SESSION JUIN/JUILLET 2024

DOSSIER PROJET GRADUATE DEVELOPPEUR WEB ET WEB MOBILE

Application Web V.Parrot

CHRISTOPHE MOSTEFAOUI

https://garage-parrot.fr/

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribué au bon déroulement de ma formation.

Je voudrais dans un premier temps remercier l'ensemble de l'équipe pédagogique de chez Studi, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je remercie également tous les formateurs et les intervenants professionnels responsables de ma formation, pour avoir assuré la partie théorique de celle-ci.

Je tiens à témoigner toute ma reconnaissance aux personnes suivantes, pour leur aide dans la réalisation de ce projet de reconversion :

Monsieur Damien Roman, mon directeur d'entreprise m'ayant accordé une période de congés de trois mois afin de me concentrer sur ma formation et qui a été d'un grand soutien dans l'élaboration de mes démarches.

L'équipe de « transition pro » qui a su me faire confiance à travers mes arguments, qui a financé en totalité la formation après étude de mon dossier.

Mme Laurence Courtade, responsable de la société « Clé de soi », qui m'a aidé à réaliser un bilan de compétences afin de confirmer mon envie d'une reconversion professionnelle vers le développement web.

Mon entourage proche pour leur soutien constant et leurs encouragements.

Sommaire

A. Introduction

1.Compétences du référentiel couvertes par le projet

a. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- **1** Maquetter une application
- **2** Réaliser une interface utilisateur web statique et adaptable
- **3** Développer une interface utilisateur web dynamique
- **4** Réaliser une interface utilisateur avec une solution de gestion de contenu

L'application est un site web disponible en local et en ligne. L'affichage des pages est géré par Twig, le moteur de template de Symphony. En utilisant le framework CSS Bootstrap, j'ai veillé à ce que l'interface du site s'adapte aux différentes résolutions d'écran. J'ai utilisé le logiciel Figma pour réaliser le maquettage du site.

b. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- **5** Créer une base de données
- **6** Développer les composants d'accès aux données
- **7** Développer la partie back-end d'une application web ou web mobile
- **8** Élaborer et mettre en œuvre des composants dans une application de gestion de contenu

Les routes et toutes les fonctionnalités de l'application ont été générées et créées en PHP grâce au framework Symphony. Ce dernier intègre Doctrine, un ORM (Object-Relationnal Mapping) qui permet de développer facilement une base de données.

L'objectif principal de ce projet sera de passer au moins une fois par chaque étape clé de conception d'une application web complète : analyse des besoins, maquettage, intégration, développement.

2. Résumé du projet

Mon projet consiste à créer et à déployer une application web vitrine pour le Garage (fictif) V. Parrot, mettant en avant les services proposés par cette entreprise.

Cette application aura pour particularité de pouvoir se connecter soit en tant qu'administrateur soit en tant qu'employé, en cliquant sur le bouton « Connexion pro ».

L'administrateur (le gérant de l'entreprise), une fois connecté, aura droit à tous les accès : il pourra ajouter ou supprimer un compte employé, modifier les heures d'ouverture du garage (présentes dans le pied de chaque page du site), ajouter ou supprimer un service qui sera visible par les utilisateurs du site. Il pourra également gérer les actions auxquelles les employés auront droit.

En effet, un employé, une fois connecté, pourra consulter la liste des messages envoyés par les utilisateurs dans le formulaire de contact ; ajouter ou supprimer un véhicule à la base de données des véhicules d'occasion avec ses caractéristiques ; valider ou supprimer un avis client déposé par un utilisateur, s'il est validé, l'avis apparaitra sur la page "avis clients" et sur la page d'accueil ; ajouter directement un avis client, qui sera visible également sur ces mêmes pages.

Ce projet répondra aux attentes de la formation en mettant en œuvre les compétences apprises (développement de la partie front-end, responsive, ainsi que la partie back-end en intégrant les paramètres de sécurité, créant une base de données ainsi qu'une interface administrateur). Le framework Symfony me paraissait le choix le plus efficace pour gagner du temps en toute simplicité. Ce choix m'a également permis de configurer rapidement la gestion des accès et des rôles de l'application.

3. Environnement technique

Je prépare ce projet en autonomie sur mon ordinateur portable sous MacOS depuis mon bureau personnel sur mon temps libre. En effet, je suis en pleine reconversion professionnelle et dois donc avancer la formation dès que je le peux. Ce projet correspond à l'ECF Studi, il me semble complet pour pouvoir démontrer les compétences acquises durant la formation. Afin de m'organiser au mieux, j'ai utilisé l'application « Trello » pour lister et planifier les différentes tâches à réaliser ce qui m'a permis de ne pas m'éparpiller dans ma mission *(exemple d'une capture d'écran en **annexe 1**)*.

B. Le projet

1. Le cahier des charges

Conception de l'Application

L'interface est conçue pour offrir une expérience utilisateur conviviale et réactive. Le design est axé sur la simplicité et la facilité de navigation. Les pages principales comprennent l'accueil, les services, les avis clients, la page de contact et les véhicules d'occasion. Le responsive permet de naviguer en toute aisance sur le site avec n'importe quel type d'écran.

L'application propose les fonctionnalités suivantes :

En tant qu'Administrateur :

- Connexion sécurisée en tant qu'administrateur.
- Gestion des comptes employés (ajout, suppression).
- Modification des heures d'ouverture du garage.
- Gestion des services (ajout, suppression).
- Attribution des droits aux employés.

En tant qu'Employé :

- Connexion sécurisée en tant qu'employé.
- Consultation des messages envoyés via le formulaire de contact.
- Gestion des véhicules d'occasion (ajout, suppression, caractéristiques).
- Validation et suppression des avis clients.
- Ajout d'avis clients.

Sécurité

La sécurité est une priorité majeure dans le développement de l'application. Les mots de passe sont stockés de manière sécurisée à l'aide de hachages cryptographiques. Les routes sensibles sont protégées par des mécanismes d'authentification et d'autorisation.

Base de Données

La base de données MySQL est utilisée pour stocker les informations relatives aux comptes utilisateurs, aux messages, aux véhicules d'occasion, à la page services et aux avis clients. La structure de la base de données est soigneusement conçue pour permettre une récupération efficace des données.

2. Spécifications techniques

L'application est conçue en utilisant une architecture client-serveur, avec une partie front-end et une partie back-end distinctes.

Les technologies utilisées pour le développement de l'application sont les suivantes :

- **Maquettes/ Illustrations/ Logo/ Images :**

- **Figma** pour réaliser les wireframes et les maquettes
- **Canva** pour créer le logo et les retouches images
- **Envato** pour sélectionner des images illustrant le site (libres de droit d'auteur)
- **Visual Paradigm** pour créer les schémas de modèles de base de données et les diagrammes
- **JMerise** pour la création du MCD

- **Front-end :**

- **HTML5** pour la structure
- **CSS3** pour le style
- **JavaScript** pour les animations et les filtres
- **Bootstrap 5** le framework CSS pour le style et le responsive
- **Twig** le moteur de template de Symphony pour structurer les pages

- **Back-end :**

- **Symphony 7** *La version 5.4 LTS (Long Term Support) de Symfony qui est une version stable qui bénéficie d'un support à long terme aurait pu être utilisée. Elle est importante car elle garantit une stabilité et une sécurité à long terme pour les projets qui l'utilisent. Cependant, il n'a pas été utile de l'utiliser pour ce projet.*
- **PHP 8.2**
- **Composer** pour la gestion de dépendances PHP
- **Doctrine l'ORM** (Object Relational Mapping), intégré à Symfony
- **MySQL** le moteur de base de données
- **PHP MyAdmin** pour stocker la base de données
- **Symfony Local Web Server** pour faire tourner mon projet en local
- **XAMPP** pour le serveur web local

- **Outil de développement :**

- **Visual Studio Code**

- **Outil d'hébergement :**

- **Github**

- **Outil de déploiement :**

- **Hostinger (Voici le lien du site déployé : https://garage-parrot.fr/)**

C. La réalisation du projet (back-end)

Avant de commencer le projet, j'ai commencé par mettre en place une organisation de travail en regroupant toutes les informations issues du pitch, de la veille réalisée en amont du projet et des premiers éléments créés.

- **Benchmark :**

En allant voir sur les sites d'autres garages automobiles, j'ai listé les différentes fonctionnalités proposées et j'ai repéré leur façon de communiquer (ce qui est mis en avant et comment).

- **Personas :**

Après analyse du dossier, j'ai pu déterminer les différents profils type des utilisateurs et j'ai établi des fiches personas avec les rôles et les besoins différents.

- **Wireframes :**

Une fois les fonctionnalités et besoins identifiés, j'ai entamé la création de wireframes afin de mettre en forme les fonctionnalités attendues à travers différentes pages (Accueil, Connexion, Services, Avis clients, Authentification…). ***(Annexe 4)***

- **Charte graphique :**

Pour ce projet, j'ai imaginé une interface simple et attractive. A travers le logo, j'ai voulu mettre des couleurs bleu blanc rouge ce qui m'a permis d'identifier par la suite les couleurs, les icones ainsi que les typographies adéquates***. (Annexes 2 et 3)***

- **Diagrammes :**

J'ai réalisé différents diagrammes pour assurer une bonne conduite au sein du projet : un diagramme de cas d'utilisation, un MCD et un diagramme de séquence. ***(Pages n°9 à n°13)***

- **Trello :**

Pour finir, j'ai regroupé toutes ces informations dans un tableau, ce qui m'a permis de commencer à séparer les différentes tâches à effectuer selon leur degré d'importance et de faire avancer le projet de manière constante. ***(Annexe 1)***

1.Conception et modélisation de la base de données

Dans un premier temps, j'ai commencé par lister toutes les données à stocker. Pour modéliser la base de données, je dois développer un module d'authentification. Une fois l'analyse finie, j'ai pu schématiser celle-ci à l'aide du logiciel Visual Paradigm afin d'établir un **diagramme de cas d'utilisation** pour établir les relations entre les différentes entités.

Le diagramme de cas d'utilisation

C'est le diagramme le plus utilisé lors de l'élaboration de spécifications fonctionnelles, puisqu'il permet de dresser une liste exhaustive et claire des fonctionnalités de l'application. Il va également permettre de définir des dépendances entre les différentes fonctionnalités et de gérer leurs accès selon les différents utilisateurs.

Il est composé de trois éléments principaux : **les acteurs humains, les cas d'utilisation et les relations entre eux.**

Les acteurs humains sont représentés par un bonhomme bâton (stickman) et sont placés sur la gauche du diagramme.

Un cas d'utilisation représente une fonctionnalité de l'application qui va être liée à un acteur. Chaque cas d'utilisation est représenté par une ellipse contenant une courte description de la fonctionnalité.

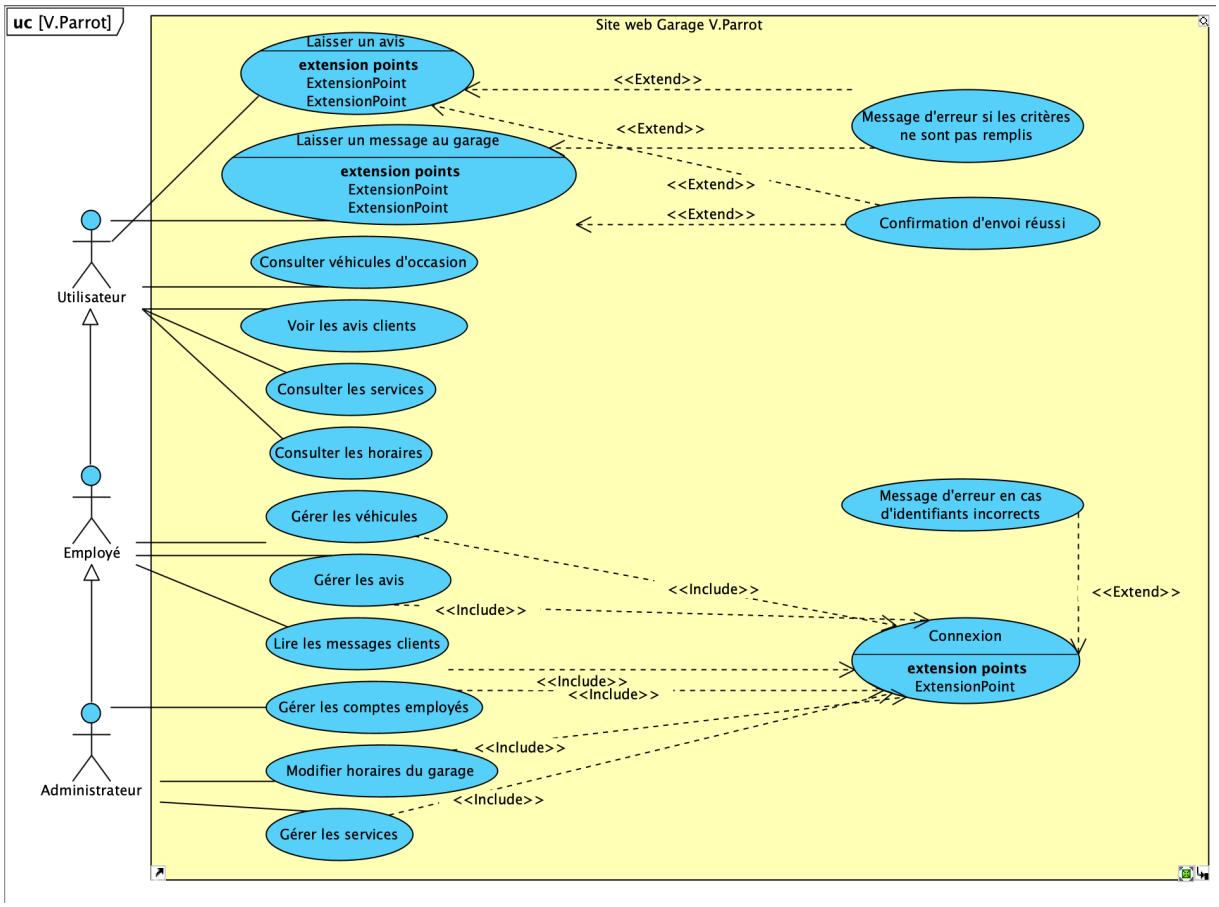
Les cas d'utilisation sont contenus dans un système dont les frontières sont représentées par un rectangle.

En plus de pouvoir relier des acteurs à des fonctionnalités, il est également possible de relier deux fonctionnalités entre elles. Il existe trois types de relations principales : les inclusions, les extensions et les généralisations.

Les **inclusions** permettent de définir une dépendance forte : la fonctionnalité incluse doit être réalisée en amont pour que l'autre puisse être appelée. Cette relation est symbolisée par une flèche pointillée portant le label <<include>>.

Les **extensions**, au contraire des inclusions, permettent de définir une dépendance faible. Si une fonctionnalité en étend une autre, cela signifie qu'elle pourra potentiellement être appelée lorsque l'autre sera exécutée. Cette relation est symbolisée par une flèche pointillée portant le label <<extend>> et par la création d'un point d'extension dans le cas d'utilisation, qui permet d'attribuer un nom à chaque extension. La liste des points d'extension doit être précédée par le titre « extension points ». Chaque relation d'extension peut être accompagnée d'une note indiquant à quel point d'extension elle fait référence, et quelle est la condition d'exécution du cas d'utilisation étendu.

Les labels <<include>> et <<extend>> sont appelés des « stéréotypes ». Leur rôle est d'expliciter la sémantique de la flèche pointillée, qui symbolise une dépendance entre deux fonctionnalités.



Détails du diagramme :

Les acteurs :

Visiteur : toute personne visitant le site sans se connecter.
Employé : un utilisateur avec des droits limités pour effectuer certaines tâches.
Administrateur (ou Patron) : l'utilisateur avec tous les droits pour la gestion du site.

Cas d'utilisation :

Pour le Visiteur :

- Consulter les véhicules : Voir la liste des véhicules disponibles.
- Consulter les services : Voir les différents services offerts par le garage.
- Laisser un message : Utiliser le formulaire de contact pour envoyer un message.
- Laisser un avis : Poster un avis sur le site.
- Voir les avis : Voir les avis des autres clients.
- Consulter les horaires : Voir les horaires d'ouverture du garage.

Pour l'Employé :

- Gérer les véhicules : Ajouter, modifier ou supprimer des véhicules de la base de données.
- Gérer les avis clients : Créer, valider ou supprimer des avis postés par les visiteurs.
- Consulter les messages : laissés par les visiteurs via le formulaire de contact.

Pour l'Administrateur :

- Gérer les comptes employés : Ajouter ou supprimer des comptes employés.
- Modifier les horaires du garage : horaires d'ouverture et de fermeture.
- Gérer les services : Ajouter, modifier ou supprimer des services offerts par le garage.

Relations :
Les acteurs Employé et Administrateur héritent des cas d'utilisation du Visiteur car ils peuvent également effectuer ces actions.
L'Administrateur a un accès supplémentaire à des cas d'utilisation de gestion plus avancés.

Diagramme : Dans le diagramme, on a trois "bulles" (acteurs) : Visiteur, Employé, Administrateur. Chaque acteur est relié par des lignes aux cas d'utilisation appropriés, représentés par des ovales dans la partie droite du diagramme. Les employés et l'administrateur sont reliés aux cas d'utilisation de visiteur pour indiquer l'héritage des fonctionnalités.

Le MCD

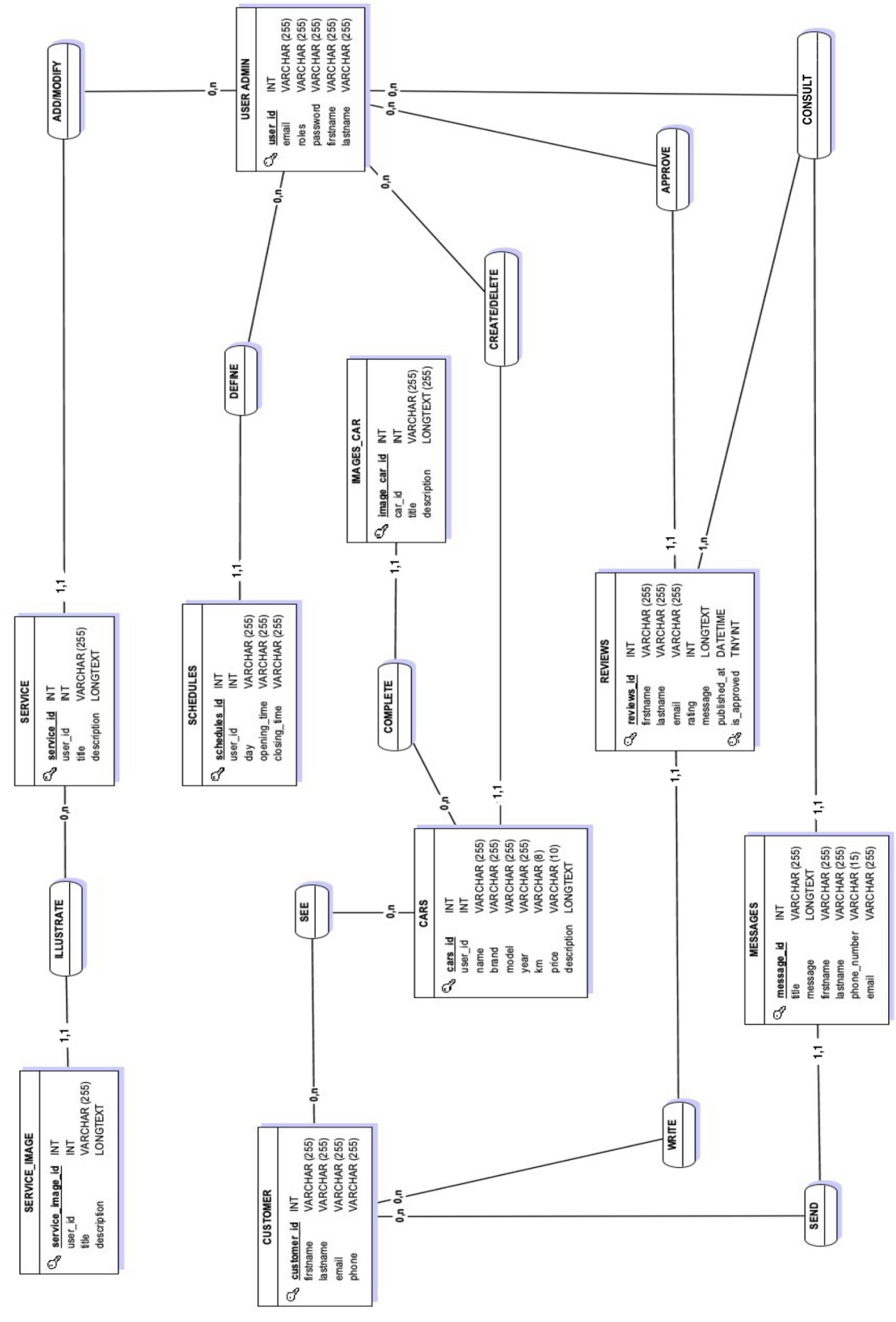
Une fois ce schéma établi, j'ai continué par élaborer un **MCD (Modèle Conceptuel de Données)** avec la méthode Merise pour clarifier l'organisation des différentes fonctionnalités du site en entités et pour décrire les relations entre elles. Cela facilitera la compréhension de la structure du site web.

La construction d'un MCD passe par plusieurs étapes :

- Identification des Entités : Ce sont les objets ou concepts du monde réel pertinents pour le système.
- Définition des Attributs : Ce sont les propriétés ou caractéristiques de chaque entité.
- Détermination des Relations : Il s'agit des liens logiques entre les entités.
- Établissement des Cardinalités : Pour chaque relation, on définit combien d'instances d'une entité peuvent être associées à combien d'instances de l'autre.

Voici quelques exemples pour comprendre les associations et cardinalités entre les tables présentes dans le schéma :

User - Cars :
Un utilisateur peut enregistrer plusieurs véhicules (0,N), mais chaque véhicule est enregistré par un seul utilisateur (1,1).
Cars - CarsImage :
Un véhicule peut avoir plusieurs images (0,N), mais chaque image est associée à un seul véhicule (1,1).
Services - ServiceImage :
Un service peut avoir plusieurs images (0,N), mais chaque image est associée à un seul service (1,1).
Customers - Reviews :
Un client peut laisser plusieurs avis (0,N), mais chaque avis est laissé par un seul client (1,1).
Customers - Messages :
Un client peut envoyer plusieurs messages (0,N), mais chaque message est envoyé par un seul client (1,1).


USER ADMIN
user_id INT
email VARCHAR (255)
roles VARCHAR (255)
password VARCHAR (255)
firstname VARCHAR (255)
lastname VARCHAR (255)
ADD/MODIFY
CONSULT
APPROVE
DEFINE
CREATE/DELETE
SERVICE
service_id INT
user_id INT
title VARCHAR (255)
description LONGTEXT
SCHEDULES
schedules_id INT
user_id INT
day VARCHAR (255)
opening_time VARCHAR (255)
closing_time VARCHAR (255)
IMAGES_CAR
image_car_id INT
car_id INT
title VARCHAR (255)
description LONGTEXT (255)
COMPLETE
REVIEWS
reviews_id INT
firstname VARCHAR (255)
lastname VARCHAR (255)
email VARCHAR (255)
rating INT
message LONGTEXT
published_at DATETIME
is_approved TINYINT
ILLUSTRATE
SEE
CARS
cars_id INT
user_id INT
name VARCHAR (255)
brand VARCHAR (255)
model VARCHAR (255)
year VARCHAR (8)
km VARCHAR (10)
price VARCHAR (10)
description LONGTEXT
MESSAGES
message_id INT
title VARCHAR (255)
message LONGTEXT
firstname VARCHAR (255)
lastname VARCHAR (255)
phone_number VARCHAR (15)
email VARCHAR (255)
SERVICE_IMAGE
service_image_id INT
user_id INT
title VARCHAR (255)
description LONGTEXT
CUSTOMER
customer_id INT
firstname VARCHAR (255)
lastname VARCHAR (255)
email VARCHAR (255)
phone VARCHAR (255)
WRITE
SEND

Le diagramme de séquence

Ensuite, afin de bien décrire comment les différentes fonctionnalités interagissent entre elles pour fournir une expérience utilisateur cohérente, j'ai réalisé le **diagramme de séquence** suivant. C'est une représentation visuelle de l'interaction entre les différents éléments.

Voici une analyse de ce que le diagramme montre :

- ***Visitor*** *:*

Peut remplir et envoyer un message via le formulaire de contact. L'application stocke ensuite le message dans la base de données.

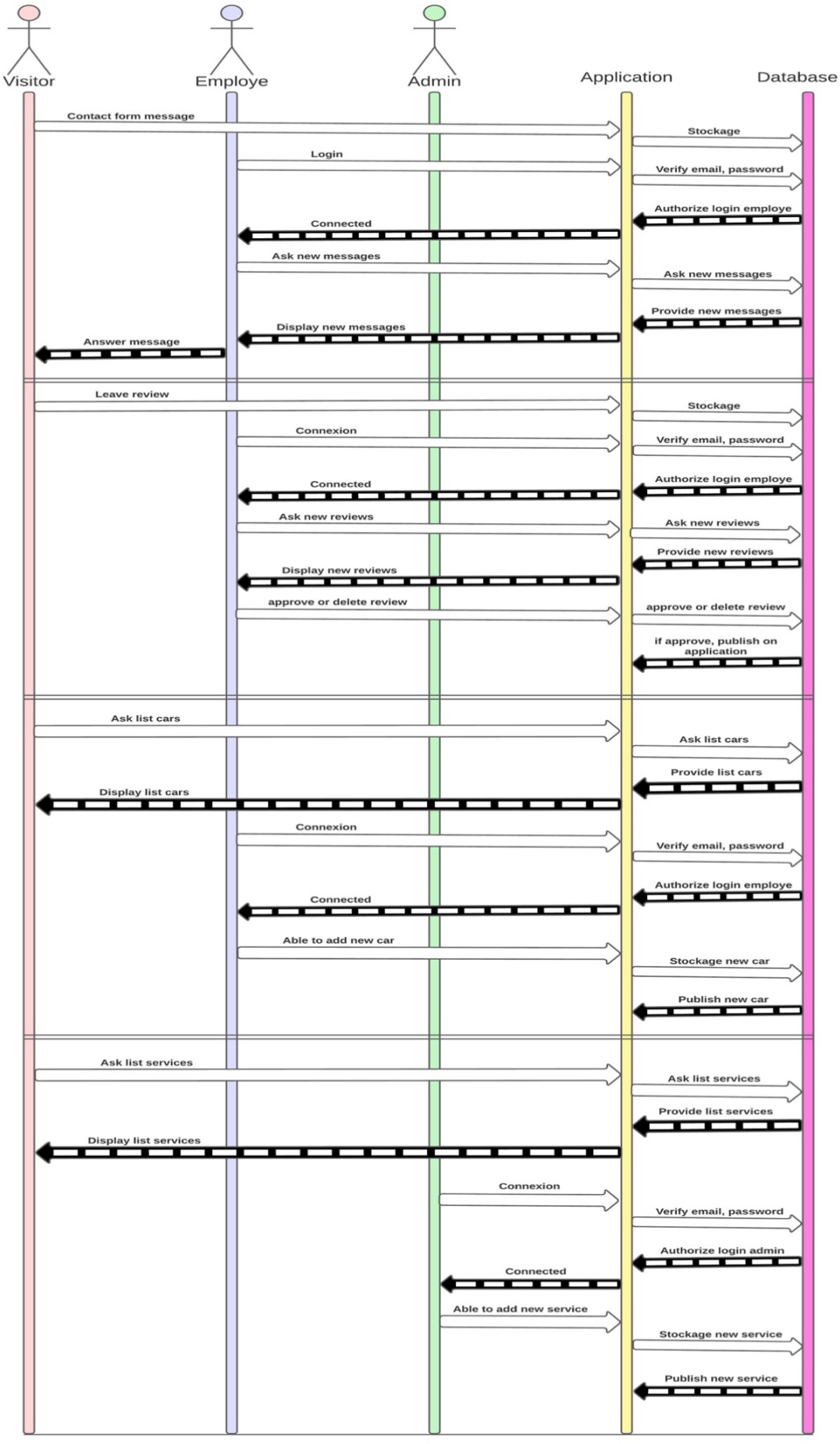
- ***Employee*** *:*

Se connecte à l'application, ce qui déclenche une vérification de l'email et du mot de passe dans la base de données.
Peut demander à voir les nouveaux messages ou les nouveaux avis (reviews), qui sont alors récupérés de la base de données et affichés.
A la possibilité d'approuver ou de supprimer des avis.
Peut demander et afficher la liste des voitures (list cars).

- ***Admin*** *:*

Dispose de fonctionnalités similaires à l'employé pour la connexion, l'affichage et la gestion des messages et des avis.
Peut aussi ajouter de nouvelles voitures et de nouveaux services, qui seront stockés dans la base de données et potentiellement publiés sur l'application.

Le diagramme montre également les interactions entre les acteurs et l'application, ainsi que la façon dont l'application interagit avec la base de données.


Visitor
Employe
Admin
Application
Database
Contact form message
Stockage
Login
Verify email, password
Authorize login employe
Connected
Ask new messages
Ask new messages
Provide new messages
Display new messages
Answer message
Leave review
Stockage
Connexion
Verify email, password
Connected
Authorize login employe
Ask new reviews
Ask new reviews
Provide new reviews
Display new reviews
approve or delete review
approve or delete review
if approve, publish on application
Ask list cars
Ask list cars
Provide list cars
Display list cars
Connexion
Verify email, password
Connected
Authorize login employe
Able to add new car
Stockage new car
Publish new car
Ask list services
Ask list services
Provide list services
Display list services
Connexion
Verify email, password
Authorize login admin
Connected
Able to add new service
Stockage new service
Publish new service

Synthèse des schémas :

Une fois sur l'application, vous pouvez vous connecter soit en tant qu'administrateur soit en tant qu'employé, en cliquant sur le bouton **"Connexion pro"** présent en haut à droite dans la Navbar de chaque page. Les identifiants nécessaires pour se connecter sont une adresse mail et un mot de passe.

L'administrateur, Vincent Parrot, une fois connecté, peut :

- Ajouter ou supprimer un compte employé, les comptes employés crées apparaitront directement sur la page d'accueil de l'administrateur et il lui sera possible d'en supprimer un en cliquant sur le bouton "supprimer", un message de confirmation sera ouvert.

- Modifier les heures d'ouverture du garage (présentes dans le pied de chaque page du site)

- **Ajouter un service et une image reliée à ce service** qui seront visibles par les utilisateurs du site en cliquant dans la barre de menu sur l'onglet "nos services".

- Supprimer un service, tous les services crées seront également présentés dans la page d'accueil de l'administrateur et disposent d'un bouton pour les supprimer, un message de confirmation apparaitra.

- Il peut également faire tout ce que l'employé va pouvoir faire :

Un employé, une fois connecté, peut :

- Consulter la liste des messages envoyés par les utilisateurs dans le formulaire de contact (présent sur la page "contact").

- Ajouter un nouveau véhicule à la base de données des véhicules d'occasion. Le nouveau véhicule apparaitra à la fin de la liste des véhicules.

- Supprimer un véhicule présent dans la base de données, il pourra consulter la liste de tous les véhicules de manière concise et cliquer sur le bouton "supprimer" présent sous chaque véhicule, un message de confirmation sera ouvert.

- Valider ou supprimer un avis client déposé par un utilisateur, s'il est validé, l'avis apparaitra sur la page "commentaires".

- Ajouter directement un avis client, qui sera visible sur la page d'accueil du site.

Une fois l'élaboration de la structure de la base de données terminée, j'ai procédé au développement de la partie **back-end** afin de générer la base de données.

2.Initialisation du projet

Installation de Composer :

« Composer » est un outil de gestion des dépendances pour les projets PHP. Il facilite l'installation, la mise à jour et la gestion des bibliothèques externes dont le projet pourrait avoir besoin. C'est un moyen efficace de gérer les dépendances de l'application et de garantir que toutes les bibliothèques requises sont disponibles.

```
chris@MBPdeChristophe templatemo_464_ultra_profile % composer --version
Composer version 2.6.5 2023-10-06 10:11:52
```

Installation de Symfony CLI :

Installer « Symfony CLI » est une étape importante pour un développeur Symfony. Symfony CLI est un outil en ligne de commande spécifique à Symfony qui facilite diverses tâches liées au développement Symfony.
En l'installant, on obtient un ensemble de commandes pratiques pour créer de nouveaux projets Symfony, générer des contrôleurs, exécuter le serveur de développement, et bien plus encore. Cela simplifie le processus de développement en automatisant certaines tâches récurrentes, ce qui peut nous faire gagner du temps et améliorer l'efficacité du flux de travail. En résumé, Symfony CLI rend le développement avec Symfony plus fluide et plus efficace.

```
Symfony CLI version 5.8.6 (c) 2021-2024 Fabien Potencier (2024-01-30T13:12:32Z - stable)
```

Symfony new :

Afin de commencer le développement du projet, j'ai commencé par initialiser Symfony sur mon ordinateur grâce au terminal et à la commande « *symfony new –webapp GarageAuto* ». La version utilisée de Symfony est la 7.

```
chris@MBPdeChristophe GarageAuto % symfony new --webapp GarageAuto
 * Creating a new Symfony project with Composer
   (running /usr/local/bin/composer create-project symfony/skeleton /Users/chris/Desktop/GarageAuto/GarageAuto  --no-interaction)

 * Setting up the project under Git version control
   (running git init /Users/chris/Desktop/GarageAuto/GarageAuto)

   (running /usr/local/bin/composer require webapp --no-interaction)

 [OK] Your project is now ready in /Users/chris/Desktop/GarageAuto/GarageAuto
```

La version de PHP installée est la 8.2.10.
La version de MySQL est la 8.1.0.
Une fois installé, j'ai relié le projet à un nouveau dépôt Github (git init).

Configuration du fichier « .env.local » :

J'ai créé un fichier « .env.local » afin de ne pas risquer d'exposer des données sensibles dans un dépôt git public.

On peut y trouver une variable APP_ENV en dev car on est en cours de développement, et on la passera en prod au moment de la production.

Le APP_SECRET correspond à une clé qui sera utilisé pour générer les jetons CSRF au niveau de la validation des formulaires, on pourra la modifier quand on passera en prod (par sécurité).

La partie MESSENGER_TRANSPORT_DSN (pour la partie notifications, envois de mails avec files d'attente par exemple) sert à choisir où sera stockée la file d'attente.

La partie DATABASE_URL sert à configurer la connexion à la base de données (nom d'utilisateur, mot de passe, nom de la base de données).

La partie MAILER_DSN sera utilisée avec un intercepteur d'email (« Mailhog » par exemple) pour tester les envois d'emails en mode dev.

```
APP_ENV=dev
APP_SECRET=3cc13b2c8403dd8c174b149bc3572766
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root@127.0.0.1:3306/GarageAuto?serverVersion=8.0.32&charset=utf8mb4"
# DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
###< doctrine/doctrine-bundle ###

###> symfony/messenger ###
# Choose one of the transports below
# MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
# MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
MESSENGER_TRANSPORT_DSN=doctrine://default?auto_setup=0
###< symfony/messenger ###

###> symfony/mailer ###
MAILER_DSN=smtp://localhost:1025
###< symfony/mailer ###
```

3.Création de la base de données

J'ai ensuite créé la base de données en local grâce à Doctrine depuis le fichier *.env.local* et la commande « *symfony console doctrine database:create* ».

```
chris@MBPdeChristophe GarageAuto % symfony console doctrine:database:create
Created database `GarageAuto` for connection named default
```

Voici l'URL du fichier .env.local configuré pour se connecter à ma base de données MySQL :

```
DATABASE_URL="mysql://root@127.0.0.1:3306/GarageAuto?serverVersion=8.0.32&charset=utf8mb4"
```

A l'aide des schémas de modèles de base de données réalisés en amont du projet, j'ai créé les différentes entités de la base via la commande « *symfony console make:entity* » avec laquelle j'ai pu ajouter les propriétés et leur type (string, int…).

Voici deux exemples des créations d'entités « User » et « Services »

```
chris@MBPdeChristophe GarageAuto % symfony console make:user

The name of the security user class (e.g. User) [User]:
>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!

Next Steps:
  - Review your new App\Entity\User class.
  - Use make:entity to add more fields to your User entity and then run make:migration.
  - Create a way to authenticate! See https://symfony.com/doc/current/security.html
```

```
chris@macbook-pro-de-christophe Garage_Vincent_Parrot % symfony console make:entity

 Class name of the entity to create or update (e.g. DeliciousElephant):
 > Services

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > ServicesImages

 Field type (enter ? to see all types) [string]:
 > relation

 What class should this entity be related to?:
 > ServicesImages

What type of relationship is this?
 ------------ ---------------------------------------------------------------------------
  Type         Description
 ------------ ---------------------------------------------------------------------------
  ManyToOne    Each Services relates to (has) one ServicesImages.
               Each ServicesImages can relate to (can have) many Services objects.

  OneToMany    Each Services can relate to (can have) many ServicesImages objects.
               Each ServicesImages relates to (has) one Services.

  ManyToMany   Each Services can relate to (can have) many ServicesImages objects.
               Each ServicesImages can also relate to (can also have) many Services objects.

  OneToOne     Each Services relates to (has) exactly one ServicesImages.
               Each ServicesImages also relates to (has) exactly one Services.
 ------------ ---------------------------------------------------------------------------

 Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
 > OneToMany

 A new property will also be added to the ServicesImages class so that you can access and set the related Services object from it.

 New field name inside ServicesImages [services]:
 >

 Is the ServicesImages.services property allowed to be null (nullable)? (yes/no) [yes]:
 >
```

J'ai facilement établi les relations entre les entités grâce à la propriété « relation » que propose Symfony en fonction des types de relations choisies (ManyToMany, OneToMany, OneToOne), la console m'a configuré automatiquement des propriétés et des méthodes dans les entités me permettant de faire le pont entre chacune.

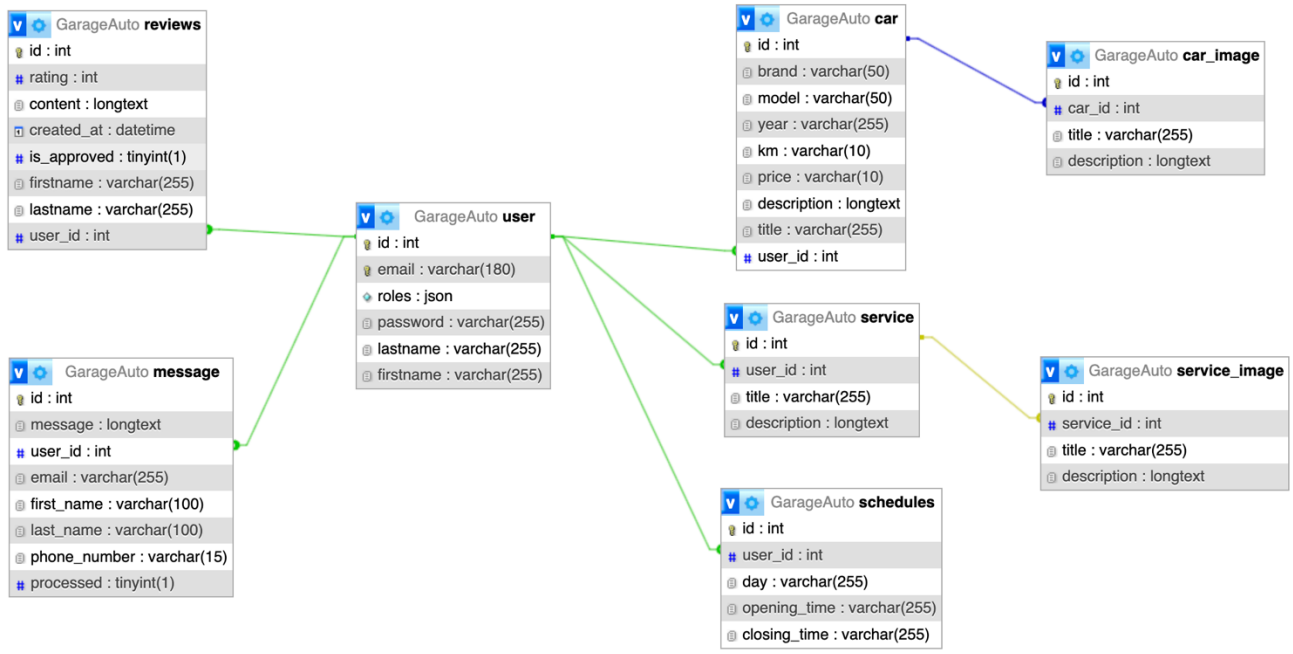
Enfin, pour pousser toutes ces nouvelles entités en base, j'ai utilisé le terminal et Doctrine pour générer et envoyer les migrations « *symfony console make:migration // symfony console doctrine:migrations:migrate* ».

```
chris@macbook-pro-de-christophe VParrot % symfony console make:migration

  created: migrations/Version20231106153745.php

  Success!

  Review the new migration then run it with symfony console doctrine:migrations:migrate
  See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
chris@macbook-pro-de-christophe VParrot %
```

```
chris@macbook-pro-de-christophe VParrot % symfony console doctrine:migrations:migrate

 WARNING! You are about to execute a migration in database "vparrot" that could result in schema changes and da
tinue? (yes/no) [yes]:
 >

[notice] Migrating up to DoctrineMigrations\Version20231106153745
[notice] finished in 119.1ms, used 20M memory, 1 migrations executed, 14 sql queries

[OK] Successfully migrated to version : DoctrineMigrations\Version20231106153745
```

Une fois la migration terminée, voici ma base de données qui apparait dans mon phpmyadmin local :

	Table	Action	Lignes	Type	Interclassement	Taille	Perte
☐	car	Parcourir Structure Rechercher Insérer Vider Supprimer	8	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	car_image	Parcourir Structure Rechercher Insérer Vider Supprimer	18	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	doctrine_migration_versions	Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8mb3_unicode_ci	16,0 kio	–
☐	message	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	messenger_messages	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	64,0 kio	–
☐	reviews	Parcourir Structure Rechercher Insérer Vider Supprimer	8	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	schedules	Parcourir Structure Rechercher Insérer Vider Supprimer	7	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	service	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	service_image	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
☐	user	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_unicode_ci	32,0 kio	–
	10 tables	**Somme**	**63**	**InnoDB**	**utf8mb4_0900_ai_ci**	**336,0 kio**	**0 o**



J'ai également créé la base de données à l'aide de requêtes SQL

Ce fichier s'occupe de la création de la base de données, des tables, des clés étrangères, du compte admin de Vincent Parrot, des horaires. En voici un extrait expliqué :

```sql
-- Création de la base de données
CREATE DATABASE `GarageAuto`;
USE `GarageAuto`;


-- Table pour les horaires d'ouverture des services
CREATE TABLE Schedules (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    userID INT,
    Day INT,
    OpeningTime TIME,
    ClosingTime TIME,
    FOREIGN KEY (userID) REFERENCES Users(ID)
);
```

« CREATE DATABASE » et « USE » sont deux commandes SQL utilisées pour créer une nouvelle base de données appelée "GarageAuto" et sélectionner cette base de données pour y effectuer des opérations.

Ensuite, le code SQL créé une table appelée "Schedules" (horaires) dans une base de données relationnelle.

« CREATE TABLE Schedules » : Cette ligne indique le début de la définition de la table. Le nom de la table est "Schedules".
« ID INT AUTO_INCREMENT PRIMARY KEY » : Cette ligne définit une colonne appelée "ID" avec le type de données INT (entier). La clause "AUTO_INCREMENT" indique que cette colonne se remplira automatiquement avec une valeur incrémentielle à chaque nouvel enregistrement, ce qui est couramment utilisé pour les clés primaires. La clause "PRIMARY KEY" déclare cette colonne comme la clé primaire de la table, ce qui signifie qu'elle doit contenir des valeurs uniques et qu'elle sera utilisée pour identifier de manière unique chaque enregistrement dans la table.
« userID INT » : Cette ligne est destinée à stocker l'identifiant d'un utilisateur associé à l'entrée du programme.
« Day INT » : Cette ligne définit une colonne appelée "Day" avec le type de données INT. Elle est prévue pour stocker le jour de la semaine pour lequel le programme est défini.
« OpeningTime TIME » : Cette ligne définit une colonne appelée "OpeningTime" avec le type de données TIME. Cette colonne stockera l'heure d'ouverture du service pour le jour spécifié.
« ClosingTime TIME » : Cette ligne définit une colonne appelée "ClosingTime" avec le type de données TIME. Cette colonne stockera l'heure de fermeture du service pour le jour spécifié.
« FOREIGN KEY (userID) REFERENCES Users(ID) » : Cette ligne établit une contrainte de clé étrangère. Elle indique que la colonne "userID" de la table "Schedules" est une clé étrangère qui fait référence à la colonne "ID" de la table "Users". Cela signifie que chaque valeur dans la colonne "userID" doit correspondre à une valeur existante dans la colonne "ID" de la table "Users". Cela crée une relation entre les deux tables, reliant les horaires aux utilisateurs auxquels ils appartiennent.

Exemple de l'entité « car » :

```php
#[ORM\Entity(repositoryClass: CarRepository::class)]
class Car
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 50)]
    private ?string $brand = null;

    #[ORM\Column(length: 50)]
    private ?string $model = null;

    #[ORM\Column]
    private ?string $year = null;

    #[ORM\Column(length: 10)]
    private ?string $price = null;

    #[ORM\Column(length: 10)]
    private ?string $km = null;
```

Au-dessus de chaque propriété, on a ce que l'on appelle des annotations permettant de leur donner des spécificités.
Par exemple on peut voir qu'au-dessus de la propriété id, on a orm id qui indique la clé primaire, orm generated value indique que c'est le sgbd qui va gérer l'auto incrémentation de la clé primaire et que la clé primaire est de type integer.

4.Création des Datafixtures

L'étape suivante consiste à la création de data fixtures qui seront utilisées pour peupler ma base de données avec des données de tests ou initiales. Cela va permettre de travailler avec un ensemble de données cohérent et prérempli. J'exécute la commande suivante sur le terminal :

```
chris@MBPdeChristophe GarageAuto % composer require orm-fixtures
chris@MBPdeChristophe GarageAuto % symfony console make:fixtures ReviewsFixtures
 created: src/DataFixtures/ReviewsFixtures.php

 Success!

 Next: Open your new fixtures class and start customizing it.
 Load your fixtures by running: php bin/console doctrine:fixtures:load
 Docs: https://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html
```

Voici un exemple de création de Datafixtures concernant les avis clients :

```php
<?php

namespace App\DataFixtures;

use App\Entity\Reviews;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class ReviewsFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        $review=new Reviews();      You, le mois dernier • corrections base de donnees et fixtures
        $review->setFirstname('Stacy');
        $review->setLastname('Moss');
        $review->setRating(5);
        $review->setContent('Au top ce garage !');
        $review->setCreatedAt (\DateTimeImmutable::createFromFormat('Y-m-d H:i:s', '2024-02-19 12:00:00'));
        $review->setIsApproved(true);
        $manager->persist($review);
```

Je charge ensuite les fixtures grâce à la commande suivante :

```
chris@MBPdeChristophe GarageAuto % symfony console doctrine:fixtures:load

  Careful, database "garageauto" will be purged. Do you want to continue? (yes/no) [no]:
  > yes

    > purging database
    > loading App\DataFixtures\CustomersFixtures
```

Et ma base de données se remplit comme on peut le voir :

	id	rating	content	created_at (DC2Type:datetime_immutable)	is_approved	firstname	lastname	user_id
Éditer Copier Supprimer	24	5	Au top ce garage !	2024-02-19 12:00:00	1	Stacy	Moss	NULL
Éditer Copier Supprimer	25	5	Je suis très satisfait, ma formule 1 avit un probl...	2023-11-14 11:00:00	1	Esteban	Ocon	NULL
Éditer Copier Supprimer	26	4	Génial, un point en moins pour la fermeture le dim...	2023-09-10 12:00:00	1	Emilia	Clark	NULL
Éditer Copier Supprimer	27	5	Le meilleur garage au monde !	2024-01-31 12:00:00	1	Lebron	James	NULL
Éditer Copier Supprimer	28	5	Ils ont pu réparer ma fusée en 5minutes, de vrais ...	2024-11-05 12:00:00	1	Elon	Musk	NULL
Éditer Copier Supprimer	29	4	Sympathique mais locaux un peu petits...	2023-12-28 12:00:00	1	Luc	Skywalker	NULL
Éditer Copier Supprimer	54	5	super mega genial	2024-03-10 14:23:00	1	CHRISTOPHE	MOSTEFAOUI	NULL
Éditer Copier Supprimer	55	1	nul comme garage	2024-03-10 13:24:03	1	Mohamed	MOSTEFAOUI	NULL

J'ai donc créé ces données initiales pour les entités horaires, avis clients, employés, voitures, services.

J'aurais pu installer le composant « *fakerphp* » afin de créer des fausses données sans avoir à les rentrer moi-même mais cela n'a pas été indispensable pour la création de ce site.

J'ai également utilisé des commandes directement en SQL comme je l'ai appris en cours afin de rajouter des données comme l'exemple que vous pouvez voir ou je rajoute une datafixture :

```sql
INSERT INTO reviews (firstname, lastname, rating, content, created_at, is_approved)
VALUES ('Stacy', 'Moss', 5, 'Au top ce garage !', '2024-02-19 12:00:00', true);
```

Vérification de la base de données :

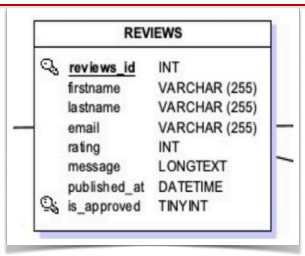
Une fois que j'ai conçu ma base de données, j'ai ouvert mon terminal pour me connecter à mon serveur de base de données et en tapant la commande SQL **SHOW TABLES** voilà la liste des tables que j'ai obtenues.

```
chris@MBPdeChristophe GarageAuto % mysql -u root -p -h 127.0.0.1 -P 3306 GarageAuto
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.1.0 Homebrew

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW TABLES;
+------------------------------+
| Tables_in_garageauto         |
+------------------------------+
| car                          |
| car_image                    |
| doctrine_migration_versions  |
| message                      |
| messenger_messages           |
| reviews                      |
| schedules                    |
| service                      |
| service_image                |
| user                         |
+------------------------------+
10 rows in set (0,01 sec)
```

Par exemple, pour voir en détail la structure de la table « reviews », je tape la requête **SELECT * FROM reviews;**

On peut voir qu'elle correspond exactement à ce que je voulais dans le MCD

```
mysql> SELECT * FROM Reviews;
+----+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-------
------+------------+------------+---------+
| id | rating | content                                                                                                                                                     | created_at          | is_app
roved | firstname  | lastname   | user_id |
+----+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-------
------+------------+------------+---------+
| 24 |      5 | Au top ce garage !                                                                                                                                          | 2024-02-19 12:00:00 |
    1 | Stacy      | Moss       |    NULL |
| 25 |      5 | Je suis très satisfait, ma formule 1 avit un problème pour accélérer, Mr Parrot a su trouver le problème rapidement je recommande ce garage!               | 2023-11-14 11:00:00 |
    1 | Esteban    | Ocon       |    NULL |
| 26 |      4 | Génial, un point en moins pour la fermeture le dimanche !                                                                                                   | 2023-09-10 12:00:00 |
    1 | Emilia     | Clark      |    NULL |
| 27 |      5 | Le meilleur garage au monde !                                                                                                                               | 2024-01-31 12:00:00 |
    1 | Lebron     | James      |    NULL |
| 28 |      5 | Ils ont pu réparer ma fusée en 5minutes, de vrais pro !                                                                                                     | 2024-11-05 12:00:00 |
    1 | Elon       | Musk       |    NULL |
| 29 |      4 | Sympathique mais locaux un peu petits...                                                                                                                    | 2023-12-28 12:00:00 |
    1 | Luc        | Skywalker  |    NULL |
| 54 |      5 | super mega genial                                                                                                                                           | 2024-03-10 14:23:00 |
    1 | CHRISTOPHE | MOSTEFAOUI |    NULL |
| 55 |      1 | nul comme garage                                                                                                                                            | 2024-03-10 13:24:03 |
    1 | Mohamed    | MOSTEFAOUI |    NULL |
+----+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-------
------+------------+------------+---------+
8 rows in set (0,01 sec)
```



J'en ai profité pour faire une sauvegarde de cette base de données avec le mysqldump

```
chris@MBPdeChristophe GarageAuto % mysqldump -u root -p -h 127.0.0.1 GarageAuto > GarageAuto_backup.sql
```

5.Mise en place des routes et des contrôleurs :

Pour créer les pages de l'application, j'ai utilisé la commande « *symfony console make:controller* » qui m'a permis de créer les routes et les fonctions associées.

```php
<?php

namespace App\Controller;

use App\Repository\ReviewsRepository;
use App\Repository\SchedulesRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

You, le mois dernier | 1 author (You)
class HomeController extends AbstractController
{
    #[Route('/', name: 'app_home')]
    public function index(SchedulesRepository $schedulesRepository, ReviewsRepository $reviewsRepository): Response
    {
        @var string[] $days
        $days = ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche'];
        $reviews = $reviewsRepository->findApprovedReviews();

        foreach ($days as $day) {
            $workingHours[$day] = $schedulesRepository->findWorkingHoursByDay($day);
        }

        return $this->render('home/index.html.twig', [
            'workingHours' => $workingHours,
            'reviews' => $reviews
        ]);
    }
}

```

Ici l'exemple du controller de la page d'accueil. Il est assez simple ; il n'a qu'une seule action (index) et une seule route, qui correspond à l'URL racine.

On a tout d'abord la déclaration du contrôleur sous ***le namespace*** *App\Controller.*
Il utilise plusieurs composants et services de Symfony, comme ReviewsRepository et SchedulesRepository pour accéder aux avis et aux horaires de travail.

ClassHomeController *hérite de AbstractController, ce qui lui permet d'utiliser des fonctionnalités de Symfony comme la méthode render. La méthode render est celle qui est utilisée pour générer et renvoyer le contenu d'un template (vue) dans le cadre d'une réponse HTTP. Elle prépare donc la vue qui sera affichée à l'utilisateur. Elle fait partie de la classe AbstractController, que les contrôleurs Symfony héritent la plupart du temps.*

Ensuite, on a la ***méthode index*** *qui est associée à la route '/' grâce à l'annotation #[Route('/', name: 'app_home')]. Cela signifie que cette méthode est appelée lorsque quelqu'un accède à la page d'accueil de notre site.*

Pour la récupération des données, on a ***2 objets*** *qui sont injectés dans la méthode index : SchedulesRepository et ReviewsRepository. Ils permettent de récupérer des données depuis la base de données. findApprovedReviews() du ReviewsRepository est appelé pour obtenir les avis approuvés.*

Une ***liste de jours de la semaine*** *est définie ($days).*

- *Pour chaque jour de la semaine, la méthode findWorkingHoursByDay($day) du SchedulesRepository est appelée pour obtenir les horaires de travail correspondant. Les résultats sont stockés dans le tableau $workingHours.*

Pour ***le rendu de la vue****, le contrôleur utilise la méthode render pour afficher la vue home/index.html.twig. Il passe deux variables à la vue : workingHours (les horaires de travail pour chaque jour) et reviews (les avis approuvés).*

Les controllers sont nécessaires pour gérer les requêtes HTTP entrantes, récupérer les données de la requête (paramètres, données de formulaire, etc.), prendre des décisions en fonction des données, interagir avec la base de données, et renvoyer une réponse appropriée au client, cela peut inclure le rendu de vues à l'aide de Twig dans mon cas. Les vues représentent généralement la partie présentation de l'application. Les contrôleurs sont associés à des routes spécifiques qui déterminent quels contrôleurs et actions (méthodes) doivent être appelés en fonction de l'URL demandée.

6.EasyAdmin et la création de l'interface administration :

Lorsque l'on souhaite mettre en place les fonctions de CRUD (Create, Read, Update, Delete) sur une entité, Symfony met à disposition plusieurs méthodes. J'ai choisi d'utiliser *EasyAdmin* qui est une extension Symfony qui facilite la création d'interfaces d'administration pour les applications web. Il fournit un ensemble d'outils et de fonctionnalités préconstruits qui permettent de créer rapidement des panneaux d'administration pour la gestion des entités de l'application. *EasyAdmin* prend également en compte la sécurité en intégrant des fonctionnalités d'autorisation.

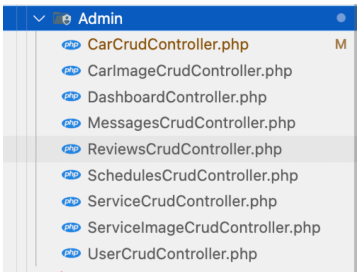
La commande suivante m'a permis l'installation de l'extension :
"composer require easycorp/easyadmin-bundle"

Voici les étapes qui m'ont permis la création de l'interface administrateur :

- Création du dashboard :
  "symfony console make:admin:dashboard"

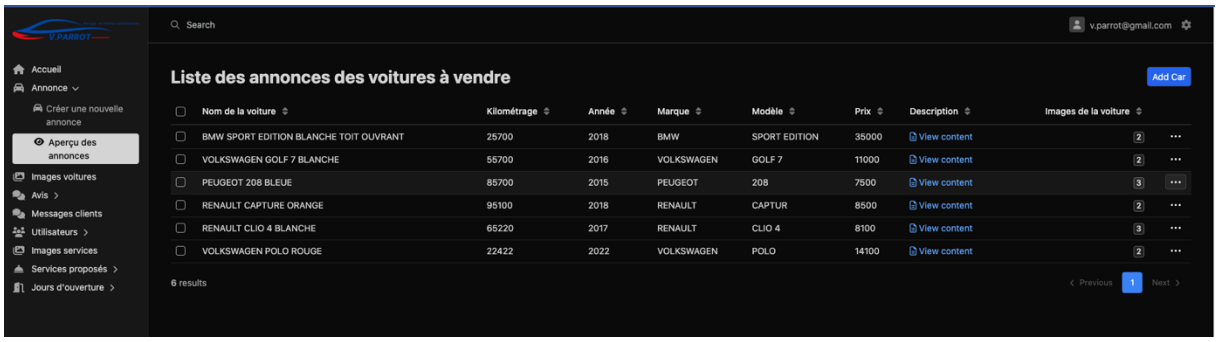
- Création des CrudControllers :
  Ils vont permettre de créer les pages qui vont nous permettre de consulter les données de nos différentes entités.

  « symfony console make:admin:crud



Les Cruds Controllers servent à créer les pages qui vont nous permettre de consulter les données de nos différentes entités.

J'ai ensuite personnalisé les pages de l'administration en rajoutant le logo de l'entreprise, en ajoutant ou en supprimant des champs nécessaires à la navigation confortable des utilisateurs. Voici un aperçu du dashboard en étant Vincent Parrot :



Voici un exemple de la configuration du CarCrudController

```php
<?php

namespace App\Controller\Admin;

use App\Entity\Car;
use EasyCorp\Bundle\EasyAdminBundle\Config\Crud;
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;
use EasyCorp\Bundle\EasyAdminBundle\Field\CollectionField;
use EasyCorp\Bundle\EasyAdminBundle\Field\TextEditorField;
use EasyCorp\Bundle\EasyAdminBundle\Field\TextField;
use EasyCorp\Bundle\EasyAdminBundle\Field\AssociationField;

You, il y a 13 secondes | 1 author (You)
class CarCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Car::class;
    }

    public function configureFields(string $pageName): iterable
    {
        return [
            TextField::new('title')->setLabel('Nom de la voiture'),
            TextField::new('km')->setLabel('Kilométrage'),
            TextField::new('year')->setLabel('Année'),
            TextField::new('brand')->setLabel('Marque'),
            TextField::new('model')->setLabel('Modèle'),
            TextField::new('price')->setLabel('Prix'),
            TextEditorField::new('description')->setLabel('Description'),
            CollectionField::new('carImages')
            ->onlyOnForms()
            ->setLabel('Images de la voiture')
            ->setTemplatePath('admin/car_images.html.twig'), // chemin vers le template Twig pour afficher les images
            AssociationField::new('carImages')
            ->setLabel('Images de la voiture')
            ->setFormTypeOptions([
                'by_reference' => false,
                ])
            ];
}

    public function configureCrud(Crud $crud): Crud
    {
        return $crud
            ->setPageTitle('index', 'Liste des annonces des voitures à vendre')
            ->setPageTitle('new', 'Créer une nouvelle annonce de voiture à vendre')
            ->setPaginatorPageSize(20);
    }
}
```

Cliquez pour réduire

Namespace : Le code est placé dans le namespace App\Controller\Admin, indiquant que ce contrôleur est destiné à être utilisé dans la section d'administration de l'application.

Héritage : La classe CarCrudController étend AbstractCrudController fourni par EasyAdminBundle. Cela signifie que CarCrudController bénéficie de fonctionnalités spécifiques pour la gestion des entités dans une interface d'administration.

getEntityFqcn : Cette méthode statique renvoie le nom complet de la classe de l'entité associée, Car::class.

configureFields : Cette méthode configure les champs qui seront affichés dans les différentes pages de l'interface d'administration. On y trouve des champs tels que TextField, TextEditorField, AssociationField, et CollectionField, chacun correspondant à un type de champ différent.

TextField est utilisé pour les champs de texte simples.

TextEditorField est utilisé pour les champs de texte plus longs, souvent utilisés pour les descriptions.

AssociationField est utilisé pour représenter des relations entre entités.

CollectionField est utilisé pour gérer des collections d'entités (des images).

configureCrud : Cette méthode configure des paramètres spécifiques pour la gestion du CRUD en utilisant l'objet Crud. On y trouve des configurations telles que le titre de la page, le nombre d'éléments par page dans la pagination, etc.

setPageTitle est utilisé pour définir le titre de la page pour différentes actions (index, new, etc.).

setPaginatorPageSize définit le nombre d'éléments à afficher par page dans la liste.

En résumé, ce code simplifie la gestion des annonces de voitures dans l'application grâce à EasyAdmin, en offrant une interface d'administration complète et personnalisable.

7.Gestion des droits :

La notion des rôles et des droits a été une notion importante dans ce projet.
Lors de la réalisation des personas, j'ai identifié deux rôles différents comme précédemment évoqués (un rôle super administrateur pour le patron : ROLE_SUPER_ADMIN et un rôle administrateur pour les employés : ROLE_ADMIN).
Par défaut, Symfony injecte le statut de ROLE_USER à chaque nouvel objet créé de type User que j'ai modifié en ROLE_ADMIN. Depuis le fichier *security.yaml*, j'ai utilisé la hiérarchie des rôles pour déterminer les droits de chaque type d'utilisateur. Cette fonctionnalité utilise l'héritage pour gérer les droits de chacun (le SUPER_ADMIN aura forcément accès aux droits du ROLE_ADMIN).

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }

role_hierarchy:      You, il y a 4 jours • roles admin
    ROLE_SUPER_ADMIN: ROLE_ADMIN
```

	id	email	roles	password	lastname	firstname
Éditer Copier Supprimer	51	v.parrot@gmail.com	["ROLE_SUPER_ADMIN"]	$2y$13$D2lCuz6vCKpFt8nDbGKiiOXAu7J6F4V8rgLdmKYS0N3...	Parrot	Vincent
Éditer Copier Supprimer	52	cmurcie18@gmail.com	["ROLE_ADMIN"]	$2y$13$4KFKbgALbGVfplwzcQTQYuwHLlo1RRAFCbFrBDZloox...	Murcie	Cyril
Éditer Copier Supprimer	53	festayre64@gmail.com	["ROLE_ADMIN"]	$2y$13$ZG2LYBR2rylyS9JUNC1LsepUvUZI4Dk9WOF5j43Wklb...	Duval	Marc
Éditer Copier Supprimer	54	Raki457@yahoo.fr	["ROLE_ADMIN"]	$2y$13$dFf8uKi5lyE9PPSl9YmZCu.5bvFMr7/yKmJ79amjFDw...	Raki	Chris
Éditer Copier Supprimer	55	rlegrand@gmail.com	["ROLE_ADMIN"]	$2y$13$zdX1Rf4lCBWISptTWD8KXu0FvXbwsBc.hBGSj1BlnRB...	Legrand	Romain
Éditer Copier Supprimer	56	c.mostefaoui@yahoo.fr	["ROLE_SUPER_ADMIN", "ROLE_ADMIN"]	$2y$13$nmmg6z7RQhuTGWCHpNrW5O9HXdW6LxWA9RMjcx4vgB8...	MOSTEFAOUI	CHRISTOPI

J'ai donc déterminé le rôle de SUPER_ADMIN pour Vincent Parrot et le rôle ADMIN pour les employés dans les fixtures.

Système d'authentification :

Pour permettre à l'utilisateur de se créer un compte et de se connecter, j'ai utilisé les fonctionnalités disponibles sur Symfony.

J'avais créé une entité utilisateur grâce à la commande « *symfony console make:user* » qui génère automatiquement l'entité avec des propriétés par défaut : email, mot de passe et rôles.

J'ai ensuite créé un formulaire d'inscription avec la commande « *symfony console make:form* » relié directement à l'entité User. Une fois les utilisateurs (employés) validés par le patron, ils peuvent se connecter grâce à un formulaire de connexion. Je me suis aidé du Maker de Symfony grâce à « *symfony console make:auth* ». Cette commande a généré automatiquement les fonctions de connexion/déconnexion avec les routes et les templates associés.

```
chris@MBPdeChristophe GarageAuto % symfony console make:auth

What style of authentication do you want? [Empty authenticator]:
  [0] Empty authenticator
  [1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> UsersAuthentificator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
>

Do you want to generate a '/logout' URL? (yes/no) [yes]:
>

Do you want to support remember me? (yes/no) [yes]:
>

How should remember me be activated? [Activate when the user checks a box]:
  [0] Activate when the user checks a box
  [1] Always activate remember me
> 0

created: src/Security/UsersAuthentificatorAuthenticator.php
updated: config/packages/security.yaml
updated: src/Controller/SecurityController.php
created: templates/security/login.html.twig

 Success!

Next:
- Customize your new authenticator.
- Finish the redirect "TODO" in the App\Security\UsersAuthentificatorAuthenticator::onAuthenticationSuccess() method.
- Review & adapt the login template: templates/security/login.html.twig.
```

Voici un extrait de la page Users authentificator

```php
    public function authenticate(Request $request): Passport
    {
        $email = $request->request->get('email', '');

        $request->getSession()->set(SecurityRequestAttributes::LAST_USERNAME, $email);

        return new Passport(
            new UserBadge($email),
            new PasswordCredentials($request->request->get('password', '')),
            [
                new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token')),
                new RememberMeBadge(),
            ]
        );
    }

    public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
    {
        if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
            return new RedirectResponse($targetPath);
        }

        // For example:
        return new RedirectResponse($this->urlGenerator->generate('app_home'));
        // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
    }

    protected function getLoginUrl(Request $request): string
    {
        return $this->urlGenerator->generate(self::LOGIN_ROUTE);
    }
}
```

La méthode authentificate extrait l'email et le mot de passe de la requête et crée un objet Passport pour l'authentification. Le Passport contient :

- *UserBadge qui identifie l'utilisateur basé sur l'email.*
- *PasswordCredentials qui vérifie les informations d'identification fournies (mot de passe).*
- *CsrfTokenBadge qui vérifie le token CSRF pour sécuriser le formulaire*
- *RememberMeBadge qui gère l'option "Se souvenir de moi" pour la session.*
- *La méthode onAuthentificationsuccess est appelée lorsque l'authentification réussit. Elle redirige l'utilisateur vers la page qu'il essayait d'atteindre avant de se connecter (targetPath) ou vers la page d'accueil (app_home).*
- *La méthode getLoginUrl génère l'URL de la page de login.*

Pour résumer, cette class UsersAuthentificatorAuthenticator est responsable du processus de connexion utilisateur, en utilisant le mécanisme de Passport pour vérifier les informations d'identification et gérer les badges de sécurité comme le CSRF et le "Remember Me". Elle assure également la redirection adéquate après une connexion réussie.

L'interface de connexion :

```twig
{% extends 'base.html.twig' %}

{% block title %}Connexion administrateur{% endblock %}

{% block body %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <form method="post" class="text-center">
                {% if error %}
                    <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
                {% endif %}

                {% if app.user %}
                    <div class="mb-3">
                        Connecté en tant que {{ app.user.userIdentifier }}, <a href="{{ path('app_logout') }}">Déconnexion</a>
                    </div>
                {% endif %}

                <h1 class="h3 mb-3 font-weight-normal">Identifiez-vous</h1>

                <label for="inputEmail" class="sr-only">Email</label>
                <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control" placeholder="Email" autocomplete="email" required autofocus>

                <label for="inputPassword" class="sr-only">Mot de passe</label>
                <input type="password" name="password" id="inputPassword" class="form-control" placeholder="Mot de passe" autocomplete="current-password" required>

                <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

                <button class="btn btn-lg btn-primary" type="submit">
                    Connexion à l'espace administrateur
                </button>
            </form>
        </div>
    </div>
</div>
{% endblock %}
```

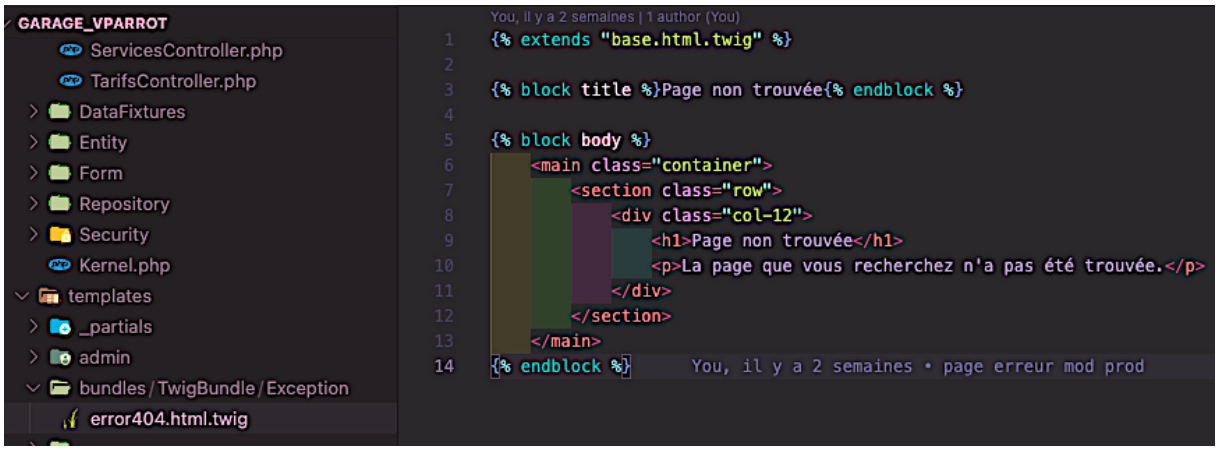
L'interface de connexion a été automatiquement générée dans le template security avec le nom « login.html.twig » avec le code ci-dessus.

On y affiche un message d'erreur si une erreur de connexion survient.
Si un utilisateur est déjà connecté (app.user), un message s'affiche indiquant son nom d'utilisateur et un lien de déconnexion.

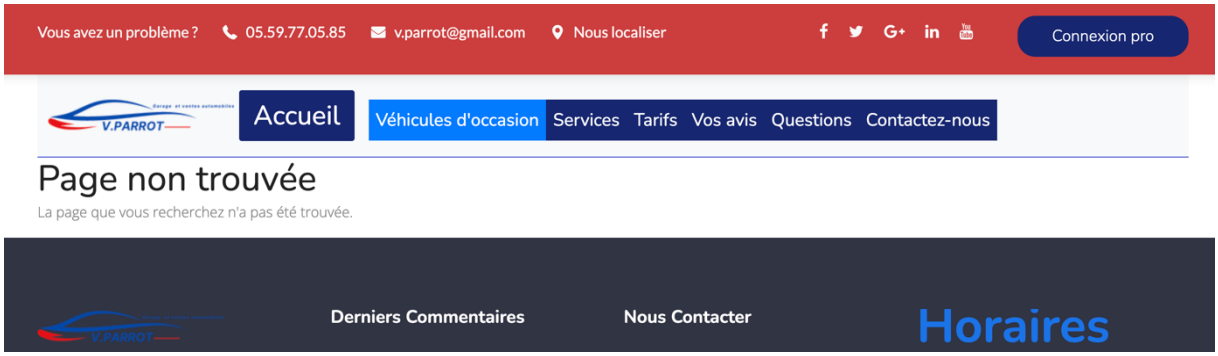
Pour résumé, Ce template Twig fournit une interface de connexion pour les administrateurs avec un formulaire sécurisé par un jeton CSRF et des messages d'erreur en cas de connexion incorrecte.

8.Pages d'erreurs

En cas d'erreur de page en mode prod, une page blanche s'affiche automatiquement. Il convient donc de configurer une page d'erreur afin de rendre plus agréable la navigation à l'utilisateur, grâce au « Twigpack ». *« composer require symfony/twig-pack »*



Voici le rendu visuel :



D. La réalisation du projet (front-end)

Dans un premier temps, j'ai désinstallé de Symfony le « webpack encore » grâce à la commande suivante :

« composer remove webpack »

Webpack est un outil puissant et offre de nombreuses fonctionnalités avancées pour la gestion des ressources front-end, mais il peut être complexe à configurer et à maintenir, en particulier pour les petits projets ou pour les développeurs qui ne sont pas familiers avec son fonctionnement, ce qui est mon cas.

Vues dynamiques :

Pour la partie front-end, j'ai utilisé Twig, le moteur de template Symfony. Avec ses variables, j'ai pu afficher des données dynamiques au sein des pages.
L'accès aux données présentes en base de données est rendu possible grâce aux contrôleurs dans lesquels j'ai fait passer les variables nécessaires aux vues.

Par exemple, voici une partie de la page services en twig affichant les images et la description des services venant de la base de données :

```
<h1>Les services que nous proposons actuellement :</h1>

<section class="service-area service-page">
  <div class="container">
    <div class="row">
      {% for service in services %}
        <div class="col-lg-3 col-md-3 col-sm-6 col-xs-12">
          <div class="single-service-item text-center" id="service-{{ loop.index }}">
            <div class="img-holder">
              {% if service.getServiceImages().first() %}
                <img src="{{ asset('/uploads/Services/' ~ service.getServiceImages().first().getTitle()) }}" alt="{{ service.getTitle() }}">
              {% else %}
                <img src="{{ asset('/uploads/Services/vidange.jpg') }}" alt="Default Image">
              {% endif %}
              <div class="overlay-style-one">
                <div class="box">
                  <div class="content">
                    <i class="fa fa-link" aria-hidden="true"></i>
                  </div>
                </div>
              </div>
            </div>
            <div class="text-holder">
              <h3>{{ service.getTitle() }}</h3>
              <p>{{ service.getDescription() }}</p>
            </div>
          </div>
        </div>
      {% endfor %}
    </div>
  </div>
</section>

{% endblock %}
```

Voici la vue de cette partie en version desktop et en version mobile :

Bootstrap :

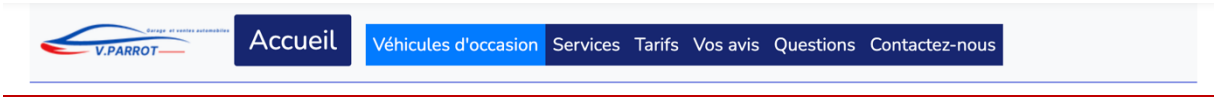
Afin d'optimiser mon temps, j'ai préféré utiliser le framework CSS Bootstrap 5 tout en restant fidèle aux maquettes et en y ajoutant du style personnalisé avec CSS. Celui-ci permet également d'automatiser le responsive du site, adaptable sur n'importe quelle taille d'écran.

Voici un exemple de l'utilisation de Bootstrap pour réaliser ma barre de menu rapidement :

```
  <!-- Main Menu -->
  <section class="mainmenu-area stricky">
    <div class="container">
      <div class="row">
        <div class="col-lg-12 clearfix">
          <!-- Menu Navbar -->
          <nav class="navbar navbar-expand-lg navbar-light bg-light navbar-transparent custom-navbar">
            <div class="container">
              <div class="logo pull-left">
                <a href="{{ path('app_home') }}">
                  <img src="assets/resources/logo.png" alt="Logo" />
                </a>
              </div>

              <a class="navbar-brand" href="{{ path('app_home') }}">Accueil</a>

              <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNavDropdown"
aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
              </button>

              <div class="collapse navbar-collapse" id="navbarNavDropdown">
                <ul class="navbar-nav">
                  <li class="nav-item active">
```

```
                <a class="nav-link" href="{{ path('app_car') }}">Véhicules d'occasion<span class="sr-only"></span></a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{{ path('app_services') }}">Services<span class="sr-only"></span></a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{{ path('app_tarifs') }}">Tarifs<span class="sr-only"></span></a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{{ path('app_reviews') }}">Vos avis<span class="sr-only"></span></a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{{ path('app_f_a_q') }}">Questions<span class="sr-only"></span></a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="{{ path('app_contact') }}">Contactez-nous<span class="sr-only"></span></a>
              </li>
            </ul>
          </div>
        </div>
      </nav>
      <!-- End main menu and navbar -->
```

Voici le rendu en version desktop et en version mobile :



CSS :

J'ai utilisé du CSS afin de personnaliser le style des pages du site.

Voici un exemple de CSS personnalisant l'apparence du titre de la page « services » :

```
<style>
    h1 {
        text-align: center;
        color: #233588;
        margin-top: 50px;
        line-height: 1.5;

         }
       </style>
<h1>Les services que nous proposons actuellement :</h1>
```

Voici le rendu :

Les services que nous proposons actuellement :

Javascript :

J'ai rajouté quelques éléments du site en Javascript afin de le rendre plus dynamique et plus attrayant comme par exemple le système de filtrage au niveau de la page des voitures d'occasion. En voici un exemple :

```
document.addEventListener("DOMContentLoaded", function() {

    // Éléments des curseurs
    const kilometerSlider = document.getElementById("kilometer-max");
    const priceSlider = document.getElementById("price-max-input");
    const yearSlider = document.getElementById("year-max-input");


    // Éléments pour afficher les valeurs actuelles des curseurs
    const kilometerDisplay = document.getElementById("kilometer-max-value");
    const priceDisplay = document.getElementById("price-max-value");
    const yearDisplay = document.getElementById("year-max-value");

    const resetButton = document.getElementById("reset-button");

    // Mise à jour des valeurs et affichage des véhicules correspondants
    function updateFilter() {
        const maxKilometer = parseInt(kilometerSlider.value, 10);
        const maxPrice = parseInt(priceSlider.value, 10);
        const maxYear = parseInt(yearSlider.value, 10);

        kilometerDisplay.textContent = `${maxKilometer} km`;
        priceDisplay.textContent = `${maxPrice} €`;
        yearDisplay.textContent = maxYear;

        // Filtrage des véhicules
        const cars = document.querySelectorAll(".cars-filter");
        cars.forEach(car => {
            const kilometer = parseInt(car.getAttribute("data-kilometer"), 10);
            const price = parseInt(car.getAttribute("data-price"), 10);
            const year = parseInt(car.getAttribute("data-year"), 10);

            if (kilometer <= maxKilometer && price <= maxPrice && year <= maxYear)
{
                car.style.display = "";
            } else {
                car.style.display = "none";
            }
        });
    }
    kilometerSlider.addEventListener("input", updateFilter);
```

```
    priceSlider.addEventListener("input", updateFilter);
    yearSlider.addEventListener("input", updateFilter);

    // Fonction de réinitialisation
    function resetFilters() {
        kilometerSlider.value = 180000;
        priceSlider.value = 100000;
        yearSlider.value = 2023;

        kilometerDisplay.textContent = "180000 km";
        priceDisplay.textContent = "100000 €";
        yearDisplay.textContent = "2023";

        const cars = document.querySelectorAll(".cars-filter");
        cars.forEach(car => car.style.display = "");

        updateFilter();
    }
    resetButton.addEventListener("click", resetFilters);
    updateFilter();
});
```

Voici une explication détaillée de ce code :

Les curseurs (kilometerSlider, priceSlider, yearSlider) et les éléments d'affichage (kilometerDisplay, priceDisplay, yearDisplay) sont récupérés à partir du DOM. Ces éléments représentent les contrôles de filtre et les éléments d'affichage des valeurs correspondantes.

La fonction updateFilter est responsable de mettre à jour les valeurs des filtres, d'afficher les valeurs, et de filtrer les véhicules en fonction des critères définis par les curseurs.

Des écouteurs d'événements sont ajoutés aux curseurs pour déclencher la fonction updateFilter lorsqu'un utilisateur modifie les valeurs des curseurs.

La fonction resetFilters réinitialise les valeurs des curseurs et réaffiche tous les véhicules, annulant ainsi tous les filtres appliqués.

Un écouteur d'événements est ajouté au bouton de réinitialisation pour déclencher la fonction resetFilters lorsqu'il est cliqué.

Enfin, la fonction updateFilter est appelée une fois au chargement initial de la page pour appliquer les filtres initiaux.

En résumé, ce script gère la mise en place de filtres pour des véhicules permettant à l'utilisateur de définir des critères de filtrage via des curseurs, avec la possibilité de réinitialiser les filtres à leurs valeurs par défaut.

E. LES VULNERABILITÉS

Comme tout projet configuré par des rôles avec des droits et des accès différents, il existe plusieurs types de vulnérabilités, dont trois qui sont les plus répandues :

La faille CSRF

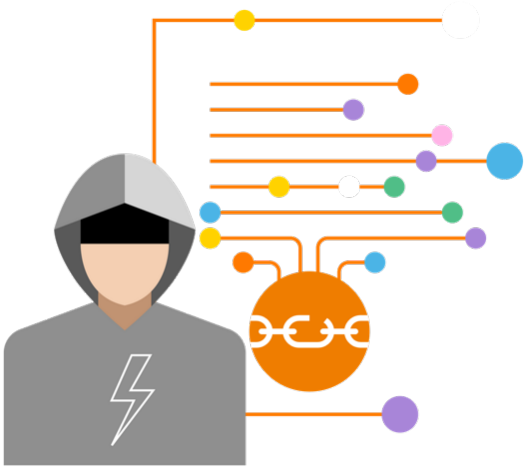
Une des plus connues dont il faut se préparer est la faille CSRF (Cross-Site Request Forgery). Ce type de faille consiste à faire exécuter à une victime une requête HTTP à son insu. Sans le savoir, la victime devient complice car elle exécute les actions d'une page avec des privilèges. Par exemple, sur le site du garage, seuls les utilisateurs ayant le rôle employé ou administrateur ont la possibilité de supprimer une voiture sur les annonces. Un utilisateur visitant le site ne pourra donc pas accéder à la requête HTTP associée à la suppression. Si un visiteur malveillant décide d'utiliser un CSRF pour rediriger un utilisateur employé vers la page associée à la suppression, le code sera exécuté et la voiture supprimée.

La faille XSS

Une autre vulnérabilité est **la faille XSS** (Cross Site Scripting) accompagnée du slogan « Never Trust User Input ». Cette faille consiste à injecter du code interprétable par le navigateur web (comme un script JavaScript ou du HTML). Elle intervient en général au niveau des formulaires, lorsque l'utilisateur renseigne les champs demandés et qu'il soumet les données.

L'injection SQL

Une autre faille répandue est **l'injection SQL**. C'est un type d'attaque dans laquelle un utilisateur malveillant injecte du code SQL sur le site web et le trompe pour le forcer à accéder à la base de données.



Comment se protéger ?

En utilisant Symfony et en appliquant les règles de sécurité, toutes les failles citées peuvent être évitées.

Pour se protéger des attaques CSRF, Symfony embarque par défaut un outil dans ses formulaires, le CsrfToken. À chaque fin de formulaire, un token aléatoire est stocké dans la session de l'utilisateur par le biais d'un input hidden et sera validé lorsque l'utilisateur enverra le formulaire. Il est impossible de valider le formulaire sans ce token. Par exemple, voici qui est généré un jeton CSRF pour sécuriser notre formulaire :

```
<input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">
```

Pour éviter qu'un utilisateur malveillant n'injecte du code dans notre application, il est important de filtrer les entrées. Pour sécuriser rapidement un formulaire, Symfony met à disposition des contraintes de validation (Symfony Validator).

De plus, Twig, le moteur de templates de Symfony, protège contre les failles XSS par défaut. Il le fait en échappant automatiquement toutes les variables sorties dans les templates. Cela signifie que le code injecté par un utilisateur malveillant ne sera pas exécuté comme du code HTML ou JavaScript, mais sera affiché en tant que texte brut.

Pour finir, contre l'injection SQL, j'ai utilisé Doctrine avec le « Querybuilder » et sa méthode « setParameter » qui permet d'exécuter des requêtes préparées.

Si malgré ces règles appliquées, un utilisateur réussit à entrer dans une base de données, alors il est indispensable que les mots de passe des utilisateurs du garage soient cryptés.

Pour crypter les mots de passe, j'ai utilisé la fonction « *hasher* » de l'interface « *UserPassworldHasherinterface* » lors de la validation du formulaire. Cette fonction utilise des algorithmes sécurisés pour empêcher la lecture des mots de passe en clair.

F. LES TESTS UNITAIRES AVEC PHPUNIT

Afin de m'assurer que mon code fonctionne, j'ai effectué des tests à plusieurs endroits de l'application, notamment sur les contraintes de validation et sur les données enregistrées en base de données.

Un exemple de test unitaire :

Pour vérifier la fonctionnalité permettant la création d'un employé, j'ai créé la classe *UserTest* qui étend de la classe *TestCase.*

tests > Entity > UserTest.php > UserTest > testUserCreation

```php
<?php

namespace App\Tests\Entity;

use App\Entity\User;
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase
{
    public function testUserCreation(): void
    {
        $user = new User();
        $user->setEmail('john.doe@gmail.com')
            ->setPassword('password123')
            ->setLastname('Doe')
            ->setFirstname('John');

        $this->assertSame('john.doe@gmail.com', $user->getEmail());
        $this->assertSame('password123', $user->getPassword());
        $this->assertSame('Doe', $user->getLastname());
        $this->assertSame('John', $user->getFirstname());
    }

    public function testUserRoles(): void
    {
        $user = new User();
        $this->assertContains('ROLE_ADMIN', $user->getRoles());

        $user->setRoles(['ROLE_USER']);
        $this->assertContains('ROLE_USER', $user->getRoles());
        $this->assertContains('ROLE_ADMIN', $user->getRoles());
    }

    public function testUserIdentifier(): void
    {
        $user = new User();
        $user->setEmail('john.doe@gmail.com');

        $this->assertSame('john.doe@gmail.com', $user->getUserIdentifier());
    }


}
```

Explications :

« testUserCreation » vérifie la création d'un utilisateur avec des propriétés de base : email, mot de passe, nom de famille et prénom.
Les assertions ***(assertSame)*** vérifient que les valeurs définies sont correctement récupérées.
« testUserRoles » vérifie que chaque utilisateur a au moins le ROLE_ADMIN par défaut.
Il vérifie également la possibilité d'ajouter d'autres rôles et de garantir que le ROLE_ADMIN est toujours présent.
« testUserIdentifier » vérifie que l'identifiant utilisateur (qui est l'email) est correctement défini et récupéré.

J'ai également testé les erreurs, en renseignant volontairement des données non compatibles avec les contraintes (nom trop court, mail invalide, mot de passe invalide).

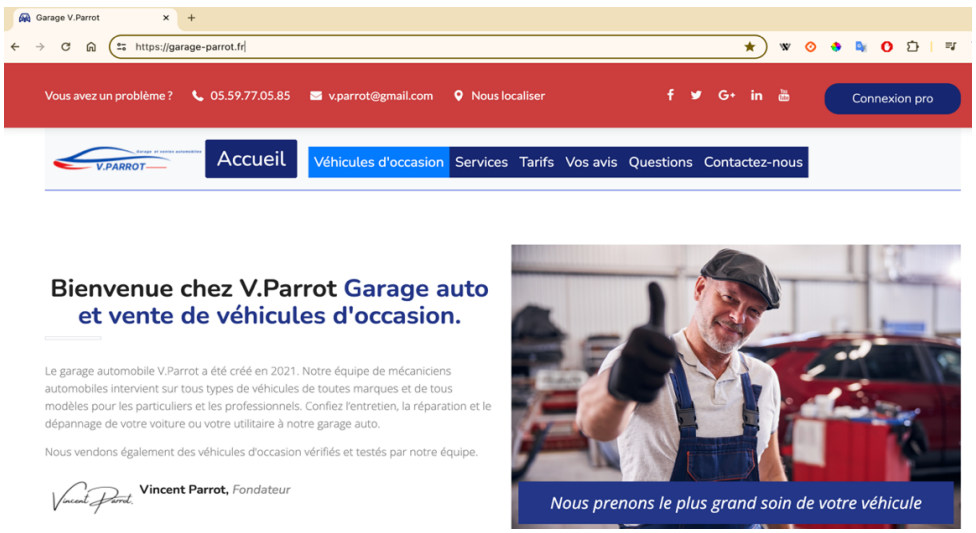
J'ai ensuite exécuté les tests avec la commande « *php bin/php unit* », ils sont tous passé avec succès.

```
chris@MBPdeChristophe GarageAuto % php bin/phpunit

PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
.....                                                               5 / 5 (100%)

Time: 00:00.301, Memory: 32.00 MB

OK (5 tests, 10 assertions)
chris@MBPdeChristophe GarageAuto %
```

G. LE DÉPLOIEMENT DU SITE EN LIGNE

Pour déployer mon application Symfony, j'ai choisi l'hébergeur Hostinger avec lequel je possède un abonnement. J'ai choisi le nom de domaine « garage-parrot.fr ».

Il a fallu passer par plusieurs étapes comme se connecter en FTP avec FileZilla et transférer le projet dans le dossier public_html (en excluant certains dossiers et fichiers comme tests, vendor, .git et .gitignore), se connecter en SSH pour installer et mettre à jour Composer, passer Symfony en mode production en modifiant le fichier .env, configurer la base de données en ajustant DATABASE_URL dans le fichier .env. Pour résoudre une erreur 403, j'ai créé des fichiers .htaccess pour indiquer à Apache où trouver le fichier index.php. Toutes ces étapes m'ont permises d'héberger le site, c'était la dernière étape de mon projet.



H - RECHERCHE ANGLOPHONE ET TRADUCTION FRANCAISE

Parmi les nombreuses recherches que j'ai effectuées pour avancer ce projet, en voici une qui m'a permis de me renseigner sur le hachage et la vérification des mots de passe sur le site officiel de Symfony :

Password Hashing and Verification

Edit this page

Most applications use passwords to login users. These passwords should be hashed to securely store them. Symfony's PasswordHasher component provides all utilities to safely hash and verify passwords.

Make sure it is installed by running:

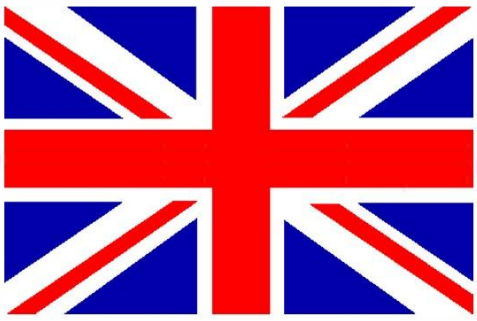
```
$ composer require symfony/password-hasher
```

Configuring a Password Hasher

Before hashing passwords, you must configure a hasher using the `password_hashers` option. You must configure the *hashing algorithm* and optionally some *algorithm options*:

YAML | XML | PHP | Standalone Use

```yaml
# config/packages/security.yaml
security:
    # ...

    password_hashers:
        # auto hasher with default options for the User class (and children)
        App\Entity\User: 'auto'

        # auto hasher with custom options for all PasswordAuthenticatedUserInterface insta
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
            algorithm: 'auto'
```



Voici une traduction que j'ai faite de cet article :

Hachage et vérification du mot de passe

La plupart des applications utilisent des mots de passe pour connecter les utilisateurs. Ces mots de passe peuvent être hachés pour les stocker en toute sécurité. Le composant PasswordHasher de Symfony fournit tous les utilitaires permettant de hacher et de vérifier les mots de passe en toute sécurité.

Assurez-vous qu'il est installé en exécutant la commande
composer require symfony/password-hasher
Configuration d'un hacheur de mot de passe
Avant de hacher les mots de passe, vous devez configurer un hachage en utilisant password_hashers option. Vous devez configurer l'algorithme de hachage et éventuellement certaines options de l'algorithme :

Conclusion

Ce projet de développement web m'a permis de mettre en pratique les compétences acquises tout au long de la formation. La réalisation de fonctionnalités complexes, la conception d'une interface utilisateur attrayante et la mise en œuvre de mesures de sécurité ont contribué à renforcer mes connaissances en développement.

Les défis rencontrés lors du projet ont été surmontés grâce à une approche méthodique et à une recherche approfondie. En effet, la contrainte de temps m'a poussé à prendre certaines décisions notamment l'utilisation du framework Symfony.

J'ai réalisé que la lecture de la documentation et de toutes les veilles associées dans le milieu du développement représentait une très grosse partie du travail de développeur.

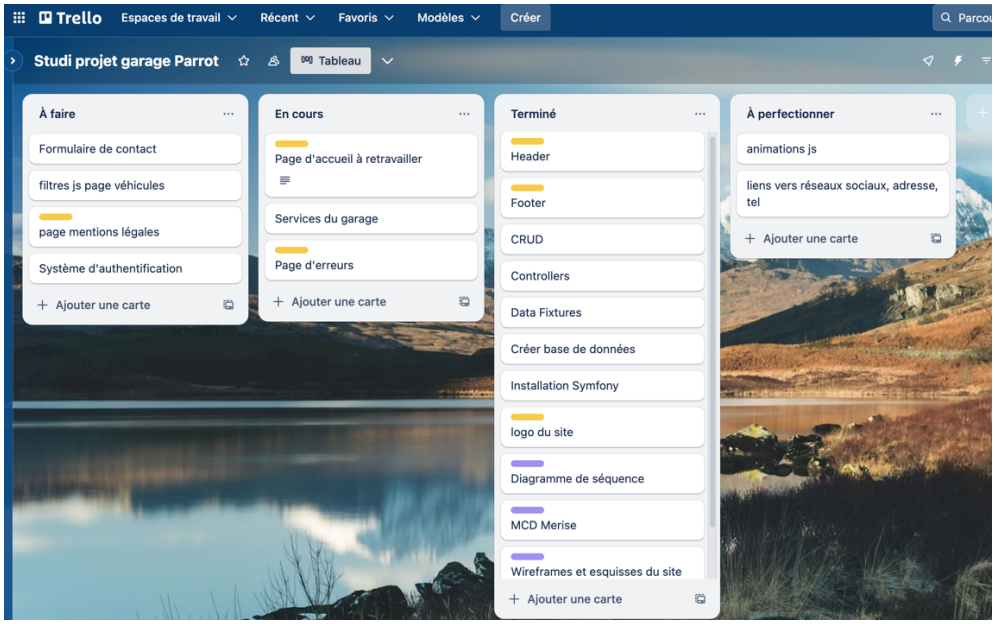
Ce dossier a offert un aperçu détaillé du processus de développement, des choix technologiques et des compétences appliquées, démontrant ainsi la réussite de ce projet de développement full stack pour le Garage V. Parrot.

Vendeur multimédia de métier, j'ai cherché à me former en tant que développeur web afin d'approfondir mes connaissances dans le milieu informatique et cette formation m'a appris énormément de facettes du métier que je n'aurais imaginé auparavant, un métier fascinant et en constante évolution. J'adore l'environnement de PHP avec SYMFONY, c'est une technologie vraiment très agréable avec des standards qui nous guident et qui nous aident surtout en tant que jeune développeur. Conscient que la route est encore très longue pour en apprendre d'avantage, je suis très motivé à continuer d'exercer dans le développement, j'ai hâte de pouvoir en faire mon métier et échanger avec des professionnels.

ANNEXES

Annexe 1 : Trello

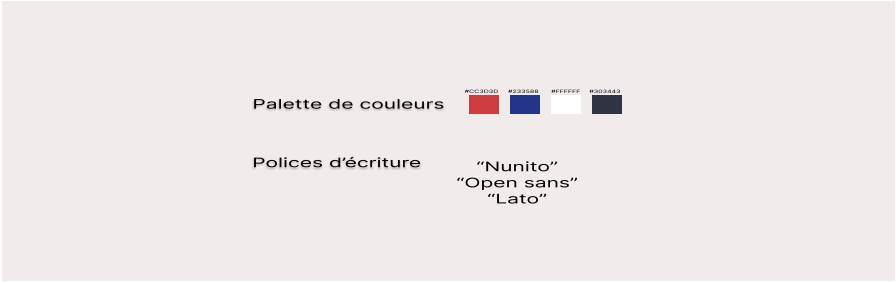
Aperçu de tableau de bord des missions à réaliser sur Trello :



Annexe 2 : Le logo

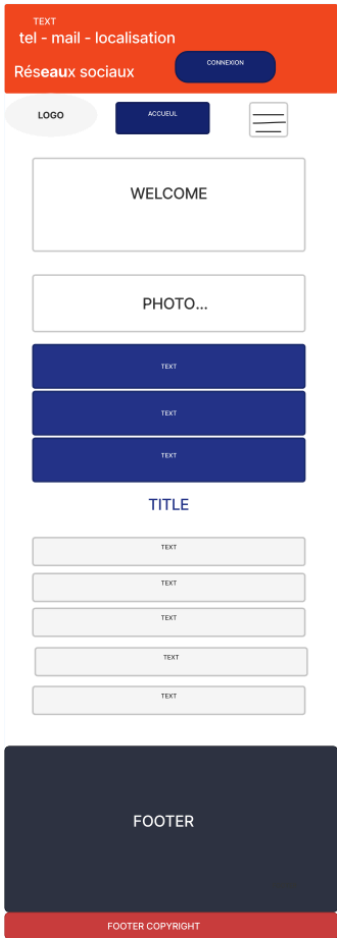
Logo de l'entreprise représentant une voiture sous les couleurs bleu, blanc, rouge (créé à l'aide de Canva) :

Annexe 3 : La charte graphique

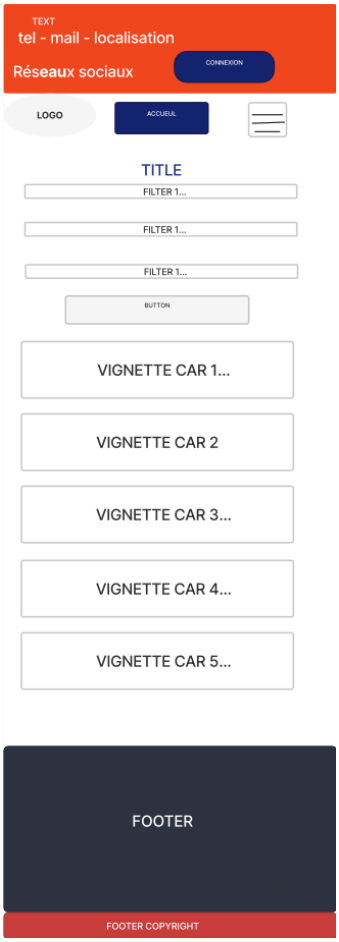


Annexe 4 : Les wireframes

Voici 3 maquettes du site version mobile (mobile firts) réalisées en amont



Page d'accueil sur mobile

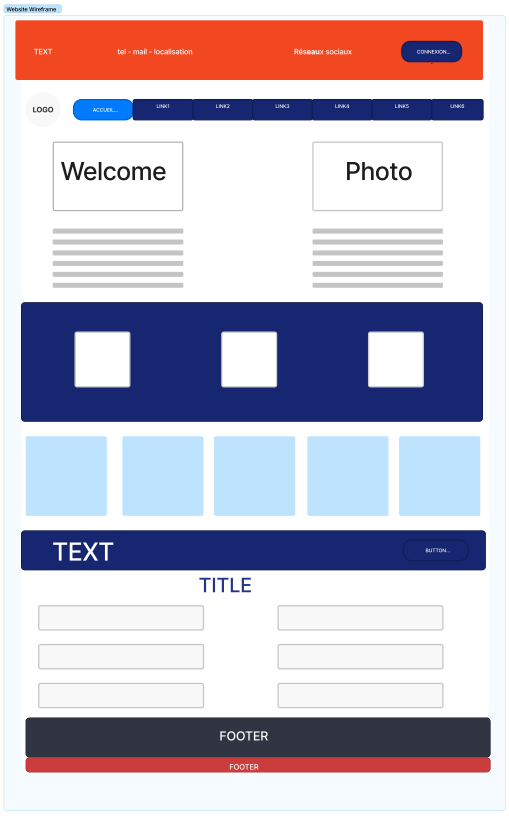


Page véhicules occasion sur mobile

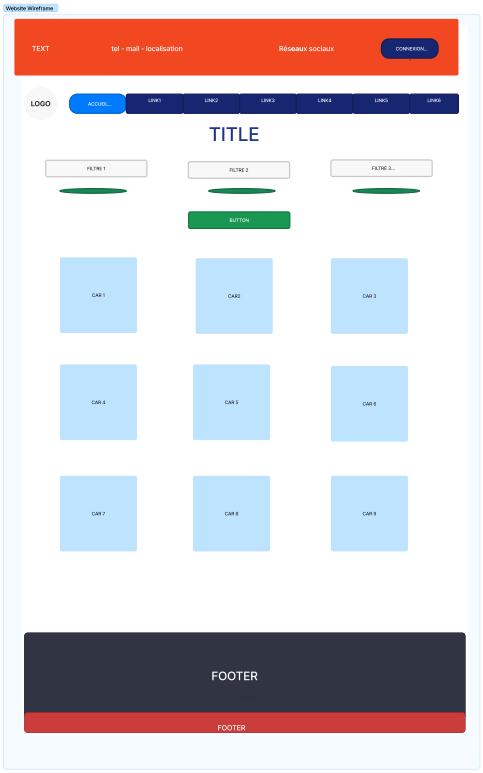
Page contact sur mobile

J'ai réalisé ces wireframes à l'aide de Figma.

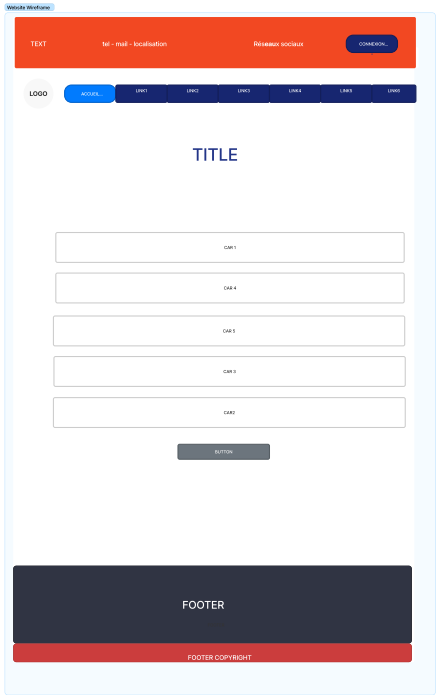
Voici 3 maquettes du site version Desktop réalisées en amont



Page d'accueil version Desktop



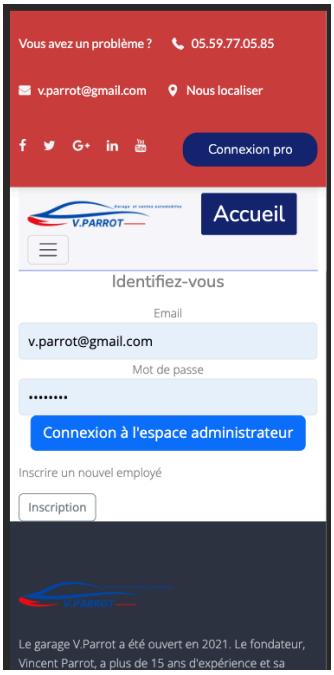
Page véhicules d'occasion version Desktop



Page contact version Desktop

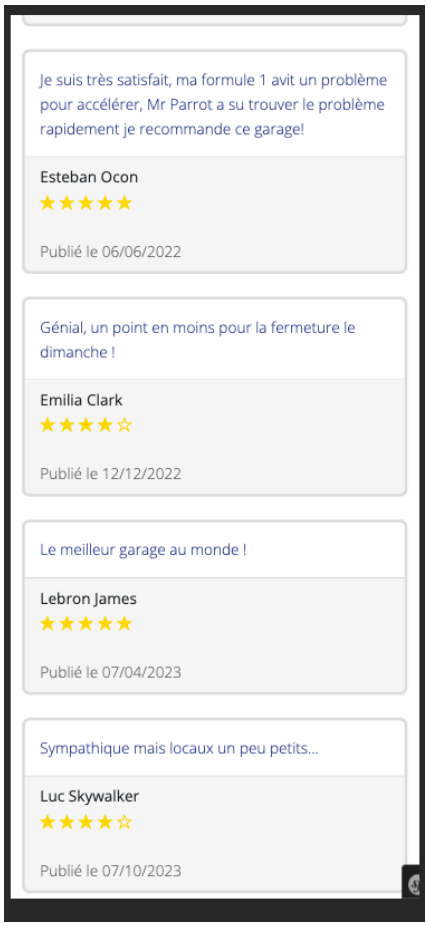
Annexe 5 : Écrans développés : ci-dessous quelques captures d'écran du rendu de l'application :

Page d'identification de l'utilisateur

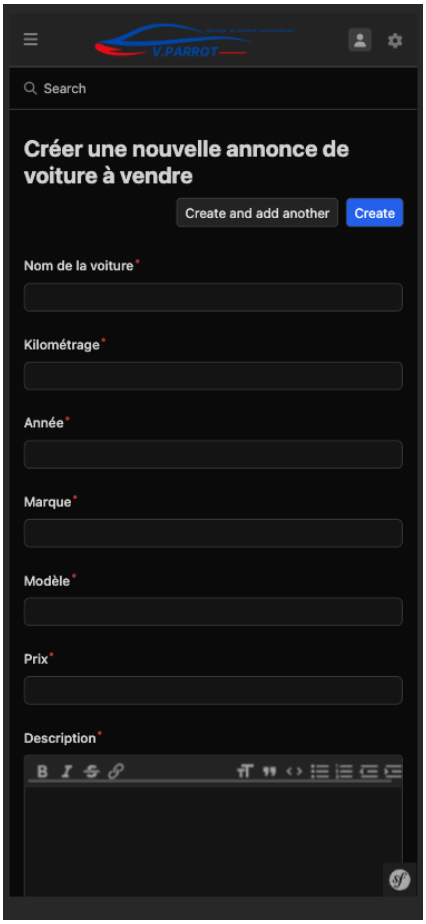


Formulaire de contact

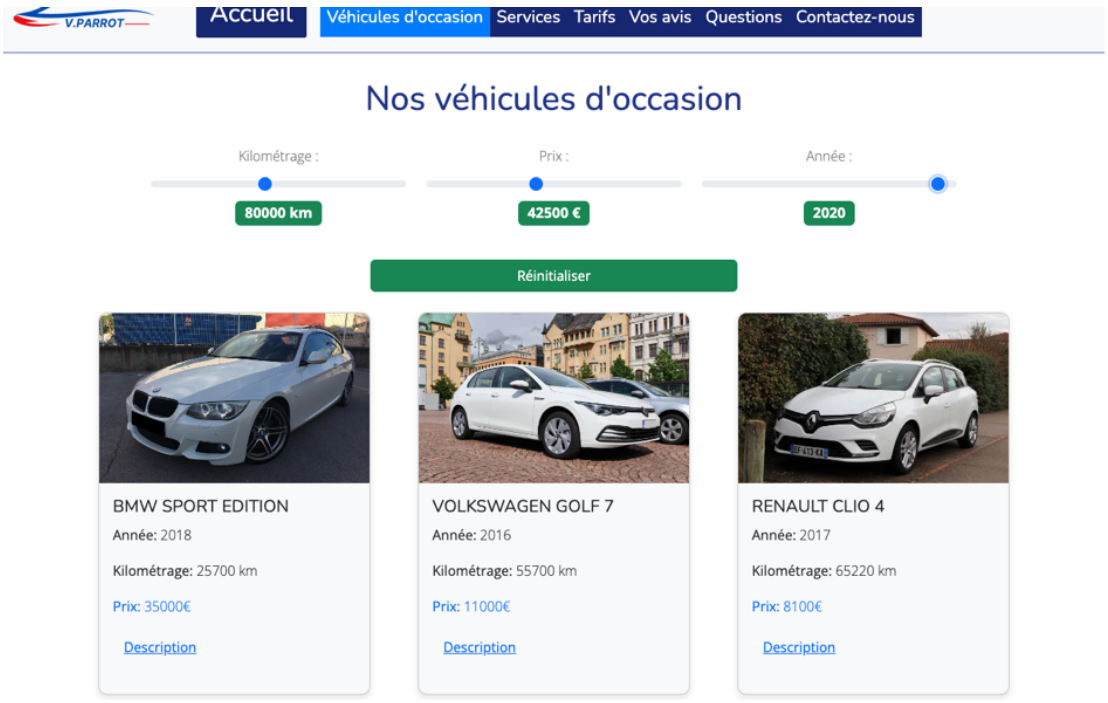
Les avis clients



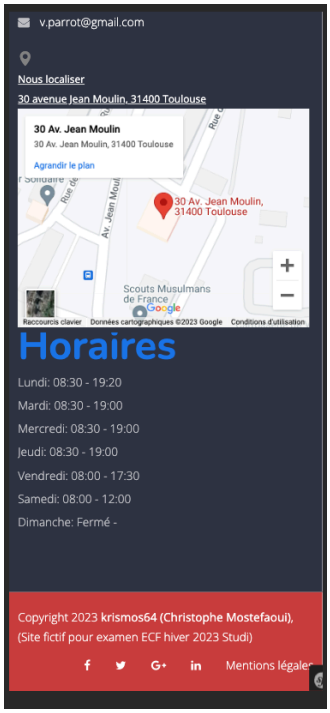
Le dashboard administrateur

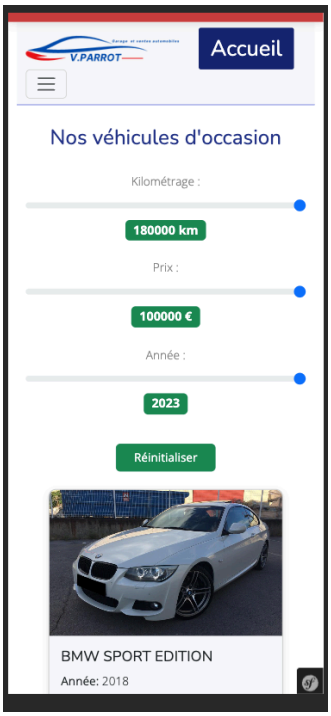


Page des véhicules d'occasion :



Le footer avec l'affichage des horaires d'ouverture



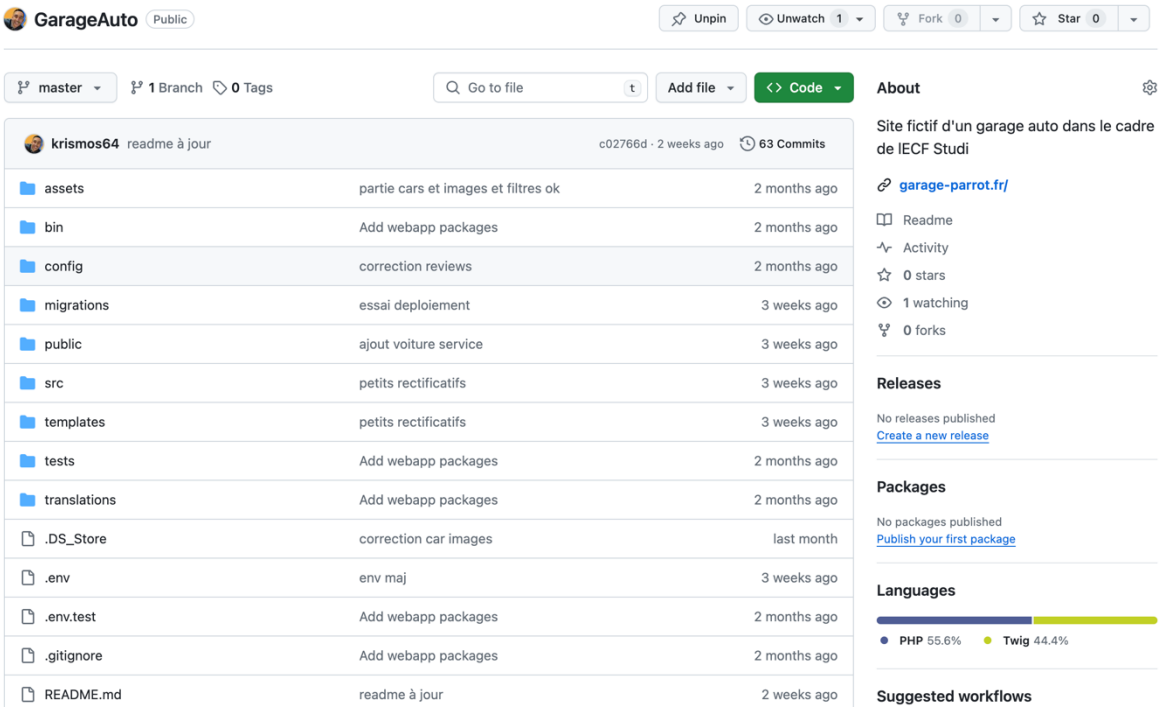


La page de filtre de recherche de véhicules

Page des avis clients

Annexe 6 : Github

Vue depuis mon Github



Annexe 7 : Mon ficher Readme

GARAGEAUTO

Projet Garage V. PARROT Il s'agit d'un projet pour un garage fictif proposant des services automobiles ainsi que la vente de véhicules d'occasion. L'administrateur (gérant) dispose d'une interface back-end lui permettant de gérer l'ensemble du site (création d'utilisateurs/employés, publication d'annonces pour les véhicules d'occasion, gestion des demandes clients, gestion des témoignages clients, etc.). Les employés ont également accès au back-end, mais avec des autorisations restreintes.

Voici le lien du site déployé : https://garage-parrot.fr/

Pour vous connecter en tant que SUPER_ADMIN, cliquez sur le bouton "connexion pro" en haut à droite de la top-bar. L'identifiant du patron est "v.parrot@gmail.com et le mot de passe est "password". Une fois connecté, vous aurez accès au bouton "espace pro" en haut à droite qui vous amènera à l'interface administrateur qui permet au gérant d'actualiser son site et aux employés également mais avec un accès limité.

Cloner le projet : Pour cloner le projet, exécutez la commande suivante :

(https://github.com/krismos64/GarageAuto.git)git clone

Prérequis : PHP 8.2 ou version supérieure

MySQL ou un autre serveur de base de données compatible avec Symfony, je recommande PhpMyAdmin

Un serveur Web (par exemple Apache ou Nginx), je recommande Xampp

Composer (gestionnaire de dépendances PHP)

Etapes :

Accédez au répertoire du projet :

« cd GarageAuto »

Installez les dépendances requises en utilisant Composer :

« composer install » « composer require webapp »

Modifier le fichier .env à la racine du projet:

Ouvrez le fichier '.env' et ajoutez les valeurs des variables d'environnement suivant votre configuration locale :

DATABASE_URL="mysql://nom_utilisateur:mot_de_passe@127.0.0.1:port/nom_du_projet?serverVersion=8&charset=utf8mb4"

Créez la base de données en exécutant la commande suivante :

« symfony console doctrine:database:create »

Créez les tables en utilisant les entités de votre application, attention vous aurez besoin de MakerBundle de Symfony pour exécuter ces commandes :

« symfony console make:migration « « symfony console doctrine:migrations:migrate »

Démarrez le serveur Web interne de Symfony en exécutant la commande suivante :

« symfony serve »

Votre application Symfony est maintenant déployée et accessible à l'adresse http://127.0.0.1:8000

Pour vous connecter en tant qu'admin sur le site, il vous faudra créer un utilisateur avec un 'ROLE_ADMIN' qui pourra gérer les fonctionnalités.

Rendez-vous en base de données via PhpMyAdmin

Lancez la requête SQL :

~ INSERT INTO user (id, email, roles, password) VALUES (NULL, 'v.parrot@gmail.com', '["ROLE_SUPER_ADMIN"]', 'vparrot')
Vous voilà prêt à exploiter le back-end du site, n'oubliez pas de hacher le mot de passe pour que celui-ci soit sécurisé.