur le menu il peut choisir de se connecter si'il ne l'est pas encore ou de se déconnecter. Voici une capture d'écran de ce menu qui persistait dans toutes les activités de l'application :



FIGURE 7.9 : menu

5.Connexion du client

Lorsqu'un utilisateur clique sur le menu et il choisit l'option se connecter il sera amené vers l'interface ci-dessous ;

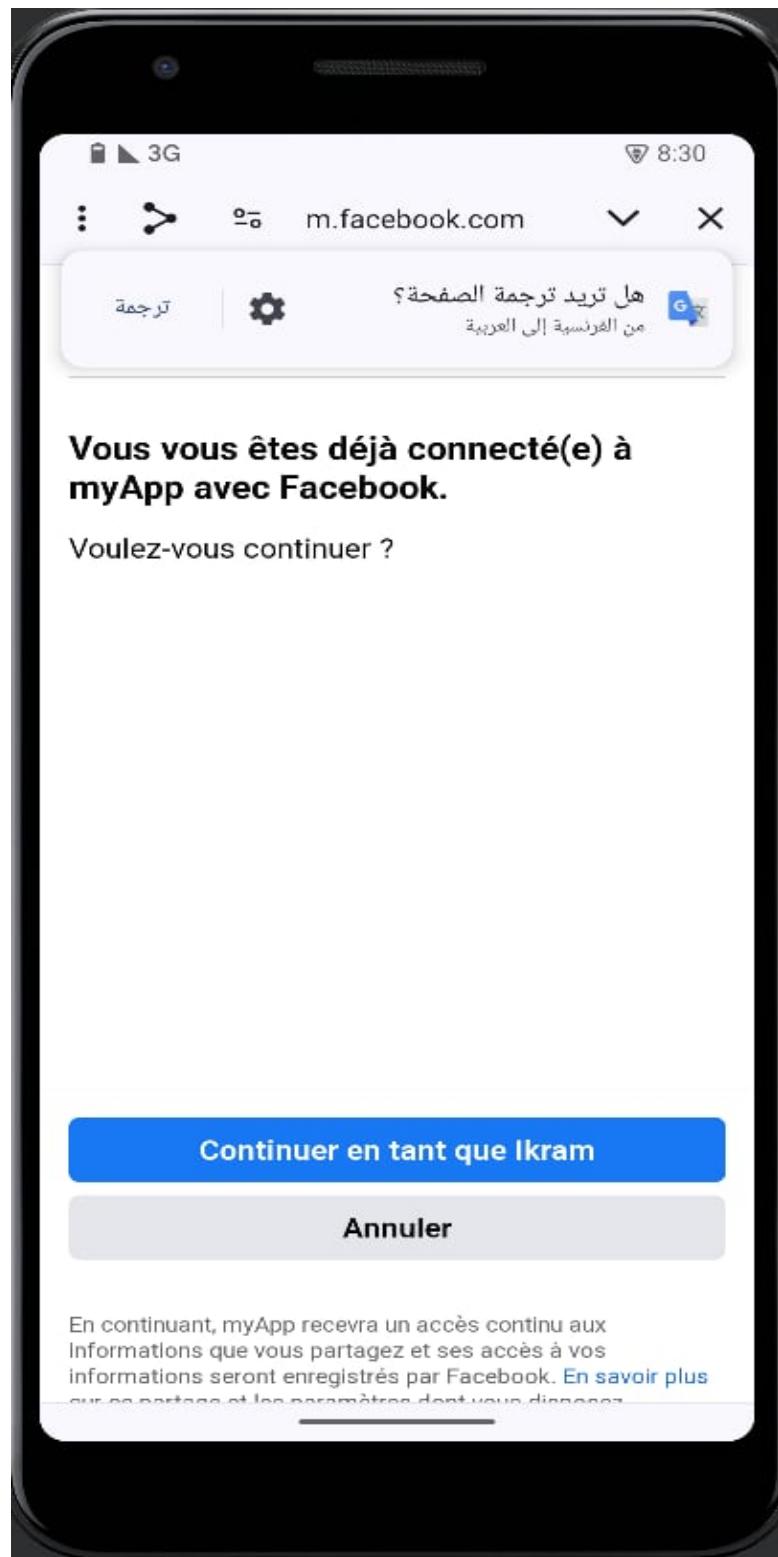


FIGURE 7.10 : Connexion du client

6. Dashboard du client

Un fois connecté le client va se trouver devant cette interface ;

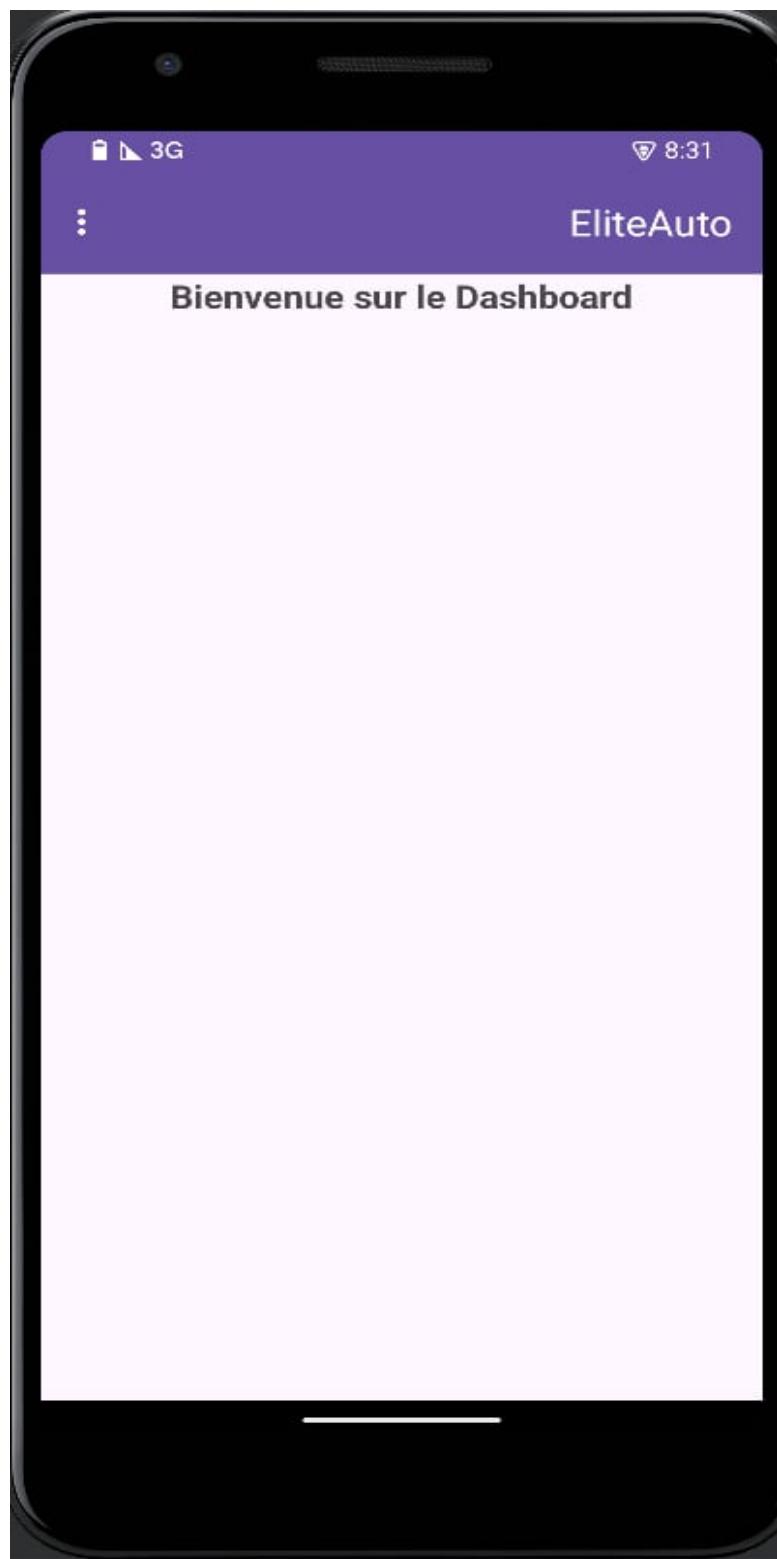


FIGURE 7.11 : Dashbord du client

7.10 Les tests faits

7.10.1 introduction

Pour tester la partie mobile de notre projet, voici une suite de tests structurée avec des scénarios pour plusieurs types de test ; test unitaire, intégration des composants, intégration système

et acceptation utilisateur.

1. Test unitaire

Objectif : Vérifier la conversion des disponibilités au format LocalDate dans la classe Disponibilite.

Objet de test : La méthode getDateDebutDisponibilite() de la classe Disponibilite. getDateDebutDisponibilite() la classe Disponibilite.

Propriété à tester : La méthode doit convertir correctement une date au format String (e.g., "2024-12-14") en un objet LocalDate.

Données de test : dateDebutDisponibilite : "2024-11-14"

Étapes du test :

- 1) Instancier un objet Disponibilite avec dateDebutDisponibilite = "2024-12-14".
- 2) Appeler la méthode getDateDebutDisponibilite().
- 3) Vérifier que le résultat est un objet LocalDate avec la valeur 2024-12-14.

Résultat attendu : La méthode retourne un objet LocalDate avec la valeur exacte.

2. Test d'Intégration Composants

Objectif : Vérifier que l'adaptateur VoitureAdapter lie correctement les données d'une liste de voitures à l'interface utilisateur.

Objet de test : Classe VoitureAdapter. **Propriété à tester :** Les données (marque, modèle, prix, disponibilités,nombres de porte...) sont affichées correctement dans la vue.

Données de test : Liste de voitures avec : marque = "Audi" modèle = "A4" prixParJour = 260
Disponibilités : Du 2024-11-14 au 2024-11-16 Du 2024-11-22 au 2024-12-08

Étapes du test :

- 1) Créer une liste de voitures avec les données de test.
- 2) Injecter cette liste dans une instance de VoitureAdapter.
- 3) Charger l'adaptateur dans une vue de liste (RecyclerView).
- 4) Vérifier manuellement que les données s'affichent correctement dans chaque élément de la liste.

Résultat attendu : Les champs marque, modèle, prix, et disponibilités,nombres de porte... s'affichent comme prévu.

3. Test d'Intégration Système

Objectif : Vérifier que l'application mobile communique correctement avec l'API backend pour récupérer la liste des voitures.

Objet de test : Méthode `getAllVoitures()` de l'interface `ApiService`. **Propriété à tester :** Communication réseau correcte et La méthode récupère une liste valide de voitures depuis l'API.

Données de test : Endpoint API : `http://10.0.2.2:8081/api/voitures`.

Base de données du backend contenant au moins une voiture avec des disponibilités. **Étapes du test :**

- 1) Connecter l'application mobile au backend.
- 2) S'assurer que le RecyclerView affiche correctement les données après la récupération en vérifiant les informations de la voiture affichée.

Résultat attendu : La liste affichée dans l'application correspond aux données de l'API.

4. Test d'acceptation utilisateur

Objectif : Vérifier que l'utilisateur peut consulter les détails d'une voiture et voir une image.

Objet de test : Flux utilisateur complet pour consulter une voiture. **Propriété à tester :** Base de données avec des voitures disponibles.

Étapes du test :

- 1) Lancer l'application et naviguer vers la liste des voitures.
- 2) Sélectionner une voiture dans la liste. **Résultat attendu :** La liste affichée dans l'application correspond aux données de l'API.
- 3) Vérifier que les détails affichés correspondent à la voiture sélectionnée (marque, modèle, prix...).
- 4) Vérifier que l'image s'affiche sans erreur.

Résultat attendu : Les détails affichés et l'image sont corrects et cohérents avec la sélection de l'utilisateur.

5. Rapport sommaire des tests

Type de Test	Objet de Test	Propriété à Tester	Données de Test	Étapes de Test	Résultat Attendu	Résultat Obtenu	Statut	Date Exécution Test	Risk Level	Fonctionnel ou Non Fonctionnel	Condition de Test
Test Unitaire	Méthode getDateDebutDisponibilite()	Conversion d'une date String en LocalDate	dateDebutDisponibilite = "2024-12-14"	1. Instancier un objet Disponibilite 2. Appeler getDateDebutDisponibilite() 3. Vérifier le résultat avec LocalDate.parse("2024-12-14")	Retourner un objet LocalDate avec la valeur "2024-12-14"	OK	Passé	2024-12-14	Faible	Fonctionnel	Test effectué en conditions normales sur l'app mobile avec la classe Disponibilite instanciée.
Test d'Intégration Composants	VoitureAdapter	Affichage correct des données dans la RecyclerView	Données fictives d'une voiture (marque, modèle, prix, disponibilités, image...)	1. Créer une liste de voitures avec les données de test 2. Injecter cette liste dans un VoitureAdapter 3. Vérifier l'affichage dans la RecyclerView	Les données (marque, modèle, prix, disponibilités) doivent s'afficher correctement dans la RecyclerView	OK	Passé	2024-12-14	Faible	Fonctionnel	Test effectué dans un environnement où l'adaptateur est utilisé avec une RecyclerView valide.
Test d'Intégration Système	Méthode getAllVoitures() de ApiService	Communication correcte avec l'API backend	Identifiant d'une voiture existante	1. Lancer l'application et connecter l'API. 2. Appeler getAllVoitures() 3. Vérifier que les données requises sont correctement affichées dans la liste de voitures.	Les voitures doivent s'afficher correctement dans l'application en fonction des données de l'API	OK	Passé	2024-12-14	Modéré	Fonctionnel	Connexion à l'API backend via réseau local (émulateur Android).
Test d'Acceptation Utilisateur	Fonctionnalité de consultation des détails de voiture	Expérience utilisateur : fluidité absence d'erreurs	Voiture avec : id = 1, marque = "Toyota", modèle = "Corolla", prixParJour = 50, imagePath = "/voitures/getImage/1"	1. Lancer l'application. 2. Naviguer vers la liste des voitures. 3. Sélectionner une voiture. 4. Vérifier que les détails de la voiture sont affichés correctement.	Les détails (marque, modèle, prix, image) doivent s'afficher correctement dans la vue des détails	OK	Passé	2024-12-14	Faible	Fonctionnel	Test effectué sur l'application mobile en conditions normales avec l'image accessible.

TABLEAU 7.1 : Tableau des tests effectués

7.10.2 Conclusion

Cette section présente les tests effectués sur notre application, visant à valider son bon fonctionnement et à garantir la qualité de l'expérience utilisateur.

Conclusion

Dans cette partie de l'application mobile, nous avons mis en œuvre une navigation fluide entre les différentes pages en utilisant des intents explicites. Cette approche nous a permis de passer d'une activité à une autre de manière précise et contrôlée, assurant ainsi une navigation cohérente pour l'utilisateur. Nous avons également créé un service pour vérifier périodiquement la disponibilité des voitures. Nous avons aussi gérer l'authentification via Facebook, garantissant que les utilisateurs peuvent se connecter de manière sécurisée et rapide. Après avoir développé cette fonctionnalité, nous avons effectué des tests rigoureux pour nous assurer de son bon fonctionnement. Ces tests ont validé la performance de l'authentification, la navigation entre les pages et la communication avec le backend. Grâce à ces efforts, l'application est désormais prête à offrir une expérience utilisateur fluide et sécurisée.

Conclusion et Perspective

Ce rapport présente le travail réalisé dans le cadre de la matière Projet d'Intégration. Tout au long de ce projet, nous avons été confrontés aux défis du travail en équipe et de l'application de la méthodologie Scrum, ce qui a considérablement enrichi notre expérience.

La location de voiture, aujourd'hui, peut souvent être perçue comme un processus complexe. Pour y remédier, nous avons développé une plateforme qui simplifie la mise en relation entre les propriétaires de véhicules et les locataires. L'objectif principal de notre projet est de permettre aux propriétaires de mettre en location leurs voitures de manière simple et sécurisée, tout en offrant aux clients la possibilité de trouver une voiture à un prix plus abordable que celui des agences de location classiques, grâce à l'absence d'intermédiaires. Ce modèle réduit les coûts et améliore l'expérience utilisateur des deux parties.

Notre plateforme facilite cette mise en relation en offrant une interface fluide et fonctionnelle qui permet aux locataires de rechercher, consulter et réserver des voitures en ligne, tout en garantissant une gestion optimale des disponibilités et des paiements. De plus, elle assure la transparence des transactions et la confiance au sein de la communauté grâce à un système de notation et de suivi des locations.

Sur le plan technique, ce stage nous a été une bonne opportunité pour développer nos compétences en développement des projets web et mobiles.

Loin d'être complètement achevé ce projet est toujours en cours d'avancement. Ce projet est apte à l'ajout d'autre enrichissement tels que le paiement via la carte D17 ou l'intégration de l'intelligence artificielle pour offrir au un système de recommandation personnalisé aux clients.

webographie

- [https://fr.wikipedia.org/wiki/Scrum_\(d%C3%A9veloppement\)#:~:text=Scrum%20est%20un%20framework%20ou,la%20plus%20grande%20valeur%20possible%20%C2%BB](https://fr.wikipedia.org/wiki/Scrum_(d%C3%A9veloppement)#:~:text=Scrum%20est%20un%20framework%20ou,la%20plus%20grande%20valeur%20possible%20%C2%BB).
- <https://spring.io/projects/spring-boot>
- <https://angular.dev/>
- <https://app.diagrams.net/>

