EPSI ARRAS SN2

SHARE

Nom-Prénom des personnes de l'équipe : POTEAUX Agathe, BRUYE Antoine, PARISOT Clément et BOUDRY Mélanie en SN2 EPSI ARRAS

Année 2023 - 2024

TABLE DES MATIERES

Lien du Trello pour les deux : https://trello.com/b/WBz35VhO/share
Lien GitHub application web : https://github.com/MelB04/Share
Lien du site web complet de l'équipe : https://s4-8057.nuage-peda.fr/share/
Lien GitHub application mobile : https://github.com/SuperKiment/share-app

1 L'APPLICATION SHARE

Share est un site web permettant de partager des fichiers entre utilisateurs. Par la suite, afin d'augmenter sa portée et de la rendre accessible sur mobile, nous avons décidé de créer une version mobile sous le même nom, en utilisant React Native. Cette application mobile offre les mêmes fonctionnalités que sa version web : les utilisateurs peuvent partager leurs fichiers avec d'autres personnes et gérer leurs partages à la fois sur le site web et sur l'application. De plus, l'application mobile propose une fonctionnalité supplémentaire : un espace d'échange sous forme de forum, permettant une interaction plus dynamique entre les utilisateurs.

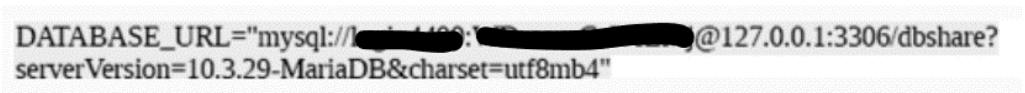
Langages utilisés :

- Pour le web :
  - Symfony
  - phpMyAdmin
- Pour la version mobile :
  - React Native
  - Api Plateform

2 LA BASE DE DONNEES SHARE

2.1 COMMENT CREER UNE BASE DE DONNEES ?

Il faut faire une copie du .env en .env.local.



Sur cette ligne, on doit spécifier mon login, mon mot de passe ainsi que le nom de la base de données.

Ligne de commandes :

- php bin/console doctrine:database:create : créer la base de données
- php bin/console doctrine:database:drop –force : si on veut la supprimer

- php bin/console make:entity : créer une nouvelle entité et répondre aux différentes questions (nom de l'attribut, la taille, les orphelins...)
- php bin/console make:migration : préparer la migration
- php bin/console doctrine:migrations:migrate : mettre à jour la BDD

Méthodes pour une entité :

- Méthode qui retourne la valeur nulle ou un contact dont l'identifiant sera passé en paramètre

  @method Contact|null find($id, $lockMode = null, $lockVersion = null)

- Méthode qui retourne la valeur nulle ou un contact par rapport à un ensemble de critères que nous pouvons passer en paramètre

  @method Contact|null findOneBy(array $criteria, array $orderBy = null)

- Méthode qui retourne un tableau comprenant tous les contacts présents dans la table « Contact »

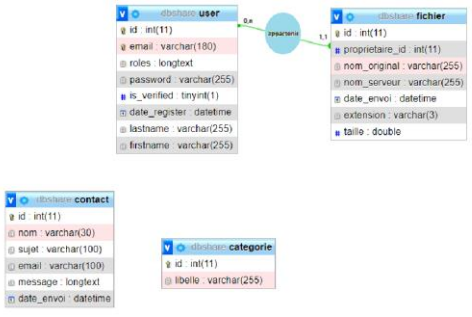
  @method Contact[] findAll()

- Méthode qui retourne un tableau comprenant tous les contacts présents dans la table « Contacts » qui vérifient des critères passés en paramètre.

  @method Contact[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)

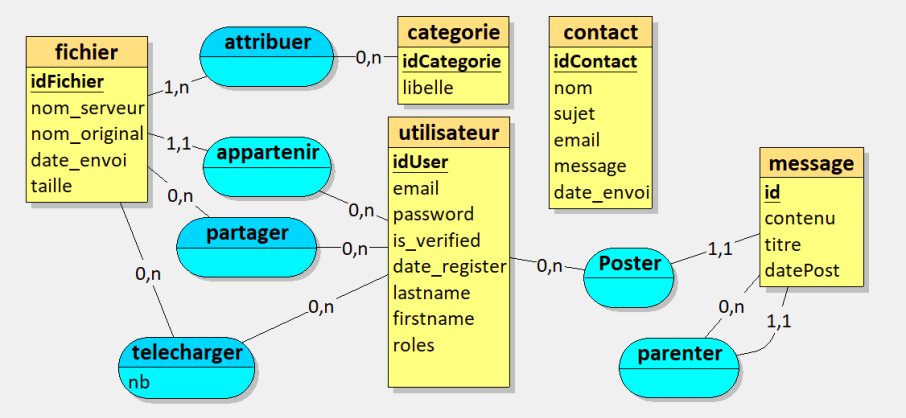
Relations :

- ManyToMany : Dans une relation ManyToMany, chaque entité d'une classe peut être liée à plusieurs entités d'une autre classe, et vice versa. Par exemple, une relation entre les entités Product et Category, où un produit peut appartenir à plusieurs catégories et une catégorie peut contenir plusieurs produits.
- ManyToOne : Dans une relation ManyToOne, chaque instance d'une entité appartient à exactement une instance d'une autre entité. Par exemple, une relation entre User et Fichier, où chaque fichier est associé à un seul user. Faire l'ajout de relation dans l'entité Fichier en ajoutant un attribut avec comme type ManyToOne.

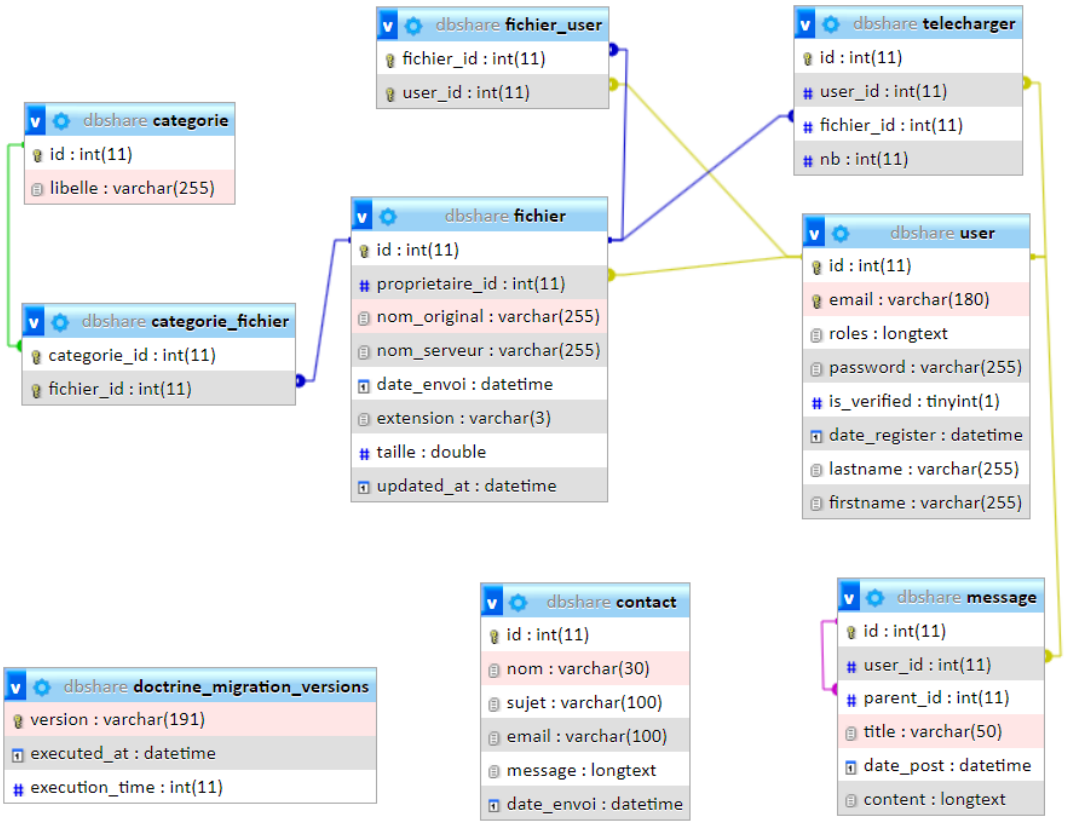


- Pour une association x,n x,n avec un attribut, créer l'entité avec des relation ManyToOne.

2.2 LE MCD :



2.3 LE MLD :



Nous avons mis en place l'entité message pour l'application mobile.

3 L'APPLICATION WEB

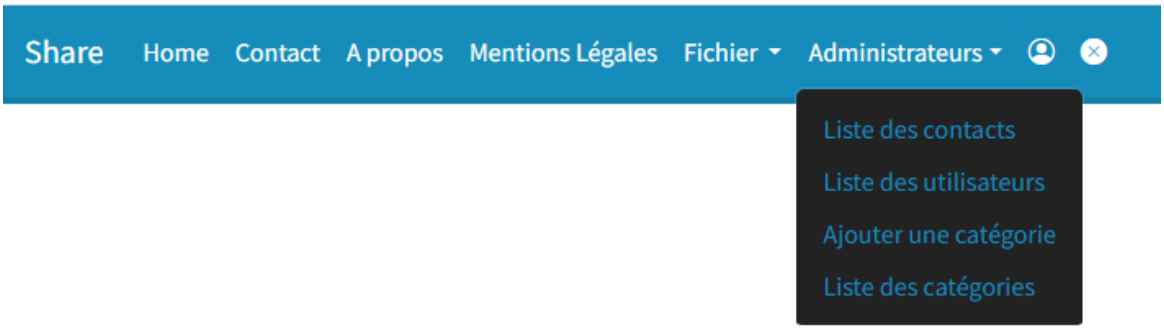
- php bin/console make:form
- php bin/console make:controller
- php bin/console make:controller NomDuController
- composer create-project symfony/skeleton share 6.3.*

Nous avons mis en place le projet en intégrant Bootstrap. Nous avons ajouté une barre de menu en fonction du rôle de la personne présente sur le site (visiteur, administrateur connecté et simple utilisateur connecté).

Simple visiteur :

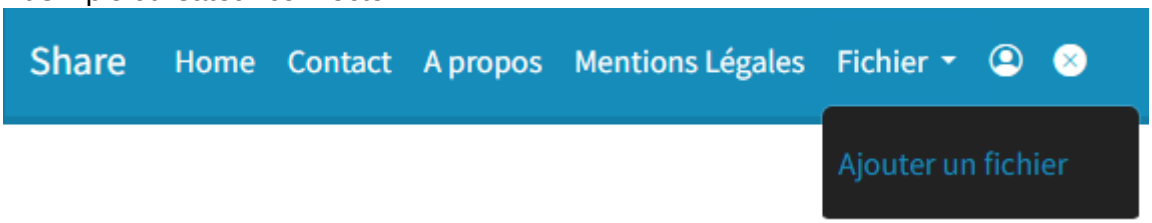

```
<li class="nav-item">
    <a class="nav-link active" href="{{path('app_accueil')}}">Home
        <span class="visually-hidden">(current)</span>
    </a>
</li>
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_contact')}}">Contact</a>
</li>
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_apropos')}}">A propos</a>
</li>
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_mentionslegales')}}">Mentions Légales</a>
</li>
{% if not is_granted('IS_AUTHENTICATED_FULLY') %}
    <li class="nav-item">
        <a class="nav-link text-white" href="{{path('app_login')}}">Se connecter
        </a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-white" href="{{path('app_register')}}">S'inscrire
        </a>
    </li>
```

Administrateur connecté :




```
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle text-white" href="#" id="navbarScrollingDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
        Fichier
    </a>
    <ul class="dropdown-menu" aria-labelledby="navbarScrollingDropdown">
        <li class="nav-item">
            <a class="nav-link text-primary" href="{{path('app_profil-ajout-fichier')}}">Ajouter un fichier
            </a>
        </li>
    </ul>
</li>
{% if is_granted('ROLE_ADMIN') or is_granted('ROLE_MOD')%}
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle text-white" data-bs-toggle="dropdown" href="#" role="button" aria-haspopup="true" aria-expanded="false">Administrateurs</a>
        <div class="dropdown-menu">
            <a class="dropdown-item text-primary" href="{{path('app_liste_contacts')}}">Liste des contacts</a>
            <a class="dropdown-item text-primary" href="{{path('app_liste_users')}}">Liste des utilisateurs</a>
            <a class="dropdown-item text-primary" href="{{path('app_categorie')}}">Ajouter une catégorie</a>
            <a class="dropdown-item text-primary" href="{{path('app_liste_categories')}}">Liste des catégories</a>
        </div>
    </li>
{% endif %}
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_profil')}}">
        <i class="bi bi-person-circle"></i>
    </a>
</li>
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_logout')}}">
        <i class="bi bi-x-circle-fill text-white"></i>
    </a>
</li>
```

Et simple utilisateur connecté :



Le simple utilisateur ne dispose pas de l'affichage de la liste déroulante consacré aux administrateurs.

En plus de tout cela, nous avons mis en place notre page d'accueil.

3.1 MISE EN PLACE DE LA SECURITE

Pour avoir accès à un plus grand nombre de fonctionnalités et avoir accès au but de l'application, une authentification avec email et mot de passe est requise. Cette authentification se base donc sur un espace de connexion, d'inscription et de déconnexion.

Certaines actions expliqué prochainement sont des actions avec un certain droit d'accès (tout le monde ou seulement administrateur).

Symfony propose des outils permettant de gérer simplement et rapidement la sécurité.

```
composer require symfony/security-bundle
```

```php
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['user:list', 'user:item'])]
    private ?int $id = null;

    #[ORM\Column(length: 180, unique: true)]
    #[Groups(['user:list', 'user:item'])]
    private ?string $email = null;

    #[ORM\Column]
    #[Groups(['user:item'])]
    private array $roles = [];

    /**
     * @var string The hashed password
     */
    #[ORM\Column]
    private ?string $password = null;

    #[ORM\Column(type: 'boolean')]
    private $isVerified = false;

    #[ORM\Column(type: Types::DATETIME_MUTABLE)]
    private ?\DateTimeInterface $dateRegister = null;
```

Nous allons devoir créer une entité spéciale qui fait appel à « UserInterface », qui est une interface proposant de gérer la connexion, le mot de passe à notre place...

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

Il nous faut également créer une classe User qui va stocker nos utilisateurs dans la base de données. C'est l'email qui permet de se connecter. On hache les mots de passe. On met tous les attributs que l'on souhaite.

Une entité « User » et le « UserRepository » ont été créé.

Ensuite on fait une mise à jour de la base de données.

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

3.1.1 L'authentification

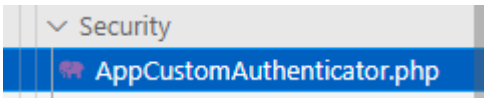
Symfony nous permet de créer très facilement le formulaire de connexion via le package :

```
composer require symfony/maker-bundle --dev
```

Et ensuite la commande :

```
php bin/console make:auth
```

Il nous faut ensuite sélectionner le type d'authentification, on choisit « Login form authenticator ». On accepte le nom proposé. Ensuite, on crée un contrôleur pour la sécurité. On accepte par la même occasion de créer la fonctionnalité permettant de se déconnecter avec l'url « /logout ».



Ainsi, à la suite de cette commande, nous avons créé :

- Le fichier « App\Security\AppCustomAuthenticator.php » qui va personnaliser le comportement de la connexion.
- « SecurityController » dans « src\controller ».
- Le fichier « templates/security/login.html.twig » pour la vue du formulaire de connexion.

3.1.1.1 La connexion

Une nouvelle route a donc été créé pour se connecter (/login).

```php
#[Route(path: '/login', name: 'app_login')]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // if ($this->getUser()) {
    //     return $this->redirectToRoute('target_path');
    // }

    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();
    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

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

Si la connexion échoue, le contrôleur récupère l'erreur qui la stocke dans $error. Il récupère également l'email saisi lors de la tentative de connexion qui la stocke dans $lastUsername. Il envoie ces 2 nouvelles variables à la vue (au fichier TWIG) pour les afficher.

Un template a par la même occasion été créé. Nous avons le modifier pour le mettre en forme.

```twig
<form method="post">
    {% if error %}
        <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
    {% endif %}

    {% if app.user %}
        <div class="mb-3">
            You are logged in as
            {{ app.user.lastname }},
            <a href="{{ path('app_logout') }}">Logout</a>
        </div>
    {% endif %}
```

```
    <label for="inputEmail" class="fw-bold">Email</label>
    <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control" autoc
    <label for="inputPassword" class="fw-bold">Mot de passe</label>
    <input type="password" name="password" id="inputPassword" class="form-control" autocomplete="current-pa

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

    {#
        Uncomment this section and add a remember_me option below your firewall to activate remember me
        See https://symfony.com/doc/current/security/remember_me.html

        <div class="checkbox mb-3">
            <label>
                <input type="checkbox" name="_remember_me"> Remember me
            </label>
        </div>
    #}
    <div class="text-center">
    <button class="btn bg-primary text-white m-4" type="submit">
        CONNEXION
    </button>
    </div>
</form>
```

Voici le visuel de ce dernier :

Des messages d'erreur s'affichent en cas de problèmes sur l'authentification.

3.1.1.2 Inscription

On installe le composant permettant d'écrire des règles de validation pour les formulaires.

```
composer require validator
```

Ensuite on crée un nouveau formulaire

```
php bin/console make:registration-form
```

Ensuite, on doit répondre à quelques questions :

- Un compte est unique dans notre base de données.

- On souhaite envoyer un email de confirmation après inscription. (avec l'email de l'émetteur.
- On ne souhaite pas connecter la personne automatiquement après la validation. En mentionnant la route de redirection.

Pour pouvoir envoyer des emails de confirmation, on installe ce package :

```
composer require symfonycasts/verify-email-bundle
```

On a, à la suite de ça, un nouveau controleur appelé « RegistrationController.php » dans laquelle on a deux fonctions (/register et /verify/email).

```php
#[Route('/register', name: 'app_register')]
public function register(Request $request, UserPasswordHasherInterface $userPasswordHashe
{
    $user = new User();
    $form = $this->createForm(RegistrationFormType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $user->setDateRegister(new \Datetime());
        // encode the plain password
        $user->setPassword(
            $userPasswordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );

        $entityManager->persist($user);
        $entityManager->flush();

        // generate a signed url and email it to the user
        $this->emailVerifier->sendEmailConfirmation('app_verify_email', $user,
            (new TemplatedEmail())
                ->from(new Address('melanie.boudry@ecoles-epsi.net', 'Melanie BOUDRY'))
                ->to($user->getEmail())
                ->subject('Please Confirm your Email')
                ->htmlTemplate('registration/confirmation_email.html.twig')
        );
        // do anything else you need here, like send an email

        return $this->redirectToRoute('app_login');
    }

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

```php
#[Route('/verify/email', name: 'app_verify_email')]
public function verifyUserEmail(Request $request, UserRepository $userRepository): Re
{
    $id = $request->query->get('id');

    if (null === $id) {
        return $this->redirectToRoute('app_register');
    }

    $user = $userRepository->find($id);

    if (null === $user) {
        return $this->redirectToRoute('app_register');
    }

    // validate email confirmation link, sets User::isVerified=true and persists
    try {
        $this->emailVerifier->handleEmailConfirmation($request, $user);
    } catch (VerifyEmailExceptionInterface $exception) {
        $this->addFlash('verify_email_error', $exception->getReason());

        return $this->redirectToRoute('app_register');
    }

    // @TODO Change the redirect on success and handle or remove the flash message in
    $this->addFlash('success', 'Your email address has been verified.');

    return $this->redirectToRoute('app_login');
}
```

Ensuite, on a également un formulaire automatiquement créé permettant de nous inscrire (« src/Form/RegistrationFormType.php ») avec les champs nécessaires :

```php
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
        ->add('email', EmailType::class, ['attr' => ['class'=> 'form-control'], 'label_attr' => ['class'=> 'fw-bold']])
        ->add('lastname', TextType::class, ['label' => 'Nom','attr' => ['class'=> 'form-control'], 'label_attr' => ['clas
        ->add('firstname', TextType::class, ['label' => 'Prénom','attr' => ['class'=> 'form-control'], 'label_attr' => ['
        ->add('agreeTerms', CheckboxType::class, [
                'label' => 'J\'accepte les termes',
                'label_attr' => ['class'=> 'fw-bold pe-4 pt-4'],
                'mapped' => false,
                'constraints' => [
                    new IsTrue([
                        'message' => 'You should agree to our terms.',
                    ]),
                ],
            ])
```

```
        ->add('plainPassword', PasswordType::class, [
                'label_attr' => ['class'=>'fw-bold'],
                // instead of being set onto the object directly,
                // this is read and encoded in the controller
                'mapped' => false,
                'attr' => ['autocomplete' => 'new-password','class'=> 'form-control' ],
                'constraints' => [
                    new NotBlank([
                        'message' => 'Please enter a password',
                    ]),
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Your password should be at least {{ limit }} characters',
                        // max length allowed by Symfony for security reasons
                        'max' => 4096,
                    ]),
                ],
            ])
            ->add('Register', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-white m-4' ],'row_attr' => ['

        ;
    }
```

Les contraintes possibles :

- new IsTrue qui impose que la case soit cochée sinon on affiche le 'message' => 'You should agree to our terms.',
- NotBlank qui impose la saisie d'une donnée.
- Length qui impose une condition sur la longueur d'une chaine de caractères. « Min » permet de donner le nombre minimum de caractères

Enfin, la vue de l'interface d'inscription se trouve dans les templates « registration/register.html.twig » :

```
<div class="container-fluid">
    <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">Inscription</h1>

    <div class="row justify-content-center">
        <div class="col-12 col-md-6 bg-white p-4 m-0 text-primary">

            {% for flash_error in app.flashes('verify_email_error') %}
                <div class="alert alert-danger" role="alert">{{ flash_error }}</div>
            {% endfor %}

            {{ form_errors(registrationForm) }}

            {{ form_start(registrationForm) }}
            {{ form_row(registrationForm.email) }}
            {{ form_row(registrationForm.lastname) }}
            {{ form_row(registrationForm.firstname) }}
            {{ form_row(registrationForm.plainPassword, {
        label: 'Mot de Passe'
    }) }}
            {{ form_row(registrationForm.agreeTerms) }}

            {{ form_end(registrationForm) }}

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

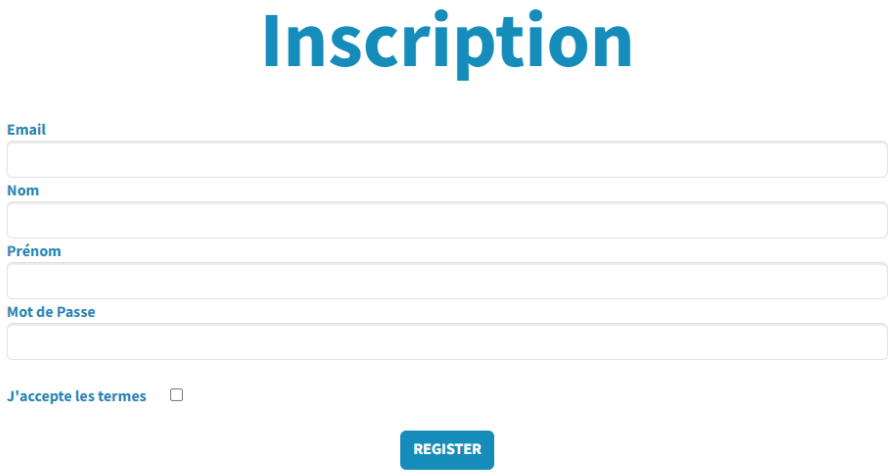
L’email de confirmation se trouve dans le template « /confirmation_email.html.twig ».

```
<h1>Hi! Please confirm your email!</h1>

<p>
    Please confirm your email address by clicking the following link: <br><br>
    <a href="{{ signedUrl|raw }}">Confirm my Email</a>.
    This link will expire in {{ expiresAtMessageKey|trans(expiresAtMessageData, 'VerifyEmailBundle
</p>

<p>
    Cheers!
</p>
```

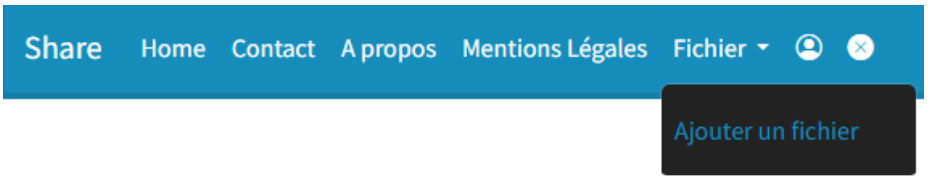
Concernant l’envoi de l’email, c’est la classe « EmailVerifier » qui se situe dans « src/Security/EmailVerifier.php » qui s’en charge.

```
public function sendEmailConfirmation(string $verifyEmailRouteName, UserInterface $user,
{
    $signatureComponents = $this->verifyEmailHelper->generateSignature(
        $verifyEmailRouteName,
        $user->getId(),
        $user->getEmail(),
        ['id' => $user->getId()]
    );

    $context = $email->getContext();
    $context['signedUrl'] = $signatureComponents->getSignedUrl();
    $context['expiresAtMessageKey'] = $signatureComponents->getExpirationMessageKey();
    $context['expiresAtMessageData'] = $signatureComponents->getExpirationMessageData();

    $email->context($context);

    $this->mailer->send($email);
}
```

Voici le visuel de la page d’inscription :



Lorsqu'on est connecté, la navbar de la page se change.



3.1.2 Déconnexion

Pour ce qui est de la déconnexion, nous avons simplement ajouté la redirection vers la déconnexion (/app_logout).

```
<li class="nav-item">
    <a class="nav-link text-white" href="{{path('app_logout')}}">
        <i class="bi bi-x-circle-fill text-white"></i>
    </a>
</li>
```

Cette dernière va utiliser la fonction /logout.

```
#[Route(path: '/logout', name: 'app_logout')]
public function logout(): void
{
    throw new \LogicException('This method can be blank - it wil
}
```

3.2 LES ROLES ET FONCTIONNALITES POSSIBLES

Dans notre application, nous avons 3 types d'utilisateurs :

- L'administrateur qui a accès à toutes les pages du site, il a le droit de manipuler les données. Il a les droits du modérateur et de l'utilisateur. Il est au-dessus de tout le monde.
- Le modérateur qui aura le rôle utilisateur plus celui de pouvoir consulter la liste des contacts.
- L'utilisateur qui pourra gérer seulement ses données.

Nom	Rôle	Le rôle comprend les droits :
Administrateur	ROLE_ADMIN	ROLE_MOD et ROLE_USER
Modérateur	ROLE_MOD	ROLE_USER
Utilisateur	ROLE_USER	

Nous pouvons maintenant mettre en place une hiérarchie des utilisateurs dans les droits en paramétrant le fichier « /packages/security.yaml ».

```yaml
role_hierarchy:
    ROLE_MOD: [ROLE_USER]
    ROLE_ADMIN: [ROLE_MOD]
```

Dans ce même fichier, il nous reste à spécifier les pages accessibles pour chaque rôle.

```yaml
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
    - { path: ^/private, roles: ROLE_MOD }
```

Un utilisateur inscrit aura automatiquement un « ROLE_USER ».

```php
/**
 * @see UserInterface
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```

Pour la navbar, il suffit de mettre des conditions :

```twig
{% if is_granted('ROLE_ADMIN')
or is_granted('ROLE_MOD')%}
{% endif %}
```

On peut se baser sur l'URL pour protéger une page.

Pour cela, il suffit d'ajouter le mot « private » au début de l'URL.

```php
#[Route('/private-liste-contacts', name: 'app_liste_contacts')]
public function listeContacts(ContactRepository $contactRepository
```

L'administrateur peut visualiser l'ensemble des personnes inscrites à l'application.

Liste des utilisateurs

Email	Date d'inscription	Rôle	Vérifié
Ajdhde	22-04-2024 à 13:19:50	Utilisateur	Non validé
Laurence.leroy@live.com	24-03-2024 à 07:17:34	Utilisateur	Oui
Luc.joseph@hotmail.fr	17-03-2024 à 15:20:08	Utilisateur	Oui
Dominique.thomas@hotmail.fr	07-03-2024 à 17:05:49	Utilisateur	Oui
Élise.lemaire@laposte.net	26-02-2024 à 11:45:06	Utilisateur	Oui
Louise.didier@orange.fr	24-02-2024 à 00:30:11	Utilisateur	Oui

Le principe est le même que l'explication 3.3.2.

3.3 LES CONTACTS

A travers l'application web, il est possible de contacter les administrateurs en leur passant un message. Cela est possible qu'on soit connecté ou non. Chaque prise de contact est conservée en base de données. Nous avons créé un dossier « contact » dans le dossier « templates » pour segmenter toutes les actions.

3.3.1 Ajout de contact

Nous avons commencé par créer un contrôleur ContactController.php dans le dossier « src » et un nouveau template « contact.html.twig » dans le dossier « Contact » du dossier « templates ».

Pour créer le formulaire, il nous faut utiliser les commandes :

- composer require symfony/form (installation du package formulaire)
- php bin/console make:form (création du formulaire) appelé « Contact »
- Relié le formulaire à une entité appelé « Contact »

Un fichier « ContactType.php » a donc été créé dans le dossier « src/Form ». Il nous faut simplement ajouter du style au élément et un bouton d'envoi. La méthode buildForm va nous permettre d'ajouter tous les composants nécessaires à la création du formulaire.

On associe également tous les champs libellés du formulaire avec un type de composant.

```php
class ContactType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nom', TextType::class, ['attr' => ['class'=> 'form-control'], 'label_attr' =>
            ->add('sujet', TextType::class, ['attr' => ['class'=> 'form-control'], 'label_attr' =
            ->add('email', EmailType::class, ['attr' => ['class'=> 'form-control'], 'label_attr'
            ->add('message', TextareaType::class, ['attr' => ['class'=> 'form-control', 'rows'=>'
            ->add('envoyer', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-white
        ;
    }

    public function configureOptions(OptionsResolv
    {
        $resolver->setDefaults([
            'data_class' => Contact::class,
        ]);
    }
}
```

Nom du composant Symfony
TextType
TextType
EmailType
TextareaType

Ensuite, l'intégration de ce dernier se fait très facilement :

- Dans le controleur :

Nous importons la classe « ContactType » avec un « use » tout en haut du fichier.

```php
use App\Form\ContactType;
```

Et ensuite dans une fonction appelé « contact », on crée le formulaire en lui donnant la nouvelle classe créée. Le formulaire sera stocké dans la variable « $form ».

```php
#[Route('/contact', name: 'app_contact')]
public function contact(Request $request, EntityManagerInterface $em): Response
{
    $contact = new Contact();
    $form = $this->createForm(ContactType::class, $contact);
```

Ensuite on donne le formulaire généré en code HTML (createView()) à la vue lorsque nous appelons le fichier TWIG.

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

- Dans le template (ce qui est visible sur la page web) :

Dans le fichier correspondant, on appelle le formulaire en utilisant la variable utilisé lors de l'envoi au TWIG.

```twig
{% block body %}
<div class="container-fluid">
    <div class="row justify-content-center">
        <div class="col-12 col-md-5 text-primary">
            <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">CONTACT</h1>
            {{form(form)}}
        </div>
    </div>
</div>
```

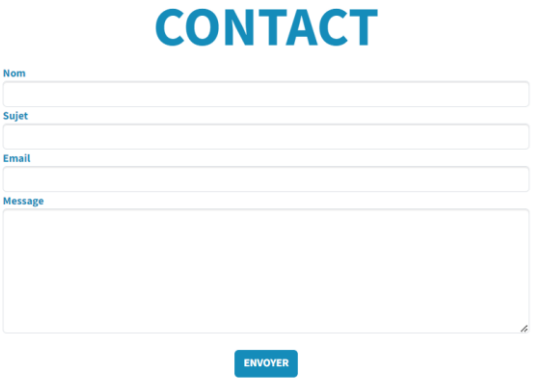
Pour récupérer les données du formulaire, on installe le composant « Request » qui va contenir les informations sur la requête HTTP qu'on passe ensuite en paramètre de la fonction « contact ».

```php
use Symfony\Component\HttpFoundation\Request;
```

On déclare la variable $contact = new Contact() étant l'objectif du formulaire. On passe cette variable au formulaire, de cette manière au retour de formulaire les données remplis seront associé à ceux de l'entité contact.

Si on appuie sur le bouton d'envoi, on vérifie que la requête est bien de type POST et que le formulaire a bien été envoyé et est valide. A l'intérieur de cette condition, nous préparons l'ajout, le confirmons et le réalisons. Après cela, un nouveau contact a été ajouté en base de données. Apres l'ajout des données en base de données, un message s'affiche sur une nouvelle page de formulaire de contact vierge.

```php
#[Route('/contact', name: 'app_contact')]
public function contact(Request $request, EntityManagerInterface $em): Response
{
    $contact = new Contact();
    $form = $this->createForm(ContactType::class, $contact);

    if($request->isMethod('POST')){
        $form->handleRequest($request);
        if ($form->isSubmitted()&&$form->isValid()){
            $contact->setDateEnvoi(new \Datetime());
            $em->persist($contact);
            $em->flush();
            $this->addFlash('notice','Message envoyé');
            return $this->redirectToRoute('app_contact');
        }
    }

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



3.3.2 Visualisation de l'ensemble des contacts

L'administrateur connecté a la possibilité de voir l'ensemble de demandes de contact faites à travers le formulaire expliqué précédemment.

On ajoute une nouvelle fonction dans le contrôleur de Contact, qui s'appelle listeContact en spécifiant l'url, la route ainsi que le fichier TWIG créé.

Pour récupérer l'ensemble des contacts dans la table Contact de la base de données, il faut importer le ContactRepository et le manager d'entité dans le fichier.

```
use App\Entity\Contact;
use Doctrine\ORM\EntityManagerInterface;
```

La Méthode Doctrine utilisé pour retourner un tableau comprenant tous les contacts présents dans la table Contact est findAll(). Et enfin on envoie à la vue la variable « contacts » avec tous les contacts.

```
#[Route('/private-liste-contacts', name: 'app_liste_contacts')]
public function listeContacts(ContactRepository $contactRepository): Response
{
    $contacts = $contactRepository->findAll();
    return $this->render('contact/index.html.twig', [
        'contacts' => $contacts
    ]);
}
```

Et ensuite, pour le visuel, j'ai créé un tableau qui affichait chaque contact avec les informations avec un tri.

```
<table class="table table-hover">
    <thead>
        <tr class="fw-bold text-primary">
            <th scope="col">Nom</th>
            <th scope="col">Sujet</th>
            <th scope="col">Email</th>
            <th scope="col">Date d'envoi</th>
        </tr>
    </thead>
    <tbody>
        <!-- si le a est inferieur à b alors on fait ca et ensuite ca -->
        {% for contact in contacts |sort((a, b) => b.dateEnvoi <=> a.dateEnvoi) %}
        <tr class="{{ cycle(['table-primary','table-secondary'], loop.index0) }}">
            <td>{{ contact.nom |capitalize }}</td>
            <td>{{ contact.sujet | capitalize }}</td>
            <td>{{ contact.email | lower }}</td>
            <td>{{ contact.dateEnvoi | date("d-m-Y à H:i:s") }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

3.4 LES CATEGORIES

Nous avons créé un contrôleur pour gérer l'ensemble des actions effectuées sur des catégories. Ces actions sont accessibles que par les administrateurs de l'application.

3.4.1 L'ajout des catégories

J'ai fait exactement la même chose que pour l'ajout de contact. (3.3.1)


```php
#[Route('/private-addCategorie', name: 'app_categorie')]
public function categorie(Request $request, EntityManagerInterface $em): R
{
    $categorie = new Categorie();
    $form = $this->createForm(CategorieType::class, $categorie);

    if($request->isMethod('POST')){
        $form->handleRequest($request);
        if ($form->isSubmitted()&&$form->isValid()){
            $categorie -> setLibelle(ucfirst(strtolower($categorie->getLibelle())));

            $em->persist($categorie);
            $em->flush();
            $this->addFlash('notice','Catégorie créée');
            return $this->redirectToRoute('app_categorie');
        }
    }

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

```php
class CategorieType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('libelle', TextType::class, ['attr' => ['class'=> 'form-control']
            ->add('envoyer', SubmitType::class, ['attr' => ['class'=> 'btn bg-prima
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Categorie::class,
        ]);
    }
}
```

3.4.2 Le tableau des catégories

J'ai également fait la même chose que la partie 3.3.2.

L'administrateur connecté a la possibilité de voir l'ensemble de catégories.

```
#[Route('/private-liste-categories', name: 'app_liste_categories', methods: ['GET', 'POST'])]
public function listeCategories(CategorieRepository $categoriesRepository, Request $request, E
{
    $categories = $categoriesRepository->findAll();

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

3.4.3 La modification d'une catégorie

Nous avons donc ajouté une nouvelle fonction dans le contrôleur Categorie ainsi qu'un template. On ajoute maintenant dans le tableau précédemment créé une nouvelle colonne qui fait référence aux actions (modifier et supprimer). Nous incluons dans cette colonne un bouton <a> icône permettant de modifier cette catégorie, en lui transmettant l'identifiant correspondant en paramètre.

```
<td><a href="{{path('app_modif_categorie', {'id':categorie.id})}}" class="text-black"><i class="bi bi-pencil-square"></i></a></td>
```

On réutilise le formulaire créé pour l'ajout de catégorie. Mais on pourrait en recréer un tout neuf.

```php
#[Route('/private-modif-categorie/{id}', name: 'app_modif_categorie')] #il sait maintenant qu'il faut
public function modifCategorie(Categorie $categorie, Request $request, EntityManagerInterface $em): R
{
    $form = $this->createForm(CategorieType::class, $categorie); #il recupere les informations en pré

    if($request->isMethod('POST')){
        $form->handleRequest($request);
        if ($form->isSubmitted()&&$form->isValid()){
            $categorie -> setLibelle(ucfirst(strtolower($categorie->getLibelle())));

            $em->persist($categorie); #ajouter ou modif
            $em->flush();
            $this->addFlash('notice','Catégorie modifiée');
            return $this->redirectToRoute('app_liste_categories');
        }
    }

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

Le code de l'ajout et de la modification est identique et se fait avec un « persist ». Un message s'affiche pour signaler une erreur ou alors la réussite de l'action. Ensuite on redirige vers la liste des catégories si réussite.


```twig
{% block title %}{{parent()}}Modifier une catégorie{% endblock %}

{% block body %}
<div class="container-fluid">
    <div class="row justify-content-center">
        <div class="col-12 col-md-5 text-primary">
            <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">Modifier une catégorie</h1>
            {{form(form)}}
        </div>
    </div>
</div>
```

Le visuel :

3.4.4 La suppression d'une ou des catégories

3.4.4.1 Une catégorie

On ajoute une nouvelle et dernière fonction dans le contrôleur. Et on ajoute un icone dans le tableau de la liste des catégories.

```
<td><a href="{{path('app_del_categorie', {'id':categorie.id})}}" class="text-black"><i class="bi bi-trash3-fill"></i></a></td>
```

On vérifie que la catégorie existe puis on supprime cette dernière avec la méthode remove(). On est ensuite redirigé vers la liste des catégories avec un message d'action.

```
#[Route('/private-del-categorie/{id}', name: 'app_del_categorie')]
public function delCategorie(Categorie $categorie, EntityManagerInterface $em): Response
{
    if($categorie!=null){
        $em->remove($categorie);
        $em->flush();
        $this->addFlash('notice','Catégorie supprimée');
    }
    return $this->redirectToRoute('app_liste_categories');
}
```

3.4.4.2 Plusieurs catégories à la fois

Pour supprimer plusieurs catégories à la fois avec des cases à cocher, on va créer un nouveau formulaire sans entité.

- php bin/console make:form

```
class SupprimerCategorieType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
        ->add('categories', EntityType::class, [
            'class' => Categorie::class,
            'choices' => $options['categories'],
            'choice_label' => 'libelle',
            'expanded' => true,
            'multiple' => true,
            'label' => false, 'mapped' => false])
        ->add('supprimer', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-white m-4' ],
            'row_attr' => ['class' => 'text-center'],]);
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'categories' => []
        ]);
    }
}
```

Ce formulaire ne s'appuie pas sur une entité mais il va recevoir une liste de catégories. Nous mettons le composant en tant que EntityType pour retrouver facilement les enregistrements d'une entité précise.

Nous modifions aussi notre contrôleur pour lister les catégories. On importe le formulaire, on fait la même chose que pour une catégorie, on va simplement avant vérifier que le formulaire a bien été envoyé, récupérer les données des catégories sélectionnées dans le formulaire. On boucle sur cette liste de catégories sélectionnées pour les supprimer une à une.

```php
#[Route('/private-liste-categories', name: 'app_liste_categories', methods: ['GET', 'POST'])]
public function listeCategories(CategorieRepository $categoriesRepository, Request $request, E
{
    $categories = $categoriesRepository->findAll();
    $form = $this->createForm(SupprimerCategorieType::class, null, [
        'categories' => $categories
        ]);
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $selectedCategories = $form->get('categories')->getData();
        foreach ($selectedCategories as $categorie) {
            $em->remove($categorie);
        }

        $em->flush();
        $this->addFlash('notice', 'Catégories supprimées avec succès');
        return $this->redirectToRoute('app_liste_categories');
    }

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

Dans la vue, on ajoute le formulaire avec une case à cocher.

```twig
<td>{{ form_widget(form.categories[categorie.id]) }}</td>

{{ form_widget(form.supprimer) }}
```

3.5 LES FICHIERS

Les utilisateurs connectés peuvent uploader des fichiers qui pourront par la suite partager à d'autres personnes.

3.5.1 L'ajout de fichier

Nous avons créé un formulaire en liant l'entité Fichier.

```php
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('fichier', VichFileType::class, [
            'required' => true,
            'allow_delete' => false,
            'download_uri' => false,
            'download_label' => false,
            'constraints' => [
                new File([
                    'maxSize' => '200M',
                    'mimeTypes' => [
                        'application/pdf',
                        'image/jpeg',
                        'image/png',
                    ],
                    'mimeTypesMessage' => 'Please upload a valid PDF, JPG, or PNG file.',
                ])
            ],
        ])
        ->add('categorie', EntityType::class, [
            'attr' => ['class'=> 'form-select' ],
            'class' => Categorie::class,
            'choice_label' => 'libelle',
            'multiple' => true,
            'constraints' => [
                new NotBlank([
                    'message' => 'Veuillez sélectionner au moins une catégorie.',
                ]),
            ],
        ],)
        ->add('envoyer', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-whi
    ;
}
```

On a spécifié le type d'entité, les éléments qui sont requis ainsi que la contrainte de poids du fichier. Chaque fichier ajouté doit également disposer obligatoirement d'une ou plusieurs catégories.

Dans le contrôleur et la vue, on intègre le formulaire comme précédemment.

```twig
<h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">
    Ajouter un fichier</h1>
<div class="row justify-content-center">
    <div class="col-12 col-md-6 bg-white p-4 m-0 text-primary">
        {{form(form)}}
    </div>
</div>
```

On crée un répertoire « uploads » avec un dossier « fichiers » à la racine du projet, en lui donnant tous les droits.

- chmod -R 777 uploads

Ces répertoires vont stocker les fichiers envoyés par les utilisateurs.

Lorsqu'on clique sur le bouton d'envoi du fichier. On vérifie que le formulaire est valide, on récupère les données. S'il y a une donnée, on va récupérer le nom du fichier sans son extension, puis on utilise la variable de type SluggerInterface pour formater en chaine de caractères en retirant les espaces et les caractères spéciaux. Ensuite, on ajoute l'extension à ce nom. On prépare l'insertion en remplissant les champs de l'entité.

Voici une liste avec certaines fonctionnalités MIMES :

Méthode	Description
getSize()	Récupère la taille du fichier en octets
guessExtension()	Récupère le type du fichier en lisant son MIME
getClientOriginalName()	Récupère le nom du fichier

Pour voir plus :
https://developer.mozilla.org/fr/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

```php
#[Route('/profil-ajout-fichier', name: 'app_profil-ajout-fichier')]
public function index(Request $request, SluggerInterface $slugger, CategorieRepository $categorie
{
    $form = $this->createForm(AjoutFichierType::class);

    if($request->isMethod('POST')){
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()){
            $fichier = $form->get('fichier')->getData();

            if($fichier){
                $nomFichier = pathinfo($fichier->getClientOriginalName(), PATHINFO_FILENAME);
                $nomFichier = $slugger->slug($nomFichier);
                $nomFichier = $nomFichier.'-'.uniqid().'.'.$fichier->guessExtension();
                try{
                    $f = new Fichier();
                    $f->setNomOriginal($fichier->getClientOriginalName());
                    $f->setDateEnvoi(new \Datetime());
                    $f->setExtension($fichier->guessExtension());
                    $f->setProprietaire($this->getUser());
                    $f->setFile($fichier);

                    $categories = $form->get('categorie')->getData();
                    foreach ($categories as $categorie){
                        $c = $categoriesRepository->find($categorie);
                        $f -> addCategory($c);
                    }
                    $this->addFlash('notice', 'Fichier envoyé');
                }
                catch(FileException $e){
                    $this->addFlash('notice', 'Erreur d\'envoi');
                }
```

```
                $entityManagerInterface->persist($f);
                $entityManagerInterface->flush();
            }
            return $this->redirectToRoute('app_profil-ajout-fichier');
        }
    }

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

    ]);
}
```

3.5.2 La visualisation des fichiers uploadés

J'ai également fait la même chose que la partie 3.3.2.

La personne connectée utilisateur peut voir les fichiers qu'elle a déjà uploadé. On utilise la variable de connexion stockées par l'application lors de cette dernière. Via cette variable, on récupère l'ensemble des fichiers appartenant à la personne. (app.user.fichiers).

```
<table class="table caption-top">
    <caption>Liste des fichiers</caption>
    <thead>
        <tr>
            <th scope="col">Nom</th>
            <th scope="col">Date</th>
            <th scope="col">Extension</th>
            <th scope="col">Taille</th>
            <th scope="col"></th>

        </tr>
    </thead>
    <tbody>
        {% for f in app.user.fichiers | sort((a, b) => a.nomOriginal <=> b.nomOriginal) %}
            <tr class="{{ cycle(['table-primary', 'table-secondary'], loop.index0) }}">
                <td>{{f.nomOriginal}}</td>

                <td>{{f.dateEnvoi | date("d-m-Y à H:i:s") }}</td>
                <td>
                    {% if f.extension == 'pdf'%}
                        <i class="bi bi-file-earmark-pdf"></i>
                    {% else %}
                        <i class="bi bi-file-earmark-text"></i>
                    {% endif %}
                </td>
                {% set ko = f.taille / 1000 %}
                <td>{{ko}}
                    ko</td>
                <td><a href="{{path('telechargement-fichier', {'id':f.id})}}" target="_blank"><i class="bi bi-download text

            </tr>
        {% endfor %}
    </tbody>
</table>
```

Voici le visuel :

Liste des fichiers

Nom	Date	Extension	Taille
B1-Bloc3-DELEGATION_TP.pdf	30-10-2023 à 17:02:28		39.853 ko
BDD UML.pdf	30-10-2023 à 16:52:26		34.998 ko
Certificat_.pdf	31-10-2023 à 11:11:33		104.72 ko
Certificat_.pdf	31-10-2023 à 12:15:39		104.72 ko
Certificat_.pdf	31-10-2023 à 12:31:56		104.72 ko
Certificat_conformite.pdf	31-10-2023 à 12:17:16		104.754 ko
Certificat_respacteurs.pdf	31-10-2023 à 12:14:46		104.741 ko
Certificat_respacteurs.pdf	31-10-2023 à 12:15:03		104.741 ko
Cours_Transact SQLServer_Gaber.pdf	12-04-2024 à 13:00:59		436.378 ko
Mélanie_BoudryCV.pdf	05-03-2024 à 13:57:32		473.091 ko
Work life balance fr.pdf	30-10-2023 à 16:55:36		172.013 ko
lettre de motivation CGI.pdf	11-04-2024 à 10:47:02		72.432 ko

On a ajouté la possibilité de télécharger le document.

```
#[Route('/profil-telechargement-fichier/{id}', name: 'telechargement-fichier', requirements: ["id"=>"\d+"] )]
public function telechargementFichier(int $id, EntityManagerInterface $entityManagerInterface) {

    $repoFichier = $entityManagerInterface->getRepository(Fichier::class);
    $fichier = $repoFichier->find($id);
    if ($fichier == null){
        $this->redirectToRoute('app_profil-ajout-fichier');
    }
    else{
        return $this->file($this->getParameter('file_directory').'/'.$fichier->getNomServeur(), $fichier->getNomOriginal());
    }
}
```

On cherche le fichier à l'aide de la méthode find() du manager grâce à l'id du fichier. On retourne le fichier en le téléchargeant.

3.6 PROFIL

3.6.1 Ajouter les informations de l'utilisateur connecté

Nous avons créé un template ainsi qu'un contrôleur pour faire une page de profil.

```
#[Route('/profile', name: 'app_profil')]
public function profil(): Response
{

    return $this->render('user/profil.html.twig', [

    ]);
}
```

Ensuite, dans la vue, on utilise les informations stockées par l'application lors de la connexion.

```
<div class="pt-4 pb-4">
    <p>Email : {{app.user.email}}</p>
    <hr>
    <p>Nom : {{app.user.lastname}}</p>
    <hr>
    <p>Prénom : {{app.user.firstname}}</p>
    <hr>
    <p>Date d'inscription : {{app.user.dateRegister | date("d-m-Y à H:i:s")}}</p>
    <hr>
    <p>Rôle : {% if "ROLE_ADMIN" in app.user.roles
        <td>Administrateur</td>
    {% else %}
        {% if "ROLE_MOD" in app.user.roles %}
            <td>Modérateur</td>
        {% else %}
            <td>Utilisateur</td>
        {% endif %}
    {% endif %}</p>
</div>
```

Voici le visuel :

3.6.2 L'affichage de ses propres fichiers

Nous avons utilisé le même principe que pour l'affichage des fichiers déjà uploadés par la personne connectée. Nous affichons tout simplement plus d'informations et d'actions dans ce tableau. On a la possibilité de partager le fichier à des personnes, de voir les personnes à qui on a partagé ce fichier et révoquer ce partage.

```
<div class="pt-4 pb-4">
    <p>Vous avez {{ app.user.fichiers | length }} fichier(s).</p>
    {% if app.user.fichiers | length > 0%}
        <table class="table table-hover">
            <thead>
                <tr class="fw-bold text-primary">
                    <th scope="col">Nom du fichier</th>
                    <th scope="col">Catégories</th>
                    <th scope="col">Partagé à ?</th>
                    <th scope="col">Partagé à autre</th>
                </tr>
            </thead>
            <tbody>

                {% for fichier in app.user.fichiers %}
                    <tr class="{{ cycle(['table-primary','table-secondary'], loop.index0) }}">
                        <td>{{fichier.nomOriginal }}</td>
                        <td>{% if fichier.categories | length > 0 %}
                            <ul>
                                {% for categorie in fichier.categories %}
                                    <li>{{ categorie.libelle  }}</li>
                                {% endfor %}
                            </ul>
                            {% else %}
                                <p>Aucune</p>
                            {% endif %}
                        </td>
                        <td>{% if fichier.user | length > 0 %}
                            <ul>
                                {% for userpartage in fichier.user %}
                                    <li>{{ userpartage.lastname  }} {{ userpartage.firstname  }}<a href="{{path('app_del_partagefichier'
                                {% endfor %}
                            </ul>
                            {% else %}
                                <p>Aucune</p>
                            {% endif %}
                        </td>
```

```
                    <td><a href="{{path('app_partager_fichier_with', {'id':fichier.id})}}" ><i class="bi bi-share-fill text-black"><
                </tr>
            {% endfor %}
        </tbody>
    </table>
```

Vous avez 12 fichier(s).

Nom du fichier	Catégories	Partagé à ?	Partagé à autre
BDD UML.pdf	• Librairie • Gestions de projet	• Leroy Laurence	
Work life balance fr.pdf	• Marketing	• Thomas Dominique	
B1-Bloc3-DELEGATION_TP.pdf	• Librairie	• Mélanie BOUDRY	
Certificat_.pdf	Aucune	Aucune	
Certificat_respacteurs.pdf	Aucune	Aucune	
Certificat_respacteurs.pdf	Aucune	Aucune	

3.6.3 L'affichage des fichiers partagés avec lui

Nous avons utilisé le même principe que pour l'affichage des fichiers déjà uploadés par la personne connectée. Nous affichons tout simplement ici que les fichiers qui nous ont été partagé (app.user.fichiersPartages) dans un tableau. On a la possibilité ici comme l'autre de télécharger le fichier.

Vous avez 2 fichier(s) partagé(s).

Nom du fichier	Catégories	Propriétaire	Télécharger
B1-Bloc3-DELEGATION_TP.pdf	• Librairie	Mélanie BOUDRY	
Screenshot_20240422_101127_Samsung Health Monitor.jpg	Aucune	Leroy Laurence	

```
<div class="pt-4 pb-4">
    <p>Vous avez {{ app.user.fichiersPartages | length }} fichier(s) partagé(s).</p>

    {% if app.user.fichiersPartages | length > 0%}
        <table class="table table-hover">
            <thead>
                <tr class="fw-bold text-primary">
                    <th scope="col">Nom du fichier</th>
                    <th scope="col">Catégories</th>
                    <th scope="col">Propriétaire</th>
                    <th scope="col">Télécharger</th>
                </tr>
            </thead>
            <tbody>
                {% for fichierPartage in app.user.fichiersPartages %}
                    <tr class="{{ cycle(['table-primary','table-secondary'], loop.index0) }}">
                        <td>{{fichierPartage.nomOriginal }}</td>
                        <td>{% if fichierPartage.categories | length > 0 %}
```

```
                                <ul>
                                    {% for categorie in fichierPartage.categories %}
                                        <li>{{ categorie.libelle  }}</li>
                                    {% endfor %}
                                </ul>
                                {% else %}
                                    <p>Aucune</p>
                                {% endif %}
                            </td>
                            <td>{{ fichierPartage.proprietaire.lastname  }} {{ fichierPartage.proprietaire.firstname  }} </td>
                            <td><a href="{{path('telechargement-fichier', {'id':fichierPartage.id})}}" target="_blank"><i class="b
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
        {% endif %}
    </div>
```

3.6.4 Partager un fichier avec quelqu'un

Pour le partage de fichiers à une personne, j'ai ajouté un élément dans le tableau (3.6.2).

```
<td><a href="{{path('app_partager_fichier_with', {'id':fichier.id})}}" ><i class="bi bi-share-fill text-black"></i></td>
```

J'ai créé un formulaire qui va nous permettre de sélectionner la personne via l'email à qui on veut envoyer notre fichier. Cela correspond à une liste déroulante des noms des utilisateurs inscrits dans l'application.

```
class PartageWithType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            //->add('id', HiddenType::class )
            ->add('user', EntityType::class, [
                'attr' => ['class'=> 'form-select' ],
                'class' => User::class,
                'choice_label' => 'email',
                'multiple' => false,
                'label' => 'Partager à ?',
                'mapped' => false,
            ],)
            ->add('Partager', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary tex
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            //'data_class' => Fichier::class,
        ]);
    }
}
```

Dans le controleur Fichier, j'ai ajouté une fonction pour le partage.

On vérifie et cherche le fichier à partager via son id passé en paramètre. En cas de problème, on retourne une erreur. Lorsqu'on appuie sur le bouton de partage, on vérifie que le formulaire est valide et ensuite on recherche l'utilisateur récepteur du fichier avec un find(). Si le fichier n'est pas « null », alors on y ajoute dans sa variable de personnes partagés l'utilisateur choisi. On retourne un message en cas de réussite.

```php
#[Route('/partager_fichier/{id}', name: 'app_partager_fichier_with')]
public function partagerFichier(Request $request, FichierRepository $fichierRepository, UserRep
{
    $fichier=null;
    if ($request->get('id') != null)
    {
        $id = $request->get('id');
        $fichier = $fichierRepository->find($id);
    }

    $form = $this->createForm(PartageWithType::class);
    //$idFichier = $request-> get('idfichier');

    if (!$fichier) {
        throw $this->createNotFoundException('Fichier non trouvé');
    }

    if($request->isMethod('POST')){
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()){
            $idUser = $form->get('user')->getData();

            $user = $userRepository->find($idUser);

            if (!$user) {
                throw $this->createNotFoundException('Utilisateur non trouvé');
            }

            if ($fichier != null) {
                $fichier -> addUser($user);
                $em->persist($fichier);
                $em->flush();
            }

            $this->addFlash('notice','Partage ajouté');
            return $this->redirectToRoute('app_profil');
        }
    }

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

```
{% block body %}
    <div class="container-fluid">
        <h1 class="text-center text-primary mt-4 pt
            Partager un fichier avec
        </h1>
        <div class="row justify-content-center">
            <div class="col-12 col-md-6 bg-white p-
                <p>Nom :
                    {{fichier.nomOriginal}}</p>
                <p>Propriétaire :
                    {{fichier.proprietaire.lastname}}-{{fichier.proprietaire.firstname}}</p>

                {{form(form)}}

                </div>
            </div>

        </div>
    {% endblock %}
```

3.6.5 Révoquer un partage

La personne propriétaire du fichier peut aussi annuler un partage de fichier avec quelqu'un.

Pour cela, j'ai ajouté à coté de chaque bénéficiaire du fichier une croix (bouton) pour annuler le partage.

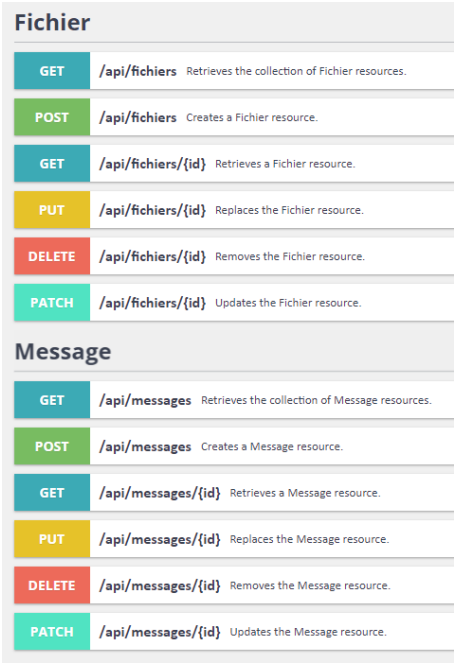
Lorsque l'on clique sur la croix à côté de l'utilisateur qu'on veut supprimer, une fonction vérifie que l'utilisateur et le fichier retournés en GET existent bien avec des find(). Ensuite on supprime la personne de la liste en utilisant la méthode removeUser() en passant en paramètre l'utilisateur à supprimer.

```
#[Route('/delete-partage-fichier/{iduser}/{idfichier}', name: 'app_del_partagefichier')]
public function delPartage(Request $request, UserRepository $userRepository, FichierRepository $fichierRepository
{
    $idUser = $request-> get('iduser');
    $idFichier = $request-> get('idfichier');

    $user = $userRepository->find($idUser);
    $fichier = $fichierRepository->find($idFichier);

    $fichier -> removeUser($user);

    $em->persist($fichier);
    $em->flush();
    $this->addFlash('notice','Partage supprimée');
    return $this->redirectToRoute('app_profil');
}
```

4 L'APPLICATION MOBILE

Pour afficher et manipuler les mêmes données que l'application web dans notre application mobile, nous avons utilisé une API grâce à la mise en place de API Plateform qui nous permet de récupérer ces informations. Pour ma part, je n'ai interagi que avec les entités Message et Fichier de l'API. J'ai manipulé l'affichage des éléments de ces derniers, mis en place des filtres, des tris pour faire correspondre avec les informations que je souhaitais afficher sur l'application mobile.

Tous les appels API retournent un JSON avec les données.



La mise en place d'une API se fait très facilement à partir d'un projet avec base de données :

- composer require api
- composer require symfony/apache-pack
- On rajoute dans tous les entités que l'on souhaite voir dans l'API
    - use ApiPlatform\Metadata\ApiResource;
    - #[ApiResource()]

Ces éléments ont pour l'entité Message et Fichier un peu évolué.

J'ai donc dans un premier temps géré les retours API de l'entité Fichier et Message.

Pour cela, j'ai premièrement mis en place les opérations possibles sur chaque entité :

- GETCollection
- POST
- GET
- PUT
- PATCH
- DELETE

Pour certains GETCollection et GET, j'ai spécifié des groupes.

Les imports :

```
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;
use ApiPlatform\Metadata\Patch;
use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\Post;
use ApiPlatform\Metadata\Put;
```

L'entité Fichier :

```php
#[ApiResource(paginationItemsPerPage: 100,operations: [
    new GetCollection(normalizationContext: ['groups' => 'fichier:list']),
    new Post(),
    new Get(normalizationContext: ['groups' => 'fichier:item']),
    new Put(),
    new Patch(),
    new Delete(),
],)]
```

```php
class Fichier
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['fichier:list', 'fichier:item'])]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    #[Groups(['fichier:list', 'fichier:item'])]
    private ?string $nomOriginal = null;

    #[ORM\Column(length: 255)]
    #[Groups(['fichier:item'])]
    private ?string $nomServeur = null;

    #[ORM\Column(type: Types::DATETIME_MUTABLE)]
    #[Groups(['fichier:item'])]
    private ?\DateTimeInterface $dateEnvoi = null;

    #[ORM\Column(length: 3)]
    #[Groups(['fichier:list', 'fichier:item'])]
    private ?string $extension = null;

    #[ORM\Column]
    #[Groups(['fichier:item'])]
    private ?float $taille = null;

    #[ORM\ManyToOne(inversedBy: 'fichiers')]
    #[ORM\JoinColumn(nullable: false)]
    #[Groups(['fichier:list', 'fichier:item'])]
    private ?User $proprietaire = null;

    #[ORM\OneToMany(mappedBy: 'fichier', targetEntity: Telecharger::class, orphar
    #[Groups(['fichier:item'])]
    private Collection $telechargers;

    #[ORM\ManyToMany(targetEntity: Categorie::class, mappedBy: 'fichier')]
    #[Groups(['fichier:item'])]
    private Collection $categories;

    #[ORM\ManyToMany(targetEntity: User::class, inversedBy: 'fichiersPartages')]
    #[Groups(['fichier:item'])]
    private Collection $user;
```

J'ai spécifié pour chaque attribut de l'entité Fichier s'il devait être inclus lors de la demande de la liste ou seulement en tant qu'élément individuel. Cela se reflète dans le code où j'ai ajouté des annotations spécifiant les groupes d'inclusion pour chaque attribut, distinguant entre ceux destinés à la liste complète et ceux destinés à un élément individuel.

L'entité Message :

En plus des groupes comme avant, j'ai ajouté de la sécurité. La modification n'est possible que quand l'objet appartient à la personne user connectée. La suppression n'est possible que par un administrateur ou la personne propriétaire du message. L'ajout d'un message ne se fait que par une personne connectée.

```php
#[ApiResource(paginationItemsPerPage: 100,operations: [
    new GetCollection(normalizationContext: ['groups' => 'messag
    new Post(security: "is_granted('ROLE_USER')"),
    new Get(normalizationContext: ['groups' => 'message:item']),
    new Put(),
    new Patch(security:"object.user == user"),
    new Delete(security: "is_granted('ROLE_ADMIN') or object.use
    ],)]
```

```php
class Message
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    #[Groups(['message:list', 'message:item'])]
    private ?int $id = null;

    #[ORM\Column(length: 50)]
    #[Groups(['message:list', 'message:item'])]
    private ?string $title = null;

    #[ORM\Column(type: Types::DATETIME_MUTABLE)]
    #[Groups(['message:list', 'message:item'])]
    private ?\DateTimeInterface $datePost = null;

    #[ORM\Column(type: Types::TEXT)]
    #[Groups(['message:list', 'message:item'])]
    private ?string $content = null;

    #[ORM\ManyToOne(inversedBy: 'messages')]
    #[ORM\JoinColumn(nullable: false)]
    #[Groups(['message:list', 'message:item'])]
    public ?User $user = null;

    #[ORM\ManyToOne(targetEntity: self::class, inversedBy: 'messages')]
    #[Groups(['message:list', 'message:item'])]
    private ?self $parent = null;

    #[ORM\OneToMany(mappedBy: 'parent', targetEntity: self::class, casca
    private Collection $messages;
```

J'ai fait de même pour les annotations spécifiant les groupes d'inclusion pour les attributs de l'entité Message.

4.1 AFFICHAGE DES FICHIERS DE LA PERSONNE ET DE CEUX PARTAGES

4.1.1 Récupérer les données avec l'API

```
#[ApiFilter(SearchFilter::class, properties: ['proprietaire' => 'exact', 'user'=>'exact'])]
```

J'ai également mis en place un filtre de recherche pour pouvoir spécifier que je cherche tous les fichiers de tel personne ou alors que je veux voir les fichiers qui disposent de cette personne comme personne partagée. Via cela, je peux chercher un ou des fichiers en fonction du propriétaire exact ou des users (personnes partagées) exact.

Pour interagir et obtenir les données que je souhaite afficher sur l'application, il suffit rentrer la bonne URL avec les bons paramètres :

- Pour obtenir les fichiers étant partagés par la personne connectée, c'est-à-dire que c'est le propriétaire du fichier, il faut utiliser l'URL :

```
https://s4-8057.nuage-peda.fr/share/api/fichiers?proprietaire=2
```

Le « 2 » après le « proprietaire= » correspond à l'id de la personne cherchée.

- Pour obtenir les fichiers partagés avec la personne connectée, c'est-à-dire que la personne connectée fait partir des users du fichier, il faut utiliser l'URL :

```
https://s4-8057.nuage-peda.fr/share/api/fichiers?user=1
```

Le « 1 » après le « user= » correspond à l'id de la personne cherchée dans la liste des users des fichiers.

4.1.1.1 Les fichiers étant partagés par la personne connectée

Pour obtenir tous les fichiers où l'utilisateur est propriétaire, il me fallait donc utiliser la première URL (**https://s4-8057.nuage-peda.fr/share/api/fichiers?proprietaire=<idPersonne>**).

Si par exemple, j'essaie d'interagir avec l'API et que je cherche les fichiers de la personne avec l'id 1 :

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/fichiers?proprietaire=1' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/fichiers?proprietaire=1
```

Server response

L’API me retourne une réponse exprimée en JSON.

Code | Details

200

Response body


```
{
  "@context": "/share/api/contexts/Fichier",
  "@id": "/share/api/fichiers",
  "@type": "hydra:Collection",
  "hydra:totalItems": 12,
  "hydra:member": [
    {
      "@id": "/share/api/fichiers/1",
      "@type": "Fichier",
      "id": 1,
      "nomOriginal": "BDD UML.pdf",
      "extension": "pdf",
      "proprietaire": "/share/api/users/1"
    },
    {
      "@id": "/share/api/fichiers/2",
      "@type": "Fichier",
      "id": 2,
      "nomOriginal": "Work life balance  fr.pdf",
      "extension": "pdf",
      "proprietaire": "/share/api/users/1"
    },
    {
      "@id": "/share/api/fichiers/3",
      "@type": "Fichier",
      "id": 3,
      "nomOriginal": "B1-Bloc3-DELEGATION_TP.pdf",
```


Il y a le nombre d’éléments (hydra:totalItems) ainsi que l’ensemble de ces 12 éléments (hydra:member).

Sur l’application mobile :

Pour utiliser cette API via cette URL, nous devons d'abord initialiser un état « fichiers » et « setFichiers » en utilisant la fonction « useState ». Ensuite, nous construisons une fonction qui envoie une requête GET à l'API (en utilisant l'URL précédemment testée) et prend en paramètre l'ID du propriétaire (user.id), qui représente la personne connectée à l'application. Cette fonction récupère les fichiers et met à jour la variable "fichiers" via "setFichiers". Selon la réponse obtenue (qu'elle soit vide ou non), nous mettons à jour l'état des fichiers avec les données de la réponse.

```
const getFichiersById = async () => {
  const data = await fetch(nuage + "api/fichiers?proprietaire=" + user.id);

  const dataJSON = await data.json();
  if (dataJSON["hydra:totalItems"] > 0) {
    setFichiers(dataJSON["hydra:member"]);
  }
};
```

Ensuite, dans un « useEffect », nous appelons la fonction. Nous avons également mis en place une variable de chargement « loading » pour afficher les éléments, le visuel que si tout est bien chargé pour ne pas rencontrer de problèmes.

```
  const itemDimension = (Dimensions.get("window").width - 60) / 3 - 20;
  const [selectedValue, setSelectedValue] = useState("MesFichiers");
  const [fichiers, setFichiers] = useState([]);
  const [fichiersShared, setFichiersShared] = useState([]);
  const { user } = useUser();
  const [loading, setLoading] = useState(true);
  const [fileToUpload, setFileToUpload] = useState(null);

  useEffect(() => {
    getFichiersById();
    getFichiersShared();
    setLoading(false);
  }, []);
```

4.1.1.2 Les fichiers partagés avec la personne connectée

Pour obtenir tous les fichiers partagés avec l'utilisateur, il me fallait donc utiliser la première URL (**https://s4-8057.nuage-peda.fr/share/api/fichiers?user=<idPersonne>**).

Si par exemple, j'essaie d'interagir avec l'API et que je cherche les fichiers partagés de la personne avec l'id 1 :

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/fichiers?user=1' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/fichiers?user=1
```

Server response

L'API me retourne une réponse exprimée en JSON.

Code	Details
200	Response body


```
{
  "@context": "/share/api/contexts/Fichier",
  "@id": "/share/api/fichiers",
  "@type": "hydra:Collection",
  "hydra:totalItems": 1,
  "hydra:member": [
    {
      "@id": "/share/api/fichiers/3",
      "@type": "Fichier",
      "id": 3,
      "nomOriginal": "B1-Bloc3-DELEGATION_TP.pdf",
      "extension": "pdf",
      "proprietaire": "/share/api/users/1"
    }
  ],
  "hydra:view": {
    "@id": "/share/api/fichiers?user=1",
```

Il y a le nombre d'éléments (hydra:totalItems) ainsi que l'ensemble de ces éléments (hydra:member).

Sur l'application mobile :

Pour utiliser cette API via cette URL, nous devons d'abord initialiser un état « fichiersShared » et « setFichiersShared » en utilisant la fonction « useState ». Ensuite, nous construisons une fonction qui envoie une requête GET à l'API (en utilisant l'URL précédemment testée) et prend en paramètre l'ID (user.id), qui représente la personne connectée à l'application. Cette fonction récupère les fichiers et met à jour la variable " fichiersShared " via " setFichiersShared ". Selon la réponse obtenue (qu'elle soit vide ou non), nous mettons à jour l'état des fichiers avec les données de la réponse.

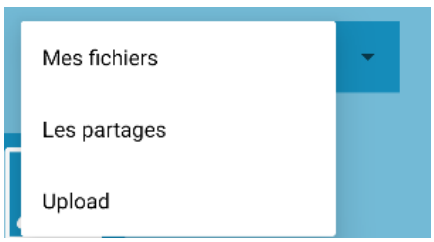
```
const getFichiersShared = async () => {
  const data = await fetch(nuage + "api/fichiers?user=" + user.id);

  const dataJSON = await data.json();
  if (dataJSON["hydra:totalItems"] > 0) {
    setFichiersShared(dataJSON["hydra:member"]);
  }
};
```

Ensuite, dans un « useEffect », nous appelons la fonction. Nous avons également mis en place une variable de chargement « loading » pour afficher les éléments, le visuel que si tout est bien chargé pour ne pas rencontrer de problèmes.

```
const itemDimension = (Dimensions.get("window").width - 60) / 3 - 20;
const [selectedValue, setSelectedValue] = useState("MesFichiers");
const [fichiers, setFichiers] = useState([]);
const [fichiersShared, setFichiersShared] = useState([]);
const { user } = useUser();
const [loading, setLoading] = useState(true);
const [fileToUpload, setFileToUpload] = useState(null);

useEffect(() => {
  getFichiersById();
  getFichiersShared();
  setLoading(false);
}, []);
```

4.1.2 Visuel

Après avoir obtenu les données de l'API, l'affichage dépend de l'élément sélectionné dans la liste déroulante. Par défaut, l'élément sélectionné est « MesFichiers ».




```
<Picker
  selectedValue={selectedValue}
  onValueChange={(itemValue, itemIndex) => {
    setSelectedValue(itemValue);
  }}
  style={styles.picker}
  mode={"dropdown"}
>
  <Picker.Item label="Mes fichiers" value="MesFichiers" />
  <Picker.Item label="Les partages" value="LesPartages" />
  <Picker.Item label="Upload" value="Upload" />
</Picker>
```

Si la variable est définie sur « MesFichiers », une grille à 3 colonnes est créée en utilisant les données API « fichiers ». Chaque case de la grille affiche le nom et une icône du fichier correspondant, et chaque case est cliquable pour accéder à un élément spécifique. Si le nom est trop long, il est coupé.

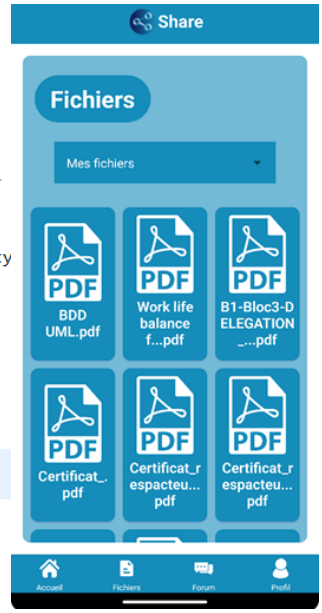
C'est un flatGrid qui crée une grid en fonction d'un tableau de données (data=fichiers) émis, et qui boucle sur ces dernières pour afficher l'image et le texte correspondant.

Si, la variable est égale à « LesPartages » alors la création de la grille se base sur la variable « fichiersShared ». Le fonctionnement se fait de la même manière.

Voici pour le coupage du titre si trop long :

```
<Text style={styles.itemCodeFichier}>
  {item.nomOriginal.length > 20
    ? item.nomOriginal.slice(0, 20) +
      "..." +
      item.extension
    : item.nomOriginal}
</Text>
```

```
{selectedValue === "MesFichiers" &&
  (fichiers.length > 0 ? (
    <FlatGrid
      itemDimension={itemDimension}
      data={fichiers}
      style={styles.gridView}
      keyExtractor={(item) => item.id.toString()}
      renderItem={({ item }) => (
        <TouchableOpacity
          style={[styles.itemContainerFichier, sty
          onPress={() => {
            navigation.navigate("Fichier", {
              type: "Fichier",
              idFichier: item.id,
            });
          }}
        >
          {item.extension === "pdf" ? (···
          ) : (···
          )}
          <Text style={styles.itemCodeFichier}>
            {item.nomOriginal.length > 20
              ? item.nomOriginal.slice(0, 20) +
                "..." +
                item.extension
              : item.nomOriginal}
          </Text>
        </TouchableOpacity>
      )}
    />
  ) : (
    <View style={[styles.loadingContainer]}>
      <Text>Pas de fichier</Text>
    </View>
  ))}
```




```
{selectedValue === "LesPartages" &&
  (fichiersShared.length > 0 ? (
    <FlatGrid
      itemDimension={itemDimension}
      data={fichiersShared}
      style={styles.gridView}
      keyExtractor={(item) => item.id.toString()}
      renderItem={({ item }) => (
        <TouchableOpacity
          style={[styles.itemContainerFichier, styles.shadowProp]}
          onPress={() => {
            navigation.navigate("Fichier", {
              type: "Partage",
              idFichier: item.id,
            });
          }}
        >
          {item.extension === "pdf" ? (···
          ) : (···
          )}
          <Text style={styles.itemCodeFichier}>
            {item.nomOriginal.length > 20
              ? item.nomOriginal.slice(0, 20) +
                "..." +
                item.extension
              : item.nomOriginal}
          </Text>
        </TouchableOpacity>
      )}
    />
  ) : (
    <View style={[styles.loadingContainer]}>
      <Text>Pas de fichier partagé avec vous</Text>
    </View>
  ))}
```

4.2 LES DETAILS DE CHAQUE FICHIER

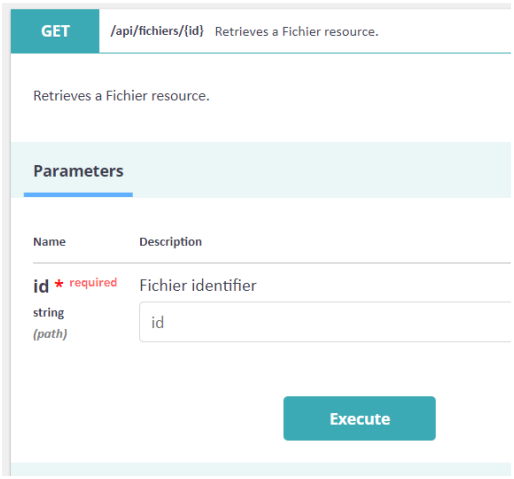
4.2.1 Récupérer les données

Pour interagir et obtenir les données que je souhaite afficher sur l'application en fonction du fichier, il suffit rentrer l'URL :

`https://s4-8057.nuage-peda.fr/share/api/fichiers/2`

Le 2 à la fin correspond à l'id du fichier recherché.

Cela se fait sur le GET avec id, la recherche d'un élément.



<u>Pour tester :</u>

On obtient toutes les informations du fichier avec l'id 2.

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/fichiers/2' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/fichiers/2
```

Server response

Code | Details

200

Response body


```
{
  "@context": "/share/api/contexts/Fichier",
  "@id": "/share/api/fichiers/2",
  "@type": "Fichier",
  "id": 2,
  "nomOriginal": "Work life balance  fr.pdf",
  "nomServeur": "Work-life-balance-fr-653fd1f8bd4be.pdf",
  "dateEnvoi": "2023-10-30T16:55:36+01:00",
  "extension": "pdf",
  "taille": 172013,
  "proprietaire": {
    "@id": "/share/api/users/1",
    "@type": "User",
    "lastname": "Mélanie",
    "firstname": "BOUDRY"
  },
  "telechargers": [],
  "categories": [
    {
      "@id": "/share/api/categories/4",
      "@type": "Categorie",
      "libelle": "Marketing"
    }
  ],
  "user": [
    {
      "@id": "/share/api/users/9",
```


Les données renvoyées sont celles spécifiées avec la balise "fichier:item" devant chaque attribut.

Sur l'application mobile :

Pour utiliser cette API via cette URL, nous devons d'abord initialiser un état « fichier » et « setFichier » en utilisant la fonction « useState ». Ensuite, nous construisons une fonction qui envoie une requête GET à l'API (en utilisant l'URL précédemment testée) et prend en paramètre l'ID du fichier cliqué. Cette fonction récupère les fichiers et met à jour la variable "fichier" via "setFichier". Selon la réponse obtenue (qu'elle soit vide ou non), nous mettons à jour l'état du fichier avec les données de la réponse.

```
const getFichierById = async (idFichier) => {
  const data = await fetch(nuage + "api/fichiers/" + idFichier);
  const dataJSON = await data.json();
  setFichier(dataJSON);
  setLoading(false);
};
```

Ensuite, dans un « useEffect », nous appelons la fonction. Nous avons également mis en place une variable de chargement « loading » pour afficher les éléments, le visuel que si tout est bien chargé pour ne pas rencontrer de problèmes.

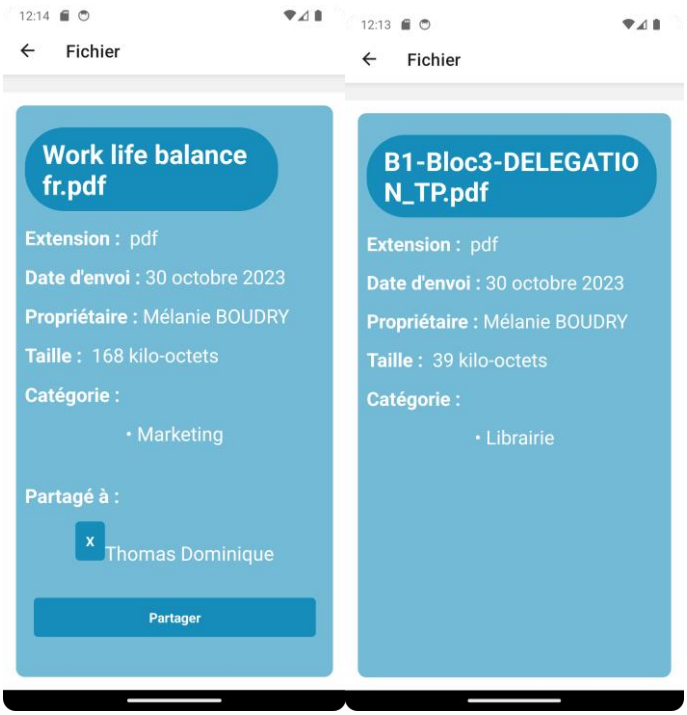
```
export default ({ route, navigation }) => {
  const { idFichier } = route.params;
  const { type } = route.params;
  const [fichier, setFichier] = useState({});
  const [loading, setLoading] = useState(true);
  const [downloaded, setDownloaded] = useState(null);

  useEffect(() => {
    getFichierByIdidFichier);
  });
```

4.2.2 Visuel

Tant que la variable n'est pas totalement chargée, la page est en mode chargement.

```
<View style={styles.body}>
  {loading ? (
    <View style={styles.blueContainer}>
      <Text>Chargement en cours...</Text>
    </View>
  ) : (
```

Quand tout est bien chargé, on affiche les éléments du fichier (nom, date d'envoi, propriétaire, taille, si catégorie la ou les catégorie(s). Si c'est un fichier dont nous sommes le propriétaire alors il sera également affiché les nom-prénom des personnes à qui on a partagé ce dernier.



Le bouton « partager » et la révocation n'ont pas été fait par moi.

```
<ScrollView style={styles.blueContainer}>
  <View style={styles.ecart}>
    <Text style={styles.blueTitre}>{fichier.nomOriginal}</Text>
    <Text style={styles.ProfilTexte}>
      <Text style={styles.gras}>Extension : </Text> {fichier.extension}
    </Text>
    <Text style={styles.ProfilTexte}>
      <Text style={styles.gras}>Date d'envoi : </Text>
      {fichier.dateEnvoi && formatDate(fichier.dateEnvoi)}
    </Text>
    <Text style={styles.ProfilTexte}>
      <Text style={styles.gras}>Propriétaire : </Text>
      {fichier.proprietaire.lastname +
        " " +
        fichier.proprietaire.firstname}
    </Text>
    <Text style={styles.ProfilTexte}>
      <Text style={styles.gras}>Taille : </Text>{" "}
      {Math.round(fichier.taille / 1024)} kilo-octets
    </Text>

    <Text style={styles.ProfilTexte}>
      <Text style={styles.gras}>Catégorie : </Text>
    </Text>      You, last month • liste des fichiers et partagé
    {fichier.categories.length > 0 ? (
      <View style={styles.categorieContainer}>
        <Text style={styles.ProfilTexte}>
          {fichier.categories.map((categorie, index) => (
            <React.Fragment key={index}>
              <Text>{"\u2022"} </Text>
              <Text>
                {categorie.libelle}
                {"\n"}
              </Text>
            </React.Fragment>
          ))}
        </Text>
      </View>
    ) : (
      <View style={styles.categorieContainer}>
        <Text style={styles.ProfilTexte}>Aucune catégorie</Text>
      </View>
    )}

    {type != "Partage" ? (
      <>
        <Text style={styles.ProfilTexte}>
          <Text style={styles.gras}>Partagé à : </Text>
        </Text>
        {fichier["user"].length > 0 ? (
          <View style={[styles.categorieContainer]}>
            <Text style={styles.Margin}>
              {fichier["user"].map((user, index) => (
                <React.Fragment key={index}>
                  <TouchableOpacity
                    onPress={() => {
                      {
                        let id = user["@id"].split("/").pop();
                        revoquer(id);
                      }
                    }}
                    style={styles.petitBouton}
                  >
                    <Text style={styles.textePetitBouton}>X</Text>
                  </TouchableOpacity>
                  <Text style={styles.PersonneTexte}>
                    {user["lastname"]} {user["firstname"]}
                    {"\n"}
                  </Text>
                </React.Fragment>
              ))}
            </Text>
          </View>
        ) : (
          <View style={styles.categorieContainer}>
            <Text style={styles.ProfilTexte}>Aucun partage</Text>
          </View>
        )}
        <TouchableOpacity
```

4.3 LE FORUM : AFFICHAGE DES SUJETS DE DISCUSSION POSSIBLE

4.3.1 Récupérer les données avec l'API

```
#[ApiFilter(ExistsFilter::class, properties: ['parent'])]
```

J'ai mis en place un filtre qui filtre en fonction de l'existence de l'élément parent du message, c'est-à-dire si c'est le message n'est pas enfant de quelqu'un d'autre. C'est donc un sujet.

```
#[ApiFilter(SearchFilter::class, properties: ['user' => 'exact', 'parent'=> 'exact'])]
#[ApiFilter(OrderFilter::class, properties: ['id' => 'ASC', 'title' => 'ASC'])]
```

De plus, on peut rechercher les éléments en fonction de leur attribut user et parent et dans l'ordre croissant du titre ou de l'id.

Pour interagir et obtenir les données que je souhaite afficher sur l'application, il suffit rentrer l'URL :

- Pour obtenir les sujets, il faut utiliser l'URL :

```
https://s4-8057.nuage-peda.fr/share/api/messages?exists%5Bparent%5D=false
```

Si par exemple, j'essaie d'interagir avec l'API et que je cherche les messages sans parent (false) :

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/messages?exists%5Bparent%5D=false' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/messages?exists%5Bparent%5D=false
```

Server response

Code Details

200

Response body


```
{
  "@context": "/share/api/contexts/Message",
  "@id": "/share/api/messages",
  "@type": "hydra:Collection",
  "hydra:totalItems": 3,
  "hydra:member": [
    {
      "@id": "/share/api/messages/1",
      "@type": "Message",
      "id": 1,
      "title": "Conseils pour choisir un portable",
      "datePost": "2024-01-24T07:35:48+01:00",
      "content": "Mon vieux portable montre des signes de faibl
toutes les options. Besoin de conseils !",
      "user": {
        "@id": "/share/api/users/15",
        "@type": "User",
        "lastname": "Guillot",
        "firstname": "Victoire"
      }
    },
    {
      "@id": "/share/api/messages/2",
      "@type": "Message",
      "id": 2,
      "title": "Langages pour débutants ?",
      "datePost": "2024-02-27T14:12:39+01:00",
```


L'API me retourne une réponse exprimée en JSON.

Il y a le nombre d'éléments (hydra:totalItems) ainsi que l'ensemble de ces éléments (hydra:member).

Sur l'application mobile :

Pour utiliser cette API via cette URL, nous devons d'abord initialiser un état « messages » et « setMessages » en utilisant la fonction « useState ». Ensuite, nous construisons une fonction qui envoie une requête GET à l'API (en utilisant l'URL précédemment testée) sans paramètre. Cette fonction récupère les messages sans parent donc les sujets de conversation et met à jour la variable "messages" via "setMessages". Selon la réponse obtenue (qu'elle soit vide ou non), nous mettons à jour l'état des messages avec les données de la réponse.

```
export default ({ navigation }) => {
  const [messages, setMessages] = useState([]);
  const { user } = useUser();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    getMessages();
    setLoading(false);
  });

  const getMessages = async () => {
    const data = await fetch(nuage + "api/messages?
    exists%5Bparent%5D=false");
    const dataJSON = await data.json();
    if (dataJSON["hydra:totalItems"] > 0) {
      setMessages(dataJSON["hydra:member"]);
    }
  };
```

Ensuite, dans un « useEffect », nous appelons la fonction. Nous avons également mis en place une variable de chargement « loading » pour afficher les éléments, le visuel que si tout est bien chargé pour ne pas rencontrer de problèmes.

4.3.2 Visuel

```
{loading ? (
  <View style={styles.blueContainer}>
    <Text>Chargement en cours...</Text>
  </View>
) : (
```

Tant que la variable n'est pas totalement chargée, la page est en mode chargement.

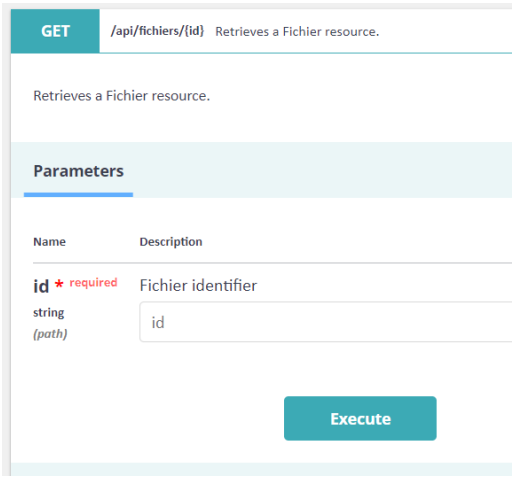
Je crée une liste qui est créé en bouclant sur les informations « messages » au format JSON. Chaque élément de la liste est cliquable pour être ensuite redirigé vers le détail du sujet.

C'est un flatList qui crée une liste en fonction d'un tableau de données (data=messages) émis, et qui boucle sur ces dernières pour le texte et l'image souhaitée.

```
<FlatList
  data={messages}
  style={styles.FlatList}
  keyExtractor={(item) => item.id.toString()}
  ItemSeparatorComponent={() => <View style={{ height: 10 }} />}
  renderItem={({ item }) => (
    <TouchableOpacity
      style={[
        styles.itemContainerFichier,
        styles.shadowProp,
        styles.flex_row,
      ]}
      onPress={() => {
        navigation.navigate("Sujet", {
          idSujet: item.id,
        });
      }}
    >
      <Image
        source={{
          uri: "https://cdn-icons-png.flaticon.com/512/6456/6456117.png",
        }}
        style={{
          tintColor: "white",
          width: 25,
          height: 25,
        }}
      />
      <View style={styles.contenuSujet}>
        <Text style={styles.sujetTitle}>{item.title}</Text>
        <Text style={styles.SousTitre}>
          {new Date(item.datePost).toLocaleTimeString("fr-FR", {
            day: "numeric",
            month: "long",
            year: "numeric",
          })}
        </Text>
      </View>
    </TouchableOpacity>
  )}
/>
```

4.4 LES DETAILS D'UN SUJET AVEC SES MESSAGES

4.4.1 Récupérer les données



Pour interagir et obtenir les données que je souhaite afficher sur l'application en fonction du fichier, il suffit rentrer l'URL :

- Pour obtenir les informations du sujet :

```
https://s4-8057.nuage-peda.fr/share/api/messages/2
```

Le 2 à la fin correspond à l'id du sujet recherché.

- Pour obtenir les messages du sujet

```
https://s4-8057.nuage-peda.fr/share/api/messages?parent=2
```

Le 2 à la fin correspond à l'id du sujet recherché.

<u>Pour tester :</u>

- Le sujet

On obtient toutes les informations du fichier avec l'id 2.

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/messages/2' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/messages/2
```

Server response

Les données renvoyées sont celles spécifiées avec la balise "fichier:item" devant chaque attribut.

Code Details

200

Response body


```
{
  "@context": "/share/api/contexts/Message",
  "@id": "/share/api/messages/2",
  "@type": "Message",
  "id": 2,
  "title": "Langages pour débutants ?",
  "datePost": "2024-02-27T14:12:39+01:00",
  "content": "Je suis nouveau en programmation. Quels langage
r ?",
  "user": {
    "@id": "/share/api/users/12",
    "@type": "User",
    "lastname": "Charpentier",
    "firstname": "Stéphane"
  }
}
```

- Les messages du sujet

Curl

```
curl -X 'GET' \
  'https://s4-8057.nuage-peda.fr/share/api/messages?parent=2' \
  -H 'accept: application/ld+json'
```

Request URL

```
https://s4-8057.nuage-peda.fr/share/api/messages?parent=2
```

Server response

Code Details

200

Response body


```
{
  "@context": "/share/api/contexts/Message",
  "@id": "/share/api/messages",
  "@type": "hydra:Collection",
  "hydra:totalItems": 1,
  "hydra:member": [
    {
      "@id": "/share/api/messages/13",
      "@type": "Message",
      "id": 13,
      "title": "Python ou JavaScript",
      "datePost": "2024-03-13T21:18:56+01:00",
      "content": "Je recommande Python ou JavaScript pour débuter. Ils sont
et faciles à apprendre.",
      "user": {
        "@id": "/share/api/users/1",
        "@type": "User",
        "lastname": "Mélanie",
        "firstname": "BOUDRY"
      },
      "parent": {
        "@id": "/share/api/messages/2",
        "@type": "Message",
        "id": 2,
        "title": "Langages pour débutants ?",
        "datePost": "2024-02-27T14:12:39+01:00",
        "content": "Je suis nouveau en programmation. Quels langages sont
```


Sur l'application mobile :

Pour utiliser cette API via cette URL, nous devons d'abord initialiser un état « sujet » et « setSujet » et « messages » et « setMessages » avec des loadings différents « loadingM » et « loadingS » en utilisant la fonction « useState ». Ensuite, nous construisons une fonction qui envoie une requête GET à l'API (en utilisant l'URL précédemment testée) et prend en paramètre l'ID du sujet cliqué. Cette fonction récupère les informations du sujet et met à jour la variable "sujet" via "setSujet".

```
const getSujet = async (idSujet) => {
  const data = await fetch(nuage + "api/messages/" + idSujet);
  const dataJSON = await data.json();
  setSujet(dataJSON);
  setLoadingS(false);
};
```

J'ai créé une autre fonction qui renvoie maintenant l'ensemble des messages du sujet.

```
const getMessagesBySujet = async (idSujet) => {
  const data = await fetch(nuage + "api/messages?parent=" + idSujet);
  const dataJSON = await data.json();
  // if (dataJSON["hydra:totalItems"] > 0) {
  setMessages(dataJSON["hydra:member"]);
  setLoadingM(false);
  // }
};
```

Ensuite, dans un « useEffect », nous appelons les fonctions. Nous avons également mis en place des variables de chargement pour afficher les éléments, le visuel que si tout est bien chargé pour ne pas rencontrer de problèmes.

```
export default ({ route, navigation }) => {
  const { idSujet } = route.params;
  const [sujet, setSujet] = useState({});
  const [messages, setMessages] = useState([]);
  const [loadingM, setLoadingM] = useState(true);
  const [loadingS, setLoadingS] = useState(true);
  const { user, token } = useContext(UserContext);

  useEffect(() => {
    getSujet(idSujet);
    getMessagesBySujet(idSujet);
  }, []);
```

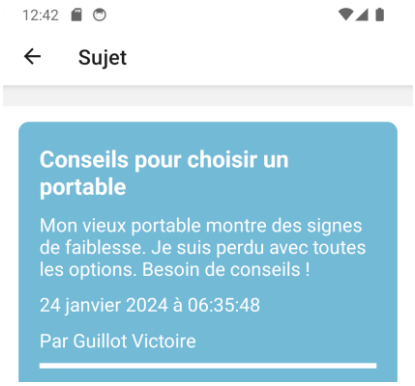
4.4.2 Visuel

Tant que la variable n'est pas totalement chargée, la page est en mode chargement.

```
{loading ? (
  <View style={styles.blueContainer}>
    <Text>Chargement en cours...</Text>
  </View>
) : (
```

Quand tout est bien chargé, on affiche le détail du sujet (titre, date de post, la personne à l'origine de ce sujet, description/contenu).

```
<ScrollView style={[styles.blueContainer]}>
  <View style={styles.sujetContainer}>
    <Text style={[styles.sujetTitle, { marginBottom: 10 }]}>
      {sujet.title}
    </Text>
    <Text style={styles.sujetElement}>{sujet.content}</Text>
    <Text style={styles.sujetElement}>
      {new Date(sujet.datePost).toLocaleTimeString("fr-FR", {
        day: "numeric",
        month: "long",
        year: "numeric",
      })}
    </Text>
    <Text style={styles.sujetElement}>
      {sujet.user["@id"] == IRInuage + "api/users/" + user.id ? (
        <>
          <View>
            <Text style={styles.messageElement}>Par vous</Text>

            <BoutonsModifierSupprimer message={sujet} />
          </View>
        </>
      ) : (
        <>
          Par {sujet.user.lastname} {sujet.user.firstname}
        </>
      )}
    </Text>
  </View>
```

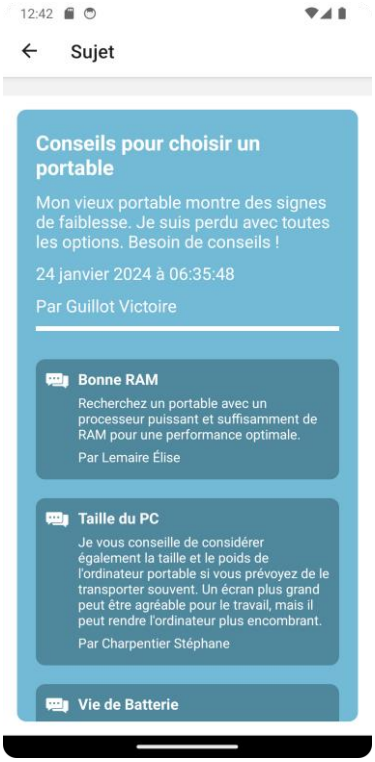


Ensuite, on affiche les messages du sujet (titre, date de post, la personne à l'origine de ce message, description/contenu).


```
<View style={styles.messagesContainer}>
  {!loadingM ? (
    messages.map((message, index) => (
      <View key={index} style={styles.messageContainer}>
        <Image
          source={{
            uri: "https://cdn-icons-png.flaticon.com/512/6456/6456117.png",
          }}
          style={styles.messageIcon}
        />
        <View style={styles.messageContent}>
          <Text style={styles.messageTitle}>{message.title}</Text>
          <Text style={styles.messageElement}>{message.content}</Text>
          <Text style={styles.messageElement}>
            {message.user["@id"] ==
            IRInuage + "api/users/" + user.id ? (
              <>
                <View>
                  <Text style={styles.messageElement}>Par vous</Text>
                  <BoutonsModifierSupprimer message={message} />
                </View>
              </>
            ) : (
              <>
                Par {message.user.lastname} {message.user.firstname}
              </>
            )}
          </Text>
        </View>
      </View>
    ))
  ) : (
    <View style={styles.noMessagesContainer}>
      <Text style={styles.noMessagesText}>Aucun message trouvé.</Text>
    </View>
  )}
</View>
```

Je n'ai pas traité la modification / suppression de message / sujet.

Le visuel →



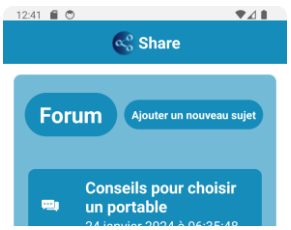
4.5 AJOUT DE SUJET

Il m'a fallu aussi pouvoir ajouter des sujets de conversation. La personne connectée peut donc de cette manière lancer un sujet qui l'intéresse avec tout le monde.

J'ai d'abord commencé à mettre en place le visuel de l'application avec les différents <TextInput> permettant à l'utilisateur d'écrire son sujet. J'ai également mis en place du CSS. J'ai également mis en place deux boutons d'action :

- Un pour annuler l'action, ce qui fait un retour en arrière dans l'application, on revient à l'ancien page visitée
- Un pour ajouter le sujet demandé, ce qui lance la fonction addSujet.

Sur la page de l'ensemble des sujets, j'ai ajouté un bouton qui nous redirige vers ce formulaire d'ajout de sujet.




```
return (
  <>
    <View>
      <Text style={styles.blueTitre}>Ajout de sujet</Text>
      <Text style={{ ...styles.gras, ...styles.ProfilTexte }}>Titre</Text>
      <TextInput
        defaultValue={titleSujet}
        style={[styles.TextInput, { height: 100 }]}
        onChangeText={setTitleSujet}
        multiline
      />
      <Text style={{ ...styles.gras, ...styles.ProfilTexte }}>
        Contenu du premier message
      </Text>
      <TextInput
        value={contentSujet}
        style={[styles.TextInput, { height: 150 }]}
        onChangeText={setContentSujet}
        placeholder="Enter text here..."
        multiline
      />
    </View>

    <TouchableOpacity onPress={addSujet} style={styles.bouton}>
      <Text style={styles.texteBouton}>Ajouter</Text>
    </TouchableOpacity>

    <TouchableOpacity
      onPress={() => {
        navigation.goBack();
      }}
      style={styles.bouton}
    >
      <Text style={styles.texteBouton}>Annuler</Text>
    </TouchableOpacity>
  </>
);
```

Lorsqu'on appuie sur le bouton, on lance cette fonction, permettant d'ajouter un sujet de discussion en envoyant une requête POST à l'API. ➔

Elle vérifie d'abord si les champs de contenu (contentSujet) <TextInput> et de titre (titleSujet) <TextInput> ne sont pas vides. Si les champs sont remplis, elle récupère l'ID de l'utilisateur (idProprietaire), la date actuelle (currentDate) et construit l'IRI de l'utilisateur (userIRI).

Ensuite, on envoie une requête POST à l'API avec les données du sujet, y compris le titre, la date de publication, le contenu, l'IRI de l'utilisateur et une liste vide de messages. Si la requête est réussie, elle redirige l'utilisateur en arrière. Et si une erreur se produit lors de la requête, elle affiche une alerte avec le message d'erreur correspondant.

```
const addSujet = async () => {
  if (contentSujet != "" && titleSujet != "") {
    let idProprietaire = user["id"];
    const currentDate = new Date();
    let userIRI = `${IRInuage}api/users/${idProprietaire}`;

    const data = await fetch(nuage + "api/messages", {
      headers: {
        Accept: "application/ld+json",
        "Content-Type": "application/ld+json",
        Authorization: `Bearer ${token}`,
      },
      method: "POST",
      body: JSON.stringify({
        title: titleSujet,
        datePost: currentDate,
        content: contentSujet,
        user: userIRI,
        messages: [],
      }),
    })
      .then(function (response) {
        navigation.goBack();
        return response.json();
      })
      .catch((error) => {
        alert(error);
      });
  } else {
    alert("Un ou des champs sont vides !");
  }
};
```