Chabha
Iddir

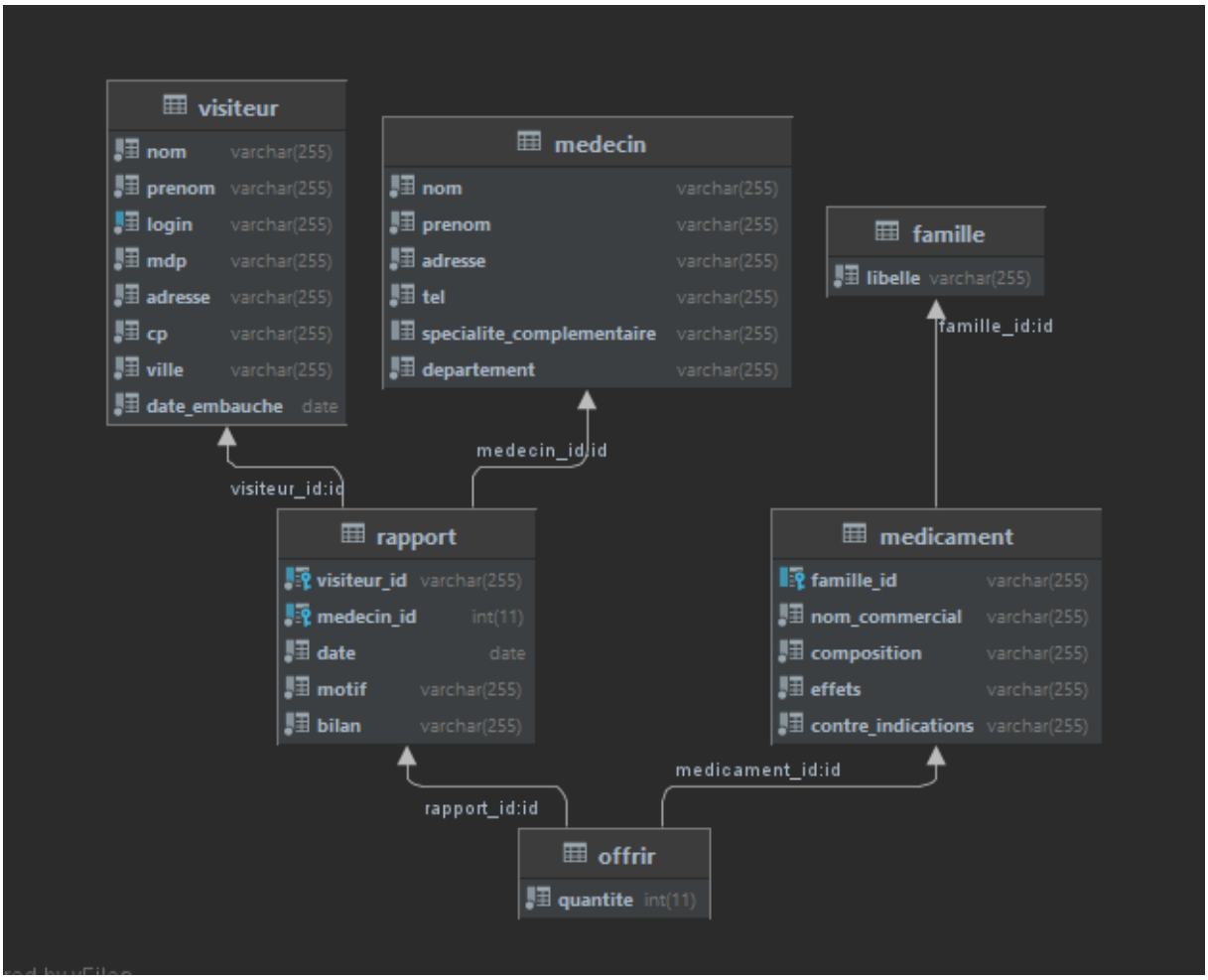
Documentation technique

AP : Gestion Visites

Ce site à était réaliser avec le Framework Symfony, qui représente un ensemble d'outils conçus pour faciliter le processus de développement d'application web. L'objectif est d'apporter les composants de bases qui constituent toutes les applications web tel un système de gestion des routes, un moteur de templates, … .

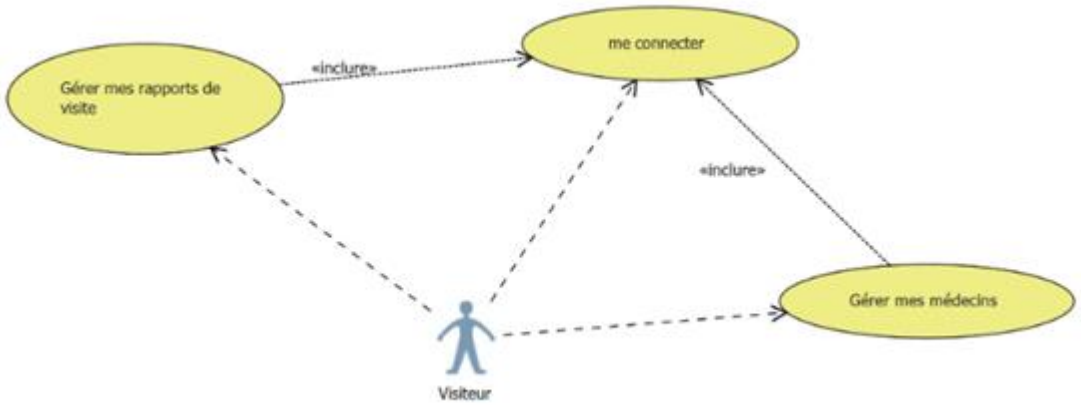
On retrouve plus bas la structure des données ainsi que le diagramme représentant le cas général d'utilisation du site.

Modèle Logique de données :



https://github.com/Chabha-02/GestionVisite/tree/master/src

Diagramme des cas d'utilisations :



Symfony en détails :

Au niveau de la base de données symfony utilisent Doctrine comme ORM (object relational mapper) qui se chargera de générer la structure de la base de données et gérer l'accès aux données à l'aide des repositories. Il fait le lien entre BDD et Poo et transforme une table en un objet facilement manipulable via ses attributs

L'intérêt principal est d'éviter beaucoup de code très similaire. Ainsi plutôt que de devoir recoder à la main les fonctions de base (CRUD– création, lecture, modification, suppression), symfony ajoutent une couche logicielle intermédiaire qui se charge de la « traduction ».

La difficulté principale consiste à trouver la « bonne traduction » pour toutes les opérations possibles.

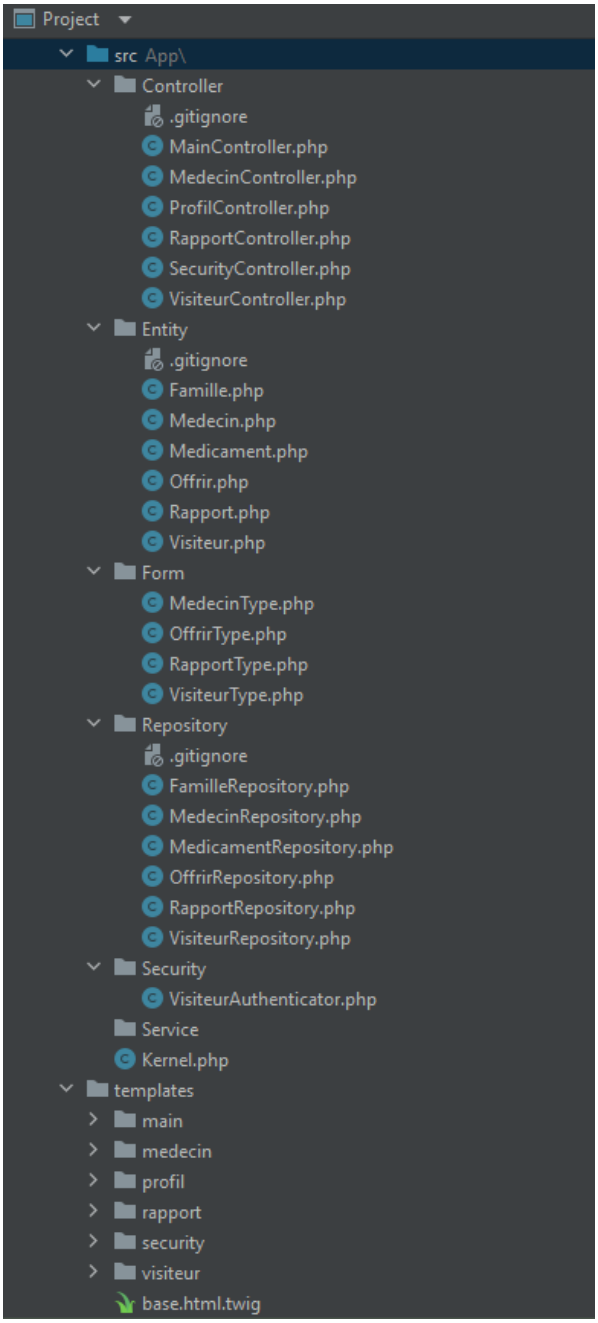
Avantages

- Réduction du code à créer
- Homogénéité du code objet
- Accélération du temps de développement

Inconvénients

- Problèmes de mise en place lorsque la base de données n'est pas faite dans les règles de l'art
- Ne permet pas de gérer les requêtes un peu complexes (jointures, groupements.

Structure de l'application:



Les controllers :

Ils vont permettre de lire une requête et renvoyer une réponse sur une route qui représente le lien entre une URL et un contrôleur. Quand le serveur reçoit une requête HTTP, symfony regarde dans le les controllers pour trouver la route.

Les entités :

Représente les objets métier de l'application.

Form :

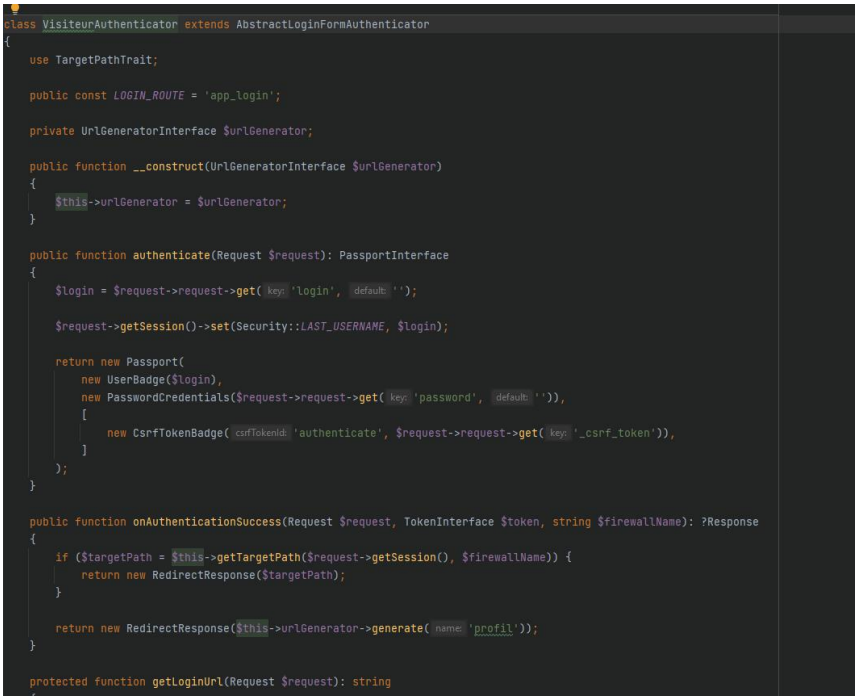
Représente des classes techniques qui permettent de faciliter la création et la validation de formulaire et dans Symfony spécifiquement les formulaires vont permettre de créer et/ou modifier une entité. On dit qu'ils sont « mappés » à une entité. Dans chaque formulaire que l'on va créer, on va renseigner l'entité que ce formulaire va modifier.

Repository :

Les repositories permettent l'accès aux données, ça va centraliser tout ce qui touche à la récupération de nos entités.

Authentification :

Pour créer une authentification Symfony aura besoin d'une entité user qui sera ici notre entité visiteur après lui avoir passé cette information il suffit de faire une commande qui est php bin/console make :auth pour crée l'app VisiteurAuthenticator qui s'occupera de la sécurité de la connexion.



```php
class VisiteurAuthenticator extends AbstractLoginFormAuthenticator
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'app_login';

    private UrlGeneratorInterface $urlGenerator;

    public function __construct(UrlGeneratorInterface $urlGenerator)
    {
        $this->urlGenerator = $urlGenerator;
    }

    public function authenticate(Request $request): PassportInterface
    {
        $login = $request->request->get( key: 'login',  default: '');

        $request->getSession()->set(Security::LAST_USERNAME, $login);

        return new Passport(
            new UserBadge($login),
            new PasswordCredentials($request->request->get( key: 'password',  default: '')),
            [
                new CsrfTokenBadge( csrfTokenId: 'authenticate', $request->request->get( key: '_csrf_token')),
            ]
        );
    }

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

        return new RedirectResponse($this->urlGenerator->generate( name: 'profil'));
    }

    protected function getLoginUrl(Request $request): string
```

Gestion des rapports et des médecins avec CRUD :

Pour gérer les rapports et les médecins on va toucher ici à l'atout majeur d'un Framework comme Symfony car on veut que notre visiteur voie ses rapports, les édites et en crée de nouveau et pour cela Symfony a également une commande préfète(C'est php bin/console make : crud) qui va générer un Controller, un form et une vue. Cette commande demandera l'entité pour fournir ses différentes opérations, pour GSB ce sera rapport et médecins.

```php
#[Route('/new', name: 'medecin_new', methods: ['GET','POST'])]
public function new(Request $request): Response
{
    $medecin = new Medecin();
    $form = $this->createForm( type: MedecinType::class, $medecin);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($medecin);
        $entityManager->flush();

        return $this->redirectToRoute( route: 'medecin_index', [], status: Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm( view: 'medecin/new.html.twig', [
        'medecin' => $medecin,
        'form' => $form,
    ]);
}

#[Route('/{id}', name: 'medecin_show', methods: ['GET'])]
public function show(Medecin $medecin): Response
{
    return $this->render( view: 'medecin/show.html.twig', [
        'medecin' => $medecin,
    ]);
}

#[Route('/{id}/edit', name: 'medecin_edit', methods: ['GET','POST'])]
public function edit(Request $request, Medecin $medecin): Response
{
    $form = $this->createForm( type: MedecinType::class, $medecin);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute( route: 'medecin_index', [], status: Response::HTTP_SEE_OTHER);
```

Controller généré par Symfony

La Recherche :

Pas de fonctions préfète ici mais les deux recherches sont assez similaires dans la base. On passe déjà par le Controller. Brièvement on applique une condition, si le champ n'est pas vide alors on passe la requête inscrite dans la Repository sinon on affiche tous les médecins.

```php
#[Route('/', name: 'medecin_index', methods: ['GET'])]
public function index(MedecinRepository $medecinRepository): Response
{
    $request = Request::createFromGlobals();
    $query = $request->query->get( key: 'nom');
    if($query != '' && $query != Null) {
        $medecins = $medecinRepository->findMedecinByNom($query);
    } else {
        $medecins = $medecinRepository->findAll();
    }

    return $this->render( view: 'medecin/index.html.twig', [
        'medecins' => $medecins,
    ]);
}
```

Dans la repository on fera notre requête avec un like pour la recherche par nom.

```php
/**
 * @return Medecin[] Returns an array of Medecin objects
 */

public function findMedecinByNom(string $nom): array
{
    $entityManager = $this->getEntityManager();

    $query = $entityManager->createQuery(
        dql: 'SELECT n
        FROM App\Entity\Medecin n
        WHERE n.nom LIKE :nom
        '
        )->setParameter( key: 'nom', value: '%'.$nom.'%');

    return $query->getResult();
}
```

On lui associera pour finir une vue dans le dossier template.

```
<div class="container">
    <div class="row height d-flex justify-content-center align-items-center">
        <form class="form-inline my-2 my-lg-0" method="get">
            <input class="form-control mr-sm-2" type="search" placeholder="Chercher un médecin" name="nom" aria-label="Recherche">
            <button class="btn btn-info my-2 my-sm-0" type="submit">Recherche</button>
        </form>
    </div>
</div>
```

UML :