Las Dunas Rent A Car

Application Web dédiée à la location de voiture

Projet réaliser pour la prestation de l’examen du titre professionnel de

Concepteur Développeur d' Applications

Presente par
Esteban BARE

Introduction

Après ma première année en DWWM, j'ai souhaité approfondir mes compétences et en acquérir de nouvelles. La formation professionnelle CDA m'a permis d'aller plus loin, notamment en travaillant sur les bases de données NoSQL, le back-end en Java, la POO et le framework Spring Boot, qui apporte sécurité et fonctionnalités avancées aux applications.

J'ai également beaucoup progressé sur les API et les microservices, ce qui m'a donné une meilleure compréhension des architectures modernes. De plus, j'ai surmonté mes difficultés avec Docker, et je suis maintenant capable de l'utiliser efficacement pour gérer et déployer mes projets.

Cette expérience m'a aussi permis de renforcer mes soft skills grâce au travail en équipe et aux échanges avec différents formateurs, ce qui m'a donné de nouvelles perspectives de développement.

Enfin, tout ce que j'ai appris au cours de cette année m'a permis de réaliser mon projet Las Dunas Rent a Car, que je présenterai pour l'examen de CDA.

Liste de compétences du référentiel couvertes par le projet

Compétence 1 — Installer et configurer son environnement

Pour démarrer le projet, j'ai installé la JDK, Maven, Node.js et Angular CLI afin de lancer les microservices Spring Boot et le front Angular. Après avoir cloné le dépôt GitHub, j'ai configuré les fichiers application.properties et vérifié le bon fonctionnement des services via SpringCloudConfig et SpringCloudGateway.

Compétence 2 — Développer des interfaces utilisateur

Dans le dossier Front-Dunas, j'ai développé l'interface utilisateur en Angular. J'ai créé des composants, connecté les services aux APIs des microservices et géré la validation des formulaires. J'ai également ajouté la gestion des erreurs côté client et préparé des tests unitaires avec Jasmine/Karma pour sécuriser l'application.

Compétence 3 — Développer des composants métier

J'ai implémenté la logique métier dans différents microservices (Ms-Rental, Ms-Pricing, Ms-Promo, Ms-Comments). Chaque service a ses entités et traitements spécifiques, et la sécurité est gérée par Ms-Security. J'ai rédigé des tests unitaires avec JUnit pour valider les règles métier et documenté le fonctionnement des APIs.

Compétence 4 — Contribuer à la gestion d'un projet informatique

J'ai utilisé GitHub pour organiser et suivre mon travail (issues, branches fonctionnelles et commits réguliers). J'ai respecté des conventions de code et documenté l'avancement par des notes de suivi, ce qui m'a permis de gérer efficacement les différentes étapes du projet et d'assurer la traçabilité.

Compétence 5 — Analyser les besoins et réaliser des wireframes

J'ai étudié les besoins liés à la réservation et la gestion de véhicules puis créé des wireframes pour représenter l'enchaînement des écrans (recherche, réservation, gestion client). Ces wireframes m'ont permis d'anticiper la navigation et la structure fonctionnelle de l'application avant de passer au développement.

Compétence 6 — Définir l’architecture logicielle

J’ai conçu une architecture multicouche reposant sur une séparation claire : Angular pour le front, Spring Cloud Gateway pour le routage, les microservices Spring Boot pour la logique métier et Spring Cloud Config pour la configuration centralisée. Cette organisation permet une meilleure évolutivité et une gestion centralisée de la sécurité.

Compétence 7 — Concevoir et mettre en place une base relationnelle

J’ai défini le schéma relationnel de l’application (clients, véhicules, réservations, tarification) et préparé les scripts SQL nécessaires à l’initialisation des bases de données des microservices. J’ai également constitué un jeu de données de test et mis en place les règles d’accès et de droits utilisateurs.

Compétence 8 — Développer des composants d’accès aux données

J’ai créé les repositories et composants d’accès aux données pour chaque microservice en appliquant des requêtes sécurisées et en gérant les transactions. J’ai validé ces accès par des tests unitaires et d’intégration, afin de garantir la fiabilité et la sécurité du stockage des informations.

Compétence 9 — Préparer et exécuter les plans de tests

J’ai élaboré un plan de tests comprenant des tests unitaires, d’intégration et de non-régression. J’ai testé la communication entre les microservices via l’API Gateway et vérifié les contrôles de sécurité des endpoints. Enfin, j’ai exécuté des scénarios utilisateurs pour valider les fonctionnalités principales de l’application.

Compétence 10 — Préparer et documenter le déploiement

J’ai rédigé une procédure de déploiement détaillant les étapes de build (Maven pour les microservices, Angular pour le front), la configuration des environnements et les vérifications après installation. J’ai défini les environnements SIT/UAT/PROD et décrit les étapes de mise en ligne pour assurer une installation maîtrisée.

Compétence 11 — Contribuer à la mise en production (DevOps)

J’ai préparé l’intégration continue en organisant les services pour être conteneurisés et déployés plus facilement. J’ai documenté la structure pour mettre en place un pipeline CI/CD avec build, tests et packaging automatisés. L’architecture microservices est prête à être intégrée dans une chaîne DevOps complète avec Docker et GitHub Actions.

Résumé du Projet

Pour ce projet, je voulais m'inspirer d'expériences que tout le monde connaît déjà : les plateformes de location en ligne. On a tous déjà réservé une voiture pour un voyage ou un déplacement, mais souvent les sites existants manquent de clarté ou de modernité. C'est là que naît **Las Dunas Rent A Car** : une application pensée pour rendre la location de véhicules plus simple, plus intuitive et plus agréable.

L'idée est d'avoir une plateforme où l'utilisateur peut chercher une voiture, voir les tarifs en temps réel, profiter de promotions et même laisser des commentaires sur son expérience. Le but est de rendre le parcours utilisateur fluide : de la recherche jusqu'à la réservation, en passant par la gestion de son profil.

Le projet est construit sur une architecture moderne, avec un front-end en Angular pour offrir une navigation rapide et agréable, et des microservices en Java/Spring Boot pour gérer la logique métier (locations, tarifs, promos, avis, sécurité). Cette organisation permet de séparer clairement chaque fonction et de rendre l'ensemble évolutif et sécurisé.

En résumé, **Las Dunas Rent A Car** se veut être bien plus qu'un simple site de location : c'est une solution moderne et accessible, pensée pour que chacun puisse trouver et réserver une voiture facilement, tout en bénéficiant d'une expérience claire, conviviale et transparente.

Cahier des charges

Objectif

L’objectif principal de mon projet est de permettre aux utilisateurs de **réserver facilement une voiture en ligne**. Le site doit offrir une navigation simple et intuitive pour rechercher un véhicule, consulter les prix et finaliser une réservation en quelques clics.

Je souhaite également intégrer des **promotions et des réductions** afin d’attirer et fidéliser les clients. Les utilisateurs pourront laisser des **commentaires et avis** sur leur expérience, ce qui apportera plus de transparence et aidera les futurs clients à faire leur choix.

Pour améliorer l’expérience, il sera possible de **trier et filtrer les véhicules** (par prix, catégorie, disponibilité) et d’afficher des suggestions en fonction des besoins de l’utilisateur.

Enfin, il est indispensable de mettre en place un **système sécurisé d’inscription et de connexion** pour protéger les données personnelles. Un **back-office** sera prévu afin de gérer les utilisateurs, les réservations, les véhicules et les promotions.

User Stories

ID	En tant que…	Je veux…	Afin de…	Priorité
US1	Utilisateur	Créer un compte et me connecter	Accéder à mes réservations et mes infos	Haute
US2	Utilisateur	Rechercher un véhicule par date, type, prix	Trouver une voiture adaptée rapidement	Haute
US3	Utilisateur	Voir le détail d'un véhicule	Choisir le véhicule le plus adapté	Haute
US4	Utilisateur	Réserver un véhicule en ligne	Gagner du temps et éviter de me déplacer	Haute
US5	Utilisateur	Payer ma réservation de manière sécurisée	Finaliser mon achat en toute confiance	Haute
US6	Utilisateur	Bénéficier de promotions et réductions	Louer une voiture moins cher	Moyenne
US7	Utilisateur	Laisser un commentaire	Partager mon avis et aider les autres	Moyenne
US8	Utilisateur	Consulter les avis des autres	Faire un choix éclairé	Moyenne
US9	Utilisateur	Modifier ou annuler une réservation	Avoir de la flexibilité	Moyenne
US10	Utilisateur	Gérer mes infos personnelles	Garder mes données à jour	Moyenne
US11	Administrateur	Gérer les véhicules	Maintenir un catalogue toujours à jour	Haute
US12	Administrateur	Gérer les utilisateurs	Assurer la sécurité de la plateforme	Haute
US13	Administrateur	Gérer les réservations	Aider les utilisateurs en cas de problème	Haute
US14	Administrateur	Créer et gérer des promotions	Attirer et fidéliser des clients	Moyenne
US15	Administrateur	Consulter des statistiques	Suivre l'activité et améliorer le service	Basse

Fonctionnalités

En suivant les user stories, je peux lister les fonctionnalités principales du projet **Las Dunas Rent A Car** :

1. **Accueil** : Présentation des véhicules disponibles et mise en avant des promotions en cours.

2. **Recherche de véhicules** : Filtrer par date, type de véhicule, prix ou catégorie.

3. **Détail d'un véhicule** : Voir la description complète, photos, tarifs, promotions et avis clients.

4. **Réservation en ligne** : Choisir un véhicule, sélectionner les dates, confirmer et payer de façon sécurisée.

5. **Gestion du profil utilisateur** : Modifier ses informations personnelles, mot de passe et consulter l'historique de ses réservations.

6. **Commentaires et avis** : Publier un avis sur une location et consulter les retours d'autres utilisateurs.

7. **Promotions et réductions** : Affichage des offres spéciales pour fidéliser les clients.

8. **Administration** : Tableau de bord pour gérer les véhicules, les utilisateurs, les réservations, les promotions et consulter des statistiques.

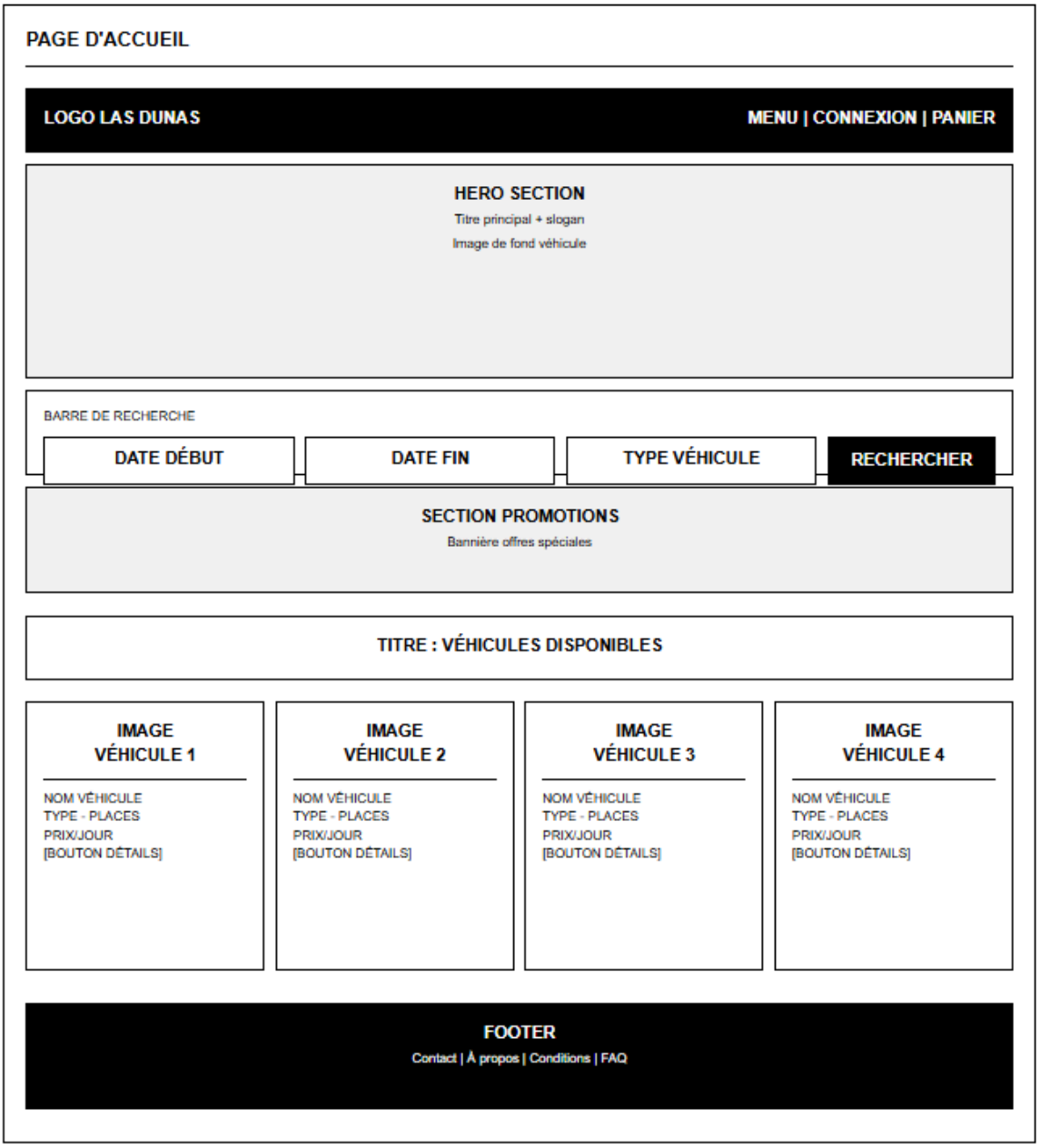
Rôles et permissions

1. **Utilisateur non authentifié** :
    - Consulter la liste des véhicules.
    - Accéder aux détails d'un véhicule et lire les avis.
    - Rechercher et filtrer les véhicules.
2. **Utilisateur authentifié** :
    - Réserver un véhicule en ligne et effectuer un paiement.
    - Gérer son profil et consulter son historique de réservations.
    - Publier et consulter des commentaires.
3. **Administrateur** :
    - Gérer les véhicules (ajouter, modifier, supprimer).
    - Gérer les utilisateurs (activer, suspendre).
    - Gérer les réservations (valider, annuler).
    - Créer et gérer des promotions.
    - Accéder à un tableau de bord avec statistiques et rapports.

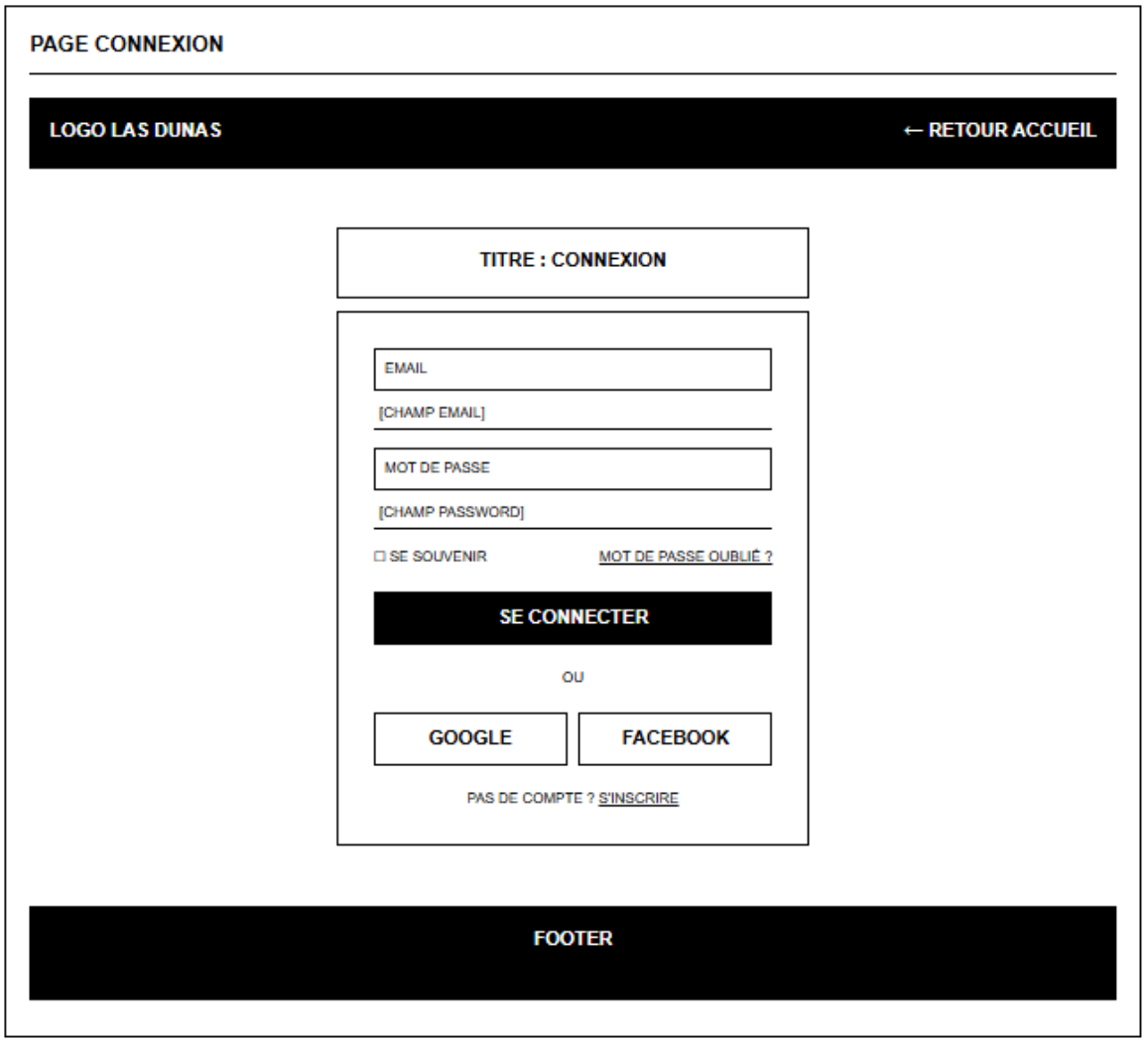
Wireframes

Voici quelques wireframes que j'ai créés afin d'anticiper la structure de mon projet et de planifier le développement de mon application.

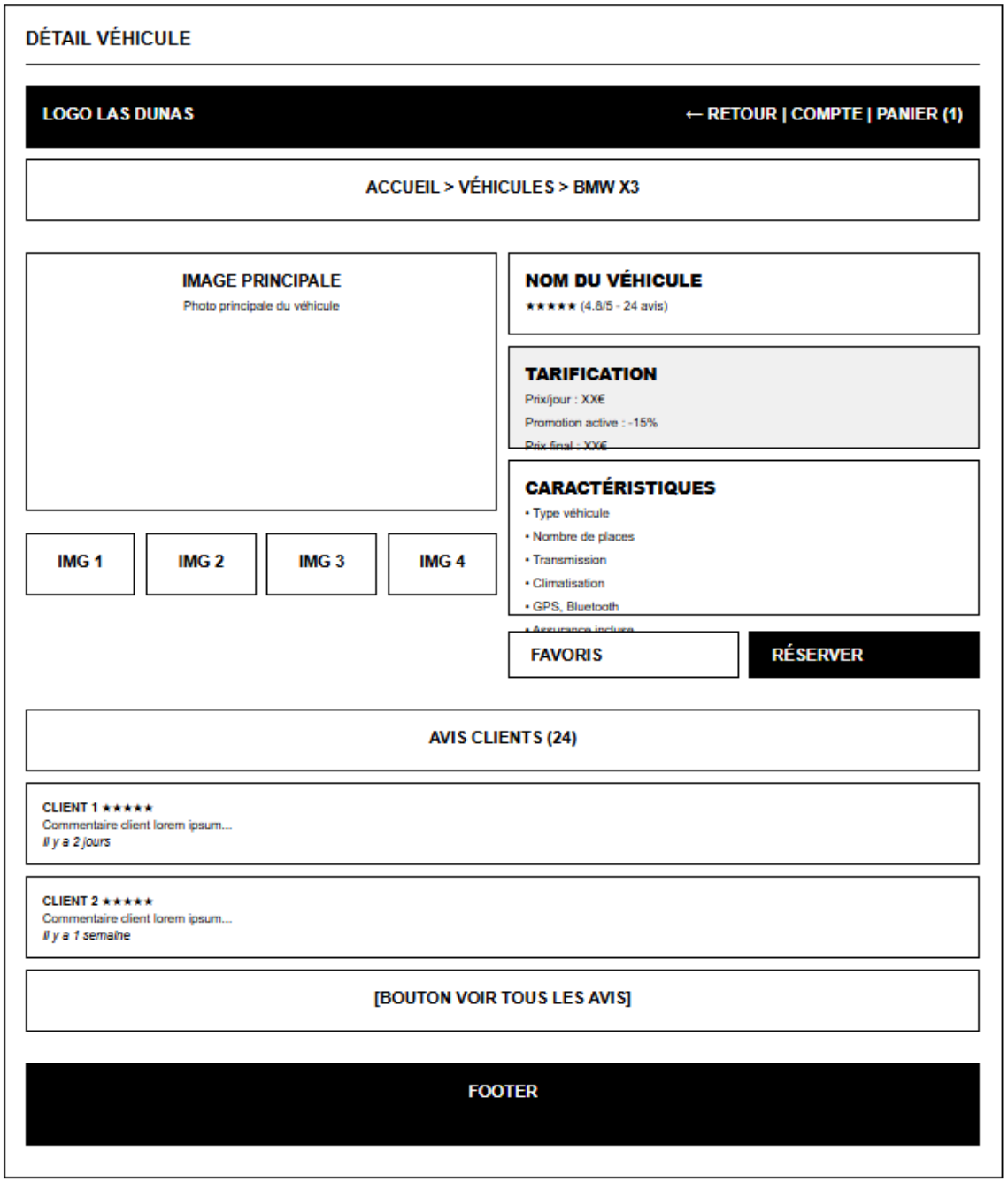
- Page d'accueil



- Page de connexion



- Détails d’un véhicule



Stack et technologies utilisées

Front-End

Pour le front-end, j'ai utilisé **Angular** avec **TypeScript**, **HTML** et **CSS**. Angular me permet de créer des interfaces dynamiques et modulaires, tout en profitant de son écosystème (Angular CLI, npm) pour gérer les dépendances et lancer le projet facilement.

Back-End

Pour le back-end, j'ai choisi **Java** avec le framework **Spring Boot**. C'est une solution fiable et largement utilisée pour développer des applications robustes et sécurisées. Spring Boot simplifie la création d'APIs REST et l'intégration avec d'autres outils de l'écosystème Java.

Base de données

Pour la base de données, j'utilise un **SGBD relationnel** avec **Spring Data JPA** pour l'accès aux données. JPA me permet de manipuler les entités directement en Java et de simplifier les opérations CRUD sans écrire trop de SQL.


Architecture de Las Dunas Rent A Car

1. Microservices Spring Boot

Le projet adopte une **architecture en microservices**, où chaque composant métier est encapsulé dans un service dédié. On retrouve notamment :

- **Ms-Rental** (gestion des locations)
- **Ms-Pricing** (gestion des tarifs)
- **Ms-Promo** (gestion des promotions)
- **Ms-Comments** (gestion des avis clients)
- **Ms-Security** (authentification et autorisation)

Chacun de ces services est développé en **Java** avec **Spring Boot**, ce qui permet une modularité, une évolutivité et une maintenance facilitées.

2. API Gateway avec Spring Cloud Gateway

L'**API Gateway** (via Spring Cloud Gateway) agit comme point d'entrée unique pour toutes les requêtes externes. Il oriente les appels vers les microservices appropriés, gère la sécurité des endpoints, les questions de CORS, et peut appliquer des filtres (auth, headers, etc.).

3. Configuration centralisée avec Spring Cloud Config

La configuration des microservices est externalisée et centralisée via **Spring Cloud Config**, ce qui permet de gérer de manière fluide et cohérente les paramètres (URL de DB, ports, clés, etc.) selon les environnements (dev, test, prod).

4. Communication inter-services via API REST

Les microservices communiquent entre eux via des **APIs REST**. Chaque service est autonome, avec son propre domaine fonctionnel et pouvant évoluer indépendamment des autres.

5. Base de données distribuée (une par service)

Chaque microservice gère sa propre **base de données relationnelle**, via **Spring Data JPA**. Cela permet de découpler la gestion des données, d'évoluer indépendamment, et de garantir la résilience et la cohérence du système.

Base de données

MCD / MLD :

Pour la modélisation de la base de données du projet **Las Dunas Rent A Car**, j'ai utilisé **Mermaid** afin de créer le schéma conceptuel (MCD) et le schéma logique (MLD).

Après la création du MCD, j'ai pu analyser les relations entre les différentes entités :

- Les **voitures** et les **catégories de véhicules** sont en relation Many-to-Many, car une voiture peut appartenir à plusieurs catégories et chaque catégorie peut regrouper plusieurs voitures. J'ai donc créé une table de pivot vehicle_category pour gérer cette relation.

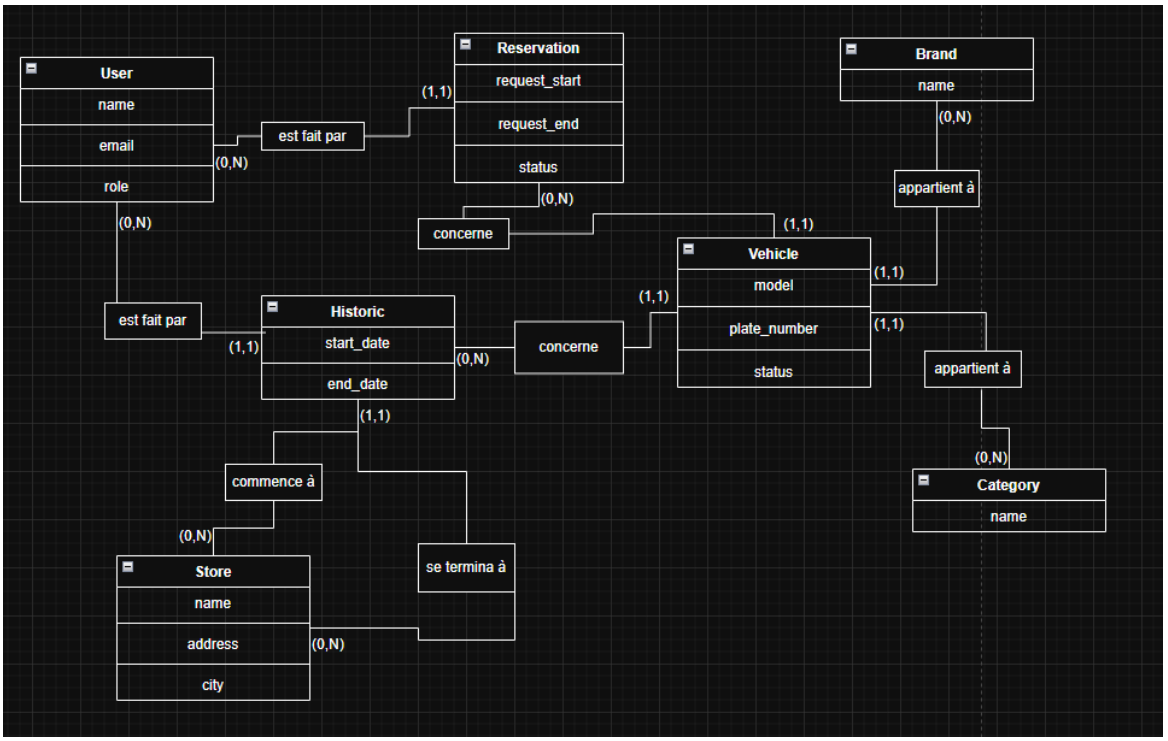
- Un **utilisateur** peut effectuer plusieurs **réservations**, mais une réservation n'appartient qu'à un seul utilisateur (relation One-to-Many).

- De même, un utilisateur peut laisser plusieurs **avis** sur les voitures, mais chaque avis est associé à un seul utilisateur et une seule voiture (relation One-to-Many pour chaque côté).

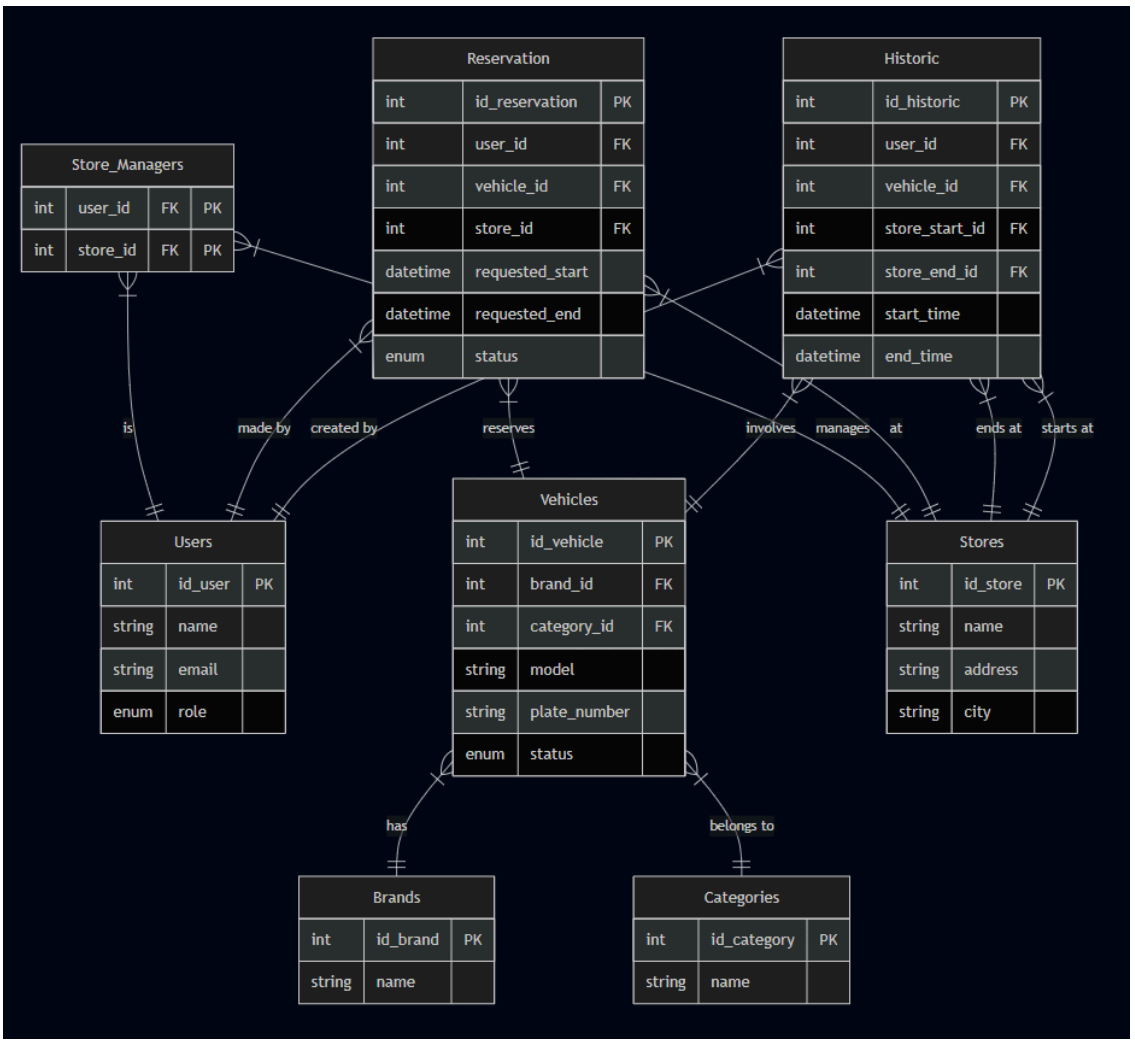
- Les relations plus simples, comme les liens entre les voitures et leurs caractéristiques, ont été modélisées directement dans le MCD avec des clés étrangères dans le MLD.

Étant donné que le projet est structuré autour d'une architecture **microservices**, je n'ai pas généré de MPD complet, car chaque microservice gère sa propre base de données et il serait trop complexe de représenter l'ensemble des données dans un seul MPD.

MCD


User
name
email
role
est fait par
(0,N)
(1,1)
Reservation
request_start
request_end
status
(0,N)
concerne
(1,1)
Brand
name
(0,N)
appartient à
Vehicle
model
plate_number
status
(1,1)
(1,1)
(1,1)
(1,1)
est fait par
(0,N)
Historic
start_date
end_date
(1,1)
(0,N)
concerne
(1,1)
appartient à
(0,N)
Category
name
(1,1)
commence à
(0,N)
Store
name
address
city
(0,N)
se termina à

MLD


Store_Managers
int user_id FK PK
int store_id FK PK
Reservation
int id_reservation PK
int user_id FK
int vehicle_id FK
int store_id FK
datetime requested_start
datetime requested_end
enum status
Historic
int id_historic PK
int user_id FK
int vehicle_id FK
int store_start_id FK
int store_end_id FK
datetime start_time
datetime end_time
is
made by
created by
reserves
involves
manages
at
ends at
starts at
Users
int id_user PK
string name
string email
enum role
Vehicles
int id_vehicle PK
int brand_id FK
int category_id FK
string model
string plate_number
enum status
Stores
int id_store PK
string name
string address
string city
has
belongs to
Brands
int id_brand PK
string name
Categories
int id_category PK
string name

Créer la base de données

Pour créer la base de données, j'utilise **JPA (Java Persistence API)** avec Spring Boot.

1. **Configuration de la source de données**
   Avant tout, je configure le fichier **application.properties** (ou **application.yml**) pour définir la connexion à la base de données. Par exemple, pour une base H2 en mémoire :

   - URL, utilisateur et mot de passe
   - Dialecte Hibernate
   - Option **ddl-auto=create** pour créer automatiquement les tables au démarrage de l'application

```
#spring.datasource.url=jdbc:h2:mem:rentaldb;DB_CLOSE_ON_EXIT=FALSE
#spring.datasource.driverClassName=org.h2.Driver
#spring.datasource.username=sa
#spring.datasource.password=password
#spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
#spring.h2.console.enabled=true

spring.datasource.url=jdbc:mysql://localhost:3308/las_rental
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

2. **Définition des entités**
   Chaque table de la base est représentée par une classe Java annotée avec **@Entity**. Les attributs de la classe correspondent aux colonnes de la table. Par exemple, une entité Vehicle possède des champs comme **id, brand** ou **model**.

```java
@Entity  Esteban
@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false)
    private String model;

    @Column(name = "plateNumber", nullable = false)
    private String plateNumber;

    @Column(name = "status", nullable = false)
    @Enumerated(EnumType.STRING)
    private StatusVehicle status;

    @Column(name = "price_par_day", precision = 10, scale = 2)
    private BigDecimal pricePerDay;

    @ManyToOne
    @JoinColumn(name = "brand_id", nullable = false)
    @JsonManagedReference
    private Brand brand;
```

3. **Création des repositories**
   Pour chaque entité, je crée un repository en étendant **JpaRepository**. Cela permet de gérer facilement les opérations CRUD sur la table correspondante.

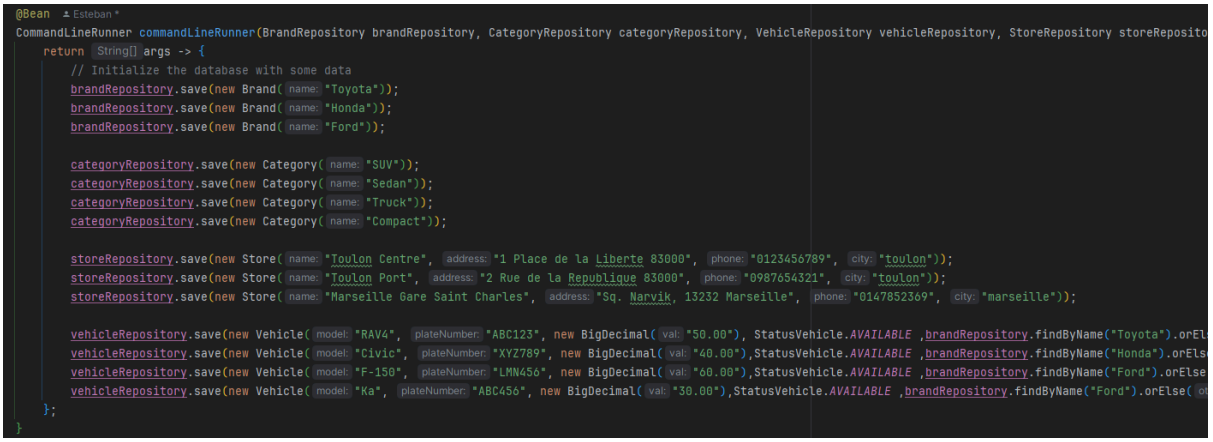
```java
@Repository  6 usages  Esteban
public interface VehicleRepository extends JpaRepository<Vehicle, Long> {
}
```

4. **Initialisation des données fictives**

Pour tester l'application, je crée des données fictives avec **JavaFaker**.

- Une classe annotée **@Component** implémente **CommandLineRunner** pour s'exécuter au démarrage.

- À l'aide de **Faker,** je génère des valeurs aléatoires et les insère dans la base avec les repositories.



Routes