Dossier Projet

Hoffmann Michel

GARAGE PARROT

TOULOUSE

CAHIER DES CHARGES

Projet Principal

Lien TRELLO : https://trello.com/b/jdyXm3Ws/conduite-de-projet

Lien github côté Front : https://github.com/Michelhof1978/GarageParrot

Lien github côté back : https://github.com/Michelhof1978/GarageBack

Lien Figma : https://www.figma.com/file/rhs91pTc1st89ApCbMHoel/GARAGE-PARROT?type=design&node-id=0-1&mode=design&t=b8DACnrXU32mmG6U-0

Projet Cv en ligne : https://cvmichel-hoffmann.fr/

Lien github : https://github.com/Michelhof1978/cvMichel

Technologies Utilisées : Html, Css, Bootstrap, javascript et php sans base de données.

Table de matière

Documents joints avec le projet :

-Maquette Figma

-User stories admin + Utilisateurs

-Cahier des charges

-Gestion des tâches avec Trello

-Diagramme de cas d'utilisation

-Diagramme de séquences

-Diagramme de classes avec Mysql

Technologies utilisées pour le projet :

-Front : React, Bootstrap, Html et Css

-Back : Php 8, MySqlworkbench et Xampp

1.0 Description du projet :

DESCRIPTION DU PROJET GARAGE PARROT

Le projet vise à développer une application web vitrine pour le **Garage V. Parrot**, un établissement automobile situé à Toulouse, qui propose une gamme de services incluant la réparation automobile, l'entretien et la vente de véhicules d'occasion. L'objectif principal de cette application est de mettre en avant la qualité des services offerts par le garage.

Voici un aperçu des principales fonctionnalités à implémenter :

1.1 Connexion : L'application doit permettre à l'administrateur et aux employés de se connecter via un formulaire d'accès à l'espace professionnel. Les identifiants requis seront le nom d'utilisateur et un mot de passe sécurisé. Seul l'administrateur principal sera autorisé à créer, modifier ou supprimer des comptes pour ses employés. Dans le tableau de bord, toutes les sections seront accessibles à tous les utilisateurs, à l'exception de la section "gestion des comptes employés", qui sera réservée à l'administrateur en chef pour gérer les comptes de ses employés.

1.2 Présentation des services : La page d'accueil de l'application devra clairement présenter les différents services de réparation automobile proposés par le garage, tant dans la barre de navigation que sous forme de diaporama sur la page d'accueil. L'administrateur devra avoir la capacité d'ajouter, de modifier ou de supprimer ces informations depuis son espace d'administration. De plus, seul le compte administrateur pourra consulter l'historique des modifications apportées par les employés, afin de retracer toute manipulation effectuée par un employé en cas de problème.

1.3 Définir les horaires d'ouverture : Les horaires d'ouverture du garage seront clairement affichés sur le site, de préférence dans le pied de page. L'administrateur aura la possibilité de spécifier ces horaires depuis son espace dédié. Les horaires seront affichés avec une coupure entre **12h et 14h** pour indiquer les heures de fermeture pendant cette période.

1.4 Présentation des voitures d'occasion : L'application devra afficher les véhicules disponibles à la vente, accompagnés d'une galerie de photos, de descriptions détaillées et d'informations techniques telles qu'un tableau de caractéristiques, la liste des équipements et options installés. Chaque véhicule devra présenter son prix, une image principale, l'année de mise en circulation et le kilométrage. Les employés auront la possibilité d'ajouter de nouvelles voitures depuis leur espace dédié. Chaque nouvelle voiture ajoutée sera automatiquement mise en avant sur la page d'accueil sous forme de carte, permettant ainsi de présenter les nouveautés aux utilisateurs. Les voitures seront triées automatiquement par ordre d'arrivée.

1.5 Filtrage de la liste des véhicules d'occasion : Pour simplifier la recherche des visiteurs, un système de filtres sera mis en place. Les visiteurs pourront affiner leurs résultats en fonction d'une fourchette de prix **(slider)**, d'un nombre de kilomètres parcourus ou de l'année de mise en circulation. Pour offrir la meilleure expérience utilisateur possible, les filtres seront appliqués en temps réel, **sans nécessiter le rechargement de la page**. Les voitures seront classées en cinq catégories : **Berline, SUV, Familiale, Utilitaire et Citadine**, avec des options de mise en avant sous forme de **boutons** (cases à cocher) pour une expérience utilisateur optimale.

1.6 Contact avec l'atelier : Les visiteurs devront avoir la possibilité de contacter facilement le garage, que ce soit par téléphone ou en remplissant un formulaire de contact qui sera ensuite redirigé vers le service approprié. L'utilisateur devra fournir son **nom, prénom, adresse e-mail, numéro de téléphone et un message**. De plus, une **liste déroulante** permettra de choisir vers quel service l'utilisateur souhaite diriger sa demande. En cas de contact concernant un véhicule en vente**, l'ID ou la référence du véhicule et la fiche du véhicule seront automatiquement inclus** dans le formulaire, accompagnés du message de l'utilisateur. Cela permettra à l'administrateur de retrouver la voiture plus facilement et de répondre de manière plus efficace à la demande de l'utilisateur, ce qui permettra de gagner du temps. L'application intégrera également des mécanismes de conformité au **RGPD** pour la protection des informations personnelles et un **captcha** de Google pour renforcer la sécurité. pour prouver que c'est un humain et éviter que le message tombe dans les spams. Un message s'ouvrira dans une autre page pour confirmation d'envois.

1.7 Collecte des témoignages clients : Une section dédiée sera créée pour permettre aux visiteurs de laisser des témoignages. Ces témoignages devront être soumis à une modération préalable par un employé du garage avant d'être affichés sur la page d'accueil. Les employés auront également la possibilité de laisser des avis client via l'espace Pro. Un système de notation à cinq étoiles sera mis en place, permettant aux utilisateurs de sélectionner leur note, d'ajouter leur nom et de laisser un commentaire. Les informations personnelles ne seront pas affichées publiquement pour préserver la vie privée des utilisateurs. Diffusion des avis sous forme de **carrousel.**

1.8 Affichage :

-Cookies (Acceptation **RGPD**, affichage page d'accueil lors de la première connexion)

Principales utilisations des cookies :

1. **Maintenir la session de l'utilisateur**
2. **Améliorer l'expérience utilisateur**
3. **Suivi de l'activité et analyse**
4. **Publicité ciblée**
5. **Mesure du trafic**

-Mentions légales

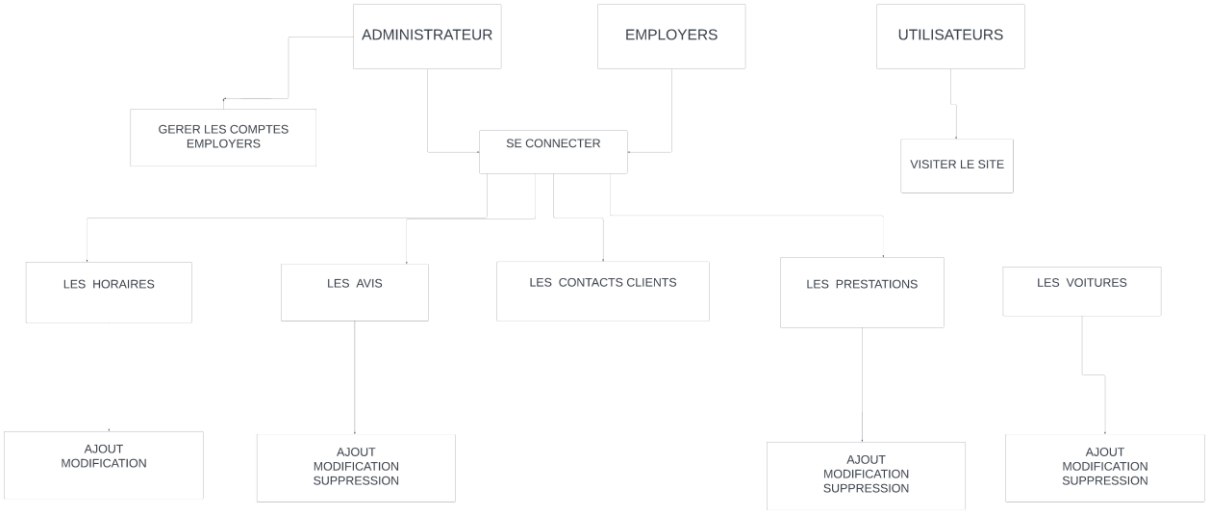
-Condition générales de ventes

-Politique de confidentialité

2.0 Diagrammes :

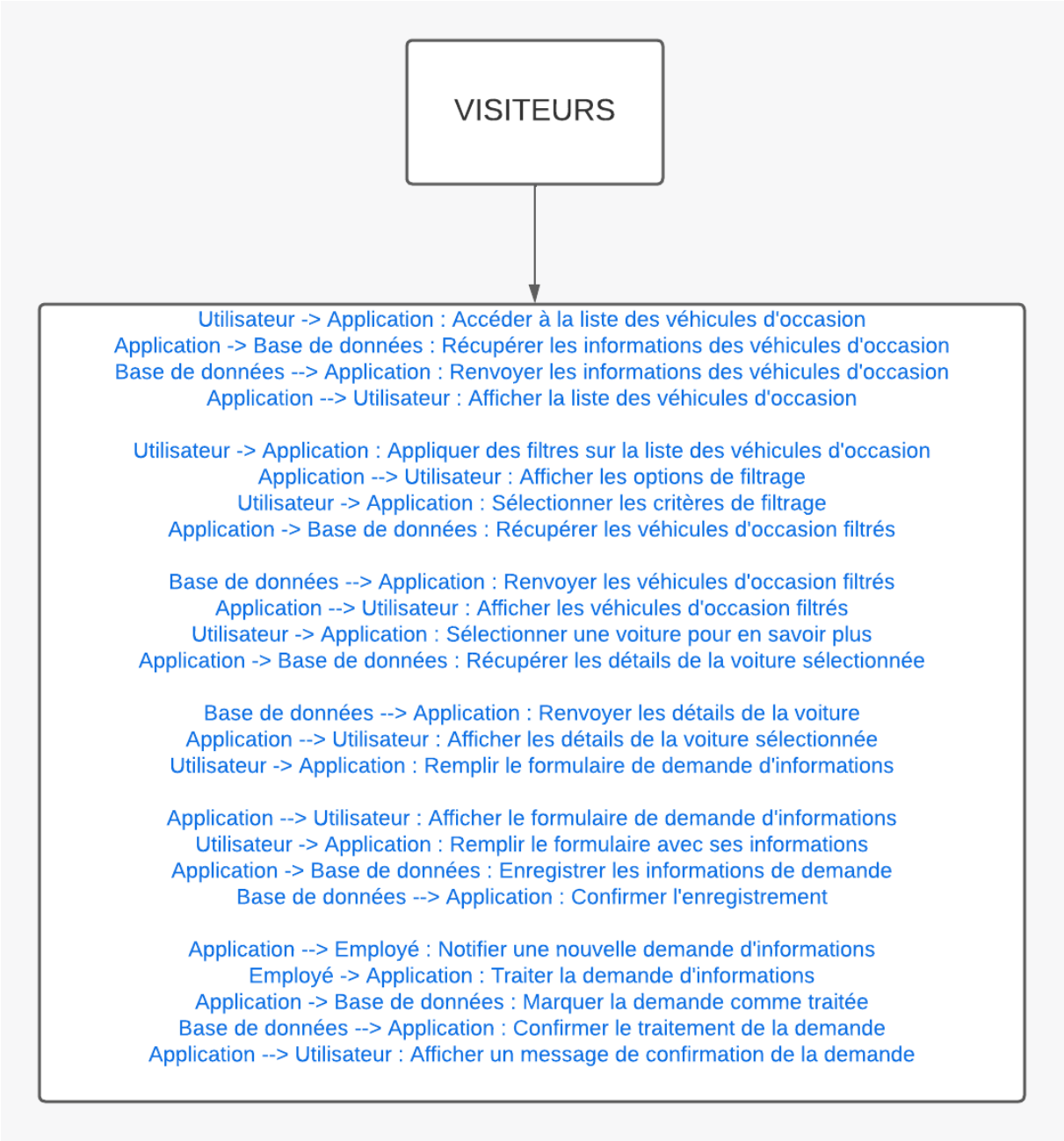
DIAGRAMMES

DIAGRAMME DE CAS D'UTILISATION

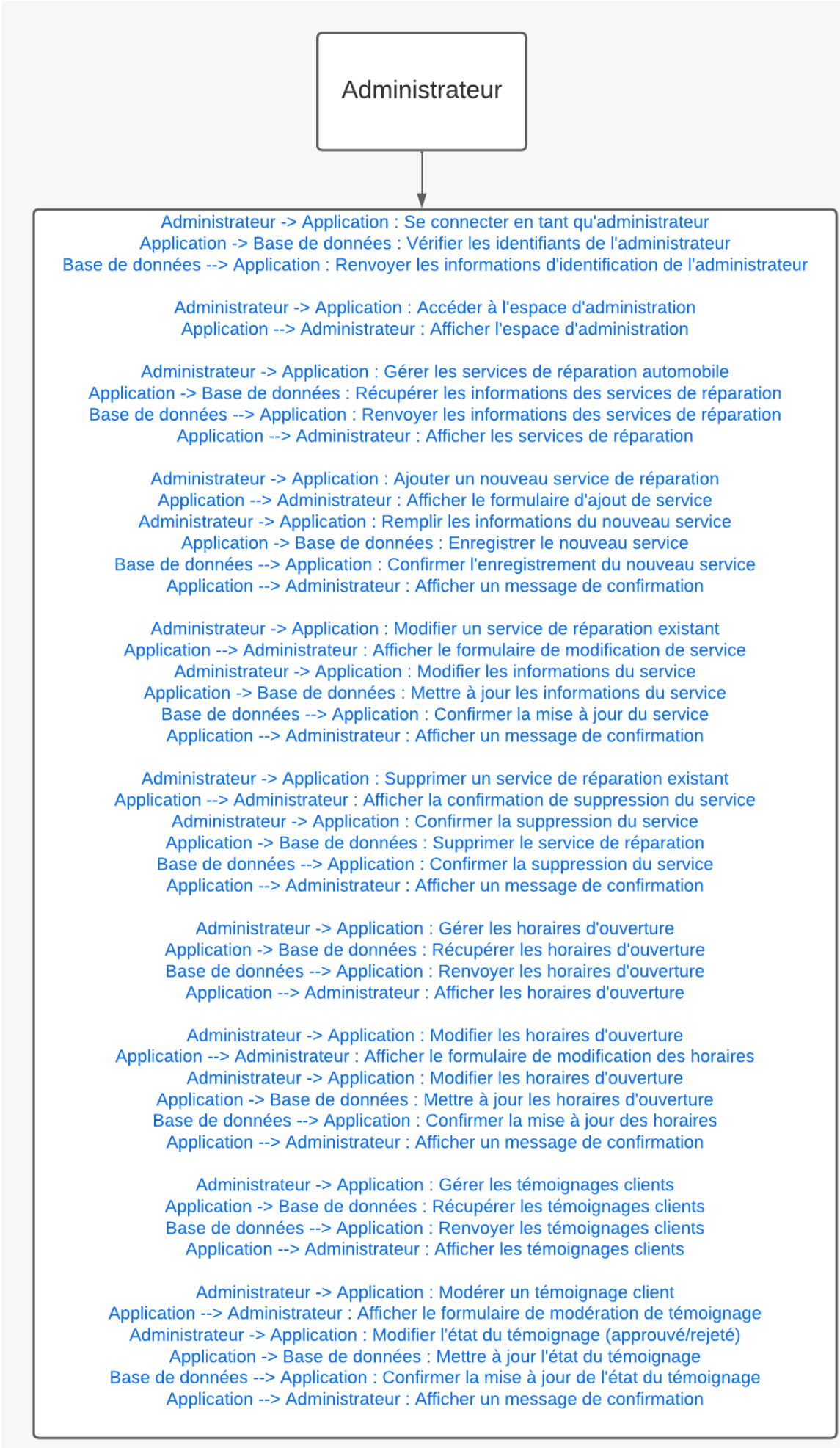


DIAGRAMMES DE SEQUENCES

Ces diagrammes de modélisation utilisés représentent les interactions entre différents objets ou acteurs dans l'application. Ils permettent de visualiser la séquence des messages échangés entre ces objets ou acteurs dans le temps. Ils sont particulièrement utiles pour décrire le flux de contrôle entre les différentes parties d'un système et pour comprendre le comportement dynamique d'un processus comme ci-dessous.



Dans ce diagramme de séquence, je décris comment un nouveau visiteur peut découvrir la liste des véhicules d'occasion, appliquer des filtres pour affiner les résultats, sélectionner une voiture spécifique, et remplir le formulaire de demande d'informations pour en savoir plus sur cette voiture. Mon application enregistre ensuite la demande et la notifie à un employé du garage, qui la traitera et la marquera comme traitée dans la base de données.


Administrateur
Administrateur -> Application : Se connecter en tant qu'administrateur
Application -> Base de données : Vérifier les identifiants de l'administrateur
Base de données --> Application : Renvoyer les informations d'identification de l'administrateur
Administrateur -> Application : Accéder à l'espace d'administration
Application --> Administrateur : Afficher l'espace d'administration
Administrateur -> Application : Gérer les services de réparation automobile
Application -> Base de données : Récupérer les informations des services de réparation
Base de données --> Application : Renvoyer les informations des services de réparation
Application --> Administrateur : Afficher les services de réparation
Administrateur -> Application : Ajouter un nouveau service de réparation
Application --> Administrateur : Afficher le formulaire d'ajout de service
Administrateur -> Application : Remplir les informations du nouveau service
Application -> Base de données : Enregistrer le nouveau service
Base de données --> Application : Confirmer l'enregistrement du nouveau service
Application --> Administrateur : Afficher un message de confirmation
Administrateur -> Application : Modifier un service de réparation existant
Application --> Administrateur : Afficher le formulaire de modification de service
Administrateur -> Application : Modifier les informations du service
Application -> Base de données : Mettre à jour les informations du service
Base de données --> Application : Confirmer la mise à jour du service
Application --> Administrateur : Afficher un message de confirmation
Administrateur -> Application : Supprimer un service de réparation existant
Application --> Administrateur : Afficher la confirmation de suppression du service
Administrateur -> Application : Confirmer la suppression du service
Application -> Base de données : Supprimer le service de réparation
Base de données --> Application : Confirmer la suppression du service
Application --> Administrateur : Afficher un message de confirmation
Administrateur -> Application : Gérer les horaires d'ouverture
Application -> Base de données : Récupérer les horaires d'ouverture
Base de données --> Application : Renvoyer les horaires d'ouverture
Application --> Administrateur : Afficher les horaires d'ouverture
Administrateur -> Application : Modifier les horaires d'ouverture
Application --> Administrateur : Afficher le formulaire de modification des horaires
Administrateur -> Application : Modifier les horaires d'ouverture
Application -> Base de données : Mettre à jour les horaires d'ouverture
Base de données --> Application : Confirmer la mise à jour des horaires
Application --> Administrateur : Afficher un message de confirmation
Administrateur -> Application : Gérer les témoignages clients
Application -> Base de données : Récupérer les témoignages clients
Base de données --> Application : Renvoyer les témoignages clients
Application --> Administrateur : Afficher les témoignages clients
Administrateur -> Application : Modérer un témoignage client
Application --> Administrateur : Afficher le formulaire de modération de témoignage
Administrateur -> Application : Modifier l'état du témoignage (approuvé/rejeté)
Application -> Base de données : Mettre à jour l'état du témoignage
Base de données --> Application : Confirmer la mise à jour de l'état du témoignage
Application --> Administrateur : Afficher un message de confirmation

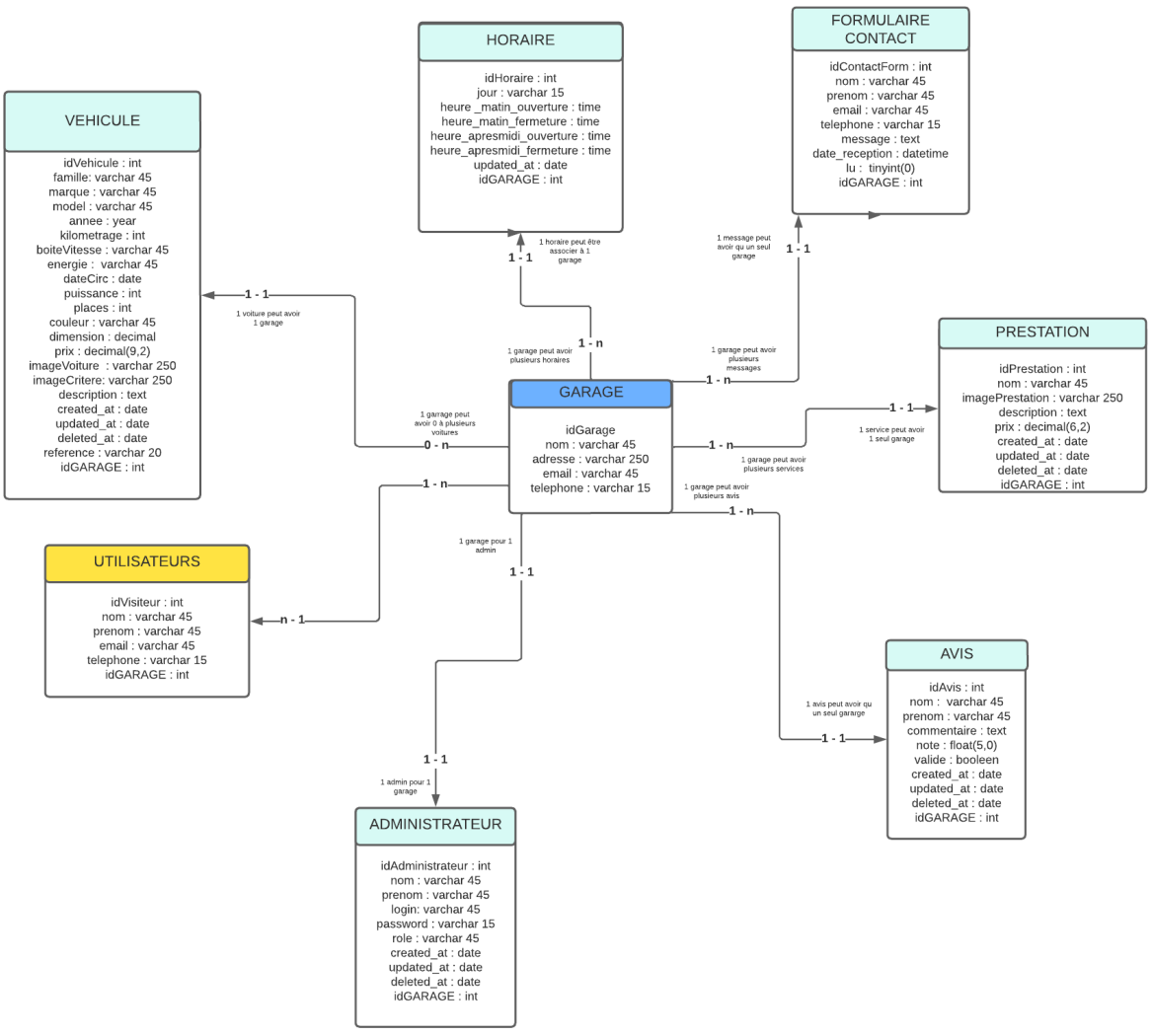
Dans ce diagramme de séquence, je décris comment je peux me connecter à l'application, accéder à l'espace d'administration, gérer les services de réparation automobile (ajouter, modifier, supprimer), gérer les horaires d'ouverture, et modérer les témoignages clients.

Mon application interagit avec la base de données pour récupérer et mettre à jour les informations pertinentes, et elle affiche des messages de confirmation pour m'indiquer le succès des opérations effectuées.

DIAGRAMME DE CLASSES

Ce diagramme de classes illustre la structure de base du modèle de données pour le

Garage V. Parrot



Dans ce diagramme de classes que je viens de réaliser, le Garage est la classe principale et je lui ai attribué des méthodes pour gérer les **services**, les **horaires**, les **véhicules d'occasion**, les **contacts** avec le service client ainsi que les **avis clients**.

-**La classe Service** représente un service de réparation automobile, et j'ai défini des méthodes pour y accéder et modifier ses attributs.

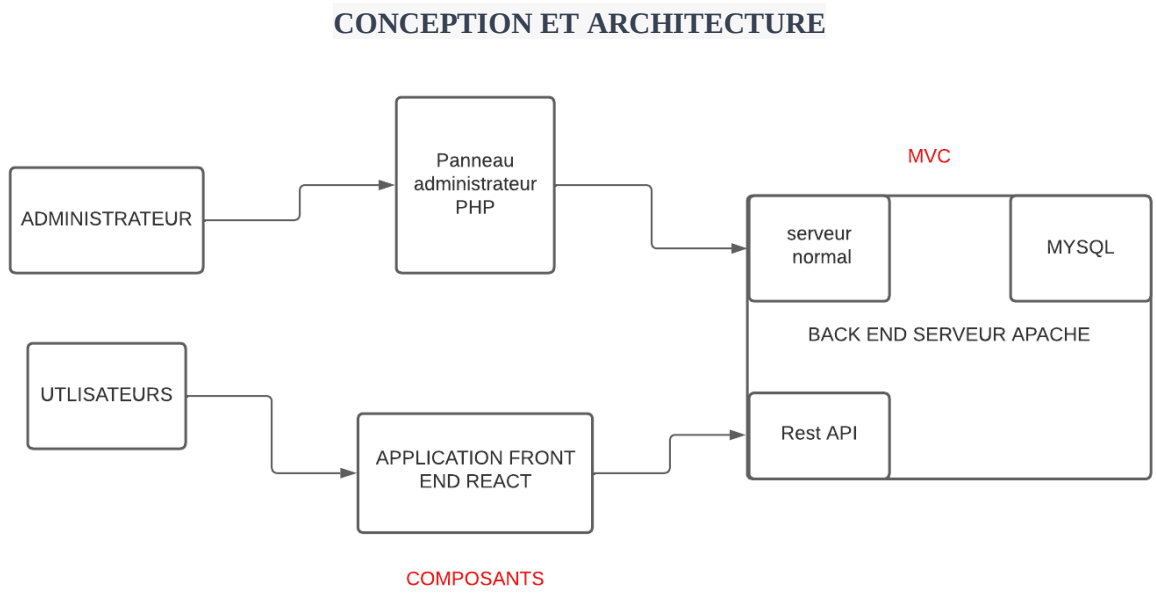
-**La classe Véhicule** représente un véhicule d'occasion, et j'ai également ajouté des méthodes pour accéder et modifier ses attributs.

-**La classe Message** est utilisée pour stocker les informations de contact, telles que les demandes de contact de clients.

-**La classe Horaire** est dédiée à la gestion des horaires d'ouverture du garage, et elle peut avoir des méthodes pour accéder et modifier ces attributs.

-**La classe Avis** est conçue pour recueillir les avis ou les témoignages des clients concernant le garage ou ses services. J'y ai également inclus des méthodes pour accéder et modifier les attributs des avis, ainsi que pour gérer les opérations liées à ces avis.

3.0 Conception et Architecture :



Le projet contiendra 2 parties distinctes :

-La **partie back end** qui sera réalisé en Php et placé sur un serveur Apache avec une base de données mysql / apache.

-Le **panneau administrateur** alimentateur et génèrera cette base de données en PHP. Toute la partie Serveur utilisera l'architecture modèle du contrôleur et sera programmée en **POO**.

-L'utilisateur utilisera les données de **l'API REST PHP** qui devra être programmé au niveau serveur et qui seront en format **JSON**.

Le serveur devra renvoyer uniquement les données en **Json** et ça sera **React** qui se chargera de les afficher.

4.0 Code référence véhicule :

CODE REFERENCE VEHICULE

Référence id unique par véhicule : exemple = **P-C-208-année**

Code référence véhicule par marque:

Peugeot = P

Renault = R

Citroen = C

Wolkswagen = w

Fiat = F

Code référence par famille :

Utilitaire = U

Berline = B

familiale = F

Citadine = C

Suv = S

Code référence véhicule par marque/model

Peugeot = 208/405/2008/3008/5008

Renault = clio/megane/zoe/twingo/kadjar

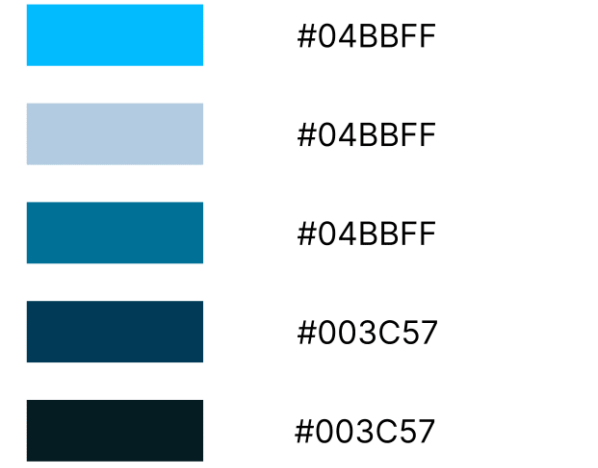
Citroen = C3/C4/C5/berlingo/spacetourer

Wolkswagen = golf/polo/passat/tiguan/touareg

Fiat = 500/panda/ducato/punto/tipo

5.0 UI :

PALETTE DE COULEURS



POLICES

- **LEAD** de Bootstrap

- LEAD n'est pas reconnu par Figma, je mettrais en remplacement sur la maquette la police 'Time New Roman'

-Unkempt

User story

GARAGE PARROT

Administrateur

2024

Table de matière

Préambule

Ce document décrit les fonctionnalités du rôle pour chaque administrateur de notre plateforme web et mobile.

CRUD (create, read, update, delete) (créer, lire, mettre à jour, supprimer) est un acronyme pour les façons dont on peut fonctionner sur des données stockées. C'est un moyen némotechnique pour les quatre fonctions de base du stockage persistant.

1.0 Création de compte

CREATION DE COMPTE	
Utilisateurs cibles	**ADMINISTRATEUR**
Description	En tant qu'administrateur, je souhaite pouvoir créer ou supprimer un espace admin pour mes employés.
Règles de gestion	-uniquement l'admin pourra le faire -un mot de passe aléatoire sera créer -à voir si l'employé pourra modifier son mdp -pouvoir donner un rôle à chaque admin (autorisation ou pas) -l'admin pourra ensuite modifier son mdp par sécurité en 2 étapes
Critères d'acceptation (conditions)	Etant donné que je suis le seul à pouvoir prendre les décisions à gérer le contenu du site, je déleste en parti mon travaille à certains de mes employés en leur créant un espace unique pour chaque personne

1.1 Se connecter

SE CONNECTER	
Utilisateurs cibles	ADMINISTRATEUR ET EMPLOYES
Description	En tant qu'administrateur, je souhaite pouvoir gérer le site en me connectant sur l'espace pro.
Règles de gestion	-avoir un compte admin -email et mdp obligatoire

	-modification de mot de passe affiché sous la connexion, un lien sera envoyé pour réinitialiser le mdp -création d'un journal des connexions journalière date + heure dans l'espace Admin uniquement -création d'un journal de toutes les tâches effectuées par les employés dans l'espace Admin uniquement
Critères d'acceptation **(conditions)**	Etant donné que je dois gérer le contenu du site, j'ai le moyen de le faire en me connectant sur la plateforme pro

1.2 Déconnexion

DECONNEXION	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir me déconnecter à tout moment par sécurité.
Règles de gestion	-avoir un compte admin -se connecter sera remplacé par déconnexion lorsqu'il sera connecté -déconnexion automatique après 5mn d'inactivité
Critères d'acceptation (conditions)	Etant donné que j'ai terminé ce que je voulais faire ds l'espace pro, j'ai le moyen de me déconnecter à tout moment par sécurité.

1.3 Règles de gestion du Mot de passe

REGLES DE GESTION DU MOT DE PASSE	
Utilisateurs cibles	**ADMINISTRATEUR**
Description	En tant qu'administrateur, je souhaite pouvoir modifier mon mdp si j'en ai besoins
Règles de gestion	-être admin -mdp associé au mail -réinitialisation du mdp et valider
Critères d'acceptation **(Conditions)**	Etant donné que j'ai oublié mon mdp, je clique sur le lien mdp oublié qui est sur la page connexion, j'inscris mon mail pour pouvoir recevoir une réinitialisation de mdp au compte associé au mail

4.0 Formulaire de contact (navbar + prestations)

FORMULAIRE DE CONTACT (navbar + prestations)	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir répondre aux questions des utilisateurs
Règles de gestion	-avoir un compte admin -pouvoir répondre aux utilisateurs -pouvoir transférer le message au service concerné -avoir un historique de la conversation -notification de modification, supprimer ou envoyé
Critères d'acceptation (conditions)	Etant donné que je dois fournir un service de communication pour la satisfaction du client, je dois pouvoir lui répondre ds les plus brefs délais.

4.1 Formulaire de contact (fiche voiture)

FORMULAIRE DE CONTACT (fiche voiture)	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir ajouter, modifier ou supprimer certains contenus comme des images, textes
Règles de gestion	-avoir un compte admin - L'ID ou référence du véhicule accompagné du message de l'utilisateur devront être afficher pour une meilleure expérience -notification de modification, supprimer ou envoyé
Critères d'acceptation (conditions)	Etant donné que je dois fournir un service de communication pour la satisfaction du client, je dois pouvoir lui répondre ds les plus brefs délais.

5.0 Dashboard - Page d'accueil et prestations

DASHBOARD - PAGE D'ACCUEIL ET PRESTATIONS	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir ajouter, modifier ou supprimer certains contenus comme images, textes…
Règles de gestion	-avoir un compte admin - **Section 1**-> photo modifiable - **Section 2**-> photo modifiable - **Section 3**-> photos + texte modifiable - **Section 4**-> photos modifiables + texte modifiable - **Section 5**-> photo modifiable - **Section 6**-> + texte modifiable -notification de modification, supprimer ou création - historique des modifications pour chaque employé.
Critères d'acceptation (conditions)	Etant donné que je dois gérer le contenu de mon site, j'ai le moyen de le faire en tant qu'administrateur dans le Dashboard dédié.

5.1 Dashboard - Page Fiche Voiture

DASHBOARD - PAGE FICHE VOITURE	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir ajouter, modifier ou supprimer certains contenus comme des images, textes.
Règles de gestion	-avoir un compte admin - Section 4-> photos modifiables + texte modifiable en ajoutant les caractéristiques de chaque véhicule. -notification de modification, supprimer ou création - historique des modifications pour chaque employé.

Critères d'acceptation (conditions)	Etant donné que je dois gérer le contenu de mon site, j'ai le moyen de le faire en tant qu'administrateur dans le Dashboard dédié

5.2 Dashboard - Avis Clients

DASHBOARD – AVIS CLIENTS	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir gérer les avis clients et pouvoir aussi les publier en ligne. J'aurais le choix ou pas de les publier.
Règles de gestion	-avoir un compte admin -ajout, supprimer avis client en ligne -possibilité d'ajouter avis client écrit sur papier -répondre aux avis -les infos persos ne devront pas être publié en ligne, uniquement le prénom
Critères d'acceptation (conditions)	Etant donné que je dois attirer un maximum de clientèle, je peux exposer sur le site quelques avis clients.

5.3 Dashboard - Horaires

DASHBOARD - HORAIRES	
Utilisateurs cibles	**ADMINISTRATEUR ET EMPLOYES**
Description	En tant qu'administrateur, je souhaite pouvoir modifier les horaires d'ouverture de l'entreprise.
Règles de gestion	-avoir un compte admin -affichage dans footer -notification de modification, supprimer ou création -ajout, modifié, supprimer horaires

	-heure, fermé, vacances -coupure entre 12h et 14h
Critères d'acceptation (**conditions**)	Etant donné que je dois informer la clientèle des horaires d'ouverture du garage, j'ai le moyen de les informer par le site.

User story

GARAGE PARROT

Utilisateurs

2024

Table des matières

Préambule

Ce document décrit les fonctionnalités par rôle pour chaque utilisateur ou système de notre plateforme web et mobile.

Voici les rôles :

Les utilisateurs *bénéficient de tous les services que propose le site sans devoir être authentifiés.*

EXEMPLE A VIDE :

TITRE	
Utilisateurs cibles	**Utilisateurs et/ou Anonymes**
Description	En tant que….. , Je souhaite…., afin de(facultatif)….
Règles de gestion	-fonctionnalités
Critères d'acceptation (conditions)	Etant donné que….(situation de départ),…, que …, Quand….(action) et que…. Alors….(resultat)

1.0 Header

HEADER	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'Utilisateur, j'ai accès en premier plan à des offres ou promotions sous forme de bannière.
Règles de gestion	
Critères d'acceptation (conditions)	Etant donné que je suis sur le site pour un sujet bien précis, je peux éventuellement profiter d'offres qui pourrait être décisif dans mes futurs choix.

1.1 Header - Navbar

HEADER - NAVBAR	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'Utilisateur, je veux pouvoir accéder facilement aux différentes sections du site depuis la navbar, afin de naviguer rapidement vers les pages qui m'intéressent.
Règles de gestion	-cliquer sur le service désiré -menu Accueil(logo)/Pneumatique/ Mécanique/Contact……
Critères d'acceptation (conditions)	Etant donné que j'aimerais naviguer facilement sur le site, j'ai le moyen de le faire en cliquant sur le menu qui m'est proposé

2.0 Formulaire De Contact

FORMULAIRE DE CONTACT	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je veux pouvoir contacter le service clientèle via un formulaire de contact pour poser des questions ou signaler un problème.
Règles de gestion	-remplir tous les champs du formulaire -email : invalide si pas @, .fr .com -numéro de téléphone : minimum 10 chiffres -nom, prénom : minimum 3 lettres -prouver que je suis bien un humain en cochant la checkbox captcha
Critères d'acceptation (conditions)	Etant donné que je souhaite un renseignement sur un produit ou service, prendre rdv ou demander un devis, je le fais avec le formulaire de contact qui m'est proposé par le site.

3.0 Footer (pied de page)

FOOTER	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, j'aimerais suivre via les réseaux sociaux l'entreprise et connaître les horaires d'ouverture. Je suis aussi avisé des règles légales de l'entreprise.
Règles de gestion	-Doit avoir une page les mentions légales -Doit avoir une page de confidentialité -Doit avoir condition générale de vente -Doit avoir réseaux sociaux -Doit avoir les horaires d'ouverture -Doit avoir les coordonnées du site pour avoir meilleur référencement
Critères d'acceptation (conditions)	Etant donné que je souhaite me renseigner un peu plus sur l'entreprise et connaître mes droits en tant qu'utilisateur, je peux y accéder directement via le footer

4.0 Page d'accueil

PAGE D'ACCUEIL	
Utilisateurs cibles	Utilisateurs
Description	En tant qu'utilisateur, je veux voir une interface claire et attrayante sur la page d'accueil pour faciliter la navigation et susciter mon intérêt dès le premier regard.
Règles de gestion	
Critères d'acceptation (conditions)	-Etant donnée que je cherche à consommer, la page d'accueil influencera mes choix et d'y rester ou pas.

4.1 Page d'accueil – Laisser un avis

PAGE D'ACCUEIL- LAISSER UN AVIS	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je souhaite pouvoir laisser des commentaires et des évaluations sur la qualité des services de l'entreprise pour aider les autres utilisateurs dans leur décision d'achat.
Règles de gestion	-remplir tous les champs du formulaire -email : invalide si pas @, .fr .com -numéro de téléphone : minimum 10 chiffres -nom, prénom : minimum 3 lettres -prouver que je suis bien un humain en cochant la checkbox captcha
Critères d'acceptation (conditions)	Etant donné que j'ai été client du site, je laisse un avis pour aider les futurs clients dans leurs choix

4.3 Page d'accueil – Lire un avis

PAGE D'ACCUEIL – LIRE UN AVIS	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'Utilisateur, je souhaite pouvoir lire les avis d'autres clients pour savoir si l'entreprise est sérieuse ou pas.
Règles de gestion	-Défilement des avis par flèches gauche ou droite.
Critères d'acceptation (conditions)	Etant donné que je souhaite acheter un service ou produit, je me renseigne déjà sur les avis clients pour savoir si l'entreprise est honnête ou pas.

4.4 Page d'accueil - Cookies

PAGE D'ACCUEIL - COOKIES	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, j'ai le choix d'accepter ou pas que l'on utilise mes informations pour une meilleure navigation dans le futur.
Règles de gestion	-Affichage : Accepter ou refuser les cookies lors de la connexion sur le site
Critères d'acceptation (conditions)	Etant donné que je souhaite me connecter au site, j'ai l'obligation d'accepter ou pas les cookies pour pouvoir faire disparaître le popup et pouvoir visiter le site

5.0 Page Prestations

PAGE – PRESTATIONS	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je souhaite pouvoir parcourir facilement les différentes prestations offertes par l'entreprises afin de trouver celles qui correspondent le mieux à mes besoins.
Règles de gestion	-Aucunes
Critères d'acceptation (conditions)	Etant donné que j'ai un but précis dans mes recherches, je vais directement et rapidement au service voulu.

5.1 Page Prestations - Contact

PAGE PRESTATIONS - CONTACT	
Utilisateurs cibles	**Utilisateurs**

Description	En tant qu'utilisateur, je veux pouvoir contacter le service clientèle via un formulaire de contact pour avoir des renseignements sur une prestation et obtenir un devis
Règles de gestion	-remplir tous les champs du formulaire -email : invalide si pas @, .fr .com -numéro de téléphone : minimum 10 chiffres -nom, prénom : minimum 3 lettres -prouver que je suis bien un humain en cochant la checkbox captcha
Critères d'acceptation (conditions)	Etant donné que je souhaite un renseignement ou un devis sur un produit ou service, prendre rdv, je le fais via le formulaire de contact qui m'est proposé par le site.

6.0 Page Voitures d'Occasion

PAGE VOITURES D'OCCASION	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je souhaite me renseigner sur les véhicules que propose le site pour un éventuel achat.
Règles de gestion	-Aucunes
Critères d'acceptation (conditions)	Etant donné que je souhaite acheter une voiture, j'ai le choix de faire ma sélection par rapport au catalogue qui m'est proposé.

6.1 Page Voitures d'Occasion - Filtres

PAGE VOITURES D'OCCASION - FILTRES	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur et ayant un but précis, je souhaite pouvoir trouver des véhicules plus facilement via le système de filtres.
Règles de gestion	-choix par famille -choix par kilométrage, marque, modèle, prix et année
Critères d'acceptation (conditions)	Etant donné que je souhaite acheter un véhicule, j'ai le moyen de le faire grâce aux filtres pour une recherche plus rapide.

7.0 Page Fiche Voiture

PAGE FICHE VOITURE	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je peux voir toutes les caractéristiques de chaque véhicules
Règles de gestion	-Aucunes

Critères d'acceptation (conditions)	Etant donné que je souhaite acheter un véhicule et que je sais quel genre de véhicule je veux, j'ai le moyen de le faire grâce à la fiche technique du véhicule.

7.1 Page Fiche Voiture - Contact

PAGE FICHE VOITURE - CONTACT	
Utilisateurs cibles	**Utilisateurs**
Description	En tant qu'utilisateur, je peux demander des renseignements complémentaires sur le véhicule, le réserver ou obtenir un rdv pour le voir.
Règles de gestion	-remplir tous les champs du formulaire -email : invalide si pas @, .fr .com -numéro de téléphone : minimum 10 chiffres -nom, prénom : minimum 3 lettres -prouver que je suis bien un humain en cochant la checkbox captcha -affichage automatique de l'Id du véhicule ds le formulaire pour que l'admin puisse trouver le véhicule plus rapidement.
Critères d'acceptation (conditions)	Etant donné que je souhaite me renseigner de ce que propose le site et savoir si c'est adapté à mes exigences, je vais me renseigner sur la page en question.

FRONT END

PARTIE 1

Ce rapport détaille les étapes cruciales et les réalisations majeures dans le développement Front-End, mettant spécifiquement l'accent sur l'établissement de l'environnement de travail, l'utilisation de React, la structuration efficace des dossiers, et l'intégration stratégique de Bootstrap via les CDN.

Table de matière

1.0 Mise en Place de l'Environnement de Travail

L'une des premières tâches du projet a été la mise en place de l'environnement de travail.

Mon choix s'est porté sur la bibliothèque JavaScript **React** pour le développement des interfaces utilisateur.

J’ai configuré les outils et dépendances nécessaires pour pouvoir démarrer mon projet.

1.1 React Router

React Router a été installé à l'aide de la commande **`npm install react-router-dom`.**

Une fois installé, J’ai importé dans le fichier `**App.js**` pour être utilisé dans la gestion des routes de l'application.

Le composant `Site` a été configuré pour contenir toutes les routes de l'application.

1.2 Gestion des Dossiers et Problèmes d'Importation

Une des premières difficultés rencontrées a été liée à la gestion des dossiers et à l'importation de composants.

J’ai observé que React ne reconnaissait pas automatiquement les majuscules dans les noms de dossiers, obligeant à les saisir manuellement.

Ce problème a également été rencontré précédemment sur un autre de mes projets, il doit s’agir d’un problème de React.

1.3 Intégration de Bootstrap via les CDN

Pour améliorer l'aspect visuel de l'application, Bootstrap a été intégré en utilisant des CDN (Content Delivery Network), par la suite, je l’ai installé via le terminal par rapport à React.

Cette approche a permis d'inclure rapidement Bootstrap dans le projet sans avoir à télécharger et à configurer les fichiers localement.

1.4 Création de Composants SF et SL

J’ai créé 2 types de composants à l’avance pour pouvoir gagner un peu plus de temps :ce sont les composants **Stateless** et les composants **Stateful**.

Les composants SF sont des composants React simples qui ne gèrent pas d'état interne.

Les composants SL, en revanche, gèrent un état interne et sont utilisés pour des éléments interactifs de l'interface utilisateur.

1.5 Utilisation d'Axios pour les Requêtes

J’ai installé Axios, une bibliothèque JavaScript populaire pour effectuer des requêtes HTTP, à l'aide de la commande **`npm install axios`.**

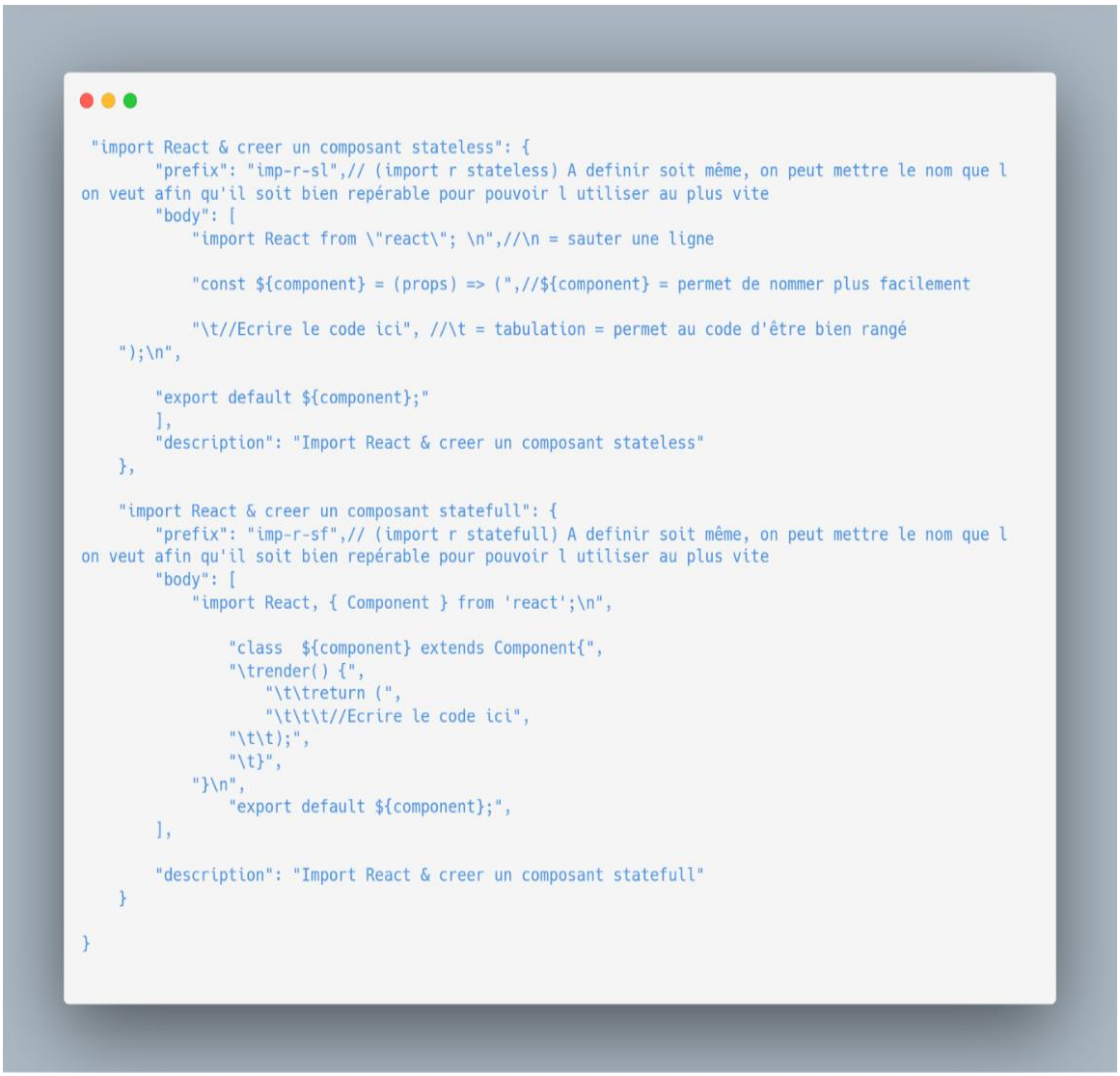
Axios sera utilisé pour effectuer des requêtes vers le backend ou d'autres services par la suite.

1.6 Optimisation du Flux de Travail avec des Snippets

Pour faciliter le développement, j’ai créé dans Visual Studio Code des extraits de code **(snippets).**

Ces extraits permettent de générer automatiquement la structure de base des composants **SF et SL**, accélérant ainsi le processus de création de composants.

Pour utiliser ces extraits, Je suis allé dans le menu "Fichier" (File), puis "Préférences" (Préférences), et enfin "Extraits d'utilisateur" (User Snippets), puis de sélectionner le langage (dans ce cas-là, JavaScript).



```
 "import React & creer un composant stateless": {
        "prefix": "imp-r-sl",// (import r stateless) A definir soit même, on peut mettre le nom que l
on veut afin qu'il soit bien repérable pour pouvoir l utiliser au plus vite
        "body": [
            "import React from \"react\"; \n",//\n = sauter une ligne

            "const ${component} = (props) => (",//${component} = permet de nommer plus facilement

            "\t//Ecrire le code ici", //\t = tabulation = permet au code d'être bien rangé
    ");\n",

        "export default ${component};"
        ],
        "description": "Import React & creer un composant stateless"
    },

    "import React & creer un composant statefull": {
        "prefix": "imp-r-sf",// (import r statefull) A definir soit même, on peut mettre le nom que l
on veut afin qu'il soit bien repérable pour pouvoir l utiliser au plus vite
        "body": [
            "import React, { Component } from 'react';\n",

                "class  ${component} extends Component{",
                "\trender() {",
                    "\t\treturn (",
                    "\t\t\t//Ecrire le code ici",
                "\t\t);",
                "\t}",
            "}\n",
                "export default ${component};",
        ],

        "description": "Import React & creer un composant statefull"
    }

}
```

Je détaille ici les activités effectuées au cours de la phase de développement côté Front. Dans cette section, je me pencherai sur les éléments suivants :

La conception du composant ‘Navbar’ en utilisant Bootstrap, la création d'une page d'erreur 404, l'intégration d'Axios pour gérer les requêtes, la mise en place du composant de cartes pour afficher les véhicules, l'implémentation d'un système de pagination pour les cartes, ainsi que le développement d'un système de recherche de véhicules basé sur des filtres….

2.0 Création des composants

2.1 Création du Composant ‘Navbar’ avec Bootstrap

J’ai créé un composant ‘**Stateless’** nommé Navbar.js pour pouvoir afficher la barre de navigation de l'application.

-Mode desktop

-Mode Mobile (Hamburger)

J’ai inséré dans le composant des images et du code **JSX**, avec des classes **`className`** pour pouvoir mettre du style en majorité Bootstrap.

J’ai présenté les services de réparation sous forme de menus déroulants (**dropdowns**) dans la barre de navigation.

Le tout en responsive bien sûr, un menu de type ‘hamburger’ sera affiché en mode mobile.

2.2 Création du composant de la Page d'Erreur 404

J’ai ajouté une route de type ‘**Error 404’** pour pouvoir gérer les cas où une page n'existe pas.

Cette page d'erreur fournira une réponse aux utilisateurs en cas d'accès à des pages inexistantes.

```jsx
const error = (props) => (
  <div style={{ backgroundColor: "#FF0000", color: "#FFFFFF", padding: "20px"
}}> <TitreH1>Erreur{props.type}</TitreH1>
    <div>
      {props.children}
    </div>
  </div>
)

export default error
```

2.3 Création du Composant ‘Card’ pour afficher les Véhicules

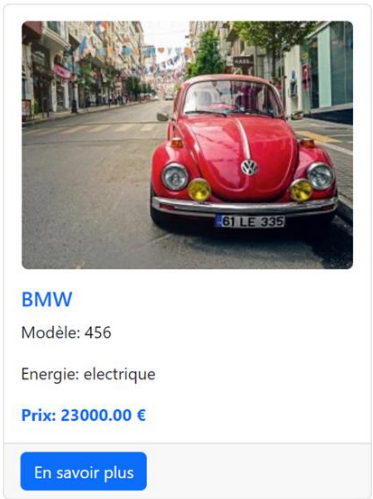
J’ai voulu présenter les véhicules aux utilisateurs sous forme de cartes Bootstrap, mettant en avant quelques informations essentielles concernant chaque véhicule.

Composant ‘Card’

Pour réaliser cela, j'ai connecté ce composant directement à la base de données, extrayant les données de la table 'véhicule' à l'aide d'Axios.

Cela me permettra de récupérer toutes les informations relatives à chaque véhicule.

Le composant **'Card'** a été conçu pour être réutilisable, affichant de manière individualisée les détails d'un véhicule à la fois, sous forme d'une carte distincte.



```jsx
  const Card = (props) => {
  const image = `http://localhost/garageback/public/images/${props.image}`;
  return (
<>
    <div className="card">
      <div className="card-body">
        <a
          href={props.image}
          target="_blank"
          rel="noopener noreferrer"
        >
          <img src={image} alt={props.marque} className="img-fluid rounded mx-auto d-block mb-
3" />    </a>

        <h5 className="card-title text-primary">{props.marque.toUpperCase()}</h5>
        <p className="card-text">Modèle: {props.modele} </p>
        <p className="card-text">Energie: {props.energie} </p>
        <p className="card-text fw-bold text-primary">Prix: {props.prix} €</p>
      </div>
      <div className="card-footer">
        <Link
          to={`/vehiculefiche/${props.id}`}
          className="btn btn-primary"
        >
          En savoir plus
        </Link>
      </div>
    </div>
    </>
  );
};

export default Card;
```

Je vais ajouter des paramètres aux propriétés du composant **'Card'** pour représenter les informations du véhicule, telles que l'image, la marque, le modèle, etc.

Pour obtenir l'image, je vais construire le chemin en utilisant l'URL de base, indiquant où sont stockées les images.

Ensuite, je vais transmettre cette URL via la propriété **`props.image`**, ce qui me permettra d'éviter de réécrire l'URL chaque fois que j'aurai besoin de récupérer une image .

Composant ‘vehiculeCard’

Pour ce composant, je vais importer plusieurs dépendances, notamment **`useState`**, qui me permettra de gérer l'état local pour stocker la liste des véhicules, le numéro de la page actuelle **`currentPage`**, et le nombre de cartes affichées par page que je vais définir ultérieurement **`cardsPerPage`**.

```
 const VehiculesCard = () => {
//State pour stocker la liste des véhicules
  const [vehicules, setVehicules] = useState([]);

//State pour gérer la pagination
  const [currentPage, setCurrentPage] = useState(1);
  const cardsPerPage = 6;

//Utilisation de useEffect qui va effectuer des opérations asynchrones lors du rendu d'un composant.
  useEffect(() => {

//Utilisation de la bibliothèque Axios pour effectuer une requête HTTP GET vers l'URL spécifiée
    axios
      .get("http://localhost/garageback/front/voiturefiche/all")

//La méthode .then est appelée sur la promesse renvoyée par la requête HTTP. Elle prend une fonction
callback avec le paramètre response qui représente la réponse de la requête.
      .then((response) => {

//Extraction des données Json de la réponse HTTP.
        const jsonData = response.data;

//Tri des données par date de création
        jsonData.sort((a, b) => new Date(a.created_at) - new Date(b.created_at));

        setVehicules(jsonData);
      })
      .catch((error) => {
        console.error("Erreur lors de la récupération des véhicules :", error);
      });
```

Mon objectif est de permettre au composant d'effectuer une requête à partir de **l'API** 'véhicule' que j'ai créée du côté serveur, afin d'obtenir la liste de tous les véhicules à afficher.

Pour cela, je vais utiliser **'axios'** dans la fonction **'useEffect'**. Mon intention est de faire en sorte que les cartes s'affichent sur la page d'accueil en fonction de leur date de création la plus récente.

Cela permettra de renouveler régulièrement les cartes et de proposer aux utilisateurs les derniers véhicules en stock.

Le composant va ensuite mapper les véhicules actuellement affichés dans une liste de composants **'Card'** créés précédemment, en transmettant les informations du véhicule en tant que propriétés à chaque composant 'Card'.

```
<div className="row">
        {currentCards.map((vehicule) => (
          <div
            key={vehicule.idVehicule}
            className="col-lg-4 col-md-4 col-sm-6 col-6 mt-3"
          >
            <Card
              image={vehicule.imageVoiture}
              marque={vehicule.marque}
              nom={vehicule.nom}
              modele={vehicule.modele}
              energie={vehicule.energie}
              prix={vehicule.prix}
              id={vehicule.idVehicule}
            />
          </div>
```

2.4 Mise en Place de la Pagination des cartes

Je vais mettre en œuvre un système de pagination pour diviser les véhicules en groupes, limitant ainsi le nombre de véhicules affichés par page grâce à la variable **'cardsPerPage'**.

Ce mécanisme permettra aux utilisateurs de naviguer d'une page à l'autre.

Le nombre total de pages sera calculé en fonction du nombre total de véhicules et du nombre de cartes par page, une valeur que je vais définir ultérieurement.

```
 const indexOfLastCard = currentPage * cardsPerPage;
  const indexOfFirstCard = indexOfLastCard - cardsPerPage;
  const currentCards = vehicules.slice(indexOfFirstCard, indexOfLastCard);

  const paginate = (pageNumber) => {
    setCurrentPage(pageNumber);
  };
console.log(currentCards);
        console.error("Erreur lors de la récupération des véhicules :",
error)});
```

```
<Pagination>
          {Array.from(
            { length: Math.ceil(vehicules.length / cardsPerPage) },
            (_, index) => (
              <Pagination.Item
                key={index}
                active={index + 1 === currentPage}
                onClick={() => paginate(index + 1)}
              >
                {index + 1}
              </Pagination.Item>
```

Le nombre maximal de cartes affichées sera de 9 véhicules, ce qui correspond à 3 cartes par ligne en mode desktop et 2 en mode mobile, tant sur la page d'accueil que sur la page de voitures d'occasion résultant d'une recherche par filtres ou d'un classement par ordre de date de création.

Cette limitation permettra d'améliorer significativement les performances de l'application et éviter que tous les véhicules de la base de données s’affichent en même temps.

Affichage des informations complète du véhicule

Lorsque l'utilisateur clique sur "**plus d'infos**" sur la carte, une nouvelle page s'affichera, présentant toutes les informations du véhicule de manière plus détaillée, toujours sous forme de carte, mais de manière plus large.

Si l'utilisateur souhaite obtenir davantage d'informations sur un produit, je ferai en sorte à le rediriger vers la page de contact.

Dans une étape future, que je n'ai pas encore eu l'occasion de mettre en place, je prévois d'automatiquement inclure l'ID du véhicule dans le formulaire de contact.

Cela permettra à l'administrateur de retrouver plus facilement le véhicule en se référant à son identifiant, ce qui lui entraînera un gain de temps en évitant de rechercher manuellement.

```
<Pagination>
          {[...Array(pagesCount)].map((_, index) => (
            <Pagination.Item
              key={index}
              active={index === pageNumber}
              onClick={() => handlePageClick(index)}
            >
              {index + 1}
            </Pagination.Item>
          ))}
        </Pagination>
```

2 .5 Création des composants UI

-Création de tous les composant boutons, cards ….

3.0 Page d'Accueil

-La page d'accueil a été mise en place avec l'ajout d'un composant `**h1**` pour afficher le titre.

-J'ai ajouté le composant **Navbar**.

- Pour améliorer la présentation, j'ai installé la bibliothèque **Font Awesome** à l'aide de la commande `**npm install --save font-awesome**`.

Composant caroussel

J'ai intégré le composant carrousel pour présenter les services que le garage propose.

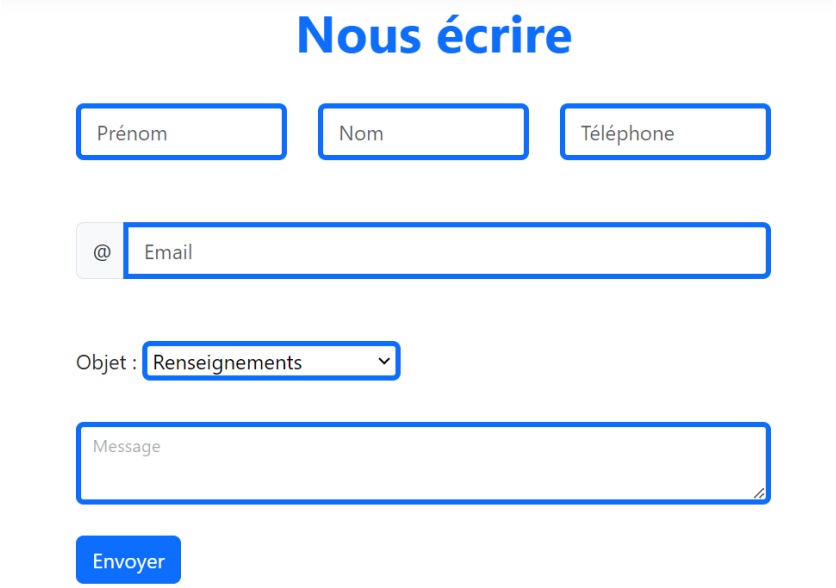
Lorsque l'utilisateur cliquera sur le bouton du carrousel, il sera redirigé vers une autre route que j'aurai créée au préalable.

Cette nouvelle route affichera une présentation des prestations du garage, également sous forme de cartes.

J’ai ajouté le composant ‘**cards’** pour pouvoir présenter à l’utilisateur les derniers véhicules en stock arrivés .

4.0 Page de Contact

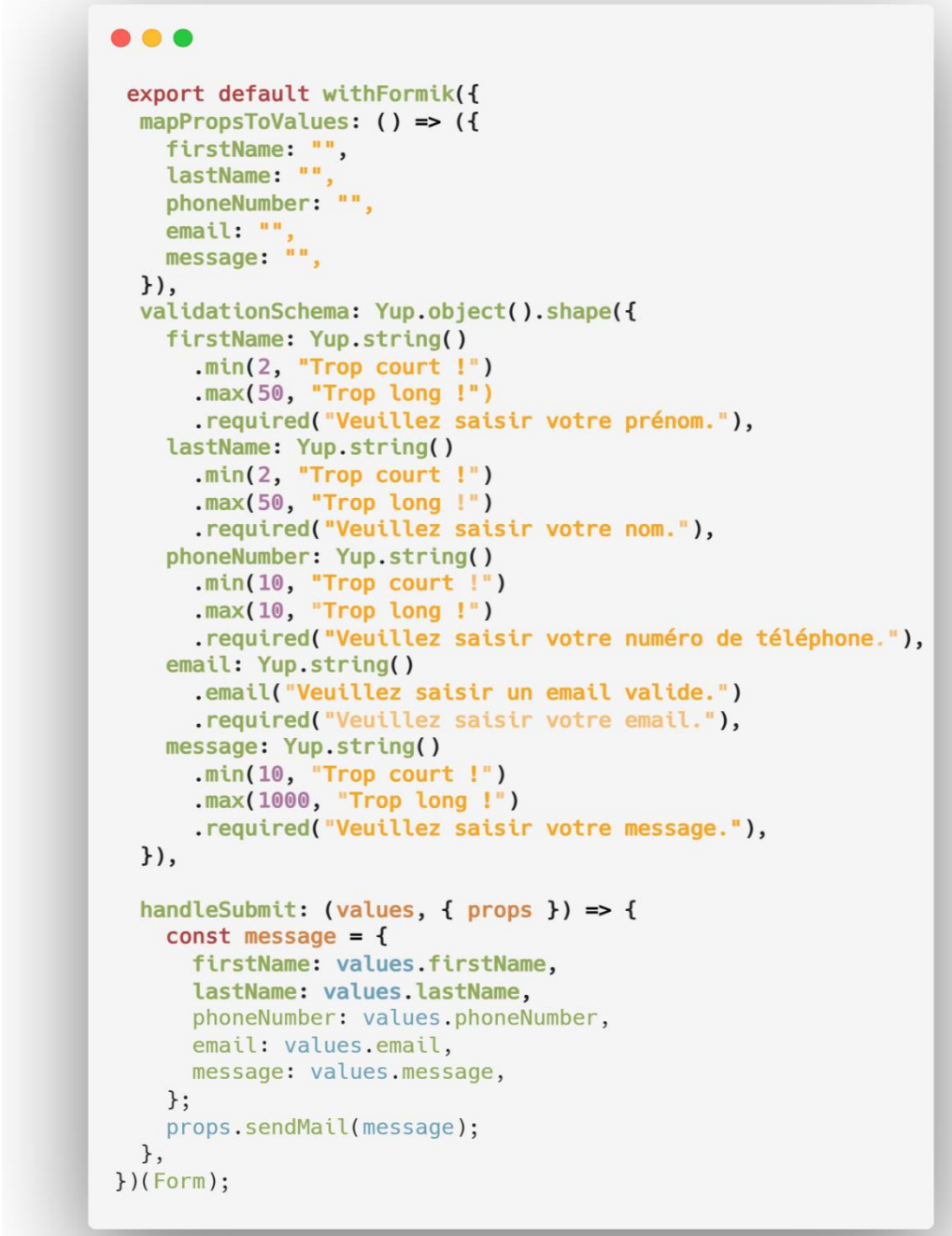
Je viens d’ajouter une route vers le formulaire de contact qui a été créée dans le fichier `Site.js`, avec son composant associé.



J’ai développé la ‘**view’** du formulaire de la page de contact sous ‘**Bootstrap’** dans un contenaire.

J’ai utilisé la bibliothèque ‘**Formik’** pour faciliter la gestion des formulaires et la bibliothèque ‘**Yup’** pour imposer des restrictions lors du remplissage des champs du formulaire.

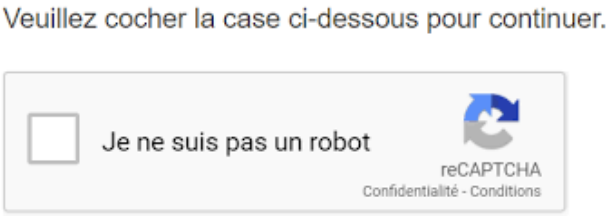
J’ai ajouté certaines restrictions pour les champs de formulaire pour éviter d’éventuelles erreurs que pourrait faire l’utilisateur.



```
export default withFormik({
 mapPropsToValues: () => ({
   firstName: "",
   lastName: "",
   phoneNumber: "",
   email: "",
   message: "",
 }),
 validationSchema: Yup.object().shape({
   firstName: Yup.string()
     .min(2, "Trop court !")
     .max(50, "Trop long !")
     .required("Veuillez saisir votre prénom."),
   lastName: Yup.string()
     .min(2, "Trop court !")
     .max(50, "Trop long !")
     .required("Veuillez saisir votre nom."),
   phoneNumber: Yup.string()
     .min(10, "Trop court !")
     .max(10, "Trop long !")
     .required("Veuillez saisir votre numéro de téléphone."),
   email: Yup.string()
     .email("Veuillez saisir un email valide.")
     .required("Veuillez saisir votre email."),
   message: Yup.string()
     .min(10, "Trop court !")
     .max(1000, "Trop long !")
     .required("Veuillez saisir votre message."),
 }),

 handleSubmit: (values, { props }) => {
   const message = {
     firstName: values.firstName,
     lastName: values.lastName,
     phoneNumber: values.phoneNumber,
     email: values.email,
     message: values.message,
   };
   props.sendMail(message);
 },
})(Form);
```

Comme le formulaire n'est pas en ligne pour l'instant, je ne peux pas savoir s'il est fonctionnel ou pas mais dans d'autres projets que je vais présenter par la suite, ils le seront.

4.1 Ajout de Google reCAPTCHA



Afin de renforcer la sécurité du formulaire de contact et prévenir les spams, j'ai intégré Google '**reCAPTCHA'**.

L'installation du reCAPTCHA a été effectuée en utilisant la commande `**npm install react-google-recaptcha**`, puis je l'ai importé par la suite dans le projet.

J'ai généré un identifiant unique de sécurité grâce à Google reCAPTCHA. Cet identifiant doit être stocké dans un fichier caché, de manière à ce qu'il ne soit pas accessible en ligne, ce qui renforcera la sécurité de l'application.

```
import ReCAPTCHA from "react-google-recaptcha";

const Recaptcha = ({ onChange }) => {
  return (
    <ReCAPTCHA

 sitekey={onChange}NEXrK24tmIgsSaLO_ZdUaJzXJ"
    />
  );
};

export default Recaptcha;
```

5.0 Page Voitures d'Occasions

Mise en Place des Composants de Recherche et de Filtrage

Au cours de cette phase de développement, j'ai conçu des éléments d'interface utilisateur (composants), notamment des '**sliders'**, des sélecteurs '**dropdowns**', et des cases à cocher '**checkboxs**', visant à faciliter la recherche et le filtrage des données.

Pour améliorer l'expérience utilisateur, je vais créer des ‘**sliders’ => prix, km et année** en offrant la possibilité à l'utilisateur de définir à la fois une valeur **minimale et maximale** pour chaque paramètre.

Cela permettra aux utilisateurs de mieux cibler leurs critères de recherche.

J'ai structuré le code en créant des composants distincts pour chaque élément de filtrage, qu'il s'agisse de **cases à cocher, de menus déroulants, ou de sliders**.

Ces composants individuels communiquent avec un composant parent centralisé nommé "**VehiculeFilters**" pour gérer l'état des filtres de manière cohérente.

Ne pouvant directement communiquer entre eux, je vais créer pour les composants parents et fils une fonction ‘**handlechange’** qui prendra en paramètre un évènement lorsque l’utilisateur cliquera sur le bouton rechercher.

Elle sera définie pour mettre à jour l’état des filtres lorsque l’utilisateur effectuera une sélection dans l’un des composant filtre.

Je la passerais en tant que ‘**props’** aux composants fils pour qu’ils puissent **mettre à jour** l’état parent.

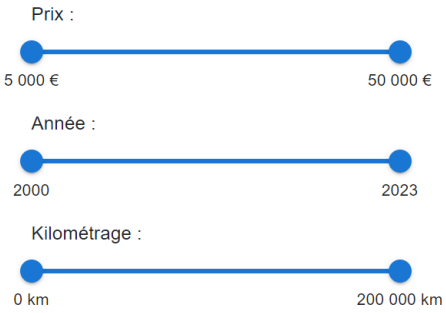
Chaque composant a été créé en utilisant la bibliothèque ‘**Material-UI (https://mui.com/)’.**

J'ai installé les dépendances nécessaires pour garantir leur bon fonctionnement ce qui m’a beaucoup aidé car au départ, j’ai voulu créer les composants de filtres par moi-même, ça fonctionnait à peu près mais visuellement, ce n’était pas terrible.

5.1 Composant BasicRange

Je vais faire en sorte de créer la fonction qui attendra en paramètre un objet et qui utilisera le ‘**destructuring’** en extrayant les propriétés de l’objet dans des variables locales.

L’objet passé en paramètre ne contient pas de propriétés spécifiques, alors je mettrais une valeur par défaut.

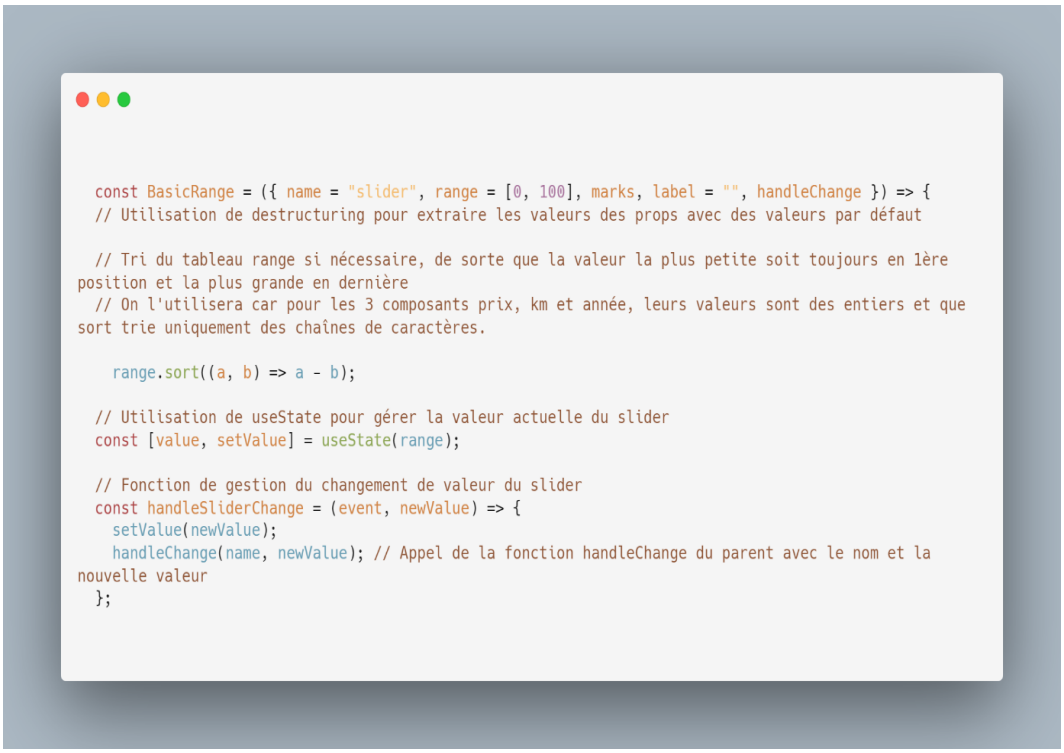


Après avoir effectué des recherches, j'ai conclu que le ‘**destructuring’** est une technique efficace pour extraire des valeurs à partir d'objets ou de tableaux, ce qui a donc motivé mon choix.

Pour ce qui est du tri d'un tableau, j'ai opté pour l'utilisation de la fonction `**sort()**` sans recourir au ‘**destructuring’**.

Cette fonction trie les éléments du tableau en les comparant les uns aux autres, ce qui me permettra de réorganiser les éléments du tableau de manière à les placer dans un ordre spécifique.

```
range.sort((a, b) => a - b);
```



```
  const BasicRange = ({ name = "slider", range = [0, 100], marks, label = "", handleChange }) => {
  // Utilisation de destructuring pour extraire les valeurs des props avec des valeurs par défaut

  // Tri du tableau range si nécessaire, de sorte que la valeur la plus petite soit toujours en 1ère
position et la plus grande en dernière
  // On l'utilisera car pour les 3 composants prix, km et année, leurs valeurs sont des entiers et que
sort trie uniquement des chaînes de caractères.

    range.sort((a, b) => a - b);

  // Utilisation de useState pour gérer la valeur actuelle du slider
  const [value, setValue] = useState(range);

  // Fonction de gestion du changement de valeur du slider
  const handleSliderChange = (event, newValue) => {
    setValue(newValue);
    handleChange(name, newValue); // Appel de la fonction handleChange du parent avec le nom et la
nouvelle valeur
  };
```

Le composant BasicRange comporte les éléments suivants :

- **handleChange** : Cette fonction sera appelée à chaque modification de la valeur du slider et recevra deux paramètres : le nom du paramètre (comme prix, kilométrage, année) et la nouvelle valeur sélectionnée.

- **label** : Un libellé qui s'affiche au-dessus du slider pour aider l'utilisateur à comprendre les valeurs sélectionnées.

- **name** : Le nom du slider.

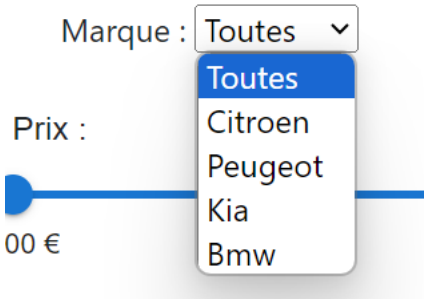
- **marks** : Un tableau d'objets spécifiant les marques (valeurs) sur le slider, chaque objet ayant une valeur et une étiquette (label). Par exemple, pour le prix : "5 000 - 50 000".

- **range** : La plage de valeurs possibles du slider. J'ai utilisé une plage de base de 0 à 100, qui convient à de nombreuses situations.

J'ai veillé à ce que la plus petite valeur soit toujours en première position en utilisant une fonction de tri pour maintenir l'ordre correct.

```
 return (
    <>
      <Typography id="input-slider" gutterBottom>
        {label}
      </Typography>

      <Slider
        name={name}
        min={range[0]}
        max={range[1]}
        onChange={handleSliderChange}
        size="medium"
        valueLabelDisplay="auto" // Afficher la valeur actuelle
        marks={marks || [{ value: range[0], label: range[0].toString() }, { value: range[1], label:
range[1].toString() }]}
        value={value}

      />
    </>
  );
};

export default BasicRange;
```

5.2 Composant BasicSelect

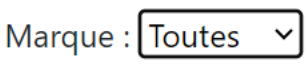


J'ai créé ce composant en un menu déroulant avec des options configurables, telles que les marques de voitures.

Lorsque l'utilisateur sélectionne une option, la fonction '**handleChange**' sera utilisée pour communiquer avec le composant parent.

Cette fonction sera appelée à chaque fois que l'utilisateur modifie la sélection dans le menu déroulant.

Elle prendra en paramètre l'objet '**event**'.



Je vais utiliser le '**destructuring**' pour extraire les propriétés '**value et name**' de l'objet '**event**'.

Je vais ensuite appeler la fonction '**props.handleChange(name, value)**' pour transmettre les valeurs au composant parent ce qui permettra à ce composant de réagir à la sélection effectuée dans le menu déroulant.

```
 const BasicSelect = (props) => {

  const handleChange = (event) => {
      const {value, name} = event.target; //on extrait 2 propriètes value et name de l'objet
      props.handleChange(name, value); //handleChange va transmettre la valeur value et name  au
composant parent
  };

  return (
    <>
      <label>{props.label}</label>

      <select
        name={props.name}

        onChange={handleChange}
      >

      {props.options.map((ele, idx) => { //element , index

        if(
            idx = 0

        )

        {   return(<option selected value={ele.value}>{ele.text}</option>)

         }else{ return(<option value={ele.value}>{ele.text}</option>) }

      })
      }

      </select>
    </>
  );
};

export default BasicSelect;
```

5.3 Composant BasicCheckbox

Ce composant servira à sélectionner une famille de véhicule.

L'utilisateur pourra en sélectionner plusieurs s'il le souhaite ou si rien de cocher, je ferais en sorte de lui présenter tout type de famille de véhicules.

Comme pour les autres composants filtres, la fonction '**handleChange'** sera appelée chaque fois que l'utilisateur coche ou décoche une case.

La fonction prendra en charge un objet '**événement'** en tant que paramètre.

Je vais par la suite transmettre cet événement au composant parent grâce à la fonction '**props.handleCheckBoxChange(e)'**, cela permettra au composant parent de réagir à l'état de la case si cochée ou pas.

```
const BasicCheckbox = (props) =>{

    const handleChange = (e) =>
{props.handleCheckBoxChange(e)
    }
   return ( <label>
    <input
      type="checkbox"
      name={props.name}
      value={props.value}

      onChange={handleChange}
    />
  {props.label }
  </label>
  )
}
export default BasicCheckbox;
```

5.4 Composant Parent 'VehiculeFilters'

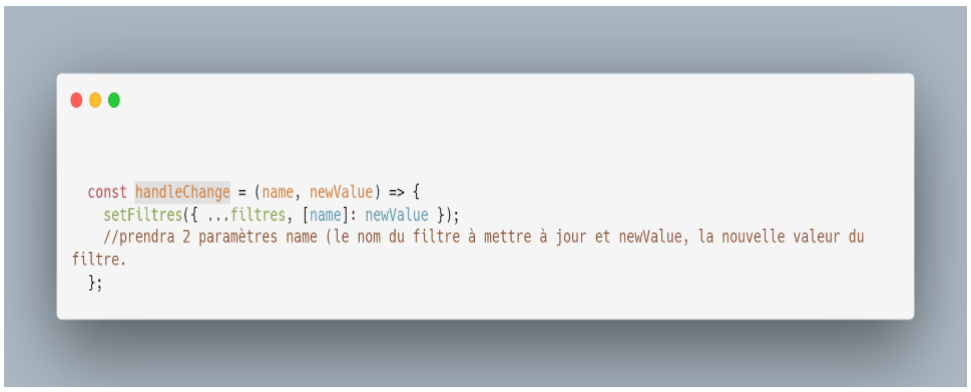
La page nommée "**Voitures d'occasion**" servira donc de parent pour intégrer les composants enfants que j'ai créés précédemment, à savoir `**BasicCheckbox**`, `**BasicSelect**`, et `**BasicRange**`.

C'est effectivement sur cette page que l'utilisateur effectuera sa recherche en ajustant les filtres en fonction de ses choix.

Pour le slider **"année",** j'ai créé une fonction permettant de mettre à jour la date de fin du slider en temps réel, évitant ainsi une mise à jour manuelle chaque année.

```
//Fonction pour obtenir l'année actuelle en utilisant l'objet date.
 const getCurrentYear = () => {
   const dateActuelle = new Date();
   const anneeActuelle = dateActuelle.getFullYear();
   return anneeActuelle;
 };
```

J'ai implémenté la fonction '**handleChange'**, qui est transmise aux composants enfants via les 'props' et qui mettra à jour l'état local des filtres en fonction des modifications apportées par l'utilisateur.



```
  const handleChange = (name, newValue) => {
    setFiltres({ ...filtres, [name]: newValue });
    //prendra 2 paramètres name (le nom du filtre à mettre à jour et newValue, la nouvelle valeur du
filtre.
  };
```

La fonction **handleCheckboxChange** sera utilisée pour gérer les cases à cocher. Elle mettra à jour l'état des filtres en fonction de ce qui est coché ou pas.

Dans cette fonction, j'extrais les 3 propriétés de l'objet '**événement'** : '**name'** qui est le nom de la case à cocher, '**value'** qui est la valeur associée à la case à cocher et '**checked'** qui sera un booléen qui indiquera **'true'** si cochée ou **'false'** si décochée.

```
  const handleCheckBoxChange = (e) => {
    const { name, value, checked } = e.target;
    if (checked) {
      setFiltres({ ...filtres, [name]: [...filtres.famille, value]
}); } else {
      setFiltres({
        ...filtres,
        [name]: filtres.famille.filter((ele) => ele !== value),
      });
    }
  };
```

5.5 Hook UseEffect

Je vais ensuite utiliser le '**hook useEffect**' pour effectuer les requêtes http grâce à '**Fetch**' lorsqu'il sera monté (affiché pour la première fois) grâce à **AXIOS** qui effectuera ces requêtes et ainsi récupérer les données des véhicules en fonction des paramètres de filtres choisi par l'utilisateur.

```
useEffect(() => {
   fetch(
     //fetch effectue une requête http, si reponse, elle sera encapsulé dans une promesse
    lien
     // "http://localhost/GarageBack/API/vehicule.php?
kilometremin=0&kilometremax=200000&anneemin=2000&anneemax=2023&prixmin=5000&prixmax=50000"
   )
   //Si reponse reçu de la requête http, then va gére la reponse de cette promesse et va prendre une
fonction de rappel en argument:

     .then((res) => res.json())// Va extraire les données de l'API sous format json
     .then((data) => {
       setCards(data)
       console.log();
     })//data ou on aurait pu mettre un nom représente la réponse de la requête http
     .catch((err) => console.log(err));//Si erreur de la requête, catch retourne une erreur
 }, [lien]);
```

5.6 Fonction handleClick

Je vais créer une fonction qui va construire une Url dynamique en fonction des filtres sélectionnés dans l'objet '**lienObjet**'. Il supprimera le dernier caractère '**&**' pour obtenir **une Url propre**.

```
const handleClick = ()=> {


        let lienTmp = "http://localhost/garageback/API/vehicules.php?";
        let lienObject = {kilometremin:filtres.kilometrage[0],
                          kilometremax:filtres.kilometrage[1],
                          prixmin:filtres.prix[0],
                          prixmax:filtres.prix[1],
                          anneemin:filtres.annee[0],
                          anneemax:filtres.annee[1],

                        };
        if(filtres.marque.length !== 0){
          lienObject.marque = filtres.marque;
        }

        if(filtres.famille.length !== 0){
          lienObject.famille = filtres.famille.join(",");
        }

          for(const [cle, valeur] of Object.entries(lienObject)){
            lienTmp = lienTmp + `${cle}=${valeur}&`
          }
          lienTmp = lienTmp.slice(0, -1);
          // console.log(lienTmp)
          setLien(lienTmp)
  }
```

6.0 Postman

Avant de créer la variable **`Lien`**, j'ai effectué des tests sur mes URLs de recherche par filtres en utilisant **Postman**, ce qui m'est avéré être d'une grande aide pour la gestion des erreurs.

J’ai renseigné dans mon URL les valeurs **minimales et maximales** des filtres que j'avais précédemment définies.

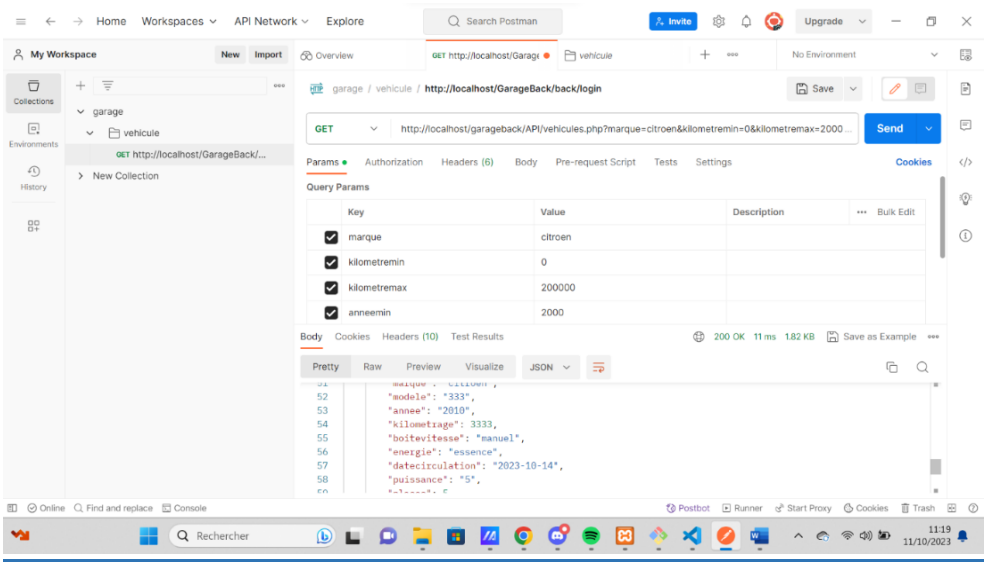
Exemples de Liens de test en méthode GET

Pour tester la recherche par marque :

http://localhost/GarageBack/API/vehicule.php?marque=citroen

Pour tester la recherche de tous les filtres avec les valeurs de début et de fin renseignées :

http://localhost/GarageBack/API/vehicule.php?kilometremin=0&kilometremax=200000&anneemin=2000&anneemax=2023&prixmin=5000&prixmax=50000



Bien sûr après que tous ces paramétrages ont été réalisé, je retourne tous les composants de filtrage.

Difficultés rencontrées ‘Recherches par filtres’ :

Cette partie du développement a été l'une des plus complexes pour moi.

Tout d'abord, j'ai initialement essayé de créer mes propres composants de sliders, de cases à cocher, etc., alors que des bibliothèques existaient pour simplifier ce processus mais au départ, je n’en avais pas eu connaissance, n’étant pas satisfait du résultat, j’ai réalisé plusieurs recherches pour trouver une solution et proposer au client un front plus présentable.

J'ai aussi éprouvé des difficultés à bien cibler mes recherches, mais j'ai heureusement bénéficié de conseils avisés qui m'ont facilité la tâche.

Mon apprentissage s'améliore au fil du temps que je pratique, et je suis désormais conscient des ressources disponibles pour résoudre de tels problèmes techniques.

De plus, j'ai rencontré des défis lors de la mise en place de la fonction ‘handleChange’ et de sa transmission en tant que ‘props’ aux composants enfants, mais j'ai compris l'importance de cette fonction pour établir la communication entre les composants.

J'ai également été confronté à plusieurs erreurs liées à des noms de composants mal orthographiés, tant au niveau des noms de fichiers que des imports.

Je suis conscient de la sensibilité à la casse dans ce contexte, mais dans ce cas précis, j'ai eu du mal à identifier l'origine de l'erreur, car tous les noms semblaient correctement écrits.

Pour résoudre ce problème, j'ai dû supprimer tous les fichiers des composants et de les recréer un à un afin que l'erreur cesse de s'afficher.

C'est l'un des mystères de l'informatique qui peut parfois entraîner une perte de temps considérable pour ce qui semble être une petite erreur en apparence.

La convention stipule que les fichiers doivent commencer par une lettre majuscule mais de manière inexplicable, il arrive parfois que certains fichiers se renomment automatiquement, perdant ainsi leur lettre majuscule initiale.

Cela provoque des erreurs lors de l'importation de ces fichiers. Je ne parviens pas à expliquer pourquoi cela se produit, mais cette situation devient de plus en plus frustrante à chaque occurrence.

7.0 Prestations

En ce qui concerne la présentation des services du garagiste, le système est presque identique à celui utilisé pour afficher les informations des véhicules.

Les services sont affichés sous forme de "cartes" Bootstrap.

8.0 Avis Clients

Cette partie représente la dernière étape que j'ai pu accomplir dans ce projet, mais elle n'est pas encore achevée à ce stade.

Je récupère bien mes données en Get mais en Post, je n'y suis pas encore parvenu, j'ai des problèmes au niveau de la requête.

J'ai créé un composant d'étoiles pour la notation et j'ai installé la bibliothèque **'npm install react-simple-star-rating**'. **Opérationnel**

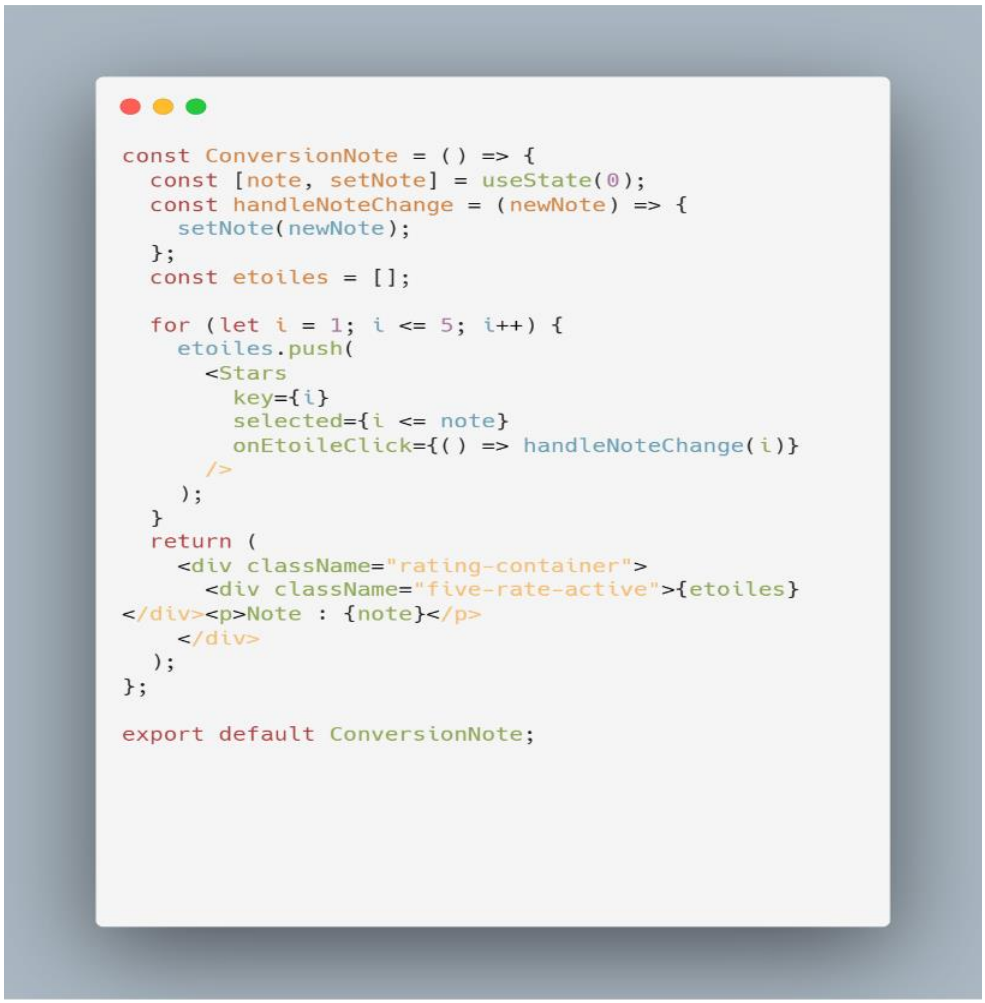
Il n'y a même pas besoins de cliquer sur les étoiles pour les sélectionner, un survol de la souris suffira.

J'ai décidé ici de ne pas inclure les demis étoile pour l'instant, je verrais cela plus tard.

```
const Stars = () => {
    const [rating, setRating] = useState(100) // Va jusqu'à 100 donc la 5 étoiles
    const handleRating = (rate) => {
      console.log(rate)
      setRating(rate)

    }
    return (
      <Rating
        fillColor="#F0C300"
        //allowHalfIcon //Pour les demi étoile, là, on est à true
        // tooltipArray={['nul', 'bof', 'moyen', 'top', 'génial']}
        transition
        // showTooltip
        onClick={handleRating}
        ratingValue={rating}
      />
    )
  }
  export default Stars;
```

Je vais maintenant convertir les étoiles en note en créant un nouveau composant '**ConversionNote**'

A l'intérieur de la fonction, je vais créer une variable d'état 'note' à l'aide du Hook '**UseState**' et je vais l'initialiser à **0**.

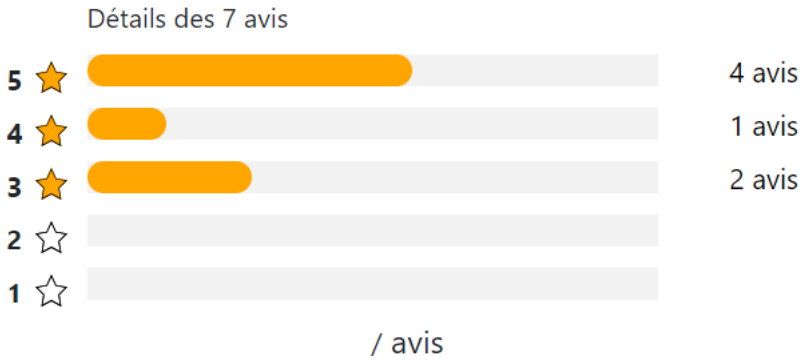


```
const ConversionNote = () => {
  const [note, setNote] = useState(0);
  const handleNoteChange = (newNote) => {
    setNote(newNote);
  };
  const etoiles = [];

  for (let i = 1; i <= 5; i++) {
    etoiles.push(
      <Stars
        key={i}
        selected={i <= note}
        onEtoileClick={() => handleNoteChange(i)}
      />
    );
  }
  return (
    <div className="rating-container">
      <div className="five-rate-active">{etoiles}
</div><p>Note : {note}</p>
    </div>
  );
};

export default ConversionNote;
```

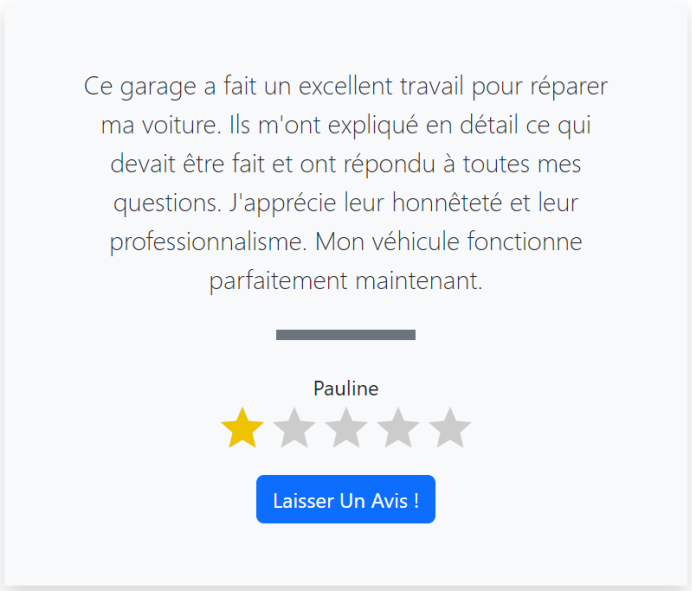
J’ai testé dans la console en y ajoutant un echo, ça fonctionne parfaitement


```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Error while trying to use the following icon from the Manifest: http://localhost:3000/logo192.png (Download error or resource isn't a valid image)   avisClients:1
4                                                                                                                                                    Rating.js:8
4                                                                                                                                                    Rating.js:8
>
```

J'ai créé un composant destiné à informer l'utilisateur et à générer un bilan des avis clients (bien que non opérationnel pour le moment).



Par la suite, sur la page d'accueil, j'ai l'intention d'afficher les avis de manière dynamique. Pour ce faire, je vais générer un nombre aléatoire qui sélectionnera aléatoirement un avis à afficher.



9.0 Conclusion

En fin de compte, la mise en place des composants en ce qui concerne la recherche par filtre a été une étape assez instructive en ce qui me concerne par rapport à mon apprentissage.

J'ai été confronté à de nombreux problèmes, que ce soit avec Git et Github, la connexion à la base de données, la création des tables et bien entendu le code.

Souvent, je passais trop de temps à me battre avec un problème, mais j'ai fini par réaliser que cela ne servait à rien de rester des jours à essayer de le résoudre.

Maintenant, je préfère passer à autre chose temporairement, le temps de réfléchir à la meilleure solution pour résoudre le problème. Cette méthode me convient assez bien.

Cette expérience m'a permis de développer mes compétences en gestion de bases de données et d'acquérir une compréhension plus approfondie des aspects pratiques du développement.

Mon apprentissage continu dans ce domaine m'aidera à aborder de futurs projets avec plus de confiance et de compétence.

J'ai pris conscience que la mémorisation forcée ne mène à rien, car il est impossible d'absorber l'ensemble des informations par cœur.

La méthode qui fonctionne le mieux pour moi consiste à pratiquer de manière intensive tout en effectuant des recherches constantes.

FRONT END

PARTIE 2

Ce rapport présente en détail les étapes clés et les réalisations majeures au cours du développement Front-End d'un projet client que j'ai entièrement pris en charge, de la conception initiale à la mise en œuvre finale.

Pour la réalisation de ce projet, j'ai opté pour un code en pur JavaScript, sans recourir à l'utilisation de frameworks, et j'ai optimisé l'esthétique en intégrant Bootstrap de manière exhaustive via les CDN.

Projet Client ‘ Les Caravanes De La Besbre’ : https://lescaravanesdelabesbre.fr/

Lien github : https://github.com/Michelhof1978/lesCaravanesDeLaBesbre

Technologies Utilisées : Html, Css, Bootstrap, Javascript et Php.

Table de matière

1.0 Zoom image lors du survol de la souris

Dans ce code JavaScript, j'ai défini 2 fonctions, **zoomIn** et **zoomOut**, qui permettront respectivement de **zoomer** sur une **image** en augmentant son **échelle** à 1.5 fois sa taille initiale, et de **dézoomer** en rétablissant son échelle à la taille initiale (1).

Les transitions seront animées sur une durée de 0.5 seconde pour un effet visuel fluide.

Ces fonctions seront utilisées lors **survol de la souris** pour créer une interaction **dynamique** avec les images de ma page web.

-Agrandir l'image 1.5 fois et remettre l'image à sa taille d'origine

```
<script>
    function zoomIn(img) {
        img.style.transform = "scale(1.5)";
        img.style.transition = "transform 0.5s";
    }

    function zoomOut(img) {
        img.style.transform = "scale(1)";
        img.style.transition = "transform 0.5s";
    }
</script>
```

2.0 Mise à jour automatique de la date du Copyright du Footer

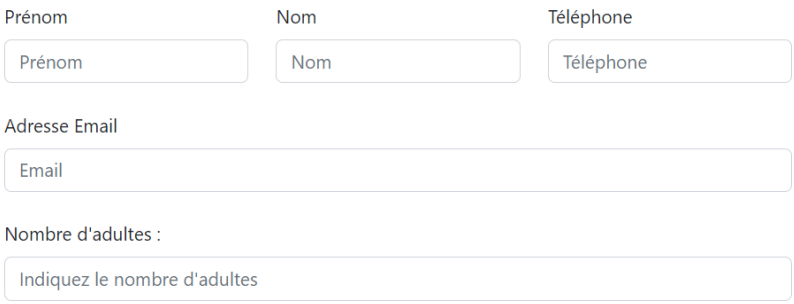
J'ai mis en place ce code JavaScript qui va récupérer **l'année actuelle**, puis va mettre à jour le contenu de l'élément **HTML** du footer ayant ***l'ID "date"*** avec l'année actuelle.

Ensuite, je vais appeler la fonction toutes les **secondes** à l'aide de **setInterval**, assurant ainsi que l'année affichée est constamment mise à jour en temps réel dans le footer.

```
<script>
    function afficherDate() {

        var date = new Date().getFullYear();
        document.getElementById('date').textContent = date;
    }
    setInterval(afficherDate, 1000);
</script>
```

3.0 Formulaire de réservation

3.1 Ajout de restrictions lors du remplissage des champs par l'utilisateurs.



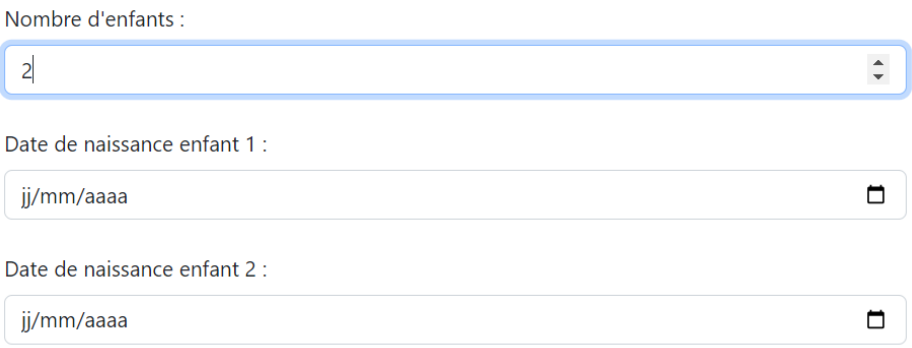
```
<script>
    function validateForm() {
        let emailInput = document.getElementById('email');
        let emailValue = emailInput.value.trim();
        let emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

        if (!emailRegex.test(emailValue)) {
            alert('Veuillez saisir une adresse email valide.');/
            emailInput.focus();
            return false;
        }

        let phoneNumberInput = document.getElementById("phoneNumber");
        let phoneNumberValue = phoneNumberInput.value;
        let phoneRegex = /^[0-9]+$/;

        if (!phoneRegex.test(phoneNumberValue)) {
            alert("Veuillez saisir uniquement des chiffres pour le numéro de téléphone.");
            return false;
        }
```

Dans ce script, je vais faire en sorte que toutes les **conditions** soient réunies à la **validation** d'un formulaire, je vais vérifier ainsi la validité de **l'adresse e-mail** et du **numéro de téléphone** saisis par l'utilisateur.

Je vais utiliser les expressions régulières pour m'assurer que l'adresse e-mail est correcte et que le numéro de téléphone ne contient que des chiffres et rien d'autres.

En cas d'échec de validation, des alertes ciblés devront être affichées pour informer l'utilisateur, avec une attention particulière portée sur le champ d'adresse e-mail, où le focus est automatiquement déplacé pour faciliter la correction.

3.2 Ajout dynamique de nouveaux champs de date de naissance en fonction du nombre d'enfants sélectionnés.



-Fonction pour ajouter dynamiquement les champs de date de naissance des enfants

```
function ajouterChampsDateNaissance() {
        const nombreEnfants = document.getElementById('nombreEnfants').value;
        const containerDatesNaissance = document.getElementById('containerDatesNaissance');

        containerDatesNaissance.innerHTML = '';

        for (let i = 1; i <= nombreEnfants; i++) {
            const divRow = document.createElement('div');
            divRow.className = 'row mb-4';

            const divCol = document.createElement('div');
            divCol.className = 'col';

            const label = document.createElement('label');
            label.className = 'form-label';
            label.setAttribute('for', 'dateNaissanceEnfant' + i);
            label.innerText = 'Date de naissance enfant ' + i + ' :';

            const inputDate = document.createElement('input');
            inputDate.name = 'dateNaissanceEnfant' + i;
            inputDate.type = 'date';
            inputDate.id = 'dateNaissanceEnfant' + i;
            inputDate.className = 'form-control';
            inputDate.required = true;

            divCol.appendChild(label);
            divCol.appendChild(inputDate);
            divRow.appendChild(divCol);

            containerDatesNaissance.appendChild(divRow);
        }
    }
```

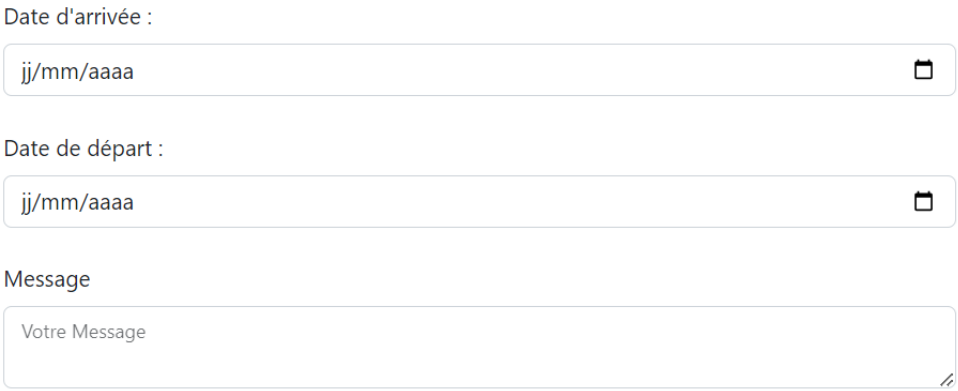
Je vais mettre en place cette fonction JavaScript, '**ajouterChampsDateNaissance()**' qui permettra d'ajouter dynamiquement des champs de date de naissance pour chaque enfant en fonction du nombre spécifié par l'utilisateur.

Je vais faire en sorte d'obtenir le nombre d'enfants à partir d'un champ de formulaire, puis de créer des éléments HTML (divisions, labels, et champs de date) à l'intérieur d'un conteneur dédié.

Je vais faire en sorte que chaque champ de date soit associé à un numéro d'enfant unique.

Je vais mettre en place la fonction qui utilisera une boucle '**for**' pour générer ces éléments de manière itérative, cela va assurer ainsi une création dynamique en fonction du nombre d'enfants spécifié dans le formulaire.

3.3 Validation des dates d'arrivée et de départ.



-Validation des dates d'arrivée et de départ

```
let dateArriveeInput = document.getElementById('dateArrivee');
        let dateDepartInput = document.getElementById('dateDepart');
        let dateArriveeValue = dateArriveeInput.value.trim();
        let dateDepartValue = dateDepartInput.value.trim();


        if (dateArriveeValue === '' || dateDepartValue === '') {
            alert('Les dates d\'arrivée et de départ ne peuvent pas être
vides.');    return false;
        }

        if (new Date(dateArriveeValue) > new Date(dateDepartValue)) {
            alert('La date de départ doit être ultérieure à la date
d\'arrivée.');    dateDepartInput.focus();
            return false;
        }
```

Ensuite, je vais valider les champs de date d'arrivée et de départ dans ce formulaire de réservation.

On va obtenir les valeurs de ces champs, qui va supprimer les espaces blancs, puis effectuera les vérifications suivantes :

1- Je vais mettre en place une vérification si l'une des dates (d'arrivée ou de départ) soit vide ou pas.

En cas de champ vide, cela affichera une alerte indiquant que les dates d'arrivée et de départ ne peuvent pas être vides, et renvoie **'false'** pour signaler que la validation a échoué.

2- Ensuite, j'ai pensé à faire vérifier si la date de départ est ultérieure à la date d'arrivée en cas d'erreur de saisie par l'utilisateur.

En cas de non-respect de cette condition, cela affichera une **alerte** informant que la date de départ doit être **ultérieure** à la date d'arrivée, place le **focus** sur le champ de date de départ pour attirer l'attention de l'utilisateur, et renvoie false pour indiquer que la validation a échoué.

3.4 Vérification si la case du consentement RGPD et CGV sont bien cochées.

-Vérification si la case du consentement RGPD est cochée

Je vais faire en sorte d'obliger l'utilisateur à valider si une case à cocher liée à la politique de confidentialité **(RGPD)** est cochée dans un formulaire, s'il refuse, le formulaire ne pourra pas être soumis.

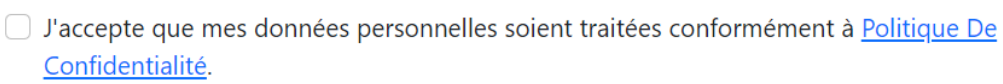
Si la case n'est pas cochée, une alerte sera affichée demandant à l'utilisateur d'accepter **la politique de confidentialité**.

De plus, le focus est placé sur la case à cocher pour attirer l'attention de l'utilisateur, et la fonction renvoie **false**, signalant ainsi que la validation du formulaire a échoué.

Le Règlement Général sur la Protection des Données (RGPD), également connu sous l'acronyme anglais GDPR (General Data Protection Regulation), est une réglementation européenne qui régit la protection et le traitement des données personnelles des citoyens de l'Union européenne (UE).

```
let rgpdCheckbox = document.getElementById('rgpdCheckbox');
if (!rgpdCheckbox.checked) {
    alert('Vous devez accepter la politique de confidentialité.');
    rgpdCheckbox.focus();
    return false;
}
```

Il a été adopté en 2016 et est entré en vigueur le 25 mai 2018. Le RGPD vise à renforcer la protection de la vie privée des individus et à harmoniser les règles relatives à la protection des données au sein de l'UE.

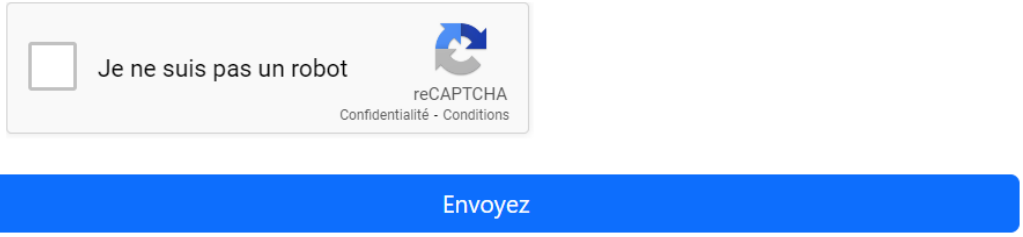


-Vérification si la case du consentement RGPD est cochée

En ce qui concerne les conditions générales de vente, c'est un outil important pour établir un cadre légal et transparent dans les transactions en ligne, pouvant assurer une relation équitable entre le vendeur et l'acheteur, tout en respectant les exigences légales en vigueur.

```
let cgvCheckbox = document.getElementById('cgvCheckbox');
      if (!cgvCheckbox.checked) {
          alert('Vous devez accepter les Conditions Générales de Vente.');
          cgvCheckbox.focus();
          return false;
      }
```

3.5 Vérification si la réponse reCAPTCHA n'est pas vide.



-Vérification si la réponse reCAPTCHA n'est pas vide

```
            let recaptchaResponse =
grecaptcha.getResponse();
            if (recaptchaResponse.length == 0) {
                alert('Veuillez cocher le reCAPTCHA.');
                return false;
            }
            return true;
        }
</script>
```

Je vais faire pareil que pour le **rgpd et le cgv**, je vais faire en sorte de vérifier la validité de la réponse **reCAPTCHA** du formulaire.

Il aura l'obtention de la réponse **reCAPTCHA** à l'aide de la bibliothèque **reCAPTCHA API**, puis effectue les étapes suivantes :

Cela va vérifier si la réponse **reCAPTCHA** est vide **(non cochée)**.

En cas de réponse non cochée, cela affichera une alerte demandant à l'utilisateur de cocher le **reCAPTCHA**.

Renvoie false pour indiquer que la validation a échoué en raison du **reCAPTCHA** non cochée, ce qui empêchera l'envoi du formulaire.

Si toutes les validations précédentes réussissent, la fonction renvoie '**true'**, indiquant que le formulaire est valide.

C'est une pratique recommandée en matière de sécurité lors de l'utilisation de clés API, telles que celles nécessaires pour intégrer le service reCAPTCHA.

En règle générale, les clés API sont des chaînes de caractères sensibles et confidentielles qui permettent d'authentifier et d'autoriser l'accès à des services tiers.

Pour éviter tout accès non autorisé ou malveillant, j'ai fait en sorte de ne pas exposer ces clés directement dans le code source accessible au public.

Dans le contexte spécifique du reCAPTCHA, la clé **API** est généralement utilisée pour valider les réponses du reCAPTCHA côté serveur.

Pour garantir la sécurité, j'ai donc stocké la clé dans un fichier de configuration '**.gitignore'** en lui indiquant l'emplacement du fichier à protéger et qui sera donc à part du code source principal de l'application.

Cela me permettra qu'elle soit exposée à quiconque aurait accès au code source de la page web.

Difficultés rencontrées :

Initialement, le **reCAPTCHA** n'avait pas été intégré en raison de contraintes de temps que j'avais.

J'ai informé le client de cette situation et l'ai alerté sur la nécessité de vérifier ses courriers indésirables, mais malheureusement, cela n'a pas été pris en compte.

De nombreux contacts de clients potentiels se retrouvaient dans les spams, entraînant des pertes financières pour l'entreprise du client qui n'a donc pas pensé à vérifier ses spams.

Après la mise en place du **reCAPTCHA** et en dirigeant les messages des utilisateurs via le service de mailing d'**OVH**, les filtres anti-spam ont réussi à acheminer correctement les messages vers le dossier de réception, améliorant ainsi la gestion des communications.

4.0 Affichage d’un Popup

J’ai créé un pop-up en image avec un message centré sur la page web lors d’une première connexion d’un utilisateur.

Il utilisera donc des **cookies** pour s'assurer que le pop-up puisse être affiché **qu'une seule fois** par utilisateur.

Les principales étapes sont les suivantes :

1-Vérification du Popup : J'ai créé une fonction `**isPopupShown**` qui examine les cookies pour déterminer si le popup a déjà été affiché. Cela va garantir que le popup n'est pas répété à chaque fois que la page a été rechargée.

```
function isPopupShown() {
    return document.cookie.indexOf("popupShown=true") !==
-1}
```

2-Définition du Cookie : Une fonction `**setPopupShown**` est mise en place pour définir un cookie indiquant que le popup a été affiché.

Cela permet de mémoriser l'état du popup.

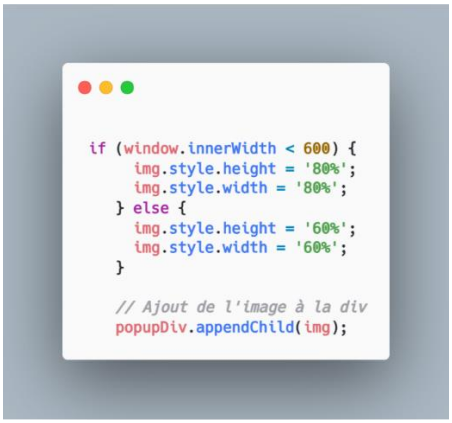
```
function setPopupShown() {
  document.cookie = "popupShown=true; expires=Thu, 01 Jan 2030 00:00:00 UTC; path=/";
}
```

3-Création du Popup : Si le popup n'a pas encore été montré, une nouvelle div est générée dynamiquement.

Cette div contiendra une image, un bouton de fermeture (croix) ainsi qu'un message.

```
if (!isPopupShown()) {
    let popupDiv = document.createElement("div");
    popupDiv.style.position = "fixed";
    popupDiv.style.top = "50%";
    popupDiv.style.left = "50%";
    popupDiv.style.transform = "translate(-50%,
-50%)";popupDiv.style.zIndex = "9999";
    popupDiv.style.textAlign = "center";
    popupDiv.style.background = "white";
    popupDiv.style.padding = "20px";
    popupDiv.style.border = "1px solid #ccc";
    popupDiv.style.borderRadius = "8px";

    // Création de l'image à afficher
    let img = document.createElement("img");
    img.src = "../images/lePal2024.png";
```

Adaptation de la Taille de l'Image : La taille de l'image est ajustée en fonction de la largeur de l'écran. Si la largeur est inférieure à 600 pixels, l'image occupe 80% de l'écran, sinon elle prendra 60%.



```
if (window.innerWidth < 600) {
      img.style.height = '80%';
      img.style.width = '80%';
    } else {
      img.style.height = '60%';
      img.style.width = '60%';
    }

    // Ajout de l'image à la div
    popupDiv.appendChild(img);
```

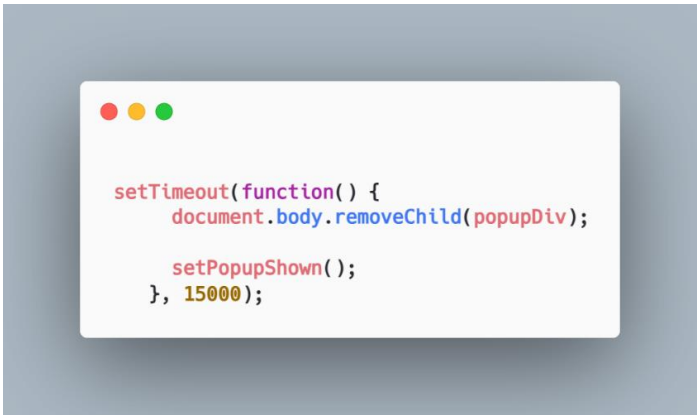
5-Fermeture du Popup : Un bouton de fermeture est ajouté au popup, et un gestionnaire d'événements est attaché pour fermer le popup lorsque le bouton est cliqué.

La div du popup est également supprimée du corps de la page.

```
let closeButton = document.createElement("button");
   closeButton.innerHTML = "X";
   closeButton.style.position = "absolute";
   closeButton.style.top = "10px";
   closeButton.style.right = "10px";
   closeButton.style.cursor = "pointer";
   closeButton.style.border = "none";
   closeButton.style.background = "transparent";
   closeButton.style.fontSize = "16px";
   closeButton.addEventListener("click", function () {
     document.body.removeChild(popupDiv);
     setPopupShown();
   });

   popupDiv.appendChild(closeButton);

   let message = document.createTextNode("");
   popupDiv.appendChild(document.createElement("br"));
   popupDiv.appendChild(message);

   document.body.appendChild(popupDiv);
```

6-Durée d'Affichage : Le popup est affiché pendant 15 secondes avant d'être automatiquement supprimé.

Cette durée peut être ajustée en fonction des besoins.

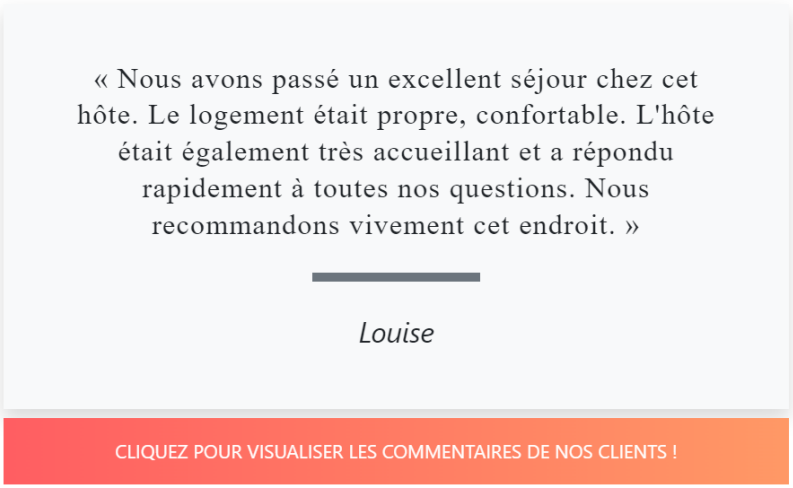


```
setTimeout(function() {
     document.body.removeChild(popupDiv);

     setPopupShown();
   }, 15000);
```

J’ai fait en sorte que ce code puisse assurer une expérience utilisateur engageante en présentant un popup avec une image et un message, tout en veillant à ne pas gêner l'utilisateur avec des affichages répétitifs grâce à l'utilisation de cookies.

La taille de l'image est également adaptée pour offrir une présentation optimale sur différents appareils.

5.0 Affichage dynamique avis clients



Je vais mettre en place un affichage **aléatoire** d'avis clients sur la page principale du site avec l’aide d’un bouton.

Je vais utiliser des **événements** de clic pour générer **aléatoirement** un avis à partir d'un **tableau** prédéfini.

Les principales étapes que j’ai effectuées incluront la création **d'événements de clic**, la **génération d'indices aléatoires** distincts, la **mise à jour des éléments HTML** avec **le texte de l’avis** et le **nom du client auteur**, je vais ensuite éviter la répétition de l'affichage du même avis consécutivement.

```
let nouveau = document.querySelector('#nouveau');
let avi = document.querySelector('#avi');
let auteur = document.querySelector('#auteur');
let dernier = 0;
let nombreAleatoire = 0;

let avis = [
  ["Notre séjour chez Isabelle a été super ! L'hôte était très accueillant et la caravane était
propre, confortable et bien situé. Nous avons vraiment apprécié notre séjour et nous recommandons
vivement cet endroit !", "Carole"],
  ["Isabelle était très attentionnée et serviable, nous donnant des conseils sur les meilleurs
endroits à visiter dans le coin. Nous avons adoré notre séjour ici et nous y retournerons certainement
!", "Bastien"],

];
```

Tout d'abord, je vais déclarer une variable '**nouveau**' en utilisant '**document.querySelector ('#nouveau')**'.

Cette variable représente l'élément **HTML** avec **l'ID** "nouveau" sur ma page, qui sera un bouton qui déclenchera l'événement lors du clic de l'utilisateur.

Ensuite, je vais déclarer deux autres variables, **avi** et **auteur**, en utilisant **document.querySelector('#avi')** et **document.querySelector('#auteur')**, respectivement. Ces variables représentent des éléments HTML avec les ID "avi" et "auteur".

J'initialise également deux autres variables, '**dernier**' **et** '**nombreAleatoire**', à **0**. '**dernier**' sera utilisé pour suivre le dernier avis affiché, et **nombreAleatoire** sera utilisé pour stocker le nombre généré aléatoirement.

Ensuite, je vais créer un tableau appelé '**avis**' qui contiendra les commentaires clients sous forme de tableaux imbriqués.

Chaque sous-tableau a deux éléments : le commentaire lui-même et le nom de l'auteur qui sera associé au commentaire.

-Fonction qui génère un nombre aléatoire entre 0 (inclus) et max (exclus).

```
function genererNombreEntier(max) {
   return Math.floor(Math.random() * Math.floor(max));
 }

 nouveau.addEventListener('click', () => {
   do {
     nombreAleatoire =
genererNombreEntier(avis.length);
   } while (nombreAleatoire === dernier)

   avi.textContent = avis[nombreAleatoire][0];
   auteur.textContent = avis[nombreAleatoire][1];
 });
```

Je vais définir par la suite une fonction appelée' **genererNombreEntier(max)**'.

Cette fonction utilise '**Math.random()**' qui va générer un nombre décimal entre 0 (inclus) et 1 (exclus), puis le multiplie par '**max'** et applique '**Math.floor'** pour obtenir un nombre entier.

En d'autres termes, elle génère un nombre entier aléatoire entre **0 et max – 1** (derniers indice -1 du tableau comme on comme à l'indice 0).

Cette fonction sera utilisée pour obtenir un index aléatoire dans le tableau avis.

Ensuite, je vais attacher un écouteur d'événements au bouton identifié par '**nouveau'**.

Lorsque ce bouton est cliqué, la fonction fléchée anonyme sera déclenchée.

À l'intérieur de cette fonction, j'utilise une boucle **'do...while'** pour générer un nouveau nombre aléatoire (**nombreAleatoire**) en utilisant la fonction **genererNombreEntier(avis.length).**

La boucle continuera à générer des nombres aléatoires tant que le nombre généré est égal au dernier nombre généré (**dernier**). Cela garantit que l'avis affiché ne soit pas le même que le précédent.

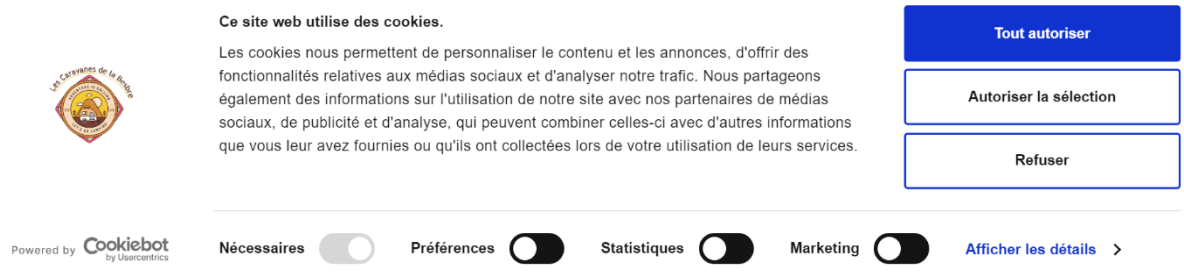
Une fois qu'un nombre aléatoire différent est généré, je ferais en sorte d'accéder au tableau avis à l'index '**nombreAleatoire'**. J'utiliserai cet index pour extraire le commentaire (**avis[nombreAleatoire][0])** et le nom de l'auteur (**avis[nombreAleatoire][1]**).

Enfin, pour términer, je vais attribuer le commentaire à l'élément HTML avec l'ID "**avi**" en utilisant '**avi.textContent'** et le nom de l'auteur à l'élément avec l'**ID "auteur**" en utilisant '**auteur.textContent'**.

Cela mettra à jour le contenu de la page avec le nouvel avis et son auteur à chaque clic sur le bouton "**nouveau**".

6.0 Les Cookies

-Avec CookiesBot (non concluant en ce qui concerne le service gratuit)



Je vais intégrer un script **Cookiebot** qui est un service de gestion des cookies, sur le site.

L'identifiant unique (**data-cbid**) spécifié sera associé à la configuration du compte **Cookiebot** que j'ai créé auparavant.

Le mode de blocage des cookies est défini sur "auto", ce qui signifiera que le service bloquera automatiquement les cookies non essentiels jusqu'à ce que l'utilisateur donne son consentement.

-Création manuellement du consentement des Cookies

Je vais créer manuellement une bannière de cookies avec une demande explicite d'acceptation ou de refus de l'utilisation des cookies.

Les actions associées aux boutons sont définies comme des fonctions JavaScript externes (**acceptCookies** et **refuseCookies**) qui devraient être implémentées ailleurs dans le code.

-Template du consentement des Cookies

```
<div id="cookie-banner">
    <p>Nous utilisons des cookies pour améliorer votre expérience sur notre site. Acceptez-vous
l'utilisation des cookies ?</p>
    <button onclick="acceptCookies()">Accepter</button>
    <button onclick="refuseCookies()">Refuser</button>
</div>
```

-Fonction pour obtenir la valeur d'un cookie

```
function getCookie(name) {
        var match = document.cookie.match(new RegExp('(^| )' + name + '=
([^;]+)'));if(match) return match[2];
    }
```

Je vais créer une fonction que je vais appeler '**getCookie(name)**' qui me permettra de récupérer la valeur d'un cookie spécifique à partir du document.

Lorsque j'appellerais cette fonction, je devrais fournir le nom du cookie que je souhaite récupérer en tant que paramètre.

À l'intérieur de cette fonction, je vais utiliser l'objet '**document.cookie'**, qui contient toutes les informations sur les cookies associés à la page sous forme d'une chaîne de caractères.

Pour rechercher le cookie spécifique que je veux, j'utiliserai une expression régulière (**new RegExp(...)**) qui ciblera le nom du cookie dans la chaîne '**document.cookie'**. L'expression régulière identifiera le début du cookie, le nom du cookie, le signe égal séparant le nom et la valeur, et enfin, elle capturera la valeur du cookie jusqu'au prochain point-virgule.

Si le cookie est trouvé, la méthode '**match'** renvoie un **tableau** avec les correspondances.

En utilisant la condition **if (match)**, je vais faire en sorte de vérifier si le cookie existe bel et bien.

Si c'est le cas, je retourne la valeur du cookie, qui est située à la deuxième position du tableau **(match[2]**). Si le cookie n'a pas été pas trouvé, la fonction renverra '**undefined'**.

-Fonction pour définir un cookie avec une durée d'expiration (en jours)

```
function setCookie(name, value, days) {
        var expires = '';
        if (days) {
            var date = new Date();
            date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
            expires = '; expires=' + date.toUTCString();
        }
        document.cookie = name + '=' + value + expires + '; path=/';
    }
```

Je crée ici une fonction nommée '**setCookie** avec en paramètre **(name, value, days)'**, ce qui me permettra de définir un cookie avec un nom, une valeur et une durée d'expiration en jours.

Lorsque j'appellerai cette fonction, je devrais fournir le nom du **cookie, sa valeur, et éventuellement, le nombre de jours** pendant lesquels je souhaite que le cookie soit valide.

À l'intérieur de cette fonction, je vais commencer par définir une **chaîne vide** nommée '**expires'**. Ensuite, je vais vérifier si j'ai bien spécifié une durée d'expiration en jours.

Si c'est le cas, je vais créer par la suite un objet '**Date'** représentant le moment actuel, j'ajoute le nombre de jours spécifié en **millisecondes**, puis j'obtiens une représentation en **chaîne** de cette date au format **UTC** (**AAAA-MM-JJTHH:MM:SSZ**).

En fonction de la présence d'une durée d'expiration, j'ajouterais la partie correspondante à l'expiration dans la chaîne expires.

Si aucune durée d'expiration n'est spécifiée, la chaîne '**expires'** reste vide.

Enfin, j'utilise la propriété '**document.cookie**' pour définir le cookie en concaténant le nom, la valeur, l'expiration, et le chemin du cookie. Le cookie sera donc créé avec les paramètres qui lui auront été spécifiés.

-Fonction appelée lorsqu'un utilisateur accepte les cookies

```
function acceptCookies() {
        setCookie('cookieConsent', 'accepted', 365);

        document.getElementById('cookie-banner').style.display =
'none';}
```

Sur cette partie de code, je vais créer une fonction nommée '**acceptCookies**' qui, lorsqu'elle sera appelée, définira un **cookie** indiquant que l'utilisateur a accepté les cookies.

Je nommerais le cookie **'cookieConsent'** et est défini comme **'accepted'** pour une durée de **365 jours**.

Ensuite, je vais ajouter du code supplémentaire pour définir d'autres cookies ou effectuer des actions nécessaires.

Enfin, je manipulerai l'élément **HTML** avec **l'ID 'cookie-banner'**, en le masquant pour indiquer que l'utilisateur a accepté les cookies

-Vérifier si l'utilisateur a déjà accepté les cookies

```
function refuseCookies() {

        document.getElementById('cookie-banner').style.display = 'none';
    }

    if (getCookie('cookieConsent') !== 'accepted') {
        setTimeout(function() {
            document.getElementById('cookie-banner').style.display =
'block';}, 2000);
    }
```

Pour finir, j'ai créé une fonction appelée `**refuseCookies**` qui, lorsqu'elle sera appelée, masque la bannière de cookies en définissant la propriété `**display**` de l'élément avec **l'ID 'cookie-banner' sur 'none'**.

Je vais ainsi vérifier si l'utilisateur a déjà accepté les cookies ou pas en consultant le cookie **'cookieConsent'**.

Si l'utilisateur n'a pas encore accepté les cookies, j'afficherai la bannière de consentement après un délai de 2000 millisecondes (2 secondes) en modifiant la propriété `**display**` de l'élément **'cookie-banner'** à **'block'**.

7.0 Conclusion

Gestion des Spams

Durant la création de ce projet, j'ai rencontré des difficultés mineures, notamment la réception de courriels de clients potentiels dirigés vers les spams, ce qui a été source de frustration pour moi et le client.

Conformité Réglementaire

Une autre priorité a été de garantir la conformité avec la législation française et européenne. Cela a impliqué l'intégration du RGPD, l'obtention des autorisations pour les cookies, ainsi que l'ajout des mentions légales et politiques de confidentialité.

Accessibilité aux Normes WCAG

Un aspect essentiel de mon projet était de rendre le site accessible à tous, y compris aux **personnes handicapées**, en respectant le plus possible des normes **A, AA et AAA du Web Content Accessibility Guidelines (WCAG) édité par le World Wide Web Consortium (W3C)**.

Niveau A : Imposant les exigences fondamentales d'accessibilité.

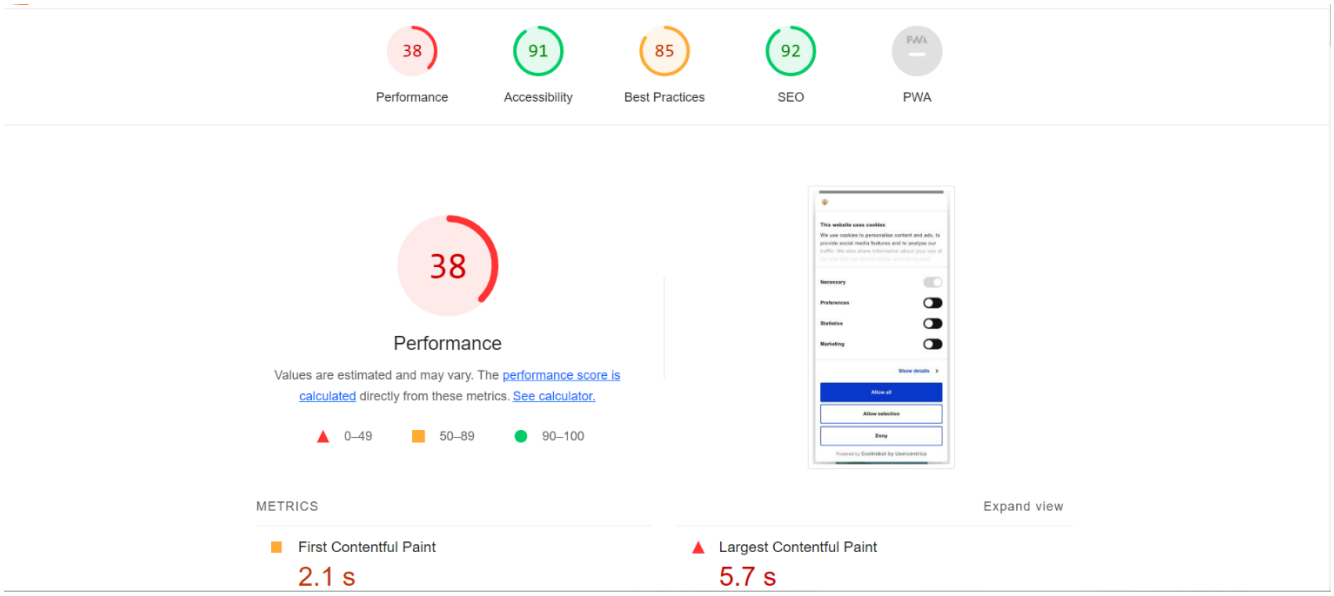
Niveau AA : Recommandé pour atteindre un niveau d'accessibilité décent, souvent considéré comme la norme minimale.

Niveau AAA : Imposant les exigences les plus strictes en matière d'accessibilité.

Les détails exhaustifs des normes d'accessibilité se trouvent sur le site officiel du **W3C**, où sont publiées les **Web Content Accessibility Guidelines (WCAG).**

Ces directives détaillent les critères d'accessibilité à différents niveaux, offrant des conseils précieux pour rendre les contenus web accessibles à un large public, y compris les personnes handicapées.

J'ai également utilisé l'extension **Lighthouse** de Google, qui m'a fourni des conseils avisés pour améliorer mon site conformément à mes objectifs en atteignant **91%** en moyenne d'accessibilité.



Optimisation du Référencement SEO

J'ai pris en charge le référencement **SEO** gratuit sur **Google** en effectuant une recherche approfondie des mots-clés les plus pertinents, en créant des liens utiles et en intégrant des balises **H1** aussi pertinentes que possible.

En effectuant cette démarche, j'ai rencontré un succès notable en positionnant le site en bonne place dans les résultats de recherche de la première page de Google.

J'ai profité de l'opportunité pour élaborer une **fiche établissement sur Google**, augmentant ainsi la visibilité de mon site et renforçant sa présence en ligne.

HOFFMANN MICHEL