DOSSIER PROJET

STAGE DU 1/03 AU 26/04

Développeur web et web mobile

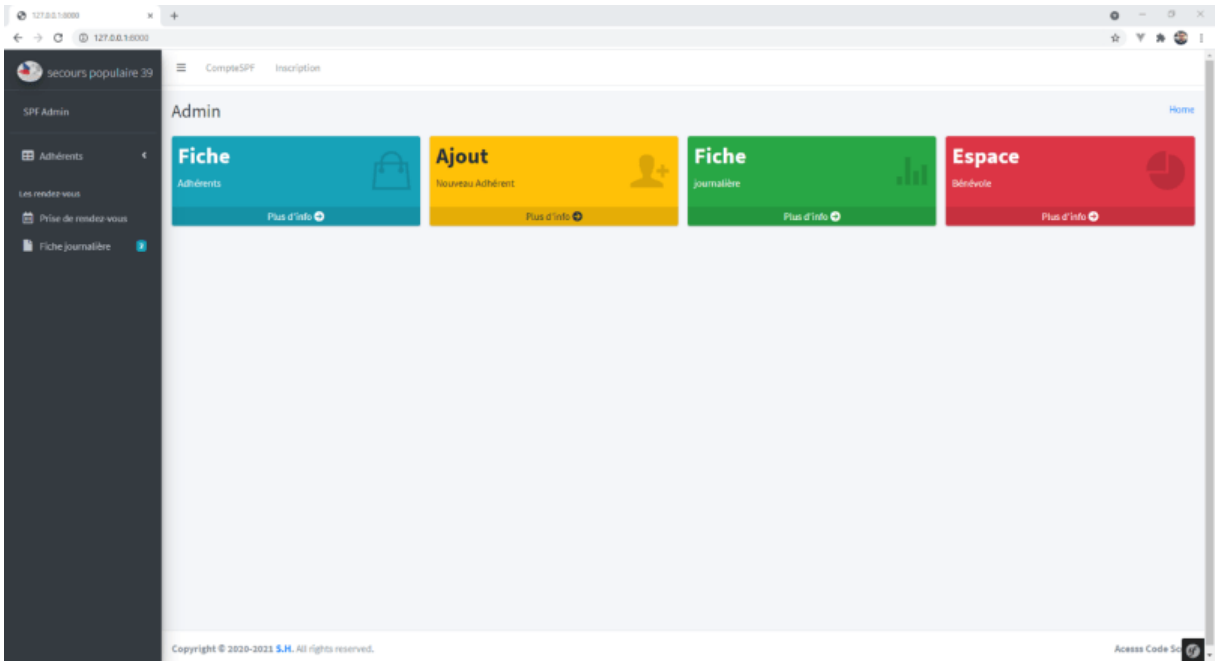
Réalisation d'une administration par

Hajji Sami

Access Code School

Table des matières

1- Liste des compétences du référentiel couvertes par le projet



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

-Réaliser une interface utilisateur web statique et adaptable
-Développer une interface utilisateur web dynamique

1.2 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 et web mobile

2 -Résumé du projet

J’ai effectué mon stage d’une durée de 2 mois du 01/03/2021 au 26/04/2021 au sein du secours populaire à Dole. C'est une association française qui intervient sur le plan matériel, médical, moral et juridique.

J’ai travaillé en tant que bénévole en faisant des livraisons et des distributions de colis alimentaires.

Durant ma formation, j’ai proposé au directeur Mickaël Nassom de lui créer une application qui faciliterait les tâches des bénévoles pour gérer la distribution alimentaire.

Dès le début de mon stage, j’ai étudié leurs besoins.

Il fallait faire une application très simple et facile d’utilisation car la plupart du personnel a un niveau bas en informatique.

J'ai accompli plusieurs missions, voici les plus importantes :

- Création login pour la sécurité de l'application
- CRUD adhérent
- Création d’un moteur de recherche pour filtrer
- Création d’un système de prise de rendez vous
- Création d’un listing journalier

Pour ce projet, j'ai utilisé le framework Symfony ainsi que les bibliothèques "fullcalendar", “axios”, "bootstrap" et “select2”.

Ce projet m’a permis d’évaluer mes capacités d’adaptation et mes compétences.

3 - Cahier des charges

Il y a 1200 personnes qui bénéficient de l'aide alimentaire.

Après concertation avec le directeur de l'association, nous avons décider de mettre en place un site qui facilite les tâches des bénévoles.

Le côté administration :

- Inscrire les nouveaux adhérents suivants plusieurs critères :
  -fréquence 1 = 1 colis/mois PAYANT
  -fréquence 2 = 2colis/mois GRATUIT (+ lessive, hygiène et/ou couches 1x par mois)
  -cas exceptionnel (suivant le cas)

-Il existe 4 types de colis calculés par le nombre de personne résidant dans le foyer, adulte(s) et enfant(s) :

- colis 1 = 1 personne

-colis 2 = 2-3 personnes

-colis 3 = 4-5 personnes

-colis 4 = 6 personnes et +

-Modifier ou supprimer un adhérent

Et pour le côté accueil :

- Etablir la prise de rendez-vous et paiement
- Sortir le détail de la fiche journalière
- Constituer l'historique des bénéficiaires absent/présent
- Détailler le nombre de passage par an
- Consulter la dernière fois que le bénéficiaire fréquence 2 a eu hygiène lessive couches

4 - Spécifications techniques du projet

J'étais le seul développeur à travailler sur le site, étant donné que le directeur et certains bénévoles travaillent uniquement Excel.

Lors de la présentation du projet, j'ai pu identifier que le principal besoin requiert l'utilisation d'une technologie back-end, performante et sécurisée. J'ai donc choisi le Framework Symfony pour son côté système de gestion de base de données.

Le Framework Symfony

-l'architecture MVC
Mon choix s'est porté sur le Framework Symfony 4.4 pour son organisation selon une architecture MVC (Modèle Vue Contrôleur). Il correspond aux réels besoins fonctionnels du site.

-la sécurité dans Symfony
L'essentiel dans le site consiste au traitement des données soumises par les utilisateurs. La question de la sécurisation à tous les niveaux est primordiale dans les valeurs transmises par le Secours Populaire.
Ce Framework permet d'empêcher de nombreuses failles de sécurité : grâce à Twig, nous empêchons les failles. Une utilisation malveillante peut supprimer ou modifier une table de la base de données.
Toutes les requêtes de Symfony sont préparées avant l'envoi en base de données ce qui évite les injections SQL.

-L'efficacité de Symfony
Les entités sont la solution à une première problématique de sécurité. La gestion de la base de données avec Symfony 4 se fait via l'ORM Doctrine qui permet de ne plus avoir à créer de table via PhpMyAdmin, mais exploite les données comme des objets. Une entité est un objet que l'ORM va manipuler et enregistrer automatiquement. Il est possible de créer des relations entre les entités, comme les clés étrangères d'une base de données.

Fullcalendar

-Fullcalendar est une bibliothèque JavaScript.
J'ai choisi Fullcalendar pour afficher un calendrier qui permet les prises de rendez-vous facilement. Grâce à la documentation, je peux changer l'apparence et le comportement du calendrier. Les options sont définies et modifiables.

Axios

-Axios est une bibliothèque JavaScript. Elle permet de communiquer avec des API en utilisant des requêtes. Il est possible de créer des requêtes avec la méthode POST. Dans mon cas, je l'ai utilisé pour récupérer des données sans charger la page.

Select2

-Select2 est une librairie JavaScript/jQuery qui améliore l'usage des sélecteurs (balises <select>) natifs des navigateurs en facilitant entre autres la recherche d'un terme dans la liste. Dans mon projet, je l'ai utilisé pour une datalist.

Le Framework Bootstrap

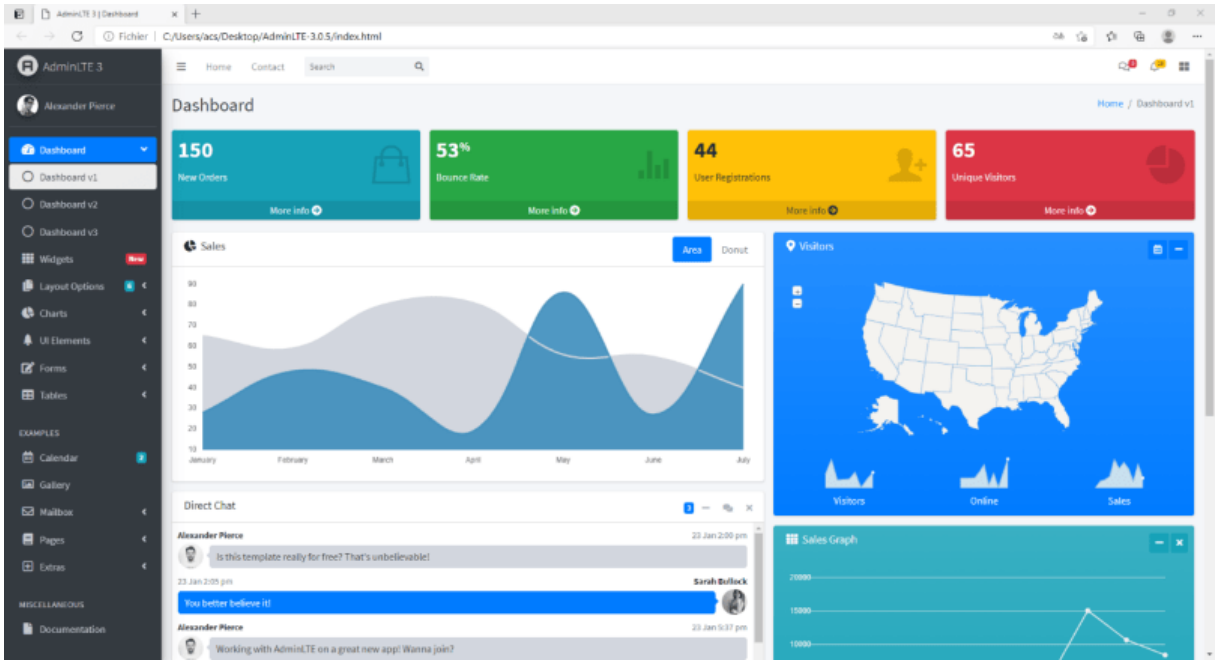
-Bootstrap est un Framework open source de développement web orienté interface graphique. Ce Framework utilisant des langages HTML, CSS et JavaScript fournis aux développeurs des outils pour créer un site facilement. Pour mon site, j'ai utilisé Bootstrap pour me permettre d'obtenir d'une part une mise en page rapidement et d'autre part pour gérer la partie responsive du site.

GitHub

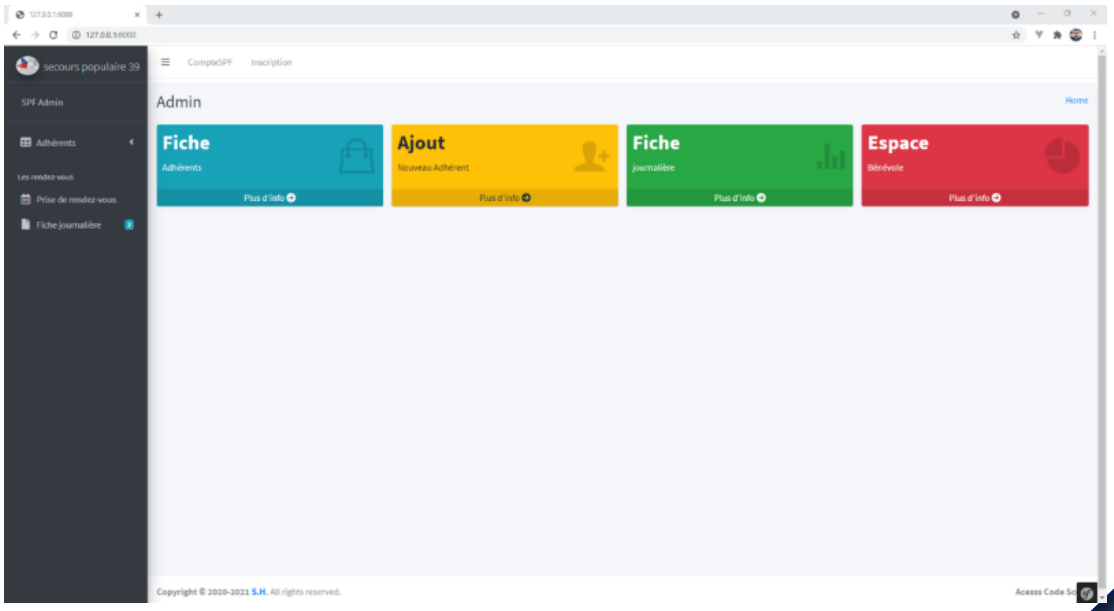
-GitHub est un site de partage de code, sur lequel nous pouvons publier des projets dont le code est géré avec le système de gestion de version Git. Pour ne pas perdre mon travail du jour, je mets mon code sur GitHub quotidiennement.

5 -Réalisations

5.1 Choisir un Template et l'adapter



Ce Template détermine la structure, la façon dont il s'affichera. La mise en page et le design sont facile et simple d'utilisation. Je l'ai adapté aux besoins du site.



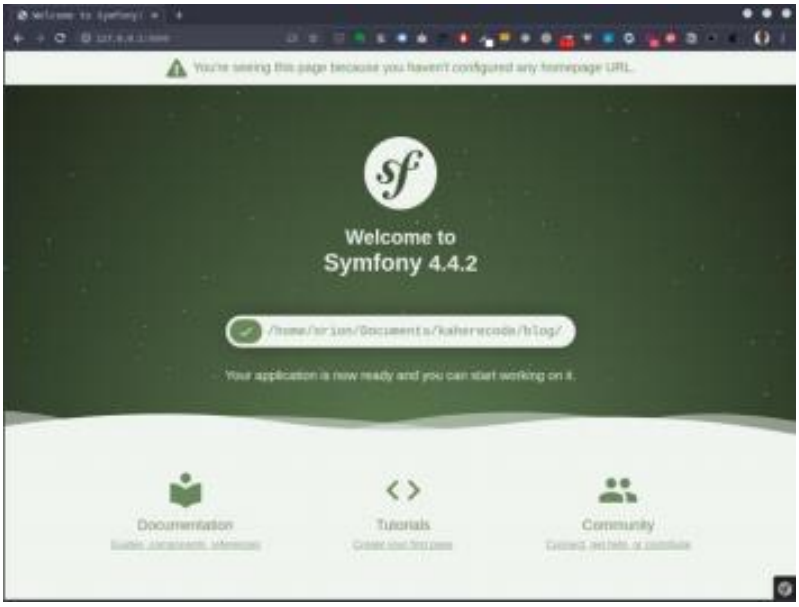
5.2 Installation Symfony 4.4

composer create-project symfony/website-skeleton admin_secours_populaires "4.4.*"

Mon projet a été créé, je me déplace dans le dossier du projet et je lance le serveur :
Cd admin_secours_populaires
Symfony server : start

```
[OK] Web server listening

     The Web server is using PHP CGI 7.3.12

     http://127.0.0.1:8000
```

Dans le navigateur, la page s'ouvre :



5.3 Conception et création de la base de données

Pour crée la base de données avec symfony 4 , on commence par éditer le fichier .env qui se trouve à la racine , ce fichier contient les configurations de notre application comme les informations sur la base de données.

DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/spf39?serverVersion=5.7"
```

Ensuite grâce à la commande suivante j'ai créé la base de données que je peux la voir dans phpMyAdmin.

```
> php bin/console doctrine:database:create
```

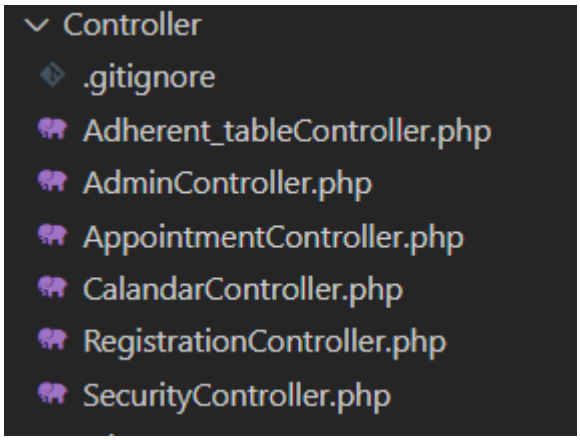
5.4 Création des contrôleurs

Pour la petite définition, le contrôleur est juste une méthode qui va nous permettre de retourner une réponse.

Pour cela, j'utilise la ligne de commande :

```
$ ./bin/console make:controller
```

Dans mon projet j'ai créé les contrôleurs :



5.5 Création des entités

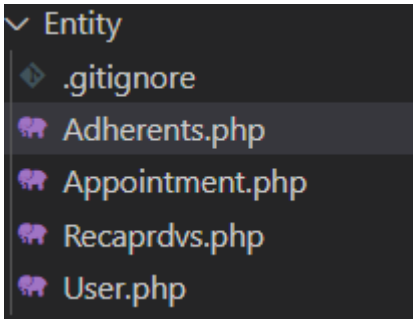
Symfony utilise Doctrine pour la gestion de la base de données. Doctrine nous permet d'écrire et lire dans notre base de données en utilisant du PHP.

Les entités sont des classes PHP qui contient des propriétés. Et son rôle est aussi de représenter une table coté base de données.

```
$ php bin/console make:entity
```

Un fichier PHP va être généré dans le dossier src/Entity avec le nom de la classe choisie et les propriétés de cette classe est automatiquement la commande va générer un fichier dans repository pour permettre de communiquer avec la base de données.

Dans mon projet j'ai créé des entités :



Puis j'ai mis à jour la structure de la base de données chaque fois grâce à cette ligne de commande

```
$ php bin/console doctrine:schema:update --force
```

Après grâce aux relations entre entités, j'ai effectué une relation entre adhérent et appointement OneToMany c'est à dire un adhérent peut avoir plusieurs rendez-vous.

```
/**
 * @ORM\OneToMany(targetEntity=Appointment::class, mappedBy="adherents")
 */
private $Appointment;

public function __construct()
{
    $this->Appointment = new ArrayCollection();
}
```

5.6 Création des formulaires

Quand j'utilise les formulaires en Symfony, je dois passer par trois étapes :

-D'abord je construis le formulaire dans le contrôleur ou je lui définissais une classe

-J'affiche le formulaire dans une vue

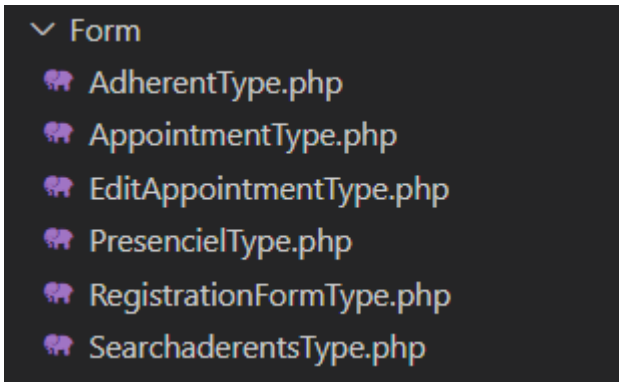
-Je traite le formulaire pour valider les données et exécuter les actions demandées

J'ai créé mes formulaires avec cette ligne de commande :

```
$ php bin/console make:form
```

Ensuite Symfony me demande de rentrer le nom du formulaire de la classe, chaque formulaire est attaché à une entité.

Dans mon projet, j'ai utilisé des formes :



5.7 Gestion des adhérents

J’ai établi une interface où il y a plusieurs informations :

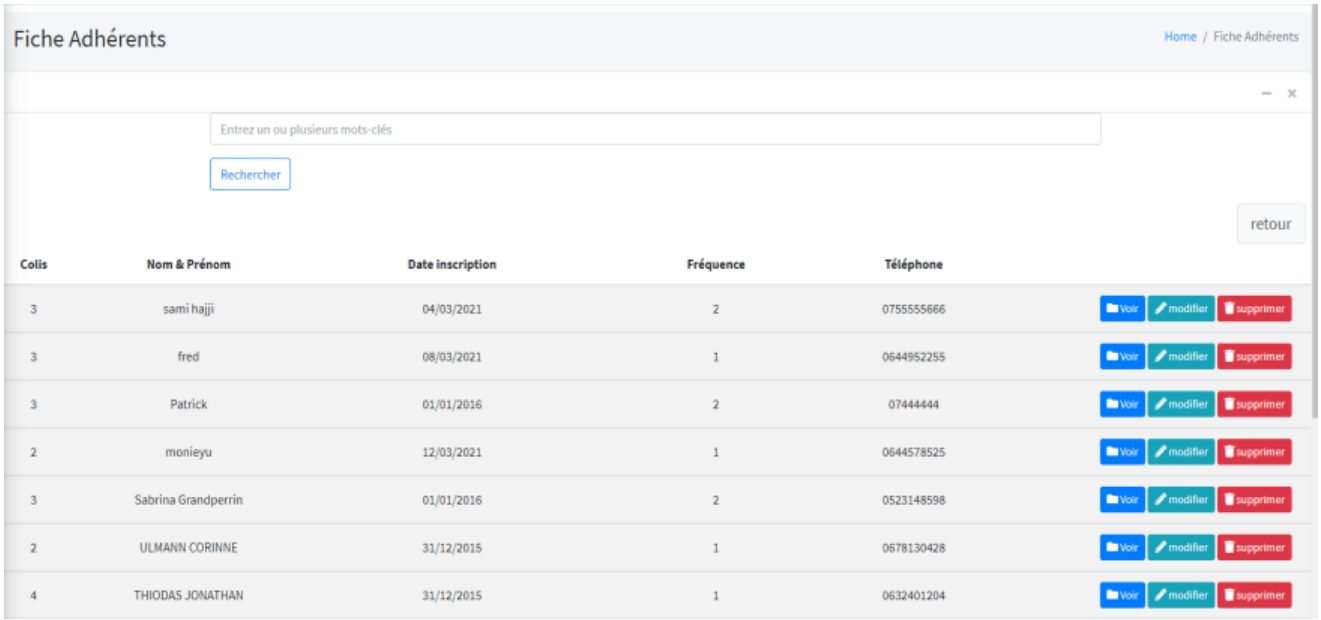
a-Afficher/filtrer la liste adhérents

b-modifier des informations

c-supprimer une fiche adhérent

d-voir la fiche adhérent

e-inscription d’un adhérent



a-Afficher/filtrer les adhérents

Dans mon contrôleur, j'ai créé une fonction avec une condition pour me récupérer tous les adhérents grâce à la méthode findAll ( ) et leurs informations ou sinon un système de filtrage par nom/prénom/numéro de dossier à l'aide de Repository.

```php
/**
 * @Route("/data/tables/", name="data_tables" , methods={"GET","POST"})
 */
public function AficheAdhrent(AdherentsRepository $AdherentsRepository, request $request): Response
{

  $Adherents = $AdherentsRepository->findAll();

  $form = $this->createForm(SearchaderentsType::class);


  $result = [];

  if ($form->handleRequest($request)->isSubmitted() && $form->isValid()) {

    $datas = $form->getData();
    $Adherent = $datas->getNomPrenom();
    $result = $AdherentsRepository->search($Adherent);
  }
  return $this->render('data_tables/adherents.html.twig', [
    'Adherents' => $Adherents,
    'search' => $form->createView(),
    'result' => $result

  ]);
}
```

Dans le Repository, voilà la fonction search qui me permet de filtrer les adhérents en utilisant le QueryBuilder :

```php
/**
* @return Adherents[] Returns an array of Adherents objects
*/
public function search ($critere)
{
    return $this->createQueryBuilder('a')

        ->andWhere('a.NomPrenom LIKE :NomPrenom')
        ->setParameter('NomPrenom', '%'.$critere.'%')
        ->orWhere('a.Dossier LIKE :Dossier')
        ->setParameter('Dossier', '%'.$critere.'%')
        ->orderBy('a.id', 'ASC')
        ->setMaxResults(10)
        ->getQuery()
        ->getResult()
    ;
}
```

Dans le vue dans TWIG, j'ai établi une boucle for qui me permet d'afficher tous les adhérents et une condition pour le filtrage, sans oublier que j'ai élaboré 2 formes : une pour adhérent et l'autre searchadherents

```php
class SearchaderentsType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
        ->add('NomPrenom', SearchType::class, [
            'label' => false,
            'attr' => [
                'class' => 'form-control',
                'placeholder' => 'Entrez un ou plusieurs mots-clés'
            ],
            'required' => false
        ])

        ->add('Rechercher', SubmitType::class, [

            'attr' => [
                'class' => 'btn btn-outline-primary',

            ]
        ]);


    }
```

b-modifier des informations

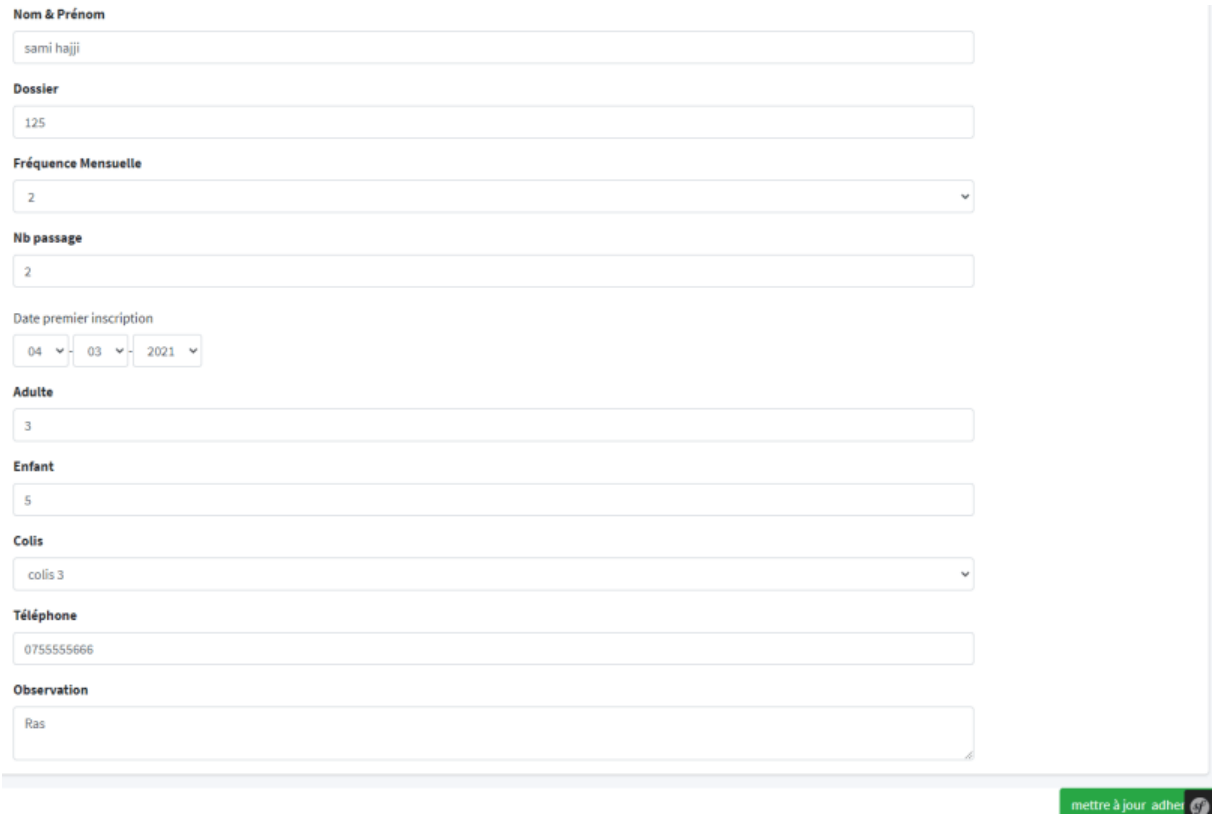
Dans mon contrôleur, j'ai établi une fonction edit et je l'ai lié avec une création d'un formulaire AdherentType. En cliquant sur modifier dans la vue, je récupère l'identifiant de l'adhérent comme ça je peux afficher ses informations.

```php
/**
 * @Route("/editAdherent/{id}", name="editAdherent")
 */

public function edit(Request $request, Adherents $projects): Response
{

  $entityManager = $this->getDoctrine()->getManager();


  $form = $this->createForm(AdherentType::class, $projects);
  $form->handleRequest($request);
  if ($form->isSubmitted() && $form->isValid()) {

    $entityManager->persist($projects);
    $entityManager->flush();
    return $this->redirectToRoute('data_tables');
  }
  return $this->render('data_tables/editAdhrent.html.twig', [
    'form' => $form->createView()
  ]);
}
```

Dans la vue TWIG, j'ai affiché le formulaire en utilisant :

```twig
            {{ form_start(form) }}
            <div class="container">
              <div class="row w-100">
                <div class="col-12">
                  {{ form_widget(form) }}
                </div>
              </div>
            </div>
          </div>
        </select>
      </div>
    </div>

    <!-- /.card-body -->
  </div>
  <input
  type="submit" value="mettre à jour  adhérent" class="btn btn-success float-right">
<!-- /.card -->
</div>
 {{ form_end(form) }}
```

Voilà le formulaire sur le site :



c-supprimer une fiche adhérent

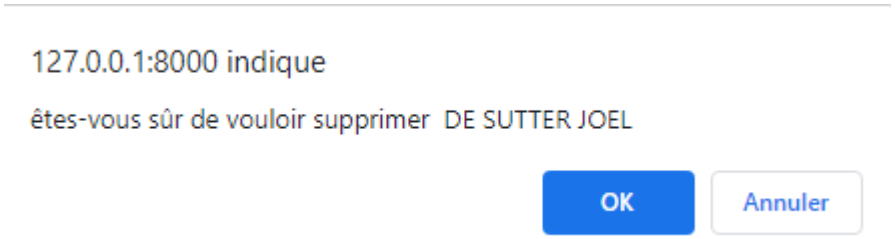
Dans mon contrôleur, j'ai établi une fonction deleteadherent en passant l'identifiant dans la route et après la suppression reste sur la même page

```php
/**
 * @Route("deleteadherent/{id}", name="delete_adherent")
 */
public function deleteadherent(Request $request, Adherents $adhrent): Response
{

  $entityManager = $this->getDoctrine()->getManager();
  $entityManager->remove($adhrent);
  $entityManager->flush();
  return $this->redirect($request->getUri());
}
```

Dans la vue, pour la suppression, j'ai mis en place un message de confirmation

```twig
<a class="btn btn-danger btn-sm"
    onclick="return confirm('êtes-vous sûr de vouloir supprimer  {{ Adherent.NomPrenom}}')"
    href="{{path ('delete_adherent', {'id' : Adherent.id})}}">
    <i class="fas fa-trash"></i>
    supprimer
</a>
```

Voilà le message de confirmation sur le site :



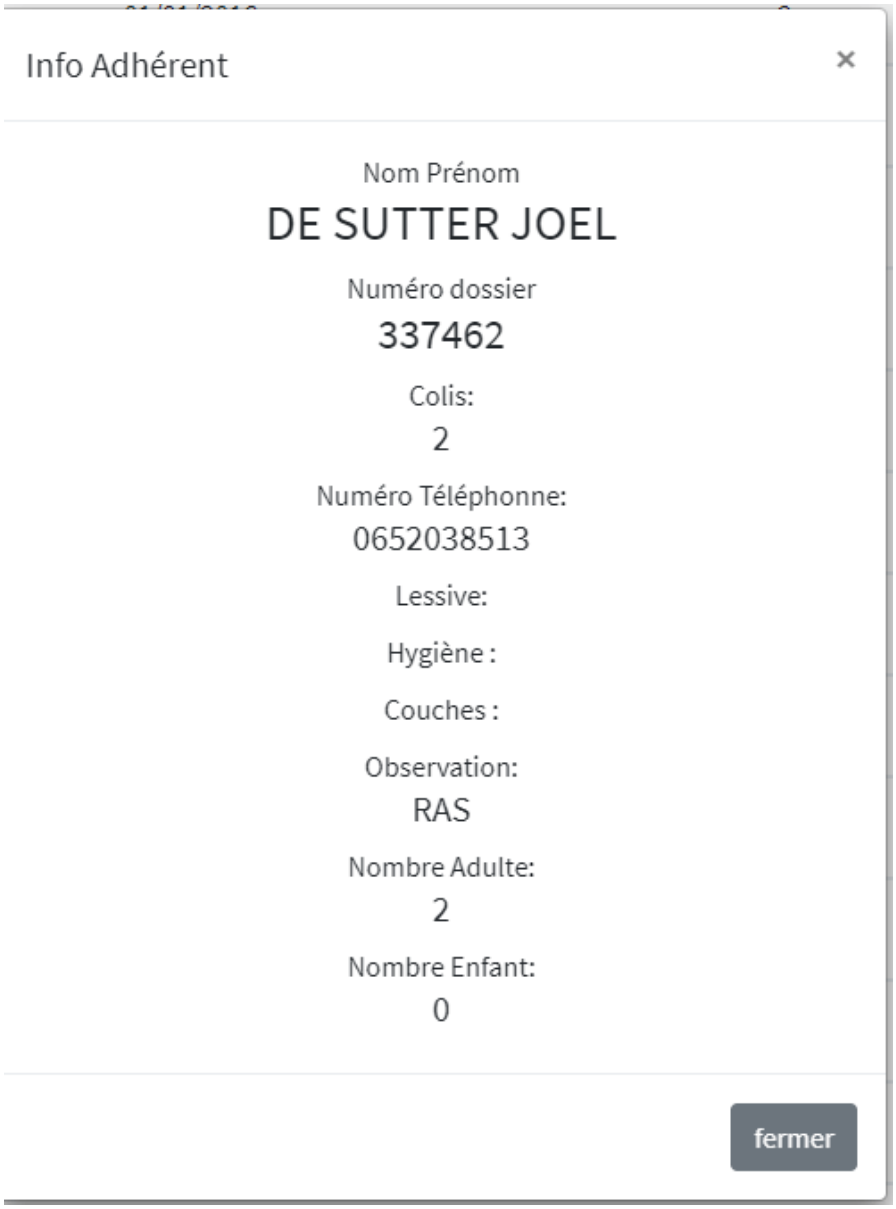
d-voir la fiche adhérent

J’ai créé un bouton qui ouvre un modal de Bootstrap en passant l’identifiant de la personne choisie. Les informations des adhérents sont affichées dans un popup.

Dans le JavaScript, j’ai créé une fonction event

```
$('#exampleModal').on('show.bs.modal', function (event) {
    var button = $(event.relatedTarget) // Button that triggered the modal

    var nom_user = button.data('nom')
    var colis = button.data('colis')

    var Telephone = button.data('numero')
    var Observation = button.data('observation')
    var hygiene = button.data('hygiene')
    var Lessive = button.data('lessive')
    var adulte = button.data('adulte')
    var enfant = button.data('enfant')
    var dossier = button.data('dossier')
    var couches = button.data('couches')

    var modal = $(this)

    modal.find('#nameuser').text(nom_user)
    modal.find('#colis').text(colis)
    modal.find('#Telephone').text(Telephone)
    modal.find('#Observation').text(Observation)
    modal.find('#hygiene').text(hygiene)
    modal.find('#Adulte').text(adulte)
    modal.find('#enfant').text(enfant)
    modal.find('#Dossier').text(dossier)
    modal.find('#Lessive').text(Lessive)
    modal.find('#couche').text(couches)

})
```

Dans la vue, j’ai passé les informations dans une boucle for et dans le contenu de modal, j’ai mis les informations en passant des id



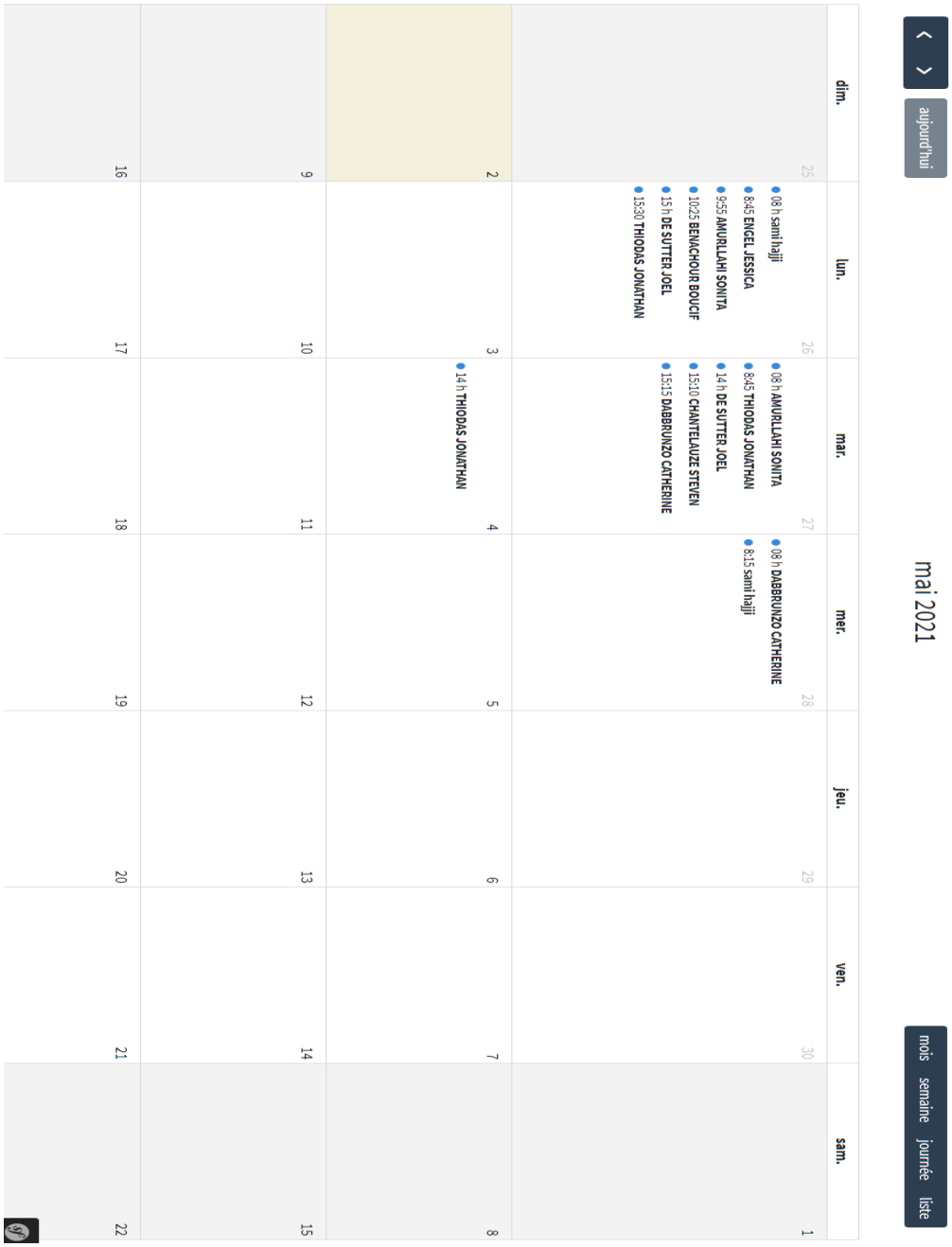
e-inscription d’un adhérent

J’ai édité une fonction pour l’ajout d’un nouvel adhérent, en créant dans la vue un formulaire.

```php
/**
 * @Route("/AjoutAdhrent", name="Addadherent")
 */

public function AjoutAdhrent(Request $request, AdherentsRepository $adherentsRepository)

{
  setlocale(LC_TIME, 'fr_FR.UTF8', 'fr.UTF8', 'fr_FR.UTF-8', 'fr.UTF-8');
  $beneficiaire = new Adherents();
  $form = $this->createForm(AdherentType::class, $beneficiaire);
  $form->handleRequest($request);
  $projets = $adherentsRepository->findAll();

  if ($form->isSubmitted() && $form->isValid()) {
    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->persist($beneficiaire);
    $entityManager->flush();
    return $this->redirectToRoute('data_tables');
  }

  return $this->render('data_tables/addAdherent.html.twig', [
    'form' => $form->createView()

  ]);
}
```

5.8 Prise de rendez-vous

Cette partie a été la plus compliquée à mettre en place. J'ai utilisé la bibliothèque JavaScript FullCalendar et je l'ai intégrée dans mon projet en la modifiant pour les besoins du projet (langue française, horaires, planning …)
Ensuite, j'ai créé une entité Appointment et un formulaire où je me suis permis d'utiliser l'EntityType pour récupérer les noms/prénoms des adhérents de l'autre table adhérent.



Dans cette partie, c'est un mélange entre le back-end et le front-end. En commençant par le back, j'ai créé 2 contrôleurs. Le premier, juste pour l'affichage du calendrier, le deuxième pour les évènements.

L'affichage du calendrier, voici la fonction :

```php
class CalandarController extends AbstractController
{
    /**
     * @Route("/calandar", name="calandar",methods={"GET"})
     */
    public function index(Request $request): Response
    {
        $form = $this->createForm(AppointmentType::class);

        return $this->render('calandar/calandar.html.twig');
    }



}
```

Pour les évènements :

a-récupération des informations de la base de données

J'ai récupéré les informations de la base de données en format json en utilisant SerializerInterface. Je suis passé par la fonction findAll ( )

```php
/**
 * @Route("/appointment/showappeintment", name="showappeintment")
 */


public function showappeintment(AppointmentRepository $appointmentRepository, SerializerInterface $serializer): Response
{

    $appointments = $appointmentRepository->findAll();


    $result = new Response($serializer->serialize($appointments, 'json', ['groups' => ['Appointment:list']]));

    return $result;
}
```

```php
class Adherents
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     *
     * @Groups({"Appointment:list"})
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     *
     * @Groups({"Appointment:list"})
     */
    private $NomPrenom;
```

Dans le JavaScript, j'ai créé une event où je passe l'url, ensuite j'ai établi une fonction où je récupère le nom de l'adhérent et l'heure/date de rendez-vous.

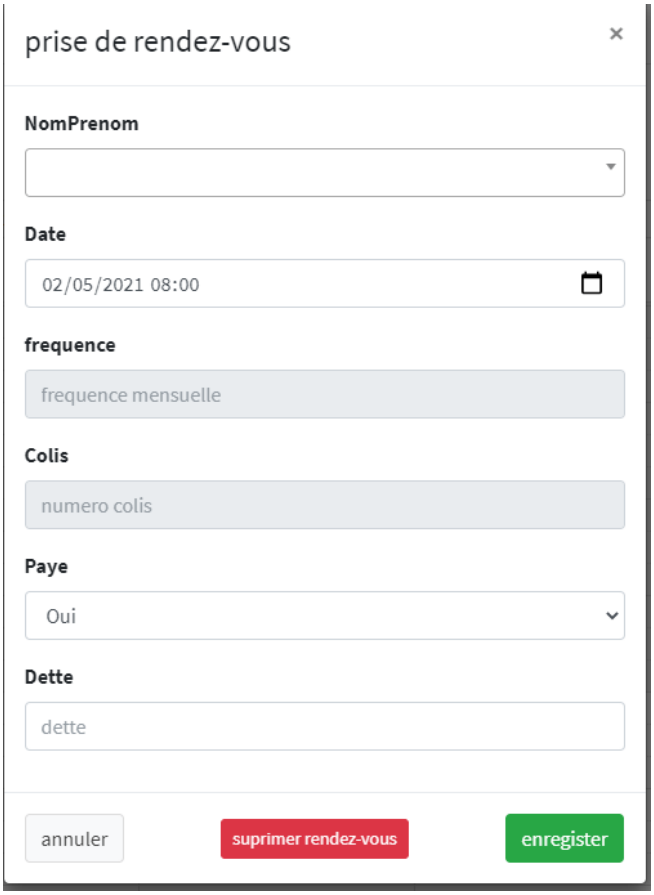
```
events: "/appointment/showappeintment",
eventSourceSuccess: function (content, xhr) {
  content.forEach((e) => console.log(e));
  console.log(content);
  return content.map((appointment) => ({
    id: appointment.id,
    title: appointment.adherents.NomPrenom,
    start: appointment.date,
  }));
},
```

b-prise/modification de rendez-vous

Dans cette partie, je passe les informations dans un modal JavaScript et j'envoie ma requête avec une fonction dans mon contrôleur

```
/**
 * @Route("/appointment/create", name="create_appointment", methods={"GET","POST"})
 * @Route("/appointment/{id}/edit", name="edit_appointment", methods={"GET","POST"})
 */
public function save(Request $request, Appointment $appointment = null)
{
    $form = $this->createForm(AppointmentType::class, $appointment);

    if ($form->handleRequest($request)->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($form->getData());
        $entityManager->flush();
        return $this->redirectToRoute('calandar');
    }

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

La fonction du FullCalendar dateClick me sert à prendre le rendez-vous en cliquant sur l'horaire et le jour choisi. J'inscris le nom et prénom du bénéficiaire dans le modal et l'heure est affichée automatiquement ainsi que les informations du formulaire.



Grâce à la bibliothèque jquery select2, je peux faire une datalist de mon entityType pour récupérer le nom/prénom des adhérents. En utilisant la bibliothèque axios et en passant par une requête, j'ai transformé les informations en format json. Je fais la même chose pour récupérer les informations d'un adhérent sans charger la page et sans modifier les données (readonly) à l'aide de son identifiant.

```php
/**
 * @Route("/appointment/howcolisfrequence/{id}", name="showafrequence")
 */


public function showcolisfrequence(AdherentsRepository $adherentsRepository, SerializerInterface $serializer,$id): Response
{

    $adherents = $adherentsRepository->find($id);



    $result = new Response($serializer->serialize($adherents, 'json', ['groups' => ['Appointment:list']]));

 return $result;

}
```

```javascript
$dialog.modal();
const result = document.getElementById(
  "select2-appointment_Adherents-container"
);
console.log(result);
result.addEventListener("change", (e) => {
  console.log(e.target.value);
});
$(".select2").on("select2:select", function (e) {
  // Do something
  axios
    .get(`/appointment/howcolisfrequence/${e.target.value}`)
    .then((result) => {
      var frequence = result.data.FrequenceMensuelle;
      var colis = result.data.Colis;
      console.log(frequence);
      console.log(colis);
      console.log(result.data);
      document.getElementById('frequence').value=frequence;
      document.getElementById('colis').value=colis;
      //$dialog.find("#appointment-form-container").write("toto");
    });
```

c-supprimer un rendez-vous

J'ai créé une fonction dans le JavaScript eventClick, elle me permet de supprimer un rendez-vous en passant l'url de la requête.

```
eventClick: function (calEvent, jsEvent, view) {
  console.log(calEvent);
  $dialog = $("#add-appointment-dialog");
  $dialog
    .find("#delete-appointment-link")
    .prop("href", `appointment/${calEvent.event.id}/delete`);
  $dialog.find("#appointment-form-container").empty();

  $dialog
    .find("#appointment-form-container")
    .load(`appointment/${calEvent.event.id}/edit`, () => {
      $dialog.modal();



    });
},
```

```
/**
 * @Route("appointment/{id}/delete", name="delete_appointment")
 */
public function delete(Request $request, Appointment $appointment): Response
{
    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->remove($appointment);
    $entityManager->flush();
    return $this->redirectToRoute('calandar');
}
```

5.9 Fiche journallière

J’ai créé dans le repository une requêtte personalisée où je recupère les adhérents qui ont un rendez vous ce jour et je la passe vers le contrôleur où j’ai créé une fonction

```php
public function showafiche(AppointmentRepository $appointmentRepository): Response
{

    $now = new \DateTime('now', new \DateTimeZone('Europe/Paris'));
    $Fiche = $appointmentRepository->findByfichejournaliere($now);



    return $this->render('appointment/fichejournaliere.html.twig', [
        'Fiche' => $Fiche,
    ]);
}
```

```php
public function findByfichejournaliere(\DateTimeInterface $date)
{
    $datestart = new \DateTime($date->format('Y-m-d') . '00:00:00', new \DateTimeZone('Europe/Paris')
    $dateend = new \DateTime($date->format('Y-m-d') . '23:59:59',  new \DateTimeZone('Europe/Paris'))
    return $this->createQueryBuilder('a')
        ->andWhere('a.date BETWEEN :datestart AND :dateend')
        ->setParameter('datestart', $datestart)
        ->setParameter('dateend', $dateend)
        ->getQuery()
        ->getResult();
}
```

Dans la vue j’ai effectué un boucle for dans TWIG (twig est un moteur de templates pour le langage de programation PHP)
J’ai ajouté un bouton qui me permet d’imprimer la fiche journalière.

5.10 Création login

J'ai d'abord créé une entité User , les infos sur l'utilisateur vont être enregistrées en base de données et c'est là bas que je recupére le login et mot de passe pour s'identifier .Pour créer l'entité User, j'ai utilisé la commande suivante :

```
$ php bin/console make:user
```

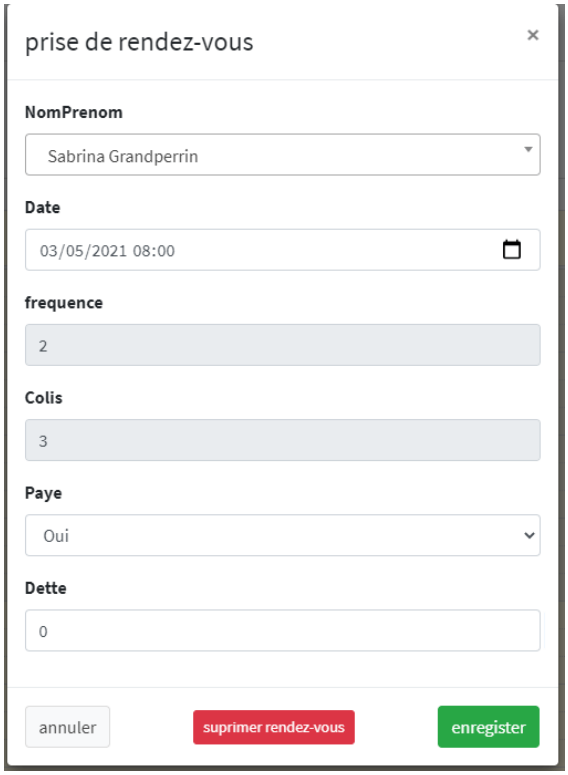
Une fois les questions terminées, la classe User a éte créé dans le fichier src/Entity/User.php
Maintenant que j'ai tous les attributs de l'entité, je transfert les données à la BDD

```
$ php bin/console doctrine:schema:update --force
```

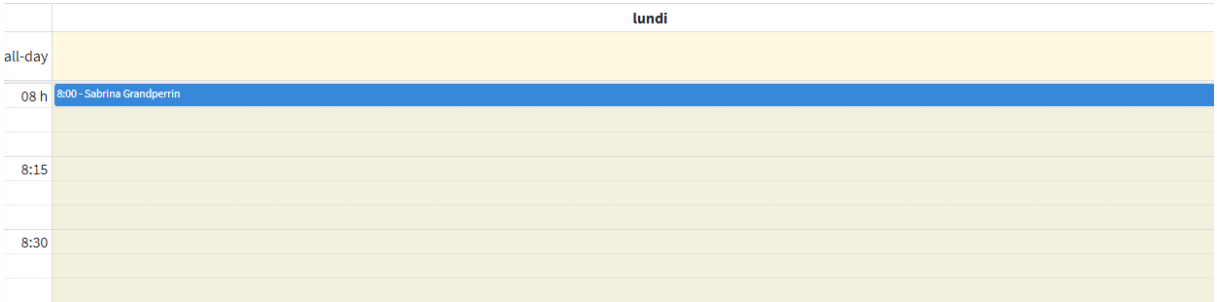
6. JEU D'ESSAI DE LA FONCTIONNALITÉ LA PLUS REPRÉSENTATIVE

J'ai choisi la prise de rendez-vous comme essai de la fonctionnalité :

En cliquant sur l'horaire et le jour choisi. J'inscris le nom et prénom du bénéficiaire dans le modal et l'heure est affichée automatiquement ainsi que les informations (numéro de colis, fréquence), en appuyant sur le bouton enregistrer(submit) j'envoie l'information à la base de données.



Et là, je vois bien le rendez-vous que j'ai pris a bien était ajouté sur mon agenda :



Et comme j'ai pris un rendez-vous pour aujourd'hui je recupère ce rendez-vous dans ma liste journalière :

Colis	Nom & Prenom	Date et heure	Frequence	Telephone	dossier
3	Sabrina Grandperrin	2021-05-03 08:00:00	2	0523148598	45201

7. Veille sur les vulnérabiltés de sécurité

PHP est le langage web le plus utilisé. La dernière montre que PHP est le langage web où il y a le plus de vulnérabilités.

En effet, sur l'ensemble du trafic malveillant en 2018, 81 % étaient liés au langage PHP. Ces chiffres sont cependant à nuancer, notamment au regard des pratiques et actions mises en place par 2le pour réduire au maximum les vulnérabilités PHP. Tout d'abord, il faut savoir que PHP est utilisé pour la création de plus de 80 % des sites web mondiaux. Il est donc logique que les hackers recherchent plus facilement les vulnérabilités de ce langage car les informations à dérober sont plus nombreuses. De plus, par sa facilité de compréhension, PHP est utilisé par beaucoup de débutants en informatique. Ces derniers n'ont pas encore l'expérience nécessaire pour sécuriser les applications.

L'utilisation de Symfony pour sécuriser les applications Le framework Symfony permet d'empêcher aisément de nombreuses failles de sécurité : grâce à Twig, nous empêchons les failles XSS. Le principe de cette faille est d'injecter un code malveillant en langage de script dans un site web vulnérable, par exemple en déposant un message dans un forum qui redirige l'internaute vers un faux site (phishing) ou qui vole vos informations (cookies). Un autre exemple : la gestion des formulaires de Symfony embarque un outil permettant de gérer les erreurs CRSF. Il s'agit d'effectuer une action visant un site ou une page précise en utilisant l'utilisateur comme déclencheur, sans qu'il en ait conscience.

Vulnérabilités PHP :

Injection SQL Pour éviter toute attaque par Injection SQL, l'outil que nous utilisons, doctrine, permet d'empêcher nativement les failles d'injection de la plupart des requêtes.

8.Description d'une situation ayant nécessité une recherche à partir de site anglophone

Pendant la durée de mon stage, j'ai consulté principalement 2 documentations officielles de Symfony, qui est rédigée en anglais https://symfony.com/ et FullCalendar https://fullcalendar.io/ .

Je me suis aidé également avec des vidéos « live coding » de nouvelle techno et Lior Chamla sur YouTube.

Lors de mon intégration du calendrier de FullCalendar, j'ai été bloqué par le fait que je voulais faire un planning de rendez-vous toutes les 5 min et cela ne fonctionnait pas.

J'ai dû regarder dans le fichier JavaScript et la solution était juste un réglage des minutes dans cette ligne de code :

```
defaultTimedEventDuration: "00:05:00",
```

9. Extrait du site anglophone et traduction

The Serializer Component

single: Serializer single: Components; Serializer

The Serializer component is meant to be used to turn objects into a specific format (XML, JSON, YAML, ...) and the other way around.

In order to do so, the Serializer component follows the following schema.

As you can see in the picture above, an array is used as an intermediary between objects and serialized contents. This way, encoders will only deal with turning specific formats into arrays and vice versa. The same way, Normalizers will deal with turning specific objects into arrays and vice versa.

Serialization is a complex topic. This component may not cover all your use cases out of the box, but it can be useful for developing tools to serialize and deserialize your objects

Traduction :

Dans ce projet, à l'aide d'une requête, j'ai transformé des objets dans un format JSON

Le composant Serializer est destiné à être utilisé pour transformer des objets dans un format spécifique (XML, JSON, YAML, ...) et inversement.

10. Annexes

Conception base de données