

# Système d'Enchères Automobiles

Conception et Implémentation d'une Application Mobile en Temps Réel

Rapport de Projet

*Auteur*

24 avril 2025

## Dédicaces

Je dédie ce travail

À mes chers parents,  
Pour leur soutien inconditionnel et leurs encouragements constants tout au long de mes études.

À ma famille,  
Pour leur présence et leur support moral dans tous mes projets.

À mes professeurs,  
Pour leur dévouement et la qualité de leur enseignement.

À mes amis et collègues,  
Pour leur aide et leur collaboration précieuse.

À tous ceux qui ont contribué de près ou de loin à ma formation et à la réalisation de ce projet.

# **Remerciements**

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet de fin d'études.

Mes remerciements s'adressent particulièrement à :

**Mon encadrant,**

Pour son suivi régulier, ses conseils précieux et sa disponibilité tout au long de ce projet.

**L'équipe pédagogique,**

Pour la qualité de la formation dispensée et leur engagement dans notre réussite.

**Les membres du jury,**

Pour l'intérêt qu'ils portent à ce travail et le temps qu'ils consacrent à son évaluation.

**Mes collègues de promotion,**

Pour leur esprit d'entraide et les moments de collaboration enrichissants.

Un remerciement spécial à tous ceux qui m'ont aidé à surmonter les défis techniques, notamment dans :

- La résolution des problèmes de ports et de configuration
- La mise en place du système de mise à jour automatique des enchères
- L'optimisation des performances de l'application
- L'implémentation des fonctionnalités en temps réel

Enfin, je remercie ma famille pour leur soutien moral et leurs encouragements constants durant toute la période de réalisation de ce projet.

# Résumé

Ce projet consiste en le développement d'une application d'enchères automobiles en temps réel utilisant React Native pour le frontend mobile, React.js pour le tableau de bord administratif, et Node.js avec MongoDB pour le backend. L'application permet aux utilisateurs de participer à des enchères de voitures, de placer des offres en temps réel, et de suivre leurs enchères gagnées ou perdues. Le système comprend également une interface d'administration pour la gestion des enchères, des utilisateurs et des véhicules.

Les principales fonctionnalités incluent :

- Gestion automatique des statuts d'enchères
- Notifications en temps réel via Socket.IO
- Interface mobile native pour iOS
- Tableau de bord administratif complet
- Base de données MongoDB pour la persistance des données

**Mots clés :** Enchères en temps réel, React Native, Node.js, MongoDB, Socket.IO, Application mobile

# **Abstract**

This project involves the development of a real-time car auction application using React Native for the mobile frontend, React.js for the admin dashboard, and Node.js with MongoDB for the backend. The application allows users to participate in car auctions, place real-time bids, and track their won or lost auctions. The system also includes an administration interface for managing auctions, users, and vehicles.

Key features include :

- Automatic auction status management
- Real-time notifications via Socket.IO
- Native iOS mobile interface
- Comprehensive admin dashboard
- MongoDB database for data persistence

**Keywords :** Real-time auctions, React Native, Node.js, MongoDB, Socket.IO, Mobile application

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contexte et Motivation . . . . .	1
1.2	Problématique . . . . .	1
1.3	Objectifs du Projet . . . . .	1
1.3.1	Objectifs Fonctionnels . . . . .	1
1.3.2	Objectifs Techniques . . . . .	2
1.3.3	Objectifs Pédagogiques . . . . .	2
1.4	Périmètre du Projet . . . . .	2
1.4.1	Fonctionnalités Couvertes . . . . .	2
1.4.2	Limites et Exclusions . . . . .	2
1.5	Méthodologie . . . . .	2
1.5.1	Approche de Développement . . . . .	2
1.5.2	Phases du Projet . . . . .	3
1.5.3	Outils de Gestion . . . . .	3
1.6	Structure du Rapport . . . . .	3
1.7	Aperçu de l'Application . . . . .	3
<b>2</b>	<b>État de l'Art</b>	<b>4</b>
2.1	Introduction aux Systèmes d'Enchères en Ligne . . . . .	4
2.1.1	Évolution Historique . . . . .	4
2.1.2	Typologies d'Enchères . . . . .	4
2.2	Analyse des Plateformes d'Enchères Automobiles Existantes . . . . .	5
2.2.1	Acteurs Majeurs du Marché . . . . .	5
2.2.2	Analyse Comparative . . . . .	5
2.3	Technologies Clés pour les Systèmes d'Enchères Modernes . . . . .	5
2.3.1	Technologies Frontend . . . . .	6
2.3.2	Technologies Backend . . . . .	6
2.3.3	Technologies de Base de Données . . . . .	7
2.4	Sécurité et Authentification . . . . .	7
2.4.1	Mécanismes d'Authentification Modernes . . . . .	7
2.4.2	Sécurisation des Transactions . . . . .	7
2.5	Tendances Émergentes et Innovations . . . . .	8
2.5.1	Intelligence Artificielle et Machine Learning . . . . .	8
2.5.2	Blockchain et Contrats Intelligents . . . . .	8
2.5.3	Réalité Augmentée et Virtuelle . . . . .	8
2.6	Défis et Opportunités du Marché . . . . .	8
2.6.1	Défis Actuels . . . . .	8

2.6.2	Opportunités de Marché . . . . .	9
2.7	Conclusion et Positionnement de Notre Solution . . . . .	9
<b>3</b>	<b>Technologies et Outils Utilisés</b>	<b>10</b>
3.1	Vue d'Ensemble de la Stack Technologique . . . . .	10
3.1.1	Architecture Globale . . . . .	10
3.2	Technologies Frontend . . . . .	10
3.2.1	React Native . . . . .	10
3.2.2	React Navigation . . . . .	11
3.2.3	Gestion de l'État . . . . .	11
3.2.4	Bibliothèques UI Complémentaires . . . . .	13
3.3	Technologies Backend . . . . .	13
3.3.1	Node.js et Express . . . . .	13
3.3.2	Architecture API RESTful . . . . .	14
3.3.3	Sécurité . . . . .	15
3.4	Communication Temps Réel avec Socket.io . . . . .	16
3.4.1	Principes de Fonctionnement . . . . .	16
3.4.2	Intégration Côté Client . . . . .	18
3.5	Base de Données MongoDB . . . . .	20
3.5.1	Choix de MongoDB . . . . .	20
3.5.2	Modélisation des Données . . . . .	20
3.5.3	Optimisation des Performances . . . . .	24
3.6	Outils de Développement et Déploiement . . . . .	24
3.6.1	Environnement de Développement . . . . .	24
3.6.2	CI/CD et Déploiement . . . . .	24
3.6.3	Monitoring et Logging . . . . .	24
3.7	Choix Technologiques et Alternatives . . . . .	25
3.7.1	Justification des Choix . . . . .	25
3.7.2	Alternatives Considérées . . . . .	25
<b>4</b>	<b>Manuel d'Utilisation</b>	<b>26</b>
4.1	Introduction à l'Application . . . . .	26
4.1.1	Public Cible . . . . .	26
4.1.2	Prérequis Techniques . . . . .	26
4.2	Installation et Configuration . . . . .	26
4.2.1	Téléchargement de l'Application . . . . .	26
4.2.2	Création de Compte . . . . .	26
4.2.3	Connexion . . . . .	27
4.3	Interface Utilisateur . . . . .	27
4.3.1	Navigation Principale . . . . .	27
4.3.2	Écran d'Accueil . . . . .	27
4.4	Participation aux Enchères . . . . .	27
4.4.1	Recherche et Consultation des Enchères . . . . .	27
4.4.2	Détails d'une Enchère . . . . .	28

4.4.3	Placement d'une Enchère . . . . .	28
4.4.4	Suivi des Enchères . . . . .	28
4.5	Gestion du Profil Utilisateur . . . . .	28
4.5.1	Modification des Informations Personnelles . . . . .	28
4.5.2	Gestion des Notifications . . . . .	28
4.5.3	Historique des Transactions . . . . .	29
4.6	Fonctionnalités Administrateur . . . . .	29
4.6.1	Gestion des Véhicules . . . . .	29
4.6.2	Création et Gestion des Enchères . . . . .	29
4.6.3	Gestion des Utilisateurs . . . . .	30
4.7	Résolution des Problèmes Courants . . . . .	30
4.7.1	Problèmes de Connexion . . . . .	30
4.7.2	Problèmes avec les Enchères . . . . .	30
4.7.3	Problèmes Techniques Généraux . . . . .	30
4.8	Support et Assistance . . . . .	30
4.8.1	Centre d'Aide . . . . .	30
4.8.2	Contact du Support . . . . .	31
4.9	Interface de l'Application . . . . .	31
4.9.1	Authentification et Inscription . . . . .	32
4.9.2	Page d'Accueil et Enchères . . . . .	33
4.9.3	Gestion des Enchères . . . . .	34
4.9.4	Profil Utilisateur . . . . .	35
4.9.5	Détails du Véhicule et Enchère . . . . .	36
4.9.6	Historique des Enchères . . . . .	37
4.10	Fonctionnalités Principales . . . . .	38
4.11	Conseils d'Utilisation . . . . .	38
<b>5</b>	<b>Analyse de l'Existant</b>	<b>39</b>
5.1	Solutions d'Enchères Existantes . . . . .	39
5.1.1	Solutions Traditionnelles . . . . .	39
5.1.2	Solutions Numériques Actuelles . . . . .	39
5.2	Limitations des Solutions Existantes . . . . .	39
5.2.1	Problèmes Techniques . . . . .	39
5.2.2	Problèmes Fonctionnels . . . . .	39
5.3	Opportunités d'Amélioration . . . . .	39
5.3.1	Aspects Techniques . . . . .	39
5.3.2	Aspects Fonctionnels . . . . .	39
5.4	Analyse des Risques . . . . .	40
<b>6</b>	<b>Analyse des Besoins</b>	<b>41</b>
6.1	Méthodologie d'Analyse . . . . .	41
6.1.1	Collecte des Besoins . . . . .	41
6.1.2	Priorisation des Besoins . . . . .	41
6.2	Exigences Fonctionnelles . . . . .	41

6.2.1	Gestion des Utilisateurs . . . . .	41
6.2.2	Gestion des Véhicules . . . . .	42
6.2.3	Gestion des Enchères . . . . .	42
6.2.4	Notifications et Communication . . . . .	43
6.2.5	Recherche et Filtrage . . . . .	43
6.2.6	Administration . . . . .	43
6.3	Exigences Non Fonctionnelles . . . . .	44
6.3.1	Performance . . . . .	44
6.3.2	Sécurité . . . . .	44
6.3.3	Fiabilité . . . . .	45
6.3.4	Utilisabilité . . . . .	45
6.3.5	Compatibilité . . . . .	45
6.4	Contraintes Techniques . . . . .	45
6.5	Scénarios d'Utilisation . . . . .	46
6.5.1	Scénario 1 : Inscription et Première Enchère . . . . .	46
6.5.2	Scénario 2 : Gestion d'une Enchère par un Administrateur . . . . .	46
6.5.3	Scénario 3 : Finalisation d'une Enchère Réussie . . . . .	46
6.6	Matrice de Traçabilité . . . . .	47
<b>7</b>	<b>Conception du Système d'Enchères</b>	<b>48</b>
7.1	Architecture Globale . . . . .	48
7.1.1	Vue d'ensemble . . . . .	48
7.2	Diagramme de Cas d'Utilisation . . . . .	48
7.2.1	Acteurs du Système . . . . .	48
7.2.2	Cas d'Utilisation Principaux . . . . .	48
7.3	Diagramme de Séquence . . . . .	48
7.3.1	Analyse du Flux d'Enchère . . . . .	49
7.4	Diagramme de Classes . . . . .	49
7.4.1	Entités Principales . . . . .	49
7.4.2	Relations Entre Entités . . . . .	49
7.5	Modèles de Données . . . . .	49
7.5.1	Modèle Utilisateur . . . . .	49
7.5.2	Modèle Enchère . . . . .	50
7.5.3	Modèle Voiture . . . . .	50
7.5.4	Modèle Offre . . . . .	50
7.6	Architecture Technique . . . . .	50
7.6.1	Architecture Frontend . . . . .	50
7.6.2	Architecture Backend . . . . .	51
7.6.3	Communication en Temps Réel . . . . .	51
7.6.4	Base de Données . . . . .	51
<b>8</b>	<b>Réalisation Technique</b>	<b>55</b>
8.1	Implémentation Frontend . . . . .	55
8.1.1	Structure de l'Application Mobile . . . . .	55

8.1.2	Composants Réutilisables . . . . .	55
8.1.3	Gestion du State . . . . .	56
8.1.4	Communication Temps Réel . . . . .	57
8.2	Implémentation Backend . . . . .	57
8.2.1	Architecture API . . . . .	57
8.2.2	Contrôleurs . . . . .	58
8.2.3	Middleware . . . . .	60
8.2.4	Services . . . . .	60
8.3	Intégration et Tests . . . . .	62
8.3.1	Tests Unitaires . . . . .	62
8.3.2	Déploiement . . . . .	63
8.3.3	Sécurité . . . . .	63
8.4	Défis Techniques et Solutions . . . . .	63
8.4.1	Concurrence dans les Enchères . . . . .	63
8.4.2	Performances en Temps Réel . . . . .	64
8.4.3	Gestion des Images . . . . .	64
<b>9</b>	<b>Défis d'Implémentation et Perspectives d'Évolution</b>	<b>65</b>
9.1	Défis Techniques Rencontrés . . . . .	65
9.1.1	Gestion de la Concurrence . . . . .	65
9.1.2	Optimisation des Performances . . . . .	66
9.1.3	Sécurité et Authentification . . . . .	67
9.2	Perspectives d'Évolution . . . . .	68
9.2.1	Fonctionnalités Additionnelles . . . . .	68
9.2.2	Améliorations Techniques . . . . .	68
9.2.3	Expansion Internationale . . . . .	69
9.3	Analyse Comparative avec des Solutions Existantes . . . . .	69
9.3.1	Forces du Système . . . . .	69
9.3.2	Opportunités d'Amélioration . . . . .	69
9.4	Impact Écologique et Économique . . . . .	70
9.4.1	Réduction de l'Empreinte Carbone . . . . .	70
9.4.2	Démocratisation du Marché Automobile . . . . .	70
<b>10</b>	<b>Conclusion</b>	<b>71</b>
10.1	Synthèse du Projet . . . . .	71
10.1.1	Réalisations Principales . . . . .	71
10.1.2	Objectifs Atteints . . . . .	71
10.2	Compétences Développées . . . . .	71
10.2.1	Compétences Techniques . . . . .	71
10.2.2	Compétences Méthodologiques . . . . .	71
10.3	Apports Personnels et Professionnels . . . . .	72
10.3.1	Développement Personnel . . . . .	72
10.3.2	Développement Professionnel . . . . .	72
10.4	Perspectives d'Avenir . . . . .	72

10.4.1	Évolution du Projet . . . . .	72
10.4.2	Applications des Connaissances Acquises . . . . .	72
10.5	Mot de Fin . . . . .	72
<b>11</b>	<b>Annexes</b>	<b>74</b>
11.1	Guide d'Installation . . . . .	74
11.1.1	Prérequis . . . . .	74
11.1.2	Installation du Projet . . . . .	74
11.1.3	Configuration . . . . .	74
11.2	Documentation API . . . . .	74
11.2.1	Routes d'Authentification . . . . .	74
11.2.2	Routes des Enchères . . . . .	75
11.3	Schéma de la Base de Données . . . . .	75
11.3.1	Collection Users . . . . .	75
11.3.2	Collection Auctions . . . . .	75
11.4	Captures d'écran de l'application . . . . .	76
11.5	Logs et Débogage . . . . .	76
11.5.1	Exemple de Logs . . . . .	76
11.6	Scripts Utiles . . . . .	76
11.6.1	Gestion des Processus . . . . .	76

## Table des figures

4.1	Navigation principale de l'application . . . . .	27
4.2	Interface de placement d'enchère . . . . .	28
4.3	Écrans d'authentification de l'application . . . . .	32
4.4	Pages d'accueil avec différentes vues des enchères . . . . .	33
4.5	Écrans de gestion des enchères . . . . .	34
4.6	Gestion du profil utilisateur . . . . .	35
4.7	Détails du véhicule et système d'enchères . . . . .	36
4.8	Historique des enchères de l'utilisateur . . . . .	37
7.1	Diagramme de cas d'utilisation du système d'enchères . . . . .	52
7.2	Diagramme de séquence pour le processus d'enchère . . . . .	53
7.3	Diagramme de classes des principaux composants du système . . . . .	54

## **Liste des tableaux**

2.1 Comparaison des principales plateformes d'enchères automobiles . . . . .	5
3.1 Comparaison des technologies considérées . . . . .	25
6.1 Extrait de la matrice de traçabilité . . . . .	47

# Chapitre 1

## Introduction

### 1.1 Contexte et Motivation

Le marché des ventes de véhicules d'occasion connaît une transformation numérique significative, avec une demande croissante pour des plateformes permettant d'acheter et de vendre des véhicules en ligne. Cette évolution est d'autant plus marquée que les consommateurs recherchent aujourd'hui des expériences d'achat plus accessibles, transparentes et efficaces.

Dans ce contexte, les systèmes d'enchères en ligne pour automobiles représentent une solution particulièrement adaptée, offrant un mécanisme de marché dynamique où la valeur des véhicules est déterminée collectivement par les acheteurs potentiels. Contrairement aux plateformes traditionnelles d'annonces automobiles, les systèmes d'enchères permettent une valorisation plus précise des véhicules et offrent une expérience interactive engageante pour les participants.

Cependant, la mise en œuvre d'un tel système présente des défis techniques considérables, notamment en termes de gestion des transactions en temps réel, de sécurisation des échanges, et de conception d'interfaces utilisateur intuitives sur appareils mobiles. Ces défis constituent le point de départ de notre projet.

### 1.2 Problématique

Le développement d'une application d'enchères automobiles en temps réel soulève plusieurs questions essentielles :

- Comment concevoir un système d'enchères qui garantit l'équité et la transparence pour tous les participants ?
- Quelles architectures techniques permettent de gérer efficacement la concurrence et le temps réel dans un contexte d'enchères ?
- Comment assurer une expérience utilisateur fluide et accessible sur plateformes mobiles, malgré les contraintes de connectivité et les variations de performances des appareils ?
- Quels mécanismes de sécurité mettre en place pour protéger les transactions et les données personnelles des utilisateurs ?
- Comment structurer l'application pour qu'elle puisse évoluer et s'adapter aux besoins futurs du marché ?

Ces problématiques constituent le cœur de notre réflexion et ont guidé les choix techniques et fonctionnels tout au long du développement de l'application.

### 1.3 Objectifs du Projet

Dans le cadre de ce projet, nous nous sommes fixé plusieurs objectifs :

#### 1.3.1 Objectifs Fonctionnels

- Développer une application mobile permettant aux utilisateurs de consulter et participer à des enchères automobiles

- Mettre en place un système d'enchères en temps réel avec mise à jour instantanée des prix
- Implémenter des mécanismes de notification pour informer les utilisateurs des événements importants (surenchère, fin d'enchère, etc.)
- Créer un panneau d'administration pour la gestion des véhicules, des enchères et des utilisateurs
- Assurer une présentation détaillée et attrayante des véhicules mis aux enchères

### 1.3.2 Objectifs Techniques

- Concevoir une architecture évolutive et maintenable basée sur des technologies modernes
- Implémenter des communications bidirectionnelles en temps réel entre clients et serveur
- Garantir la sécurité et l'intégrité des données dans un environnement d'enchères concurrentiel
- Optimiser les performances pour assurer une expérience fluide, même lors de pics d'activité
- Développer une interface utilisateur réactive et intuitive sur plateformes iOS et Android

### 1.3.3 Objectifs Pédagogiques

- Approfondir la maîtrise du développement d'applications mobiles avec React Native
- Explorer les techniques de communication en temps réel avec Socket.io
- Appliquer les principes de conception UML à un projet concret
- Développer des compétences en architecture de systèmes distribués
- Mettre en pratique les méthodes de gestion de projet agile

## 1.4 Périmètre du Projet

### 1.4.1 Fonctionnalités Couvertes

Le projet couvre les fonctionnalités suivantes :

- Inscription et authentification des utilisateurs
- Consultation des enchères en cours et à venir
- Participation aux enchères avec mise à jour en temps réel
- Système de notifications pour les événements importants
- Gestion de profil utilisateur
- Interface d'administration pour la gestion des enchères et des véhicules
- Visualisation détaillée des véhicules avec galerie d'images
- Historique des enchères et des transactions

### 1.4.2 Limites et Exclusions

Le projet n'inclut pas :

- Le traitement des paiements en ligne (prévu pour une version ultérieure)
- L'intégration avec des services de vérification d'historique de véhicules
- La gestion logistique de livraison des véhicules
- Un système de messagerie instantanée entre acheteurs et vendeurs
- Une version web desktop complète (focus sur applications mobiles)

## 1.5 Méthodologie

### 1.5.1 Approche de Développement

Le projet a été mené selon une approche agile, avec :

- Des sprints de deux semaines

- Des réunions quotidiennes pour synchroniser l'équipe
- Des revues de sprint pour valider les fonctionnalités développées
- Une amélioration continue basée sur les retours des tests utilisateurs

### 1.5.2 Phases du Projet

Le projet s'est déroulé en plusieurs phases :

1. **Analyse des besoins** : Identification des exigences fonctionnelles et techniques
2. **Conception** : Élaboration de l'architecture et modélisation UML
3. **Développement** : Implémentation itérative des fonctionnalités
4. **Tests** : Validation technique et fonctionnelle
5. **Déploiement** : Mise en production de l'application
6. **Maintenance** : Correction des bugs et améliorations continues

### 1.5.3 Outils de Gestion

Pour la gestion du projet, nous avons utilisé :

- **Jira** : Suivi des tâches et des sprints
- **GitHub** : Gestion de versions et revue de code
- **Figma** : Conception des interfaces utilisateur
- **Lucidchart** : Création des diagrammes UML
- **Slack** : Communication d'équipe

## 1.6 Structure du Rapport

Ce rapport est organisé en plusieurs chapitres qui couvrent l'ensemble du processus de développement :

**Chapitre 1-4** Introduction, remerciements et résumé.

**Chapitre 5** Introduction générale au projet.

**Chapitre 6** Présentation détaillée des technologies et outils utilisés.

**Chapitre 7** Manuel d'utilisation de l'application.

**Chapitre 8** Analyse des besoins et des exigences du système.

**Chapitre 9** Conception du système avec diagrammes UML.

**Chapitre 10** Implémentation technique et détails de réalisation.

**Chapitre 11** Défis d'implémentation et perspectives d'évolution.

**Chapitre 12** Conclusion et bilan du projet.

Chaque chapitre vise à présenter de manière claire et structurée les différents aspects du développement de notre application d'enchères automobiles, depuis l'analyse initiale jusqu'à la mise en production et aux perspectives futures.

## 1.7 Aperçu de l'Application

Notre application d'enchères automobiles se distingue par plusieurs caractéristiques clés :

- Une interface utilisateur moderne et intuitive, optimisée pour les appareils mobiles
- Un système d'enchères en temps réel permettant une expérience interactive
- Une architecture technique robuste basée sur des technologies éprouvées
- Une attention particulière à la sécurité et à la protection des données
- Une conception évolutive permettant l'ajout futur de fonctionnalités

Ces caractéristiques constituent les fondements de notre solution et seront détaillées dans les chapitres suivants, accompagnées d'illustrations, de diagrammes et d'exemples de code.

## Chapitre 2

### État de l'Art

#### 2.1 Introduction aux Systèmes d'Enchères en Ligne

Les enchères en ligne ont considérablement évolué depuis leur apparition dans les années 1990. Ce qui a commencé comme de simples plateformes de vente aux enchères s'est transformé en écosystèmes sophistiqués intégrant des technologies avancées pour améliorer l'expérience utilisateur, la sécurité des transactions et l'efficacité globale du processus d'enchaînement.

##### 2.1.1 Évolution Historique

L'histoire des enchères en ligne peut être divisée en plusieurs phases distinctes :

1. **Première génération (1995-2000)** : Caractérisée par des plateformes comme eBay, offrant des fonctionnalités basiques d'enches avec des interfaces utilisateur simples et des mécanismes de communication asynchrones.
2. **Deuxième génération (2000-2010)** : Introduction de fonctionnalités avancées comme les enches automatiques, les systèmes de réputation, et l'intégration de moyens de paiement sécurisés.
3. **Troisième génération (2010-2015)** : Développement de plateformes spécialisées par secteur (art, automobiles, immobilier), avec des fonctionnalités adaptées aux spécificités de chaque domaine.
4. **Quatrième génération (2015-présent)** : Intégration de technologies temps réel, d'applications mobiles avancées, d'intelligence artificielle, et de mécanismes de vérification d'identité renforcés.

##### 2.1.2 Typologies d'Enchères

Les systèmes d'enches en ligne peuvent implémenter différents modèles d'enches, chacun avec ses propres règles et mécanismes :

- **Enchères anglaises** : Le modèle le plus courant, où les participants encherissent à la hausse jusqu'à ce que plus personne ne surenchérisse.
- **Enchères hollandaises** : Commençant à un prix élevé qui diminue progressivement jusqu'à ce qu'un acheteur accepte le prix courant.
- **Enchères à prix secret** : Les participants soumettent leurs offres en privé, et le plus offrant remporte l'encheûtre.
- **Enchères à paliers** : Le prix augmente par incrémentations prédefinies à intervalles réguliers.
- **Enchères inversées** : Les vendeurs encherissent à la baisse pour offrir le prix le plus bas pour un service ou produit.

Notre système d'enches automobiles s'appuie principalement sur le modèle d'enches anglaises, tout en intégrant certaines fonctionnalités avancées comme l'extension automatique du temps lors d'enches de dernière minute.

## 2.2 Analyse des Plateformes d'Enchères Automobiles Existantes

Plusieurs plateformes d'enquêtes automobiles existantes ont été analysées pour identifier les meilleures pratiques et les opportunités d'innovation.

### 2.2.1 Acteurs Majeurs du Marché

**Bring a Trailer** Plateforme spécialisée dans les véhicules de collection et d'enthousiaste, caractérisée par des descriptions détaillées, une communauté active et un modèle d'enquêtes de 7 jours avec extension automatique.

**Cars & Bids** Focalisée sur les véhicules modernes enthousiastes (moins de 30 ans), avec une interface utilisateur moderne, des frais réduits, et une vérification des annonces.

**eBay Motors** Plateforme généraliste avec une section dédiée aux véhicules, offrant à la fois des options d'achat immédiat et d'enquêtes, avec une large audience mondiale.

**Copart** Spécialisée dans les véhicules accidentés et de récupération, avec un système d'enquêtes hybride (en personne et en ligne) destiné principalement aux professionnels.

**CarOnSale** Plateforme européenne dédiée aux professionnels de l'automobile, avec un système d'enquêtes B2B et des services d'inspection des véhicules.

### 2.2.2 Analyse Comparative

Caractéristique	Bring a Trailer	Cars & Bids	eBay Motors	Copart	CarOnSale
Public cible	Collectionneurs	Enthousiastes modernes	Grand public	Professionnels	Professionnels
Durée d'enquête	7 jours	7 jours	Variable	Court terme	24-48h
Extension auto.	Oui	Oui	Non	Limitée	Oui
Mobile App	Limitée	Oui	Complète	Complète	Basique
Communication TR	Modérée	Avancée	Limitée	Basique	Modérée
Inspection	Par la communauté	Vérification photos	Optionnelle	Complète	Standardisée
Modèle de revenus	Commission vendeur et acheteur	Commission vendeur	Commission vendeur	Frais d'inscription et commission	Commission variable

TABLE 2.1 – Comparaison des principales plateformes d'enquêtes automobiles

Cette analyse comparative révèle plusieurs tendances et opportunités :

- L'importance croissante des fonctionnalités mobiles complètes
- La valeur ajoutée des communications en temps réel pendant les enquêtes
- L'évolution vers des systèmes d'extension automatique pour éviter le "sniping" (enquêtes de dernière seconde)
- Le besoin d'équilibrer la simplicité d'utilisation et la richesse fonctionnelle

## 2.3 Technologies Clés pour les Systèmes d'Enquêtes Modernes

Le développement d'un système d'enquêtes automobiles moderne nécessite l'intégration de plusieurs technologies avancées.

### 2.3.1 Technologies Frontend

#### Applications Mobiles Natives vs Hybrides

Le débat entre applications natives et hybrides reste pertinent pour les plateformes d'enchères :

- **Applications natives** : Offrent des performances optimales et un accès complet aux fonctionnalités du dispositif, mais nécessitent un développement séparé pour chaque plateforme.
- **Applications hybrides** : Permettent un développement unique pour plusieurs plateformes, avec des performances qui se sont considérablement améliorées grâce à des frameworks comme React Native et Flutter.

Pour notre système, nous avons opté pour React Native, qui offre un bon compromis entre performances natives et efficacité de développement multiplateforme.

#### Interfaces Utilisateur Réactives

Les tendances modernes en matière d'interface utilisateur pour les systèmes d'enchères incluent :

- **Design minimaliste** : Interfaces épurées qui mettent l'accent sur le contenu et les fonctionnalités essentielles.
- **Navigation intuitive** : Structures de navigation simples et cohérentes qui facilitent l'accès aux différentes fonctionnalités.
- **Retours visuels immédiats** : Animations et transitions subtiles qui fournissent un feedback instantané aux actions de l'utilisateur.
- **Microinteractions** : Petites animations fonctionnelles qui enrichissent l'expérience utilisateur, particulièrement importantes dans le contexte des enchères où chaque interaction peut être critique.

### 2.3.2 Technologies Backend

#### Architectures API

Plusieurs approches architecturales sont utilisées dans les systèmes d'enchères modernes :

- **REST** : Architecture mature et largement adoptée, bien adaptée pour les opérations CRUD standard.
- **GraphQL** : Offre une flexibilité accrue pour les requêtes complexes et réduit le surfetching, particulièrement utile pour les applications mobiles avec des contraintes de bande passante.
- **gRPC** : Protocole haute performance utilisant Protocol Buffers, adapté pour les communications entre microservices.

Notre système utilise principalement une API RESTful pour sa simplicité et sa large compatibilité, tout en intégrant des WebSockets pour les communications en temps réel.

#### Communication Temps Réel

Les technologies de communication temps réel sont essentielles pour les systèmes d'enchères :

- **WebSockets** : Protocole établissant une connexion bidirectionnelle persistante entre le client et le serveur, idéal pour les mises à jour d'enchères en temps réel.
- **Server-Sent Events (SSE)** : Alternative plus légère aux WebSockets, adaptée pour les scénarios où la communication est principalement du serveur vers le client.
- **Socket.IO** : Bibliothèque qui simplifie l'utilisation des WebSockets avec des fonctionnalités de fallback et de reconnexion automatique.

Notre choix s'est porté sur Socket.IO pour sa robustesse, sa facilité d'intégration et ses mécanismes de fallback qui garantissent une expérience utilisateur optimale même dans des conditions réseau variables.

### 2.3.3 Technologies de Base de Données

#### Bases de Données SQL vs NoSQL

Le choix entre SQL et NoSQL dépend des besoins spécifiques du système :

- **SQL** : Offre des garanties ACID fortes et une structure relationnelle bien adaptée pour les données structurées avec des relations complexes.
- **NoSQL** : Propose une meilleure scalabilité horizontale et une flexibilité de schéma adaptée aux données semi-structurées et aux évolutions rapides.

Les systèmes d'enchères modernes utilisent souvent une approche hybride, avec :

- Des bases SQL pour les données critiques nécessitant une intégrité forte (transactions, informations utilisateurs)
- Des bases NoSQL pour les données à haute vitesse ou nécessitant une évolutivité particulière (historique des enchères, logs d'activité)

Notre système utilise MongoDB comme solution principale, complétée par Redis pour le caching et la gestion des sessions.

#### Technologies de Cache

Les mécanismes de cache sont cruciaux pour maintenir les performances sous charge :

- **Redis** : Solution de stockage en mémoire polyvalente, utilisée pour le caching, les sessions, et les files d'attente de messages.
- **Memcached** : Alternative plus simple et focalisée uniquement sur le caching.
- **CDN** : Pour la distribution optimisée des assets statiques comme les images de véhicules.

## 2.4 Sécurité et Authentification

### 2.4.1 Mécanismes d'Authentification Modernes

Les systèmes d'enchères nécessitent des mécanismes d'authentification robustes :

- **JWT (JSON Web Tokens)** : Standard moderne pour l'authentification sans état, bien adapté aux architectures distribuées.
- **OAuth 2.0 / OpenID Connect** : Protocoles standardisés pour l'authentification déléguée et la fédération d'identité.
- **Authentification Multi-facteurs** : Couche de sécurité supplémentaire devenue standard pour les plateformes manipulant des transactions financières.
- **Authentification biométrique** : Intégration des capteurs d'empreintes digitales et de reconnaissance faciale des appareils mobiles.

### 2.4.2 Sécurisation des Transactions

La sécurité des transactions est particulièrement critique dans les systèmes d'enchères automobiles :

- **Validation stricte des enchères** : Vérifications côté serveur pour prévenir les manipulations.
- **Prévention des conditions de course** : Mécanismes de verrouillage et transactions atomiques pour garantir l'intégrité des enchères concurrentes.
- **Audit trails** : Journalisation immuable de toutes les actions critiques pour la traçabilité et la résolution de litiges.

- **Chiffrement de bout en bout** : Protection des communications sensibles entre l'utilisateur et le serveur.
- **Détection de fraude** : Algorithmes de détection d'anomalies pour identifier les comportements suspects.

## 2.5 Tendances Émergentes et Innovations

### 2.5.1 Intelligence Artificielle et Machine Learning

L'IA et le ML transforment plusieurs aspects des systèmes d'enchères :

- **Estimation automatique de valeur** : Algorithmes prédictifs pour suggérer des prix de départ optimaux basés sur les caractéristiques du véhicule et les données historiques.
- **Détection de fraude avancée** : Modèles de ML pour identifier les schémas d'activités suspectes en temps réel.
- **Recommandations personnalisées** : Suggestion de véhicules pertinents basée sur l'historique et les préférences de l'utilisateur.
- **Analyse d'images** : Validation automatique des photos de véhicules et détection des défauts ou incohérences.

### 2.5.2 Blockchain et Contrats Intelligents

La technologie blockchain offre des perspectives intéressantes pour les systèmes d'enchères :

- **Transparence et immuabilité** : Enregistrement inaltérable de l'historique des enchères et des transactions.
- **Contrats intelligents** : Automatisation des règles d'enchères et des processus post-enchère (paiement, transfert de propriété).
- **Tokenisation** : Représentation numérique de la propriété d'un véhicule facilitant les transferts sécurisés.
- **Vérification d'identité décentralisée** : Systèmes KYC (Know Your Customer) plus robustes et respectueux de la vie privée.

Bien que prometteuses, ces technologies restent émergentes dans le secteur des enchères automobiles et font face à des défis d'adoption liés à la complexité technique et aux questions réglementaires.

### 2.5.3 Réalité Augmentée et Virtuelle

Les technologies immersives commencent à être intégrées dans les plateformes d'enchères automobiles avancées :

- **Visites virtuelles** : Exploration détaillée des véhicules en 3D sans nécessité de déplacement physique.
- **Superposition d'informations** : Utilisation de la RA pour afficher des informations contextuelles sur les différents éléments d'un véhicule.
- **Simulation de conduite** : Expériences virtuelles permettant d'avoir un aperçu du comportement routier d'un véhicule.

## 2.6 Défis et Opportunités du Marché

### 2.6.1 Défis Actuels

Le développement et l'exploitation de systèmes d'enchères automobiles font face à plusieurs défis :

- **Confiance et vérification** : Établir la confiance dans un environnement en ligne où l'acheteur ne peut pas inspecter physiquement le véhicule.
- **Réglementation** : Navigation dans un paysage réglementaire complexe et variable selon les juridictions.
- **Concurrence** : Différenciation dans un marché de plus en plus saturé.
- **Expérience mobile** : Offrir une expérience d'enchère complète et fluide sur des appareils mobiles malgré les limitations d'interface.
- **Internationalisation** : Adaptation aux spécificités régionales tout en maintenant une plateforme cohérente.

### 2.6.2 Opportunités de Marché

Plusieurs tendances créent des opportunités significatives :

- **Digitalisation accélérée** : Adoption croissante des solutions numériques pour l'achat et la vente de véhicules, accélérée par la pandémie de COVID-19.
- **Évolution des préférences consommateurs** : Augmentation de la confiance dans les achats en ligne de biens de valeur élevée.
- **Segment des véhicules électriques** : Croissance rapide créant de nouveaux segments de marché avec des besoins spécifiques.
- **Marchés émergents** : Expansion dans des régions où le commerce électronique automobile est encore en développement.
- **Services complémentaires** : Intégration de services à valeur ajoutée (financement, assurance, logistique) dans l'écosystème d'enchères.

## 2.7 Conclusion et Positionnement de Notre Solution

L'analyse de l'état de l'art révèle un secteur des enchères automobiles en ligne en pleine évolution, avec une sophistication croissante des technologies et des attentes utilisateurs de plus en plus élevées.

Notre système d'enchères automobiles se positionne à l'intersection de ces tendances, avec :

- Une approche centrée sur l'expérience mobile, reflétant la transition vers un usage majoritairement mobile
- Un système de communication en temps réel optimisé, crucial pour l'engagement des utilisateurs
- Une architecture évolutive capable d'intégrer progressivement les innovations technologiques
- Un focus sur la transparence et la sécurité pour établir la confiance des utilisateurs

Les choix technologiques et fonctionnels détaillés dans ce rapport s'inscrivent dans cette vision, visant à créer une plateforme qui répond aux besoins actuels tout en anticipant les évolutions futures du marché.

# Chapitre 3

## Technologies et Outils Utilisés

### 3.1 Vue d'Ensemble de la Stack Technologique

Le développement d'une application d'enchères automobiles en temps réel nécessite une pile technologique robuste et moderne, capable de gérer efficacement les communications bidirectionnelles, le traitement des données et l'expérience utilisateur. Notre choix s'est porté sur un ensemble de technologies complémentaires formant un écosystème cohérent.

#### 3.1.1 Architecture Globale

L'architecture technique du système repose sur quatre composants principaux :

- **Frontend Mobile** : Interface utilisateur développée avec React Native
- **Backend API** : Serveur RESTful développé avec Node.js et Express
- **Communication Temps Réel** : Implémentée avec Socket.io
- **Base de Données** : MongoDB pour le stockage persistant des données

Cette architecture permet une séparation claire des responsabilités tout en assurant une communication fluide entre les différentes couches du système.

### 3.2 Technologies Frontend

#### 3.2.1 React Native

React Native constitue le choix central pour le développement de l'interface utilisateur mobile. Cette technologie présente plusieurs avantages décisifs :

- **Développement multiplateforme** : Une base de code unique déployable sur iOS et Android
- **Performances natives** : Rendu d'interfaces utilisateur natives, offrant des performances proches des applications développées en code natif
- **Architecture à composants** : Organisation modulaire du code favorisant la réutilisation et la maintenance
- **Hot Reloading** : Cycle de développement accéléré grâce au rechargement à chaud
- **Écosystème riche** : Large éventail de bibliothèques tierces disponibles

#### Composants React Native Principaux

L'application utilise intensivement les composants React Native suivants :

```
import React, { useState, useEffect } from 'react';
import {
  View,
  Text,
  Image,
  TouchableOpacity,
```

```
FlatList,
StyleSheet,
ActivityIndicator,
Alert
} from 'react-native';
```

### 3.2.2 React Navigation

La navigation entre les écrans est gérée par React Navigation, qui offre :

- Une navigation fluide et performante entre les écrans
- La gestion de l'historique de navigation
- Différents types de navigateurs (stack, tab, drawer)
- Une intégration profonde avec les gestes natifs

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

// Configuration de base de la navigation
function AppNavigator() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{
            title: 'Accueil',
            headerStyle: styles.header
          }}
        />
        <Stack.Screen name="AuctionDetail" component={AuctionDetailScreen} />
        {/* Autres écrans... */}
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

### 3.2.3 Gestion de l'État

La gestion de l'état de l'application utilise plusieurs approches complémentaires :

- **Hooks React** : useState et useEffect pour l'état local des composants
- **Context API** : Pour le partage d'état entre composants proches
- **AsyncStorage** : Pour la persistance locale des données comme les tokens d'authentification

```
// Exemple d'utilisation du Context API pour l'authentification
const AuthContext = React.createContext();
```

```
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  // Vérification du token au démarrage
  useEffect(() => {
    const loadUserFromStorage = async () => {
      try {
        const token = await AsyncStorage.getItem('userToken');
        if (token) {
          const userData = await authService.getUserData(token);
          setUser(userData);
        }
      } catch (error) {
        console.log('Failed to load user:', error);
      } finally {
        setLoading(false);
      }
    };
    loadUserFromStorage();
  }, []);

  // Fonctions d'authentification
  const login = async (email, password) => {
    setLoading(true);
    try {
      const response = await authService.login(email, password);
      await AsyncStorage.setItem('userToken', response.token);
      setUser(response.user);
      return { success: true };
    } catch (error) {
      return { success: false, error: error.message };
    } finally {
      setLoading(false);
    }
  };

  const logout = async () => {
    await AsyncStorage.removeItem('userToken');
    setUser(null);
  };

  // Valeur fournie par le contexte
  const authContextValue = {
    user,
    loading,
    login,
    logout,
    isAuthenticated: !!user
  };
}
```

```

};

return (
  <AuthContext.Provider value={authContextValue}>
    {children}
  </AuthContext.Provider>
);
};

```

### 3.2.4 Bibliothèques UI Complémentaires

Pour enrichir l'interface utilisateur, plusieurs bibliothèques complémentaires sont utilisées :

- **React Native Elements** : Composants UI pré-stylisés et personnalisables
- **React Native Vector Icons** : Collection d'icônes vectorielles
- **React Native Gesture Handler** : Gestion avancée des gestes tactiles
- **React Native Reanimated** : Animations fluides et performantes

## 3.3 Technologies Backend

### 3.3.1 Node.js et Express

Le backend est développé avec Node.js et Express, offrant :

- **Architecture événementielle non-bloquante** : Idéale pour les applications en temps réel
- **Performance** : Traitement efficace des requêtes simultanées
- **Ecosystème npm** : Accès à un vaste répertoire de packages
- **JavaScript full-stack** : Uniformité du langage entre frontend et backend

```

const express = require('express');
const app = express();
const cors = require('cors');
const morgan = require('morgan');
const helmet = require('helmet');

// Middleware de base
app.use(cors());
app.use(express.json());
app.use(morgan('dev'));
app.use(helmet());

// Routes API
app.use('/api/auth', require('./routes/authRoutes'));
app.use('/api/auctions', require('./routes/auctionRoutes'));
app.use('/api/cars', require('./routes/carRoutes'));
app.use('/api/users', require('./routes/userRoutes'));

// Middleware de gestion d'erreurs
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    message: 'Une erreur est survenue',
  });
});

```

```

        error: process.env.NODE_ENV === 'production' ? {} : err
    });
});

// Démarrage du serveur
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});

```

### 3.3.2 Architecture API RESTful

L'API backend suit les principes REST avec :

- Organisation des endpoints par ressources
- Utilisation appropriée des méthodes HTTP (GET, POST, PUT, DELETE)
- Réponses structurées avec codes HTTP standards
- Pagination des résultats pour les listes volumineuses

```

// Exemple de contrôleur RESTful pour les enchères
const auctionController = {
    // GET /api/auctions
    getAll: async (req, res) => {
        try {
            const page = parseInt(req.query.page) || 1;
            const limit = parseInt(req.query.limit) || 10;
            const skip = (page - 1) * limit;

            const auctions = await Auction.find()
                .sort({ createdAt: -1 })
                .skip(skip)
                .limit(limit)
                .populate('car', 'brand model year images');

            const total = await Auction.countDocuments();

            res.json({
                data: auctions,
                meta: {
                    total,
                    page,
                    lastPage: Math.ceil(total / limit)
                }
            });
        } catch (error) {
            res.status(500).json({ message: error.message });
        }
    },
};

// GET /api/auctions/:id
getById: async (req, res) => {
    try {

```

```

const auction = await Auction.findById(req.params.id)
  .populate('car')
  .populate({
    path: 'bids',
    options: { sort: { amount: -1 } },
    populate: { path: 'user', select: 'username profileImage' }
  });

if (!auction) {
  return res.status(404).json({ message: 'Enchère non trouvée' });
}

res.json(auction);
} catch (error) {
  res.status(500).json({ message: error.message });
}
};

// Autres méthodes...
};

```

### 3.3.3 Sécurité

La sécurité du backend est assurée par plusieurs mécanismes :

- **JSON Web Tokens (JWT)** : Pour l'authentification sécurisée
- **bcrypt** : Pour le hachage sécurisé des mots de passe
- **Helmet** : Protection contre les vulnérabilités web courantes
- **express-rate-limit** : Limitation du nombre de requêtes pour prévenir les attaques par force brute
- **express-validator** : Validation des entrées utilisateur

```

// Middleware d'authentification
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const auth = async (req, res, next) => {
  try {
    const token = req.header('Authorization').replace('Bearer ', '');
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findOne({
      _id: decoded.id,
      'tokens.token': token
    });

    if (!user) {
      throw new Error();
    }

    req.token = token;
    req.user = user;
    next();
  }
};

```

```

    } catch (error) {
      res.status(401).json({ message: 'Veuillez vous authentifier' });
    }
  };
}

```

## 3.4 Communication Temps Réel avec Socket.io

### 3.4.1 Principes de Fonctionnement

Socket.io est utilisé pour établir une communication bidirectionnelle en temps réel entre le client et le serveur, permettant :

- Mise à jour instantanée des enchères
- Notifications en temps réel
- Actualisation du statut des enchères sans rechargement
- Support de la reconnexion automatique

```

// Configuration Socket.io côté serveur
const http = require('http');
const socketIo = require('socket.io');
const jwt = require('jsonwebtoken');

const server = http.createServer(app);
const io = socketIo(server, {
  cors: {
    origin: '*',
    methods: ['GET', 'POST']
  }
});

// Middleware d'authentification Socket.io
io.use((socket, next) => {
  if (socket.handshake.query && socket.handshake.query.token) {
    const token = socket.handshake.query.token;
    jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
      if (err) return next(new Error('Authentication error'));
      socket.decoded = decoded;
      next();
    });
  } else {
    next(new Error('Authentication error'));
  }
});

// Gestion des événements Socket.io
io.on('connection', (socket) => {
  console.log('User connected:', socket.decoded.id);

  // Rejoindre une salle d'enchère spécifique
  socket.on('join_auction', (auctionId) => {
    socket.join(`auction_${auctionId}`);
    console.log(`User joined auction: ${auctionId}`);
  });
});

```

```
});

// Soumettre une enchère
socket.on('place_bid', async (data) => {
  try {
    const { auctionId, amount } = data;
    const userId = socket.decoded.id;

    // Traitement de l'enchère via le service
    const result = await auctionService.placeBid(auctionId, userId, amount);

    if (result.success) {
      // Diffuser la mise à jour à tous les participants
      io.to(`auction_${auctionId}`).emit('bid_update', {
        auctionId,
        newPrice: result.data.amount,
        bidder: result.data.bidderName,
        timestamp: new Date()
      });

      // Notifier l'enchérisseur précédent
      if (result.data.previousBidder) {
        io.to(`user_${result.data.previousBidder}`).emit('outbid', {
          auctionId,
          carName: result.data.carName
        });
      }
    } else {
      // Informer l'utilisateur de l'échec
      socket.emit('bid_error', {
        auctionId,
        message: result.message
      });
    }
  } catch (error) {
    socket.emit('error', { message: error.message });
  }
});

// Quitter une salle d'enchère
socket.on('leave_auction', (auctionId) => {
  socket.leave(`auction_${auctionId}`);
});

// Connecter l'utilisateur à sa salle personnelle pour les notifications
socket.join(`user_${socket.decoded.id}`);

socket.on('disconnect', () => {
  console.log('User disconnected:', socket.decoded.id);
});
```

```
});
```

### 3.4.2 Intégration Côté Client

L'intégration de Socket.io côté client est réalisée comme suit :

```
// Service Socket.io Frontend
import io from 'socket.io-client';
import { API_URL } from '../config';

class SocketService {
    constructor() {
        this.socket = null;
        this.listeners = {};
    }

    initialize(token) {
        if (this.socket) {
            this.disconnect();
        }

        this.socket = io(API_URL, {
            query: { token },
            transports: ['websocket'],
            reconnection: true,
            reconnectionAttempts: 5,
            reconnectionDelay: 1000
        });

        this.socket.on('connect', () => {
            console.log('Socket connected');
        });

        this.socket.on('connect_error', (error) => {
            console.error('Socket connection error:', error);
        });

        this.socket.on('error', (error) => {
            console.error('Socket error:', error);
        });

        // Réattacher les écouteurs après reconnexion
        this.socket.on('reconnect', () => {
            console.log('Socket reconnected');
            Object.keys(this.listeners).forEach(event => {
                this.listeners[event].forEach(callback => {
                    this.socket.on(event, callback);
                });
            });
        });
    }
}
```

```
disconnect() {
    if (this.socket) {
        this.socket.disconnect();
        this.socket = null;
    }
}

// Rejoindre une salle d'enchère
joinAuction(auctionId) {
    if (this.socket) {
        this.socket.emit('join_auction', auctionId);
    }
}

// Quitter une salle d'enchère
leaveAuction(auctionId) {
    if (this.socket) {
        this.socket.emit('leave_auction', auctionId);
    }
}

// Placer une enchère
placeBid(auctionId, amount) {
    if (this.socket) {
        this.socket.emit('place_bid', { auctionId, amount });
    }
}

// Gestion des écouteurs d'événements
on(event, callback) {
    if (this.socket) {
        this.socket.on(event, callback);

        // Enregistrer l'écouteur pour réattachement après reconnexion
        if (!this.listeners[event]) {
            this.listeners[event] = [];
        }
        this.listeners[event].push(callback);
    }
}

off(event, callback) {
    if (this.socket) {
        this.socket.off(event, callback);

        // Nettoyer l'écouteur de la liste
        if (this.listeners[event]) {
            this.listeners[event] = this.listeners[event]
                .filter(cb => cb !== callback);
        }
    }
}
```

```

        }
    }
}

export default new SocketService();

```

## 3.5 Base de Données MongoDB

### 3.5.1 Choix de MongoDB

MongoDB a été sélectionné comme système de gestion de base de données pour plusieurs raisons :

- **Schéma flexible** : Adapté à l'évolution des modèles de données
- **Format JSON natif** : Compatibilité naturelle avec JavaScript
- **Performances** : Bonnes performances en lecture et écriture
- **Scalabilité** : Facilité de mise à l'échelle horizontale
- **Support des requêtes géospatiales** : Utile pour les recherches basées sur la localisation

### 3.5.2 Modélisation des Données

La modélisation des données avec Mongoose (ODM pour MongoDB) s'articule autour de plusieurs schémas principaux :

```

// Schéma Utilisateur
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    trim: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    validate: {
      validator: function(v) {
        return /^[^@\s+\.\.\s+$/.test(v);
      },
      message: props => `${props.value} n'est pas un email valide!`
    }
  },
  password: {
    type: String,
    required: true,
    minlength: 8
  },
  profileImage: String,
  role: {
    type: String,

```

```
enum: ['user', 'admin'],
default: 'user'
},
createdAt: {
  type: Date,
  default: Date.now
},
lastLogin: Date,
tokens: [
  token: {
    type: String,
    required: true
  }
]
},
{
  timestamps: true
});

// Schéma Voiture
const carSchema = new mongoose.Schema({
  brand: {
    type: String,
    required: true,
    trim: true
  },
  model: {
    type: String,
    required: true,
    trim: true
  },
  year: {
    type: Number,
    required: true
  },
  mileage: {
    type: Number,
    required: true
  },
  color: String,
  fuel: {
    type: String,
    enum: ['essence', 'diesel', 'électrique', 'hybride', 'gpl']
  },
  transmission: {
    type: String,
    enum: ['manuelle', 'automatique']
  },
  description: {
    type: String,
    required: true
  }
});
```

```
},
condition: {
  type: String,
  enum: ['neuf', 'excellent', 'bon', 'moyen', 'à restaurer']
},
images: [String],
features: [String],
addedAt: {
  type: Date,
  default: Date.now
}
},
{
  timestamps: true
});

// Schéma Enchère
const auctionSchema = new mongoose.Schema({
  car: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Car',
    required: true
  },
  startPrice: {
    type: Number,
    required: true,
    min: 0
  },
  currentPrice: {
    type: Number,
    required: true,
    min: 0
  },
  minIncrement: {
    type: Number,
    required: true,
    default: 100,
    min: 1
  },
  startTime: {
    type: Date,
    required: true
  },
  endTime: {
    type: Date,
    required: true,
    validate: [
      validator: function(v) {
        return v > this.startTime;
      },
      message: 'La date de fin doit être postérieure à la date de début!'
    ]
  }
});
```

```
        }
    },
    status: {
        type: String,
        enum: ['pending', 'active', 'ended', 'cancelled'],
        default: 'pending'
    },
    winner: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    bids: [{

        type: mongoose.Schema.Types.ObjectId,
        ref: 'Bid'
    }],
    createdBy: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    }
}, {
    timestamps: true
});

// Schéma Offre
const bidSchema = new mongoose.Schema({
    auction: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Auction',
        required: true
    },
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    amount: {
        type: Number,
        required: true,
        min: 0
    },
    timestamp: {
        type: Date,
        default: Date.now
    }
}, {
    timestamps: true
});
```

### 3.5.3 Optimisation des Performances

Pour optimiser les performances de la base de données, plusieurs stratégies ont été mises en œuvre :

- **Indexation** : Création d'index sur les champs fréquemment recherchés
- **Dénormalisation contrôlée** : Stockage de données redondantes stratégiques pour réduire les jointures
- **Pagination** : Limitation du nombre de résultats par requête
- **Projection** : Sélection des champs nécessaires uniquement

```
// Ajout d'index pour optimiser les requêtes
userSchema.index({ username: 1 });
userSchema.index({ email: 1 });
carSchema.index({ brand: 1, model: 1 });
carSchema.index({ year: -1 });
auctionSchema.index({ status: 1 });
auctionSchema.index({ endTime: 1 });
auctionSchema.index({ car: 1 });
bidSchema.index({ auction: 1, amount: -1 });
bidSchema.index({ user: 1 });
```

## 3.6 Outils de Développement et Déploiement

### 3.6.1 Environnement de Développement

Le développement s'appuie sur plusieurs outils :

- **Git** : Contrôle de version distribuée
- **ESLint** : Analyse statique du code JavaScript
- **Prettier** : Formatage automatique du code
- **Jest** : Framework de tests pour JavaScript
- **Postman** : Tests d'API REST
- **React Native Debugger** : Débogage de l'application React Native

### 3.6.2 CI/CD et Déploiement

Le pipeline d'intégration continue et de déploiement continu utilise :

- **GitHub Actions** : Automatisation des workflows de CI/CD
- **Docker** : Conteneurisation des applications
- **Kubernetes** : Orchestration des conteneurs
- **MongoDB Atlas** : Base de données MongoDB gérée
- **AWS/Google Cloud** : Infrastructure cloud

### 3.6.3 Monitoring et Logging

La surveillance de l'application en production s'appuie sur :

- **New Relic** : Monitoring des performances applicatives
- **Sentry** : Suivi des erreurs en temps réel
- **ELK Stack** : Collecte et analyse des logs
- **Prometheus** : Métriques système et applicatives
- **Grafana** : Visualisation des métriques de performance

## 3.7 Choix Technologiques et Alternatives

### 3.7.1 Justification des Choix

Les choix technologiques ont été guidés par plusieurs critères :

- **Performance** : Capacité à gérer efficacement les communications en temps réel
- **Évolutivité** : Facilité d'adaptation à l'évolution des besoins
- **Maturité** : Technologies éprouvées avec communautés actives
- **Cohérence** : Écosystème JavaScript homogène (Node.js, React Native)
- **Productivité** : Outils et frameworks accélérant le développement

### 3.7.2 Alternatives Considérées

Plusieurs alternatives ont été évaluées avant la sélection finale :

- **Frontend** : Flutter vs React Native
- **Backend** : Django/Python vs Node.js/Express
- **Base de données** : PostgreSQL vs MongoDB
- **Temps réel** : Firebase Realtime Database vs Socket.io

Domaine	Solution retenue	Alternative	Raison du choix
Frontend mobile	React Native	Flutter	Expertise équipe, écosystème JS
Backend	Node.js/Express	Django/Python	Performances temps réel, cohérence JS
Base de données	MongoDB	PostgreSQL	Flexibilité du schéma, scaling horizontal
Communication	Socket.io	Firebase	Contrôle fin, coûts prévisibles

TABLE 3.1 – Comparaison des technologies considérées

# Chapitre 4

## Manuel d'Utilisation

### 4.1 Introduction à l'Application

Ce chapitre présente un guide détaillé d'utilisation de l'application d'enchères automobiles, destiné aux différents utilisateurs du système. L'application permet de participer à des enchères de véhicules en temps réel, de suivre les enchères en cours, et de gérer son profil utilisateur.

#### 4.1.1 Public Cible

L'application s'adresse à deux catégories principales d'utilisateurs :

**Utilisateurs Standard** Personnes intéressées par l'achat de véhicules via le système d'enchères.

Ces utilisateurs peuvent consulter les enchères disponibles, placer des enchères, et suivre leurs activités.

**Administrateurs** Gestionnaires de la plateforme responsables de la configuration des enchères, de l'ajout des véhicules, et de la gestion des utilisateurs.

#### 4.1.2 Prérequis Techniques

Pour utiliser l'application, les conditions suivantes sont nécessaires :

- Un smartphone sous iOS (version 12 ou supérieure) ou Android (version 8 ou supérieure)
- Une connexion internet stable (Wi-Fi ou données mobiles)
- Un espace de stockage minimal de 100 Mo sur l'appareil
- Un compte email valide pour l'inscription

### 4.2 Installation et Configuration

#### 4.2.1 Téléchargement de l'Application

L'application peut être téléchargée depuis les plateformes officielles :

- **iOS** : App Store
- **Android** : Google Play Store

La recherche peut être effectuée avec les mots-clés "Car Auction App" ou en scannant le QR code disponible sur le site web officiel.

#### 4.2.2 Crédit de Compte

Pour utiliser l'application, un compte utilisateur est nécessaire :

1. Ouvrir l'application après installation
2. Sélectionner l'option "S'inscrire"
3. Remplir le formulaire d'inscription avec :

- Nom d'utilisateur
- Adresse email
- Mot de passe (au moins 8 caractères, incluant lettres, chiffres et caractères spéciaux)
- Informations de profil optionnelles

4. Accepter les conditions d'utilisation
5. Cliquer sur "Créer un compte"
6. Vérifier l'email de confirmation reçu et suivre les instructions

### 4.2.3 Connexion

Une fois le compte créé, la connexion s'effectue comme suit :

1. Ouvrir l'application
2. Saisir l'email et le mot de passe
3. Sélectionner "Se souvenir de moi" (optionnel)
4. Appuyer sur "Se connecter"

En cas d'oubli du mot de passe, l'option "Mot de passe oublié" permet de recevoir un email de réinitialisation.

## 4.3 Interface Utilisateur

### 4.3.1 Navigation Principale

L'interface de l'application est organisée autour de cinq sections principales accessibles via la barre de navigation inférieure :

**Accueil** Affiche les enchères en vedette et les annonces importantes

**Enchères** Liste toutes les enchères disponibles avec options de filtrage

**Mes Enchères** Suivi des enchères auxquelles l'utilisateur participe

**Notifications** Centre de notifications pour les alertes et mises à jour

**Profil** Gestion des informations personnelles et préférences

FIGURE 4.1 – Navigation principale de l'application

### 4.3.2 Écran d'Accueil

L'écran d'accueil présente :

- Un carrousel des enchères populaires ou se terminant bientôt
- Des statistiques sur le marché (nombre d'enchères actives, etc.)
- Un accès rapide aux dernières enchères consultées
- Les actualités et annonces de la plateforme

## 4.4 Participation aux Enchères

### 4.4.1 Recherche et Consultation des Enchères

Pour trouver des véhicules :

1. Accéder à l'onglet "Enchères"
2. Utiliser la barre de recherche pour des mots-clés spécifiques
3. Appliquer des filtres selon :
  - Marque et modèle
  - Année de fabrication
  - Prix (fourchette)
  - Kilométrage
  - Type de carburant
  - Statut de l'encheré (à venir, en cours, terminée)
4. Trier les résultats par prix, popularité ou date de fin
5. Sélectionner un véhicule pour afficher sa fiche détaillée

#### 4.4.2 Détails d'une Enchère

La fiche détaillée d'un véhicule comprend :

- Galerie d'images du véhicule
- Informations techniques (marque, modèle, année, kilométrage, etc.)
- Description et état du véhicule
- Historique d'entretien (si disponible)
- Prix actuel et historique des enchères
- Nombre d'enchérisseurs
- Temps restant avant la fin de l'enchère
- Bouton pour placer une enchère

#### 4.4.3 Placement d'une Enchère

Pour participer à une enchère :

1. Sur la fiche détaillée du véhicule, appuyer sur "Placer une enchère"
2. Saisir le montant proposé (doit être supérieur au prix actuel)
3. Vérifier les informations et confirmer
4. Attendre la confirmation que l'enchère a été acceptée

FIGURE 4.2 – Interface de placement d'enchère

#### 4.4.4 Suivi des Enchères

Pour suivre les enchères auxquelles l'utilisateur participe :

1. Accéder à l'onglet "Mes Enchères"
2. Visualiser les enchères actives, remportées et perdues
3. Sélectionner une enchère pour :
  - Consulter son statut actuel
  - Voir l'historique des enchères placées
  - Recevoir des notifications pour cette enchère
  - Placer une nouvelle enchère (si active)

### 4.5 Gestion du Profil Utilisateur

#### 4.5.1 Modification des Informations Personnelles

Pour mettre à jour son profil :

1. Accéder à l'onglet "Profil"
2. Sélectionner "Modifier le profil"
3. Mettre à jour les informations souhaitées :
  - Photo de profil
  - Nom d'utilisateur
  - Coordonnées (téléphone, adresse)
  - Préférences de notification
4. Enregistrer les modifications

#### 4.5.2 Gestion des Notifications

Pour configurer les notifications :

1. Accéder à l'onglet "Profil"

2. Sélectionner "Paramètres de notification"
3. Activer/désactiver les types de notifications :
  - Surenchère sur un véhicule
  - Rappel de fin d'enchère proche
  - Enchère remportée/perdue
  - Nouvelles enchères correspondant aux préférences
  - Annonces système
4. Choisir le mode de notification (push, email, ou les deux)

#### 4.5.3 Historique des Transactions

Pour consulter l'historique :

1. Accéder à l'onglet "Profil"
2. Sélectionner "Historique des transactions"
3. Visualiser :
  - Enchères remportées
  - Paiements effectués
  - Factures et documents associés

### 4.6 Fonctionnalités Administrateur

Cette section concerne uniquement les utilisateurs disposant des droits d'administration.

#### 4.6.1 Gestion des Véhicules

Les administrateurs peuvent gérer les véhicules :

1. Accéder au panneau d'administration
2. Sélectionner "Gestion des véhicules"
3. Pour ajouter un véhicule :
  - Cliquer sur "Ajouter un véhicule"
  - Remplir les informations détaillées
  - Ajouter des images (minimum 3, maximum 10)
  - Enregistrer le véhicule
4. Pour modifier un véhicule :
  - Rechercher et sélectionner le véhicule
  - Modifier les informations nécessaires
  - Enregistrer les modifications
5. Pour supprimer un véhicule :
  - Rechercher et sélectionner le véhicule
  - Confirmer la suppression

#### 4.6.2 Crédation et Gestion des Enchères

Pour gérer les enchères :

1. Accéder au panneau d'administration
2. Sélectionner "Gestion des enchères"
3. Pour créer une enchère :
  - Cliquer sur "Créer une enchère"
  - Sélectionner le véhicule concerné
  - Définir le prix de départ
  - Configurer la date et l'heure de début/fin

- Définir les incrémentations minimales
  - Activer/désactiver les fonctionnalités spéciales
  - Publier l'enchère
4. Pour gérer une enchère existante :
- Rechercher et sélectionner l'enchère
  - Modifier les paramètres (si l'enchère n'a pas commencé)
  - Surveiller l'activité en temps réel
  - Intervenir en cas de problème (suspension, annulation)

#### 4.6.3 Gestion des Utilisateurs

Pour administrer les comptes utilisateurs :

1. Accéder au panneau d'administration
2. Sélectionner "Gestion des utilisateurs"
3. Rechercher un utilisateur par nom, email ou ID
4. Visualiser les informations complètes du compte
5. Actions possibles :
  - Modifier les détails du compte
  - Réinitialiser le mot de passe
  - Suspender/réactiver un compte
  - Attribuer des droits d'administration
  - Consulter l'historique d'activité

### 4.7 Résolution des Problèmes Courants

#### 4.7.1 Problèmes de Connexion

- **Identifiants refusés** : Vérifier l'orthographe, utiliser "Mot de passe oublié"
- **Compte bloqué** : Contacter le support après plusieurs tentatives échouées
- **Erreur de connexion** : Vérifier la connexion internet, redémarrer l'application

#### 4.7.2 Problèmes avec les Enchères

- **Enchère refusée** : Vérifier que le montant est supérieur au prix actuel et respecte l'incrément minimal
- **Enchère non enregistrée** : Rafraîchir la page, vérifier la connexion
- **Retard dans les mises à jour** : Fermer et rouvrir l'application pour rétablir la connexion Socket.io

#### 4.7.3 Problèmes Techniques Généraux

- **Application lente** : Libérer de l'espace sur l'appareil, fermer les applications en arrière-plan
- **Plantages** : Mettre à jour l'application, redémarrer l'appareil
- **Images non chargées** : Vérifier la connexion, patienter ou rafraîchir

### 4.8 Support et Assistance

#### 4.8.1 Centre d'Aide

Un centre d'aide est accessible depuis l'application :

1. Accéder à l'onglet "Profil"
2. Sélectionner "Aide et support"

3. Consulter les articles d'aide classés par catégorie
4. Utiliser la recherche pour trouver des réponses spécifiques

#### 4.8.2 Contact du Support

Pour contacter l'équipe d'assistance :

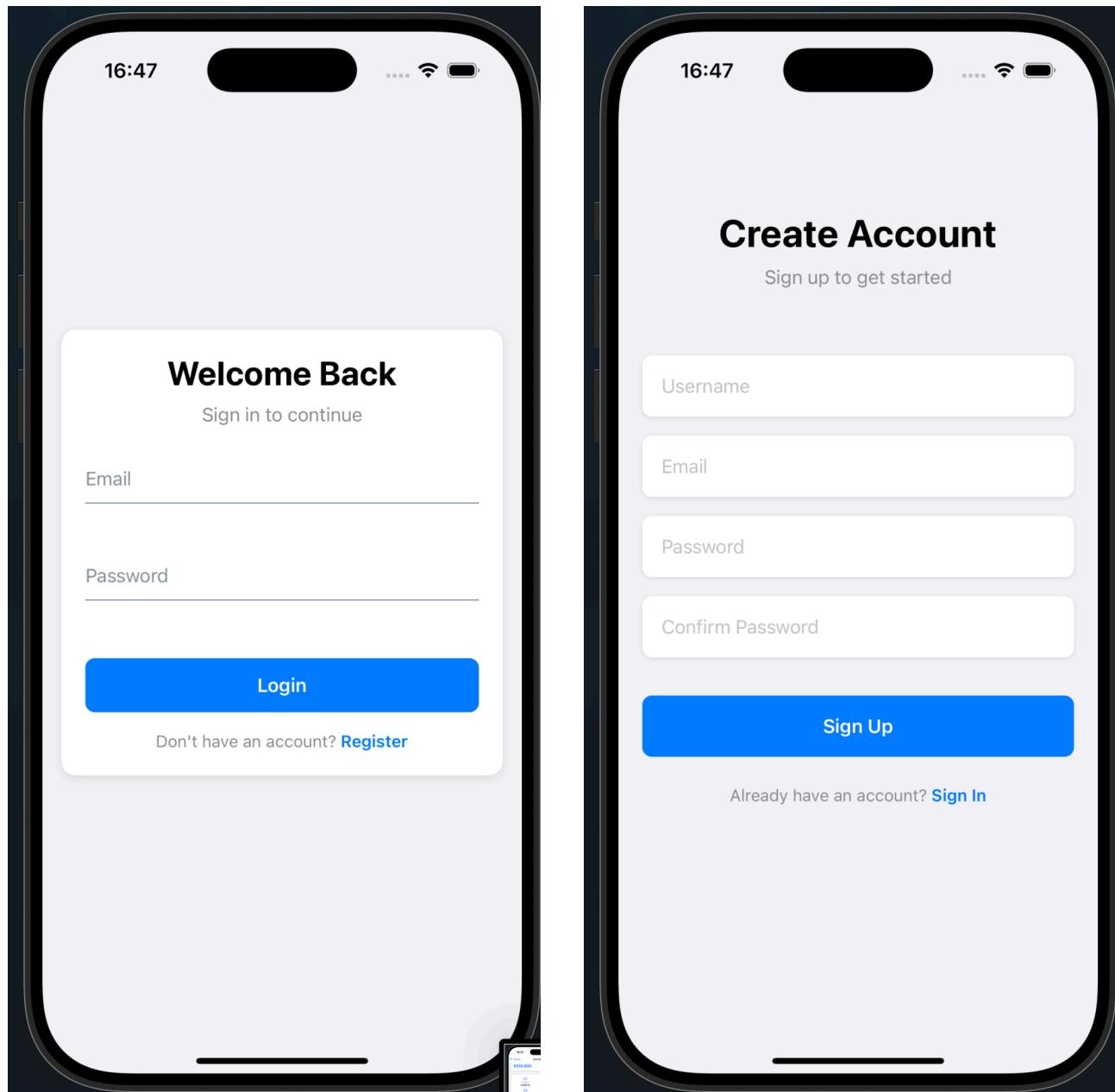
1. Accéder à l'onglet "Profil"
2. Sélectionner "Aide et support"
3. Choisir "Contacter le support"
4. Sélectionner le type de problème
5. Décrire le problème rencontré
6. Joindre des captures d'écran si nécessaire
7. Soumettre la demande

Le délai de réponse habituel est de 24 à 48 heures ouvrables.

### 4.9 Interface de l'Application

L'application de vente aux enchères d'automobiles offre une interface utilisateur intuitive et conviviale, conçue pour faciliter la participation aux enchères automobiles. Voici les principales fonctionnalités et écrans de l'application :

#### 4.9.1 Authentification et Inscription



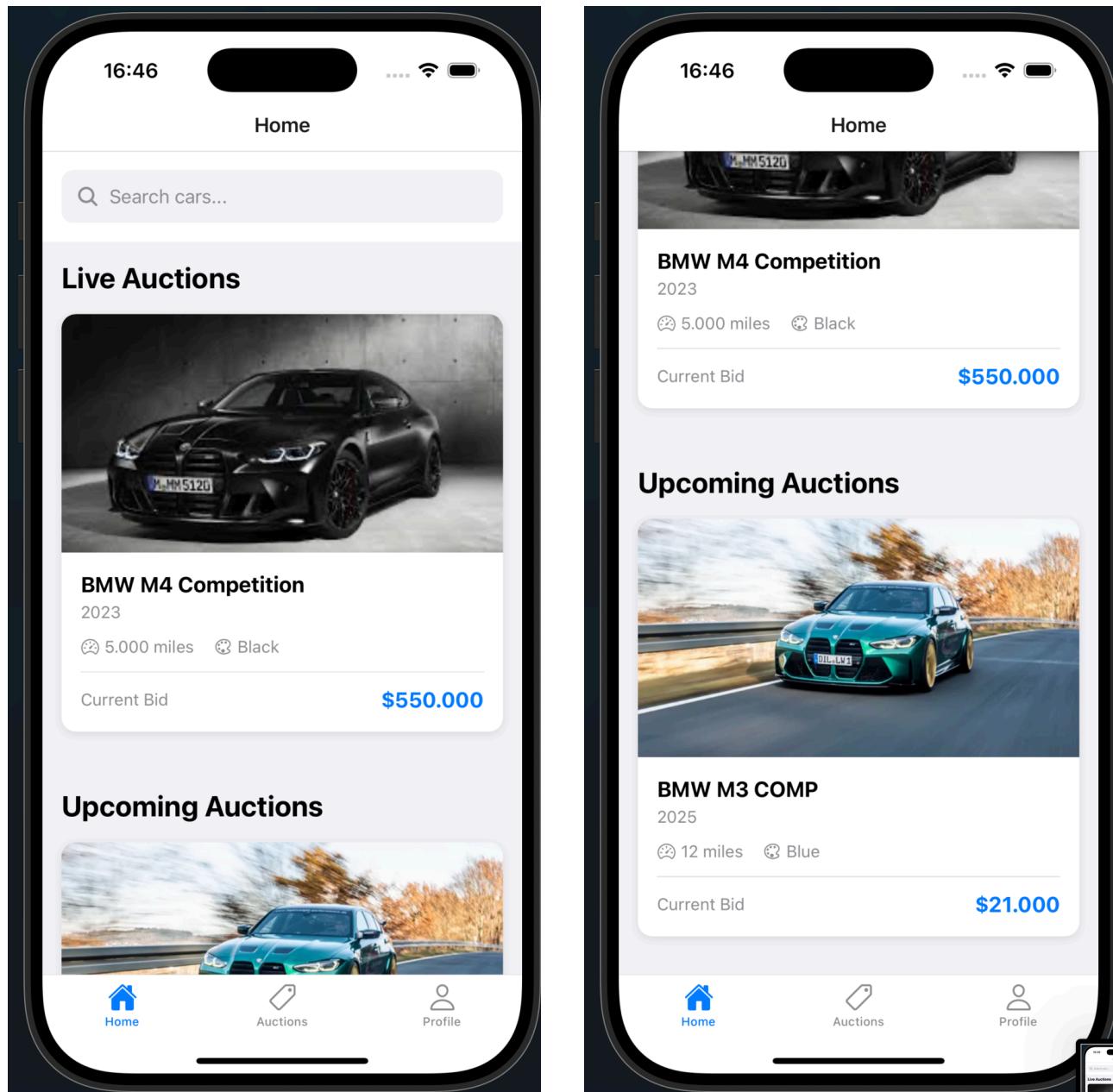
(a) Écran de connexion

(b) Écran d'inscription

FIGURE 4.3 – Écrans d'authentification de l'application

L'utilisateur peut accéder à l'application via la page de connexion (Figure 4.3a) ou créer un nouveau compte via la page d'inscription (Figure 4.3b). Ces écrans permettent une entrée sécurisée dans le système.

#### 4.9.2 Page d'Accueil et Enchères



(a) Enchères en cours

(b) Enchères à venir

FIGURE 4.4 – Pages d'accueil avec différentes vues des enchères

La page d'accueil présente les enchères en cours (Figure 4.4a) et les enchères à venir (Figure 4.4b), permettant aux utilisateurs de parcourir facilement les véhicules disponibles. On y trouve des informations essentielles comme la marque, le modèle, l'année, le kilométrage et le prix actuel de l'enchère.

### 4.9.3 Gestion des Enchères

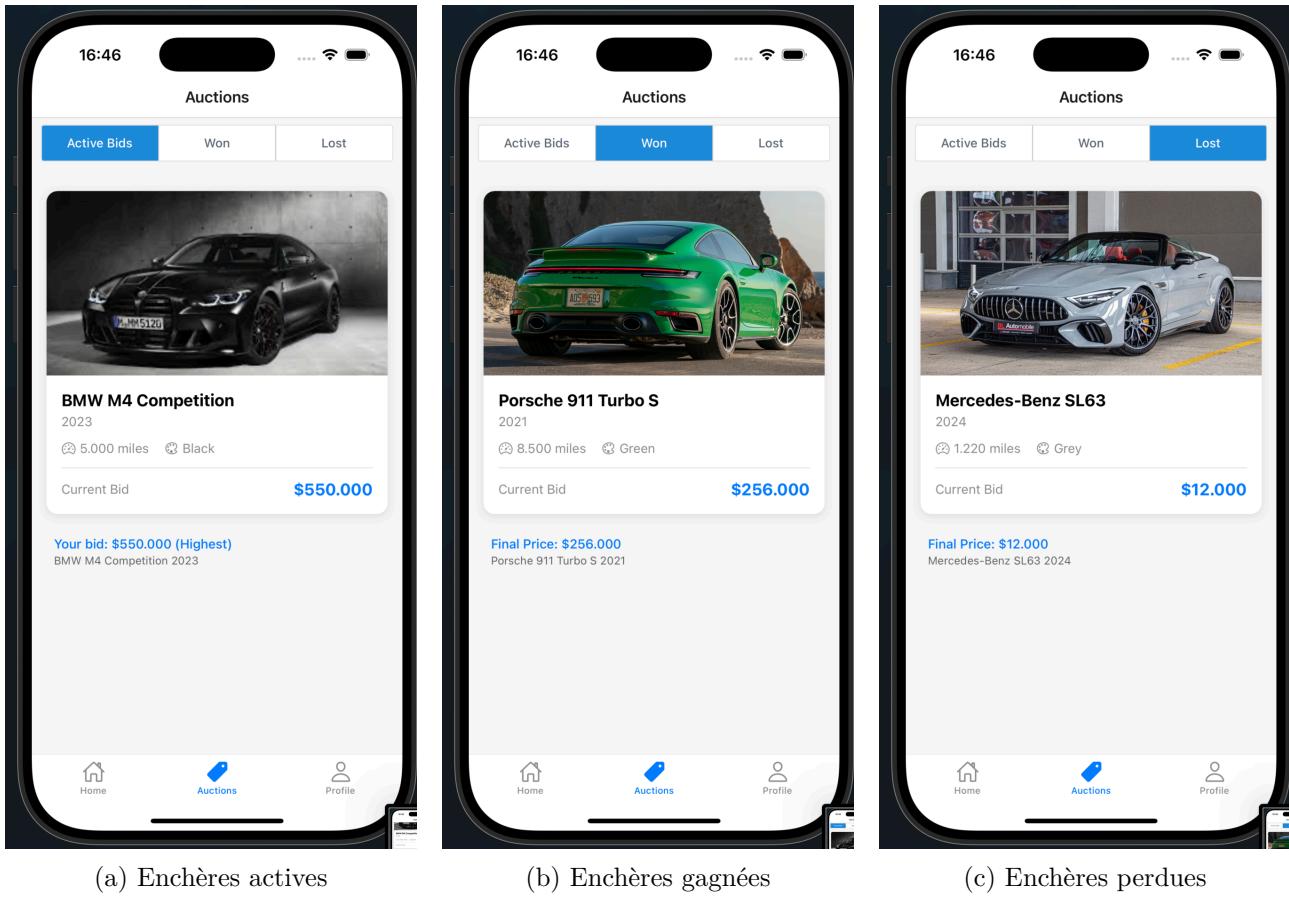


FIGURE 4.5 – Écrans de gestion des enchères

Les utilisateurs peuvent suivre leurs enchères via différents onglets : enchères actives (Figure 4.5a), enchères gagnées (Figure 4.5b) et enchères perdues (Figure 4.5c).

#### 4.9.4 Profil Utilisateur

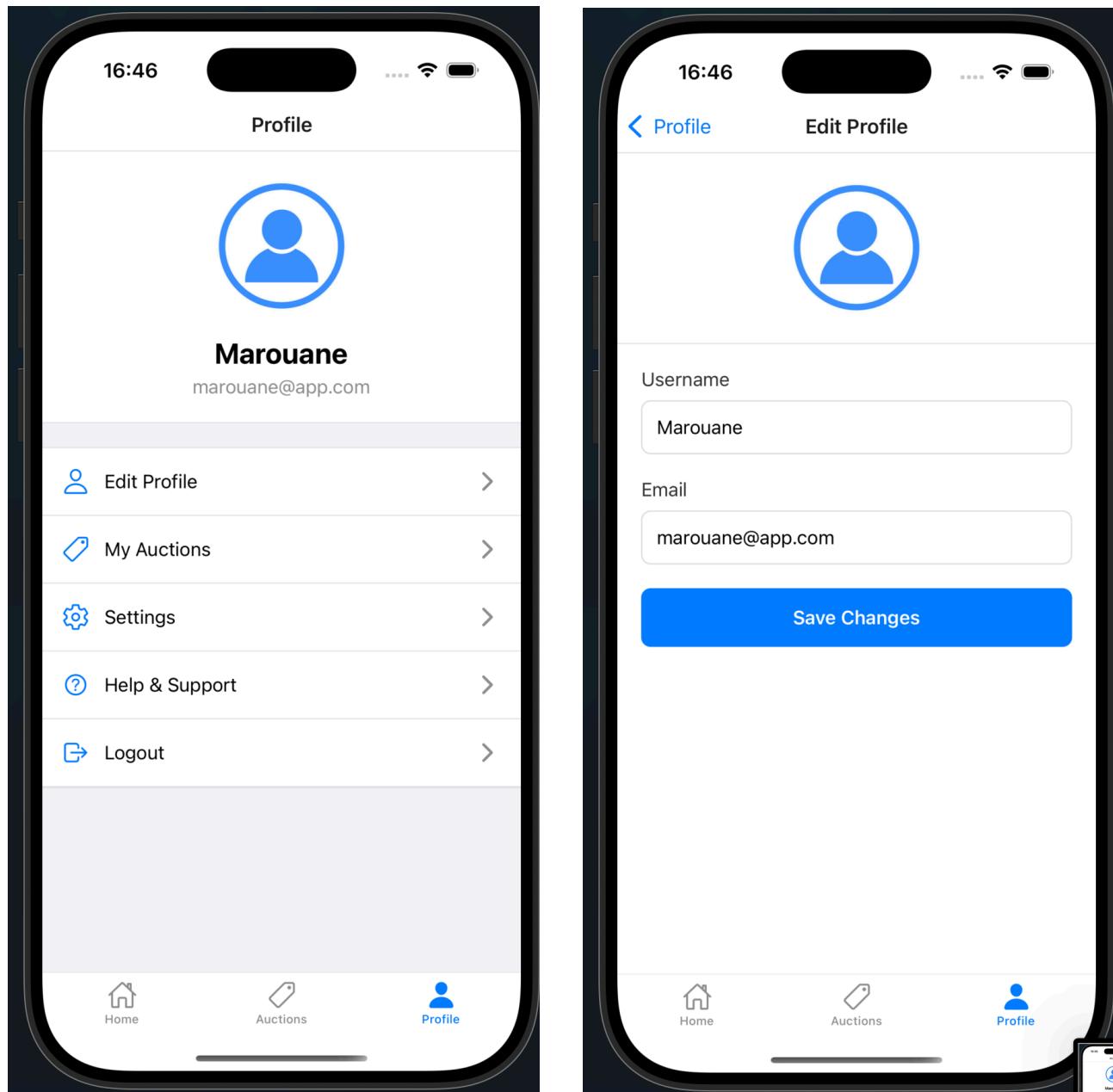


FIGURE 4.6 – Gestion du profil utilisateur

L'écran de profil (Figure 4.6a) permet d'accéder aux informations personnelles de l'utilisateur et à l'historique de ses enchères. L'écran de modification du profil (Figure 4.6b) permet de mettre à jour les informations personnelles.

#### 4.9.5 Détails du Véhicule et Enchère

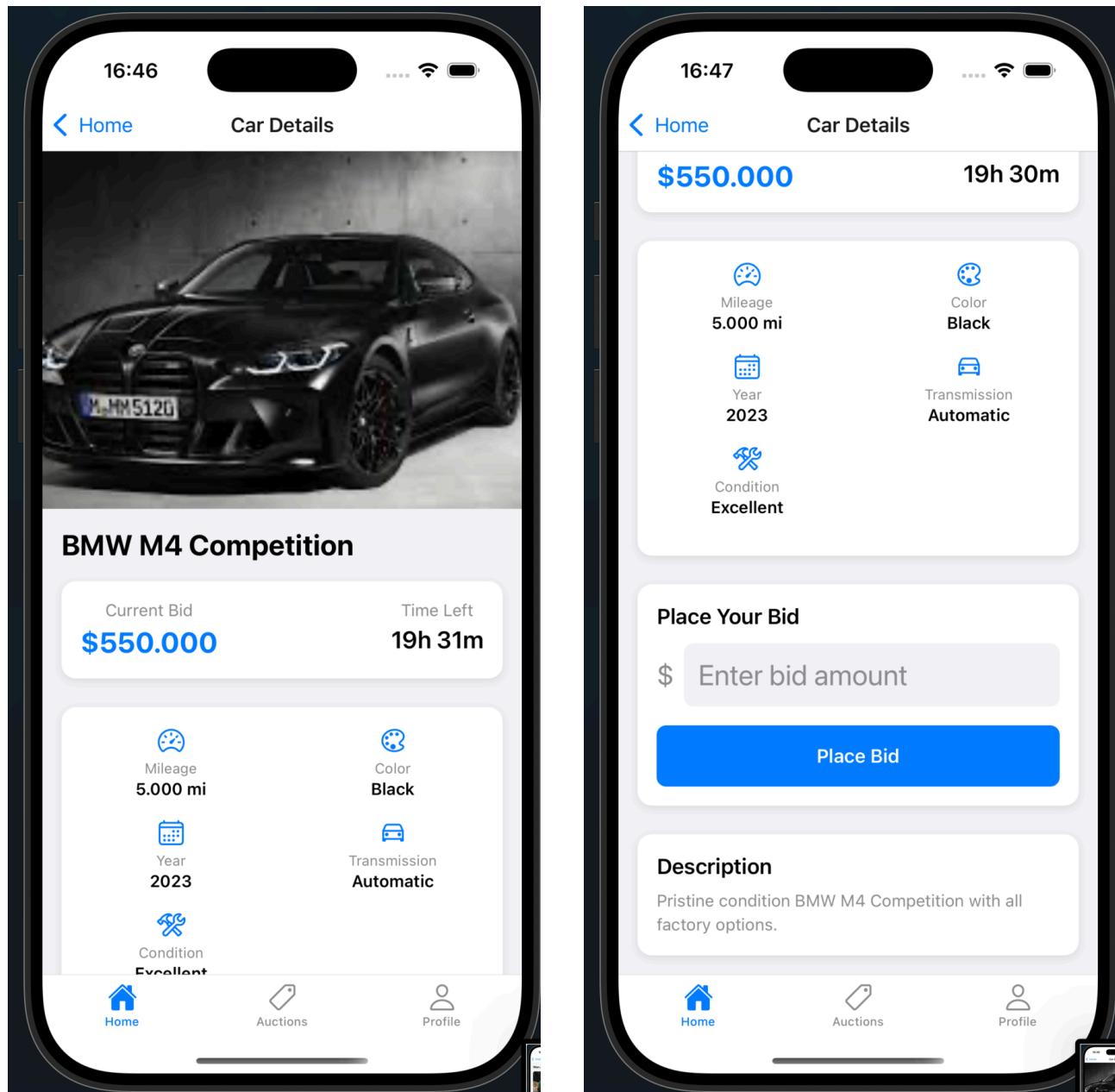


FIGURE 4.7 – Détails du véhicule et système d'enchères

Les écrans de détails du véhicule (Figures 4.7a et 4.7b) présentent toutes les informations pertinentes sur le véhicule et permettent de placer une enchère directement depuis l'application.

#### 4.9.6 Historique des Enchères

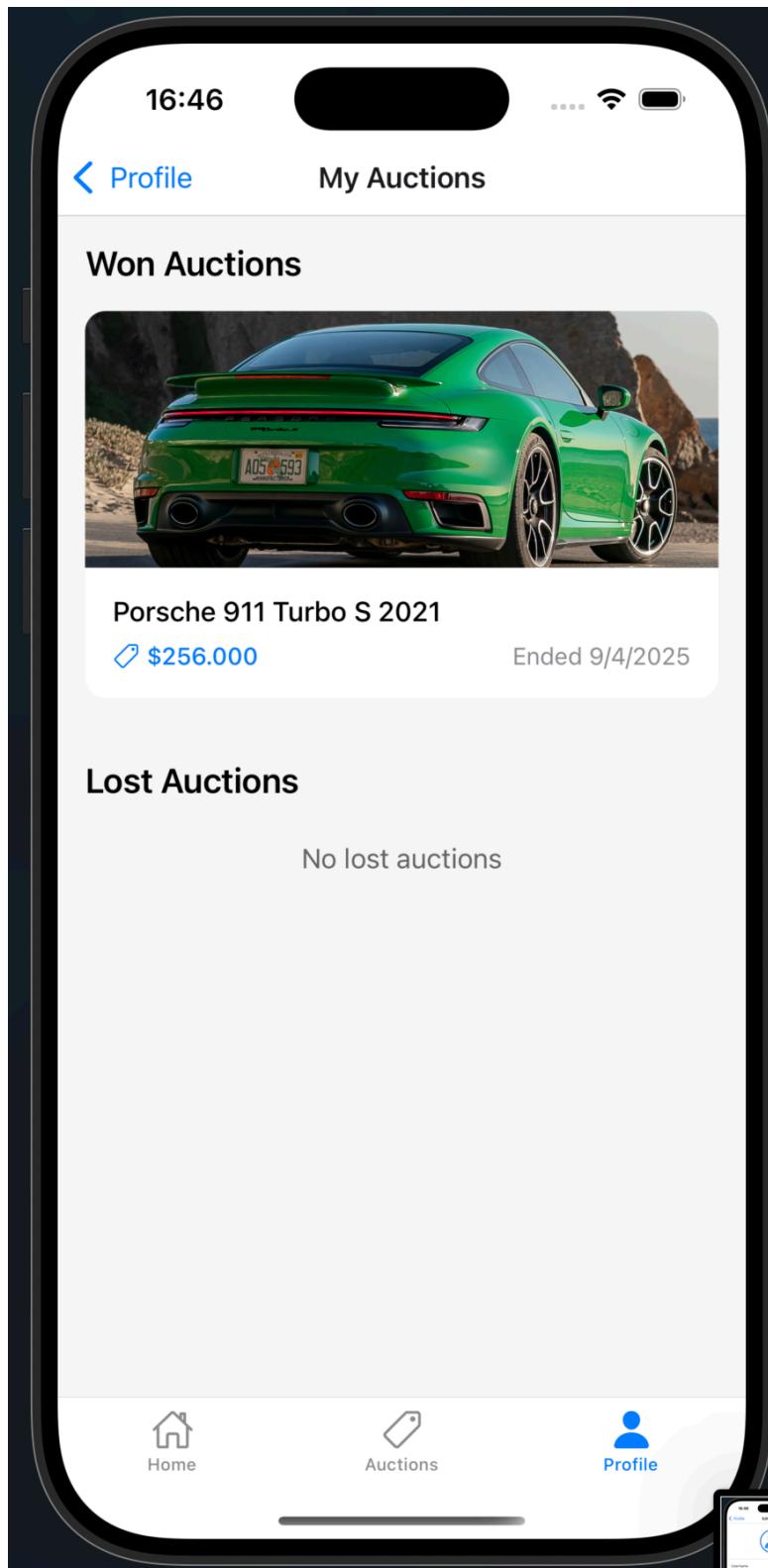


FIGURE 4.8 – Historique des enchères de l'utilisateur

L'écran d'historique des enchères (Figure 4.8) permet à l'utilisateur de consulter l'ensemble de ses activités d'enchères passées, incluant les véhicules sur lesquels il a enchéri, les montants proposés et les résultats obtenus.

## 4.10 Fonctionnalités Principales

- **Inscription et connexion** : Création de compte et authentification sécurisée
- **Navigation des enchères** : Parcourir les enchères en cours et à venir
- **Participation aux enchères** : Placer des enchères sur les véhicules désirés
- **Suivi des enchères** : Consulter l'état de vos enchères actives, gagnées et perdues
- **Gestion du profil** : Mettre à jour vos informations personnelles
- **Notifications** : Recevoir des alertes en temps réel sur les enchères suivies

## 4.11 Conseils d'Utilisation

- Configurez les notifications pour être informé des changements d'enchères
- Vérifiez régulièrement l'onglet "Enchères actives" pour suivre vos enchères en cours
- Complétez intégralement votre profil pour faciliter les transactions
- Consultez les détails complets du véhicule avant de placer une enchère
- Définissez un budget maximum pour chaque enchère pour éviter les dépassements

# **Chapitre 5**

## **Analyse de l'Existant**

### **5.1 Solutions d'Enchères Existantes**

#### **5.1.1 Solutions Traditionnelles**

Les solutions traditionnelles d'enches automobiles presentent plusieurs caracteristiques :

- Enches physiques necessitant une presence sur place
- Documentation papier et processus manuel
- Limitation geographique des participants
- Delais importants dans le traitement des enches

#### **5.1.2 Solutions Numériques Actuelles**

Plusieurs plateformes numeriques existent sur le marche :

- Plateformes web uniquement, sans application mobile native
- Applications avec fonctionnalites limitees
- Systèmes sans mise à jour en temps réel
- Interfaces utilisateur complexes

### **5.2 Limitations des Solutions Existantes**

#### **5.2.1 Problèmes Techniques**

- Absence de notifications en temps réel
- Manque d'intégration mobile native
- Gestion manuelle des statuts d'enches
- Problèmes de performance avec les mises à jour

#### **5.2.2 Problèmes Fonctionnels**

- Interface administrateur limitée
- Absence de suivi des enches perdues
- Processus d'enche non optimisé
- Manque de transparence dans les résultats

### **5.3 Opportunités d'Amélioration**

#### **5.3.1 Aspects Techniques**

- Implémentation de Socket.IO pour les mises à jour en temps réel
- Développement d'une application mobile native avec React Native
- Automatisation de la gestion des statuts d'enches
- Optimisation des performances serveur avec Node.js

#### **5.3.2 Aspects Fonctionnels**

- Interface utilisateur intuitive et moderne

- Système complet de notifications
- Tableau de bord administratif avancé
- Historique détaillé des enchères

#### 5.4 Analyse des Risques

- Gestion des conflits de ports (3000, 5001, 8081)
- Synchronisation des données en temps réel
- Sécurité des transactions
- Performance sous charge importante

# Chapitre 6

## Analyse des Besoins

### 6.1 Méthodologie d'Analyse

L'analyse des besoins constitue une étape fondamentale dans le développement de notre système d'enchères automobiles. Cette analyse a été menée selon une approche structurée visant à identifier avec précision les exigences fonctionnelles et non fonctionnelles du système.

#### 6.1.1 Collecte des Besoins

La collecte des besoins a été réalisée à travers plusieurs méthodes complémentaires :

- **Entretiens avec les parties prenantes** : Sessions de discussion avec des acheteurs et vendeurs potentiels, des experts du marché automobile, et des professionnels du développement d'applications
- **Analyse concurrentielle** : Étude approfondie des plateformes d'enchères existantes pour identifier les meilleures pratiques et les opportunités d'innovation
- **Ateliers de brainstorming** : Sessions collaboratives pour générer des idées de fonctionnalités et identifier les priorités
- **Questionnaires** : Sondages ciblés pour recueillir les attentes des utilisateurs potentiels
- **Analyse des tendances du marché** : Étude des évolutions récentes dans le domaine des enchères en ligne et du commerce électronique automobile

#### 6.1.2 Priorisation des Besoins

Les besoins identifiés ont été priorisés selon la méthode MoSCoW :

**Must Have** Fonctionnalités essentielles sans lesquelles le système ne peut pas fonctionner correctement

**Should Have** Fonctionnalités importantes mais non critiques pour le lancement initial

**Could Have** Fonctionnalités souhaitables qui apportent une valeur ajoutée mais peuvent être reportées

**Won't Have** Fonctionnalités intéressantes mais explicitement exclues de la version actuelle

Cette priorisation a permis de définir clairement le périmètre du projet et d'établir une feuille de route pour les développements futurs.

## 6.2 Exigences Fonctionnelles

Les exigences fonctionnelles définissent les comportements spécifiques et les fonctionnalités que le système doit offrir.

#### 6.2.1 Gestion des Utilisateurs

##### 1. Inscription des utilisateurs (Must Have)

- Le système doit permettre la création de nouveaux comptes utilisateurs
- Les informations minimales requises incluent : nom d'utilisateur, email, mot de passe
- Le système doit vérifier l'unicité des emails et noms d'utilisateur

- Une confirmation par email doit être envoyée lors de l'inscription

## 2. Authentification (Must Have)

- Les utilisateurs doivent pouvoir se connecter avec leur email et mot de passe
- Le système doit offrir une fonction de récupération de mot de passe
- L'authentification doit être sécurisée par des tokens JWT
- Les sessions doivent expirer après une période d'inactivité

## 3. Gestion de profil (Should Have)

- Les utilisateurs doivent pouvoir modifier leurs informations personnelles
- Le téléchargement d'une photo de profil doit être possible
- L'historique des activités (enchères placées, remportées) doit être consultable
- Les préférences de notification doivent être configurables

## 4. Rôles et permissions (Must Have)

- Le système doit distinguer au moins deux types d'utilisateurs : standard et administrateur
- Les administrateurs doivent avoir accès à des fonctionnalités de gestion avancées
- Les permissions doivent être vérifiées pour toutes les actions sensibles

### 6.2.2 Gestion des Véhicules

#### 1. Ajout de véhicules (Must Have)

- Les administrateurs doivent pouvoir ajouter de nouveaux véhicules au système
- Les informations à saisir incluent : marque, modèle, année, kilométrage, couleur, etc.
- Le téléchargement de multiples images doit être supporté
- Une description détaillée du véhicule doit être possible

#### 2. Modification des véhicules (Should Have)

- Les informations des véhicules doivent être modifiables tant qu'ils ne sont pas en enchère
- L'ajout, la suppression et la réorganisation des images doivent être possibles
- Un historique des modifications doit être conservé

#### 3. Suppression des véhicules (Should Have)

- Les véhicules sans enchère active doivent pouvoir être supprimés
- Une confirmation doit être demandée avant suppression
- La suppression doit être logique plutôt que physique pour préserver l'historique

#### 4. Visualisation des véhicules (Must Have)

- Les utilisateurs doivent pouvoir consulter les détails complets des véhicules
- Une galerie d'images avec fonction de zoom doit être disponible
- Les caractéristiques techniques doivent être présentées de manière structurée
- L'historique d'entretien, si disponible, doit être accessible

### 6.2.3 Gestion des Enchères

#### 1. Création d'encheres (Must Have)

- Les administrateurs doivent pouvoir créer de nouvelles enchères
- Chaque enchère doit être associée à un véhicule spécifique
- Les paramètres à définir incluent : prix de départ, incrément minimal, dates de début et fin
- Des règles spécifiques (extension automatique, prix de réserve) doivent être configurables

#### 2. Participation aux enchères (Must Have)

- Les utilisateurs authentifiés doivent pouvoir placer des enchères
- Le montant proposé doit être supérieur au prix actuel plus l'incrément minimal
- L'utilisateur doit recevoir une confirmation immédiate de son enchère

- L'historique des enchères doit être visible pour tous les participants

### 3. Suivi des enchères (Must Have)

- Les utilisateurs doivent pouvoir suivre l'évolution des enchères en temps réel
- Un compte à rebours doit indiquer le temps restant
- Les notifications doivent être envoyées en cas de surenchère
- Les enchères terminées doivent être clairement identifiées avec leur résultat

### 4. Finalisation des enchères (Should Have)

- Les enchères doivent se terminer automatiquement à la date et heure définies
- Le gagnant doit être déterminé et notifié automatiquement
- Un récapitulatif doit être généré pour le vendeur et l'acheteur
- Les enchères sans offre valide doivent être marquées comme non attribuées

## 6.2.4 Notifications et Communication

### 1. Notifications en temps réel (Must Have)

- Les utilisateurs doivent recevoir des notifications push pour les événements importants
- Les notifications doivent inclure : surenchères, fin imminente, enchère remportée/perdue
- Les notifications doivent être personnalisables selon les préférences utilisateur

### 2. Alertes par email (Should Have)

- Des emails doivent être envoyés pour les événements majeurs
- Les contenus des emails doivent être personnalisés et professionnels
- Une option de désabonnement doit être disponible

### 3. Centre de notifications (Could Have)

- Un centre de notifications dans l'application doit centraliser tous les messages
- Les notifications doivent pouvoir être marquées comme lues/non lues
- Un historique des notifications doit être conservé

## 6.2.5 Recherche et Filtrage

### 1. Recherche de véhicules (Must Have)

- Les utilisateurs doivent pouvoir rechercher des véhicules par mots-clés
- La recherche doit porter sur les marques, modèles, descriptions, etc.
- Les résultats doivent être pertinents et classés par ordre de pertinence

### 2. Filtrage avancé (Should Have)

- Des filtres multiples doivent être disponibles : marque, modèle, année, prix, etc.
- Les filtres doivent être combinables pour affiner les résultats
- Les valeurs de filtrage doivent être présentées de manière intuitive (sliders, listes déroulantes)

### 3. Tri des résultats (Should Have)

- Les résultats doivent pouvoir être triés selon différents critères
- Les options de tri incluent : prix (croissant/décroissant), popularité, fin prochaine
- Le tri sélectionné doit être mémorisé pendant la session

## 6.2.6 Administration

### 1. Tableau de bord administrateur (Must Have)

- Un tableau de bord doit présenter les statistiques clés du système
- Des graphiques d'activité doivent visualiser les tendances
- Des alertes doivent signaler les situations nécessitant attention

### 2. Gestion des utilisateurs (Should Have)

- Les administrateurs doivent pouvoir consulter, modifier et suspendre des comptes

- Des filtres doivent permettre de rechercher rapidement des utilisateurs
- L'historique d'activité des utilisateurs doit être accessible

### 3. Modération des enchères (Should Have)

- Les administrateurs doivent pouvoir intervenir sur les enchères en cours
- Les options incluent : pause, annulation, extension, modification du prix
- Toute intervention doit être journalisée avec justification

### 4. Rapports et analyses (Could Have)

- Des rapports détaillés doivent être générables sur différentes métriques
- Les rapports peuvent concerter : ventes, utilisateurs, performances, etc.
- L'exportation des données doit être possible en différents formats

## 6.3 Exigences Non Fonctionnelles

Les exigences non fonctionnelles définissent les critères de qualité et les contraintes auxquelles le système doit se conformer.

### 6.3.1 Performance

#### 1. Temps de réponse (Must Have)

- Le temps de chargement initial de l'application ne doit pas excéder 3 secondes
- Les interactions utilisateur doivent recevoir une réponse en moins de 1 seconde
- La mise à jour des enchères en temps réel doit s'effectuer en moins de 500 ms

#### 2. Capacité de charge (Should Have)

- Le système doit supporter simultanément au moins 1000 utilisateurs actifs
- Jusqu'à 100 enchères simultanées doivent être gérables sans dégradation
- Le système doit pouvoir traiter au moins 50 enchères par minute

#### 3. Scalabilité (Should Have)

- L'architecture doit permettre une mise à l'échelle horizontale
- Les pics d'activité doivent être absorbables par allocation dynamique de ressources
- La performance ne doit pas se dégrader significativement avec l'augmentation du volume de données

### 6.3.2 Sécurité

#### 1. Authentification et autorisation (Must Have)

- Les mots de passe doivent être stockés avec un hachage fort (bcrypt)
- L'authentification multi-facteurs doit être disponible
- Les sessions doivent expirer automatiquement après 30 minutes d'inactivité
- Les autorisations doivent être vérifiées à chaque action sensible

#### 2. Protection des données (Must Have)

- Toutes les communications doivent être chiffrées via HTTPS
- Les données sensibles doivent être chiffrées au repos
- Les accès aux données doivent être tracés et auditables
- La conformité RGPD doit être assurée pour les utilisateurs européens

#### 3. Sécurité applicative (Should Have)

- Protection contre les injections SQL et NoSQL
- Défense contre les attaques XSS et CSRF
- Limitation du débit des requêtes pour prévenir les attaques DDoS
- Validation stricte des entrées utilisateur côté serveur

### 6.3.3 Fiabilité

#### 1. Disponibilité (Must Have)

- Le système doit garantir une disponibilité de 99,9% (hors maintenance planifiée)
- Les temps d'arrêt planifiés doivent être annoncés au moins 24h à l'avance
- Un plan de reprise après sinistre doit être en place avec RTO < 4h

#### 2. Robustesse (Should Have)

- Le système doit gérer gracieusement les erreurs côté client et serveur
- Les transactions critiques doivent être protégées contre les défaillances
- La reconnexion automatique doit être implémentée pour les communications temps réel

#### 3. Intégrité des données (Must Have)

- Les transactions impliquant des enchères doivent être atomiques
- Des sauvegardes régulières doivent être effectuées avec RPO < 1h
- La cohérence des données doit être vérifiable par des processus automatisés

### 6.3.4 Utilisabilité

#### 1. Interface utilisateur (Must Have)

- L'interface doit être intuitive et nécessiter un minimum de formation
- La navigation doit être cohérente à travers toute l'application
- Le design doit s'adapter aux différentes tailles d'écran (responsive)

#### 2. Accessibilité (Should Have)

- L'application doit être conforme aux normes WCAG 2.1 niveau AA
- Le contraste des couleurs doit être suffisant pour les utilisateurs malvoyants
- Des alternatives textuelles doivent être fournies pour tous les éléments visuels

#### 3. Internationalisation (Could Have)

- L'interface doit supporter plusieurs langues
- Les formats de date, heure et devise doivent s'adapter aux paramètres régionaux
- Le contenu doit être adaptable aux spécificités culturelles

### 6.3.5 Compatibilité

#### 1. Compatibilité mobile (Must Have)

- L'application doit fonctionner sur iOS 12+ et Android 8+
- L'interface doit s'adapter aux différentes résolutions d'écran
- Les fonctionnalités doivent être cohérentes entre plateformes

#### 2. Compatibilité navigateur (Should Have)

- Une version web progressive doit être compatible avec les navigateurs modernes
- Le support minimum inclut : Chrome 70+, Firefox 60+, Safari 12+, Edge 79+
- Les fonctionnalités dégradées gracieusement selon les capacités du navigateur

#### 3. Intégration API (Could Have)

- Des API publiques doivent être disponibles pour intégration tierce
- Les API doivent être documentées selon les standards OpenAPI
- Des environnements de test et de staging doivent être fournis

## 6.4 Contraintes Techniques

### 1. Infrastructure

- Le système doit être déployable sur des infrastructures cloud (AWS, GCP, Azure)
- Les composants doivent être conteneurisés pour faciliter le déploiement

- L'architecture doit supporter la réPLICATION géographique

## 2. Développement

- Le code source doit être versionné avec Git
- Le développement doit suivre une approche CI/CD
- Des tests automatisés doivent couvrir au moins 80% du code

## 3. Maintenance

- La documentation technique doit être maintenue à jour
- Le code doit suivre des standards de qualité vérifiables automatiquement
- Des logs détaillés doivent être générés pour faciliter le débogage

## 6.5 Scénarios d'Utilisation

Pour illustrer concrètement les besoins identifiés, plusieurs scénarios d'utilisation ont été définis :

### 6.5.1 Scénario 1 : Inscription et Première Enchère

Jean découvre l'application d'enchères automobiles et décide de s'inscrire. Après avoir créé son compte et complété son profil, il parcourt les enchères en cours. Un véhicule attire son attention : une Peugeot 308 de 2018 avec 45,000 km. L'encheré se termine dans 3 heures et le prix actuel est de 12,500€. Jean décide de placer une enchère à 12,700€. Il reçoit une confirmation immédiate et peut voir son nom apparaître comme enchérisseur principal. Une heure plus tard, il reçoit une notification l'informant qu'il a été surenchéri. Il retourne dans l'application et place une nouvelle enchère à 13,100€.

Ce scénario illustre le processus d'inscription, la navigation dans les enchères, le placement d'enchères et le système de notification.

### 6.5.2 Scénario 2 : Gestion d'une Enchère par un Administrateur

Marie, administratrice de la plateforme, doit ajouter un nouveau véhicule : une Audi A3 de 2020. Elle se connecte au panneau d'administration, crée une nouvelle fiche véhicule avec toutes les informations techniques, télécharge 8 photos de la voiture, et rédige une description détaillée. Elle configue ensuite une nouvelle enchère avec un prix de départ de 18,000€, un incrément minimal de 200€, et programme son démarrage pour le lendemain à 10h00, avec une durée de 3 jours. Deux jours plus tard, elle constate que l'encheré génère beaucoup d'intérêt et décide de vérifier les participants. Elle peut voir que 15 personnes ont placé un total de 28 enchères, et que le prix actuel est de 22,600€.

Ce scénario illustre les fonctionnalités d'administration, la création de véhicules et d'enchères, ainsi que le suivi des enchères en cours.

### 6.5.3 Scénario 3 : Finalisation d'une Enchère Réussie

L'encheré pour la Toyota Yaris 2019 vient de se terminer. Thomas, qui avait placé la dernière enchère à 14,800€, reçoit une notification l'informant qu'il a remporté l'encheré. Simultanément, l'administrateur reçoit une alerte de fin d'encheré réussie. Le système met automatiquement à jour le statut de l'encheré à "Terminée" et marque Thomas comme gagnant. Thomas reçoit un email récapitulatif avec les détails du véhicule et les prochaines étapes pour finaliser l'achat. Il peut également consulter ces informations dans la section "Mes encheres remportées" de son profil.

Ce scénario illustre le processus de finalisation d'une enchère, les notifications automatiques et la gestion post-enchère.

## 6.6 Matrice de Traçabilité

Une matrice de traçabilité a été établie pour assurer que toutes les exigences identifiées sont correctement adressées dans la conception et l'implémentation du système. Cette matrice met en relation les exigences avec les composants logiciels et les cas de test correspondants.

Exigence	Composant logiciel	Cas de test
Inscription des utilisateurs	AuthController, UserModel	Test_Registration_Valid, Test_Registration_Invalid
Placement d'enchères	BidController, BidService, SocketHandler	Test_Bid_Valid, Test_Bid_Concurrent, Test_Bid_Notification
Notifications en temps réel	NotificationService, SocketHandler	Test_Notification_Outbid, Test_Notification_EndingSoon
Sécurité des transactions	AuthMiddleware, TransactionService	Test_Security_Authentication, Test_Transaction_Integrity

TABLE 6.1 – Extrait de la matrice de traçabilité

Cette matrice est maintenue tout au long du projet pour garantir que toutes les exigences sont correctement implémentées et testées.

# Chapitre 7

## Conception du Système d'Enchères

### 7.1 Architecture Globale

Ce chapitre présente l'architecture et la conception détaillée du système d'enquêtes d'automobiles. Notre approche de conception suit les principes de l'architecture client-serveur moderne, optimisée pour les applications en temps réel avec des exigences élevées en matière de réactivité et de fiabilité.

#### 7.1.1 Vue d'ensemble

Le système est structuré selon une architecture à plusieurs couches comprenant :

- Une application mobile frontend développée avec React Native
- Un backend serveur RESTful basé sur Node.js et Express
- Une couche de communication en temps réel utilisant Socket.io
- Une base de données MongoDB pour le stockage persistant

### 7.2 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation ci-dessous illustre les principales fonctionnalités du système du point de vue des acteurs principaux : l'utilisateur standard et l'administrateur.

#### 7.2.1 Acteurs du Système

- **Utilisateur** : Représente un client qui peut consulter les enchères, s'inscrire, se connecter, placer des enchères, gérer son profil et suivre ses enchères.
- **Administrateur** : Dispose de priviléges étendus permettant de gérer les voitures, les utilisateurs et créer de nouvelles enchères.

#### 7.2.2 Cas d'Utilisation Principaux

**Consulter les enchères** Permet aux utilisateurs de visualiser toutes les enchères actives.

**S'inscrire** Création d'un nouveau compte utilisateur.

**Se connecter** Authentification dans le système.

**Placer une enchère** Soumettre une offre sur un véhicule en enchère.

**Gérer profil** Modifier les informations personnelles.

**Voir mes enchères** Consulter l'historique personnel des enchères.

**Gérer les voitures** Fonctionnalité réservée à l'administrateur pour ajouter, modifier ou supprimer des véhicules.

**Gérer les utilisateurs** Fonctionnalité réservée à l'administrateur pour gérer les comptes utilisateurs.

**Créer une enchère** Fonctionnalité réservée à l'administrateur pour initier une nouvelle enchère.

### 7.3 Diagramme de Séquence

Le diagramme de séquence suivant illustre le flux d'interaction entre les différentes composantes du système lors du processus d'enquête.

### 7.3.1 Analyse du Flux d'Enchère

Le diagramme de séquence décompose le processus d'enchère en plusieurs étapes clés :

1. L'utilisateur accède à une enchère via l'interface frontend
2. Le frontend demande les détails de l'enchère au backend via une requête GET
3. Le backend interroge la base de données pour récupérer les informations
4. L'utilisateur place une enchère qui est envoyée au serveur
5. Le backend vérifie la validité de l'enchère
6. Si l'enchère est valide :
  - L'enchère est enregistrée dans la base de données
  - Une notification en temps réel est émise via Socket.io
  - Tous les clients connectés sont informés de la mise à jour
7. Si l'enchère est invalide, un message d'erreur est retourné
8. Le système met à jour automatiquement le statut des enchères terminées

## 7.4 Diagramme de Classes

Le diagramme de classes suivant représente la structure des entités principales du système et leurs relations.

### 7.4.1 Entités Principales

**Utilisateur** Contient les informations personnelles et d'authentification. Un utilisateur peut participer à plusieurs enchères et recevoir des notifications.

**Enchère** Représente une session d'encheres avec une date de début, de fin, un prix de départ et un prix actuel. Une enchère concerne un véhicule spécifique et contient plusieurs offres.

**Voiture** Contient les informations détaillées d'un véhicule mis aux enchères, comme la marque, le modèle, l'année, et des images.

**Offre** Représente une proposition financière faite par un utilisateur dans le cadre d'une enchère.

**Notification** Permet d'informer les utilisateurs des événements importants comme les surenchères ou la fin d'une enchère.

### 7.4.2 Relations Entre Entités

- Un utilisateur peut participer à plusieurs enchères (relation n-n)
- Une enchère concerne exactement une voiture (relation 1-1)
- Un utilisateur peut faire plusieurs offres (relation 1-n)
- Une enchère contient plusieurs offres (relation 1-n)
- Un utilisateur reçoit plusieurs notifications (relation 1-n)
- Une enchère génère plusieurs notifications (relation 1-n)

## 7.5 Modèles de Données

### 7.5.1 Modèle Utilisateur

```
const userSchema = new Schema({
  id: String,
  username: String,
  email: String,
  password: String,
  profileImage: String,
  role: String,
  dateCreation: Date,
```

```
    dernièreConnexion: Date
});
```

### 7.5.2 Modèle Enchère

```
const encheresSchema = new Schema({
  id: String,
  voiture: { type: Schema.Types.ObjectId, ref: 'Voiture' },
  dateDebut: Date,
  dateFin: Date,
  prixDepart: Number,
  prixActuel: Number,
  statut: String,
  gagnant: { type: Schema.Types.ObjectId, ref: 'Utilisateur' },
  encheres: [{ type: Schema.Types.ObjectId, ref: 'Offre' }]
});
```

### 7.5.3 Modèle Voiture

```
const voitureSchema = new Schema({
  id: String,
  marque: String,
  modele: String,
  annee: Number,
  couleur: String,
  kilometrage: Number,
  images: [String],
  description: String,
  condition: String,
  transmission: String,
  dateAjout: Date
});
```

### 7.5.4 Modèle Offre

```
const offreSchema = new Schema({
  id: String,
  encheres: { type: Schema.Types.ObjectId, ref: 'Enchere' },
  utilisateur: { type: Schema.Types.ObjectId, ref: 'Utilisateur' },
  montant: Number,
  dateOffre: Date,
  statut: String
});
```

## 7.6 Architecture Technique

### 7.6.1 Architecture Frontend

L'application frontend est développée avec React Native, ce qui permet un déploiement sur iOS et Android à partir d'une base de code unique. Les principales composantes de l'architecture frontend sont :

- **Écrans** : Représentent les différentes vues de l'application

- **Composants** : Éléments d'interface réutilisables
- **Services** : Modules pour la communication avec le backend
- **État global** : Gestion centralisée de l'état de l'application
- **Navigateurs** : Gestion de la navigation entre écrans

### 7.6.2 Architecture Backend

Le backend est construit sur Node.js avec Express.js comme framework web. L'architecture est organisée selon les principes MVC (Modèle-Vue-Contrôleur) adaptés aux API REST :

- **Routes** : Définissent les points d'entrée API
- **Contrôleurs** : Contiennent la logique métier
- **Modèles** : Représentent les entités de base de données
- **Middleware** : Composants pour l'authentification et la validation
- **Services** : Logique partagée entre différents contrôleurs

### 7.6.3 Communication en Temps Réel

La communication en temps réel est gérée par Socket.io, qui permet :

- Des mises à jour instantanées des enchères
- Des notifications en temps réel pour les utilisateurs
- La gestion de salles d'encheres virtuelles
- Une reconnexion automatique en cas de perte de connexion

### 7.6.4 Base de Données

MongoDB a été choisi comme système de base de données pour sa flexibilité et sa scalabilité. Les caractéristiques clés de notre implémentation MongoDB sont :

- Schémas flexibles permettant une évolution rapide
- Relations entre documents gérées par références
- Indexation pour optimiser les performances de requête
- Agrégations pour des analyses complexes

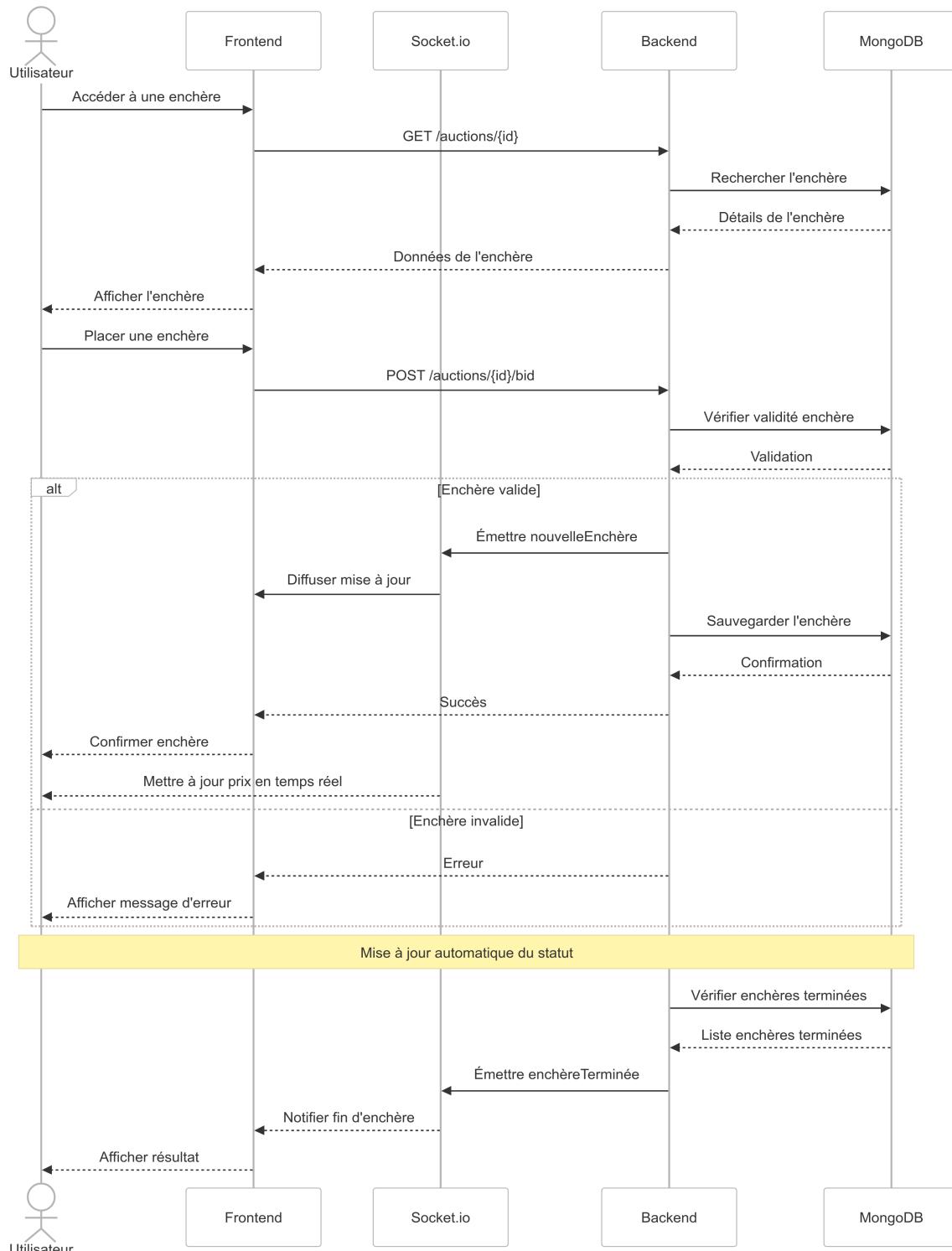
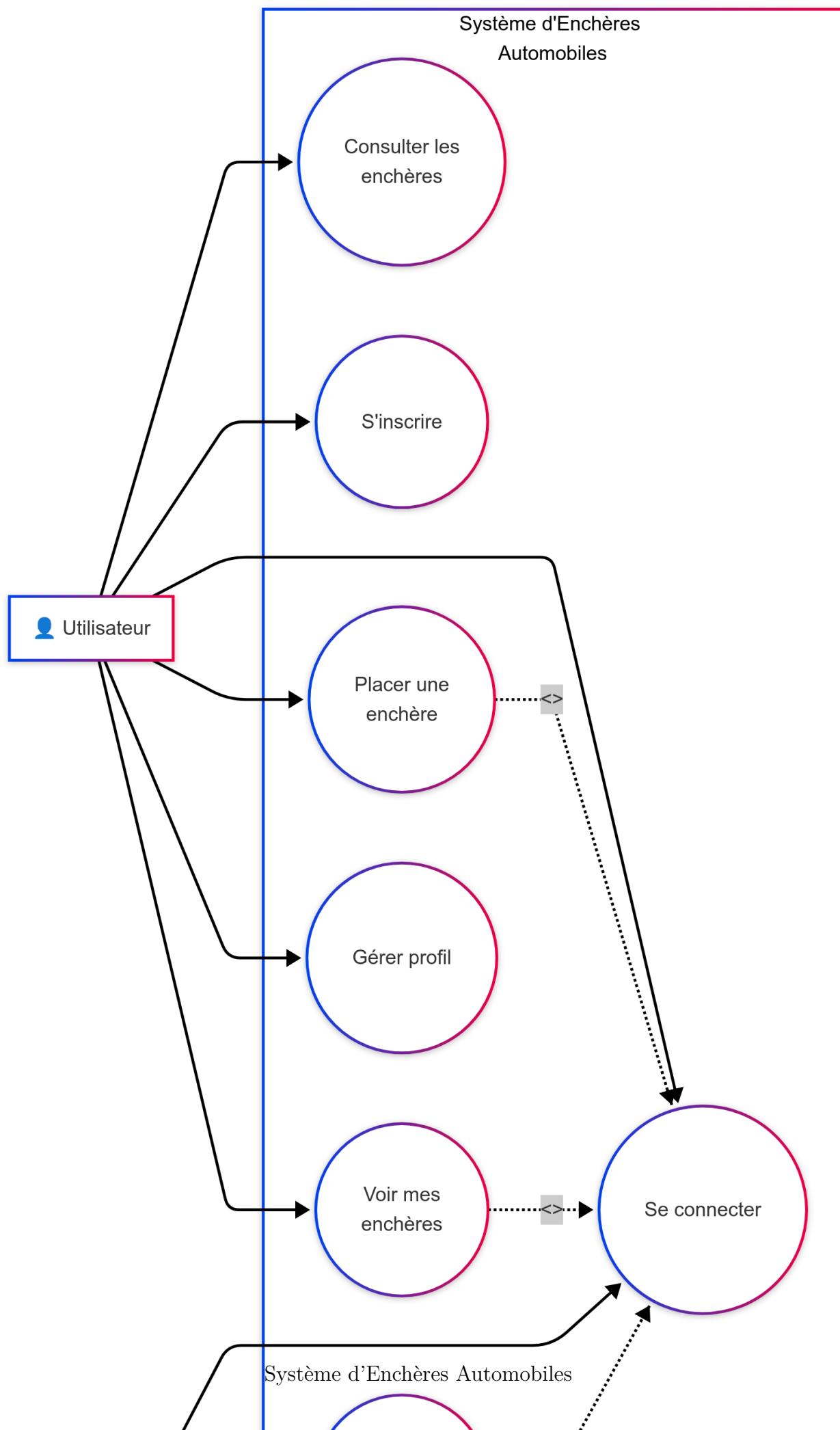


FIGURE 7.1 – Diagramme de cas d'utilisation du système d'enchères



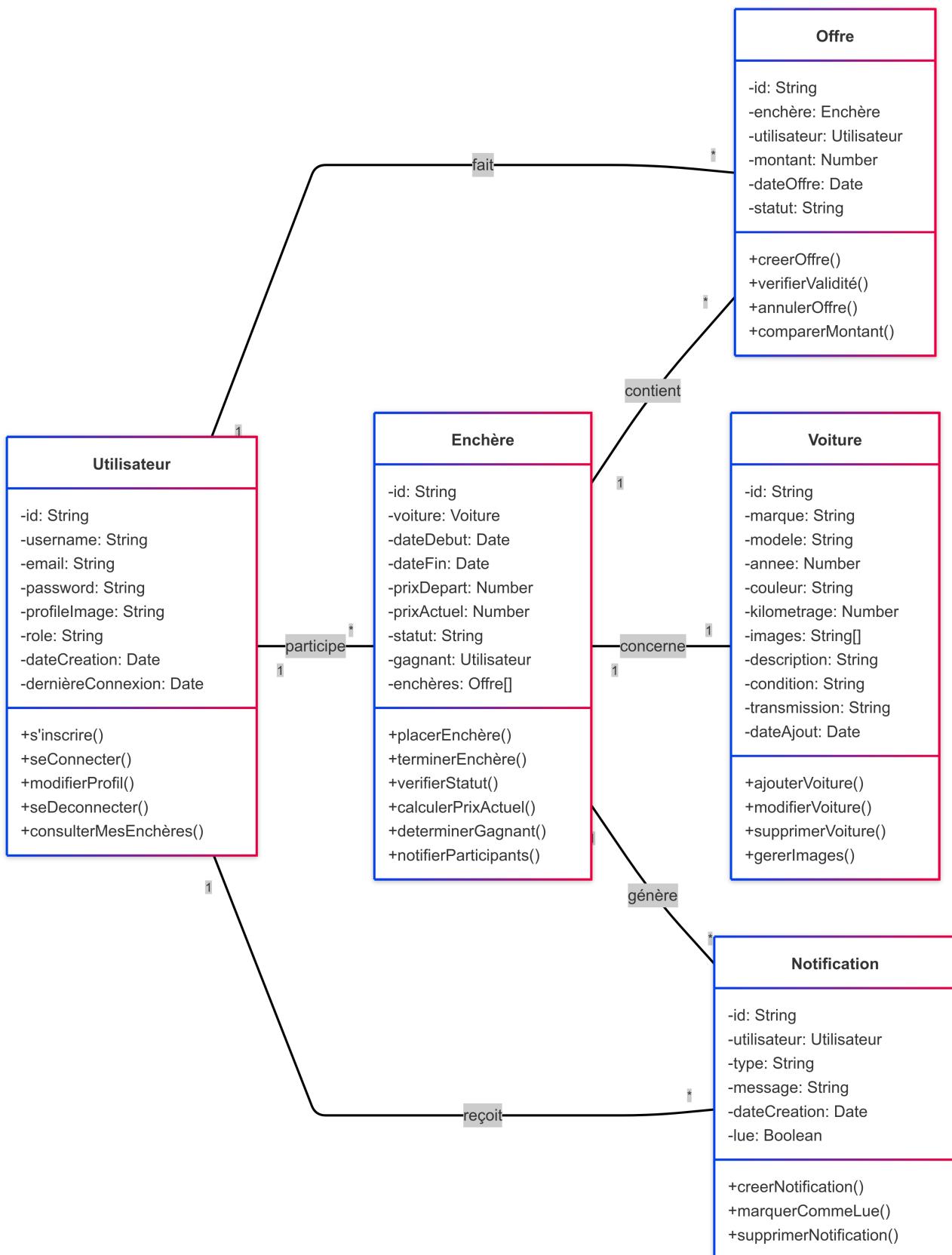


FIGURE 7.3 – Diagramme de classes des principaux composants du système

# Chapitre 8

## Réalisation Technique

### 8.1 Implémentation Frontend

#### 8.1.1 Structure de l'Application Mobile

L'application mobile représente l'interface principale par laquelle les utilisateurs interagissent avec le système d'enchères. Développée avec React Native, elle offre une expérience native sur les plateformes iOS et Android à partir d'une base de code unique.

#### Organisation des Écrans

L'application est structurée autour de plusieurs écrans principaux :

- **AuthenticationScreen** : Gère l'inscription et la connexion des utilisateurs
- **AuctionListScreen** : Affiche la liste des enchères disponibles
- **AuctionDetailScreen** : Présente les détails d'une enchère spécifique
- **BiddingScreen** : Interface pour placer des enchères
- **ProfileScreen** : Gestion du profil utilisateur
- **MyAuctionsScreen** : Historique des enchères de l'utilisateur

#### Navigation

La navigation est gérée via React Navigation, permettant une expérience fluide entre les différents écrans :

```
const Stack = createStackNavigator();

function AppNavigator() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="AuctionDetail" component={AuctionDetailScreen} />
      <Stack.Screen name="Bidding" component={BiddingScreen} />
      <Stack.Screen name="Profile" component={ProfileScreen} />
      <Stack.Screen name="MyAuctions" component={MyAuctionsScreen} />
    </Stack.Navigator>
  );
}
```

#### 8.1.2 Composants Réutilisables

Pour maintenir la cohérence de l'interface et optimiser le développement, plusieurs composants réutilisables ont été créés :

```
// Exemple de composant réutilisable pour les enchères
const AuctionCard = ({ auction, onPress }) => (
  <TouchableOpacity
    style={styles.card}
    onPress={() => onPress(auction.id)}
  >
    <Image
      source={{ uri: auction.car.images[0] }}
      style={styles.carImage}
    />
    <View style={styles.infoContainer}>
      <Text style={styles.title}>
        {auction.car.brand} {auction.car.model}
      </Text>
      <Text style={styles.price}>
        Prix actuel: {auction.currentPrice} €
      </Text>
      <Text style={styles.time}>
        Fin: {formatDate(auction.endTime)}
      </Text>
    </View>
  </TouchableOpacity>
);

```

### 8.1.3 Gestion du State

La gestion de l'état de l'application utilise les hooks React pour les composants locaux et un système centralisé pour l'état global :

```
// Exemple d'utilisation des hooks dans un écran d'enchère
function AuctionDetailScreen({ route }) {
  const { auctionId } = route.params;
  const [auction, setAuction] = useState(null);
  const [loading, setLoading] = useState(true);
  const [bidAmount, setBidAmount] = useState('');

  useEffect(() => {
    const fetchAuction = async () => {
      try {
        const data = await AuctionService.getAuctionById(auctionId);
        setAuction(data);
      } catch (error) {
        console.error('Error fetching auction:', error);
      } finally {
        setLoading(false);
      }
    };
    fetchAuction();
  });
}

// Configuration Socket.io pour les mises à jour en temps réel
```

```

socket.on('auction_update', (updatedAuction) => {
  if (updatedAuction.id === auctionId) {
    setAuction(updatedAuction);
  }
});

return () => {
  socket.off('auction_update');
};

}, [auctionId]);
}

// Reste de la logique du composant...
}

```

### 8.1.4 Communication Temps Réel

L'intégration de Socket.io au frontend permet de recevoir des mises à jour en temps réel des enchères :

```

// Service Socket.io
const initializeSocket = (userId) => {
  socket = io(API_URL);

  socket.on('connect', () => {
    console.log('Connected to Socket.io');
    socket.emit('user_connected', { userId });
  });

  socket.on('new_bid', (data) => {
    // Mise à jour de l'interface en temps réel
    notifyBidUpdate(data);
  });

  socket.on('auction_ended', (data) => {
    // Notification de fin d'enchère
    notifyAuctionEnded(data);
  });

  return socket;
};

```

## 8.2 Implémentation Backend

### 8.2.1 Architecture API

Le backend est implémenté en utilisant Node.js et Express.js, suivant une architecture RESTful pour exposer les points d'entrée API :

```

// Configuration principale Express
const express = require('express');
const app = express();
const http = require('http').createServer(app);

```

```

const io = require('socket.io')(http);
const mongoose = require('mongoose');
const cors = require('cors');

// Middleware
app.use(cors());
app.use(express.json());

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/auctions', auctionRoutes);
app.use('/api/cars', carRoutes);
app.use('/api/users', userRoutes);

// Gestion Socket.io
io.on('connection', (socket) => {
    console.log('New client connected');

    socket.on('join_auction', (auctionId) => {
        socket.join(`auction_${auctionId}`);
    });

    socket.on('place_bid', async (data) => {
        // Traitement de l'enchère
        const result = await auctionService.placeBid(data);

        if (result.success) {
            io.to(`auction_${data.auctionId}`).emit('bid_update', result.data);
        }
    });
});

// Démarrage du serveur
mongoose.connect(DB_URI)
.then(() => {
    console.log('Connected to MongoDB');
    http.listen(PORT, () => {
        console.log(`Server running on port ${PORT}`);
    });
});

```

### 8.2.2 Contrôleurs

Les contrôleurs implémentent la logique métier principale :

```

// Contrôleur d'enchères
const auctionController = {
    // Récupérer toutes les enchères actives
    getAllActive: async (req, res) => {
        try {
            const auctions = await Auction.find({ status: 'active' })

```

```
.populate('car')
  .sort({ endTime: 1 });

  res.json(auctions);
} catch (error) {
  res.status(500).json({ message: error.message });
}
},

// Récupérer une enchère par son ID
getById: async (req, res) => {
  try {
    const auction = await Auction.findById(req.params.id)
      .populate('car')
      .populate({
        path: 'bids',
        populate: { path: 'user', select: 'username' }
      });

    if (!auction) {
      return res.status(404).json({ message: 'Enchère non trouvée' });
    }

    res.json(auction);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
},

// Placer une enchère
placeBid: async (req, res) => {
  try {
    const { auctionId, amount } = req.body;
    const userId = req.user.id;

    const result = await auctionService.placeBid({
      auctionId,
      userId,
      amount
    });

    if (result.success) {
      res.json(result.data);
    } else {
      res.status(400).json({ message: result.message });
    }
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
}
```

```
};
```

### 8.2.3 Middleware

Les middleware gèrent l'authentification et la validation :

```
// Middleware d'authentification
const authMiddleware = async (req, res, next) => {
  try {
    const token = req.header('Authorization').replace('Bearer ', '');
    const decoded = jwt.verify(token, JWT_SECRET);
    const user = await User.findById(decoded.id);

    if (!user) {
      throw new Error();
    }

    req.token = token;
    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ message: 'Veuillez vous authentifier' });
  }
};
```

### 8.2.4 Services

Les services encapsulent la logique réutilisable entre différents contrôleurs :

```
// Service d'enchères
const auctionService = {
  // Placer une enchère
  placeBid: async ({ auctionId, userId, amount }) => {
    // Validation des entrées
    if (!mongoose.Types.ObjectId.isValid(auctionId)) {
      return { success: false, message: 'ID d\'enchère invalide' };
    }

    const amountNum = parseFloat(amount);
    if (isNaN(amountNum)) {
      return { success: false, message: 'Montant invalide' };
    }

    // Début de transaction
    const session = await mongoose.startSession();
    session.startTransaction();

    try {
      // Récupérer l'enchère avec verrouillage
      const auction = await Auction.findById(auctionId).session(session);
```

```
if (!auction) {
    throw new Error('Enchère non trouvée');
}

if (auction.status !== 'active') {
    throw new Error('L\'enchère n\'est plus active');
}

if (auction.endTime < new Date()) {
    throw new Error('L\'enchère est terminée');
}

if (amountNum <= auction.currentPrice) {
    throw new Error('Le montant doit être supérieur au prix actuel');
}

// Créer la nouvelle enchère
const bid = new Bid({
    auction: auctionId,
    user: userId,
    amount: amountNum,
    createdAt: new Date()
});

await bid.save({ session });

// Mettre à jour l'enchère
auction.currentPrice = amountNum;
auction.bids.push(bid._id);
await auction.save({ session });

// Valider la transaction
await session.commitTransaction();

// Populer les données pour la réponse
const populatedBid = await Bid.findById(bid._id)
    .populate('user', 'username')
    .lean();

return {
    success: true,
    data: {
        bid: populatedBid,
        newPrice: amountNum,
        auctionId
    }
};
} catch (error) {
    // Annuler la transaction en cas d'erreur
    await session.abortTransaction();
```

```

        return { success: false, message: error.message };
    } finally {
        session.endSession();
    }
},

// Vérifier et mettre à jour les enchères terminées
checkEndedAuctions: async () => {
    const now = new Date();

    const endedAuctions = await Auction.find({
        status: 'active',
        endTime: { $lte: now }
    });

    for (const auction of endedAuctions) {
        // Trouver le gagnant
        if (auction.bids.length > 0) {
            const highestBid = await Bid.findOne({ auction: auction._id })
                .sort({ amount: -1 })
                .populate('user');

            if (highestBid) {
                auction.winner = highestBid.user._id;
            }
        }

        auction.status = 'ended';
        await auction.save();

        // Émettre un événement de fin d'enchère
        io.to(`auction_${auction._id}`).emit('auction_ended', {
            auctionId: auction._id,
            winner: auction.winner
        });
    }
}

return endedAuctions;
}
};

```

## 8.3 Intégration et Tests

### 8.3.1 Tests Unitaires

Les tests unitaires couvrent les fonctionnalités critiques du système :

```

// Test du service d'enchères
describe('Auction Service Tests', () => {
    test('Devrait rejeter une enchère inférieure au prix actuel', async () => {
        // Configuration du test

```

```

const auctionId = new mongoose.Types.ObjectId();
const userId = new mongoose.Types.ObjectId();

// Mock de l'enchère
jest.spyOn(Auction, 'findById').mockResolvedValue({
  _id: auctionId,
  currentPrice: 1000,
  status: 'active',
  endTime: new Date(Date.now() + 3600000), // +1 heure
  bids: [],
  save: jest.fn().mockResolvedValue(true)
});

// Exécution du test
const result = await auctionService.placeBid({
  auctionId,
  userId,
  amount: 900 // Inférieur au prix actuel
});

// Vérification des résultats
expect(result.success).toBe(false);
expect(result.message).toContain('supérieur au prix actuel');
});
});

```

### 8.3.2 Déploiement

L'application est déployée à l'aide d'une infrastructure cloud garantissant haute disponibilité et scalabilité :

- **Frontend** : Déployé via les app stores (Google Play et App Store)
- **Backend** : Conteneurisé avec Docker et déployé sur une plateforme Kubernetes
- **Base de données** : MongoDB Atlas pour une solution gérée et scalable
- **Monitoring** : Intégration de solutions de monitoring pour surveiller les performances et la disponibilité

### 8.3.3 Sécurité

Plusieurs mesures de sécurité ont été implémentées :

- Stockage sécurisé des mots de passe avec bcrypt
- Protection contre les attaques par injection via la validation des entrées
- Authentification basée sur les tokens JWT
- Gestion sécurisée des sessions
- Protection contre les attaques CSRF
- Configuration HTTPS pour toutes les communications

## 8.4 Défis Techniques et Solutions

### 8.4.1 Concurrence dans les Enchères

Un défi majeur était de gérer la concurrence lorsque plusieurs utilisateurs placent des enchères simultanément :

- **Problème** : Risque de conditions de course (race conditions) entraînant des incohérences dans les données.
- **Solution** : Utilisation de transactions MongoDB avec verrouillage optimiste pour garantir l'intégrité des données.

#### 8.4.2 Performances en Temps Réel

Le maintien des performances pour un grand nombre d'utilisateurs simultanés était essentiel :

- **Problème** : Surcharge potentielle du serveur lors de pics d'activité.
- **Solution** :
  - Optimisation de l'architecture Socket.io avec des espaces de noms et des salles
  - Mise en cache des données fréquemment accédées
  - Architecture horizontalement scalable

#### 8.4.3 Gestion des Images

La gestion efficace des images de véhicules représentait un défi important :

- **Problème** : Stockage et livraison optimisés des images des véhicules.
- **Solution** :
  - Utilisation d'un CDN pour la distribution des images
  - Redimensionnement automatique des images en fonction du contexte d'affichage
  - Compression progressive pour améliorer la vitesse de chargement

# Chapitre 9

## Défis d'Implémentation et Perspectives d'Évolution

### 9.1 Défis Techniques Rencontrés

#### 9.1.1 Gestion de la Concurrence

Un des défis majeurs dans le développement de cette application d'enchères a été la gestion de la concurrence. Les enchères étant un environnement où plusieurs utilisateurs peuvent agir simultanément sur les mêmes ressources, nous avons dû mettre en place des mécanismes solides pour éviter les conflits et garantir l'intégrité des données.

#### Problématique

Les problèmes spécifiques liés à la concurrence incluaient :

- Risque de conditions de course lors de la mise à jour du prix actuel d'une enchère
- Possibilité d'enchères soumises après la date de fin
- Conflit potentiel lors de la détermination du gagnant si plusieurs offres arrivent simultanément

#### Solutions Implémentées

Pour résoudre ces problèmes, nous avons mis en œuvre :

- Une architecture basée sur les transactions MongoDB, garantissant l'atomicité des opérations
- Un mécanisme de verrouillage optimiste pour prévenir les mises à jour simultanées contradictoires
- Une validation côté serveur systématique avec horodatage précis pour les enchères
- Un système de file d'attente pour traiter les enchères dans l'ordre chronologique

```
// Exemple de gestion de transaction pour placer une enchère
const placeBid = async (auctionId, userId, amount) => {
    const session = await mongoose.startSession();
    session.startTransaction();

    try {
        // Verrouillage optimiste avec version
        const auction = await Auction.findById(auctionId).session(session);
        if (!auction) throw new Error('Enchère non trouvée');

        // Vérifications de validité
        if (auction.status !== 'active') throw new Error('Enchère non active');
        if (auction.endTime < new Date()) throw new Error('Enchère terminée');
        if (amount <= auction.currentPrice) throw new Error('Montant insuffisant');

        // Création de l'offre et mise à jour de l'enchère
        const bid = new Bid({ auction: auctionId, user: userId, amount });
        auction.bids.push(bid);
        auction.currentPrice = amount;
        auction.lastBidder = userId;
        auction.lastBidTime = new Date();
        auction.save();
    } catch (err) {
        session.abortTransaction();
        throw err;
    } finally {
        session.endSession();
    }
}
```

```
    await bid.save({ session });

    auction.currentPrice = amount;
    auction.bids.push(bid._id);
    await auction.save({ session });

    await session.commitTransaction();
    return { success: true, bid };
} catch (error) {
    await session.abortTransaction();
    return { success: false, error: error.message };
} finally {
    session.endSession();
}
};

});
```

### 9.1.2 Optimisation des Performances

La nature temps réel de l'application exigeait une attention particulière aux performances, surtout pendant les périodes de forte activité.

## Problématique

Les défis de performance incluaient :

- Latence potentielle lors de la mise à jour en temps réel des prix d'enchères
  - Surcharge du serveur pendant les dernières minutes d'une enchère populaire
  - Gestion efficace d'un grand nombre de connexions simultanées
  - Temps de réponse de la base de données sous charge élevée

## Solutions Implémentées

Pour optimiser les performances, nous avons :

- Implémenté une architecture basée sur des microservices pour une meilleure scalabilité horizontale
  - Utilisé Redis comme solution de cache pour réduire la charge sur MongoDB
  - Optimisé les requêtes MongoDB avec des index appropriés
  - Implémenté un système de limitation de débit (rate limiting) pour éviter les surcharges
  - Configuré Socket.io avec des espaces de noms et des salles pour optimiser la diffusion des messages

```
// Configuration Socket.io avec espaces de noms et salles
const io = require('socket.io')(server);
const auctionNamespace = io.of('/auctions');

auctionNamespace.on('connection', (socket) => {
    // Rejoindre des salles spécifiques pour les enchères
    socket.on('join_auction', (auctionId) => {
        socket.join(`auction_${auctionId}`);
    });
});
```

```
// Diffusion ciblée aux participants d'une enchère spécifique
socket.on('place_bid', async (data) => {
  const result = await auctionService.placeBid(data);
  if (result.success) {
    auctionNamespace.to(`auction_${data.auctionId}`).emit('bid_update', result.data);
  } else {
    socket.emit('bid_error', { message: result.error });
  }
});
});
```

### 9.1.3 Sécurité et Authentification

La sécurité était une priorité absolue pour protéger les données des utilisateurs et maintenir l'intégrité des enchères.

#### Problématique

Les défis de sécurité incluaient :

- Protection contre les attaques par force brute
- Risque de manipulation des enchères par des utilisateurs malveillants
- Sécurisation des communications en temps réel
- Protection des données sensibles des utilisateurs

#### Solutions Implémentées

Pour renforcer la sécurité, nous avons :

- Implémenté un système d'authentification robuste basé sur JWT avec rotation des tokens
- Mis en place une validation stricte des entrées côté serveur
- Configuré HTTPS pour toutes les communications
- Appliqué des politiques de mot de passe fort avec hachage bcrypt
- Implémenté une détection d'activité suspecte avec blocage temporaire des comptes
- Sécurisé les connexions WebSocket avec des tokens d'authentification

```
// Middleware d'authentification pour Socket.io
const authenticateSocket = (socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) {
    return next(new Error('Authentication error'));
  }

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    socket.user = decoded;
    next();
  } catch (error) {
    next(new Error('Invalid token'));
  }
};

io.use(authenticateSocket);
```

## 9.2 Perspectives d'Évolution

### 9.2.1 Fonctionnalités Additionnelles

Sur la base du système actuel, plusieurs fonctionnalités pourraient être ajoutées pour enrichir l'expérience utilisateur :

#### Système de Paiement Intégré

L'intégration d'une solution de paiement permettrait de finaliser les transactions directement dans l'application :

- Intégration avec des API de paiement populaires (Stripe, PayPal)
- Gestion de dépôts de garantie pour les enchères
- Système de facturation automatisé
- Suivi des transactions et historique des paiements

#### Système de Notation et d'Avis

Un système permettant aux utilisateurs d'évaluer les vendeurs et les véhicules après achat :

- Notations et commentaires sur les vendeurs
- Vérification de l'authenticité des avis
- Impact sur la réputation et la visibilité des vendeurs
- Système de badges pour les vendeurs fiables

#### Assistant d'Enchères Intelligent

Un assistant basé sur l'IA pour aider les utilisateurs dans leur stratégie d'enchères :

- Prédiction des tendances de prix basée sur l'historique
- Recommandations personnalisées de véhicules
- Aide à la décision pour le montant optimal des enchères
- Alertes intelligentes pour les opportunités d'enchères

#### Système de Messagerie Interne

Une plateforme de communication directe entre acheteurs et vendeurs :

- Messagerie privée sécurisée
- Possibilité de poser des questions sur les véhicules
- Négociation post-enchère
- Partage sécurisé de documents

### 9.2.2 Améliorations Techniques

#### Architecture Serverless

Migration vers une architecture serverless pour améliorer la scalabilité et réduire les coûts :

- Utilisation de AWS Lambda ou Azure Functions pour le backend
- API Gateway pour la gestion des requêtes
- WebSockets managés avec des services cloud
- Base de données distribuée pour une meilleure résilience

## Progressive Web App (PWA)

Transformation de l'application en PWA pour améliorer l'accessibilité :

- Fonctionnalités hors ligne avec synchronisation
- Installation sur l'écran d'accueil sans passer par les app stores
- Push notifications pour les mises à jour d'enchères
- Expérience utilisateur améliorée sur tous les appareils

## Intelligence Artificielle et Machine Learning

Intégration de fonctionnalités IA pour enrichir l'application :

- Estimation automatique de la valeur des véhicules
- Détection de fraude et d'activités suspectes
- Optimisation dynamique des prix de départ basée sur des données historiques
- Recommandations personnalisées basées sur le comportement des utilisateurs

### 9.2.3 Expansion Internationale

#### Multilinguisme et Localisation

Pour atteindre un public international :

- Support de multiples langues dans l'interface
- Adaptation aux différentes devises et formats régionaux
- Conformité aux réglementations locales sur les enchères
- Personnalisation des contenus selon les marchés cibles

## Infrastructure Géo-Distribuée

Pour offrir une latence minimale aux utilisateurs du monde entier :

- Déploiement multi-régional des serveurs
- Réseau de diffusion de contenu (CDN) pour les ressources statiques
- RéPLICATION géographique des bases de données
- Monitoring et métriques par région

## 9.3 Analyse Comparative avec des Solutions Existantes

### 9.3.1 Forces du Système

Par rapport aux plateformes d'enchères automobile existantes, notre solution présente plusieurs avantages :

- Interface utilisateur intuitive et moderne
- Système de notification en temps réel plus réactif
- Architecture technique plus flexible et évolutive
- Optimisation pour les appareils mobiles
- Meilleure intégration des fonctionnalités sociales

### 9.3.2 Opportunités d'Amélioration

Des domaines où notre système pourrait s'améliorer :

- Enrichir les fonctionnalités de vérification des véhicules
- Développer des partenariats avec des services d'inspection

- Renforcer les outils d'analyse de marché
- Améliorer les fonctionnalités pour les vendeurs professionnels

## 9.4 Impact Écologique et Économique

### 9.4.1 Réduction de l'Empreinte Carbone

L'application peut contribuer positivement à l'environnement :

- Réduction des déplacements pour voir des véhicules grâce aux informations détaillées
- Optimisation de la réutilisation des véhicules d'occasion
- Possibilité de promotion des véhicules électriques et hybrides

### 9.4.2 Démocratisation du Marché Automobile

Impact économique positif :

- Accès facilité au marché pour les petits vendeurs
- Transparence des prix favorisant une concurrence équitable
- Réduction des intermédiaires et des coûts associés
- Potentiel de création d'emplois dans le secteur des services automobiles associés

# Chapitre 10

## Conclusion

### 10.1 Synthèse du Projet

Le développement du système d'enchères automobiles a représenté un défi technique significatif, nécessitant une approche méthodique et une architecture robuste pour répondre aux exigences de performance, sécurité et expérience utilisateur. Ce projet a permis de mettre en œuvre des solutions innovantes pour résoudre des problématiques complexes liées aux applications en temps réel.

#### 10.1.1 Réalisations Principales

Le système développé se distingue par plusieurs aspects notables :

- Une architecture complète client-serveur utilisant des technologies modernes (React Native, Node.js, MongoDB, Socket.io)
- Un système d'enchères en temps réel performant et fiable, gérant efficacement la concurrence
- Une interface utilisateur intuitive et réactive sur plateformes mobiles
- Un backend sécurisé avec une gestion robuste de l'authentification et des autorisations
- Une conception évolutive permettant l'ajout futur de fonctionnalités

#### 10.1.2 Objectifs Atteints

Le projet a répondu avec succès aux objectifs initiaux :

- Permettre aux utilisateurs de participer facilement à des enchères automobiles depuis leurs appareils mobiles
- Garantir l'équité et la transparence du processus d'enchère
- Fournir une expérience utilisateur fluide et engageante
- Assurer la sécurité des données et des transactions
- Offrir une plateforme adaptable aux évolutions du marché

### 10.2 Compétences Développées

La réalisation de ce projet a permis de développer et d'approfondir plusieurs compétences techniques et organisationnelles :

#### 10.2.1 Compétences Techniques

- Maîtrise du développement d'applications mobiles avec React Native
- Conception et implémentation d'API RESTful avec Node.js et Express
- Gestion efficace des communications en temps réel avec Socket.io
- Modélisation et optimisation de bases de données NoSQL (MongoDB)
- Mise en œuvre de mécanismes d'authentification et de sécurité avancés
- Déploiement et maintenance d'applications sur infrastructures cloud

#### 10.2.2 Compétences Méthodologiques

- Application des principes de conception UML pour modéliser le système

- Gestion de projet agile permettant une adaptation continue aux besoins
- Tests et assurance qualité systématiques
- Documentation approfondie du code et de l'architecture
- Analyse et résolution de problèmes complexes

## 10.3 Apports Personnels et Professionnels

Ce projet a constitué une expérience enrichissante tant sur le plan personnel que professionnel :

### 10.3.1 Développement Personnel

- Renforcement des capacités d'analyse et de résolution de problèmes
- Amélioration des compétences en gestion du temps et des priorités
- Développement de la persévérance face aux obstacles techniques
- Perfectionnement de l'autonomie dans l'apprentissage de nouvelles technologies

### 10.3.2 Développement Professionnel

- Acquisition d'une expérience pratique avec des technologies de pointe
- Compréhension approfondie des enjeux de développement d'applications commerciales
- Constitution d'un portfolio démontrant des compétences diversifiées
- Préparation aux défis du marché du travail dans le développement d'applications

## 10.4 Perspectives d'Avenir

### 10.4.1 Évolution du Projet

Le système d'enchères automobiles dispose d'un potentiel d'évolution important :

- Intégration de fonctionnalités basées sur l'intelligence artificielle pour l'estimation des valeurs de véhicules et les recommandations personnalisées
- Expansion vers une marketplace complète incluant services et pièces automobiles
- Développement d'une version web progressive en complément des applications mobiles
- Internationalisation pour atteindre des marchés globaux

### 10.4.2 Applications des Connaissances Acquises

Les compétences et connaissances acquises durant ce projet peuvent être appliquées à divers domaines :

- Développement d'autres applications de commerce électronique en temps réel
- Conception de systèmes financiers nécessitant intégrité et performance
- Création de plateformes collaboratives avec interactions en temps réel
- Implémentation de solutions de communication sécurisées

## 10.5 Mot de Fin

La réalisation de ce système d'enchères automobiles représente une étape significative dans notre parcours professionnel. Au-delà des aspects techniques, ce projet nous a permis de comprendre l'importance de concevoir des solutions centrées sur l'utilisateur, robustes techniquement et évolutives pour s'adapter aux besoins futurs.

Les défis rencontrés et surmontés tout au long du développement ont renforcé notre confiance en notre capacité à aborder des problèmes complexes, à rechercher et implémenter des solutions innovantes, et à livrer des produits de qualité.

Nous sommes convaincus que les connaissances et l'expérience acquises constitueront une base solide pour nos futurs projets et contribueront significativement à notre évolution professionnelle dans le domaine du développement d'applications.

# Chapitre 11

## Annexes

### 11.1 Guide d'Installation

#### 11.1.1 Prérequis

- Node.js v23.10.0 ou supérieur
- MongoDB
- Xcode (pour iOS)
- npm ou yarn

#### 11.1.2 Installation du Projet

```
1 # Clone du projet
2 git clone [URL_DU_PROJET]
3
4 # Installation des dépendances backend
5 cd backend
6 npm install
7
8 # Installation des dépendances frontend admin
9 cd ../admin-dashboard
10 npm install
11
12 # Installation des dépendances mobile
13 cd ../mobile
14 npm install
```

#### 11.1.3 Configuration

```
1 # Configuration des ports
2 Backend: 5001
3 Admin Dashboard: 3000
4 Metro Bundler: 8081
5
6 # En cas de conflit de ports
7 lsof -ti:5001,3000,8081 | xargs kill -9
```

## 11.2 Documentation API

### 11.2.1 Routes d'Authentification

- POST /api/auth/register

```
1 {
2     "email": "string",
3     "password": "string",
4     "name": "string"
5 }
6
```

— POST /api/auth/login

```

1  {
2      "email": "string",
3      "password": "string"
4  }
5

```

### 11.2.2 Routes des Enchères

— GET /api/auctions

— POST /api/auctions

```

1  {
2      "title": "string",
3      "description": "string",
4      "startPrice": "number",
5      "startTime": "date",
6      "endTime": "date"
7  }
8

```

— POST /api/auctions/ :id/bid

```

1  {
2      "amount": "number"
3  }
4

```

## 11.3 Schéma de la Base de Données

### 11.3.1 Collection Users

```

1  {
2      "_id": ObjectId,
3      "email": String,
4      "password": String,
5      "name": String,
6      "role": String,
7      "createdAt": Date
8 }

```

### 11.3.2 Collection Auctions

```

1  {
2      "_id": ObjectId,
3      "title": String,
4      "description": String,
5      "startPrice": Number,
6      "currentPrice": Number,
7      "startTime": Date,
8      "endTime": Date,
9      "status": String,
10     "winner": ObjectId,
11     "bids": Array
12 }

```

## 11.4 Captures d'écran de l'application

## 11.5 Logs et Débogage

### 11.5.1 Exemple de Logs

```
1 # Logs de mise à jour des statuts
2 Status updates: 0 became active, 0 ended
3 Processed winners for 2 newly ended auctions
4
5 # Logs de démarrage du serveur
6 Upload directories created successfully
7 Server running on port 5001
8 Connected to MongoDB
```

## 11.6 Scripts Utiles

### 11.6.1 Gestion des Processus

```
1 # Démarrage du backend
2 cd backend && npm start
3
4 # Démarrage du dashboard admin
5 cd admin-dashboard && npm start
6
7 # Démarrage du bundler React Native
8 npx react-native start --reset-cache
9
10 # Lancement de l'app iOS
11 npx react-native run-ios
```