ESME SUDRIA

Luxury Car

Projet JavaScript

Thibault Masse

Nicolas Félis

Bastien Fernandez

Table des matières

1. Introduction

Dans le cadre de notre second semestre de quatrième année à l'ESME, nous avons été amenés à réaliser un projet web complet dans un environnement de développement full stack. Notre groupe a décidé de concevoir **LuxuryCar**, une plateforme de location de voitures de luxe destinée aux particuliers. Le projet repose sur une architecture mêlant React.js pour le frontend et Node.js/Express pour le backend, avec une base de données relationnelle gérée via Sequelize et hébergée sur Google Cloud SQL.
LuxuryCar propose aux utilisateurs une interface intuitive pour consulter un catalogue de véhicules de luxe, filtrer selon la marque, effectuer des réservations pour une période définie, et consulter l'historique de leurs locations. Le site intègre également une interface d'administration sécurisée permettant la gestion des utilisateurs, des véhicules et des réservations.
Ce projet a constitué pour nous une opportunité de mobiliser un ensemble de compétences essentielles : conception d'API REST, structuration d'une base de données, gestion de l'authentification sécurisée, interactions entre client et serveur, et développement d'une interface utilisateur claire et ergonomique.

2. Backend

Le backend a été codé avec Node.js et Express.js, structuré autour d'une architecture RESTful. Nous avons utilisé Sequelize comme ORM afin de faciliter les interactions avec notre base de données Google Cloud. La configuration de la base de données est centralisée dans un fichier database.js, qui se connecte via les variables d'environnement.
Plusieurs modèles ont été définis :

- User.js : pour gérer les comptes utilisateurs, incluant les rôles (user/admin), l'authentification et les informations de base.
- Car.js : pour décrire les véhicules avec des attributs comme le nom, la marque, la vitesse maximale, le type (route ou circuit), le prix et l'image.
- Reservation.js : pour enregistrer chaque réservation, en lien avec un utilisateur et un véhicule, avec les dates de début et de fin, le statut (« en cours » ou « terminé »), et le prix calculé.

Côté sécurité et logique métier, nous avons :

- Empêché les réservations sur des dates passées.
- Empêché le chevauchement de réservation pour un même véhicule.
- Ajouté un mécanisme de mise à jour automatique du statut des réservations : toute réservation dont la date de fin est antérieure à la date actuelle est marquée comme « terminée ».
- La gestion complète des erreurs et des réponses serveur pour chaque action critique (inscription, réservation, suppression…).

Les routes sont divisées en plusieurs fichiers (UserRoutes.js, CarRoutes.js, ReservationRoutes.js) pour une meilleure lisibilité et maintenabilité. Chaque route est testée pour garantir le bon fonctionnement du backend.

Nous avons pris soin d'intégrer une sécurisation des mots de passe. Lors de l'inscription d'un nouvel utilisateur, le mot de passe est automatiquement haché à l'aide de la bibliothèque bcrypt avant d'être enregistré dans la base de données. Cette opération empêche tout stockage en clair, protégeant ainsi les données personnelles des utilisateurs en cas de fuite. Lors de la connexion, le mot de passe saisi est comparé au hash en base, assurant un processus

d’authentification fiable et sécurisé. Cette bonne pratique est aujourd’hui un standard incontournable dans le développement d’applications web.
Pour la gestion du paiement, nous avons intégré Stripe au sein du backend en utilisant leur API. Lorsqu'un utilisateur confirme une réservation depuis le frontend, une requête est envoyée au serveur qui crée une session de paiement Stripe. Cette session contient toutes les informations nécessaires : le montant total à régler, la description de la location, et l'URL de redirection en cas de succès ou d'échec.
Le backend génère cette session grâce au SDK Stripe Node.js et renvoie au frontend l'URL sécurisée de paiement. Lorsque le paiement est validé, Stripe déclenche un webhook qui permet au serveur de confirmer automatiquement la transaction et d'enregistrer la réservation dans la base de données. Cette approche garantit que seules les réservations effectivement payées sont enregistrées.
Ainsi, le backend pour la partie paiement assure :

- La création de la session de paiement Stripe.
- La réception et le traitement du webhook Stripe après paiement réussi.
- L'enregistrement en base de données d'une réservation uniquement après validation du paiement.

3.Frontend

Concernant le frontend, nous avons utilisé React.js, ce qui nous a permis de construire une interface dynamique, ergonomique et responsive. L'interface est divisée en plusieurs fichiers source, chacun ayant une fonctionnalité spécifique :

- App.js est le cœur de l'application React. Il gère les données principales de l'application, effectue les appels API nécessaires, s'occupe des interactions utilisateur et affiche les composants de la page (Header, CarCarousel, AdminPanel, etc.).
  C'est un fichier qui centralise la logique de notre application.

- Api.js est l'interface qui permet la communication entre le frontend et le backend. Il sert à envoyer les requêtes HTTP via Axios vers le serveur (backend) et à interagir indirectement avec la base de données.

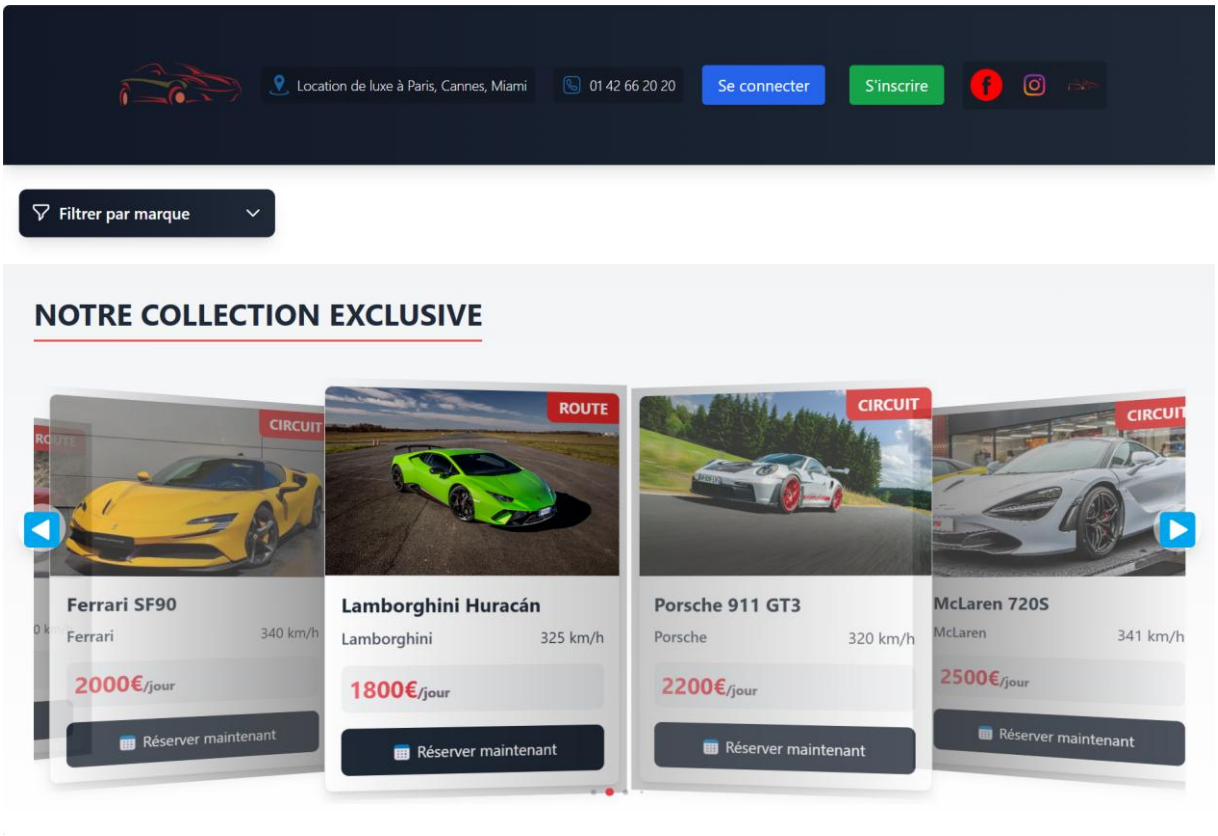
- Header.js affiche l'en-tête de la page, comprenant le logo, les coordonnées, les réseaux sociaux, ainsi que les boutons de connexion, d'inscription et de déconnexion.

- Index.css est la feuille de style principale de l'application. Elle est combinée avec Tailwind CSS pour personnaliser les animations, les effets visuels et certains styles globaux.

- CarCarousel.js : Il s'agit du carrousel principal de l'application, affichant le catalogue de voitures proposé aux clients. Il intègre également le mécanisme de réservation pour chaque véhicule.

- BranFilter.js : C'est un composant de filtrage permettant de trier les voitures affichées dans le carrousel en fonction de leur marque.

Vu global de l’application :



Fonctionnalités utilisateur :

- Connexion / inscription sécurisée avec validation côté client.
- Affichage des voitures disponibles avec détails.
- Réservation d’un véhicule avec vérification de disponibilité.
- Suivi des réservations (voiture, date, prix, statut).
- Possibilité d’annuler une réservation.

Fonctionnalités administrateur :

- Panneau d’administration accessible uniquement par les comptes admin.
- Ajout / suppression de voitures.

- Gestion des utilisateurs (suppression ou promotion en admin).
- Suivi global des réservations, avec option de suppression.

Le composant AdminPanel.js centralise toutes les fonctionnalités de gestion. Les appels API sont effectués via api.js, un fichier intermédiaire qui organise les requêtes HTTP en fonctions claires et réutilisables.

Un soin particulier a été apporté à l'expérience utilisateur : messages d'erreur explicites, mise à jour automatique des listes après chaque action, et indications visuelles lors des actions (confirmation, alertes, etc.)

Pour la réservation côté frontend on peut remarquer tous ces éléments :

- Lorsqu'il clique sur "Réserver", une fenêtre modale s'ouvre pour sélectionner les dates de début et de fin de la location.
- En cliquant sur "Confirmer la réservation", une requête API est envoyée au backend pour créer une session de paiement Stripe.
- Une fois la session créée, le frontend redirige automatiquement l'utilisateur vers la page de paiement Stripe sécurisée grâce à la librairie Stripe.js.
- L'utilisateur saisit ses coordonnées bancaires directement sur l'interface Stripe.
- À la validation du paiement, Stripe redirige automatiquement l'utilisateur vers une page de confirmation du site LuxuryCar.

4. Conclusion

Ce projet a été extrêmement enrichissant sur plusieurs aspects. Il nous a permis de mettre en pratique nos compétences techniques en développement web full stack, mais également de travailler en équipe sur un projet concret.

Nous avons appris à :

- Concevoir une architecture REST propre et scalable.
- Gérer une base de données relationnelle avec Sequelize.
- Sécuriser un site avec des rôles et des restrictions.
- Intégrer une interface frontend intuitive avec React.
- Maintenir une cohérence entre le backend et le frontend.

Nous sommes fiers du résultat obtenu. La plateforme LuxuryCar est fonctionnelle, modulaire et extensible. Ce projet nous a également sensibilisés à l'importance de la collaboration, du versionning (via GitHub) et de la rigueur dans la structuration du code.

5. Perspective d’évolution

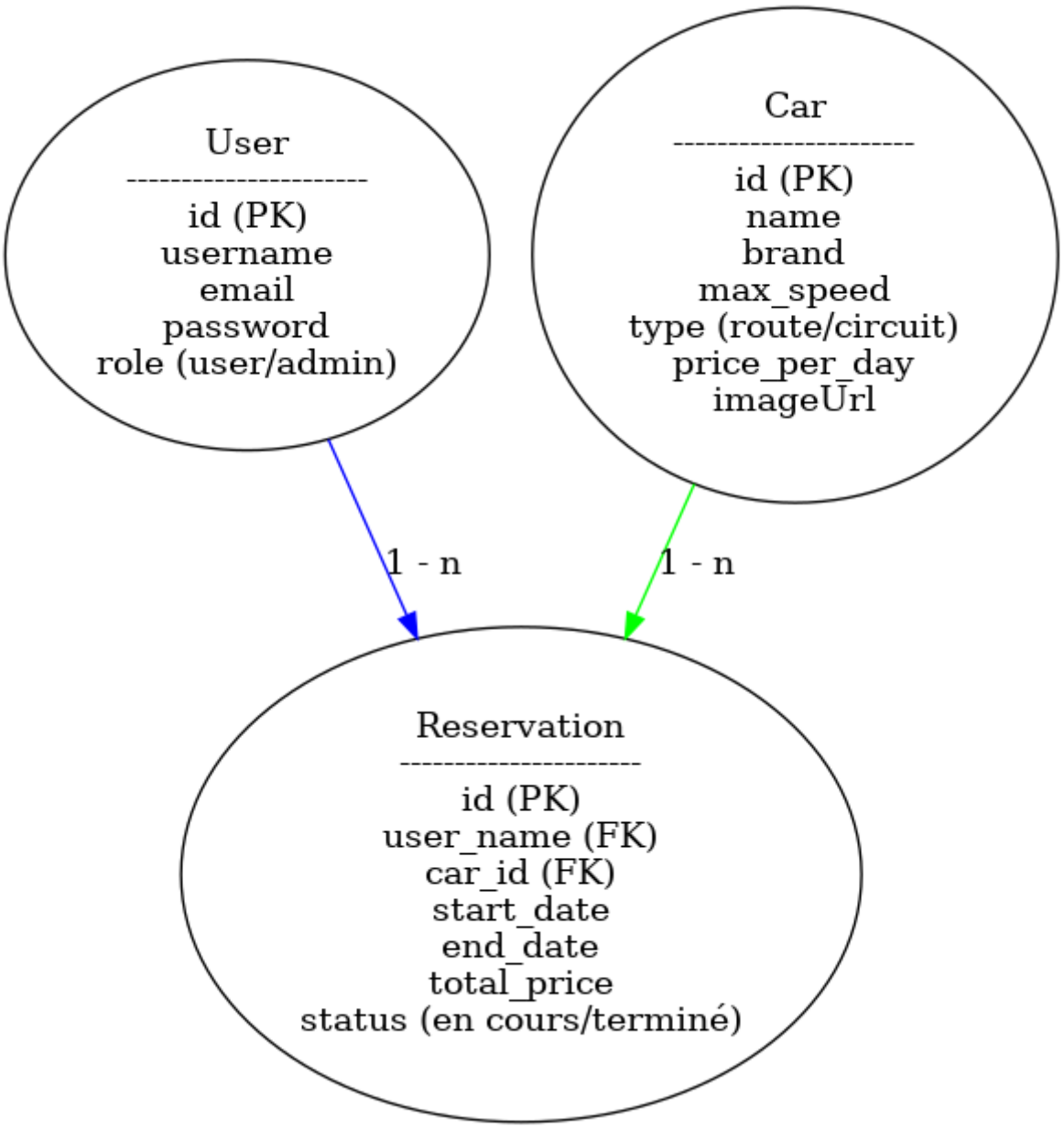
Bien que la plateforme LuxuryCar soit déjà fonctionnelle et complète dans sa version actuelle, plusieurs pistes d’évolution peuvent être envisagées pour enrichir l’expérience utilisateur.

Ensuite, un calendrier interactif pour chaque voiture serait un atout ergonomique majeur. Cela permettrait de visualiser en un coup d'œil les disponibilités d’un véhicule, d’éviter les erreurs de sélection et de fluidifier le processus de réservation.

De plus, la gestion d’un système d’évaluation et de notation des véhicules et du service pourrait renforcer la confiance entre utilisateurs et administrateurs.

Enfin, une fonctionnalité avancée de gestion de flotte pourrait être développée, incluant le suivi des entretiens, les kilomètres parcourus, etc…

Côté technique, le déploiement de l’application sur une plateforme cloud (comme Vercel pour le frontend et Render/Heroku/Google Cloud Run pour le backend) permettrait une accessibilité publique. La mise en place de tests unitaires et d’intégration via Jest et Supertest renforcerait la robustesse du code. Un système d’authentification plus avancé (comme JWT) pourrait également être introduit pour mieux sécuriser le site.


User
id (PK)
username
email
password
role (user/admin)
Car
id (PK)
name
brand
max_speed
type (route/circuit)
price_per_day
imageUrl
1 - n
1 - n
Reservation
id (PK)
user_name (FK)
car_id (FK)
start_date
end_date
total_price
status (en cours/terminé)