

# PARTIE BACK END



## Introduction

Ce projet a été élaboré pour un client fictif, le propriétaire d'un garage qui a exprimé le souhait de vendre des voitures d'occasion et de proposer ses services de réparation de véhicules à travers un site web.

Dans le cadre de la mise en place de la partie Back-end de ce projet, j'ai choisi d'utiliser des technologies telles que PHP, MySQL, ainsi que les environnements de développement Wamp et XAMPP au début.

Toutefois, en raison des problèmes rencontrés avec Xampp, j'ai décidé de migrer vers une autre solution plus adaptée à mes besoins qui est Wampp.

Une décision importante a été de ne pas recourir à Symfony pour ce projet, préférant opter pour une approche de développement en PHP pur.

On m'a fréquemment conseillé, en tant que débutant, de maîtriser les bases de la programmation en travaillant directement avec du code non abstrait.

Cette approche m'a grandement aidé à mieux appréhender la logique sous-jacente du code. Toutefois, l'absence d'un framework pour accélérer le développement a entraîné une augmentation du temps nécessaire à la réalisation du projet, qui reste partiellement achevé à ce stade.

Aujourd'hui, je suis prêt à vous présenter le résultat de mes efforts de 4 mois de travail.

Pour chaque fonctionnalité de l'appli, une branche sera créée vers Git.

# Table de matière

1.0	Base de données.....	3
1.1	Schéma de la base de données.....	3
1.2	Configuration de la Base de D.....	3
1.3	Importation des Tables depuis MySQL Workbench .....	4
1.4	Ajout Manuel des Contraintes de Clé Étrangère .....	4
1.5	Création de Données Temporaires dans PHPMyAdmin.....	5
1.6	Ajout de Données dans la Table Horaire.....	6
1.7	Création de la table avis.....	6
2.0	Mise en place de l'environnement de travail.....	6
2.1	Création du fichier index.php .....	6
2.2	Création du fichier .htaccess .....	6
2.3	Bloc "try" "catch": Gestion des Exceptions.....	7
2.3.1	Affichage de messages d'erreur conviviaux pour les utilisateurs.....	7
2.3.2	Journalisation des erreurs.....	8
2.3.3	Gestion spécifique des différents types d'exceptions.....	8
2.3.4	Propagation de l'exception.....	8
3.0	Mise en Place du Système de Routage.....	8
3.1	Création du fichier index.php.....	8
3.2	Tests du Système de Routage.....	9
3.3	Personnalisation des URLs.....	10
4.0	Le Rôle du Routeur.....	10
5.0	La Structure MVC.....	10
5.1	Le Contrôleur.....	11
5.2	Le Modèle.....	11
5.3	La Vue.....	11

6.0 Mise en Place du Contrôleur et du modèle Front-End .....	11
7.0 Utilisation de PDO.....	12
8.0 Tests avec de Faux Données.....	13
9.0 Liaison de toutes les champs à la base de données "Garage" .....	14
10.0 Mise en Place des Filtres pour les véhicules.....	14
11.0 Gestion des Problèmes Techniques.....	14
12.0 Développement du Contrôleur Front-End véhicules.....	15
13.0 Espace Administrateur.....	15
13.1.0 : Page Login.....	16
13.1.1 Mise en Place du Système de Connexion.....	16
13.1.2 Création des Champs de Connexion.....	16
13.1.3 Utilisation de password_hash().....	16
13.1.4 Modification des Tables "Employés" et "Admin" .....	16
13.1.5 Tests du Système de Connexion.....	17
13.1.6 Insertion de Données Factices.....	17
13.1.6 Validation des Tests et Conclusions.....	17
13.1.7 Sécurisation et Vérification des Informations de Connexion.....	18
14.0 Tests et affichage des données factices dans la table administrateur.....	19
14.1 Correction des Erreurs de Connexion.....	20
14.2 Tests de Données.....	21
14.3 Mise en Forme des Données.....	21
15.0 Implémentation du Bouton de Suppression.....	21
15.1 Création d'une Nouvelle Route.....	21
15.2 Conversion de l'ID en Entier.....	21
15.3 Mise en Place d'Alertes JavaScript.....	21
15.4 Gestion des Redirections.....	22
16.0 Bouton modifier.....	23
17.0 Bouton création d'un véhicule.....	24
18.0 Ajout d'images lors de la création.....	26

19.0 Suppression d'images.....	27
20.0 Gestion des dates.....	28
21.0 Mise en place du back recherche par filtres.....	29
22.0 API vehicules.php.....	31
23.0 vehicule_model.php.....	32
24.0 Avis Clients.....	34
25.0 Prestation du garage.....	34
26.0 Conclusion.....	35

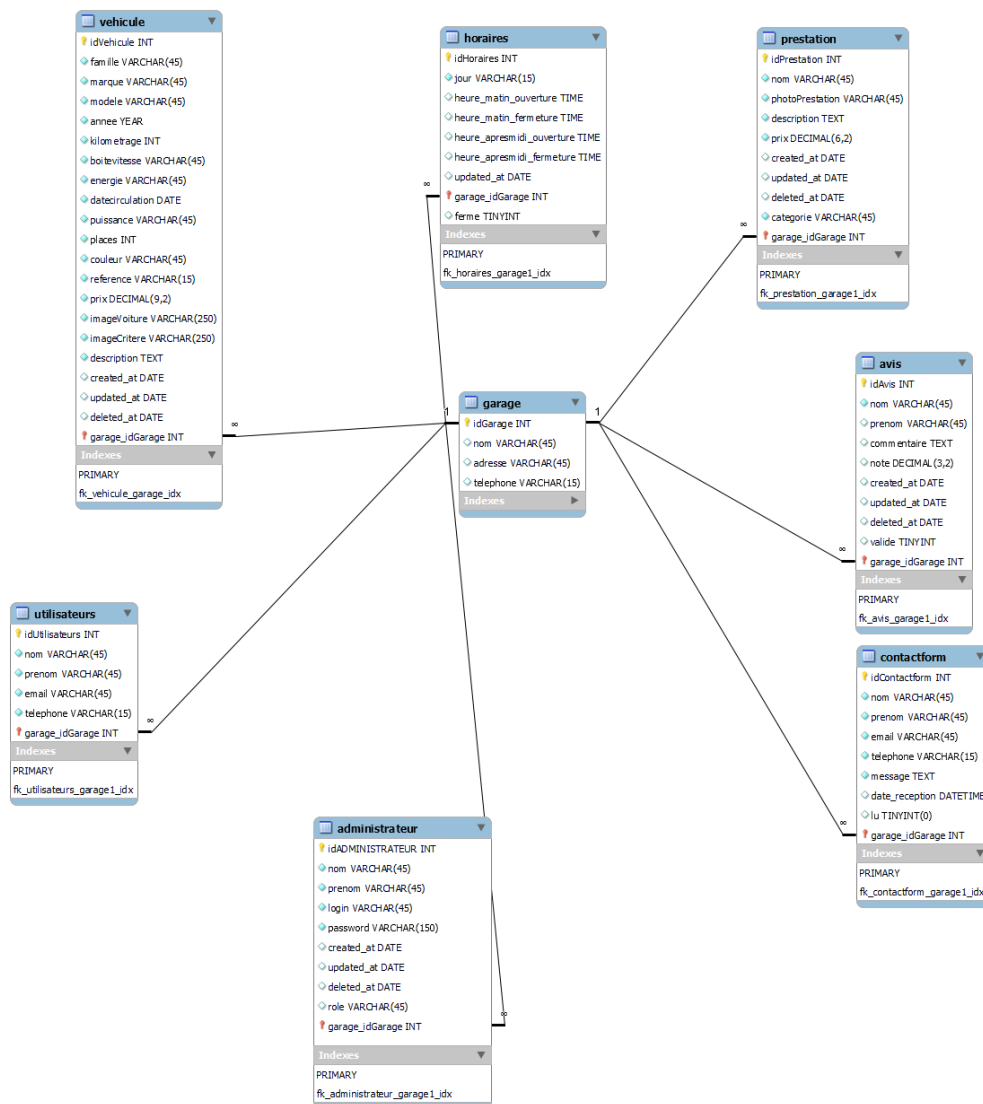
## 1.0 Base de données

# Base De Données

## 1.1 Schéma de la base de données

En accord avec les spécifications que j'ai élaborées, j'ai conçu le schéma de la base de données, un processus qui a connu diverses évolutions au fil de l'avancement du projet.

Il est fréquemment complexe d'anticiper tous les aspects dès le départ, d'où la nécessité d'ajustements progressifs en réponse aux changements et aux besoins qui émergent au cours du développement.



## 1.2 Configuration de la Base de Données

Pour commencer, je vais configurer la base de données et vous citer tous les défis que j'ai pu rencontrer tout au long du processus.

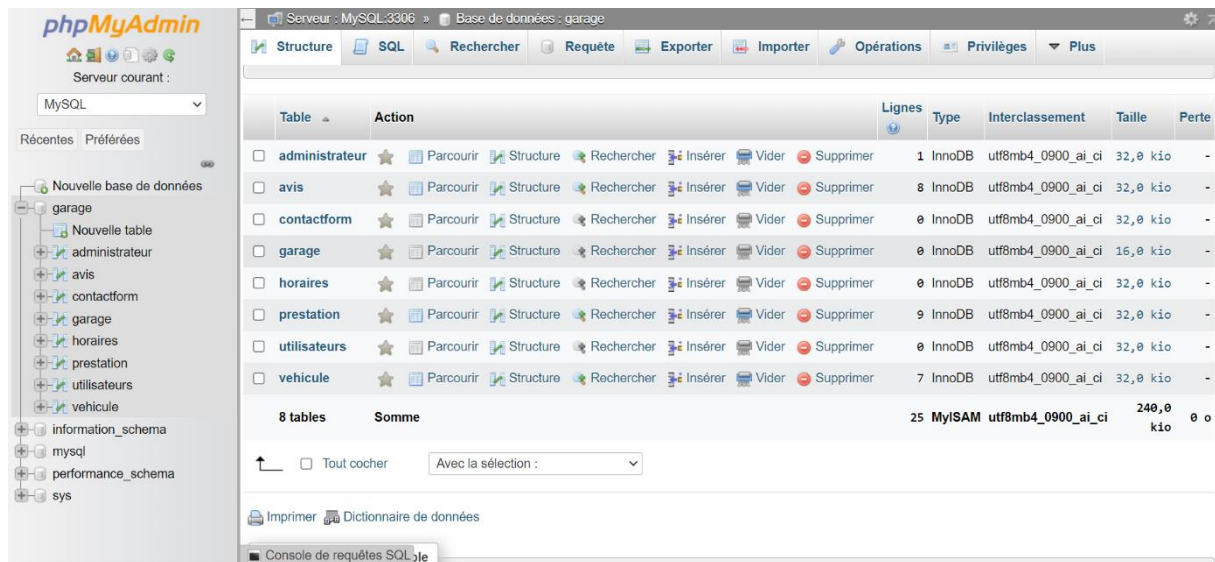
L'objectif principal était de concevoir une base de données fonctionnelle et efficace pour pouvoir stocker les données de mon application.

## 1.3 Importation des Tables depuis MySQL Workbench

Au départ, j'ai utilisé MySQL Workbench pour concevoir le modèle conceptuel de la base de données, y compris les relations entre les tables.

Cependant, lors de l'exportation vers MySQL PHPMyAdmin, j'ai rencontré des problèmes liés aux cardinalités entre les tables.

Malgré mes efforts pour configurer les relations correctement dans MySQL Workbench, les contraintes de clé étrangère n'ont pas été prises en compte lors de l'exportation.




The screenshot shows the phpMyAdmin interface for a MySQL server. The left sidebar displays a tree view of the database structure, including a 'garage' database with several tables. The main panel shows a table structure overview for the 'garage' database. The table list includes: administrateur, avis, contactform, garage, horaires, prestation, utilisateurs, and vehicule. A summary row at the bottom indicates 8 tables with a total of 25 MyISAM lines and a size of 240,0 kio.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
administrateur	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
avis	Parcourir Structure Rechercher Insérer Vider Supprimer	8	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
contactform	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
garage	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_0900_ai_ci	16,0 kio	-
horaires	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
prestation	Parcourir Structure Rechercher Insérer Vider Supprimer	9	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
utilisateurs	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
vehicule	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	utf8mb4_0900_ai_ci	32,0 kio	-
<b>8 tables</b>	<b>Somme</b>	<b>25</b>	<b>MyISAM</b>	<b>utf8mb4_0900_ai_ci</b>	<b>240,0 kio</b>	<b>0 0</b>

## 1.4 Ajout Manuel des Contraintes de Clé Étrangère

Pour résoudre, enfin, je croyais, ce problème, j'ai dû ajouter manuellement les contraintes de clé étrangère pour chaque table à l'aide de commandes SQL.

Cela a permis d'établir les relations souhaitées entre les tables. Voici un exemple de commande que j'ai utilisée pour ajouter une contrainte de clé étrangère :



```
1. ALTER TABLE table_enfant
2. ADD CONSTRAINT fk_nom_contrainte
3. FOREIGN KEY (colonne_etrangere) REFERENCES table_parente(colonne_primaire);
```

Cette solution a normalement permis l'intégrité des données dans ma base de données.

Suite à la résolution initiale de mon problème, j'ai vérifié le modèle conceptuel dans PHPMysqlAdmin et j'ai constaté que les cardinalités n'avaient toujours pas été prises en compte.

Face à cette situation, j'ai dû prendre une décision pour avancer dans le projet. J'ai choisi de passer par le panneau de commande SQL pour ajouter manuellement les contraintes de clé étrangère, table par table.

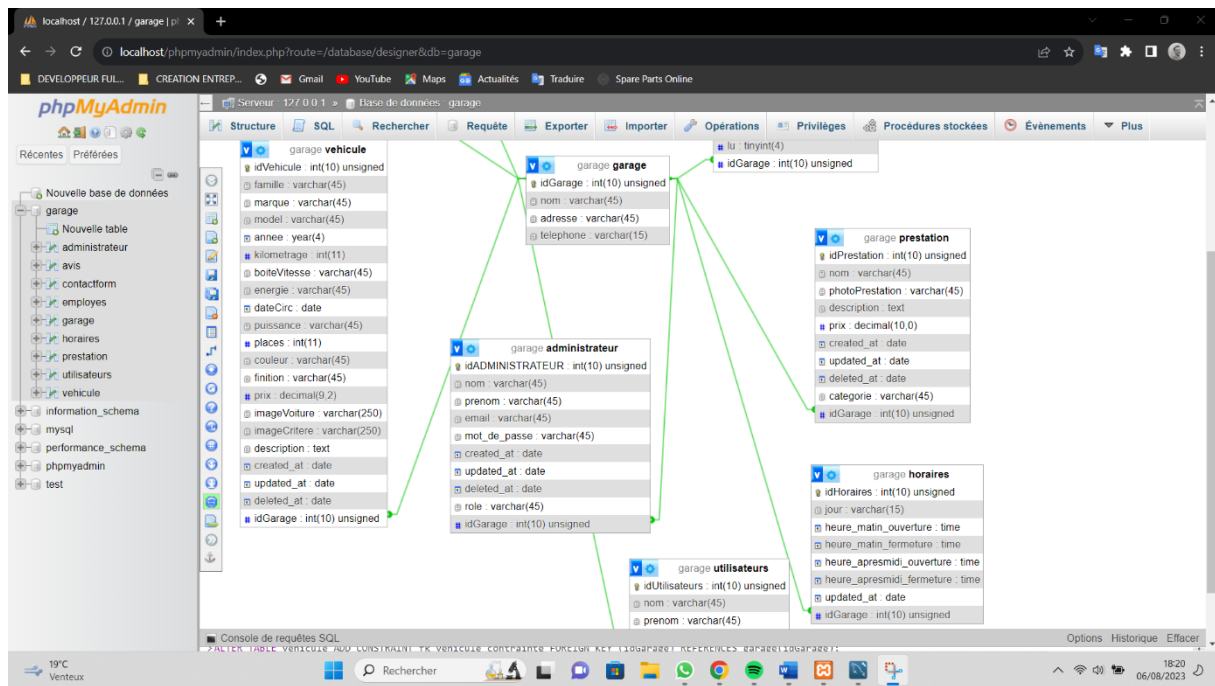
Bien que cette solution ait permis de créer les relations entre les tables conformément à nos besoins, je reste conscient que cela peut être considéré comme une approche moins automatisée et plus sujette aux erreurs.

Cependant, dans le contexte de mon projet et compte tenu du temps limité, c'était la solution la plus pratique pour pouvoir avancer.

Pour l'avenir, je vais continuer à travailler sur l'optimisation de notre base de données et des contraintes de clé étrangère, et je réévaluerai la possibilité de les réactiver une fois que je serai certain que toutes les données sont conformes aux contraintes.

Cette approche garantira l'intégrité des données à long terme tout en me permettant de poursuivre le développement sans interruption.

En fin de compte, cette expérience m'a montré l'importance de la flexibilité et de l'adaptabilité dans le processus de développement, ainsi que la nécessité de prendre des décisions pragmatiques pour faire progresser un projet, même en cas de difficultés imprévues.



## 1.5 Création de Données Temporaires dans PHPMyAdmin

Après avoir résolu les problèmes liés aux contraintes de clé étrangère, j'ai entrepris de saisir de fausses données temporaires dans les tables de ma base de données.

L'objectif était de créer un environnement de test pour vérifier le fonctionnement de mon application.

Cependant, j'ai de nouveau rencontré des erreurs liées aux clés primaires lors de l'insertion de ces données.

Après une analyse plus approfondie, j'ai pu identifier la source du problème.

J'avais mal interprété les cardinalités des relations entre les tables.

J'avais créé manuellement les clés étrangères alors que si j'avais choisi les bonnes cardinalités, les clés auraient été créées automatiquement dans chaque table.

Une fois que j'ai ajusté les cardinalités correctement, j'ai pu exporter les données sans problème.

## 1.6 Ajout de Données dans la Table Horaire

Lorsque j'ai commencé à ajouter des données dans la table "horaire", j'ai réalisé que j'avais initialement considéré uniquement les jours d'ouverture, négligeant les jours de fermeture.

J'ai rapidement rectifié cette omission en ajoutant une colonne supplémentaire appelée "ferme" à la table.

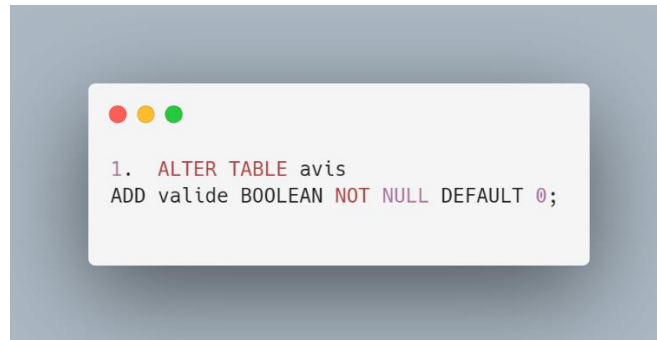
Cette colonne prendra la valeur 0 si l'établissement est ouvert et 1 si c'est fermé, me permettant ainsi de gérer correctement les jours de fermeture.



## 1.7 Création de la table avis

Je vais initialiser dans la table 'avis' un état **valide(1) ou non valide(0)** pour l'espace administrateur sous forme de **booléen**.

Je vais l'initialiser au départ dans la base de données à 0 comme non valide avec une commande Sql :

A screenshot of a terminal window with a light blue background. The terminal shows a SQL command: 

```
1. ALTER TABLE avis
ADD valide BOOLEAN NOT NULL DEFAULT 0;
```

Chaque **Avis** sera donc initialisé à **0** automatiquement comme non valide car après, l'administrateur devra le valider ou pas dans son espace admin après vérification.

## 2.0 Mise en place de l'environnement de travail

# Mise en Place de l'Environnement de Travail

Pour optimiser mon environnement de développement, j'ai réalisé plusieurs étapes clés :

### 2.1 Création du fichier index.php

Ce fichier joue un rôle central dans le modèle MVC (Modèle-Vue-Contrôleur) que j'ai créé.

Toutes les pages qui seront réalisés ultérieurement seront redirigées vers index.php grâce à la méthode GET.

Cela permettra une gestion centralisée des requêtes et des vues.

### 2.2 Création du fichier .htaccess

J'ai également créé un fichier .htaccess pour configurer le serveur Apache de XAMPP.

Ce fichier va mettre en place un système de réécriture des URLs, ce qui rendra les URLs de l'application plus compréhensibles pour les utilisateurs.

Ce fichier est apparemment inaccessible au public, c'est un fichier caché comme on dit.

Ce fichier a permis de configurer le serveur Apache de XAMPP pour la réécriture des URLs, rendant ainsi les URLs plus compréhensibles et lisibles.

Dans l'ensemble, ces étapes m'ont permis de mettre en place un environnement de développement je pense, robuste et de progresser dans la création de l'application.

Je continue à travailler sur l'ajout de données factices dans nos tables pour tester et affiner le fonctionnement de l'application.

## 2.3 Bloc "try" "catch": Gestion des Exceptions

Je vais gérer le bloc "catch" dans le cadre du développement de l'application, car il permet de gérer les exceptions de manière efficace.

### Voici comment ce mécanisme fonctionne :

Dans le bloc "try", j'ai entouré le code susceptible de générer des exceptions, telles que des opérations de base de données ou des appels à des services externes.

L'objectif était de détecter et de capturer toute erreur ou exception qui pourrait être générée lors de l'exécution de ces opérations potentiellement risquées.

```
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $username,
    $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Erreur de connexion à la base de données: " . $e->getMessage());
}
```

Le bloc "catch" est l'endroit où je défini comment gérer précisément chaque type d'exception qui a été lancé dans le bloc "try".

Cette personnalisation me permet de répondre de manière adaptée à chaque situation.

Voici quelques exemples de ce que j'ai pu faire dans le bloc "catch" en fonction de nos besoins spécifiques :

### 2.3.1 Affichage de messages d'erreur conviviaux pour les utilisateurs

J'ai pu capturer l'exception, extraire les informations pertinentes et afficher un message d'erreur clair et compréhensible pour les utilisateurs.

Cela améliore l'expérience utilisateur en les aidant à comprendre ce qui s'est mal passé.

### 2.3.2 Journalisation des erreurs

Pour un débogage ultérieur, j'ai pu enregistrer les détails de l'exception dans un fichier journal.

Cette journalisation permet de suivre les erreurs qui se produisent en production, ce qui est essentiel pour diagnostiquer et résoudre les problèmes.

### 2.3.3 Gestion spécifique des différents types d'exceptions

En fonction de la nature de l'exception (par exemple, une erreur de base de données ou une erreur de fichier), j'ai pu avoir plusieurs blocs "catch" spécifiques pour chaque type d'exception. Chacun de ces blocs peut gérer l'exception de manière appropriée en fonction du contexte.

### 2.3.4 Propagation de l'exception

Dans certains cas, j'ai pu choisir de propager l'exception vers un niveau supérieur de l'application, où elle peut être traitée de manière plus globale.

Cela peut être utile lorsque nous ne savons pas comment gérer l'exception à l'endroit précis où elle a été lancée.

En utilisant judicieusement les blocs **"try" et "catch"**, j'ai pu rendre l'application plus robuste et résiliente aux erreurs, ce qui sera je pense essentiel pour assurer un bon fonctionnement, fournir une bonne expérience utilisateur et faciliter le débogage en cas de problèmes.

## 3.0 Mise en Place du Système de Routage

Une autre étape a été la mise en place d'un système de routage pour organiser les URL de l'application.

Mon objectif était de distinguer clairement la partie **"front"** pour l'espace administrateur de la partie **"back"** qui va faire tourner mon application, tout en permettant une gestion flexible des pages.

Je souhaitais que l'URL contienne deux informations après le "/", ce qui permettait une meilleure organisation du système de routage. Par exemple, notre URL ressemblait à ceci :

```
`http://localhost/garageback/back/nom_de_la_page`.
```

Je rajouterais à la fin de l'url quand ça sera nécessaire, l'Id de l'objet sélectionné.

**Pour mettre en place ce système de routage, j'ai suivi les étapes suivantes :**

### 3.1 Création du fichier `index.php` :

J'ai créé un fichier `index.php` qui joue un rôle central dans notre modèle MVC (Modèle-Vue-Contrôleur).

Toutes les pages que j'ai développées ultérieurement ont été redirigées vers `index.php` grâce à la méthode GET.

Ce système de routage a contribué à l'efficacité de l'application en permettant une gestion plus claire des pages front-end et back-end, tout en offrant une expérience utilisateur améliorée.

```

<?php

error_reporting(E_ALL);
ini_set('display_errors', '1');

session_start();

define("URL", str_replace("index.php", "", (isset($_SERVER['HTTPS']) ? "https" : "http") .
"://".$_SERVER[HTTP_HOST].$_SERVER[PHP_SELF]));

define('__ROOT__', dirname(__FILE__));

require_once("controllers/front/vehicule_controller.php");
$apiController = new VehiculeController();

require_once("controllers/back/espacepro_controller.php");
$espacepro_controller = new EspaceproController();

try {
    if (empty($_GET['page'])) {
        throw new Exception("La page n'existe pas"); // Si l'URL est vide ou faussée, on lève une
exception et on affiche une page d'erreur.
    } else {
        $url = explode("/", filter_var($_GET['page'], FILTER_SANITIZE_URL)); // On récupère l'URL et on
la filtre pour pouvoir la mieux sécuriser.
        if (count($url) < 2) {
            throw new Exception("La page n'existe pas");
        }
        switch ($url[0]) {
            case "front":
                switch ($url[1]) {
                    case "voiturefiche":
                        if (count($url) < 3) {
                            throw new Exception("L'identifiant de la voiture est manquant");
                        }
                        $apiController->getCarsByFilters($url[2]);
                        break;

                    default:
                        throw new Exception("La page n'existe pas");
                }
                break;
            case "back":
                switch ($url[1]) {
                    case "login":
                        $admin_controller->getPageLogin();
                        break;
                    case "connexion":
                        $admin_controller->connexion();
                        break;
                    case "espacepro":
                        if (count($url) < 3) {
                            throw new Exception("La page n'existe pas");
                        }
                        switch ($url[2]) {
                            case "visualisationprestation":
                                $espacepro_controller->visualisationprestation();
                                break;
                            case "modificationprestation":
                                $espacepro_controller->modificationprestation();
                                break;
                            case "suppressionprestation":
                                $espacepro_controller->suppressionprestation();
                                break;
                        }
                }
                break;
            default:
                throw new Exception("La page n'existe pas");
        }
    }
} catch (Exception $e) {
    $msg = $e->getMessage();
}
?>

```

### 3.2 Tests du Système de Routage

Après avoir mis en place le système de routage pour distinguer clairement les parties "front" et "back" de mon application, j'ai entrepris de tester ce système pour m'assurer de son bon fonctionnement.

J'ai dû faire plusieurs tests pour m'assurer que les pages étaient correctement acheminées vers les contrôleurs appropriés.

Pour effectuer ces tests, j'ai utilisé la fonction ``echo`` pour afficher des informations sur l'URL et vérifier comment le système de routage réagissait.

Cela m'a permis de diagnostiquer rapidement les éventuels problèmes de routage et de les corriger en conséquence.

### 3.3 Personnalisation des URLs pour les Pages "voituresFiche" et "prestations" et ainsi de suite

Pour les pages **"voituresFiche"** et **"prestations"**, j'ai mis en place une personnalisation supplémentaire de l'URL pour une expérience utilisateur améliorée.

Mon objectif était de permettre l'accès direct aux fiches de voitures et aux informations sur les prestations en utilisant l'ID unique de chaque objet.

Concrètement, j'ai ajouté l'ID unique à l'indice 2 de l'URL.

**Par exemple, l'URL ressemblait à ceci :**

``http://localhost/garageback/back/voituresFiche/123`` ou  
``http://localhost/garageback/back/prestations/456``.

Cette personnalisation des URLs permettait aux utilisateurs d'accéder rapidement aux informations spécifiques qu'ils recherchaient, améliorant ainsi l'expérience de navigation et la convivialité de l'application.

### 4.0 Le Rôle du Routeur

J'ai dû ensuite mettre en un place un système de routage.

Le permet de faire l'association entre les demandes de l'utilisateur et la logique du site. Cela signifie qu'il décidera comment chaque demande HTTP doit être gérée et quel contrôleur doit être appelé pour traiter la demande qui sera faites.

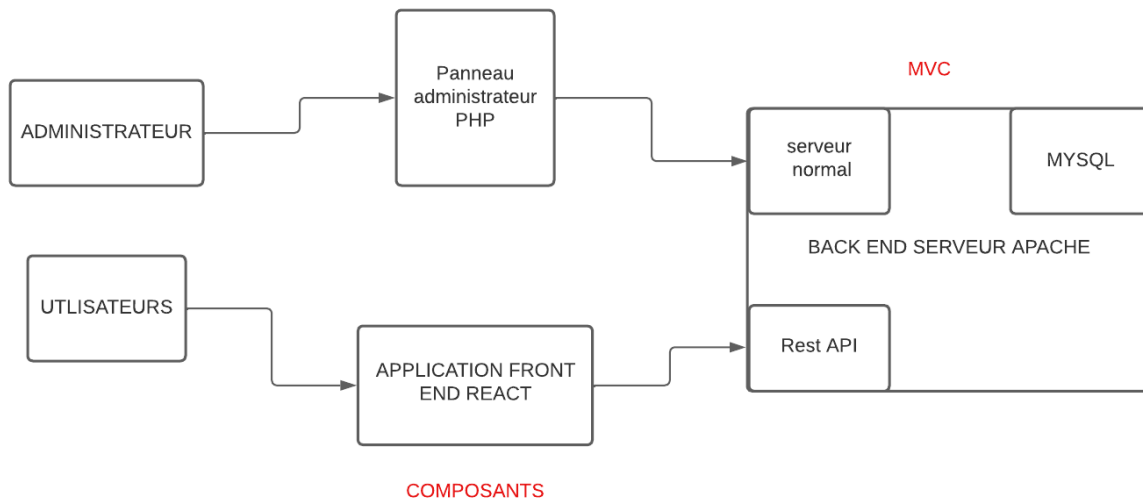
Le routeur permet d'acheminer les utilisateurs vers les bonnes pages en fonction de l'URL demandée.

Il joue un rôle crucial dans la navigation et l'expérience utilisateur.

## 5.0 La Structure MVC (Modèle-Vue-Contrôleur)

J'ai adopté pour l'architecture MVC (Modèle-Vue-Contrôleur) pour organiser ainsi mon code de manière efficace et modulaire.

Cette structure est le moteur dans la séparation des responsabilités et la gestion de l'application.



Voici comment elle fonctionne :

### 5.1 Le Contrôleur

Le contrôleur est le cœur de l'application. Il gère la partie logique et pilote la demande du client.

C'est lui qui décide quels modèles doivent être sollicités pour récupérer les données nécessaires, puis il transmet ces données à la vue.

Le contrôleur coordonne ce qui doit être mis en place pour renvoyer le résultat attendu.

### 5.2 Le Modèle

Le modèle est responsable de la récupération des données demandées par le contrôleur. Il interagit avec la base de données ou d'autres sources de données pour collecter les informations nécessaires.

### 5.3 La Vue

La vue est chargée de l'affichage des pages demandées. Elle utilise les données transmises par le contrôleur pour créer des pages web dynamiques.

Cette architecture MVC a permis une organisation claire et modulaire du code, facilitant ainsi la maintenance de l'application et le travail collaboratif entre les membres de l'équipe.

Ce fût pour moi, l'étape la plus facile de ce projet.





## 8.0 Tests avec de Fauses Données et la Page "voituresFiche"

Pour effectuer ces tests, j'ai utilisé des données factices que j'avais préalablement créées dans la base de données. Cette approche m'a permis de simuler un environnement réel et de vérifier si tout fonctionnait comme prévu.

Lors de ces tests, j'ai spécifié un ID dans l'URL, ce qui a permis d'identifier la voiture spécifique que je souhaitais afficher. Les données de cette voiture ont été récupérées à partir de la base de données et renvoyées sous forme de tableau.

Je vais par la suite formater ces données au format JSON pour une meilleure manipulation et un affichage dynamique sur la page. Cette étape va me permettre de garantir que l'application fonctionne correctement et renvoie les informations attendues aux utilisateurs.

```
Array
(
  [0] => Array
  (
    [idVehicule] => 1
    [famille] => Berline
    [marque] => Peugeot
    [model] => 208
    [annee] => 2022
    [kilometrage] => 23000
    [boiteVitesse] => Automatique
    [energie] => Essence
    [dateCirc] => 2023-08-21
    [puissance] => 5
    [places] => 5
    [couleur] => Rouge
    [reference] => 2023-lk-12
    [prix] => 12000.00
    [imageVoiture] => peugeot.webp
    [imageCritere] => critereA.webp
    [description] => 1 ligne insérée.
    Identifiant de la ligne insérée : 1
    Warning: #1265 Data truncated for column 'prix' at row 1
    Warning: #1366 Incorrect integer value: " for column `garage`.`vehicule`.`garage_idGarage` at row 1
    [created_at] =>
    [updated_at] =>
    [deleted_at] =>
    [garage_idGarage] => 1
  )
)
```

## 9.0 Liaison de toutes les champs à la base de données "Garage"

J'ai relié les champs à la table **Garage** en utilisant des clés étrangères.

Initialement, j'ai rencontré un problème où il ne semblait pas être capable de localiser la table "**Garage**" avec son ID.

Après une analyse plus approfondie, j'ai découvert que le problème résidait dans les cardinalités que j'avais choisies.

Une fois que j'ai ajusté les cardinalités appropriées, le problème de la jointure a été résolu.

## 10.0 Mise en Place des Filtres pour les véhicules

J'ai développé la mise en place de filtres pour trier et afficher les données de manière sélective. J'ai suivi une approche basée sur le modèle **MVC** en utilisant du code pur comme cité ci-dessus.

Le modèle **MVC** impliquait la connexion à la base de données dans le modèle, la création d'un contrôleur qui renvoyait les données au format **JSON**, et la mise en place de l'API pour gérer les filtres.

J'ai utilisé Postman pour effectuer des tests d'URL et résoudre les problèmes qui me sont présentés. J'ai pu ainsi développer une fonctionnalité de filtrage robuste, enfin, j'espère, pour mon application.

## 11.0 Gestion des Problèmes Techniques

Je me suis confronté à plusieurs problèmes techniques.

L'un d'entre eux concernait **MariaDB** de **XAMPP**, où je ne pouvais plus me connecter et où de nombreuses erreurs s'affichaient dans PHPMyAdmin.

Après avoir cherché une solution, j'ai remarqué que MySQL ne fonctionnait pas correctement dans le gestionnaire de tâches.

J'ai donc pris la décision de réinstaller XAMPP en sauvegardant mes fichiers, ce qui a permis de résoudre ce problème et de rétablir la connexion à la base de données.

## 12.0 Développement du Contrôleur Front-End véhicules

Avec la gestion des données et des filtres en place, j'ai pu me concentrer sur le développement du contrôleur front-end de l'application.

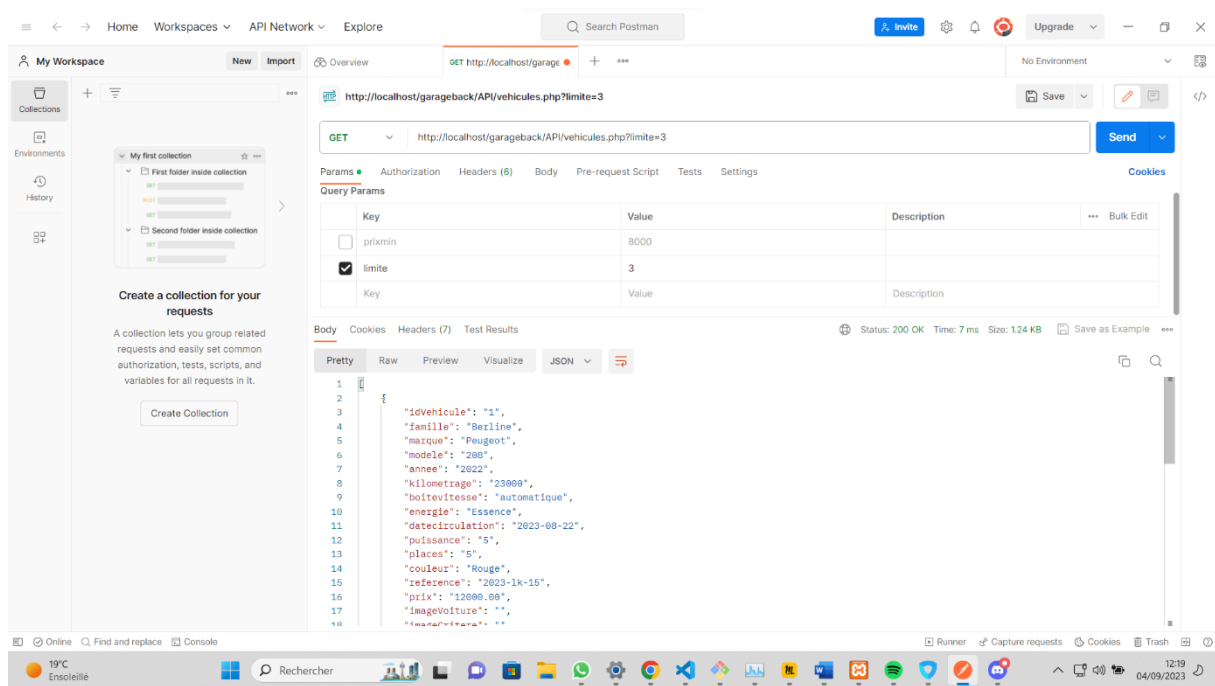
Ce contrôleur a été chargé de manipuler les données des véhicules dans l'application.

J'ai créé un fichier séparé du modèle (**VehiculeModel**) pour accéder aux données et gérer les opérations liées aux véhicules.

En utilisant des tests d'URL et en traitant les requêtes HTTP entrantes, j'ai pu mettre en place différentes actions en fonction de la méthode de la requête et des paramètres fournis dans l'URI.

J'ai également pris en compte la gestion des erreurs (par exemple, une réponse 404 en cas de problème).

Cette phase du projet a permis de préparer le terrain pour le développement du front-end de l'application.



## 13.0 Espace Administrateur

# ESPACE ADMINISTRATEUR

### 13.1.0 : Page Login

## PAGE LOGIN

### 13.1.1 Mise en Place du Système de Connexion pour l'Espace Professionnel

Une étape de mon travail qui a été la mise en place du système de connexion pour l'espace professionnel de l'application.

**Voici les étapes une par une pour créer ce système :**

### 13.1.2 Création des Champs de Connexion

Pour gagner un peu de temps, j'ai utilisé des champs Bootstrap préexistants pour créer les champs de connexion.

Cela m'a permis de concevoir plus rapidement l'interface de connexion administrateur.

### 13.1.3 Utilisation de `password_hash()` pour la Sécurité des Mots de Passe

La sécurité sur internet est primordiale, je dois donc faire mon maximum pour que le site ne soit pas piratable.

Je vais donc utiliser la fonction `password_hash()` pour hacher (crypter) les mots de passe des utilisateurs.

Cette fonction génère un hachage sécurisé qui peut être stocké en toute sécurité dans la base de données.

J'ai opté pour l'option `PASSWORD_DEFAULT`, considérée comme je pense d'après mes recherches, la plus sécurisée à ce jour.

J'aurais qu'à plus ajouter un mot de passe qui sera haché automatiquement que j'insérerais dans la Bdd par la suite.

```
○ ○ ○  
  
public function GetPageLogin() {  
    require_once(__ROOT__ . '\views\login_view.php');  
  
    }  
  
public function connexion() {  
  
    echo password_hash("michelaquiche", PASSWORD_DEFAULT);  
    // echo "connexion";  
    }  
}
```

#### 13.1.4 Modification des Tables "Employés" et "Admin"

En voulant supprimer certaines informations inutiles dans les tables "Employés" et "Admin", notamment l'e-mail que j'avais noté auparavant, je vais pour l'instant, conserver uniquement les champs **"Login"** et **"Password"**.

A ce stade, le login sera généré manuellement, tandis que le mot de passe sera sécurisé par `password_hash()`.

J'ai pensé ensuite, qu'une seule table devrait suffire pour l'espace admin au lieu de créer 2 tables admin et employés, je vais plutôt attribuer des rôles à chacun des administrateurs que je ferai plus tard.

#### 13.1.5 Tests du Système de Connexion

J'ai effectué des tests en appuyant sur le bouton "Valider".

Le système a généré automatiquement un mot de passe sécurisé sur la page de Login, que j'ai pu vérifier en le comparant avec le mot de passe haché stocké dans la base de données.

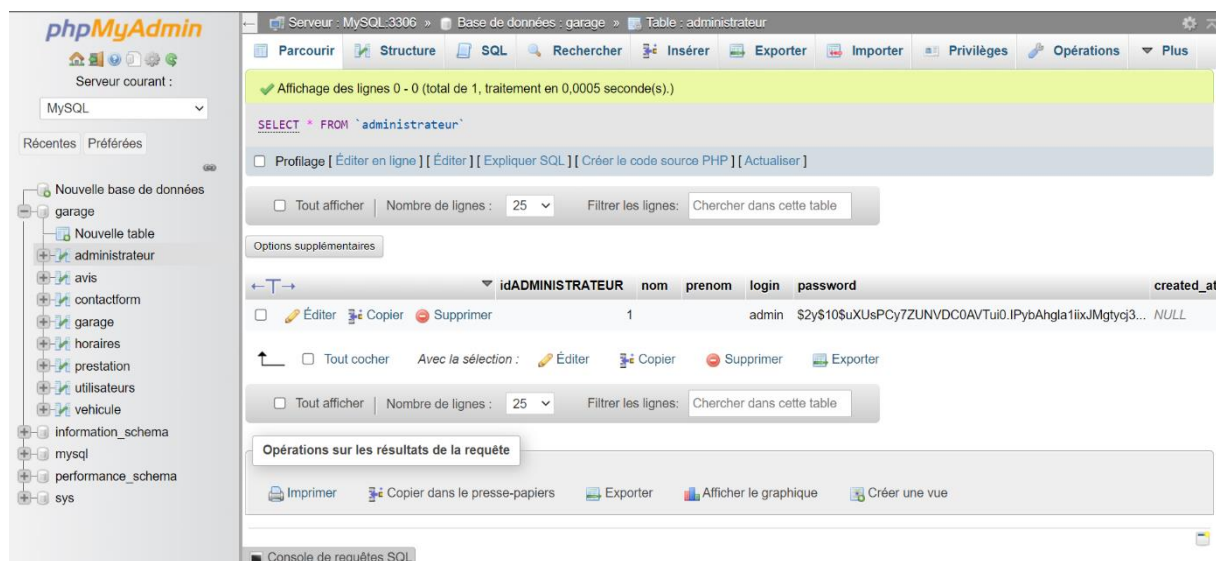
À chaque régénération de la page, un nouveau mot de passe haché sera généré mais mon vrai mot de passe ne changera pas.

Ce mot de passe haché se générera à chaque fois que je cliquerais sur le bouton ce qui ajoutera une sécurité en plus. Dans ces conditions-là, il sera sans doute impossible de le pirater.

### 13.1.6 Insertion de Données Factices

Pour effectuer des tests plus poussés, j'ai inséré de fausses données dans la table, en copiant le mot de passe généré automatiquement et en l'insérant dans le champ "**password**" et admin pour "**Login**" de ma table "**Administrateur**".

La mise en place de ce système de connexion est une étape importante qui garantira la sécurité des données sensibles des administrateurs et ainsi respecter les conditions de RGPD que l'on nous impose sous peine de grosses amendes par la suite.



### 13.1.6 Validation des Tests et Conclusions

En conclusion de cette phase de développement, j'ai réussi à mettre en place un système de gestion de mot de passe sécurisé pour l'espace professionnel de l'application.

Le processus de validation a été concluant, confirmant l'efficacité du hachage de mon mot de passe.

Pour valider le système, j'ai utilisé la fonction `password_hash()` pour hacher un mot de passe spécifique, en l'occurrence "**michelaquiche**".

Le test a été un succès, puisque le mot de passe généré automatiquement correspondait au hachage attendu.

### 13.1.7 Sécurisation et Vérification des Informations de Connexion

```
public function connexion(){
    if (!empty($_POST["login"]) && !empty($_POST["password"])) {
        $login = Securite::secureHtml($_POST["login"]); // securite en lien avec security.class.php
        $password = Securite::secureHtml($_POST["password"]);

        if($this->AdminManager->isConnexionValid($login, $password)) {
            $_SESSION['access'] = "admin"; // Pour activer les variables de session, il va falloir
            que je les active en début de page ds index.php
            header('Location: '.URL."back/admin");
            exit();
        } else {
            header('Location: '.URL."back/login");
            exit();
        }
    }
}
```

La sécurité de l'application est très importante, notamment lorsqu'il s'agit de gérer les informations sensibles dont les mots de passe.

**Voici les étapes que j'ai suivies pour renforcer la sécurité de la gestion des comptes professionnels**

#### Ajustement de la Longueur des Mots de Passe

Lors de l'insertion des données dans la table, j'ai rencontré une erreur liée à la longueur du mot de passe haché.

En effet, j'ai pensé à mettre assez de place pour un mot de passe ordinaire mais je n'avais pas pensé au cryptage qui est largement plus long.

Pour remédier à cela, j'ai ajusté la longueur du champ VARCHAR, en veillant à ce qu'il soit suffisamment long pour accueillir le hachage dans le champ.

J'ai également constaté que l'utilisation de caractères spéciaux dans le mot de passe pouvait entraîner des problèmes, j'ai donc préféré n'utiliser que des caractères alphanumériques pour éviter d'éventuels problèmes même si je sais que je ne devrais pas le laisser en l'état.

Je m'en occuperais sans doute plus tard si le temps me le permet.

## Mise en Place de Vérifications

J'ai créé une page dédiée à la sécurité où je vais vérifier notamment ce qui se passe au niveau des formulaires notamment il vérifiera si les utilisateurs ont bien accès au site. J'ai également converti en HTML les caractères spéciaux pour éviter certains problèmes de sécurité.

J'ai aussi mis en place un système de vérification pour m'assurer que les champs de connexion sont correctement remplis.

J'ai rajouté une sécurité en plus où j'extrait les valeurs soumises pour les champs "login" et "password" que je vais faire passer par une fonction nommée 'secureHtml', elle effectuera des opérations de nettoyage pour éviter les attaques de sécurité, comme l'injection de code malveillant.

J'ai également prévu de créer un lien de déconnexion qui supprimera la variable de **session** lorsque l'utilisateur se déconnectera.

Je rajouterai une fonction dans le manager qui fera des actions de vérifications de connexion en renvoyant 'true' ("Authentification réussie") ou false ("Authentification échouée").

## Gestion des Sessions

Je vais créer dans le controller des variables de sessions et que si les informations de connexion sont correctes (vérifiées en utilisant des identifiants préalablement créés), une session est générée, et l'utilisateur est redirigé vers la page administrative, en n'oubliant pas de le déclarer en début de la page **index.php** pour que les pages puissent par la suite communiquer entre elles.



## Validation des Informations de Connexion

J'ai développé une fonction de vérification des informations de connexion pour m'assurer que l'utilisateur a rempli correctement les champs requis.

Cela garantit également que lors de la déconnexion, l'accès à la page d'administration est désactivé donc quitté la Session.

```

public function connexion(){
    if (!empty($_POST["login"]) && !empty($_POST["password"])) {
        $login = Securite::secureHtml($_POST["login"]); // securite en lien avec security.class.php
        $password = Securite::secureHtml($_POST["password"]);

        if($this->AdminManager->isConnexionValid($login, $password)) {
            $_SESSION['access'] = "admin"; // Pour activer les variables de session, il va falloir
            que je les active en début de page ds index.php
            header('Location: '.URL."back/admin");
            exit();
        } else {
            header('Location: '.URL."back/login");
            exit();
        }
    }
}

```

## Mise en Place de la 'Navbar' Admin

J'ai préparé une navbar pour l'administration, présentant toutes les fonctionnalités autorisées sous forme de menus déroulants.

La navbar est visible uniquement lorsque l'administrateur est connecté.

Pour ce faire, j'ai intégré du code PHP dans le code HTML existant.

## Résolution des Problèmes de Connexion

J'ai rencontré des problèmes de connexion, mais en examinant attentivement le code, j'ai identifié une erreur de syntaxe avec les instructions `require_once`.

Après correction, la connexion a fonctionné correctement.

Ces étapes ont garanti que les informations de connexion soient bien sécurisées et que seuls les utilisateurs autorisés auront accès à la page d'administration

Garrage Parrot - Espace Pro Connexion

## Connexion Espace Pro

Identifiant

Mot De Passe



## 14.0 Tests des données factices dans la table administrateur et Affichage des Données

Voici comment j'ai procédé pour afficher la table "Véhicule" dans l'espace professionnel :

```
class Model {
    private static $pdo;

    private static function setBdd(){
        self::$pdo = new PDO("mysql:host=localhost;dbname=garage;charset=utf8","root","");
        self::$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
    }

    protected function getBdd(){
        if(self::$pdo === null){
            self::setBdd();
        }
        return self::$pdo;
    }

    public static function sendJSON($info){
        header("Access-Control-Allow-Origin: *"); /site sera en ligne
        header("Content-Type: application/json");
        echo json_encode($info);
    }
}
```

### 14.1 Correction des Erreurs de Connexion

Tout d'abord, j'ai résolu les problèmes de connexion qui empêchaient l'accès à la page. Une fois la connexion réussie, les données se sont affichées.

Pour une meilleure lisibilité, je vais faire en sorte de les afficher sous forme de tableau car assez compliqué de s'y retrouver surtout s'il y a des centaines de données à afficher. Je vais utiliser un tableau Bootstrap pour plus de simplicité.

### 14.2 Tests de Données

Pour vérifier que tout fonctionnait correctement, j'ai ajouté des données factices à la table et j'ai effectué des tests en utilisant le contrôleur correspondant.

Cela m'a permis de m'assurer que les données étaient correctement récupérées depuis la base de données.

Ces tests ont été fait pour chaque table.

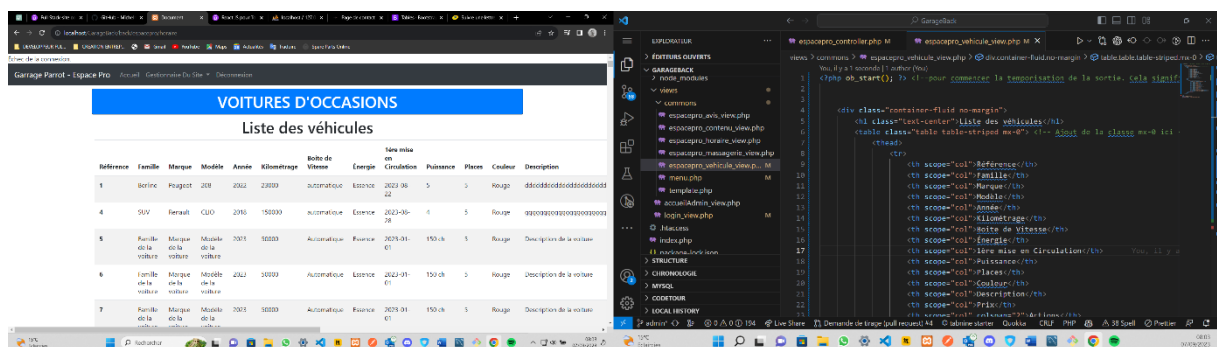
### 14.3 Mise en Forme des Données

Pour améliorer la présentation des données, j'ai utilisé une table Bootstrap prête à l'emploi. Cela m'a permis de gagner du temps tout en offrant une interface utilisateur plus conviviale.

J'ai également préparé des boutons pour les opérations de base et je vais ajouter les CRUD (**Create, Read, Update, Delete**) sous format date actuelle et non américain.

L'affichage des données dans l'espace professionnel va pouvoir fournir aux utilisateurs un accès rapide et efficace aux informations dont ils ont besoin.

Les prochaines étapes consisteront à mettre en place les fonctionnalités de modification et de suppression.



### 15.0 Implémentation du Bouton de Suppression

J'ai créé le bouton de suppression pour garantir la sécurité des données et éviter toute suppression accidentelle. Voici comment j'ai mis en place ce mécanisme dans l'espace professionnel :

#### 15.1 Création d'une Nouvelle Route

Tout d'abord, j'ai créé une nouvelle route pour gérer l'action de suppression. Cette route permettra de transmettre l'identifiant du véhicule à supprimer depuis l'URL.

#### 15.2 Conversion de l'ID en Entier

Étant donné que l'URL transmet l'identifiant en tant que chaîne de caractères, j'ai dû convertir cette chaîne en entier dans le contrôleur correspondant. Cela a permis d'éviter les erreurs et les problèmes de sécurité potentiels.

#### 15.3 Mise en Place d'Alertes JavaScript

J'ai préparé des alertes en JavaScript pour confirmer l'action de suppression.

Ces alertes sont conçues pour n'apparaître qu'une seule fois par page, et j'ai veillé à supprimer la variable de session associée une fois que le message a été affiché.

#### 15.4 Gestion des Redirections

Pendant, j'ai rencontré un problème de redirection après la suppression. Bien que le chemin de redirection soit correct, j'ai dû le commenter temporairement pour résoudre le problème.

La mise en place de ce mécanisme de suppression sécurisé que j'ai effectué servira à protéger les données de l'application et garantir que les opérations de suppression sont effectuées de manière intentionnelle.

## 16.0 Bouton modifier **NON OPERATIONNEL**

Dans la vue, je vais créer des champs de saisie qui vont s'afficher sous la ligne de l'objet à modifier pour chaque attribut de la table "véhicule", permettant ainsi la modification de tous les détails, à l'exception de l'identifiant (ID) bien sûr. Par la suite, un bouton "Valider" sera ajouté.

Ensuite, je vais m'occuper de la partie côté serveur lorsque l'utilisateur appuiera sur le bouton "Valider" par rapport aux nouvelles informations insérées.

### Difficultés rencontrées

À deux reprises, j'ai rencontré des problèmes d'accès au serveur SQL, où l'accès m'a été complètement refusé sans raison apparente, malgré les tests que j'avais effectués.

J'ai dû réinstaller XAMPP encore une fois, mais le problème est que j'ai perdu mes données de table, bien que j'aie sauvegardé le dossier SQL auparavant.

Il doit exister une solution pour restaurer ces données, donc je vais effectuer des recherches à ce sujet.

Par la suite, j'ai appris à les sauvegarder en exportant la Bdd et me le renvoi en **nonDuFichier.sql** mais je devrais par la suite le compresser en fichier ZIP pour pouvoir de nouveau l'importer.

J'ai aussi un problème mineur que je n'ai pas encore trouvé, il faut que je clique 2 fois sur les boutons modifier, supprimé et déconnexion pour que l'opération s'effectue, peut être un problème de rafraichissement de page, à suivre...

**(J'ai trouvé plus tard le problème, il s'agissait d'un inversement de codes qui n'étaient pas à la bonne place).**



Je me suis retrouvé dans une impasse après la réinstallation de XAMPP, avec différents problèmes.

J'ai souhaité générer un nouveau mot de passe pour l'insérer dans la base de données, mais j'ai rencontré des difficultés et des messages d'erreur liés à un ancien code du contrôleur qui n'existait même plus.

J'ai vidé le cache, redémarré XAMPP, isolé tout le reste de mon code, mais rien n'a changé.

En cherchant des solutions, j'ai découvert que je pouvais consulter d'autres messages d'erreur dans un fichier journal (log) de XAMPP, en désactivant d'abord Apache pour éviter de me tromper et ainsi cibler le message d'erreur spécifique.

Cependant, les avertissements (warnings) que j'ai trouvés ne semblaient concerner que le certificat SSL, qui doit être en HTTPS, mais étant donné que je travaille en local, cela ne me concernait pas directement.

Par la suite, mon ordinateur a subi une panne et j'ai dû tout réinstaller, puis récupérer mon travail en utilisant "git clone".

J'ai exporté à nouveau mes tables vers la base de données et généré un nouveau mot de passe que j'ai inséré dans la base de données car je ne savais pas encore comment enregistrer ma Bdd. Cependant, lors de la saisie du nom d'utilisateur (login) et du mot de passe, il était impossible de se connecter.

J'ai mis 6 jours à identifier le problème. Dans le gestionnaire (manager), il était noté "login" et "password", tandis que dans la base de données, c'était écrit "login" et "mot\_de\_passe", d'où l'erreur. C'était comme chercher une aiguille dans une botte de foin.

Après plusieurs bugs de Xampp, je suis passé à Wamp et j'ai cette fois-ci bien enregistré ma Bdd au format Zip au cas où j'aurais d'autres problèmes de fichiers corrompus.

## 17.0 Bouton création d'un véhicule

En ce qui concerne le view, et pour une meilleure expérience utilisateur, j'ai préparé des propositions par rapport aux caractéristiques de la voiture sous forme de dropdown sauf pour la famille des véhicules qui seront en checkboxes pour mettre un peu de piquant, je mettrai une règle pour que l'admin puisse uniquement cocher une seule case pour éviter certaines erreurs.

```
<div class="form-check">
  <input type="radio" id="utilitaire" name="famille" value="Utilitaire" class="form-check-
input"> <label for="utilitaire" class="form-check-label">Utilitaire</label>
</div>
```

J'ai récemment créé une nouvelle route et ajouté un modèle Bootstrap à mon application, en incorporant des cellules de remplissage pour chaque caractéristique du véhicule.

J'ai choisi de ne pas insérer manuellement l'ID lors de l'ajout d'un véhicule, car j'ai configuré la base de données pour qu'elle utilise l'auto incrémentation, ce qui simplifiera le processus.

Pour gérer cette fonctionnalité, j'ai adapté mon gestionnaire (manager) en utilisant une approche similaire à celle que j'avais employé pour la modification.

Cette fois-ci, j'ai élaboré la commande SQL `INSERT INTO` pour l'ajout des données.

Pour obtenir l'ID généré par la base de données, j'ai fait appel à la fonction `lastInsertId()`, qui est une fonctionnalité de l'extension PDO.

```
return $this->getBdd()-  
>lastInsertId();
```

Ensuite, j'ai pris soin de transmettre cet ID au contrôleur responsable de la récupération du nouvel ID.

Pour garantir une expérience utilisateur fluide, j'ai ajouté un message de confirmation à l'intention de l'administrateur, l'informant du nouvel ID du véhicule.

```
$_SESSION['alert'] = [  
    "message" => "Le véhicule a bien été créé sous l'identifiant : " .  
    $idVehicule, "type" => "alert-success"  
];
```

```
public function createVehicule($imageVoiture, $famille, $marque, $modele, $annee,
    $kilometrage, $boitevitesses, $energie, $datecirculation,
    $puissance, $places, $couleur, $description, $prix, $imageCritere, $created_at)
{
    $req = "INSERT INTO vehicule (imageVoiture, famille, marque, modele, annee, kilometrage,
        boitevitesses, energie, datecirculation, puissance, places, couleur, description, prix,
        imageCritere, created_at)
        VALUES (:imageVoiture, :famille, :marque, :modele, :annee, :kilometrage, :boitevitesses,
            :energie, :datecirculation, :puissance, :places, :couleur, :description, :prix, :imageCritere,
            :created_at)";
```

Cependant, j'ai rencontré un problème lors de la saisie des informations dans les champs de formulaire. Les données étaient bien enregistrées dans la base de données, mais au lieu de mes entrées, elles étaient enregistrées sous la forme de '000000'.

La source du problème résidait dans l'utilisation de la fonction ``Securite::secureHTML``, que j'ai dû temporairement désactiver pour résoudre ce problème.

Bien que cette désactivation ait permis de corriger l'anomalie, je suis conscient que cela pourrait potentiellement compromettre la sécurité de mon code.

Je cherche actuellement une solution plus sécurisée pour résoudre ce problème tout en maintenant la protection de mon application."

Malheureusement, même après avoir retiré l'utilisation de cette sécurité, mon bouton de modification continue de ne pas fonctionner.

Cela signifie que la désactivation de cette fonction n'est pas la cause du problème.

Comme la sécurité est importée pour mon application, j'envisage plus tard de trouver des solutions alternatives pour résoudre ce problème tout en préservant la sécurité de mon code.

Ce passage a été marqué par pas mal de défis inattendus et des problèmes techniques mais je pense avoir assez bien géré sur ce coup-là.

# Ajouter un Vehicule

Image Voiture

Choisir un fichier Aucun fichier choisi

Famille

- Utilitaire
- Berline
- Familiale
- Citadine
- SUV

marque

citroen

Modele

## 18.0 Ajout d'images lors de la création

Dans la vue, je vais modifier le type de champ en "file" pour permettre le téléchargement d'image.

Ensuite, je vais créer un nouveau fichier de contrôleur que je nommerai "**regles\_utiles.php**".

C'est dans ce fichier que je vais définir les règles de téléchargement, telles que la taille et le format des images qui devront être respectées, je fais cela pour éviter d'alourdir le site et ainsi le ralentir, ce qui pourrait faire fuir les utilisateurs si la page met trop longtemps à charger.

```

function ajoutImage($file, $dir){
    if(!isset($file['name']) || empty($file['name']))//vérification si l image a bien été saisie
        throw new Exception("Vous devez indiquer une image");

    if(!file_exists($dir)) mkdir($dir,0777);//Si pas de répertoire de crée alors il va en créer un sur
    le serveur

    $extension = strtolower(pathinfo($file['name'],PATHINFO_EXTENSION));
    $random = rand(0,99999);//on va générer un nombre aléatoire pour donner un nom unique au fichier.
    $target_file = $dir.$random."_".$file['name'];

    if(!getimagesize($file["tmp_name"]))//vérifier que le fichier est bien une image
        throw new Exception("Le fichier n'est pas une image");
        //Vérification de la bonne extension
    if($extension !== "jpg" && $extension !== "jpeg" && $extension !== "png" && $extension !== "gif")
        throw new Exception("L'extension du fichier n'est pas reconnu");
    if(file_exists($target_file))
        throw new Exception("Le fichier existe déjà");
    if($file['size'] > 900000)//De préférence, mettre 500000
        throw new Exception("Le fichier est trop gros");
    if(!move_uploaded_file($file['tmp_name'], $target_file))
        throw new Exception("L'ajout de l'image n'a pas fonctionné");
    else return ($random."_".$file['name']);
}

```

Je vais aussi générer un nombre aléatoire pour m'assurer que le nom du fichier téléchargé sera unique.

Je vais également apporter quelques modifications au fichier **"espacepro\_controller.php"** pour lier ces règles que j'ai définies en utilisant des fonctions appropriées.

Je n'oublierai pas de spécifier le répertoire dans lequel je souhaite stocker les images téléchargées.

## 19.0 Suppression d'images :

Lors de la suppression d'un identifiant de véhicule, il est nécessaire que je prenne des mesures pour supprimer les images associées qui sont encore présentes dans le répertoire que j'ai sélectionné.

Pour accomplir cette tâche, je vais ajouter quelques lignes de code supplémentaires à mon contrôleur de suppression dans l'espace professionnel et utiliser la fonction ``unlink()`.

Cela permettra de nettoyer efficacement les fichiers image associés au véhicule supprimé et ainsi alléger le site.





```
public function getImageVoiture($idVehicule){
    $req = "SELECT imageVoiture from vehicule where idVehicule = :idVehicule";
    $stmt = $this->getBdd()->prepare($req);
    $stmt->bindValue(":idVehicule",$idVehicule,PDO::PARAM_INT);
    $stmt->execute();
    $image = $stmt->fetch(PDO::FETCH_ASSOC);
    $stmt->closeCursor();
    return $image['imageVoiture'];
}


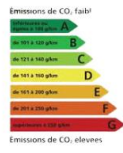

public function getImageCritere($idVehicule){
    $req = "SELECT imageCritere from vehicule where idVehicule = :idVehicule";
    $stmt = $this->getBdd()->prepare($req);
    $stmt->bindValue(":idVehicule",$idVehicule,PDO::PARAM_INT);
    $stmt->execute();
    $image = $stmt->fetch(PDO::FETCH_ASSOC);
    $stmt->closeCursor();
    return $image['imageCritere'];
}
```

## RESULTAT

### 1<sup>ère</sup> partie :

Référence	Image Véhicule	Famille	Marque	Modèle	Année	Kilométrage	Boîte de Vitesse	Énergie	1ère mise en Circulation	Puissanc
128		Berline	citroen	2222	2010	222222	manuel	essence	0000-00-00	5
129		Citadine	citroen	20008	2015	120000	automatique	diesel	0000-00-00	4
130		Berline	citroen	gg	2019	111111	manuel	essence	0000-00-00	5

## 2ème partie :

Puissance	Places	Couleur	Description	Prix	Image Critère	Actions
5	5	blanc	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua	12345.00		<a href="#">Modifier</a> <a href="#">Supprimer</a>
4	6	vert	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua	13500.00		<a href="#">Modifier</a> <a href="#">Supprimer</a>
5	4	bleu	DDDDDDDDDDDDDDDDDDDDDDDDDDDD	12000.00		<a href="#">Modifier</a> <a href="#">Supprimer</a>

## 20.0 Gestion des dates

### J'ai identifié deux problèmes liés à la gestion des dates :

1. Le format de la date (0000-00-00) n'était pas adéquat, j'ai dû le convertir au format européen (00-00-0000). Pour ce faire, j'ai effectué quelques modifications du côté du gestionnaire (manager) afin de garantir la conversion correcte de la date au format souhaité.

2. Une fois que le format de la date était corrigé, je me suis heurté à un deuxième problème : l'incapacité à enregistrer la date dans la base de données car elle sera toujours au format 00-00-000.

La solution était, lors du traitement du formulaire côté serveur, j'ai récupéré la date au **format 'd-m-y'** et je l'ai préalablement formatée en **'y-m-d'** avant de la stocker dans la base de données.

**Problèmes à résoudre ultérieurement : prévisualisation des images en création + Impossibilité de modifier un véhicule**

## 21.0 Mise en place de l'API 'véhicule' par rapport à la recherche par filtres

Je vais utiliser l'architecture du modèle MVC en continuité de la logique de mon code.

### vehicule\_controller.php

Je vais créer le contrôleur qui va gérer les requêtes des filtres pour récupérer les données de la table véhicule à partir du modèle **'vehiculeModel'** .

A ce contrôleur, je vais lui créer une classe **'VehiculeModel'** dans la propriété **'\$apiManager'**.

```
public function __construct() {  
    $this->espaceproManager = new EspaceproManager();  
}
```

J'ai ajouté à cette méthode un constructeur de classe. Je vais utiliser cette méthode pour initialiser chaque objet lorsqu'une instance de cette classe sera créé.

Elle sera appelée automatiquement à chaque fois que je créerais un nouvel objet, ici les filtres, à partir de cette classe.

J'ai ajouté **\$this** qui est un mot clé en Php et qui fait référence à l'objet de lui-même (instance de classe dans laquelle le code est en cours d'exécution).

Il s'agit donc d'un opérateur de sélection de propriété et qui va accéder à chaque propriété de l'objet filtre.

```
public function getCarsByFilters($filtres)
```

Cette instance sera utilisée pour interagir avec **le modèle de données**.

Ensuite, je vais définir une méthode pour récupérer les véhicules en fonction des filtres passés en paramètre et après, je ferais appel au modèle pour récupérer les données.

Pour finaliser le paramétrage du contrôleur, je vais configurer les en-têtes http avec '**header()**' pour spécifier le type de contenu **JSON** et permettre les requêtes en spécifiant les domaines autorisés comme les méthodes acceptées ou les en-têtes.



```
//Configurez les entêtes avant d'envoyer la réponse
header('Content-Type: application/json');
header("Access-Control-Allow-Origin: http://localhost:3000");
header("Access-Control-Allow-Methods: GET, POST, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type");
```

**Header()** est une fonction en Php qui permet de contrôler les en-têtes http comme expliqué ci-dessus pour pouvoir personnaliser la manière dont le serveur communique avec le navigateur et permettre ainsi d'effectuer **des redirections, définir des cookies, spécifier le type de contenu comme ici du JSON** et apparemment encore beaucoup d'autres choses mais que je ne suis pas encore assez expérimenté pour en dire plus.

## 22.0 API vehicules.php

C'est dans ce code, que je vais gérer les requêtes http entrantes en fonction de la méthode qui sera ici en **GET** pour pouvoir récupérer les **paramètres sous format Json** fournis dans l'**Url**.

Je vais vérifier que si la méthode de la requête est GET, je ferais en sorte de continuer à traiter la requête.

Je vais faire vérifier les paramètres que j'ai effectué sur mon système de recherche par filtre et que je stockerais dans un tableau nommé \$filters, s'ils sont bien sûr définis par l'utilisateur dans la requête.

```

    $filters = array();

    if (isset($_GET['famille'])) {
        $filters["famille"] = $_GET['famille'];
    }

    if (isset($_GET['marque'])) {
        $filters["marque"] = $_GET['marque'];
    }

    if (isset($_GET['kilometremin'])) {
        $filters['kilometremin'] =
    }intval($_GET['kilometremin']);

```

Ensuite je vais créer une instance de classe **'VehiculeController'** et je vais appeler la méthode **'getCarsByFilters(\$filters)'** sur cette instance.

Cela me permettra d'interagir entre le contrôleur des véhicules pour récupérer toute leurs données en fonction des filtres choisi par l'utilisateur.

```

    $controller = new VehiculeController();
    $controller->getCarsByFilters($filters);

```

Je vais utiliser la fonction de php **'intval()'** pour pouvoir convertir une valeur en un entier que je stockerais ensuite dans une variable, cela me permettra de m'assurer qu'elle contienne uniquement une valeur numérique en ignorant tout autre caractère non numérique présent dans la chaîne d'origine.

```
if (isset($_GET['kilometremin'])) {  
    $filters['kilometremin'] =  
}intval($_GET['kilometremin']);
```

Cela me permettra de traiter les données provenant d'une source externe pour assurer que la valeur entrée ici, par la recherche par filtre, sera interprétée uniquement comme un entier.

Comme pour le reste de mon projet, en ajoutant une condition, s'il y a une erreur comme une méthode qui n'est pas **GET**, il renverra une **réponse d'erreur 404**.

```
http_response_code(404);  
echo json_encode(["error" => "endpoint not found"]);
```

### 23.0 vehicule\_model.php

Je vais créer la classe '**VehiculeModel**' qui va être utilisée pour effectuer des recherches de véhicules dans la Bdd.

Dans le constructeur de la classe, je vais préparer la connexion à la Bdd en utilisant le **PDO** de Php. J'ai défini les identifiants de connexion pour avoir accès à la Bdd.

Je définie les infos de connexion dans des variables et si une erreur de connexion se passe, j'afficherais un message d'erreur.

```

public function __construct()
{
    $dsn = 'mysql:host=localhost;dbname=garage;charset=utf8';
    $user = 'root';
    $password = '';

    try {
        // Connexion à la base de données
        $this->dbh = new PDO($dsn, $user, $password);
    } catch (PDOException $e) {
        die('Erreur de connexion : ' . $e->getMessage());
    }
}

```

Je vais créer une autre classe `getCarsByFilters` qui sera un **tableau associatif** contenant les filtres pour la recherche de véhicules.

La méthode va construire une requête Sql de base qui va sélectionner toutes les colonnes de la table `vehicule` avec une condition **WHERE 1**, cela signifie que la requête devra toujours être vraie même s'il n'y a pas de filtres spécifiés.

```

$sql = "SELECT * FROM vehicule WHERE 1";

```

En fonction des filtres choisis par l'utilisateur dans le tableau `$filters`, la méthode va ajouter des conditions supplémentaires à la requête Sql.

Je vais donner un exemple, si le filtre `'famille'` est choisi, la méthode ajoutera une clause `'AND famille = :famille'` à la requête et ainsi de suite pour les autres filtres.

La fonction `explode()` va diviser la valeur du filtre en un tableau en utilisant la virgule comme séparateur.

Elles seront ensuite combiné en une chaîne unique séparée par des virgules à l'aide de la fonction `implode()`.

`Str_replace` va supprimer les virgules de chaque valeur et va les transformer en chaîne de caractère et la valeur sera ensuite concaténée.

---

```
if (isset($filters['famille'])) {les AND 1 par 1
    $values = explode(",", $filters['famille']);
    $namedPlaceholders = implode(' ', array_map(function ($value) {
        return ':value_' . str_replace(',', '', $value);
    }, $values));
    $sql .= " AND famille IN ($namedPlaceholders)";
```

J'ai ajouté un filtre limite qui permettra de limiter le nombre de résultats retournés pour éviter que toutes les données soient retournées d'un seul coup et ne serait pas joli visuellement même si je l'ai déjà aussi paramétré du côté React en limitant le nombre d'objet à afficher et aussi en incluant une pagination.

Au départ, j'ai préparé la requête Sql en liant les valeurs des filtres aux paramètres de la requête en utilisant `bindParam()` qui est une méthode de classe de PDO.



```
if (isset($filters['marque'])) {
    $sql .= " AND marque = :marque";
}

if (isset($filters['kilometremin'])) {
    $sql .= " AND kilometrage >= :kilometremin";
}
```

J'avais un doute si je devais utiliser la fonction **bindParam()** ou **bindValue()** car apparemment, ces 2 fonctions sont assez similaires.

Après quelques recherches, j'ai décidé d'utiliser plutôt la fonction **bindParam()** car elle permet de lier une variable par référence, ce qui signifie que toute modifications de filtres apportés à la variable « **filtres** » affectera automatiquement à la requête préparée.

Dans les 2 cas, je pense que cela aurait sans doute fonctionné car le plus important, c'est de bien spécifier le type de données des champs qui sont dans la table de la Bdd pour éviter les certaines erreurs.

Je vais ensuite exécuter la requête en récupérant les résultats sous forme de tableau associatif en utilisant **fetchAll()** qui est aussi une classe de PDO.

Je vais par la suite stocker ces résultats dans une variable et renvoyer la méthode.

## 24.0 Avis Clients

J'ai utilisé le modèle MVC comme référence pour la gestion des avis des clients dans la section réservée aux professionnels.

J'ai mis en place un mécanisme où, par défaut, le champ "valide" dans la table des avis est initialisé à "0", ce qui indique que l'avis est en attente de validation.

Lorsque l'administrateur valide un avis, son état passe automatiquement à 1, ce qui signifie qu'il est validé.

J'ai également ajouté une fonctionnalité supplémentaire sous la forme d'un bouton de validation.

Cela donne à l'administrateur le choix de mettre en ligne un avis ou non. Une fois qu'un avis est validé, le bouton de validation disparaît pour améliorer l'expérience utilisateur dans cet espace.

							Supprimer
5	Müller	Hans	4	Sehr zufrieden	29-10-2023	Validé	Modifier Supprimer
18	Martin	Sophie	5	Service exceptionnel ! Le personnel était très professionnel et ma voiture est comme neuve.	28-10-2023	Validé	Modifier Supprimer
19	Tremblay	Pierre	2	Très déçu de l'expérience. La réparation a pris beaucoup plus de temps que prévu et le coût était élevé.	27-10-2023	Non Validé	Modifier Supprimer Valider
20	Dubois	Marie	4	J'ai été impressionnée par la qualité du service. Ils ont réparé ma voiture rapidement et à un prix raisonnable.	26-10-2023	Non Validé	Modifier Supprimer Valider

## 25.0 Prestation du garage

La seule difficulté était du côté de la création de l'API, cela, m'a fait planter tout mon back end. J'ai dû faire quelques modifications au niveau du MVC. Le problème est résolu.

### Liste des prestations

Référence	Image prestation	Nom	Description	Prix à partir de	Date de création	Date de mise à jour	Actions
4		amortisseurs	Lorem ipsum dolor sit amet. Quo officia corporis aut voluptatem ratione et deserunt rerum. Nam laudantium distinctio id debitis culpa non blanditiis deleniti.	120.00	29-10-2023		Modifier Supprimer
6		embrayage	Lorem ipsum dolor sit amet. Quo officia corporis aut voluptatem ratione et deserunt rerum. Nam laudantium distinctio id debitis culpa non blanditiis deleniti.	200.00	29-10-2023		Modifier Supprimer

## 26.0 Conclusion

Dans la conception de l'interface en HTML, j'ai élaboré un tableau destiné à afficher la liste complète des véhicules.

J'ai aussi implémenté une boucle "**foreach**" pour itérer à travers les données du tableau, permettant ainsi l'affichage détaillé de chaque véhicule.

Ici, je parle de la table véhicule mais dès que j'ai trouvé la solution au départ comment savoir gérer les données, j'ai pu facilement le reproduire sur les autres tables.

De plus, j'ai associé les fonctionnalités de modification et de suppression à des formulaires HTML dédiés.

Ces formulaires fournissent une interface utilisateur intuitive pour gérer les opérations de mise à jour et de suppression des véhicules de manière efficace.

Pour renforcer la sécurité et éviter toute suppression accidentelle, j'ai intégré une demande de confirmation en JavaScript pour le bouton "Supprimer".

Cette mesure préventive assure une expérience utilisateur plus fiable.

J'ai mis en place une condition pour vérifier si un formulaire de modification a été soumis pour un véhicule spécifique.

En cas de soumission, un formulaire de modification est généré pour le véhicule concerné, avec des champs préremplis (inputs) facilitant ainsi la modification des détails mais qui ne fonctionne pas à ce jour.

Je pense que cette approche permet une gestion fluide et sécurisée des véhicules au sein de l'application, tout en offrant une expérience utilisateur optimale, enfin, je l'espère.

