GARAGE PARROT

PARTIE BACK END



**-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.

**-Importation des Tables depuis MySQL Workbench :**

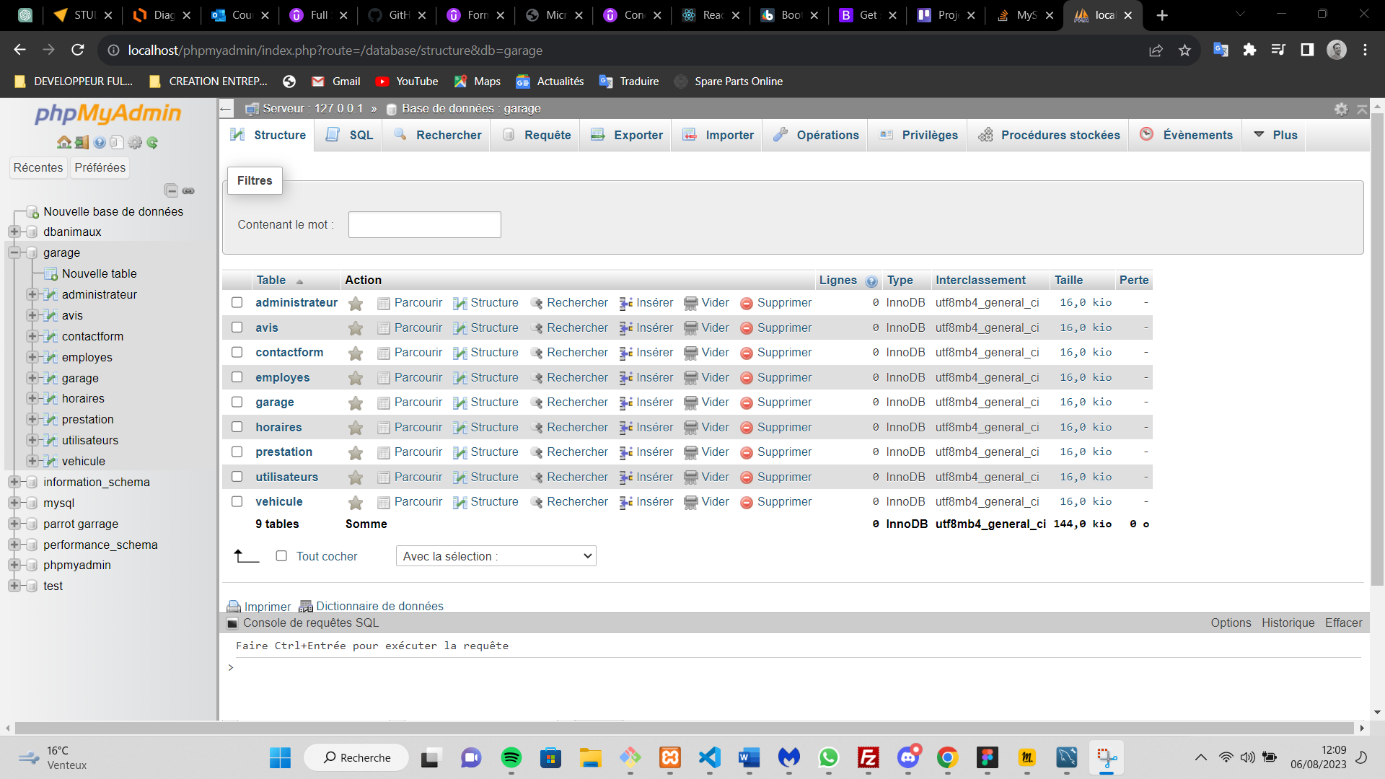
Au départ, j'ai utilisé MySQL Workbench pour concevoir le modèle conceptuel de la base de données, y compris les relations entre les tables. Cependant, lors de l'exportation vers MySQL PHPMyAdmin, j'ai rencontré des problèmes liés aux cardinalités entre les tables. Malgré mes efforts pour configurer les relations correctement dans MySQL Workbench, les contraintes de clé étrangère n'ont pas été prises en compte lors de l'exportation.

**-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);

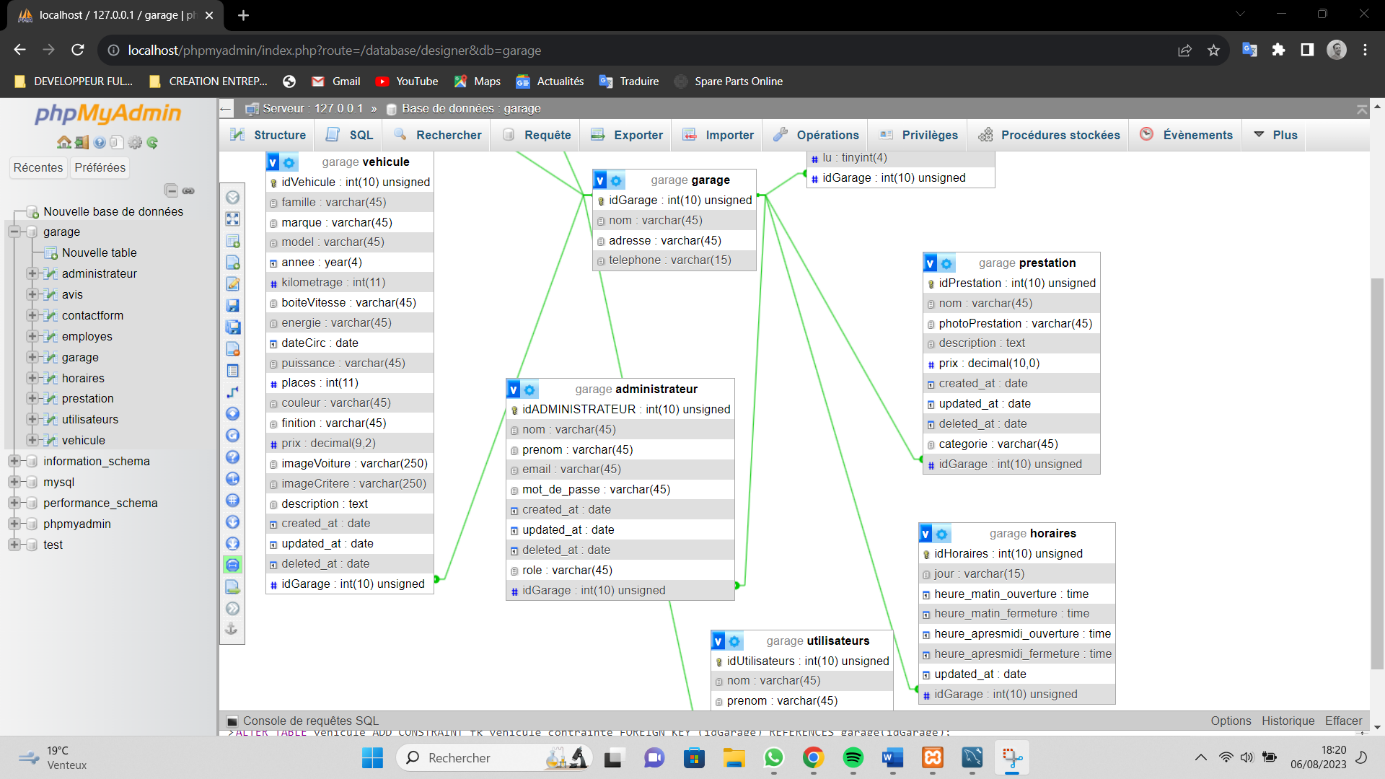
Cette solution a permis de garantir l'intégrité des données dans ma base de données.



Suite à la résolution initiale de mon problème, j'ai vérifié le modèle conceptuel dans PHPMyAdmin 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.

**-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 identifer 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.

**-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.

**Mise en Place de l'Environnement de Travail**

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

**- Création du fichier index.php :** Ce fichier joue un rôle central dans notre modèle MVC (Modèle-Vue-Contrôleur). Toutes les pages que je créerai ultérieurement seront redirigées vers index.php grâce à la méthode GET. Cela permettra une gestion centralisée de nos requêtes et de nos vues.

**- 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.

Dans l'ensemble, ces étapes m'ont permis de mettre en place un environnement de développement robuste et de progresser dans la création de notre application. Je continue à travailler sur l'ajout de données factices dans nos tables pour tester et affiner le fonctionnement de l'application.

-création d’une branche que je vais nommer routage avec Git.

**-Bloc "catch" : Gestion des Exceptions**

Le bloc "catch" est essentiel 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.

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 :

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 considérablement leur expérience en les aidant à comprendre ce qui s'est mal passé.

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.

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.

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 est essentiel pour assurer son bon fonctionnement, fournir une bonne expérience utilisateur et faciliter le débogage en cas de problèmes.

**-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" de la partie "back" de l'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`.

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

**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.

**2.Création du fichier .htaccess** : Pour rendre notre application conviviale et améliorer l'expérience utilisateur, j'ai mis en place un fichier .htaccess. Ce fichier a permis de configurer le serveur Apache de XAMPP pour la réécriture des URLs, rendant ainsi les URLs plus compréhensibles et lisibles.

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

// 1. Création du fichier index.php dans lequel on définit une constante URL.

define("URL", str\_replace("index.php","",(isset($\_SERVER['HTTPS']) ? "https" : "http").

"://$\_SERVER[HTTP\_HOST]$\_SERVER[PHP\_SELF]"));

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(empty($url[0]) || empty($url[1])) throw new Exception ("La page n'existe pas");//On va vérifier que l'URL contient bien 2 paramètres en ajoutant une auytre exeption.Si à l'indice 0 ou 1 de l'url après le / n'existe pas aprés la page front ou back, ds ce cas la je vais lever une erreur.

        switch($url[0]){

            case "front" : //On vérifie la valeur de l'url 0, si elle est égale à front, on va vérifier la valeur de l'url 1.

                switch($url[1]){//On vérifie la valeur de l'url 1, si elle est égale à accueil, on affiche la page accueil.

                    case "accueil": echo "page accueil";

                    break;

                    case "prestations": echo " page prestations".$url[2]." demandées";//On affiche la page prestation et on ajoute à l'indice 2 la prestation demandée avec un Id.

                    break;

                    case "voituresFiltre": echo "page voituresFiltre";

                    break;

                    case "voitureFiche": echo "page voitureFiche".$url[2]." demandées";//On affiche la page voitureFiche et on ajoute à l'indice 2 la voiture selectionnée avec un Id.

                    break;

                    case "contact": echo "page contact";

                    break;

                    case "avis": echo "page avis";

                    break;

                    default : throw new Exception ("La page n'existe pas");

                }

            break;

            case "back" : echo "page back end demandée";

            break;

            default : throw new Exception ("La page n'existe pas");//On léve encore une exeption que s'il n'y a pas front ou back mais autre chose de marqué ds l'url, on affichera un message d'erreur.

        }

    }

} catch (Exception $e){

    $msg = $e->getMessage();

    echo $msg;

}

**-Tests du Système de Routage :**

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

**- Personnalisation des URLs pour les Pages "voituresFiche" et "prestations" :**

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 notre application.

**- Le Rôle du Routeur :**

Un élément central de mon travail lors de mon stage a été la mise en place d'un routeur. Le routeur est un composant essentiel dans une application web, car il permet de faire l'association entre les demandes de l'utilisateur et la logique du site. Cela signifie qu'il décide comment chaque demande HTTP doit être gérée et quel contrôleur doit être appelé pour traiter la demande.

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.

**-La Structure MVC (Modèle-Vue-Contrôleur) :**

Lors de mon stage, j'ai adopté une architecture MVC (Modèle-Vue-Contrôleur) pour organiser notre code de manière efficace et modulaire. Cette structure a été fondamentale dans la séparation des responsabilités et la gestion de l'application.

**Voici comment elle fonctionne :**

**- 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.

**- 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.

**- 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.

**-Mise en Place du Contrôleur Front-End :**

Pour commencer à organiser notre système de routage, j'ai créé un contrôleur du côté front-end. Ce contrôleur a regroupé toutes les routes nécessaires pour gérer les demandes des utilisateurs. Par exemple, une URL comme `http://localhost/garageback/front/accueil` a été associée à une fonction spécifique dans le contrôleur.

En utilisant des fonctions et des tests `echo`, j'ai pu vérifier si le système de routage fonctionnait correctement. Cette étape a été cruciale pour s'assurer que les utilisateurs étaient redirigés vers les bonnes pages en fonction de leurs demandes.

**- Gestion des Données avec le Modèle et l'API :**

L'une des étapes cruciales dans le développement de l'application a été la gestion des données. Pour cela, j'ai mis en place une architecture modèle-vue-contrôleur (MVC) pour séparer clairement les responsabilités. Après avoir créé le contrôleur front-end, la prochaine étape a été de mettre en place le modèle pour gérer les données.

J'ai créé un fichier manager dans la partie front du modèle. Ce manager a été chargé de gérer les données en utilisant une API (Interface de Programmation Applicative). L'API a été un élément clé pour interagir avec la base de données et récupérer les informations nécessaires pour l'application.

**- Utilisation de PDO pour une Liaison Sécurisée avec la Base de Données :**

Pour établir une connexion sécurisée entre notre application et la base de données, j'ai choisi d'utiliser **PDO (PHP Data Objects)**. PDO est une interface de programmation permettant d'interagir avec différentes bases de données de manière sécurisée et portable.

L'utilisation de PDO a permis de prévenir les failles de sécurité potentielles telles que les injections SQL. En utilisant des requêtes préparées, j'ai pu garantir que les données entrantes étaient correctement traitées et échappées, réduisant ainsi les risques liés à la sécurité de l'application.

La liaison sécurisée entre l'application et la base de données a été un élément essentiel pour garantir l'intégrité des données et la confidentialité des informations stockées.

// Création du controlleur du côté front qui va regrouper toutes nos routes

class APIController{

    public function getAccueil(){

        echo "accueil";

    }

    public function getPrestations($idPrestations){//On récupére en paramétre 2 de l'url l'Id

        echo " page prestations".$idPrestations." demandées";

    }

    public function getVoituresfiltre(){

        echo "voiture filtre";

    }

    public function getVoiturefiche($idVoiturefiche){//On récupére en paramétre 2 de l'url l'Id

        echo " page voiturefiltre".$idVoiturefiche." demandées";

    }

    public function getContact(){

        echo "contact";

    }

    public function getAvis(){

        echo "avis";

    }

}

abstract class Model{

    //On va créer une instance à PDO, il y aura qu'une seul instance pour toute la BDD

    private static $pdo;

    private static function setBdd(){//On va se connecter à la BDD

        self::$pdo = new PDO("mysql:host=localhost;dbname=garage;charset=utf8","root","");

        self::$pdo->setAttribute(PDO::ATTR\_ERRMODE,PDO::ERRMODE\_WARNING);//On va afficher les erreurs SQL

    }

}

**-Tests avec de Faux 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.

À l'avenir, nous prévoyons de formater ces données au format JSON pour une meilleure manipulation et un affichage dynamique sur la page. Cette étape a été cruciale pour garantir que notre 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

)

**-Liaison des Tables "Véhicule" et "Prestation" à la Table "Garage" :**

Au cours de mon stage, j'ai entrepris de relier les tables "Véhicule" et "Prestation" 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. Cette étape a été essentielle pour créer des relations significatives entre les données dans la base de données.

**- Mise en Place des Filtres :**

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 brut.

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 se sont présentés. J'ai pu ainsi développer une fonctionnalité de filtrage robuste pour notre application.

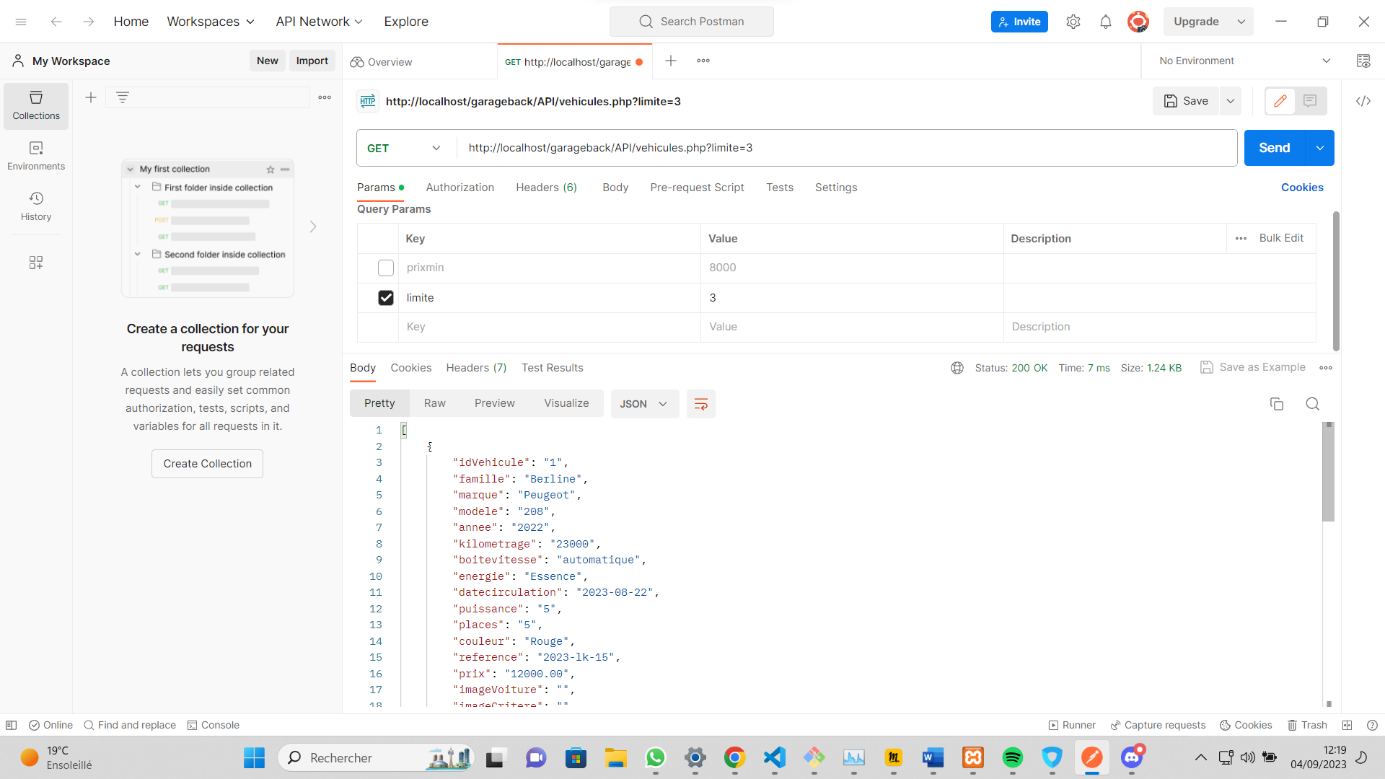
**-Gestion des Problèmes Techniques :**

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.

**-Développement du Contrôleur Front-End :**

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.

****

**ESPACE ADMINISTRATEUR**

**PAGE LOGIN**

**-Mise en Place du Système de Connexion pour l'Espace Professionnel :**

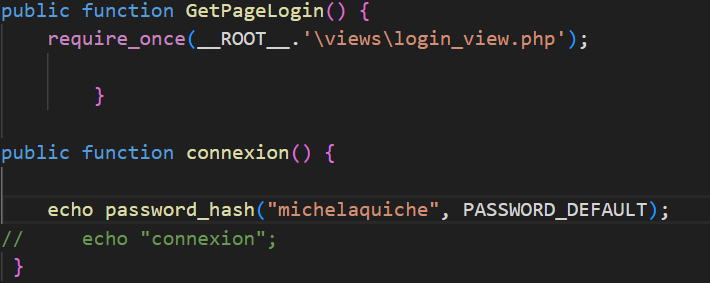
Une étape de mon travail qui a été la mise en place du système de connexion pour l'espace professionnel de l'application. Voici les étapes une par une pour créer ce système :

**- Création des Champs de Connexion :**

Pour gagner un peu 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.

**- 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 la plus sécurisée à ce jour. J’aurais qu’à plus ajouter un

****

**- 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 "Mot de Passe". 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, je vais par la suite plutôt attribuer des rôles à chacun des administrateurs que je ferais plus tard.

**- Tests du Système de Connexion :**

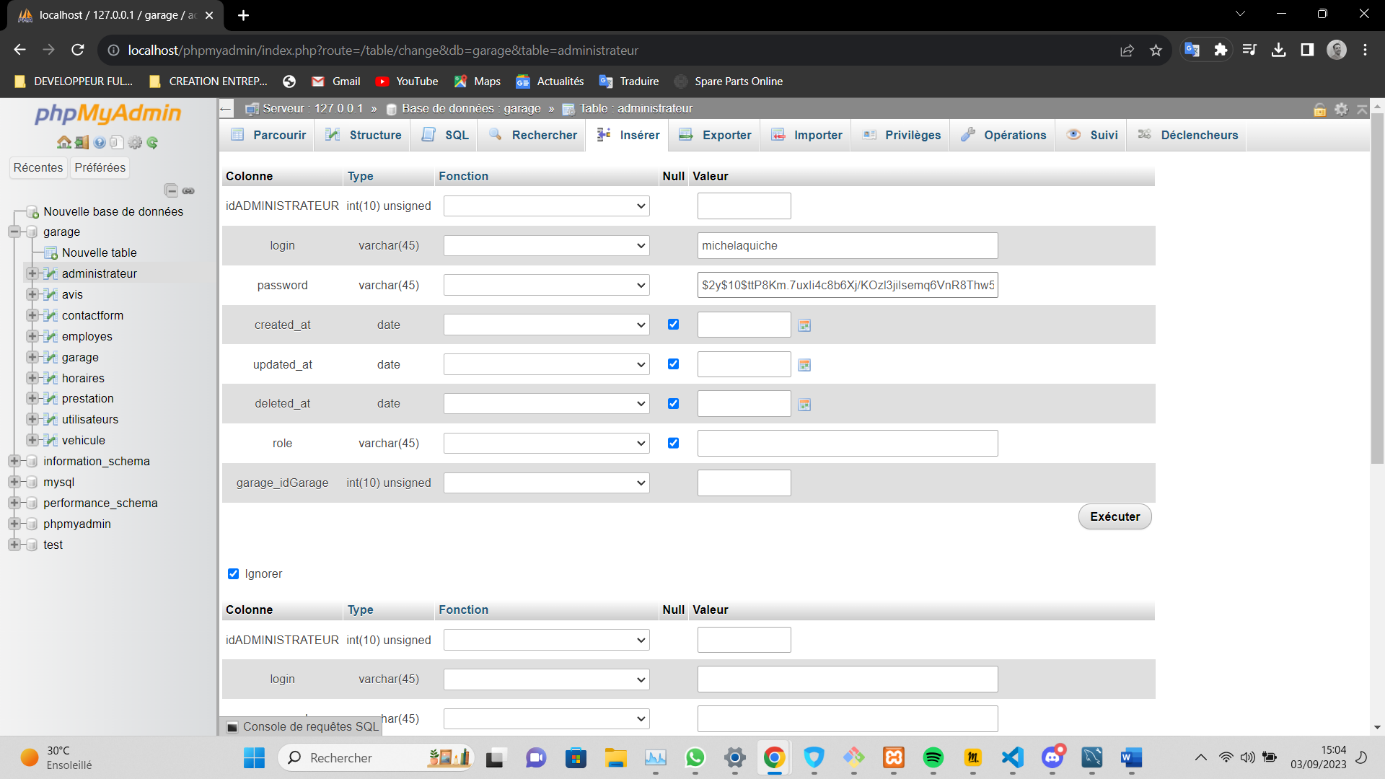
J'ai effectué des tests en appuyant sur le bouton "Valider". Le système a généré automatiquement un mot de passe sécurisé, que j'ai pu vérifier en le comparant avec le mot de passe haché stocké dans la base de données. À chaque régénération de la page, un nouveau mot de passe était généré.

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.

**- 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"** de ma table"**Administrateur**".

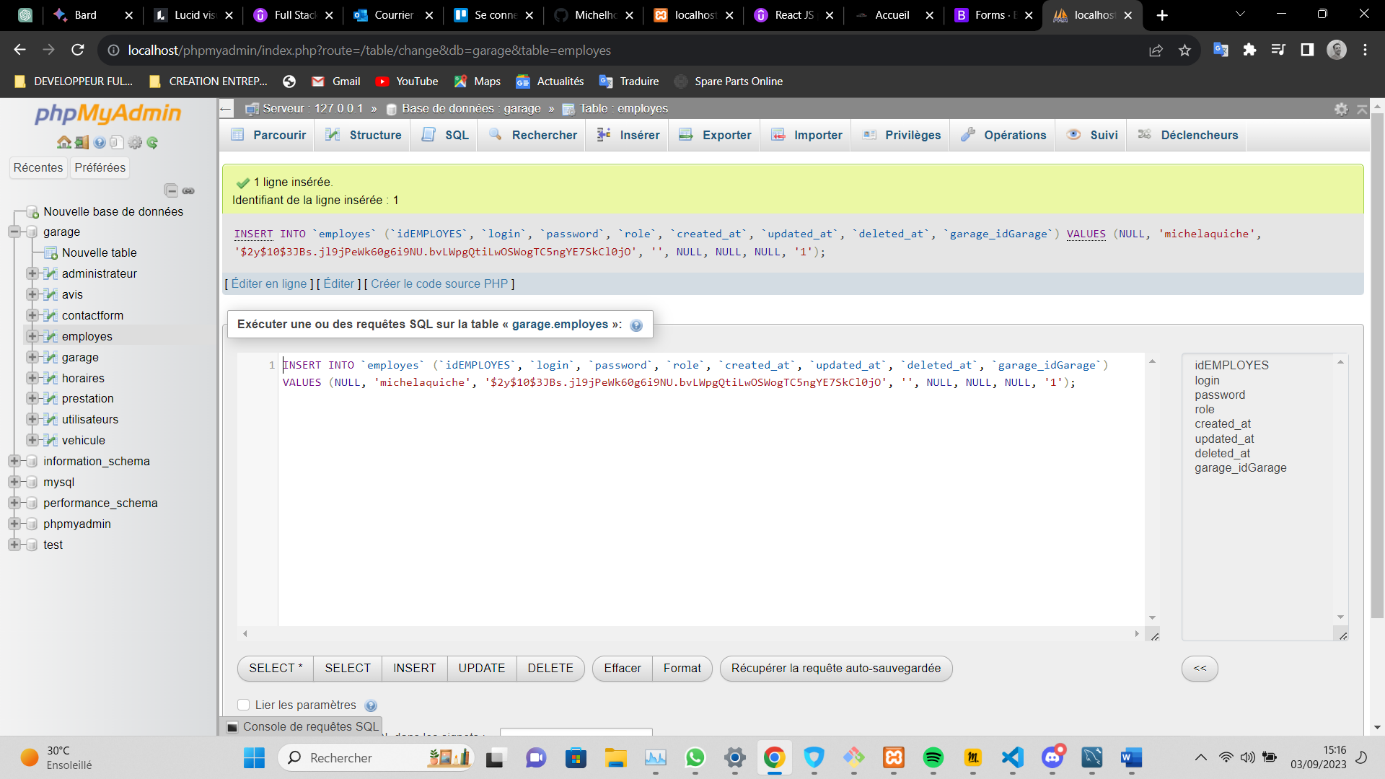
La mise en place de ce système de connexion est une étape importante qui garantira la sécurité des données sensibles des administrateurs.



**-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.

****

**- Sécurisation et Vérification des Informations de Connexion :**

****

La sécurité de l'application est une priorité majeure, notamment lorsqu'il s'agit de gérer les informations de connexion. 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. 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 mais si je sais que je ne devrais pas laissé en l’état. Je m’en occuperais sans doute plus tard si le temps m’en permets.

**- 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 formulaire 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 également mis en place un système de vérification pour s'assurer que les champs de connexion sont correctement remplis. J’ai rajouté une sécurité en plus où j’extrais 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 rajouterais une fonction dans le manager qui fera des actions de vérifications de connexion en renvoyant true (“Authentification réussi“) ou false (“Authentification échouée“).

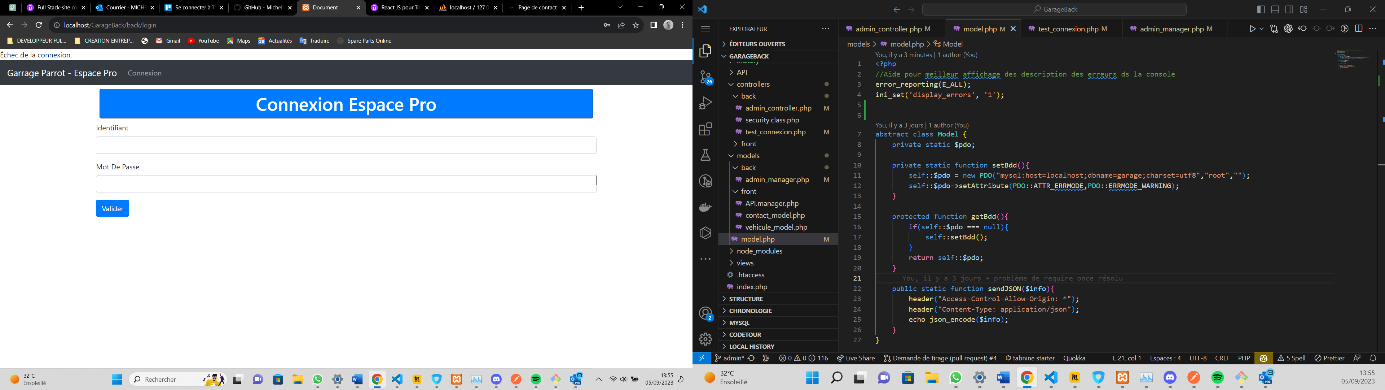
**- Gestion des Sessions :** Je vais créer dans le controller des variables de sessions et que si les informations de connexion sont correctes (vérifiées en utilisant des identifiants préalablement créés), une session est générée, et l'utilisateur est redirigé vers la page administrative, en n’oubliant 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.

**- 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 été essentielles pour garantir que les informations de connexion sont sécurisées et que seuls les utilisateurs autorisés ont accès à la page d'administration.



**-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 :

**- 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.

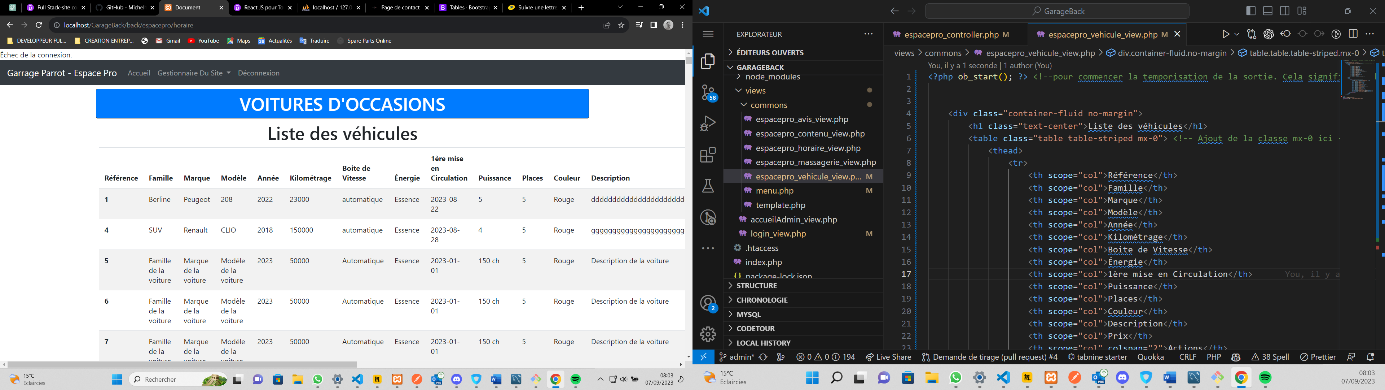
**- Tests de Données :**

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.

**- 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 du CRUD **(Create, Read, Update, Delete).**

L'affichage des données dans l'espace professionnel est une étape essentielle pour 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.



**- Implémentation du Bouton de Suppression**

J’ai crée 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 :

**- 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.

**- 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.

**- 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.

**- Gestion des Redirections :** Cependant, 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.

**-Bouton modifier :**

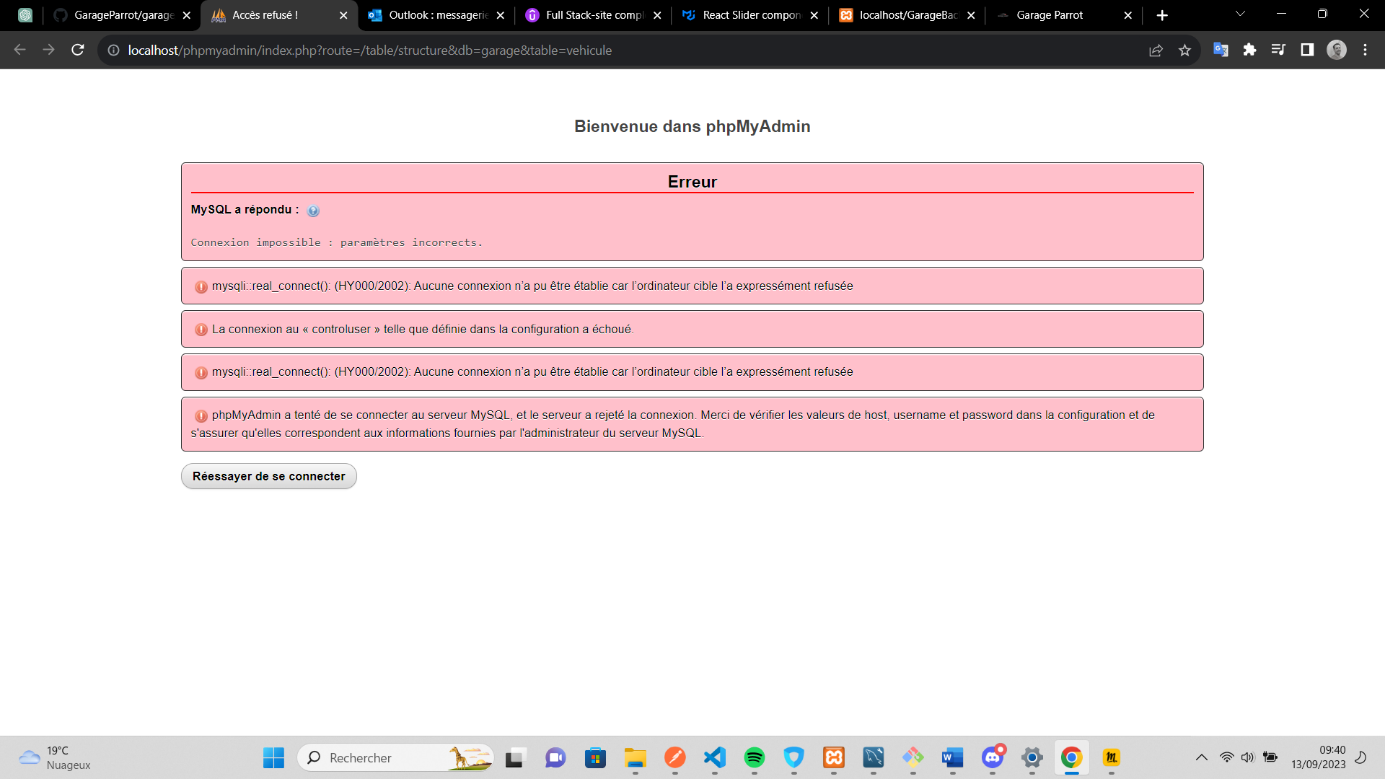
Dans la vue, je vais créer des champs de saisie pour chaque attribut de la table "véhicule", permettant ainsi la modification de tous les détails, à l'exception de l'identifiant (ID) bien sûr. Par la suite, un bouton "Valider" sera ajouté.

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, 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.

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.

****

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 concerne 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. 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.

**-Bouton création d’un véhicule :**

En ce qui concerne le view, et pour une meilleure expérience utilisateur, je vais préparer des propositions de caractéristiques de la voiture sous forme de dropdown sauf pour la famille des véhicules qui seront en checkboxs, je mettrai une règle pour que l’admin puisse uniquement cocher une seule case pour éviter certaines erreurs..

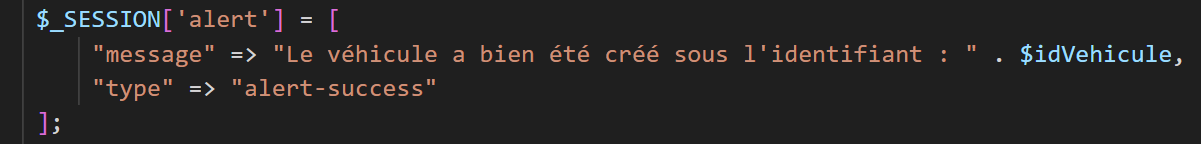
J'ai récemment créé une nouvelle route et ajouté un modèle Bootstrap à mon application, en incorporant des cellules de remplissage pour chaque caractéristique du véhicule. J'ai choisi de ne pas insérer manuellement l'ID lors de l'ajout d'un véhicule, car j'ai configuré la base de données pour qu'elle utilise l'auto-incrémentation, ce qui simplifie le processus.

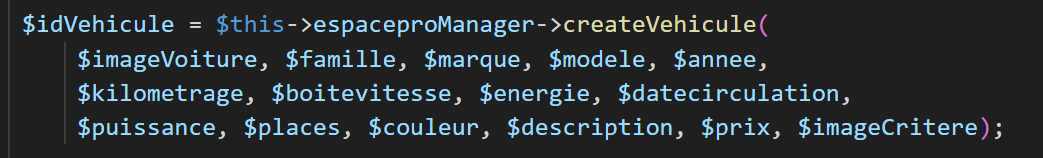
Pour gérer cette fonctionnalité, j'ai adapté mon gestionnaire (manager) en utilisant une approche similaire à celle que j'avais employé pour la modification. Cette fois-ci, j'ai élaboré la commande SQL INSERT INTO pour l'ajout des données.

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.



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.

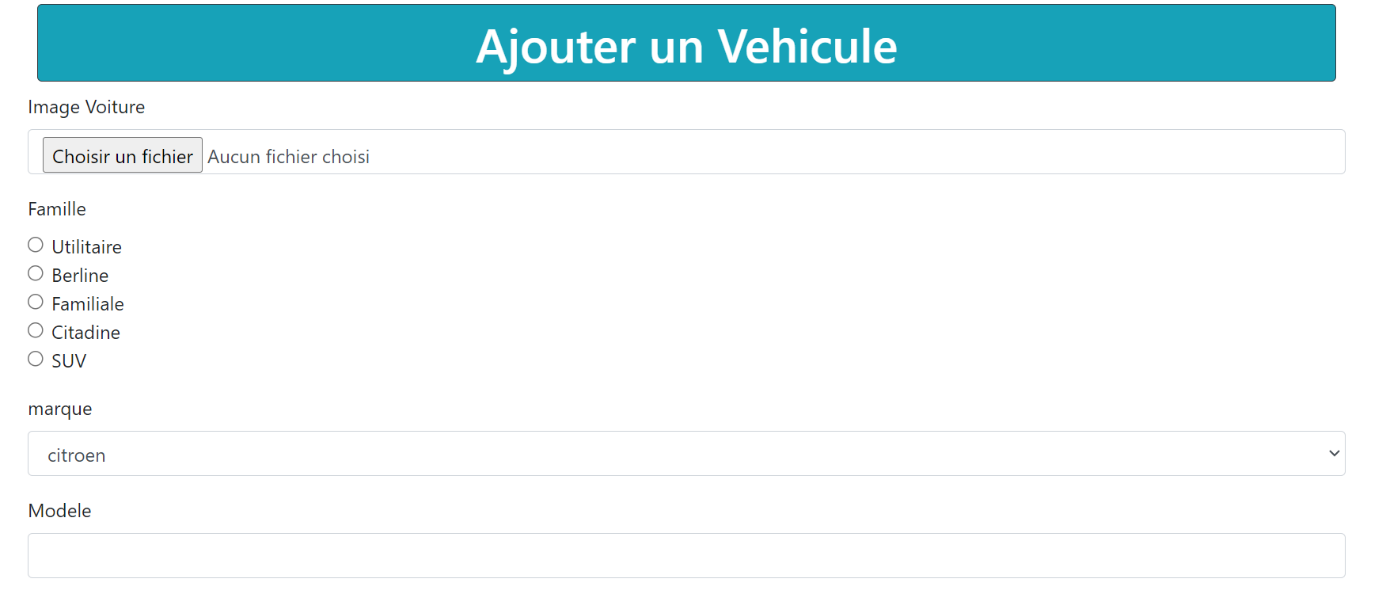




Cependant, j'ai rencontré un problème lors de la saisie des informations dans les champs de formulaire. Les données étaient bien enregistrées dans la base de données, mais au lieu de mes entrées, elles étaient enregistrées sous la forme de '000000'. La source du problème résidait dans l'utilisation de 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 **Securite::secureHTML**, mon bouton de modification continue de ne pas fonctionner. Cela signifie que la désactivation de cette fonction n'est pas la cause du problème. Il est essentiel de maintenir la sécurité de mon application, j'envisage donc des solutions alternatives pour résoudre ce problème tout en préservant la sécurité de mon code

Cette expérience a été marquée par des défis inattendus et des problèmes techniques mais je pense avoir assez bien géré sur ce coup-là.



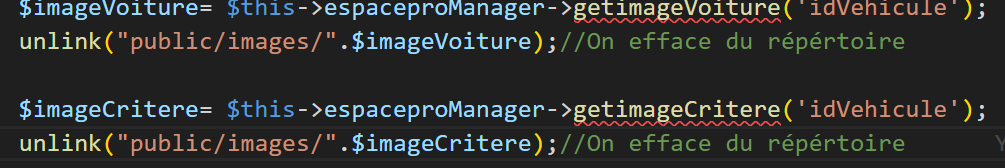
**Ajout d’images lors de la création :**

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

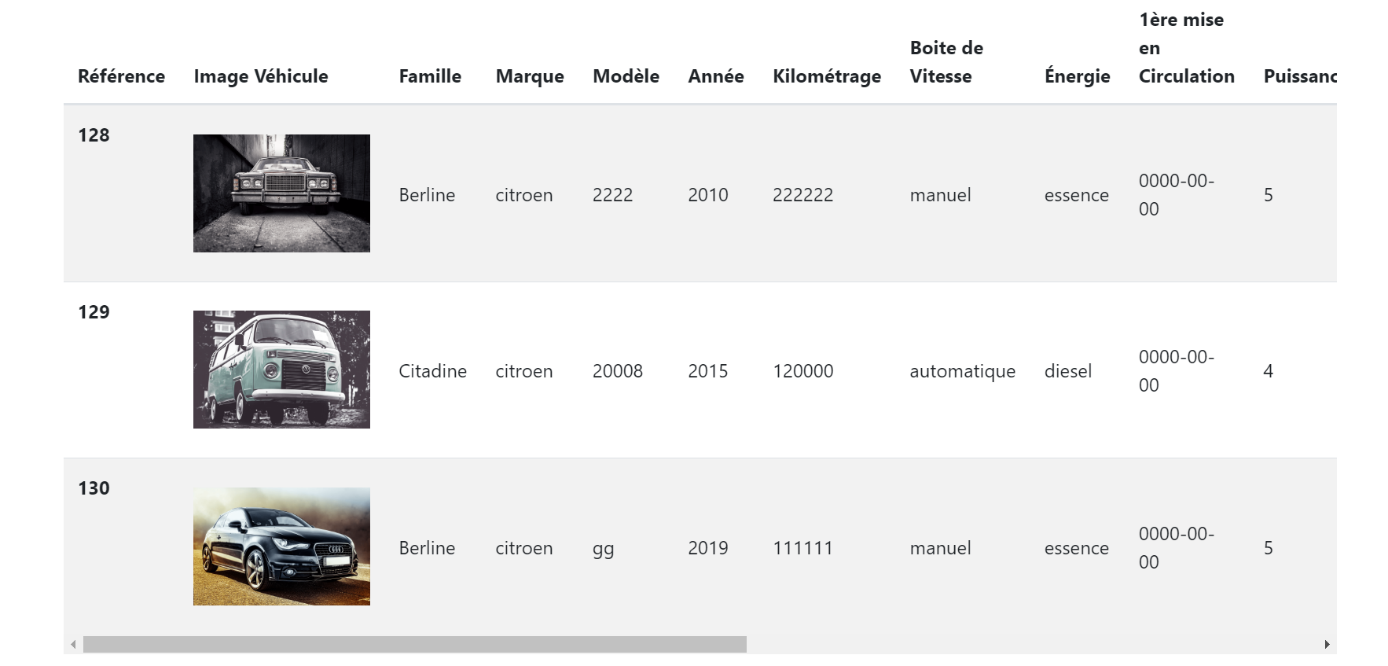
**Suppression d’images :**

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



**RESULTAT**

**1ère partie :**

****

**2ème partie :**

****

**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. 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.

**Conclusion :**

Dans la conception de l'interface en HTML, j'ai élaboré un tableau destiné à afficher la liste complète des véhicules. À cette fin, j'ai implémenté une boucle "foreach" pour itérer à travers les données du tableau, permettant ainsi l'affichage détaillé de chaque véhicule.

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.

En outre, 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 pré-remplis (inputs) facilitant ainsi la modification des détails.

Cette approche globale garantit une gestion fluide et sécurisée des véhicules au sein de l'application, tout en offrant une expérience utilisateur optimale.

**INSERT INTO `vehicule` (`idVehicule`, `famille`, `marque`, `modele`, `annee`, `kilometrage`, `boitevitesse`, `energie`, `datecirculation`, `puissance`, `places`, `couleur`, `reference`, `prix`, `imageVoiture`, `imageCritere`, `description`, `created\_at`, `updated\_at`, `deleted\_at`, `garage\_idGarage`) VALUES (NULL, 'berline', 'peugeot', '208', '2020', '150000', 'manuel', 'diesel', '2020-09-09', '5', '5', 'rouge', 'hhhhhhhhh', '11000.00', '', '', 'hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh', NULL, NULL, NULL, '1');**