PROJET PROSER

Dossier de Synthèse

Ce projet est réalisé dans le cadre d'une formation pour le « Titre de Développeur Web et Web mobile ». Cette formation a été faite à ELAN Formation de Strasbourg durant la période du 19 février au 18 Octobre 2024.

MARZAK Ali

01/10/2024

SOMMAIRE

Table des matières

I. Introduction 5
A. Contexte 5
B. Compétences couverts (REAC) 5
II. Présentation Générale du Système 6
III. Cahier des Charges 6
IV. Technologies utilisées 7
A. Backend 7
1. Symfony 7.1 7
2. Doctrine 8
B. Frontend 9
1. React JS 9
2. Tailwind CSS 9
C. Communication Backend – Frontend 10
V. Environnement de travail 12
A. Utilisation de Git 12
B. Gestion des dépendances avec Composer 12
C. Gestion des Assets avec npm 13
D. Gestion de Projet 13
E. GitHub : Project & Issues 14
VI. SEO (Search Engine Optimization) 15
A. Optimisation des Performances 15
B. Structure du Contenu 16
C. Accessibilité 16
D. Gestion des URLs et Redirections 18
E. Responsive Design 18
VII. Conception du Système 19
A. Architecture du Système 19
1. Architecture Globale 19
2. Architecture REACT 20
B. Maquettage 21
1. Accessibilité 21
2. UX / UI 22
C. Gestion de la Base de données 24
VIII. Sécurité du Système 26

A. Connexion 27
1. Register 27
2. Login 30
3. Logout 31
B. Sécurisation des données 31
1. Hachage du mot de passe 31
2. Protection contre les injections SQL 32
3. Protection contre les attaques XSS 34
4. Protection contre les attaques CSRF 35
5. Protection contre les attaques par Force brute 38
IX. Fonctionnalités principales 43
A. Annonces (Filtrage + Pagination) 43
1. Ajout d'un Slug pour les Annonces 44
2. Filtrage des Annonces 44
3. Pagination 45
B. Messagerie 46
C. Gestion des Rendez-vous 48
1. Création du Rendez-vous 48
2. Compléter le rendez-vous 49
3. Affichage du rendez-vous 51
4. Validation du Rendez-vous 52
D. Gestion de Paiement 53
E. Barre de Recherche 55
F. Dashboard Entrepreneur / Client 56
X. Conformité RGPD 59
A. Collecte et Gestion des Données Personnelles 59
1. Type de données collecté et minimisation 59
2. Mise en place de consentement 60
3. Accessibilité de la politique de Confidentialité 60
B. Droits des Utilisateurs 61
1. Droits d'accès et de modification des données personnels 61
2. Droit à l'oubli 62
C. Sécurité des données personnels 65
XI. Conclusion et Perspectives d'améliorations 66
XII. Remerciements 66
XIII. ANNEXES 67
A. Documentation Technique 67
B. Références et Ressources 71

I. Introduction

A. Contexte

Dans un monde où l'entrepreneuriat prend de plus en plus de place dans le monde professionnel et où les personnes sont de plus en plus connectées, la mise en relation entre des Entrepreneurs et des Clients est devenue primordial pour répondre rapidement et efficacement à divers besoins.

Le projet PROSER s'inscrit dans ce contexte en créant une plateforme numérique permettant de faciliter ces interactions.

Ainsi, nous trouvons des clients recherchant des professionnels compétents et indépendants pour réaliser des services, mais aussi des entrepreneurs cherchant à élargir leur clientèle et mettre en avant leurs compétences.

PROSER offre ainsi un espace sécurisé où chaque partie peut publier des annonces, échanger via une messagerie intégrée, gérer les Rendez-Vous et accompagner la transaction en toute sécurité.

PROSER permet de répondre à tous ces services pour assurer une expérience utilisateur optimale et sécurisée.

B. Compétences couverts (REAC)

AT 1 - Développer la partie Front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- ✓ Maquetter une application
- ✓ Réaliser une interface utilisateur web statique et adaptable
- ✓ Développer une interface utilisateur web dynamique
- ✓ Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

AT 2 - Développer la partie Back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- ✓ Créer une base de données
- ✓ Développer les composants d'accès aux données
- ✓ Développer la partie Back-end d'une application web ou web mobile
- ✓ Mettre en œuvre une solution de gestion de contenu ou e-commerce

II. Présentation Générale du Système

Le système propose un accès authentifié ou non permettant de visualiser des annonces de type Service/Besoin selon différentes catégories.

Une authentification permettrait d'accéder à des fonctionnalités beaucoup plus importantes.

Parmi ces fonctionnalités, nous avons la possibilité de faire une demande pour avoir le Rôle d'entrepreneur dans l'application ce qui permettrait de switcher entre le mode client/Entrepreneur.

Nous avons aussi la possibilité de **publier une Annonce**, de la modifier ou encore de la supprimer, ce qui permettrait par la suite d'échanger via messagerie.

Une fonctionnalité pour la **gestion des rendez-vous** est mise en place avec pour l'entrepreneur, un planning qui lui est dédié.

Une interface de paiement via **Stripe** est accessible pour le client permettant ainsi de valider un rendez-vous.

Le système respecte les règles du **RGPD** en permettant notamment une transparence des données pour l'utilisateur,

III. Cahier des Charges

- Inscription & Authentification
- Formulaire d'Inscription pour avoir le rôle d'Entrepreneur (ROLE_PRO)
- Bouton pour switcher entre les 2 rôles (ROLE_PRO et ROLE_USER)
- Publication d'Annonce de type Service (Entrepreneur) ou Besoin (Client)
- Page Annonce avec des filtres + un slug
- Envoi de 1er message + Messagerie
- Ajout d'un Rendez-vous par l'Entrepreneur via le canal de discussion
- Affichage des rendez-vous dans un Planning pour l'Entrepreneur
- Affichage des rendez-vous via une liste pour les clients
- Intégration d'un formulaire de paiement pour les clients pour compléter un rendez-vous.

IV. Technologies utilisées

A. Backend

1. Symfony 7.1

Pour gérer le côté Serveur de mon application, j'ai fait le choix d'utiliser Symfony qui est un Framework basé sur le langage PHP.

FIGURE 1 SYMFONY 7.1

Le projet est basé sur la dernière version terminée **Symfony 7.1** avec l'utilisation de la version 8.3.9 de **PHP** pour avoir les dernières fonctionnalités.

Pourquoi avoir choisi ce Framework ?

Tout d'abord, Symfony reste le 1er Framework que j'ai appris durant ma formation et que j'ai apprécié durant son utilisation dans mes différents projets de Formation et de Stage.

Ensuite, l'utilisation du langage PHP reste un atout majeur pour un Framework permettant ainsi une bonne gestion des erreurs notamment avec les dernières versions de PHP.

Enfin, Symfony comporte d'innombrables avantages :

- ✓ Une communauté très riche
- ✓ Offre un écosystème riche (bundles, etc…)
- ✓ Architecture modulaire permet une meilleure flexibilité et une meilleure évolution du projet
- ✓ Offre une sécurité complète avec des mises à jour continue

2. Doctrine

Si l'on parle de Symfony, nous sommes dans l'obligation de parler de Doctrine. En effet, la gestion des bases de données est effectuée avec Doctrine.

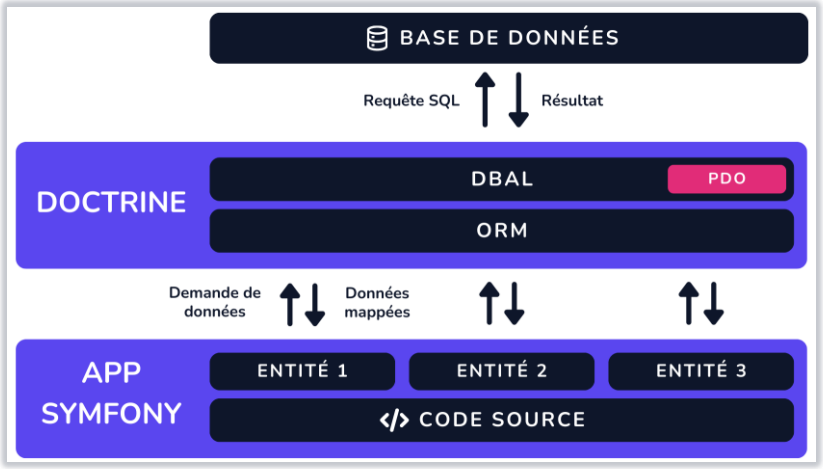


FIGURE 2 COMMUNICATION DE L'APP SYMFONY AVEC DOCTRINE

Doctrine offre un ensemble de bibliothèques PHP pour la gestion des bases de données. Parmi ses composants, nous avons **Doctrine ORM** (Object-Relational Mapping) qui va permettre de faire le mappage des objets PHP aux tables d'une base de données relationnelle.

L'utilisation de Doctrine ORM rend la manipulation de la base de données beaucoup plus simple et intuitive.

De plus, avec l'utilisation du **QueryBuilder**, nous sommes en capacité de construire des requêtes dynamiques DQL qui remplace les requêtes SQL brutes en proposant une version orientée objet.

```sql
SELECT a.*, c.*
FROM articles a
LEFT JOIN commentaires c ON a.id = c.article_id;
```

FIGURE 3 REQUETE SQL

```php
$queryBuilder = $entityManager->createQueryBuilder();
$query = $queryBuilder
    ->select('a, c')
    ->from('App\Entity\Article', 'a')
    ->leftJoin('a.commentaires', 'c')
    ->getQuery();
$result = $query->getResult();
```

FIGURE 4 REQUETE DQL

Enfin, Doctrine offre la possibilité de versionner les modifications lors des migrations de schéma de base de données.

B. Frontend

1. React JS

Dans mon projet PROSER, j'ai fait le choix d'utiliser React JS pour toute la partie Frontend de mon application qui est une bibliothèque en Javascript

J'utilise ainsi la version 18.0 de React JS que j'ai installé via le bundle Symfony **UX-react**.

Ce choix a été guidé particulièrement par mon envie de découvrir cette bibliothèque qui fait l'actualité des développeurs Web.

En effet, la particularité de ReactJS est qu'elle propose une avancée majeure dans le développement d'interfaces utilisateur (UI) très dynamiques et réactives.

Pourquoi avoir choisi Symfony et React JS ?

Il n'y a pas longtemps, l'utilisation de ces 2 aurait été compliqués, mais l'apparition des dernières versions de Symfony 7.1 permet l'utilisation de React sans avoir de soucis et avec des outils en plus coté Serveur permettant une bonne relation (Authentification et Sécurité adapté).

De plus, il s'agit d'une stratégie permettant de maximiser l'efficacité et la qualité de notre projet. En effet, chacun joue un rôle bien défini dans l'architecture de l'application permettant à celle-ci d'être plus propre et d'offrir de meilleures perspectives d'améliorations, mais aussi d'exploiter pleinement leurs capacités.

Dans Ract.js, j'ai fait le choix d'utiliser du JSX (JavaScriptXML) qui est une syntaxe d'extension pour Javascript qui permet d'écrire des éléments React de manière lisible et intuitive.

2. Tailwind CSS

Pour la partie Front, nous combinons React JS avec le framework **Tailwind CSS 3.4.6**. Il s'agit d'un framework de design CSS qui se base sur une approche de classes utilitaires.

Pour ma part, l'utilisation de Tailwind avec React reste une solution très puissante puisqu'il offre une meilleure gestion du code et une meilleure visualisation de l'interface. En effet, j'aime beaucoup savoir le comportement visuel de chaque composant.

Voici les raisons de l'utilisation de Tailwind :

- ✓ Nommage des classes : je fais des parties des personnes qui peuvent être rapidement à court d'idée lorsqu'il s'agit de trouver des noms de classes et de vite me mêler les pinceaux. Tailwind CSS m'évite ainsi ce problème.
- ✓ Gain de temps considérable : En gérant le design directement au niveau du composant React, cela m'évite de faire des allers-retours vers des fichiers CSS.
- ✓ Gain de Performance : Tailwind CSS élimine directement le CSS non utilisé ce qui va réduire considérablement la taille finale des fichiers CSS et donc améliorer les performances.

C. Communication Backend – Frontend

Puisque j'ai utilisé Symfony pour le Back et React pour le Front, qui n'est pas la combinaison optimale, j'ai utilisé 2 méthodes de communication :

- Via le moteur de Template Twig

En effet, j'ai décidé d'adopter cette méthode particulière pour les grandes pages de mon site, comme la page d'accueil, les annonces, et les messages. Bien qu'il existe plusieurs approches possibles, comme utiliser uniquement des requêtes AJAX et intégrer un système de routage de React, j'ai choisi de continuer avec ma configuration initiale, qui ne reposait pas sur un routeur.

Dans l'image ci-dessous, j'intègre mon composant parent de la page Annonce dans mon Twig via la fonction « react_component » qui permettra d'avoir le rendu du composant parent « PageAdvert ».

```
1  {% extends 'base.html.twig' %}
2
3  {% block title %}Annonce{% endblock %}
4
5  {% block body %}
6      <section class="section_advert">
7          <div {{ react_component('PageAdvert', { initialAdverts: adverts, categoryId:categoryId, searchInput:searchInput }) }}>
8              Loading... <i class="fas fa-cog fa-spin fa-3x"></i>
9          </div>
10     </section>
11 {% endblock %}
```

FIGURE 5 FONCTION DE RENDU D'UN COMPOSANT REACT SUR TWIG

- Via des **requêtes AJAX**

Une fois que je parcours les composants React, je communique avec mon Serveur via des requêtes AJAX (Asynchronous JavaScript and XML) qui sont des **requêtes asynchrones** permettant de charger des données en arrière-plan sans avoir à charger la page.

L'élément principal de l'AJAX est l'objet « XMLHttpRequest » qui permet d'envoyer des requêtes HTTP (Hypertext Transfer Protocol) avec sa méthode (GET, POST, etc …) du navigateur au serveur.

```
4   var xhttp = new XMLHttpRequest();
5   xhttp.open("GET", "advert", true);
6   xhttp.send();
```

FIGURE 6 REQUETE AJAX AVEC LA METHODE GET VERS LA ROUTE ADVERT

Pour mon projet, j'ai utilisé une API nommée « fetch » qui est une version plus améliorée de la méthode ci-dessus via l'objet « XMLHttpRequest ».

Elle offre une interface plus simple et une méthode plus puissante pour faire des requêtes.

```
try {
    // Envoie une requête POST à l'URL '/publishAdvert'
    const response = await fetch('/publishAdvert', {
        method: 'POST', // Méthode HTTP utilisée pour la requête
        headers: {
            'X-CSRF-TOKEN': csrfToken // Ajoute le token CSRF pour la protection contre les attaques CSRF
        },
        body: data, // Données à envoyer avec la requête
    });
    // Vérifie si la réponse est OK (statut HTTP 200-299)
    if (!response.ok) {
        throw new Error('Failed to submit form'); // Lance une erreur si la réponse n'est pas réussie
    }
    // Analyse la réponse JSON pour obtenir les données renvoyées par le serveur
    const result = await response.json();
    console.log('Form submitted successfully:', result); // Affiche le résultat de la soumission
} catch (error) {
    // Capture et affiche toute erreur survenue lors de la soumission du formulaire
    console.error('Error submitting form:', error);
}
```

FIGURE 7 REQUETE POST AVEC FETCH API

Les particularités de l'API fetch sont :

- L'utilisation de promesse, d'où l'utilisation de « async/await ».
- La gestion des erreurs de réseau en utilisant « catch ».

V. Environnement de travail

A. Utilisation de Git

Dans mon projet, j'ai utilisé Git qui est un système de contrôle de version qui a joué un rôle primordial dans la gestion du code source de mon projet.

Les **fonctionnalités principales de Git** sont :

- Le contrôle de Version

Git permet de conserver un **historique complet des modifications** que j'ai pu apporter à mon code permettant ainsi de récupérer une ancienne version de mon code, de comparer des versions différentes ou encore de suivre l'évolution du projet.

- Branches et Merges

Git donne la possibilité de travailler sur des fonctionnalités du code via une version parallèle du code qu'on appelle **Branches** permettant ainsi de ne pas affecter la version principale.

Lorsque la/les fonctionnalité(s) sont terminées et correctement testées, on peut fusionner cette branche à notre branche principale en effectuant un **merge**.

Dans mon projet, via la Fonctionnalité **Project** de GitHub, j'ai pu organiser l'ensemble de mon projet en créant des **Issues** qui seront liées à des branches, me permettant ainsi de développer mon code selon les fonctionnalités.

B. Gestion des dépendances avec Composer

Dans un projet, nous sommes amenés à utiliser des bibliothèques que l'on a besoin et qu'on souhaite intégrer dans notre code. Dans mon cas, j'ai utilisé **Composer** qui est un outil de gestion de dépendances pour les projets PHP et qui est nécessaire avant l'installation de Symfony.

Les dépendances seront gérées via le fichier **composer.json** dans lequel nous allons trouver les bibliothèques nécessaires au projet avec leurs versions. Cela va nous permettre une cohérence et une stabilité du code.

De plus, composer va nous permettre de mettre à jour les bibliothèques tout en prenant en compte leurs compatibilités et contraintes.

Ensuite, une autre fonctionnalité de Composer est l'**Autoloading**. En effet, il génère un fichier d'Autoload qui va nous permettre le chargement des classes sans avoir à les inclure manuellement.

C. Gestion des Assets avec npm

Puisque j'ai fait le choix d'utiliser React.js dans mon projet, j'ai installé le gestionnaire de paquet **npm** qui est un gestionnaire par défaut l'environnement node.js.

Ce gestionnaire me permettra d'installer facilement des bibliothèques et des outils Front-end nécessaires au projet.

D. Gestion de Projet

Pour la gestion de mon projet, j'ai décidé d'utiliser la **méthode Agile** dont l'idée est d'apporter une souplesse et performance à mon projet.

Cette méthode est basée sur 3 piliers :

- Transparence : Garantir que tous les membres ont une compréhension commune du projet.
- Inspection : Rendre le travail examinable et évaluable régulièrement.
- Adaptation : Garantir une adaptation dans le développement pour répondre aux besoins et aux exigences des parties prenantes.

Pour la gestion de mon projet, j'ai utilisé l'outil **Trello**, qui m'a permis d'afficher et d'organiser efficacement toutes les tâches. J'ai structuré mon tableau Trello en m'inspirant de deux méthodologies Agile : **Scrum** et **MoSCoW**. Cela m'a permis de visualiser le progrès du projet et de prioriser les tâches de manière efficace.

FIGURE 8 TABLEAU TRELLO AVEC LA METHODE SCRUM ET MOSCOW

J'ai donc utilisé la méthode Scrum qui est une méthode qui se concentre sur des cycles de développement courts appelés « sprints » généralement de 2 à 4 semaines. Pour cela, j'ai organisé mon tableau Trello en plusieurs sections. Tout d'abord, j'ai une table **BackLog** qui contient toutes les fonctionnalités de mon application à mettre en place.

De plus, j'ai aussi une table **Sprint** dans laquelle je déplace les fonctionnalités sélectionnées pour le cycle en cours. Enfin une table **Terminée** qui contient toutes les fonctionnalités qui ont été développées, testées et validées.

Pour la **priorisation des tâches**, j'ai décidé d'utiliser la **méthode MoSCoW**.

Chaque élément de la table BackLog a été priorisé selon la méthode MoSCoW à partir des étiquettes définies.

En se basant sur cette méthode, nous avons 4 étiquettes :

E. GitHub : Project & Issues

Pour mon projet, j'ai aussi utilisé quelques fonctionnalités sympas que propose GitHub comme l'outil **Project** permettant de gérer son projet de manière visuelle comme Trello, mais avec des avantages supplémentaires pour notre code.

En effet, celui-ci offre la possibilité d'interagir avec notre code via une fonctionnalité qui s'appelle **Issues**. En effet, chaque Issue peut être liée à une nouvelle branche de notre code permettant ainsi de mieux organiser le développement de notre application. Pour ma part, lorsque je créais une nouvelle Issue pour la mise en place d'une fonctionnalité, j'ouvrais une nouvelle branche dédiée que je m'assignais. Ensuite, je développais cette fonctionnalité dans sa branche. Lorsque la fonctionnalité a été développée et validée, je fusionnais cette branche à ma branche principale et fermais l'Issue correspondante.

VI. SEO (Search Engine Optimization)

Le **SEO** ou **Search Engine Optimization** correspond à l'ensemble des techniques permettant l'amélioration de la visibilité de notre application dans un moteur de recherche comme Google.

A. Optimisation des Performances

1. Conversion des images en Webp

Etant donné que j'utilise des images dans mon application, j'ai décidé de mettre en place une conversion des images au format Web lorsque l'utilisateur intègre une image d'un autre format.

Pour cela j'ai créé une fonction privée dans mon contrôleur qui va me permettre de convertir mes images en ce format avec les fonctions natives de PHP.

```php
private function convertImageToWebP(string $originalImagePath, string $webpImagePath):
{
    $image = null;
    // Récupère l'extension de l'image
    $extension = pathinfo($originalImagePath, PATHINFO_EXTENSION);
    // Conversion selon le type d'image
    switch ($extension) {
        case 'jpeg':
        case 'jpg':
            $image = imagecreatefromjpeg($originalImagePath);
            break;
        case 'png':
            $image = imagecreatefrompng($originalImagePath);
            break;
        case 'gif':
            $image = imagecreatefromgif($originalImagePath);
            break;
        default:
            throw new \InvalidArgumentException('Type d\'image non supporté');
    }
    if ($image) {
        imagewebp($image, $webpImagePath);
        imagedestroy($image);
    }
}
```

2. Ajout d'un Lazy Loading pour les images

Il s'agit d'une technique d'optimisation qui consiste à charger les ressources uniquement lorsque celle-ci est nécessaire, c'est-à-dire qu'elles deviennent visibles à l'écran. Pour ma part, j'ai décidé de le mettre en place pour ma pagination de la page annonce ou encore pour les images comme ci-dessous.

```jsx
<div className='w-80 h-36 flex justify-center overflow-hidden'>
    <img src={`${ advert.mainImage }.webp`} loading='lazy' alt={`
</div>
```

3. Optimisation grâce aux requêtes préparées

Il faut savoir que lorsqu'on utilise des requêtes préparées, celle-ci est préparé qu'une seule fois et ensuite l'exécution se fait plusieurs fois avec des paramètres différentes. Ce qui améliore les temps de réponse et donc un temps de chargement beaucoup plus bas, ainsi améliorer le référencement.

B. Structure du Contenu

Pour avoir un meilleur SEO, il est primordial de respecter les conventions de la W3C (World Wide Web Consortium) qui sont des standards et des recommandations permettant de garantir l'accessibilité et la pérennité du site Web.

Parmi ces standards, on a l'utilisation des bonnes balises HTML pour avoir une bonne sémantique dans la structure de notre page.

Ou encore l'utilisation de style pour la présentation visuelle des pages. Dans mon cas, j'utilise Tailwind CSS.

C. Accessibilité

Une des solutions pour améliorer le SEO est l'Accessibilité de notre application. En effet, il est essentiel de pouvoir garantir l'accès à toutes les personnes (Handicap Visuels, Auditifs, Moteurs et Cognitifs.

Pour l'accessibilité, il existe deux références importantes qui sont :

- **WCAG** (Web Content Accessibility Guidelines : Il s'agit d'un ensemble des directives développées par le W3C
- **RGAA** (Référentiel Général d'amélioration de l'accessibilité) : Il s'agit d'un référentiel français qui adapte les directives WCAG. Il s'articule autour des mêmes principes que celui des WCAG.

Les **4 principes fondamentaux** que nous allons détailler dans le maquettage sont :

- Perceptibilité
- Utilisabilité
- Compréhensibilité
- Robustesse

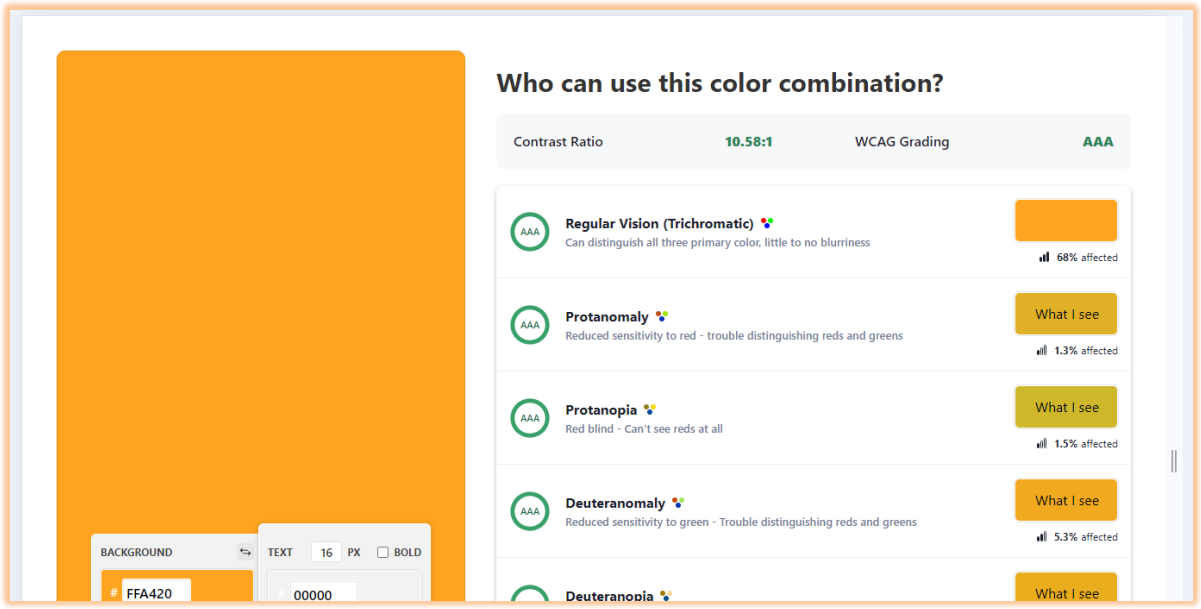
Ainsi, pour répondre à tout cela, j'ai mis en place :

- Ajout d'un attribut **Alt** pour les balises image proposant **un texte alternatif** pour les personnes malvoyantes ou encore pour une image qui ne charge pas.

```
25    <div className='w-80 h-36 flex justify-center overflow-hidden'>
26        <img src={`${ advert.mainImage }.webp`} loading='lazy' alt={`Image principale de l'Annonce ProSer : ${advert.title}`} c
27    </div>
```

- Rendre le site navigable grâce à une bonne sémantique HTML
- Assurer un bon contraste de notre application avec le choix des couleurs

Avec l'utilisation de l'outil en ligne WhoCanUse, je peux vérifier si les couleurs sont compatibles et offrent un bon contraste.



D. Gestion des URLs et Redirections

1. URL & Slug/ url conviviales

Ensuite, pour un meilleur SEO, il est très important d'avoir un URL qui soit lisible et agréable à lire. Pour cela, j'ai utilisé un **Slug** pour l'affichage d'une annonce. L'ajout de ce slug va permettre aussi de mieux référencer cette **Annonce** et donc il est plus accessible via un moteur de recherche.

2. Redirections

De plus, il est très important de pouvoir rediriger un utilisateur lorsqu'il y a une erreur que ça soit côté Serveur ou Client pour qu'il puisse rester un maximum sur notre application et donc améliorer le référencement naturel.

J'ai pour cela mis en place des pages d'erreur pour les erreurs courantes comme erreur 404 ou erreur 403.

E. Responsive Design

Avoir une application qui est totalement Responsive améliore le SEO, puisqu'à notre époque, la majorité de la navigation sur le Web se fait à partir de téléphone. C'est pour cela qu'il est recommandé de développer une application en **Mobile First**. Une recommandation que j'ai suivie puisque mon application Web a été d'abord pensée en version mobile.

De plus, j'ai décidé d'utiliser **Tailwind CSS** qui facilite la mise en place d'un responsive.

VII. Conception du Système

Après avoir énoncé les différentes technologies utilisées dans mon projet et mon environnement de travail, il est très important de se pencher sur la conception du Système. Notamment, sur le choix de l'architecture de notre système qui nous permettra d'avoir une meilleure structure de notre application. Dans mon cas, j'ai dû faire le choix d'une architecture côté Serveur avec l'utilisation du Framework Symfony, mais aussi d'une architecture côté client avec l'utilisation de la bibliothèque React.js.

A. Architecture du Système

1. Architecture Globale

Pour mon projet PROSER, il est essentiel de faire le choix d'un cadre de développement pour une meilleure organisation du code et une facilité lors de la maintenance.

Parmi les différentes Design Pattern, j'ai décidé d'utiliser l'architecture MVC (Model-View-Controller) qui a fait ses preuves dans la gestion d'applications complexes et qui le design pattern utilisé avec le Framework Symfony.

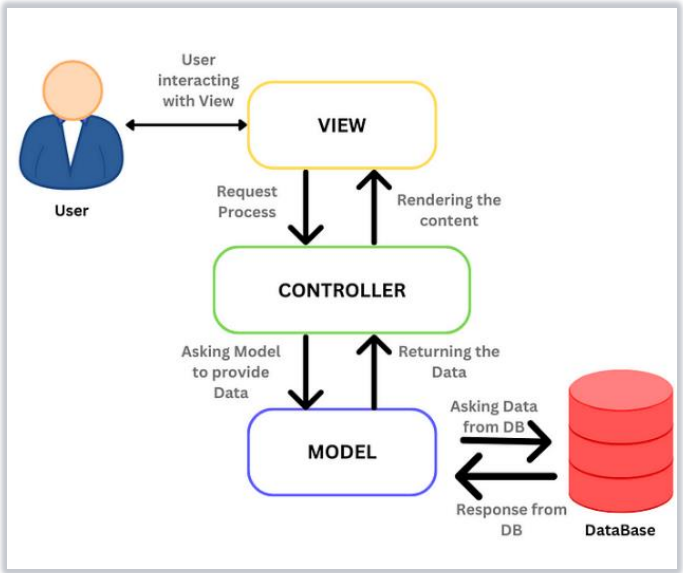


FIGURE 9 ARCHITECTURE MVC

Cette architecture a la particularité de diviser l'application en 3 parties offrant une meilleure organisation, une réutilisation de code et une meilleure maintenance :

- **Model**

Le modèle est la partie qui est responsable de la gestion des données. Il interagit avec la base de données en effectuant des opérations (CRUD) de Création, Lecture, Mise à jour et de Suppression.

- **View**

Ce composant représente l'interface utilisateur et l'affichage des données. Se concentre uniquement sur la présentation des informations. Sur Symfony, on utilise le moteur de Templates Twig.

- **Controller**

Le Controller est la partie qui fera l'intermédiaire entre le modèle et la vue ou encore entre le client et la base de données.

Il s'occupera de recevoir les requêtes HTTP de l'utilisateur, de faire les vérifications nécessaires et d'interroger le modèle pour récupérer les données nécessaires ou d'effectuer des actions nécessaires sur la base de données.

2. Architecture REACT

Il faut savoir que React.js est basé sur le concept de composants, on dit qu'il a un **Design atomique**. En effet, lorsqu'on utilise du React, l'objectif est de séparer notre interface utilisateur en des composants de plus en plus petits dont l'objectif est de les rendre réutilisable.

Ainsi, un composant pourra gérer son propre état et ses propriétés ce qui va nous permettre de rendre une page beaucoup plus dynamique entre les différents composants.

Dans React.js, il existe 2 types de composants :

- **Composants de classe** : Les premières versions de React ont débuté avec ce type de composants. Ce sont des classes Javascript qui étendent la classe React.component.
- **Composants Fonctionnels** : ces types de composants ont vu le jour par la suite qui rendent le développement beaucoup plus simple et favorise la réutilisation des composants. Ces fonctions acceptent des arguments en entrée qu'on appelle **props** et retournent un élément React.

Avec l'arrivée des composants fonctionnels, les Hooks ont vu le jour. Ce sont des fonctions de React qui nous permettent d'utiliser les états, le cycle de vie et d'autres fonctionnalités de React dans nos composants fonctionnels.

Parmi ces Hooks, j'en ai utilisé 3 dans mon projet :

- **useState** : il permet de gérer l'état local d'un composant

```
5   // State contenant le tableau des catégories
6   const [categories, setCategories] = useState([]);
7   // State contenant le tableau des catégories sélectionnés
8   const [selectedCategories, setSelectedCategories] = useState(initialCategories);
```

- **useEffect** : il permet de gérer le cycle de vie d'un composant. Il remplace les méthodes de cycle de vie des composants de classe comme

componentWillUnmount. Ce Hook m'a souvent permis de communiquer avec ma base de données lorsque je souhaité envoyer une requête Ajax quand le composant était monté.

- **useContext** : il permet d'accéder à des states sans avoir à les passer au niveau des props.

B. Maquettage

Pour la partie Front de mon application, j'ai décidé d'utiliser l'outil **Figma** pour mon Maquettage. En effet, le maquettage est une étape essentielle dans le processus de conception d'une application qui va nous permettre de visualiser notre application avant de passer à la phase de développement.

Pour le maquettage, il existe 3 types de maquettage :

- **Wireframe** : Il s'agit d'un schéma simplifié de notre application qui va mettre en avant la structure de nos pages.
- **MockUp** : Il s'agit d'une version beaucoup plus réaliste de notre application. Contrairement aux Wireframe, celui-ci inclut des éléments graphiques représentant l'interface utilisateur de notre application.
- **Prototype** : C'est la dernière étape du maquettage de notre site. Il s'agit d'une version fonctionnelle de l'application qui sera donc interactive.

Pour ma part, je suis partie sur un maquettage beaucoup plus proche d'un **Wireframe** que d'un MockUp malgré que j'aie ajouté des couleurs.

De plus, j'ai fait le choix de faire du Mobile First Design qui est une approche qui privilégie une conception des interfaces pour les appareils mobiles avant de les adapter aux écrans plus grands.

1. Accessibilité

Pour le maquettage, j'ai pris en considération l'accessibilité qui se repose sur 4 principes fondamentaux qui sont :

- Perceptibilité : Tous les éléments de notre site doivent être perçus par l'utilisateur.
- Utilisabilité : L'utilisateur doit être en capacité d'utiliser tous les composants de notre application sans rencontrer de problème.
- Compréhensibilité : Chaque élément de notre application doit être facile à comprendre. Les informations doivent être présentées de manière claire et logique en utilisant par exemple un langage simple.
- Robustesse : Notre application doit être compatible à n'importe quel environnement.

2. UX / UI

UX Expérience Utilisateur :

L'expérience utilisateur correspond à la facilité avec laquelle un utilisateur va parcourir une application Web.

Pour cela, il existe plusieurs choses que j'ai mises en place pour améliorer l'expérience utilisateur :

- **Affordance** : Elle fait référence à la manière dont les éléments de l'interface suggèrent leur utilisation. Par exemple, lorsqu'un utilisateur voit un bouton, il doit savoir que ce bouton est présent pour être cliqué ou qu'un lien hypertexte va nous rediriger vers une autre page. Pour cela, j'ai fait en sorte d'ajouter un effet ombre à mes boutons ou encore des Hover et pour les boutons avec des icones comme dans ma NavBar, j'ai décidé d'ajouter un mot pour décrire l'icône comme pour la redirection vers la Page Home.
- **Loi de Fitts** : Cette loi nous dit que plus l'Utilisateur atteint rapidement son objectif, plus son expérience utilisateur sera positive. J'ai donc mis en place les règles des 3 cliques qui vont permettre à l'utilisateur d'accéder à n'importe quelle partie de mon application en moins de 3 cliques.
- **Loi de Hicks** : Scientifiquement, il est prouvé que plus un individu a de choix, plus la décision sera longue à prendre. Pour cela, j'ai fait en sorte que mon application soit simple et épurée, proposant un nombre d'options limité ce qui va mener l'utilisateur vers une navigation plus fluide de mon application Web.
- **Choix des mots** : De plus, il est très important de choisir des mots clairs et pertinents que ça soit au niveau du titre ou des descriptives ce qui va rendre l'application plus simple à comprendre et donc à naviguer. En effet, un utilisateur qui ne comprend pas un bouton aura moins envie de cliquer dessus.

UI Interface Utilisateur :

L'Interface Utilisateur se rapporte à tous l'environnement graphique ou visuelle auquel un utilisateur va faire face lorsqu'il accède à une application Web.

Pour cela, il est très important d'avoir une interface agréable et facile d'utilisation car plus un utilisateur est sur un site agréable à visiter et plus il va rester sur notre application.

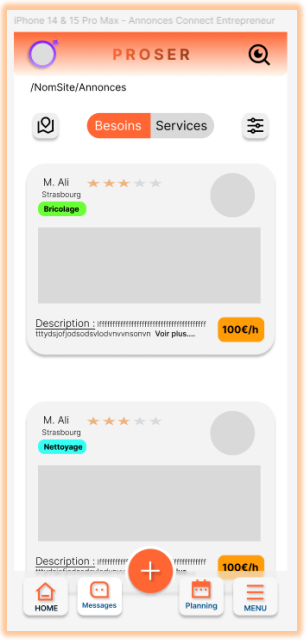
Pour améliorer l'interface utilisateur, j'ai fait plusieurs choix importants :

- **Cohérence visuelle** : J'ai décidé pour mon application d'utiliser des couleurs plus conviviales et chaleureuses. Pour cela, j'ai décidé de partir sur une couleur Orange qui se rapproche un petit peu du rouge. Pour renforcer cette cohérence visuelle, j'ai aussi utilisé les palettes de couleurs proposées par Tailwind CSS qui ont été choisis pour développer le front d'une application de manière cohérente.

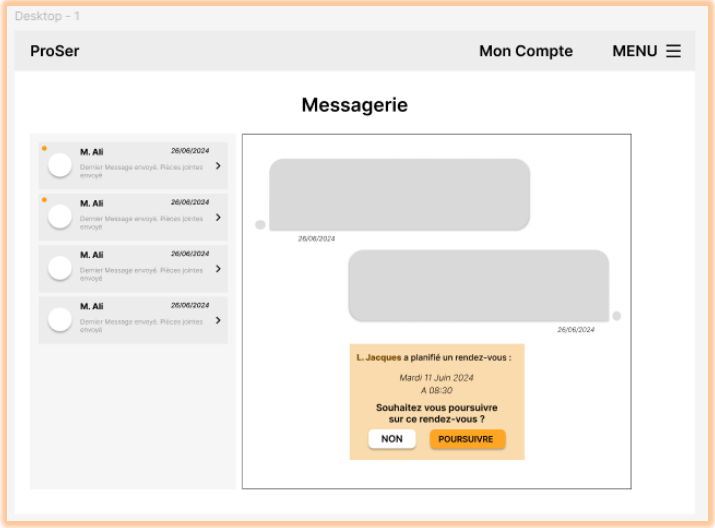
- **Réactivité et feedback visuel** : Lorsque l'utilisateur interagit avec l'interface, il reçoit un feedback visuel immédiat, soit via un changement de couleur, une animation permettant à l'utilisateur de comprendre rapidement l'effet de ses actions. C'est aussi pour cela que j'ai décidé d'utiliser la bibliothèque React.js qui permet de rendre mon application beaucoup plus dynamique et réactive.
- **Responsive Design** : En effet, que mon application s'adapte à n'importe quel type d'écran rend l'interface travaillée quel que soit l'appareil utilisé et donc l'utilisateur garde une interface agréable.

En respectant tous ces points, j'ai été amené à ce maquettage :

➢ Wireframe Version mobile



➢ Wireframe Version Tablette/Desktop



C. Gestion de la Base de données

Dans une application Web, lorsque nous sommes amenés à manipuler des informations, il est primordial d'utiliser une méthode pour la conception d'une base de données.

Dans mon application, j'ai décidé d'utiliser la **méthode MERISE** qui une méthode française d'analyse, de conception et de réalisations de systèmes d'informations.

Cette méthode se repose sur une séparation claire de 3 niveaux : niveau conceptuel, niveau logique et niveau physique.

Dans notre cas, nous avons tout d'abord une étude préliminaire dans laquelle on identifie les besoins et on définit les objectifs du projet. Suite à cela, on met en place un modèle qui représente les données qu'on appelle **Modèle conceptuel des données (MCD)**. Ensuite, on adapte ce MCD à un modèle plus détaillé qui est le Modèle logique des données (MLD).

Modèle conceptuel de Données

Le MCD est considéré comme un schéma qui va modéliser la structure de notre base de données. Dans ce schéma, on va y trouver des entités qui sont des objets pour notre système, accompagnés de leurs attributs qui sont des caractéristiques qui les décrivent et nous permet également de concevoir les relations entre ces entités qui vont permettre d'illustrer comment elles interagissent et se connectent les unes aux autres.

Il faut que savoir que notre MCD se base sur les besoins que nous avons énoncés plus tôt mais aussi sur le respect des principes du Règlement général de protection des données, c'est-à-dire que nous avons pris en compte la sécurité et la minimisation des données dès cette phase de conception, ainsi que sur les éléments réfléchis lors de mon maquettage.

Le MCD de mon projet est en annexe ici.

Comme dit, dans un MCD, nous avons des entités mais aussi des cardinalités qui sont des couples de valeur que l'on va trouver entre chaque entité et ses associations liées.

Dans ce Projet, j'ai mis en place 10 entités :

- **Adverts** :
    - **ManyToMany** à l'entité **Category**
    - **OneToMany** à l'entité **Images_to_advert**
    - **OneToMany** à l'entité **Appointment**
    - **OneToMany** à l'entité **Discussion**
    - **ManyToOne** à l'entité **User**

- **Appointment** :
    - **OneToMany** à l'entité **ValidationCode**
    - **2 ManyToOne** à l'entité **User** (Entrepreneur et Client)
    - **ManyToOne** à l'entité **Discussion**
- **Discussion**
    - 3 **ManyToOne** à l'entité **User** (Client et Entrepreneur et dernier utilisateur)
    - **OneToMany** à l'entité **Message**
- **Message**
    - 2 **ManyToOne** à l'entité **User** (Expéditeur et destinataire)
- **User**
    - **OneToMany** à l'entité **AccountManagement**
    - **OneToMany** à l'entité **PasswordChangeRequest**
    - **OneToOne** à l'entité **Opinion**

Modèle logique de Données

Le MLD est une étape clé de la conception de notre base de données puisqu'elle permet de traduire le MCD en un modèle plus détaillé qui prend en compte la structure logique de nos données.

Dans ce schéma, on aura les relations entre les différents objets qui sont représentés par des tables. De plus, c'est dans ce schéma que seront définis les types de données, les clés primaires, mais aussi les clés étrangères résultant des relations. Pour ma part, dans mon MCD, j'ai une cardinalité en ManyToMany qui va donner dans le MLD, la création d'une table associative qui contient les clés étrangères des 2 tables. Sur Symfony, on aura donc la création d'une entité à part entière.

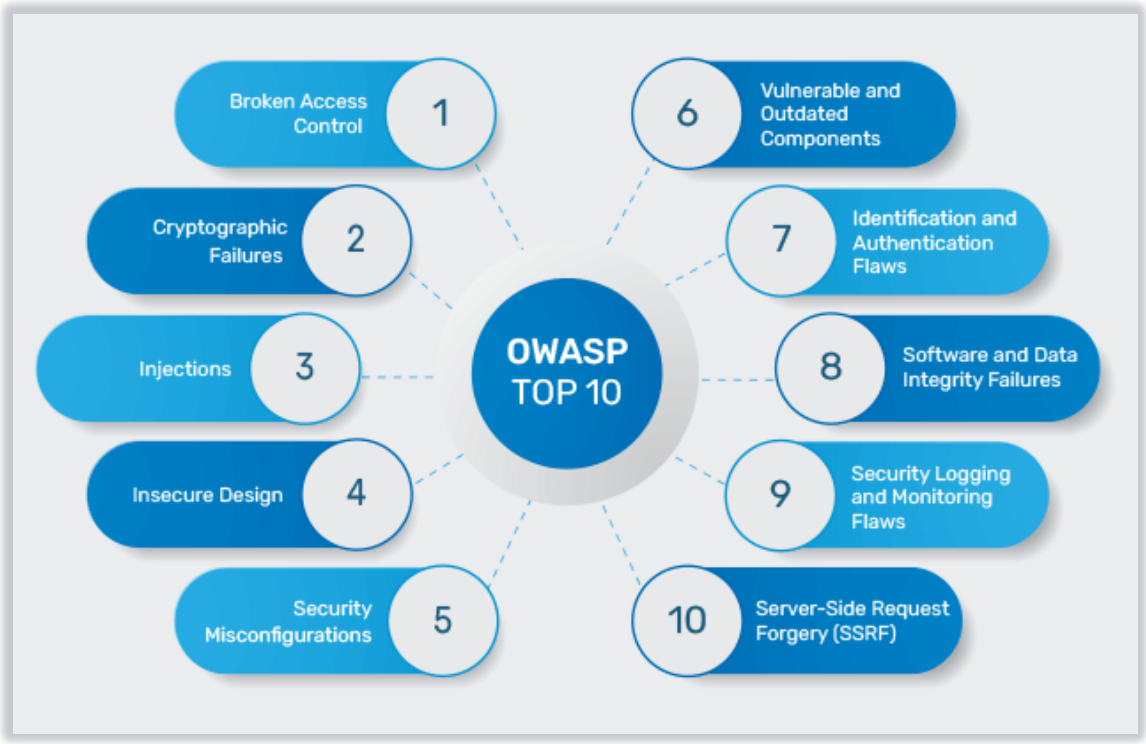
Le MLD de mon projet est en annexe ici.

VIII. Sécurité du Système

Lorsqu'on fait la conception d'un Système, il est primordial de penser à la sécurité de notre système. Enfin dans le monde numérique actuel, la sécurisation des systèmes d'information reste un enjeu crucial dans lequel nous avons une évolution constante des menaces. Ainsi pour répondre à ces attentes et se prémunir face à ces risques, des organismes ont vu le jour telles que **l'OWASP** (Open Web Application Security Project) et **l'ANSSI** (Agence nationale de la sécurité des systèmes d'information).

L'OWASP est une communauté en ligne à but non lucratif qui se consacre à la sécurité des applications Web. L'un des contenus qu'elle propose est le « Top 10 » qui est un rapport qu'elle met à jour régulièrement présentant les 10 risques les plus critiques en matière de sécurité des applications Web.

La dernière mise à jour du **Top 10** date de 2021 :



On y trouve par exemple (voir Annexe pour plus d'informations) :

- En tête de liste, une défaillance dans le **contrôle d'accès**. Ainsi des utilisateurs non autorisés peuvent accéder à des ressources.
- Des vulnérabilités liées à **des injections** : Injection SQL ou XSS.
- L'utilisation de bibliothèques ou Framework **obsolètes**.
- Des failles dans les mécanismes d'**Authentification** ou de **gestion de Session.**

L'ANSSI quant à elle est un service français créé par décret en 2009. Elle offre donc un cadre national pour la protection des systèmes d'information en fournissant des **bonnes pratiques**, des recommandations mais aussi des certifications.

Elle propose **10 bonnes pratiques** pour se protégeait de la plupart des risques numériques, voici quelques-unes :

- Gestion du mot de passe : Mot de passe différent pour chaque accès, mot de passe complexe, ne pas le communiquer à un tiers.
- Mis à jour régulier
- Protection face au virus et logiciels malveillants
- Evitez les réseaux publics

A. Connexion

Dans mon projet WEB, j'ai décidé de mettre en place une **authentification** pour les utilisateurs. Il se trouve que Symfony propose la gestion d'inscription et de connexion avec une prise en charge de certaines sécurités. Pour la gestion de l'authentification, Symfony fournit « Security Bundle » contenant un ensemble d'outils de sécurité.

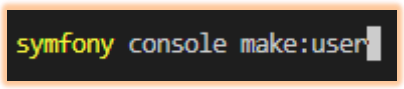
1. Register

Avant de pouvoir s'authentifier, il est primordial de s'inscrire à notre application en stockant nos informations dans la base de données pour pouvoir être repéré lors de l'Authentification.

Pour pouvoir gérer cela, nous allons le faire en **5 parties :**

- Création de **l'Entité User**

Via la commande Symfony suivante :



Cette entité User stockera les informations du nouvel utilisateur dans notre base de données, tel que :

- L'Email
- L'Empreinte numérique du mot de passe
- La vérification du compte
- Les informations personnelles : Nom, prénom, etc. …

- Mise en place du **formulaire d'inscription**

Etant donné que j'utilise **React.js**, j'ai fait mon **formulaire d'inscription manuellement** sans utiliser le formulaire proposé par Symfony via la commande :

```
symfony console make:registration-form
```

- **Mise en place du contrôleur** pour gérer l'inscription

Dans le contrôleur « RegistrationController », on récupère les entrées du formulaire d'inscription via la requête HTTP et on les stocke dans notre base de données.

On fait aussi une vérification de l'email en envoyant une confirmation par Email pour valider le compte.

- **Hachage du mot de passe**

En effet, avant de stocker nos informations, on fait un **hachage irréversible** du mot de passe permettant de stocker une **empreinte numérique du mot de passe** rendant le mot de passe **irrécupérable** en clair.

Pour hacher notre mot de passe, Symfony propose une fonction « hashPassword » qui gère cela, via l'interface d'objet « UserPasswordHasherInterface ».

```
81
82          // Insertion à l'utilisateur du mot de passe haché
83          $user->setPassword(
84              // Fonction de hachage gérée par Symfony via "UserPasswordHasherInterface"
85              $userPasswordHasher->hashPassword(
86                  $user,                              // User lié à ce mot de passe
87                  $data['plainPassword']              // Mot de plasse en clair
88              )
89          );
```

Ainsi cette fonction nous renverra une empreinte numérique que l'on pourra stocker dans notre base de données en tant que mot de passe.

Pour avoir plus détails sur le Hachage du mot de passe et son fonctionnement, voir la rubrique « Sécurisation des données ».

- **Vérification de l'Email** de l'utilisateur

Pour la vérification, on envoie d'abord un mail qui va nous permettre de vérifier notre compte. Pour cela, on utilise la classe **« EmailVerifier »** de Symfony contenant la méthode **« sendEmailConfirmation »** qui sera responsable de l'envoi de l'email de vérification.

```php
        // Envoi mail de confirmation
        $this->emailVerifier->sendEmailConfirmation('app_verify_email', $user,
            (new TemplatedEmail())
                ->from(new Address('support@proser.com', 'proser'))
                ->to($user->getEmail())
                ->subject('Please Confirm your Email')
                // Utilisation d'un template Twig pour le mail
                ->htmlTemplate('registration/confirmation_email.html.twig')
        );
```

FIGURE 10 CREATION ET ENVOI D'UN EMAIL DE CONFIRMATION

Lorsque l'email est envoyé, on doit passer à la phase de vérification lorsque l'utilisateur valide l'email.

Pour cela, dans le contrôleur, nous avons une fonction qui est appelé via une route lorsqu'on clique dans l'email de confirmation.

Cette fonction va donc gérer la confirmation de l'email en utilisant la méthode **«handleEmailConfirmation »** de la classe **« EmailVerifier »** et donc permettre l'accès au compte.

```php
    #[Route('/verify/email', name: 'app_verify_email')]
    public function verifyUserEmail(Request $request, TranslatorInterface $translator): Response
    {
        // Empêche l'accès à la méthode à toute personne non authentifiée
        $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

        // Traite la confirmation de l'email et gère les redirections
        try {
            $this->emailVerifier->handleEmailConfirmation($request, $this->getUser());
        } catch (VerifyEmailExceptionInterface $exception) {
            $this->addFlash('verify_email_error', $translator->trans($exception->getReason(), [], 'VerifyEmailBundle'));

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

        $this->addFlash('success', 'Your email address has been verified.');

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

FIGURE 11 FONCTION DU REGISTRATIONCONTROLLER POUR LA VERIFICATION DE L'EMAIL

2. Login

Pour l'authentification d'un utilisateur, il nous faut ajouter un formulaire de connexion. Fort heureusement, « Security Bundle » est fourni avec différents authentificateurs (Voir Annexe). Parmi eux, nous avons le **« FormLoginAuthenticator »** proposant un formulaire avec un Email et un mot de passe.

Nous l'installons via la commande Symfony :

```
symfony console make:security:form-login
```

Ensuite dans le fichier de configuration « security.yaml », nous apportons quelques modifications au form-login pour spécifier les routes vers le formulaire et la vérification d'authentification, la redirection lors d'une authentification réussi mais aussi les protections nécessaires tel que le CSRF :

```yaml
firewalls:
    dev: ...
    main:
        lazy: true
        provider: app_user_provider

        # FormLoginAuthenticator
        form_login:
            # Route permettant d'accéder au formulaire de connexion
            login_path: app_login
            # Route vers la vérification des identifiants lors de la soumission du formulaire
            check_path: app_login
            # Page de redirection lors d'une authentification réussi
            default_target_path: app_home
            # Activation de la protection contre la faille CSRF
            enable_csrf: true
```

FIGURE 12 CONFIGURATION DU FORMULAIRE DE CONNEXION DANS LE FICHIER "SECURITY.YAML"

Dans le contrôleur de connexion, nous pouvons gérer les erreurs d'authentification à l'aide de la classe « AuthenticationUtils » proposant 2 fonctions : celle de récupérer l'erreur d'authentification et l'autre de récupérer le dernier identifiant d'authentification.

```php
class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // Récupère l'erreur lors de l'authentification si il y en a une
        $error = $authenticationUtils->getLastAuthenticationError();

        // Récupère le dernier identifiant inséré dans le formulaire de connexion
        $lastUsername = $authenticationUtils->getLastUsername();

        // Retourne le formulaire d'Authentification avec erreur ou non
        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }
```

FIGURE 13 FONCTION D'AUTHENTIFICATION DANS LE CONTROLEUR DE SECURITE

3. Logout

Lorsqu'on parle d'authentification, on parle aussi de déconnexion. Pour la configurer, on s'en occupe dans le fichier de configuration « security.yaml » :

```
31          logout:
32              # Spécifie la route permettant la déconnexion
33              path: app_logout
34              # Page de redirection lors d'une déconnexion
35              target: app_login
```

FIGURE 14 CONFIGURATION DE DECONNEXION DANS "SECURITY.YAML"

B. Sécurisation des données

1. Hachage du mot de passe

Lorsqu'on intègre une authentification, il est très important qu'une personne malveillante n'est pas accès aux mots de passe des utilisateurs, cela pourrait avoir des conséquences désastreuses pour les utilisateurs et leur données personnelles.

Pour s'assurer qu'il soit impossible de récupérer un mot de passe en clair, on utilise donc une méthode qu'on appelle « **hachage** » qui consiste à transformer de manière **irréversible** un mot de passe en clair en une chaîne de caractères unique de longueur fixe, appelée **empreinte numérique** à l'aide d'un algorithme de hachage.

Il existe de nombreux algorithmes de hachage, qu'on peut catégoriser dans 2 groupes :

- Algorithmes Forts : Considérés comme très sécurisé. Exemple : **Bcrypt**, Argon.
- Algorithmes Faibles : Considérés comme peu sécurisé. Exemple : MD5, SHA.

Lorsqu'on utilise des fonctions natives PHP pour le hachage de mot de passe, celui-ci s'occupe d'utiliser automatiquement un algorithme de hachage fort qui est en tendance. Par défaut, il utilise actuellement l'algorithme **Bcrypt**. Dans mon projet, Symfony propose également une fonction de hachage de mot de passe qui fonctionne sur le même principe.

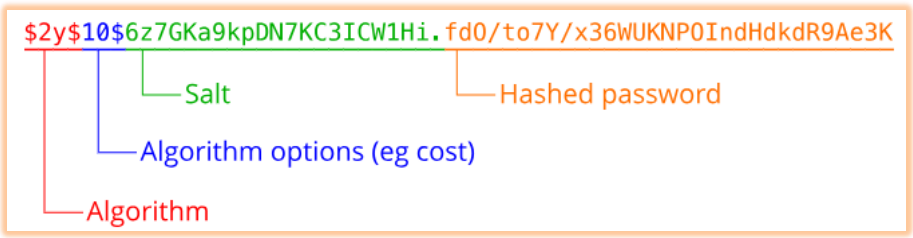


FIGURE 15 EMPREINTE NUMERIQUE APRES LE HACHAGE DU MOT DE PASSE AVEC BCRYPT

Comme nous pouvons le constater, une empreinte numérique est décomposée en 4 parties :

- **Version** de l'algorithme : Elle indique l'algorithme de hachage utilisé et sa version. Dans cet exemple, **$2y$** correspond à une des versions de l'algorithme Bcrypt.
- **Facteur de coût** (Cost) : Elle détermine le nombre de fois que l'algorithme va exécuter son processus de hachage. Ainsi un coût de 10 comme ici signifie que l'algorithme effectuera **2^10 fois** le processus de hachage.
- **Sel** (Salt) : Il s'agit d'une valeur aléatoire qu'on ajoute à notre mot de passe avant son hachage. Ainsi, avec l'ajout du sel, 2 même mots de passe ne peuvent obtenir le même mot de passe haché. De plus, elle permet une protection contre les attaques par table arc-en-ciel, qui sont des tables précalculées de hachages pour des mots de passe courants.
- **Mot de passe haché** : La dernière partie correspond à notre mot de passe haché.

L'ajout d'un Cost et d'un Salt nous permet aussi de protéger contre les attaques par force brute. En effet, plus le facteur de coût est élevé, plus les ressources utilisées et la durée de hachage sont élevées. Ainsi, que le Salt qui complexifie le processus de hachage, rendant la durée de hachage plus élevée.

2. Protection contre les injections SQL

➢ Les Injections SQL, c'est quoi ?

Il s'agit d'une faille de sécurité présents dans les applications Web, qui se produit lorsqu'une personne mal intentionnée parvient à insérer ou manipuler des requêtes SQL dans une base de données par le biais d'un champ de saisie utilisateur ou par Url (méthode GET).

Cette faille permettrait à l'attaquant de contourner les contrôles d'accès pour pouvoir corrompre des données ou encore pour récupérer des données sensibles venant de la base de données.

➢ Comment s'en protégeait ?

Pour se protégeait de cette faille, nous devons utiliser des **requêtes préparées**. La préparation des requêtes SQL est une technique qui se déroule en 2 étapes :

- La Préparation :

Cette étape consiste à tout d'abord envoyer la structure de la requête à la base de données en remplaçant les données dynamiques provenant du côté Client par des paramètres. Cette étape de préparation peut être divisée en 2 phases :

- Phase de **préparation** dans laquelle on construit la requête en spécifiant sa structure.
- Phase de **compilation** dans laquelle on vérifie la syntaxe de notre requête par le serveur.

Voici une requête SQL non préparé :

```
4
5   SELECT * FROM users WHERE username ='admin' AND password ='password123';
6
```

Et voici une requête SQL préparé sous PHP :

```
39
40  // Requête préparé avec des marqueurs nominatifs
41  $stmt = $pdo->prepare('SELECT * FROM users WHERE username = :username AND password = :password');
42
```

- L'exécution :

Après avoir préparé la requête SQL, on passe à l'exécution de la requête qui consiste à lier nos valeurs provenant du coté client aux paramètres présents dans les requêtes préparées.

```
40  $username = 'admin';
41  $password = 'password123';
42
43  $stmt->execute([':username' => $username, ':password' => $password]);
44
```

➢ Protection dans mon projet

En utilisant Symfony, je n'utilise plus les requêtes SQL mais plutôt des **Requêtes DQL**.La logique reste la même sauf que la syntaxe de la requête change :

```
20      public function find(User $username, string $password)
21      {
22          return $this->createQueryBuilder('u')
23              ->where('u.username = :username')
24              ->andWhere('u.password = :password')
25              ->setParameter('username', $username)
26              ->setParameter('password', $password)
27              ->getQuery()
28              ->getResult();
29      }
```

3. Protection contre les attaques XSS

Il s'agit d'une faille de sécurité qui se produit lorsqu'une personne mal intentionnée parvient à injecter du code Javascript malveillant dans des pages Web. L'injection d'un script malveillant peut amener à la collecte d'informations sensibles.

Il existe 3 types d'attaques XSS :

- **XSS permanent** (stocké) : Dans ce cas, le script malveillant est stocké sur le serveur et est donc exécuté à chaque fois qu'un utilisateur y accède soit en visitant une page ou en effectuant une action côté Serveur.
- **XSS non permanent** (réfléchi) : Ici, le script malveillant est injecté dans une requête HTTP et qu'il est renvoyé par le serveur sans être stocké d'où le fait qu'on dit XSS réfléchi.
- **Self XSS** : Il s'agit d'une attaque XSS où l'utilisateur est trompé, par un attaquant via par exemple de techniques de pishing, en injectant un script malveillant depuis sa machine.

Pour se protégeait de cette faille de sécurité, il existe plusieurs moyens de s'en protéger :

- Filtrage des données

Pour se protéger de cette faille, j'ai décidé de filtrer mes entrées grâce à des fonctions natives PHP de filtrage qui permettent de filtrer selon différents filtres.

```php
// Filtrage des entrées contre les failles XSS via des filtres spécifiques
$title = filter_input(INPUT_POST, 'title', FILTER_SANITIZE_STRING);
$description = filter_input(INPUT_POST, 'description', FILTER_SANITIZE_STRING);
$expirationDate = filter_input(INPUT_POST, 'expirationDate', FILTER_SANITIZE_STRING);
$city = filter_input(INPUT_POST, 'city', FILTER_SANITIZE_STRING);
$mobility = filter_var($request->request->get('mobility'), FILTER_VALIDATE_INT);
$price = filter_var($request->request->get('price'), FILTER_VALIDATE_FLOAT);
```

- Echappement des données

Nous pouvons utiliser la fonction native **Htmlspecialchars** qui nous permet d'échapper les données et de se protégeait contre les failles XSS.

- Validation des Entrées Utilisateurs

```php
// Validation des champs obligatoires
if (empty($title) || empty($description) || empty($city) || $mobility === false || $price === false) {
    return new JsonResponse(['error' => 'Missing or invalid required fields'], JsonResponse::HTTP_BAD_REQUEST);
}

// Validation des catégories sélectionnées
$selectedCategories = $request->request->all('categories', []);
if (!is_array($selectedCategories) || array_filter($selectedCategories, 'is_numeric') !== $selectedCategories) {
    return new JsonResponse(['error' => 'Catégories Invalides'], JsonResponse::HTTP_BAD_REQUEST);
}
```

4. Protection contre les attaques CSRF

Les attaques CSRF (Cross-Site Request Forgery) ou encore « Falsifications de demande intersite » sont des attaques où un utilisateur authentifié est amené à effectuer une action à son insu sur une applicationWeb où il est connecté.

➢ Comment fonctionne cette attaque CSRF ?

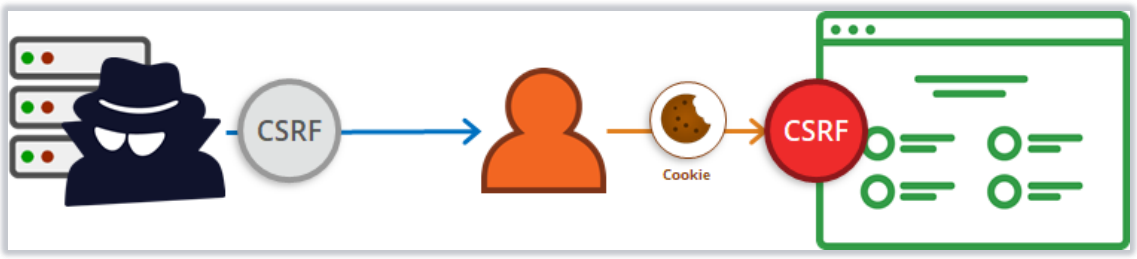


FIGURE 16 ILLUSTRATION D'UNE ATTAQUE CSRF

Pour comprendre son fonctionnement, nous allons prendre un exemple concret :

- Tout d'abord, nous avons un utilisateur qui se connecte à une application Web « A » comme à son habitude. Ainsi, un cookie de session est généré pour cette session.
- L'attaquant va envoyer à la victime un mail par exemple contenant un lien d'un site malveillant contenant un formulaire ressemblant à celui du site « A ».
- L'utilisateur va appuyer sur le lien et ce site malveillant va envoyer une requête HTTP au site officiel en utilisant les cookies de session stocké de l'utilisateur, à son insu.
- Le site officiel va recevoir la requête et l'exécuter en pensant que l'utilisateur est derrière cette action.

➢ Comment s'en protégeait ?

Pour s'en protégeait, nous allons créer un Token de CSRF unique côté Serveur que nous allons stocker dans la session et qu'on envoie côté client avec le formulaire. Lorsque le client soumet le formulaire, il enverra les champs de formulaire ainsi que le Token CSRF qui est unique.

Côté serveur, lorsqu'on reçoit les champs, on vérifie d'abord si le Token CSRF correspond à celui qui est stocké dans la session.

➢ Protection dans mon projet

Etant donné que j'utilise React.js pour les formulaires, j'ai dû mettre en place manuellement la protection contre les attaques CSRF en 3 étapes :

- Création d'une fonction permettant de générer un Token CSRF

Tout d'abord, dans un nouveau contrôleur, on créer une fonction « getCsrfToken » qui va nous permettre de **générer un Token CSRF**.

Dans cette fonction, nous commençons par démarrer une session si aucune n'est active. Ensuite, nous générons un objet CsrfToken à partir de la **méthode « getToken »** et récupérons sa valeur sous forme de chaîne de caractères.

Enfin, nous **stockons cette valeur** dans la session et la retournons.

```php
#[Route('/api/csrf-token/{tokenId}', name: 'csrf_token', methods: ['GET'])]
public function getCsrfToken(CsrfTokenManagerInterface $csrfTokenManager, string $tokenId, SessionInterface $session): JsonResponse
{
    // Démarrer une Session temporaire si aucune est démarrée
    if (!$session->isStarted()) {
        $session->start();
    }
    // On récupère un objet CsrfToken généré et ensuite retourne une chaine de caractère
    $csrfToken = $csrfTokenManager->getToken($tokenId)->getValue();

    // Stockage du token dans la session temporaire
    $session->set('csrf_token', $csrfToken);

    return new JsonResponse(['csrfToken' => $csrfToken]);
}
```

- Récupération de ce Token sur React.js à partir de l'appel de cette fonction

L'objectif ici étant de générer côté serveur et récupérer un **nouveau CSRF Token** lorsque j'affiche un formulaire. Sur react.js, il existe un Hooks qui permet d'effectuer des actions lorsqu'un composant est **monté** (signifie qu'il est affiché par le navigateur), qui est le Hooks « useEffect ».

Dans l'exemple ci-dessous, je me trouve dans le composant contenant le formulaire d'Inscription. Ainsi, lorsque ce composant est monté, j'envoie une requête HTTP faisant appel à la fonction vue précédemment générant mon CSRF Token. Je spécifie l'Id du token (register).

En réponse à cette requête, je récupère la valeur du CSRF Token en chaîne de caractère que je stocke dans le state CsrfToken.

```javascript
// Demande d'un nouveau Csrf token lorsque le composant est monté
useEffect(() => {
    fetch('api/csrf-token/register')
        .then(response => response.json())
        .then(data => setCsrfToken(data.csrfToken));
}, []);
```

- Envoi de la valeur du CSRF Token lors de la soumission du formulaire d'inscription

Lorsque l'utilisateur a soumis le formulaire d'inscription, il envoie aussi la valeur du CSRF Token dans le Header de la requête HTTP.

```
106     // Fonction pour soumettre le formulaire
107     const handleSubmit = async (e) => {
108         // Empêche le comportement par défaut du formulaire (rechargement de la page)
109         e.preventDefault();
110         try {
111             const response = await fetch('/register', {
112                 method: 'POST',
113                 headers: {
114                     'Content-Type': 'application/json',
115                     // Ajout de la valeur du CSRF
116                     'X-CSRF-TOKEN': csrfToken
117                 },
118                 body: JSON.stringify(formData)
119             });
```

- Vérification du CSRF dans le contrôleur

Après l'envoi du formulaire au Serveur, dans le contrôleur, on fait la vérification du Token CSRF.

Pour cela, on **récupère la valeur du token** à partir du Header de la requête HTTP avec laquelle on va **générer un nouvel objet CsrfToken** que l'on va **comparer** avec celui stocké en Session via la méthode « isTokenValid ».

Facultativement, on peut supprimer l'objet CsrfToken qui est stocké.

```
40          // Récupérer le Csrf Token de la requête HTTP
41          $csrfTokenValue = $request->headers->get('X-CSRF-TOKEN');
42          // Créer un objet CsrfToken
43          $csrfToken = new CsrfToken('register', $csrfTokenValue);
44
45          // Vérifier l'objet CsrfToken avec celui stocké en Session
46          if(!$csrfTokenManager->isTokenValid($csrfToken)){
47              return $this->json([
48                  'message' => 'Accès non authorisé',
49              ], Response::HTTP_UNAUTHORIZED);
50          }
51          // Supprimer l'objet Csrf Token (facultatif)
52          $csrfTokenManager->removeToken($csrfToken);
```

5. Protection contre les attaques par Force brute

➢ Qu'est-ce qu'une attaque par force brute ?

La **CNIL** propose une définition de cette faille :

« Une attaque par force brute (bruteforce attack) consiste à tester, l'une après l'autre, chaque combinaison possible d'un mot de passe ou d'une clé pour un identifiant donné afin se connecter au service ciblé. »

➢ Comment s'en protégeait ?

Pour s'en protéger, il existe plusieurs méthodes :

- REGEX

Très souvent, pour sécuriser ou contrôler un accès, on met en place une authentification par mot de passe qui reste un des moyens les plus simples et le moins couteux à déployer.

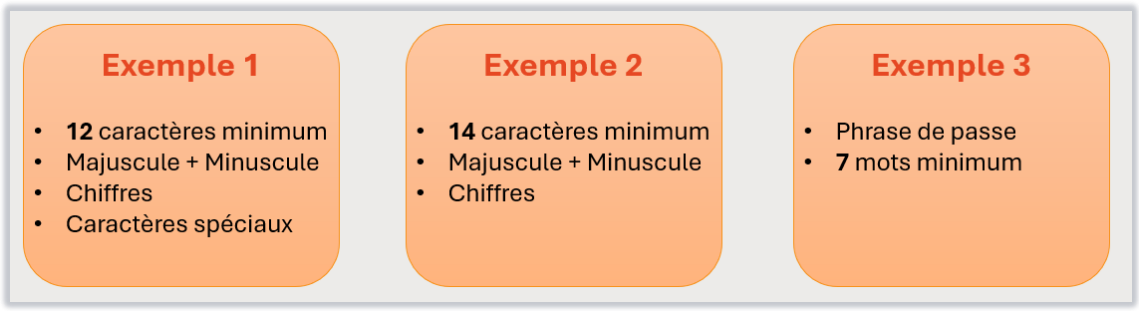
Toutefois, on se trouve souvent face à un niveau de sécurité faible dû à un mauvais choix de mot de passe qui n'est pas assez suffisant pour résister à une attaque par une personne malveillante.

En réponse à cette problématique, la **CNIL** a émis une recommandation en 2017 concernant les mots de passe qui sera par la suite **mis à jour en 2022**.

Cette recommandation est la mise en place d'un **« REGEX »** qui est une séquence de caractères permettant de vérifier le format d'un mot de passe afin de renforcer sa sécurité.

Ainsi pour garantir la robustesse d'un mot de passe, il est essentiel de définir des critères de complexité et de longueur. La CNIL recommande un niveau **d'entropie de 80 bits** pour les mots de passe afin d'assurer leur résistance aux attaques.

La CNIL met donc à disposition **3 exemples de politiques de mot de passe** qui respectent cette entropie, qui sont :



- Rate Limiter

Une autre méthode pour se protégeait contre les attaques de force brute serait de contrôler le nombre de requêtes qu'un utilisateur peut effectuer sur un **service donné** et sur une **période donnée**. Il s'agit donc d'une stratégie de sécurité permettant de **prévenir ces attaques**, qu'on appelle « Rate Limiter ».

Ainsi, cela consistera à limiter le nombre de requête effectué (par exemple, 5 tentatives) sur une période spécifiée (par exemple, 1 minutes), ce qui va réduire la possibilité d'une attaque automatique.

Lorsque cette limitation est dépassée, des mesures de sécurité peuvent être mis en place tel que le blocage du compte ou encore l'ajout de protection supplémentaire tel que l'ajout d'un reCAPTCHA ou encore une double authentification.

➢ Protection dans mon projet

Dans mon projet, j'ai mis en place le **REGEX** à deux reprises : une fois avec React.js et une autre avec Symfony.

Le regex sur React.js est une mesure de sécurité supplémentaire mais elle me permet surtout d'améliorer **l'interface utilisateur (UI)** en guidant l'utilisateur sur le choix de son mot de passe de manière dynamique.

- Regex sur React.js :

```
37
38    // Regex du mot de passe via React
39    const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{12,}$/;
40
```

Ce regex peut être décomposé en plusieurs parties :

1) /^ et **$/** : Ils indiquent **le début et la fin de la chaîne de caractères** pour la vérification
2) (?=.*[a-z]) : Il s'agit d'une **assertion positive** vérifiant qu'il y ait au moins une **lettre minuscule** dans la chaîne de caractère.
3) (?=.*[A-Z]) : Il s'agit d'une assertion positive vérifiant qu'il y ait au moins une **lettre majuscule** dans la chaîne de caractère.
4) (?=.*\d) : Assertion positive vérifiant qu'il y ait au moins un **chiffre** dans la chaîne de caractères.
5) (?=.*[@$!%*?&]) : Assertion positive vérifiant qu'il y ait au moins un **caractère spécial** parmi cette liste.
6) {12,} : Indique que la longueur du mot de passe doit être d'au moins de **12 caractères**.

Lorsqu'on a mis en place le regex du mot de passe et qu'on a un mot de passe à vérifier, on passe à la vérification du format en utilisant la méthode « test » de l'objet RegExp du langage Javascript.

```
56          }
57          // Vérifie le format du mot de passe via la méthode "test"
58          else if(!passwordRegex.test(formData.plainPassword)) {
59              setValid(false);
60          }
```

- Regex sur Symfony

```
64
65      // Regex du mot de passe via Symfony
66      $pattern = "/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{12,}$/";
67
```

Ce regex peut être décomposée de la même manière que sur react.js.

Ce qui va changer est la vérification. En effet, nous utiliserons une fonction native de PHP qui est « preg_match ».

```
68      // Vérifie la correspondance du mot de passe via la fonction "preg_match"
69      $regex = preg_match($pattern, $data['plainPassword']);
```

Après avoir récupéré un booléen spécifiant la correspondance, on bloque l'accès si la correspondance n'est pas valide.

```
71      // Condtion de validation
72      if(!$regex){
73          return new JsonResponse(
74              ['error' => 'Le mot de passe doit contenir au moins 12 caractères, une majuscule, une minuscule, un chiffre et un caractère spécial.'],
75              Response::HTTP_BAD_REQUEST);
76      }
```

Pour ce qui est du Rate Limiter, Symfony propose de le gérer pour l'authentification en l'activant dans le fichier security.yaml.

Dans mon cas, j'ai décidé de personnaliser mon Rate Limiter en ajoutant un écouteur d'événement (EventListener).

L'ajout de ce Rate Limiter personnalisé se fait en **3 étapes** :

- Configuration des différents Rate Limiter

J'ai décidé de configurer 3 Rate Limiter pour différents formulaires : Inscription, Authentification et les autres formulaires.

Par exemple, pour la connexion, j'ai configuré le Limiteur de débit de telles sortes qu'une **IP** puisse effectuer jusqu'à **10 tentatives** sur une durée de **1 minutes**.

```yaml
framework:
    # Configuration de 3 Limiteurs de Débit
    rate_limiter:
        # Limiteur de débit pour la connexion
        login_limiter:
            policy: 'sliding_window'
            # Une IP peut effectuer jusqu'à 10 Tentatives
            limit: 10
            # 10 Tentatives durant 1 minutes
            interval: '1 minute'
            # Données stockés dans le cache
            cache_pool: 'cache.app'
        # Limiteur de débit pour l'inscription
        register_limiter:
            policy: 'sliding_window'
            limit: 30
            interval: '10 minutes'
            cache_pool: 'cache.app'
        # Limiteur de débit pour les formulaires
        form_limiter:
            policy: 'sliding_window'
            limit: 10
            interval: '5 minutes'
            cache_pool: 'cache.app'
```

FIGURE 17 CONFIGURATION DE 3 LIMITEURS DE DEBIT DANS LE FICHIER RATE_LIMITER.YAML

```yaml
    # Création d'un service pour la classe RateLimitListener
    App\EventListener\RateLimitListener:
        arguments:
            $loginRateLimiter: '@limiter.login_limiter'
            $registerRateLimiter: '@limiter.register_limiter'
            $formRateLimiter: '@limiter.form_limiter'
        # Associe le service à un événement
        # Indique que cette classe est un écouteur d'événement et que la
        # méthode onKernelRequest sera appelé à chaque requête HTTP
        tags:
            - { name: kernel.event_listener, event: kernel.request, method: onKernelRequest }
```

FIGURE 18 SERVICE POUR LA CLASSE RATELIMITLISTENER DANS LE FICHIER SERVICES.YAML

- Ajout de la méthode « onKernelRequest » qui se déclenche lorsque le kernel reçoit une requête http.

```php
    // Méthode déclenché lorsque le kernel de Symfony reçoit une requête HTTP
    public function onKernelRequest(RequestEvent $event): void
    {
        // Récupère l'objet Request
        $request = $event->getRequest();
        // Récuèpre le chemin de la requête
        $path = $request->getPathInfo();
        // Récupère la méthode HTTP utilisé
        $method = $request->getMethod();

        // Les Limiteurs sont appliqués uniquement pour les méthodes POST
        if ($method === 'POST') {
            if ($path === '/login' || $path ==='/passwordChangeRequest') {
                // Pour les chemins de Connexion et de changement de mot de passe,
                //       on appelle la fonction de vérification
                $this->checkRateLimit($this->loginRateLimiter, $request->getClientIp());
            } elseif ($path === '/register') {
                // Pour le chemin d'Inscription, on appelle la fonction de vérification
                $this->checkRateLimit($this->registerRateLimiter, $request->getClientIp());
            } elseif (preg_match('/^\/form/', $path)) {
                // Pour les chemins de formulaire, on appelle la fonction de vérification
                $this->checkRateLimit($this->formRateLimiter, $request->getClientIp());
            }
        }
    }
```

- Ajout de la méthode pour la vérification du rate limit sur une requête provenant d’une adresse IP.

```php
    // Méthode pour la vérification du rate limit sur une requête donnée
    private function checkRateLimit(RateLimiterFactory $rateLimiterFactory, string $identifier): void
    {
        // Création d'un instance de limiteur pour l'IP de l'utilisateur
        $limiter = $rateLimiterFactory->create($identifier);
        // Spécifie qu'un jeton du limiteur a été consommé
        $consumption = $limiter->consume(1);

        // Vérifie si l'utilisateur a dépassé le nombre de requête autorisé
        if (!$consumption->isAccepted()) {
            throw new TooManyRequestsHttpException('Too many requests, please try again later.');
        }
    }
}
```

IX. Fonctionnalités principales

Mon application Web est un site de Multiservice qui permet de mettre en relation des entrepreneurs et des particuliers par le biais d'annonces. Ainsi ma fonctionnalité principale est l'annonce qui ensuite résulte à d'autres fonctionnalités comme la Messagerie ou encore la gestion des Rendez-vous.

A. Annonces (Filtrage + Pagination)

Comme nous avons vu précédemment dans la gestion de la base de données, j'ai créé une entité **Avdert** (annonce) qui a une relation avec l'entité **Category** mais aussi avec l'entité **Images_to_advert** (Images supplémentaires). En effet une annonce peut avoir plusieurs catégories et peut stocker jusqu'à 3 images supplémentaires.

Une annonce est caractérisée par un Titre, une description, le **type de service**, les dates de création et d'expiration, d'une image principale, d'un **Slug** et d'autres informations supplémentaires.

Il faut savoir qu'une Annonce peut être de 2 types :

- **Service** s'il a été publié par un Entrepreneur
- **Besoin** s'il a été publié par un particulier

J'ai mis en place un système de statut pour donner la possibilité à l'utilisateur de pouvoir **désactiver** son Annonce et ne pas le rendre visible ou encore **d'activé** l'annonce pour une durée donnée en entrant une **date d'expiration** qui n'est pas obligatoire.

Les 3 statuts d'une annonce :

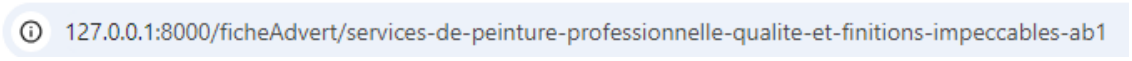
- **Active**
- **Expired**
- **Desactivated**

1. Ajout d'un Slug pour les Annonces

Pour les annonces, j'ai décidé d'ajouter un Slug. Pour cela, j'ai installé via Composer la librairie Crocur/Slugify :

```
composer require cocur/slugify
```

Un slug est la partie de l'url qui va permettre d'identifier une page spécifique tout en étant lisible et significative. Pour mon application, je génère un slug à partir du titre de l'annonce, ce qui permet d'avoir au niveau de l'url, un titre beaucoup plus lisible :




```php
// Création d'une nouvelle instance de la classe Slugify permettant de générer des slugs
$slugify = new Slugify();
// Utilisation de la méthode slugify pour transformer le titre en un slug
$baseSlug = $slugify->slugify($title);
// Initialisation de la variable $slug avec $baseSlug, qui sera utilisée pour vérifier l'unicité.
$slug = $baseSlug;

// Boucle qui s'exécute tant que le slug est déjà existant dans la base de donnée.
while ($entityManager->getRepository(Advert::class)->findOneBy(['slug' => $slug])) {
    // Génération d'un suffixe unique en utilisant random_bytes pour garantir une unicité élevée.
    $uniqueSuffix = substr(bin2hex(random_bytes(2)), 0, 3);
    // Nouveau slug avec l'ajout d'un suffixe unique pour éviter les doublons.
    $slug = $baseSlug . '-' . $uniqueSuffix;
}
```

Puisque mon titre n'est pas unique, j'ai décidé d'ajouter un **suffixe** de 3 caractères à la fin du slug pour éviter d'avoir des **doublons** au niveau de mon Slug. De plus, je génère mon slug en vérifiant dans ma base de données, qu'elle ne soit pas déjà existante via une boucle **While**.

2. Filtrage des Annonces

Lorsque l'utilisateur arrive sur ma page d'annonce, il a la possibilité de pouvoir filtrer les Annonces selon :

- Type d'Annonce : Double Bouton Service ou Besoin
- Catégories : Choix des catégories avec un Multiple Select
- Prix : Choix de mettre un prix minimum ou un prix maximum

L'utilisateur a aussi la possibilité de faire un tri des Annonces selon la date d'activation de l'annonce ou encore selon le prix des annonces.

Il faut savoir que le filtrage est effectué sans rechargement de page à partir de requêtes AJAX via l'API Fetch permettant de faire des requêtes asynchrones.

3. Pagination

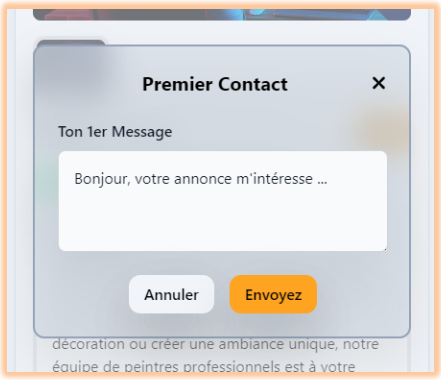
Pour la Pagination de mes Annonces, j'ai décidé de partir sur une pagination dite « Infinite Scrolling » qui consiste à afficher des annonces supplémentaires par lot automatiquement lorsque l'utilisateur atteint le bas de la page.

J'ai fait le choix d'utiliser cette pagination pour améliorer l'expérience utilisateur le rendant plus fluide et évitant à l'utilisateur de devoir cliquer sur des boutons pour pouvoir changer de page.

Comme toute bonne chose, il y a des inconvénients. Malheureusement, l'utilisation de cette pagination rend plus difficile l'accès à des pages précis mais c'est un choix que j'ai décidé de faire puisque je considère qu'il existe beaucoup plus de personne qui vont scroller en bas, que de personne souhaitant revenir sur des annonces précédemment vue.

B. Messagerie

Lorsqu'une annonce est mise en ligne, un utilisateur peut envoyer un **Premier message** à partir de la fiche d'annonce :



S'il s'agit d'une annonce de type Service, seul un utilisateur qui est connecté en tant que particulier peut envoyer un premier message. S'il s'agit d'une annonce de type Besoin, seul un utilisateur qui est connecté en tant qu'entrepreneur peut envoyer un premier message.

Une fois que le Premier message est envoyé, une discussion est ouverte et est accessible dans la catégorie **Message**. L'utilisateur peut voir tous les messages qui sont **actives**. En effet, lorsque le rendez-vous est terminé, la discussion est désactivée et n'est plus affichée dans la Messagerie.

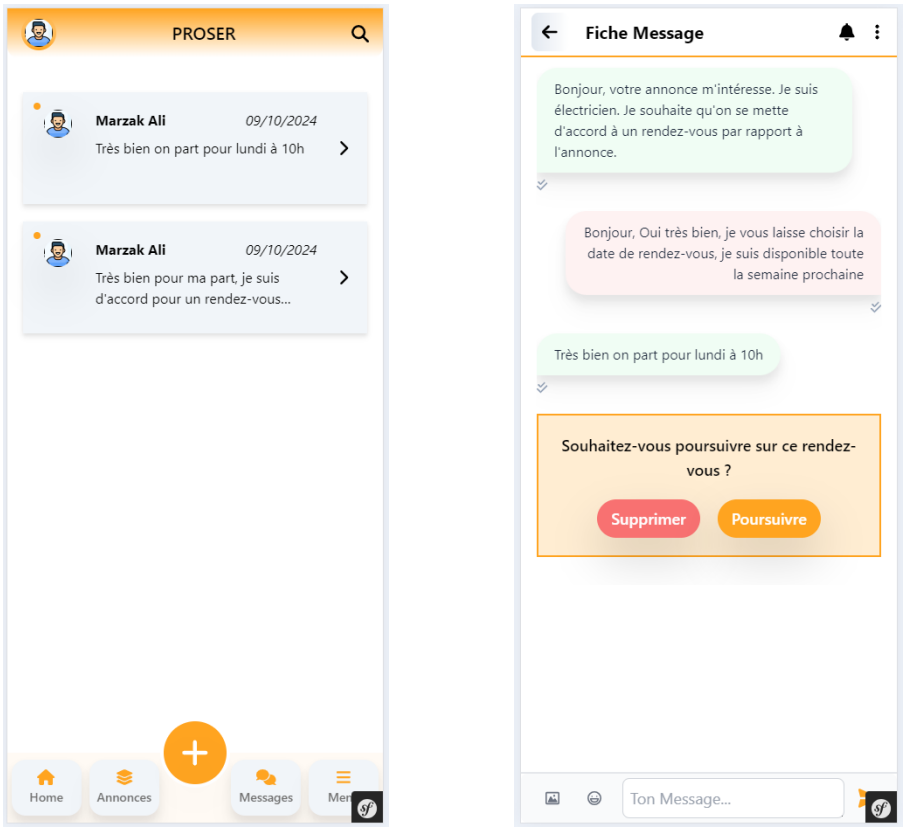
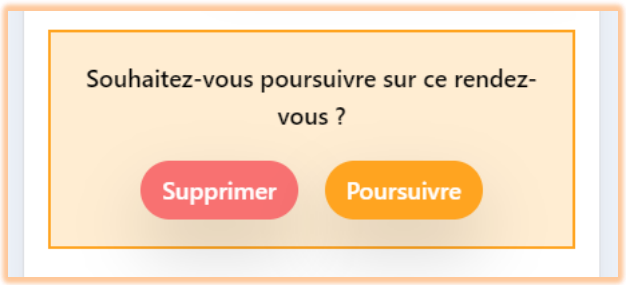


FIGURE 19 LISTE DES DISCUSSIONS & PAGE DISCUSSION

Dans ma base de données, j'ai décidé d'avoir 2 entités **Discussion** et **Message** qui ont une cardinalité en OneToMany. Une discussion peut avoir plusieurs messages. Dans la discussion, je stocke l'id de l'entrepreneur, mais aussi l'id du particulier qui va m'aider dans l'affichage des discussions selon le rôle de la personne lors de sa connexion.

Pour les messages, je stocke l'id de l'expéditeur et du destinataire, le contenu du message et sa date d'envoi. Mais aussi le type de message. En effet, on peut avoir des messages de type **text** qui sont utilisés pour échanger mais aussi des messages de type **appointment** qui sont des messages permettant de confirmer un rendez-vous.

Lorsque les 2 parties (Entrepreneur et Client) se mettent d'accord, l'entrepreneur va créer un rendez-vous via un formulaire, ce qui va envoyer un message de confirmation de type **appointment** dans la discussion permettant au client de **compléter son rendez-vous** en passant au paiement. L'Entrepreneur et le client ont la possibilité de **supprimer ce rendez-vous** si les dates ne correspondent pas. Voici le message de confirmation :



Il faut savoir que la messagerie n'est pas en temps réel à l'heure actuelle et qu'il est nécessaire de sortir de la discussion pour pouvoir afficher les nouveaux messages.

C. Gestion des Rendez-vous

1. Création du Rendez-vous

Comme nous l'avons dit, l'Entrepreneur s'occupe de créer un rendez-vous à partir d'un formulaire de création de rendez-vous présent dans le canal de Discussion :

Lorsque l'utilisateur entre une date de rendez-vous, j'envoie une requête AJAX pour récupérer les rendez-vous de ce jour permettant ainsi à l'entrepreneur de pouvoir choisir une date sans avoir de conflits avec d'autres rendez-vous.

De plus, il ne peut pas choisir une date précédent la date actuelle et l'heure de fin doit être après l'heure de début.

Une fois que le rendez-vous est créé, il est ajouté dans la base de données et j'ai mis en place un système de statut pour avoir un aperçu de l'état du rendez-vous.

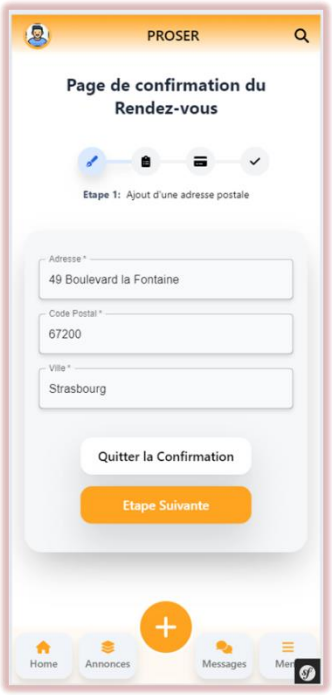
Ainsi un rendez-vous peut avoir **6 états** :

- **AwaitingCompletion** : Il s'agit du statut par défaut lorsque le rendez-vous est créé. En effet, le rendez-vous est en attente d'être complété par le client.
- **Completed** : Le rendez-vous est donc complété par le client et il est à venir prochainement. Il est placé dans le planning de l'Entrepreneur.
- **InProgress** : Le rendez-vous est en cours. Le rendez-vous possède ce statut lorsque l'heure actuelle est entre le début et la fin de la date de rendez-vous.
- **Finished** : Le rendez-vous est considéré terminé lorsque ce dernier a été validé par les 2 parties par le biais d'un code qu'on verra par la suite.
- **Rescheduled** : Si la date du rendez-vous a été décalé par l'entrepreneur, cela permet aux deux parties d'avoir un indice sur le fait que le rendez-vous a été décalé. Il est considéré comme **Completed**.
- **Cancelled** : lorsque le rendez-vous est annulé par l'entrepreneur.

2. Compléter le rendez-vous

Lorsque le rendez-vous est créé, le client doit compléter ce rendez-vous. Pour cela, il y a 4 étapes :

- **Etape 1 :** Ajout d'une adresse postale

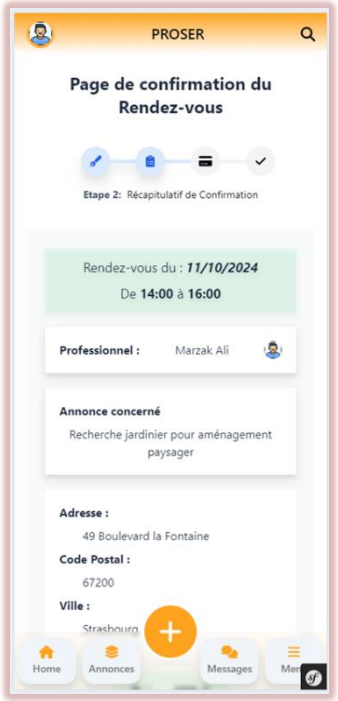


Cette étape consiste à entrer une adresse postale dans laquelle l'entrepreneur doit aller pour le rendez-vous.

Si le client a déjà mis ces coordonnées dans ses paramètres d'utilisateur, les champs seront automatiquement complétés avec cette adresse.

Ainsi, le client peut garder cette adresse ou modifier s'il doit en mettre une nouvelle.

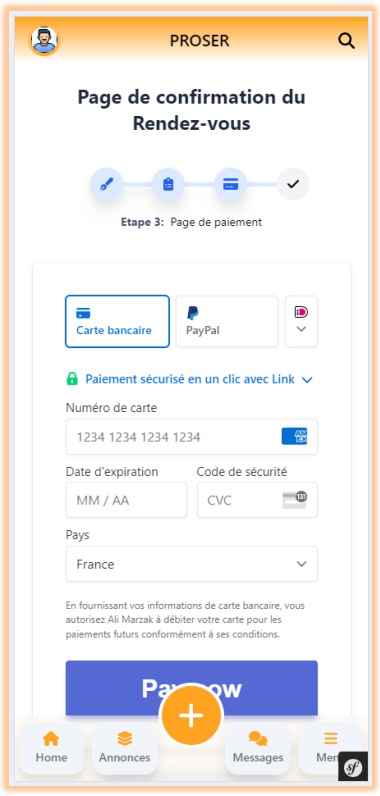
- **Etape 2 :** Vérification concernant le rendez-vous



Cette partie est un récapitulatif du rendez-vous qui va permettre au client d'être sûr des informations avant de pouvoir passer au passement.

Il trouvera la date du rendez-vous, l'annonce qui est concerné, le prix et l'adresse qu'il a entré pour le rendez-vous. Il a aussi la possibilité de revenir en arrière pour modifier son adresse.

- Etape 3 : Paiement du rendez-vous

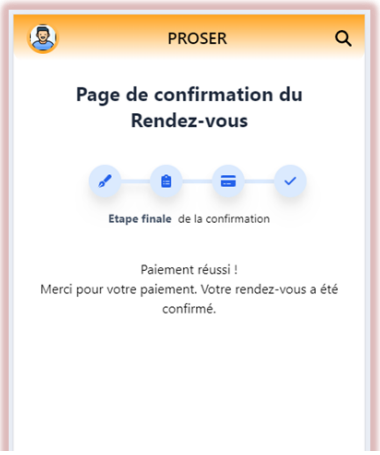


Pour le paiement, j'ai décidé d'utiliser le service **Stripe** qui va permettre aux clients de payer vers l'organisme ProSer.

Cette page va nous afficher le **formulaire** fourni par Stripe que j'ai intégré via **React**.

Il s'agit d'un composant récupéré qui permet d'intégrer le paiement directement dans notre application.

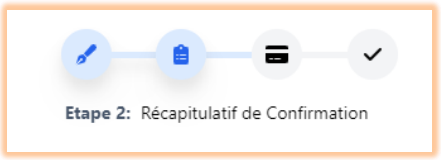
- Etape Finale : Page de confirmation



Enfin, lorsque le paiement a été validé, le client arrive sur cette page qui lui confirme que le rendez-vous a été bien complété.

Suite à cela le rendez-vous passe au statut **Completed** et il est donc affiché dans le planning de l'Entrepreneur.

Enfin pour améliorer l'interface utilisateur de cette page, j'ai décidé d'utiliser un **Stepper** qui permet de guider l'utilisateur et ainsi de l'encourager à continuer vers la fin du processus.



3. Affichage du rendez-vous

Pour voir les rendez-vous, j'ai mis en place une page « Mes Rendez-vous ». Pour un entrepreneur, cette page affichera un **planning** que j'ai codé manuellement et pour un client, cette page affichera la **liste des rendez-vous** selon les statuts.

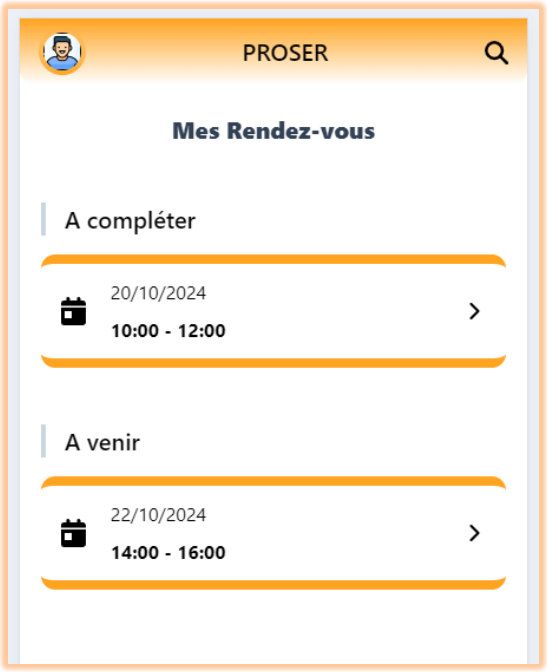


FIGURE 20 VERSION CLIENT

Il s'agit de l'affichage des rendez-vous pour les clients.

Tout d'abord, j'affiche les rendez-vous **En Cours**.

Ensuite les rendez-vous qui sont **à compléter** par le client.

Puis les rendez-vous qui sont **complétés ou reprogrammés**.

Enfin, on affiche les rendez-vous qui sont annulés par l'entrepreneur.

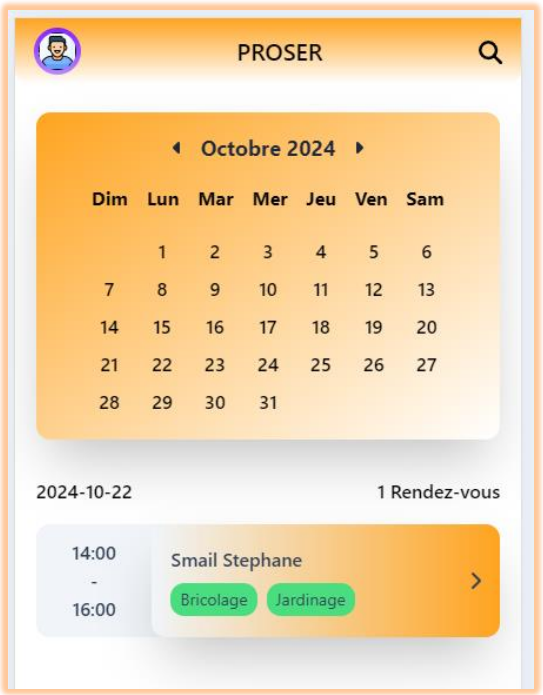


FIGURE 21 VERSION ENTREPRENEUR

Pour les entrepreneurs, j'ai décidé mettre en place un planning qui va afficher seulement les rendez-vous complétés et reprogrammés.

Le fonctionnement du planning est simple, j'envoie une requête AJAX contenant le mois et l'année ce qui va me permettre de récupérer un tableau des jours.

Et lorsque je clique sur un jour en particulier, j'envoie une requête pour récupérer les rendez-vous de ce jour.

4. Validation du Rendez-vous

Lorsque le jour du rendez-vous est arrivé et que le rendez-vous s'est bien passé, il est primordial de mettre en place un système de validation pour pouvoir valider le rendez-vous des 2 côtés. Pour cela, j'ai mis une vérification avec un code à 6 digits.

Dans ma base de données, j'ai donc ajouté une entité **Validation_code** qui a une cardinalité en **ManyToOne** avec l'entité **Appointment** qui stocke les rendez-vous.

Ainsi, je donne la possibilité au client de générer un code à partir de la fiche du rendez-vous qui sera stocké dans cette entité avec une date. Le client peut remplacer ce code à l'infini en appuyant sur le bouton Générer un code.

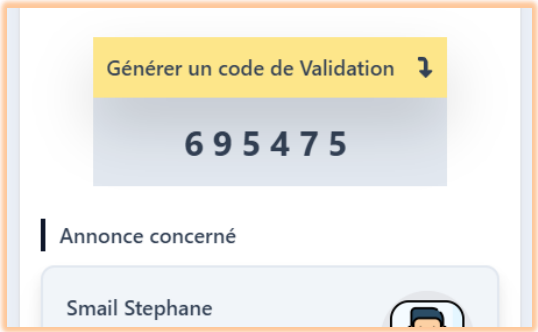


FIGURE 22 GENERATION DE CODE DE VALIDATION PAR LE CLIENT

Ainsi, une fois que le code est généré par le client, l'entrepreneur peut saisir ce code et donc validé le rendez-vous et le faire passer au statut **Finished**. Bien évidemment avant de valider, une vérification sera effectuée côté serveur pour vérifier que le rendez-vous est bien au statut **InProgress**.

FIGURE 23 SAISIS DU CODE DE VALIDATION PAR L'ENTREPRENEUR

D. Gestion de Paiement

Pour la partie paiement lors de la confirmation du rendez-vous par le client, j'ai décidé d'utiliser **Stripe**, qui est une plateforme de paiement en ligne qui permet aux organismes de traiter des paiements sur internet.

Les **choix** qui ont fait que j'ai utilisé Stripe sont :

- Elle offre la possibilité de payer selon différentes **méthodes de paiement**.
- Elle est facile à intégrer dans un projet. En effet, propose une intégration pour différentes technologies. Dans mon cas, Stripe propose une **intégration avec React.js**.
- Stripe offre une **bonne gestion des erreurs** dont je peux récupérer les messages facilement.
- Enfin, Stripe propose ses services avec des **mesures de sécurité avancées** et très robustes.

Il faut savoir que Stripe propose 3 Services :

- **Payment Links** : ce service permet de créer des liens de redirections de paiement très facilement que l'on peut partager avec les clients
- **Checkout** : ce service fournit une page de paiement prête à l'emploi et non personnalisable.
- **Eléments** : ce service fournit une collection de composants UI permettant de créer des formulaires de paiements personnalisables dans notre application.

Dans mon cas, j'ai décidé de choisir le service **Eléments** qui me permet d'intégrer le paiement selon l'interface utilisateur que je souhaitais mettre en place.

Pour pouvoir établir un paiement avec Stripe, nous devons créer une Instance de paiement via **l'API Stripe**.

Pour effectuer cela, lorsque le composant affichant la page de paiement est monté, j'effectue une requête Ajax vers mon contrôleur qui gère la création de cette instance.

```
16  // Envoi de la requête lors du montage du composant
17  useEffect(() => {
18    // Requête Ajax pour la création de l'instance de paiement
19    fetch("/create-payment-intent", {
20        method: "POST",
21        headers: { "Content-Type": "application/json" },
22        body: JSON.stringify({ numberToken: numberToken }),
23      })
24      .then((res) => res.json())
25      // Récupération de l'id pour le côté client
26      .then((data) => setClientSecret(data.clientSecret));
27  }, []);
```

FIGURE 24 HOOKS USEFFECT QUI ENVOIE UNE REQUETE AJAX LORSQUE LE COMPOSANT EST MONTE

Ainsi de ce contrôleur, j'attends qu'il me renvoie la clé secrète du client, qui est une clé unique associée à un **PaymentIntent donnée**, obtenue via l'API Stripe.

Dans ce contrôleur, j'utilise plusieurs services utilisant des méthodes fournis par Stripe que j'ai mis en place qui vont me permettre de créer cette Instance de paiement.

Ainsi, dans mon fichier StripeService.php, j'ai 3 fonctions :

- **SearchCustomer** qui va me permettre de récupérer un **Customer** s'il est déjà existant, c'est-à-dire qu'un de mes utilisateurs a déjà effectuer un paiement.

```php
    // Appelle l'API pour rechercher un Customer
    public function searchCustomer(string $email)
    {
        $result = $this->stripe->customers->search([
            'query' => 'email:\'' . $email . '\'',
        ]);

        return $result->data;
    }
```

- **CreateCustomer** qui va me permettre de créer un Customer à partir d'une donnée unique comme un email et de son nom.

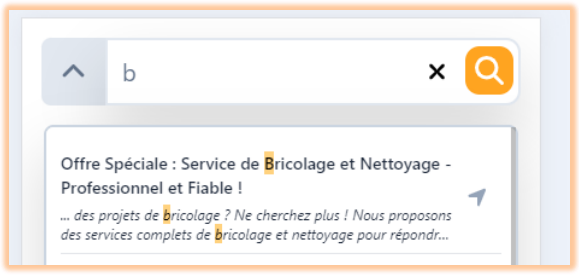
```php
    // Appelle l'API pour créer un Customer
    public function createCustomer(string $email, string $name)
    {
        return $this->stripe->customers->create([
            'name' => $name,
            'email' => $email,
        ]);
    }
```

- **CreatePaymentIntent** qui va permettre de créer cette Instance de paiement grâce aux informations qu'on fournit comme le Montant, la devise, le **Customer** et d'autres éléments personnalisables.

```php
    // Crée un nouvel Intent de Paiement via l'API Stripe
    public function createPaymentIntent(int $amount, string $email, string $id_client)
    {
        return $this->stripe->paymentIntents->create([
            'amount' => $amount * 100,
            'currency' => 'eur',
            'receipt_email' => $email,
            'customer' => $id_client,
            'description' => "Paiement d'une annonce PROSER",
            'setup_future_usage' => 'off_session',
            'automatic_payment_methods' => [
                'enabled' => true,
            ],
        ]);
    }
```

Ainsi, dans mon contrôleur, je m'occupe tout d'abord de récupérer les informations nécessaires sur le rendez-vous comme le montant, ou encore les informations sur le client. Ce qui va me permettre d'utiliser le Service de recherche de Customer. Si je ne trouve pas de Customer lié à ce client, je m'occupe d'en créer un avec le service de Création de Customer. Une fois que j'ai l'id du Customer, j'utilise le Service de Création d'Instance de paiement qui va me permettre de récupérer la clé secrète du client que j'envoie à mon composant React permettant d'afficher le formulaire pour ce paiement donné.

E. Barre de Recherche



Une autre fonctionnalité que j'ai mis en place est la Barre de Recherche. En effet, dans un contexte où j'utilise des annonces, il est très important de mettre en place une barre de recherche facilitant les utilisateurs dans leurs recherches.

Pour cela, j'ai décidé de mettre en place dans mon application une recherche textuelle avancée.

- **Indexage Full Texte**

Pour commencer, il fallait mettre en place l'Indexage Full Texte qui va nous permettre d'améliorer les performances de recherche.

Cela consiste à ajouter un index spécial sur les colonnes de la base de données auquel on souhaite effectuer nos recherches. Pour ma part, les recherches s'effectueront sur le titre et la description des Annonces.

```
17    // Index fulltext sur les colonnes 'title_ad' et 'description'
18    #[ORM\Index(name: 'advert', columns: ['title_ad', 'description'], flags: ['fulltext'])]
```

- Ajout de l'extension **MATCH AGAINST**

Pour effectuer la recherche sur notre base de données, nous devons passer par des requêtes SQL et donc pour faire une recherche de texte intégral, on utilise la commande Match Against.

Toutefois, dans mon projet nous sommes sur Symfony et nous utilisons le langage DQL qui ne présente pas cette commande. J'ai donc dû utiliser une extension qu'on trouve sur GitHub de manière OpenSource.

Ainsi j'ai pu l'utiliser dans mes requête DQL pour effectuer les recherches souhaitées :

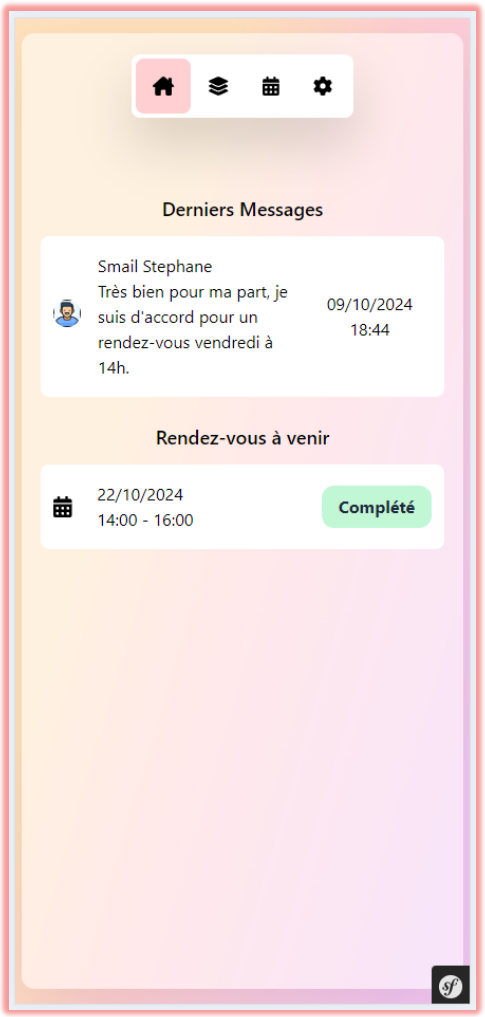
```
20    // Fonction contenant une requête DQL pour les recherches d'annonces
21    public function suggestionsSearch(string $searchInput): array
22    {
23        $status = "active";
24        $query = $this->createQueryBuilder('a');
25        $query->where("a.status = :status");
26        $query->setParameter('status', $status);
27        if($searchInput){
28            $query->andWhere('MATCH_AGAINST(a.titleAd, a.description) AGAINST (:searchInput boolean)>0');
29            $query->setParameter('searchInput', '*'. $searchInput .'*');
30        }
31
32        return $query->getQuery()->getResult();
33    }
```

F. Dashboard Entrepreneur / Client

Pour que les Entrepreneurs ou les Clients puissent **Afficher** ou **Modifier** ou **Supprimer** leurs annonces ou rendez-vous, il est primordial que les utilisateurs puissent avoir un Dashboard correspondant à leur rôle (Entrepreneur ou Client).

Le Dashboard est constitué de **3 Pages** :

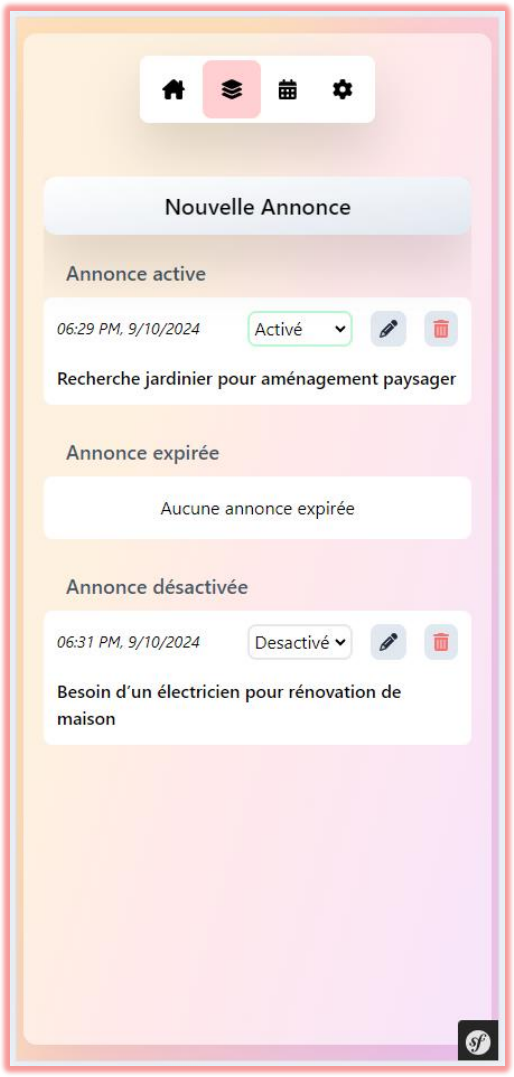
- Page Home



Cette page contient les **5 derniers messages** reçues par l'Entrepreneur ou le Client.

De plus, il liste les Rendez-vous à venir qui contient les rendez-vous **Completed**, **Rescheduled** ou **InProgress**.

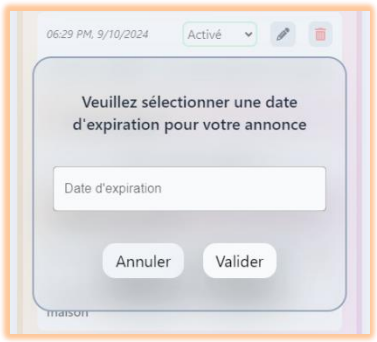
- Page Annonce



Cette page permet d'ouvrir un formulaire pour **publier** une nouvelle Annonce.

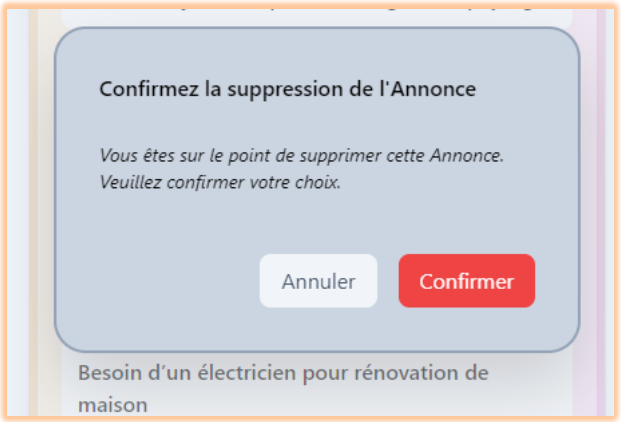
Elle affiche les annonces qui sont de statut **active**, **expired** et **deactivated**.

L'utilisateur peut désactiver une annonce ou l'activer en spécifiant une date d'expiration ou non :



L'utilisateur peut aussi **modifier** le contenu de son annonce.

Enfin il peut **supprimer** une annonce et une confirmation sera demandé via une modale.



- Page Rendez-vous

Pour les rendez-vous, seul **l'entrepreneur** peut **modifier** ou **supprimer** le rendez-vous. Le client peut seulement **voir** les rendez-vous.

Dans cette page, j'affiche les rendez-vous qui ont un des statuts : **InProgress**, **Completed** & **Rescheduled** et **AwaitingCompletion**.

Pour la **modification**, l'entrepreneur peut seulement changer la date de début et de fin du rendez-vous.

Et tout comme pour les annonces, une confirmation sera demandée pour la **suppression** d'un rendez-vous.

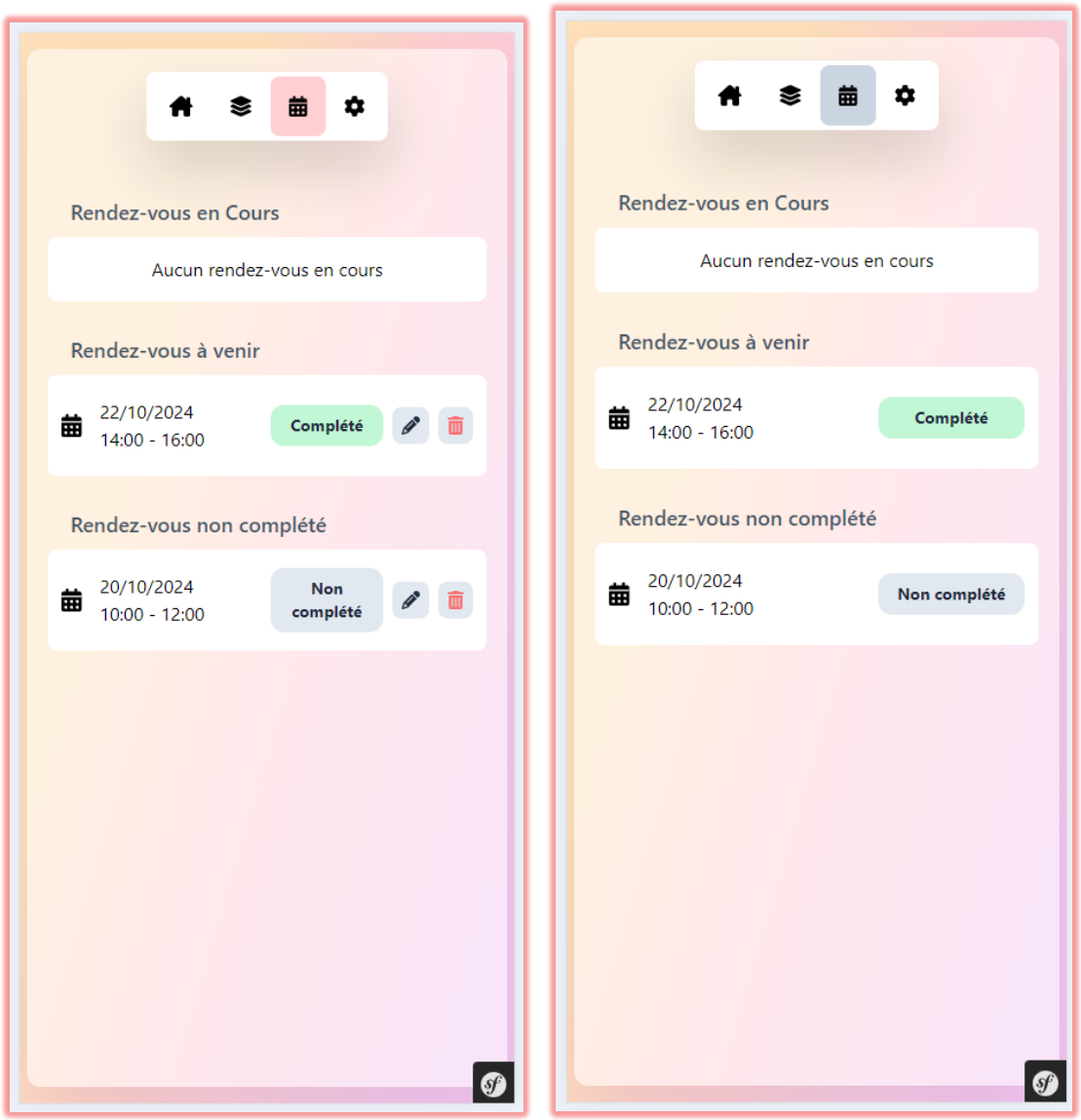


FIGURE 25 VUE DE LA PAGE RENDEZ-VOUS DU DASHBOARD ENTREPRENEUR / CLIENT

X. Conformité RGPD

Lorsqu'on navigue dans une application Web, nous sommes souvent amenés à entrer des données nous concernant qui seront stockés ou non dans leur base de données.

Cela a donné lieu à de nombreuses dérives en matière de protection de la vie privée et de gestion des données.

Pour faire face à ces enjeux, la **CNIL** (Commission Nationale de l'Informatique et des Libertés) a mis en place le **RGPD** (Règlement Général sur la Protection des Données) dont l'objectif est de « responsabiliser les organismes publics et privés qui traitent leurs données.

Nous allons donc voir les différentes dispositions du RGPD.

A. Collecte et Gestion des Données Personnelles

1. Type de données collecté et minimisation

Toute d'abord, lorsqu'on développe une application, il est primordial de savoir quel type de données, nous avons besoin. En effet, lorsque nous faisons un traitement de données, nous devons avoir une finalité claire c'est-à-dire que nous ne « pouvons pas collecter ou traiter des **données personnelles** simplement au cas où cela vous serait utile un jour » (selon la CNIL).

Nous avons parlé de données personnelles mais de quoi s'agit-il ?

La CNIL répond à cette question en nous donnant une définition précise d'une **donnée à caractère personnel** :

C'est toute information relative à une personne physique susceptible d'être identifiée, directement ou indirectement.

Dans mon projet, j'ai respecté ce qui est recommandé par le RGPD en ayant une approche de **minimisation des données**, c'est-à-dire que je demande à l'utilisateur seulement les données dont le traitement a une finalité.

A titre d'exemple, j'ai décidé pour l'inscription que l'utilisateur n'avait à entrer seulement **l'Email**, la **Civilité**, le **Nom** et **Prénom** comme donnée à caractère personnel.

Si le client est amené à devoir prendre un rendez-vous, il serait dans l'obligation d'entrer son **Numéro de téléphone** et son **Adresse postale** pour pouvoir confirmer son rendez-vous sinon il ne pourra pas passer au rendez-vous et donc ne pas confirmer son rendez-vous.

2. Mise en place de consentement

Lorsqu'on utilise des données à caractère personnel, il est primordial d'avoir le consentement de l'utilisateur qui est exigé par le RGPD.

Le RGPD nous donne une définition claire du consentement :

Le consentement est défini comme « toute manifestation de volonté, libre, spécifique, éclairée et univoque par laquelle la personne concernée accepte, par une déclaration ou par un acte positif clair, que des données à caractère personnel la concernant fassent l'objet d'un traitement ».

Ainsi, pour qu'un consentement soit correctement validé, il faut qu'il respecte 4 critères :

- Consentement **libre** : l'utilisateur ne doit être ni influencé ni forcé et ne doit pas subir de conséquences s'il décide de ne pas être consentant.
- Consentement **spécifique** : lorsqu'un utilisateur est consentant, c'est seulement pour un seul traitement qui a une finalité déterminée.
- Consentement **éclairé** : avant que l'utilisateur ne soit consentant ou non, l'utilisateur doit pouvoir avoir des informations tel que : l'identité du responsable du traitement, les catégories de données collectés, les finalités poursuivies et l'existence d'un droit de retrait du consentement.
- Consentement **univoque** : il ne doit pas avoir d'ambiguïté dans le recueil du consentement.

3. Accessibilité de la politique de Confidentialité

Dans une application web, il essentielle de respecter 3 règles sur la politique de Confidentialité :

- Claire et Compréhensible : elle doit être écrit avec un langage simple de telle sorte que toute utilisateur soit dans la capacité de comprendre facilement comment leurs données seront utilisées.
- Visible : elle doit être accessible à partir de n'importe quel page de l'application. Dans mon application, celui-ci est disponible au menu de ma NavBar dans la version mobile et au pied de page dans la version Desktop.
- Détaillée : la politique de confidentialité doit donner des détails sur les types de données collectées, la finalité de chaque traitement de donnée et les droits des utilisateurs.

Voici quelques visuelles de mon application qui montre l'accès à la politique de confidentialité selon le type d'écran (version mobile et version Tablette/Desktop) :

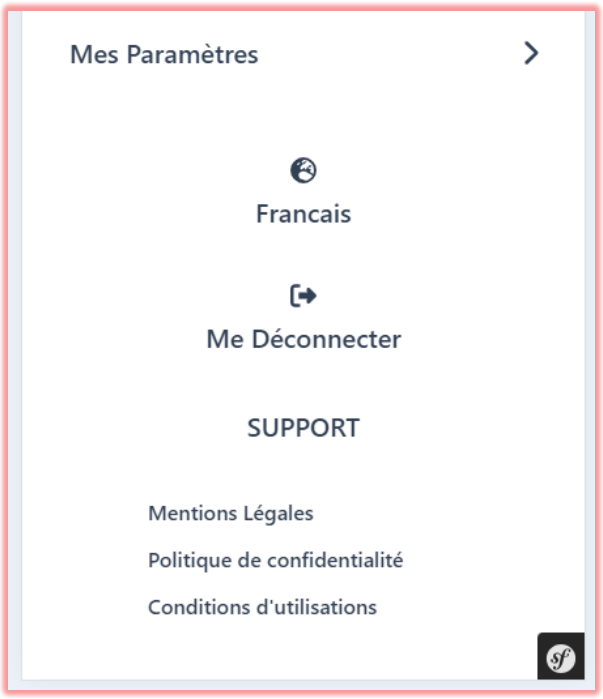


FIGURE 26 POLITIQUE DE CONFIDENTIALITE AU MENU DE LA VERSION MOBILE

FIGURE 27 POLITIQUE DE CONFIDENTIALITE AU FOOTER DE LA VERSION DESKTOP

B. Droits des Utilisateurs

1. Droits d'accès et de modification des données personnels

Lorsqu'on utilise des données personnelles d'utilisateurs, le RGPD préconise un **Droit de rectification** à l'utilisateur de ces données. Ainsi, un utilisateur peut accéder à ses données et pouvoir les modifier ou les effacer. La CNIL nous donne une définition claire de ce qu'est le Droit de rectification :

Toute personne peut faire rectifier, compléter, actualiser, verrouiller ou effacer des informations la concernant lorsqu'ont été décelées des erreurs, des inexactitudes ou la présence de données dont la collecte, l'utilisation, la communication ou la conservation est interdite.

Pour ma part, je permets à chaque utilisateur de pouvoir accéder à ses données mais aussi de pouvoir les modifier dans la rubrique **Paramètres/Informations personnelle**.

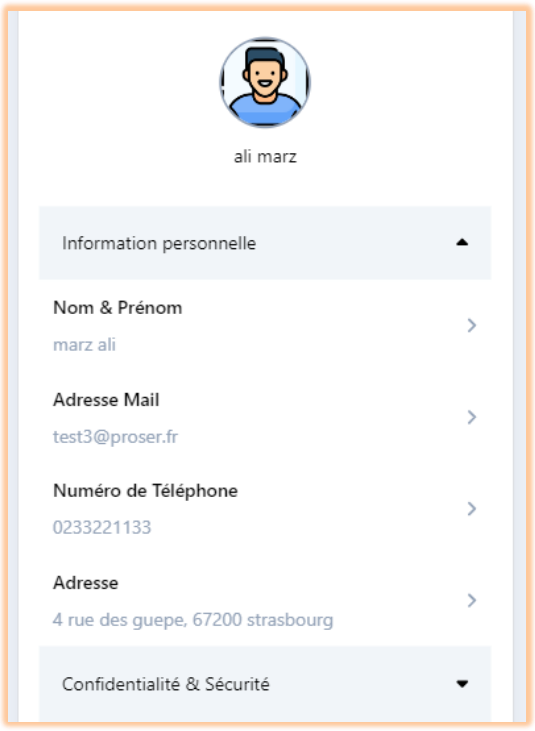


FIGURE 28 INFORMATION PERSONNELLE DE L'UTILISATEUR

2. Droit à l'oubli

Un autre droit de l'utilisateur qui doit être respecté selon la CNIL est le Droit à l'oubli. Un utilisateur peut tout à fait demander à qu'il n'y ait plus aucune de ses données personnelles stockées dans la base de données.

Dans les paramètres de mon application, l'utilisateur peut avoir la possibilité de **Désactiver son compte pendant 30 jours** ou encore **Supprimer définitivement** son compte avec une confirmation via email.

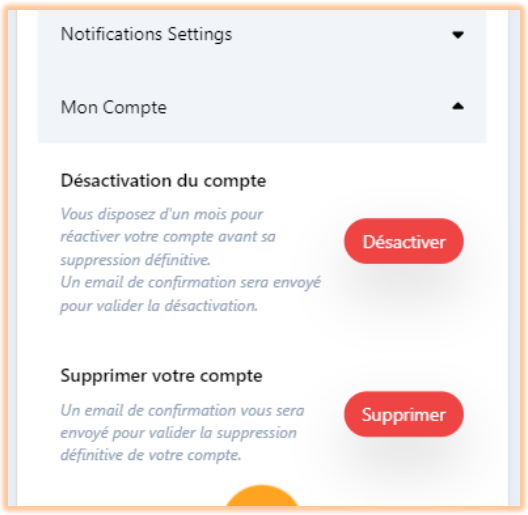


FIGURE 29 AFFICHAGE DE LA RUBRIQUE MON COMPTE DANS MES PARAMETRES

Pour gérer cela côté Serveur, j'ai ajouté une entité **AccountManagement** qui va me permettre de gérer la confirmation via Email mais aussi de garder une trace des dernières actions effectués. Au niveau de la base de données, cette entité aura une relation ManyToOne avec l'entité User. Ci-dessous vous trouverez le MCD et le MLD qui représente ces éléments :

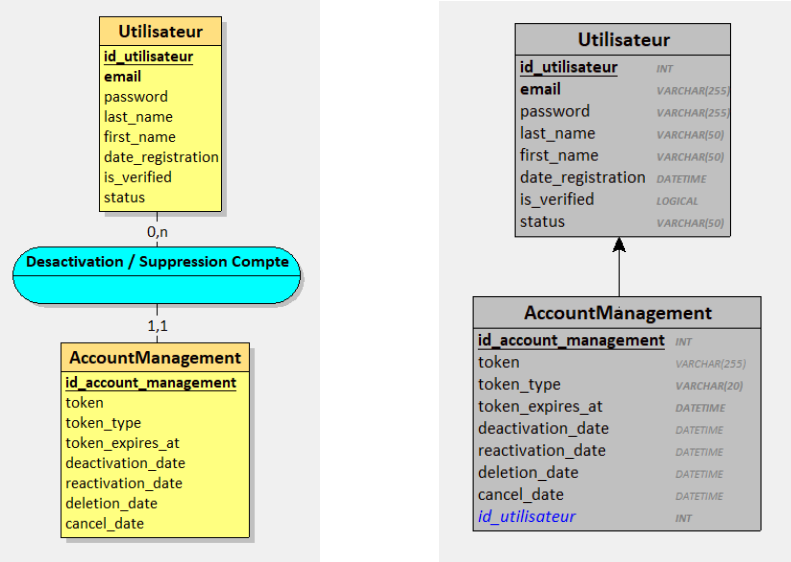


FIGURE 30 MCD ET MLD DE LA GESTION DE DESACTIVATION ET DE SUPPRESSION DE COMPTE

Bien évidemment pour pouvoir désactiver et supprimer son compte, il ne faut pas que l'utilisateur possède des rendez-vous à venir et donc il doit attendre que ses rendez-vous soit finit ou que l'Entrepreneur annule le rendez-vous.

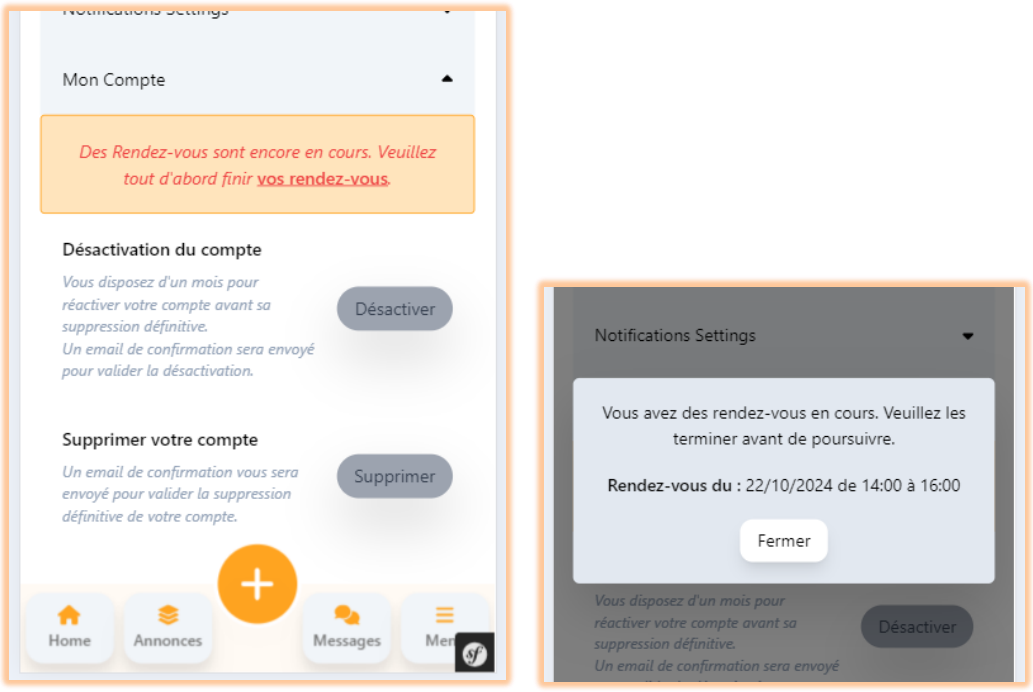


FIGURE 31 BLOCAGE DE LA DESACTIVATION ET DE LA SUPPRESSION DE COMPTE

Lorsque l'utilisateur désactive ou supprime définitivement son compte, une nouvelle instance de la classe **AccountManagement** sera créer qui contiendra :

- **Token** généré grâce au standard **UUID** version 4 qui permet de générer des identifiants aléatoires les rendant pratiquement impossibles à dupliquer et donc uniques. Ce token va permettre de repérer cette instance de la classe lorsqu'on annule ou confirme via l'email.
- **Date d'expiration** : Si l'email de confirmation n'a pas été confirmé avant 1 Heure après l'envoi de l'email, la désactivation n'est plus possible.
- **Type** de token qui correspond au type d'action. Il y a 2 actions soient « deactivate » pour la désactivation ou « delete » pour la suppression définitive.

➢ Désactivation de compte

Pour le cas de désactivation de compte, l'utilisateur aura le statut **pendingDeactivate** qui va nous permettre de faire la vérification lors de la confirmation, évitant ainsi une désactivation sans que la demande ait été faites par l'utilisateur.

Une fois que l'email aura été envoyé via **EmailVerifier**, l'utilisateur aura 2 choix :

- Annuler : Le statut de l'utilisateur repasse en état **active**. Une date d'annulation sera ajoutée à cette instance de classe pour avoir un historique de la manipulation. Et le Token sera mis à **null** pour m'assurer l'unicité des Tokens.
- Confirmer : Le statut de l'utilisateur passe en état **deactivated**. Une date de désactivation sera ajoutée à cette instance de classe pour avoir un historique de la manipulation. Tous ces annonces passent au statut **deactivated**.

Ainsi, lorsque la désactivation est effectuée, l'utilisateur a 1 mois pour se reconnecter et donc réactivé son compte. Sinon, si ce délai de 1 mois est dépassé alors le compte est supprimé définitivement et donc le statut de l'utilisateur passe en **deleted**.

Lors de la réactivation de son compte, une date de réactivation est ajoutée à l'instance concerné de la classe **AccountManagement** et le statut de l'utilisateur passe en **active**.

➢ Suppression de compte

Pour le cas de suppression de compte, l'utilisateur aura le statut **pendingDelete** qui nous permet de faire la vérification lors de la confirmation comme pour la désactivation du compte.

L'utilisateur aura à faire le même choix lors de la réception de l'email, soit d'annuler ou de confirmer la suppression de compte.

Si l'utilisateur confirme la suppression, tous ces annonces seront supprimées et l'utilisateur sera **anonymisé.** Je m'occupe d'hacher son email avec **l'algorithme de hachage sha256**, je remplace son nom et prénom par des valeurs prédéfinis et je mets à **null** les autres données personnelles.

De plus, je supprime toutes les annonces le concernant. Etant donné que les rendez-vous sont supprimés après avoir été terminé, les données personnelles stockés n'ont pas besoin d'être supprimé.

Après avoir confirmé la suppression définitive du compte, l'utilisateur ne pourra plus se connecter avec ce compte.

C. Sécurité des données personnels

Une des préoccupations du RGPD est la sécurité des données personnelles. En effet, chaque organisme doit pouvoir s'assurer que les données personnelles d'un utilisateur soient gérées et stockées de manière sécurisé. Pour cela, nous devons mettre en place des mesures nécessaires.

Tout d'abord, nous devons assurer le chiffrement des données comme le hachage du mot de passe avec un algorithme de hachage fort. Ce chiffrement garantit que même en cas de fuite de données, les informations sensibles restent inaccessibles.

Ensuite, il est impératif de mettre en place un contrôle d'accès rigoureux pour restreindre les droits des utilisateurs en fonction de leurs rôles.

De plus, nous devons mettre en place une protection pour l'ensemble des failles de sécurité (Injections SQL, Injections XSS, faille CSRF et Attaque par force brute) qui sont énoncés par des organismes comme l'OWASP. Cette mesure de sécurité nous évite des fuites de données personnelles.

Enfin, on doit assurer une maintenance régulière de notre application pour offrir à nos utilisateurs une version à jour pour bénéficier des dernières améliorations en matière de sécurité.

En adoptant ces mesures, les organismes peuvent non seulement protéger les données personnelles, mais aussi instaurer un climat de confiance avec leurs utilisateurs et donc garantir une conformité avec le RGPD.

XI. Conclusion et Perspectives d'améliorations

Pour conclure, il est très intéressant d'utiliser Symfony et React. Chacune de ses bibliothèques offre des éléments intéressants. Symfony avec son architecture robuste et ses composants modulaires permet de construire un application Back-End très performantes et sécurisées. De son côté, React facilite énormément la création d'interfaces utilisateur dynamiques et réactives, ce qui a apportait à mon application une meilleure expérience utilisateur.

Toutefois l'utilisation de ces 2 dans un même projet reste très délicat et présente des défis puisque j'ai fait énormément de choses manuellement tel que la partie Sécurité qui a demandé des configurations spécifiques.

Au final, c'était une aventure enrichissante pour ma part.

Pour ce qui est des perspectives d'améliorations, il serait intéressant de mettre en place un système de facture grâce au bundles DomPDF qui permet de créer une facture lorsque le client passe au paiement ou encore que l'entrepreneur reçoit son paiement.

Je souhaiterais aussi mettre en place le paiement via Stripe lorsque le rendez-vous a été terminé et validé par les 2 cotés.

Aussi, ajouter un système de notification avec un serveur mercure pour recevoir les notifications en temps-réel et enfin ajouter une double authentification pour assurer une meilleure sécurité.

XII. Remerciements

Je souhaite tout d'abord remercier les formateurs (Stephane SMAIL, Mickael MURMANN et Quentin MATHIEU) qui ont été là durant ma formation pour me guider et m'aider lorsque j'en avais besoin.

Je souhaite aussi remercier l'ensemble des formateurs et formatrices présents à ELAN Formation. Je pense notamment à Sophie qui s'est occupée de nous pour nous aider à préparer l'entretien final, mais aussi de présenter notre projet.

XIII. ANNEXES

A. Documentation Technique

Ci-dessous, vous trouvez le MCD et le MLD

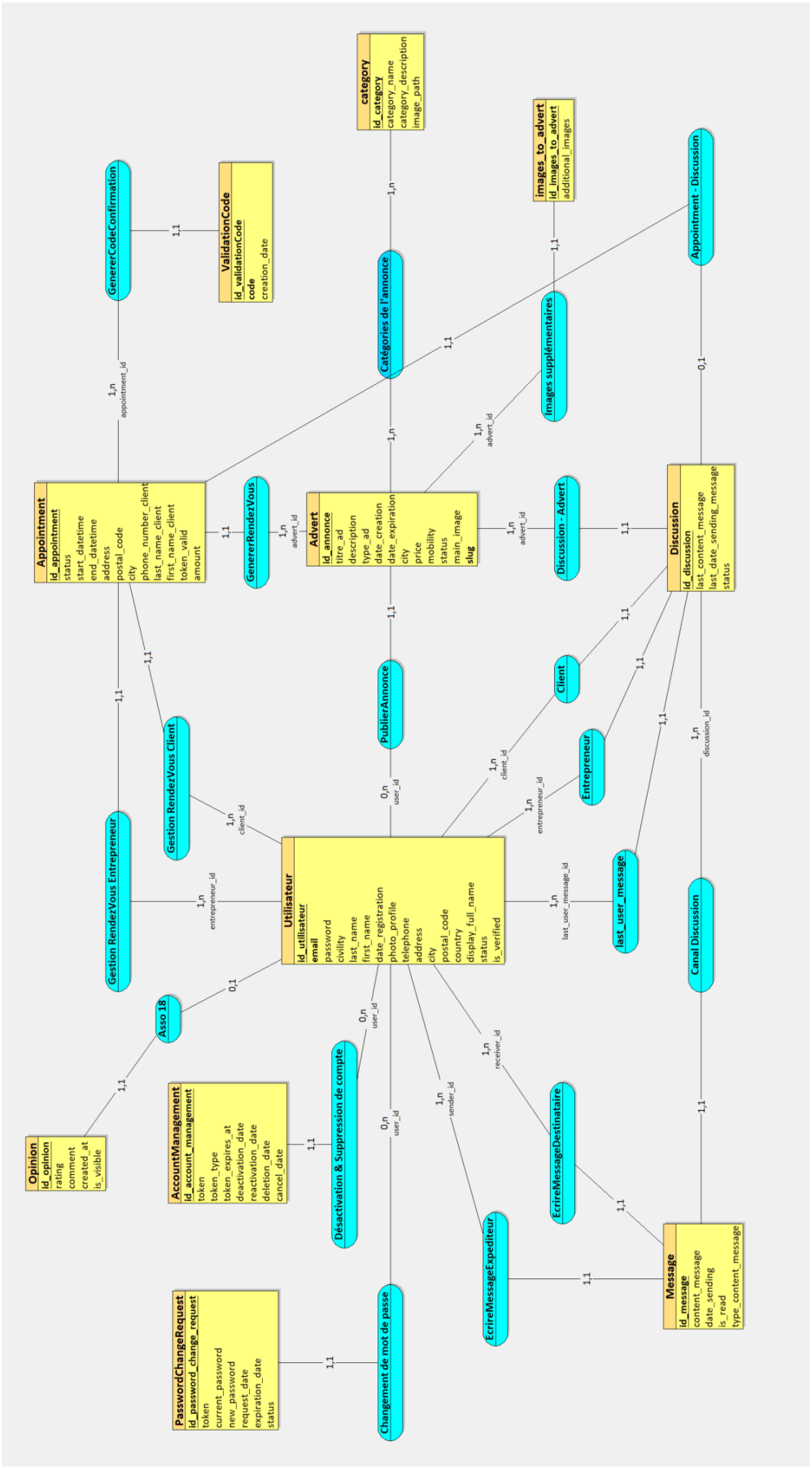


FIGURE 32 MODELE CONCEPTUEL DE DONNEE

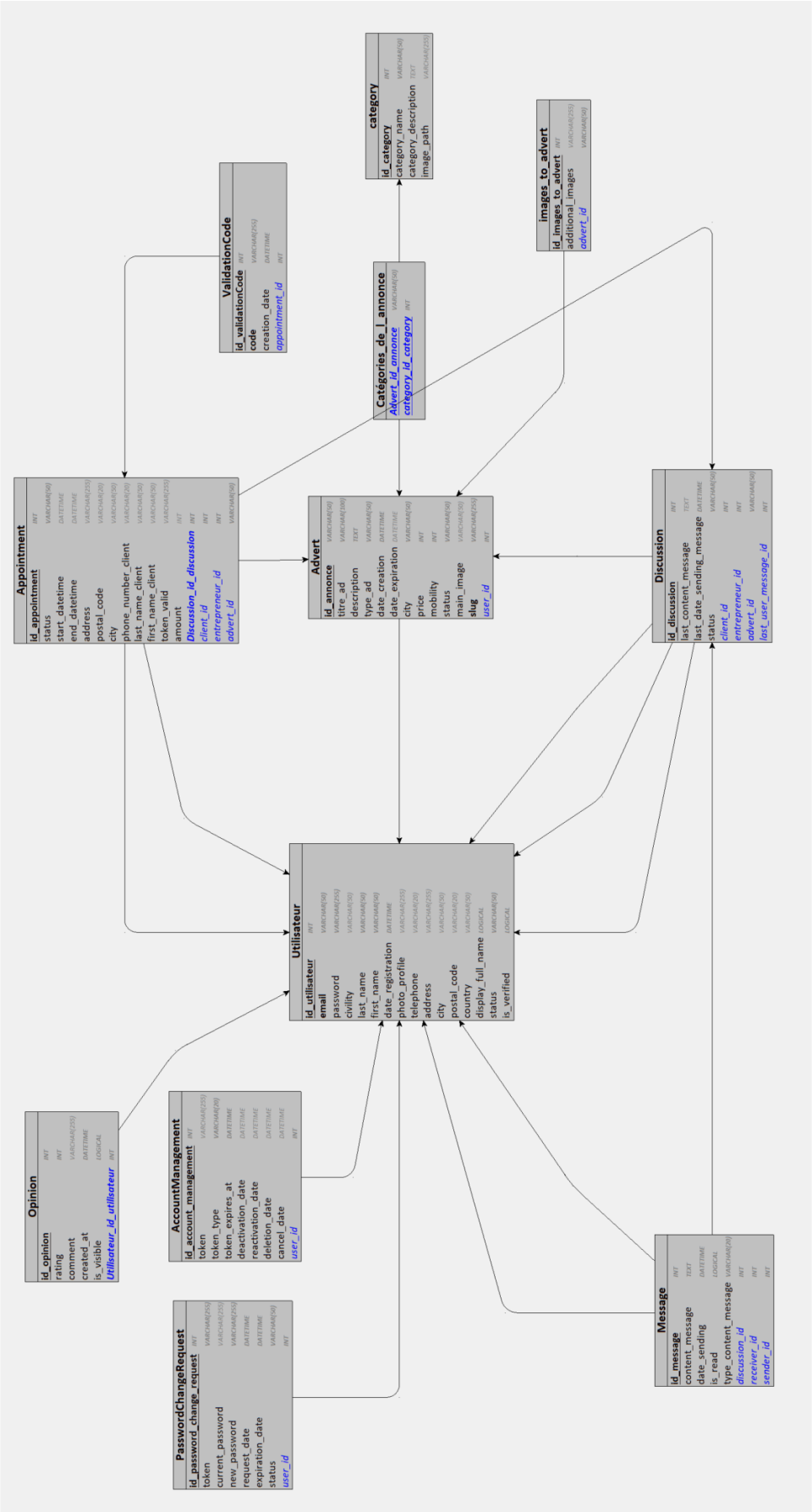


FIGURE 33 MODELE LOGIQUE DE DONNEE

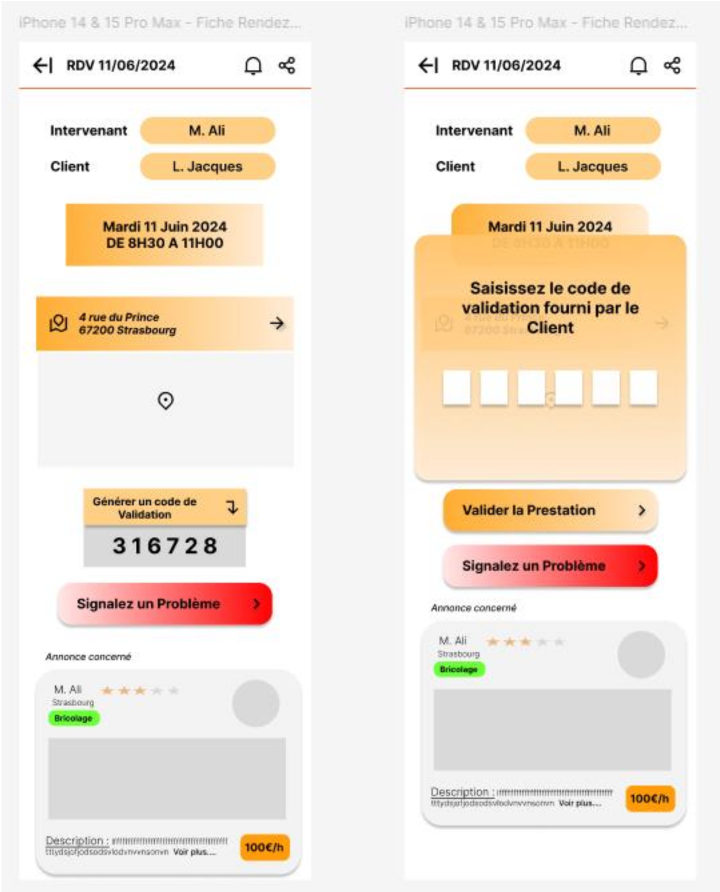


FIGURE 34 MOCKUP DE LA FICHE DU RENDEZ-VOUS

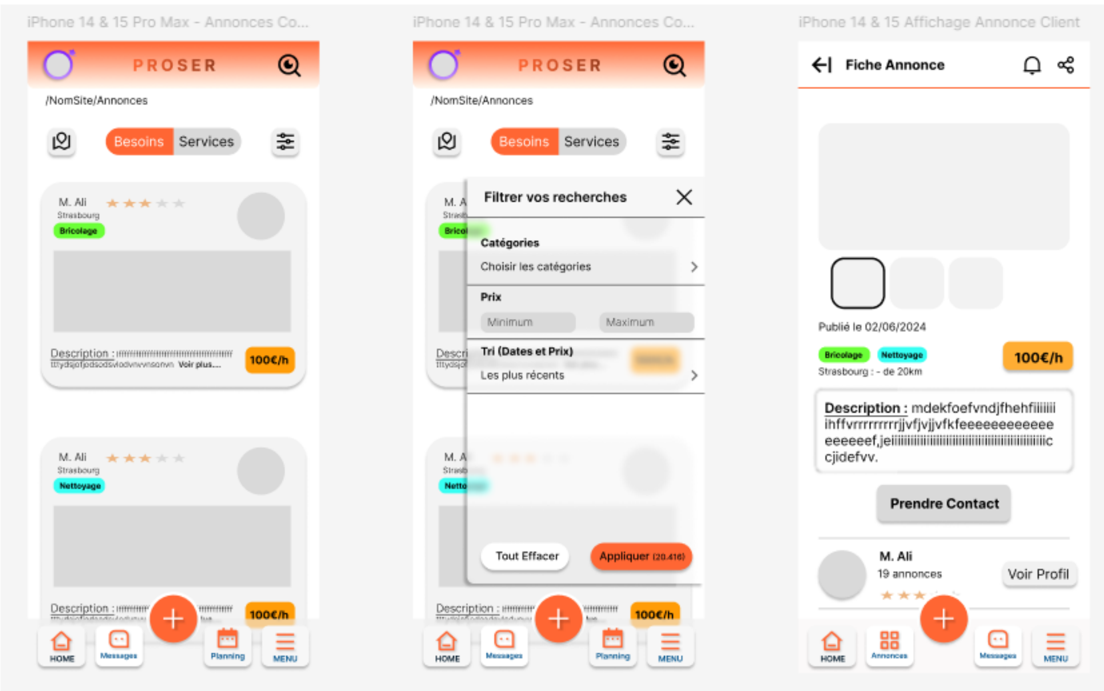


FIGURE 35 MOCKUP DE LA LISTE DES ANNONCES + FILTRAGE + FICHE DE L'ANNONCE

B. Références et Ressources

Références

Figure1 : https://blog.sensiolabs.com/fr/2023/11/29/lessentiel-sur-symfony-7-avec-nicolas-grekas/

Figure2 : https://laconsole.dev/formations/symfony/bdd-entites-relations/

Figure3 : https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2

Figure4 : https://www.linkedin.com/pulse/prevent-csrf-attacks-ayemun-hossain

Ressources utilisées

https://kinsta.com/fr/blog/meilleures-pratiques-react/

https://dev.to/sathishskdev/part-1-naming-conventions-the-foundation-of-clean-code-51ng

https://www.w3.org/

https://www.cnil.fr/fr

https://owasp.org/

https://cyber.gouv.fr/

https://accessibilite.numerique.gouv.fr/

https://symfony.com/doc/current/index.html

https://tailwindcss.com/