# Documentation Technique MyCarDeals

Plateforme de Vente de Véhicules en Ligne

Projet de Stage

*Yassine TAMANI & Souheyl LABIDI*

23 mars 2025

# Table des matières

# Chapitre 1

# Introduction

## 1.1 Présentation du Projet

MyCarDeals est une application web complète dédiée à la vente de véhicules en ligne, développée dans le cadre d'un stage par Yassine TAMANI et Souheyl LABIDI. Cette plateforme se compose de deux interfaces distinctes : une interface publique destinée aux acheteurs potentiels, leur permettant de consulter les véhicules disponibles, et une interface d'administration réservée aux gestionnaires du site pour gérer efficacement le parc automobile.

## 1.2 Objectifs du Projet

Les objectifs principaux de la plateforme MyCarDeals sont :

- Créer une vitrine en ligne moderne et attractive pour présenter les véhicules à vendre
- Fournir une interface détaillée pour chaque véhicule incluant les caractéristiques techniques, équipements et tarifs
- Offrir un système d'administration complet permettant la gestion des véhicules (ajout, modification, suppression)
- Implémenter une architecture robuste et évolutive basée sur des technologies modernes
- Assurer une expérience utilisateur intuitive et responsive sur tous les appareils

## 1.3 Stack Technologique

Le projet MyCarDeals utilise les technologies suivantes :

### 1.3.1 Frontend

- HTML5
- CSS3
- JavaScript (ES6+)
- TailwindCSS pour le design responsive

### 1.3.2 Backend

- Node.js

— Express.js (framework web)
— Firebase Admin SDK

### 1.3.3 Base de données et Services

— Firebase Firestore (base de données NoSQL)
— Firebase Authentication (gestion des utilisateurs et authentification)
— Firebase Hosting (déploiement)

### 1.3.4 Sécurité

— JWT (JSON Web Tokens) pour l'authentification
— Helmet.js pour la sécurité HTTP
— Express Validator pour la validation des données
— CORS pour la gestion des requêtes cross-origin

### 1.3.5 Outils de développement

— npm (gestionnaire de paquets)
— dotenv (gestion des variables d'environnement)
— Winston (journalisation)

# Chapitre 2

# Architecture du Projet

## 2.1 Structure des Répertoires

Le projet suit une architecture organisée avec une séparation claire entre le frontend et le backend :

Listing 2.1 – Structure principale du projet

```
mycardeals/
        backend/                  # Serveur et API
            config/               # Configuration Firebase
            middleware/           # Middleware d'authentification
            routes/               # Routes API
                admin.js          # Routes d'administration
                auth.js           # Routes d'authentification
                car.js            # Routes pour les v hicules
            .env                  # Variables d'environnement
            server.js             # Point d'entr e du serveur
        frontend/                 # Interface utilisateur
            admin/                # Interface d'administration
                css/              # Styles sp cifiques    l'admin
                js/               # Scripts sp cifiques    l'admin
                    admin.js      # Fonctions d'administration
                    cars.js       # Gestion des v hicules
                    dashboard.js  # Tableau de bord
                    login.js      # Authentification admin
                cars.html         # Page de gestion des v hicules
                dashboard.html    # Tableau de bord admin
                index.html        # Page de connexion admin
            css/                  # Styles g n raux
                styles.css        # Feuille de style principale
            img/                  # Images statiques
            js/                   # Scripts du frontend public
                admin.js          # Fonctions admin partag es
                car-details.js    # D tails des v hicules
                listings.js       # Liste des v hicules
                main.js           # Script principal
                tailwind-config.js # Configuration Tailwind
```

```
        utils.js          # Fonctions utilitaires
    car-details.html      # Page de d tail de v hicule
    index.html            # Page d'accueil
    listings.html         # Page de liste des v hicules
    404.html              # Page d'erreur
scripts/                  # Scripts utilitaires
    addCars.js            # Ajout de v hicules d mo
    createAdmin.js        # Cr ation d'un admin
    migrateCarsToVehicles.js # Migration de donn es
firebase.json             # Configuration Firebase
firestore.indexes.json    # Index Firestore
firestore.rules           # R gles de s curit  Firestore
package.json              # D pendances npm
package-lock.json         # Verrouillage des versions
serviceAccountKey.json    # Cl  d'API Firebase (priv e)
```

## 2.2 Architecture Backend

Le backend est construit suivant une architecture modulaire, avec une séparation claire des responsabilités entre les différents composants.

### 2.2.1 Serveur Express

Le point d'entrée du backend est le fichier `server.js` qui configure et lance le serveur Express :

Listing 2.2 – Extrait de server.js

```
const express = require('express');
const helmet = require('helmet');
const cors = require('cors');
const compression = require('compression');
const path = require('path');
const { logger } = require('./middleware/logger');
require('dotenv').config();

// Routes
const adminRoutes = require('./routes/admin');
const carRoutes = require('./routes/car');

const app = express();
const PORT = process.env.PORT || 3000;

// Middlewares
app.use(helmet());
app.use(cors());
app.use(compression());
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(logger);

// API Routes
app.use('/api/admin', adminRoutes);
app.use('/api/cars', carRoutes);

// Servir les fichiers statiques
```

```
29 app.use(express.static(path.join(__dirname, '../frontend')));
30
31 // Route g n rique pour le frontend (SPA)
32 app.get('*', (req, res) => {
33   res.sendFile(path.join(__dirname, '../frontend/index.html'));
34 });
35
36 // D marrage du serveur
37 app.listen(PORT, () => {
38   console.log(`Serveur d marr  sur le port ${PORT}`);
39 });
```

### 2.2.2 Configuration Firebase

La connexion à Firebase est définie dans le fichier `backend/config/firebase-config.js` :

Listing 2.3 – Configuration Firebase

```
1  const admin = require('firebase-admin');
2  const dotenv = require('dotenv');
3  const path = require('path');
4
5  dotenv.config({ path: path.resolve(__dirname, '../.env') });
6
7  // V rification des variables d'environnement
8  const serviceAccount = {
9    "type": process.env.FIREBASE_TYPE || "service_account",
10   "project_id": process.env.FIREBASE_PROJECT_ID,
11   "private_key_id": process.env.FIREBASE_PRIVATE_KEY_ID,
12   "private_key": process.env.FIREBASE_PRIVATE_KEY ?
13                  process.env.FIREBASE_PRIVATE_KEY.replace(/\\n/g, '\n') :
14                  undefined,
15   "client_email": process.env.FIREBASE_CLIENT_EMAIL,
16   "client_id": process.env.FIREBASE_CLIENT_ID,
17   "auth_uri": process.env.FIREBASE_AUTH_URI,
18   "token_uri": process.env.FIREBASE_TOKEN_URI,
19   "auth_provider_x509_cert_url": process.env.FIREBASE_AUTH_PROVIDER_CERT_URL
       ,
20   "client_x509_cert_url": process.env.FIREBASE_CLIENT_CERT_URL
21 };
22
23 // Initialisation de Firebase Admin
24 admin.initializeApp({
25   credential: admin.credential.cert(serviceAccount)
26 });
27
28 const db = admin.firestore();
29
30 module.exports = { admin, db };
```

### 2.2.3 Middleware d'Authentification

Le middleware d'authentification vérifie la validité des tokens JWT et les droits d'accès administrateur :

Listing 2.4 – Middleware d'authentification

```
1 const { admin, db } = require('../config/firebase-config');
2 const jwt = require('jsonwebtoken');
```

```
3
4  const authenticateAdmin = async (req, res, next) => {
5      try {
6          const token = req.headers.authorization?.split(' ')[1];
7
8          if (!token) {
9              return res.status(401).json({ error: 'Non autoris : Token
                   manquant' });
10         }
11
12         // V rifier le token
13         const decoded = jwt.verify(token, process.env.JWT_SECRET);
14
15         // V rifier si l'utilisateur existe et est un admin
16         const userSnapshot = await db.collection('admins').doc(decoded.uid).
               get();
17
18         if (!userSnapshot.exists) {
19             return res.status(401).json({ error: 'Non autoris : Utilisateur
                    non trouv ' });
20         }
21
22         const userData = userSnapshot.data();
23
24         if (!userData.isAdmin) {
25             return res.status(403).json({ error: 'Acc s  refus : Droits
                   administrateur requis' });
26         }
27
28         // Ajouter les informations de l'utilisateur    la requ te
29         req.user = {
30             uid: decoded.uid,
31             email: userData.email,
32             isAdmin: userData.isAdmin
33         };
34
35         next();
36     } catch (error) {
37         console.error('Erreur d\'authentification:', error);
38         return res.status(401).json({ error: 'Non autoris : Token invalide'
               });
39     }
40 };
41
42 module.exports = { authenticateAdmin };
```

### 2.2.4 Routes API

Les routes API sont organisées en deux fichiers principaux :

#### Routes Publiques (car.js)

Ces routes gèrent les opérations de lecture des véhicules accessibles au public :

Listing 2.5 – Extrait des routes publiques

```
1 const express = require('express');
2 const router = express.Router();
3 const { db } = require('../config/firebase-config');
```

```
4
5 // R cup rer tous les v hicules
6 router.get('/', async (req, res) => {
7     try {
8         const carsSnapshot = await db.collection('vehicles').get();
9         const cars = [];
10
11         carsSnapshot.forEach(doc => {
12             cars.push({
13                 id: doc.id,
14                 ...doc.data()
15             });
16         });
17
18         res.json(cars);
19     } catch (error) {
20         console.error('Erreur lors de la r cup ration des v hicules:',
               error);
21         res.status(500).json({ error: 'Erreur serveur' });
22     }
23 });
24
25 // R cup rer un v hicule sp cifique
26 router.get('/:id', async (req, res) => {
27     try {
28         const carDoc = await db.collection('vehicles').doc(req.params.id).
               get();
29
30         if (!carDoc.exists) {
31             return res.status(404).json({ error: 'V hicule non trouv ' });
32         }
33
34         res.json({
35             id: carDoc.id,
36             ...carDoc.data()
37         });
38     } catch (error) {
39         console.error('Erreur lors de la r cup ration du v hicule:',
               error);
40         res.status(500).json({ error: 'Erreur serveur' });
41     }
42 });
43
44 // R cup rer les suggestions de v hicules similaires
45 router.get('/suggestions', async (req, res) => {
46     try {
47         const excludeId = req.query.exclude;
48         const limit = parseInt(req.query.limit) || 3;
49
50         let query = db.collection('vehicles');
51
52         if (excludeId) {
53             query = query.where('__name__', '!=', excludeId);
54         }
55
56         const suggestionsSnapshot = await query.limit(limit).get();
57         const suggestions = [];
58
59         suggestionsSnapshot.forEach(doc => {
60             suggestions.push({
```

```
                id: doc.id,
                ...doc.data()
            });
        });

        res.json(suggestions);
    } catch (error) {
        console.error('Erreur lors de la r cup ration des suggestions:', error);
        res.status(500).json({ error: 'Erreur serveur' });
    }
});

module.exports = router;
```

### Routes d'Administration (admin.js)

Ces routes gèrent les opérations CRUD réservées aux administrateurs :

Listing 2.6 – Extrait des routes d'administration

```
const express = require('express');
const router = express.Router();
const { db, admin } = require('../config/firebase-config');
const { authenticateAdmin } = require('../middleware/auth-middleware');
const { body, validationResult } = require('express-validator');
const jwt = require('jsonwebtoken');

// Authentification administrateur
router.post('/login', async (req, res) => {
    try {
        const { email, password } = req.body;

        if (!email || !password) {
            return res.status(400).json({ error: 'Email et mot de passe requis' });
        }

        // Authentifier via Firebase Auth
        const userRecord = await admin.auth().getUserByEmail(email);

        // V rifier si l'utilisateur est un administrateur
        const adminDoc = await db.collection('admins').doc(userRecord.uid).get();

        if (!adminDoc.exists || !adminDoc.data().isAdmin) {
            return res.status(403).json({ error: 'Acc s refus : Droits administrateur requis' });
        }

        // G n rer un token JWT
        const token = jwt.sign(
            { uid: userRecord.uid, email: userRecord.email },
            process.env.JWT_SECRET,
            { expiresIn: '24h' }
        );

        res.json({ token, user: { uid: userRecord.uid, email: userRecord.email } });
    } catch (error) {
```

```
        console.error('Erreur d\'authentification:', error);
        res.status(401).json({ error: 'Identifiants invalides' });
    }
});

// Routes prot g es par le middleware d'authentification
router.use(authenticateAdmin);

// Ajouter un v hicule
router.post('/cars', [
    body('name').notEmpty().withMessage('Le nom est requis'),
    body('price').isNumeric().withMessage('Le prix doit  tre  un nombre'),
    body('year').isNumeric().withMessage('L\'ann e doit  tre  un nombre')
], async (req, res) => {
    // Validation des donn es
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
    }

    try {
        const carData = {
            name: req.body.name,
            brand: req.body.brand,
            model: req.body.model,
            year: parseInt(req.body.year),
            price: parseFloat(req.body.price),
            description: req.body.description,
            mileage: parseInt(req.body.mileage) || 0,
            fuel: req.body.fuel,
            transmission: req.body.transmission,
            power: parseInt(req.body.power) || 0,
            color: req.body.color,
            imageUrl: req.body.imageUrl,
            available: req.body.available === true,
            equipments: req.body.equipments || {},
            createdAt: admin.firestore.FieldValue.serverTimestamp(),
            updatedAt: admin.firestore.FieldValue.serverTimestamp()
        };

        const newCarRef = await db.collection('vehicles').add(carData);

        res.status(201).json({
            id: newCarRef.id,
            ...carData
        });
    } catch (error) {
        console.error('Erreur lors de l\'ajout du v hicule:', error);
        res.status(500).json({ error: 'Erreur serveur' });
    }
});

// Autres routes CRUD pour les v hicules...

module.exports = router;
```

## 2.3 Architecture Frontend

Le frontend de l'application utilise une approche basée sur des pages HTML distinctes avec JavaScript pour la dynamisation, et TailwindCSS pour le design.

### 2.3.1 Structure des Pages

#### Pages Publiques

**Page d'Accueil (index.html)** Cette page présente les véhicules en vedette et les fonctionnalités principales du site.

**Page de Listings (listings.html)** Affiche la liste complète des véhicules avec des options de filtrage.

**Page de Détails (car-details.html)** Présente les informations détaillées d'un véhicule spécifique.

#### Pages d'Administration

**Page de Connexion (admin/index.html)** Permet l'authentification des administrateurs.

**Tableau de Bord (admin/dashboard.html)** Fournit une vue d'ensemble des statistiques.

**Gestion des Véhicules (admin/cars.html)** Interface complète pour gérer les véhicules (ajout, modification, suppression).

### 2.3.2 JavaScript Frontend

Les fichiers JavaScript sont organisés par fonctionnalité :

#### Scripts Publics

**main.js** Gère l'initialisation et les fonctionnalités de la page d'accueil.

**listings.js** Gère le chargement et le filtrage des véhicules dans la page de liste.

**car-details.js** Gère l'affichage des détails d'un véhicule spécifique.

Voici un extrait du fichier `car-details.js` montrant comment les équipements d'un véhicule sont affichés :

Listing 2.7 – Extrait de car-details.js

```
1 // Fonction pour mettre    jour les   quipements
2 const updateCarEquipments = (equipements) => {
3     const equipmentsContainer = document.getElementById('carEquipments');
4     if (!equipmentsContainer) return;
5
6     // Nouveau format d'  quipements   structur  (confort, securite,
          multimedia)
7     if (equipements && typeof equipements === 'object' && (equipements.
          confort || equipements.securite || equipements.multimedia)) {
8         let equipmentsHTML = '';
```

```
9
10          // Fonction pour g n rer les items d' quipement  activ s
11          const generateEquipmentItems = (equipment) => {
12              let items = [];
13              if (!equipment) return items;
14
15              // Parcourir chaque propri t  de l'objet et n'ajouter que
                    celles qui sont true
16              Object.entries(equipment).forEach(([key, enabled]) => {
17                  if (enabled === true) {
18                      // Transformer la cl  en texte lisible
19                      let label = '';
20                      switch (key) {
21                          // Confort
22                          case 'climatisation': label = 'Climatisation
                                automatique'; break;
23                          case 'regulateur': label = 'R gulateur de vitesse
                                adaptatif'; break;
24                          case 'siegesChauf': label = 'Si ges chauffants';
                                break;
25                          case 'demarrageSansCle': label = 'D marrage sans
                                cl '; break;
26
27                          // S curit
28                          case 'freinageUrgence': label = 'Freinage d\'urgence
                                 automatique'; break;
29                          case 'detectionAngles': label = 'D tection des
                                angles morts'; break;
30                          case 'cameraRecul': label = 'Cam ra de recul';
                                break;
31                          case 'aideStationnement': label = 'Aide au
                                stationnement'; break;
32
33                          // Multim dia
34                          case 'ecranTactile': label = ' cran  tactile 9"';
                                break;
35                          case 'carPlay': label = 'Apple CarPlay/Android Auto
                                '; break;
36                          case 'bluetooth': label = 'Bluetooth'; break;
37                          case 'gps': label = 'Navigation GPS'; break;
38
39                          default: label = key.charAt(0).toUpperCase() + key.
                                slice(1); // Capitaliser par d faut
40                      }
41
42                      items.push(label);
43                  }
44              });
45
46              return items;
47          };
48
49          // G n rer les sections pour chaque cat gorie
50          const categories = [
51              { id: 'confort', name: 'Confort' },
52              { id: 'securite', name: 'S curit ' },
53              { id: 'multimedia', name: 'Multim dia' }
54          ];
55
56          categories.forEach(category => {
```

```
57              const items = generateEquipmentItems(equipements[category.id]);
58
59              if (items.length > 0) {
60                  equipmentsHTML += `
61                      <div class="mb-6">
62                          <h3 class="text-lg font-semibold mb-3 text-secondary
                               ">${category.name}</h3>
63                          <ul class="grid grid-cols-1 md:grid-cols-2 gap-2">
64                              ${items.map(item => `
65                                  <li class="flex items-center">
66                                      <svg class="w-5 h-5 text-accent mr-2
                                           flex-shrink-0" fill="none" stroke="
                                           currentColor" viewBox="0 0 24 24">
67                                          <path stroke-linecap="round" stroke-
                                               linejoin="round" stroke-width="2"
                                                d="M5 13l4 4L19 7"></path>
68                                      </svg>
69                                      <span>${item}</span>
70                                  </li>
71                              `).join('')}
72                          </ul>
73                      </div>
74                  `;
75              }
76          });
77
78          if (equipmentsHTML) {
79              equipmentsContainer.innerHTML = equipmentsHTML;
80          } else {
81              equipmentsContainer.innerHTML = '<p class="text-secondary">Aucun
                     quipement   sp cifi  </p>';
82          }
83      }
84  };
```

### Scripts d'Administration

**admin.js** Module centralisé d'administration qui définit les objets pour gérer l'authentification et les requêtes API.

**login.js** Gère la connexion des administrateurs.

**dashboard.js** Initialise et affiche les données du tableau de bord.

**cars.js** Gère les opérations CRUD pour les véhicules.

# Chapitre 3

# Modèle de Données

## 3.1 Collections Firebase

Les données sont stockées dans Firestore, organisées en collections :

### 3.1.1 Collection "vehicles"

Cette collection contient les véhicules dans le format principal :

Listing 3.1 – Structure d'un document dans la collection "vehicles"

```
{
  "id": "string",
  "name": "string",
  "brand": "string",
  "model": "string",
  "year": number,
  "price": number,
  "description": "string",
  "mileage": number,
  "fuel": "string",
  "transmission": "string",
  "power": number,
  "color": "string",
  "imageUrl": "string",
  "available": boolean,
  "equipments": {
    "confort": {
      "climatisation": boolean,
      "regulateur": boolean,
      "siegesChauf": boolean,
      "demarrageSansCle": boolean
    },
    "securite": {
      "freinageUrgence": boolean,
      "detectionAngles": boolean,
      "cameraRecul": boolean,
      "aideStationnement": boolean
    },
    "multimedia": {
      "ecranTactile": boolean,
      "carPlay": boolean,
      "bluetooth": boolean,
```

```
      "gps": boolean
    }
  },
  "createdAt": timestamp,
  "updatedAt": timestamp
}
```

### 3.1.2 Collection "admins"

Cette collection stocke les informations des administrateurs :

Listing 3.2 – Structure d'un document dans la collection "admins"

```
{
    "uid": "string",
  "email": "string",
  "isAdmin": boolean,
  "createdAt": timestamp
}
```

### 3.1.3 Migration entre Formats

Le projet maintient deux formats de données pour assurer la compatibilité :

- **Ancien format** : Collection "cars" avec une structure différente
- **Nouveau format** : Collection "vehicles" avec une structure optimisée

Un script de migration est fourni pour convertir les données de l'ancien format vers le nouveau :

Listing 3.3 – Script de migration des données

```
// migrateCarsToVehicles.js
const { db } = require('../backend/config/firebase-config');

async function migrateData() {
    try {
        console.log("D but de la migration des donn es...");

        // R cup rer tous les v hicules de l'ancienne collection
        const carsSnapshot = await db.collection('cars').get();

        if (carsSnapshot.empty) {
            console.log("Aucune donn e    migrer dans la collection 'cars
                '");
            return;
        }

        console.log(`${carsSnapshot.size} v hicules trouv s    migrer`);

        let migratedCount = 0;

        // Migrer chaque v hicule
        const batch = db.batch();

        carsSnapshot.forEach(doc => {
            const oldData = doc.data();

            // Cr er la nouvelle structure
```

```
27              const newVehicle = {
28                  name: oldData.name || `${oldData.specifications?.marque ||
                        ''} ${oldData.specifications?.modele || ''}`,
29                  brand: oldData.specifications?.marque || '',
30                  model: oldData.specifications?.modele || '',
31                  year: parseInt(oldData.specifications?.annee) || 0,
32                  price: parseFloat(oldData.prix) || 0,
33                  description: oldData.description || '',
34                  mileage: parseInt(oldData.specifications?.kilometrage) || 0,
35                  fuel: oldData.specifications?.carburant || '',
36                  transmission: oldData.specifications?.transmission || '',
37                  power: parseInt(oldData.specifications?.puissance) || 0,
38                  color: oldData.specifications?.couleur || '',
39                  imageUrl: oldData.imageUrl || '',
40                  available: oldData.status === 'disponible',
41
42                  // Convertir les   quipements
43                  equipments: {
44                      confort: {
45                          climatisation: oldData.equipements?.climatisation
                                === true,
46                          regulateur: oldData.equipements?.regulateur === true
                                ,
47                          siegesChauf: oldData.equipements?.siegesChauf ===
                                true,
48                          demarrageSansCle: oldData.equipements?.
                                demarrageSansCle === true
49                      },
50                      securite: {
51                          freinageUrgence: oldData.equipements?.
                                freinageUrgence === true,
52                          detectionAngles: oldData.equipements?.
                                detectionAngles === true,
53                          cameraRecul: oldData.equipements?.cameraRecul ===
                                true,
54                          aideStationnement: oldData.equipements?.
                                aideStationnement === true
55                      },
56                      multimedia: {
57                          ecranTactile: oldData.equipements?.ecranTactile ===
                                true,
58                          carPlay: oldData.equipements?.carPlay === true,
59                          bluetooth: oldData.equipements?.bluetooth === true,
60                          gps: oldData.equipements?.gps === true
61                      }
62                  },
63
64                  createdAt: oldData.createdAt || new Date(),
65                  updatedAt: new Date(),
66                  originalId: doc.id // R f rence    l'ID d'origine
67              };
68
69              // Ajouter    la nouvelle collection
70              const newRef = db.collection('vehicles').doc();
71              batch.set(newRef, newVehicle);
72
73              migratedCount++;
74          });
75
76          // Ex cuter les op rations par lots
```

```
77          await batch.commit();
78
79          console.log(`Migration termin e. ${migratedCount} v hicules
                migr s avec succ s.`);
80
81      } catch (error) {
82          console.error("Erreur lors de la migration:", error);
83      }
84  }
85
86  migrateData();
```

# Chapitre 4

# Fonctionnalités Principales

## 4.1 Interface Publique

L'interface publique du site offre plusieurs fonctionnalités clés pour les visiteurs :

### 4.1.1 Page d'Accueil

La page d'accueil présente :

- Une section héro avec slogan et appel à l'action
- Une sélection de véhicules en vedette
- Une section de recherche rapide
- Des témoignages clients
- Une présentation des avantages du service

### 4.1.2 Catalogue de Véhicules

La page de listings offre :

- Une liste complète des véhicules disponibles
- Des filtres de recherche (marque, prix, année, kilométrage)
- Une présentation en grille des véhicules avec photos et informations clés
- Une pagination pour naviguer dans la liste

### 4.1.3 Détails des Véhicules

La page de détails d'un véhicule comprend :

- Une galerie d'images du véhicule
- Les caractéristiques techniques complètes
- Les équipements organisés par catégorie (Confort, Sécurité, Multimédia)
- Une description détaillée
- Des suggestions de véhicules similaires
- Un formulaire ou des options de contact

### 4.1.4 Affichage des Équipements

Les équipements sont affichés de manière organisée et claire pour l'utilisateur, regroupés en trois catégories principales :

- **Confort** : Climatisation, régulateur de vitesse, sièges chauffants, démarrage sans clé
- **Sécurité** : Freinage d'urgence, détection des angles morts, caméra de recul, aide au stationnement
- **Multimédia** : Écran tactile, CarPlay/Android Auto, Bluetooth, GPS

## 4.2 Interface d'Administration

L'interface d'administration est réservée aux gestionnaires du site et offre des fonctionnalités complètes pour la gestion du parc automobile :

### 4.2.1 Authentification

Le système d'authentification utilise Firebase Authentication couplé à JWT :

- Connexion via email et mot de passe
- Génération d'un token JWT stocké localement
- Protection des routes d'administration
- Vérification des droits dans la collection "admins"

### 4.2.2 Tableau de Bord

Le tableau de bord administrateur présente :

- Des statistiques générales (nombre de véhicules, marques, prix moyen)
- L'état des véhicules (disponibles/vendus)
- Des liens rapides vers les différentes sections

### 4.2.3 Gestion des Véhicules

L'interface de gestion des véhicules permet :

- La consultation de la liste complète des véhicules
- L'ajout de nouveaux véhicules via un formulaire détaillé
- La modification des véhicules existants
- La suppression de véhicules
- La configuration détaillée des caractéristiques et équipements

#### Ajout et Modification de Véhicules

Le formulaire d'ajout/modification comprend :

- Informations de base (nom, marque, modèle, année, prix)
- Caractéristiques techniques (kilométrage, carburant, transmission, puissance)
- Lien vers l'image du véhicule
- Description détaillée
- Configuration des équipements par catégorie
- État du véhicule (disponible/vendu)

# Chapitre 5

# Sécurité

La sécurité est un aspect crucial de l'application et plusieurs mesures ont été mises en place :

## 5.1 Authentification

— **Firebase Authentication** : Utilisé pour l'authentification des administrateurs
— **JWT** : Tokens avec expiration pour maintenir les sessions
— **Vérification double** : Vérification dans Firebase Auth et dans la collection "admins"

## 5.2 Protection des Routes

— **Middleware d'authentification** : Vérifie la validité des tokens pour les routes sensibles
— **Vérification des droits** : S'assure que l'utilisateur a le rôle d'administrateur

## 5.3 Sécurité HTTP

— **Helmet.js** : Configure les en-têtes de sécurité HTTP
— **CORS** : Contrôle les requêtes cross-origin
— **Protection CSRF** : Implémentée sur le serveur

## 5.4 Validation des Données

— **Express Validator** : Valide les entrées côté serveur
— **Sanitisation** : Nettoie les données avant le stockage
— **Vérification des types** : S'assure que les données ont le format attendu

## 5.5 Règles Firestore

Des règles de sécurité sont configurées dans Firestore pour protéger les données :

Listing 5.1 – Règles Firestore

```
rules_version = '2';
service cloud.firestore {
```

```
  match /databases/{database}/documents {
    // Acc s public aux v hicules (lecture seule)
    match /vehicles/{vehicleId} {
      allow read: true;
      allow write: if request.auth != null && exists(/databases/$(database)/d
    }

    // Acc s public aux anciennes donn es de v hicules (lecture seule)
    match /cars/{carId} {
      allow read: true;
      allow write: if request.auth != null && exists(/databases/$(database)/d
    }

    // Protection des donn es administrateur
    match /admins/{userId} {
      allow read: if request.auth != null && request.auth.uid == userId;
      allow write: if false; //   criture   uniquement via le backend
    }
  }
}
```

# Chapitre 6

# Scripts Utilitaires

Le projet inclut plusieurs scripts utilitaires pour faciliter la configuration et la maintenance :

## 6.1 Script de Création d'Administrateur

Le script `createAdmin.js` permet de créer un utilisateur administrateur :

Listing 6.1 – Script de création d'administrateur

```
const { admin, db } = require('../backend/config/firebase-config');
require('dotenv').config();

async function createAdmin() {
  try {
    // Cr er un utilisateur dans Firebase Auth
    const email = "admin@autoelite.com";
    const password = "Admin123!"; // Utilisez un mot de passe fort en
       production

    let userRecord;

    try {
      // V rifier si l'utilisateur existe d j
      userRecord = await admin.auth().getUserByEmail(email);
      console.log('Utilisateur existant trouv :', userRecord.uid);
    } catch (error) {
      // Cr er l'utilisateur s'il n'existe pas
      userRecord = await admin.auth().createUser({
        email: email,
        password: password,
        emailVerified: true
      });
      console.log('Nouvel utilisateur cr    :', userRecord.uid);
    }

    // Ajouter l'utilisateur    la collection admins dans Firestore
    await db.collection('admins').doc(userRecord.uid).set({
      uid: userRecord.uid,
      email: email,
      isAdmin: true,
      createdAt: admin.firestore.FieldValue.serverTimestamp()
    });

```

```
34      console.log('Utilisateur administrateur cr    avec succ s!');
35      console.log('Email:', email);
36      console.log('Mot de passe:', password);
37
38    } catch (error) {
39      console.error('Erreur lors de la cr ation de l\'administrateur:', error
          );
40    }
41
42    process.exit();
43  }
44
45  createAdmin();
```

## 6.2 Script d'Ajout de Véhicules

Le script `addCars.js` permet d'ajouter des véhicules de démonstration :

Listing 6.2 – Script d'ajout de véhicules de démonstration

```
1  const { db } = require('../backend/config/firebase-config');
2
3  async function addDemoCars() {
4    try {
5      console.log("Ajout des v hicules de d monstration...");
6
7      const demoCars = [
8        {
9          name: "Renault Clio 5 TCe 100",
10         brand: "Renault",
11         model: "Clio",
12         year: 2021,
13         price: 15990,
14         description: "Magnifique Renault Clio 5 en excellent  tat , faible
             kilom trage...",
15         mileage: 25000,
16         fuel: "Essence",
17         transmission: "Manuelle",
18         power: 100,
19         color: "Rouge Flamme",
20         imageUrl: "https://example.com/images/clio.jpg",
21         available: true,
22         equipments: {
23           confort: {
24             climatisation: true,
25             regulateur: true,
26             siegesChauf: false,
27             demarrageSansCle: true
28           },
29           securite: {
30             freinageUrgence: true,
31             detectionAngles: false,
32             cameraRecul: true,
33             aideStationnement: true
34           },
35           multimedia: {
36             ecranTactile: true,
37             carPlay: true,
38             bluetooth: true,
```

```
            gps: false
          }
        }
      },
      // Autres v hicules...
    ];

    // Ajouter en lot
    const batch = db.batch();

    demoCars.forEach(car => {
      const carRef = db.collection('vehicles').doc();
      batch.set(carRef, {
        ...car,
        createdAt: admin.firestore.FieldValue.serverTimestamp(),
        updatedAt: admin.firestore.FieldValue.serverTimestamp()
      });
    });

    await batch.commit();
    console.log(`${demoCars.length} v hicules de d monstration ajout s
       avec succ s!`);

  } catch (error) {
    console.error("Erreur lors de l'ajout des v hicules:", error);
  }
}

addDemoCars();
```

# Chapitre 7

# Flux d'Utilisation

## 7.1 Flux Utilisateur Public

Le parcours utilisateur standard sur le site public se déroule comme suit :

1. L'utilisateur visite la page d'accueil (`index.html`)
2. Il peut parcourir les véhicules en vedette
3. Il peut accéder à la liste complète des véhicules (`listings.html`)
4. Il peut filtrer les véhicules par marque, prix, année
5. Il peut cliquer sur un véhicule pour voir ses détails (`car-details.html`)
6. Sur la page de détails, il peut voir toutes les informations, équipements et contacter le vendeur

## 7.2 Flux Administrateur

Le parcours d'un administrateur se déroule comme suit :

1. L'administrateur accède à `/admin` (redirigé vers `/admin/index.html`)
2. Il se connecte avec ses identifiants (email : admin@autoelite.com, password : Admin123 !)
3. Après connexion, il accède au tableau de bord (`dashboard.html`)
4. Il peut consulter les statistiques du site
5. Il peut accéder à la gestion des véhicules (`cars.html`)
6. Il peut voir tous les véhicules existants
7. Il peut ajouter un nouveau véhicule via le formulaire
8. Il peut modifier ou supprimer des véhicules existants
9. Lors de l'ajout/modification, il peut définir tous les détails et équipements

# Chapitre 8

# Compatibilité et Migrations

## 8.1 Gestion des Formats de Données

L'application gère deux formats de données de véhicules pour assurer la compatibilité :

- **Ancien format** (`cars`) : structure avec prix/specifications/equipements
- **Nouveau format** (`vehicles`) : structure avec price/brand/model/equipments

## 8.2 Stratégie de Migration

Le système est conçu pour :

1. Lire les données depuis les deux collections
2. Normaliser les données pour l'affichage cohérent
3. Écrire les nouvelles données dans le format `vehicles`
4. Maintenir les références croisées si nécessaire
5. Gérer intelligemment différentes orthographes (equipments/equipements)

# Chapitre 9

# Maintenance et Évolution

## 9.1 Remarques sur la Maintenance

— Les véhicules sont enregistrés dans `vehicles` avec le nouveau format
— La rétrocompatibilité est maintenue pour les anciens véhicules dans `cars`
— Les requêtes API gèrent intelligemment les deux collections
— Les équipements sont structurés par catégorie pour une meilleure organisation
— La normalisation des données a lieu à plusieurs niveaux pour assurer la cohérence

## 9.2 Évolutions Futures

Voici quelques pistes d'évolution pour le projet :

— Implémentation d'un système de recherche avancée
— Ajout d'un panneau de réservation ou d'achat en ligne
— Intégration d'un système de messagerie entre clients et vendeurs
— Développement d'une application mobile
— Système d'alerte pour les véhicules correspondant aux critères des utilisateurs
— Tableau de bord statistique avancé pour l'administration

# Chapitre 10

# Conclusion

Le projet MyCarDeals est une application web complète et moderne pour la vente de véhicules en ligne. Développé par Yassine TAMANI et Souheyl LABIDI dans le cadre d'un stage, il offre une interface publique attractive pour les acheteurs potentiels et une interface d'administration puissante pour les gestionnaires.

Grâce à l'utilisation de technologies modernes comme Express.js pour le backend et TailwindCSS pour le frontend, combinées à la puissance de Firebase pour la base de données et l'authentification, l'application offre une expérience utilisateur fluide et des fonctionnalités complètes pour la gestion des véhicules.

Le projet a été conçu avec une attention particulière à la sécurité, à la scalabilité et à la maintenance. La structure modulaire du code, l'organisation claire des données et la documentation complète en font une base solide pour d'éventuelles évolutions futures.

# Annexe A

# Guide d'Installation

## A.1 Prérequis

— Node.js (v14 ou supérieur)
— npm (v6 ou supérieur)
— Compte Firebase

## A.2 Configuration Firebase

1. Créer un projet Firebase sur la console Firebase
2. Activer Firestore et Firebase Authentication
3. Générer une clé de service privée (fichier JSON)
4. Placer le fichier de clé à la racine du projet sous le nom `serviceAccountKey.json`

## A.3 Installation des Dépendances

```
# Installer les d pendances
npm install
```

## A.4 Configuration des Variables d'Environnement

Créer un fichier `.env` à la racine du projet avec les variables suivantes :

```
PORT=3000
JWT_SECRET=votre_cle_secrete_pour_jwt
FIREBASE_TYPE=service_account
FIREBASE_PROJECT_ID=votre_project_id
FIREBASE_PRIVATE_KEY_ID=votre_private_key_id
FIREBASE_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\nVotre_Cle_Privee\n-----END
FIREBASE_CLIENT_EMAIL=votre_client_email
FIREBASE_CLIENT_ID=votre_client_id
FIREBASE_AUTH_URI=https://accounts.google.com/o/oauth2/auth
FIREBASE_TOKEN_URI=https://oauth2.googleapis.com/token
FIREBASE_AUTH_PROVIDER_CERT_URL=https://www.googleapis.com/oauth2/v1/certs
```

```
FIREBASE_CLIENT_CERT_URL=votre_client_cert_url
```

## A.5 Création d'un Administrateur

```
# Executer le script de cration d'administrateur
node scripts/createAdmin.js
```

## A.6 Lancement de l'Application

```
# Dmarrer le serveur
npm start
```

L'application sera accessible à l'adresse `http://localhost:3000`

# Annexe B

# Guide d'Utilisation

## B.1 Interface Publique

— Parcourir les véhicules sur la page d'accueil
— Utiliser la page de listings pour filtrer les véhicules
— Consulter les détails d'un véhicule sur sa page spécifique

## B.2 Interface d'Administration

— Accéder à `/admin` et se connecter avec les identifiants administrateur
— Utiliser le tableau de bord pour une vue d'ensemble
— Gérer les véhicules via l'interface dédiée (ajout, modification, suppression)