DOSSIER DE PROJET

Mélanie Bruzac

Développeur Web et Web mobile Full stack
Studi

Application web vitrine Garage V. Parrot

Studi
DIGITAL EDUCATION

Sommaire

Introduction

Je suis Mélanie Bruzac en formation de graduate développeur web et web mobile full stack et à côté je fais des heures de ménage. Ma formation a commencé en septembre 2022. Titulaire d'un Bachelor chargé d'affaires Europe-Asie, j'ai voulu changer de domaine pour aller vers le développement web.

J'ai choisi de me diriger vers ce domaine, car c'est un domaine hyper enrichissant qui est en permanente évolution et développement. Ce côté de toujours être en mouvement et d'apprendre de nouvelles choses en continu me plaît.

Étant en option marketing digital durant mon bachelor et ayant eu des échos à propos du développement web pendant ces cours de marketing, je me suis un peu plus intéressé en faisant des recherches de mon côté. J'ai donc commencé à suivre des vidéos sur le HTML, le CSS et le Javascript et à m'intéresser à la base de données pour comprendre un peu plus sur le fonctionnement d'un site internet. Cela m'a passionné, j'ai alors décidé de sauter le pas et de me diriger vers le monde du développeur web.

Ma formation se concrétise donc avec ce projet. Dans ce dossier, je vous présente le développement du site vitrine du Garage V. Parrot propose dans l'ECF. Pendant 4 mois, j'ai travaillé sur ce projet. Cela m'a permis de mettre en pratique mes compétences vues durant la formation et aussi de développer directement des nouvelles compétences.

I. Compétences du référentiel couvertes par le projet

1. **Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité**
    - Maquetter une application
    - Réaliser une interface utilisateur web statique et adaptable
    - Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

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

    - Créer une base de données
    - Développer les composants d'accès aux données
    - Développer la partie back-end d'une application web ou web mobile
    - Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

II. Résumé du projet

Le garage automobile V. Parrot est ouvert depuis 2021 à Toulouse. Son propriétaire Vincent Parrot, 15 ans d'expérience dans la réparation automobile, propose plusieurs services : réparation de carrosserie et mécanique, entretien et vente de voitures d'occasion. Par ailleurs, le garage n'a aucune visibilité sur internet. M. Parrot reconnaît qu'il serait bien utile d'avoir un site internet pour se faire une place parmi la concurrence.

Les fonctionnalités souhaitées par le propriétaire sont :

- un compte administrateur où il pourra gérer les informations, ainsi que créer les comptes de ses employés. Le compte employé sera capable d'ajouter de nouvelles voitures, de modérer et d'ajouter les témoignages. Ils pourront, administrateur et employés, se connecter avec le même formulaire de connexion en entrant une adresse e-mail et un mot de passe.
- une présentation claire et concise des différents services de réparation et d'entretien.
- une indication des horaires d'ouverture placée dans le pied de page.
- une présentation des voitures d'occasion à vendre avec des photos, une description détaillée et des informations techniques.
- un filtrage de la liste des véhicules d'occasion.
- une page contact avec le numéro de téléphone et un formulaire de contact. De plus, en dessous de chaque annonce, un lien sera disposé pour diriger le visiteur vers la page contact.
- une section de témoignages sur la page d'accueil où les visiteurs pourront laisser un témoignage composé d'une note, d'un nom, de la date de visite au garage et d'un commentaire.

Le site internet sera développé avec le framework Symfony, ainsi que du HTML, CSS et Bootstrap pour le front-end, du PHP et Mysql pour le back-end.

Le site internet sera composé :

- d'une page d'accueil avec une présentation du garage, suivie d'une présentation des services, puis une section avec un carrousel de témoignage.
- d'une page présentant la liste des véhicules en vente avec la possibilité d'appliquer des filtres pour faciliter la recherche. Les voitures seront présentées dans des cartes avec une photo, le nom, le prix, les km, l'année de circulation et un lien pour accéder à la fiche détails du véhicule.
- d'une page contact avec le numéro de téléphone, l'adresse et un formulaire de contact.

Le projet

I. Expression des besoins

Le garage V. Parrot n'a aucune visibilité sur internet. Pour se démarquer de ces concurrents, le propriétaire souhaite une application web vitrine qui reflète la philosophie du garage et qui permet une navigation et une communication efficace.

Plusieurs fonctionnalités ont été demandées.

- La possibilité de se connecter avec soit un compte administrateur, pour le propriétaire, soit un compte employé. Le compte administrateur peut gérer les informations sur le site et créer un compte employé. Le compte employé peut gérer les annonces et gérer les témoignages. Ils pourront se connecter avec une adresse e-mail et un mot de passe sécurisé.
- La présentation des différents services proposés.
- Les horaires d'ouverture présents sur le pied de page.
- La présence de la liste d'annonces de voitures d'occasions qui pourra être filtrée en fonction de la fourchette de prix, des km parcourus et de l'année de mise en circulation. Chaque annonce aura des images et une description détaillées.
- Information de contact par téléphone ou formulaire de contact pour contacter le garage à tout moment. Le formulaire doit être composé du nom, prénom, adresse mail, d'un message et de son sujet. Le sujet doit être automatique ajusté en fonction du titre de l'annonce. De ce fait, un lien vers le contact doit être mis sur chaque annonce.
- Le visiteur doit avoir la possibilité d'ajouter un témoignage. Les employés auront la possibilité de modérer et d'ajouter des témoignages.

II. Environnement

a. Humain

J'ai essentiellement travaillé seul avec l'aide du retour du formateur sur mon ECF, des lives Studi et des forums.

b. Technique

L'ordinateur portable utilisé pour le développement est sous Windows 11.
J'ai utilisé différentes technologie pour réaliser ce projet :

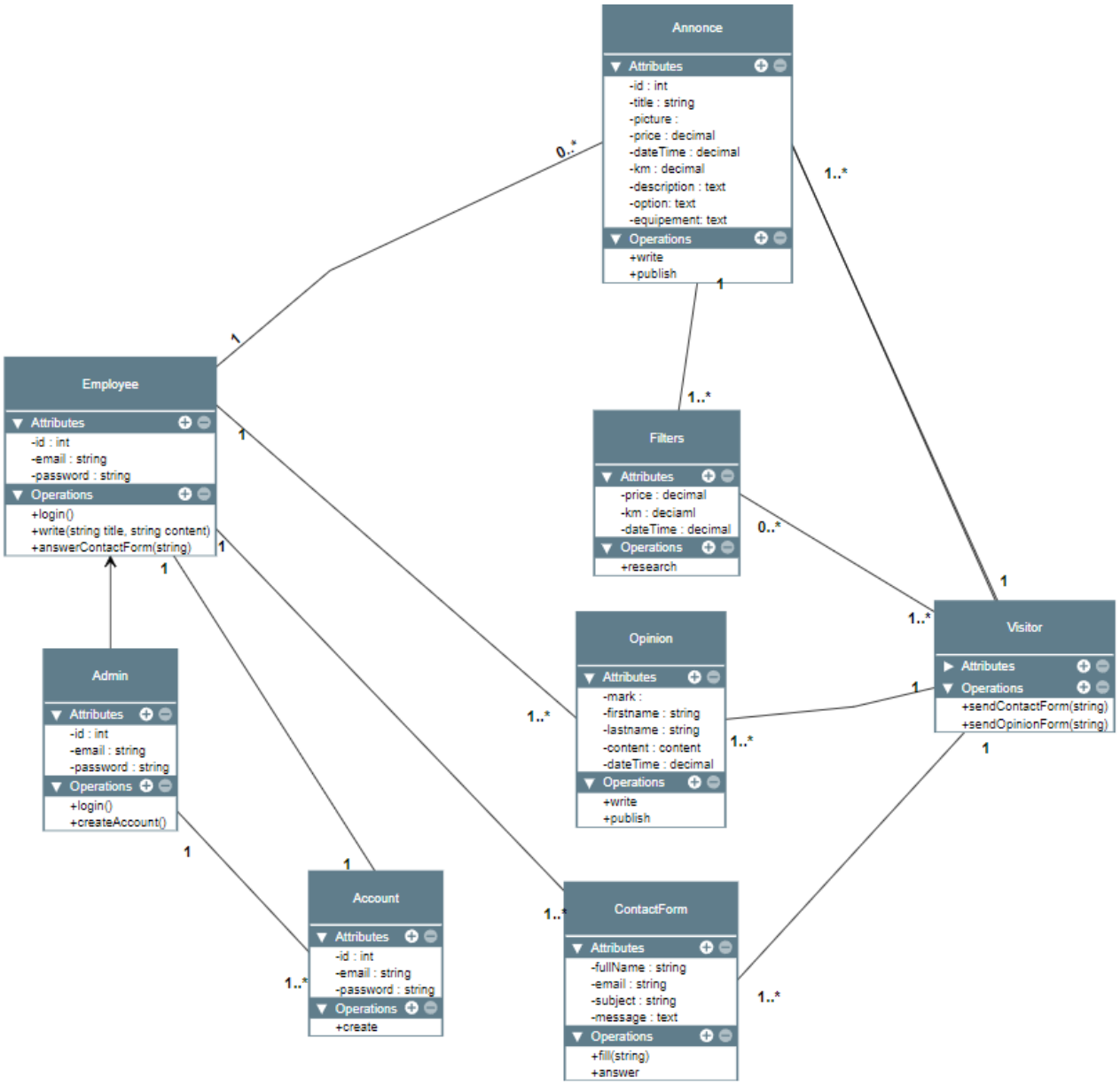
- **Environnement de travail** :
    - Visual Studio code permet de développer des projets.
    - Git permettant de sauvegarder différentes versions du projet et de les comparer.
    - GitHub permettant de déposer le code.
    - Xampp : serveur local
    - Heroku : développement en ligne

- **Partie Front** :
    - Bootstrap 5.3.0 est un kit d'outils qui facilite la création du design des applications web.
    - HTML 5
    - CSS 3

- **Partie Back** :
    - Symfony CLI 5.5.8 est un framework PHP qui facilite le développement.
    - Mysql
    - PHP 8.2.4

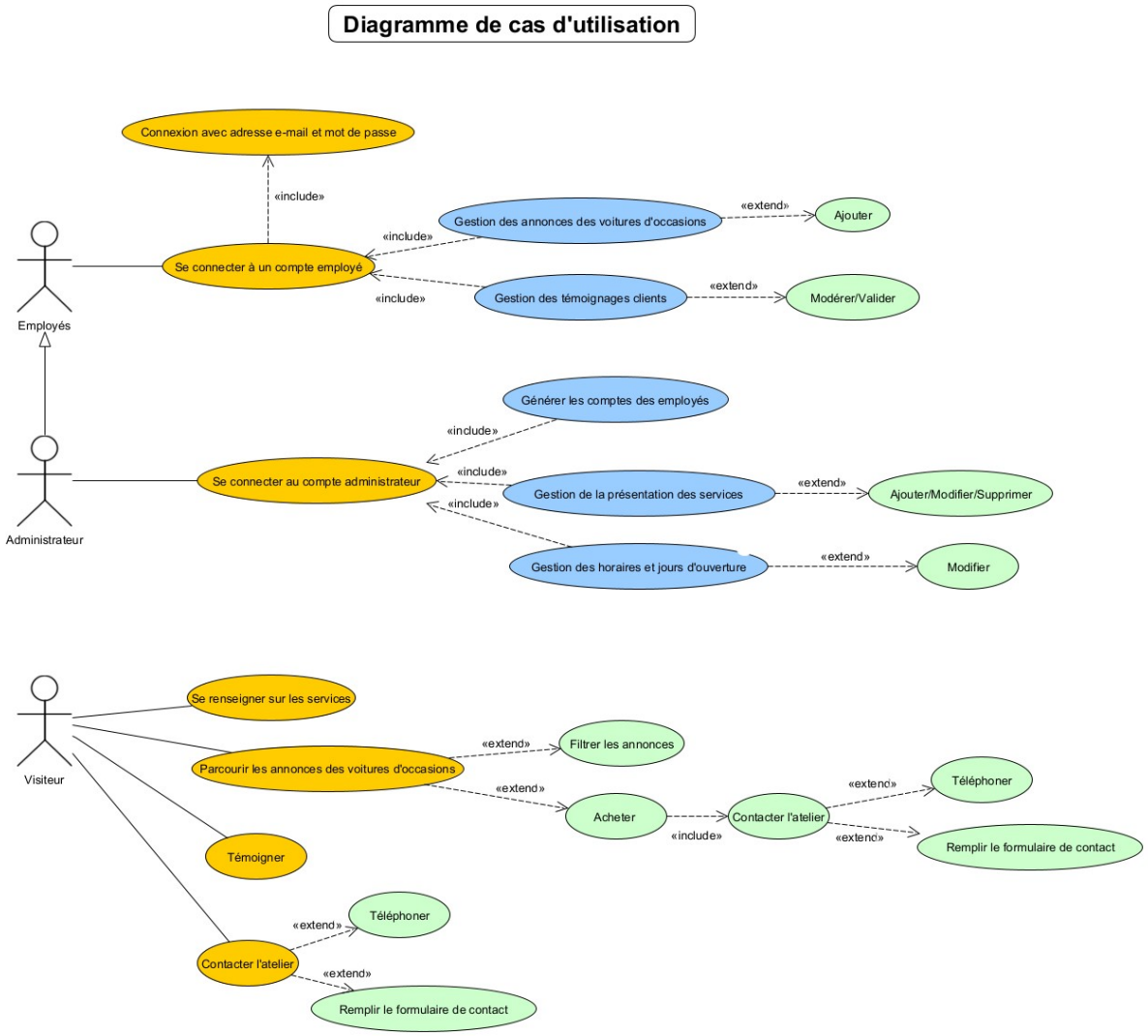
III. Analyse

Dans un premier temps, j'ai élaboré le diagramme de classe qui me permet de visualiser les différentes structures de données et les relations qu'il y a entre elles.



Nous pouvons voir qu'il y a 3 utilisateurs, l'admin, l'employé et le visiteur. L'admin peut créer un ou plusieurs comptes employés et il hérite aussi du rôle employé qui peut écrire et publier un ou plusieurs annonces, modérer les témoignages et répondre au demande de contact. Le visiteur quant à lui peut cliquer sur le détail des annonces, filtrer la liste d'annonce, ajouter un témoignage et envoyer un formulaire de contact.

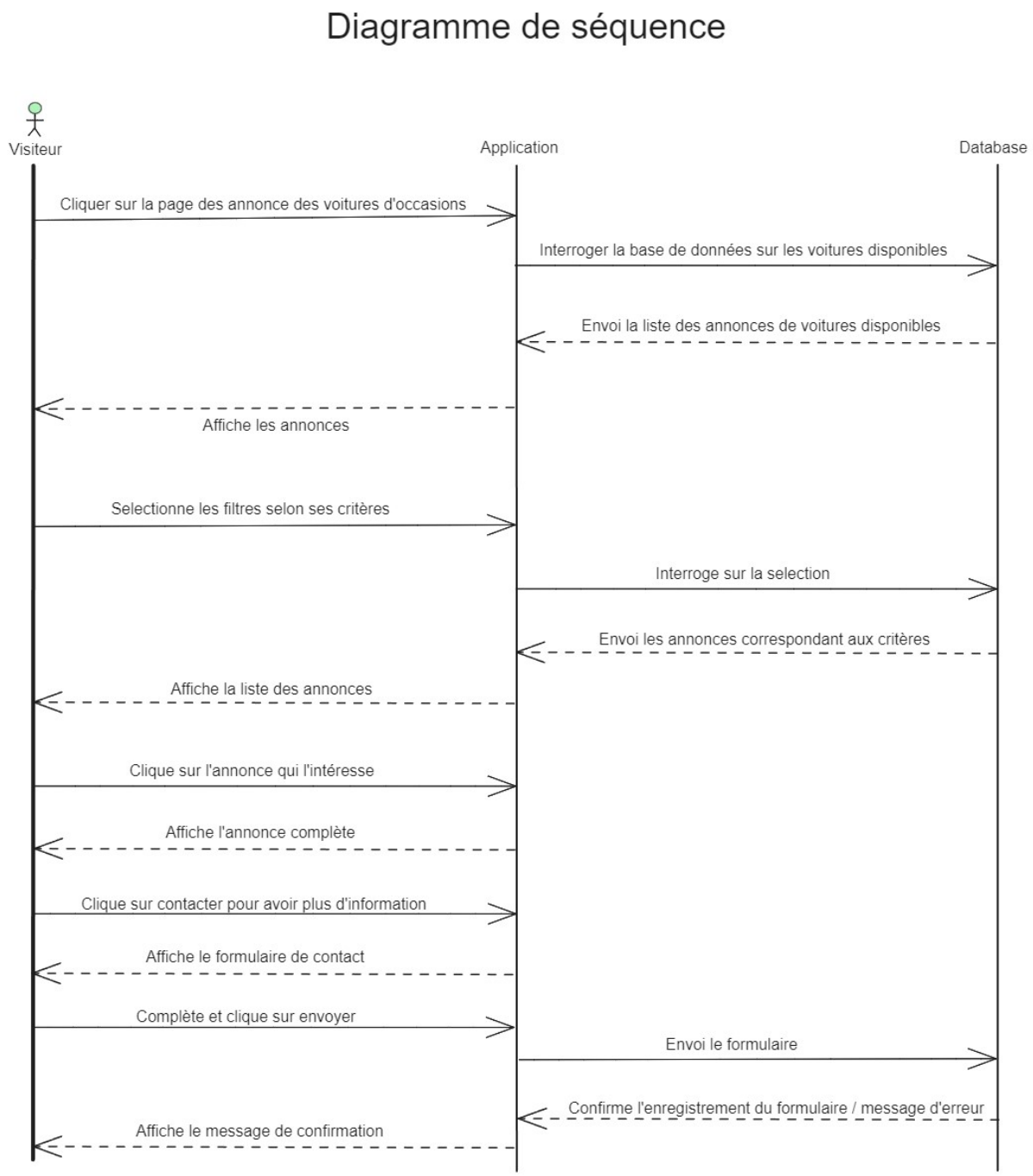
Ensuite j'ai réalisé un diagramme de cas d'utilisation pour pouvoir avoir une représentation des interactions que peuvent avoir les utilisateurs et les relations entre les fonctionnalités.



L'administrateur et les employés se connectent sur le même formulaire de connexion qui les dirige vers la même interface administrative mais pas tout à fait les mêmes fonctionnalités. Ces deux utilisateurs peuvent s'occuper de la gestion des annonces en les ajoutant, modifiant et supprimant, ainsi que la gestion des témoignages en les modérant et ajoutant. L'admin a en plus, la création de compte employé, la gestion de la présentation de la page d'accueil. Le visiteur peut se renseigner sur les services, parcourir les annonces avec l'aide des filtres et acheter une voiture en contactant le garage avec le formulaire de contact ou en téléphonant, ainsi que témoigner.

J'ai aussi réalisé un diagramme de séquence pour le chemin de navigation d'un nouveau visiteur qui découvre la liste des véhicules d'occasions, utilise les filtres avec ces critères

recherchés et remplit le formulaire pour avoir plus d'informations sur la voiture sélectionnée.



Pour finir, j'ai réalisé une maquette desktop et mobile ainsi qu'une charte graphique (annexe page).

Garage V. Parrot

Titre

Front : Quicksand bold

Text

Front : Roboto

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ceci est un bouton

Front : Roboto bold

Palette de couleurs :

Police d'écriture

Quicksand
Roboto

Charte graphique

La réalisation du projet

I. Conception de la base de données

Initialisation et configuration

Dans un premier temps j'ai installé Doctrine avec **`composer require --dev symfony/maker-bundle`**. Doctrine est un ensemble de librairies PHP pour travailler avec la base de données. Ensuite j'ai configuré la base de données dans .env.

```
DATABASE_URL="mysql://root:KriJoMaMaMeLy56.@127.0.0.1:3306/app?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
```

Extrait de .env

Dans un deuxième temps, j'ai créé la base de données avec **`php bin/console doctrine:database:create`**.

Création de la base de donnée

Ensuite j'ai créé des entités, ce sujet est développé en partie II. Puis j'ai fait php bin/console make:migration pour créer une migration qui contient des requêtes SQL puis php bin/console doctrine:migrations:migrate pour pousser les données vers la base de données. Voici une requête SQL de la création de l'entité annonce user :

```
CREATE TABLE user
id INT AUTO_INCREMENT NOT NULL,
email VARCHAR(255) NOT NULL,
UNIQUE INDEX UNIQ_1483A5E9E7927C74 (email),
password VARCHAR(255) NOT NULL,
roles LONGTEXT NOT NULL COMMENT \'(DC2Type:json)\',
PRIMARY KEY(id)) DEFAULT CHARACTER
SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
```

Et voici la requête qui a été fait quand j'ai ajouter les propriétés prénom et nom dans l'entité

```
ALTER TABLE user ADD prenom VARCHAR(255) NOT NULL, ADD nom
VARCHAR(255) NOT NULL
```

Jeu de fausses données

Pour pouvoir tester l'application, j'ai utilisé un jeu de fausse donnée avec DoctrineFixturesBundle. Je l'ai donc installé avec **`composer require --dev orm-fixtures`** et écrit les fixtures (annexe page 38 et 39).

II. Développement du back-end

Initialisation du projet

Pour l'initialisation du projet je me suis aidé de la documentation du site officiel de Symfony (symfony.com). En amont j'ai installé PHP et composer.

Par la suite j'ai créé l'application via le terminal sur VSCode avec la commande **`symfony new garage_v_parrot --webapp`**.

Pour démarrer le serveur localement la commande **`symfony server`** est utilisée.

Initialisation de Git

Ensuite, pour que le projet soit versionné, j'ai utilisé GitHub. J'ai donc créer un dépôt sur GitHub, puis j'ai entré les commandes suivantes dans le terminal de VSCode pour le lier le projet avec le dépôt : **`git init`**

`git commit`

`git remote add`...

`git push`

Au fur et à mesure du projet j'ai fait plusieurs commits pour mettre à jour le dépôt.

Création des entités

Tout d'abord, J'ai créé les entités avec la commande **php bin/console make:entity**. Voici un extrait de la documentation de Symfony utilisée pour m'aider à créer une entité. Il provient du site officiel de Symfony.

Creating an Entity Class

Suppose you're building an application where products need to be displayed. Without even thinking about Doctrine or databases, you already know that you need a `Product` object to represent those products.

You can use the `make:entity` command to create this class and any fields you need. The command will ask you some questions - answer them like done below:

```
$ php bin/console make:entity

Class name of the entity to create or update:
> Product

```

Avec l'aide du diagramme de classe, j'ai conçu les entités en commençant par user (annexe p. 40) qui permet à l'administrateur et aux employés de se connecter à un compte. J'ai donc implémenté un id, un email, un nom et prénom, un rôle et un mot de passe, ainsi qu'une relation OneToMany avec l'entité annonce qui permet à l'utilisateur d'avoir plusieurs annonces.

```php
#[ORM\OneToMany(targetEntity: "App\Entity\Annonce", mappedBy: "user")]
2 references
private $annonce;
```

Extrait de user.php - indique sa relation avec annonce

J'ai continué avec l'entité annonce (annexe page 41) où j'ai instauré les propriétés id, titre, image, prix, année, km, carburant description, option et équipement, ainsi qu'une relation ManyToOne et un slug. Le slug permet de générer une URL sûre pour les entités via le titre de l'élément annonce. Dans mon projet, le slug me permet d'amener sur une fiche détails d'une voiture. Il est placé dans le lien "Détails" de ma page annonce.

```twig
<a class="btn text-center" href="{{ path('annoncedetails', {'slug': annonce.slug }) }}">Détails</a>
```

Extrait de _annonce.html.twig

Pour faciliter le téléchargement des images, j'ai utilisé VichUploaderBundle. J'ai donc utilisé **`composer require vich/uploader-bundle`** pour l'installer et j'ai créé dans config/package un fichier pour configurer le bundle. Dans ce fichier, j'ai inscrit la façon dont la base de données est gérée, avec ORM, le type qui est attribut et le nom du mapping pour localiser les images quand elles sont téléchargées. Dans ce projet, elles sont situées dans le dossier public.

```yaml
vich_uploader:
    db_driver: orm

    metadata:
        type: attribute

    mappings:
        annonce_images:
            uri_prefix: /images/annonces
            upload_destination: '%kernel.project_dir%/public/images/annonces'
            namer: Vich\UploaderBundle\Naming\SmartUniqueNamer

            inject_on_load: false
            delete_on_update: false
            delete_on_remove: true
```

Extrait de vich_uploader.yaml

Voici comment le bundle est implémenté dans l'entité annonce :

```php
#[Vich\UploadableField(mapping: 'annonce_images', fileNameProperty: 'imageName')]
2 references
private ?File $imageFile = null;

#[ORM\Column(nullable: true)]
```

```php
public function setImageFile(?File $imageFile = null): void
{
    $this->imageFile = $imageFile;

    if ($imageFile) {
        $this->updatedAt = new \DateTimeImmutable();
    }
}

0 references | 0 overrides
public function getImageFile(): ?File
{
    return $this->imageFile;
}

0 references | 0 overrides
public function setImageName(?string $imageName): void
{
    $this->imageName = $imageName;
}

0 references | 0 overrides
public function getImageName(): ?string
{
    return $this->imageName;
}
```

Extraits de annonce.php

J'ai fini avec l'entité contact (annexe page 42) avec les propriétés id, nom complet, email, objet, message et la date de création.

Création des contrôleurs et des routes

Pour pouvoir avoir accès aux différentes entités j'ai développé les contrôleurs suivent :

- HomeController qui a pour route ' / ' et qui mène à la page d'accueil base.html.twig.

```php
class HomeController extends AbstractController
{
    #[Route('/', name: 'home')]
4 references | 0 overrides
public function index(): Response
    {


        return $this->render('base.html.twig', [
            'controller_name' => 'HomeController',

        ]);

    }
}
```

AnnonceController.php

-ContactController (annexe page 43) qui a pour route '/contact' et qui mène au formulaire de contact :

```php
    return $this->render('contact/index.html.twig', [
        'form' => $form->createView(),
    ]);
```

Elle envois le formulaire ContactType (annexe page 46) à la base de donnés :

```php
$contact = new Contact();
$form=$this-> createForm(ContactType::class, $contact);
$form->handleRequest($request);
if ($form->isSubmitted() && $form->isValid()) {
    $contact = $form->getData();
    $em->persist($contact);
    $em->flush();
```

Extrait de ContactController.php

L'Entity Manager permet d'interagir avec la base de données. Les informations sont soumises et vérifiées, puis elles sont sauvegardées et envoyées vers la base de données avec les méthodes persist et flush. Si le formulaire est bien passé un message apparaît :

```php
$this->addFlash(
    'success',
    'Votre message a été envoyé avec succès !'
);

return $this->redirectToRoute('contact');
```

Extrait de ContactController.php

Le formulaire est quant à lui amené à la vue sur la plage contact (index.html.twig) avec les fonctions twig. Form_label renvoi l'étiquette/le nom du champ et form_widget renvoi le widget.

```
<form>
    {{ form_start(form) }}
        <div class="form-group">
            <div > {{ form_label(form.fullName) }}  </div>
            <div type="text"> {{ form_widget(form.fullName) }} </div>
        </div>
        <div class="form-group">
            <div>{{ form_label(form.email) }}</div>
            <div  type="email"> {{ form_widget(form.email) }} </div>
        </div>
        <div class="form-group">
            <div>{{ form_label(form.subject) }} </div>
            <div type="text"> {{ form_widget(form.subject) }} </div>
        </div>
        <div class="form-group">
            <div for="message"> {{ form_label(form.message) }}  </div>
            <div >{{ form_widget(form.message) }} </div>
            <div class="mlForm">
                <p >Vos informations sont utilisées uniquement pour répondre à votre demande.
                    Plus d'informations sur le traitement des données, rendez-vous sur notre page
                    <a href="{{ path('mentions_legales')}}">mentions légales</a> .</p>
            </div>
        </div>
        <div> {{ form_end(form) }} </div>
</form>
```

Formulaire de contact sur la page contact

-AnnonceController (annexe page 44) qui a pour route '/voitures-doccasions' et qui mène vers la liste des annonces et un slug permettant d'aller sur le détail d'une annonce.

```
    return $this->render('annonce/index.html.twig', [
        'annonces'=>$annonces,
        'form' => $form->createView(),
    ]);

}

#[Route('/{slug}', name: 'details')]
10 references | 0 overrides
public function details(Annonce $annonce): Response
{
    return $this->render('annonce/details.html.twig', compact('annonce'));
}
```

Extrait d'AnnonceController

Dans ce controller, j'ai ajouté la méthode data pour le filtre. Elle permet d'amener les annonces avec les critères que le visiteur à demander :

```php
$data = new SearchData();
$form = $this->createForm(SearchForm::class, $data);

$form->handleRequest($request);
$annonces = $annonceRepository-> findBySearch($data);


return $this->render('annonce/index.html.twig', [
    'annonces'=>$annonces,
    'form' => $form->createView(),
]);
```

Extrait d'AnnonceController

Data est liée à SearchData (annexe page 44) qui implémente les propriétés du filtre. Les annonces sont, grâce à la fonction findBySearch, placées dans AnnonceRepository (annexe page 45)

-MentionsLegaleController :

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;


8 references | 0 implementations
class MentionsLegalesController extends AbstractController
{
    #[Route('/mentionsLégales', name: 'mentions_legales')]
    4 references | 0 overrides
    public function index(): Response
    {
        return $this->render('mentions_legales/index.html.twig', [
            'controller_name' => 'MentionsLegalesController',
        ]);
    }
}
```

-SecurityController qui a pour route '/connexion' et qui permet de se connecter avec un formulaire de connexion (annexe page 48) pour avoir accès à l'interface administrative.

```php
class SecurityController extends AbstractController
{
    #[Route(path: '/connexion', name: 'login')]
    12 references | 0 overrides
    public function login(AuthenticationUtils $authenticationUtils) : Response
    {
        // Erreur de connection si il en a
        $error = $authenticationUtils->getLastAuthenticationError();
        // le dernier identifient utilisé par l'utilisateur
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security\login.html.twig', [
            'last_username' =>$lastUsername,
            'error' => $error,
        ]);
    }

    #[Route(path: '/deconnexion', name: 'logout')]
    4 references | 0 overrides
    public function logout(): void
    {
        throw new \LogicException('This method can be blank - it will be intercepted
        by the logout key on your firewall.');
    }
```

SecurityController.php

J'ai configuré security.yaml pour que les utilisateurs puissent se connecter et se déconnecter à leur compte et pour qu' après la connexion l'utilisateur soit dirigé vers l'interface administrative grâce à **`default_target_path: admin`**.

```yaml
form_login :
    login_path: login
    check_path: login
    enable_csrf: true
    default_target_path: admin

logout:
    path: logout
```

Extrait du pare-feu dans security.yaml

Création d'une interface administrative avec EasyAdmin

J'ai décidé d'utiliser EasyAdmin pour faciliter le développement de l'interface administrative qui permet aux utilisateurs d'interagir avec la base de données. J'ai donc utilisé **`composer`**

require easycorp/easyadmin-bundle et créé le dashboard avec **php bin/console make:admin:dashboard** qui a pour route '/admin'.

```php
class DashboardController extends AbstractDashboardController
{
    #[Route('/admin', name: 'admin')]
    6 references | 0 overrides
    public function index(): Response
    {
        return $this->render('admin/dashboard.html.twig');
    }

    0 references | 0 overrides
    public function configureDashboard(): Dashboard
    {
        return Dashboard::new()
            ->setTitle('Garage V. Parrot');
    }

    0 references | 0 overrides
    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
        yield MenuItem::linkToCrud('Employés', 'fas fa-user', User::class)
                ->setPermission('ROLE_ADMIN');
        yield MenuItem::linkToCrud('Annonce', 'fas fa-edit', Annonce::class);
        yield MenuItem::linkToCrud('Demandes de contact', 'fas fa-envelope', Contact::class);
    }
}
```

Extrait du DashboardCrudController.php

J'ai sécurisé ce contrôleur en configurant dans security.yaml access_control pour limiter l'accès du dashboard à l'admin et aux employés.

```yaml
access_control:
    - { path: ^/admin, roles: ROLE_USER }
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

Extrait de security.yaml

J'ai développé trois CRUD contrôleurs :

-UserCrudController (annexe page) qui est lié à l'entité user, permet de gérer les comptes. Il n'y a que l'admin qui y a accès. J'ai pu restreindre l'accès avec set Permission ajouté à la configuration du menu comme ci-dessous :

```php
public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
    yield MenuItem::linkToCrud('Employés', 'fas fa-user', User::class)
            ->setPermission('ROLE_ADMIN');
    yield MenuItem::linkToCrud('Annonce', 'fas fa-edit', Annonce::class);
    yield MenuItem::linkToCrud('Demandes de contact', 'fas fa-envelope', Contact::class);
}
```

Extrait du DashboardCrudController.php

-AnnonceCrudController (annexe page) qui est lié à l'entité annonce. Il permet d'ajouter, modifier et supprimer une annonce.

-ContactCrudController (annexe page) qui est lié à l'entité contact. Il permet de voir les demandes de contact.

Sécurité

Pour simplifier et sécuriser la validation des formulaires j'ai mis en place le composant Validator avec **`composer req validator annotations`**. J'ai par la suite défini des règles de validation dans les entités. Par exemple avec la propriété email de user ou avec title de annonce :

```php
#[Assert\NotBlank(
    message: "L'eamil ne doit pas être vide !")]
#[Assert\Length(
    min: 2,
    max: 180,
    minMessage: "L'adresse email ne doit pas faire moins de 2 caractères",
    maxMessage: "L'adresse email ne doit pas faire plus de 180 caractères", )]
#[ORM\Column(length: 180, unique: true)]
#[Assert\Email()]
#[Assert\Length(min: 2, max: 180)]
3 references
private ?string $email = null;
```

Extrait de l'entité user

```php
#[ORM\Column()]
#[Assert\NotBlank(message: 'Le contenu ne doit pas être vide !')]
#[Assert\Length(
    min: 1, max: 150,
    minMessage : "Le titre ne doit pas faire moins de 1 caractères",
    maxMessage : "Le titre ne doit pas faire plus de 150 caractères"
    )]
3 references
private ?string $title = null;
```

Extrait de l'entité annonce

Cela permet de cadrer l'entrée des données sur les formulaires de création d'annonces et de demandes de contacts.

Ensuite, pour sécuriser le mot de passe, j'ai installé le hasher password avec **`composer require symfony/security-bundle`**. Ensuite j'ai appliqué la propriété plainPassword sans annotation pour pas qu'elle n'aille dans la base de donnés. J'ai importé

la logique du hasher password avec un Entity Listerner. Je l'ai ajouté à annonce avec une annotation comme ci-dessous :

```php
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[UniqueEntity("email", message: "L'email' est déjà pris...")]
#[ORM\EntityListeners(['App\EntityListener\UserListener'])]
36 references | 0 implementations
class User implements UserInterface, PasswordAuthenticatedUserInterface
```

Extrait d'annonce.php

Dans le EntityListerner, j'ai développé une fonction qui écoute les actions faites sur le user avec public function prePersist, ainsi qu'une fonction qui encode le mot de passe encodePassword.

```php
class UserListener
{
    2 references
    private UserPasswordHasherInterface $hasher;

    /** @param UserPasswordHasherInterface $hasher a service locator for listeners */
    4 references | 0 overrides
    public function __construct(UserPasswordHasherInterface $hasher)
    {
        $this->hasher = $hasher;
    }

    //écoute les actions faites sur le user
    0 references | 0 overrides
    public function prePersist(User $user)
    {
        $this->encodePassword($user);
    }

/**
 * Encode password based on plain password
 *
 * @param User $user
 * @return void
 */
//
    1 reference | 0 overrides
    public function encodePassword(User $user)
    {
        if($user->getPlainPassword() === null) { //vérifie si le plainPassword est vide
            return;  // si vide retroune en arrière
        }

        $user->setPassword( // si pas vide il set le password
            $this->hasher->hashPassword(
                $user,
                $user->getPlainPassword()
            )
        );

        $user->setPlainPassword(null); // pour être sur que rien ne fuite
    }
```

UserListener.php

Voici le résultat du hashage du mot de passe dans la base de données.

	id	email	password	roles (DC2Type:json)	prenom	nom
Éditer Copier Supprimer	45	admin@test.com	$2y$13$.SXbTBqbipYFon6kpyCuDOdNfmGFsS0UZtJmeWx8Taf...	["ROLE_USER","ROLE_ADMIN"]	Vincent	Parrot
Éditer Copier Supprimer	46	user@test.com	$2y$13$LvBi9NlgGkIlJvM6ztSHtuOfvBxWx0OX2k61Cl5SWin...	[]	Mallory	Pellerin
Éditer Copier Supprimer	48	user2@test.com	$2y$13$ZjcVkFzHCNKljA1gFbl3DuoLTgv23QeR/lbYgGwrFl8...	[]	Gwen	Morvan

Résumé des fonctionnalités

Les fonctionnalités que Vincent Parrot demande et qui sont développées dans cette partie sont :

-la connexion avec adresse email et mot de passe sur le compte admin et les comptes employés avec un formulaire de connexion unique.

-un compte admin qui peut créer, modifier et supprimer un compte employé et qui à les mêmes possibilités que le compte employé.

-un compte employé qui permet d'ajouter, modifier et supprimer les annonces

-le filtrage de la liste des voitures

-un formulaire de contact qui est visible sur les comptes.

II. Développement de la partie front-end

Initialisation du projet

J'ai choisi d'utiliser le bundle Webpack Encore pour gérer le CSS et le Javascript.

Auparavant j'ai installé Node.js pour les requêtes en JavaScript. J'ai donc fait un **`composer require symfony/webpack-encore-bundle`** pour installer Webpack et un **`npm install`**. J'ai configuré le fichier webpack.config.js comme ci-dessous :

```
const Encore = require("@symfony/webpack-encore");

// Manually configure the runtime environment if not already configured yet by the "encore" command.
// It's useful when you use tools that rely on webpack.config.js file.
if (!Encore.isRuntimeEnvironmentConfigured()) {
  Encore.configureRuntimeEnvironment(process.env.NODE_ENV || "dev");
}

Encore
  // directory where compiled assets will be stored
  .setOutputPath("public/build/")
  // public path used by the web server to access the output path
  .setPublicPath("/build")
  // only needed for CDN's or sub-directory deploy
  //.setManifestKeyPrefix('build/')

  /*
   * ENTRY CONFIG
   *
   * Each entry will result in one JavaScript file (e.g. app.js)
   * and one CSS file (e.g. app.css) if your JavaScript imports CSS.
   */
  .addEntry("app", "./assets/app.js")

  // enables the Symfony UX Stimulus bridge (used in assets/bootstrap.js)
  .enableStimulusBridge("./assets/controllers.json")

  // When enabled, Webpack "splits" your files into smaller pieces for greater optimization.
  .splitEntryChunks()

  // will require an extra script tag for runtime.js
  // but, you probably want this, unless you're building a single-page app
  .enableSingleRuntimeChunk()

  /*
   * FEATURE CONFIG
   *
   * Enable & configure other features below. For a full
   * list of features, see:
   * https://symfony.com/doc/current/frontend.html#adding-more-features
   */
  .cleanupOutputBeforeBuild()
  .enableBuildNotifications()
  .enableSourceMaps(!Encore.isProduction())
  // enables hashed filenames (e.g. app.abc123.css)
```

Extrait de webpack.config.js

Ensuite, j'ai mis en place les autres fichiers permettant au bundle de fonctionner : app.js qui est le point d'entrée et où j'ai inclus app.scss comme ci-dessous :

```
// any CSS you import will output into a single css file (app.css in this case)
import './styles/app.scss';

// start the Stimulus application
import './bootstrap';
```

Extrait de app.js

Cela permet le fonctionnement du CSS.

J'ai installé SASS avec la commande npm install node-sass sass-loader --save-dev et je l'ai activée dans webpack.config.js.

```
// enables Sass/SCSS support
.enableSassLoader()
```

J'ai aussi ajouté bootstrap.js initialisant Stimulus Bridge qui permet de faire fonctionner chaque librairie JavaScript avec Symfony. Ainsi que controllers.json qui s'intègre dans le système Stimulus.

```
// assets/bootstrap.js
import { startStimulusApp } from "@symfony/stimulus-bridge";

// Registers Stimulus controllers from controllers.json and in the controllers/ directory
export const app = startStimulusApp(
  require.context(
    "@symfony/stimulus-bridge/lazy-controller-loader!./controllers",
    true,
    /\.(j|t)sx?$/
  )
);
```

bootstrap.js

```
{
    "controllers": [],
    "entrypoints": []
}
```

controllers.json

Pour exécuter les mêmes commandes j'ai ajouté à package.json une section scripts.

```
"scripts": {
    "dev-server": "encore dev-server",
    "dev": "encore dev",
    "watch": "encore dev --watch",
    "build": "encore production --progress"
},
```

Extrait de package.json

Pour que les ajouts et les modifications faites dans le dossier assets soient pris en compte, j'ai compilé avec : **`npm run dev`** pour compiler une seule fois ou **`npm run watch`** pour compiler et re-compiler quand les fichiers changent.

J'ai bien sûr inclus dans le fichier base.html.twig le chemin vers le dossier assets.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %} Garage V. Parrot {% endblock %}</title>
        <link rel="icon" href="img/LogoNom.png">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

            <!-- Feuilles de styles-->
        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}
            <!-- Feuilles de styles-->
        {% block javascripts %}
            {{ encore_entry_script_tags('app') }}
        {% endblock %}
    </head>
```

Extrait de base.html.twig

J'ai choisi de développer à l'aide du framework Bootstrap qui permet un site responsive. J'ai donc intégré Bootstrap avec **`npm install bootstrap --save-dev`**. Ensuite je l'ai inclus dans app.scss.

```
@import "~bootstrap/scss/bootstrap";
```

Les importations dans app.scss

Une fois cette partie initialisée, j'ai pu commencer à développer.

Création des pages twig

J'ai tout d'abord créé les pages Twig en HTML. Les premières pages développées sont le header, le footer et la page d'accueil (voir annexe page 49). Pour baliser le header et le footer je me suis aidé d'un modèle sur le site Bootstrap.

Dans la page d'accueil j'ai inclus le header, le footer et la page d'accueil.

```
  <body>
{% include 'base/nav.html.twig' %}

        {% block body %}
{% include 'base/accueil.html.twig' %}
        {% endblock %}

{% include 'base/footer.html.twig' %}
  </body>
```

Extrait de base.html.twig

Ensuite j'ai développé la page annonce (annexe page) qui contient une description du service, un filtre et la liste des annonces. Les templates des deux derniers sont inclus avec {% include %} pour que la page soit plus facile à lire.

```
<section>
<div class="list container-fluid">
    <div class="row">

    <div id="filtre" class="col-md-3">
      {% include 'annonce/filter.html.twig' %}
    </div>

    <div id="annonces" class="col-8 offset-md-2 text-center">
      <div class="row row-cols-1 row-cols-md-2 g-4 gx-5 gy-5">
      {% include 'annonce/_annonce.html.twig' %}
      </div>
    </div>

  </div>
</div>
</section>
```

Extrait de annonce.html.twig

La liste des annonces est affichée grâce à la fonction{% for....in…. %}. Les informations sont affichées grâce à la fonction {{.....}}, exemple de {{annonce.title}} qui appelle la propriété titre d'annonce. Cette fonction amène les informations de l'entité dans la vue.

```twig
{% for annonce in annonces %}
<div class="col ">
    <div class="card">
        <img class="img-fluid" src="{{ vich_uploader_asset(annonce, 'imageFile') }} ">
        <div class="card-body">
            <h5>{{annonce.title}}</h5>
            <div class="card-content">
                <p>
                    Prix : {{annonce.price | number_format(0, ',', ' ') }}€ <br>
                    Km : {{annonce.km | number_format(0, ',', ' ') }} Km<br>
                    Année : {{annonce.dateTime}}
                </p>
                <a class="btn text-center" href="{{ path('annoncedetails', {'slug': annonce.slug }) }}">Détails</a>
            </div>
        </div>
    </div>
</div>
{% endfor %}
```

_annonce.html.twig

CSS responsive

Pour avoir une application responsive j'ai utilisé des composant Boostrap comme card pour les annonces ou navbar pour la barre de navigation.

```twig
<nav class="navbar navbar-expand-lg">
  <div class="container-fluid">

    <!--logo -->
    <a class="navbar-brand" href="#">
      <img src="/img/Logo_transparent.png" width="300" height="auto" alt="">
    </a>

    <!--Barre de navigation -->
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon bg-white"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
      <div class="navbar-nav">
        <a class="nav-link" href="{{ path('home')}}">Accueil</a>
        <a class="nav-link" href="{{ path('annonceindex')}}">Voitures d'occasions</a>
        <a class="nav-link" href="{{ path('contact')}}">Contact</a>
      </div>
    </div>

      <!--n°téléphone -->
        <div class="navbar-nav">
      <a class="nav-link"><img src="/img/telephone.png" width="25" height="25" alt=""></img>02 61 25 86 84</a>
        </div>
    </div>

  </div>
</nav>
```

Nav.html.twig

J'ai développé un fichier responsive.scss pour les pages s'adaptent aux écrans de portable :

```scss
/* mobile*/
@media (max-width: 767px) {
  h1 {
    font-size: 22px;
    margin-bottom: 30px;
  }
  .navbar-brand img {
    margin: 5px 0;
  }
  .nav-link body {
    padding: 60px 0px 60px 0px;
    font-size: 22px;
  }

  .nav-link body p {
    font-size: 20px;
  }
  .contact {
    padding: 40px 0px;
  }

}

/* small desktops */
@media (min-width: 992px) and (max-width: 1199px) {
    .nav-link .body .h2 {
        font-size: 40px;
    }

    .h3 a {
        font-size: 14px;
    }
}
```

responsive.scss

J'ai aussi modifié les éléments des pages avec de fichiers css que j'ai import dans app.scss comme ci-dessous :

```scss
@import "~bootstrap/scss/bootstrap";

@import './base/footer.scss';
@import './base/accueil.scss';
@import './base/nav.scss';
@import './avis.scss';
@import './mentionsLegales.scss';

@import './annonce/accueilAnnonce.scss';
@import './annonce/details.scss';


@import './contact.scss';
@import './login.scss';

@import './responsive.scss';
```

```scss
.btn{
    font-weight: bold;
    background: #055C9D;
    color : white;
    text-align: center;
}

body {
    font-family: Roboto, sans-serif;
    font-size: larger;
    background: white;
    color: black

}

h1, h2 {
    font-family: Quicksand, sans-serif;
    font-weight: bold;
    color: black;
    text-align: center;
    padding-bottom: 2%;
    padding-top: 2%;

}

h6 {
    font-weight: bold;
    color: #DCB233;
}

p {
    color: black;
    text-align: center;
```

Extrait d'app.scss montrant les paramètres css global

Sécurité

Pour la protection du front, j'ai installé une protection contre les CSRF avec **`composer require symfony/security-csrf`**. J'ai donc configuré framework.yaml pour permettre cette protection :

```yaml
framework:
    secret: '%env(APP_SECRET)%'
    csrf_protection: true
    http_method_override: false
    handle_all_throwables: true
```

Extrait de framework.yaml

J'ai par la suite installer la méthode dans le formulaire de contact, 'csrf_protection' et dans la page login.html.twig avec le méthode Post:

```php
public function configureOptions(OptionsResolver $resolver): void
{
    $resolver->setDefaults([
        'data_class' => Contact::class,
        'method' => 'GET',
        'csrf_protection' => true
    ]);
}
```

Extrait de ContactType.php

```twig
<form  action="{{ path('login') }}" method="post" name="login">
        <div class="form-group">
            <label for="username" class="form-label mt-4">Adresse
```

Extrait de login.html.twig

Résumé des fonctionnalités

Les fonctionnalités que Vincent Parrot demande et qui sont développées dans cette partie sont :

- La présentation des services
- La présence des horaires sur le pied de page
- L'exposition des annonces qui ont chacune un lien vers plus de détails.

Conclusion

Pour conclure, l'application est fonctionnelle avec la présence des pages demandées et une bonne partie des fonctionnalités désirées qui sont développées. Par ailleurs, du fait que le projet s'est fait sur une petite période, toutes les fonctionnalités n'ont pas été développées telles qu'une section témoignage ainsi que la possibilité pour l'administrateur de modifier les informations sur la page d'accueil et les horaires. C'est pourquoi je les ajouterai par la suite. Je rajoutais aussi une pagination pour que la navigation soit plus agréable et des cookies, ainsi qu'un reCaptcha sur le formulaire de contact pour plus de sécurité.

Des fonctionnalités sont aussi à améliorer :

- Les filtres : j'ajouterai du ajax pour que le changement de filtre soit plus rapide et agréable à utiliser. Je transformerai aussi le filtre en menu déroulant pour que ça prenne moins de place et je le placerai à droite pour les écrans mobiles.
- L'objet du formulaire de contact sera déjà défini quand on clique sur "nous contacter" via une annonce. Elle comportera le titre de l'annonce.
- La visibilité du mot de passe quand on se connecte ou quand on change de mot de passe.
- Faire répéter le mot de passe quand on le change.

Malgré pas mal d'améliorations à faire, ce projet m'a apporté beaucoup de choses, notamment la compréhension du fonctionnement du back-end dans une application et de développer les compétences vues pendant la formation. Il m'a permis de renforcer mon choix de devenir développeur web. Il a été un bon moyen de me plonger dans le milieu du développement web.

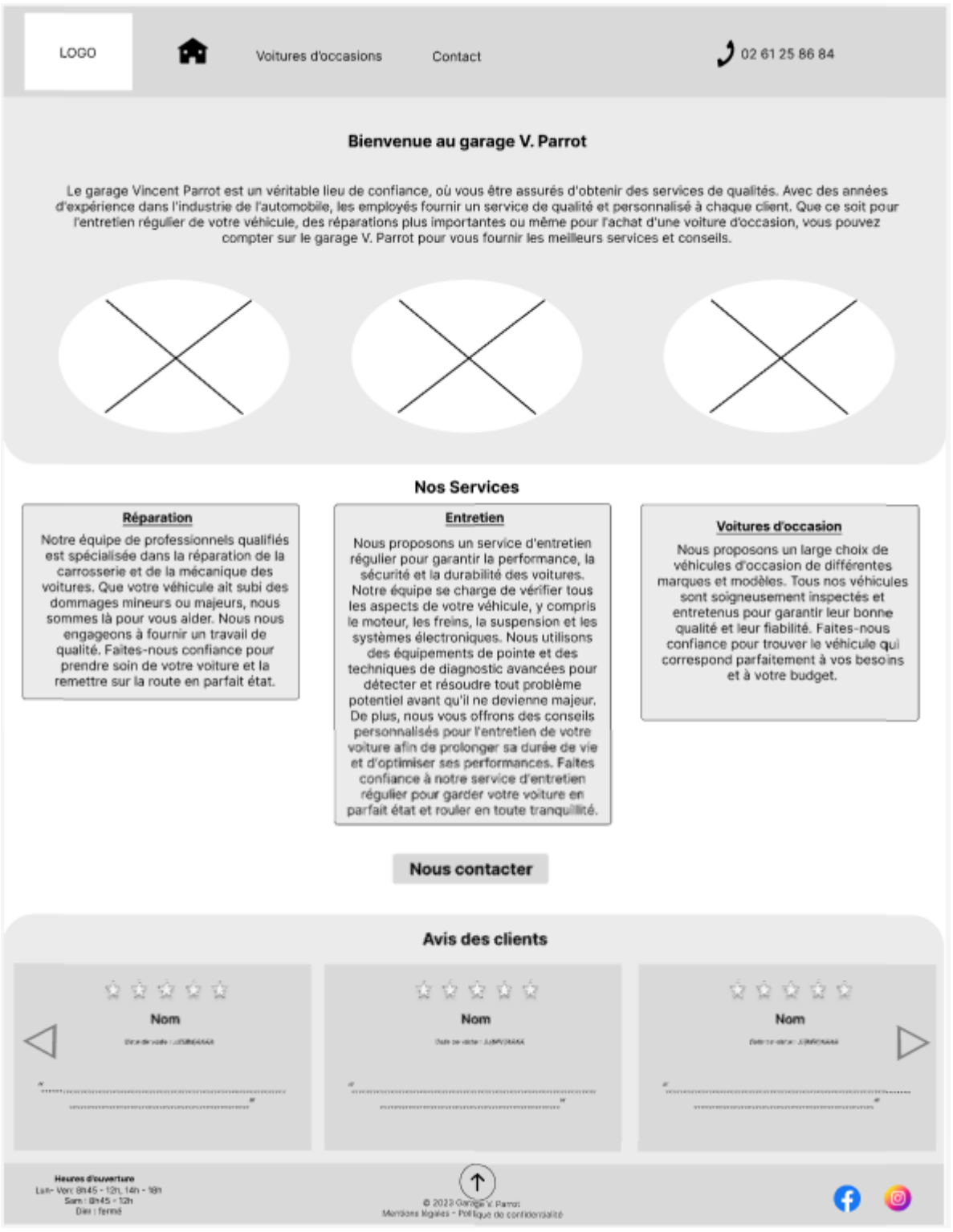
A la suite de cette formation, j'aimerais m'améliorer encore plus dans Symfony et apprendre React js et React Native. J'ai aussi l'opportunité de réaliser le site vitrine d'une école de natation de ma région. Cela me permettra de pratiquer avec Wordpress, HTML , CSS et JavaScript.

J'aimerais aussi faire un stage pour commencer en douceur ma vie professionnelle dans le développement web, ainsi que pour prendre un peu plus confiance en moi avant de postuler pour un poste de développeur fullstack en entreprise ou dans une agence.

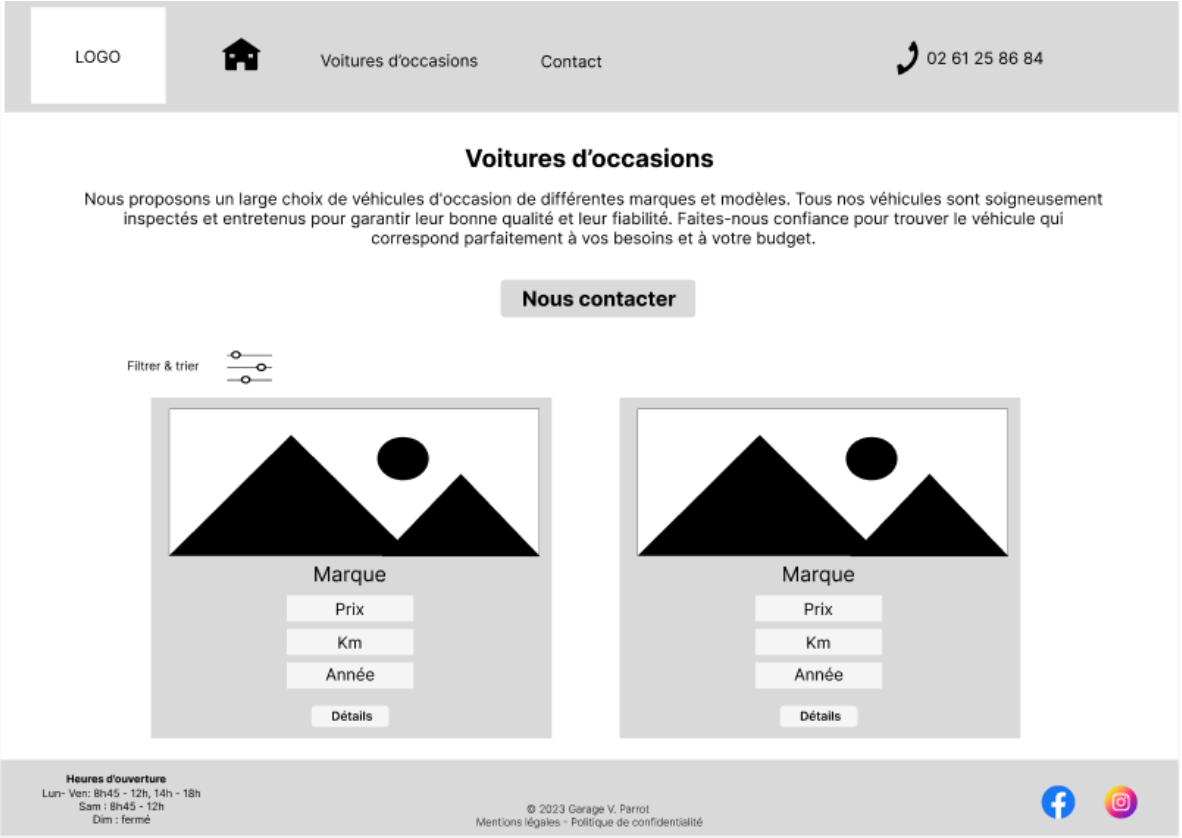
Annexes

Wireframes

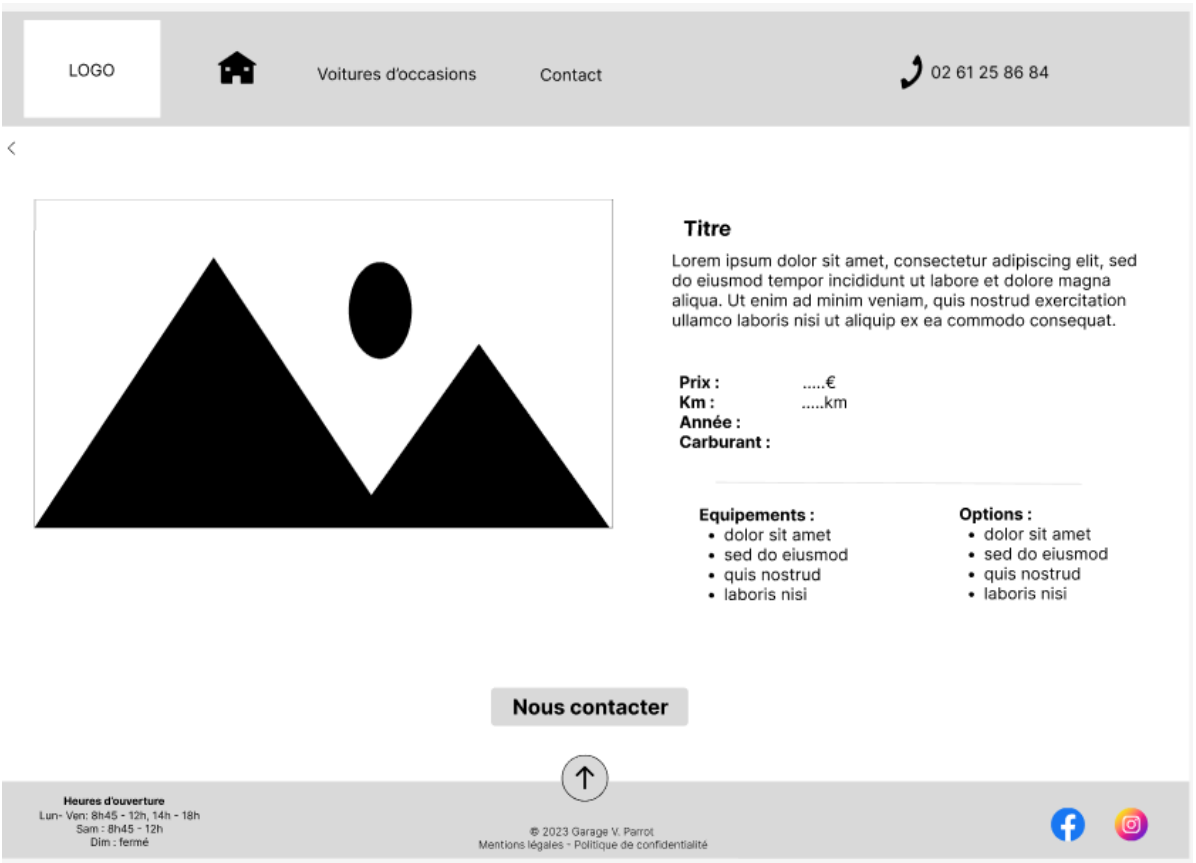
-Desktop



Page d'accueil



Page des voitures d'occasions avec la liste

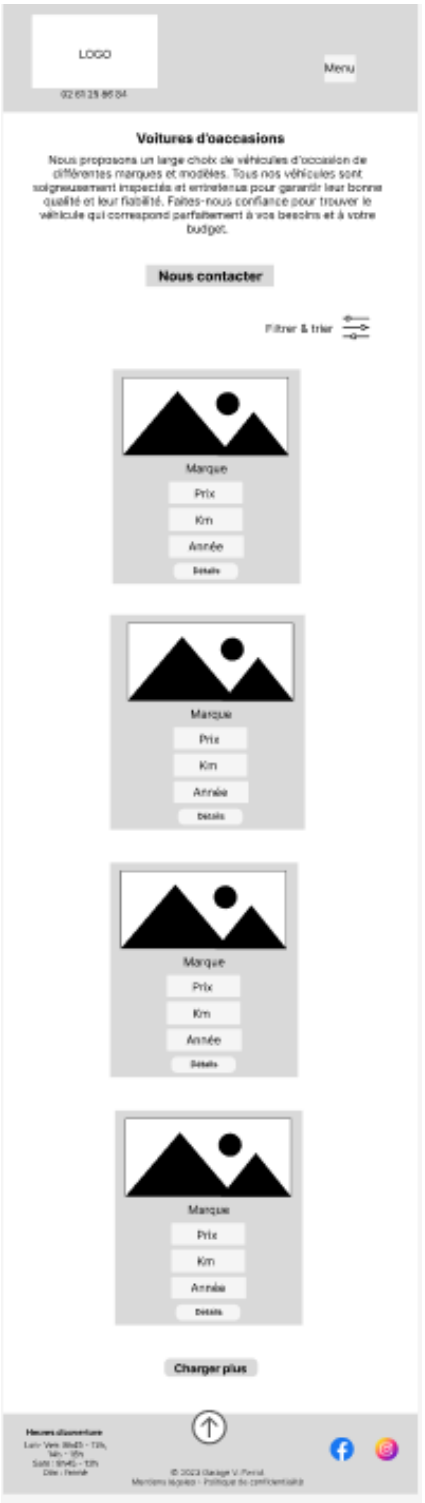


Page détails d'une annonce

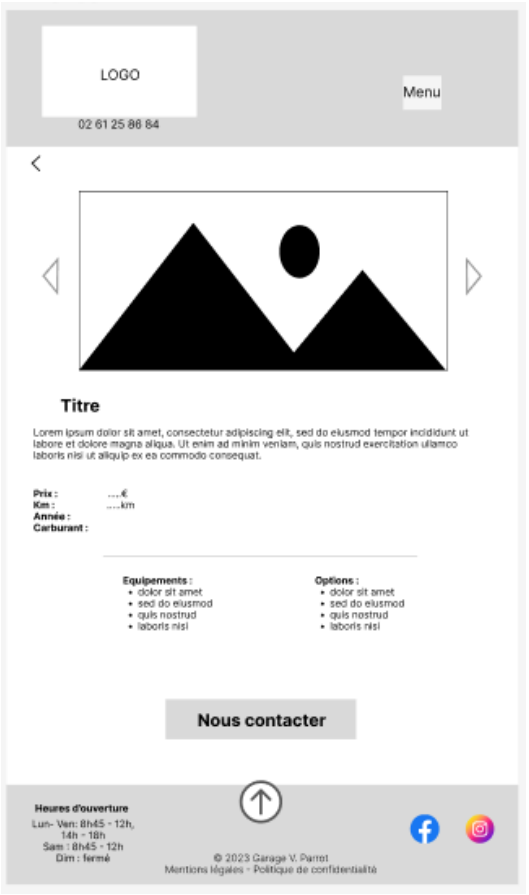
Page Contact

- Mobile

Page d'accueil



Page des voitures d'occasions avec la liste



Page détails d'une annonce

Page contact

Fixtures

```php
private Generator $faker;

2 references
protected $slugger;

3 references | 0 overrides
public function __construct(SluggerInterface $slugger)
{
    $this->slugger = $slugger;
    $this->faker = Factory::create('fr_FR');
}

0 references | 0 overrides
public function load(ObjectManager $manager): void
{

        $user = [];

        //Creation de l'admin
        $admin = new User();
        $admin->setEmail('admin@test.com')
            ->setPrenom('Vincent')
            ->setNom('Parrot')
            ->setRoles(['ROLE_USER', 'ROLE_ADMIN'])
            ->setPlainPassword('password');
        $user[] = $admin;
        $manager->persist($admin);


        //Creation de l'employé
    for ($i = 1; $i < 5; $i++) {
        $user = new User();
        $user->setEmail($this->faker->email())
            ->setPrenom($this->faker->firstName)
            ->setNom($this->faker->lastName)
            ->setRoles(['ROLE_USER'])
            ->setPlainPassword('password');

        $manager->persist($user);
```

Extrait de AppFixtures.php

```php
// Creation d'une annonce
for ($i=0; $i < 10; $i++) {
    $annonce = new Annonce();

    $annonce->setTitle($this->faker->words(2, true))
            ->setPrice($this->faker->randomFloat(100, 9999))
            ->setdateTime($this->faker->date('2000', '2022'))
            ->setKm($this->faker->randomFloat(2, 100, 9999))
            ->setCarburant($this->faker->words(2, true))
            ->setDescription($this->faker->text(500))
            ->setOption($this->faker->text(500))
            ->setEquipement($this->faker->text(500))
            ->setSlug(strtolower($this->slugger->slug($annonce->getTitle())));

    $manager->persist($annonce);
}

$manager->flush();
```

Extrait de AppFixtures.php

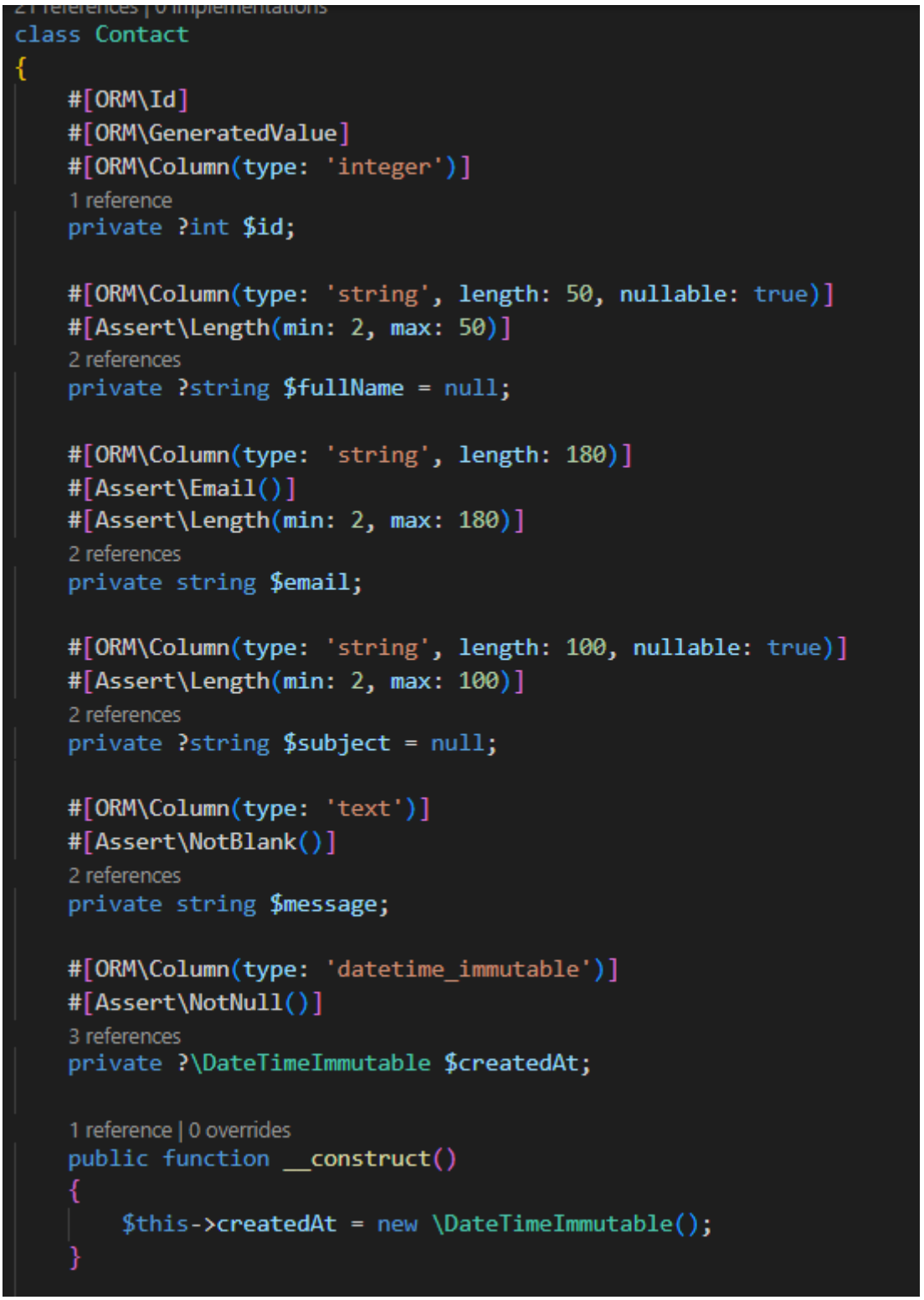
Entités

```php
class User implements UserInterface, PasswordAuthenticatedUserInterface
{

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    1 reference
    private ?int $id = null;

    #[Assert\NotBlank(
        message: "L'eamil ne doit pas être vide !")]
    #[Assert\Length(
        min: 2,
        max: 180,
        minMessage: "L'adresse email ne doit pas faire moins de 2 caractères",
        maxMessage: "L'adresse email ne doit pas faire plus de 180 caractères", )]
    #[ORM\Column(length: 180, unique: true)]
    #[Assert\Email()]
    #[Assert\Length(min: 2, max: 180)]
    3 references
    private ?string $email = null;

    #[ORM\Column(length: 255)]
    2 references
    private ?string $prenom = null;

    #[ORM\Column(length: 255)]
    2 references
    private ?string $nom = null;

    #[ORM\Column]
    #[Assert\NotNull()]
    2 references
    private array $roles = [];

    2 references
    private ?string $plainPassword = null;

    #[ORM\Column(type: 'string', length: 255)]
    #[Assert\NotBlank(
        message: "Le mot de passe ne peut pas être vide !",
    )]
    2 references
    private ?string $password = 'password';
```

user.php

```php
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    1 reference
    private ?int $id = null;


    #[ORM\Column()]
    #[Assert\NotBlank(message: 'Le contenu ne doit pas être vide !')]
    #[Assert\Length(
        min: 1, max: 150,
        minMessage : "Le titre ne doit pas faire moins de 1 caractères",
        maxMessage : "Le titre ne doit pas faire plus de 150 caractères"
        )]
    3 references
    private ?string $title = null;

    #[Vich\UploadableField(mapping: 'annonce_images', fileNameProperty: 'imageName')]
    2 references
    private ?File $imageFile = null;

    #[ORM\Column(nullable: true)]
    2 references
    private ?string $imageName = null;

    #[ORM\Column(nullable: true)]
    4 references
    private ?\DateTimeImmutable $updatedAt = null;

    #[ORM\Column(type: Types::DECIMAL, precision: 10, scale: '0')]
    #[Assert\NotBlank(message: 'Le contenu ne doit pas être vide !')]
    2 references
    private ?float $price = null;

    #[ORM\Column(type: 'string', nullable: true)]
    #[Assert\NotBlank(message: 'Le contenu ne doit pas être vide !')]
    2 references
    private $dateTime;

    #[ORM\Column(type: Types::DECIMAL, precision: 10, scale: '0')]
    #[Assert\NotBlank(message: 'Le contenu ne doit pas être vide !')]
    2 references
    private ?float $km = null;
```

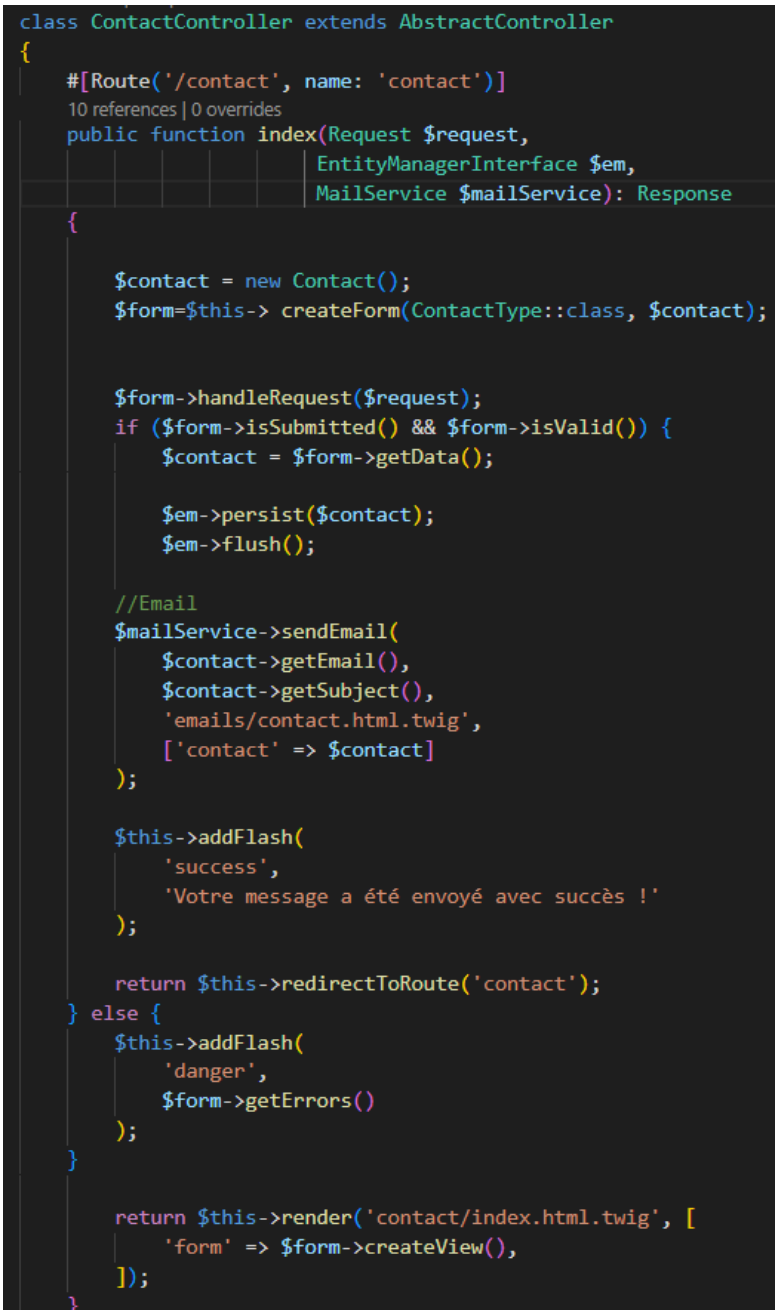
Extrait de annonce.php



```
class Contact
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    1 reference
    private ?int $id;

    #[ORM\Column(type: 'string', length: 50, nullable: true)]
    #[Assert\Length(min: 2, max: 50)]
    2 references
    private ?string $fullName = null;

    #[ORM\Column(type: 'string', length: 180)]
    #[Assert\Email()]
    #[Assert\Length(min: 2, max: 180)]
    2 references
    private string $email;

    #[ORM\Column(type: 'string', length: 100, nullable: true)]
    #[Assert\Length(min: 2, max: 100)]
    2 references
    private ?string $subject = null;

    #[ORM\Column(type: 'text')]
    #[Assert\NotBlank()]
    2 references
    private string $message;

    #[ORM\Column(type: 'datetime_immutable')]
    #[Assert\NotNull()]
    3 references
    private ?\DateTimeImmutable $createdAt;

    1 reference | 0 overrides
    public function __construct()
    {
        $this->createdAt = new \DateTimeImmutable();
    }
```

contact.php

Contrôleurs



```php
class ContactController extends AbstractController
{
    #[Route('/contact', name: 'contact')]
    10 references | 0 overrides
    public function index(Request $request,
                          EntityManagerInterface $em,
                          MailService $mailService): Response
    {

        $contact = new Contact();
        $form=$this-> createForm(ContactType::class, $contact);


        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $contact = $form->getData();

            $em->persist($contact);
            $em->flush();

        //Email
        $mailService->sendEmail(
            $contact->getEmail(),
            $contact->getSubject(),
            'emails/contact.html.twig',
            ['contact' => $contact]
        );

        $this->addFlash(
            'success',
            'Votre message a été envoyé avec succès !'
        );

        return $this->redirectToRoute('contact');
    } else {
        $this->addFlash(
            'danger',
            $form->getErrors()
        );
    }

        return $this->render('contact/index.html.twig', [
            'form' => $form->createView(),
        ]);
    }
```

contactController.php

```php
#[Route('/voitures-doccasions', name: 'annonce')]
6 references | 0 implementations
class AnnonceController extends AbstractController
{
    #[Route('/', name: 'index')]
    10 references | 0 overrides
    public function Annonce (AnnonceRepository $annonceRepository,
                             Request $request
                            ): Response
    {


        $data = new SearchData();
        $form = $this->createForm(SearchForm::class, $data);

        $form->handleRequest($request);
        $annonces = $annonceRepository-> findBySearch($data);


        return $this->render('annonce/index.html.twig', [
            'annonces'=>$annonces,
            'form' => $form->createView(),
        ]);

    }

    #[Route('/{slug}', name: 'details')]
    10 references | 0 overrides
    public function details(Annonce $annonce): Response
    {
        return $this->render('annonce/details.html.twig', compact('annonce'));
    }
}
```

Extrait d'AnnonceController.php

Extrait de SearchData.php

```php
class SearchData
{

    /**
     * @var int
     */
    0 references
    public $page = 1;

    /**
     * @var string
     */
    2 references
    public $q = '';


    /**
     * @var null|integer
     */
    1 reference
    public $max;

    /**
     * @var null|integer
     */
    3 references
    public $min;

    /**
     * @var null|integer
     */
    2 references
    public $kmMax;

    /**
     * @var null|integer
     */
    2 references
    public $kmMin;

    /**
     * @var null|integer
     */
    2 references
    public $dateTimeMax;
```

Repository

```php
public function findBySearch(SearchData $searchData): array
{
    $query = $this
        ->createQueryBuilder('p');

    if (!empty($searchData->q)){
        $query = $query
            ->andWhere('p.title LIKE :q')
            ->setParameter('q', "%{$searchData->q}%");

    }

    if (!empty($searchData->min)){
        $query = $query
            ->andWhere('p.price >= :min')
            ->setParameter('min', $searchData->min);

    }

    if (!empty($searchData->min)){
        $query = $query
            ->andWhere('p.price <= :max')
            ->setParameter('max', $searchData->max);

    }

    if (!empty($searchData->kmMin)){
        $query = $query
            ->andWhere('p.km >= :kmMin')
            ->setParameter('kmMin', $searchData->kmMin);

    }

    if (!empty($searchData->kmMax)){
        $query = $query
            ->andWhere('p.km <= :kmMax')
```

Extrait d'AnnonceRepository.php

Formulaire

```php
<?php

namespace App\Form;

use App\Entity\Contact;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints as Assert;

8 references | 0 implementations
class ContactType extends AbstractType
{
    0 references | 0 overrides
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('fullName', TextType::class, [
                'attr' => [
                    'class' => 'form-control',
                    'minlenght' => '2',
                    'maxlenght' => '50',
                ],
                'label' => 'Nom / Prénom',
                'label_attr'=> [
                    'class' => 'form-label  mt-4'
                ]
            ])
            ->add('email', EmailType::class, [
                'attr' => [
                    'class' => 'form-control',
                    'minlenght' => '2',
                    'maxlenght' => '180',
                ],
                'label' => 'Adresse email',
                'label_attr' => [
                    'class' => 'form-label  mt-4'
                ],
```

Extrait 1 de ContactType.php

```
            'constraints' => [
                new Assert\NotBlank(),
                new Assert\Email(),
                new Assert\Length(['min' => 2, 'max' => 180])
            ]
        ])
        ->add('subject', TextType::class, [
            'attr' => [
                'class' => 'form-control',
                'minlenght' => '2',
                'maxlenght' => '100',
            ],
            'label' => 'Objet',
            'label_attr' => [
                'class' => 'form-label  mt-4'
            ],
            'constraints' => [
                new Assert\Length(['min' => 2, 'max' => 100])
            ]
        ])
        ->add('message', TextareaType::class, [
            'attr' => [
                'class' => 'form-control',
            ],
            'label' => 'Message',
            'label_attr' => [
                'class' => 'form-label mt-4'
            ],
            'constraints' => [
                new Assert\NotBlank()
            ]
        ])

        ->add('submit', SubmitType::class, [
            'attr' => [
                'class' => 'btn btn-primary mt-4'
            ],
            'label' => 'Envoyer'
        ]);
}

0 references | 0 overrides
public function configureOptions(OptionsResolver $resolver): void
{
    $resolver->setDefaults([
```

Extrait 2 de ContactType.php

```php
class LoginType extends AbstractType
{
    0 references | 0 overrides
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add("email", TextType::class, [
                "label" => "email",
                "required" => true,
                //'row_attr' => ['class' => 'nom', 'id' => 'name'],
                "constraints" => [
                    new Length(["min" => 2, "max" => 180, "minMessage" => "Le nom d'utilisateur ne doit pas faire moins de 2 caractères",
                    "maxMessage" => "Le nom d'utilisateur ne doit pas faire plus de 180 caractères"]),
                    new NotBlank(["message" => "Le nom d'utilisateur ne doit pas être vide !"])
                ]
            ])
            ->add("plainPasword", PasswordType::class, [
                "label" => "Mot de passe",
                "required" => true,
                "constraints" => [
                    new NotBlank(["message" => "Le mot de passe ne peut pas être vide !"])
                ]
            ])
            ->add('submit', SubmitType::class, [
                'attr' => [
                    'class' => 'btn btn-primary mt-4'
                ],
                'label' => 'Se connecter'
            ]);

    }
```

LoginType

CRUD Controllers

```php
public function configureCrud(Crud $crud): Crud
{

    return $crud
        ->setEntityLabelInSingular('Annonce')
        ->setEntityLabelInPlural('Annonces')

        ->setPageTitle('index', "Administration des voitures d'occasions")
        ->setPaginatorPageSize(10)
    ;
}

0 references | 0 overrides
public function configureFields(string $pageName): iterable
{
    return [
        IdField::new('id')
            ->hideOnForm(),
        TextField::new('title'),
        NumberField::new('price'),
        TextField::new('dateTime'),
        NumberField::new('km'),
        TextField::new('carburant'),
        TextareaField::new('description'),
        TextareaField::new('option'),
        TextareaField::new('equipement'),
       TextField::new('imageFile')
            ->setFormType(VichImageType::class),
        ImageField::new('imageName')
            ->setBasePath('/images/annonces')
            ->onlyOnIndex(),
        SlugField::new('slug')
            ->setTargetFieldName('title')
            ->hideOnIndex(),
    ];
}
```

AnnonceCrudController.php

```php
class ContactCrudController extends AbstractCrudController
{
    0 references | 0 overrides
    public static function getEntityFqcn(): string
    {
        return Contact::class;
    }

    24 references | 0 overrides
    public function configureCrud(Crud $crud): Crud
    {

        return $crud
            ->setEntityLabelInSingular('Demande de contact')
            ->setEntityLabelInPlural('Demandes de contact')
            ->setPageTitle("index", "Administration des demandes de contact")
            ->setPaginatorPageSize(30)

        ;
    }


    0 references | 0 overrides
    public function configureFields(string $pageName): iterable
    {
        return [
            IdField::new('id')
                ->hideOnForm(),
            TextField::new('fullName'),
            TextField::new('email'),
            TextareaField::new('message')
                ->hideOnIndex(),
            DateTimeField::new('createdAt')
                ->hideOnForm()
        ];
    }
```

ContactCrudController.php

```
ntroller > Admin > UserCrudController.php > PHP Intelephense > UserCrudController > configureFields
class UserCrudController extends AbstractCrudController
{

    4 references | 0 overrides
    public function __construct(public UserPasswordHasherInterface $userPasswordHasher)
    {
    }

    0 references | 0 overrides
    public static function getEntityFqcn(): string
    {
        return User::class;
    }

    24 references | 0 overrides
    public function configureCrud(Crud $crud): Crud
    {

        return $crud
            ->setEntityLabelInSingular('Employé')
            ->setEntityLabelInPlural('Employés')
            ->setPageTitle("index", "Administration des employés")
        ;
    }


    0 references | 0 overrides
    public function configureFields(string $pageName): iterable
    {
        return [
            IdField::new('id')
                ->hideOnForm(),
            EmailField::new('email'),
            TextField::new('Prenom'),
            TextField::new('Nom'),
            TextField::new('plainPassword', 'Mot de passe')
                ->setFormType(PasswordType::class)
                ->setRequired($pageName === Crud::PAGE_NEW)
                ->onlyOnForms(),
            ArrayField::new('roles')
                ->hideOnIndex(),
        ];
```

UserCrudController.php

Pages Twig

```
<footer id="footer" class="d-flex flex-wrap justify-content-between align-items-center reveal">
    <div class="col-4 text-center"> <img width="30" height="30" src="/img/horloge.svg"></img><br>
        Lundi-Vendredi : 8h45-12h/14h-16h<br>Samedi : 8h45-12h<br>Dimanche : fermé
    </div>

    <div id="lienML" class="col-md-4 d-flex align-items-center justify-content-center text-center ">
        <p >&copy; 2023 Garage V. Parrot<br>
            <a style="text-decoration:none" href="{{ path('mentions_legales')}}"> Mentions légales - Politique de confidentialité</a>
        </p>
    </div>



    <ul class=" col-md-4 justify-content-end text-center">
        <img width="30" height="30" href="#" src="/img/facebook.png"></img>
        <img width="30" height="30" href="#" src="/img/instagram.png"></img>
    </ul>

</footer>
```

footer.html.twig