Documentation technique

A. Technologies utilisées

Pour le back

- Symfony 6.3 avec PHP 8.2.6
- Doctrine bundle 2.10
- composer

Pour le front

- HTML 5 & Twig
- CSS 3 & Bootstrap & Bootswatch
- Javascript

Serveurs

- Heroku avec JAWS_DB pour le déploiement sur le internet
- Mysql 8.0.32
- Serveur de développement local intégré à PHP

Pourquoi ce choix ?

Pour des raisons de préférences personnelles, je voulais initialement réaliser ce projet avec React et une API écrite en PHP. Après avoir évalué la demande du client, j'ai beaucoup hésité quant au choix de la technologie à utiliser. J'ai dû peser le pour et le contre de chacune des options que j'avais à disposition. Mon choix s'est finalement orienté vers le framework symfony. Il y a plusieurs raisons principales à cela :

- La rapidité de création et de mise en place des éléments constitutifs de l'application, tant au niveau du front-end que du back-end. Cela est permis notamment par le maker bundle.
- La sécurité apportée par symfony, notamment au niveau du moteur de template HTML Twig, qui s'occupe (entre autres) de générer les tokens csrf dans les formulaires, permettant au développeur de se concentrer sur d'autres aspects du projet.
- Le référencement qui est un aspect important pour un site dit « vitrine ». En effet, l'objectif de cette application est d'offrir à Vincent Parrot une visibilité sur internet, afin qu'il puisse maintenir sa position face à la concurrence. Un site construit avec React aurait péché à ce niveau, de part le fonctionnement même de cette technologie. Il existe cependant des solutions utilisant le *server side rendering* pour retrouver le référencement avec React, mais je ne les maîtrise pas (pour l'instant).

Toutes ces raisons font de symfony un très bon choix pour répondre avec efficacité à la demande.

B. Diagramme de cas d'utilisation

disponible dans le dossier **Garage_V_Parrot/ECF_livrables/diagrammes**

C. Diagramme de séquence (*disponible dans **ECF_livrables/diagrammes**)*

Explications du diagramme :

1. Le visiteur clique sur le lien pointant vers la page des annonces auto
2. L'application requête la base de données pour récupérer les données des véhicules dans la limite de 5 résultats (valeur par choisie par défault), puis les renvois au visiteur dans une page html. Le visiteur aura alors un système de pagination pour parcourir tous les résultats
3. Le visiteur filtre les résultats selon ses critères de recherche. L'application requête la base de données de la même manière que précédemment mais avec des paramètres qui seront inclus dans la clause WHERE de la requête SQL. L'affichage est mis à jour sans rechargement de la page.
4. Le visiteur clique sur la voiture qui l'intéresse, faisant apparaitre dans une fenêtre modale un formulaire de contact pré-rempli avec les informations du véhicule.
5. Le visiteur remplit et soumet le formulaire. L'application le traite et l'enregistre en base de donnée. Une fois l'opération effectuée, elle notifie le visiteur de la bonne réception de son message.

D. Modèle conceptuel de données(*disponible dans **ECF_livrables/diagrammes**)*

Vous trouverez sur le MCD des bulles informatives apportant des précisions sur certains choix. Ce MCD a été fait avec Lucidchart. Le nombre de formes étant limité dans la version gratuite, voici des explications complémentaires :

- L'entité *Schedules* représente les horaires d'ouverture apparaissant dans le pied de chaque page. Une seule occurence de cette entité contient les horaires des sept jours de la semaine contenus dans l'attribut openedDays. Une occurence de l'entité *Garages* ne peut donc suivre qu'une seule occurence de l'entité *Schedules*, et une occurence de *Schedules* peut être liée à plusieurs garages. Dans le code, je fait en sorte que l'admin ne puissent pas créer/supprimer de nouvelle entité *Schedules*. En effet, je veux qu'il puisse seulement faire des modifications d'horaire sur une entité que j'aurais préalablement insérée en base de données. Cela induit donc que tous les garages crées suivront les mêmes horaires.
- Dans la plupart des relations entre les entités, j'ai fait le choix d'une valeur de cardinalité minimale égale à zéro. Ainsi l'administrateur a la possibilité de créer des services qui ne sont proposés par aucun garage, ou alors de créer des garages qui ne proposent pour le moment aucune voiture. Cela permet plus de souplesse à l'administrateur.
- Pour des raisons de simplification de l'exercice, j'impose qu'un service ne possède qu'une image au maximum, contrairement aux véhicules qui peuvent avoir plusieurs photos. Cela induit donc une relation 1:1 entre l'entité *Photos* et *Services.* C'est pourquoi cela se traduit dans le MLD par l'ajout d'un attribut imageName dans l'entité *Services.*
- L'entité *Users* représente l'admin (Vincent Parrot) et les employés, mais nullement les visiteurs qui n'ont pas la possibilité d'avoir de compte. Conformément à la demande client, seul Vincent Parrot a la possibilité de créer des comptes pour ses employés. Les visiteurs ne sont donc pas représentés dans ce MCD. On pourra envisager dans un second temps, le besoin de créer des comptes clients.

***E*. Modèle logique de données** (*disponible dans **ECF_livrables/diagrammes**)*

Voir les bulles informatives directement sur le MLD

F. Modèle physique de données (*disponible dans **ECF_livrables/diagrammes**)*

Voir directement sur le MPD