

---

# DOSSIER PROJET

---

Titre Professionnel Développeur Web et Web Mobile



Guillaume  
BURGNIES

Février 2022

# Table des matières

<b>Partie A : Introduction du dossier projet ..</b>	<b>5</b>
<b>1. Introduction.....</b>	<b>5</b>
1.1 Parcours personnel .....	5
1.2 Compétences couvertes par le dossier .....	6
<b>2. Présentation succincte des projets.....</b>	<b>7</b>
2.1 Projet en entreprise BS Next.....	7
2.2 Projet personnel PostYours.....	7
<b>Partie B : Projet en entreprise BS Next.....</b>	<b>7</b>
<b>1. Introduction .....</b>	<b>7</b>
1.1 Présentation de l'entreprise BS Next.....	7
1.2 Résumé du projet en entreprise .....	8
1.3 Technologies utilisées (BS Next) .....	8
<b>2. Quelques outils .....</b>	<b>9</b>
2.1 Git : Outil de gestion de version.....	9
2.2 Autres outils utilisés .....	10
<b>3. Cahier des charges .....</b>	<b>11</b>
3.1 Contexte .....	11
3.2 Spécifications .....	11
<b>4. Maquettage .....</b>	<b>12</b>
4.1 Diagramme.....	12
4.2 Maquettage du site.....	12
<b>5. Réalisation de la page d'accueil .....</b>	<b>15</b>
5.1 Route et composants .....	15
5.2 Adaptation aux écrans (responsive) .....	18
5.3 Site multilingue avec i18Next.....	19

<b>6. Réalisation des autres pages</b>	<b>20</b>
6.1 Page formation, développement, mentions légales	20
6.2 Formik et Yup pour la validation des formulaires	21
6.3 Connexion et inscription	22
6.4 Avis laissés par les utilisateurs	23
6.5 Les routes protégées	27
6.6 Sécurité	28
<b>7. Tests unitaires</b>	<b>29</b>
<b>8. Avenir du projet</b>	<b>29</b>
<b>Partie C : Projet personnel</b>	<b>30</b>
<b>1. Introduction</b>	<b>30</b>
1.1 Résumé du projet personnel	30
1.2 Technologies utilisées (BS Next)	30
<b>2. Front-end et généralités</b>	<b>31</b>
<b>3. Base de données</b>	<b>32</b>
3.1 Connexion de la base de données à Symfony	32
3.2 Création de la base de données	32
3.3 Création des entités et des tables	32
<b>4. Back-end de l'application</b>	<b>34</b>
4.1 Création de l'inscription et la connexion	34
a) L'inscription	34
b) La connexion	37
4.2 Création de la page d'accueil	38
4.3 Création de la page personnelle (utilisateurs)	40
4.4 Création, modification et suppression d'un « post » (image)	40
4.5 Création de la page pour modifier le mot de passe	41

<b>5. Filtre, Back office Sécurité et déploiement .....</b>	<b>42</b>
5.1 Création d'un filtre .....	42
5.2 Création de la partie administrateur .....	44
5.3 Sécurité .....	44
5.3 Mise en production .....	45
<b>6. Jeu d'essai .....</b>	<b>46</b>
6.1 Tests unitaires .....	46
6.2 Test utilisateurs .....	47
 <b>Partie D : Veille technologiques .....</b>	 <b>49</b>
1. Les failles de sécurité les plus courantes .....	49
2. La sécurité et la validation des mots de passe .....	51
3. Recherche à partir d'un site anglophone .....	52
 <b>Conclusion .....</b>	 <b>54</b>
<b>Remerciements .....</b>	<b>55</b>
 <b>Annexes .....</b>	 <b>56</b>
<b>Annexe 1 : Projet en entreprise .....</b>	<b>57</b>
1.1 Le maquettage .....	57
1.2 Le questionnaire .....	59
 <b>Annexe 2 : Projet personnel .....</b>	 <b>62</b>
2.1 Page d'accueil (Code Twig) .....	62
2.2 Page « show » affichage d'une image (Code Twig) .....	63
2.3 Page compte utilisateur (Code Twig) .....	64
2.4 Contrôleur image .....	65
2.5 Formulaire .....	67

# Partie A : Introduction du dossier projet

## 1. Introduction

### 1.1 Parcours personnel

Après l'obtention de mon BAC S, je me suis dirigé vers l'université pour y faire une licence de Physique-Chimie puis par la suite j'ai changé et je me suis dirigé en licence de Physique.

Durant mon parcours, j'ai côtoyé l'informatique et plus particulièrement la programmation notamment les langages C et C++ qui sont beaucoup utilisés pour la modélisation de phénomènes physiques par exemple.

Cette partie m'a toujours beaucoup intéressé et je n'avais pas une idée très précise du métier que je voulais faire par la suite.

Lors de ma dernière année en Physique, j'ai réalisé un stage dans le domaine des fibres optiques avec une présentation oral en fin de stage. Cette présentation devait être faite à l'aide d'un site web comme support, et c'est ainsi que j'ai pu découvrir le langage HTML, CSS et la conception d'un site web.

Cette expérience m'a convaincu de l'intérêt que j'avais pour le Développement Web et d'en faire mon métier.

C'est ainsi que j'ai découvert la formation diplômante de développeur web et web mobile, où j'ai pu découvrir et apprendre le métier de développeur web via l'intégration HTML/CSS, la programmation avec PHP et le concept de Programmation Orientée Objet, la manipulation du DOM avec Javascript, la gestion d'une base de données, les bonnes pratiques web, ainsi que gérer un déploiement et réaliser les documents propres à un projet.

Dans le cadre de la fin de la formation et en vue du Titre Professionnel Développeur Web & Web Mobile, j'ai réalisé un stage de deux mois en entreprise (distanciel), au cours duquel j'ai réalisé un projet pour l'entreprise et un projet personnel, tout cela dans le but de la certification finale.

Au cours de ce dossier, je vais donc présenter ces deux projets ; le site web de l'entreprise que j'ai réalisé et l'application web qui est un projet personnel.

## 1.2 Compétences couvertes par le dossier

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

- Maquetter une appliaction (CP1)
- Réaliser une interface utilisateur web statique et adaptable (CP2)
- Réaliser une interface utilisateur web dynamique (CP3)

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

- Créer une base de données (CP5)
- Développer les composants d'accès aux données (CP6)
- Développer la partie back-end d'une application web ou web mobile (CP7)

**Compétences transversales :**

- Utiliser l'Anglais dans son activité professionnelle en développement web et web mobile
- Actualiser et partager ses compétences en développement web et web mobile

## 2. Présentation succincte des projets

### 2.1 Projet en entreprise BS Next

Le premier projet consiste à réaliser « un site vitrine » pour l'entreprise BS Next qui permettra de faire connaître les services que celle-ci propose, de la contacter, de laisser un avis etc... L'application sera développée avec React pour la partie front et avec Symfony pour la partie back-end.

Il y aura aussi la possibilité d'afficher le site en Français ou en Anglais.

### 2.2 Projet personnel PostYours

Le second projet consiste à réaliser une application qui a pour but principal de permettre à un utilisateur de partager des photos/images mais aussi de laisser des commentaires sur les photos d'autres utilisateurs. Cette application est développée entièrement avec le Framework Symfony et une base de données MySQL.

## Partie B : Projet en entreprise BS Next

### 1. Introduction

#### 1.1 Présentation de l'entreprise BS Next

BSNext est une Auto-Entreprise fondée en juin 2020, localisée dans les Hauts de France. Le fondateur, ingénieur de formation, propose les mêmes services qu'une entreprise de service numérique (ESN).

L'entreprise est divisée en 3 pôles :

1. Applications
2. Formations
3. Conseils

La clientèle concerne les ESN et les écoles du numérique. Le chiffre d'affaires annuel de l'entreprise est de 20 000 euros. L'entreprise est en phase d'expansion.

## 1.2 Résumé du projet en entreprise BS Next

Le projet consiste à réaliser « un site vitrine » pour l'entreprise qui permettra de faire connaître les services que celle-ci propose, de la contacter, de laisser un avis etc...

L'application sera développée avec React pour la partie front-end et avec Symfony pour la partie back-end.

Cela étant fait grâce à l'aide du webpack « Encore » qui permet d'utiliser React pour la vue dans Symfony et de non pas utilisé Twig comme cela peut être fait classiquement avec ce Framework PHP. Il y aura aussi la possibilité d'afficher le site en Français ou en Anglais

L'application permettra à un utilisateur, s'il le souhaite, de s'inscrire et se connecter. Il pourra ainsi par exemple laisser un avis, tout cela sera géré par Firebase : le service d'authentification proposé (géré par OAuth) et la « RealTime Database » (NoSQL).

## 1.3 Technologies utilisées (BS Next)

**React JS** : La partie front-end a été réalisée avec React, Framework développé par Facebook utilisant Javascript et un DOM virtuel bien plus rapide que le DOM original, la possibilité de réutiliser les composants de React permet de gagner du temps, de plus la librairie est open-source ce qui facilite la recherche d'informations et de résolution de problème.

React permet de faire ce que l'on appelle une SPA (Single Page Application), j'ai pu utiliser React Router afin d'avoir des routes virtuelles : seuls les composants présents sur la route sont placés dans le DOM virtuel, le résultat est direct. L'utilisateur n'a pas l'impression d'avoir subi un chargement de page.

**CSS 3 et Bootstrap** : Tous deux pour « l'habillage » du site et l'adaptation à la taille de l'écran.

**Formik et Yup** : Utilisés pour la validation des formulaires côté client.

**Symfony** : Pour la partie back-end du site.

**Firebase** : Pour la base de données (RealTime DataBase). Firebase est un ensemble de services d'hébergement pour n'importe quel type d'application (Android, iOS, Javascript, Node.js, Java, Unity, PHP, C++ ...). Il propose d'héberger en NoSQL et en temps réel des bases de données, du contenu, de l'authentification sociale (Google, Facebook, Twitter et Github), et des notifications, ou encore des

services, tel que par exemple un serveur de communication temps réel. Il appartient aujourd'hui à la maison mère de Google : Alfabé



## 2. Quelques outils

### 2.1 Git : Outil de gestion des versions

Le « versioning » pour le site vitrine a été réalisé à l'aide de l'outil Git, l'outil open-source le plus utilisé permettant de gérer différents projets en les envoyant sur un serveur. Ce dernier est connecté à l'ordinateur d'autres développeurs qui envoient leur code et récupèrent le vôtre. Toute personne qui travaille sur un projet est connectée avec les autres, tout est synchronisé. Quant à Github, il s'agit d'un logiciel, ou plateforme, son rôle est d'héberger ces différents projets qui utilisent Git. Concrètement, ils proposent une interface graphique qui en simplifie grandement l'utilisation.

Pourquoi utiliser Git ?

- L'aspect backup régulier du code, le « versioning », est essentiel dans le développement web. Une panne de disque dur pourrait être fatale.
- L'aspect collaboratif qu'offre Git devient rapidement essentiel lorsqu'il y a plus d'un développeur dans le projet.
- Il permet de garder une trace des versions antérieures du site. Très utile en cas de débogage.

Des règles précises ont été appliquées pour la création de branches et les « commits » par exemple.

Pour le nom des branches par exemple, le format suivant a été suivi : type / name / issue\_ID.

Le type permet d'identifier le but de la branche :

- **Feature** : Ajout d'une nouvelle fonctionnalité
- **bugfix** : Correction d'un bug
- **hotfix** : Correction d'un bug critique
- **chore** : Nettoyage du code
- **experiment** : Expérimentation de fonctionnalités

Le nom de la branche décrit succinctement le but de celle-ci. Certaines règles doivent être respectées :

- Le nom doit faire moins de 50 caractères
- Le nom doit respecter la convention kebab-case (les mots doivent être en minuscule et liés par des tirets "-")
- Enfin « issue\_ID » sert pour le suivi des tickets, que ce soit sur [Github](#), [Gitlab](#) ou tout autre outil comme [JIRA](#) par exemple, il peut être utile d'ajouter le numéro de ceux-ci dans le nom des branches. Il est ainsi possible par exemple de fermer automatiquement les tickets lorsque la branche sera « mergée ».

## 2.2 Autres outils utilisés

### **Communication :**

- Slack, utilisé tout au long du projet et où nous étions connectés en permanence facilitant nos échanges.
- Google Meet pour les visioconférences plusieurs fois par semaine.

### **Documentation :**

- Google Drive/Docs, utilisation de Docs pour la prise de note (cahier des charges annexes...).
- Utilisation de Drive pour le partage de fichiers comme les logos de l'entreprise pour le site.

### **Gestion de projet :**

- Jira pour le suivi et la création de tickets.

### **Développement :**

- VS Code, éditeur de code open-source. Très complet, utile tout au long de ma formation et customisable à l'aide de ses nombreuses extensions gratuites communautaires pour augmenter sa productivité

## 3. Cahier des charges

### 3.1 Contexte

L'entreprise BS Next a besoin d'un site vitrine afin de se faire connaître et attirer de nouveaux clients et expliquer ce que l'entreprise propose.

Nous avons utilisé la méthode agile pour la réalisation du projet, le stage s'est déroulé entièrement en distanciel et j'ai travaillé seul sur le projet mis à part une collaboration avec un Webdesigner.

### 3.2 Spécifications

Le but du site est de détailler les services proposés, proposer un catalogue de formation, un espace de témoignages de la part des clients.

Le site doit être utilisable par tous types d'appareil et doit donc s'adapter à la taille de l'écran.

- Il doit y avoir la page d'accueil qui contiendra la plupart des informations sur l'entreprise.
- Une page pour chaque service proposé : formation, développement, gestion de projet, dashboarding.
- Une page de contact ainsi que les mentions légales.
- La possibilité de se connecter pour les utilisateurs afin de laisser un avis.
- Un back office pour gérer et valider les avis notamment.
- Un code couleur pour le site : blanc, noir et doré.
- Le site doit être développé avec React JS et Symfony.

Echéances : Le projet a débuté le 21 novembre 2021 et le développement de celui-ci doit se terminer le 04 février 2022.

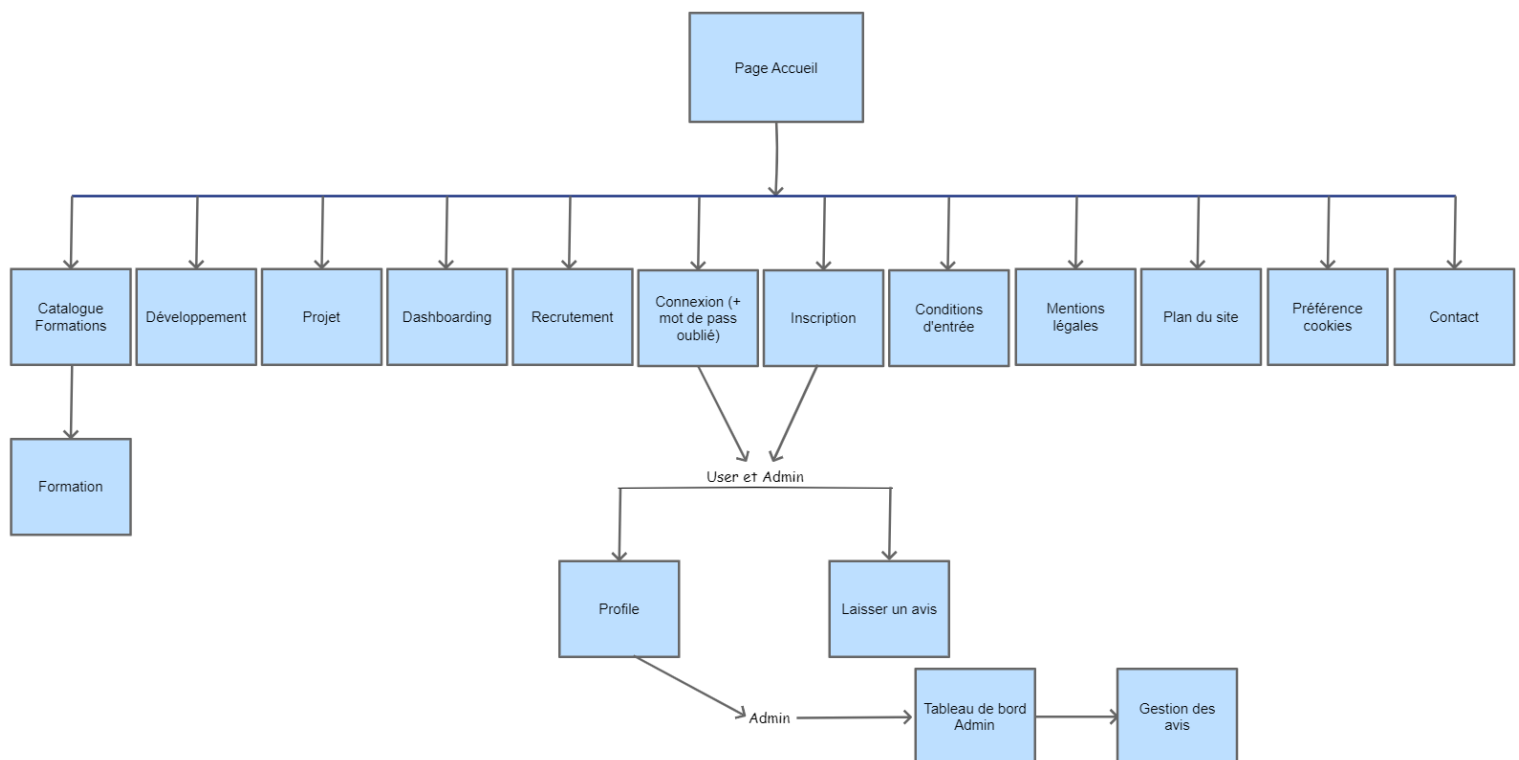
## 4. Maquettage

### 4.1 Diagramme

En se basant sur le cahier des charges, il a d'abord fallu réaliser le diagramme (arborescence) de l'application.

Ceci a été fait grâce au logiciel « Pencil » (également utilisé pour le maquettage).

Ce schéma représente les différentes pages, liées entre elles et hiérarchisées par niveaux de profondeur (voir ci-dessous).



### 4.2 Maquettage du site

Après réalisation de l'arborescence du site, il a été possible de commencer à penser à la conception même du site à l'aide d'une maquette. La maquette est un schéma utilisé pour définir les zones et les composants d'une interface utilisateur.

J'ai donc réalisé la maquette en suivant une charte graphique définie auparavant, notamment pour les couleurs et la police du texte et en m'inspirant d'une maquette qui avait déjà été établie avec Adobe XD.

Police la plus utilisée : Bahnschrift.

Couleurs principales utilisées : #C2912B (doré), #313131 (noir), #FBFBFB (blanc).

Le maquetage a été réalisé à l'aide du logiciel Pencil, j'ai donc créé un nouveau projet pour l'application, et commencé à réaliser la maquette pour un affichage sur ordinateur.

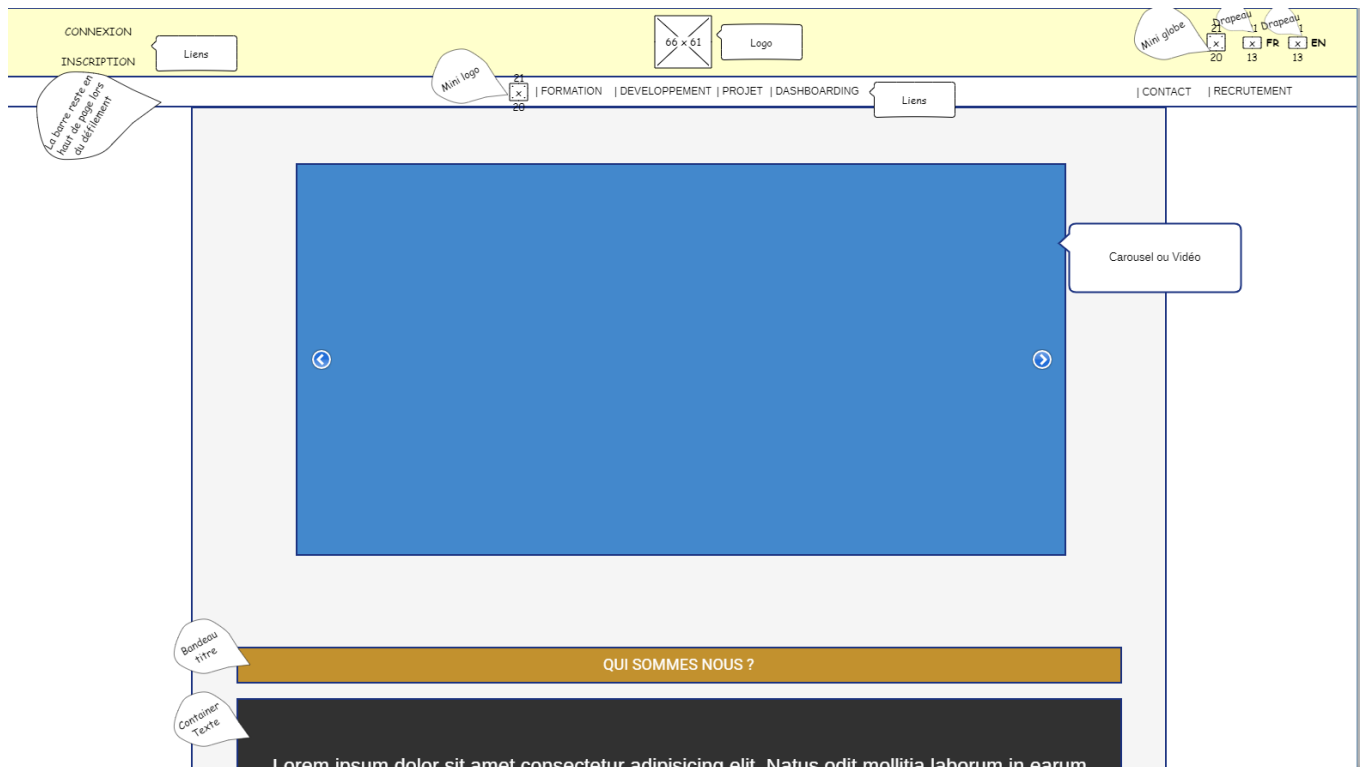
Pour la page d'accueil qui contient la plupart des informations, j'ai commencé par mettre en place le header avec le logo au centre et les boutons de connexion et inscription à gauche et finalement le choix de la langue à droite.

En dessous, on retrouve une barre qui sert à la navigation avec différents liens centrés. À droite on retrouve un lien pour la page contenant le formulaire de contact et un lien pour le recrutement.

Cette seconde barre de navigation reste fixe en haut de la page lorsque l'on fait défiler celle-ci, ce qui contribue à améliorer l'expérience utilisateur en facilitant la navigation.

En dessous nous avons un carrousel ou une vidéo au choix.

Le « header » est réutilisé pour chaque page et il en est de même pour le « footer ».

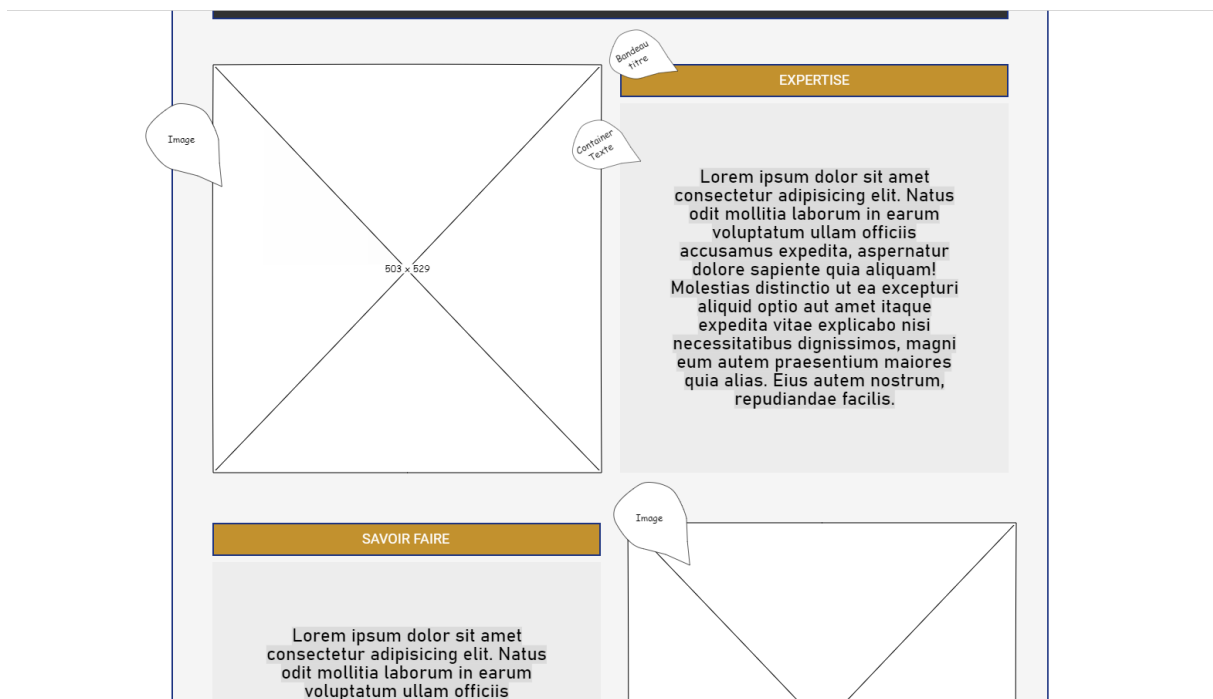


**Maquette page d'accueil**

Ensuite, nous avons tout le contenu de la page d'accueil avec des boutons « en savoir plus » qui mène vers une page spécifique suivant la section où l'on se trouve, il y a également en bas de page les avis et pour finir bien sûr le footer.

Le footer contient un abonnement à une newsletter, contact rapide et adresse de l'entreprise. Des liens pour les réseaux sociaux et pour finir des liens en bas pour les mentions légales, plan du site ect...

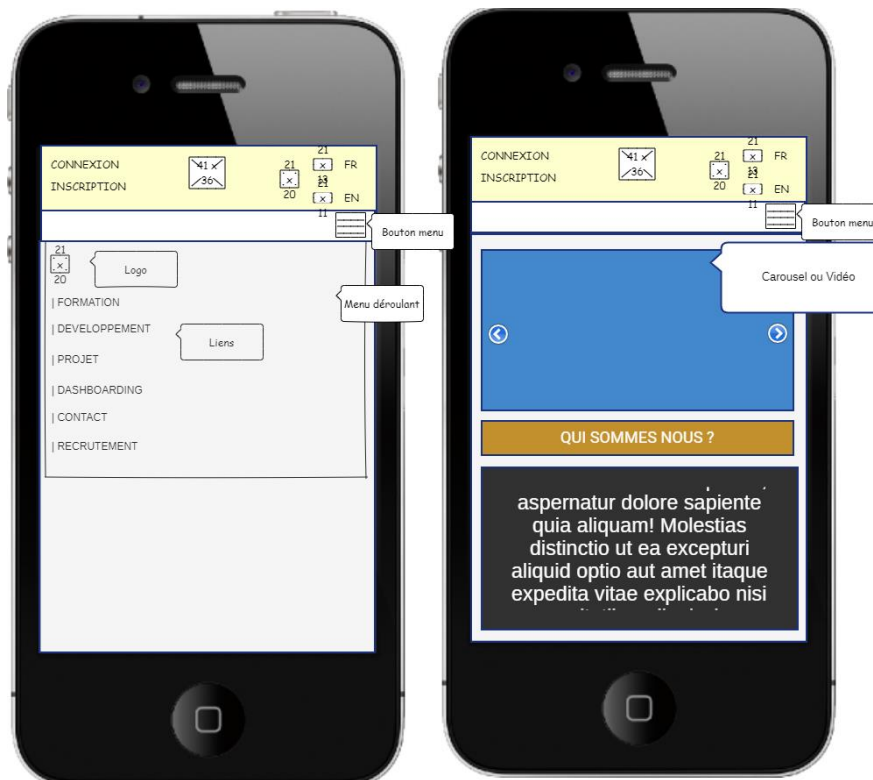
Les autres pages sont réalisées selon le même principe en reprenant le header et footer et en respectant la charte graphique du site.



**Maquette page d'accueil**

Ensuite, j'ai également réalisé la maquette pour les versions mobiles et tablettes.

Ceci en suivant le même principe mais en adaptant le contenu le plus logiquement et simplement à des écrans plus petits. Le menu pour les écrans plus petits se retrouve dans un menu « burger » avec un bouton pour accéder à celui-ci (voir image mobile).



Maquette page d'accueil (mobile)

## 5. Réalisation de la page d'accueil

### 5.1 Route et composants

La page d'accueil de l'application est en grande partie une page statique, puisque son contenu est écrit dans le code (JSX) et aucune base de données y intervient.

Il a donc fallu reproduire la maquette à l'aide de React qui gère la partie front du site et permet grâce au JSX d'écrire du code identique au HTML qui est ensuite compilé.

React propose deux différentes syntaxes : les composants fonctionnels ou les composants de classe, personnellement j'ai utilisé les composants fonctionnels que je préfère aux composants de classe.

L'affichage des différents composants et donc pages est contrôlé par le système de route utilisé par React, appelé React-router-dom.

Celui-ci permet de rendre l'affichage de tel ou tel élément suivant l'url ou chemin (path).

Par exemple on a : `<Route path="/contact" element=<Contact /> />`

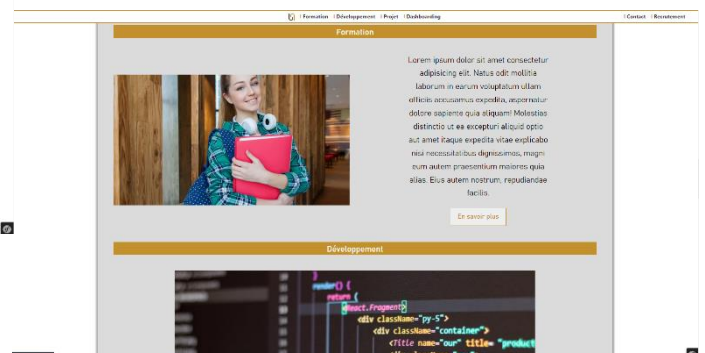
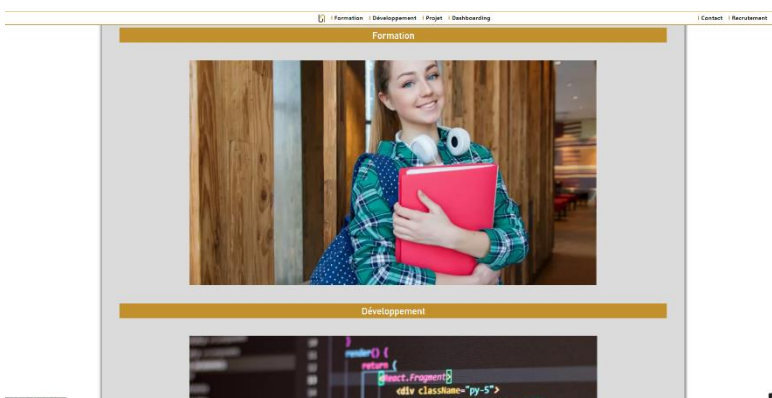
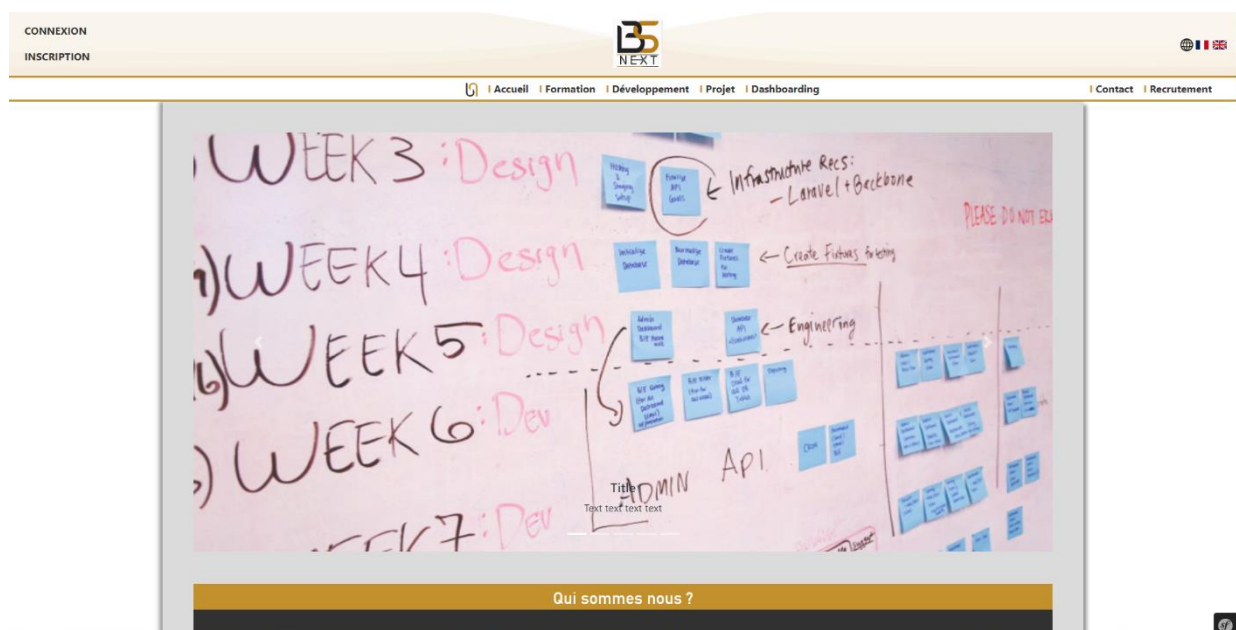
Ainsi sur le « path » /contact on aura donc l'élément « Contact » qui correspond au fichier « contact.js » dont l'affichage sera rendu dans le dom virtuel.

On peut si on le souhaite segmenter nos pages et on peut également créer des composants et les réutiliser dans différentes pages.

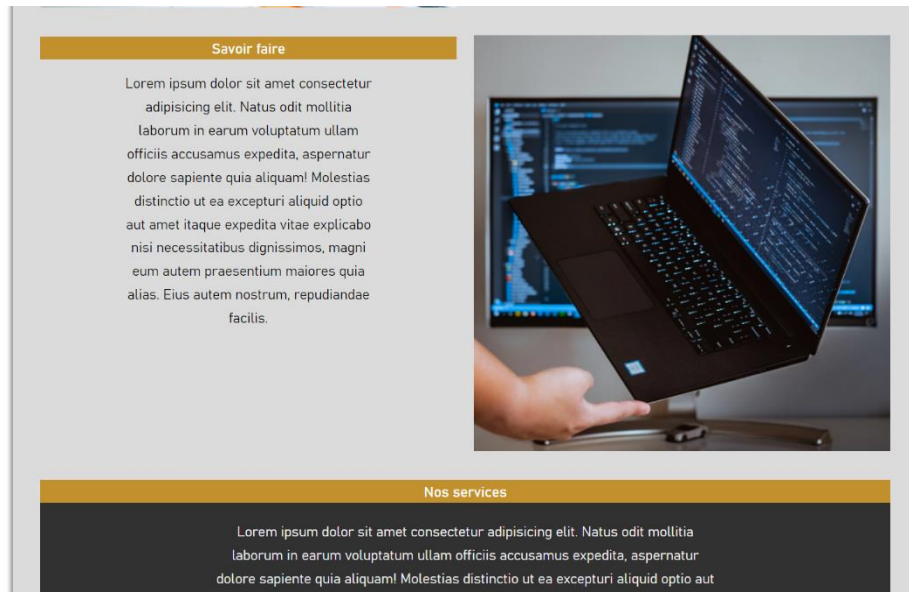
Toutes ces routes sont contenues dans un même fichier (Home.js) qui va gérer suivant la route quel contenu est affiché et ce contenu est ensuite affiché via une « <div id= 'route'> » qui se trouve dans le fichier « template/default/index.html.twig ». Ce rendu est donc fait dans le twig et génère la vue de notre application.

Tout cela est fait grâce au webpack « Encore » qui permet de travailler avec JavaScript (mais aussi du SCSS par exemple) dans Symfony.

### Images du site







```

---
<div className="row">
  <div className="col-xl-6">
    <div className="label-title">
      <p>
        {t('main1.savoir-faire', {ns: 'homepage'})}
      </p>
    </div>
    <div className="label-content-nbg">
      <p>
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus odit mollitia laborum in earum voluptatum ullam officiis accusamus expedita, aspernatur dolore sapiente quia aliquam! Molestias distinctio ut ea excepturi aliquid optio aut amet itaque expedita vitae explicabo nisi necessitatibus dignissimos, magni eum autem praesentium maiores quia alias. Eius autem nostrum, repudiandae facilis.
      </p>
      <div className="text-center"><button className="button-more">{t('button.en-savoir-plus', {ns: 'homepage'})}</button></div>
    </div>
  </div>
  <div className="col-xl-6">
    <img className="img" src={img2} alt="Ordinateur portable" />
  </div>
</div>

---
<div className="label-title">
  <p>
    {t('main1.nos-services', {ns: 'homepage'})}
  </p>
</div>
<div className="label-content">
  <p>
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus odit mollitia laborum in earum voluptatum ullam officiis accusamus expedita, aspernatur dolore sapiente quia aliquam! Molestias distinctio ut ea excepturi aliquid optio aut amet itaque expedita vitae explicabo nisi necessitatibus dignissimos, magni eum autem praesentium maiores quia alias. Eius autem nostrum, repudiandae facilis.
  </p>
  <div className="text-center"><button className="button-more-gold">{t('button.en-savoir-plus', {ns: 'homepage'})}</button></div>
</div>

```

Extrait du site avec son code

## 5.2 Adaptation aux écrans (responsive)

L'application ayant été composée en version ordinateur ou grand écran, il a fallu adapter son affichage selon les supports utilisés (téléphones, tablettes...) afin d'optimiser l'affichage, ceci a été fait grâce à l'utilisation d'unités de mesure ainsi que des Media Queries dans les feuilles de style CSS ainsi que du Framework de style Bootstrap.

Le choix des règles à appliquer repose principalement sur la résolution d'écran, le type d'écran ainsi que son orientation.

```

<div className="row">
  <div className="col-xl-6 justify-content-center">
    <img className="img" src={img1} alt="Image 1" />
  </div>
  <div className="col-xl-6">
    <div className="label-title">
      <p>
        {t('main1.expertise', {ns: 'homepage'})}
      </p>
    </div>
    <div className="label-content-nbg">
      <p>
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus odit mollitia laborum in earum voluptatum ullam officiis accusamus expedita, aspernatur dolore sapiente quia aliquam! Molestias distinctio ut ea excepturi aliquid optio aut amet itaque expedita vitae explicabo nisi necessitatibus dignissimos, magni eum autem praesentium maiores quia alias. Eius autem nostrum, repudiandae facilis.
      </p>
    </div>
    <div className="text-center"><button className="button-more">{t('button.en.savoir.plus', {ns: 'homepage'})}</button></div>
  </div>
</div>

```

Image 1

Image 2

```

533 @media screen and (max-width:1400px){
534   .height-img {
535     height: 500px;
536   }
537   .height-video {
538     height: 450px;
539   }
540 }
541 @media screen and (max-width:1199px){
542   .navbar{
543     height: 60px;
544     background-color: #fff;
545   }
546   .top-nav {
547     height: 120px;
548   }
549   .navbar-collapse
550   {
551     background-color: #fff;
552     margin-top: 9px;
553     margin-left: 0;
554     border: solid 3px #C2912E;
555   }
556 }
557 @media screen and (max-width:1100px){
558   .main-container{
559     width: 100%;
560   }
561   .height-img {
562     height: 450px;
563   }
564   .height-video {
565     height: 400px;
566   }
567   .contact-form-main-container {
568     width: 95%;
569   }
570   .contact-form-container {
571     width: 80%;
572   }
573 }
574 }
575 }

```

Ci-contre (image 2) différents « media queries » suivant la taille de l'écran pour adapter le contenu à différentes tailles suivant la largeur en pixel.

Sur l'image 1, l'utilisation de Bootstrap est ici illustrée, une première « div » avec la classe (className) « row » permet d'afficher les éléments sur une seule ligne.

Bootstrap utilise une grille répartie sur 12 colonnes, c'est ainsi que pour chaque élément compris dans cette « div » on lui attribue la classe « col-xl-6 ».

« Col » pour colonne, « XL » pour la taille de l'écran à partir de laquelle les éléments ne devront plus s'afficher en ligne et côte à côte mais l'un en dessous de l'autre. Ce qui correspond au « wrap » que l'on retrouve avec les Flexbox.

Pour finir le nombre « 6 » correspond aux colonnes occupées par l'élément. Comme dit précédemment il y a 12 colonnes, dans notre cas ayant deux éléments à afficher (une image et un texte) chacun peut occuper 6 colonnes.

Le « justify-content-center » permet de centrer verticalement l'image.

## 5.3 Site multilingues avec i18Next

Le site laisse la possibilité à l'utilisateur de choisir entre la langue Française et Anglaise. Cela a été fait grâce à i18Next, une librairie JavaScript.

Le fonctionnement est simple : il faut créer un dossier par langage dans lequel on va créer des fichiers JSON où l'on va écrire le texte que l'on veut afficher avec une clef et une valeur comme ci-dessous (Français à gauche, Anglais à droite) :

```

20 .....main1": {
21 .....  "qui-sommes-nous": "Qui sommes nous...",
22 .....  "expertise": "Expertise",
23 .....  "savoir-faire": "Savoir-faire",
24 .....  "nos-services": "Nos services"
25 .....
26 .....},
27
28 .....main2": {
29 .....  "formation": "Formation",
30 .....  "developpement": "Développement",
31 .....  "projet": "Projet"
32 .....},
33
34 .....main3": {
35 .....  "conditions d'entrées": "CONDITIONS D'ENTRÉE"
36 .....},
37
38 .....main4": {
39 .....  "votre avis compte": "VOTRE AVIS COMPTE",
40 .....  "veuillez vous connecter": "Veuillez vous connecter",
41 .....  "se connecter": "Se connecter",
42 .....  "laissez votre avis": "Laissez votre avis",
43 .....  "profession": "Votre profession"
44 .....}
45 }

```

Image 3

```

20 .....main1": {
21 .....  "qui-sommes-nous": "WHO ARE WE?",
22 .....  "expertise": "ASSESSMENT",
23 .....  "savoir-faire": "KNOW-HOW",
24 .....  "nos-services": "OUR SERVICES"
25 .....
26 .....},
27
28 .....main2": {
29 .....  "formation": "TRAINING",
30 .....  "developpement": "DEVELOPMENT",
31 .....  "projet": "PROJECT"
32 .....},
33
34 .....},
35
36 .....main3": {
37 .....  "conditions d'entrées": "ENTRY CONDITIONS"
38 .....},
39
40 .....main4": {
41 .....  "votre avis compte": "YOUR OPINION MATTERS",
42 .....  "veuillez vous connecter": "Please sign-in",
43 .....  "se connecter": "Sign in",
44 .....  "laissez votre avis": "Give us your feedback",
45 .....  "profession": "Your job"
46 .....}
47 }

```

Image 4

Ensuite il n'y a plus qu'à utiliser les clefs et suivant le langage choisi, il sera affiché le bon langage correspondant. Bien évidemment les clefs doivent être identiques pour chaque langage et dans chaque fichier Json. Pour la sélection du langage cela se fait automatiquement au chargement du site selon le langage du navigateur de celui-ci.

Si l'utilisateur souhaite quand même changer de langage il peut le faire en sélectionnant le drapeau correspondant en haut à droite. Cela va alors créer un cookie suivant le langage choisi et le cookie étant prioritaire sur le langage du navigateur, on peut modifier la langue d'affichage à souhait.

## 6. Réalisation des autres pages

### 6.1 Page formation, développement, mentions légales ...

J'ai ensuite réalisé différentes pages nécessaires au site, comme la page pour les formations ou encore la page pour les mentions légales.

La page pour les formations se décompose sous la forme d'un carrousel où chaque formation est présentée et l'on peut cliquer sur ce carrousel qui nous amène vers une nouvelle page avec les détails de la formation en question.

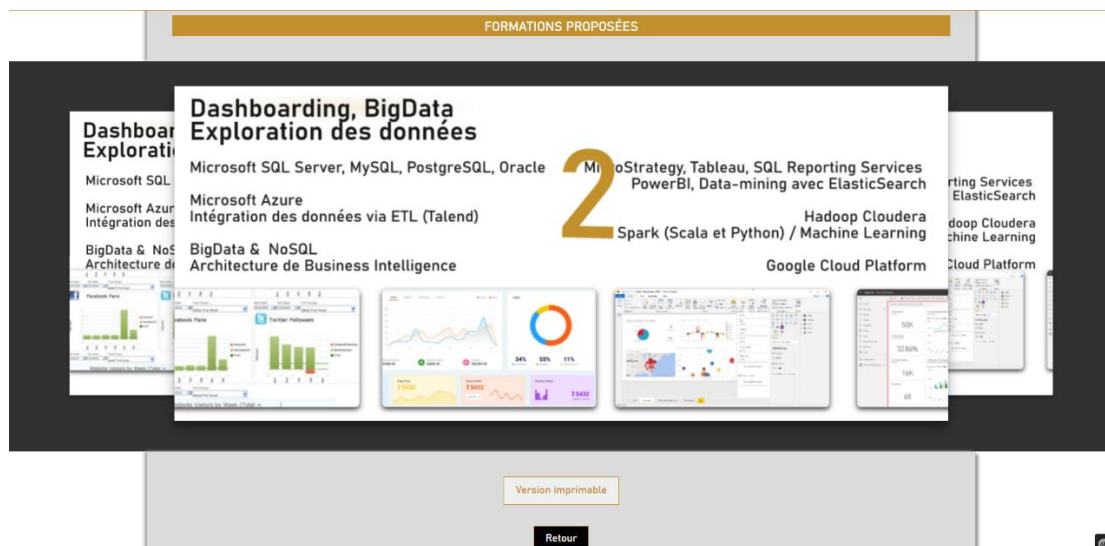


Image du carrousel (Formations)

## 6.2 Formik et Yup pour la validation des formulaires

Pour la validation des formulaires, j'ai utilisé « Formik » et « Yup » qui fonctionne très bien ensemble.

On peut prendre l'exemple du formulaire de contact pour illustrer leur fonctionnement.

```
const validation = Yup.object({
  ...name: Yup.string()
  ... .max(15, '15 caractères maximum')
  ... .required('Ce champ est requis'),
  ...firstname: Yup.string()
  ... .max(15, '15 caractères maximum')
  ... .required('Ce champ est requis'),
  ...email: Yup.string()
  ... .email('Email invalide')
  ... .required('Ce champ est requis'),
  ...phone: Yup.string()
  ... .max(15, '15 caractères maximum')
  ... .min(10, '10 caractères minimum')
  ... .number()
  ... .required('Ce champ est requis'),
  ...companyName: Yup.string(),
  ...companyAddress: Yup.string(),
  ...job: Yup.string()
  ... .required('Ce champ est requis'),
  ...message: Yup.string()
  ... .required('Ce champ est requis')
})
```

Image 5

```
36 ...const [msg, setMsg] = useState('');
37 ...
38 ...const formik = useFormik({
39 ...  initialValues: {
40 ...    name: '',
41 ...    firstname: '',
42 ...    email: '',
43 ...    phone: '',
44 ...    companyName: '',
45 ...    companyAddress: '',
46 ...    job: '',
47 ...    message: '',
48 ...  },
49 ...  onSubmit: (values) => {
50 ...
51 ...    console.log(JSON.stringify(values));
52 ...
53 ...    axios({
54 ...      method: 'post',
55 ...      url: 'http://127.0.0.1:8000/api/contacts',
56 ...      data: {
57 ...        name: formik.values.name,
58 ...        firstname: formik.values.firstname,
59 ...        email: formik.values.email,
60 ...        phone: formik.values.phone,
61 ...        companyName: formik.values.companyName,
62 ...        companyAddress: formik.values.companyAddress,
63 ...        job: formik.values.job,
64 ...        message: formik.values.message,
65 ...      },
66 ...      headers: {
67 ...        'Access-Control-Allow-Origin': '*',
68 ...        'Content-Type': 'application/json',
69 ...      }
70 ...    }).then(response => {
71 ...      console.log(response.data);
72 ...      setMsg(response.data);
73 ...    })
74 ...    formik.resetForm();
75 ...  },
76 ...
77 ...  validationSchema: validation
78 ...})
79 ...
```

Image 6

On commence par définir un schéma de validation (image 5). On définit une fonction pour gérer la validation. Elle reçoit les valeurs des données du formulaire et valide chaque propriété en fonction des règles définies. Chaque clef de l'objet doit correspondre au nom du champ de saisie.

On a ensuite un objet « initialValues » pour les valeurs initiales des champs du formulaire. Ensuite lors de la soumission du formulaire s'il n'y a aucune erreur on exécute la requête avec Axios en récupérant les informations saisies grâce à Formik.

```
<input className="int form-control shadow-none" placeholder={t('nom', {ns: 'contact'})} name="name" type="text" value={formik.values.name} onChange={formik.handleChange} />
<small className="text-danger margin">{formik.touched.name && formik.errors.name}</small>
```

Image 7

Ici un exemple d'un champ de saisie (image 7), on a « onChange » qui s'agit du gestionnaire d'événement de changement d'entrée et qui est passé dans le champ de saisie.

Cela met à jour la valeur (value).

En dessous la balise « small » permet d'afficher les erreurs lorsqu'il y en a.

## 6.3 Connexion et inscription

Les visiteurs du site ont la possibilité de s'inscrire et se connecter. Tout cela est géré par Firebase et son système d'authentification (OAuth). Un utilisateur peut se connecter/s'inscrire avec son compte Facebook, Twitter ou Google ou bien encore avec une adresse électronique et un mot de passe.

Image 8

```
function register(email, password) {
  return createUserWithEmailAndPassword(auth, email, password);
}
function signIn(email, password) {
  return signInWithEmailAndPassword(auth, email, password);
}
function signInWithGoogle() {
  const googleProvider = new GoogleAuthProvider();
  return signInWithPopup(auth, googleProvider);
}
function signInWithFacebook() {
  const facebookProvider = new FacebookAuthProvider();
  return signInWithPopup(auth, facebookProvider);
}
function signOutAccount() {
  return signOut(auth);
}
function forgotPassword(email) {
  return sendPasswordResetEmail(auth, email);
}
```

Image 8 bis



La connexion se fait grâce aux outils du SDK Firebase (Software Development Kit) comme par exemple avec « `SignInWithEmailAndPassword()` », qui permet de connecter un utilisateur à l'aide du mot de passe et de l'email (voir image 8).

Tout cela (image 8) est placé dans ce que l'on appelle un contexte (le contexte offre un moyen de faire passer des données à travers l'arborescence du composant sans avoir à passer manuellement les « props » à chaque niveau. Celui-ci permet notamment de récupérer l'utilisateur connecté à n'importe quel niveau de l'application.

## 6.4 Avis laissés par les utilisateurs

En fin de page on retrouve les avis laissés par les utilisateurs pour donner un retour sur leur expérience avec l'entreprise BS Next (Image 9).



Image 9

Image 10

Pour communiquer avec Firebase, cela se fait avec la partie Backend, c'est-à-dire Symfony qui est utilisé comme une API.

La base de données en temps réel de Firebase est utilisée (NoSQL), celle-ci fonctionne comme un fichier JSON avec une arborescence.

On peut donc définir ce qu'on appelle des références ou nodes dans lesquelles on place les données. Pour l'affichage des commentaires ceux-ci se trouvent sous « feedback » (image 10).

Enfin pour les requêtes entre React et Symfony, Axios a été utilisé.

Axios est un client HTTP basé sur les promesses (Une promesse est un objet qui représente la complétion ou l'échec d'une opération asynchrone).

Chaque avis laissé est d'abord affiché côté administrateur pour être soumis à validation (ou non) par mesure de sécurité et donc avoir un contrôle de ce qui sera affiché sur le site.



Image 11

Sur l'image 12 nous avons un « useEffect » qui permet d'exécuter le code qu'il contient après que le composant a été monté.

On fait appel à Axios et la méthode get pour récupérer les données qu'on affecte à un « hook » « dataDisplay ».

Sur l'image 11 on affiche les données avec la fonction « map » pour boucler sur les données récupérées.

```
<div className="m-auto">
  { dataDisplay.reverse().map((data, index) =>
    <div className="mb-3 feedback-item-3" key={data.uid + index}>
      <img src={data.photo ? data.photo : profil} alt="Photo Profil"/>
      <strong><p>{data.nom}</p></strong>
      <i><p>{data.date}</p></i>
      <p>{data.note == '1' && <img src={star1} alt="1 étoile/star"/>}</p>
      <p>{data.note == '2' && <img src={star2} alt="2 étoiles/stars"/>}</p>
      <p>{data.note == '3' && <img src={star3} alt="3 étoiles/stars"/>}</p>
      <p>{data.note == '4' && <img src={star4} alt="4 étoiles/stars"/>}</p>
      <p>{data.note == '5' && <img src={star5} alt="5 étoiles/stars"/>}</p>
      <p><strong>Profession / Job :</strong> {data.profession}</p>
      <p><strong>Avis / Feedback :</strong> {data.message}</p>
    </div>
  )}
</div>
```

Image 12

```
//Affiche les Avis au chargement
useEffect(() => {
  axios.get('http://127.0.0.1:8000/api/feedback_disp')
    .then(response => {
      console.log(response.data)
      setDataDisplay(response.data);
    });
}, []);

const handleSubmit = async (e) => {
  e.preventDefault();
  if (starr == "0") {
    setInfo('Choisissez une note / Select a grade');
  }
  else {
    await axios({
      method: 'post',
      url: `${apiUrl}feedback_submit`,
      data: {
        UID: currentUser.uid,
        nom: currentUser.displayName,
        idToken: idToken,
        photo: currentUser.photoURL,
        prof: prof,
        message: message,
        note: starr,
        date: `Le ${date} à ${time}`
      },
      headers: {
        'Access-Control-Allow-Origin': '*',
        'Content-Type': 'application/json',
      }
    }).then(response => {
      setInfo(response.data);
      setProf('');
      setMessage('');
    })
  }
}
```

Sur l'image 13, on a le code pour envoyer un nouvel avis pour la validation par l'administrateur.

On va maintenant s'intéresser à Symfony et voir comment les avis sont gérés dans cette partie.

Image 13



```

/**
 * @Route("/api/feedback_submit", name="feedback_submit")
 */
public function index(Request $request)
{
    //On récupère les données venant du front
    $content = json_decode($request->getContent());

    //On assigne chaque donnée récupérée à une variable
    $uid = $content->uid;
    $nom = $content->nom;
    $photo = $content->photo;
    $idToken = $content->idToken;
    $prof = htmlspecialchars($content->prof);
    $message = htmlspecialchars($content->message);
    $note = $content->note;
    $date = $content->date;

    //Vérification utilisateur connecté
    $verifiedIdToken = $this->auth->verifyIdToken($idToken);
    $uidCheck = $verifiedIdToken->claims()->get('sub');
    $user = $this->auth->getUser($uidCheck);

    //Instancie la classe Response
    $response = new Response();

    $response->headers->set('Content-Type', 'application/json');
    $response->headers->set('Access-Control-Allow-Origin', '*');

    if ($user) {
        //Déclare le tableau pour remplir la BDD
        $postData = [
            'id' => 'default',
            'uid' => $uid,
            'nom' => $nom,
            'photo' => $photo,
            'profession' => $prof,
            'message' => $message,
            'note' => $note,
            'date' => $date
        ];

        try {
            //Envoie à la BDD du Tableau ci-dessus
            $push = $this->database->getReference('feedbackPending')->push($postData);

            //On change l'ID par la clef
            $key = $push->getKey();

            $ref = '/feedbackPending/' . $key . '/' . $id;
            $this->database->getReference($ref)->set($key);

            $response->setContent(json_encode('Votre avis a été soumis / Feedback submitted successfully'));

            return $response;
        } catch (FirebaseException $e) {
            $response->setContent(json_encode('Erreur : votre avis n\'a pas pu être ajouté / Your feedback couldn\'t be added : ' . $e->getMessage()));

            return $response;
        } catch (Throwable $e) {
            $response->setContent(json_encode('Erreur : votre avis n\'a pas pu être ajouté / Your feedback couldn\'t be added : ' . $e->getMessage()));

            return $response;
        }
    } else {
        $response->setContent(json_encode('Erreur : votre avis n\'a pas pu être ajouté / Your feedback couldn\'t be added'));

        return $response;
    }
}

```

Image 14 : Contrôleur pour soumettre un avis

On a ici la méthode index qui permet de gérer la soumission du formulaire pour laisser un avis qui sera enregistré dans « feedbackPending » qui permet de mettre le commentaire en attente et qui permet à l'administrateur de valider ou non l'avis dans le back office.

Première chose qu'on le fait est de récupérer les données grâce à l'objet « Request ». On affecte ensuite chaque donnée récupérée (nom, message...) à une variable.

Ensuite, on vérifie l'utilisateur qui émet cet avis grâce à « l'idToken », Firebase permet de vérifier l'utilisateur grâce à ce token et on peut récupérer « l'UID » de l'utilisateur correspondant.

S'il y a un problème on retourne une erreur sinon on commence par déclarer un tableau avec les données à envoyer à la « Realtime Database » de Firebase.

On utilise ensuite un bloc « try & catch » pour récupérer les exceptions ou erreurs potentielles.

On envoie les données à la base de données et ensuite on récupère la valeur de la clef de la nouvelle « node ou référence » créée pour l'enregistrer également un niveau plus bas dans notre arborescence. C'est-à-dire au même niveau que les données de l'avis laissé.

Image 15 : Contrôleur pour valider un avis (back office)

```

/**
 * @Route("/api/feedback_validate", name="feedback_validate")
 */
public function validateFeedback(Request $request)
{
    //On récupère les données venant du front
    $content = json_decode($request->getContent());

    //On assigne chaque donnée récupérée à une variable
    $id = $content->id;
    $idToken = $content->idToken;

    //Verification des droits (admin)
    $verifiedIdToken = $this->auth->verifyIdToken($idToken);
    $uid = $verifiedIdToken->claims()->get('sub');
    $claims = $this->auth->getUser($uid)->customClaims;

    if ($claims['admin'] === true) {
        $reference = 'feedbackPending/'.$id;
        $snapshot = $this->database->getReference($reference)->getSnapshot();
        $postData = $snapshot;

        try {
            //Envoie à la BDD du commentaire validé
            $push = $this->database->getReference('feedback')->push($postData);

            //On change l'ID par la clef
            $key = $push->getKey();
            $ref = '/feedback/'.$key.'/id';
            $this->database->getReference($ref)->set($key);

            //Suppression du message ajouter dans feedbackPending
            $this->database->getReference('feedbackPending/'.$id)->remove();

            $response->setContent(json_encode('Avis Validé'));
            return $response;
        } catch (FirebaseException $e) {
            $response->setContent(json_encode('Une erreur s\'est produite : '.$e->getMessage()));
            return $response;
        } catch (Throwable $e) {
            $response->setContent(json_encode('Une erreur s\'est produite : '.$e->getMessage()));
            return $response;
        }
    } else {
        $response->setContent(json_encode('Impossible de trouver les droits d\'Administrateur nécessaire'));
        return $response;
    }
}

```

On a ici la méthode « validateFeedback » qui permet à un administrateur de valider un avis laissé. On récupère encore une fois les données grâce à l'objet « Request ». Tout ce dont on a besoin c'est de récupérer « l'ID » ou plus précisément la valeur de la clef qui correspond à la « node ou référence » que l'on veut ici valider.

Puis il nous faut également récupérer « l'UID » de l'utilisateur pour vérification des droits.

En théorie un utilisateur n'ayant pas le rôle « admin » ne peut pas accéder à cette partie mais par mesure de sécurité, je vérifie à nouveaux si les droits sont valides avant toutes opérations.

Ayant récupéré ces deux données on peut tout d'abord vérifier les droits grâce aux différentes méthodes du SDK de Firebase comme la méthode « verifyIdToken ».

S'il n'y a pas les droits nécessaires on retourne une erreur sinon grâce à « l'ID » récupéré on récupère les données de l'avis qui se trouvent dans « feedbackPending » et on va ensuite les enregistrer dans « feedback » qui contient les avis affichés sur le site.

Ensuite on va supprimer l'avis dans « feedbackPending » puisque celui-ci a été validé.

## 6.5 Les routes protégées

Certaines parties du site ne doivent être accessibles qu'à un utilisateur connecté, ou encore la partie administration (back office) ne doit être accessible qu'à un utilisateur avec les droits nécessaires (admin). On ne veut pas non plus qu'un utilisateur connecté puisse accéder au formulaire de connexion ou d'inscription.

Tout ceci est fait grâce à ce que l'on appelle les routes protégées, un fichier nommé « ProtectedRoute.js » contrôle cela.

Dans ce fichier on utilise le contexte utilisé pour l'authentification :

```
import { useAuth } from "../contexts/AuthContext";
```

Ensuite, on peut récupérer l'utilisateur connecté, si dorénavant il y en a un, et suivant si un utilisateur est connecté ou pas il peut accéder à la route demandée sinon il est redirigé vers le formulaire de connexion :

```
export const ProtectedRouteNotSignedIn = () => {
  const { currentUser } = useAuth();
  return currentUser ? <Outlet/> : <Navigate to="/signinr" /> };
```

Ce fichier contenant les fonctions pour protéger les routes est ensuite utilisé dans « Home.js » qui gère toutes les routes de l'application, il suffit donc d'entourer les routes que l'on veut protéger comme ci-dessous :

```
<Route element={<ProtectedRouteNotSignedIn/>}>
  <Route path="/profil" element={<Profile />} />
</Route>
```

Ainsi si je veux me rendre sur le profil et que je ne suis pas connecté je serai redirigé vers le formulaire de connexion (/signinr).

À savoir qu'il y a aussi une vérification côté Back-end pour par exemple vérifier qu'un utilisateur a bien les droits de laisser un commentaire (l'utilisateur doit être connecté) ou bien valider ou supprimer un commentaire pour un administrateur.

## 6.6 Sécurité

Pour l'authentification Firebase utilise OAuth qui fonctionne avec des Json Web Token (JWT). Du côté de la base de données Firebase (Realtime Database), la sécurité y est très bien gérée également. Firebase propose de définir des règles pour contrôler qui peut écrire ou lire des données. Ci-dessous on autorise l'accès aux données en écriture et lecture sous « feedbackPending » seulement si l'utilisateur est connecté et qu'il a le rôle Admin.

```
{
  "rules": {
    "feedbackPending": {
      ".read": "auth.token.admin === true",
      ".write": "auth.token.admin === true",
    }
  }
}
```

En revanche en utilisant le back-end avec « admin SDK » comme lors de l'ajout d'un avis par exemple, ces règles ne sont pas prises en compte, on peut donc mettre « read & write » à false.

Autre point de sécurité : CORS (Cross Origin Resource Sharing) qui permet de définir quel domaine a accès à l'API (Symfony) et interdit donc l'accès à tout autre domaine que celui qui est défini. Pour finir il y a les Json Web Token (JWT) qui sont notamment utilisés pour l'authentification.

## 7. Tests unitaires

J'ai mis en place des tests unitaires dans la partie front de l'application afin de vérifier par exemple le bon rendu d'un composant. Ces tests ont été réalisés grâce à « Jest » et « react testing library ».

Je teste ici par exemple le bon rendu du composant « Services » en lui passant des « props » et en vérifiant que l'affichage ensuite est correct.

Sur l'image ci-contre je définis donc 4 « props » : title, image, link et children. Je passe ensuite ces « props » au composant. On vérifie ensuite que les éléments sont dans le document et que les valeurs attendues (props) sont également présentes.

```
describe('services', () => {
  it('uses props', () => {
    const title = 'titre';
    const image = 'image.jpg';
    const link = '/test';
    const children = 'Lorem ipsum';

    render(
      <Services
        title={title}
        image={image}
        link={link}
      >
        {children}
      </Services>
    );

    const elementTitle = screen.getByTestId('title');
    expect(elementTitle).toBeInTheDocument();
    expect(elementTitle).toHaveTextContent('titre')

    const elementImage = screen.getByTestId('image');
    expect(elementImage).toHaveAttribute('src', 'image.jpg');

    const elementLink = screen.getByTestId('link');
    expect(elementLink).toBeInTheDocument();
    expect(elementLink).toHaveAttribute('to', '/test');

    const elementChildren = screen.getByTestId('children');
    expect(elementChildren).toBeInTheDocument();
    expect(elementChildren).toHaveTextContent('lorem ipsum');
  });
});
```

## 8. Avenir du projet

En fin de stage, mon tuteur m'a demandé de rajouter une fonctionnalité qui était de pouvoir créer des questionnaires et que les utilisateurs puissent y répondre. Le développement n'étant pas totalement terminé j'ai placé cette partie en Annexe (Annexe 1).

Le projet a été développé avec React en front ce qui était un souhait de l'entreprise (mon tuteur) au départ. Nous avons beaucoup discuté de l'importance du référencement pour ce site et j'ai lui est donc fait remarquer le problème de l'utilisation de React pour le site et qu'un Framework comme Next JS serait beaucoup plus adapté.

En effet React est une application à page unique (SPA) et cela n'est pas très performant en termes de référencement car non compréhensible pas le moteur d'indexation de google.

Les « SPA » sont chargées sur une seule page, les données changent grâce au Javascript et les robots ne comprennent pas, ils n'y voient qu'une page blanche.

À l'inverse Next JS propose des fonctionnalités comme « SSG » ou « SSR » ou génération de sites statiques ou le rendu côté serveur. Cela améliorant grandement le référencement du site et également les performances. En effet Next JS fait un rendu côté serveur là où React JS fait un rendu côté client.

C'est donc pourquoi en fin de stage mon tuteur m'a demandé d'entreprendre le changement de la partie Front-end du site pour passer de React JS à Next JS. Malheureusement avec tous les changements et développements arrivés en fin de stage je n'ai pas eu le temps d'arriver à l'étape de la mise en production.

# Partie C : Projet personnel PostYours

## 1. Introduction

### 1.1 Présentation du projet personnel

Ce second projet consiste à réaliser une application qui a pour but principal de permettre à un utilisateur de partager des photos/images mais aussi de laisser des commentaires sur les photos d'autres utilisateurs.

Comme précisé dans la partie technologies cette application est développée entièrement avec le Framework Symfony et une base de données MySQL.

Chaque photo publiée comprend également un titre et une description.

La page d'accueil affichera toutes les photos/images publiées, les plus récentes seront affichées en premières. En cliquant sur une image on accède à une page avec cette même image en plus grand, le titre et la description complète. Cela donne aussi la possibilité de voir les commentaires qui ont pu être laissés et d'en laisser un dans le cas où l'on est connecté.

Un utilisateur pourra également s'inscrire, se connecter depuis la page d'accueil et devra évidemment être inscrit et connecté pour pouvoir accéder à la page de publication d'une photo. Il en sera de même pour accéder à sa page profil où il pourra y retrouver ses informations (qu'il pourra modifier), modifier son mot de passe et également les photos qu'il a publiées sur l'application. Lorsqu'une photo sera publiée il aura la possibilité de la modifier, de modifier le titre, la description et bien évidemment de simplement la supprimer.

### 1.2 Technologies utilisées

**Twig** : Pour rendre la vue de l'application.

**CSS 3 et Bootstrap** : Tous deux pour « l'habillage » du site et l'adaptation à la taille de l'écran.

**Symfony** : Pour la partie back-end du site.

**Wamp** : Pour le serveur local et la base de données MySQL

## 2. Front-end de l'application et généralités

La partie front-end étant couverte par la partie B, le but ici est juste de préciser quelques éléments et remettre dans le contexte.

Symfony utilise le modèle MVC (Modèle-vue-contrôleur), c'est-à-dire que l'on a un contrôleur qui fait le lien entre la vue (Twig dans Symfony) et le modèle, qui gère l'accès aux données, la base de données dans notre cas. Le contrôleur gère toutes les actions à effectuer.

Dans ce projet, Twig (le plus utilisé dans le cas de Symfony) est utilisé pour générer la vue (ce que l'utilisateur voit). Le code est donc écrit avec le langage HTML associé bien évidemment avec le langage CSS pour le style. Twig permet de créer un modèle (template) de base que l'on peut réutiliser dans toutes les pages grâce notamment à des « blocks » que l'on définit (voir image ci-contre).

Cela permet par exemple de réutiliser le footer et la navigation sans avoir à les remettre manuellement dans chaque nouveau fichier.

Twig va bien plus loin que du simple langage HTML et offre la possibilité d'utiliser des boucles, des conditions, ce qui devient nécessaire étant donné que l'on n'utilise pas de code PHP dans le fichier Twig.

Pour le style Bootstrap a été également utilisé en grande partie pour ce projet.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>

    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Montserrat&display=swap" rel="stylesheet">

    {% block stylesheets %}
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
      integrity="sha384-1BmE4kWBq78iYhFtdvKuhfTAU6au08ttT94WPHftjdBrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
    <link rel="stylesheet" type="text/css" href="{{asset('assets/css/bootstrap.min.css')}}">
    <link rel="stylesheet" type="text/css" href="{{asset('assets/css/app.css')}}">
    {% endblock %}
  </head>
  <body>

    <{{include ("components/_nav.html.twig")}}>

    <main role="main" class="container-fluid">
      {% for label, messages in app.flashes %}
      {% for message in messages %}
        <div class="text-center alert alert-{{label}}">
          {{ message }}
        </div>
      {% endfor %}
      {% endfor %}
    </main>

    {% block body %}
    {% endblock %}
  </body>

  <{{include ("components/_footer.html.twig")}}>
```

## 3. Base de données

### 3.1 Connexion de la base de données à Symfony

Dans le dossier du projet, les informations de connexion ont été stockées dans une variable d'environnement appelé 'DATABASE\_URL', elle-même placée dans le fichier « .env » à la racine du projet. Cette variable permet la connexion à la base de données (serveur Wamp), elle contient les identifiants ainsi que le nom de la base de données.

### 3.2 Création de la base de données

J'ai choisi d'utiliser une base de données MySQL, car c'est une base de données relationnelle qui correspond aux attentes et aux spécificités du projet.

La base de données a été créée en utilisant l'ORM Doctrine de Symfony.

Il est possible de créer la base de données après avoir renseigné la variable d'environnement avec la commande : « symfony console doctrine:database:create ». Ainsi après cela si l'on se rend dans PhpMyAdmin on y trouvera notre base de données fraîchement créée (vide bien évidemment).

L'étape suivante est de créer les tables de notre base de données, pour cela on utilise évidemment Doctrine et l'on va créer les entités (Entity) ainsi que les « repository », qui correspondent à chaque table.

Doctrine permet la génération des instructions SQL nécessaires pour créer les tables de la base de données lors de la création des entités ce que l'on va voir dans la prochaine partie.

### 3.3 Création des entités et des tables

Pour accéder aux données il a fallu commencer par créer des classes entités (Entity) contenant un objet propre à ce que l'on souhaite représenter. Ceci permet la génération des instructions SQL qui permettront de créer les tables dans la base de données.

Cette entité est ensuite connectée à la table correspondante de la base de données via Doctrine.

Concrètement, on peut modéliser rapidement des entités via la commande « Symfony console make:entity » celui-ci nous pose alors des questions sur le nom de l'entité désirée, ainsi que le nom de chaque attribut de classe à ajouter, son type et sa longueur.



Une fois tous les attributs énumérés et la validation dans la console le fichier contenant l'entité est créé (ainsi que le repository) et est disponible dans le dossier des entités. Ce fichier contient les attributs déclarés ainsi que leurs « getters et setters » et les méthodes obligatoires à implémenter.

```
<?php

namespace App\Entity;

use App\Repository\ImagesRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\HttpFoundation\File\File;
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: ImagesRepository::class)]
#[ORM\HasLifecycleCallbacks]
/**
 * @Vich\Uploadable
 */
class Images
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[Assert\NotBlank(message: 'Ce champ ne peut être vide')]
    #[Assert\Length(
        min: 2,
        max: 25,
        minMessage: 'Le titre doit comprendre 2 caractères minimum',
        maxMessage: 'Le titre ne doit pas dépasser 25 caractères',
    )]
    #[ORM\Column(type: 'string', length: 255, nullable: false)]
    private $title;

    #[Assert\Length(
        max: 300,
        maxMessage: 'La description ne doit pas dépasser 300 caractères',
    )]
    #[ORM\Column(type: 'text', nullable: true)]
    private $description;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTitle(): ?string
    {
        return $this->title;
    }

    public function setTitle(string $title): self
    {
        $this->title = $title;

        return $this;
    }

    public function getDescription(): ?string
    {
        return $this->description;
    }

    public function setDescription(?string $description): self
    {
        $this->description = $description;

        return $this;
    }
}
```

### Extraits de l'entité Images

```
final class Version20211224092641 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your
        // needs
        $this->addSql('CREATE TABLE images (id INT AUTO_INCREMENT NOT NULL, category_id INT NOT NULL, user_id INT NOT NULL, title VARCHAR(255) NOT NULL, description LONGTEXT DEFAULT NULL, INDEX IDX_E01FBE6A12469DE2 (category_id), INDEX IDX_E01FBE6AA76ED395 (user_id), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
        $this->addSql('ALTER TABLE images ADD CONSTRAINT FK_E01FBE6A12469DE2 FOREIGN KEY (category_id) REFERENCES category (id)');
        $this->addSql('ALTER TABLE images ADD CONSTRAINT FK_E01FBE6AA76ED395 FOREIGN KEY (user_id) REFERENCES user (id)');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to
        // your needs
        $this->addSql('DROP TABLE images');
    }
}
```

À ce stade l'entité est créée, mais la table n'existe pas dans la base de données. Il faut créer un fichier de migration qui va contenir le code SQL à exécuter dans le gestionnaire de base de données (image ci-contre). Ceci se fait via la commande « symfony console make:migration » dans le terminal. Puis, il faut faire la mise à jour la base de données en fonction du fichier précédemment généré en utilisant la ligne de commande « symfony console doctrine:migrations:migrate ».

### Extrait du fichier de migration

## 4. Back-end de l'application

### 4.1 Inscription et connexion

#### a) L'inscription

Les entités et la base de données avec ses tables étant créées, la prochaine étape est de créer l'inscription et la connexion pour les utilisateurs.

Pour cela j'ai commencé par créer un contrôleur grâce à la commande « symfony console make:controller ». Il nous est alors demandé d'entrer le nom (Register) puis de valider.

Deux fichiers sont alors créés un contrôleur (src/controller/RegisterController.php) et un fichier Twig (templates/register/index.html.twig) placer dans un dossier « register » pour la vue.

La dernière étape est de créer le formulaire d'inscription avec la commande « symfony console make:form », là encore plusieurs questions nous sont posées notamment à quelle entité il faut relier le formulaire et finalement un fichier est créé dans src/Form/RegisterType.php.

Les fichiers nécessaires étant maintenant créés, on peut s'occuper d'écrire le code pour générer le formulaire.

Je commence par construire le formulaire grâce au fichier « RegisterType » qui contient une méthode « buildForm » à laquelle on a injecté la « FormBuilderInterface » tout cela permettant de définir les champs du formulaire.

La méthode « add() » permet de définir les champs du formulaire, elle prend en premier argument l'attribut de l'entité à laquelle elle se réfère. Puis le type de champ (text, email, mot de passe...) et enfin un tableau d'options.

Dans ce tableau on peut déclarer plusieurs choses comme le « label », des contraintes et des attributs comme le « placeholder » par exemple (ci-contre le fichier final obtenu).

```

16 class RegisterType extends AbstractType
17 {
18     public function buildForm(FormBuilderInterface $builder, array $options): void
19     {
20         $builder
21             ->add('firstname', TextType::class, [
22                 'label' => 'Prénom',
23                 'constraints' => new Length(['min' => 2, 'max' => 30]),
24                 'attr' => [
25                     'placeholder' => 'Entrez votre prénom'
26                 ]
27             ])
28             ->add('lastname', TextType::class, [
29                 'label' => 'Nom',
30                 'constraints' => new Length(['min' => 2, 'max' => 30]),
31                 'attr' => [
32                     'placeholder' => 'Entrez votre nom'
33                 ]
34             ])
35             ->add('email', EmailType::class, [
36                 'label' => 'Email',
37                 'constraints' => new Length(['min' => 2, 'max' => 50]),
38                 'attr' => [
39                     'placeholder' => 'Entrez votre adresse email'
40                 ]
41             ])
42             ->add('password', RepeatedType::class, [
43                 'type' => PasswordType::class,
44                 'invalid_message' => 'Le mot de passe et la confirmation doivent être identiques',
45                 'label' => 'Mot de passe',
46                 'required' => true,
47                 'first_options' => [
48                     'label' => 'Mot de passe',
49                     'attr' => [
50                         'placeholder' => 'Entrez votre mot de passe',
51                     ]
52                 ],
53                 'second_options' => [
54                     'label' => 'Confirmez votre mot de passe',
55                     'attr' => [
56                         'placeholder' => 'Confirmation du mot de passe'
57                     ]
58                 ]
59             ])

```

Cette étape étant terminée nous pouvons alors nous rendre dans le contrôleur, dans la méthode `index` et commencer par instancier la classe « User » pour créer un nouvel objet. Ceci nous permettra donc de créer un nouvel utilisateur dans la table User de notre base de données.

Ensuite on peut déclarer une variable « form » et on utilise la méthode « `createForm` » et celle-ci requière deux arguments le formulaire qu'on veut créer par cette méthode, qui est donc « RegisterType » et la classe que nous avons instanciée : « User ».

À partir de ce moment on peut passer le formulaire à notre vue Twig et voir si celui-ci est rendu. Pour cela on lui passe la variable « form » de notre contrôleur contenant notre formulaire et on utilise la méthode « `createView()` » pour créer la vue du formulaire.

Du côté de Twig il suffit d'utiliser la fonction « `form()` » et de lui passer en argument le nom donné au formulaire côté contrôleur, dans notre cas : formulaire.

Comme prévu on obtient bien notre formulaire du côté de la vue.

```

..#[Route('/inscription', name: 'register')]

..public function index(Request $request, UserPasswordHasherInterface $encoder, MailerInterface $mailer): Response
..{
.....if ($this->getUser()) {
.....|..return $this->redirectToRoute('home');
.....}
.....
.....$user = new User();
.....$form = $this->createForm(RegisterType::class, $user);

.....$form->handleRequest($request);

.....if ($form->isSubmitted() && $form->isValid()) {
.....|..$user = $form->getData();

.....|..$password = $encoder->hashPassword($user, $user->getPassword());
.....|..$user->setPassword($password);
.....|..$user->setToken($this->generateToken());
.....|..$user->setVerified(false);
.....|..
.....|..$this->entityManager->persist($user);
.....|..$this->entityManager->flush();

.....|..$this->sendEmail($user->getEmail(), $user->getToken(), $mailer);

.....|..$this->addFlash("success", "Email de confirmation envoyé à votre adresse");
.....|..
.....}

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

```

### Extrait contrôleur inscription

Maintenant que notre formulaire est rendu dans la vue, il faut gérer la soumission de celui-ci, vérifier qu'il soit valide et insérer les données dans notre table User en base de données.

Pour cela on reprend notre formulaire créé et l'on va utiliser la méthode « `handleRequest()` » qui prendra en argument l'objet « Request » qui représente une requête HTTP (ceci correspond aux variables « superglobales » `$_GET`, `$_POST` etc...).

Pour cela on va utiliser l'injection de dépendance, qui permet d'utiliser la méthode `index` en « embarquant » l'objet Request dont on affecte la référence à la variable « `$request` » (voir image ci-dessus).

Ainsi il ne reste plus qu'à récupérer les données lors d'une requête.

Finalement on vérifie grâce à une condition si le formulaire est soumis et s'il est valide, si c'est le cas nous n'avons plus qu'à récupérer les données du formulaire avec la méthode « `getData()` » et ensuite de les affecter à notre objet `User`.

Il faut ensuite ne pas oublier de « hasher » le mot de passe avant de l'envoyer à la base de données. Pour cela on utilise l'interface pour « hasher » les mots de passe en utilisant les « getters et setter » de notre entité `User`.

Ensuite, il faut générer un Token pour la confirmation ultérieure du compte.

Dernière étape, il faut enregistrer les données dans notre base de données. Nous avons donc besoin d'utiliser « l'entity manager interface » de Doctrine. Pour cela on déclare un constructeur dans notre classe et on utilise l'injection de dépendance pour utiliser notre interface et l'affecte à la variable « `$entityManager` » que l'on peut ensuite utiliser n'importe où dans notre classe.

Il suffit finalement de « figer » ou suivre l'objet avec la méthode « `persist()` » en lui passant cet objet (`User`) en argument puis

d'insérer les données en base de données avec la méthode

« `flush()` » ou en d'autres mots synchroniser le contexte de

persistance en base de données. Pour finir il faut envoyer un email avec un lien pour la confirmation du compte.

Nous avons maintenant notre formulaire d'inscription qui est parfaitement fonctionnel.

```

/**
 * @Route("/confirmer/{token}", name="confirm_account")
 * @param string $token
 */
public function confirmAccount(string $token, UserRepository $userRepo)
{
    $user = $userRepo->findOneBy(["token" => $token]);
    if($user){
        $user->setToken(null);
        $user->setVerified(true);
        $user->setRoles(['ROLE_USERACTIVE']);
        $this->entityManager->persist($user);
        $this->entityManager->flush();
        $this->addFlash("success", "Compte activé !");
        return $this->redirectToRoute("home");
    } else {
        $this->addFlash("error", "Ce compte n'existe pas !");
        return $this->redirectToRoute("home");
    }
}

/**
 * @return string
 * @throws \Exception
 */
private function generateToken()
{
    return rtrim(strtr(base64_encode(random_bytes(108)), '+/', '-_'), '=');
}

```

### Méthode pour vérifier un compte et pour générer un token

## b) La connexion

Pour la connexion il existe une ligne de commande « symfony console make :auth ». Cette ligne de commande permet de créer les fichiers nécessaires à la connexion d'un utilisateur dans notre application.

Cela va créer 3 fichiers, un « template » pour Twig qui sera la vue pour afficher notre formulaire, une classe que l'on appelle « LoginFormAuthenticator » qui contient toutes les méthodes nécessaires à l'authentification d'un utilisateur.

Pour finir un contrôleur appelé « SecurityController » qui contient les routes de connexion et déconnexion de l'application.

Tous les fichiers n'ont besoin de quasiment aucune modification, dans le fichier Twig on retrouve un formulaire qui a été créé avec un « input » de type « hidden » pour le token CSRF.

J'ai également créé une page pour réinitialiser le mot de passe. L'utilisateur entre son adresse dans un formulaire puis le soumet. On récupère alors dans le contrôleur le token CSRF et l'adresse email. Si un utilisateur possédant cette adresse existe en base de données et que le token CSRF est valide alors un token de réinitialisation pour le mot de passe est généré et enregistré en base de données. Enfin un email est envoyé avec un lien vers la route pour réinitialiser le mot de passe. Cette route utilise le token en base de données, si un utilisateur ayant ce token existe alors il peut réinitialiser son mot de passe.

```

.../**
... * @Route("/connexion/motdepasse/reinitialiser", name="pw_forgot")
... */
... public function forgotPasswordEmail(Request $request, UserRepository $userRepo, MailerInterface $mailer)
... {
...     $content = $request->request->all();
...
...     if ($content) {
...         $user = $userRepo->findOneBy(['email' => $content['email']]);
...
...         if ($user && $this->isCsrfTokenValid('forgot_password', $content['_csrf_token'])) {
...             $user->setPwToken($this->generateToken());
...             $this->entityManager->flush();
...
...             $this->sendEmail($user->getEmail(), $user->getPwToken(), $mailer);
...
...             $this->addFlash("success", "Email de réinitialisation envoyé à votre adresse");
...         }
...         else {
...             $this->addFlash("error", "Email non valide");
...         }
...     }
...
...     return $this->render('account/forgotpassword.html.twig');
... }

```

**Méthode du contrôleur pour envoyer un email et réinitialiser le mot de passe**

```

96  ..../**
97  .... * @Route("/mot-de-passe/reinitialiser/{token}", name="reset_password")
98  .... * @param string $token
99  .... */
100  ....public function PasswordReset(string $token, Request $request, UserRepository $userRepo, UserPasswordHasherInterface $hasher)
101  ....{
102  ....    $user = $userRepo->findOneBy(["pw_token" => $token]);
103  ....
104  ....    $form = $this->createForm(ResetPasswordType::class, $user);
105  ....
106  ....    $form->handleRequest($request);
107  ....
108  ....
109  ....    if ($user) {
110  ....        if ($form->isSubmitted() && $form->isValid()) {
111  ....            $user = $form->getData();
112  ....            $password = $hasher->hashPassword($user, $user->getPassword());
113  ....            $user->setPassword($password);
114  ....            $user->setPwToken(null);
115  ....            $this->entityManager->flush();
116  ....
117  ....            $this->addFlash("error", "Mot de passe réinitialisé avec succès");
118  ....            $this->redirectToRoute('home');
119  ....        }
120  ....    }
121  ....
122  ....    else {
123  ....        $this->addFlash("error", "Erreur");
124  ....        return $this->redirectToRoute('home');
125  ....    }
126  ....
127  ....    return $this->render('account/resetpassword.html.twig', [
128  ....        'formulaire' => $form->createView(),
129  ....    ]);
130  ....
131  ....}

```

**Méthode pour la vérification du token et réinitialisation du mot de passe**

## 4.2 Création de la page d'accueil

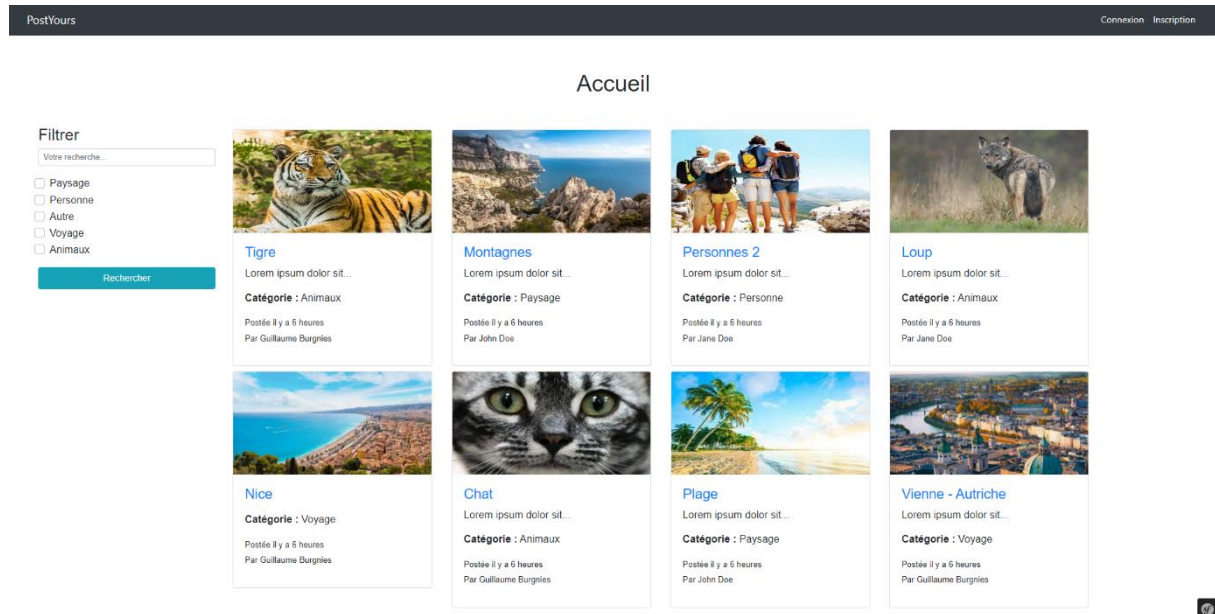
Comme on l'a vu précédemment avec l'inscription, on commence par créer un contrôleur avec la console, on lui donne le nom de « Home ».

Cela va créer deux fichiers un contrôleur et un fichier Twig pour la vue (HomeController & home/index.html.twig).

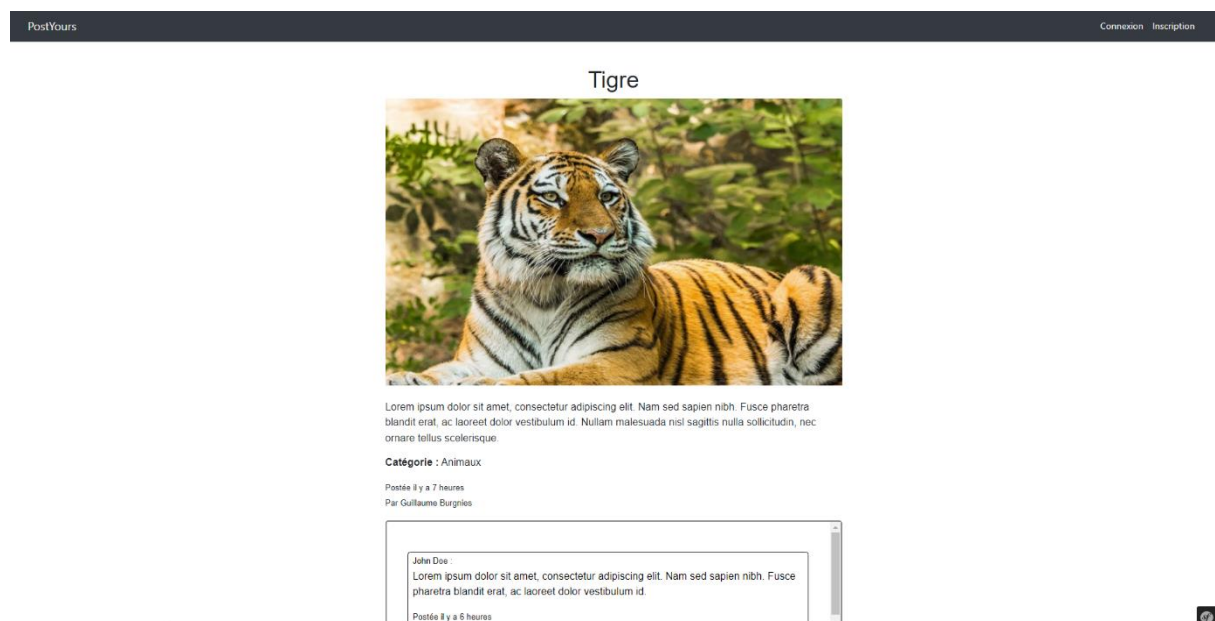
On rajoute également un fichier « show.html.twig » dans le dossier « templates/home » créé qui servira lorsqu'on cliquera sur une image en particulier et que l'on se rendra sur une page pour l'affichage de celle-ci, avec le titre, la description et les commentaires notamment.

Dans le contrôleur, il faut créer une route pour la page d'accueil « / » dans la méthode index et il suffira de récupérer dans la base de données toutes les images postées par les utilisateur (en utilisant le repository et la méthode « findby() ») et de les afficher du côté de notre vue Twig.

Ensuite, il faut créer une route avec un paramètre variable (id) pour afficher une image particulière et on nommera la méthode « show ».



Page d'accueil



Page d'une image (show)

## 4.3 Création de la page personnelle (utilisateurs)

Pour la page personnelle d'un utilisateur, il faut créer encore une fois un contrôleur avec son fichier Twig pour la vue via la console. Le nom donné est « Account ».

Dans cette page l'utilisateur verra les images qu'il a posté et pourra les modifier ou les supprimer.

Pour cela il faut donc dans notre contrôleur récupérer toutes les images postées par cet utilisateur est seulement celui-ci.

```

18 class AccountController extends AbstractController
19 {
20     #[Route('/compte', name: 'account')]
21     #[Security("is_granted('ROLE_USERACTIVE')")]
22     public function index(ImagesRepository $imagesRepo, Request $request): Response
23     {
24         .....
25         $user = $this->getUser();
26         $userId = $user->getId();
27         .....
28         $images = $imagesRepo->findby(['user' => $userId], ['createdAt' => 'DESC']);
29         .....
30         $search = new Search();
31         $form = $this->createForm(SearchAccountType::class, $search);
32         .....
33         $form->handleRequest($request);
34         .....
35         if ($form->isSubmitted() && $form->isValid()) {
36             .....
37             if ($search->categories !== []) {
38                 $search = $form->getData();
39                 $images = $imagesRepo->findWithCatAccount($search, $userId);
40             }
41             else {
42                 $images = $imagesRepo->findby(['user' => $userId], ['createdAt' => 'DESC']);
43             }
44         }
45     }

```

## 4.4 Création, modification et suppression d'un « post » (image)

Pour poster une image il faut créer une nouvelle page et donc un contrôleur et sa vue Twig et également un formulaire comme on a pu le voir avec l'inscription (4.1).

On obtient donc un contrôleur « ImageController » dans lequel on va gérer le formulaire, on va créer une nouvelle route avec la méthode index qui affichera tout d'abord le formulaire créé dans la vue Twig.

Ensuite, lorsque le formulaire sera soumis il faut récupérer les données vérifier si tout est valide et si tel est le cas envoyer les informations à la base de données pour créer un nouveau « post » (image).

On ajoute ensuite une autre méthode (edit) pour modifier une image, on utilise le même formulaire et une route avec un paramètre variable (id de l'image). On pourra donc récupérer les données de l'image en question dans le formulaire et enregistrer à la soumission de celui-ci.

Une dernière méthode est la méthode « delete » pour supprimer une image et toutes les informations (titre, description...). De la même manière il faut une route avec un paramètre variable pour ensuite supprimer l'image correspondante (Images du contrôleur Image disponible en annexe 2 / 2.4).



## 4.5 Création d'une page pour modifier le mot de passe

Dans le dossier templates/account, j'ai créé un nouveau fichier que l'on va appeler changePassword.html.twig qui sert à afficher le formulaire pour changer le mot de passe. Avec la console j'ai aussi créé un formulaire de la même façon que l'on a pu le faire auparavant. Dans notre contrôleur on récupère dans la variable « \$user » l'utilisateur actuellement connecté avec la méthode « getUser() »

Ceci étant fait on peut créer le formulaire avec la méthode « createForm() » puis on lui passe le formulaire créer « ChangePasswordType » et la variable (référence à l'objet) « \$user ».

Il ne reste plus qu'à « écouter » la requête grâce à la méthode « handleRequest » et on utilise l'objet Request qui est embarqué grâce à l'injection de dépendance dans notre méthode index. On vérifie si le formulaire est valide et soumis et si c'est le cas on va récupérer l'ancien mot de passe entré par l'utilisateur.

Ensuite, avec une condition on va vérifier si le mot de passe est correct et correspond bien avec la méthode « isValid() ». Si ce n'est pas le cas on informe l'utilisateur avec un message Flash. En revanche si c'est correct on récupère le nouveau mot de passe entré, on actualise l'objet « user » avec le nouveau mot de passe et on peut envoyer ces changements à la base de données. On termine en affichant un message flash que la modification a été faite avec succès.

```
.../**
... * @Route("/compte/modifier_motdepasse", name="account_password")
... */
... public function index(Request $request, UserPasswordHasherInterface $hasher)
... {
...     $user = $this->getUser();
...     $form = $this->createForm(ChangePasswordType::class, $user);
...     $form->handleRequest($request);
...     if ($form->isSubmitted() && $form->isValid()) {
...         $old_password = $form->get('old_password')->getData();
...         if ($hasher->isPasswordValid($user, $old_password)) {
...             $new_password = $form->get('new_password')->getData();
...             $password = $hasher->hashPassword($user, $new_password);
...             $user->setPassword($password);
...             $this->entityManager->flush();
...             $this->addFlash('success', "Votre mot de passe a bien été mis à jour");
...         }
...         else {
...             $this->addFlash('danger', "Le mot de passe actuel est incorrect");
...         }
...     }
...     return $this->render('account/changepassword.html.twig', [
...         'formulaire' => $form->createView()
...     ]);
... }
```

## 5. Filtre, back-office, sécurité et déploiement

### 5.1 Création d'un filtre

Dans cette partie, on veut donner la possibilité à l'utilisateur d'afficher les images en filtrant par catégorie ou en effectuant une recherche sur les titres des images.

Pour cela on va créer une nouvelle classe qu'on appellera « Search » dans laquelle on aura deux attributs : « string » et « catégories ».

La prochaine étape est de créer un formulaire grâce à la console, on appelle celui-ci SearchType.

Celui-ci aura un champ pour écrire une recherche par titre et des « checkboxes » pour sélectionner différentes catégories. Ce deuxième champ est donc relié à l'entité « Category » pour pouvoir afficher toutes les catégories existantes.

```

...public function buildForm(FormBuilderInterface $builder, array $options): void
...{
...    $builder
...        ->add('string', TextType::class, [
...            'label' => false,
...            'required' => false,
...            'attr' => [
...                'placeholder' => 'Votre recherche...',
...                'class' => 'form-control-sm'
...            ]
...        ])
...        ->add('categories', EntityType::class, [
...            'label' => false,
...            'required' => false,
...            'class' => Category::class,
...            'multiple' => true,
...            'expanded' => true,
...            'attr' => [
...                'placeholder' => 'Votre recherche...'
...            ]
...        ])
...        ->add('submit', SubmitType::class, [
...            'label' => 'Rechercher',
...            'attr' => [
...                'class' => 'btn-block btn-info'
...            ]
...        ])
...    ];
...}

```

Il faut maintenant créer les instructions SQL pour afficher les images suivant la recherche effectuée.

Tout d'abord on va créer la méthode qui pourra être utiliser dans le contrôleur, on place celle-ci dans le « Repository » Image. On appelle cette méthode « findWithCat() » et on lui passera en argument l'objet « Search » qui est un objet de la classe que l'on a créée auparavant.

Ensuite on fait notre requête, on sélectionne les entités « Category » et « Image », qui représentent les tables respectives de notre base de données, et on réalise une jointure.

Puis on vérifie si notre objet contient quelque chose dans « categories » ou « string ». Lorsque que quelque chose est présent dans « categories » on sélectionne

toutes les images ayant la catégorie ou les catégories sélectionnées.

Pour l'attribut « string » c'est le même principe mais avec le titre des images.

Notre méthode étant créée, on va pouvoir l'utiliser dans notre « HomeController ».

On notera l'important d'utiliser la méthode « setParameter » qui permet de se protéger contre les injections SQL.

Comme j'en ai parlé dans la partie sur la page d'accueil, on a d'abord une variables « \$images » qui va contenir les images qui ont été postées par tous les utilisateurs.

Il faut donc maintenant s'occuper de la recherche (filtre), dans le cas où l'utilisateur souhaite effectuer une recherche. On commence par instancier la classe « search » et on affecte à « \$search » la référence de ce nouvel objet.

On va également créer le formulaire pour avoir du côté de la vue la barre de recherche et les « checkboxes » qui correspondent à chaque catégorie.

On va ensuite « écouter » la requête et vérifier si le formulaire est valide et soumis par l'utilisateur. Si c'est le cas on récupère les données dans l'objet « \$search ».

Finalement, on récupère les images avec la méthode créée « findWithCat() » en appelant celle-ci grâce au « repository » de « Images » qui est « embarqué » grâce à l'injection de dépendance. Finalement on retourne le formulaire et l'objet « \$images » à notre vue Twig grâce à la méthode « render() ».

```

/**
 * Affichage du filtre par catégories page Home
 * @return Images[]
 */
public function findWithCat(Search $search)
{
    $query = $this
        ->createQueryBuilder('i')
        ->select('i', 'category')
        ->join('i.category', 'category')
        ;

    if (!empty($search->categories)) {
        $query = $query
            ->andWhere('category IN (:categories)')
            ->orderBy('i.updatedAt', 'DESC')
            ->setParameter('categories', $search->categories);
    }

    if (!empty($search->string)) {
        $query = $query
            ->andWhere('i.title LIKE :string')
            ->orderBy('i.updatedAt', 'DESC')
            ->setParameter('string', "%{$search->string}%");
    }

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

#[Route('/', name: 'home')]
public function index(ImagesRepository $imagesRepo, Request $request): Response
{
    $images = $imagesRepo->findBy([], ['createdAt' => 'DESC']);

    $search = new Search();
    $form = $this->createForm(SearchType::class, $search);

    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $search = $form->getData();

        $images = $imagesRepo->findWithCat($search);
    }

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

```

## 5.2 Création de la partie Administrateur

Pour la partie administration j'ai utilisé le Bundle « easy admin » qui permet la création d'un back-office très facilement. Cela est notamment utile pour gérer les utilisateurs, pouvoir ajouter, supprimer, modifier les catégories et également pour gérer les images ou commentaires.

## 5.3 Sécurité

La sécurité pour l'authentification est gérée par le Firewall de Symfony et « access control » pour l'autorisation (rôles) tout cela étant défini dans le fichier security.yaml.

Pour la protection contre les failles CSRF ou XSRF lors de la création d'un formulaire avec la commande « make :form » Symfony ajoute un champ caché au formulaire avec un token CSRF pour se protéger contre ce type de faille.

Il est aussi possible d'ajouter manuellement ce champ et de vérifier ensuite le token dans le contrôleur avant toute action.

Un bon exemple est lors de la suppression d'une image :

```

..#[Route('/image/{id<[0-9]+>}/supprimer', name: 'delete_image', methods: 'POST')]
..#[Security("is_granted('ROLE_USERACTIVE') and image.getUser() == user")]

..public function delete(Request $request, EntityManagerInterface $em, Images $image)
..{
..    $csrf_token = $request->request->all();
..    ..
..    if ($this->isCsrfTokenValid('imageDelete', $csrf_token['csrf_token'])) {
..        ..
..        $em->remove($image);
..        $em->flush();
..        ..
..        $this->addFlash('success', 'Image supprimée avec succès');
..        ..
..        return $this->redirectToRoute('account');
..    }
..}

```

On a ci-dessus la méthode « delete » qui permet de supprimer une image. Dans Twig, j'ai créé un formulaire avec un champ caché pour le token CSRF.

Lorsque que l'utilisateur supprime une image, on récupère le token présent dans le champ caché et on va vérifier que celui-ci est bien valide, si c'est le cas l'image (et titre, description...) peut être supprimée.

Autre point de sécurité illustré ici : l'attribut « security » en dessous de la route (voir image précédente) qui permet de restreindre l'accès, on vérifie ici que l'utilisateur a bien le rôle « USERACTIVE » qui indique que son compte est vérifié et on vérifie surtout que cet utilisateur est bien le « propriétaire » de l'image qui doit être supprimée, dans le cas contraire l'action ne peut pas être réalisée.

Pour les failles XSS, Twig échappe automatiquement tous caractères non-désirés, tel que du code JavaScript par exemple.

Autre faille de sécurité très problématique : les injections SQL. Dans la pratique on peut utiliser les requêtes préparées.

Dans ce projet j'utilise l'ORM Doctrine qui protège contre ce type de failles, il est possible de faire des requêtes plus personnelles grâce à DQL (Doctrine Query Language) comme dans le cas du filtre (5.1). Dans ce cas-là on peut utiliser des méthodes comme « setParameter » pour se protéger contre les injections SQL.

## 5.4 Mise en production

Le site a été déployé sur la plateforme Heroku, j'ai utilisé la CLI (Command line interface) heroku qui permet de déployer l'application avec GIT. J'ai donc ajouté un fichier Procfile ainsi qu'un fichier htaccess pour Apache.

## 6. Jeux d'essai

### 6.1 Tests unitaires

J'ai réalisé des tests unitaires, notamment pour tester les entités, les contrôleurs.

Ci-contre un extrait des tests sur l'entité User. Je fais des tests sur les contraintes liées à l'entité, comme par exemple le mot de passe qui doit avoir 8 caractères minimum, 1 majuscule, 1 minuscule, 1 chiffre et un caractère spécial. Ceci permet de vérifier que lors de l'inscription un utilisateur ne pourra pas laisser un champ vide ou encore utiliser un email non conforme.

```

public function getUserEntity() : User
{
    return (new User())
        ->setEmail("johndoe@gmail.com")
        ->setPassword("P@assword1")
        ->setFirstname('john')
        ->setLastname('Doe')
        ->setVerified(false)
        ->setCreatedAt(new DateTimeImmutable())
        ->setToken('6+2565+56gvxrg6g52xt6b5gx');
}

public function assertHasErrorsUser(User $user, int $number = 0) {
    self::bootKernel();
    $error = self::getContainer()->get('validator')->validate($user);
    $this->assertCount($number, $error);
}

public function testUserEntity() {
    $user = $this->getUserEntity();
    $this->assertHasErrorsUser($user, 0);
}

public function testUserEntityWrongEmailType() {
    $user = $this->getUserEntity()->setEmail('email.fr');
    $this->assertHasErrorsUser($user, 1);
}

public function testUserEntityWrongPasswordType() {
    $user = $this->getUserEntity()->setPassword('password');
    $this->assertHasErrorsUser($user, 1);
}

```

```

public function testAccountControllerRedirect() {
    $client = static::createClient();
    $client->request('GET', '/compte');
    $this->assertResponseRedirects('/connexion');
}

public function testImageControllerRedirectForNewImage() {
    $client = static::createClient();
    $client->request('GET', '/nouvelle/image');
    $this->assertResponseRedirects('/connexion');
}

```

Ci-dessus, un extrait pour des tests sur les redirections vers la page de connexion lorsqu'un utilisateur n'est pas connecté.

Enfin ci-contre un extrait pour tester la redirection lors de la connexion avec des identifiants corrects et avec des identifiants incorrects.

```

public function testLoginControllerWrongCredentials() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/connexion');
    $form = $crawler->selectButton('Se connecter')->form([
        'email' => 'guillaume.burgnies@hotmail.fr',
        'password' => '123',
    ]);
    $client->submit($form);
    $this->assertResponseRedirects('/connexion');
}

public function testLoginControllerGoodCredentials() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/connexion');
    $form = $crawler->selectButton('Se connecter')->form([
        'email' => 'guillaume.burgnies@hotmail.fr',
        'password' => '123456',
    ]);
    $client->submit($form);
    $this->assertResponseRedirects('/compte');
}

```

## 6.2 Test utilisateurs

Le principe pour ce test est de tester l'inscription d'un nouvel utilisateur. Lorsqu'un utilisateur s'inscrit il a plusieurs informations à fournir, les données à entrer sont donc :

- Nom (Champ qui ne peut être vide)
- Prénom (Champ qui ne peut être vide)
- Email (Ne peut être vide, doit être une adresse valide et qui n'existe pas déjà en base de données)
- Mot de passe (Ne peut être vide et doit contenir au moins 8 caractères, 1 majuscule, 1 minuscule, 1 caractère spécial et 1 chiffre)
- Confirmation de mot de passe (Ne peut être vide et doit être identique au mot de passe entré précédemment)

PostYours Connexion Inscription

### Inscription

Prénom :

Nom :

Email :

Mot de passe :

Confirmez votre mot de passe :

Si tout est correct, en plus des informations saisies, un token est généré et toutes ces informations sont enregistrées en base de données et on affecte également false à l'attribut « verified » correspondant à notre colonne « verified » en base de données. Finalement un mail est envoyé à l'adresse email entrée par l'utilisateur avec un lien (contenant le token généré) pour confirmer le compte.


id	email	roles	password	firstname	lastname	created_at (DC2Type:datetime_immutable)	updated_at (DC2Type:datetime_immutable)	token	verified
8	test@hotmail.fr	[]	\$2y\$13\$W12amnOksTJz8NX.klqXW0E6NusqspV4zIBhKbq/QQy...	Guillaume	Burgnies	2022-02-02 10:11:48	2022-02-02 10:11:48	R4FA50_M_CUaHJICB0HeilA95T0pdfgSP3BomP-HE4mZ8H5u5...	0

Les données obtenues sont donc conformes à ce qui était attendu.

En cas d'erreur : un champ laissé vide, un email non conforme... un message d'erreur sera affiché à l'utilisateur pour le champ correspondant, ceci étant des vérifications côté client comme dans l'image ci-dessous.

Email :


Mot de passe :

 Veuillez renseigner ce champ.

Dans le cas d'un utilisateur malveillant qui modifierait le code html par exemple et remplacerait l'attribut de l'input : type= 'email' par type='text', sans vérification côté serveur il pourrait s'enregistrer avec une

adresse email non conforme. C'est pourquoi que dans ce cas un message sera également affiché à la suite d'une vérification côté serveur comme sur l'image ci-dessous :

Email :



l'email n'est pas valide

Les données obtenues et le résultat sont donc encore une fois ce qui était attendu.



# Partie D : Veille technologique

## 1. Les failles de sécurité les plus courantes

Durant le stage, j'ai effectué des recherches sur les failles de sécurité les plus courantes et surtout quels sont les moyens pour s'en prémunir notamment avec les Frameworks que j'ai pu être amené à utiliser.

J'ai effectué ces recherches en Français et en Anglais avec des mots clefs comme failles, sécurité et web ou en Anglais les mots web, security et threats.

Ci-dessous un résumé des informations que j'ai pu trouver.

### Injection SQL

Une injection SQL se produit lorsque des pirates prennent le contrôle de sites qui génèrent des requêtes SQL à partir de données fournies par les utilisateurs sans vérifier au préalable si ces données sont valides. Ces pirates sont alors en position d'envoyer des requêtes SQL malveillantes et de transmettre des commandes directement à une base de données.

### Cross-site scripting (XSS)

Les attaques XSS (Cross-site scripting) visent les utilisateurs d'une application en injectant du code, généralement un script côté client (tel que du JavaScript), dans la sortie d'une application Web. Dès que la sortie ou la page infectée est affichée, le navigateur exécute le code, permettant ainsi à un pirate de détourner des sessions utilisateur, de rediriger l'utilisateur vers un site malveillant ou simplement d'endommager la page. Les attaques XSS sont possibles dans le contenu d'une page générée dynamiquement, à partir du moment où une application intègre des données fournies par l'utilisateur sans les valider ni les convertir correctement.

Pour empêcher les attaques par injection et les attaques XSS, il faut configurer une application en partant du principe que toutes les données, qu'elles proviennent d'un formulaire, d'une URL, d'un cookie voire de la base de données de l'application, proviennent d'une source non fiable. L'utilisation des Frameworks et aussi un moyen de s'en protéger.

Le ***Cross-site request forgery***, abrégé **CSRF** ou **XSRF**,

L'objet de cette attaque est de transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits. L'utilisateur devient donc complice d'une attaque sans même s'en rendre compte. L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

Pour empêcher ces attaques on peut :

- Utiliser des jetons de validité (ou *Token*) dans les formulaires est basé sur la création du token via le chiffrement d'un identifiant utilisateur, un nonce et un horodatage. Le serveur doit vérifier la correspondance du jeton envoyé.
- Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions : cette technique va naturellement éliminer des attaques simples basées sur les images, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.

### Problèmes d'authentification et de gestion des sessions

Les applications Web doivent gérer l'authentification des utilisateurs et établir des sessions pour suivre les demandes, car le protocole HTTP ne dispose pas de cette fonction.

Si toutes les informations d'authentification et les identifiants de session ne sont pas en permanence protégés par chiffrement, et protégés contre toute divulgation par d'autres failles telles que le XSS, un pirate peut détourner une session active et usurper l'identité d'un utilisateur.

### Références directes non sécurisées à un objet

Cette autre faille découle d'une mauvaise conception des applications, reposant sur l'hypothèse erronée que les utilisateurs suivent toujours les règles des applications. Par exemple, si l'ID de compte d'un utilisateur apparaît dans l'URL de la page ou dans un champ masqué, un utilisateur malveillant peut deviner l'ID d'un autre utilisateur et soumettre de nouveau la demande d'accès à ses données, en particulier si l'ID revêt une valeur prévisible.

Les meilleurs moyens de combattre cette vulnérabilité consistent à utiliser des ID et des noms de fichier et d'objet aléatoires, non prévisibles, et de ne jamais laisser apparaître les noms réels des objets.

Les données de ce type sont généralement exposées de façon incorrecte dans des URL et des liens, des champs de formulaire masqués, l'état d'affichage non protégé dans ASP.NET, des zones de liste déroulante, du code JavaScript et des objets côté client, tels que des applets Java.

En cas d'accès à des fichiers ou à du contenu sensible, vérifiez que l'utilisateur dispose des autorisations nécessaires.

### Mauvaise configuration de la sécurité

L'infrastructure qui prend en charge une application Web comporte de nombreux appareils et logiciels : serveurs, pare-feu, bases de données, systèmes d'exploitation et applications. Tous ces éléments doivent être configurés et gérés de façon sécurisée, l'application étant exécutée avec les privilèges minimaux requis. L'une des causes principales d'une administration système médiocre tient au manque de formation

adéquate des personnes responsables de la gestion des applications Web et de l'infrastructure sous-jacente.

Si la sécurité et la confidentialité doivent rester une priorité dans toutes les phases du processus de développement, il est tout aussi essentiel de fournir une formation et des ressources appropriées aux personnes chargées de la gestion quotidienne des réseaux et des applications.

Enfin, planifiez un test d'intrusion pour les applications Web qui traitent des données sensibles. Cette méthode proactive permet d'évaluer la capacité à résister à une attaque et de détecter les vulnérabilités avant les pirates.

## 2. La sécurité et la validation des mots de passe

### Sécurité

Un mot de passe doit avoir un niveau minimum de complexité pour ne pas être facilement deviné à des fins malveillantes. Pour se faire, on avertit généralement l'utilisateur si la complexité n'est pas suffisante selon les bonnes pratiques. Dans mes deux projet ce procédé est géré grâce aux expressions régulières, grâce à un pattern Regex on peut vérifier que le mot de passe entré dispose bien d'au moins :

- 8 caractères au total
- une lettre majuscule
- une lettre minuscule
- un chiffre
- un caractère spécial

Si la condition n'est pas validée, alors une erreur est affichée sous le champ concerné à la soumission du formulaire.

### Validation

En plus de la sécurité, dans le cadre de l'inscription d'un utilisateur on place un autre champ qui permet de confirmer le mot de passe entré précédemment. Ce champ obligatoire dispose dans ses règles d'une validation de la valeur à celle présente dans le champ précédent (mot de passe). Si la valeur de ce champ n'est pas remplie et n'est pas égale au mot de passe entré auparavant alors une erreur est affichée et le formulaire ne sera pas soumis.

```
password: Yup.string()
  // 8 caractères Min, 1 Majuscule, 1 minuscule, 1 caractère spécial, 1 chiffre
  .matches(/^(?=.*{8,})(?=.*[!@#$%^&*()\-_+={};:,<.>]){1})(?=.*\d)((?=.*[a-z]){1})(?=.*[A-Z]){1}).*$/, {
    'signup.minimum', {ns: 'auth'}}
  .required(t('signup.ce champ est requis', {ns: 'auth'})),
```

Expression régulière utilisée avec React

### 3. Recherche à partir d'un site anglophone

Dans le projet en entreprise, j'ai utilisé Formik et Yup pour la validation des formulaires, ne connaissant pas ces bibliothèques j'ai effectué une recherche sur leur utilisation. Les mots clefs que j'ai employé sont « Formik », « Yup » et « React ».

En troisième position j'ai trouvé un article très intéressant et sur le site « smashingmagazine.com » dont je fais la traduction de deux extraits ci-dessous.

As developers, it is our job to ensure that when users interact with the forms we set up, the data they send across is in the form we expect.

In this article, we will learn how to handle form validation and track the state of forms without the aid of a form library. Next, we will see how the Formik library works. We'll learn how it can be used incrementally with HTML input fields and custom validation rules. Then we will set up form validation using Yup and Formik's custom components and understand how Yup works well with Formik in handling Form validation. We will implement these form validation methods to validate a simple sign up form I have set up.

En tant que développeurs, il est de notre devoir de nous assurer que lorsque les utilisateurs interagissent avec les formulaires que nous avons configurés, les données qu'ils envoient sont sous la forme que nous attendons. Dans cet article, nous allons apprendre à gérer la validation des formulaires et à suivre l'état des formulaires sans l'aide d'une bibliothèque de formulaires. Ensuite, nous verrons comment fonctionne la bibliothèque Formik. Nous apprendrons comment il peut être utilisé de manière incrémentielle avec des champs de saisie HTML et des règles de validation personnalisées. Ensuite, nous allons configurer la validation de formulaire à l'aide des composants personnalisés de Yup et Formik et comprendre comment Yup fonctionne aussi bien avec Formik dans la gestion de la validation de formulaire. Nous mettrons en œuvre ces méthodes de validation de formulaire pour valider un simple formulaire d'inscription que j'ai mis en place.

Formik is a flexible library. It allows you to decide when and how much you want to use it. We can control how much functionality of the Formik library we use. It can be used with HTML input fields and custom validation rules, or Yup and the custom components it provides. Formik makes form

validation easy! When paired with Yup, they abstract all the complexities that surround handling forms in React.


Yup is a JavaScript object schema validator. While it has many powerful features, we'll focus on how it helps us create custom validation rules so we don't have to. This is a sample Yup object schema for a sign-up form. We'll go into Yup and how it works in depth later in the article.


Formik est une bibliothèque flexible. Il vous permet de décider quand nous voulons l'utiliser et nous pouvons également contrôler la quantité de fonctionnalités de la bibliothèque Formik que nous utilisons. Il peut être utilisé avec des champs de saisie HTML et des règles de validation personnalisées, ou avec Yup et les composants personnalisés qu'il fournit. Formik facilite la validation des formulaires !


Lorsqu'il est associé à Yup, tous deux font abstraction de toutes les complexités qui entourent la gestion des formulaires dans React. Yup est un validateur de schéma d'objet JavaScript. Bien qu'il possède de nombreuses fonctionnalités puissantes, nous allons nous concentrer sur la façon dont il nous aide à créer des règles de validation personnalisées afin que nous n'ayons pas à le faire. Il s'agit d'un exemple de schéma d'objet Yup pour un formulaire d'inscription. Nous verrons plus en profondeur comment Yup fonctionne plus loin dans l'article.

J'ai également rencontré des difficultés lors de la configuration d'un bundle (Kreait Firebase SDK PHP) utilisé dans Symfony pour la base de données en temps réel. J'ai donc posé une question sur Discord sur le salon dédié à ce bundle.

Voici donc des captures d'écrans de la conversation :

 **Guillaume Burgnies** 23/11/2021  
Hello, I'm using Symfony 5.3.10 and php 8.0.13. I installed the kreait/firebase-bundle. I have this error when using dependency injection : "Cannot autowire service "App\Controller\TestController": argument "\$database" of method "\_\_construct()" references class "Kreait\Firebase\Database" but no such service exists. You should maybe alias this class to the existing "kreait\_firebase.my\_project.database" service." I suppose I have a problem with the configuration, would be glad to have some help, thanks.

 **jeromegamez** 23/11/2021  
I just checked, and my examples repo still works as it is 😊. I think you'll need to provide me with more information, e.g. how your configuration looks like.

 **Guillaume Burgnies** 24/11/2021  
I resolved the problem, it was with the config file. Thanks for your help.

# Conclusion

Pour conclure, ce stage m'a permis d'approfondir énormément mes connaissances, notamment en travaillant sur différents Frameworks. Il m'a aussi fait prendre conscience des responsabilités et des difficultés que l'on peut avoir dans la réalisation d'un projet complet. L'expérience acquise au fil de ma formation pour devenir développeur et mon implication personnelle m'ont été bénéfiques, et je pense avoir pris les bonnes décisions au cours des tâches qui m'étaient attribuées ainsi que dans mon implication dans celles-ci.

## Remerciements

Je tiens dans un premier temps à remercier tout particulièrement mon tuteur de stage et fondateur de BS Next, Mr Samir BOUDJEMA, pour son investissement à mes côtés, et pour le temps qu'il m'a consacré tout au long de mon stage.

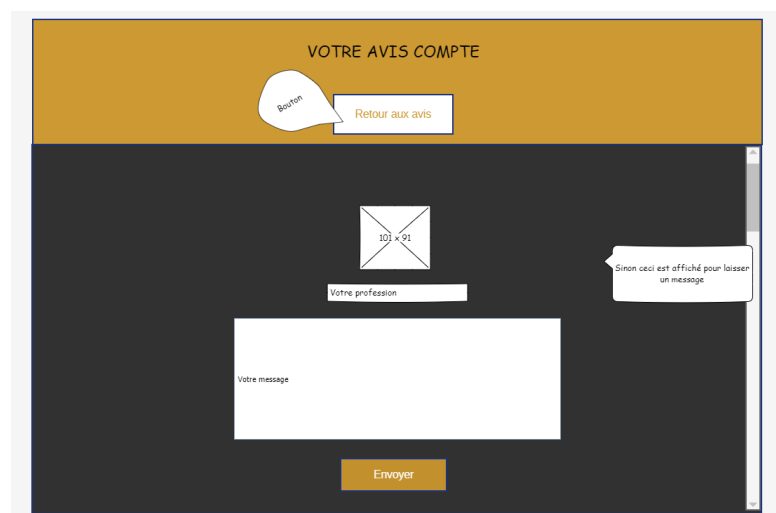
Je voudrais également remercier les formateurs de l'AFPA pour leur bienveillance et sympathie tout au long de ma formation. Pour finir, je voudrais aussi remercier le centre de formation AFPA de Nice qui m'a permis de suivre cette formation de Développeur Web et Web Mobile.

# Annexes



# Annexe 1 : Projet en entreprise

## 1.1 Le Maquettage



CONTACT RAPIDE 0666666666 Contact@contact.fr	ABONNEZ-VOUS A NOTRE NEWSLETTER <input type="text"/> <input type="button" value="Valider"/>	NOTRE ADRESSE 00 RUE DE NUL PART 00000 VILLE
RETROUVEZ NOUS SUR LES RESEAUX		
FB TW In		
MENTIONS LEGALES PLAN DU SITE PREFERENCES COOKIES CONTACT		



## 1.2 Le questionnaire

L'intégration du questionnaire n'était pas prévue au départ dans le cahier des charges. Cela étant intervenu en fin de stage et le développement n'étant pas totalement terminé, c'est pour ces deux raisons que ceci a été placé en annexe.

Les utilisateurs ont la possibilité de participer à un questionnaire. Ces questionnaires sont créés par l'administrateur grâce au back-office. Il y a également la possibilité de modifier un questionnaire, supprimer une ou des questions, ou même supprimer entièrement un questionnaire. Je ne vais pas détailler tout le code dans cette partie mais je vais seulement prendre un extrait en l'occurrence le code qui permet d'enregistrer les résultats d'un utilisateur après la participation à un questionnaire.

La première étape est de récupérer les réponses de l'utilisateur au questionnaire et toutes les données nécessaires.

On vérifie ensuite que la session de l'utilisateur et si tout est correct, on commence le traitement des données.

On va d'abord récupérer les questions (et bons résultats) du questionnaire en question et on va également récupérer le nombre de questions de ce questionnaire. On initialise ensuite deux variables à zéro pour le score et le total.

```

../**
... * @Route("/api/answer_quiz", name="answer_quiz")
... */
... public function answerQuiz(Request $request)
... {
...     //On récupère les données venant du front
...     $content = json_decode($request->getContent());

...     //On assigne chaque donnée récupérée à une variable
...     $questionnaire = $content->questionnaire;
...     $email = $content->email;
...     $answer = $content->answer;
...     $time = $content->time;
...     $tokenId = $content->tokenId;

...     //J'enlève le point dans l'email (non accepté par firebase)
...     $emailChange = str_replace('.', '', $email);

...     //Vérification de l'utilisateur
...     $verifiedIdToken = $this->auth->verifyIdToken($tokenId);
...     $uid = $verifiedIdToken->claims()->get('sub');
...     $user = $this->auth->getUser($uid);

...     if ($user) {
...         //Récupère les questions du quiz et on calcule le nombre de questions
...         $reference = 'quiz/'.$questionnaire;
...         $snapshot = $this->database->getReference($reference)->getSnapshot();
...         $val = array_values($snapshot->getValue());
...         $length = count($val);

...         //Initialise les variables
...         $score = 0;
...         $total = 0;

...         //Prépare email et id par défaut pour BDD
...         $postData = [
...             'id' => 'default',
...             'email' => $email,
...         ];
    
```

Pour préparer la base de données, on envoie « l'id » par défaut et l'email de l'utilisateur dans deux « nodes ou références » différentes : une pour les résultats affichés à l'utilisateur et l'autre pour le back office.

Dans la suite on récupère la valeur de la clef pour chaque « node ou référence » où les données se sont enregistrées.

```

    try {
        //Envoi email et id par défaut à BDD des résultats pour User et admin
        $push1 = $this->database->getReference('resultQuiz/'.$questionnaire->push($postData));
        $push2 = $this->database->getReference('resultQuizUser/'.$emailChange.'/'.$questionnaire->push($postData));
    } catch (FirebaseException $e) {
        return $this->json([
            'Une erreur s'est produite : ' . $e->getMessage()
        ]);
    } catch (Throwable $e) {
        return $this->json([
            'Une erreur s'est produite : ' . $e->getMessage()
        ]);
    }

    //Change id dans BDD
    try {
        $key1 = $push1->getKey();
        $ref1 = 'resultQuiz/'.$questionnaire.'/'.$key1.'/'.$id';
        $this->database->getReference($ref1)->set($key1);

        $key2 = $push2->getKey();
        $ref2 = 'resultQuizUser/'.$emailChange.'/'.$questionnaire.'/'.$key2.'/'.$id';
        $this->database->getReference($ref2)->set($key2);
    } catch (FirebaseException $e) {
        return $this->json([
            'Une erreur s'est produite : ' . $e->getMessage()
        ]);
    } catch (Throwable $e) {
        return $this->json([
            'Une erreur s'est produite : ' . $e->getMessage()
        ]);
    }
}

```

Ensuite, ici grâce à une boucle « for » on va affecter à un tableau les bons résultats pour chaque question. Par exemple la position « 0 » du tableau aura la bonne réponse de la question 1, c'est-à-dire un nombre 1, 2, 3 ou 4.

Puis on va utiliser ce tableau pour comparer aux résultats de l'utilisateur. Encore une fois grâce à une boucle, on va alors affecter +1 à la variable « score » ou non, enregistrer si la réponse est correcte ou non et également la réponse donnée par l'utilisateur.

```

    //On récupère dans un tableau les numéros des bonnes réponses pour chaque question
    for ($i = 0; $i < $length; $i++) {
        $tab1 = $val[$i];
        $tab2 = $tab1['reponse'];
        $result[$i] = $tab2;
    }

    //On utilise le tableau précédent pour comparé aux réponses données et on envoi les résultats à la BDD
    for ($j = 0; $j < $length; $j++) {
        if ($answer[$j] != null) {
            $question = $answer[$j]->id;
            $total = $length;

            if ($question === $result[$j]) {
                $score ++;
                $postData = [
                    'question' . ($j+1) => 'Correct',
                    'repQuestion' . ($j+1) => $answer[$j]->value,
                ];
            } else {
                $postData = [
                    'question' . ($j+1) => 'Incorrect',
                    'repQuestion' . ($j+1) => $answer[$j]->value,
                ];
            }
        } else {
            $postData = [
                'question' . ($j+1) => 'Aucune réponse',
                'repQuestion' . ($j+1) => 'Aucune réponse',
            ];
        }
    }
}

```

Pour finir à chaque fois que l'on boucle on envoie les données à la base de données. Lorsque la boucle est terminée on envoie alors à la base de données, le temps de réponse le score et le total. Tout cela en utilisant le bloc « try & catch » pour récupérer les erreurs éventuelles.

```

try {
    $this->database->getReference('resultQuiz/'.$questionnaire.'/'.$key1)->update($postData);
    $this->database->getReference('resultQuizUser/'.$emailChange.'/'.$questionnaire.'/'.$key2)->update($postData);
} catch (FirebaseException $e) {
    return $this->json([
        'Une erreur s\'est produite : '.$e->getMessage()
    ]);
} catch (Throwable $e) {
    return $this->json([
        'Une erreur s\'est produite : '.$e->getMessage()
    ]);
}

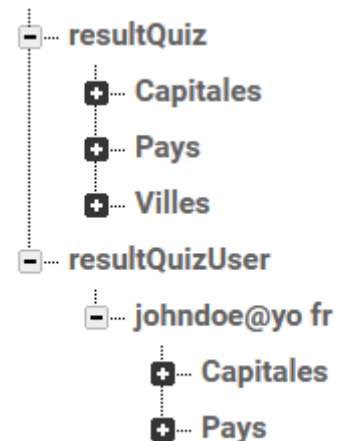
//Déclare le tableau pour remplir la BDD
$postData = [
    'temps' => $time->h.'h'. $time->m.'min'. $time->s.'sec',
    'score' => $score,
    'total' => $total
];

//Envoie à la BDD du Tableau ci-dessus
try {
    $this->database->getReference('resultQuiz/'.$questionnaire.'/'.$key1)->update($postData);
    $this->database->getReference('resultQuizUser/'.$emailChange.'/'.$questionnaire.'/'.$key2)->update($postData);

    return $this->json([
        'Résultats enregistrés'
    ]);
} catch (FirebaseException $e) {
    return $this->json([
        'Une erreur s\'est produite : '.$e->getMessage()
    ]);
} catch (Throwable $e) {
    return $this->json([
        'Une erreur s\'est produite : '.$e->getMessage()
    ]);
}
}

```

On a ici à droite un exemple de la base de données de tests dans Firebase. On voit que l'on a deux « nodes ou références » différentes : « resultQuiz » pour le bock office où l'on a chaque questionnaire. Dans notre cas on voit 3 questionnaires (Capitales, Pays et Villes) qui sont donc des questionnaires où un ou des utilisateurs ont participé.

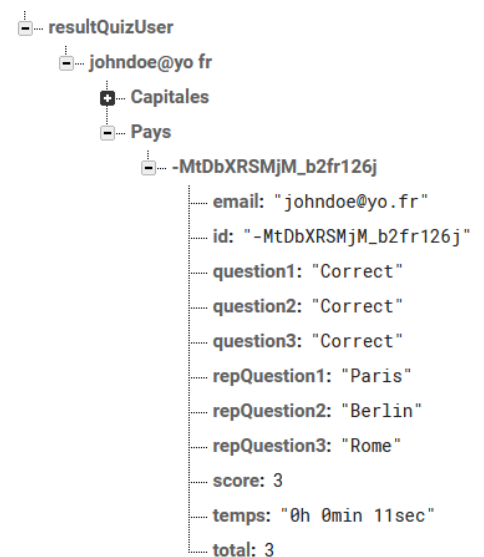


En dessous, on a « resultQuizUser » qui contient les résultats affichés pour les utilisateurs. On voit qu'au premier niveau on a l'adresse email des utilisateurs, au deuxième niveau le nom des questionnaires auxquels l'utilisateur a participé.

Sur l'image à droite, on voit ici les résultats de l'utilisateur « johndoe » pour le questionnaire pays.

Comme on a pu le voir dans le traitement du contrôleur on a ici tous les résultats enregistrés.

On a la réponse donnée à chaque question, si la réponse est correcte ou non et également le score, le temps de réponse etc...



# Annexe 2 : Projet personnel

## 2.1 Page d'accueil (Code Twig)

```

1  {% extends 'base.html.twig' %}
2
3  {% block title %}Accueil{% endblock %}
4
5  {% block body %}
6
7  <div class="wrapper">
8      <h1 class="text-center mb-5">Accueil</h1>
9
10     <div class="row">
11         <div class="col-xl-2">
12             <h3>Filtre</h3>
13             {{form(formulaire)}}
14         </div>
15         <div class="col-xl-9">
16             <div class="row">
17                 {% for image in images %}
18                 <div class="col-md-3 mt-2">
19
20                     <div class="card mr-1" style="width:;>
21                         <a href="{{ path('home_image_show', {id: image.id}) }}">
22                             
23                         </a>
24
25                         <div class="card-body">
26                             <h4><a href="{{ path('home_image_show', {id: image.id}) }}">{{image.title}}</a></h4>
27                             <p>{{image.description | u.truncate(25, '...')}}</p>
28                             <p><strong>Catégorie :</strong> {{image.category.name}}</p>
29                             <p>
30                                 <small>Postée {{image.createdAt | ago}}</small></br>
31
32                             {% if app.user == image.user %}
33                                 <small>Par vous</small>
34                             {% else %}
35                                 <small>Par {{ image.user.firstname }} {{ image.user.lastname }}</small>
36                             {% endif %}
37                         </div>
38                     </div>
39                 </div>
40             </div>
41         {% endfor %}
42     </div>

```

## 2.2 Page « show » affichage d'une image (Code Twig)

```
{% extends 'base.html.twig' %}

{% block title %}{{image.title}}{% endblock %}

{% block body %}

<div class="example-wrapper">
... <h1 class="text-center">{{image.title}}</h1>
...
... <div>
... <div class="">
...
... 
...
... <div class="mt-4">
... <p>{{image.description}}</p>
... <p><strong>Catégorie :</strong> {{image.category.name}}</p>
... <p>
... <small>Postée {{image.createdAt | ago}}</small></br>
... <% if app.user == image.user %>
... <small>Par vous</small>
... <% else %>
... <small>Par {{ image.user.firstname }} {{ image.user.lastname }}</small>
... <% endif %>
... </p>
...
... <% if app.user == image.user %>
... <a href="{{ path('edit_image', {id: image.id}) }}">Modifier</a><span> | </span>
... <a href="#" onclick="event.preventDefault();
... confirm('Êtes-vous certain de supprimer cette image ?') && document.getElementById('image_delete').submit();"
... >
... Supprimer
... </a>
...
... <form id="image_delete" action="{{ path('delete_image', {id: image.id}) }}" method="post">
... <input type="hidden" name="csrf_token" value="{{ csrf_token('imageDelete') }}">
... <input type="hidden" name="_method" value="DELETE">
... </form>
... <% endif %>
... </div>
... </div>
... </div>

... <% if app.user %>
... <section class="mt-5">
... <div>
... {{form(formulaire)}}
... </div>
... </section>
... <% endif %>

... <section>
... <% if showComments != null %>
... <div class="comments-border">
...
... <% for showComment in showComments %>
...
... <div class="comment-border pl-2 mt-3">
... <div class="comment-container">
... <small class="">{{showComment.user.firstname}} {{showComment.user.lastname}} :</small><br>
... <p>{{showComment}}</p>
... <small class="mt-2">Postée {{showComment.createdAt | ago}}</small>
... <% if app.user == showComment.user %>
... <small class="ml-2"><a href="#" onclick="event.preventDefault();
... confirm('Êtes-vous certain de supprimer ce commentaire ?') && document.getElementById('image_delete').submit();"
... >
... Supprimer
... </a></small>
...
... <form id="image_delete" action="{{ path('delete_comment', {id: showComment.id}) }}" method="post">
... <input type="hidden" name="csrf_token" value="{{ csrf_token('commentDelete') }}">
... <input type="hidden" name="_method" value="DELETE">
... </form>
... <% endif %>
... </div>
... </div>
... <% endfor %>
... </div>
... <% else %>
... <p>Pas encore de commentaire{% if app.user != image.user %}, soyez le premier ou la première !{% endif %}</p>
... <% endif %>
... </section>
...
... </div>
{% endblock %}
```

## 2.3 Page compte utilisateur (Code Twig)

```

<div class="wrapper">
....<h1>Mon compte</h1>
....<p>Bienvenue {{app.user.firstname}}</p>

....<p><a href="{{ path('account_password') }}">Modifier mon mot de passe</a></p>

....</div>

<div class="row mt-5">
....<div class="col-xl-2">
....<h3>Filtre</h3>
....{{form(formulaire)}}
....</div>
....<div class="col-xl-9">
....<div class="row">
....    {% if images %}
....    {% for image in images %}
....    <div class="col-md-3 mt-2">
....
....        <div class="card mr-1" style="width:;>
....            <a href="{{ path('home_image_show', {id: image.id}) }}">
....            
....            </a>
....
....            <div class="card-body">
....                <h3><a href="{{ path('home_image_show', {id: image.id}) }}">{{image.title}}</a></h3>
....                <p>{{image.description | u.truncate(25, '...')}}</p>
....                <p>{{image.category.name}}</p>
....                <p><small>Posté {{image.createdAt | ago}}</small></p>
....                <a href="{{ path('edit_image', {id: image.id}) }}">Modifier</a><span> | </span>
....                <a href="#" onclick="event.preventDefault();
....                confirm('Êtes-vous certain de supprimer cette image ?') && document.getElementById('image_delete').submit();"
....                >Supprimer
....                </a>
....
....                <form id="image_delete" action="{{ path('delete_image', {id: image.id}) }}" method="post">
....                <input type="hidden" name="csrf_token" value="{{ csrf_token('imageDelete') }}">
....                <input type="hidden" name="_method" value="DELETE">
....                </form>
....            </div>
....        </div>
....    </div>
....    </div>
....    </div>
....    {% endfor %}
....    {% else %}
....    <p class="text-center">Aucune image postée</p>
....    {% endif %}
....</div>
</div>

```



## 2.4 Contrôleur Image

```

19 ...#[Route('/nouvelle/image', name: 'create_image')]
20 ...#[Security("is_granted('ROLE_USERACTIVE')")]
21
22 ...public function index(Request $request, EntityManagerInterface $em): Response
23 ...{
24
25     ...$image = new Images;
26     ...$form = $this->createForm(ImageType::class, $image);
27     ...
28     ...$form->handleRequest($request);
29
30     ...if ($form->isSubmitted() && $form->isValid())
31     ...{
32         ...if (null !== $image->getImageFile()) {
33             ...$image->setUser($this->getUser());
34             ...$em->persist($image);
35             ...$em->flush();
36
37             ...$this->addFlash('success', 'Image postée avec succès');
38             ...return $this->redirectToRoute('home');
39         }
40         ...else {
41
42             ...$this->addFlash('danger', 'Veuillez choisir une image');
43         }
44     }
45
46     ...return $this->render('image/create.html.twig', [
47         ...'formulaire' => $form->createView(),
48     ]);
49 ...}

```

```

53 ...#[Route('/image/{id<[0-9]+>}/modifier', name: 'edit_image')]
54 ...#[Security("is_granted('ROLE_USERACTIVE') and image.getUser() == user")]
55
56 ...public function edit(Request $request, EntityManagerInterface $em, Images $image)
57 ...{
58     ...$form = $this->createForm(ImageType::class, $image);
59
60     ...$form->handleRequest($request);
61
62     ...if ($form->isSubmitted() && $form->isValid()) {
63
64         ...if (null !== $image->getImageName()) {
65             ...$em->flush();
66
67             ...$this->addFlash('success', 'Image modifiée avec succès');
68             ...return $this->redirectToRoute('account');
69         }
70         ...else {
71             ...$this->addFlash('success', 'Veuillez choisir une image');
72         }
73     }
74
75     ...return $this->render('image/edit.html.twig', [
76         ...'image' => $image,
77         ...'formulaire' => $form->createView(),
78     ]);
79 ...}

```

```
..#[Route('/image/{id<[0-9]+>}/supprimer', name: 'delete_image', methods: 'POST')]
..#[Security("is_granted('ROLE_USERACTIVE') and image.getUser() == user")]

..public function delete(Request $request, EntityManagerInterface $em, Images $image)
..{
..    $csrf_token = $request->request->all();
..    ..
..    if ($this->isCsrfTokenValid('imageDelete', $csrf_token['csrf_token'])) {
..        ..
..        $em->remove($image);
..        $em->flush();
..
..        $this->addFlash('success', 'Image supprimée avec succès');
..        ..
..        return $this->redirectToRoute('account');
..    }
..}
```

## 2.5 Formulaires

PostYours

ConnexionInscription

Inscription

Prénom :

Entrez votre prénom

Nom :

Entrez votre nom

Email :

Entrez votre adresse email

Mot de passe :

Entrez votre mot de passe

Confirmez votre mot de passe :

Confirmation du mot de passe

S'inscrire

© 2022 Mentions légales À propos

PostYours

AdminPosterMon compteDéconnexion

Poster une nouvelle image

Titre

Description

Catégorie

Paysage

Image

Choisir un fichier

Aucun fichier choisi

Valider

© 2022 Mentions légales À propos